ZILOG

Z80®
Microprocessor Family

Microprocessor Solutions For Datacommunications and Computer Peripheral Applications

Includes User Manual Specifications for the following parts:

Z80® CPU	Z80® PIO
Z80® CTC	Z80® SIO
Z80® DMA	

User's Manual

ZiLOG

Z80® Family User's Manual

Includes User Manual Specifications for the following parts:

- Z80® CPU
- Z80® CTC
- Z80® DMA
- Z80® PIO
- Z80® SIO

DC 8309-00

INTRODUCTION

The Zilog Z80® family of components can be configured with any type of standard semiconductor memory to generate computer systems with an extremely wide range of capabilities. For example, as few as two LSI circuits and three standard TTL MSI packages can be combined to form a simple controller. With additional memory and I/O devices a computer can be constructed with capabilities that only a minicomputer could previously deliver. This wide range of computational power allows standard modules to be constructed by a user that can satisfy the requirements of an extremely wide range of applications.

The major reason for MOS LSI domination of the microcomputer market is the low cost of these few LSI components. For example, MOS LSI microcomputers have already replaced TTL logic in such application as terminal controllers, peripheral device controllers, traffic signal controllers, point of sale terminals, intelligent terminals, and test systems. In fact, the MOS LSI microcomputer is finding its way into almost every product that now uses electronics and it is even replacing many mechanical systems such as weight scales and automobile controls.

The MOS LSI microcomputer market is already will established and new products using them are being developed at an extraordinary rate. The Zilog Z80 component set has been designed to fit into this market by the following factors:

- The Z80 is fully software compatible with the popular 8080A CPU offered from several sources. Existing designs can be easily converted to include the Z80 as a superior alternative.

- The Z80 component set is superior in both software and hardware capabilities to any other microcomputer system on the market. These capabilities provide the user with significantly lower hardware and software development costs while also offering additional system features.

- A complete line of software support with strong emphasis on high-level languages and a disk-based development system with advanced real-time debug capabilities is offered to enable the user to easily develop new products.

Microcomputer systems are extremely simple to construct using Z80 components. Any such system consists of three parts:

- CPU (Central Processing Unit)
- Memory
- Interface Circuits to Peripheral Devices

The CPU is the heart of the system. Its function is to obtain instructions from the memory and perform the desired operations. The memory is used to contain instructions and, in most cases, data that is to be processed. For example, a typical instruction sequence may be to read data from a specific peripheral device, store it in a location in memory, check the parity, and write it to another peripheral device. Note that the Zilog component set includes the CPU and various general-purpose I/O device controllers, while a wide range of memory devices may be used from any source. Thus, all requited components can be connected together in a very simple manner with virtually no other external logic. The user's effort then becomes primarily one of software development. That is, the user can concentrate on describing the problem and translating it into a series of instructions that can be loaded into the microcomputer memory. Zilog is dedicated to making this step of software generation as simple as possible. A good example of this is the assembly language in which a simple mnemonic is used to represent every instruction that the CPU can perform. This language is self-documenting in such a way that from the mnemonic the user can understand exactly what the instruction is doing without constantly checking back to a complex cross listing.

TABLE OF CONTENTS

A

List of Figures

List of Tables

CHAPTER 1
ARCHITECTURE

1.0 INTRODUCTION

A block diagram of the internal architecture of the Z80 CPU is shown in Figure 1-1. The diagram shows all of the major elements in the CPU and it should be referred to throughout the following descriptions.

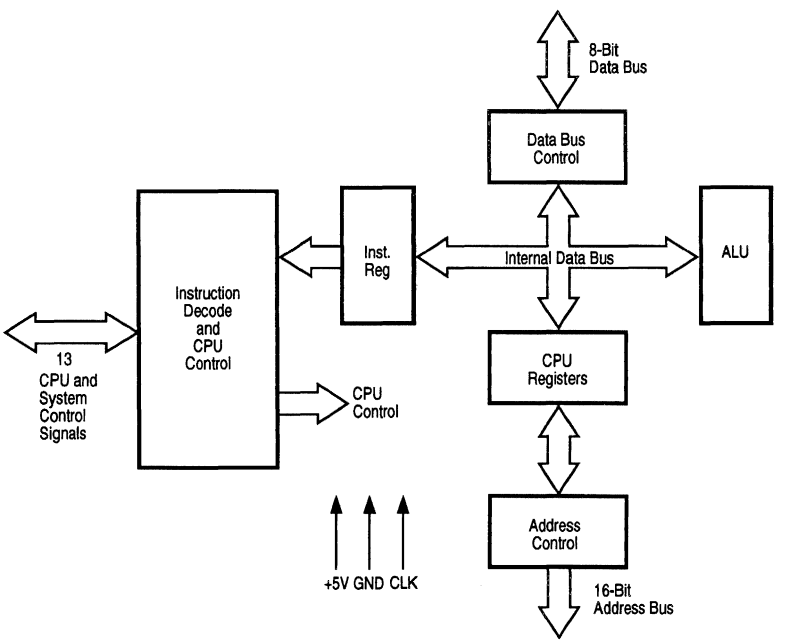


Figure 1-1. Z80 CPU Block Diagram

1.1 CPU REGISTERS

The Z80 CPU contains 208 bits of R/W memory that are accessible to the programmer. Figure 1-2 illustrates how this memory is configured into eighteen 8-bit registers and four 16-bit registers. All Z80 registers are implemented using static RAM. The registers include two sets of six general-purpose registers that may be used individually as 8-bit registers or in pairs as 16-bit registers. There are also two sets of accumulator and flag registers and six special-purpose registers.

1.1.1 Special-Purpose Registers

Program Counter (PC). The program counter holds the 16-bit address of the current instruction being fetched from memory. The PC is automatically incremented after its contents have been transferred to the address lines. When a program jump occurs the new value is automatically placed in the PC, overriding the incrementer.

Stack Pointer (SP). The stack pointer holds the 16-bit address of the current top of a stack located anywhere in external system RAM memory. The external stack memory is organized as a last-in first-out (LIFO) file. Data can be pushed onto the stack from specific CPU registers or popped off of the stack into specific CPU registers through the execution of PUSH and POP instructions. The data popped from the stack is always the last data pushed onto it. The stack allows simple implementation of multiple level interrupts, unlimited subroutine nesting and simplification of many types of data manipulation.

Two Index Registers (IX and IY). The two independent index registers hold a 16-bit base address that is used in indexed addressing modes. In this mode, an index register is used as a base to point to a region in memory from which data is to be stored or retrieved. An additional byte is included in indexed instructions to specify a displacement from this base. This displacement is specified as a two's complement signed integer. This mode of addressing greatly simplifies many types of programs, especially where tables of data are used.

Interrupt Page Address Register (I). The Z80 CPU can be operated in a mode where an indirect call to any memory location can be achieved in response to an interrupt. The I register is used for this purpose to store the high order eight bits of the indirect address while the interrupting device provides the lower eight bits of the address. This feature allows interrupt routines to be dynamically located anywhere in memory with absolute minimal access time to the routine.

Memory Refresh Register (R). The Z80 CPU contains a memory refresh counter to enable dynamic memories to be used with the same ease as static memories. Seven bits of this 8-bit register are automatically incremented after each instruction fetch. The eighth bit will remain as programmed as the result of an LD R, A instruction. The data in the refresh counter is sent out on the lower portion of the address bus along with a refresh control signal while the CPU is decoding and executing the fetched instruction. This mode of refresh is totally transparent to the programmer and does not slow down the CPU operation. The programmer can load the R register for testing purposes, but this register is normally not used by the programmer. During refresh, the contents of the I register are placed on the upper eight bits of the address bus.

1.1.2 Accumulator and Flag Registers

The CPU includes two independent 8-bit accumulators and associated 8-bit flag registers. The accumulator holds the results of 8-bit arithmetic or logical operations while the flag register indicates specific conditions for 8-bit or 16-bit operations, such as indicating whether or not the result of an operation is equal to zero. The programmer selects the accumulator and flag pair with a single exchange instruction so that it is possible to work with either pair.

1.1.3 General Purpose Registers

There are two matched sets of general-purpose registers, each set containing six 8-bit registers that may be used individually as 8-bit registers or as 16-bit register pairs. One set is called BC, DE, and HL while the complementary set is called BC', DE', and HL'. At any one time, the programmer can select either set of registers to work with through a single exchange command for the entire set. In systems where fast interrupt response is required, one set of general-purpose registers and an accumulator/flag register may be reserved for handling this very fast routine. Only a simple exchange command need be executed to go between the routines. This greatly reduces interrupt service time by eliminating the requirement for saving and retrieving register contents in the external stack during interrupt or subroutine processing. These general-purpose registers are used for a wide range of applications by the programmer. They also simplify programing, especially in ROM based systems where little external read/write memory is available.

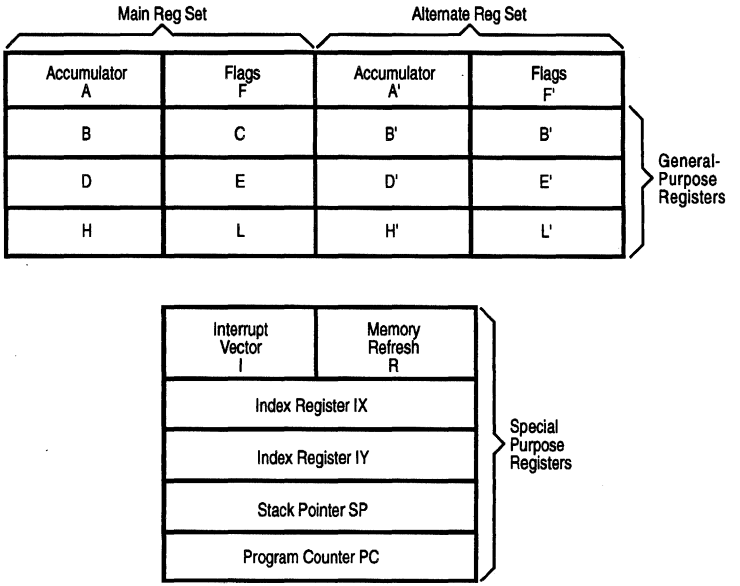


Figure 1-2. Z80 CPU Register Configuration

1.2 ARITHMETIC LOGIC UNIT (ALU)

The 8-bit arithmetic and logical instructions of the CPU are executed in the ALU. Internally the ALU communicates with the registers and the external data bus on the internal data bus. The type of functions performed by the ALU include:

- Add
- Subtract
- Logical AND
- Logical OR
- Logical Exclusive OR
- Compare
- Left or Right Shifts or Rotates (Arithmetic and Logical)
- Increment
- Decrement
- Set Bit
- Reset Bit
- Test bit

1.3 INSTRUCTION REGISTER AND CPU CONTROL

As each instruction is fetched from memory, it is placed in the instruction register and decoded. The control sections performs this function and then generates and supplies all of the control signals necessary to read or write data from or to the registers, control the ALU, and provide all required external control signals.

CHAPTER 2
PIN DESCRIPTION

2.0 INTRODUCTION

The Z80 CPU I/O pins are shown in Figure 2-1 and the function of each is described below.

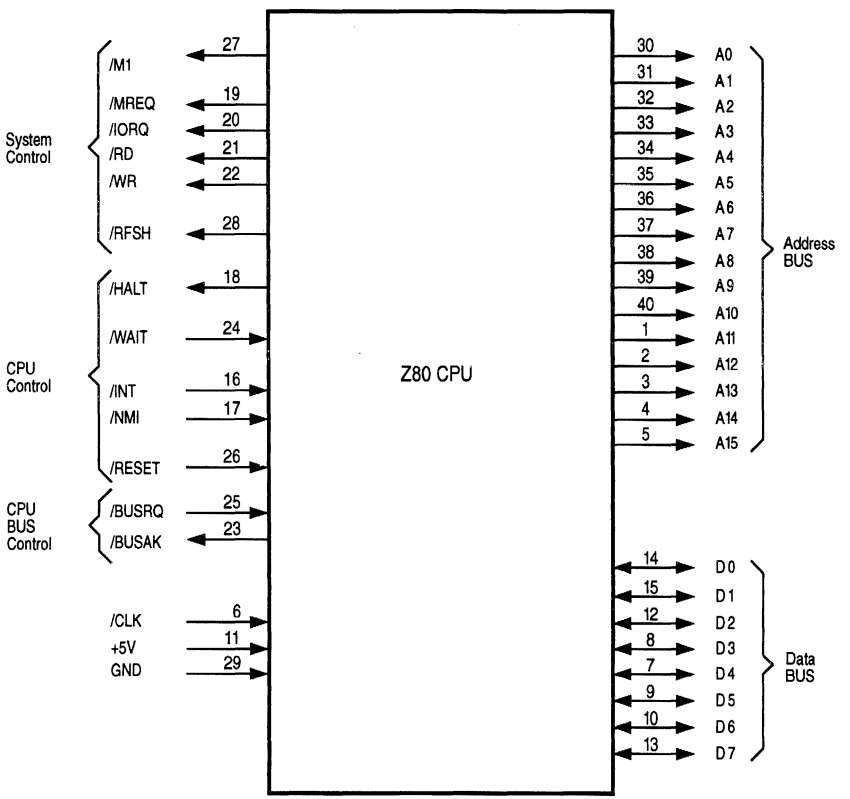


Figure 2.1 Z80 Pin Configuration

2.1 PIN FUNCTIONS

A15-A0 *Address Bus* (output, active High, tri-state), A15-A0 form a 16-bit address bus. The Address Bus provides the address for memory data bus exchanges (up to 64 Kbytes) and for I/O device exchanges.

/BUSACK *Bus Acknowledge* (output, active Low). Bus Acknowledge indicates to the requesting device that the CPU address bus, data bus, and control signals /MREQ, /IORQ, /RD, and /WR have entered their high-impedance states. The external circuitry can now control these lines.

/BUSREQ *Bus Request* (input, active Low). Bus Request has a higher priority than /NMI and is always recognized at the end of the current machine cycle, /BUSREQ forces the CPU address bus, data bus, and control signals /MREQ, /IORQ, RD, and WR to go to a high-impedance state so that other devices can control these lines. /BUSREQ is normally wired-OR and requires an external pull-up for these applications. Extended /BUSREQ periods due to extensive DMA operations can prevent the CPU from properly refreshing dynamic RAMs.

D7-D0 *Data Bus* (input/output, active High, tri-state). D7-D0 constitute an 8-bit bidirectional data bus, used for data exchanges with memory and I/O.

/HALT *HALT State* (output, active Low). /HALT indicates that the CPU has executed a HALT instruction and is awaiting either a non-maskable or a maskable interrupt (with the mask enabled) before operation can resume. During HALT, the CPU executes NOPs to maintain memory refresh.

/INT *Interrupt Request* (input, active Low). Interrupt Request is generated by I/O devices. The CPU honors a request at the end of the current instruction if the internal software-controlled interrupt enable flip-flop (IFF) is enabled. /INT is normally wired-OR and requires an external pull-up for these applications.

/IORQ *Input/Output Request* (output, active Low, tri-state). /IORQ indicates that the lower half of the address bus holds a valid I/O address for an I/O read or write operation. /IORQ is also generated concurrently with /M1 during an interrupt acknowledge cycle to indicate that an interrupt response vector can be placed on the data bus.

/M1 *Machine Cycle One* (output, active Low). /M1, together with /MREQ, indicates that the current machine cycle is the opcode fetch cycle of an instruction execution. /M1, together with /IORQ, indicates an interrupt acknowledge cycle.

/MREQ *Memory Request* (output, active Low, tri-state). /MREQ indicates that the address bus holds a valid address for a memory read of memory write operation.

/NMI *Non-Maskable Interrupt* (input, negative edge-triggered). /NMI has a higher priority than /INT. /NMI is always recognized at the end of the current instruction, independent of the status of the interrupt enable flip-flop, and automatically forces the CPU to restart at location 0066H.

/RD *Read* (output, active Low, tri-state). /RD indicates that the CPU wants to read data from memory or an I/O device. The addressed I/O device or memory should use this signal to gate data onto the CPU data bus.

/RESET *Reset* (input, active Low). /RESET initializes the CPU as follows: it resets the interrupt enable flip-flop, clears the PC and registers I and R, and sets the interrupt status to Mode 0. During reset time, the address and data bus go to a high-impedance state, and all control output signals go to the inactive state. Note that /RESET must be active for a minimum of three full clock cycles before the reset operation is complete.

/RFSH *Refresh* (output, active Low). /RFSH, together with /MREQ, indicates that the lower seven bits of the system's address bus can be used as a refresh address to the system's dynamic memories.

/WAIT *WAIT* (input, active Low). /WAIT indicates to the CPU that the addressed memory or I/O devices are not ready for a data transfer. The CPU continues to enter a /WAIT state as long as this signal is active. Extended /WAIT periods can prevent the CPU from properly refreshing dynamic memory.

/WR *Write* (output, active Low, tri-state). /WR indicates that the CPU data bus holds valid data to be stored at the addressed memory or I/O location.

/CLK *Clock* (input). Single-phase MOS-level clock.

CHAPTER 3
TIMING

3.0 INTRODUCTION

The Z80 CPU executes instructions by stepping through a very precise set of a few basic operations. These include:

- Memory Read or Write
- I/O Device Read or Write
- Interrupt Acknowledge,

All instructions are merely a series of these basic operations. Each of these basic operations can take from three to six clock periods to complete or they can be lengthened to synchronize the CPU to the speed of external devices. The basic clock periods are referred to as T (time) cycles and the basic operations are referred to as M (machine) cycles. Figure 3-1 illustrates how a typical instruction is merely a series of specific M and T cycles. Notice that this instruction consists of three machine cycles (M1, M2, and M3) The first machine cycle of any instruction is a fetch cycle which is four, five, or six T cycles long (unless lengthened by the WAIT signal which will be fully described in the next section). The fetch cycle (M1) is used to fetch the opcode of the next instruction to be executed. Subsequent machine cycles move data between the CPU and memory or I/O devices, and they may have anywhere from three to five T cycles (again they may be lengthened by wait states to synchronize the external devices to the CPU). The following paragraphs describe the timing which occurs within any of the basic machine cycles.

During T2 and every subsequent Tw, the CPU samples the WAIT line with the failing edge of Clock. If the WAIT line is active at this time, another WAIT state will be entered during the following cycle. Using this technique the read can be lengthened to match the access time of any type of memory device.

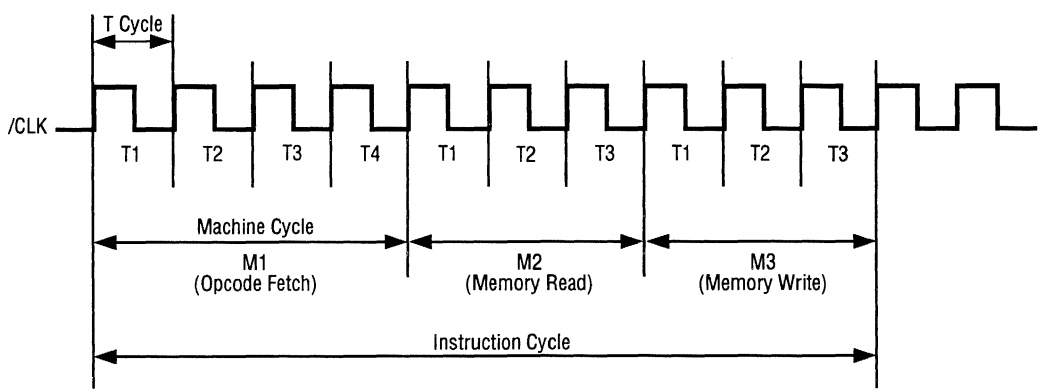


Figure 3-1. Basic CPU Timing Example

3.1 INSTRUCTION FETCH

Figure 3-2 shows the timing during an M1 (opcode fetch) cycle. The PC is placed on the address bus at the beginning of the M1 cycle. One half clock cycle later the /MREQ signal goes active. At this time the address to the memory has had time to stabilize so that the failing edge of /MREQ can be used directly as a chip enable clock to dynamic memories. The /RD line also goes active to indicate that the memory read data should be enabled onto the CPU data bus. The CPU samples the data from the memory on the data bus with the rising edge of the clock of state T3 and this same edge is used by the CPU to turn off the /RD and /MREQ signals. Thus, the data has already been sampled by the CPU before the /RD signal becomes inactive. Clock state T3 and T4 of a fetch cycle are used to refresh dynamic memories. (The CPU uses this time to decode and execute the fetched instruction so that no other operation could be performed at this time.)

During T3 and T4, the lower seven bits of the address bus contain a memory refresh address and the /RFSH signal becomes active to indicate that a refresh read of all dynamic memories should be accomplished. Notice that an /RD signal is not generated during refresh time to prevent data from different memory segments from being gated onto the data bus. The /MREQ signal during refresh time should be used to perform a refresh read of all memory elements. The refresh signal can not be used by itself since the refresh address is only guaranteed to be stable during /MREQ time.

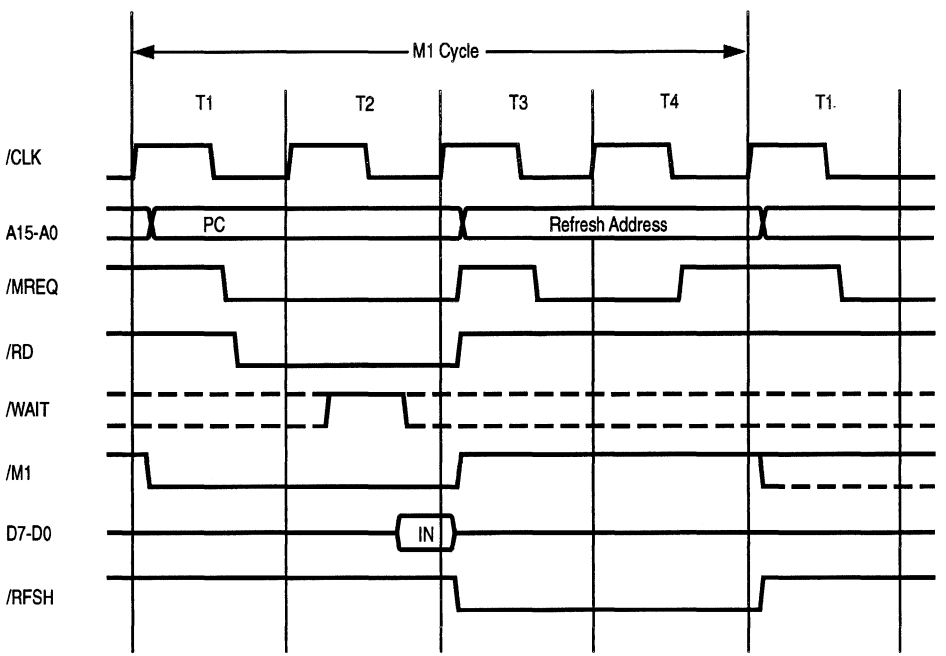


Figure 3-2. Instruction Opcode Fetch

3.2 MEMORY READ OR WRITE

Figure 3-3 illustrates the timing of memory read or write cycles other than an opcode fetch cycle. These cycles are generally three clock periods long unless wait states are requested by the memory through the /WAIT signal. The /MREQ signal and the /RD signal are used the same as in the fetch cycle. In the case of a memory write cycle, the /MREQ also becomes active when the address bus is stable so that it call be used directly as a chip enable for dynamic memories. The /WR line is active when data on the data bus is stable so that it can be used directly as a R/W pulse to virtually any type of semiconductor memory. Furthermore, the /WR signal goes inactive one-half T state before the address and data bus contents are changed so that the overlap requirements for virtually any type of semiconductor memory type will be met.

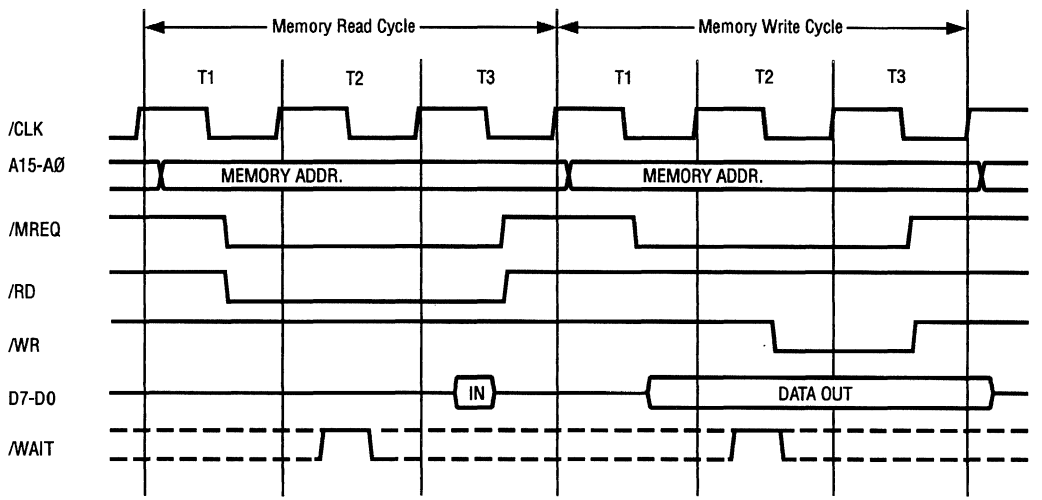


Figure 3-3. Memory Read or Write Cycle

3.3 INPUT OR OUTPUT CYCLES

Figure 3-4 illustrates an I/O read or I/O write operation. Notice that during I/O operations a single wait state is automatically inserted. The reason is that during I/O operations, the time from when the /IORQ signal goes active until the CPU must sample the /WAIT line is very short, and without this extra state, sufficient time does not exist for an I/O port to decode its address and activate the /WAIT line if a wait is required. Also, without this wait state, it is difficult to design MOS I/O devices that can operate at full CPU speed. During this wait state time, the /WAIT request signal is sampled.

During a read I/O operation, the /RD line is used to enable the addressed port onto the data bus just as in the case of a memory read. For I/O write operations, the /WR line is used as a clock to the I/O port.

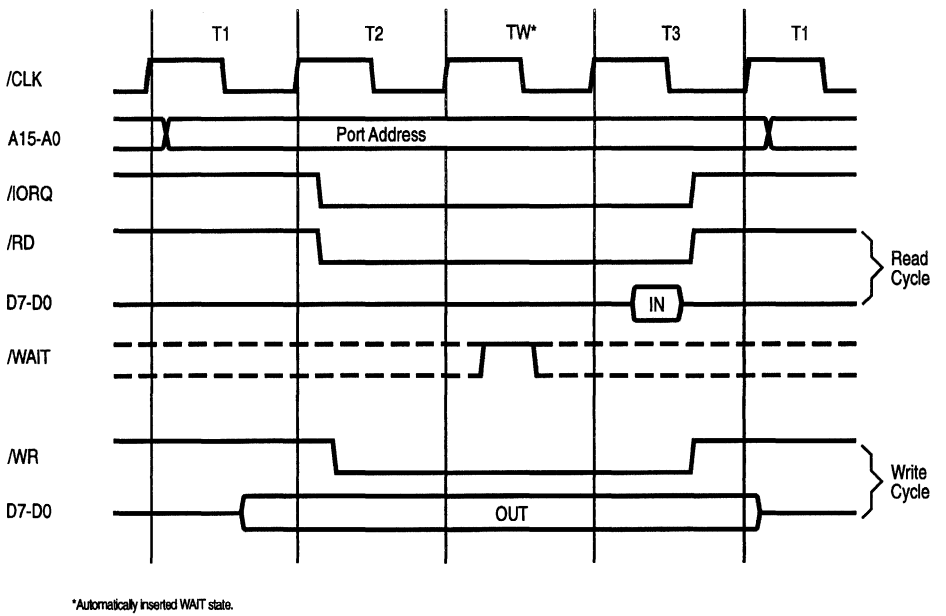


Figure 3-4. Input or Output Cycles

3.4 BUS REQUEST/ACKNOWLEDGE CYCLE

Figure 3-5 illustrates the timing for a Bus Request/Acknowledge cycle. The /BUSREQ signal is sampled by the CPU with the rising edge of the last clock period of any machine cycle. If the /BUSREQ signal is active, the CPU will set its address, data, and tri-state control signals to the high-impedance state with the rising edge of the next clock pulse. At that time, any external device can control the buses to transfer data between memory and I/O devices. (This is generally known as Direct Memory Access [DMA] using cycle stealing.) The maximum time for the CPU to respond to a bus request is the length of a machine cycle and the external controller can maintain control of the bus for as many clock cycles as is desired. Note, however, that if very long DMA cycles are used, and dynamic memoriesies are being used, the external controller must also perform the refresh function. This situation only occurs if very large blocks of data are transferred under DMA control. Also note that during a bus request cycle, the CPU cannot be interrupted by either an /NMI or an /INT signal.

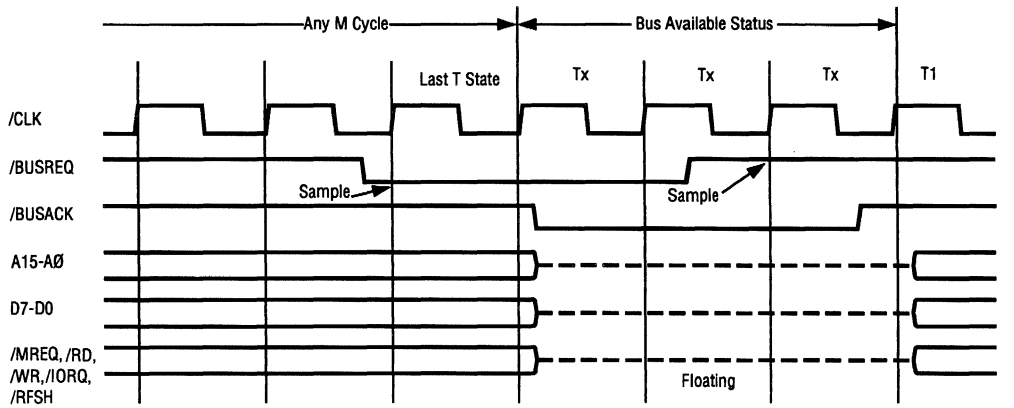


Figure 3-5. Bus Request/Acknowledge Cycle

3.5 INTERRUPT REQUEST/ACKNOWLEDGE CYCLE

Figure 3-6 illustrates the timing associated with an interrupt cycle. The interrupt signal (/INT) is sampled by the CPU with the rising edge of the last clock at the end of any instruction. The signal will not be accepted if the internal CPU software controlled interrupt enable flip-flop is not set or if the /BUSREQ signal is active. When the signal is accepted, a special M1 cycle is generated. During this special M1 cycle, the /IORQ signal becomes active (instead of the normal /MREQ) to indicate that the interrupting device can place an 8-bit vector on the data bus. Notice that two wait states are automatically added to this cycle. These states are added so that a ripple priority interrupt scheme can be easily implemented. The two wait states allow sufficient time for the ripple signals to stabilize and identify which I/O device must insert the response vector. Refer to Chapter 6 for details on how the interrupt response vector is utilized by the CPU.

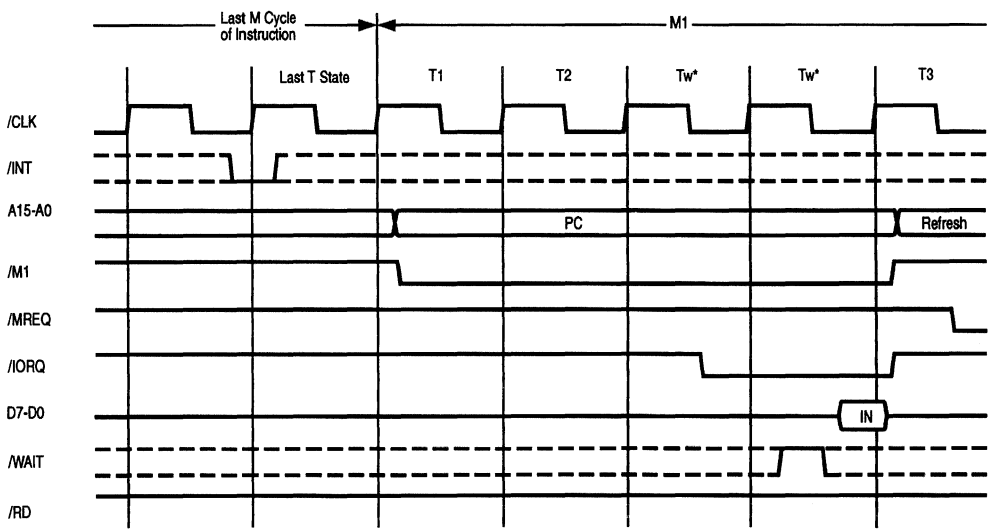


Figure 3-6. Interrupt Request/Acknowledge Cycle

3.6 NON-MASKABLE INTERRUPT RESPONSE

Figure 3-7 illustrates the request/acknowledge cycle for the non-maskable interrupt. This signal is sampled at the same time as the interrupt line, but this line has priority over the normal interrupt and it can not be disabled under software control. Its usual function is to provide immediate response to important signals such as an impending power failure. The CPU response to a non-maskable interrupt is similar to a normal memory read operation. The only difference being that the content of the data bus is ignored while the processor automatically stores the PC in the external stack and jumps to location 0066H. The service routine for the non-maskable interrupt must begin at this location if this interrupt is used.

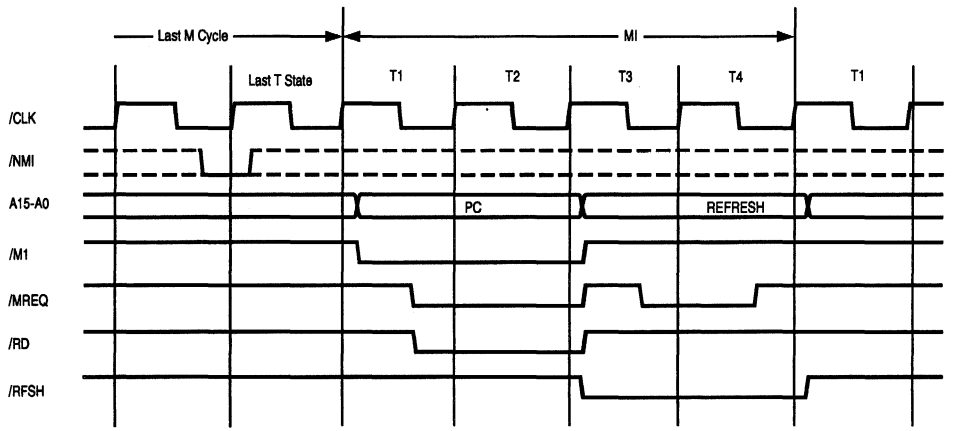


Figure 3-7. Non-Maskable Interrupt Request Operation

3.7 HALT EXIT

Whenever a software HALT instruction is executed, the CPU begins executing NOPs until an interrupt is received (either a non-maskable or a maskable interrupt while the interrupt flip flop is enabled). The two interrupt lines are sampled with the rising clock edge during each T4 state as shown in Figure 3-8. If a non-maskable interrupt has been received or a maskable interrupt has been received and the interrupt enable flip-flop is set, then the HALT state will be exited on the next rising clock edge. The following cycle will then be an interrupt acknowledge cycle corresponding to the type of interrupt that was received. If both are received at this time, then the non-maskable one will be acknowledged since it has highest priority. The purpose of executing NOP instructions while in the HALT state is to keep the memory refresh signals active. Each cycle in the HALT state is a normal M1 (fetch) cycle except that the data received from the memory is ignored and a NOP instruction is forced internally to the CPU. The HALT acknowledge signal is active during this time to indicate that the processor is in the HALT state.

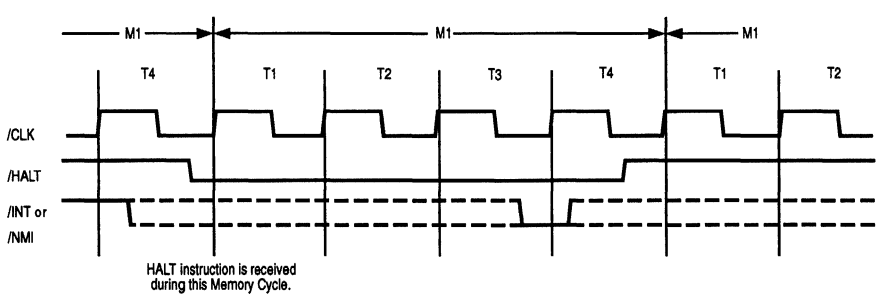


Figure 3-8. HALT Exit

3.8 POWER-DOWN ACKNOWLEDGE CYCLE

When the clock input to the CMOS Z80 CPU is stopped at either a High or Low level, the CMOS Z80 CPU stops its operation and maintains all registers and control signals. However, ICC2 (standby supply current) is guaranteed only when the system clock is stopped at a Low level during T4 of the machine cycle following the execution of the HALT instruction. The timing diagram for the power-down function, when implemented with the HALT instruction, is shown in Figure 3-9.

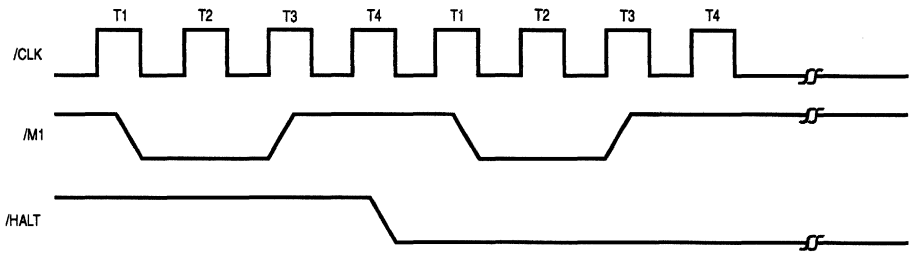


Figure 3-9. Power-Down Acknowledge

3.9 POWER-DOWN RELEASE CYCLE

The system clock must be supplied to the CMOS Z80 CPU to release the power-down state. When the system clock is supplied to the CLK input, the CMOS Z80 CPU restarts operations from the point at which the power-down state was implemented. The timing diagrams for the release from power-down mode are shown in Figures 3-10 to 3-12.

When the HALT instruction is executed to enter the power-down state, the CMOS Z80 CPU will also enter the HALT state. An interrupt signal (either /NMI or /INT) or a /RESET signal must be applied to the CPU after the system clock is supplied in order to release the power-down state.

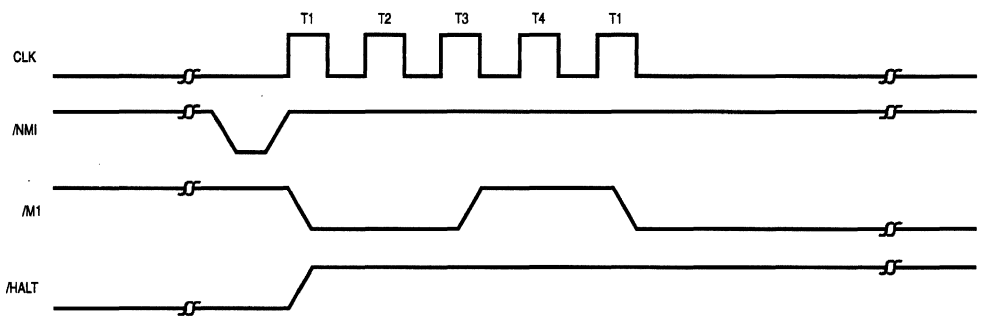


Figure 3-10. Power-Down Release Cycle No. 1

3.9 POWER DOWN RELEASE CYCLE (Continued)

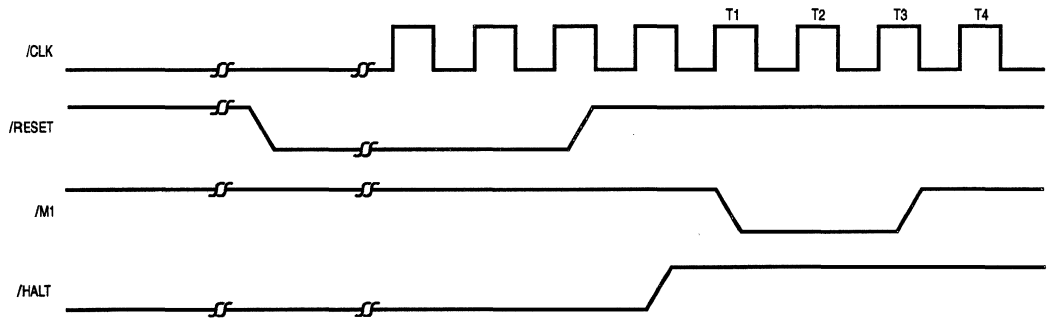


Figure 3-11. Power-Down Release Cycle No. 2

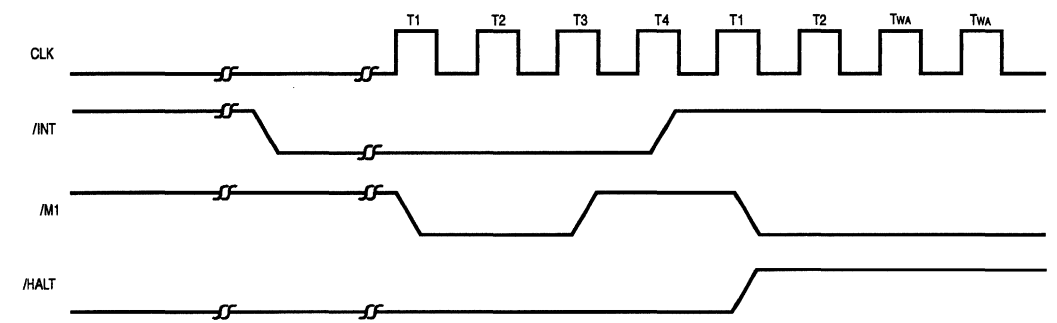


Figure 3-12. Power-Down Release Cycle No. 3

CHAPTER 4
Z80 CPU INSTRUCTION SET

4.0 INTRODUCTION

The Z80 CPU can execute 158 different instruction types including all 78 of the 8080A CPU. The instructions can be broken down into the following major groups:

- Load and Exchange
- Block Transfer and Search
- Arithmetic and Logical
- Rotate and Shift
- Bit Manipulation (Set, Reset, Test)
- Jump, Call, and Return
- Input/Output
- Basic CPU Control

4.1 INSTRUCTION TYPES

The load instructions move data internally between CPU registers or between CPU registers and external memory. All of these instructions must specify a source location from which the data is to be moved and a destination location. The source location is not altered by a load instruction. Examples of load group instructions include moves between any of the general-purpose registers such as move the data to register B from register C. This group also includes load-immediate to any CPU register or to any external memory location. Other types of load instructions allow transfer between CPU registers and memory locations. The exchange instructions can trade the contents of two registers.

A unique set of block transfer instructions is provided in the Z80. With a single instruction, a block of memory of any size can be moved to any other location in memory. This set of block moves is extremely valuable when large strings of data must be processed. The Z80 block search instructions are also valuable for this type of processing. With a single instruction, a block of external memory of any desired length can be searched for any 8-bit character. Once the character is found or the end of the block is reached, the instruction automatically terminates. Both the block transfer and the block search instructions can be interrupted during their execution so as to not occupy the CPU for long periods of time.

The arithmetic and logical instructions operate on data stored in the accumulator and other general-purpose CPU registers or external memory locations. The results of the operations are placed in the accumulator and the appropriate flags are set according to the result of the operation. An example of an arithmetic operation is adding the accumulator to the contents of an external memory location. The results of the addition are placed in the accumulator. This group also includes 16-bit addition and subtraction between 16-bit CPU registers.

The rotate and shift group allows any register or any memory location to be rotated right or left, with or without carry either arithmetic or logical. Also, a digit in the accumulator can be rotated right or left with two digits in any memory location.

The bit manipulation instructions allow any bit in the accumulator, any general-purpose register, or any external memory location to be set, reset or tested with a single instruction. For example, the most significant bit of register H can be reset. This group is especially useful in control applications and for controlling software flags in general-purpose programming.

The jump, call, and return instructions are used to transfer between various locations in the user's program. This group uses several different techniques for obtaining the new program counter address from specific external memory locations. A unique type of call is the restart instruction. This instruction actually contains the new address as a part of the 8-bit opcode. This is possible since only eight separate addresses located in page zero of the external memory may be specified. Program jumps may also be achieved by loading register HL, IX, or IY directly into the PC, thus allowing the jump address to be a complex function of the routine being executed.

4.1 INSTRUCTION TYPES (Continued)

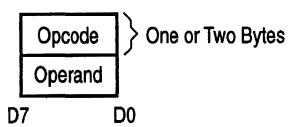
The input/output group of instructions in the Z80 allow for a wide range of transfers between external memory locations or the general-purpose CPU registers, and the external I/O devices. In each case, the port number is provided on the lower eight bits of the address bus during any I/O transaction. One instruction allows this port number to be specified by the second byte of the instruction while other Z80 instructions allow it to be specified as the content of the C register. One major advantage of using the C register as a pointer to the I/O device is that it allows different I/O ports to share common software driver routines. This is not possible when the address is part of the opcode if the routines are stored in ROM. Another feature of these input instructions is that they set the flag register automatically so that additional operations are not required to determine the state of the input data (for example its parity). The Z80 CPU includes single instructions that can move blocks of data (up to 256 bytes) automatically to or from any I/O port directly to any memory location. In conjunction with the dual set of general-purpose registers, these instructions provide for fast I/O block transfer rates. The value of this I/O instruction set is demonstrated by the fact that the Z80 CPU can provide all required floppy disk formatting (i.e., the CPU provides the preamble, address, data, and enables the CRC codes) on double density floppy disk drives on an interrupt driven basis.

Finally, the basic CPU control instructions allow various options and modes. This group includes instructions such as setting or resetting the interrupt enable flip-flop or setting the mode of interrupt response.

4.1 ADDRESSING MODES

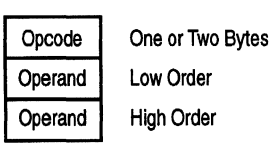
Most of the Z80 instructions operate on data stored in internal CPU registers, external memory, or in the I/O ports. Addressing refers to how the address of this data is generated in each instruction. This section gives a brief summary of the types of addressing used in the Z80 while subsequent sections detail the type of addressing available for each instruction group.

Immediate. In this mode of addressing the byte following the opcode in memory contains the actual operand.



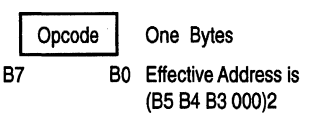
Examples of this type of instruction would be to load the accumulator with a constant, where the constant is the byte immediately following the opcode.

Immediate Extended. This mode is merely an extension of immediate addressing in that the two bytes following the opcodes are the operand.

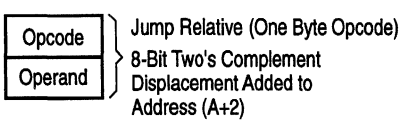


Examples of this type of instruction would be to load the HL register pair (16-bit register) with 16 bits (two bytes) of data.

Modified Page Zero Addressing. The Z80 has a special single byte CALL instruction to any of eight locations in page zero of memory. This instruction (which is referred to as a restart) sets the PC to an effective address in page zero. The value of this instruction is that it allows a single byte to specify a complete 16-bit address where commonly called subroutines are located, thus saving memory space.



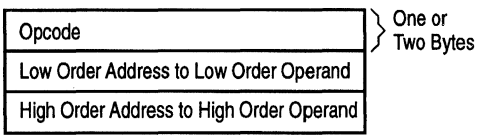
Relative Addressing. Relative addressing uses one byte of data following the opcode to specify a displacement from the existing program to which a program jump can occur. This displacement is a signed two's complement number that is added to the address of the opcode, of the following instruction.



The value of relative addressing is that it allows jumps to nearby locations while only requiring two bytes of memory space. For most programs, relative jumps are by far the most prevalent type of jump due to the proximity of related program segments. Thus, these instructions can Significantly reduce memory space requirements. The signed displacement can range between +127 and –128 from

A+2. This allows for a total displacement of +129 to −126 from the jump relative opcode address. Another major advantage is that it allows for relocatable code.

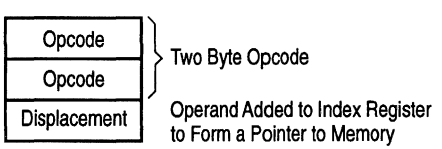
Extended Addressing. Extended Addressing provides for two bytes (16 bits) of address to be included in the instruction. This data can be an address to which a program can jump or it can be an address where an operand is located.



Extended addressing is required for a program to jump from any location in memory to any other location, or load and store data in any memory location.

When extended addressing is used to specify the source or destination address of an operand, the notation (nn) will be used to indicate the content of memory at nn, where nn is the 16-bit address specified in the instruction. This means that the two bytes of address nn are used as a pointer to a memory location. The use of the parentheses always means that the value enclosed within them is used as a pointer to a memory location. For example, (1200) refers to the contents of memory at location 1200.

Indexed Addressing. In this type of addressing, the byte of data following the opcode contains a displacement which is added to one of the two index registers (the opcode specifies which index register is used) to form a pointer to memory. The contents of the index register are not altered by this operation.



An example of an indexed instruction would be to load the contents of the memory location (Index Register + Displacement) into the accumulator. The displacement is a signed two's complement number. Indexed addressing greatly simplifies programs using tables of data since the index register can point to the start of any table. Two index registers are provided since very often operations require two or more tables. Indexed addressing also allows for relocatable code.

The two index registers in the Z80 are referred to as IX and IY. To indicate indexed addressing the notation:

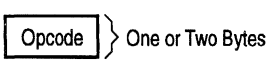
(IX+d) or (IY+d)

is used. Here d is the displacement specified after the opcode. The parentheses indicate that this value is used as a pointer to external memory.

Register Addressing. Many of the Z80 opcodes contain bits of information that specify which CPU register is to be used for an operation. An example of register addressing would be to load the data in register B into register C.

Implied Addressing. Implied addressing refers to operations where the opcode automatically implies one or more CPU registers as containing the operands. An example is. the set of arithmetic operations where the accumulator is always implied to be the destination of the results.

Register Indirect Addressing. This type of addressing specifies a 16-bit CPU register pair (such as HL) to be used as a pointer to any location in memory. This type of instruction is very powerful and it is used in a wide range of applications.



An example of this type of instruction would be to load the accumulator with the data in the memory location pointed to by the HL register contents. Indexed addressing is actually a form of register indirect addressing except that a displacement is added with indexed addressing. Register indirect addressing allows for very powerful but simple to implement memory accesses. The block move and search commands in the Z80 are extensions of this type of addressing where automatic register incrementing, decrementing, and comparing has been added. The notation for indicating register indirect addressing is to put parentheses around the name of the register that is to be used as the pointer. For example, the symbol

(HL)

specifies that the contents of the HL register are to be used as a pointer to a memory location. Often register indirect addressing is used to specify 16-bit operands. In this case, the register contents point to the lower order portion of the operand while the register contents are automatically incremented to obtain the upper portion of the operand.

4.1 ADDRESSING MODES (Continued)

Bit Addressing. The Z80 contains a large number of bit set, reset, and test instructions. These instructions allow any memory location or CPU register to be specified for a bit operation through one of three previous addressing modes (register, register indirect, and indexed) while three bits in the opcode specify which of the eight bits is to be manipulated.

4.1.1 Addressing Mode Combinations

Many instructions include more than one operand (such as arithmetic instructions or loads). In these cases, two types of addressing may be employed. For example, load can use immediate addressing to specify the source and register indirect or indexed addressing to, specify the destination.

4.2 INSTRUCTION OPCODES

This section describes each of the Z80 instructions and provides tables listing the opcodes for every instruction. In each of these tables, the opcodes in shaded areas are identical to those offered in the 8080A CPU. Also shown is the assembly language mnemonic that is used for each instruction. All instruction opcodes are listed in hexadecimal notation. Single byte opcodes require two hex characters while double byte opcodes require four hex characters. The conversion from hex to binary is repeated here for convenience.

Table 4.1. Hex to Binary Conversion Table

Hex		Binary		Decimal
0	=	0000	=	0
1	=	0001	=	1
2	=	0010	=	2
3	=	0011	=	3
4	=	0100	=	4
5	=	0101	=	5
6	=	0110	=	6
7	=	0111	=	7
8	=	1000	=	8
9	=	1001	=	9
A	=	1010	=	10
B	=	1011	=	11
C	=	1100	=	12
D	=	1101	=	13
E	=	1110	=	14
F	=	1111	=	15

The Z80 instruction mnemonics consist of an opcode and zero, one, or two operands. Instructions in which the operand is implied have no operand. Instructions which have only one logical operand of those in which one operand is invariant (such as the Logical OR instruction) are represented by a one operand mnemonic. Instructions which may have two varying operands are represented by two operand mnemonics.

4.2.1 Load and Exchange

Figure 4-1 defines the opcode for all of the 8-bit load instructions implemented in the Z80 CPU. Also shown in this table is the type of addressing used for each instruction. The source of the data is found on the top horizontal row while the destination is specified by the left hand column. For example, load register C from register B uses the opcode 48H. In all of the figures, the opcode is specified in hexadecimal notation and the 48H (0100 1000 binary) code is fetched by the CPU from the external memory during M1 time, decoded and then the register transfer is automatically performed by the CPU.

The assembly language mnemonic for this entire group is LD, followed by the destination followed by the source (LD DEST, SOURCE). Note that several combinations of addressing modes are possible. For example, the source may use register addressing and the destination may be register indirect; such as load the memory location pointed to by register HL with the contents of register D. The opcode for this operation would be 72. The mnemonic for this load instruction would be as follows:

LD (HL), D

The parentheses around the HL means that the contents of HL are used as a pointer to a memory location. In all Z80 load instruction mnemonics, the destination is always listed first, with the source following. The Z80 assembly language has been defined for ease of programming. Every instruction is self documenting and programs written in Z80 language are easy to maintain.

Note in Figure 4-1, some load opcodes that are available in the Z80 use two bytes. This is an efficient method of memory utilization since 8-, 16-, 24-, or 32-bit instructions are implemented in the Z80. Thus, often utilized instructions such as arithmetic or logical operations are only eight bits which results in better memory utilization than is achieved with fixed instruction sizes such as 16 bits.

DESTINATION		SOURCE: IMPLIED I	IMPLIED R	REGISTER A	REGISTER B	REGISTER C	REGISTER D	REGISTER E	REGISTER F	REGISTER L	REG INDIRECT (HL)	REG INDIRECT (BC)	REG INDIRECT (DE)	11INDEXED (IX+d)	11INDEXED (IY+d)	EXT. ADDR. (nn)	IMME. n
REGISTER	A	ED 57	ED 5F	7F	78	79	7A	7B	7C	7D	7E	0A	1A	DD 7E d	DD 7E d	DD 7E d	DD 7E d
	B			47	40	41	42	43	44	45	46			DD 46 d	FD 46 d		DD 7E d
	C			4F	48	49	4A	4B	4C	4D	4E			DD 4E d	DD 4E d		DD 7E d
	D			57	50	51	52	53	54	55	56			DD 56 d	FD 56 d		DD 7E d
	E			5F	58	59	5A	5B	5C	5D	5E			DD 5E d	FD 5E d		DD 7E d
	H			67	60	61	62	63	64	65	66			DD 66 d	FD 66 d		DD 7E d
	L			6F	68	69	6A	6B	6C	6D	6E			DD 6E d	FD 6E d		DD 7E d
REG INDIRECT	(HL)			77	70	71	72	73	74	75							DD 7E d
	(BC)			02													
	(DE)			12													
INDEXED	(IX+d			DD 77 d	DD 70 d	DD 71 d	DD 72 d	DD 73 d	DD 74 d	DD 75 d							DD 36 d n
	(IY+d			FD 77 d	FD 70 d	FD 71 d	FD 72 d	FD 73 d	FD 74 d	FD 75 d							FD 36 d n
EXT. ADDR	(nn)			32 n n						FD 75 d							
IMPLIED	I			ED 47													
	R			ED 4F													

Figure 4-1. 8-Bit Load Group 'LD'

All load instructions using indexed addressing for either the source or destination location actually use three bytes of memory with the third byte being the displacement d. For example, a load register E with the operand pointed to by IX with an offset of +8 would be written:

LD E, (IX + 8)

The instruction sequence for this in memory would be:

Address A	DD	Opcode
A + 1	5E	
A + 2	08	Displacement Operand

The two extended addressing instructions are also three byte instructions. For example, the instruction to load the accumulator with the operand in memory location 6F32H would be written:

LD A, (6F 32H)

and its instruction sequence would be:

Address A	3A	Opcode
A + 1	32	Low Order Address
A + 2	6F	High Order Address

Notice that the low order portion of the address is always the first operand.

The load immediate instructions for the general-purpose 8-bit registers are two-byte instructions. The instruction load register H with the value 36H would be written:

LD H, 36H

and its sequence would be:

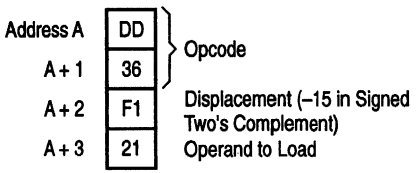
Address A	26	Opcode
A + 1	36	Operand

4.2 INSTRUCTION OPCODES (Continued)

Loading a memory location using indexed addressing for the destination and immediate addressing for the source requires four bytes. For example:

LD (IX – 15), 21H

would appear as:



Notice that with any indexed addressing the displacement always follows directly after the opcode.

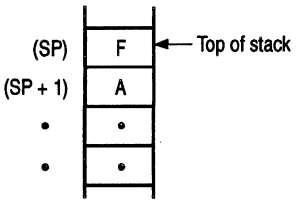
Figure 4-2 specifies the 16-bit load operations. This table is very similar to the previous one. Notice that the extended addressing capability covers all register pairs. Also notice that register indirect operations specifying the stack pointer are the PUSH and POP instructions. The mnemonic for these instructions is "PUSH" and "POP." These differ from other 16-bit loads in that the stack pointer is automatically decremented and incremented as each byte is pushed onto or popped from the stack respectively. For example, the instruction:

PUSH AF

is a single byte instruction with the opcode of F5H. When this instruction is executed the following sequence is generated:

Decrement SP
LD (SP), A
Decrement SP
LD (SP), F

Thus, the external stack now appears as follows:



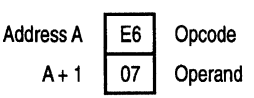
The POP instruction is the exact reverse of a PUSH. Notice that all PUSH and POP instructions utilize a 16-bit operand and the high order byte is always pushed first and popped last. That is a:

PUSH BC	is PUSH B then C
PUSH DE	is PUSH D then E
PUSH HL	is PUSH H then L
POP HL	is POP L then H

The instruction using extended immediate addressing for the source obviously requires two bytes of data following the opcode. For example:

LD DE, 0659H

will be:



In all extended immediate or extended addressing modes, the low order byte always appears first after the opcode.

Figure 4-3 lists the 16-bit exchange instructions implemented in the Z80. Opcode 08H allows the programmer to switch between the two pairs of accumulator flag registers while D9H allows the programmer to switch between the duplicate set of six general-purpose registers. These opcodes are only one byte in length to absolutely minimize the time necessary to perform the exchange so that the duplicate banks can be used to effect very fast interrupt response times.

A

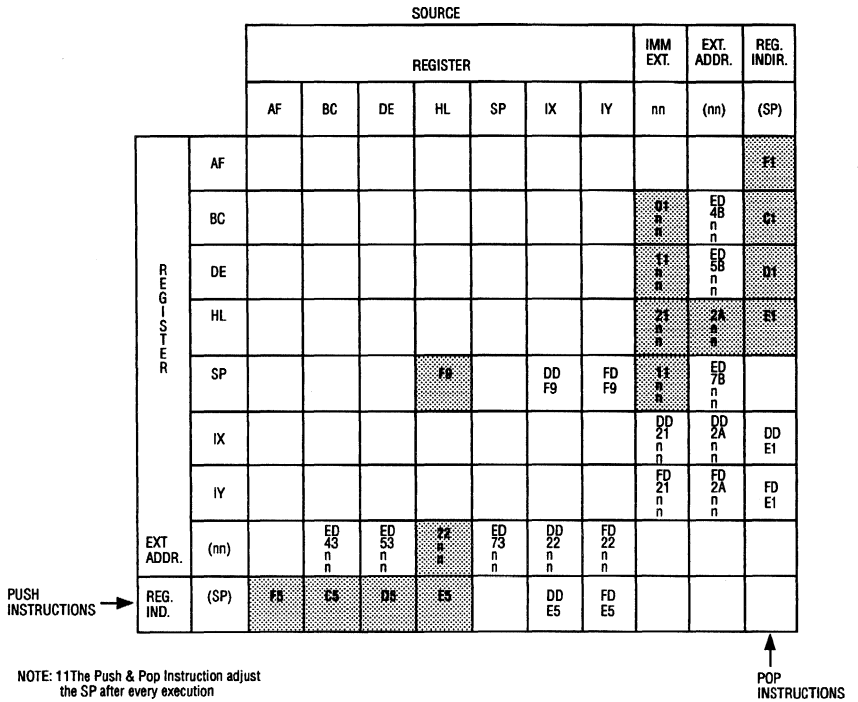


Figure 4-2. 16-Bit Load Group LD, PUSH and POP

		IMPLIED ADDRESSING				
		AF'	BC', DE' & HL'	HL	IX	IY
IMPLIED	AF	08				
	BC, DE & HL		D9			
	DE			EB		
REG. INDIR.	(SP)			E3	DD EE	FD EE

Figure 4-3. Exchanges EX and EXX

4.2.2 Block Transfer and Search

Figure 4-4 lists the extremely powerful block transfer instructions. All of these instructions operate with three registers.

HL points to the source location.
DE points to the destination location.
BC is a byte counter.

After the programmer has initialized these three registers, any of these four instructions may be used. The LDI (Load and Increment) instruction moves one byte from the location pointed to by HL to the location pointed to by DE. Register pairs HL and DE are then automatically incremented and are ready to point to the following locations. The byte counter (register pair BC) is also decremented at this time. This instruction is valuable when blocks of data must be moved but other types of processing are required between each move. The LDIR (Load, Increment and Repeat) instruction is an extension of the LDI instruction. The same load and increment operation is repeated until the byte counter reaches the count of zero. Thus, this single instruction can move any block of data from one location to any other.

Note that since 16-bit registers are used, the size of the block can be up to 64 Kbytes (1K = 1024) long and it can be moved from any location in memory to any other location. Furthermore, the blocks can be overlapping since there are absolutely no constraints on the data that is used in the three register pairs.

The LDD and LDDR instructions are very similar to the LDI and LDIR. The only difference is that register pairs HL and DE are decremented after every move so that a block transfer starts from the highest address of the designated block rather than the lowest.

Figure 4-5 specifies the opcodes for the four block search instructions. The first, CPI (Compare and Increment) compares the data in the accumulator, with the contents of the memory location pointed to by register HL. The result of the compare is stored in one of the flag bits (see section 5.2 for a detailed explanation of the flag operations) and the HL register pair is then incremented and the byte counter (register pair BC) is decremented.

The instruction CPIR is merely an extension of the CPI instruction in which the compare is repeated until either a match is found or the byte counter (register pair BC) becomes zero. Thus, this single instruction can search the entire memory for any 8-bit character.

The CPD (Compare and Decrement) and CPDR (Compare, Decrement, and Repeat) are similar instructions, their only difference being that they decrement HL after every compare so that they search the memory in the opposite direction. (The search is started at the highest location in the memory block.)

It should be emphasized again that these block transfer and compare instructions are extremely powerful in string manipulation applications.

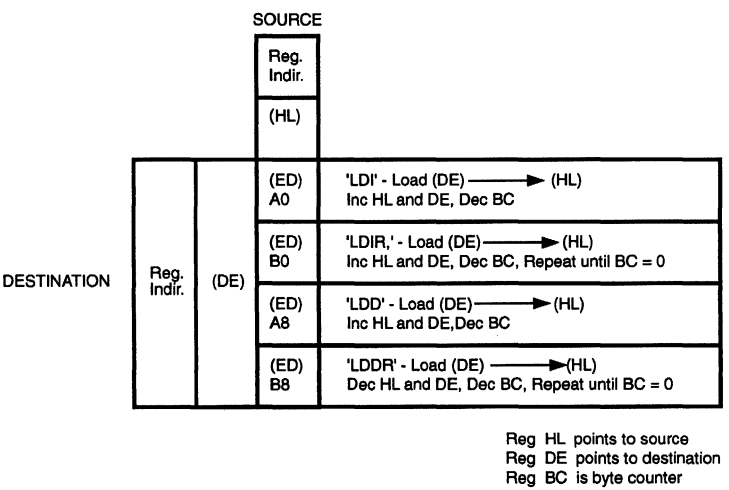


Figure 4-4. Block Transfer Group

SEARCH LOCATION	
Reg. Indir.	
(HL)	
(ED) A1	'CPI' Inc HL, Dec BC
(ED) B1	'CPRI', Inc HL, Dec BC Repeat untill BC = 0 or find match
(ED) A9	'CPD' Dec HL and BC
(ED) B9	'CPDR' Dec HL and BC Repeat untill BC = 0 or find match

HL points to location in memory to be complared with accumulator contents
BC is byte counter

Figure 4-5. Block Search Group

4.2.3 Arithmetic and Logical

Figure 4-6 lists all of the 8-bit arithmetic operations that can be performed with the accumulator, also listed are the increment (INC) and decrement (DEC) instructions. In all of these instructions, except INC and DEC, the specified 8-bit operation is performed between the data in the accumulator and the source data specified in the figure. The result of the operation is placed in the accumulator with the exception of compare (CP) that leaves the accumulator unaffected. All of these operations affect the flag register as a result of the specified operation. (Section 5.2 provides all of the details on how the flags are affected by any instruction type.) INC and DEC instructions specify a register or a memory location as both source and destination of the result. When the source operand is addressed using the index registers, the displacement must follow directly. With immediate addressing the actual operand will follow directly. For example, the instruction:

AND 07H

would appear as:

Address A	E6	Opcode
A + 1	07	Operand

Assuming that the accumulator contained the value F3H, the result of 03H would be placed in the accumulator:

Accumulator before operation	1111 0011 = F3H
Operand	0000 0111 = 07H
Result to Accumulator	0000 0011 = 03H

The Add instruction (ADD) performs a binary add between the data in the source location and the data in the accumulator. The Subtract (SUB) does a binary subtraction. When the Add with Carry is specified (ADC) or the Subtract with Carry (SBC), then the Carry flag is also added or subtracted respectively. The flags and decimal adjust instruction (DAA) in the Z80 (fully described in Chapter 5.2) allow arithmetic operations for:

- Multiprecision packed BCD numbers.
- Multiprecision signed or unsigned binary numbers.
- Multiprecision two's complement signed numbers.

Other instructions in this group are logical and (AND), logical or (OR), exclusive or (XOR), and compare (CP).

There are five general-purpose arithmetic instructions that operate on the accumulator or carry flag. These five are listed in Figure 4-7. The decimal adjust instruction can adjust for subtraction as well as addition, thus making BCD arithmetic operations simple. Note that to allow for this operation the flag N is used. This flag is set if the last arithmetic operation was a subtract. The negate accumulator (NEG) instruction forms the two's complement of the number in the accumulator. Finally, notice that a reset carry instruction is not included in the Z80 since this operation can be easily achieved through other instructions such as a logical AND of the accumulator with itself.

Figure 4-8 lists all of the 16-bit arithmetic operations between 16-bit registers. There are five groups of instructions including add with carry and subtract with carry. ADC and SBC affect all of the flags. These two groups simplify address calculation operations or other 16-bit arithmetic operations.

4.2 INSTRUCTION OPCODES (Continued)

SOURCE

	REGISTER ADDRESSING							REG. INDIR.	INDEXED		IMMED.
	A	B	C	D	E	H	L	(HL)	(IX+d)	(IT+d)	n
'ADD'	87	80	81	82	83	84	85	86	DD 86 d	FD 86 d	C6 n
ADD w CARRY 'ACC'	8F	88	89	8A	8B	8C	8D	8E	DD 8E d	FD 8E d	CE n
SUBTRACT 'SUB'	97	90	91	92	93	94	95	96	DD 96 d	FD 96 d	D6 n
SUB w CARRY 'SBC'	9F	98	99	9A	9B	9C	9D	9E	DD 9E d	FD 9E d	DE n
'ADD'	AF	A0	A1	A2	A3	A4	A5	A6	DD A6 d	FD A6 d	E6 n
'XOR'	AF	A8	A9	AA	AB	AC	AD	AE	DD AE d	FD AE d	EE n
'OR'	B7	B0	B1	B2	B3	B4	B5	B6	DD B6 d	FD B6 d	F6 n
COMPARE 'CP'	BF	B8	B9	BA	BB	BC	BD	BE	DD BE d	FD BE d	FE n
INCREMENT 'INC'	3C	04	0C	14	1C	24	2C	34	DD 34 d	FD 34 d	
DECREMENT 'DEC'	3D	05	0D	15	1D	25	2D	36	DD 35 d	FD 35 d	

Figure 4-6. 8-Bit Arithmetic and Logic

Decimal Adjust Acc, 'DAA'	27
Complement Acc, "CPL'	2F
Negate Acc, 'NEG' (2's complement	ED 44
Complement Carry Flag, 'CCF'	2F
Set Carry Flag, 'SCF'	37

Figure 4-7. General-Purpose AF Operation

		SOURCE					
		BC	DE	HL	SP	IX	IY
DESTINATION	HL	09	19	29	39		
	'ADD' IX	DD 09	DD 19		DD 39	DD 29	
	IY	FD 09	FD 19		FD 39		FD 29
	ADD WITH CARRY AND SET FLAGS 'ADC' HL	ED 4A	ED 5A	ED 6A	ED 7A		
	SUB WITH CARRY AND SET FLAGS 'SBC' HL	ED 42	ED 52	ED 62	ED 72		
	INCREMENT 'INC'	03	13	23	33	DD 23	FD 23
	DECREMENT 'DEC'	0B	1B	2B	3B	DD 2B	FD 2B

Figure 4-8. 16-Bit Arithmetic

4.2.4 Rotate and Shift

A major capability of the Z80 is its ability to rotate or shift data in the accumulator, any general-purpose register, or any memory location. All of the rotate and shift opcodes are shown in Figure 4-9. Also included in the Z80 are arithmetic and logical shift operations. These operations are useful in an extremely wide range of applications including integer multiplication and division. Two BCD digit rotate instructions (RRD and RLD) allow a digit in the accumulator to be rotated with the two digits in a memory location pointed to by register pair HL (See Figure 4-9). These instructions allow for efficient BCD arithmetic.

Source and Destination

TYPE OF ROTATE SHIFT	A	B	C	D	E	H	L	(HL)	(IX+d)	(IY+d)
'RCL'	CB 07	CB 00	CB 01	CB 02	CB 03	CB 04	CB 05	CB 06	DD CB d 06	FD CB d 06
'RRC'	CB 0F	CB 08	CB 09	CB 0A	CB 06	CB 0C	CB 0D	CB 0E	DD CB d 0E	FD CB d 0E
'RL'	CB 17	CB 10	CB 11	CB 12	CB 13	CB 14	CB 15	CB 16	DD CB d 16	FD CB d 16
'RR'	CB 1F	CB 18	CB 19	CB 1A	CB 1B	CB 1C	CB 1D	CB 1E	DD CB d 1E	FD CB d 1E
'SLA'	CB 27	CB 20	CB 21	CB 22	CB 23	CB 24	CB 25	CB 26	DD CB d 26	FD CB d 26
'SRA'	CB 2F	CB 28	CB 29	CB 2A	CB 2B	CB 2C	CB 2D	CB 2E	DD CB d 2E	FD CB d 2E
'SRL'	CB 3F	CB 38	CB 39	CB 3A	CB 3B	CB 3C	CB 3D	CB 3E	DD CB d 3E	FD CB d 3E
'SRL'								ED 6F		
'SRL'								ED 67		

	A
RLCA	07
RRCA	0F
RLA	17
RRA	1F

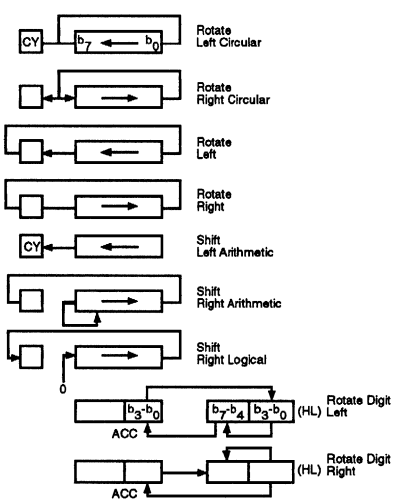


Figure 4-9. Rotates and Shifts

4.2 INSTRUCTION OPCODES (Continued)

4.2.5 Bit Manipulation

The ability to set, reset and test individual bits in a register or memory location is needed in almost every program. These bits may be flags in a general-purpose software routine, indications of external control conditions or data packed into memory locations to make memory utilization more efficient.

The Z80 has the ability to set, reset, or test any bit in the accumulator, any general-purpose register or any memory location with a single instruction. Figure 4-10 lists the 240 instructions that are available for this purpose. Register addressing can specify the accumulator or any general-purpose register on which the operation is to be performed. Register indirect and indexed addressing are available to operate on external memory locations. Bit test operations set the Zero flag (Z) if the tested bit is a zero. (Refer to section 5.2 for further explanation of flag operation.)

4.2.6 Jump, Call, and Return

Figure 4-11 lists all of the jump, call, and return instructions implemented in the Z80 CPU. A jump is a branch in a program where the program counter is loaded with the 16-bit value as specified by one of the three available addressing modes (Immediate Extended, Relative, or Register Indirect). Notice that the jump group has several different conditions that can be specified to be met before the jump will be made. If these conditions are not met, the program merely continues with the next sequential instruction. The conditions are all dependent on the data in the flag register. (Refer to section 5.2 for details on the flag register.) The immediate extended addressing is used to jump to any location in the memory. This instruction requires three bytes (two to specify the 16-bit address) with the low order address byte first followed by the high order address byte.

For example an unconditional jump to memory location 3E32H would be:

Address A	C3	Opcode
A + 1	32	Low Order Address
A + 2	3E	High Order Address

The relative jump instruction uses only two bytes, the second byte is a signed two's complement displacement from the existing PC. This displacement can be in the range of + 129 to −126 and is measured from the address of the instruction opcode.

Three types of register indirect jumps are also included. These instructions are implemented by loading the register pair HL or one of the index registers IX or IY directly into the PC. This capability allows for program jumps to be a function of previous calculations.

A call is a special form of a jump where the address of the byte following the call instruction is pushed onto the stack before the jump is made. A return instruction is the reverse of a call because the data on the top of the stack is popped directly into the PC to form a jump address. The call and return instructions allow for simple subroutine and interrupt handling. Two special return instructions have been included in the Z80 family of components. The return from interrupt instruction (RETI) and the return from nonmaskable interrupt (RETN) are treated in the CPU as an unconditional return identical to the opcode C9H. The difference is that (RETI) can be used at the end of an interrupt routine and all Z80 peripheral chips will recognize the execution of this instruction for proper control of nested priority interrupt handling. This instruction coupled with the Z80 peripheral devices implementation simplifies the normal return from nested interrupt. Without this feature, the following software sequence would be necessary to inform the interrupting device that the interrupt routine is completed:

Disable Interrupt —	Prevent interrupt before routine is exited.
LD A, n —	Notify peripheral that service
OUT n, A	routine is complete.
Enable Interrupt	
Return	

This seven byte sequence can be replaced with the one byte EI instruction and the two byte RETI instruction in the Z80. This is important since interrupt service time often must be minimized.

	Bit	REGISTER ADDRESSING							REG. INDIR.	INDEXED	
		A	B	C	D	E	H	L	(HL)	(IX+d)	(IY+d)
TEST 'BIT'	0	CB 47	CB 40	CB 41	CB 42	CB 43	CB 44	CB 45	CB 46	DD CB d 46	FD CB d 46
	1	CB 4F	CB 48	CB 49	CB 4A	CB 4B	CB 4C	CB 4D	CB 4E	DD CB d 4E	FD CB d 4E
	2	CB 57	CB 50	CB 51	CB 52	CB 53	CB 54	CB 55	CB 56	DD CB d 56	FD CB d 56
	3	CB 5F	CB 58	CB 59	CB 5A	CB 5B	CB 5C	CB 5D	CB 5E	DD CB d 46	FD CB d 46
	4	CB 67	CB 60	CB 61	CB 62	CB 63	CB 64	CB 65	CB 66	DD CB d 66	FD CB d 66
	5	CB 6F	CB 68	CB 69	CB 6A	CB 6B	CB 6C	CB 6D	CB 6E	DD CB d 6E	FD CB d 6E
	6	CB 77	CB 70	CB 71	CB 72	CB 73	CB 74	CB 75	CB 76	DD CB d 76	FD CB d 76
	7	CB 7F	CB 78	CB 79	CB 7A	CB 7B	CB 7C	CB 7D	CB 7E	DD CB d 46	DD CB d 46
REST BIT 'RES'	0	CB 87	CB 80	CB 81	CB 82	CB 83	CB 84	CB 85	CB 86	DD CB d 86	FD CB d 86
	1	CB 8F	CB 88	CB 89	CB 8A	CB 8B	CB 8C	CB 8D	CB 8E	DD CB d 8E	FD CB d 8E
	2	CB 97	CB 90	CB 91	CB 92	CB 93	CB 94	CB 95	CB 96	DD CB d 96	FD CB d 96
	3	CB 9F	CB 98	CB 99	CB 9A	CB 9B	CB 9C	CB 9D	CB 9E	DD CB d 9E	FD CB d 9E
	4	CB A7	CB A0	CB A1	CB A2	CB A3	CB A4	CB A5	CB A6	DD CB d A6	FD CB d A6
	5	CB AF	CB A8	CB A9	CB AA	CB AB	CB AC	CB AD	CB AE	DD CB d AE	FD CB d AE
	6	CB B7	CB B0	CB B1	CB B2	CB B3	CB B4	CB B5	CB B6	DD CB d B6	FD CB d B6
	7	CB BF	CB B8	CB B9	CB BA	CB BB	CB BC	CB BD	CB BE	DD CB d BE	DD CB d BE
SET BIT 'SET'	0	CB C7	CB C0	CB C1	CB C2	CB C3	CB C4	CB C5	CB C6	DD CB d C6	FD CB d C6
	1	CB CF	CB C8	CB C9	CB CA	CB CB	CB CC	CB CD	CB CE	DD CB d CE	FD CB d CE
	2	CB D7	CB D0	CB D1	CB D2	CB D3	CB D4	CB D5	CB D6	DD CB d D6	FD CB d D6
	3	CB DF	CB D8	CB D9	CB DA	CB DB	CB DC	CB DD	CB DE	DD CB d DE	FD CB d DE
	4	CB E7	CB E0	CB E1	CB E2	CB E3	CB E4	CB E5	CB E6	DD CB d E6	FD CB d E6
	5	CB EF	CB E8	CB E9	CB EA	CB EB	CB EC	CB ED	CB EE	DD CB d 46	FD CB d EE
	6	CB F7	CB F0	CB F1	CB F2	CB F3	CB F4	CB F5	CB F6	DD CB d F6	FD CB d F6
	7	CB FF	CB F8	CB F9	CB FA	CB FB	CB FC	CB FD	CB FE	DD CB d FE	FD CB d FE

Figure 4-10. Bit Manipulation Group

4.2 INSTRUCTION OPCODES (Continued)

			CONDITION									
			UN-COND.	CARRY	NON CARRY	ZERO	NON ZERO	PARITY EVEN	PARITY ODD	SIGN NEG	SIGN POS	REG B≠0
JUMP 'JP'	IMMED. EXT.	nn	C3 n n	Da n n	D2 n n	CA n n	C2 n n	EA n n	E2 n n	FA n n	F2 n n	
JUMP 'JR'	RELATIVE	PC+e	18 e-2	38 e-2	30 e-2	28 e-2	20 e-2					
JUMP 'JP'	REG. INDIR.	(HL)	E9									
JUMP 'JP'		(IX)	DD E9									
JUMP 'JP'		(IY)	FD E9									
'CALL'	IMMED EXT.	nn	CD n n	DC n n	D4 n n	CC n n	C4 n n	EC n n	E4 n n	FC n n	F4 n n	
DECREMENT B, JUMP IF NON ZERO 'DJNZ'	RELATIVE	PC+e										10 e-2
RETURN 'RET'	REFISTER INDIR.	(SP) (SP+1)	C9	D8	D0	C8	C0	E8	E0	F8	F0	
RETURN FROM INT 'RETI'	REG. INDIR.	(SP) (SP+1)	ED 4D									
RETURN FROM NON MASKABLE INT 'RETN'	REG. INDIR.	(SP) (SP+1)	ED 45									

NOTE-CERTAIN FLOAGS HAVE MORE

Figure 4-11. Jump, Call, and Return Group

To facilitate program loop control the instruction DJNZ e can be used advantageously. This two byte, relative jump instruction decrements the B register and the jump occurs if the B register has not been decremented to zero. The relative displacement is expressed as a signed two's complement number. A simple example of its use might be:

Address	Instruction	Comments
N, N + 1	LD B, 7	: set B register to count of 7
N + 2 to N + 9	(Perform a sequence of instructions)	: loop to be performed 7 times
N + 10, N + 11	DJNZ –8	: to jump from N + 12 to N + 2
N + 12	(Next Instruction)	

Figure 4-12 lists the eight opcodes for the restart instruction. This instruction is a single byte call to any of the eight addresses listed. The simple mnemonic for these eight calls is also shown. The value of this instruction is that frequently used routines can be called with this instruction to minimize memory usage.

4.2.7 Input/Output

The Z80 has an extensive set of Input and Output instructions as shown in Figures 4-13 and 4-14. The addressing of the input or output device can be either absolute or register indirect, using the C register. Notice that in the register indirect addressing mode data can be transferred between the I/O devices and any of the internal registers. In addition, eight block transfer instructions have been implemented. These instructions are similar to the memory block transfers except that they use register pair HL for a pointer to the memory source (output commands) or destination (input commands) while register B is used as a byte counter. Register C holds the address of the port for which the input or output command is desired. Since register B is eight bits in length, the I/O block transfer command handles up to 256 bytes.

In the instructions IN A, and OUT n, A, the I/O device address n appears in the lower half of the address bus (A7-A0) while the accumulator content is transferred in the upper half of the address bus. In all register indirect input output instructions, including block I/O transfers the content of register C is transferred to the lower half of the address bus (device address) while the content of register B is transferred to the upper half of the address bus.

4.2.8 CPU Control Group

Figure 4-15 illustrates the six general-purpose CPU control instructions. The NOP is a do-nothing instruction. The HALT instruction suspends CPU operation until a subsequent interrupt is received, while the DI and EI are used to lock out and enable interrupts. The three interrupt mode commands set the CPU into any of the three available interrupt response modes as follows. If mode zero is set, the interrupting device can insert any instruction on the data bus and allow the CPU to execute it. Mode 1 is a simplified mode where the CPU automatically executes a restart (RST) to location 0038H so that no external hardware is required. (The old PC content is pushed onto the stack). Mode 2 is the most powerful in that it allows for an indirect call to any location in memory. With this mode, the CPU forms a 16-bit memory address where the upper eight bits are the content of register I and the lower eight bits are supplied by the interrupting device. This address points to the first of two sequential bytes in a table where the address of the service routine is located. The CPU automatically obtains the starting address and performs a CALL to this address.

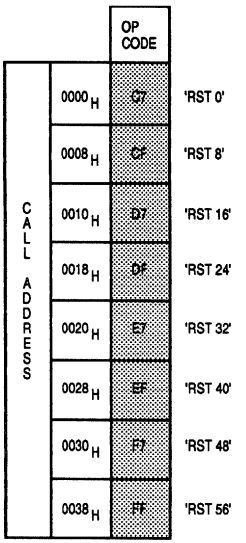
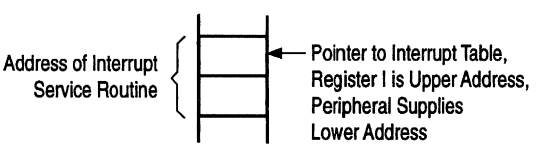


Figure 4-12. Restart Group



4.2 INSTRUCTION OPCODES (Continued)

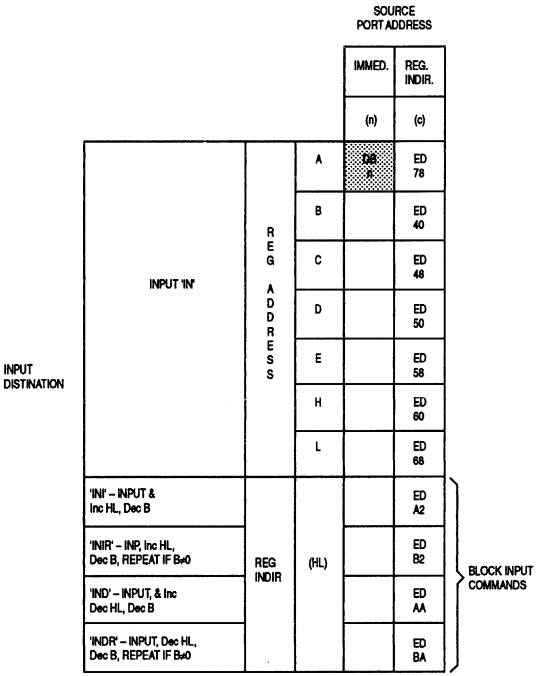


Figure 4-13. Input Group

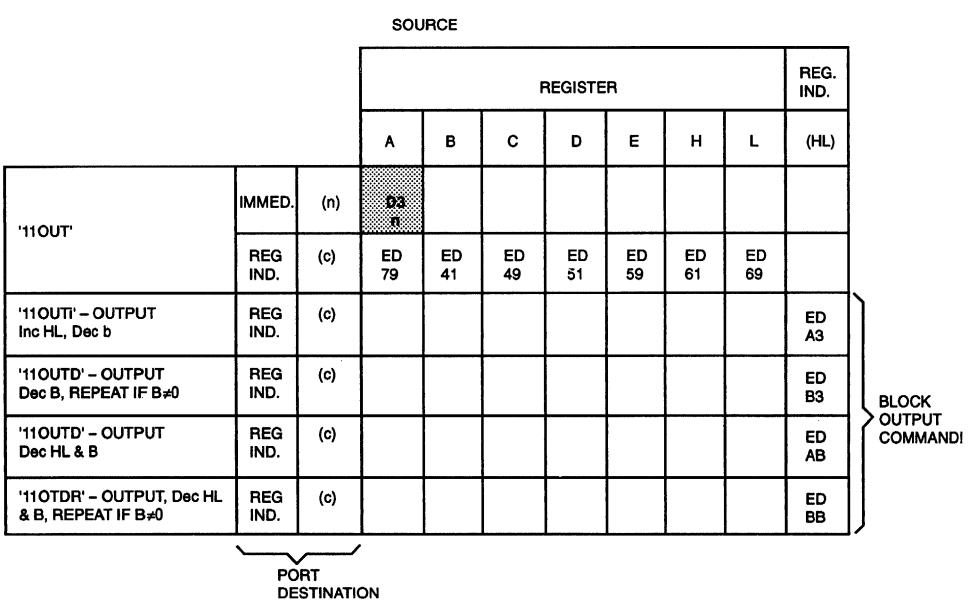


Figure 4-14. Output Group

A

'NOP'	00	
'HALT'	76	
DISABLE INT '(DI)'	F3	
ENABLE INT '(EI)'	FB	
SET INT MODE 0 'IM0'	ED 46	8080A MODE
SET INT MODE 1 'IM1'	ED 56	CALL TO LOCATION 0038_H
SET INT MODE 2 'IM2'	ED 5E	INDIRECT CALL USING REGISTER I AND 8 BITS FROM INTERR DEVICE AS A POINTER

Figure 4-15. Miscellaneout CPU Control

CHAPTER 5
Z80 INSTRUCTION DESCRIPTION

5.0 INTRODUCTION: Z80 ASSEMBLY LANGUAGE

The assembly language provides a means for writing a program without having to be concerned with actual memory addresses or machine instruction formats. It allows the use of symbolic addresses to identify memory locations and mnemonic codes (opcodes and operands) to represent the instructions themselves. Labels (symbols) can be assigned to a particular instruction step in a source program to identify that step as an entry point for use in subsequent instructions. Operands following each instruction represent storage locations, registers, or constant values. The assembly language also includes assembler directives that supplement the machine instruction. A pseudo-op, for example, is a statement which is not translated into a machine instruction, but rather is interpreted as a directive that controls the assembly process.

A program written in assembly language is called a source program. It consists of symbolic commands called statements. Each statement is written on a single line and may consist of from one to four entries: A label field, an operation field, an operand field and a comment field. The source program is processed by the assembler to obtain a machine language program (object program) that can be executed directly by the Z80 CPU.

Zilog provides several different assemblers which differ in the features offered. Both absolute and relocatable assemblers are available with the Development and Microcomputer Systems. The absolute assembler is contained in base level software operating in a 16K memory space while the relocating assembler is part of the RIO environment operating in a 32K memory space.

5.1 Z80 STATUS INDICATORS (FLAGS)

The flag register (F and F') supplies information to the user regarding the statue of the Z80 at any given time. The bit positions for each flag is shown below:

7	6	5	4	3	2	1	0
S	Z	X	N	X	P/V	N	C

where:

C	=	Carry Flag
N	=	Add/Substract
P/V	=	Parity/Overflow Flag
H	=	Half Carry Flag
Z	=	Zero Flag
S	=	Sign Flag
X	=	Not Used

Each of the two Z80 Flag Registers contains six bits of statue information which are set or reset by CPU operations. (Bits 3 and 5 are not used.) Four of these bits are testable(C, P/V, Z, and S) for use with conditional Jump, Call, or Return instructions. Two flags are not testable (H, N) and are used for BCD arithmetic.

5.1.1 Carry Flag (C)

The carry bit is set or reset depending on the operation being performed. For 'ADD' instructions that generate a carry and 'SUBTRACT' instructions that generate a borrow, the Carry flag will be sit. The Carry flag is reset by an ADD that does not generate a carry and a 'SUBTRACT' that generates no borrow. This saved carry facilitates software routines for extended precision arithmetic. Also, the 'DAA' instruction will set the Carry flag if the conditions for making the decimal adjustment are met.

For instructions RLA, RRA, RLS, and RRS, the carry bit is used as a link between the LSB and MSB for any register or memory location. During instructions RLCA, RLC s, and SLA s, the carry contains the last value shifted out of bit 7 of any register or memory location. During instructions RRCA, RRC s, SRA s, and SRL s the carry contains the last value shifted out of bit 0 of any register or memory location.

For the logical instructions AND s, OR s, and XOR s, the carry will be reset.

The Carry flag can also be set (SCF) and complemented (CCF).

5.1 Z80 STATUS INDICATORS (FLAGS) (Continued)

5.1.2 Add/Substract Flag (N)

This flag is used by the decimal adjust accumulator instruction (DAA) to distinguish between 'ADD' and 'SUBTRACT' instructions. For all 'ADD' instructions, N will be set to a '0'. For all 'SUBTRACT' instructions, N will be set to a to a '1'.

5.1.3 Parity/Overflow Flag (P/V)

This flag is set to a particular state depending on the operation being performed.

For arithmetic operations, this flag indicates an overflow condition when the result in the Accumulator is greater than the maximum possible number (+127) or is less than the minimum possible number (–128). This overflow condition can be determined by examining the sign bits of the operands.

For addition, operands with different signs will never cause overflow. When adding operands with like signs and the result has a different sign, the overflow flag is set. For example:

+120	=	0111 1000	ADDEND	
+105	=	0110 1001	AUGEND	
+225	=	1110 0001	(–95)	SUM

The two numbers added together has resulted in a dumber that exceeds +127 and the two positive operands has resulted in a negative number (–95) which is incorrect. The Overflow flag is therefore set.

For subtraction, overflow can occur for operands of unlike signs. Operands of like sign will never cause overflow. For example:

+127	0111 1111	MINUEND
(–) –64	1100 0000	SUBTRAHEND
+191	1011 1111	DIFFERENCE

The minuend sign has changed from a positive to a negative, giving an incorrect difference. Overflow is therefore set.

Another method for predicting an overflow is to observe the carry into and out of the sign bit. If there is a carry in and no carry out, or if there is no carry in and a carry out, then overflow has occurred.

This flag is also used with logical operations and rotate instructions to indicate the parity of the result. The number of '1' bits in a byte are counted. If the total is odd, 'ODD' parity (P=0) is flagged. If the total is even, 'EVEN' parity is flagged (P=1).

During search instructions (CPI, CPIR, CPD, CPDR) and block transfer instructions (LDI, LDIR, LDD, LDDR) the P/V flag monitors the state of the byte count register (BC). When decrementing, the byte counter results in a zero value, the flag is reset to 0, otherwise the flag is a Logic 1.

During LD A, I and LD A, R instructions, the P/V flag will be set with the contents of the interrupt enable flip-flop (IFF2) for storage or testing.

When inputting a byte from an I/O device, IN r, (C), the flag will be adjusted to indicate the parity of the data.

5.1.4 Half Carry Flag (H)

The Half Carry Flag (H) will be set or reset depending on the carry and borrow status between bits 3 and 4 of an 8-bit arithmetic operation. This flag is used by the decimal adjust accumulator instruction (DAA) to correct the result of a packed BCD add or subtract operation. The H flag will be set (1) or reset (0) according to the following table:

H	Add	Substract
1	There is a carry from Bit 3 to Bit 4	There is borrow from Bit 4
0	There is no carry from Bit 3 to Bit 4	There is borrow from Bit 4

5.1.5 Zero Flag (Z)

The Zero Flag (Z) is set or reset if the result generated by the execution of certain instructions is a zero.

For 8-bit arithmetic and logical operations, the Z flag will be set to a '1' if the resulting byte in the Accumulator is zero. In the byte is not zero, the Z flag is reset to '0'.

For compare (search) instructions, the Z flag will be set to a '1' if a comparison is found between the value in the Accumulator and the memory location pointed to by the contents of the register pair HL.

When testing a bit in a register or memory location, the Z flag will contain the complemented state of the indicated bit (see Bit b, s).

When inputting or outputting a byte between a memory location and an I/O device (INI, IND, OUTI, and OUTD), if the result of B–1 is zero, the Z flag is set, otherwise it is reset. Also for byte inputs from I/O devices using IN r, (C), the Z Flag is set to indicate a zero byte input.

5.1.6 Sign Flag(S)

The Sign Flag (S) stores the state of the most significant bit of the Accumulator (bit 7). When the Z80 performing arithmetic operations on signed numbers, binary two's complement notation is used to represent and process numeric information. A positive number is identified by a '0' in bit 7. A negative number is identified by a '1'. The binary equivalent of the magnitude of a positive number is stored in bits 0 to 6 for a total range of from 0 to 127. A negative number is represented by the two's complement of the equivalent positive number. The total range for negative numbers is from –1 to –128.

When inputting a byte from an I/O device to a register, IN r, (C), the S flag will indicate either positive (S=0) or negative (S=1) data.

5.2 Z80 INSTRUCTION DESCRIPTION

NOTE: Execution time (E.T.) for each instruction is given in microseconds for in assumed 4 MHz clock. Total machine cycles (M) are indicated with total clock periods (T States). Also indicated are the number of T States for each M cycle. For example:

M Cycles: 2 T States: 7(4,3) 4 MHz E.T.: 1.75

indicates that the instruction consists of 2 machine cycles. The first cycle contains 4 clock periods (T States). The second cycle contains 3 clock periods for a total of 7 clock periods or T States. The instruction will execute in 1.75 microseconds.

Register format is shown for each instruction with the most significant bit to the left and the least significant bit to the right.

A

Z80®
INSTRUCTION DESCRIPTION

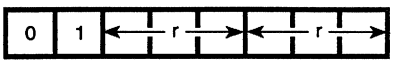
8-BIT LOAD GROUP

LD r, r'

Operation: r, ← r'

Opcode: LD

Operands: r, r'



Description: The contents of any register r' are loaded into any other register r, Note: r, r' identifies any of the registers A, B, C, D, E, H, or L, assembled as follows in the object code:

Register	r, r'
A	111
B	000
C	001
D	010
E	011
H	100
L	101

M Cycles	T States	MHz E.T.
1	4	1.0

Condition Bits Affected:
None.

Example: If the H register contains the number 8AH, and the E register contains 10H, the instruction

LD H, E

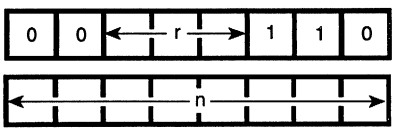
would result in both registers containing 10H.

LD r, n

Operation: r ← n

Opcode: LD

Operands: r, n



Description: The 8-bit integer n is loaded into any register r, where r identifies register A, B, C, D, E, H, or L, assembled as follows in the object code:

Register	r
A	111
B	000
C	001
D	010
E	011
H	100
L	101

M Cycles	T States	4 MHz E.T.
2	7 (4, 3)	1.75

Conditions Bits Affected:

None.

Example: After the execution of

```
LD E, A5H
```

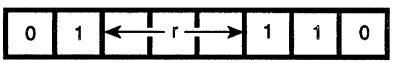
the contents of register E will be A5H.

LD r, (HL)

Operation: r ← (HL)

Opcode: LD

Operands: r, (HL)



Description: The 8-bit contents of memory location (HL) are loaded into register r, where r identifies register A, B, C, D, E, H, or L, assembled as follows in the object code:

Register	r
A	111
B	000
C	001
D	010
E	011
H	100
L	101

M Cycles	T States	4 MHz E.T.
2	7 (4,6)	1.75

Condition Bits Affected:
None

Example: If register pair HL contains the number 75A1H, and memory address 75A1H contains the byte 58H, the execution of

```
LD C, (HL)
```

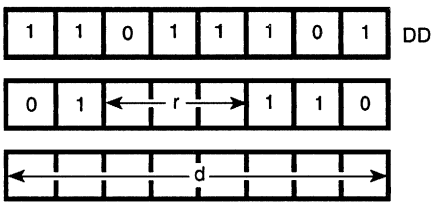
will result in 58H in register C.

LD r, (IX+d)

Operation: r ← (IX+d)

Opcode: LD

Operands: r, (IX+d)



Description: The operand (IX+d), (the contents of the Index Register IX summed with a two's complement displacement integer d) is loaded into register r, where r identifies register A, B, C, D, E, H, or L, assembled as follows in the object code:

Register	r
A	111
B	000
C	001
D	010
E	011
H	100
L	101

M Cycles	T States	4 MHz E.T.
5	19 (4, 4, 3, 5, 3)	2.50

Conditions Bits Affected:

None.

Example: If the Index Register IX contains the number 25AFH, the instruction

LD B, (IX+19H)

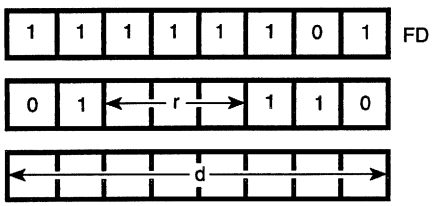
will cause the calulation of the sum 25AFH + 19H, which points to memory location 25C8H. If this address contains byte 39H, the instruction will result in register B also containing 39H.

LD r, (IY+d)

Operation: r ← (IY+D)

Opcode: LD

Operands: r, (IY+d)



Description: The operand (IY+d) (the contents of the Index Register IY summed with a two's complement displacement integer (d) is loaded into register r, where r identifies register A, B, C, D, E, H, or L, assembled as follows in the object code:

Register	r
A	111
B	000
C	001
D	010
E	011
H	100
L	101

M Cycles	T States	4 MHz E.T.
5	19 (4, 4, 3, 5, 3)	4.75

Condition Bits Affected:

None.

Example: If the Index Register IY contains the number 25AFH, the instruction

LD B, (IY+19H)

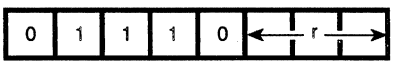
will cause the calculation of the sum 25AFH + 19H, which points to memory location 25C8H. If this address contains byte 39H, the instruction will result in register B also containing 39H.

LD (HL), r

Operation: (HL) ← r

Opcode: LD

Operands: (HL), r



Description: The contents of register r are loaded into the memory location specified by the contents of the HL register pair. The symbol r identifies register A, B, C, D, E, H, or L, assembled as follows in the object code:

Register	r
A	111
B	000
C	001
D	010
E	011
H	100
L	101

M Cycles	T States	4 MHz E.T.
2	7 (4, 3)	1.75

Conditions Bits Affected:

None.

Example: If the contents of register pair HL specifies memory location 2146H, and the B register contains the byte 29H, after the execution of

```
LD (HL), B
```

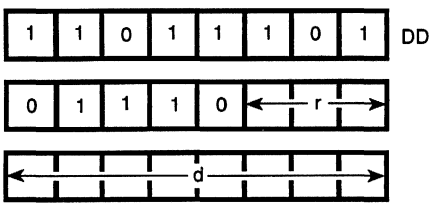
memory address 2146H will also contain 29H.

LD (IX+d), r

Operation: (IX+d) ← r

Opcode: LD

Operands: (IX+d), r



Description: The contents of register r are loaded into the memory address specified by the contents of Index Register IX summed with d, a two's complement displacement integer. The symbol r identifies register A, B, C, D, E, H, or L, assembled as follows in the object code:

Register	r
A	111
B	000
C	001
D	010
E	011
H	100
L	101

M Cycles	T States	4 MHz E.T.
5	19 (4, 4, 3, 5, 3)	4.75

Conditions Bits Affected:

None.

Example: If the C register contains the byte 1CH, and the Index Register IX contains 3100H, then the instruction

```
LD (IX+6H), C
```

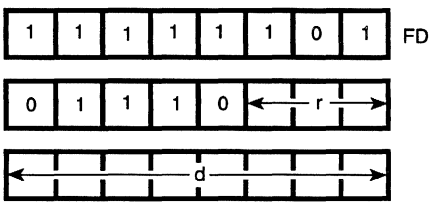
will perform the sum 3100H + 6H and will load 1CH into memory location 3106H.

LD (IY+d), r

Operation: (IY+d) ← r

Opcode: LD

Operands: (IY+d), r



Description: The contents of resister r are loaded into the memory address specified by the sum of the contents of the Index Register IY and d, a two's complement displacement integer. The symbol r is specified according to the following table.

Register	r
A	111
B	000
C	001
D	010
E	011
H	100
L	101

M Cycles	T States	4 MHz E.T.
5	19 (4, 4, 3, 5, 3)	4.75

Conditions Bits Affected:

None.

Example: If the C register contains the byte 48H, and the Index Register IY contains 2A11H, then the instruction

```
LD (IY+4H), C
```

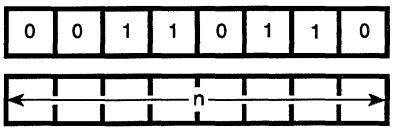
will perform the sum 2A11H + 4H, and will load 48H into memory location 2A15.

LD (HL), n

Operation: (HL) ← n

Opcode: LD

Operands: (HL), n



Description: Integer n is loaded into the memory address specified by the contents of the HL register pair.

M Cycles	T States	4 MHz E.T.
3	10 (4, 3, 3)	2.50

Condition Bits Affected

None.

Example: If the HL register pair contains 4444H, the instruction

LD (HL), 28H

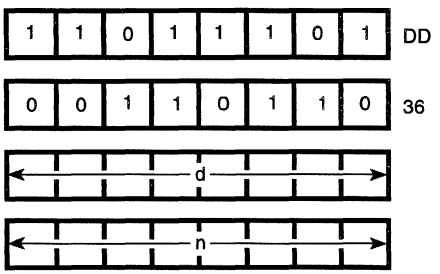
will result in the memory location 4444H containing the byte 28H.

LD (IX+d), n

Operation: (IX+d) ← n

Opcode: LD

Operands: (IX+d), n



Description: The n operand is loaded into the memory address specified by the sum of the Index Register IX an the two's complement displacement operand d.

M Cycles	T States	4 MHz E.T.
5	19 (4, 4, 3, 5, 3)	4.75

Condition Bits Affected:

None.

Example: If the Index Register IX contains the number 219AH, the instruction

LD (IX+5H), 5AH

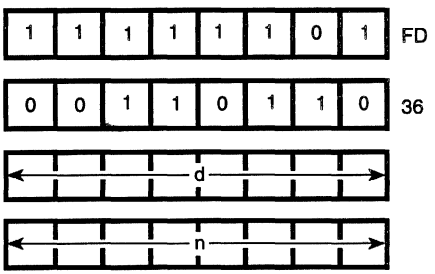
would result in the byte 5AH in the memory address 219FH.

LD (IY+d), n

Operation: (IY+d) ← n

Opcode: LD

Operands: (IY+d), n



Description: Integer n is loaded into the memory location specified by the contents of the Index Register summed with the two's complement displacement integer d.

M Cycles	T States	4 MHz E.T.
5	19 (4, 4, 3, 5, 3)	2.50

Conditions Bits Affected:

None.

Example: If the Index Register IY contains the number A940H, the instruction

LD (IY+10H), 97H

would result in byte 97 in memory location A950H.

LD A, (BC)

Operation: A ← (BC)

Opcode: LD

Operands: A, (BC)

0	0	0	0	1	0	1	0	0A

Description: The contents of the memory location specified by the contents of the BC register pair are loaded into the Accumulator.

M Cycles	T States	4 MHz E.T.
2	7 (4, 3)	1.75

Conditions Bits Affected:

None.

Example: If the BC register pair contains the number 4747H, and memory address 4747H contains the byte 12H, then the instruction

```
LD A, (BC)
```

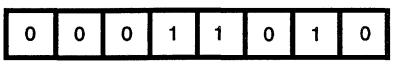
will result in byte 12H in register A.

LD A, (DE)

Operation: A ← (DE)

Opcode: LD

Operands: A, (DE)



Description: The contents of the memory location specified by the register pair DE are loaded into the Accumulator.

M Cycles	T States	4 MHz E.T.
2	7 (4, 3)	1.75

Conditions Bits Affected:

None.

Example: If the DE register pair contains the number 30A2H and memory address 30A2H contains the byte 22H, then the instruction

```
LD A, (DE)
```

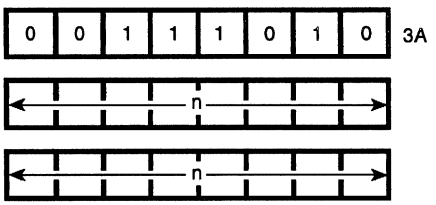
will result in byte 22H in register A.

LD A, (nn)

Operation: A ← (nn)

Opcode: LD

Operands: A, (nn)



Description: The contents of the memory location specified by the operands nn are loaded into the Accumulator. The first n operand after the opcode is the low order byte of a 2-byte memory address.

M Cycles	T States	4 MHz E.T.
4	13 (4, 3, 3, 3)	3.25

Condition Bits Affected:

None.

Example: If the contents of nn is number 8832H, and the content of memory address 8832H is byte 04H, after the instruction

```
LD A, (nn)
```

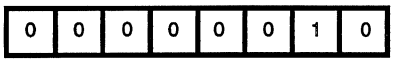
byte 04H will be in the Accumulator.

LD (BC), A

Operation: (BC) ← A

Opcode: LD

Operands: (BC), A



Description: The contents of the Accumulator are loaded into the memory location specified by the contents of the register pair BC.

M Cycles	T States	4 MHz E.T.
2	7 (4, 3)	1.75

Conditions Bits Affected:

None.

Example: If the Accumulator contains 7AH and the BC register pair contains 1212H the instruction

LD (BC), A

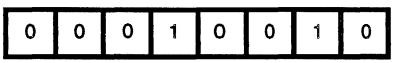
will result in 7AH being in memory location 1212H.

LD (DE), A

Operation: (DE) ← A

Opcode: LD

Operands: (DE), A



Description: The contents of the Accumulator are loaded into the memory location specified by the contents of the DE register pair.

M Cycles	T States	4 MHz E.T.
2	7 (4, 3)	1.75

Conditions Bits Affected:

None.

Example: If the contents of register pair DE are 1128H, and the Accumulator contains byte A0H, the instruction

```
LD (DE), A
```

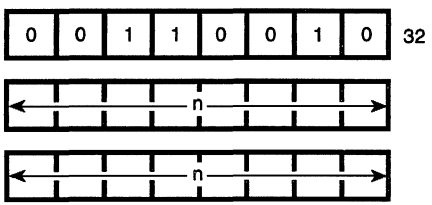
will result in A0H being in memory location 1128H.

LD (nn), A

Operation: (nn) ← A

Opcode: LD

Operands: (nn), A



Description: The contents of the Accumulator are loaded into the memory address specified by the operand nn. The first n operand after the opcode is the low order byte of nn.

M Cycles	T States	4 MHz E.T.
4	13 (4, 3, 3, 3)	3.25

Conditions Bits Affected:

None.

Example: If the contents of the Accumulator are byte D7H, after the execution of

```
LD (3141H), A
```

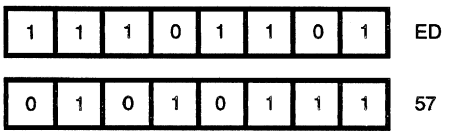
D7H will result in memory location 3141H.

LD A, I

Operation: A ← I

Opcode: LD

Operands: A, I



Description: The contents of the Interrupt Vector Register I are loaded into the Accumulator.

M Cycles	T States	MHz E.T.
2	9 (4, 5)	2.25

Condition Bits Affected:

S: Set if I-Register is negative; reset otherwise
Z: Set if I-Register is zero; reset otherwise
H: Reset
P/V: Contains contents of IFF2
N: Reset
C: Not affected

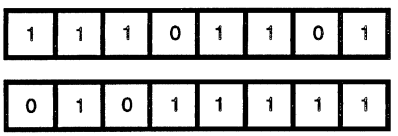
Note: If an interrupt occurs during execution of this instruction, the Parity flag will contain a 0.

LD A, R

Operation: A, ← R

Opcode: LD

Operands: A, R



Description: The contents of Memory Refresh Register R are loaded into the Accumulator.

M Cycles	T States	MHz E.T.
2	9 (4, 5)	2.25

Condition Bits Affected:

S: Set if, R-Register is negative; reset otherwise
Z: Set if R-Register is zero; reset otherwise
H Reset
P/V: Contains contents of IFF2
N: Reset
C: Not affected

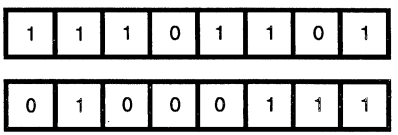
Note: If an interrupt occurs during execution of this instruction, the parity flag will contain a 0.

LD I, A

Operation: I ← A

Opcode: LD

Operands: I, A



Description: The contents of the Accumulator are loaded into the Interrupt Control Vector Register, I.

M Cycles	T States	MHz E.T.
2	9 (4, 5)	2.25

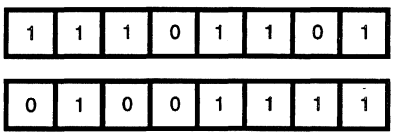
Condition Bits Affected:
None

LD R, A

Operation: R ← A

Opcode: L D

Operands: R, A



Description: The contents of the Accumulator are loaded into the Memory Refresh register R.

M Cycles	T States	MHz E.T.
2	9 (4, 5)	2.25

Condition Bits Affected:
None

A

Z80®
INSTRUCTION DESCRIPTION

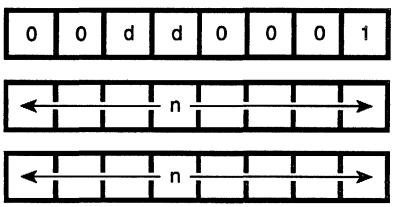
16-BIT LOAD GROUP

LD dd, nn

Operation: dd ← nn

Opcode: LD

Operands: dd, nn



Description: The 2-byte integer nn is loaded into the dd register pair, where dd defines the BC, DE, HL, or SP register pairs, assembled as follows in the object code:

Pair	dd
BC	00
DE	01
HL	10
SP	11

The first n operand after the opcode is the low order byte.

M Cycles	T States	4 MHz E.T.
2	10 (4, 3, 3)	2.50

Condition Bits Affected:
None

Example: After the execution of

LD HL, 5000H

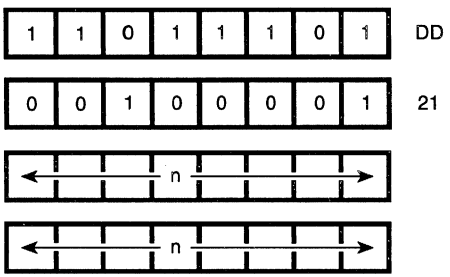
the contents of the HL register pair will be 5000H.

LD IX, nn

Operation: Ix ← nn

Opcode: LD

Operands: IX, nn



Description: Integer nn is loaded into the Index Register IX. The first n operand after the opcode is the low order byte.

M Cycles	T States	4 MHz E.T.
4	14 (4, 4, 3, 3)	3.50

Condition Bits Affected:

None

Example: After the instruction

```
LD IX, 45A2H
```

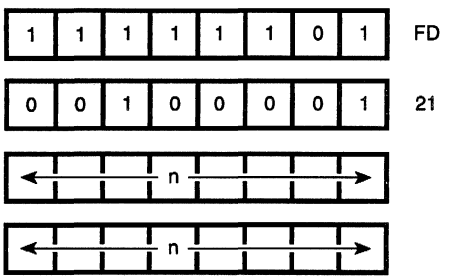
the Index Register will contain integer 45A2H.

LD IY, nn

Operation: IY ← nn

Opcode: LD

Operands: IY, nn



Description: Integer nn is loaded into the Index Register IY. The first n operand after the opcode is the low order byte.

M Cycles	T States	4 MHz E.T.
4	14 (4, 4, 3, 3)	3.50

Condition Bits Affected:
None

Example: After the instruction

LD IY, 7733H

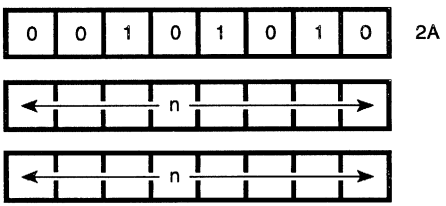
the Index Register IY will contain the integer 7733H.

LD HL, (nn)

Operation: H ← (nn+1), L ← (nn)

Opcode: LD

Operands: HL, (nn)



Description: The contents of memory address (nn) are loaded into the low order portion of register pair HL (register L), and, the contents of the next highest memory address (nn+1) are loaded into the high order portion of HL (register H). The first n operand after the opcode is the low order byte of nn.

M Cycles	T States	4 MHz E.T.
5	16 (4, 3, 3, 3, 3)	4.00

Condition Bits Affected:

None

Example: If address 4545H contains 37H, and address 4546H, contains A1H after the instruction

LD HL, (4545H)

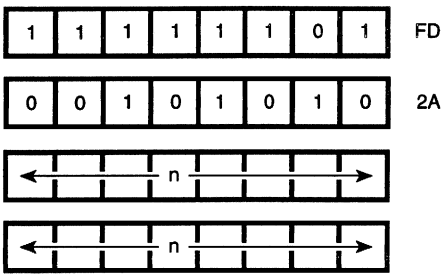
the HL register pair will contain A137H.

LD IY, (nn)

Operation: IYh ← (nn+1), IYl ← nn)

Opcode: LD

Operands: IY, (nn)



Description: The contents of address (nn) are loaded into the low order portion of Index Register IY, and the contents of the next highest memory address (nn+1) are loaded into the high order portion of IY. The first n operand after the opcode is the low order byte of nn.

M Cycles	T States	4 MHz E.T.
6	20 (4, 4, 3, 3, 3, 3)	5.00

Condition Bits Affected:
None

Example: If address 6666H contains 92H, and address 6667H contains DAH, after the instruction

LD IY, (6666H)

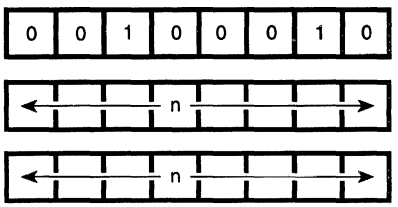
the Index Register IY will contain DA92H.

LD (nn), HL

Operation: (nn+1) ← H, (nn) ← L

Opcode: LD

Operands: (nn), HL



Description: The contents of the low order portion of register pair HL (register L) are loaded into memory address (nn), and the contents of the high order portion of HL (register H) are loaded into the next highest memory address (nn+1). The first n operand after the opcode is the low order byte of nn.

M Cycles	T States	4 MHz E.T.
6	20 (4, 4, 3, 3, 3, 3)	5.00

Condition Bits Affected:

None

Example: If the content of register pair HL is 483AH, after the instruction

LD (B229H), HL

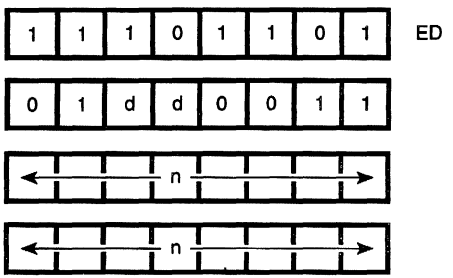
address B229H will contain 3AH, and address B22AH will contain 48H.

LD (nn), dd

Operation: (nn+1) ← ddh, (nn) ← ddl

Opcode: LD

Operands: (nn), dd



Description: The low order byte of register pair dd is loaded into memory address (nn); the upper byte is loaded into memory address (nn+1). Register pair dd defines either BC, DE, HL, or SP, assembled as follows in the object code:

Pair	dd
BC	00
DE	01
HL	10
SP	11

The first n operand after the opcode is the low order byte of a two byte memory address.

M Cycles	T States	4 MHz E.T.
6	20 (4, 4, 3, 3, 3, 3)	5.00

Condition Bits Affected:
None

Example: If register pair BC contains the number 4644H, the instruction

LD (1000H), BC

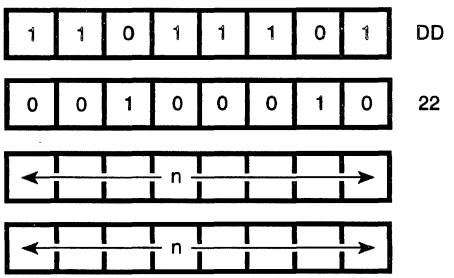
will result in 44H in memory location 1000H, and 46H in memory location 1001H.

LD (nn), IX

Operation: (nn+1) ← IXh, (nn) ← IXl

Opcode: LD

Operands: (nn), IX



Description: The low order byte in Index Register IX is loaded into memory address (nn); the upper order byte is loaded into the next highest address (nn+1). The first n operand after the opcode is the low order byte of nn.

M Cycles	T States	4 MHz E.T.
6	20 (4, 4, 3, 3, 3, 3)	5.00

Condition Bits Affected:
None

Example: If the Index Register IX contains 5A30H, after the instruction

LD (4392H), IX

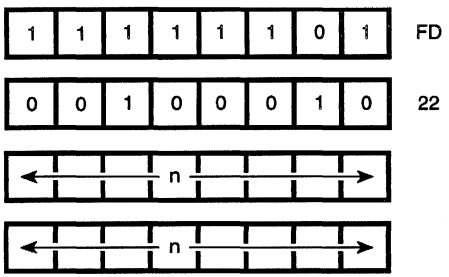
memory location 4392H will contain number 30H, and location 4393H will contain 5AH.

LD (nn), IY

Operation: (nn+1) ← IYh, (nn) ← IYl

Opcode: LD

Operands: (nn), IY



Description: The low order byte in Index Register IY is loaded into memory address (nn); the upper order byte is loaded into memory location (nn+1). The first n operand after the opcode is the low order byte of nn.

M Cycles	T States	4 MHz E.T.
6	20 (4, 4, 3, 3, 3, 3)	5.00

Condition Bits Affected:
None

Example: If the Index Register IY contains 4174H after the instruction

LD (8838H), IY

memory location 8838H will contain number 74H, and memory location 8839H will contain 41H.

LD SP, HL

Operation: SP ← HL

Opcode: LD

Operands: SP, HL

1	1	1	1	1	0	0	1	F9

Description: The contents of the register pair HL are loaded into the Stack Pointer (SP).

M Cycles	T States	4 MHz E.T.
6	20 (4, 4, 3, 3, 3, 3)	5.00

Condition Bits Affected:
None

Example: If the register pair HL contains 442EH, after the instruction

```
LD SP, HL
```

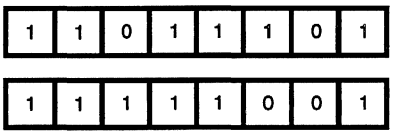
the Stack Pointer will also contain 442EH.

LD SP, IX

Operation: SP ← IX

Opcode: LD

Operands: SP, IX



Description: The 2-byte contents of Index Register IX are loaded into the Stack Pointer (SP).

M Cycles	T States	4 MHz E.T.
2	10 (4, 6)	2.50

Condition Bits Affected:
None

Example: If the contents of the Index Register IX are 98DAH, after the instruction

```
LD SP, IX
```

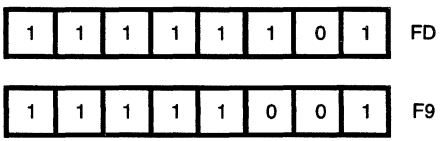
the contents of the Stack Pointer will also be 98DAH.

LD SP, IY

Operation: SP ← IY

Opcode: LD

Operands: SP, IY



Description: The 2-byte contents of Index Register IY are loaded into the Stack Pointer SP.

M Cycles	T States	4 MHz E.T.
2	10 (4, 6)	2.50

Condition Bits Affected:
None.

Example: If Index Register IY contains the integer A227H, after the instruction

LD SP, IY

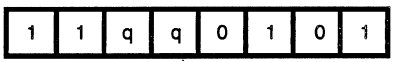
the Stack Pointer will also contain A227H.

PUSH qq

Operation: (SP–2) ← qqL, (SP–1) ← qqH

Opcode: PUSH

Operands: qq



Description: The contents of the register pair qq are pushed into the external memory LIFO (last-in, first-out) Stack. The Stack Pointer (SP) register pair holds the 16-bit address of the current "top" of the Stack. This instruction first decrements the SP and loads the high order byte of register pair qq into the memory address now specified by the SP; then decrements the SP again and loads the low order byte of qq into the memory location corresponding to this new address in the SP. The operand qq identifies register pair BC, DE, HL, or AF, assembled as follows in the object code:

Pair	qq
BC	00
DE	01
HL	10
AF	11

M Cycles	T States	4 MHz E.T.
3	11 (5, 3, 3)	2.75

Condition Bits Affected:
None.

Example: If the AF register pair contains 2233H and the Stack Pointer contains 1007H, after the instruction

```
PUSH AF
```

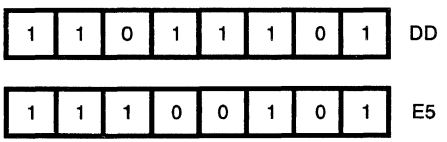
memory address, 1006H will contain 22H, memory address 1005H will contain 33H, and the Stack Pointer will contain 1005H.

PUSH IX

Operation: (SP–2) ← IXL, (SP–1) ← IXH

Opcode: PUSH

Operands: IX



Description: The contents of the Index Register IX are pushed into the external memory LIFO (last-in, first-out) Stack. The Stack Pointer (SP) register pair holds the 16-bit address of the current "top" of the Stack. This instruction first decrements the SP and loads the high order byte of IX into the memory address now specified by the SP; then decrements the SP again and loads the low order byte into the memory location corresponding to this new address in the SP.

M Cycles	T States	4 MHz E.T.
3	15 (4, 5, 3, 3)	3.75

Condition Bits Affected:
None.

Example: If the Index Register IX contains 2233H and the Stack Pointer contains 1007H, after the instruction

```
PUSH IX
```

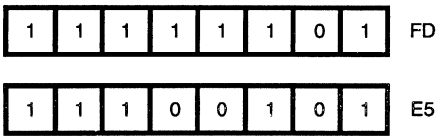
memory address 1006H will contain 22H, memory address 1005H will contain 33H, and the Stack Pointer will contain 1005H.

PUSH IY

Operation: (SP–2) ← IYL, (SP–1) ← IYH

Opcode: PUSH

Operands: IY



Description: The contents of the Index Register IY are pushed into the external memory LIFO (last-in, first-out) Stack. The Stack Pointer (SP) register pair holds the 16-bit address of the current "top" of the Stack. This instruction first decrements the SP and loads the high order byte of IY into the memory address now specified by the SP; then decrements the SP again and loads the low order byte into the memory location corresponding to this new address in the SP.

M Cycles	T States	4 MHz E.T.
4	15 (4, 5, 3, 3)	3.75

Condition Bits Affected:
None.

Example: If the Index Register IY contains 2233H and the Stack Pointer Contains 1007H, after the instruction

```
PUSH IY
```

memory address 1006H will contain 22H, memory address 1005H will contain 33H, and the Stack Pointer will contain 1005H.

POP qq

Operation: qqH ← (SP+1), qqL ← (SP)

Opcode: POP

Operands: qq

1	1	q	q	0	0	0	1

Description: The top two bytes of the external memory LIFO (last-in, first-out) Stack are popped into register pair qq. The Stack Pointer (SP) register pair holds the 16-bit address of the current "top" of the Stack. This instruction first loads into the low order portion of qq, the byte it the memory location corresponding to the contents of SP; then SP is incriminated and the contents of the corresponding adjacent memory location are loaded into the high order portion of qq and the SP is now incriminated again. The operand qq identifies register pair BC, DE, HL, or AF, assembled as follows in the object code:

Pair	r
BC	00
DE	01
HL	10
AF	11

M Cycles	T States	4 MHz E.T.
3	10 (4, 3, 3)	2.50

Condition Bits Affected:
None.

Example: If the Stack Pointer contains 1000H, memory location 1000H contains 55H, and location 1001H contains 33H, the instruction

```
POP HL
```

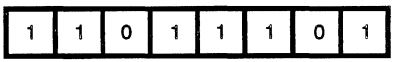
will result in register pair HL containing 3355H, and the Stack Pointer containing 1002H.

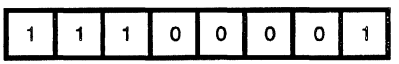
POP IX

Operation: IXH ← (SP+1), IXL ← (SP)

Opcode: POP

Operands: IX





Description: The top two bytes of the external memory LIFO (list-in, first-out) Stack are popped into Index Register IX. The Stack Pointer (SP) register pair holds the 16-bit address of the current "top" of the Stack. This instruction first loads into the low order portion of IX the byte at the memory location corresponding to the contents of SP; then SP is incremented and the contents of the corresponding adjacent memory location are loaded into the high order portion of IX. The SP is now incremented again.

M Cycles	T States	4 MHz E.T.
4	14 (4, 4, 3, 3)	3.50

Condition Bits Affected:
None.

Example: If the Stack Pointer contains 1000H, memory location 1000H contains 55H, and location 1001H contains 33H, the instruction

```
POP IX
```

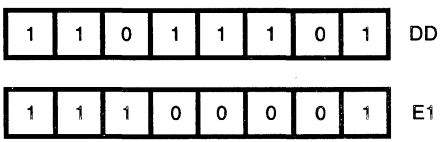
will result in Index Register IX containing 3355H, and the Stack Pointer containing 1002H.

POP IY

Operation: IYH ← (SP+1), IXL ← (SP)

Opcode: POP

Operands: IX



Description: The top two bytes of the external memory LIFO (last-in, first-out) Stack are popped into Index Register IY. The Stack Pointer (SP) register pair holds the 16-bit address of the current "top" of the Stack. This instruction first loads into the low order portion of IY the byte at the memory location corresponding to the contents of SP; then SP is incremented and the contents of the corresponding adjacent memory location are loaded into the high order portion of IY. The SP is now incremented again.

M Cycles	T States	4 MHz E.T.
4	14 (4, 4, 3, 3)	3.50

Condition Bits Affected:
None.

Example: If the Stack Pointer Contains 1000H, memory location 1000H contains 55H, and location 1001H contains 33H, the instruction

```
POP IY
```

will result in Index Register IY containing 3355H, and the Stack Pointer containing 1002H.

A

Z80®
INSTRUCTION DESCRIPTION

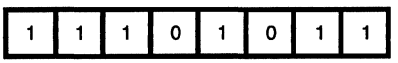
EXCHANGE, BLOCKTRANSFER, AND SEARCH GROUP

EX DE, HL

Operation: DE ↔ HL

Opcode: EX

Operands: DE, HL



Description: The 2-byte contents of register pairs DE and HL are exchanged.

M Cycles	T States	4 MHz E.T.
1	4	1.00

Condition Bits Affected:
None.

Example: If the content of register pair DE is the number 2822H, and the content of the register pair HL is number 499AH, after the instruction

```
EX DE, HL
```

the content of register pair DE will be 499AH and the content of register pair HL will be 2822H.

EX AF, AF'

Operation: AF ↔ AF'

Opcode: EX

Operands: AF, AF'

0	0	0	0	1	0	0	0	0B

Description: The 2-byte contents of the register pairs AF and AF' are exchanged. (Note: register pair AF consists of registers A' and F'.)

M Cycles	T States	4 MHz E.T.
1	4	1.00

Condition Bits Affected:
None.

Example: If the content of register pair AF is number 9900H, and the content of register pair AF' is number 5944H, after the instruction

```
EX AF, AF'
```

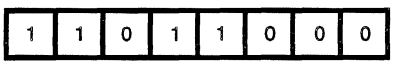
the contents of AF will be 5944H, and the contents of AF' will be 9900H.

EXX

Operation: (BC) ↔ (BC'), (DE') ↔ (DE'), (HL) ↔ (HL')

Opcode: EXX

Operands:



Description: Each 2-byte value in register pairs BC, DE, and HL is exchanged with the 2-byte value in BC', DE', and HL', respectively.

M Cycles	T States	4 MHz E.T.
1	4	1.00

Condition Bits Affected:
None.

Example: If the contents of register pairs BC, DE, and HL are the numbers 445AH, 3DA2H, and 8859H, respectively, and the contents of register pairs BC', DE', and HL' are 0988H, 9300H, and 00E7H, respectively, after the instruction

```
EXX
```

the contents of the register pairs will be as follows: BC': 0988H; DE': 9300H; HL: 00E7H; BC': 445AH; DE': 3DA2H; and HL': 8859H.

EX (SP), HL

Operation: H ↔ (SP+1), L ↔ (SP)

Opcode: EX

Operands: (SP) , HL

1	1	1	0	0	0	1	1	E3

Description: The low order byte contained in register pair HL is exchanged with the contents of the memory address specified by the contents of register pair SP (Stack Pointer), and the high order byte of HL is exchanged with the next highest memory address (SP+1).

M Cycles	T States	4 MHz E.T.
5	19 (4, 3, 4, 3, 5)	4.75

Condition Bits Affected:

None.

Example: If the HL register pair contains 7012H, the SP register pair contains 8856H, the memory location 8856H contains the byte 11H, and the memory location 8857H contains the byte 22H, then the instruction

EX (SP), HL

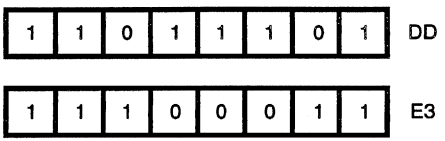
will result in the HL register pair containing number 2211H, memory location 8856H containing the byte 12H, the memory location 8857H containing the byte 70H and the Stack Pointer containing 8856H.

EX (SP), IX

Operation: IXH ↔ (SP+1), IXL ↔ (SP)

Opcode: EX

Operands: (SP) , IX



Description: The low order byte in Index Register IX is exchanged with the contents of the memory address specified by the contents of register pair SP (Stack Pointer), and the high order byte of IX is exchanged with the next highest memory address (SP+1).

M Cycles	T States	4 MHz E.T.
6	23 (4, 4, 3, 4, 3, 5)	5.75

Condition Bits Affected:
None.

Example: If the Index Register IX contains 3988H, the SP register pair Contains 0I00H, the memory location 0100H contains the byte 90H, and memory location 0101H contains byte 48H, then the instruction

EX (SP), IX

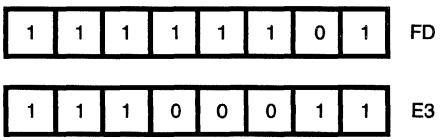
will result in the IX register pair containing number, 4890H, memory location 0100H containing 88H, memory location 0101H containing 39H and the Stack Pointer containing 0100H.

EX (SP), IY

Operation: IYH ↔ (SP+1), IYL ↔ (SP)

Opcode: EX

Operands: (SP), IY



Description: The low order byte in Index Register IY is exchanged with the contents of the memory address specified by the contents of register pair SP (Stack Pointer), and the high order byte of IY is exchanged with the next highest memory address (SP+1).

M Cycles	T States	4 MHz E.T.
6	23 (4, 4, 3, 4, 3, 5)	5.75

Condition Bits Affected:
None.

Example: If the Index Register IY contains 3988H, the SP register pair contains 0100H, the memory location 0100H contains the byte 90H, and memory location 0101H contains byte 48H, then the instruction

```
EX (SP), IY
```

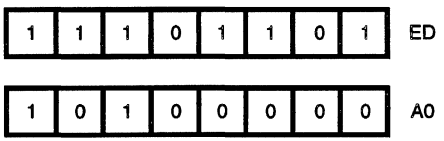
will result in the IY register pair containing number 4890H, memory location 0100H containing 88H, memory location 0101H containing 39H, and the Stack Pointer containing 0100H.

LDI

Operation: (DE) ← (HL), DE ← DE + 1, HL ← HL + 1, BC ← BC –1

Opcode: LDI

Operands: (SP) , HL



Description: A byte of data is transferred from the memory location addressed, by the contents of the HL register pair to the memory location addressed by the contents of the DE register pair. Then both these register pairs are incremented and the BC (Byte Counter) register pair is decremented.

M Cycles	T States	4 MHz E.T.
4	16 (4, 4, 3, 5)	4.00

Condition Bits Affected:

S: Not affected
Z: Not affected
H: Reset
P/V: Set if BC – 1 ≠ 0; reget otherwise
N: Reset
C: Not affected

Example: If the HL register pair contains 1111H, memory location 1111H contains contains the byte 88H, the DE register pair contains 2222H, the memory location 2222H contains byte 66H, and the BC register pair contains 7H, then the instruction

```
LDI
```

Will result in the following contents in register pairs and memory addresses:

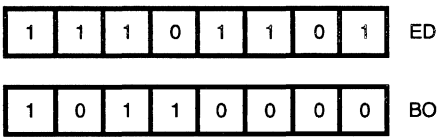
```
  HL    :  1112H
(1111H) :  88H
  DE    :  2223H
(2222H) :  88H
  BC    :  6H
```

LDIR

Operation: (DE) ← (HL), DE ← DE + 1, HL ← HL + 1, BC ← BC –1

Opcode: LDIR

Operands: B8



Description: This 2-byte instruction transfers a byte of data from the memory location addressed by the contents of the HL register pair to the memory location addressed by the DE register pair. Then both these register pairs are incremented and the BC (Byte Counter) register pair is decremented. If decrementing causes the BC to go to zero, the instruction is terminated. If BC is not zero the program counter is decremented by two and the instruction is repeated. Interrupts will be recognized and two refresh cycless will be executed after each data transfer. Note that if BC is set to zero prior to instruction execution, the instruction will loop through 64 Kbytes.

For BC ≠ 0:

M Cycles	T States	4 MHz E.T.
5	21 (4, 4, 4, 3, 5, 5)	5.25

For BC = 0:

M Cycles	T States	4 MHz E.T.
4	16 (4, 4, 3, 5)	4.00

Condition Bits Affected:

S: Not affected
Z: Not affected
H: Reset
P/V: Reset
N: Reset
C: Not affected

Example: If the HL register pair contains 11111H, the DE register pair contains 2222H, the BC register pair contains 0003H, and memory locations have these contents:

(1111H) : 88H (2222H) : 66H
(1112H) : 36H (2223H) : 59H
(1113H) : A5H (2224H) : C5H

then after the execution of

LDIR

the contents of register pairs and memory locations will be:

HL : 1114H
DE : 2225H
BC : 0000H

(1111H)	:	88H	(2222H)	:	88H
(1112H)	:	36H	(2223H)	:	36H
(1113H)	:	A5H	(2224H)	:	A5H

LDD

Operation: (DE) ← (HL), DE ← DE–1, HL ← HL–1, BC← BC –1

Opcode: LDD

Operands:

1	1	1	0	1	1	0	1	ED

1	0	1	0	1	0	0	0	A8

Description: This 2-byte instruction transfers a byte of data from the memory location addressed by the contents of the HL register pair to the memory location addressed by the contents of the DE register pair. Then both of these register pairs including the BC (Byte Counter) register pair are decremented.

M Cycles	T States	4 MHz E.T.
4	16 (4, 4, 3, 5)	4.00

Condition Bits Affected:

S: Not affected
Z: Not affected
H: Reset
P/V: Set if BC – 1 ≠ 0; reset otherwise
N: Reset
C: Not affected

Example: If the HL register pair contains 1111H, memory location 1111H contains the byte 88H, the DE register pair contains 2222H, memory location 2222H contains byte 66H, and the BC register pair contains 7H, then the instruction

```
LDD
```

will result in the following contents in register pairs and memory addresses:

```
  HL   :  1110H
(1111H) :  88H
  DE   :  2221H
(2222H) :  88H
  BC   :  6H
```

LDDR

Operation: (DE) ← (HL), DE ← DE–1, HL ← HL–1, BC ← BC–1

Opcode: LDDR

Operands:

1	1	1	0	1	1	0	1	ED
1	0	1	1	1	0	0	0	B8

Description: This 2-byte instruction transfers a byte of data from the memory location addressed by the contents of the HL register pair to the memory location addressed by the contents of the HL register pair to the memory location addressed by the contents of the DE register pair. Then both of these registers, as well as the BC (Byte Counter) are decremented. If decrementing causes the BC to go to zero, the instruction is terminated. If BC is not zero, the program counter is decremented by two and the instruction is repeated. Interrupts will be recognized and two refresh cycles will be executed after each data transfer. Note that if BC is set to zero prior to instruction execution, the instruction will loop through 64 Kbytes.

For BC ≠ 0:

M Cycles	T States	4 MHz E.T.
5	21 (4, 4, 3, 5, 5)	5.25

For BC = 0:

M Cycles	T States	4 MHz E.T.
5	21 (4, 4, 3, 5, 5)	5.25

Condition Bits Affected:

S: Not Affected:
Z: Not Affected:
H: Reset
P/V: Reset
N: Reset

Example: If the HL register pair contains 1114H, the DE register pair contains 2225H, the BC register pair contains 0003H, and memory locations have these contents:

(1114H) : A5H (2225H) : C5H
(1113H) : 36H (2224H) : 59H
(1112H) : 88H (2223H) : 66H

Then after the execution of

LDDR

the contents of register pairs and memory locations will be:

HL : 1111H
DE : 2222H
DC : 0000H

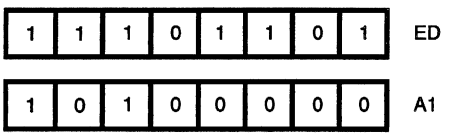
(1114H) : A5H (2225H) : A5H
(1113H) : 36H (2224H) : 36H
(1112H) : 88H (2223H) : 88H

CPI

Operation: A– (HL), HL ← HL +1, BC ← BC –1

Opcode: CPI

Operands:



Description: The contents of the memory location addressed by the HL register is compared with the contents of the Accumulator. In case of a true compare, a condition bit is set. Then HL is incremented and the Byte Counter (register pair BC) is decremented.

M Cycles	T States	4 MHz E.T.
4	16 (4, 4, 3, 5)	4.00

Condition Bits Affected:

S: Set if result is negative; reset otherwise
Z: Set if A = (HL); reset otherwise
H: Set if borrow from bit 4; reset otherwise
P/V: Set if BC – 1 ≠ 0 ; reset otherwise
N: Set
C: Not affected

Example: If the HL register pair contains 1111H, memory location 1111H contains 3BH, the Accumulator contains 3BH, and the Byte Counter contains 0001H. Then after the execution of

```
CPI
```

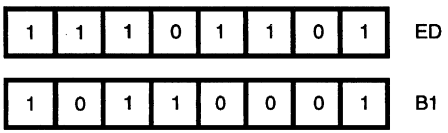
the Byte Counter will contain 0000H, the HL register pair will contain 1112H, the Z flag in the F register will be set, and the P/V flag in the F register will be reset. There will be no effect on the contents of the Accumulator or address 1111H.

CPIR

Operation: A–(HL), HL ← HL+1, BC ← BC–1

Opcode: CPIR

Operands:



Description: The contents of the memory location addressed by the HL register pair is compared with the contents of the Accumulator. In case of a true compare, a condition bit is set. The HL is incremented and the Byte Counter (register pair BC) is decremented. If decrementing causes the BC to go to zero or if A = (HL), the instruction is terminated. If BC is not zero and A ≠ (HL), the program counter is decremented by two and the instruction is repeated. Interrupts will be recognized and two refresh cycles will be executed after each data transfer. Note that if BC is set to zero before instruction execution, the instruction will loop through 64 Kbytes if no match is found.

For BC ≠ 0 and A ≠ (HL):

M Cycles	T States	4 MHz E.T.
5	21 (4, 4, 3, 5, 5)	5.25

For BC = 0 and A = (HL):

M Cycles	T States	4 MHz E.T.
4	16 (4, 4, 3, 5)	4.00

Condition Bits Affected:

S: Set if result is negative; reset otherwise
Z: Set if A = (HL); reset otherwise
H: Set if borrow from bit 4; reset otherwise
P/V: Set if BC – 1 ≠ 0; reset otherwise
N: Set
C: Not Affected:

Example: If the HL register pair contains 1111H, the Accumulator contains F3H, the Byte Counter contains 0007H, and memory locations have these contents:

(1111H) : 52H
(1112H) : 00H
(1113H) : F3H

Then after the execution of:

CPIR

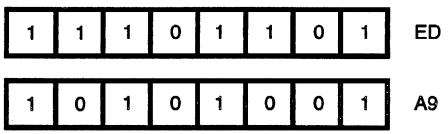
the contents of register pair HL will be 1114H, the contents of the Byte Counter will be 0004H, the P/V flag in the F register will be set, and the Z flag in the F register will be set.

CPD

Operation: A –(HL), HL ← HL –1, BC ← BC –1

Opcode: CPD

Operands:



Description: The contents of the memory location addressed by the HL register pair is compared with the contents of the Accumulator. In case of a true compare, a condition bit is set. The HL and the Byte Counter (register pair BC) are decremented.

M Cycles	T States	4 MHz E.T.
4	16 (4, 4, 3, 5)	4.00

Condition Bits Affected:

S: Set if result is negative; reset otherwise
Z: Set if A = (HL); reset otherwise
H: Set if borrow from bit 4; reset otherwise
P/V: Set if BC – 1 ≠ 0; reset otherwise
N: Set
C: Not Affected:

Example: If the HL register pair contains 1111H, memory location 1111H contains 3BH, the Accumulator contains 3BH, and the Byte Counter contains 0001H. Then after the execution of

```
CPD
```

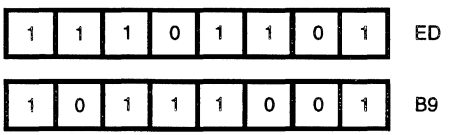
the Byte Counter will contain 0000H, the HL register pair will contain 1110H, the flag in the F register will be set, and the P/V flag in the F register will be reset. There will be no effect on the contents of the Accumulator or address 1111H.

CPDR

Operation: A –(HL), HL ← HL –1, BC ← BC –1

Opcode: CPDR

Operands:



Description: The contents of the memory location addressed by the HL register pair is compared with the contents of the Accumulator. In case of a true compare, a condition bit is set. The HL and BC (Byte Counter) register pairs are decremented. If decrementing causes the BC to go to zero or if A = (HL), the instruction is terminated. If BC is not zero and A = (HL), the program counter is decremented by two and the instruction is repeated. Interrupts will be recognized and two refresh cycles will be executed after each data transfer. Note that if BC is set to zero prior to instruction execution, the instruction will loop through 64 Kbytes if no match is found.

For BC ≠ 0 and A ≠ (HL):

M Cycles	T States	4 MHz E.T.
5	21 (4, 4, 3, 5, 5)	5.25

For BC = 0 and A = (HL):

M Cycles	T States	4 MHz E.T.
4	16 (4, 4, 3, 5)	4.00

Condition Bits Affected:

S: Set if result is negative; reset otherwise
Z: Set if A = (HL); reset otherwise
H: Set if borrow form bit 4; reset otherwise
P/V: Set if BC –1 ≠ 0 ; reset otherwise
N: Set
C: Not Affected:

Example: If the HL register pair contains 1118H, the Accumulator contains F3H, the Byte Counter contains 0007H, and memory locations have these contents.

(1118H) : 52H
(1117H) : 00H
(1116H) : F3H

Then after the execution of

CPDR

the contents of register pair HL will be 1115H, the contents of the Byte Counter will be 0004H, the P/V flag in the F register will be set, and the Z flag in the F register will be set.

A

Z80®
INSTRUCTION DESCRIPTION

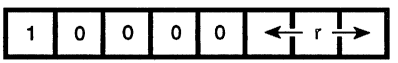
8-BIT ARITHMETIC GROUP

ADD A, r

Operation: A ← A + r

Opcode: ADD

Operands: A, r



Description: The contents of register r are added to the contents of the Accumulator, and the result is stored in the Accumulator. The symbol r identifies the registers A, B, C, D, E, H, or L, assembled as follows in the object code:

Register	r
A	111
B	000
C	001
D	010
E	011
H	100
L	101

M Cycles	T States	4 MHz E.T.
1	4	1.00

Condition Bits Affected:

S: Set if result is negative; reset otherwise
Z: Set if result is zero; reset otherwise
H: Set if carry from bit 3; reset otherwise
P/V: Set if overflow; reset otherwise
N: Reset
C: Set if carry from bit 7; reset otherwise

Example: If the contents of the Accumulator are 44H, and the contents of register C are 11H, after the execution of

```
ADD A,C
```

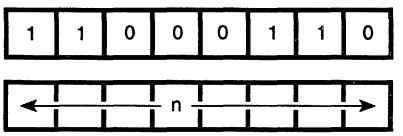
the contents of the Accumulator will be 55H.

ADD A, n

Operation: A ← A + n

Opcode: ADD

Operands: A , n



Description: The integer n is added to the contents of the Accumulator, and the results are stored in the Accumulator.

M Cycles	T States	4 MHz E.T.
2	7 (4, 3)	1.75

Condition Bits Affected:

S: Set if result is negative; reset otherwise
Z: Set if result is zero; reset otherwise
H: Set if carry from bit 3; reset otherwise
P/V: Set if overflow; reset otherwise
N: Reset
C: Set if carry from bit 7; reset otherwise

Example: If the contents of the Accumulator are 23H, after the execution of

```
ADD A, 33H
```

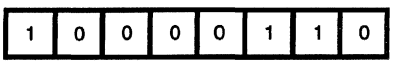
the contents of the Accumulator will be 56H.

ADD A, (HL)

Operation: A ← A + (HL)

Opcode: ADD

Operands: A , (HL)



Description: The byte at the memory address specified by the contents of the HL register pair is added to the contents of the Accumulator, and the result is stored in the Accumulator.

M Cycles	T States	4 MHz E.T.
2	7 (4, 3)	1.75

Condition Bits Affected:

S: Set if result is negative; reset otherwise
Z: Set if result is zero; reset otherwise
H: Set if carry from bit 3; reset otherwise
P/V: Set if overflow; reset otherwise
N: Reset
C: Set if carry from bit 7; reset otherwise

Example: If the contents of the Accumulator are A0H, and the content of the register pair HL is 2323H, and memory location 2323H contains byte 08H, after the execution of

```
ADD A, (HL)
```

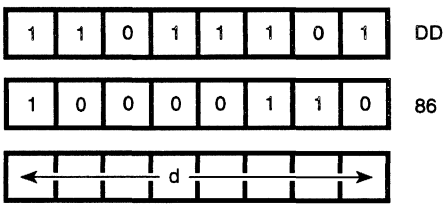
the Accumulator will contain A8H.

ADD A, (IX + d)

Operation: A ← A + (IX+d)

Opcode: ADD

Operands: A, (IX + d)



Description: The contents of the Index Register (register pair IX) is added to a two's complement displacement d to point to an address in memory. The contents of this address is then added to the contents of the Accumulator and the result is stored in the Accumulator.

M Cycles	T States	4 MHz E.T.
5	16 (4, 4, 3, 5, 3)	4.75

Condition Bits Affected:

S: Set if result is negative; reset otherwise
Z: Set if result is zero; reset otherwise
H: Set if carry from bit 3; reset otherwise
P/V: Set if overflow; reset otherwise
N: Reset
C: Set if carry from bit 7; reset otherwise

Example: If the Accumulator contents are 11H, the Index Register IX contains 1000H, and if the content of memory location 1005H is 22H, after the execution of

ADD A, (IX + 5H)

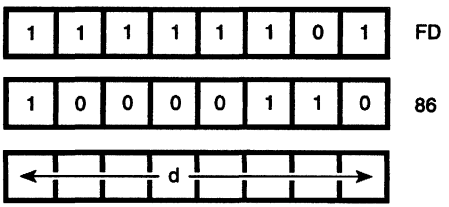
the contents of the Accumulator will be 33H.

ADD A, (IY + d)

Operation: A ← A + (ID+d)

Opcode: ADD

Operands: A, (IY + d)



Description: The contents of the Index Register (register pair IY) is added to a two's complement displacement d to point to an address in memory. The contents of this address is then added to the contents of the Accumulator, and the result is stored in the Accumulator.

M Cycles	T States	4 MHz E.T.
5	16 (4, 4, 3, 5, 3)	4.75

Condition Bits Affected:

S: Set if result is negative; reset otherwise
Z: Set if result is zero; reset otherwise
H: Set if carry from bit 3; reset otherwise
P/V: Set if overflow; reset otherwise
N: Reset
C: Set if carry from bit 7; reset otherwise

Example: If the Accumulator contents are 11H, the Index Register Pair IY contains 1000H, and if the content of memory location 1005H is 22H, after the execution of

ADD A, (IY + 5H)

the contents of the Accumulator will be 33H.

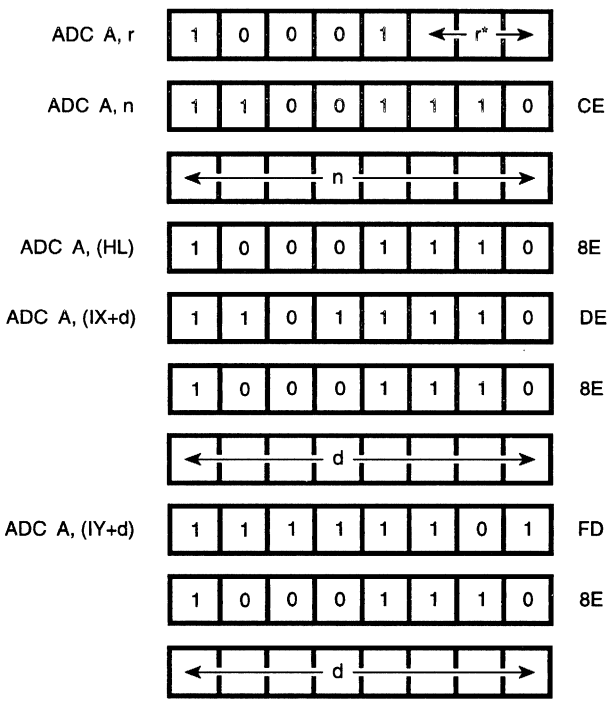
ADC A, s

Operation: A ← A + s + CY

Opcode: ADC

Operands: A, s

This s operand is any of r, n, (HL), (IX+d), or (IY+d) as defined for the analogous ADD instruction. These various possible opcode-operand combinations are assembled as follows in the object code:



*r identifies registers B, C, D, E, H, L, or A assembled as follows in the object code field above:

Register	r
B	000
C	001
D	010
E	011
H	100
L	101
A	111

Description: The s operand, along with the Carry Flag ("C" in the F register) is added to the contents of the Accumulator, and the result is stored in the Accumulator.

Instruction	M Cycle	T States	4 MHz E.T.
ADC A, r	1	4	1.00
ADC A, n	2	7 (4, 3)	1.75
ADC A, (HL)	2	7 (4, 3)	1.75
ADC A, (IX+d)	5	19 (4, 4, 3, 5, 3)	4.75
ADC A, (IY+d)	5	19 (4, 4, 3, 5, 3)	4.75

Condition Bits Affected:

S: Set if result is negative; reset otherwise
Z: Set if result is zero; reset otherwise
H: Set if carry from bit 3; reset otherwise
P/V: Set if overflow; reset otherwise
N: Reset
C: Set if carry from bit 7; reset otherwise

Example: If the Accumulator contents are 16H, the Carry Flag is set, the HL register pair contains 6666H, and address 6666H contains 10H, after the execution of

```
ADC A, (HL)
```

the Accumulator will contain 27H.

SUB s

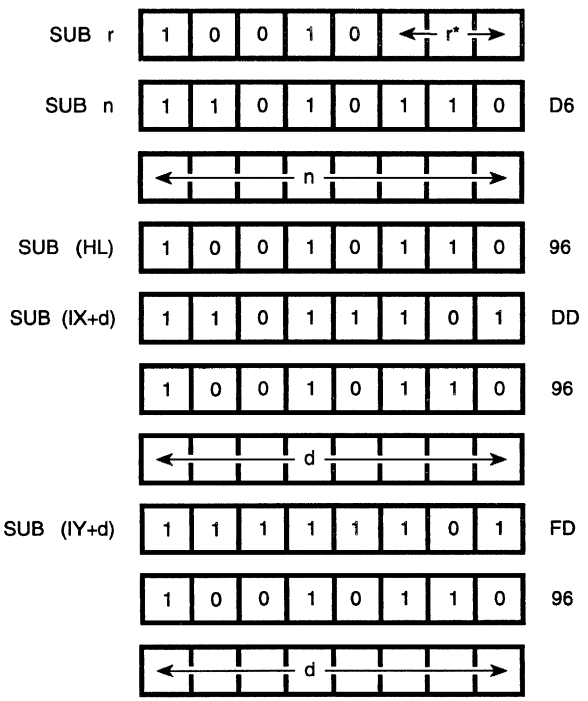
Operation: A ← A – s

Opcode: SUB

Operands: s

A

This s operand is any of r, n, (HL), (IX+d) or (IY+d) as defined for the analogous ADD instruction. These various possible opcode-operand combinations are assembled as follows in the object code:



*r identifies registers B, C, D, E, H, L, or A assembled as follows in the object code field above:

Register	r
B	000
C	001
D	010
E	011
H	100
L	101
A	111

Description: The s operand is subtracted from the contents of the Accumulator, and the result is stored in the Accumulator.

Instruction	M Cycle	T States	4 MHz E.T.
SUB r	1	4	1.00
SUB n	2	7 (4, 3)	1.75
SUB (HL)	2	7 (4, 3)	1.75
SUB (IX+d)	5	19 (4, 4, 3, 5, 3)	4.75
SUB (IY+d)	5	19 (4, 4, 3, 5, 3)	4.75

Condition Bits Affected:

S: Set if result is negative; reset otherwise
Z: Set if result is zero; reset otherwise
H: Set if borrow from bit 4; reset otherwise
P/V: Set if overflow; reset otherwise
N: Set
C: Set if borrow; reset otherwise

Example: If the Accumulator contents are 29H, and register D contains 11H, after the execution of

SUB D

the Accumulator will contain 18H.

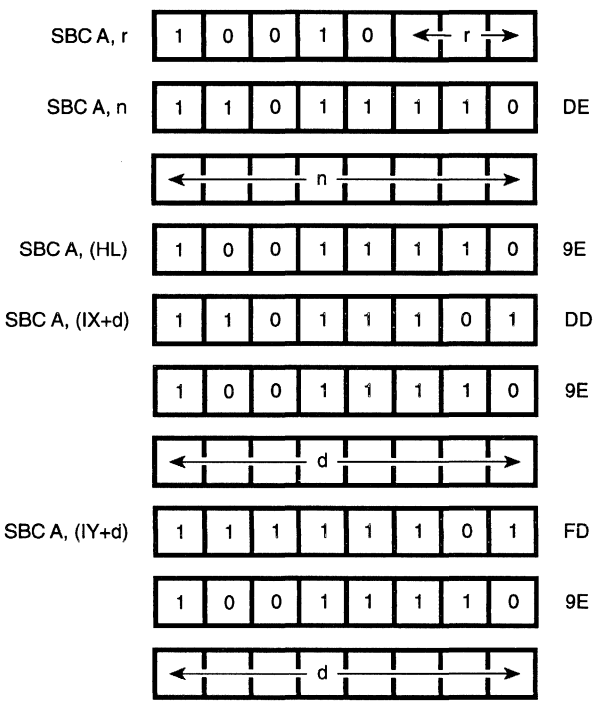
SBC A, s

Operation: A ← A – s – CY

Opcode: SBC

Operands: A, s

The s operand is any of r, n, (HL), (IX+d) or (IY+d) as defined for the analogous ADD instructions. These various possible opcode-operand combinations are assembled as follows in the object code:



*r identifies registers B, C, D, E, H, L, or A assembled as follows in the object code field above:

Register	r
B	000
C	001
D	010
E	011
H	100
L	101
A	111

Description: The s operand, along with the Carry flag ("C" in the F register) is subtracted from the contents of the Accumulator, and the result is stored in the Accumulator.

Instruction	M Cycles	T States	4 MHz E.T.
SBC A, r	1	4	1.00
SBC A, n	2	7 (4, 3)	1.75
SBC A, (HL)	2	7 (4, 3)	1.75
SBC A, (IX+d)	5	19 (4, 4, 3, 5, 3)	4.75
SBC A, (IY+d)	5	19 (4, 4, 3, 5, 3)	4.75

Condition Bits Affected:

S: Set if result is negative; reset otherwise
Z: Set if result is zero; reset otherwise
H: Set if borrow from bit 4; reset otherwise
P/V: Reset if overflow; reset otherwise
N: Set
C: Set if borrow;
reset otherwise

Example: If the Accumulator contains 16H, the carry flag is set, the HL register pair contains 3433H, and address 3433H contains 05H, after the execution of

```
SBC A, (HL)
```

the Accumulator will contain 10H.

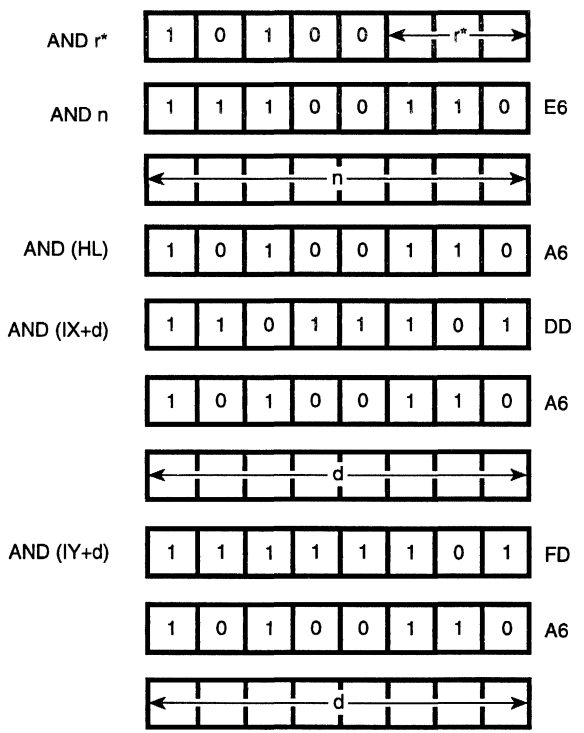
AND s

Operation: A ← A Λ s

Opcode: AND

Operands: s

The s operand is any of r, n, (HL), (IX+d), or (IY+d), as defined for the analogous ADD instructions. These various possible opcode-operand combinations are assembled as follows in the object code:



*r identifies registers B, C, D, E, H, L, or A specified as follows in the assembled object code field above:

Register	r
B	000
C	001
D	010
E	011
H	100
L	101
A	111

Description: A logical AND operation is performed between the byte specified by the s operand and the byte contained in the Accumulator; the result is stored in the Accumulator.

Instruction	M Cycles	T States	4 MHz E.T.
AND r	1	4	1.00
AND n	2	7 (4, 3)	1.75
AND (HL)	2	7 (4, 3)	1.75
AND (IX+d)	5	19 (4, 4, 3, 5, 3)	4.75
AND (IX+d)	5	19 (4, 4, 3, 5, 3)	4.75

Condition Bits Affected:

S: Set if result is negative; reset otherwise
Z: Set if result is zero; reset otherwise
H: Set
P/V: Reset if overflow; reset otherwise
N: Reset
C: Reset

Example: If the B register contains 7BH (0111 1011) and the Accumulator contains C3H (1100 0011) after the execution of

```
AND B
```

the Accumulator will contain 43H (01000011).

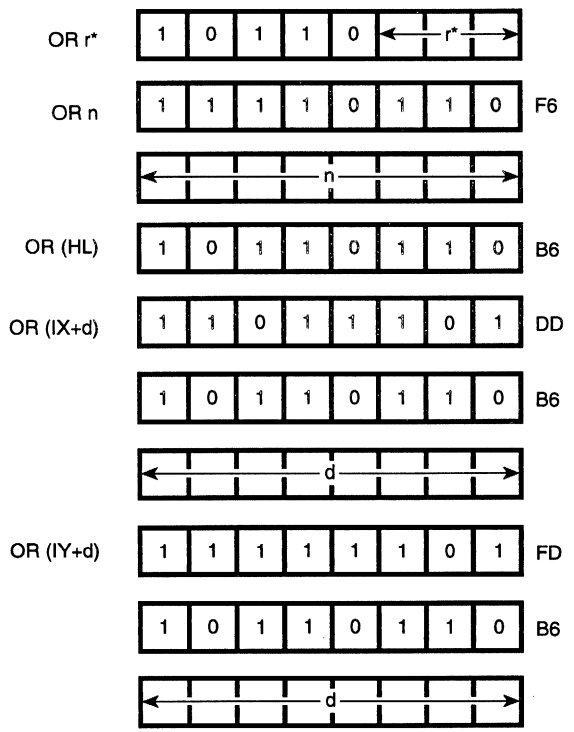
OR s

Operation: A ← A V s

Opcode: OR

Operands: s

The s operand is any of r, n, (HL), (IX+d), or (IY+d), as defined for the analogous ADD instructions. These various possible opcode-operand combinations are assembled as follows in the object code:



*r identifies registers B, C-, D, E, H, L, or A specified as follows in the assembled object code field above:

Register	r
B	000
C	001
D	010
E	011
H	100
L	101
A	111

Description: A logical OR operation is performed between the byte specified by the s operand and the byte contained in the Accumulator; the result is stored in the Accumulator.

Instruction	M Cycles	T States	4 MHz E.T.
OR r	1	4	1.00
OR n	2	7 (4, 3)	1.75
OR (HL)	2	7 (4, 3)	1.75
OR (IX+d)	5	19 (4, 4, 3, 5, 3)	4.75
OR (IY+d)	5	19 (4, 4, 3, 5, 3)	4.75

Condition Bits Affected:

S: Set if result is negative; reset otherwise
Z: Set if result is zero; reset otherwise
H: Reset
P/V: Set if overflow; reset otherwise
N: Reset
C: Reset

Example: If the H register contains 48H (0100 0100) and the Accumulator contains 12H (0001 0010) after the execution of

OR H

the Accumulator will contain 5AH (01011010).

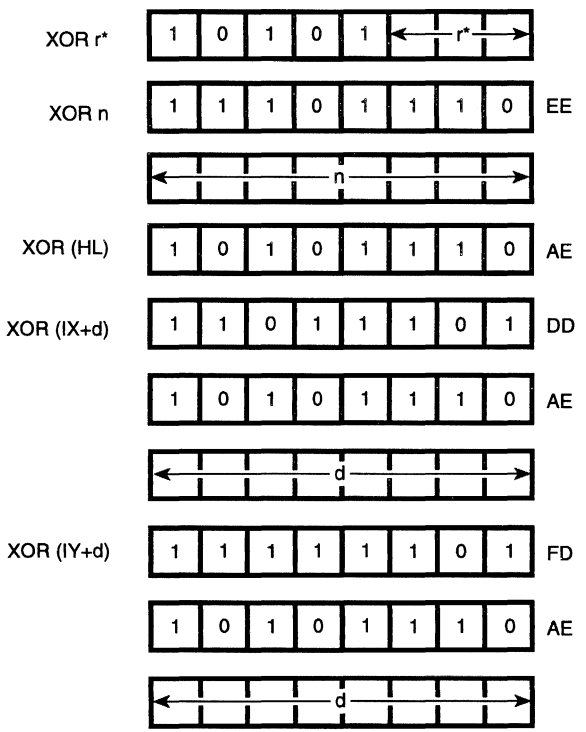
XOR s

Operation: A ← A ⊕ s

Opcode: XOR

Operands: s

The s operand is any of r, n, (HL), (IX+d), or (IY+d), as defined for the analogous ADD instructions. These various possible opcode-operand combinations are assembled as follows in the object code:



*r identifies registers B, C, D, E, H, L, or A specified as follows in the assembled object code field above:

Register	r
B	000
C	001
D	010
E	011
H	100
L	101
A	111

Description: The contents of the s operand are compared with the contents of the Accumulator. If there is a true compare, the Z flag is set. The execution of this instruction does not affect the contents of the Accumulator.

Instruction	M Cycles	T States	4 MHz E.T.
CP r	1	4	1.00
CP n	2	7 (4, 3)	1.75
CP (HL)	2	7 (4, 3)	1.75
CP (IX+d)	5	19 (4, 4, 3, 5, 3)	4.75
CP (IY+d)	5	19 (4, 4, 3, 5, 3)	4.75

Condition Bits Affected:

S: Set if result is negative; reset otherwise
Z: Set if result is zero; reset otherwise
H: Set if borrow from bit 4; reset otherwise
P/V: Set if overflow; reset otherwise
N: Set
C: Set if borrow; reset otherwise

Example: If the Accumulator contains 63H, the HL register pair contains 6000H and memory location 6000H contains 60H, the instruction

CP (HL)

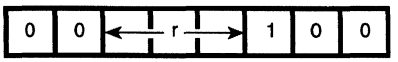
will result in the P/V flag in the F register being reset .

INC r

Operation: r ← r + 1

Opcode: INC

Operands: r



Description: Register r is incremented. r identifies any of the registers A, B, C, D, E, H, or L, assembled as follows in the object code.

Register	r
A	111
B	000
C	001
D	010
E	011
H	100
L	101

M Cycles	T States	4 MHz E.T.
1	4	1.00

Condition Bits Affected:

S: Set if result is negative; reset otherwise
Z: Set if result is zero; reset otherwise
H: Set if carry from bit 3; reset otherwise
P/V: Set if r was 7FH before operation; reset otherwise
N: Reset
C: Not affected

Example: If the contents of register D are 28H, after the execution of

```
INC D
```

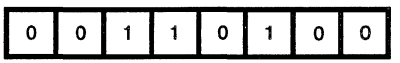
the contents of register D will be 29H.

INC (HL)

Operation: (HL) ← (HL) + 1

Opcode: INC

Operands: (HL)



Description: The byte contained in the address specified by the contents of the HL register pair is incremented.

M Cycles	T States	4 MHz E.T.
3	11 (4, 4, 3)	2.75

Condition Bits Affected:

S: Set if result is negative; reset otherwise
Z: Set if result is zero; reset otherwise
H: Set if carry from bit 3; reset otherwise
P/V: Set if (HL) was 7FH before operation; reset otherwise
N: Reset
C: Not Affected

Example: If the contents of the HL register pair are 3434H, and the contents of address 3434H are 82H, after the execution of

```
INC (HL)
```

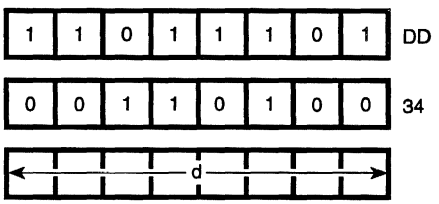
memory location 3434H will contain 83H.

INC (IX+d)

Operation: (IX+d) ← (IX+d) + 1

Opcode: INC

Operands: (IX+d)



Description: The contents of the Index Register IX (register pair IX) are added to a two's complement displacement integer d to point to an address in memory. The contents of this address are then incremented.

M Cycles	T States	4 MHz E.T.
6	23 (4, 4, 3, 5, 4, 3)	5.75

Condition Bits Affected:

S: Set if result is negative; reset otherwise
Z: Set if result is zero; reset otherwise
H: Set if carry from bit 3; reset otherwise
P/V: Set if (IX+d) was 7FH before operation; reset otherwise
N: Reset
C: Not affected

Example: If the contents of the Index Register pair IX are 2020H, and the memory location 2030H contains byte 34H, after the execution of

```
INC (IX+10H)
```

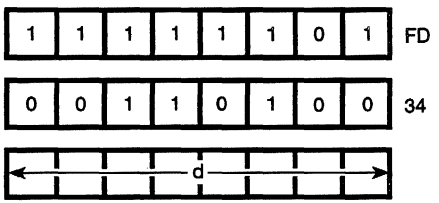
the contents of memory location 2030H will be 35H.

INC (IY+d)

Operation: (IY+d) ← (IY+d) + 1

Opcode: INC

Operands: (IY+d)



Description: The contents of the Index Register IY (register pair IY) are added to a two's complement displacement integer d to point to an address in memory. The contents of this address are then incremented.

M Cycles	T States	4 MHz E.T.
6	23 (4, 4, 3, 5, 4, 3)	5.75

Condition Bits Affected:

S: Set if result is negative; reset otherwise
Z: Set if result is zero; reset otherwise
H: Set if carry from bit 3; reset otherwise
P/V: Set if (IY+d) was 7FH before operation; reset otherwise
N: Reset
C: Not Affected

Example: If the contents of the Index Register pair IY are 2020H, and the memory location 2030H contain byte 34H, after the execution of

```
INC (IY+10H)
```

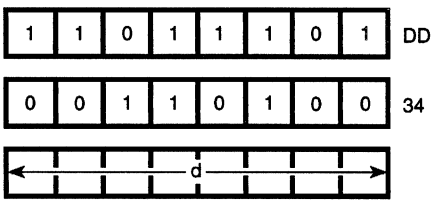
the contents of memory location 2030H will be 35H.

INC (IX+d)

Operation: (IX+d) ← (IX+d) + 1

Opcode: INC

Operands: (IX+d)



Description: The contents of the Index Register IX (register pair IX) are added to a two's complement displacement integer d to point to an address in memory. The contents of this address are then incremented.

M Cycles	T States	4 MHz E.T.
6	23 (4, 4, 3, 5, 4, 3)	5.75

Condition Bits Affected:

S: Set if result is negative; reset otherwise
Z: Set if result is zero; reset otherwise
H: Set if carry from bit 3; reset otherwise
P/V: Set if (IX+d) was 7FH before operation; reset otherwise
N: Reset
C: Not affected

Example: If the contents of the Index Register pair IX are 2020H, and the memory location 2030H contains byte 34H, after the execution of

```
INC (IX+10H)
```

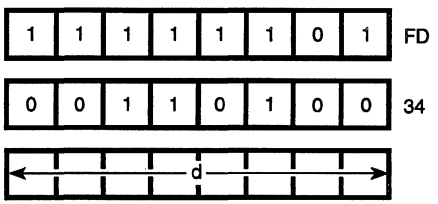
the contents of memory location 2030H will be 35H.

INC (IY+d)

Operation: (IY+d) ← (IY+d) + 1

Opcode: INC

Operands: (IY+d)



Description: The contents of the Index Register IY (register pair IY) are added to a two's complement displacement integer d to point to an address in memory. The contents of this address are then incremented.

M Cycles	T States	4 MHz E.T.
6	23 (4, 4, 3, 5, 4, 3)	5.75

Condition Bits Affected:

S: Set if result is negative; reset otherwise
Z: Set if result is zero; reset otherwise
H: Set if carry from bit 3; reset otherwise
P/V: Set if (IY+d) was 7FH before operation; reset otherwise
N: Reset
C: Not Affected

Example: If the contents of the Index Register pair IY are 2020H, and the memory location 2030H contain byte 34H, after the execution of

```
INC (IY+10H)
```

the contents of memory location 2030H will be 35H.

DEC m

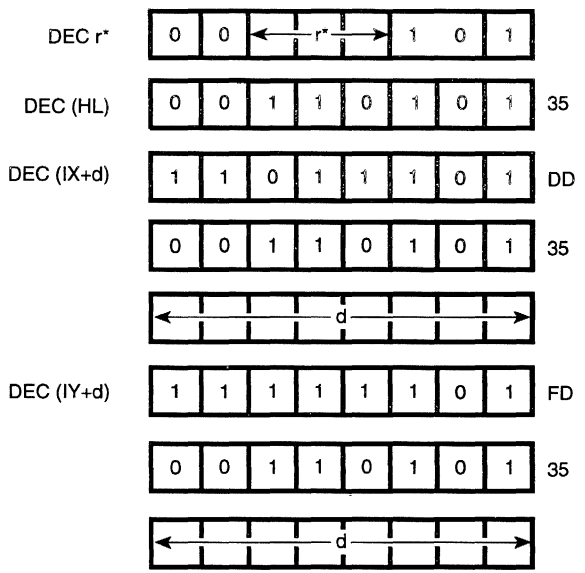
Operation: m ← m – 1

Opcode: DEC

Operands: m

A

The m operand is any of r, (HL),(IX+d), or (IY+d), as defined for the analogous INC instructions. These various possible opcode-operand combinations are assembled as follows in the object code:



*r identifies registers B, C, D, E, H, L, or A assembled as follows in the object code field above:

Register	r
B	000
C	001
D	010
E	011
H	100
L	101
A	111

Description: The byte specified by the m operand is decremented.

Instruction	M Cycles	T States	4 MHz E.T.
DEC r	1	4	1.00
DEC (HL)	3	11 (4, 4, 3)	2.75
DEC (IX+d)	6	23 (4, 4, 3, 5, 4, 3)	5.75
DEC (IY+d)	6	23 (4, 4, 3, 5, 4, 3)	5.75

Condition Bits Affected:

S: Set if result is negative; reset otherwise
Z: Set if result is zero; reset otherwise
H: Set if borrow from bit 4, reset otherwise
P/V: Set if m was 80H before operation; reset otherwise
N: Set
C: Not affected

Example: If the D register contains byte 2AH, after the execution of

```
DEC D
```

register D will contain 29H.

A

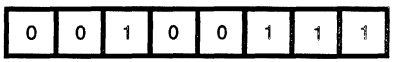
Z80®
INSTRUCTION DESCRIPTION

GENERAL-PURPOSE ARITHMETIC AND CPU CONTROL GROUPS

DAA

Operation: ———

Opcode: DAA



Description: This instruction conditionally adjusts the Accumulator for BCD addition and subtraction operations. For addition (ADD, ADC, INC) or subtraction (SUB, SBC, DEC, NEG), the following table indicates the operation performed:

Operation	C Before DAA	Hex Value In Upper Digit (bit 7-4)	H Before DAA	Hex Value In Lower Digit (bit 3-0)	Number Added To Byte	C After DAA
	0	9-0	0	0-9	00	0
	0	0-8	0	A-F	06	0
	0	0-9	1	0-3	06	0
ADD	0	A-F	0	0-9	60	1
ADC	0	9-F	0	A-F	66	1
INC	0	A-F	1	0-3	66	1
	1	0-2	0	0-9	60	1
	1	0-2	0	A-F	66	1
	1	0-3	1	0-3	66	1
SUB	0	0-9	0	0-9	00	0
SBC	0	0-8	1	6-F	FA	0
DEC	1	7-F	0	0-9	A0	1
NEG	1	6-7	1	6-F	9A	1

M Cycles	T States	4 MHz E.T.
1	4	1.00

Condition Bits Affected:

S: Set if most significant bit of Accumulator is 1 after operation; reset otherwise
Z: Set if Accumulator is zero after operation; reset otherwise
H: See instruction
P/V: Set if Accumulator is even parity after operation; reset otherwise
N: Not affected
C: See instruction

Example: If an addition operation is performed between 15 (BCD) and 27 (BCD), simple decimal arithmetic gives this result:

```
  15
+27
  42
```

But when the binary representations are added in the Accumulator according to standard binary arithmetic,

```
 0001   0101
+0010   0111
 0011   1100   = 3C
```

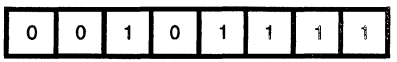
the sum is ambiguous. The DAA instruction adjusts this result so that the correct BCD representation is obtained:

```
 0011   1100
+0000   0110
 0100   0010   = 42
```

CPL

Operation: A ← /A

Opcode: CPL



Description: The contents of the Accumulator (register A) are inverted (one's complement).

M Cycles	T States	4 MHz E.T.
1	4	1.00

Condition Bits Affected:

S: Not affected
Z: Not affected
H: Set
P/V: Not affected
N: Set
C: Not affected

Example: If the contents of the Accumulator are 1011 0100, after the execution of

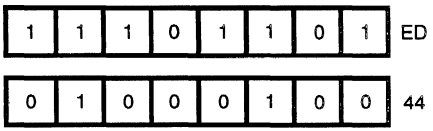
```
CPL
```

the Accumulator contents will be 0100 1011.

NEG

Operation: A ← 0 – A

Opcode: NEG



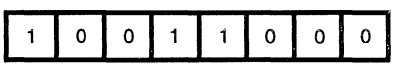
Description: The contents of the Accumulator are negated (two's complement). This is the same as subtracting the contents of the Accumulator from zero. Note that 80H is left unchanged.

M Cycles	T States	4 MHz E.T.
2	8 (4, 4)	2.00

Condition Bits Affected:

S: Set if result is negative; reset otherwise
Z: Set if result is zero; reset otherwise
H: Set if borrow from bit 4; reset otherwise
P/V: Set if Accumulator was 80H before operation; reset otherwise
N: Set
C: Set if Accumulator was not 00H before operation; reset otherwise

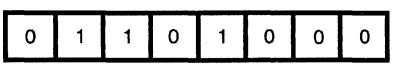
Example: If the contents of the Accumulator are



after the execution of

NEG

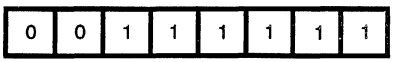
the Accumulator contents will be



CCF

Operation: CY ← /CY

Opcode: CCF



Description: The Carry flag in the F register is inverted.

M Cycles	T States	4 MHz E.T.
1	4	1.00

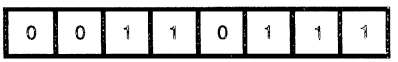
Condition Bits Affected:

S: Not affected
Z: Not affected
H: Previous carry will be copied
P/V: Not affected
N: Reset
C: Set if CY was 0 before operation; reset otherwise

SCF

Operation: CY ← 1

Opcode: SCF



Description: The Carry flag in the F register is set.

M Cycles	T States	4 MHz E.T.
1	4	1.00

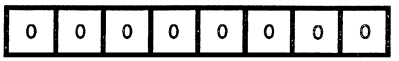
Condition Bits Affected:

S: Not affected
Z: Not affected
H: Reset
P/V: Not affected
N: Reset
C: Set

NOP

Operation: —

Opcode: NOP



Description: The CPU performs no operation during this machine cycle.

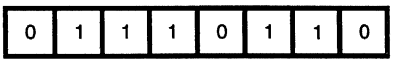
M Cycles	T States	4 MHz E.T.
1	4	1.00

Condition Bits Affected:
None.

HALT

Operation: —

Opcode: HALT



Description: The HALT instruction suspends CPU operation until a subsequent interrupt or reset is received. While in the HALT state, the processor will execute NOP's to maintain memory refresh logic.

M Cycles	T States	4 MHz E.T.
1	4	1.00

Condition Bits Affected:

None.

DI

Operation: IFF ← 0

Opcode: DI

1	1	1	1	0	0	1	1	F3

Description: DI disables the maskable interrupt by resetting the interrupt enable flip-flops (IFF1 and IFF2). Note that this instruction disables the maskable interrupt during its execution.

M Cycles	T States	4 MHz E.T.
1	4	1.00

Condition Bits Affected:
None.

Example: When the CPU executes the instruction

```
DI
```

the maskable interrupt is disabled until it is subsequently re-enabled by an EI instruction. The CPU will not respond to an Interrupt Request (INT) signal.

EI

Operation: IFF ← 1

Opcode: EI

1	1	1	1	1	0	1	1	FB

Description: The enable interrupt instruction will set both interrupt enable flip flops (IFF1 and IFF2) to a logic '1' allowing recognition of any maskable interrupt. Note that during the execution of this instruction and the following instruction, maskable interrupts will be disabled.

M Cycles	T States	4 MHz E.T.
1	4	1.00

Condition Bits Affected:
None.

Example: When the CPU executes instruction

```
EI
RETI
```

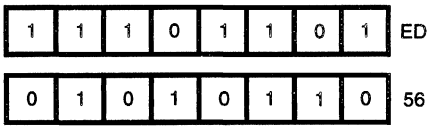
the maskable interrupt will be enabled after the execution of the RETI instruction.

IM 0

Operation: —

Opcode: IM

Operands: 0



Description: The IM 0 instruction sets interrupt mode 0. In this mode, the interrupting device can insert any instruction on the data bus for execution by the CPU. The first byte of a multi-byte instruction is read during the interrupt acknowledge cycle. Subsequent bytes are read in by a normal memory read sequence.

M Cycles	T States	4 MHz E.T.
2	8 (4, 4)	2.00

Condition Bits Affected:

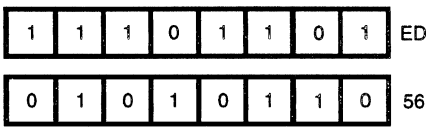
None.

IM 1

Operation: —

Opcode: IM

Operands: 1



Description: The IM 1 instruction sets interrupt mode 1. In this mode, the processor will respond to an interrupt by executing a restart to location 0038H.

M Cycles	T States	4 MHz E.T.
2	8 (4, 4)	2.00

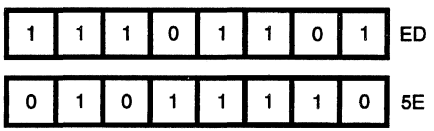
Condition Bits Affected:
None.

IM 2

Operation: —

Opcode: IM

Operands: 2



Description: The IM 2 instruction sets the vectoreed interrupt mode 2. This mode allows an indirect call to any memory location by an 8-bit vector supplied from the peripheral device. This vector then becomes the least significant eight bits of the indirect pointer while the I register in the CPU provides the most significant eight bits. This address points to an address in a vector table which is the starting address for the interrupt service routine.

M Cycles	T States 4 MHz E.T.
2	8 (4, 4) 2.00

Condition Bits Affected:
None.

A

Z80®
INSTRUCTION DESCRIPTION

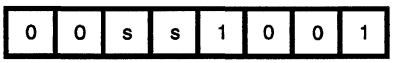
16-BIT ARITHMETIC GROUP

ADD HL, ss

Operation: HL ← HL + ss

Opcode: ADD

Operands: HL, ss



Description: The contents of register pair ss (any of register pairs BC, DE, HL, or SP) are added to the contents of register pair HL and the result is stored in HL. Operand ss is specified as follows in the assembled object code.

Register Pair	ss
BC	00
DE	01
HL	10
SP	11

M Cycles	T States	4 MHz E.T.
3	11 (4, 4, 3)	2.75

Condition Bits Affected:

S: Not affected
Z: Not affected
H: Set if carry out of bit 11; reset otherwise
P/V: Not affected
N: Reset
C: Set if carry from bit 15; reset otherwise

Example: If register pair HL contains the integer 4242H and register pair DE contains 1111H, after the execution of

```
ADD HL, DE
```

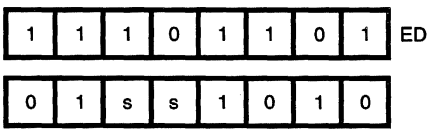
the HL register pair will contain 5353H.

ADC HL, ss

Operation: HL ← HL + ss + CY

Opcode: ADC

Operands: HL, ss



Description: The contents of register pair ss (any of register pairs BC, DE, HL, or SP) are added with the Carry flag (C flag in the F register) to the contents of register pair HL, and the result is stored in HL. Operand ss is specified as follows in the assembled object code.

Register Pair	ss
BC	00
DE	01
HL	10
SP	11

M Cycles	T States	4 MHz E.T.
4	15 (4, 4, 4, 3)	3.75

Condition Bits Affected:

S: Set if result is negative; reset otherwise
Z: Set if result is zero; reset otherwise
R: Set if carry out ot bit 11; reset otherwise
P/V: Set if overflow; reset otherwise
N: Reset
C: Set if carry from bit 15; reset otherwise

Example: If the register pair BC contains 2222H, register pair HL contains 5437H and the Carry Flag is set, after the execution of

```
ADC HL, BC
```

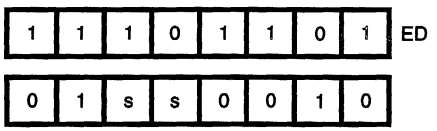
the contents of HL will be 765AH.

SBC HL, ss

Operation: HL ← HI – ss – CY

Opcode: SBC

Operands: HL, ss



Description: The contents of the register pair ss (any of register pairs BC, DE, HL, or SP) and the Carry Flag (C flag in the F register) are subtracted from the contents of register pair HL and the result is stored in HL. Operand ss is specified as follows in the assembled object code.

Register Pair	ss
BC	00
DE	01
HL	10
SP	11

M Cycles	T States	4 MHz E.T.
4	15 (4, 4, 4, 3)	3.75

Condition Bits Affected:

S: Set if result is negative; reset otherwise
Z: Set if result is zero; reset otherwise
H: Set if a borrow from bit 12; reset otherwise
P/V: Set if overflow; reset otherwise
N: Set
C: Set if borrow; reset otherwise

Example: If the contents of the HL, register pair are 9999H, the contents of register pair DE are 1111H, and the Carry flag is set, after the execution of

```
SBC HL, DE
```

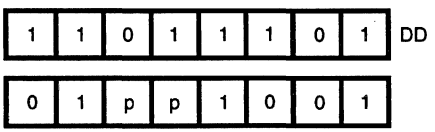
the contents of HL will be 8887H.

ADD IX, pp

Operation: IX ← IX + pp

Opcode: ADD

Operands: IX, pp



Description: The contents of register pair pp (any of register pairs BC, DE, IX, or SP) are added to the contents of the Index Register IX, and the results are stored in IX. Operand pp is specified as follows in the assembled object code.

Register Pair	pp
BC	00
DE	01
IX	10
SP	11

M Cycles	T States	4 MHz E.T.
4	15 (4, 4, 4, 3)	3.75

Condition Bits Affected:

S: Not affected
Z: Not affected
H: Set if carry out of bit 11; reset otherwise
P/V: Not affected
N: Reset
C: Set if carry from bit 15; reset otherwise

Example: If the contents of Index Register IX are 333H and the contents of register pair BC are 5555H, after the execution of

```
ADD IX, BC
```

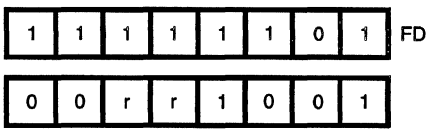
the contents of IX will be 8888H.

ADD IY, rr

Operation: IY ← IY + rr

Opcode: ADD

Operands: IY, rr



Description: The contents of register pair rr (any of register pairs BC, DE, IY, or SP) are added to the contents of Index Register IY, and the result is stored in IY. Operand rr is specified as follows in the assembled object code.

Register Pair	rr
BC	00
DE	01
IY	10
SP	11

M Cycles	T States	4 MHz E.T.
4	15 (4, 4, 4, 3)	3.75

Condition Bits Affected:

S: Not affected
Z: Not affected
H: Set if carry out of bit 11; reset otherwise
P/V: Not affected
N: Reset
C: Set if carry from bit 15; reset otherwise

Example: If the contents of Index Register IY are 333H and the contents of register pair BC are 555H, after the execution of

```
ADD IY, BC
```

the contents of IY will be 8888H.

INC ss

Operation: ss ← ss + 1

Opcode: INC

Operands: ss

0	0	s	s	0	0	1	1

Description: The contents of register pair ss (any of register pairs BC, DE, HL, or SP) are incremented. Operand ss is specified as follows in the assembled object code.

Register Pair	ss
BC	00
DE	01
HL	10
SP	11

M Cycles	T States	4 MHz E.T.
1	6	1.50

Condition Bits Affected:
None.

Example: If the register pair contains 1000H, after the execution of

```
INC HL
```

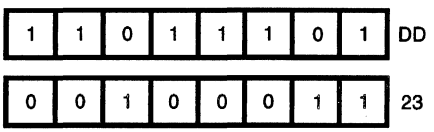
HL will contain 1001H.

INC IX

Operation: IX ← IX + 1

Opcode: INC

Operands: IX



Description: The contents of the Index Register IX are incremented.

M Cycles	T States	4 MHz E.T.
2	10 (4, 6)	2.50

Condition Bits Affected:
None.

Example: If the Index Register IX contains the integer 3300H after the execution of

```
INC IX
```

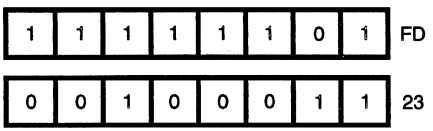
the contents of Index Register IX will be 3301H.

INC IY

Operation: IY ← IY + 1

Opcode: INC

Operands: IY



Description: The contents of the Index Register IY are incremented.

M Cycles	T States	4 MHz E.T.
2	10 (4, 6)	2.50

Condition Bits Affected:
None.

Example: If the contents of the Index Register are 2977H, after the execution of

```
INC IY
```

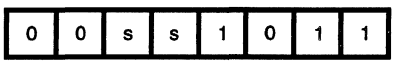
the contents of Index Register IY will be 2978H.

DEC ss

Operation: ss ← ss – 1

Opcode: DEC

Operands: ss



Description: The contents of register pair ss (any of the register pairs BC, DE, HL, or SP) are decremented. Operand ss is specified as follows in the assembled object code.

Pair	ss
BC	00
DE	01
HL	10
SP	11

M Cycles	T States	4 MHz E.T.
1	6	1.50

Condition Bits Affected:
None.

Example: If register pair HL contains 1001H, after the execution of

```
DEC HL
```

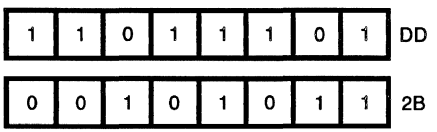
the contents of HL will be 1000H.

DEC IX

Operation: IX ← IX – 1

Opcode: DEC

Operands: IX



Description: The contents of Index Register IX are decremented.

M Cycles	T States	4 MHz E.T.
2	10 (4, 6)	2.50

Condition Bits Affected:
None.

Example: If the contents of Index Register IX are 2006H, after the execution of

```
DEC IX
```

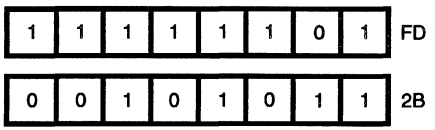
the contents of Index Register IX will be 2005H.

DEC IY

Operation: IY ← IY – 1

Opcode: DEC

Operands: IY



Description: The contents of the Index Register IY are decremented.

M Cycles	T States	4 MHz E.T.
2	10 (4, 6)	2.50

Condition Bits Affected:
None.

Example: If the contents of the Index Register IY are 7649H, after the execution of

DEC IY

the contents of Index Register IY will be 7648H.

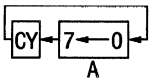
A

Z80®
INSTRUCTION DESCRIPTION

ROTATE AND SHIFT GROUP

RLCA

Operation:



Opcode: RLCA

Operands:

0	0	0	0	0	1	1	1	07

Description: The contents of the Accumulator (register A) are rotated left 1-bit position. The sign bit (bit 7) is copied into the Carry flag and also into bit 0. Bit 0 is the least significant bit.

M Cycles	T States	4 MHz E.T.
1	4	1.00

Condition Bits Affected:

S: Not affected
Z: Not affected
H: Reset
P/V: Not affected
N: Reset
C: Data from bit 7 of Accumulator

Example: If the contents of the Accumulator are

7	6	5	4	3	2	1	0
1	0	0	0	1	0	0	0

after the execution of

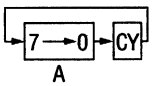
RLCA

the contents of the Accumulator and Carry flag will be

C	7	6	5	4	3	2	1	0
1	0	0	0	1	0	0	0	1

RLA

Operation:



Opcode: RLA

Operands:

0	0	0	1	0	1	1	1	17

Description: The contents of the Accumulator (register A) are rotated left 1-bit position through the Carry flag. The previous content of the Carry flag is copied into bit 0. Bit 0 is the least significant bit.

M Cycles	T States	4 MHz E.T.
1	4	1.00

Condition Bits Affected

S: Not affected
Z: Not affected
H: Reset
P/V: Not affected
N: Reset
C: Data from bit 7 of Accumulator

Example: If the contents of the Accumulator and the Carry flag are

C	7	6	5	4	3	2	1	0
1	0	1	1	1	0	1	1	0

after the execution of

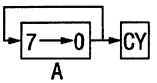
RLA

the contents of the Accumulator and the Carry flag will be

C	7	6	5	4	3	2	1	0
0	1	1	1	0	1	1	0	1

RRCA

Operation:



Opcode: RRCA

Operands:

0	0	0	0	1	1	1	1	0F

Description: The contents of the Accumulator (register A) are rotated right 1-bit position. Bit 0 is copied into the Carry flag and also into bit 7. Bit 0 is the least significant bit.

M Cycles	T States	4 MHz E.T.
1	4	1.00

Condition Bits Affected:

S: Not affected
Z: Not affected
H: Reset
P/V: Not affected
N: Reset
C: Data from bit 0 of Accumulator

Example: If the contents of the Accumulator are

7	6	5	4	3	2	1	0
0	0	0	1	0	0	0	1

after the execution of

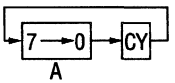
RRCA

the contents of the Accumulator and the Carry flag will be

7	6	5	4	3	2	1	0	C
1	0	0	0	1	0	0	0	1

RRA

Operation:



Opcode: RRA

Operands:

0	0	0	1	1	1	1	1	1F

Description: The contents of the Accumulator (register A) are rotated right 1-bit position through the Carry flag. The previous content of the Carry flag is copied into bit 7. Bit 0 is the least significant bit.

M Cycles	T States	4 MHz E.T.
1	4	1.00

Condition Bits Affected:

S: Not affected
Z: Not affected
H: Reset
P/V: Not affected
N: Reset
C: Data from bit 0 of Accumulator

Example: If the contents of the Accumulator and the Carry Flag are

7	6	5	4	3	2	1	0	C
1	1	1	0	0	0	0	1	0

after the execution of

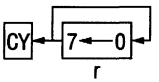
RRA

the contents of the Accumulator and the Carry flag will be

7	6	5	4	3	2	1	0	C
0	1	1	1	0	0	0	0	1

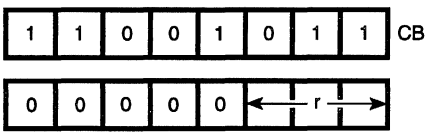
RLC r

Operation:



Opcode: RLC

Operands: r



Description: The contents of register r are rotated left 1-bit position. The content of bit 7 is copied into the Carry flag and also into bit 0. Operand r is specified as follows in the assembled object code:

Register	r
B	000
C	001
D	010
E	011
H	100
L	101
A	111

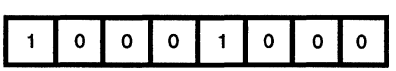
M Cycles	T States	4 MHz E.T.
2	8 (4, 4)	2.00

Condition Bits Affected:

S: Set if result is negative; reset otherwise
Z: Set if result is zero; reset otherwise
H: Reset
P/V: Set if parity even; reset otherwise
N: Reset
C: Data from bit 7 of source register

Example: If the contents of register r are

7 6 5 4 3 2 1 0

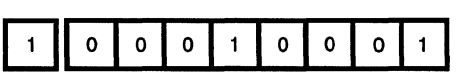


after the execution of

RLC r

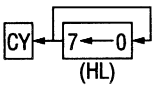
the contents of register r and the Carry flag will be

C 7 6 5 4 3 2 1 0



RLC (HL)

Operation:



Opcode: RLC

Operands: (HL)

1	1	0	0	1	0	1	1	CB
0	0	0	0	0	1	1	0	06

Description: The contents of the memory address specified by the contents of register pair HL are rotated left 1-bit position. The content of bit 7 is copied into the Carry flag and also into bit 0. Bit 0 is the least significant bit.

M Cycles	T States	4 MHz E.T.
4	15 (4, 4, 4, 3)	3.75

Condition Bits Affected:

S: Set if result is negative; reset otherwise
Z: Set if result is zero; reset otherwise
H: Reset
P/V: Set if parity even; reset otherwise
N: Reset
C: Data from bit 7 of source register

Example: If the contents of the HL register pair are 2828H, and the contents of memory location 2828H are

7	6	5	4	3	2	1	0
1	0	0	0	1	0	0	0

after the execution of

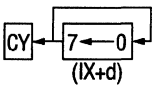
RLC (HL)

the contents of memory location 2828H and the Carry flag will be

C	7	6	5	4	3	2	1	0
1	0	0	0	1	0	0	0	1

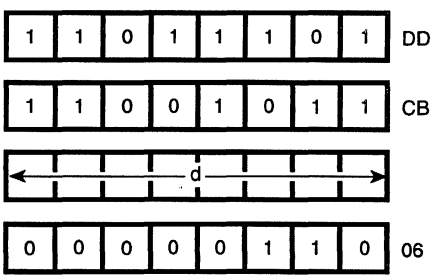
RLC (IX+d)

Operation:



Opcode: RLC

Operands: (IX+d)



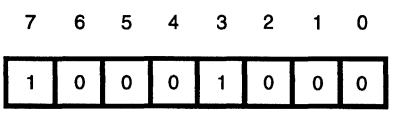
Description: The contents of the memory address specified by the sum of the contents of the Index Register IX and a two's complement displacement integer d, are rotated left 1-bit position. The content of bit 7 is copied into the Carry flag and also, into bit 0. Bit 0 is the least significant bit.

M Cycles	T States	4 MHz E.T.
6	23 (4, 4, 3, 5, 4, 3)	5.75

Condition Bits Affected:

S: Set if result is negative; reset otherwise
Z: Set if result is zero; reset otherwise
H: Reset
P/V: Set if parity even; reset otherwise
N: Reset
C: Data from bit 7 of source register

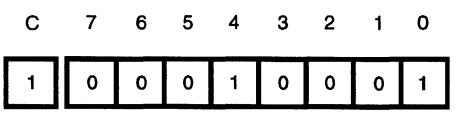
Example: If the contents of the Index Register IX are 1000H, and the contents of memory location 1022H are



after the execution of

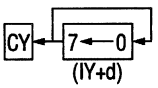
RLC (IX+2H)

the contents of memory location 1002H and the Carry flag will be



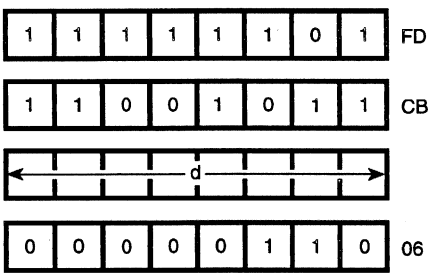
RLC (IY+d)

Operation:



Opcode: RLC

Operands: (IY+d)



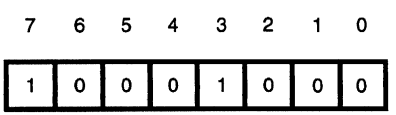
Description: The contents of the memory address specified by the sum of the contents of the Index Register IY and a two's complement displacement integer d are rotated left 1-bit position. The content of bit 7 is copied into the Carry flag and also into bit 0. Bit 0 is the least significant bit.

M Cycles	T States	4 MHz E.T.
6	23 (4, 4, 3, 5, 4, 3)	5.75

Condition Bits Affected:

S: Set if result is negative; reset otherwise
Z: Set if result is zero; reset otherwise
H: Reset
P/V: Set if parity even; reset otherwise
N: Reset
C: Data from bit 7 of source register

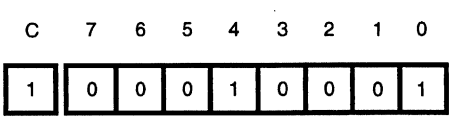
Example: If the contents of the Index Register IY are 1000H, and the contents of memory location 1002H are



after the execution of

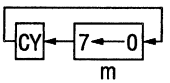
RLC (IY+2H)

the contents of memory location 1002H and the Carry flag will be



RL m

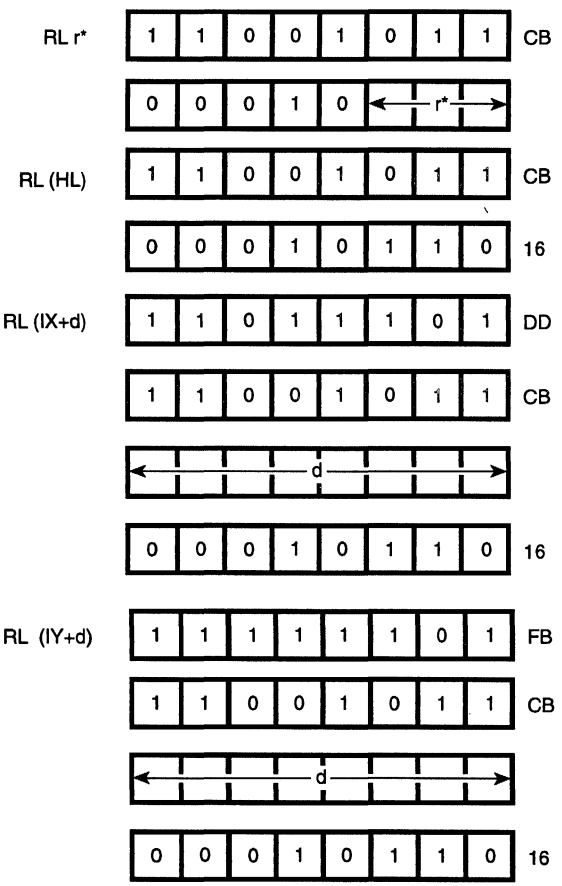
Operation:



Opcode: RL

Operands: m

The m operand is any of r, (HL), (IX+d), or (IY+d), as defined for the analogous RLC instructions. These various possible opcode-operand combinations are specified as follows in the assembled object code:



*r identifies registers B, C, D, E, H, L, or A specified as follows in the assembled object code above:

Register	r
B	000
C	001
D	010
E	011
H	100
L	101
A	111

A

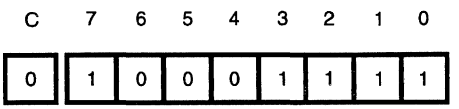
Description: The contents of the m operand are rotated left 1-bit position. The content of bit 7 is copied into the Carry flag and the previous content of the Carry flag is copied into bit 0.

Instruction	M Cycles	T States	4 MHz E.T.
RL r	2	8 (4, 4)	2.00
RL(HL)	4	15 (4, 4, 4, 3)	3.75
RL(IX+d)	6	23 (4, 4, 3, 5, 4, 3)	5.75
RL(IY+d)	6	23 (4, 4, 3, 5, 4, 3)	5.75

Condition Bits Affected:

S: Set if result is negative; reset otherwise
Z: Set if result is zero; reset otherwise
H: Reset
P/V: Set if parity even; reset otherwise
N: Reset
C: Data from bit 7 of source register

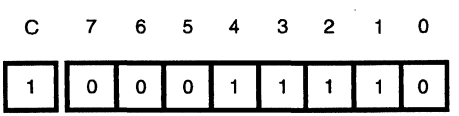
Example: If the contents of register D and the Carry flag are



after the execution of

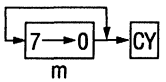
RL D

the contents of register D and the Carry flag will be



RRC m

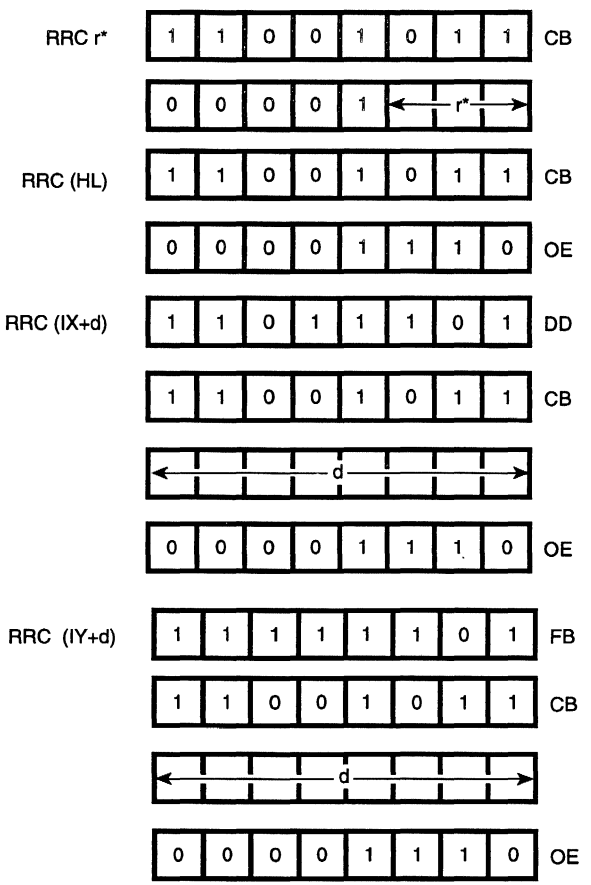
Operation:



Opcode: RRC

Operands: m

The m operand is any of r, (HL), (IX+d), or (IY+d), as defined for the analogous RLC instructions. These various possible opcode-operand combinations are specified as follows in the assembled object code:



*r identifies registers B, C, D, E, H, L, or A specified as follows in the assembled object code above:

Register	r
B	000
C	001
D	010
E	011
H	100
L	101
A	111

A

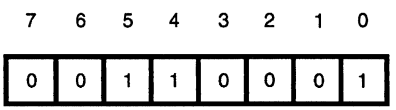
Description: The contents of the m operand are rotated right 1-bit position. The content of bit 0 is copied into the Carry flag and also into bit 7. Bit 0 is the least significant bit.

Instruction	M Cycles	T States	4 MHz E.T.
RRC r	2	8 (4, 4)	2.00
RRC (HL)	4	15 (4, 4, 4, 3)	3.75
RRC (IX+d)	6	23 (4, 4, 3, 5, 4, 3)	5.75
RRC (IY+d)	6	23 (4, 4, 3, 5, 4, 3)	5.75

Condition Bits Affected:

S: Set if result is negative; reset otherwise
Z: Set if result is zero; reset otherwise
H: Reset
P/V: Set if parity even; reset otherwise,
N: Reset
C: Data from bit 0 of source register

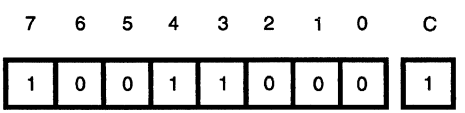
Example: If the contents of register A are



after the execution of

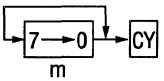
RRC A

the contents of register A and the Carry flag will be



RR m

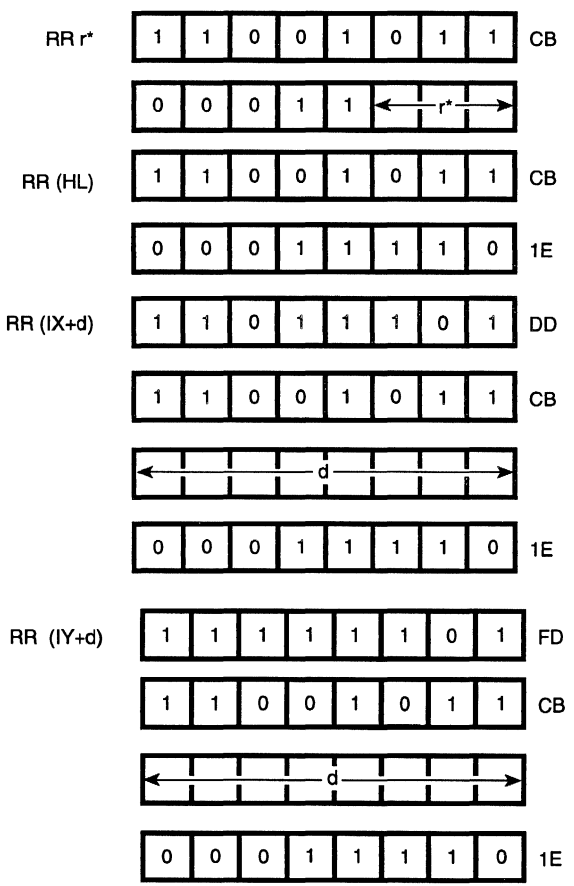
Operation:



Opcode: RR

Operands: m

The m operand is any of r, (HL), (IX+d), or (IY+d), as defined for the analogous RLC instructions. These various possible opcode-operand combinations are specified as follows in the assembled object code:



*r identifies registers B, C, D, E, H, L ,or A specified as follows in the assembled object code above:

Register	r
B	000
C	001
D	010
E	011
H	100
L	101
A	111

Description: The contents of operand m are rotated right 1-bit position through the Carry flag. The content of bit 0 is copied into the Carry flag and the previous content of the Carry flag is copied into bit 7. Bit 0 is the least significant bit.

Instruction	M Cycles	T States	4 MHz E.T.
RR r	2	8 (4, 4)	2.00
RR (HL)	4	15 (4, 4, 4, 3)	3.75
RR (IX+d)	6	23 (4, 4, 3, 5, 4, 3)	5.75
RR (IY+d)	6	23 (4, 4, 3, 5, 4, 3)	5.75

Condition Bits Affected:

S: Set if result is negative; reset otherwise
Z: Set if result is zero; reset otherwise
H: Reset
P/V: Set if parity even; reset otherwise,
N: Reset
C: Data from bit 0 of source register

Example: If the contents of the HL register pair are 4343H, and the contents of memory location 4343H and the Carry flag are

7	6	5	4	3	2	1	0	C
1	1	0	1	1	1	0	1	0

after the execution of

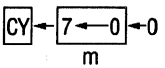
RR (HL)

the contents of location 4343H and the Carry flag will be

7	6	5	4	3	2	1	0	C
0	1	1	0	1	1	1	0	1

SLA m

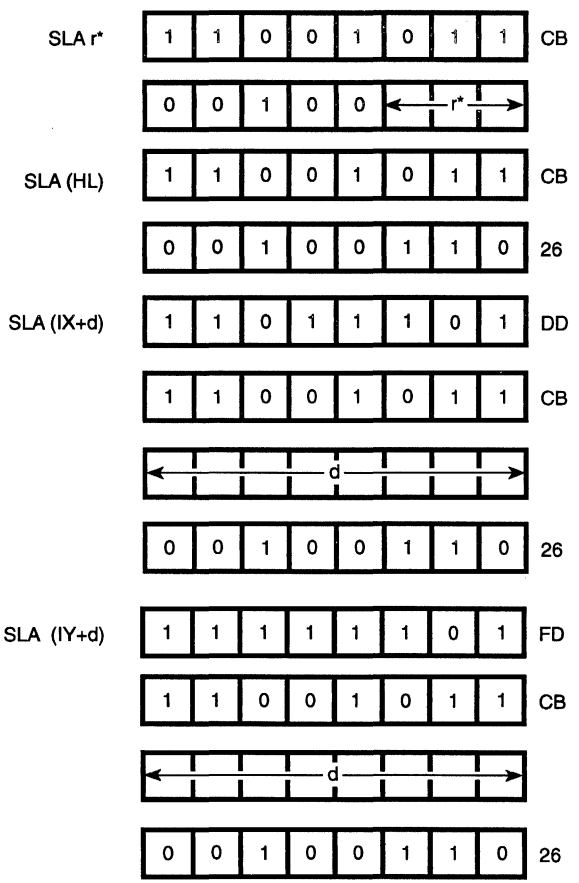
Operation:



Opcode: SLA

Operands: m

The m operand is any of r, (HL), (IX+d), or (IY+d), as defined for the analogous RLC instructions. These various possible opcode-operand combinations are specified as follows in the assembled object code:



r identifies registers B, C, D, E, H, L, or A specified as follows in the assembled object code field above:

Register	r
B	000
C	001
D	010
E	011
H	100
L	101
A	111

A

Description: An arithmetic shift left 1-bit position is performed on the contents of operand m. The content of bit 7 is copied into the Carry flag. Bit 0 is the least significant bit.

Instruction	M Cycles	T States	4 MHz E.T.
SLAr	2	8 (4, 4)	2.00
SLA(HL)	4	15 (4, 4, 4, 3)	3.75
SLA(IX+d)	6	23 (4, 4, 3, 5, 4, 3)	5.75
SLA(IY+d)	6	23 (4, 4, 3, 5, 4, 3)	5.75

Condition Bits Affected:

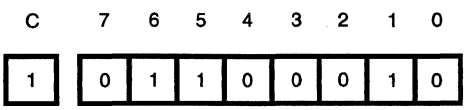
S: Set if result is negative; reset otherwise
Z: Set if result is zero; reset otherwise
H: Reset
P/V: Set if parity is even; reset otherwise
N: Reset
C: Data from bit 7

Example: If the contents of register L are

after the execution of

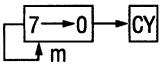
SLA L

the contents of register L and the Carry flag will be



SRA m

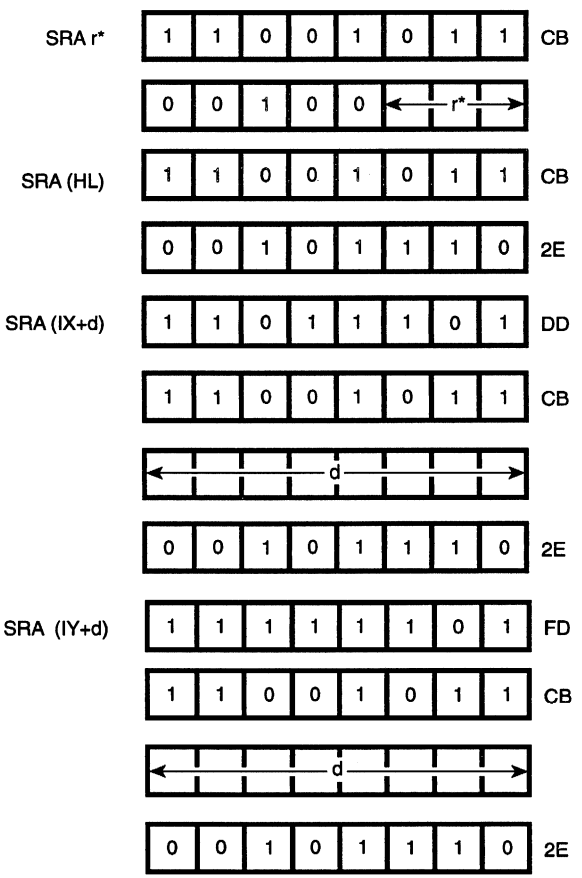
Operation:



Opcode: SRA

Operands: m

The m operand is any of r, (HL), (IX+d), or (IY+d), as defined for the analogous RLC instructions. These various possible opcode-operand combinations are specified as follows in the assembled object code:



*r identifies registers B, C, D, E, H, L, or A specified as, follows in the assembled object code field above:

Register	r
B	000
C	001
D	010
E	011
H	100
L	101
A	111

A

Description: An arithmetic shift right 1-bit position is performed on the contents of operand m. The content of bit 0 is copied into the Carry flag and the previous content of bit 7 is unchanged. Bit 0 is the least significant bit.

Instruction	M Cycles	T States	4 MHz E.T.
SRA r	2	8 (4, 4)	2.00
SRA (HL)	4	15 (4, 4, 4, 3)	3.75
SRA (IX+d)	6	23 (4, 4, 3, 5, 4, 3)	5.75
SRA (IY+d)	6	23 (4, 4, 3, 5, 4, 3)	5.75

Condition Bits Affected:

S: Set if result is negative; reset otherwise
Z: Set if result is zero; reset otherwise
H: Reset
P/V: Set if parity is even; reset otherwise
N: Reset
C: Data from bit 0 of source register

Example: If the contents of the Index Register IX are 1000H, and the contents of memory location 1003H are

7	6	5	4	3	2	1	0
1	0	1	1	1	0	0	0

after the execution of

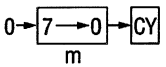
SRA (IX+3H)

the contents of memory location 1003H and the Carry flag will be

7	6	5	4	3	2	1	0	C
1	1	0	1	1	1	0	0	0

SRL m

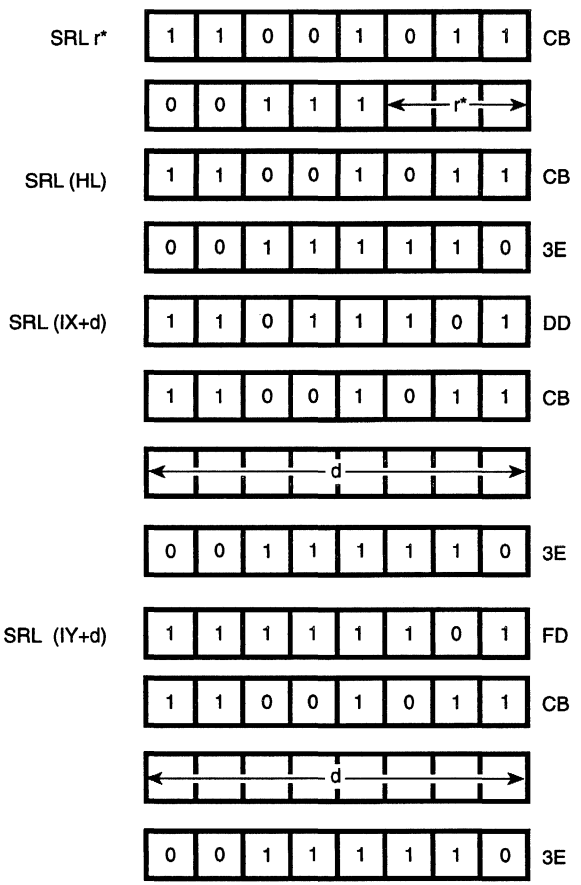
Operation:



Opcode: SRL

Operands: m

The operand m is any of r, (HL), (IX+d), or (IY+d), as defined for the analogous RLC instructions. These various possible opcode-operand combinations are specified as follows in the assembled object code:



*r identifies registers B, C, D, E, H, L, or A specified as follows in the assembled object code fields above:

Description: The contents of operand m are shifted right 1-bit position. The content of bit 0 is copied into the Carry flag, and bit 7 is reset. Bit 0 is the least significant bit.

Instruction	M Cycles	T States	4 MHz E.T.
SRL r	2	8 (4, 4)	2.00
SRL (HL)	4	15 (4, 4, 4, 3)	3.75
SRL (IX+d)	6	23 (4, 4, 3, 5, 4, 3)	5.75
SRL (IY+d)	6	23 (4, 4, 3, 5, 4, 3)	5.75

Condition Bits Affected:

S: Reset
Z: Set if result is zero; reset otherwise
H: Reset
P/V: Set if parity is even; reset otherwise
N: Reset
C: Data from bit 0 of source register

Example: If the contents of register B are

7	6	5	4	3	2	1	0
1	0	0	0	1	1	1	1

after the execution of

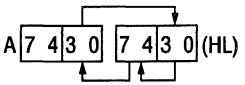
SRL B

the contents of register B and the Carry flag will be

7	6	5	4	3	2	1	0	C
0	1	0	0	0	1	1	1	1

RLD

Operation:



Opcode: RLD

Operands:

1	1	1	0	1	1	0	1	ED
0	1	1	0	1	1	1	1	6F

Description: The contents of the low order four bits (bits 3, 2, 1, and 0) of the memory location (HL) are copied into the high order four bits (7, 6, 5, and 4) of that same memory location; the previous contents of those high order four bits are copied into the low order four bits of the Accumulator (register A); and the previous contents of the low order four bits of the Accumulator are copied into the low order four bits of memory location (HL). The contents of the high order bits of the Accumulator are unaffected. Note: (HL) means the memory location specified by the contents of the HL register pair.

M Cycles	T States	4 MHz E.T.
5	18 (4, 4, 3, 4, 3)	4.50

Condition Bits Affected:

S: Set if Accumulator is negative after operation; reset otherwise
Z: Set if Accumulator is zero after operation; reset otherwise
H: Reset
P/V: Set if parity of Accumulator is even after operation; reset otherwise
N: Reset
C: Not affected

Example: If the contents of the HL register pair are 5000H, and the contents of the Accumulator and memory location 5000H are

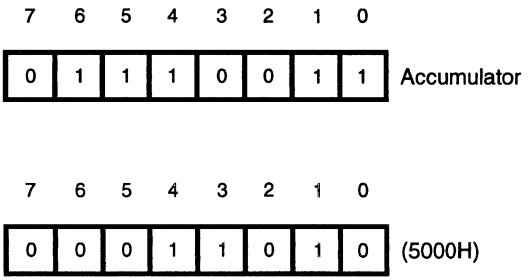
7	6	5	4	3	2	1	0	
0	1	1	1	1	0	1	0	Accumulator

7	6	5	4	3	2	1	0	
0	0	1	1	0	0	0	1	(5000H)

after the execution of

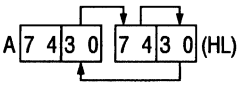
RLD

the contents of the Accumulator and memory location 5000H will be



RRD

Operation:



Opcode: RRD

Operands:

1	1	1	0	1	1	0	1	ED
0	1	1	0	0	1	1	1	67

Description: The contents of the low order four bits (bits 3, 2, 1, and 0) of memory location (HL) are copied into the low order four bits of the Accumulator (register A); the previous contents of the low order four bits of the Accumulator are copied into the high order four bits (7, 6, 5, and 4) of location (HL); and the previous contents of the high order four bits of (HL) are copied into the low order four bits of (HL). The contents of the high order bits of the Accumulator are unaffected. Note: (HL) means the memory location specified by the contents of the HL register pair .

M Cycles	T States	4 MHz E.T.
5	18 (4, 4, 3, 4, 3)	4.50

Condition Bits Affected:

S: Set if Accumulator is negative after operation; reset otherwise
Z: Set if Accumulato. is zero after operation; reset otherwise
H: Reset
P/V: Set if parity of Accumulator is even after operation; reset otherwise
N: Reset
C: Not affected

Example: If the contents of the HL register pair are 5000H, and the contents of the Accumulator and memory location 5000H are

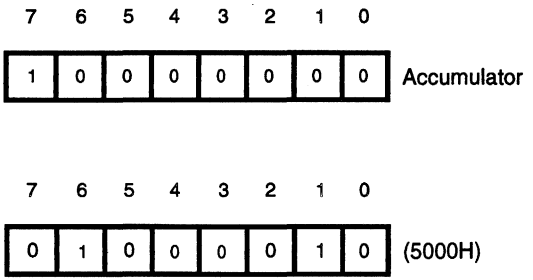
7	6	5	4	3	2	1	0	
1	0	0	0	0	1	0	0	Accumulator

7	6	5	4	3	2	1	0	
0	0	1	0	0	0	0	0	(5000H)

after the execution of

RRD

the contents of the Accumulator and memory location 5000H will be



A

A

Z80®
INSTRUCTION DESCRIPTION

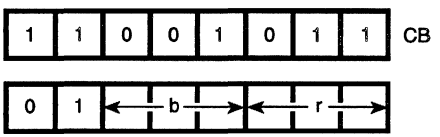
BIT SET, RESET, AND TEST GROUP

BIT b, r

Operation: Z ← /rb

Opcode: BIT

Operands: b, r



Description: This instruction tests bit b in register r and sets the Z flag accordingly. Operands b and r are specified as follows in the assembled object code:

Bit Tested	b	Register	r
0	000	B	000
1	001	C	001
2	010	D	010
3	011	E	011
4	100	H	100
5	101	L	101
6	110	A	111
7	111		

M Cycles	T States	4 MHz E.T.
2	8 (4, 4)	4.50

Condition Bits Affected:

S: Unknown
Z: Set if specified bit is 0; reset otherwise
H: Set
P/V: Unknown
N: Reset
C: Not affected

Example: If bit 2 in register B contains 0, after the execution of

```
BIT 2, B
```

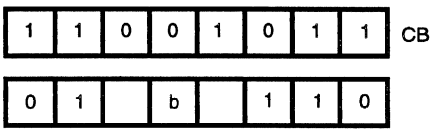
the Z flag in the F register will contain 1, and bit 2 in register B will remain 0. Bit 0 in register B is the least significant bit.

BIT b, (HL)

Operation: Z ← (/HL)b

Opcode: BIT

Operands: b, (HL)



Description: This instruction tests bit b in the memory location specified by the contents of the HL register pair and sets the Z flag accordingly. Operand b is specified as follows in the assembled object code:

Bit Tested	
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

M Cycles	T States	4 MHz E.T.
3	12 (4, 4, 4) 4	3.00

Condition Bits Affected:

S: Unknown
Z: Set if specified Bit is 0; reset otherwise
H: Set
P/V: Unknown
H: Reset
C: Not affected

Example: If the HL register pair contains 4444H, and bit 4 in the memory location 444H contains 1, after the execution of

```
BIT 4, (HL)
```

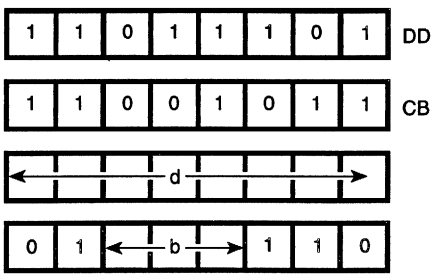
the Z flag in the F register will contain 0, and bit 4 in memory location 4444H will still contain 1. (Bit 0 in memory location 4444H is the least significant bit.)

BIT b, (IX+d)

Operation: Z ← /(IX+d)b

Opcode: BIT

Operands: b, (IX+d)



Description: This instruction tests bit b in the memory location specified by the contents of register pair IX combined with the two's complement displacement d and sets the Z flag accordingly. Operand b is specified as follows in the assembled object code.

Bit Tested	b
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

M Cycles	T States	4 MHz E.T.
5	20 (4, 4, 3, 5, 4)	5.00

Condition Bits Affected:

S: Unknown
Z: Set if specified Bit is 0; reset otherwise
H: Set
P/V: Unknown
N: Reset
C: Not affected

Example: If the contents of Index Register IX are 2000H, and bit 6 in memory location 2004H contains 1, after the execution of

```
BIT 6, (IX+4H)
```

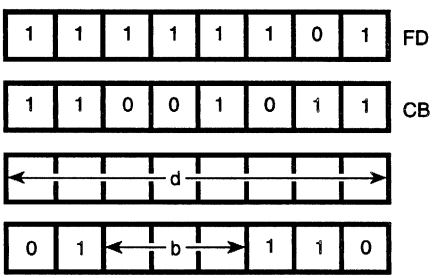
the Z flag in the F register will contain 0, and bit 6 in memory location 2004H will still contain 1. (Bit 0 in memory location 2004H is the least significant bit.)

BIT b, (IY+d)

Operation: Z ← /(IY+d)b

Opcode: BIT

Operands: b, (IY+d)



Description: This instruction tests bit b in the memory location specified by the content of register pair IY combined with the two's complement displacement d and sets the Z flag accordingly. Operand b is specified as follows in the assembled object code.

Bit Tested	b
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

M Cycles	T States	4 MHz E.T.
5	20 (4, 4, 3, 5, 4)	5.00

Condition Bits Affected:

S: Unknown
Z: Set if specified Bit is 0; reset otherwise
H: Set
P/V: Unknown
H: Reset
C: Not affected

Example: If the contents of Index Register are 2000H, and bit 6 in memory location 2004H contains 1, after the execution of

```
BIT 6, (IY+4H)
```

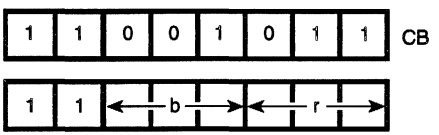
the Z flag in the F register sill contain 0, and bit 6 in memory location 2004H will still contain 1. (Bit 0 in memory location 2004H is the least significant bit.)

SET b, r

Operation: rb ← 1

Opcode: SET

Operands: b, r



Description: Bit b in register r (any of registers B, C, D, E, H, L, or A) is set. Operands b and r are specified as follows in the assembled object code:

Bit	b	Register	r
0	000	B	000
1	001	C	001
2	010	D	010
3	011	E	011
4	100	H	100
5	101	L	101
6	110	A	111
7	111		

M Cycles	T States	4 MHz E.T.
2	8 (4, 4)	2.00

Condition Bits Affected:
None.

Example: After the execution of

```
SET 4, A
```

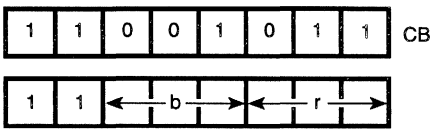
bit 4 in register A will be set. (Bit 0 is the least significant bit.)

SET b, (HL)

Operation: (HL)b ← 1

Opcode: SET

Operands: b, (HL)



Description: Bit b in the memory location addressed by the contents of register pair HL is set. Operand b is specified as follows in the assembled object code:

Bit Tested	b
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

M Cycles	T States	4 MHz E.T.
4	15 (4, 4, 4, 3)	3.75

Condition Bits Affected:
None.

Example: If the contents of the HL register pair are 3000H, after the execution of

```
SET 4, (HL)
```

bit 4 in memory location 3000H will be 1. (Bit 0 in memory location 3000H is the least significant bit.)

SET b, (IX+d)

Operation: (IX+d)b ← 1

Opcode: SET

Operands: b, (IX+d)

Description: Bit b in the memory location addressed by the sum of the contents of the IX register pair and the two's complement integer d is set. Operand b is specified as follows in the assembled object code:

Bit Tested	b
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

M Cycles	T States	4 MHz E.T.
6	23 (4, 4, 3, 5, 4, 3)	5.75

Condition Bits Affected:
None.

Example: If the contents of Index Register are 2000H, after the execution of

SET 0, (IX + 3H)

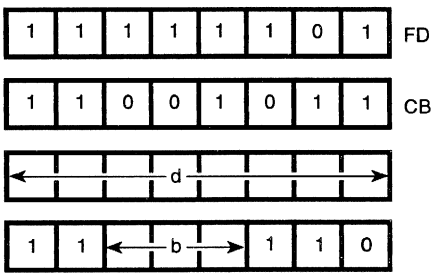
bit 0 in memory location 2003H will be 1. (Bit 0 in memory location 2003H is the least significant bit.)

SET b, (IY+d)

Operation: (IY + d) b ← 1

Opcode: SET

Operands: b, (IY + d)



Description: Bit b in the memory location addressed by the sum of the contents of the IY register pair and the two's complement displacement d is set. Operand b is specified as follows in the assembled object code:

Bit Tested	b
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

M Cycles	T States	4 MHz E.T.
6	23 (4, 4, 3, 5, 4, 3)	5.75

Condition Bits Affected:
None

Example: If the contents of Index Register IY are 2000H, after the execution of

SET 0,(IY+3H)

bit 0 in memory location 2003H will be 1. (Bit 0 in memory location 2003H is the least significant bit.)

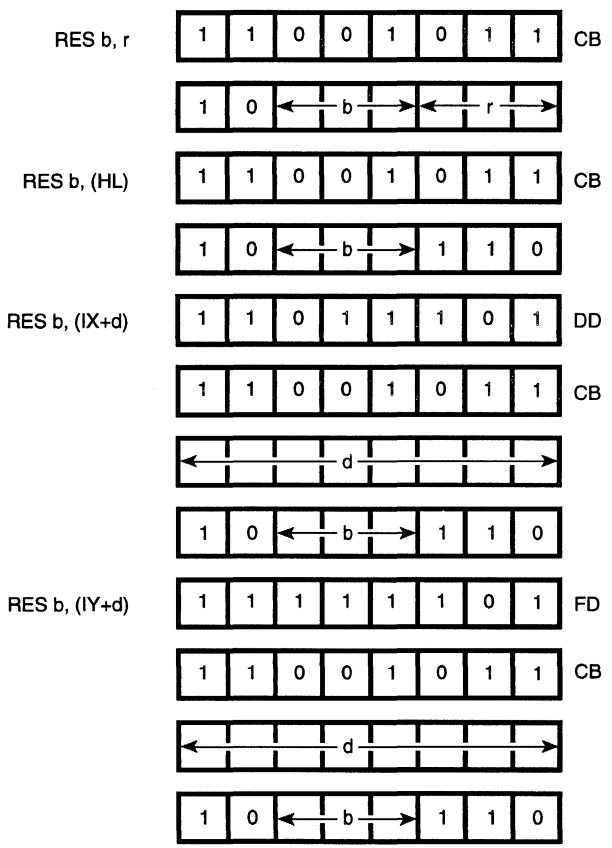
RES b, m

Operation: sb ← 0

Opcode: RES

Operands: b, m

Operand b is any bit (7 through 0) of the contents of the m operand, (any of r, (HL), (IX+d) or (IY+d)) as defined for the analogous SET instructions. These various possible opcode-operand combinations are assembled as follows in the object code:



Bit	b	Register	r
0	000	B	000
1	001	C	001
2	010	D	010
3	011	E	011
4	100	H	100
5	101	L	101
6	110	A	111
7	111		

Description: Bit b in operand m is reset.

Instruction	M Cycles	T States	4 MHz E.T.
RES r	4	8 (4, 4)	2.00
RES (HL)	4	15 (4, 4, 4, 3)	3.75
RES (IX+d)	6	23 (4, 4, 3, 5, 4, 3)	5.75
RES (IY+d)	6	23 (4, 4, 3, 5, 4, 3)	5.75

Condition Bits Affected:

None.

Example: After the execution of

```
RES 6, D
```

bit 6 in register D will be reset. (Bit 0 in register D is the least significant bit.)

A

Z80®
INSTRUCTION DESCRIPTION

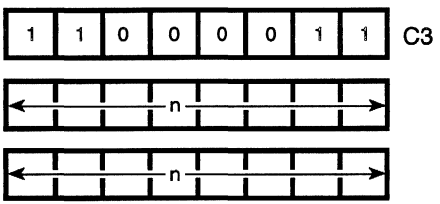
JUMP GROUP

JP nn

Operation: PC ← nn

Opcode: JP

Operands: nn



Note: The first operand in this assembled object code is the low order byte of a two-byte address.

Description: Operand nn is loaded into register pair PC (Program Counter). The next instruction is fetched from the location designated by the new contents of the PC.

M Cycles	T States	4 MHz E.T.
3	10 (4, 3, 3)	2.50

Condition Bits Affected:

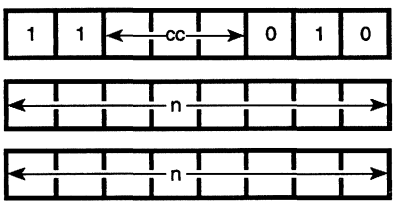
None.

JP cc, nn

Operation: IF cc true, PC← nn

Opcode: JP

Operands: cc, nn



Note: The first n operand in this assembled object code is the low order byte of a 2-byte memory address.

Description: If condition cc is true, the instruction loads operand nn into register pair PC (Program Counter), and the program continues with the instruction beginning at address nn. If condition cc is false, the Program Counter is incremented as usual, and the program continues with the next sequential instruction. Condition cc is programmed as one of eight status which corresponds to condition bits in the Flag Register (register F). These eight status are defined in the table below which also specifies the corresponding cc bit fields in the assembled object code.

cc	Condition	Relevant Flag
000	NZ non zero	Z
001	Z zero	Z
010	NC no carry	C
011	C carry	C
100	PO parity odd	P/V
101	PE parity even	P/V
110	P sign positive	S
111	M sign negative	S

M Cycles	T States	4 MHz E.T.
3	10 (4, 3, 3)	2.50

Condition Bits Affected:

None.

Example: If the Carry flag (C flag in the F register) is set and the contents of address 1520 are 03H, after the execution of

```
JP C, 1520H
```

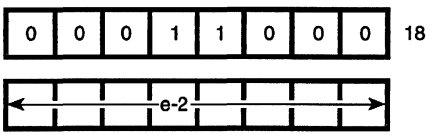
the Program Counter will contain 1520H, and on the next machine cycle the CPD will fetch from address 1520H the byte 03H.

JR e

Operation: PC ← PC + e

Opcode: JR

Operands: e



Description: This instruction provides for unconditional branching to other segments of a program. The value of the displacement e is added to the Program Counter (PC) and the next instruction is fetched from the location designated by the new contents of the PC. This jump is measured from the address of the instruction opcode and has a range of −126 to +129 bytes. The assembler automatically adjusts for the twice incremented PC.

M Cycles	T States	4 MHz E.T.
3	12 (4, 3, 5)	3.00

Condition Bits Affected:

None.

Example: To jump forward five locations from address 480, the following assembly language statement is used:

```
JR $+5
```

The resulting object code and final PC value is shown below:

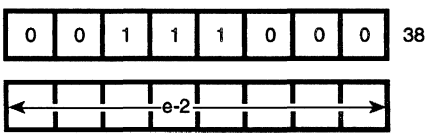
Location	Instruction
480	18
481	03
482	–
483	–
484	–
485	← PC after jump

JR C, e

Operation: If C = 0, continue
If C = 1, PC ← PC + e

Opcode: JR

Operands: C, e



Description: This instruction provides for conditional branching to other segments of a program depending on the results of a test on the Carry Flag. If the flag is equal to a '1', the value of the displacement e is added to the Program Counter (PC) and the next instruction is fetched from the location designated by the new contents of the PC. The jump is measured from the address of the instruction opcode and has a range of -126 to +129 bytes. The assembler automatically adjusts for the twice incremented PC.

If the flag is equal to a '0', the next instruction to be executed is taken from the location following this instruction.

If condition is met

M Cycles	T States	4 MHz E.T.
3	12 (4, 3, 5)	3.00

If condition is not met:

M Cycles	T States	4 MHz E.T.
2	7 (4, 3)	1.75

Condition Bits Affected:

None.

Example: The Carry flag is set and it is required to jump back four locations from 480. The assembly language statement is:

```
JR C, $ – 4
```

The resulting object code and final PC value is shown below:

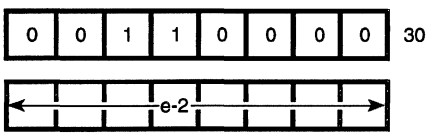
Location	Instruction
47C	← PC after jump
47D	–
47E	–
47F	–
480	38
481	FA (two's complement – 6)

JR (NC), e

Operation: If C = 1, continue
If C = 0, PC ← PC + e

Opcode: JR

Operands: NC, e



Description: This instruction provides for conditional branching to other segments of a program depending on the results of a test on the Carry Flag. If the flag is equal to '0', the value of the displacement e is added to the Program Counter (PC) and the next instruction is fetched from the location designated by the new contents of the PC. The jump is measured from the address of the instruction opcode and has a range of -126 to +129 bytes. The assembler automatically adjusts for the twice incremented PC.

If the flag is equal to a '1', the next instruction to be executed is taken from the location following this instruction.

If the condition is met:

M Cycles	T States	4 MHz E.T.
3	12 (4, 3, 5)	3.00

If the condition is not met:

M Cycles	T States	4 MHz E.T.
7	7 (4, 3)	1.75

Condition Bits Affected:

None.

Example: The Carry Flag is reset and it is required to repeat the jump instruction. The assembly language statement is:

```
JR NC, $
```

The resulting object code and PC after the jump are shown below:

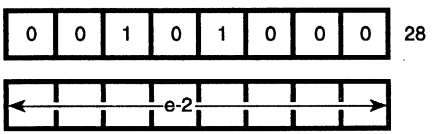
Location	Instruction
480	30 ← PC after jump
481	00

JR Z, e

Operation: If Z = 0, continue
If Z = 1, PC ← PC + e

Opcode: JR

Operands: Z, e



Description: This instruction provides for conditional branching to other segments of a program depending on the results of a test on the Zero Flag. If the flag is equal to a '1', the value of the displacement e is added to the Program Counter (PC) and the next instruction is fetched from the location designated by the new contents of the PC. The jump is measured from the address of the instruction opcode and has a range of -126 to +129 bytes. The assembler automatically adjusts for the twice incremented PC.

If the Zero Flag is equal to a '0', the next instruction to be executed is taken from the location following this instruction.

If the condition is met:

M Cycles	T States	4 MHz E.T.
3	12 (4, 3, 5)	3.00

If the condition is not met;

M Cycles	T States	4 MHz E.T.
2	7 (4, 3)	1.75

Condition Bits Affected:

None.

Example: The Zero Flag is set and it is required to jump forward five locations from address 300. The following assembly language statement is used:

```
JR Z, $ + 5
```

The resulting object code and final PC value is shown below:

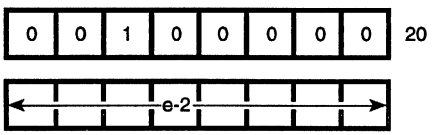
Location	Instruction
300	28
301	03
302	–
303	–
304	–
305	← PC after jump

JR NZ, e

Operation: If Z = 1, continue
If Z = 0, pc ← pc + e

Opcode: JR

Operands: NZ, e



Description: This instruction provides for conditional branching to other segments of a program depending on the results of a test on the Zero Flag. If the flag is equal to a '0', the value of the displacement e is added to the Program Counter (PC) and the next instruction is fetched from the location designated by the new contents of the PC. The jump is measured from the address of the instruction opcode and has a range of –126 to +129 bytes. The assembler automatically adjusts for the twice incremented PC.

If the Zero Flag is equal to a '1', the next instruction to be executed is taken from the location following this instruction.

If the condition is met

M Cycles	T States	4 MHz E.T.
3	12 (4, 3, 5)	3.00

If the condition is not met:

M Cycles	T States	4 MHz E.T.
2	7 (4, 3)	1.75

Condition Bits Affected:

None.

Example: The Zero Flag is reset and it is required to jump back four locations from 480. The assembly language statement is:

JR NZ, $ – 4

The resulting object code and final PC value is shown below:

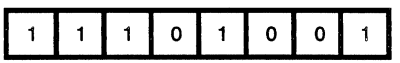
Location	Instruction
47C	← PC after jump
47D	–
47E	–
47F	–
480	20
481	FA (2' complement – 6)

JP (HL)

Operation: pc ← hL

Opcode: JP

Operands: (HL)



Description: The Program Counter (register pair PC) is loaded with the contents of the HL register pair. The next instruction is fetched from the location designated by the new contents of the PC.

M Cycles	T States	4 MHz E.T.
1	4	1.00

Condition Bits Affected:

None.

Example: If the contents of the Program Counter are 1000H and the contents of the HL register pair are 4800H, after the execution of

```
JP (HL)
```

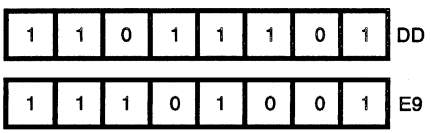
the contents of the Program Counter will be 4800H.

JP (IX)

Operation: pc ← IX

Opcode: JP

Operands: (IX)



Description: The Program Counter (register pair PC) is loaded with the contents of the IX Register Pair. The next instruction is fetched from the location designated by the new contents of the PC.

M Cycles	T States	4 MHz E.T.
2	8 (4, 4)	2.00

Condition Bits Affected:

None.

Example: If the contents of the Program Counter are 1000H, and the contents of the IX Register Pair are 4800H, after the execution of

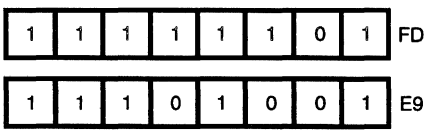
```
JP (IX)
```

the contents of the Program Counter will be 4800H.

JP (IY)

Operation: PC ← IY

Opcode: JP
Operands: (IY)



Description: The Program Counter (register pair PC) is loaded with the contents of the IY Register Pair. The next instruction is fetched from the location designated by the new contents of the PC.

M Cycles	T States	4 MHz E.T.
2	8 (4, 4)	2.00

Condition Bits Affected:

None.

Example: If the contents of the Program Counter are 1000H and the contents of the IY Register Pair are 4800H, after the execution of

```
JP (IY)
```

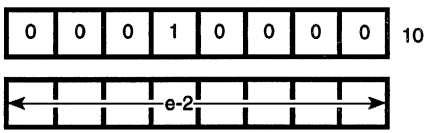
the contents of the Program Counter will be 4800H.

DJNZ, e

Operation: –

Opcode: DJNZ

Operands: e



Description: This instruction is similar to the conditional jump instructions except that a register value is used to determine branching. The B register is decremented and if a non zero value remains, the value of the displacement e is added to the Program Counter (PC). The next instruction is fetched from the location designated by the new contents of the PC. The jump is measured from the address of the instruction opcode and has a range of –126 to +129 bytes. The assembler automatically adjusts for the twice incremented PC.

If the result of decrementing leaves B with a zero value, the next instruction to be executed is taken from the location following this instruction.

if B ≠ 0:

M Cycles	T States	4 MHz E.T.
3	13 (5 ,3, 5)	3.25

If B = 0:

M Cycles	T States	4 MHz E.T.
2	8 (5, 3)	2.00

Condition Bits Affected:

None.

Example: A typical software routine is used to demonstrate the use of the DJNZ instruction. This routine moves a line from an input buffer (INBUF) to an output buffer (OUTBUF). It moves the bytes until it finds a CR, or until it has moved 80 bytes, whichever occurs first.

```
        LD      B, 80           ;Set up counter
        LD      HL, Inbuf       ;Set up pointers
        LD      DE, Outbuf

LOOP:   LD      A, (HL)         ;Get next byte from
                                ;input buffer
        LD      (DE), A         ;Store in output buffer
        CP      ODH             ;Is it a CR?
        JR      Z, DONE         ;Yes finished
        INC     HL              ;Increment pointers
        INC     DE
        DJNZ    LOOP            ;Loop back if 80
                                ;bytes have not
                                ;been moved
DONE:
```

A

Z80®
INSTRUCTION DESCRIPTION

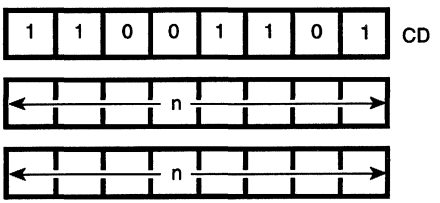
CALL AND RETURN GROUP

CALL nn

Operation: (SP–1) ← PCH, (SP–2) ← PCL, PC ← nn

Opcode: CALL

Operands: nn



Note: The first of the two n operands in the assembled object code above is the least significant byte of a 2-byte memory address.

Description: The current contents of the Program Counter (PC) are pushed onto the top of the external memory stack. The operands nn are then loaded into the PC to point to the address in memory where the first opcode of a subroutine is to be fetched. (At the end of the subroutine, a RETurn instruction can be used to return to the original program flow by popping the top of the stack back into the PC.) The push is accomplished by first decrementing the current contents of the Stack Pointer (register pair SP), loading the high-order byte of the PC contents into the memory address now pointed to by the SP; then decrementing SP again, and loading the low-order byte of the PC contents into the top of stack. Note: Because this is a 3-byte instruction, the Program Counter will have been incremented by three before the push is executed.

M Cycles	T States	4 MHz E.T.
5	17 (4, 3, 4, 3, 3)	4.25

Condition Bits Affected:

None.

Example: If the contents of the Program Counter are 1A47H, the contents of the Stack Pointer are 3002H, and memory locations have the contents:

Location	Contents
1A47H	CDH
1A48H	35H
1A49H	21H

then if an instruction fetch sequence begins, the 3-byte instruction CD3521H will be fetched to the CPU for execution. The mnemonic equivalent of this is

CALL 2135H

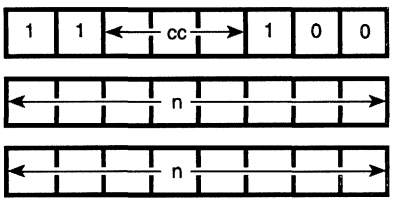
After the execution of this instruction, the contents of memory address 3001H will be 1AH, the contents of address 3000H will be 4AH, the contents of the Stack Pointer will be 3000H, and the contents of the Program Counter will be 2135H, pointing to the address of the first opcode of the subroutine now to be executed.

CALL cc, nn

Operation: IF cc true: (sp–1) ← PCH
(sp–2) ← PCL, pc ← nn

Opcode: CALL

Operands: cc, nn



Note: The first of the two n operands in the assembled object code above is the least significant byte of the 2-byte memory address.

Description: If condition cc is true, this instruction pushes the current contents of the Program Counter (PC) onto the top of the external memory stack, then loads the operands nn into PC to point to the address in memory where the first opcode of a subroutine is to be fetched. (At the end of the subroutine, a RETurn instruction can be used to return to the original program flow by popping the top of the stack back into PC.) If condition cc is false, the Program Counter is incremented as usual, and the program continues with the next sequential instruction. The stack push is accomplished by first decrementing the current contents of the Stack Pointer (SP), loading the high-order byte of the PC contents into the memory address now pointed to by SP; then decrementing SP again, and loading the low-order byte of the PC contents into the top of the stack. Note: Because this is a 3-byte instruction, the Program Counter will have been incremented by three before the push is executed. Condition cc is programmed as one of eight status which corresponds to condition bits in the Flag Register (register F). These eight status are defined in the table below, which also specifies the corresponding cc bit fields in the assembled object code:

cc	Condition	Relevant Flag
000	NZ non zero	Z
001	Z zero	Z
010	NC non carry	C
011	C carry	Z
100	PO parity odd	P/V
101	PE parity even	P/V
110	P sign positive	S
111	M sign negative	S

If cc is true:

M Cycles	T States	4 MHz E.T.
5	17 (4, 3, 4, 3, 3)	4.25

If cc is false:

M Cycles	T States	4 MHz E.T.
3	10 (4, 3, 3)	2.50

Condition Bits Affected:

None.

Example: If the C Flag in the F register is reset, the contents of the Program Counter are 1A47H, the contents of the Stack Pointer are 3002H, and memory locations have the contents:

Location	Contents
1A47H	D4H
1448H	35H
1A49H	21H

then if an instruction fetch sequence begins, the 3-byte instruction D43521H will be fetched to the CPU for execution. The mnemonic equivalent of this is

CALL NC, 2135H

After the execution of this instruction, the contents of memory address 3001H will be 1AH, the contents of address 3000H will be 4AH, the contents of the Stack Pointer will be 3000H, and the contents of the Program Counter will be 2135H, pointing to the address of the first opcode of the subroutine now to be executed.

RET

Operation: pCL ← (sp), pCH ← (sp+1)

Opcode: RET

1	1	0	0	1	0	0	1	C9

Description: The byte at the memory location specified by the contents of the Stack Pointer (SP) register pair are moved to the low order eight bits of the Program Counter (PC). The SP is now incremented and the byte at the memory location specified by the new contents of this instruction will be fetched from the memory location specified by the PC. This instruction is normally used to return to the main line program at the completion of a routine entered by a CALL instruction.

M Cycles	T States	4 MHz E.T.
3	10 (4, 3, 3)	2.50

Condition Bits Affected:

None.

Example: If the contents of the Program Counter are 3535H, the contents of the Stack Pointer are 2000H, the contents of memory location 2000H are B5H, and the contents of memory location of memory location 2001H are 18H, then after the execution of

```
RET
```

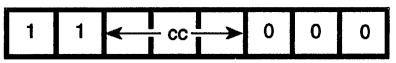
the contents of the Stack Pointer will be 2002H and the contents of the Program Counter will be 18B5H, pointing to the address of the next program opcode to be fetched.

RET cc

Operation: If cc true: PCL ← (sp), pCH ← (sp+1)

Opcode: RET

Operands: cc



Description: If condition cc is true, the byte at the memory location specified by the contents of the Stack Pointer (SP) register pair are moved to the low order eight bits of the Program Counter (PC). The SP is now incremented and the byte at the memory location specified by the new contents of the SP are moved to the high order eight bits of the PC. The SP is now incremented again. The next opcode following this instruction will be fetched from the memory location specified by the PC. This instruction is normally used to return to the main line program at the completion of a routine entered by a CALL instruction. If condition cc is false, the PC is simply incremented as usual, and the program continues with the next sequential instruction. Condition cc is programmed as one of eight status which correspond to condition bits in the Flag Register (register F). These eight status are defined in the table below, which also specifies the corresponding cc bit fields in the assembled object code.

cc	Condition	Relevant Flag
000	NZ non zero	Z
001	Z zero	Z
010	NC non carry	C
011	C carry	C
100	PO parity odd	P/V
101	PE parity even	P/V
110	P sign positive	S
111	M sign negative	S

If cc is true:

M Cycles	T States	4 MHz E.T.
3	11 (5, 3, 3)	2.75

If cc is false:

M Cycles	T States	4 MHz E.T.
1	5	1.25

Condition Bits Affected:

None.

Example: If the S flag in the F register is set, the contents of the Program Counter are 3535H, the contents of the Stack Pointer are 2000H, the contents of memory location 2000H are B5H, and the contents of memory location 2001H are 18H, then after the execution of

```
RET M
```

the contents of the Stack Pointer Will be 2002H and the contents of the Program Counter will be 18B5H, pointing to the address of the next program opcode to be fetched.

RETI

Operation: Return from Interrupt

Opcode: RETI

1	1	1	0	1	1	0	1	ED
0	1	0	0	1	1	0	1	4D

Description: This instruction is used at the end of a maskable interrupt service routine to:

1. Restore the contents of the Program Counter (PC) (analogous to the RET instruction)

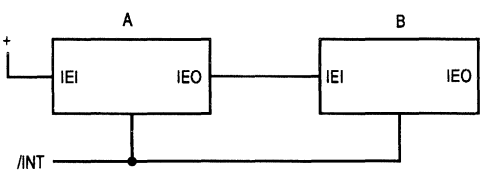
2. To signal an I/O device that the interrupt routine has been completed. The RETI instruction also facilitates the nesting of interrupts allowing higher priority devices to temporarily suspend service of lower priority service routines. Note: This instruction does not enable interrupts which were disabled when the interrupt routine was entered. Before doing the RETI instruction, the enable interrupt instruction (EI) should be executed to allow recognition of interrupts after completion of the current service routine.

M Cycles	T States	4 MHz E.T.
4	14 (4, 4, 3, 3)	3.50

Condition Bits Affected:

None.

Example: Given: Two interrupting devices, A and B connected in a daisy chain configuration with A having a higher priority than B.

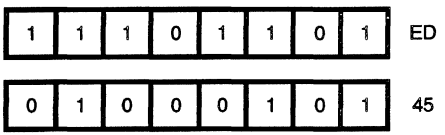


B generates an interrupt and is acknowledged. (The interrupt enable out, IE0, of B goes low, blocking any lower priority devices from interrupting while B is being serviced). Then A generates an interrupt, suspending service of B. (The IE0 of A goes 'low' indicating that a higher priority device is being serviced). The A routine is completed and a RETI is issued resetting the IE0 of A, allowing the B routine to continue. A second RETI is issued on completion of the B routine and the IE0 of B is reset (high) allowing lower priority devices interrupt access.

RETN

Operation: Return from non maskable interrupt

Opcode: RETN



Description: This instruction is used at the end of a non-maskable interrupts service routine to restore the contents of the Program Counter (PC) (analogous to the RET instruction). The state of IFF2 is copied back into IFF1 so that maskable interrupts are enabled immediately following the RETN if they were enabled before the non-maskable interrupt.

M Cycles	T States	4 MHz E.T.
4	14 (4, 4, 3, 3)	3.50

Condition Bits Affected:

None.

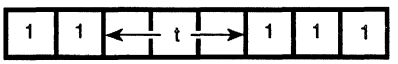
Example: If the contents of the Stack Pointer are 1000H and the contents of the Program Counter are 1A45H when a non maskable interrupt (NMI) signal is received, the CPU will ignore the next instruction and will instead restart to memory address 0066H . That is, the current Program Counter contents of 1A45H will be pushed onto the external stack address of 0FFFH and 0FFEH, high order-byte first, and 0066H will be loaded onto the Program Counter. That address begins an interrupt service routine which ends with RETN instruction. Upon the execution of RETN, the former Program Counter contents are popped off the external memory stack, low-order first, resulting in a Stack Pointer contents again of 1000H. The program flow continues where it left off with an opcode fetch to address 1A45H order-byte first, and 0066H will be loaded onto the Program Counter. That address begins an interrupt service routine which ends with RETN instruction. Upon the execution of RETN, the former Program Counter contents are popped off the external memory stack, low-order first, resulting in a Stack Pointer contents again of 1000H. The program flow continues where it left off with an opcode fetch to address 1A45H.

RST p

Operation: (SP–1) ← PCH, (SP–2) ← PCL, PCH ← 0, PCL ← P

Opcode: RST

Operands: p



Description: The current Program Counter (PC) contents are pushed onto the external memory stack, and the page zero memory location given by operand p is loaded into the PC. Program execution then begins with the opcode in the address now pointed to by PC. The push is performed by first decrementing the contents of the Stack Pointer (SP), loading the high-order byte of PC into the memory address now pointed to by SP, decrementing SP again, and loading the low-order byte of PC into the address now pointed to by SP. The ReSTart instruction allows for a jump to one of eight addresses as shown in the table below. The operand p is assembled into the object code using the corresponding T state. Note: Since all addresses are in page zero of memory, the high order byte of PC is loaded with 00H. The number selected from the "p" column of the table is loaded into the low-order byte of PC.

p	t
00H	000
08H	001
10H	010
18H	011
20H	100
28H	101
30H	110
38H	111

M Cycles	T States	4 MHz E.T.
3	11 (5, 3, 3)	2.75

Example: If the contents of the Program Counter are 15B3H, after the execution of

RST 18H (Object code 1101111)

the PC will contain 0018H, as the address of the next opcode to be fetched.

A

Z80®
INSTRUCTION DESCRIPTION

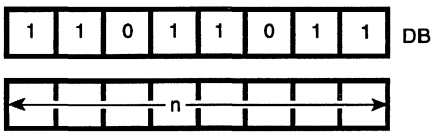
INPUT AND OUTPUT GROUP

IN A, (n)

Operation: A ← (n)

Opcode: IN

Operands: A, (n)



Description: The operand n is placed on the bottom half (A0 through A7) of the address bus to select the I/O device at one of 256 possible ports. The contents of the Accumulator also appear on the top half (A8 through A15) of the address bus at this time. Then one byte from the selected port is placed on the data bus and written into the Accumulator (register A) in the CPU.

M Cycles	T States	4 MHz E.T.
3	11 (4, 3, 4)	2.75

Condition Bits Affected:
None.

Example: If the contents of the Accumulator are 23H and the byte 7BH is available at the peripheral device mapped to I/O port address 01H, then after the execution of

```
IN A, (01H)
```

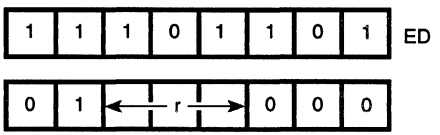
the Accumulator will contain 7BH.

IN r (C)

Operation: r ← (C)

Opcode: IN

Operands: r, (C)



Description: The contents of register C are placed on the bottom half (A0 through A7) of the address bus to select the I/O device at one of 256 possible ports. The contents of Register B are placed on the top half (A8 through A15) of the address bus at this time. Then one byte from the selected port is placed on the data bus and written into register r in the CPU. Register r identifies any of the CPU registers shown in the following table, which also shows the corresponding 3-bit "r" field for each. The flags will be affected, checking the input data.

Register	r
Flag	110 – Undefined opcode, set the flag
B	000
C	001
D	010
E	011
H	100
L	101
A	111

M Cycles	T States	4 MHz E.T.
3	12 (4, 4, 4)	3.00

Condition Bits Affected:

S: Set if input data is negative; reset otherwise
Z: Set if input data is zero; reset otherwise
H: Reset
P/V: Set if parity is even; reset otherwise
N: Reset
C: Not affected

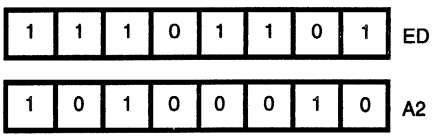
Example: If the contents of register C are 07H, the contents of register B are 10H, and the byte 7BH is available at the peripheral device mapped to I/O port address 07H, then after the execution of

IN D, (C)

INI

Operation: (HL) ← (C), B ← B – 1, HL ← HL + 1

Opcode: INI



Description: The contents of register C are placed on the bottom half (A0 through A7) of the address bus to select the I/O device at one of 256 possible ports. Register B may be used as a byte counter, and its contents are placed on the top half (A8 through A15) of the address bus at this time. Then one byte from the selected port is placed on the data bus and written to the CPU. The contents of the HL register pair are then placed on the address bus and the input byte is written into the corresponding location of memory. Finally the byte counter is decremented and register pair HL is incremented.

M Cycles	T States	4 MHz E.T.
4	16 (4, 5, 3, 4)	4.00

Condition Bits Affected:

S: Unknown
Z: Set if B – 1 = 0
reset otherwise
H: Unknown
P/V: Unknown
N: Set
C: Not affected

Example: If the contents of register C are 07H, the contents of register B are 10H, the contents of the HL register pair are 1000H, and the byte 7BH is available at the peripheral device mapped to I/O port address 07H, then after the execution of

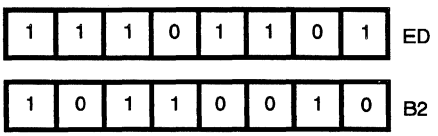
```
INI
```

memory location 1000H will contain 7BH, the HL register pair will contain 1001H, and register B will contain 0FH.

INIR

Operation: (HL) ← (C), B ← B – 1, HL ← HL +1

Opcode: INIR



Description: The contents of register C are placed on the bottom half (A0 through A7) of the address bus to select the I/O device at one of 256 possible ports. Register B is used as a byte counter, and its contents are placed on the top half (A8 through A15) of the address bus at this time. Then one byte from the selected port is placed on the data bus and written to the CPU. The contents of the HL register pair are placed on the address bus and the input byte is written into the corresponding location of memory. Then register pair HL is incremented, the byte counter is decremented. If decrementing causes B to go to zero, the instruction is terminated. If B is not zero, the PC is decremented by two and the instruction repeated. Interrupts will be recognized and two refresh cycles will be executed after each data transfer. Note that if B is set to zero prior to instruction execution, 256 bytes of data will be input.

If B ≠ 0:

M Cycles	T States	4 MHz E.T.
5	21 (4, 5, 3, 4, 5)	5.25

If B = 0:

M Cycles	T States	4 MHz E.T.
4	16 (4, 5, 3, 4)	4.00

Condition Bits Affected:

S: Unknown
Z: Set
H: Unknown
P/V: Unknown
N: Set
C: Not affected

Example: If the contents of register C are 07H, the contents of register B are 03H, the contents of the HL register pair are 1000H, and the following sequence of bytes are available at the peripheral device mapped to I/O port of address 07H:

51H
A9H
03H

then after the execution of

INIR

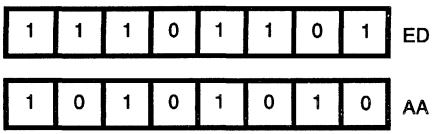
the HL register pair will contain 1003H, register B will contain zero, and memory locations will have contents as follows:

Location	Contents
1000H	51H
1001H	A9H
1002H	03H

IND

Operation: (HL) ← (C), B ← B – 1, HL ← HL – 1

Opcode: IND



Description: The contents of register C are placed on the bottom half (A0 through A7) of the address bus to select the I/O device at one of 256 possible ports. Register B may be used as a byte counter, and its contents are placed on the top half (A8 through A15) of the address bus at this time. Then one byte from the selected port is placed on the data bus and written to the CPU. The contents of the HL register pair are placed on the address bus and the input byte is written into the corresponding location of memory. Finally the byte counter and register pair HL are decremented.

M Cycles	T States	4 MHz E.T.
4	16 (4, 5, 3, 4)	4.00

Condition Bits Affected:

S: Unknown
Z: Set if B – 1= 0; reset otherwise
H: Unknown
P/V: Unknown
N: Set
C: Not affected

Example: If the contents of register C are 07H, the contents of register B are 10H, the contents of the HL register pair are 1000H, and the byte 7BH is available at the peripheral device mapped to I/O port address 07H, then after the execution of

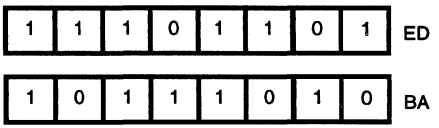
```
IND
```

memory location 1000H will contain 7BH, the HL register pair will contain 0FFFH, and register B will contain 0FH.

INDR

Operation: (HL) ← (C), B ← B1, HL ← HL1

Opcode: INDR



Description: The contents of register C are placed on the bottom half (A0 through A7) of the address bus to select the I/O device at one of 256 possible ports. Register B is used as a byte counter, and its contents are placed on the top half (A8 through A15) of the address bus at this time. Then one byte from the selected port is placed on the data bus and written to the CPU. The contents of the HL register pair are placed on the address bus and the input byte is written into the corresponding location of memory. Then HL and the byte counter are decremented. If decrementing causes B to go to zero, the instruction is terminated. If B is not zero, the PC is decremented by two and the instruction repeated. Interrupts will be recognized and two refresh cycles will be executed after each data transfer. Note that if B is set to zero prior to instruction execution, 256 bytes of data will be input.

If B ≠ 0

M Cycles	T States	4 MHz E.T.
5	21 (4, 5, 3, 4, 5)	5.25

If B = 0:

M Cycles	T States	4 MHz E.T.
4	16 (4, 5, 3, 4)	4.00

Condition Bits Affected:

S: Unknown
Z: Set
H: Unknown
P/V: Unknown
N: Set
C: Not affected

Example: If the contents of register C are 07H, the contents of register B are 03H, the contents of the HL register pair are 1000H, and the following sequence of bytes are available at the peripheral device mapped to I/O port address 07H:

51H
A9H
03H

then after the execution of

INDR

the HL register pair will contain 0FFDH, register B will contain zero, and memory locations will have contents as follows:

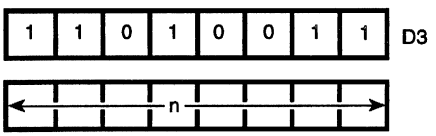
Location	Contents
0FFEH	03H
0FFFH	A9H
1000H	51H

OUT (n), A

Operation: (n) ← A

Opcode: OUT

Operands: (n), A



Description: The operand n is placed on the bottom half (A0 through A7) of the address bus to select the I/O device at one of 256 possible ports. The contents of the Accumulator (register A) also appear on the top half (A8 through A15) of the address bus at this time. Then the byte contained in the Accumulator is placed on the data bus and written into the selected peripheral device.

M Cycles	T States	4 MHz E.T.
3	11 (4, 3, 4)	2.75

Condition Bits Affected:
None.

Example: If the contents of the Accumulator are 23H the execution of then after the execution of

```
OUT (01H), A
```

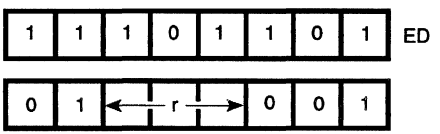
the byte 23H will have been written to the peripheral device mapped to I/O port address 01H.

OUT (C), r

Operation: (C) ← r

Opcode: OUT

Operands: (C), r



Description: The contents of register C are placed on the bottom half (A0 through A7) of the address bus to select the I/O device at one of 256 possible ports. The contents of Register B are placed on the top half (A8 through A15) of the address bus at this time. Then the byte contained in register r is placed on the data bus and written into the selected peripheral device. Register r identifies any of the CPU registers shown in the following table, which also shows the corresponding three-bit "r" field for each which appears in the assembled object code:

Register	r
B	000
C	001
D	010
E	011
H	100
L	101
A	111

M Cycles	T States	4 MHz E.T.
3	12 (4, 4, 4)	3.00

Condition Bits Affected:

None.

Example: If the contents of register C are 01H and the contents of register D are 5AH, after the execution of

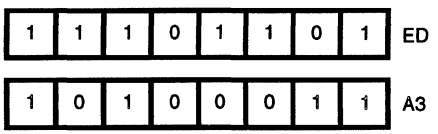
```
OUT (C),D
```

the byte 5AH will have been written to the peripheral device mapped to I/O port address 01H.

OUTI

Operation: (C) ← (HL), B ← B – 1, HL ← HL + 1

Opcode: OUTI



Description: The contents of the HL register pair are placed on the address bus to select a location in memory. The byte contained in this memory location is temporarily stored in the CPU. Then, after the byte counter (B) is decremented, the contents of register C are placed on the bottom half (A0 through A7) of the address bus to select the I/O device at one of 256 possible ports. Register B may be used as a byte counter, and its decremented value is placed on the top half (A8 through A15) of the address bus. The byte to be output is placed on the data bus and written into selected peripheral device. Finally the register pair HL is incremented.

M Cycles	T States	4 MHz E.T.
4	16 (4, 5, 3, 4)	4.00

Condition Bits Affected:

S: Unknown
Z: Set if B – 1 = 0; reset otherwise
H: Unknown
P/V: Unknown
N: Set
C: Not affected

Example: If the contents of register C are 07H, the contents of register B are 10H, the contents of the HL register pair are 1000H, and the contents of memory address 1000H are 59H, then after thee execution of

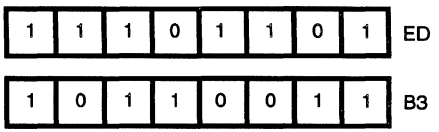
```
OUTI
```

register B will contain 0FH, the HL register pair will contain 1001H, and the byte 59H will have been written to the peripheral device mapped to I/O port address 07H.

OTIR

Operation: (C) ← (HL), B ← B – 1, HL ← HL + 1

Opcode: OTIR



Description: The contents of the HL register pair are placed on the address bus to select a location in memory. The byte contained in this memory location is temporarily stored in the CPU. Then, after the byte counter (B) is decremented, the contents of register C are placed on the bottom half (A0 through A7) of the address bus to select the I/O device at one of 256 possible ports. Register B may be used as a byte counter, and its decremented value is placed on the top half (A8 through A15) of the address bus at this time. Next the byte to be output is placed on the data bus and written into the selected peripheral device. Then register pair HL is incremented. If the decremented B register is not zero, the Program Counter (PC) is decremented by two and the instruction is repeated. If B has gone to zero, the instruction is terminated. Interrupts will be recognized and two refresh cycles will be executed after each data transfer. Note that if B is set to zero prior to instruction execution, the instruction will output 256 bytes of data.

If B ≠ 0:

M Cycles	T States	4 MHz E.T.
5	21 (4, 5, 3, 4, 5)	5.25

If B = 0:

M Cycles	T States	4 MHz E.T.
4	16 (4, 5, 3, 4)	4.00

Condition Bits Affected:

S: Unknown
Z: Set
H: Unknown
P/V: Unknown
N: Set
C: Not affected

Example: If the contents of register C are 07H, the contents of register B are 03H, the contents of the HL register pair are 1000H, and memory locations have the following contents:

Location	Contents
1000H	51H
1001H	A9H
1002H	03H

then after the execution of

OTIR

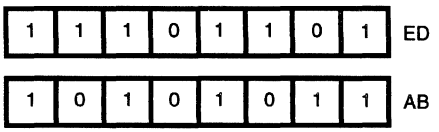
the HL register pair will contain 1003H, register B will contain zero, and a group of bytes will have been written to the peripheral device mapped to I/O port address 07H in the following sequence:

51H
A9H
03H

OUTD

Operation: (C) ← (HL), B ← B – 1, HL ← HL – 1

Opcode: OUTD



Description: The contents of the HL register pair are placed on the address bus to select a location in memory. The byte contained in this memory location is temporarily stored in the CPU. Then, after the byte counter (B) is decremented, the contents of register C are placed on the bottom half (A0 through A7) of the address bus to select the I/O device at one of 256 possible ports. Register B may be used as a byte counter, and its decremented value is placed on the top half (A8 through A15) of the address bus at this time. Next the byte to be output is placed on the data bus and written into the selected peripheral device. Finally the register pair HL is decremented.

M Cycles	T States	4 MHz E.T.
4	16 (4, 5, 3, 4)	4.00

Condition Bits Affected:

S: Unknown
Z: Set if B – 1 = 0; reset otherwise
H: Unknown
P/V: Unknown
N: Set
C: Not affected

Example: If the contents of register C are 07H, the contents of register B are 10H, the contents of the HL register pair are 1000H, and the contents of memory location 1000H are 59H, after the execution of

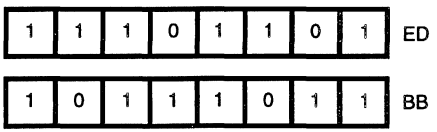
```
OUTD
```

register B will contain 0FH, the HL register pair will contain 0FFFH, and the byte 59H will have been written to the peripheral device mapped to I/O port address 07H.

OTDR

Operation: (C) ← (HL), B ← B – 1, HL ← HL – 1

Opcode: OTDR



Description: The contents of the HL register pair are placed on the address bus to select a location in memory. The byte contained in this memory location is temporarily stored in the CPU. Then, after the byte counter (B) is decremented, the contents of register C are placed on the bottom half (A0 through A7) of the address bus to select the I/O device at one of 256 possible ports. Register B may be used as a byte counter, and its decremented value is placed on the top half (A8 through A15) of the address bus at this time. Next the byte to be output is placed on the data bus and written into the selected peripheral device. Then register pair HL is decremented and if the decremented B register is not zero, the Program Counter (PC) is decremented by two and the instruction is repeated. If B has gone to zero, the instruction is terminated. Interrupts will be recognized and two refresh cycles will be executed after each data transfer. Note that if B is set to zero prior to instruction execution, the instruction will output 256 bytes of data.

If B ≠ 0:

M Cycles	T States	4 MHz E.T.
5	21 (4, 5, 3, 4, 5)	5.25

If B = 0:

M Cycles	T States	4 MHz E.T.
4	16 (4, 5, 3, 4)	4.00

Condition Bits Affected:

S: Unknown
Z: Set
H: Unknown
P/V: Unknown
N: Set
C: Not affected

Example: If the contents of register C are 07H, the contents of register B are 03H, the contents of the HL register pair are 1000H, and memory locations have the following contents:

Location	Contents
0FFEH	51H
0FFFH	A9H
1000H	03H

then after the execution of

OTDR

the HL register pair will contain 0FFDH, register B will contain zero, and a group of bytes will have been written to the peripheral device mapped to I/O port address 07H in the following sequence:

03H
A9H
51H

CHAPTER 6
INTERRUPT RESPONSE

A

6.0 INTRODUCTION

The purpose of an interrupt is to allow peripheral devices to suspend CPU operation in an orderly manner and force the CPU to start a peripheral service routine. Usually, this service routine is involved with the exchange of data, or status and control information, between the CPU and the peripheral. Once the service routine is completed, the CPU returns to the operation from which it was interrupted.

6.1 INTERRUPT ENABLE/DISABLE

The Z80 CPU has two interrupt inputs, a software maskable interrupt (/INT) and a non-maskable interrupt (/NMI). The non-maskable interrupt can not be disabled by the programmer and will be accepted whenever a peripheral device requests it. This interrupt is generally reserved for very important functions that can be enabled or disabled selectively by the programmer. This allows the programmer to disable the interrupt during periods where his program has timing constraints that do not allow interrupt. In the Z80 CPU, there is an interrupt enable flip-flop (IFF) that is set or reset by the programmer using the Enable Interrupt (EI) and Disable Interrupt (DI) instructions. When the IFF is reset, an interrupt can not be accepted by the CPU.

There are two enable flip-flops, IFF1 and IFF2.

IFF1	IFF2
Actually disables interrupts from being accepted.	Temporary storage location for IFF1.

The state of IFF1 is used to inhibit interrupts while IFF2 is used as a temporary storage location for IFF1.

A reset to the CPU forces both the IFF1 and IFF2 to the reset state so that interrupts are disabled. They can then be enabled at any time by an EI instruction by the programmer. When an EI instruction is executed, any pending interrupt request is not accepted until after the instruction following EI has been executed. This single instruction delay is necessary when the following instruction is a return instruction and interrupts must not be allowed until the return has been completed. The EI instruction sets both IFF1 and IFF2 to the enable state. When a maskable interrupt is accepted by the CPU, both IFF1 and IFF2 are automatically reset, inhibiting further interrupts until the programmer wishes to issue a new EI instruction. Note that for all of the previous cases, IFF1 and IFF2 are always equal.

The purpose of IFF2 is to save the status of IFF1 when a non-maskable interrupt occurs. When a non-maskable interrupt is accepted, IFF1 is reset to prevent further interrupts until reenabled by the programmer. Thus, after a non-maskable interrupt has been accepted, maskable interrupts are disabled but the previous state of IFF1 has been saved so that the complete state of the CPU just prior to the non-maskable interrupt can be restored at any time. When a Load Register A with Register I (LD A, I) instruction or a Load Register A with Register R (LD A, R) instruction is executed, the state of IFF2 is copied into the parity flag where it can be tested or stored.

A second method of restoring the status of IFF1 is through the execution of a Return From Non-Maskable Interrupt (RETN) instruction. Since this instruction indicates that the non-maskable interrupt service routine is complete, the contents of IFF2 are now copied back into IFF1, so that the status of IFF1 just prior to the acceptance of the non-maskable interrupt will be restored automatically.

Table 6-1 is a summary of the effect of different instructions on the two enable flip-flops.

Table 6-1. Interrupt Enable/Disable Flip-Flops

Action	IFF1	IFF2	Comments
CPU Reset	0	0	Maskable Interrupt. /INT Disabled
DI Instruction Execution	0	0	Maskable /INT Disabled
EI Instruction Execution	1	1	Maskable, /INT Enabled
LD A,I Instruction Execution	•	•	IFF2 → Parity Fag
LD A,R instruction Execution	•	•	IFF2 → Parity Flag
Accept /NMI	0	•	Maskable Interrupt
RETN instruction Execution	IFF2	•	IFF2 → IFF1 at completion of an indicates no change routine.

6.2 CPU RESPONSE

6.2.1 Non-Maskable

A non-maskable interrupt is accepted at all times by the CPU. When this occurs, the CPU ignores the next instruction that it fetches and instead does a restart to location 0066H. Thus, it behaves exactly as if it had recycled a restart instruction but, it is to a location that is not one of the eight software restart locations. A restart is merely a call to a specific address in page 0 of memory.

The CPU can be programmed to respond to the maskable interrupt in any one of three possible modes.

6.2.2 Mode 0

This mode is similar to the 8080A interrupt response mode. With this mode, the interrupting device can place any instruction on the data bus and the CPU executes it. Thus, the interrupting device provides the next instruction to be executed. Often this will be a restart instruction since the interrupting device only need supply a single byte instruction. Alternatively, any other instruction such as a 3-byte call to any location in memory could be executed.

The number of clock cycles necessary to execute this instruction is two more than the normal number for the instruction. This occurs since the CPU automatically adds two wait states to an interrupt response cycle to allow sufficient time to implement an external daisy chain for priority control. Figures 3-6 and 3-7 illustrate the detailed timing for an interrupt response. After the application of /RESET, the CPU will automatically enter interrupt Mode 0.

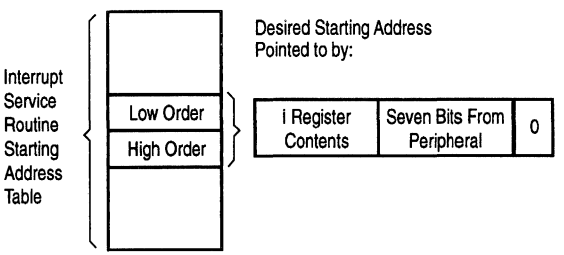
6.2.3 Mode 1

When this mode has been selected by the programmer, the CPU responds to an interrupt by executing a restart to location 0038H. Thus, the response is identical to that for a non-maskable interrupt except that the call location is 0038H instead of 0066H. The number of cycles required to complete the restart instruction is two more than normal due to the two added wait states.

6.2.4 Mode 2

This mode is the most powerful interrupt response mode. With a single 8-bit byte from the user, an indirect call can be made to any memory location.

With this mode the programmer maintains a table of 16-bit starting addresses for every interrupt service routine. This table may be located anywhere in memory. When an interrupt is accepted, a 16-bit pointer must be formed to obtain the desired interrupt service routine starting address from the table. The upper eight bits of this pointer is formed from the contents of the I register. The I register must have been previously loaded with the desired value by the programmer, i.e., LD I, A. Note that a CPU reset clears the I register so that it is initialized to zero. The lower eight bits of the pointer must be supplied by the interrupting device. Actually, only seven bits are required from the interrupting device as the least significant, bit must be a zero. This is required since the pointer is used to get two adjacent bytes to form a complete 16-bit service routine starting address and the addresses must always start in even locations.



The first byte in the table is the least significant (low order) portion of the address. The programmer must obviously fill this table in with the desired addresses before any interrupts are to be accepted.

Note that this table can be changed at any time by the programmer (if it is stored in Read/Write Memory) to allow different peripherals to be serviced by different service routines.

Once the interrupting devices supplies the lower portion of the pointer, the CPU automatically pushes the program counter onto the stack, obtains the starting address from the table, and does a jump to this address. This mode of response requires 19 clock periods to complete (seven to fetch the lower eight bits from the interrupting device, six to save the program counter, and six to obtain the jump address).

Note that the Z80 peripheral devices include a daisy chain priority interrupt structure that automatically supplies the programmed vector to the CPU during interrupt acknowledge. Refer to the Z80 PIO, Z80 SIO, and Z80 CTC manuals for details.

A

CHAPTER 7
HARDWARE IMPLEMENTATION EXAMPLES

7.0 INTRODUCTION: MINIMUM SYSTEM

This chapter is intended to serve as a basic introduction to implementing systems with the Z80 CPU. Figure 7-1 is a diagram of a very simple Z80 system. Any Z80 system must include the following elements:

- 5V Power Supply
- Oscillator
- Memory Devices
- I/O Circuits
- CPU

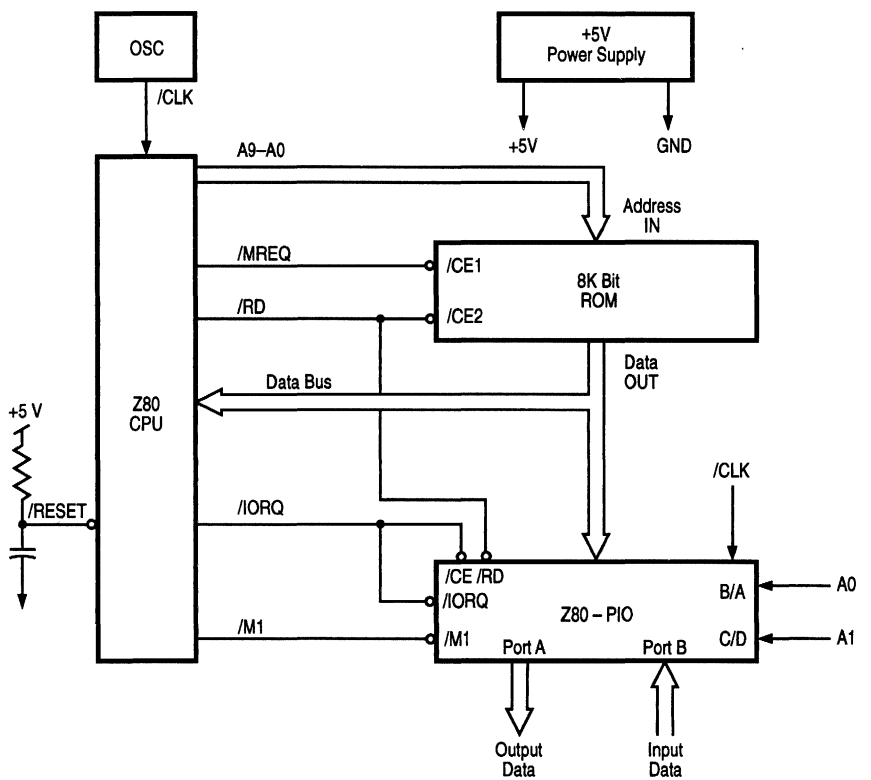


Figure 7-1. Minimum Z80 Computer System

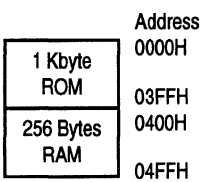
Since the Z80 CPU requires only a single 5V supply, most small systems can be implemented using only this single supply.

The external memory can be any mixture of standard RAM, ROM, or PROM. In this simple example, we have shown a single 8K bit ROM (1 Kbytes) being utilized as the entire memory system. For this example we have assumed that the Z80 internal register configuration contains sufficient Read/Write storage so that external RAM memory is not required.

Every computer system requires I/O circuits to allow it to interface to the real world. In this simple example, it is assumed that the output is an 8-bit control vector and the input is an 8-bit status word. The input data could be gated onto the data bus using any standard tri-state driver while the output data could be latched with any type of standard TTL latch. For this example we have used a Z80 PIO for the I/O circuit. This single circuit attaches to the data bus as shown and provides the required 16 bits of TTL compatible I/O. (Refer to the Z80 PIO manual for details on the operation of this circuit.) Notice in this example that with only three LSI circuits, a simple oscillator and a singe 5V power supply, a powerful computer has been implemented.

7.1 ADDING RAM

Most computer systems require some amount of external Read/Write memory for data storage and to implement a stack. Figure 7-2 illustrates how 256 bytes of static memory can be added to the previous example. In this example, the memory space is assumed to be organized as follows:



In this diagram the address space is described in hexadecimal notation. For this example address bit A10 separates the ROM space from the RAM space so that it can be used for the chip select function. For larger amounts of external ROM or RAM, a simple TTL decoder will be required to form the chip selects.

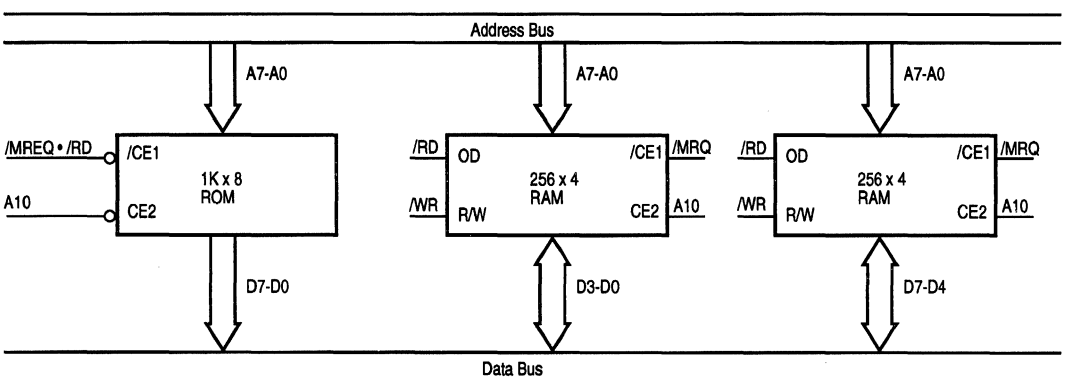


Figure 7-2 ROM and RAM Implementation

7.2 MEMORY SPEED CONTROL

For many applications, it may be desirable to use slow memories to reduce costs. The /WAIT line on the CPU allows the Z80 to operate with any speed memory. By referring back to Chapter A3, you notice that the memory access time requirements are most severe during the /M1 cycle instruction fetch. All other memory accesses have an additional one half clock cycle to be completed. For this reason it may be desirable in some applications to add one wait state to the /M1 cycle so that slower memories can be used. Figure 7-3 is an example of a simple circuit that will accomplish this task. This circuit can be changed to add a single wait state to any memory access as shown in Figure 7-4.

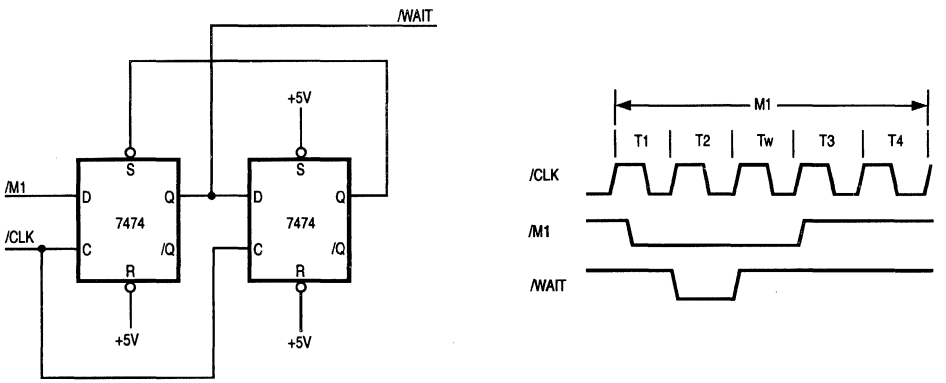


Figure 7-3. Adding One Wait State to an M1 Cycle

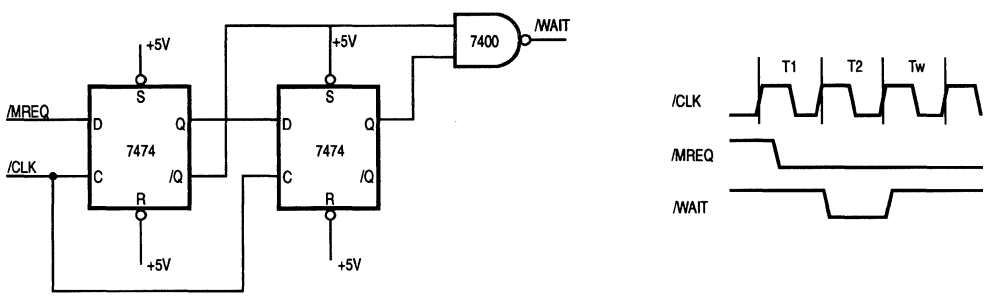


Figure 7-4. Adding One Wait State to Any Memory Cycle

7.3 INTERFACING DYNAMIC MEMORIES

This section is intended to serve as a brief introduction to interfacing dynamic memories. Each individual dynamic RAM has varying specifications that require minor modifications to the description given here and no attempt will be made in this document to give details for any particular RAM. Separate application notes are available showing how the Z80 CPU can be interfaced to most popular dynamic RAM.

Figure 7-5 illustrates the logic necessary to interface 8 Kbytes of dynamic RAM using 18-pin 4K dynamic memories. This figure assumes that the RAMs are the only memory in the system so that A12 is used to select between the two pages of memory. During refresh time, all memories in the system must be read. The CPU provides the proper refresh address on lines A0 through A6. To add additional memory to the system it is necessary to replace only the two gates that operate on A12 with a decoder that operates on all required address bits. For larger systems, buffering for the address and data bus is also generally required.

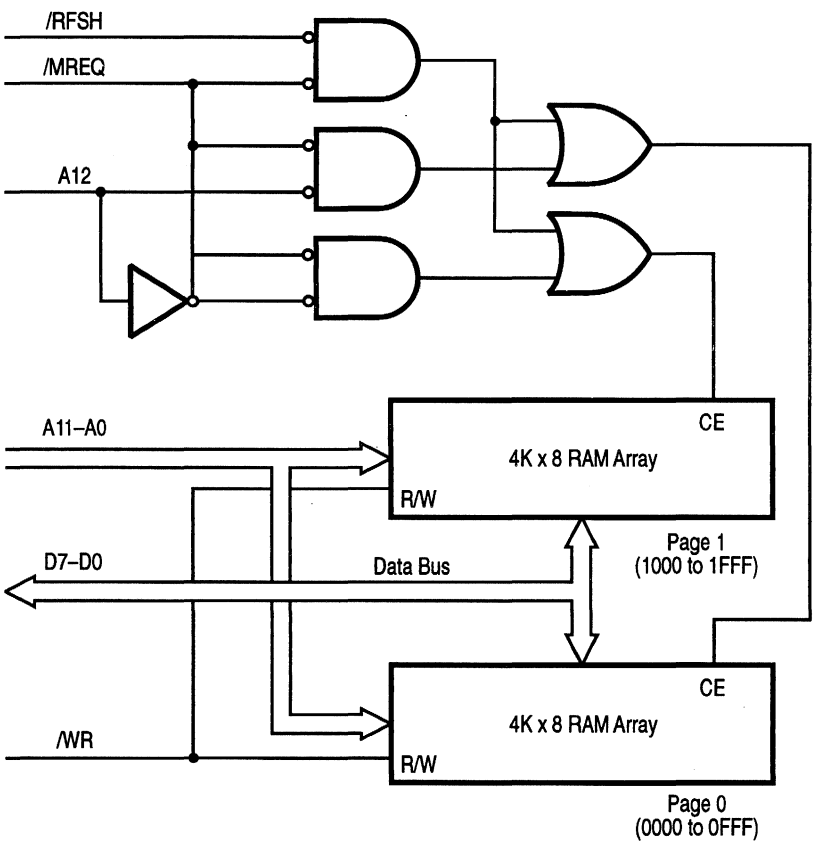


Figure 7-5. Interfacing Dynamic RAMs

CHAPTER 8
SOFTWARE IMPLEMENTATION EXAMPLES

8.0 INTRODUCTION: SOFTWARE FEATURES

The Z80 instruction set provides the user with a large and flexible repertoire of operations with which to formulate control of the Z80 CPU.

The main alternate and index registers can be used to hold the arguments of arithmetic and logical operations, or to form memory addresses, or as fast-access storage for frequently used data.

Information can be moved directly from register to register, from memory to memory, from memory to registers, or from registers to memory. In addition, register contents and register/memory contents can be exchanged without using temporary storage. In particular, the contents of main and alternate registers can be completely exchanged by executing only two instructions, EX and EXX. This register exchange procedure can be used to separate the set of working registers between different logical procedures or to expand the set of available registers in a single procedure.

Storage and retrieval of data between pairs of registers and memory can be controlled on a last-in first-out basis through PUSH and POP instructions which utilize a special stack pointer register, SP. This stack register is available both to manipulate data and to automatically store and retrieve addresses for subroutine linkage. When a subroutine is called, for example, the address following the CALL instruction is placed on the top of the push-down stack pointed to by SP. When a subroutine returns to the calling routine, the address on the top of the stack is used to set the program counter for the address of the next instruction. The stack pointer is adjusted automatically to reflect the current "top" stack position during PUSH, POP, CALL, and RET instructions. This stack mechanism allows pushdown data stacks and subroutine calls to be nested to any practical depth because the stack area can potentially be as large as memory space.

The sequence of instruction execution can be controlled by six different flags (carry, zero, sign, parity/overflow, add/subtract, half-carry) which reflect the results of arithmetic, logical, shift, and compare instructions. After the execution of an instruction which sets a flag, that flag can be used to control a conditional jump or return instruction. These instructions provide logical control following the manipulation of single bit, 8-bit byte (or) 16-bit data quantities.

A full set of logical operations, including AND, OR, XOR (exclusive-OR), CPL (NOR), and NEG (two's complement) are available for Boolean operations between the accumulator and all other 8-bit registers, memory locations, or immediate operands.

In addition, a full set of arithmetic and logical shifts in both directions are available which operate on the contents of all 8-bit primary registers or directly on any memory location. The carry flag can be included or simply set by these shift instructions to provide both the testing of shift results and to link register/register or register/memory shift operations.

8.1 EXAMPLES OF USE OF SPECIAL Z80 INSTRUCTIONS

A. Assume that a string of data in memory starting at location "DATA" is to be moved into another area of memory starting at location "BUFFER" and that the string length is 737 bytes. This operation can be accomplished as follows:

```
LD      HL , DATA       ;START ADDRESS OF DATA STRING
LD      DE , BUFFER     ;START ADDRESS OF TARGET BUFFER
LD      BC , 737        ;LENGTH OF DATA STRING
LDIR                    ;MOVE STRING - TRANSFER MEMORY POINTED TO
                        ;BY HL INTO MEMORY LOCATION POINTED TO BY DE
                        ;INCREMENT HL AND DE, DECREMENT BC
                        ;PROCESS UNTIL BC = 0.
```

Eleven bytes are required for this operation and each byte of data is moved in 21 clock cycles.

B. Assume that a string in memory starting at location "DATA" is to be moved into another area of memory starting at location "BUFFER" until an ASCII $ character (used as string delimiter) is found. Also assume that the maximum string length is 132 characters. The operation can be performed as follows:

```
        LD      HL , DATA       ;STARTING ADDRESS OF DATA STRING
        LD      DE , BUFFER     ;STARTING ADDRESS OF TARGET BUFFER
        LD      BC , 132        ;MAXIMUM STRING LENGTH
        LD      A , '$'         ;STRING DELIMITER CODE
LOOP:   CP      (HL)            ;COMPARE MEMORY CONTENTS WITH DELIMITER
        JR      Z , END - $     ;GO TO END IF CHARACTERS EQUAL
        LDI                     ;MOVE CHARACTER (HL) to (DE)
                                ;INCREMENT HL AND DE, DECREMENT BC
        JP      PE , LOOP       ;GO TO "LOOP" IF MORE CHARACTERS
END:                            ;OTHERWISE, FALL THROUGH
                                ;NOTE: P/V FLAG IS USED
                                ;TO INDICATE THAT REGISTER BC WAS
                                ; DECREMENTED TO ZERO.
```

Nineteen bytes are required for this operation.

C. Assume that a 16-digit decimal number represented in packed BCD format (two BCD digits/byte) has to be shifted as shown in the Figure 8-1 in order to mechanize BCD multiplication or division. The operation can be accomplished as follows:

```
        LD      HL  ,  DATA       ;ADDRESS OF FIRST BYTE
        LD      B   ,  COUNT      ;SHIFT COUNT
        XOR     A                 ;CLEAR ACCUMULATOR
ROTAT:  RLD                       ;ROTATE LEFT LOW ORDER DIGIT IN ACC
                                  ;WITH DIGITS IN (HL)
        INC     HL                ;ADVANCE MEMORY POINTER.
        DJNZ    ROTAT — $         ;DECREMENT B AND GO TO ROTAT IF
                                  ;B IS NOT ZERO, OTHERWISE FALL THROUGH
```

Eleven bytes are requited for this operation.

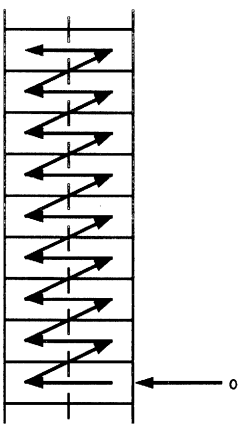


Figure 8-1. Shifting of BCD Digits/Bytes

D. Assume that one number is to be subtracted from another and that they are both in packed BCD format, that they are of equal but varying length, and that the result is to be stored in the location of the minuend. The operation can be accomplished as follows:

```
        LD      HL  ,  ARG1       ;ADDRESS OF MINUEND
        LD      DE  ,  ARG2       ;ADDRESS OF SUBTRAHEND
        LD      B   ,  LENGTH     ;LENGTH OF TWO ARGUMENTS
        AND     A                 ;CLEAR CARRY FLAG
SUBDEC: LD      A   ,  (DE)       ;SUBTRAHEND TO ACC
        SBC     A   ,  (HL)       ;SUBTRACT (HL) FROM ACC
        DAA                       ;ADJUST RESULT TO DECIMAL CODED VALUE
        LD      (HL),  A          ;STORE RESULT
        INC     HL                ;ADVANCE MEMORY POINTERS
        INC     DE
        DJNZ    SUBDEC — $        ;DECREMENT B AND GO TO "SUBDEC" IF B
                                  ;NOT ZERO, OTHERWISE FALL THROUGH
```

Seventeen bytes are required for this operation.

8.2 EXAMPLES OF PROGRAMMING TASKS

A. The following program sorts an array of numbers each in the range (0, 255) into ascending order using a standard exchange sorting algorithm.

```
01/22/76        11:14:37        BUBBLE LISTING                    PAGE 1

LOC   OBJ CODE    STMT    SOURCE STATEMENT

                    1     ;  STANDARD EXCHANGE (BUBBLE) SORT ROUTINE
                    2     ;
                    3     ;  AT ENTRY:   HL CONTAINS ADDRESS OF DATA
                    4     ;              C CONTAINS NUMBER OF ELEMENTS TO BE SORTED
                    5     ;              (1<C<256)
                    6     ;
                    7     ;  AT EXIT: DATA SORTED IN ASCENDING ORDER
                    8     ;
                    9     ;  USE OF REGISTERS
                   10     ;
                   11     ;  REGISTER     CONTENTS
                   12     ;
                   13     ;  A            TEMPORARY STORAGE FOR CALCULATIONS
                   14     ;  B            COUNTER FOR DATA ARRAY
                   15     ;  C            LENGTH OF DATA ARRAY
                   16     ;  D            FIRST ELEMENT IN COMPARISON
                   17     ;  E            SECOND ELEMENT IN COMPARISON
                   18     ;  H            FLAG TO INDICATE EXCHANGE
                   19     ;  L            UNUSED
                   20     ;  IX           POINTER INTO DATA ARRAY
                   21     ;  IY           UNUSED
                   22     ;
0000  222600       23     SORT:  LD    (DATA), HL    ; SAVE DATA ADDRESS
0003  CB84         24     LOOP:  RES   FLAG, H       ; INITIALIZE EXCHANGE FLAG
0005  41           25            LD    B, C          ; INITIALIZE LENGTH COUNTER
0006  05           26            DEC   B             ; ADJUST FOR TESTING
0007  DD2A2600     27            LD    IX, (DATA)    ; INITIALIZE ARRAY POINTER
000B  DD7E00       28     NEXT:  LD    A, (IX)       ; FIRST ELEMENT IN COMPARISON
000E  57           29            LD    D, A          ; TEMPORARY STORAGE FOR ELEMENT
000F  DD5E01       30            LD    E, (IX+1)     ; SECOND ELEMENT IN COMPARISON
0012  93           31            SUB   E             ; COMPARISON FIRST TO SECOND
0013  3008         32            JR    PC, NOEX-$    ; IF FIRST > SECOND, NO JUMP
0015  DD7300       33            LD    (IX), E       ; EXCHANGE ARRAY ELEMENTS
0018  DD7201       34            LD    (IX+I), D
001B  CBC4         35            SET   FLAG, H       ; RECORD EXCHANGE OCCURRED
0010  DD23         36     NOEX:  INC   IX            ; POINT TO NEXT DATA ELEMENT
001F  10EA         37            DJNZ  NEXT-$        ; COUNT NUMBER OF COMPARISONS
                   38                                ; REPEAT IF MORE DATA PAIRS
0021  CB44         39            BIT   FLAG, H       ; DETERMINE IF EXCHANGE OCCURRED
0023  20DE         40            JR    NZ, LOOP-$    ; CONTINUE IF DATA UNSORTED
0025  C9           41            RET                 ; OTHERWISE, EXIT
                   42     ;
0026               43     FLAG:  EQU   0             ; DESIGNATION OF FLAG BIT
0026               44     DATA:  DEFS  2             ; STORAGE FOR DATA ADDRESS
                   45            END
```

B. The following program multiplies two unsigned 16 bit integers and leaves the result in the HL register pair.

01/22/76 11:32:36 MULTIPLY LISTING PAGE 1

```
LOC  OBJ CODE  STMT    SOURCE STATEMENT
0000           1   MULT:;   UNSIGNED SIXTEEN BIT INTEGER MULTIPLY.
               2   ;        ON ENTRANCE: MULTIPLIER IN DE.
               3   ;        MULTIPLICAND IN HL.
               4   ;
               5   ;        ON EXIT: RESULT IN HL.
               6   ;
               7   ;        REGISTER USES:
               8   ;
               9   ;
               10  ;        H     HIGH ORDER PARTIAL RESULT
               11  ;        L     LOW ORDER PARTIAL RESULT
               12  ;        D     HIGH ORDER MULTIPLICAND
               13  ;        E     LOW ORDER MULTIPLICAND
               14  ;        B     COUNTER FOR NUMBER OF SHIFTS
               15  ;        C     HIGH ORDER BITS OF MULTIPLIER
               16  ;        A     LOW ORDER BITS OF MULTIPLIER
               17  ;
0000  0610     18           LD    B, 16;          NUMBER OF BITS-INITIALIZE
0002  4A       19           LD    C, D;           MOVE MULTIPLIER
0003  7B       20           LD    A, E;
0004  EB       21           EX    DE, HL;         MOVE MULTIPLICAND
0005  210000   22           LD    HL, 0;          CLEAR PARTIAL RESULT
0008  CB39     23   MLOOP:  SRL   C;              SHIFT MULTIPLIER RIGHT
000A  IF       24           RRA                   LEAST SIGNIFICANT BIT IS
               25  ;                             IN CARRY.
000B  3001     26           JR    NC, NOADD-$;    IF NO CARRY, SKIP THE ADD.
000D  19       27           ADD   HL, DE;         ELSE ADD MULTIPLICAND TO
               28  ;                             PARTIAL RESULT.
000E  EB       29   NOADD:  EX    DE, HL;         SHIFT MULTIPLICAND LEFT
000F  29       30           ADD   HL, HL;         BY MULTIPLYING IT BY TWO.
0010  EB       31           EX    DE, HL;
0011  10F5     32           DJNZ  MLOOP-$;        REPEAT UNTIL NO MORE BITS.
0013  C9       33           RET;
               34           END;
```

INDEX

Z80 CPU INSTRUCTION SET

A

ZILOG

TABLE OF CONTENTS

List of Figures

List or Tables

CHAPTER 1
INTRODUCTION

1.0 FEATURES

- Four Independently Programmable Counter/Timer Channels, Each with a Readable Down-Counter and a Selectable 16 or 256 Prescaler. Down-Counters are Reloaded Automatically at Zero Count
- Selectable Positive or Negative Trigger Initiates Timer Operation
- Three Channels Have Zero Count/Timeout Outputs Capable of Driving Darlington Transistors
- NMOS Version for High-Cost Performance Solutions
- CMOS Version for the Designs Requiring Low Power Consumption
- NMOS Z0843004 - 4 MHz, Z0843006 - 6.17 MHz
- CMOS Z84C3006 - DC to 6.17 MHz, Z84C3008 DC to 8 MHz, Z84C3010 - DC to 10 MHz
- Interfaces Directly to the Z80 CPU or — for Baud Rate Generation — to the Z80 SIO
- Standard Z80 Family Daisy-Chain Interrupt Structure Provides Fully Vectored, Prioritized Interrupts Without External Logic. The CTC May also be Used as an Interrupt Controller
- 6 MHz Version Supports 6.144 MHz CPU Clock Operation

1.1 GENERAL DESCRIPTION

The Z80 CTC, hereinafter referred to as Z80 CTC or CTC, four-channel counter/timer can be programmed by system software for a broad range of counting and timing applications. The four independently programmable channels of the Z80 CTC satisfy common microcomputer system requirements for event counting, interrupt and interval timing, and general clock rate generation.

System design is simplified because the CTC connects directly to both the Z80 CPU and the Z80 SIO with no additional logic. In larger systems, address decoders and buffers may be required.

Programming the CTC is straightforward: each channel is programmed with two bytes; a third is necessary when interrupts are enabled. Once started, the CTC counts down, automatically reloads its lime constant, and resumes counting. Software timing loops are completely eliminated. Interrupt processing is simplified because only one vector need be specified; the CTC internally generates a unique vector for each channel.

The Z80 CTC requires a single +5%V power supply and the standard Z80 single-phase system clock. It is packaged in 28-pin DIPs, a 44-pin plastic chip carrier, and a 44-pin Quad Flat Pack. Note that the QFP package is only available for CMOS versions. (Reference Chapter 3, CTC Pin Descriptions.)

CHAPTER 2
CTC ARCHITECTURE

2.0 OVERVIEW

The internal structure of the Z80 CTC consists of a Z80 CPU bus interface, Internal Control logic, four sets of Counter/Timer Channel logic, and interrupt control logic. The four independent, counter/timer channels are identified by sequential numbers from 0 to 3. The CTC has the capability of generating a unique interrupt vector. For each separate channel (for automatic vectoring to an interrupt service routine). The four channels can be connected in four contiguous slots in the standard Z80 priority chain with channel number 0 having the highest priority. The CPU bus interface logic allows the CTC device to interface directly to the CPU with no other external logic. However, port address decoders and/or line buffers may be required for large systems. A block diagram of the Z80 CTC is shown in Figure 2-1.

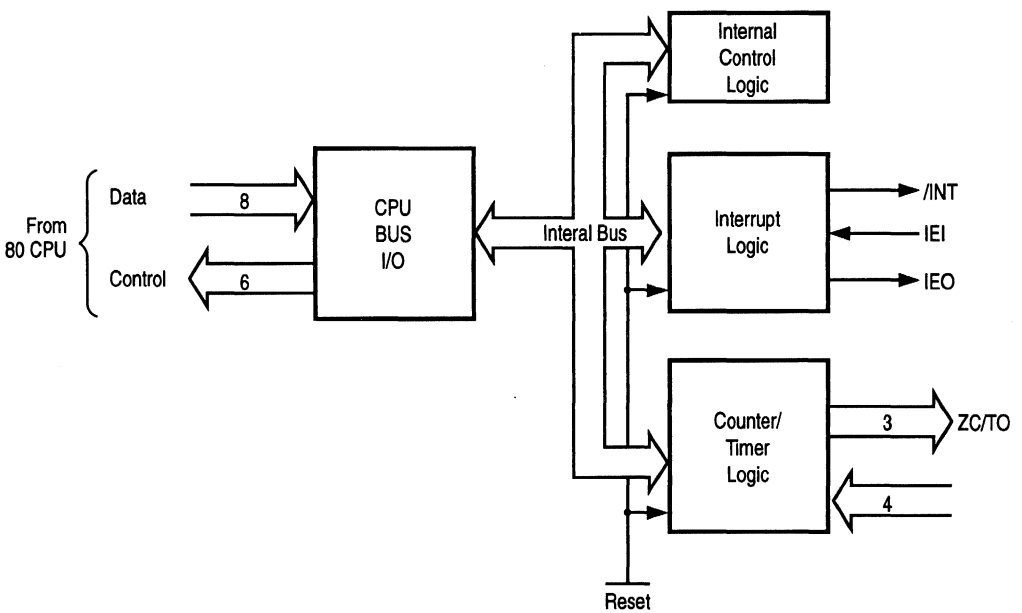


Figure 2-1. CTC Block Diagram

2.1 STRUCTURE OF CHANNEL LOGIC

The structure of one of the four sets of Counter/Timer channel logic is shown in Figure 2-2. This logic is composed of two registers, two counters and control logic. The registers consist of an 8-bit Time Constant register and an 8-bit Channel Control register. The counters consist of an 8-bit CPU-readable down-counter and an 8-bit prescaler.

2.1.1 In Channel Control Register and Logic

The Channel Control register (8-bit) and Logic is written to by the CPU to select the modes and parameters of the channel. Within the CTC device there are four such registers corresponding to the four Counter/Timer channels. The register to be written to is determined by the encoding of two channel select input pins: CS0 and CS1 (usually attached to A0 and A1 of the CPU address bus). This is illustrated in Table 2-1.

Table 2-1. Channel Select Truth Table

	CS0	CS1
Channel 0	0	0
Channel 1	0	1
Channel 2	1	0
Channel 3	1	1

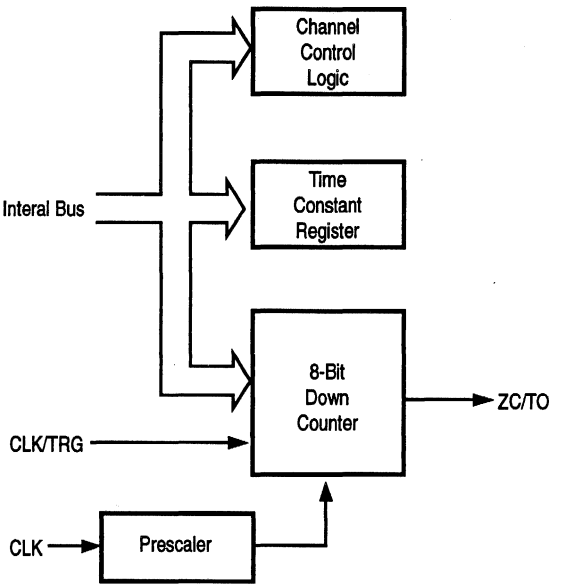


Figure 2-2. Channel Block Diagram

In the control word written to program each Channel Control register, bit 0 is always set; the other seven bits are programmed to select alternatives on the channel's operating modes and parameters. This is illustrated in Figure 2-3. (For a more complete discussion see Chapter 4, "CTC Operating Modes" and Chapter 5, "CTC Programming.")

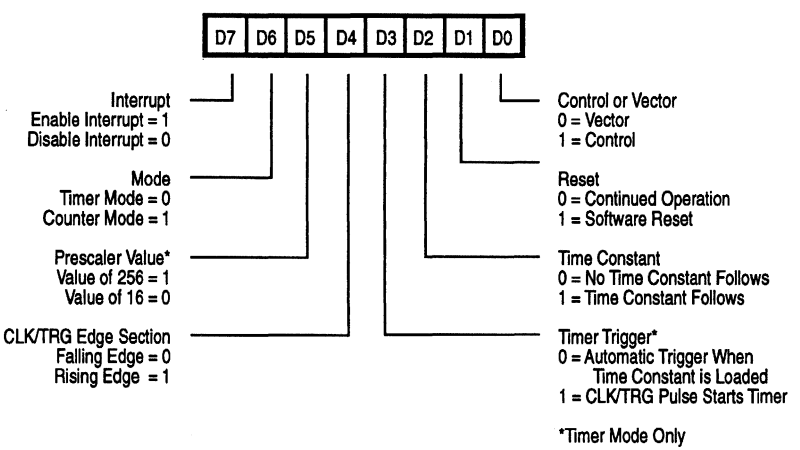


Figure 2-3. Channel Control Register

2.1.2 The Prescaler

The prescaler is an 8-bit device which is used in the timer mode only. The prescaler is programmed by the CPU through the Channel Control register to divide its input, the System clock (Φ), by 16 or 256. The output of the prescaler is then fed as an input to clock the down-counter. Initially, arid each time the down-counter clocks down to zero, the down-counter is reloaded automatically with the contents of the Time Constant register. In effect this divides the System clock by an additional factor of the time constant. Each time the down-counter counts down to zero, its output, Zero Count/Timeout (ZC/TO), is pulsed High.

2.1.3 The Time Constant Register

The 8-bit Time Constant register is used in both Counter and Timer modes. It is programmed by the CPU just after the channel control word with an integer time constant value of 1 through 256. This register loads the programmed value Into the down counter when the CTC is first initialized and reloads the same value into the down counter automatically whenever it counts down thereafter to zero. If a new time constant is loaded into the Time Constant register while a channel is counting or timing, the present down-count will be completed before the new time constant is loaded into the down counter. (For details of how a time constant is written to a CTC channel, see Chapter 5, "CTC Programming.")

2.1.4 The Down-Counter

The down-counter is an 8-bit register, which is used in both Counter and Timer modes. It is loaded by the Time Constant register both initially, and when it counts down to zero. In the Counter mode, the down-counter is decremented by each external clock edge. In the Timer mode, it is decremented by the clock output of the prescaler. By performing a simple I/O Read at the port address assigned to the selected CTC channel, the CPU can access the contents of the Downcounter and obtain the number of counts-to-zero. Any of the four CTC channels may be programmed to generate an interrupt request sequence each time the zero count is reached.

In Channels 0, 1, and 2, a signal pulse appears at the corresponding ZC/TO pin when the zero count condition is reached. Due to package pin limitations, however, Channel 3 does not have this pin and so may be used only in applications where this output pulse is not required.

B

2.2 INTERRUPT CONTROL LOGIC

The interrupt control logic insures that the CTC acts in accordance with Z80 system interrupt protocol for nested priority interrupting and return from interrupt. The priority of any system device is determined by its physical location it) a daisy chain configuration. Two signal lines (IEI and IEO) are provided in CTC devices to form this system daisy chain. The device closest to the CPU has the highest priority. Within the CTC, interrupt priority is predetermined by channel number, with Channel 0 having highest and Channel 3 the lowest priority (Figure 2-4). The purpose of a CTC-generated interrupt, as with any peripheral device, is to force the CPU to execute an interrupt service routine. According to Z80 system interrupt protocol, lower priority devices or channels may not interrupt higher priority devices or channels which have not had their interrupt service routines completed. However, high priority devices or channels may interrupt the servicing of lower priority devices or channels.

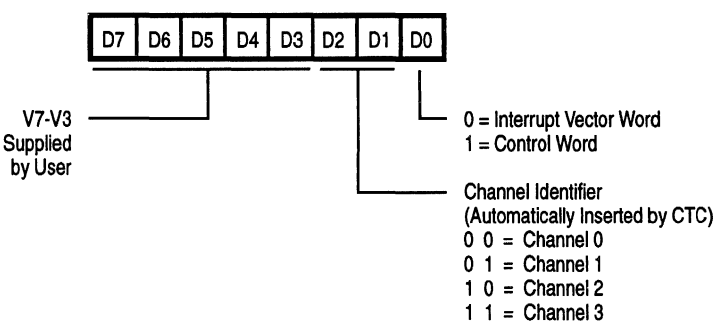


Figure 2-4. Interrupt Vector

2.2 INTERRUPT CONTROL LOGIC (Continued)

A CTC channel may be programmed to request an interrupt every time its down-counter reaches a count of zero. (To utilize this feature requires that the CPU be programmed for Interrupt Mode 2.) After the interrupt request, the CPU sends out an interrupt acknowledge. The CTC's interrupt control logic determines the highest-priority channel requesting an interrupt. If the CTC's IEI input is active, indicating that it has priority within the system daisy chain, it places an 8-bit interrupt vector on the system data bus. The high order five bits of this vector will have been written to the CTC earlier as part of the CTC initial programming process; the next two bits will be provided by the CTC's interrupt control logic as a binary code corresponding to the highest-priority channel requesting an interrupt; finally the low-order bit of the vector will always be zero according to a convention (Figure 2-4).

This interrupt vector is used to form a pointer to a location in memory where the address of the interrupt service routine is stored in a table. The vector represents the least significant eight bits. The CPU reads the contents of the I register to provide the most significant eight bits of the 16-bit pointer. The address pointed to in memory contains the low-order byte and the next highest address contains the high-order byte of an address which in turn contains the first opcode of the interrupt service routine. Thus, in Mode 2, a single 8-bit vector stored in an interrupting CTC can result in an indirect call to any memory location (Figure 2-5).

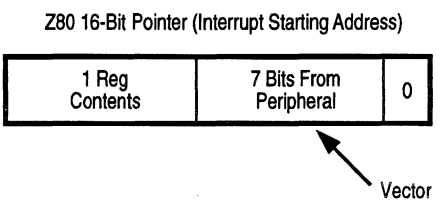


Figure 2-5. Z80 16-Bit Pointer (Interrupt Starting Address)

According to Z80 system convention, all addresses in the interrupt service routine table have their low-order byte in an even location in memory, and their high-order byte in the next highest location in memory. This location will always be odd so that the least significant bit of any interrupt vector will always be even. Thus, the least significant bit of any interrupt vector will always be zero.

The RETI instruction is used at the end of an interrupt service routine to initialize the daisy chain enable line IEO for proper control of nested priority interrupt handling. The CTC monitors the system data bus and decodes this instruction when it occurs. Thus, the CTC channel control logic knows when the CPU has completed servicing an interrupt.

CHAPTER 3
CTC PIN DESCRIPTION

3.0 PIN FUNCTIONS

Diagrams of the Z80 CTC Pin Configuration and Z80 CTC Package Configuration are shown in Figures 3-1 through 3-4, respectively. This section describes the function of each pin.

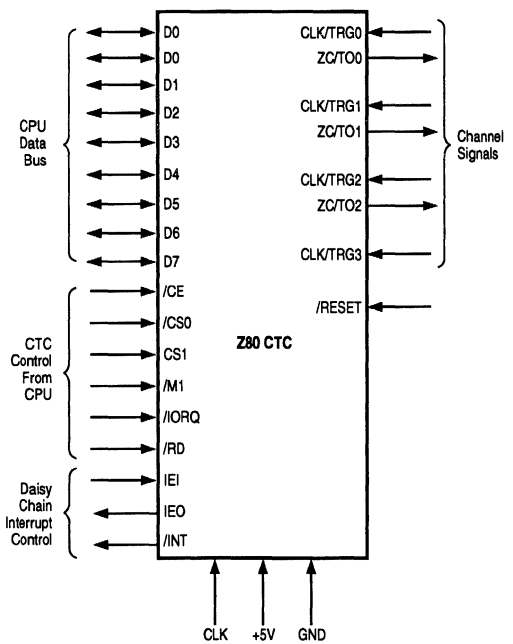


Figure 3-1. CTC Pin Configuration

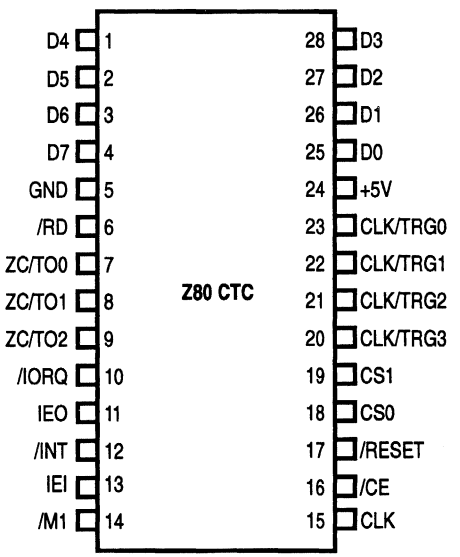


Figure 3-2. Package Configuration

3.0 PIN FUNCTIONS (Continued)

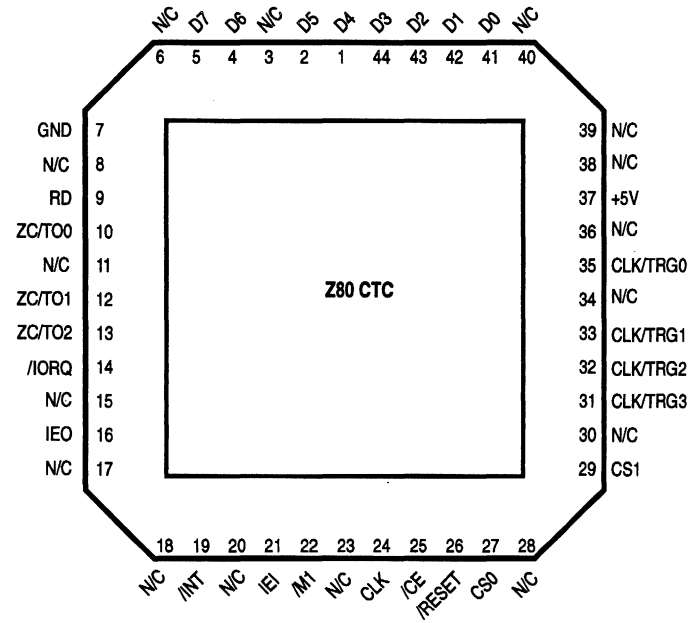


Figure 3-3. 44-pin Chip Carrier Pin Assignments

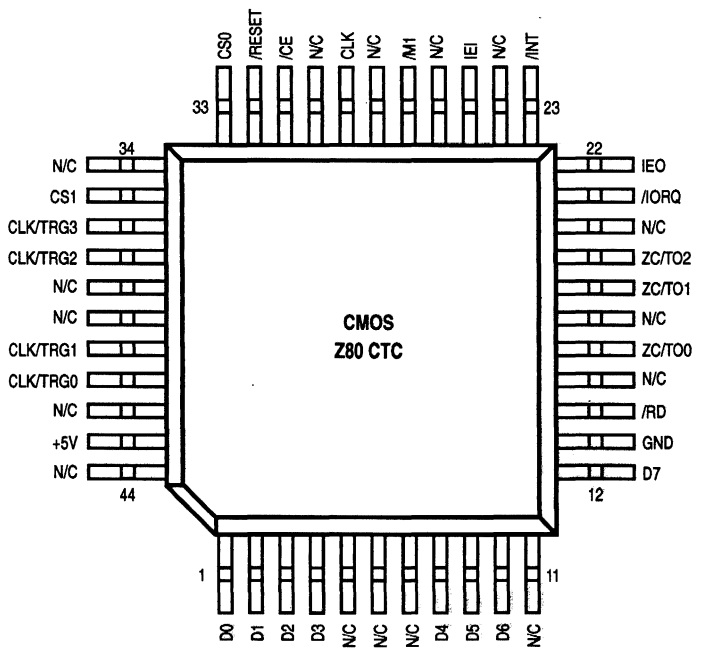


Figure 3-4. 44-pin Quad Flat Pack Pin Assignments

D7-D0 *System Data Bus* (bidirectional, tri-state). This bus is used to transfer all date and command words between the Z80 CPU and the Z80 CTC. There are eight bits on this bus, of which D0 is the least significant.

CS1-CS0 *Channel Select* (input, active High). These pins form a 2-bit binary address code for selecting one of the four independent CTC channels for an I/O Write or Read. (See Truth Table 3-1).

Table 3-1. Channel Select Truth Table

	CS1	CS0
Channel 0	0	0
Channel 1	0	1
Channel 2	1	0
Channel 3	1	1

/CE *Chip Enable* (input, active Low). A low level on this pin enables the CTC to accept control words, interrupt vectors, or time constant data words from the Z80 data bus during an I/O Write cycle; or to transmit the contents or the down-counter to the CPU during an I/O Read cycle. In most applications this signal is decoded from the eight least significant bits of the address bus for any of the four I/O port addresses that are mapped to the four Counter/Timer channels.

Clock (Φ) *System Clock* (input). This single-phase clock is used by the CTC to synchronize certain signals internally.

/M1 *Machine Cycle One Signal from CPU* (input, active low). When /M1 is active and the /RD signal is active, the CPU is fetching an instruction from memory. When /M1 is active and the /IORQ signal is active, the CPU is acknowledging an interrupt, alerting the CTC to place an interrupt vector on the Z80 data bus if it has daisy-chain priority and one of its channels has requested an interrupt.

/IORQ *Input/Output Request from CPU* (input, active Low). The /IORQ signal is used in conjunction with the /CE and /RD signals to transfer data and channel control words between the Z80 CPU and the CTC. During a CTC Write cycle, /IORQ and /CE must be true and /RD false. The CTC does not receive a specific write signal, instead it generates its own internally from the inverse of a valid /RD signal. In a CTC Read cycle, /IORQ, /CE, and /RD must be active to place the contents of the down-counter on the Z80 data bus. If /IORQ and /M1 are both true, the CPU is acknowledging an interrupt request, and the highest priority interrupting channel will place its interrupt vector on the Z80 data bus.

/RD *Read Cycle Status from the CPU* (input, active Low). The /RD signal is used in conjunction with the /IORQ and /CE signals to transfer data and channel control words between the Z80 CPU and the CTC. During a CTC Write Cycle, /IORQ and /CE must be true and /RD false. The CTC does not receive a specific write signal, instead it generates its own internally from the inverse of a valid /RD signal. In a CTC Read cycle, /IORQ, /CE, and /RD must be active to place the contents of the down-counter on the Z80 data bus.

IEI *Interrupt Enable In* (input, active High). This signal is used to form a system-wide interrupt daisy-chain which establishes priorities when more than one peripheral device in the system has interrupting capability. A high level on this pin indicates that no other interrupting devices of higher priority in the daisy chain are being serviced by the Z80 CPU.

IEO *Interrupt Enable Out* (output, active High). The IEO signal, in conjunction with IEI, is used to form a system-wide interrupt priority daisy chain. IEO is High only if IEI is High and the CPU is not servicing an interrupt from any CTC channel. Thus, this signal blocks lower priority devices from interrupting while a higher priority interrupting device is being serviced by the CPU.

/INT *Interrupt Request* (output, open-drain, active Low). This signal goes true when a CTC channel (which has been programmed to enable interrupts) has a zero-count condition in its down-counter.

/RESET *Reset* (input, active Low). This signal stops all channels from counting and resets interrupt enable bits in all control registers, thereby disabling CTC-generated interrupts. The ZC/TO and /INT outputs go to their inactive states, IEO reflects IEI, and the CTC's data bus output drivers go to the high-impedance state.

CLK/TRG3-CLK/TRG0 *External Clock/Timer Trigger* (input, user-selectable active High or Low). There are four CLK/TRG pins, corresponding to the four independent CTC channels. In the Counter mode, every active edge on this pin decrements the down-counter. In the Timer mode, an active edge on this pin initiates the timing function. The user may select the active edge to be either rising or falling.

ZC/TO2-AC/TO0 *Zero Count/Timeout* (output, active High). There are three ZC/T0 pins, corresponding to CTC Channels 2 through 0. (Due to package pin limitations Channel 3 has no ZC/TO pin.) In either Counter mode or Timer mode, when the down counter decrements to zero, an active High going pulse appears at this pin.

CHAPTER 4
CTC OPERATING MODES

4.0 INTRODUCTION

At power-on, the Z80 CTC state is undefined. Asserting /RESET puts the CTC in a known state. Before a channel can begin counting or timing, a channel control word and a time constant data word must be written to the appropriate register's of that, channel. Additionally, if a channel has been programmed to enable interrupts, an interrupt vector word must be written to the CTC's interrupt control logic. (For further details, refer to Chapter 5, "CTC Programming.") When the CPU has written all of these words to the CTC, all active channels are programmed for immediate operation in either the Counter mode or the Timer mode.

4.1 CTC COUNTER MODE

In CTC Counter mode, the CTC counts edges of the CLK/TRG input this mode is programmed for a channel when its channel control word is written with bit 6 set. The channel's external clock (CLK/TRG) input is monitored for a series of triggering edges. After each, in synchronization with the next rising edge of Φ (the System clock), the down-counter (which is initialized with the Time Constant Data word at the start of each sequence of down-counting) is decremented. Although there is no set-up time requirement between the triggering edge of the External clock and the rising edge of Φ, (Clock), the down-counter is not decremented until the following pulse. A channel's External clock input is pre-programmed by bit 4 of the channel control word to trigger the decrementing sequence with either a high- or a low-going edge.

In Channels 0, 1, or 2, when the down-counter is successively decremented from the original time constant (until it reaches zero), the Zero Count (ZC/TO) output pin for that channel will be pulsed active (High). (Due to package pin limitations, Channel 3 does not have this pin and so may only be used in applications where this output pulse is not required.) Additionally, if the channel is pre-programmed by bit 7 of the channel control word, an interrupt request sequence will be generated. (For more details, see Chapter 7, "CTC Interrupt Servicing.")

The zero-count condition also results in the automatic reload of the down-counter with the original time constant data word in the Time Constant register. There is no interruption in the sequence of continued down-counting. If the Time Constant register is written to with a new time constant data word while the down-counter is decrementing, the present count is completed before the new time constant is loaded into the down-counter.

4.2 CTC TIMER MODE

In CTC Timer mode, the CTC generates timing intervals that are an integer value of the system clock period. This is programmed for a channel when its channel control word is written with bit 6 reset. The channel then may be used to measure intervals of time based on the System clock period. The System clock is fed through the prescaler and the down-counter. Depending on the pre-programmed bit 5 in the channel control word, the prescaler divides the System clock by a factor of 16 or 256.

The output of the prescaler is then used as a clock to decrement the down-counter, which may be pre-programmed with any time constant integer between 1 and 256. The time constant is automatically reloaded into the down-counter at each zero-counter condition. At zero-count, the channel's Time Out (ZC/TO) output (which is the output of the down-counter) is pulsed, resulting in a uniform pulse train of precise period given by the product.

$$t_c * P * TC$$

Where t_c is the System clock, P is the prescaler factor of 16 or 256, and TC is the pre-programmed time constant.

Timing may be initialized automatically or with a triggering edge at the channel's Timer Trigger (CLK/TRG) input. This is determined by programming bit 3 of the channel control word. If bit 3 is reset, the timer automatically begins operation at the start of the CPU cycle following the I/O Write machine cycle that loads the time constant data word to the channel. If bit 3 is set, the timer begins operation on the second succeeding rising edge of Φ after the Timer Trigger edge following the loading of the rime constant data word. If no time constant word is to follow, the timer begins operation on the second succeeding rising edge of Φ after the Timer Trigger edge and following the control word write cycle. Bit 4 of the channel control word is pre-programmed to select whether the Timer Trigger will be sensitive to a rising or falling edge. There is no setup requirement between the active edge of the Timer Trigger and the next rising edge of Φ. If the Timer Trigger edge occurs closer than a specified minimum setup time to the rising edge of Φ, the down-counter does not begin decrementing until the following rising edge of Φ.

If bit 7 in the channel control word is set, the zero-count condition in the down-counter causes a pulse at the channel's Time Out pin, and initiates an interrupt request sequence. (For more details, see Chapter 7, "CTC Interrupt Service.")

CHAPTER 5
CTC PROGRAMMING

5.0 INTRODUCTION

To begin counting or timing operations, a channel control word and time constant data word are written to the appropriate channel by the CPU. These words are stored in the Channel Control or Time Constant registers of each channel. If a channel has been programmed to enable interrupts, an interrupt vector is written to the appropriate register in the CTC. Due to automatic features in the interrupt control logic, one pre-programmed interrupt vector suffices for all four channels.

5.1 LOADING THE CHANNEL CONTROL REGISTER

To load a channel control word, the CPU performs a normal I/O Write sequence to the port address corresponding to the desired CTC channel. The CTC input pins CS0 and CS1 are used to form a 2-bit binary address to select one of four channels within the device. (For a truth table, see section 2.1.1, "The Channel Control Register and Logic".) In many system architectures, these two input pins are connected to Address Bus lines A0 and A1, respectively, so that the four channels in a CTC device occupy contiguous I/O port addresses. A word written to a CTC channel is interpreted as a channel control word, and loaded into the channel control register (bit 0 is a logic 1). The other seven bits of this word select operating modes and conditions as indicated in Figure 5-1.

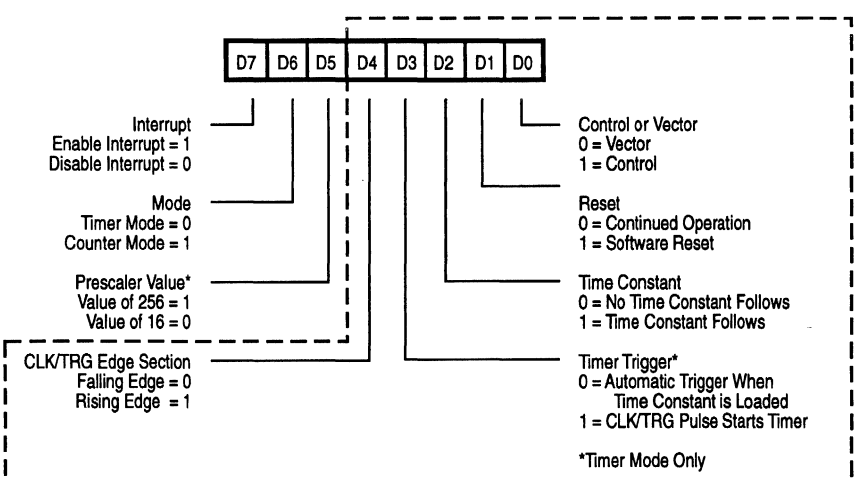


Figure 5-1. Channel Control Register

5.1 LOADING THE CHANNEL CONTROL REGISTER (Continued)

Bit 7 = 1
Each channel is enabled to generate an interrupt request sequence when the down-counter reaches a zero-count condition. To set the Interrupt bit to 1 in any of the four Channel Control registers an interrupt vector is written to the CTC before operation begins. Channel interrupts may be programmed in either Counter or Timer mode. If an updated channel control word is written to a channel in operation, with bit 7 set, the interrupt enable selection is not retroactive to a preceding zero-count condition.

Bit 7 = 0
Channel interrupts disabled.

Bit 6 = 1
Counter mode selected. The down-counter is decremented by each triggering edge of the External clock (CLK/TRG) input. The prescaler is not used.

Bit 6 = 0
Timer mode selected. The prescaler is clocked by the System clock Φ, and the output of the prescaler in turn clocks the down-counter. The output of the down-counter (the channel's ZC/TO output) is a uniform pulse train of period given by the product

$$t_C * P * TC$$

where t_C is the period of System clock, P is the prescaler factor of 16 or 256, and TC is the time constant data word.

Bit 5 = 1
(Defined for Timer mode only.) Prescaler factor is 256.

Bit 5 = 0
(Defined for Timer mode only.) Prescaler factor is 16.

Bit 4 = 1
TIMER MODE: positive edge trigger starts timer operation.

COUNTER MODE: positive edge decrements the down-counter.

Bit 4 = 0
TIMER MODE: negative edge trigger starts timer operation.

COUNTER MODE: negative edge decrements the down-counter.

Bit 3 = 1
Timer Mode Only - External trigger is valid for starting timer operation after rising edge of T2 of the machine cycle following the one that loads the time constant. The prescaler is decremented two clock cycles later if the setup time is met, otherwise three clock cycles.

Bit 3 = 0
Timer Mode Only - Timer begins operation on the rising edge of T2 of the machine cycle following the one that loads the time constant.

Bit 2 = 1
The time constant data word for the Time Constant register is the next word written to this channel. If an updated channel control word and time constant data word are written to a channel while it, is already in operation, the down-counter continues decrementing to zero before the new time constant is loaded.

Bit 2 = 0
No time constant date word for the Time Constant register is to follow. The channel control word updates the status of a channel already in operation (a channel will not operate without a correctly programmed data word in the lime Constant register). Bit 2 in the channel control word must be set in order to write to the Time Constant register.

Bit 1 = 1
Counting and/or timing operation is terminated and the channel is reset. This is not a stored condition. The bits in the Channel Control register are unchanged. If bits 1 and 2 are set to 1, the channel resumes operation upon loading a time constant.

Bit 1 = 0
Channel continues current operation.

5.2 LOADING THE TIME CONSTANT REGISTER

A time constant data word is written to the Time Constant register by the CPU. This occurs on the I/O Write Cycle following that of the channel control word. The time constant data word may be any integer value in the range 1-256 (Figure 5-2). If all eight bits in this word are zero, it is interpreted as 256. If a time constant date word is loaded to a channel already in operation, the down-counter continues decrementing to zero before the new time constant is loaded.

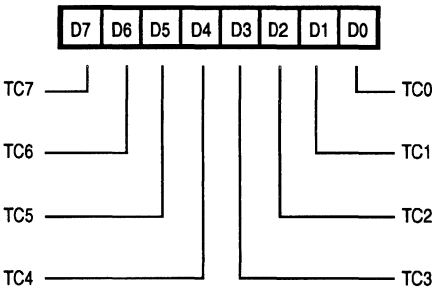


Figure 5-2. Time Constant Register

B

5.3 LOADING THE INTERRUPT VECTOR REGISTER

The Z80 CTC operates with the Z80 CPU programmed for mode 2 interrupt response. When a CTC interrupt request is acknowledged, a 16-bit pointer is formed to obtain a corresponding interrupt service routine starting address (Figure 5-3). The upper eight bits of this pointer are provided by the CPU's I register; the lower eight bits are provided by the CTC in the form of an interrupt vector unique to the requesting channel (Figure 5-4). (For further details, see Chapter 7, "CTC Interrupt Servicing".)

The five high-order bits of the interrupt vector are written to the CTC in advance as part of the initial programming sequence. The CPU writes to the I/O port address corresponding to the CTC Channel 0. A 0 in bit 0 signals the CTC to load the incoming word into the Interrupt Vector register. When the interrupt vector is placed on the Z80 data bus, the interrupt control logic of the CTC automatically supplies a binary code in bits 1 and 2 identifying which of the four CTC channels is to be serviced.

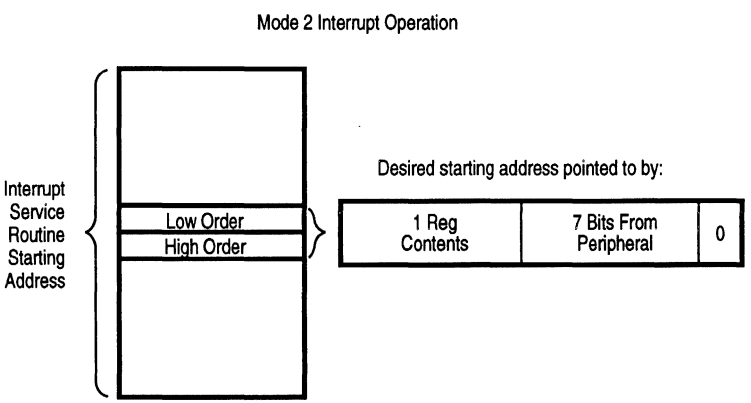


Figure 5-3. Mode 2 Interrupt Operation

5.3 LOADING THE INTERRUPT VECTOR REGISTER (Continued)

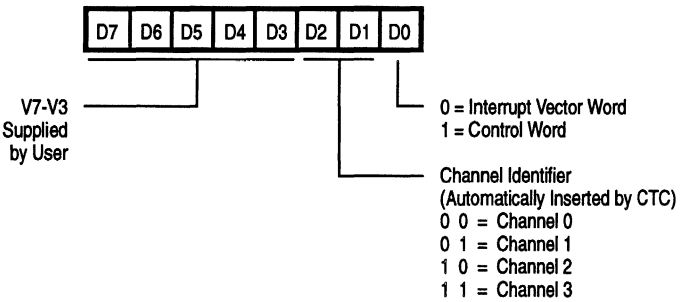


Figure 5-4. Interrupt Vector Register

CHAPTER 6
CTC TIMING

6.0 INTRODUCTION

This chapter illustrates the timing relationships of the relevant CTC pins for the following types of operation: writing a word to the CTC, reading a word from the CTC, counting, and timing. A timing diagram, Figure 7-1, relating to interrupt servicing can be found in Section 7.1.

6.1 CTC WRITE CYCLE

Figure 6-1 illustrates the timing associated with the CTC Write cycle. This sequence is applicable to loading a channel control word, an interrupt vector, or a time constant data word.

In the sequence shown, during clock cycle T1, the Z80 CPU prepares for the Write cycle with a false (High) signal at CTC input pin /RD (Read). As the CTC has no separate Write signal input, it generates its own input internally from the false /RD input. During clock cycle T2, the Z80 CPU initiates the Write cycle with true (Low) signals at CTC input pins /IORQ (I/O Request) and /CE (Chip Enable). (See Note below.) A 2-bit binary code appears at CTC inputs CS1 and CS0 (Channel Select 1 and 0), specifying which of the four CTC channels is being written to. At this time, a channel control, interrupt vector, or time constant data word may be loaded into the appropriate CTC internal register in synchronization with the rising edge beginning clock cycle T3.

Note: /M1 must be false to distinguish the cycle from an interrupt acknowledge.

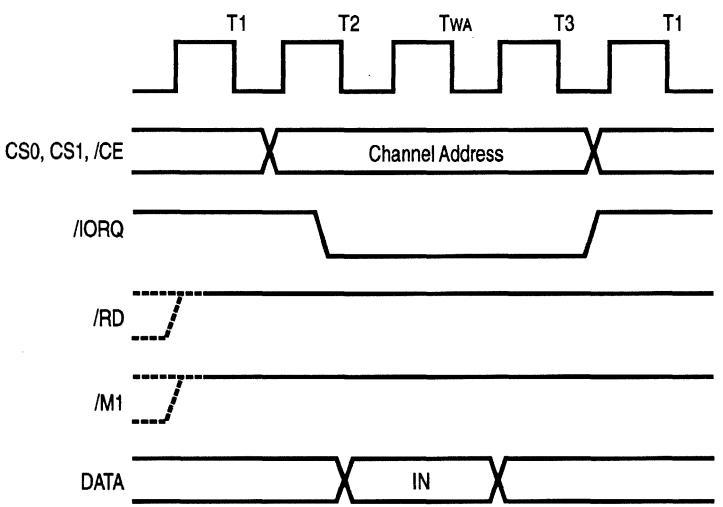


Figure 6-1. CTC Write Cycle

6.2 CTC READ CYCLE

Figure 6-2 illustrates the timing associated with the CTC Read cycle. This sequence is used when CPU reads the current contents of the down counter. During clock cycle T2, the Z80 CPU initiates the Read cycle with true signals at input pins /RD (Read), /IORQ (I/O Request), and /CE (Chip Enable). A 2-bit binary code appears at CTC inputs CS1 and CS0 (Channel Select 1 and 0), specifying which of the four CTC channels is being read from. (See Note below.) On the rising edge of the cycle T3, the valid contents of the down-counter rising edge of cycle T2 is available on the Z80 data bus. No additional wait states are allowed.

Note: /M1 must be false to distinguish the cycle from an interrupt-acknowledge.

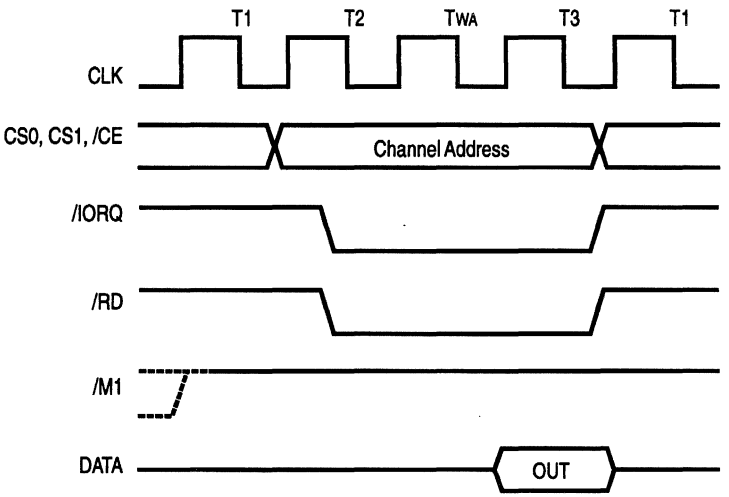


Figure 6-2. CTC Read Cycle

6.3 CTC COUNTING AND TIMING

Figure 6-3 illustrates the timing diagram for the CTC Counting and Timing modes.

In the Counter mode, the edge (rising edge is active in this example) from the external hardware connected to pin CLK/TRG, decrements the down-counter in synchronization with the System Clock Φ. This CLK/TRG pulse must have a minimum width and the minimum period must not be less than twice the System clock period. Although there is no setup time requirement between the active edge of the CLK/TRG and the rising edge of Φ, if the CLK/TRG edge occurs closer than a specified minimum time, the decrement of the down-counter will be delayed one cycle of Φ. Immediately after the 1 to 0 decrement of the down-counter, the ZC/TO output is pulsed true.

In the Timer mode, a pulse trigger (user-selectable as either active High or active Low) at the CLK/TRG pin enables the timing function on the second succeeding rising edge of Φ. As in the Counter mode, the triggering pulse is detected asynchronously and must have a minimum width. The timing function is initiated in synchronization with Φ. A minimum setup time is required between the active edge of the CLK/TRG and the rising edge of Φ. If the CLK/TRG edge occurs closer than this, the initiation of the timer function will be delayed one cycle of Φ.

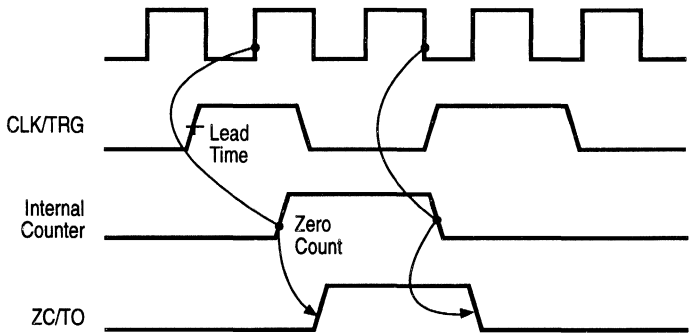


Figure 6-3. CTC Counting and Timing

CHAPTER 7
CTC INTERRUPT SERVICING

7.0 INTRODUCTION

Each CTC channel may be individually programmed to request an interrupt every time its down-counter, reaches a count of zero. The purpose of a CTC generated interrupt is to force the CPU to execute an interrupt service routine. To utilize this feature the Z80 CPU must be programmed for mode 2 interrupt response. In this mode, when a CTC channel interrupt request is acknowledged, a 16-bit pointer must be formed to obtain a corresponding interrupt service routine. The lower eight bits of the pointer are provided by the CTC in the form of an interrupt vector unique to the requesting channel. (For further details, refer to Chapter 8.0 of the Z80 CPU Technical Manual.)

The CTC's interrupt control logic insures that it acts in accordance with Z80 system interrupt protocol for nested priority interrupt and proper return from interrupt. The priority of any system device is determined by its physical location in a daisy-chain configuration. Two signal lines (IEI and IEO) are provided in the CTC to form the system daisy chain. The device closest to the CPU has the highest priority. Interrupt priority is predetermined by channel number, with Channel 0 having highest priority. According to Z80 system interrupt protocol, low priority devices or channels may not interrupt higher priority devices or channels that have not had their interrupt service routines completed. High priority devices or channels may interrupt the servicing of lower priority devices or channels. (For further details, see Chapter 2, "Interrupt Control Logic".)

Sections 7.2 and 7.3 describe the nominal timing relationships of the relevant CTC pins for the Interrupt Acknowledge cycle and the Return from Interrupt cycle. Section 7.4 discusses a typical example of daisy-chain interrupt servicing.

7.1 INTERRUPT ACKNOWLEDGE CYCLE

Figure 7-1 illustrates the timing associated with the Interrupt Acknowledge cycle. After an interrupt is requested by the CTC, the CPU sends out an interrupt acknowledge (/M1 and /IORQ). To insure that the daisy-chain enable lines stabilize, channels are inhibited from changing their interrupt request status when /M1 is active. /M1 is active two clock cycles earlier than /IORQ and /RD is false to distinguish the cycle from an instruction fetch. During this time the interrupt logic of the CTC determines the highest priority channel requesting an interrupt. If the CTC Interrupt Enable input (IEI) is active, the highest priority interrupting channel within the CTC places its interrupt vector onto the data bus when /IORQ goes active. Two wait states (TW*) are automatically inserted at this time to allow the daisy chain to stabilize. Additional wait states may be added.

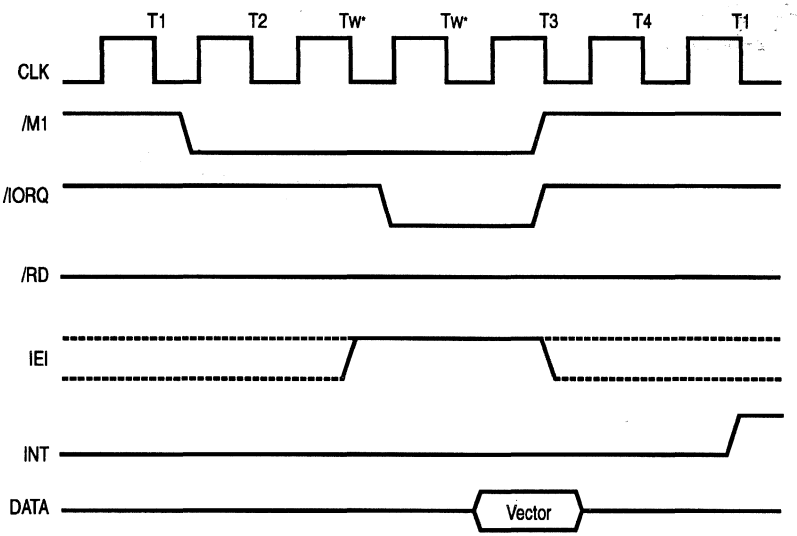
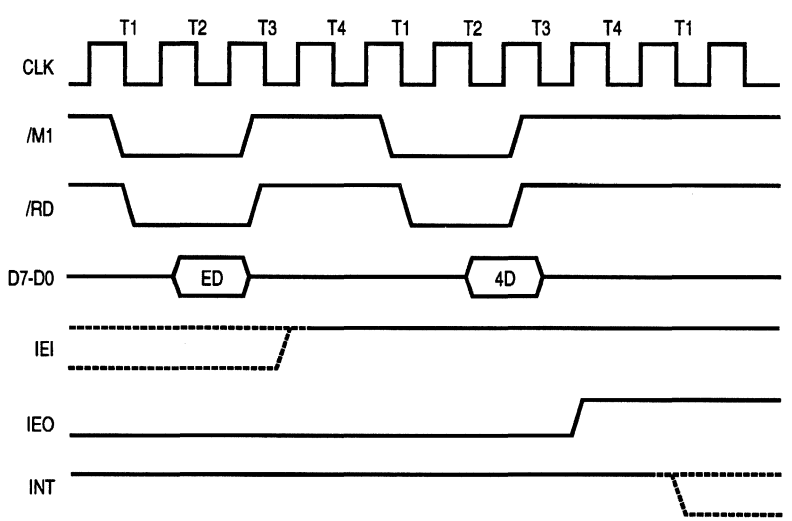


Figure 7-1. Interrupt Acknowledge Cycle

7.2 RETURN FROM INTERRUPT CYCLE

Figure 7-2 illustrates the timing associated with the RETI Instruction. This instruction is used at the end of an interrupt service routine to initialize the daisy-chain enable lines for proper control of nested priority interrupt handling. The CTC decodes the 2-byte RETI code internally and determines whether it is intended for a channel being serviced.

When several Z80 peripheral chips are in the daisy-chain, IEI will become active on the chip currently under service when an EDH opcode is decoded. If the following opcode is 4DH, the peripheral being serviced will be re-initialized and its IEO will become active.



*INT will go Low if more interrupts pending on the RTC.

Figure 7-2. Return from Interrupt Cycle

7.3 DAISY-CHAIN INTERRUPT SERVICING

Figure 7-3 illustrates a typical nested interrupt sequence which may occur in the CTC. In this example, Channel 2 interrupts and is granted service. While this channel is being serviced, higher priority Channel 1 interrupts and is granted service. The service routine for the higher priority channel is completed, and a RETI instruction is executed to signal the channel that its routine is complete (see Section 7.2 for further details). At this time, the service routine of the lower priority Channel 2 is resumed and completed.

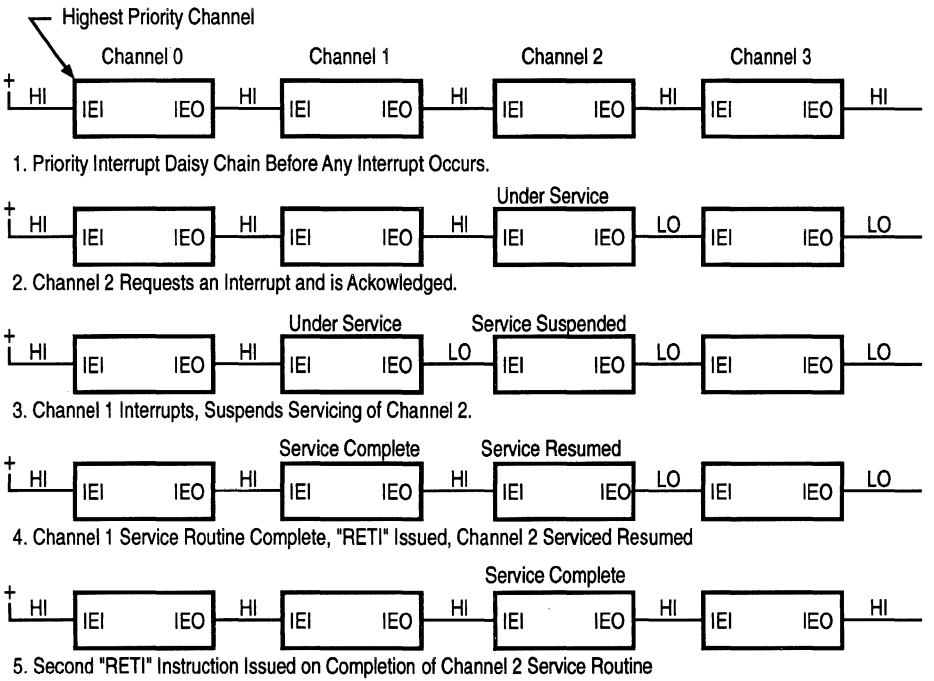


Figure 7-3. Daisy-Chain Interrupt Servicing

ZILOG

TABLE OF CONTENTS

C

List of Figures

List of Tables

CHAPTER 1
INTRODUCTION

1.0 WHY IS DMA USEFUL?

Before describing the Z80® DMA device in detail, we will review the subjects of direct memory access (DMA) and direct memory access controllers (DMACs) in general. This will provide a background of functions and terminology for 8-bit single-bus microcomputer environments used throughout the remainder of the manual.

DMACs are dedicated to the task of controlling high-speed block transfers of data independently of the CPU. Typically, these transfers are between memory and I/O, or vice versa, although a few DMACs can perform other types of transfers that have traditionally been done by the CPU. For example, the Z80 DMA can perform memory-to-memory and I/O-to-I/O transfers, as well as searching for particular patterns of bits in a byte either simultaneously with or independently of transfers.

1.0.1 CPU Transfers

In systems without DMA, data transfers must pass through the CPU and, therefore, must be implemented in software. This normally involves the execution of an instruction sequence. for inputting, outputting, and tracking each byte of data in the block to be transferred.

Figure 1-1 illustrates the minimum sequence, of instructions that must be fetched from memory and executed by conventional CPUs to transfer a block of data one. byte at a time. In fact, most CPUs require many more instructions than are shown here.

One result of this method is that CPU transfers are relatively slow and tie up the CPU for long periods of time. Another result is that response time (startup time for the first byte) is also usually slow, because the I/O device typically uses interrupts to signal its readiness, and the CPU interrupt service routine causes a significant time lag in transferring the first byte.

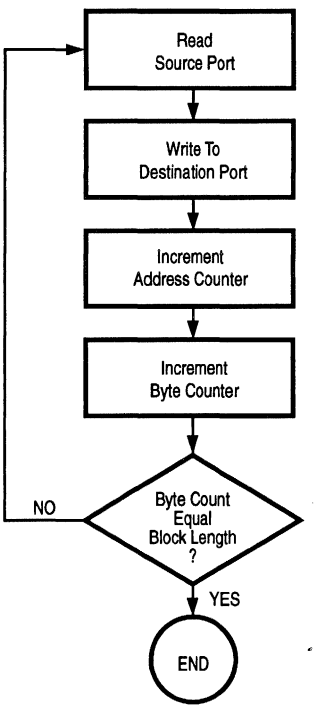


Figure 1-1. Typical CPU I/O Sequence

The Z80 and Z8000 CPUs are unique in that they both have block-transfer and string-search instructions that can operate on up to 64 Kbytes of data with a single instruction. A single block transfer instruction repetitively performs all of the functions illustrated in Figure 1-1 on an entire sequence of bytes, and the transfer rates achievable are far better than in other CPUs. The 4 MHz Z80A CPU can transfer at about 200 Kbytes/s and the 4 MHz Z8000 CPU can reach 800 Kbytes/s, which is faster than some DMACs.

WHY IS DMA USEFUL? (Continued)

The problem with CPU block transfers in the Z80 and Z8000 Families is, therefore, usually not speed of transfer but response time at startup. One of the following methods is normally used to set up for execution of the block transfer instruction:

- The I/O device interrupts the CPU and the block transfer instruction is executed in the CPU interrupt service routine. This entails a response time of at least 5 to 10 µs, even in the 4 MHz Z80A and Z8000 Families, which have one of the fastest interrupt-handling capabilities available.

- The CPU begins executing the device service routine before the I/O device is ready, and a flag bit is constantly polled by the CPU. When the bit indicates that the device is ready, the CPU jumps to the block transfer instruction. This method can sometimes allow a response time of less than 5 µs, but it totally occupies the CPU.

- The CPU begins executing the block transfer instruction in an interrupt service routine before the, I/O device is actually ready. But the I/O device idles the CPU with the Wait line just after the Read and Chip-Select lines become active. When the I/O device is ready, it releases the Wait line and the transfer is completed. This gives by far the best response time (250 ns in a 4 MHz Z80A or Z8000 CPU), but it totally ties up the bus.

In summary, both transfer and response times on most CPUs are often too slow. While transfer speed can he quite high with the Z80 and Z8000 CPUs, the response time may be too long in some interrupt-driven transfer situations.

1.0.2 DMA Transfers

A DMA controller performs direct transfers between the source and destination of data, without going through the CPU and without the instruction fetches required by the CPU. It performs all of the steps illustrated in Figure 1-1 through hardware.

In a memory-to-I/O transfer, for example, the starting address in memory and the length of the block to be transferred are written into the DMA by the CPU prior to the transfer. The DMAC quickly takes control and begins transferring data when the CPU enables the DMAC and the I/O device's Ready line becomes active. In most cases, the CPU is idled during this process. When the transfer is complete, the DMAC signals the CPU and releases control.

DMACs are used, therefore, when one or more of the following situations or requirements are present:

- The CPU is loaded down with too much I/O to perform its other tasks properly.

- The transfer must be faster than the CPU could perform it.

- The transfer response time (startup) must be faster than the CPU can conveniently provide.

Small and low-performance systems generally run without DMA. Medium-performance systems can also be designed without DMA if the CPU can handle transfers fast enough and still do its other work.

Whenever systems require fast transfers or fast response, DMACs are strong candidates for performance enhancement. Not only do they transfer faster than CPUs (with the possible exceptions of the Z80 and Z8000 CPUs), but their response time is inherently better and can be improved with the same techniques described above for CPU response.

Here are a few examples where DMA is often the best choice:

- Disk and diskette controllers

- Scanning operations, such as CRT I/O

- Data acquisition

- Memory-to-memory transfers

- Memory searches

- Backup storage (I/O-to-I/O)

- Parallel bus systems like the IEEE 488

- Fiber optic links

- Block transfers in networking, multiprocessing, or multiprogramming

The trade-off for this speed is that the CPU typically remains idle and lacks full or partial control of the system bus while the DMA is operating. This can affect not only total system throughput, but it can also affect such things as memory refresh and other interrupts.

We will discuss these issues in greater detail in the chapter entitled "Performance Limitations." Speed comparisons for the Z80 Family are given in the next chapter entitled "Functional Description of the Z80 DMA."

1.1 DMA CHARACTERISTICS

All DMACs are programmable because, at a minimum, the CPU must write a block length (byte count) and starting memory address into them before they can begin managing a data transfer. The starting address is incremented or decremented as the transfer proceeds, and the byte counter is incremented from zero up to the specified block length.

Beyond this, however, DMACs vary substantially in their characteristics and capabilities. The discussion that follows is a general overview of the characteristics of DMACs, with only occasional reference to the Z80 DMA in particular.

1.1.1 Ports and Channels

Every data transfer has a source port and a destination port. For example, in memory-to-I/O transfers, memory is the source port and I/O is the destination port. The means of controlling and tracking the exchange of data between the two ports is called a "channel." A channel includes the hardware for address and byte counting, bus control, and coordination of the entire transfer process.

Each port in a channel has its location specified either by the DMA address-generation mechanism or by hardwiring. The Z80 DMA, unlike other 8-bit DMACs, generates addresses for both memory and I/O ports during each byte transfer; in other DMACs, the I/O port is hardwired.

Some DMACs have multiple channels. This usually means that they can keep track of multiple interleaved transfers and that one DMA can be hardwired to multiple I/O devices. However, because any DMA can execute only one read and/or write cycle at a time, multiple channels do not imply higher throughput than single channels of a given speed. The Z80 DMA, which is called a single-channel device, has the distinction of having the fastest maximum transfer rate and generating authentic I/O port addresses on the address bus. Moreover, the ability to generate two addresses means that the Z80 DMA can do memory-to-memory transfers in a single channel whereas others either cannot do them at all or require two channels to do them.

The ability to perform internal byte searches is another feature available on the Z80 DMAs single channel and is not available on other 8-bit DMACs. The Z80 DMA is the only DMAC that loads bytes into an internal DMAC register, during transfers with the result that, once the byte is loaded, it can be compared with a maskable control byte. Valid comparisons can then be made to cause various actions by the Z80 DMA.

1.1.2 Transfer Methods

Earlier we mentioned the differences between handling I/O by conventional CPU instructions versus direct memory access. Figure 1-2 expands this theme by comparing not only conventional CPU instructions but also the Z80 and Z8000 CPU block-transfer instructions along with two different methods of DMA transfer. This figure shows the read and write cycles needed to accomplish the transfer of a single byte of data.

Figure 1-2a illustrates conventional CPU I/O instruction activity. The number of read and write cycles is approximate—many CPUs require more cycles than are pictured. At a minimum, the CPU instruction causes all the steps illustrated earlier in Figure 1-1, plus additional housekeeping such as testing to see if the next byte is ready for transfer.

Figure 1-2b shows Z80 and Z8000 CPU block transfer instructions. Again, these are approximate and entail somewhat more activity than one read cycle and one write cycle after initiation, particularly with the Z80 CPU. A single block-transfer instruction, however, is capable of transferring up to 64 Kbytes of data.

Figure 1-2c illustrates "sequential" or "flow-through" DMA transfer, in which a byte is read from the source port into the DMA and then subsequently written to the destination port. This method can be implemented on the Z80 DMA with no external logic in a Z80 CPU environment (its liming characteristics and pin interfacing to the CPU are highly uniform in this case). Sequential transfer provides speeds that match or exceed the capability of most serial communication processors and many other I/O or memory devices.

Figure 1-2d illustrates "simultaneous" or "flyby" DMA transfer, in which a byte is both read and written in the same machine cycle. The Read and Write control lines are both active simultaneously, and the source and destination are determined by signals that specify either a memory-read/I/O-write or an I/O read/memory-write. This is the fastest transfer method, but the external logic required makes timing interfaces to memory and I/O somewhat more complicated.

1.1 DMA CHARACTERISTICS (Continued)

Another method also used on certain DMACs is called "transparent" or "cycle-stealing" transfer. This technique works in a manner similar to Figures 1-2c and 1-2d, except instead of taking control of the bus it causes the DMA data transfers to be interleaved with CPU cycles where dynamic memory would otherwise be refreshed. This method also requires external logic and inhibits memory refresh when DMA is occurring. Additionally, it reduces DMA throughput in some cases.

All DMA transfers interrupt dynamic memory refresh by the CPU and most of them idle the CPU. It is, therefore, important to consider these implications when making the trade-off for higher DMA transfer speed.

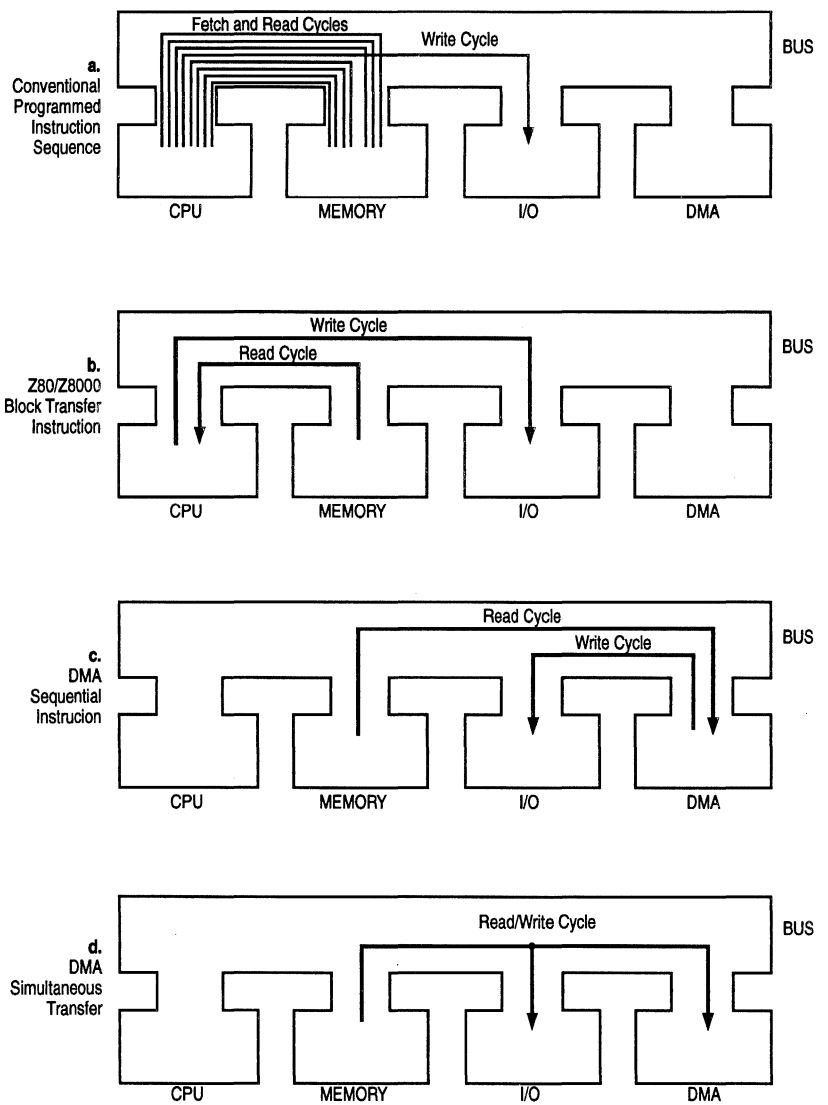


Figure 1-2. Conceptual Comparison of Various I/O Transfer Methods

1.1.3 Modes of Operation

Within each of the "methods" illustrated in Figures 1-2c and 2d there are as many as three "modes" of operation. These are termed Byte, Burst, and Continuous modes in this manual, although they are also sometimes referred to as Single, Demand, and Block modes. Figure 1-3 illustrates the typical sequence of events for each mode, once the I/O device's Ready signal to the DMAC has become active and before the DMA process reaches an end-of-block or other terminating condition. (These figures are expanded in Figures 2-3 through 2-6.)

In Byte mode, the DMAC transfers only one byte at a time while the I/O device Ready line is active. Control of the system bus is released back to the CPU between each byte. The CPU can then interleave its other activities, until the DMA makes a new request to the CPU for system bus control before transferring the next byte. Byte mode is related to the transparent method of transfer in that both cause interleaving of CPU and DMA functions. The Byte mode, however, includes the protocol of requesting and releasing the bus for each byte transfer.

In Burst mode, which is the most common mode, the DMAC continues to transfer bytes after gaining control of the system bus until the I/O device Ready line goes inactive. During this time, the CPU typically remains idle. When the Ready line goes inactive, the DMAC releases system bus control back to the CPU.

In Continuous mode, the DMAC holds the system bus until the entire block of data has been transferred. If the I/O device Ready line goes inactive before the block is completely transferred, the DMAC simply waits until it becomes active again, but the system bus is not released as in Burst mode. The Continuous mode is the fastest mode because it has the least response-time overhead when the Ready line momentarily goes inactive and returns active again. However, this mode does not allow any CPU activity for the duration of the transfer.

1.1.4 Bus Control

Most DMACs do not control the system bus in the same way that a CPU controls it. For example, many DMACs do not have a straightforward interface to the system data bus but rather multiplex a portion of the memory address onto the data bus, from which it must be latched by external logic. Nor do most DMACs generate all of the bus control signals that the CPU generates, and therefore they lack some, degree of bus control when they operate.

The Z80 DMA is unique among 8-bit DMACs in that it generates exactly the same bus control signals for read and write cycles that the Z80 CPU does, and in that it has exactly the same logical and electrical interface to the data and address buses as the CPU. This means the other system components cannot tell the difference between the Z80 DMA and CPU, control by these devices is totally interchangeable. In the sequential DMA transfer method (a read cycle followed by a write cycle), it also means that the Z80 DMA pins can be tied directly to the corresponding Z80 CPU pins without any of the external interfacing logic that other DMACs require. This property considerably simplifies design and lowers part counts.

1.1.5 Programmability

How a DMAC starts, transfers data, and stops is determined by control information written to the DMAC by the CPU prior to the transfer. Status registers, which can be read by the CPU to determine the transfer condition after the DMAC stops transferring, are also typically provided.

The degree of programmability is directly related to the DMACs flexibility for handling a variety of transfer tasks. Most DMACs are quite limited in their programmability. The Z80 DMA, by contrast, has over 140 bits of control information used to tailor the device (and retailor it between operations) for a wide variety of tasks and environments.

For example, the Z80 DMA can be programmed either to stop, interrupt the CPU, continue, or repeat a transfer when a target event such as an end-of-block, byte match, or Ready-line condition is reached. Alternatively, its buffered address counters can be reloaded during one byte-mode transfer so that the next transfer can begin quickly at a new location. Also, entire read and write cycle timings can be modified independently for each port to fit the requirements of other CPUs, memory, or I/O devices that are faster or slower than the standard Z80 Family timing.

This topic, as well as the others described earlier, are expanded in following chapters. We have introduced them here simply to give a generalized framework from which to launch a more detailed discussion of the Z80 DMA.

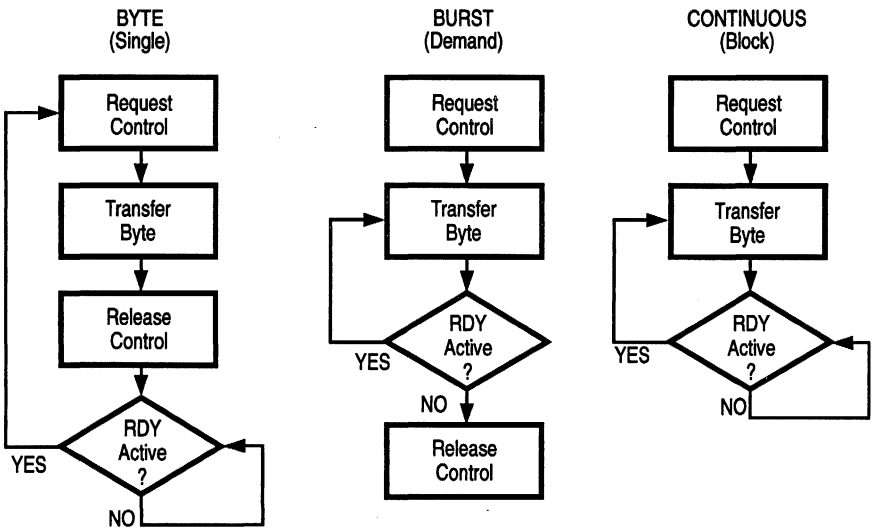


Figure 1-3. Modes of Operation
(See also Figures 2-3 through 2-6)

CHAPTER 2
FUNCTIONAL DESCRIPTION

2.0 DMA FEATURES

- Single Highly Versatile Channel
- Dual Port Address Generation with Incrementing, Decrementing, or Fixed Address in Both Ports
- Buffered Address and Block-Length Registers
- 64 Kbyte Maximum Block Length
- 2.4 or 4 MHz Clock Rates (Z80 or Z80A DMA)
- 1.25 or 2 Mbytes/s Data Rate (Z80 or Z80A DMA)
- Transfer, Search, or Transfer/Search Operations
- Bit-Maskable Byte Searching
- Sequential (Flow-Through) or Simultaneous (Flyby) Transfers
- Compatible with Z80 and Many Other CPUs
- Byte, Burst, and Continuous Modes
- Auto Restart Capability
- Variable Cycle Timing
- Wait-Line Cycle Extention
- Internally-Modifiable Interrupt Vectors
- Programmable InterruptSS on Ready, End-of-Block, Byte Match
- Hardware Priority Daisy Chains for Bus Requests and Interrupts
- Periodic Pulse Generation for External Device
- 21 Writeable Control Registers
- Seven Readable Status Registers
- Programmable Force Ready Condition
- Programmable Active State for Beady Line
- Programmable DMA Enable
- Complete System Bus Mastering
- No External Logic Needed for Sequential Transfers in Z80 Environments

2.1 OVERVIEW

The Z80 DMA performs data transfers and searches in a wide variety of 8-bit CPU environments. It is unique among DMACs in that it takes full control of the system address, data, and control buses—and is therefore a special-purpose processor—when enabled by the CPU to do so. The DMA also provides complete interfacing to the system bus. For example, in a Z80 CPU environment the Z80 DMA generates the same signal levels and timings, including tri-state control, that the Z80 CPU would generate to accomplish a transfer. It normally does this without external TTL packages, which all other DMAs require.

For this reason, and because of its extensive programmability for operating on data and data flow, the Z80 DMA can be called a special-purpose transfer processor. It unburdens not only the CPU but also the system designer.

The Z80 DMA is also unique in other respects. First, it generates two port addresses instead of one. Because both addresses can be either variable or fixed, this means that memory-to-memory or I/O-to-I/O transfers can be done with a single channel, whereas other DMACs either require more than one channel or cannot do such transfers at all.

2.1 OVERVIEW (Continued)

The capability of the Z80 DMAs channel surpasses the capability of any other available monolithic DMAC channel to service either fast or slow devices. In addition to having a Wait line for extending cycles, the basic read and write cycles can be programmed for different timing requirements. If multiple channels are needed, multiple Z80 DMAs can be integrated very easily. The interrupt structure is among the fastest and most versatile available. Interrupt signals and vectors can be generated under several conditions. Finally, the Z80 DMA is the only DMAC that passes data through itself and can therefore compare bytes against a bit-maskable match byte. An overview of Z80 DMA features are listed below and each point is described more thoroughly in this and other chapters.

Throughout the remainder of the manual we refer to the "Z80 DMA" or simply the "DMA." By this we mean either the 2.4 MHz Z80 DMA or the 4 MHz Z80A DMA device. Both parts have the same features and differ only in speed.

2.2 PROGRAMMING

The Z80 DMA has 21 writeable 8-bit control registers and seven readable 8-bit status registers available to the CPU. Control bytes can be written to the DMA or status bytes can be read from the DMA whenever the DMA is not controlling the bus.

Control bytes writeable to the DMA include those which effect immediate command actions such as enable, disable, reset, load starting addresses, continue transferring or searching, clear byte and address counters, clear status bits, and the like. In addition, many mode-setting control bytes can be written, including the class and mode of operation, port configuration, starting addresses, block length, address-counting rule, match and match-mask bytes, interrupt conditions, interrupt vector, end-of-block rule, Ready-line and Wait-line rules, and others.

Readable status registers include a general status byte reflecting Ready-line, end-of-block, byte-match, and interrupt conditions, as well as registers for the current byte count and port addresses. There is a full chapter on programming that explains these functions in detail, and most of them are described in general terms on the pages that follow.

2.3 CLASSES OF OPERATION

The Z80 DMA has three basic "classes" of operation, and two of the classes are each broken into subclasses as follows:

- Transfers of data between any two DMA ports:
  Sequential transfers (flow-through)
  Simultaneous transfers (flyby)
- Searches for a particular bit pattern within a byte at a single DMA port
- Combined transfers and searches between any two DMA ports:
  Sequential transfer/search
  Simultaneous transfer/search

Figure 2-1 illustrates these classes. The two subclasses of transfers are shown at the top, the search-only class is shown in the middle, and the two subclasses of transfer while searching are shown at the bottom. In all cases, the DMA assumes full control of the system address, data, and control buses while transferring or searching a given byte. The DMA ports are the source and destination of data: a "port" is used here to mean either memory or an I/O device.

In "sequential transfers," which are sometimes called flow-through transfers, each byte transfer entails a read cycle followed by a write cycle. The DMA reads the byte via the data bus into an internal register and sustains the byte on the data bus into the subsequent write cycle. In a Z80 CPU environment, as well as in certain other CPU environments, sequential DMA transfers can be implemented with no external logic between the DMA and the CPU.

In "simultaneous transfers," which are sometimes called flyby transfers, each byte is simultaneously read from the source into the DMA and written from the source directly to the destination in a single machine cycle. These transfers, therefore, occur at twice the rate of sequential transfers, but they require at least one external logic package to cause the proper signals to appear simultaneously on the control bus (see the "Applications" section).

In the "search only" class, each byte is read via the data bus from the source port into the DMA, where it is compared with a match byte. The match byte can optionally be masked with another byte so that only certain bits in the data and match bytes are compared. The search-only class needs no external logic between the DMA and CPU in Z80 systems and certain other CPU environments.

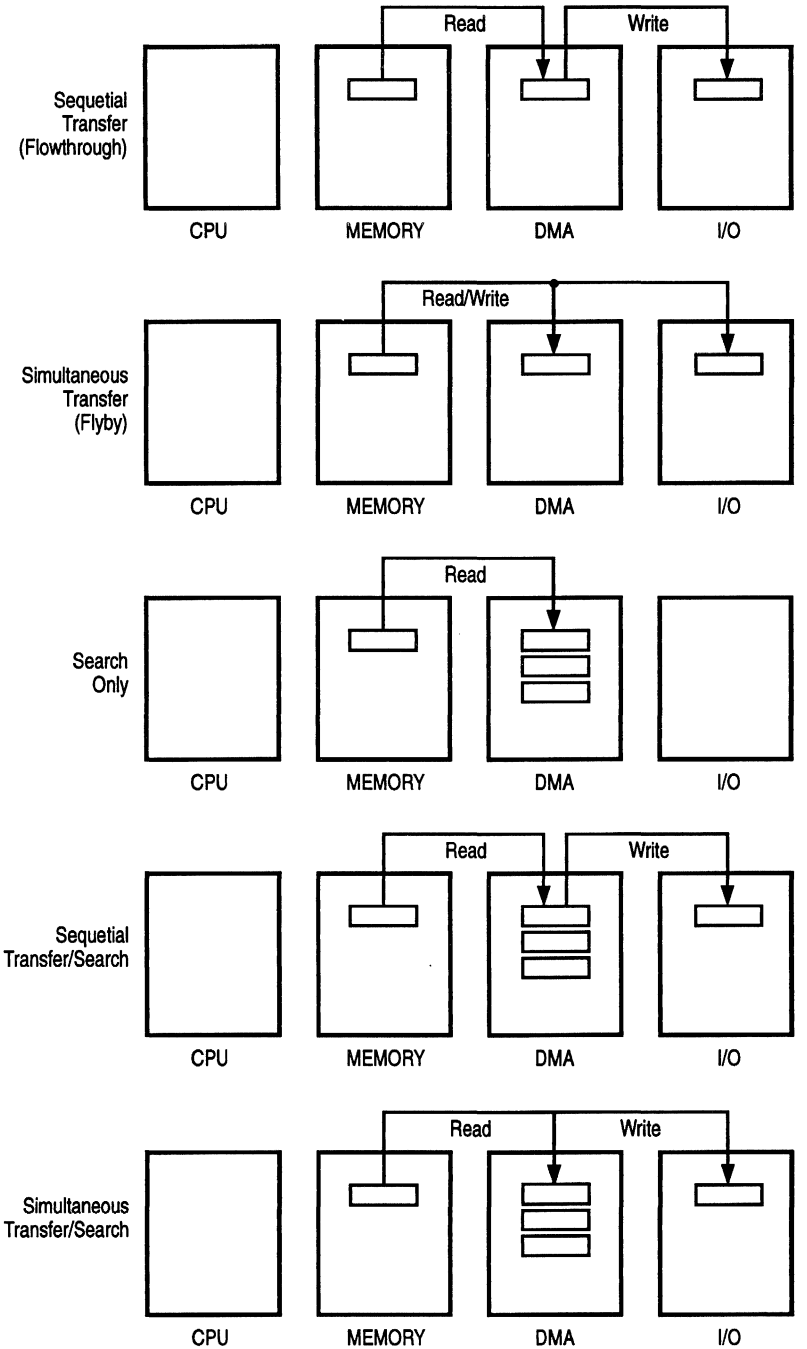


Figure 2-1. Class of Operation

2.3 CLASSES OF OPERATION (Continued)

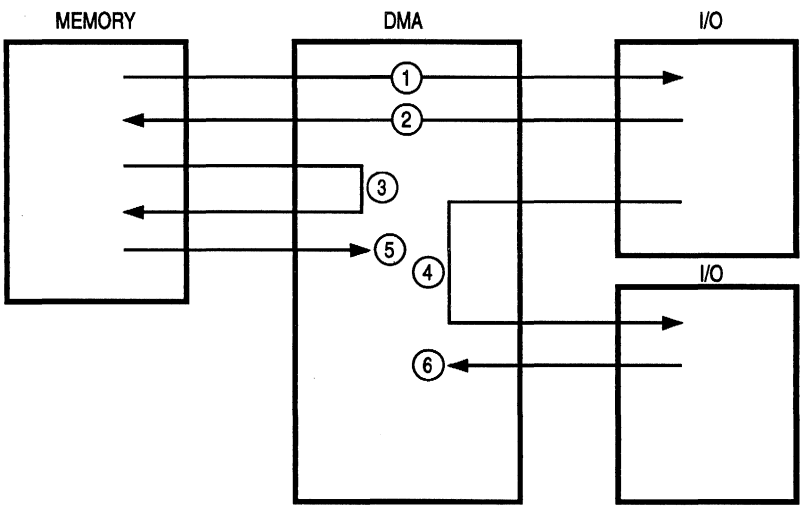
In "sequential transfer/searches," data is transferred in the same way as in the sequential transfer class and simultaneously searched, as in the search-only class. This class also needs no external logic in Z80 and some other environments.

In "simultaneous transfer/searches," data is transferred just as in the simultaneous transfer class and simultaneously searched just as in the search-only class. In this case, at least one additional external logic package is needed to obtain the doubling of speed.

Figure 2-2 summarizes the functions of each class of Z80 DMA operation with respect to the two types of addressable ports that can be selected as a source and destination. The most common applications of DMA are transfers from memory-to-I/O or from I/O-to-memory, without search, and most DMA devices are limited to these operations. The simultaneous searching function is unique to the Z80 DMA. Simultaneous transfers (flyby) are limited to transfers between memory and I/O.

Transfers from memory-to-memory are useful in relocating memory contents. They can also be used in supporting memory-mapped I/O. A Ready condition can be programmed into the DMA to simulate an active I/O Ready line for memory-to-memory transfers (see the "Programming" section).

Transfers from I/O-to-I/O can be used in applications such as real-time data acquisition where backup storage of the incoming data is required. The optional search capability allows branching to various other actions when a byte match is found as in memory-to-memory transfers. Memory searches and I/O searches, without transfer, are unique to the Z80 DMA and are useful in locating an end-of-block, check character, or other special byte in a block.



1. Transfer Memory-to-I/O (optional search).
2. Transfer I/O-to-Memory (optional search).
3. Transfer Memory-to-Memory (optional search).
4. Transfer I/O-to-I/O (optional search).
5. Search Memory.
6. Search I/O.

Figure 2-2. Basic Functions of the Z80 DMA

2.4 MODES OF OPERATION

Within any class of operation, the Z80 DMA can be programmed to operate in one of three Transfer and/or, Search modes:

Byte Mode. Data operations are performed one byte at a time. Between each byte operation the system bus is released to the CPU. The bus is requested again for each succeeding byte operation. This is also sometimes called the "Single" or "Byte-At-A-Time" mode.

Burst Mode. Data operations continue until a port's Ready line to the DMA goes inactive. The DMA then stops (releases the system bus) after completing its current byte operation. This is also called the "Demand" mode.

Continuous Mode. Data operations continue until the end of the programmed block of data or a stop-on-match condition is reached before the system bus is released. If a port's Ready line goes inactive before this occurs, the DMA simply pauses until the Ready line comes active again. This is also called the "Block" mode.

In all modes, the operation on the byte will be completed in an orderly fashion once a byte of data is read into the DMA, regardless of the state of other signals (including a port's Ready line). Figure 2-3 illustrates the sequence of events that occur in a sequential transfer/search of one byte, irrespective of the mode of operation. First, the source port address is incremented or decremented if it was programmed to he a variable-address port. Then the byte is read from that port into the DMA. Next, the destination port address is incremented or decremented if it was programmed to be a variable. The byte is then written to the destination port. If the search capability is included, the byte is compared to the match byte. When there is no byte match, the DMA increments the byte counter and continues; when a byte is found, a status bit is set and the DMA either continues by incrementing the byte counter, stops (releases the bus), or interrupts the CPU, depending on its initial programming. The next three figures illustrate the manner in which the three modes of operation behave before, during, and after the single-byte operation shown in Figure 2-3.

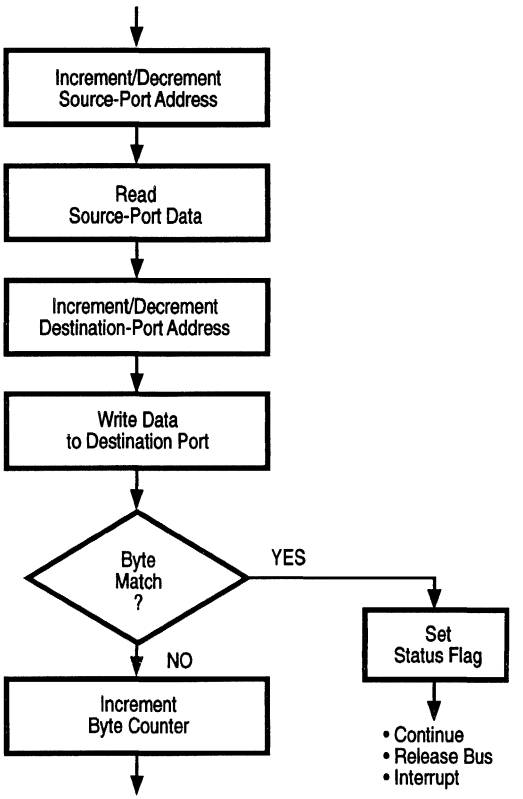


Figure 2-3. Transfer/Search One Byte

Operation in the Byte mode (Figure 2-4) begins with an enabling command from the CPU and a test of the I/O device's Ready line. When the Ready line is active, the DMA requests the system bus (address, data, and control buses) via the Bus Request line, and the CPU acknowledges and releases control to the DMA very shortly thereafter. The transfer of and/or search of one byte then takes place as in Figure 2-3. After this, a test is made for end-of-block by checking to see if the byte counter has reached the programmed block length. If the end is not reached, the DMA releases the bus back to the CPU; if the end is reached, a status bit is set and some terminating action occurs, according to the initial programming. Releasing the bus between each byte allows the CPU to execute at least one machine cycle before releasing the bus again to the DMA for the next byte transfer. This means that while the DMA operates more slowly than it could in other modes, CPU activities like interrupt acknowledgement, polling, and memory refresh can be interleaved with DMA transfers in the Byte mode.

2.4 MODES OF OPERATION (Continued)

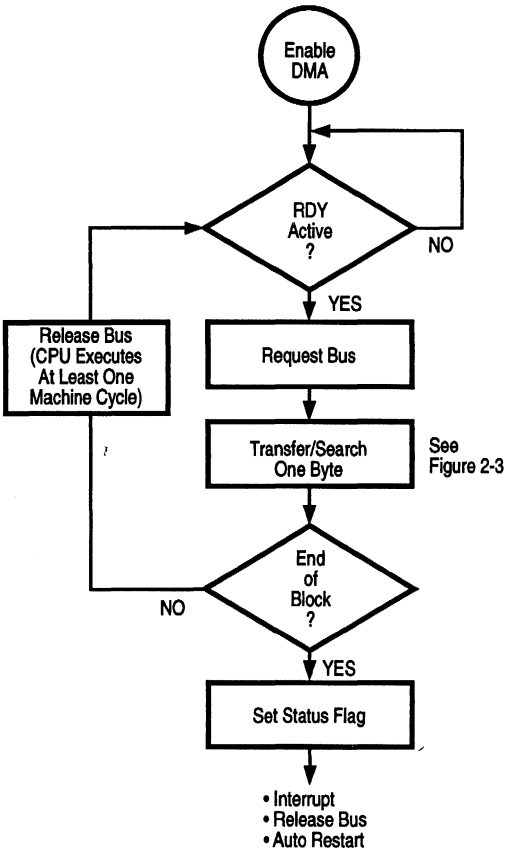


Figure 2-4. Byte Mode

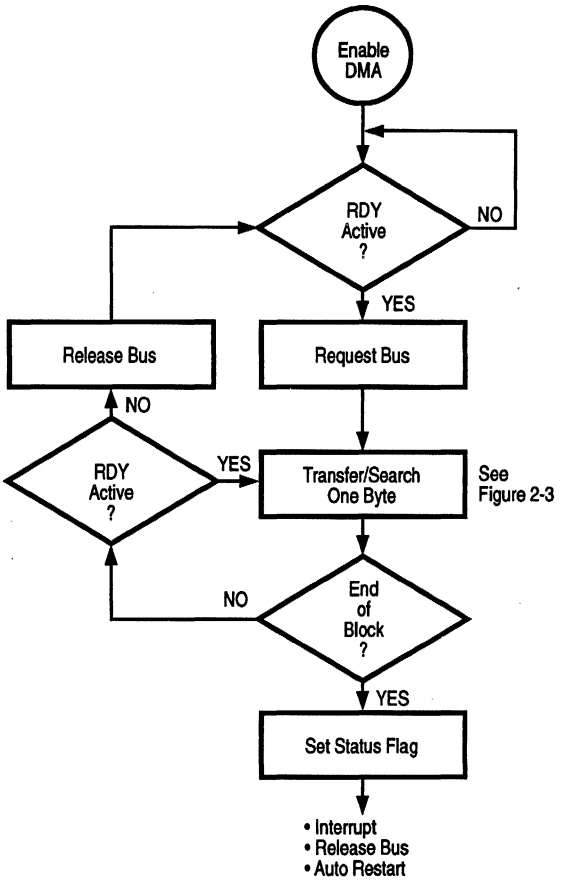


Figure 2-5. Burst Mode

In the Burst mode (Figure 2-5), the bus is requested in the same manner as previously, but once the DMA has control of the bus it continues to transfer bytes until it encounters either an inactive Ready signal from an I/O port, an end-of-block, or a byte match as in Figure 2-3. If the Ready line goes inactive before end-of-block is reached, the DMA releases the bus to the CPU and repetitively tests the Ready signal until it comes active again. Then it requests the bus again and continues its transfers. Because of this, the Burst mode is often the most useful one for general-purpose applications because it does not request the bus until it actually can use it, but once it has the bus, the transfers are made at maximum speed. If the transfers are long, however, this mode can interfere with other CPU activities, which come to a halt for the entire duration of DMA transfers.

In the Continuous mode (Figure 2-6), the DMA requests the bus in the same manner as other modes and repetitively transfers bytes in the same manner as Burst mode. However, unlike the Burst mode the bus is retained by the DMA whenever an inactive Ready signal is encountered prior to a stop on end-of-block or byte match. The DMA simply idles, while holding onto the bus, until Ready becomes active again. Then it completes the transfer sequence. This is the fastest of the three modes because it eliminates the necessity of releasing the bus and requesting it again between complete block transfers. In this mode, however, the system bus is continuously preempted by the DMA. This mode is usually used only when very fast transfers are needed and when the impact on CPU activities can be comfortably tolerated. This might be the case, for instance, when an operating system is being loaded into memory from disk.

Due to the DMAs high-speed buffered method of reading data, operations on one byte are not completed until the next byte is read in. This means that total transfer or search block lengths must be two or more bytes, even in the Byte mode, and that block lengths programmed into the DMA must be one less than the desired block length. This phenomenon is described in detail in the "Internal Structure" chapter under the section entitled, "Address and Byte Counting."

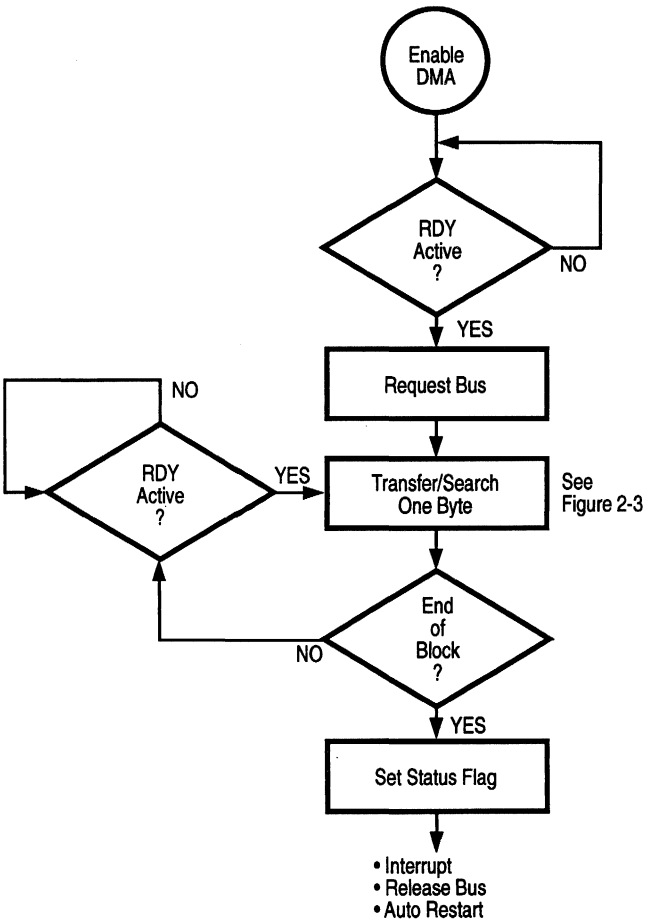


Figure 2-6. Continuous Mode

2.5 TRANSFER SPEED

The Z80 DMA has the fastest maximum transfer rate of any 8-bit DMAC device. This rate is achieved in the simultaneous transfer class of operation and, unlike the more common sequential transfer class, it requires at least one external TTL package. But because all other 8-bit DMAs require some external logic, this constitutes a legitimate speed comparison.

Table 2-1 illustrates the maximum rates that can be achieved in different classes of DMA operation. Maximum CPU block-transfer rates are also given for comparison. All DMA transfers assume the uninterrupted use of Burst or Continuous mode, and they assume read and write cycles that last two cycles (see the following "Variable Cycle," section).

Transfer speed in Byte mode depends on how long the CPU keeps the bus between each byte transfer of the DMA. Therefore, the speed is best expressed from the CPU viewpoint.

Table 2-2 shows the reduction in Z80 throughput (per Kilobaud transferred) caused by byte-mode DMA transfers, and this rate is compared with the reduction in throughput that would occur if the CPU did its own byte transfers using an interrupt service routine of six instructions (a practical lower limit). The section entitled "DMA and Z80 SIO" in the chapter on "Applications" contains more detail on this data. Note that this data assumes sequential DMA transfers with longer cycle timing than the minimum of two clock cycles per read or write. Simultaneous transfers of two clock cycles would, therefore, result in even lower impact on CPU throughput.

Table 2-2 shows that the reduction of CPU throughput with Byte mode DMA transfers is about five times less than the reduction that results when the CPU handles its own byte-mode I/O in the normal Interrupt mode. Table 2-1 shows that DMA transfer rates in Burst and Continuous modes can be up to ten times faster than Z80 CPU rates.

Table 2-1. Maximum Transfer and Search Speeds (Burst and Continuous Modes)

	Z80 (2.4 MHz)	Z80Z (4.0 MHz)
DMA Simultaneous Transfer DMA Search Only DMA Simultaneous Transfer/Search	1.25 Mbytes/s	2.0 Mbytes/s
DMA Sequential Transfer DMA Sequential Transfer/Search	0.625 Mbytes/s	1.0 Mbytes/s
CPU Block Transfer Instruction	0.125 Mbytes/s	0.200 Mbytes/s

Table 2-2. Reduction in Z80 CPU Throughput per Kilobaud for Byte Mode Transfers

	Z80 (2.4 MHz)	Z80Z (4.0 MHz)
DMA Sequential Transfer DMA Sequential Transfer/Search	0.065%	0.041%
CPU Interrupt Driven Transfer	0.340%	0.213%

2.6 ADDRESS GENERATION

Two 26-bit addresses are generated by the DMA for every transfer operation: one address for the source port and another for the destination port. The two addresses are multiplexed onto the address bus, according to whether the DMA is reading the source or writing to the destination.

The two ports are arbitrarily named Port A and Port B. Both A and B can be either source or destination, either memory or I/O, and have fixed or variable addresses.

Variable addresses can either increment or decrement automatically from the programmed starting address. Fixed addresses are useful for I/O devices and the DMAs capability to generate fixed addresses eliminates the need for transfer/search enabling wires to the I/O device (although Chip Enable hardwiring is still required, as it is with all peripheral circuits).

Two readable address counters keep the current address of each port. These counters are distinct from the starting-address registers for each port, that is, the counters are buffered by the registers. Thus new starting addresses can be written to the DMA whenever the DMA is not holding the bus (for example, between byte transfers in Byte mode). Therefore, new starting addresses for a new block of data can be loaded into the DMA before the transfer of the current block is finished. Loading new starting addresses does not disturb the contents of the associated port address counters.

The following partially summarizes the ways DMA address-generation capabilities can be used.

- Start at a base address and count up or down.
- Automatically step back to the beginning at the completion of an address sequence.
- Load in new starting addresses or reload the previous ones for the next sequence.

C

2.7 BYTE MATCHING (SEARCHING)

Searches for byte matches can be performed either as a sole function or simultaneously with transfers. When a byte match is found, a status bit in the readable status register is set and the DMA can be programmed to do one of the following:

- Stop (release the bus) immediately upon byte match.
- Stop and interrupt the CPU immediately upon byte match.
- Interrupt the CPU when the DMA stops at the end of a block.

The match byte written into the DMA is masked with another byte so that only certain bits within the match byte can be compared with the corresponding bits in the data bytes being searched.

2.8 INTERRUPTS

The DMA can be programmed to interrupt the CPU on three conditions:

- Interrupt on Ready
- Interrupt on Byte Match
- Interrupt on End-of-Block

The first condition (I/O-port Ready line becoming active) causes an interrupt before the DMA requests the bus. The other two conditions cause the DMA to interrupt the CPU after the DMA stops (releases the bus). Stopping the DMA on byte match or end-of-block is separately programmed.

Any of these conditions (Ready line becoming active, byte match, or end-of-block) causes a readable status bit to be set. In addition, when an interrupt on any of these conditions is programmed, an interrupt-pending status bit is also set, and each type of interrupt can optionally alter the DMAs interrupt vector.

The DMA shares the Z80 Family's versatile interrupt scheme, which provides fast interrupt service in real-time applications. In a Z80 CPU environment where the CPU is using its Mode 2 interrupts, the DMA passes its internally-modifiable 8-bit interrupt vector to the CPU, which attaches an additional eight bits to form the memory address of the interrupt routine table. This table contains the address of the beginning of the interrupt routine itself. In this process, CPU control is transferred to the interrupt routine, so that the next instruction executed after an interrupt acknowledge is the first instruction of the interrupt routine itself.

2.9 AUTO RESTART

Block transfers can be repeated automatically by the DMA. This function causes the byte counter to be cleared and the address counters to be reloaded with the contents of the starting-address registers.

The Auto Restart feature relieves the CPU of software overhead for repetitive operations such as CRT refresh and many others. Moreover, the CPU can write different starting addresses into the buffer registers during transfers in the Byte mode (or Burst mode when the Ready line is inactive and the bus is released) causing the Auto Restart to begin at a new location.

2.10 PULSE GENERATION

External devices can keep track of how many bytes have been transferred by using the DMAs Pulse output, which provides a signal at 256-byte intervals. The interval sequence may be offset at the beginning by 1 to 255 bytes.

The Interrupt line carries the Pulse signal in a manner that prevents interpretation by the Z80 CPU as an interrupt request, since the signal only appears when the Bus Request and Bus Acknowledge lines are both active. Under these conditions, the Z80 CPU does not monitor the Interrupt (/INT) line.

2.11 VARIABLE CYCLE

The Z80 DMA offers the unique feature of programmable operation-cycle length. This is valuable in tailoring the DMA to the particular requirements of various CPUs and other system components (fast or slow), and in maximizing the data-transfer rate. Also, it often eliminates external logic and reduces CPU software overhead.

There are two aspects to the variable cycle feature. First, the entire read and write cycles (periods) associated with the source and destination ports can be independently programmed as 2, 3, or 4 clock cycles long (more if Wait cycles are used), thereby increasing or decreasing the speed with which all DMA signals change.

Second, the four signals in each port (I/O Request, Memory Request, Read, and Write) can each have its active trailing edge terminated one-half clock cycle early. This adds a further dimension of flexibility, allowing such things as shorter-than-normal Read or Write signals that go inactive before data starts to change. Figure 2-7 illustrates the general capability, which is described later in the "Timing" chapter.

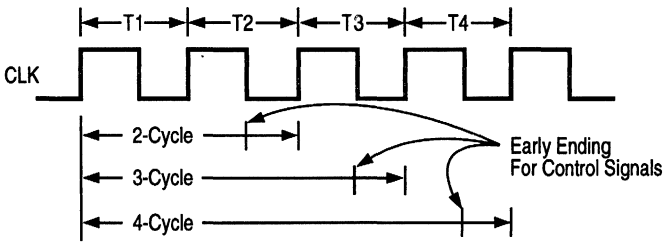


Figure 2-7. Variable Cycle Length

C

2.12 TARGETS AND ACTIONS

Table 2-3 gives an overview of the targets that can cause specific actions by the DMA, depending on how it is programmed. The targets are conditions in the DMAs internal registers, signals from the I/O device, or instructions on the data bus that DMA watches for.

Table 2-3. Events and Actions

Event	Actions Possible When Event Occurs
End-of-Block	1. Release Bus 2. Interrupt CPU 3. Auto Restart
Byte Match (Compare)	1. Release Bus 2. Interrupt CPU 3. Continue
Pulse-control Byte Matches Lower Part of Byte Counter	1. Generate Pulse
READY Inactive	1. Release Bus 2. Suspend (Continuous Mode Only)
READY Active	1. Request Bus 2. Interrupt CPU
RETI Instruction (Return from Interrupt Instruction from the Z80 CPU)	1. Request Bus

CHAPTER 3
PIN DESCRIPTION

3.0 PIN DESCRIPTION

The following pin descriptions detail the function of the Z80 DMA external pins as illustrated in Figures 3-1 through 3-4.

A15-A0 *System Address Bus* (output, tri-state). Addresses generated by the DMA are sent to both source and destination ports, either of which may be main memory or I/O peripherals.

/BAI *Bus Acknowledge In* (input, active Low). Signals that the system buses have been released for DMA control.

/BAO *Bus Acknowledge Out* (output, active Low). In multiple-DMA configurations, this pin signals that the CPU has relinquished control of the bus. /BAI and /BAO from a daisy chain for multiple-DMA priority resolution over bus control. Unlike the interrupt daisy chain formed with the IEI and IEO lines, this chain does not allow preemption of control by a high-priority DMA when a lower-priority DMA is already bus master. The DMA that has the bus is always allowed to finish, regardless of its priority in the chain.

/BUSREQ *Bus Request* (bidirectional, active Low, open-drain). As an output, it sends requests for control of the system address bus, data bus, and control bus to the CPU. As an input when multiple DMAs are strung together in a priority daisy chain through /BAI and /BAO, it senses when another DMA has requested the buses and causes this DMA to refrain from bus requesting until another DMA is finished. Because it is a bidirectional pin that allows simultaneous bidirectional signals with no means of control, there cannot be any buffers between this DMA and any other DMA. It can, however, have a buffer between it and the CPU because it is unidirectional into the CPU. A 1.8 kohms pullup resistor is typically connected to this pin.

/CE/WAIT *Chip Enable and Wait* (input, active Low). Normally, this functions only as a /CE line, but it can also be programmed to serve as a /WAIT function. As a /CE line from the CPU, it becomes active when /IORQ is active and the I/O port address (up to 16 bits) on the system address bus is the DMA's address, thereby allowing control bytes to be written from the CPU to the DMA. As a /WAIT line from memory or I/O devices, after the DMA has received a bus acknowledge (/BUSACK) from the CPU, it causes wait states to be inserted in the DMA's operation cycles, thereby slowing the DMA to a speed that matches the memory or I/O device. The "Applications" chapter contains a description of how the /CE and /WAIT inputs can be multiplexed by the CPU's /BUSACK line.

CLK *System Clock* (input). Standard Z80 single-phase clock at 2.5 MHz (Z80 DMA) or 4.0 MHz (Z80A DMA). For slower system clocks a TTL gate with a pullup resistor may be adequate to meet the timing and voltage level specifications. For higher speed systems, use a clock driver with an active pullup to meet the VIH specification and rise time requirements. There should be a resistive pullup to the power supply of 10 kohms (maximum) in all cases to ensure proper power when the DMA is reset.

D7-D0 *System Data Bus* (bidirectional, tri-state). These pins transfer control bytes from the CPU, status bytes from the DMA, and data from memory or I/O peripherals. Data transfers or searches by the DMA are done only when the DMA controls this bus and the address bus. When the CPU controls these buses, it can write or read DMA control or status bytes.

3.0 PIN DESCRIPTIONS (Continued)

IEI *Interrupt Enable In* (input, active, High). This line is used with IEO to form a priority daisy chain when there is more than one interrupting device. A High on this line indicates that no other device of higher priority is interrupting, thereby allowing this DMA to interrupt if it is otherwise enabled to.

IEO *Interrupt Enable Out* (output, active High). IEO is High only if IEI is High and this DMA is not requesting an interrupt. Thus, this signal blocks lower priority devices from interrupting while a higher priority device is being serviced by its CPU interrupt service routine. Unlike devices in a bus-request daisy chain, devices in an interrupt daisy chain can be preempted by higher priority devices before the lower priority device has been fully serviced.

/INT//PULSE *Interrupt Request* (output, active Low, open-drain). This requests a CPU interrupt when brought Low at a time when the DMA is not the bus master. The CPU acknowledges the interrupt by pulling its /IORQ output Low during an /M1 cycle. The DMA /INT pin is typically connected to the /INT pin of the CPU with a pullup resistor and tied to all other /INT pins in the system. This pin can also be used to generate periodic pulses to an external device. It can be used this way only when the DMA is bus. master (i.e., the CPU's /BUSREQ and /BUSACK lines are both Low and the CPU cannot see interrupts).

/IORQ *Input/Output Request* (bidirectional, active Low, tri-state). As an input, this indicates that the lower half of the address bus holds a valid I/O port address for transfer of control or status bytes from or to the CPU, respectively; this DMA is the addressed port if its /CE pin, /IORQ pin, and or /RD pin are simultaneously active. As an output, after the DMA has taken control of the system buses, it indicates that the address bus holds a valid 8-bit or 16-bit port address for another I/O device involved in a DMA transfer of data. When /IORQ and /M1 are both active inputs to the DMA simultaneously, an interrupt acknowledge by the CPU is indicated.

/M1 *Machine Cycle One* (input, active Low). This pin indicates that the current CPU machine cycle is an instruction fetch. It has two purposes in the DMA's interrupt structure. First, it is used by the DMA to detect return-from-interrupt instructions (RETI, or hex ED4D) fetched over the data bus by the CPU at the end of interrupt service routines. Second, an interrupt acknowledge is indicated when both /M1 and /IORQ are active inputs to the DMA. During 2-byte instruction fetches, /M1 is active as each opcode byte is fetched. In the CMOS DMA, the /M1 signal has another function: when /M1 occurs without an active /RD or /IORQ for at least two clock cycles, the internal reset is activated at the falling clock after /M1 returns to the inactive state. This internal reset lasts for three clock cycles.

/MREQ *Memory Request* (output, active Low, tri-state). This line indicates that the address bus holds a valid address for a memory read or write operation. After the DMA has taken control of the system buses, it indicates a DMA transfer request from or to memory.

/RD *Read* (bidirectional, active Low, tri-state). As an input, this signal indicates that the CPU wants to read status bytes from the DMAs read registers, if selected. As an output, after the DMA has taken control of the system buses, it indicates a DMA-controlled read from memory or I/O port address.

/RDY *Ready* (input, programmable active Low or High). This pin is monitored by the DMA to determine when a peripheral device associated with a DMA port is ready for a read or write operation. When the DMA is enabled to operate, the /RDY line indirectly controls DMA activity; the manner in which DMA activity is controlled by /RDY varies with the operating mode (Byte, Burst, or Continuous). An active /RDY line can be simulated by programming a Force Ready condition. This is useful in memory-to-memory operations. It is preferable to have the /RDY signal synchronized to the CLK signal, i.e., /RDY should become active on the rising edge of CLK. This is particularly important in the Continuous mode of operation.

/WR *Write* (bidirectional, active Low, tri-state). As an input, this indicates that the CPU wants to write control bytes to the DMA write registers when the DMA is selected. As an output, after the DMA has taken control of the system buses, it indicates a DMA-controlled write to a memory or I/O port address.

/RESET *Reset* (input active low) is only available in the CMOS PLCC version. A Low in this signal Resets the DMA.

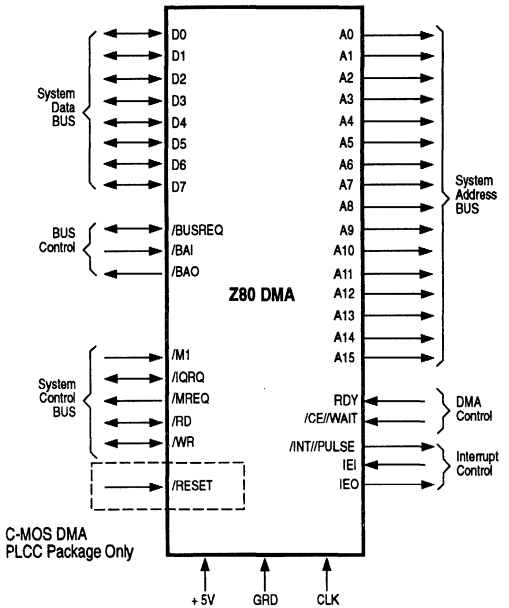


Figure 3-1. Pin Functions

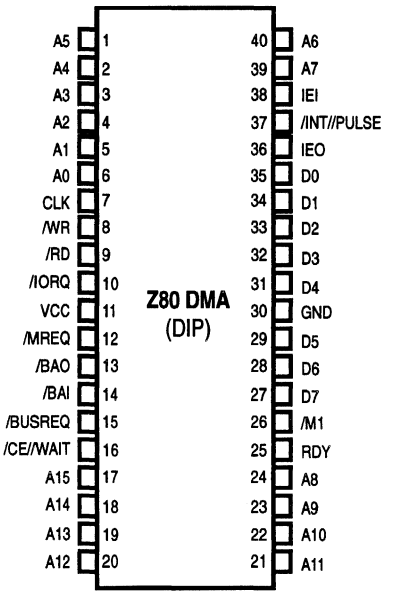


Figure 3-2. 40-Pin DIP Pin Assignments

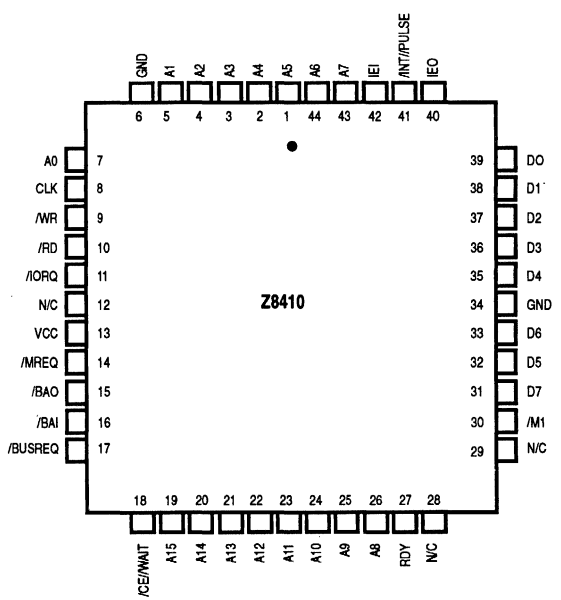


Figure 3-3. 44-Pin PLCC Pin Assignments (Z8410 NMOS)

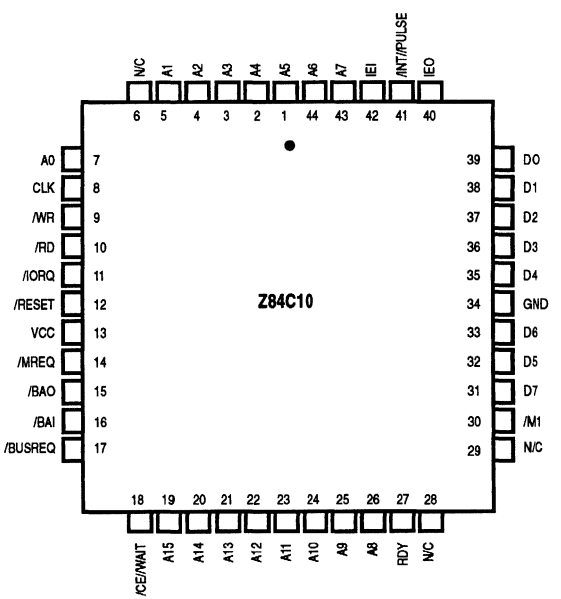


Figure 3-4. 44-Pin PLCC Pin Assignments (Z84C10 CMOS)

CHAPTER 4
INTERNAL STRUCTURE

4.0 GENERAL ORGANIZATION

The internal structure of the Z80 DMA includes driver and receiver circuitry for interfacing with an 8-bit data bus, a 16-bit address bus, and system control lines. In a Z80 CPU environment, the DMA can be tied directly to the analogous pins on the CPU with no additional buffering, except for the /CE//WAIT line, when operation is limited to sequential transfers and searches. The chapter on "Applications" provides an illustration of this.

Figure 4-1 illustrates how the DMA's internal data bus interfaces with the system data bus and services all internal logic and registers. Addresses generated for Ports A and B of the DMA's single transfer channel are multiplexed onto the system address bus.

Specialized logic circuits in the DMA are dedicated to the various functions of external bus interfacing, internal bus control, byte matching, byte counting, periodic pulse generation, CPU interrupts, bus requests, and address generation.

C

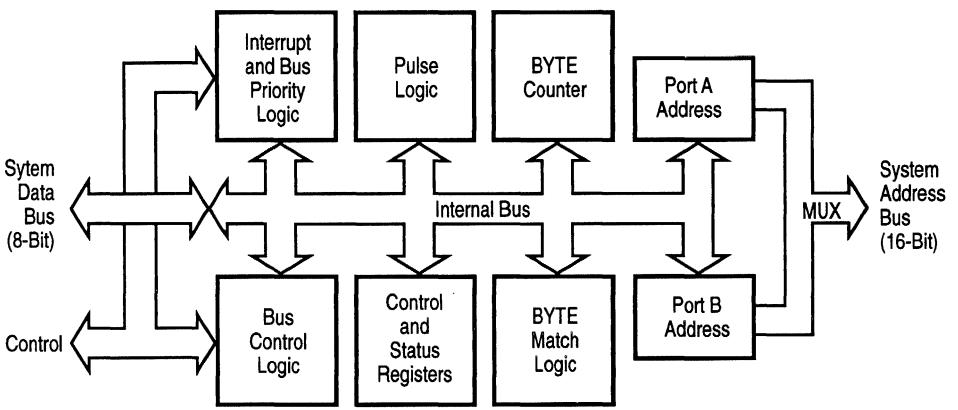


Figure 4-1. Z80 DMA Block Diagram

4.1 CONTROL AND STATUS REGISTERS

A set of 21 writeable control registers and seven readable status registers provide the means by which the CPU governs and monitors the activities of these logic circuits. All registers are eight bits wide (the width of the data bus), and double-byte information is stored in adjacent registers.

The 21 control registers writeable through the data bus are organized into seven base register groups, WR6 through WR0, most of which have multiple registers. The base registers in each group contain both control hits and pointer bits that can be set to address other registers within the group. Writing to a register within a group involves first writing to the base register for that group, with the appropriate pointer bits set, then writing to one or more of the other registers that have been pointed to within the group. The chapter on "Programming" contains a full discussion of this technique. The names of the write registers shown in Figure 4-2 do not indicate the full extent of the DMAs programmability since many modes and functions are set with single bits in the base register bytes of each group.

Figure 4-2 illustrates the write registers. These are the registers for which inputs from the data bus are shown in the figure (compare this figure with Figure 4-3, which identifies the read registers).

The two 16-bit starting address registers in groups WR4 and WR0, and the 16-bit block-length register in group WR0 have 16-bit counters associated with them. (The counters, unlike their associated registers, cannot be written to.) The two address counters generate the addresses that are put onto the address bus. They can also be read by the CPU through the data bus, as can the byte counter. All three writeable registers act as buffers for the readable counters; the contents of the registers can be changed during a block transfer without disturbing the contents of the counters. This facility is useful, for example, during Byte-mode transfers, in which control bytes can be written to the DMA while the CPU has the bus between byte transfers. This allows the next block (which can be an Auto Restart block) to begin quickly at a new location. Note that the block length counter stops (or Auto Restarts) as a result of a comparison to the block length register. In changing the register, the block length also changes with what may be unpredictable results.

The pulse-control byte illustrated in Figure 4-2 (in the WR4 group) also has a relationship to the byte counter in WR0—it can be loaded with an offset value between 0 and 255 and this value will be continuously compared with the lower byte of the byte counter. The /INT line generates a pulse each time a match occurs, which happens on every 256 bytes of transfer or search after the initial offset. Since the pulse signals generated on the /INT line only occur when the DMA has control of the system bus (i.e., when the /BUSREQ and /BUSACK lines are simultaneously active), the CPU does not see them and they can be directed exclusively to an external gate, counter, or other device.

Figure 4-3 illustrates the seven status registers readable through the data bus. Unlike the write registers, the status registers include no second-level registers or groups. These registers are accessed sequentially according to the "read mask" written into the WR6 group, except that the "status byte" can be read separately from the other read registers.

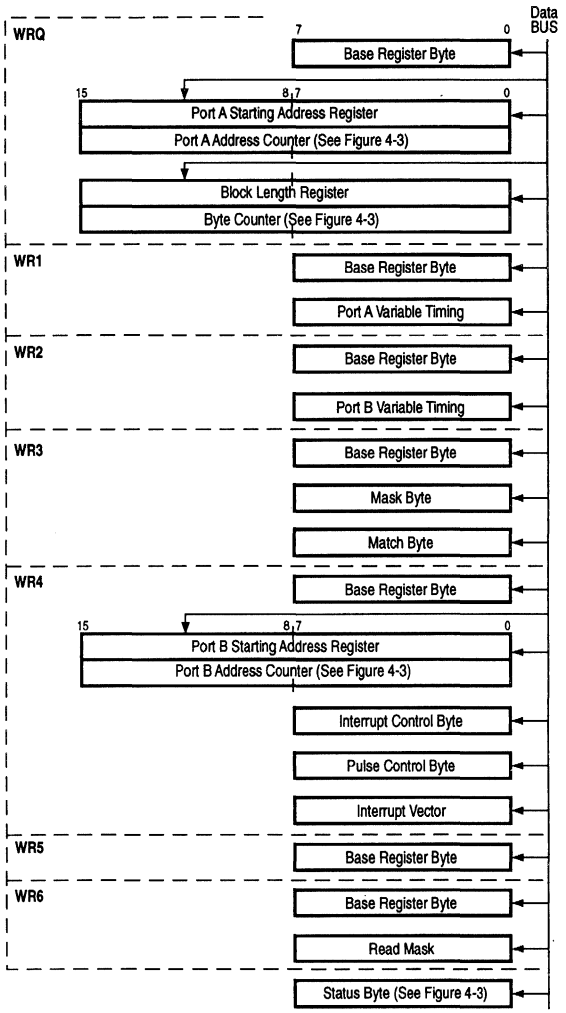


Figure 4-2. Write Register Organization

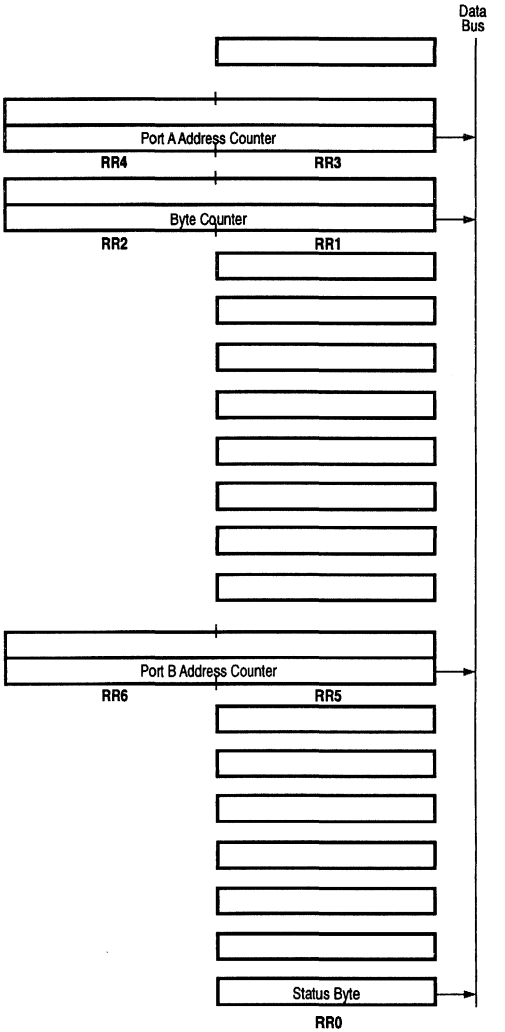


Figure 4-3. Read Register Organization

C

4.2 ADDRESS AND BYTE COUNTING

Addresses for either port may be fixed at their programmed starting address, or they may be incremented or decremented from the programmed starting address by the address counters. The block length programmed into the DMA is compared with the byte counter, which starts at zero and increments at the completion of each byte operation (Figure 2-3).

The DMA uses a high-speed buffering or pipelining scheme for reading data in. In transferring data and stopping on an end-of-block, the consequence of this pipelining is that one more transfer will be completed than is programmed into the block-length register; the only exception to this rule occurs in simultaneous transfers which use two-cycle variable timing, in which case two extra bytes are transferred if the Ready line remains active.

Table 4-1 shows the contents of the counters in the various classes and the modes of transfer involving stopping or interrupting at an end-of-block (interrupts imply prior stopping).

Search and transfer/search operations that are programmed to stop on byte match behave somewhat differently, as illustrated in Table 4-2. Matches are discovered only after the next byte is read in. In all classes of transfer/search operations, the matched byte is transferred. In simultaneous transfer/search operations, however, an additional byte is usually also transferred. The only exception to this occurs in Burst and Continuous modes when the Ready line goes inactive while the byte match is being located. With respect to simultaneous transfer/searches in Burst or Continuous mode, this special case is usually not a problem since searches are typically continuous processes performed in memory using a Force Ready condition or a Ready line that will not go inactive. However, if it is possible this situation may be encountered, the CPU can be programmed to research two bytes when such a match occurs.

Table 4-1. Contents of Counters After DMA Stops Due to End-of-Block (Transfer Operations)

Class	Mode	Programmed Block Length	Bytes Transferred At Stop	Byte Counter	Source-Port Address Counter*	Destination Port Address Counter*
Sequential Transfer	Byte	N	N + 1	N	As ± (N + 1)	As ± (N)
	Burst	N	N + 1	N	As ± (N + 1)	As ± (N)
	Continuous	N	N + 1	N	As ± (N + 1)	As ± (N)
Search Only or Simultaneous Transfer/Search	Byte	N	N + 1	N	As ± (N + 1)	***
	Burst	N	N + 1	N + 1	As ± (N + 1)	***
	Burst	N	N + 2**	N + 1**	As ± (N + 2)**	***
	Continuous	N + 1	N + 1	N + 1	As ± (N + 1)	***
	Continuous	N + 1	N + 2**	N + 1**	As ± (N + 2)**	***

Notes:

* Address can increment (+) or decrement (–) from the programmed starting address (As), which is the first address for transfer purposes.

** Occurs only in 2-cycle (variable timing) simultaneous transfers when the Ready line is still active at the end of the N + 1 byte transfer.

*** Simultaneous transfers cannot have both ports variable. This class of operation is programmed as a DMA search-only operation, with variable addresses ascribed to the programmed "source" port. In fact, wha the DMA believes is the "source" port may be either the real source or destination, as determined by external hardware. See the "Applications" chapter.

Table 4-2. Contents of Counters After DMA Stops Due to Byte Match (Search or Transfer/Search Operations)

Class	Mode	Match Occurs On This Byte	Bytes Transferred At Stop If Transferring	Byte Counter	Source-Port Address Counter*	Destination Port Address Counter*
Sequential Transfer	Byte	M	M	M – 1	As ± (M)	As ± (M – 1)
	Burst	M	M	M – 1	As ± (M)	As ± (M –)
	Continuous	M	M +	M – 1	As ± (M)	As ± (M –)
Search Only or Simultaneous Transfer/Search	Byte	M	M	M	As ± (M)	***
	Burst	M	M + 1	M + 1	As ± (M + 1)	***
	Burst	M	M**	M – 1**	As ± (M)**	***
	Continuous	M	M + 1	M + 1	As ± (M + 1)	***
	Continuous	M	M**	M – 1**	As ± (M)**	***

Motes:

* Address can increment (+) or decrement (–) from the programmed starting address (As), which is the first address for transfer or search.

** Occurs only only when the Ready line is still goes inactive just prior to the beginning of the last possible cycle in the operation (i.e., Ready iis sampled inactive on the rising edge of CLK in the last cycle of the last read operation).

*** Search only has no destination. Simultaneous transfer/search cannot have both ports variable. This class of operation is programmed as a DMA search-only operation, with variable addresses ascribed to the programmed "source" port. In fact, wha the DMA believes is the "source" port may be either the real source or destination, as determined by external hardware. See the "Applications" chapter.

C

4.3 BUS CONTROL

The DMA transfers and searches data by controlling the system buses in exactly the same way that the Z80 CPU controls them to do read and write cycles. Specifically, the DMA controls the following lines:

- Address Bus (16 bits)
- Data Bus (8 bits)
- /IORQ
- /MREQ
- /RD
- /WR

In addition, the DMA can also be programmed to watch a /WAIT line through its dual-purpose /CE//WAIT pin.

When the DMA has requested and received the bus from the CPU, other devices on the system do not perceive the change. The CPU is completely idle during this time because it cannot fetch instructions from memory.

4.3.1 Bus Requesting

Two conditions enable the DMA to request the bus from the CPU: (1) an enabling command from the CPU, and (2) an active Ready condition, which can be caused either by an active Ready line from an I/O device or a Force Ready command by the CPU.

The DMA requests the bus by latching its /BUSREQ line Low. The CPU will always respond to a bus request and it does so quickly, in no more than one machine cycle (3 to 10 clock cycles) plus one additional clock cycle—by lowering its /BUSACK line as an input to the DMA's /BAI line. Both the DMA's /BUSREQ output and the CPU's /BUSACK output will remain Low while the DMA has the bus.

The bus is released back to the CPU when the DMA's /BUSREQ line goes High; the CPU's /BUSACK line goes High in the next clock cycle. The DMA releases its /BUSREQ line under a variety of conditions, including:

- Completion of single-byte transfer (Byte mode)
- Ready line going inactive (Byte and Burst modes)
- Byte match (Burst and Continuous modes), if stop on match is programmed
- End-of-block (all modes), if stop on end-of-block is programmed

4.3 BUS CONTROL (Continued)

These conditions each have a somewhat different character and they are explained in the "Timing" chapter. Bus requests cannot be made while the CPU services an interrupt from the DMA. This is prevented by the Interrupt Under Service (IUS) latch, discussed later.

4.3.1 Bus-Request Daisy Chains

Multiple DMAs can be linked in a prioritized daisy chain for the purpose of requesting the bus. Figure 4-4 shows how this is done.

Each DMA's /BUSREQ pin is bidirectional. As an output, it requests the bus. As an input, this pin senses when another DMA in the daisy chain has requested the bus (brought the /BUSREQ line Low) and thus prevents this DMA from also requesting the bus until the other DMA has finished. Any DMA which has the bus is always allowed to finish its operation; a higher priority DMA cannot preempt it during this time.

Their proximity to the CPU determines the priority of DMAs in a daisy chain. The DMA electrically closest to the CPU (as measured along the /BUSACK//BAI lines) has the highest priority. Priority matters only when multiple DMAs request the bus on the same clock cycle. The higher priority DMA can then prevent lower priority DMAs from receiving a bus-acknowledge signal through the /BAI/ /BAO chain. The lower priority DMAs continue to hold their /BUSREQ lines Low until the higher priority DMA finishes and releases the bus, thereby allowing lower priority DMAs to contend immediately for the bus.

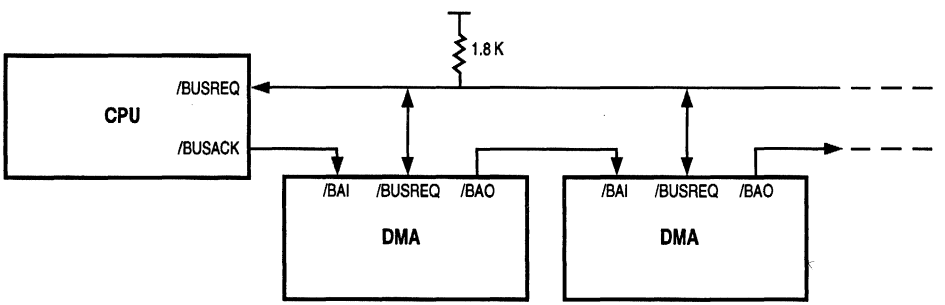


Figure 4-4. Bus-Requesting Daisy Chain

4.4 INTERRUPTS

4.4.1 Conditions and Methods

The Z80 CPU honors external events according to the following priority:

- Bus Requests (/BUSREQ)
- Non-Maskable Interrupts (NMI)
- Maskable Interrupts (/INT)

In addition to bus requests, the DMA normally uses only maskable interrupts (/INT) and it uses them in the CPU's Mode 2, which allows interrupt vectors. Non-maskable interrupts are typically reserved for extreme priority events such as power-failure signaling. A full description of this is given in Zilog Application Note 03-0041-01, The Z80 Family Program Interrupt Structure.

The DMA can be programmed to interrupt the CPU under the following conditions:

- After the DMA's RDY line has gone active and before the DMA requests the bus (interrupt on RDY).
- On an end-of-block, when the contents of the byte counter match the contents of the block-length register.
- On a byte match, when the contents of the match-byte register—after masking by the mask-byte register—corresponds to a data byte being transferred or searched.

The DMA cannot have control of the bus when it interrupts the CPU because signaling on the /INT line while the DMA is bus master is used to generate periodic pulses to an external device and is not perceived by the Z80 CPU. Therefore, after a stop on end-of-block or byte match, the DMA first releases the bus before interrupting the CPU, as shown in Figure 4-5.

If the DMA is programmed to interrupt on end-of-block and also to Auto Restart on end-of-block, an interrupt will occur (and should be acknowledged for continued operation) at each end-of-block. However, the end-of-block status bit will not be set as it would be without the Auto Restart. Therefore, the interrupt vector cannot reflect the specific interrupt cause (i.e., Status Affects Vector is not effective).

The Z80 CPU acknowledges the interrupt by pulling its /M1 and /IORQ lines Low for one machine cycle (see the "Timing" chapter). This causes the DMA to put its 8-bit interrupt vector on the data bus, thereby identifying itself and optionally identifying the origin of the interrupt. The CPU uses the vector to access an interrupt service routine, which is then executed. The interrupt service routine typically reenables the DMA to request the bus and cause interrupts again.

For CPUs that have no interrupt acknowledge or a non-compatible one, DMA control bytes can be written (usually in the interrupt service routine) to simulate the same functions.

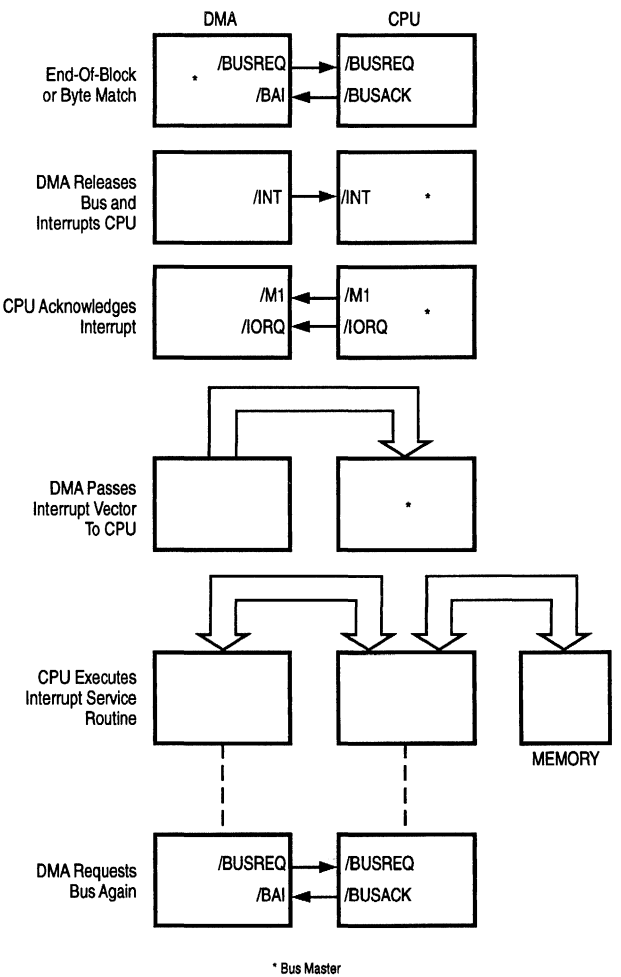


Figure 4-5. Z80 Interrupt Sequence

4.4 INTERRUPTS (Continued)

4.4.2 Interrupt Vectors

The Z80 CPU interrupt acknowledge cycle causes the DMA to put its 8-bit interrupt vector on the data bus (Figure 4-6a). This vector is read by the CPU into a temporary register. It normally identifies the interrupting device and it can also identify the cause of the interrupt (actually the current state of certain status bits). The I Register of the Z80 CPU (when the CPU is programmed to its Mode 2 state) has the upper byte of a 16-bit address which is formed with the interrupt vector, and this address points to a jump table entry in memory.

The jump table location in memory contains an address that is read into the CPU's program counter (Figure 4-6b). This address points to the first instruction of the interrupt service routine, which then begins executing. In most DMA applications, the CPU's interrupt service routine contains instructions which write control bytes back into the DMA through a register in the CPU (Figure 4-6c).

In CPU environments without interrupt vectors, the CPU must poll each peripheral or an external register to determine which device interrupted and why.

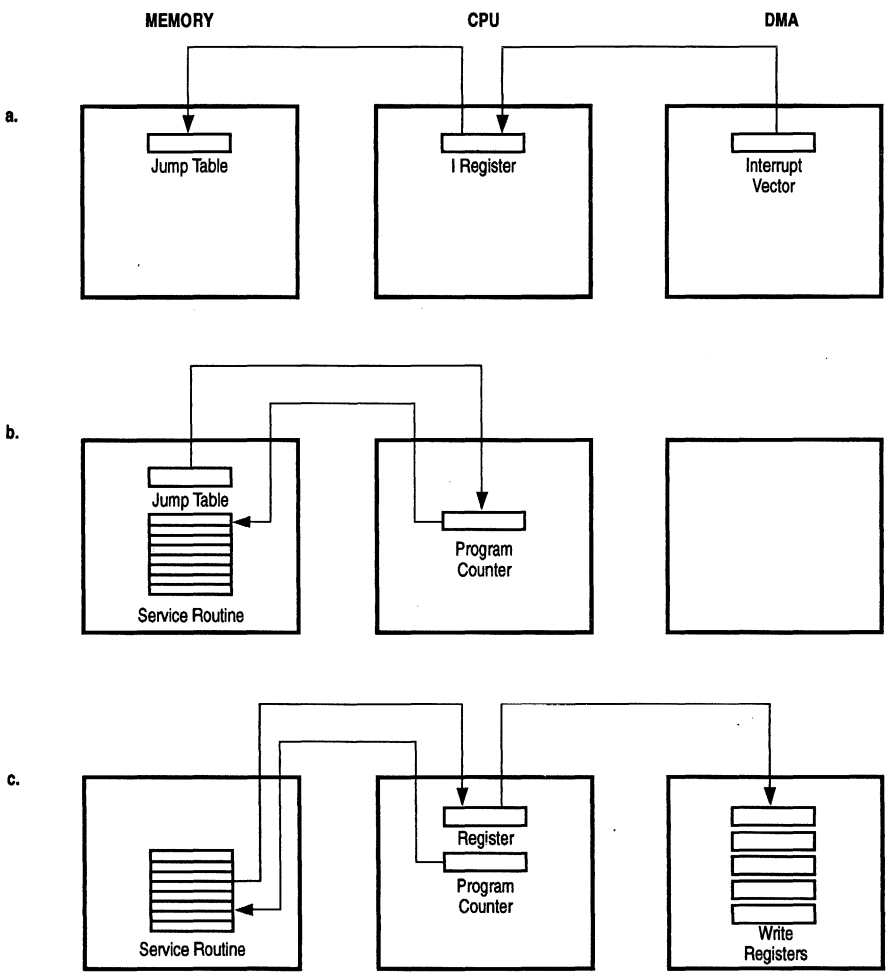


Figure 4-6. Z80 Interrupt Service Routine

4.4.3 Interrupt Latches

Two primary latches are associated with the interrupt structure:

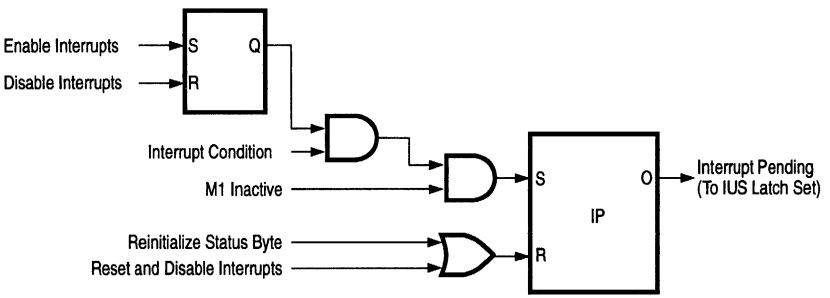
Interrupt Pending (IP). Set whenever the DMA requests an interrupt but has not yet been acknowledged. It holds the INT line Low (Figure 4-7).

Interrupt Under Service (IUS). Set when the CPU acknowledges the DMA interrupt (Figure 4-8). This does three things:

- Prevents further interrupts by this DMA
- Prevents interrupts from lower priority devices in an interrupt daisy chain
- Prevents further bus requests by this DMA

If the Interrupt on RDY (interrupt before requesting bus) option is selected, the IP latch is set when the Ready line becomes active, causing /INT to go Low.

The IP latch is reset whenever the IUS latch is set, but if the interrupt-causing condition is not removed before IUS is reset, IP becomes set again as soon as IUS is reset, therefore, causing another interrupt. The IUS latch can be reset by the Z80 CPU's Return from Interrupt (RETI) instruction or by control bytes written to the DMA.



*NOTE: Interrupt conditions can include end-or-block, byte match or active RDY line, depending on programming.

Figure 4-7. Interrupt Pending (IP) Latch

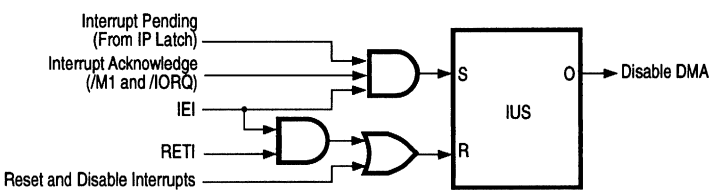


Figure 4-8. Interrupt Under Service (IUS) Latch

4.4 INTERRUPTS (Continued)

4.4.4 Interrupt On Ready

Normally, when the DMA has been enabled by the CPU to request the bus while the I/O device's Ready line is inactive, the Ready line's transition to the active state will cause the /BUSREQ line to go Low (Figure 4-9). It does so within two clock cycles if the setup time to the rising edge of CLK is met.

This does not take place, however, when the Interrupt on Ready option (also called the Interrupt Before Requesting Bus option) is selected. When this option is used, the DMA interrupts the CPU when the Ready line comes active. The CPU's interrupt service routine now writes control bytes to the DMA, which enable the DMA to request the bus after the service routine finishes.

As noted earlier, the CPU cannot respond to an interrupt when the DMA is bus master. Thus, when enabled in Continuous mode, the DMA interrupts the CPU when the Ready line first becomes active, but not on succeeding transitions.

The Interrupt on Ready option is typically used to put new starting addresses into the DMA so that transfers go to a part of memory that is dynamically determined.

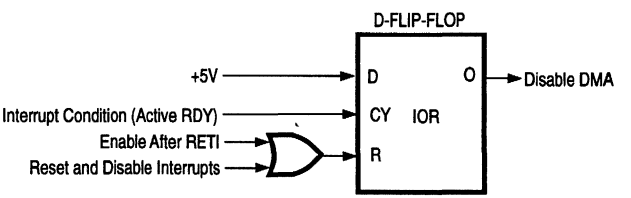
4.4.5 Interrupt Service Routines

In addition to the DMA's extensive programmability for mode-setting (usually done at power-up initialization), there are numerous commands (control bytes) designed for use in various interrupt service routines. The next chapter on "Programming," fully explains the commands, but a quick overview in the present context may be helpful.

Some typical functions for which control bytes are available for use in interrupt service routines include:

- Reset the DMA
  Enable the DMA for bus requesting
  Disable the DMA for bus requesting
- Reset and disable DMA interrupts,
  Enable DMA interrupts
  Disable DMA interrupts
- Load new starting addresses and block length
  Continue prior address counting
  clear block length counter
- Force the Ready condition
- Read the status byte
  Initiate a status-register read sequence
  Clear status

Interrupt service routines on a Z80 CPU always end with a Return From Interrupt (RETI or hex ED4D) instruction, which is now explained.



*NOTE: This latch is only set when the interrupt On Ready option is selected.

Figure 4-9. Interrupt On Ready (IOR) Latch

4.4.6 Return From Interrupt

At the end of an interrupt service routine, the Z80 CPU executes a return-from-interrupt (RETI or hex ED4D) instruction. This returns the CPU from the interrupt service routine.

The DMA also simultaneously decodes the RETI instruction, which it recognizes on the data bus as an instruction (occurring when the DMAs /M1 input is Low). This causes at least one, and possibly two, things to happen within the DMA:

- Resets the Interrupt Under Service (IUS) latch in the DMA, thereby allowing its IEO pin to go High so that lower priority devices can interrupt.
- Enables the DMA to request the bus again. (This occurs only in the Interrupt on Ready option and only when the Enable DMA control byte is also used.)

For non-Z80 environments, control bytes are provided to simulate these actions.

4.4.7 Interrupt Daisy Chains

Multiple DMAs can be chained together by their IEI and IEO lines, as shown in Figure 4-10. In the Z80 Family, the DMAs location in the IEI/IEO chain sets priority.

When peripherals simultaneously interrupt the Z80 CPU, the highest priority peripheral (nearest the +5V end of the daisy chain) is serviced. The CPU learns which peripheral won by receiving its interrupt vector; the IEI/IEO chain allows only the highest priority interrupting peripheral to place its interrupt vector on the data bus. In non-Z80 environments that have no interrupt vectors, the winning peripheral may be determined by successively reading the status of all peripherals.

For a device to have priority, its IEI line must be High. When a device needs service, it prevents downstream devices from interrupting by pulling its IEO line Low. The next device in the chain then passes this Low condition on to other downstream devices by pulling its IEO line Low, and so on.

Whenever an interrupt is acknowledged (Figure 4-5), the CPU's interrupt structure is disabled. It must subsequently be reenabled by an "enable interrupts" instruction before other devices can interrupt again. This normally takes place within the interrupt service routine. When done early in the service routine, this permits higher priority peripherals to interrupt the CPU while the latter is still executing that service routine. Thus, nested interrupts are allowed in which the higher priority peripheral suspends the execution of the lower priority peripheral's service routine.

Bus-requesting daisy chains do not have this preemption or nesting ability. Instead, any peripheral which is able to get the bus keeps it until the completion of its task.

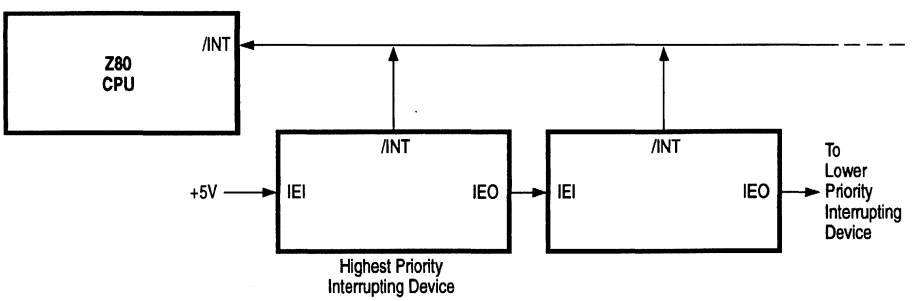


Figure 4-10. Interrupt Daisy Chain

4.4 INTERRUPTS (Continued)

4.4.8 Polling for Service Requests

When the CPU cannot detect interrupts directly, it can poll an external gate as shown in Figure 4-11.

Polling is done in the following way:

- Enable the DMA's interrupt structure with a control byte
- Poll a status bit to see when an interrupt request occurs

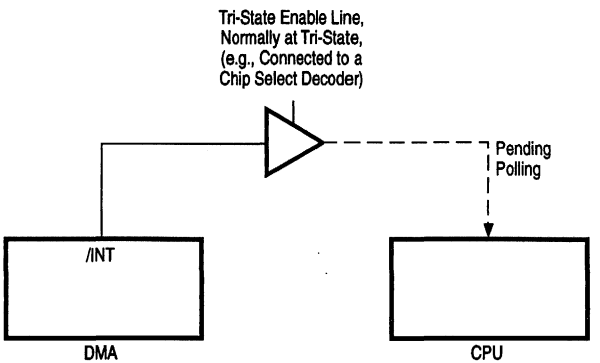


Figure 4-11. Polling for a Service Request Bit

CHAPTER 5
PROGRAMMING

5.0 OVERVIEW

The DMA must be programmed before it can be used. Its control registers have no useful default values on power up. In addition, commands are frequently written to the DMA after the initial power-up programming sets its basic operating modes; this is most commonly done within service routines for purposes such as reading status, changing starting addresses, and reenabling both interrupt and bus-request logic after a block transfer or search.

The DMA has two fundamental states that can be set programmatically: (1) an enabled state, in which it can gain control of the system buses and direct the transfer of data between ports or the searching of data from a single port; and (2) a disabled state, in which it can initiate neither bus requests nor data transfers. Table 5-1 shows these states and their sub-states in detail. When the DMA is powered up or reset by any means, it is automatically placed into the disabled state. Program commands can be written to it by the CPU in the enable/inactive state, but this automatically puts the DMA into the disabled state, which is maintained until an ENABLE DMA command is written by the CPU to the DMA's Write Register 6 (WR6).

Within the Z80 Family, the DMA normally exists as a peripheral device in system I/O space. Its Chip Enable (/CE) signal is decoded from the lower byte of the address bus for this purpose and all control bytes and status bytes are written to and read from the same I/O port address, using an output instruction such as OTIR (in the Z80 CPU).

It is possible to use the DMA in memory mapped I/O structures, but this involves some external logic (which is explained in the "Applications" chapter). It is not possible for the DMA to program itself by directing transfers of control bytes from memory to its own internal registers.

When DMA interrupt vectors are used in a Z80 environment, the Z80 CPU should be programmed for Mode-2 maskable interrupts.

Table 5-1. DMA Status

		ENABLED		
		Inactive	Active	
	DISABLED	(Stopped)	Suspended	Operating
Description	DMA cannot request the bus (cannot pull its /BUSREQ input to CPU low).	DMA can request the bus and may have had the bus immediately prior to this state, but it is not currently the bus master.	DMA is bus master but no operations are taking place.	DMA is bus master and is transferring and/or searching in one of its three modes (Byte, Burst, or Continuous).
Can the CPU write DMA control bytes or read DMA status bytes?	Yes	Yes (But write a DISABLE DMA command first)	No	No
External actions which will cause the state	Power-down	End-of-block in any mode, except with Auto Restart. Byte Match in any mode. Byte or Burst mode /BAI line inactive. Loss of power.	RDY line inactive in Continuous mode.	RDY line active in Burst mode, if DMA is enabled. RETI instruction fetched by CPU, if DMA is enabled and RDY line is active.
DMA commands (WR6 control bytes) which will cause the state.	Any command except the ENABLE DMA command (and possibly the REINITIALIZE STATUS BYTE command, if it is not preceded by some other command). The DISABLE DMA command is specifically designed for this.	ENABLE DMA if RDY line is inactive and the FORCE READY command is not used.	ENABLE DMA, if RDY line is inactive in Continuous mode.	ENABLE DMA, if RDY is active or the FORCE used and the command is outside an interrupt service routine.

5.1 WRITE REGISTERS

Control bytes must be written into all relevant registers in the DMA at power-up initialization. This section describes and illustrates each of the write registers, WR0 through WR6, into which control bytes can be written. The convention of calling the control bytes written to WR6 "commands" is often used, since they are commonly used within CPU interrupt service routines and at other times during system operation in addition to their use at power-up initialization of the DMA.

Chapter 4, "Internal Structure," gives an organizational overview of the write registers (Figure 4-2) and describes the general method of accessing them: control bytes are written into one or more of the write register groups (WR6-WR0) by first writing a byte to the "base register" in that group. All groups have base registers and most groups have additional associated registers. The associated registers in a group are sequentially accessed by first writing a byte to the base register. The base register byte contains both control bits, for DMA function control, and pointer bits (Is) to one or more of the associated registers in the base register's group.

Figure 5-1 for WR0 illustrates this. In this figure, the sequence in which associated registers within a group can be written to is shown by the vertical position of the associated registers. For example, if a byte written to the DMA contains the bits that identify WR0 (bits D0, D1, and D7), and also contains 1s in the bit positions that point to associated registers 2 and 4, then the next two bytes written to the DMA after the base register byte will be stored in these two associated registers, in that order.

Figures 5-2 through 5-8 illustrate and describe in detail each of seven base registers and their associated registers. These figures, unlike Figure 4-2, do not include the 16-bit counters associated with the starting-address and block-length registers.

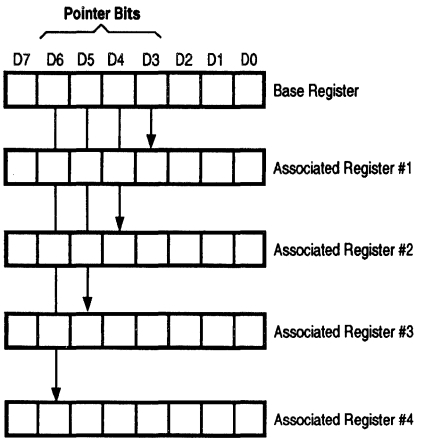


Figure 5-1. Method of Write-Register Pointing

C

5.2 WRITE REGISTER 0 GROUP

The WR0 base register byte is identified by a 0 in bit 7 and any combination but 0, 0 in bits 0 and 1 (Figure 5-2). It is used to set the following conditions.

5.2.1 Class of Operation

Bits 1 and 0 together set the class of operation as sequential transfer (0,1), search-only (1,0), or sequential/transfer/search (1,1). Simultaneous transfers or transfer/searches are obtained by selecting the search-only class here (1,0) and letting the external hardware take care of generating the appropriate bus control signals for the complete transfer (see the chapter on "Applications").

5.2.2 Source and Destination

Bit 2 declares the source port and, by implication, the destination port, if the operation is a sequential transfer. When bit 2 is 0, Port B is the source; when bit 2 is 1, Port A is the source. Search-only operations have only a source port. If the operation is a simultaneous transfer or transfer/search (where the class is set to search-only), external hardwiring determines the destination port.

NOTE: The direction of transfer should only be changed from its current setting after the DMA has been disabled by writing some other control byte to it. This means that the WR0 byte should not be the first byte written to the DMA if the direction of transfer is being changed.

5.2.3 Port A Starting Address

If Port A is used for either source or destination, its starting address must be programmed. This is done by setting bits 3 and 4 of in the base register byte to 1 so that the next two bytes written to the DMA will be recognized as the low and high bytes, respectively, of the Port A starting address. This address is interpreted in the context of the entries in WR1 bits 3 through 5, which declare the address as memory or I/O, fixed or variable, and (if variable) incrementing or decrementing. If Port A is to be a fixed address destination port, see the section following entitled "Fixed-Address Destination Ports."

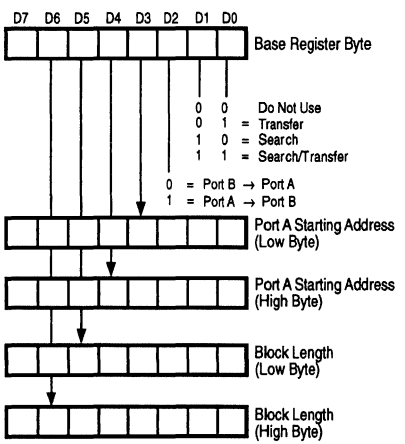


Figure 5-2. Write Register 0 Group

5.8 WRITE REGISTER 6 GROUP (Continued)

Only the source-port address counter is immediately loaded. The destination-port address counter (if used) is loaded during the first count of the destination-port address. If the destination-port address is fixed, this means that it is never loaded. This special situation is discussed in a later section entitled "Fixed-Address Destination Ports."

If the DMA is in an inactive state (Table 5-1) when the LOAD command is written, another DMA control byte must precede the LOAD. Any other command, such as DISABLE DMA, serves this purpose.

Since LOAD unforces a Forced-Ready condition, the LOAD must precede a FORCE READY command when the latter is used.

CONTINUE (D3). This command clears the byte counter to zero but leaves the address counters of both ports with their current contents. Transfers or searches continue from where they left off after an ENABLE DMA command, although the byte count starts over.

The CONTINUE command is typically used to transfer several blocks into consecutive locations in memory when it is desirable to know when each block has finished transferring. Specifically, an interrupt at the end of each block may be needed. Use this command rather than a LOAD command to transfer the next block after the interrupt. A new block length can be entered in WR0 in conjunction with the CONTINUE command.

If the DMA is in an inactive state (Table 5-1) when the CONTINUE command is written, another DMA control byte must precede the CONTINUE. Any other command, such as DISABLE DMA, serves this purpose.

DISABLE INTERRUPTS (AF). The command is used in non-Z80 CPU environments to simulate the Z80 CPU's automatic interrupt acknowledge to the DMA. When the DMA interrupts a non-Z80 CPU, the CPU writes a DISABLE INTERRUPTS to the DMA early in the service routine. This allows the /INT line to go inactive but prevents the DMA from sending subsequent interrupts while the routine is being executed. Near the end of the routine, the CPU writes an ENABLE INTERRUPTS command to the DMA, which enables it to generate a new interrupt.

This command is less extensive than the RESET AND DISABLE INTERRUPTS command because it does not reset the Interrupt Pending (IP) and Interrupt Under Service (IUS) latches.

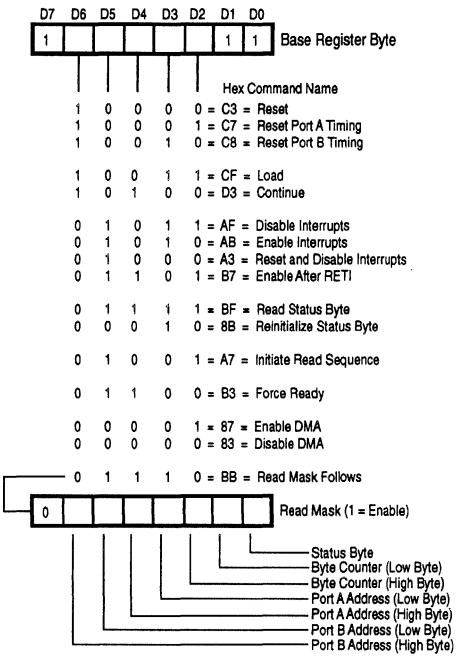


Figure 5-8. Write Register 6 Group

ENABLE INTERRUPTS (AB). See the preceding description of DISABLE INTERRUPTS. A Z80 CPU environment uses this command at power-up to enable the interrupt logic at the beginning (the DMA comes up with this logic disabled). It is not needed, however, to enable subsequent interrupts because this function is provided for by the CPU's fetching-and the DMA's decoding of the RETI instruction. The only exception to this is when the DISABLE INTERRUPTS command is used; then the ENABLE INTERRUPTS command must also be used to begin DMA operations again.

Any conditions selected to cause an interrupt are latched in the DMA even when interrupts are disabled. They can then cause a later interrupt after interrupts are reenabled.

The ENABLE INTERRUPTS command must not be written until after the DMA has been configured and the REINITIALIZE STATUS BYTE command has been written. This command has the same effect as writing a 1 to bit 5 of WR3.

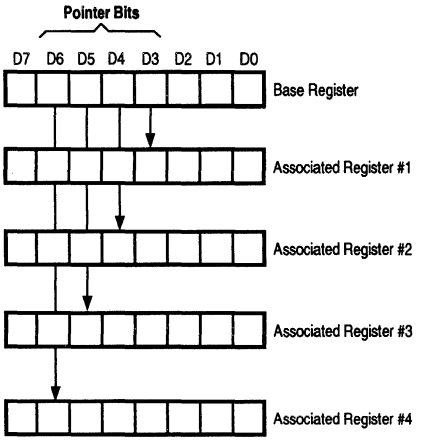


Figure 5-1. Method of Write-Register Pointing

C

5.2 WRITE REGISTER 0 GROUP

The WR0 base register byte is identified by a 0 in bit 7 and any combination but 0, 0 in bits 0 and 1 (Figure 5-2). It is used to set the following conditions.

5.2.1 Class of Operation

Bits 1 and 0 together set the class of operation as sequential transfer (0,1), search-only (1,0), or sequential/transfer/search (1,1). Simultaneous transfers or transfer/searches are obtained by selecting the search-only class here (1,0) and letting the external hardware take care of generating the appropriate bus control signals for the complete transfer (see the chapter on "Applications").

5.2.2 Source and Destination

Bit 2 declares the source port and, by implication, the destination port, if the operation is a sequential transfer. When bit 2 is 0, Port B is the source; when bit 2 is 1, Port A is the source. Search-only operations have only a source port. If the operation is a simultaneous transfer or transfer/search (where the class is set to search-only), external hardwiring determines the destination port.

NOTE: The direction of transfer should only be changed from its current setting after the DMA has been disabled by writing some other control byte to it. This means that the WR0 byte should not be the first byte written to the DMA if the direction of transfer is being changed.

5.2.3 Port A Starting Address

If Port A is used for either source or destination, its starting address must be programmed. This is done by setting bits 3 and 4 of in the base register byte to 1 so that the next two bytes written to the DMA will be recognized as the low and high bytes, respectively, of the Port A starting address. This address is interpreted in the context of the entries in WR1 bits 3 through 5, which declare the address as memory or I/O, fixed or variable, and (if variable) incrementing or decrementing. If Port A is to be a fixed address destination port, see the section following entitled "Fixed-Address Destination Ports."

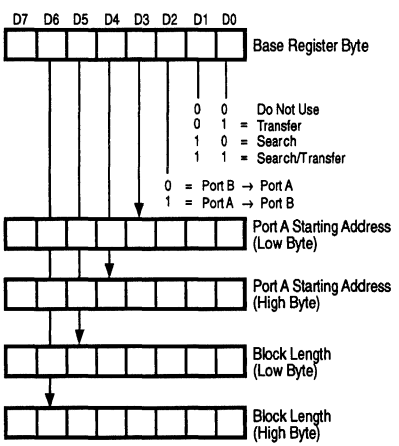


Figure 5-2. Write Register 0 Group

5.2 WRITE REGISTER 0 GROUP (Continued)

5.2.3 Block Length

All operations must have a declared block length since the default values at power-up are unpredictable for block length. These registers are written to by setting pointer bits 5 and 6 in the WR0 base register byte. The block length can be up to 64 Kbytes. Due to the pipelining method of reading in data, the number of bytes actually searched or transferred may be one or two more than the number entered here. The section on "Address and Byte, Counting" in Chapter 4, "Internal Structure" describes this (Table 4-1).

Programming a block length of zero results in the transfer or search of 216 + 1 bytes. Therefore, the shortest block length that can be entered is 1, which usually results in a transfer or search of two bytes (Table 4-2).

5.3 WRITE REGISTER 1 GROUP

Bits 7, 2, 1, and 0, as Figure 5-3 shows, select the base register byte for this group. The group is used only when Port A is used (i.e., do not program it for a search-only, simultaneous transfer, or simultaneous transfer/search with Port B as the source). It specifies the following characteristics:

5.3.1 Device Type (Port A)

Bit 3 identifies Port A as either memory or I/O. This specification, causes the proper control line /MREQ or /IORQ) to come active for cycles involving that port.

5.3.2 Variable vs Fixed Addressing (Port A)

Bits 4 and 5 specify whether the Port A address increments, decrements, or remains fixed for each byte of data transferred or searched. The first byte of data in an operation uses the starting address entered for Port A in WR0; incrementing or decrementing begins on the second byte of the operation.

5.3.3 Variable Cycle (Port A)

If bit 6 is set to 0, the DMA's variable-cycle timing feature is not used; instead, standard Z80 timing for read and write cycles is used, as described in the "Timing" chapter. If bit 6 is set to 1, the next byte written to the DMA after the WR1 base register byte will be the Port A variable-timing byte. This allows the length of the port's read and write cycles to be shortened. The choices for overall cycle timing of the DMA, including activation of the /IORQ, /MREQ, /RD, and /WR lines, are specified in bits 1 and 0 as:

4 clock cycles
3 clock cycles
2 clock cycles

In addition, bits 7, 6, 3, and 2 of the variable-timing byte allow termination of various lines 1/2 cycle earlier than specified in bits 1 and 0. The chapter on "Timing" illustrates and describes the effect of this in detail.

Particular note must be taken of the /IORQ line when variable-cycle timing is used in sequential transfers or transfer/searches. If an I/O-to-memory or memory-to-I/O operation is being done, the memory port must be programmed to have its /IORQ line ending early. (This is done in spite of the fact that the /IORQ line normally has nothing to do with memory). However, this requirement can be left off the CMOS DMA counter controller. If an I/O-to-I/O operation is being done, both ports must have their /IORQ lines ending early.

This situation arises from the fact that the /IORQ line changes logic levels off a different clock cycle edge than the other control lines when the variable-timing feature is employed.

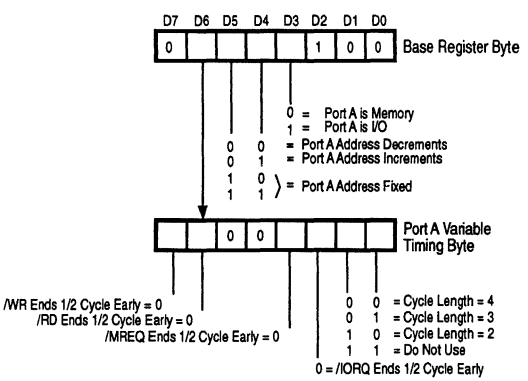


Figure 5-3. Write Register 1 Group

5.4 WRITE REGISTER 2 GROUP

Bits 7, 2, 1, and 0, as shown in Figure 5-4, specify the base register byte for this group. The group is used only when Port B is used (i.e., do not program it for a search-only, simultaneous transfer, or simultaneous transfer/search with Port A as the source). Its syntax is exactly the same as WR1.

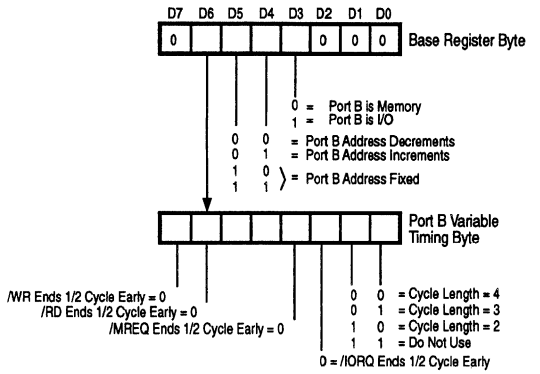


Figure 5-4. Write Register 2 Group

5.5 WRITE REGISTER 3 GROUP

Bits 7, 1, and 0, as shown in Figure 5-5, specify the base register byte for this group. The group is used primarily to specify the stop-on-match condition as well as the match byte itself for a search operation. It can do fast, one-byte enabling of both bus requests and interrupts. A description of its functions follows.

5.5.1 Stop on Match

Setting bit 2 of the base register byte to 1 causes the DMA to stop and release the bus when a data byte matches the match byte (described later). A search or transfer/search operation must be specified in WR0 to make this bit valid when set. If this bit is 0 (no stop on match), a status flag is still set in the status byte when a match occurs and there still remains the option of interrupting on match (see WR4). No stop or interrupt on match in the search class is used to obtain simultaneous transfers without searching actions.

5.5.2 Match Byte

When bit 4 of the base register is set to 1, the match byte that is compared with every data byte searched must be specified. A search operation must be specified in WR0 to make this bit valid, as shown in the following function.

5.5.3 Mask Byte

When bit 3 is set to 1, the mask byte must be subsequently specified. Bit positions that contain 1s in the mask byte will cause comparisons at those same bit positions in the match byte (preceding paragraph) to be ignored. For example, if the mask byte is 00001111, only the high four bits of the match byte will be compared to the data bytes being searched.

5.5.4 Interrupt Enable

A 1 in bit 5 of the base register enables the DMA to generate an interrupt. This function duplicates the ENABLE INTERRUPTS command in WR6.

5.5.5 DMA Enable

A 1 in bit 6 of the base register enables the DMA to request the bus. This function duplicates the ENABLE DMA command in WR6 and is used as the last control byte written to the DMA prior to allowing the DMA to usurp the bus from the CPU. The ENABLE DMA command is often better for this purpose.

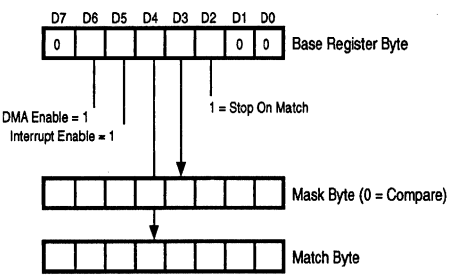


Figure 5-5. Write Register 3 Group

5.6 WRITE REGISTER 4 GROUP

Bits 7, 1, and 0, as Figure 5-6 shows, select the base register byte for this group. The group specifies the following characteristics:

5.6.1 Mode of Operation

Bits 6 and 5 of the base register specify the operating mode as Byte, Burst, or Continuous. For a review of these modes, see Figures 2-3, 2-4, 2-5, 2-6, and Tables 4-1 and 4-2.

5.6.2 Starting Address (Port B)

The starting address for Port B in the next two bytes may be specified by setting bits 2 and 3 of the base register to 1. This is only needed if Port B is used, and then it specifies the first address at which a byte will be read from or written to, depending on whether the port is declared a source or destination in WR0. If Port B is to be a fixed-address destination, see the following section entitled "Fixed-Address Destination Ports."

5.6.3 Interrupts

Bit 4 of the base register byte can point to the interrupt control byte, and bits 4 and 3 of the interrupt control byte can point to the interrupt vector and pulse control bytes, respectively. The interrupt control byte also specifies one or more of the following three interrupt conditions:

- Interrupt on match (bit 0), if stop on match or stop on end-of-block is also programmed.
- Interrupt at end-of-block (bit 1), if stop on end-of-block is also programmed.
- Interrupt on Ready (bit 6), i.e., interrupt before requesting the bus when the Ready line becomes active.

Setting any of these bits to 1 enables the interrupt condition but not the interrupt circuitry itself, which is enabled either through the ENABLE INTERRUPTS command in WR6 or through bit 5 in WR3. Interrupts do not occur on these conditions if their associated bits are 0 in the interrupt control byte. Tables 4-1 and 4-2 in the previous chapter apply to these interrupt conditions since the DMA releases the bus (stops) before interrupting the CPU.

5.6.4 Interrupt Vector

Bit 4 of the interrupt control byte allows the interrupt vector to be entered. In addition, when bit 5 of the interrupt control byte (Status Affects Vector) is set to 1, bits 1 and 2 of the interrupt vector are modified to reflect the cause of the interrupt (i.e., the state of the Ready line or Status latches) before the vector is placed on the data bus in response to the CPU's interrupt acknowledge.

The Status Affects Vector mode should not be used when both Auto Restart and interrupt on end-of-block have been programmed, because the interrupt vector sent at the end of each block in this case cannot be modified to reflect the end-of-block status.

5.6.5 Pulse Generation

Pulse generation is caused by (1) pointing to the interrupt control byte with the base register byte, (2) setting bits 2 and 3 of the interrupt control byte, and (3) entering an offset value in the pulse control byte. The pulse control byte is compared with the lower byte of the byte counter and a pulse is generated on the /INT line whenever a match occurs, which is every 256-byte transfers or searches after the initial offset number of bytes.

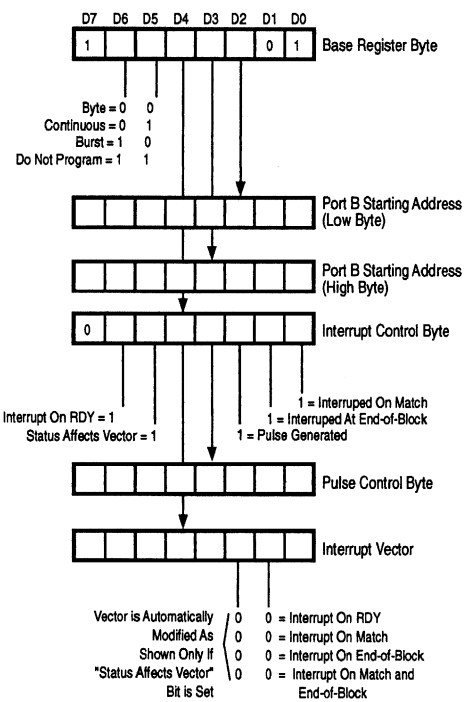


Figure 5-6. Writs Register 4 Group

5.7 WRITE REGISTER 5 GROUP

Bits 7, 6, 2, 1, and 0, as Figure 5-7 shows, specify the base register byte for this one register group. The byte is used to specify these characteristics:

5.7.1 End-of-Block Action

Bit 5 specifies either a stop (bus release) or an auto repeat at the end of the block length programmed in WR0. To interrupt at the end of a block (WR4), bit 5 should be 0 since, the DMA must reset the end-of-block status bit to proceed with a new block (in Auto Restart, the end-of-block status bit is also reset).

5.7.2 /CE//WAIT Line Usage

Bit 4 specifies that the DMA's /CE//WAIT line is to be used in one of two ways:

/CE Only. The /CE/WAIT line functions only as a chip-enable line to allow CPU writing and reading of control/status bytes when the DMA is not bus master (see the "Applications" chapter for the method by which this time is decoded from the address bus).

/CE//WAIT Multiplex. This line functions as described in "/CE only" when the DMA is not bus master. When the DMA has the bus, however, the line allows external Wait inputs from external logic to extend the DMA's cycle programmed in WR1 and/or WR2. (See the "Applications" chapter for hardware interfacing of this option.)

5.7.3 Ready-Line State

Bit 3 specifies that the DMA interprets the Ready (RDY) line as active when High or active when Low. This allows flexibility in interfacing to a variety of other devices.

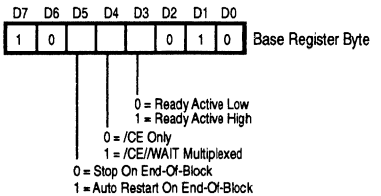


Figure 5-7. Write Register 5 Group

C

5.8 WRITE REGISTER 6 GROUP

The base register byte for this group has bits 7, 1, and 0 set to one, as Figure 5-8 shows. The remaining bits specify 16 commands commonly used after DMA initialization (e.g., within CPU interrupt service routines), and to point to a read mask for the read registers.

Each of these commands, except the ENABLE DMA command, disables the DMA. Therefore, the ENABLE DMA command must be the last command written before DMA bus requests can begin.

RESET (C3). This command is used at power-up and when aborting a program sequence to do the following:

- Disable interrupt and bus-request logic.
- Reset interrupt latches.
- Unforce a FORCE READY condition.
- Reset the Auto Repeat function (see WR5).
- Reset the Wait function (See WR5).
- Reinitialize Ports A and B to standard Z80 cycle timing (see WR1 and WR2).

At power-up, one reset command should be sent to the DMA prior to the initialization program. When aborting an operation sequence, sending six reset commands guarantees resetting (this results from WR4 having five associated registers that can potentially be pointed to).

The RESET command does not perform a complete DMA reset. For example, it does not reset the read sequence, which is set by the INITIATE READ SEQUENCE command.

RESET PORT A TIMING (C7). Resets the Port A variable-timing byte in WR1 to standard Z80 timing. (The RESET command also perform this function.)

RESET PORT B TIMING (CB). Resets the Port B variable-timing byte in WR2 as described in RESET PORT A TIMING (C7).

LOAD (CF). This command must be used to write new addresses to the address registers (WR0 and/or WR4) or to restart an operation (except Auto Restart) at the same addresses. It loads the contents of both starting-address registers into their associated address counters (Figure 4-2). It also clears the byte counter associated with the block-length register, and it unforces an internal Force-Ready condition. The starting addresses must be written in WR0 and/or WR4 before the LOAD command is written, if they are to differ from the previous starting addresses.

5.8 WRITE REGISTER 6 GROUP (Continued)

Only the source-port address counter is immediately loaded. The destination-port address counter (if used) is loaded during the first count of the destination-port address. If the destination-port address is fixed, this means that it is never loaded. This special situation is discussed in a later section entitled "Fixed-Address Destination Ports."

If the DMA is in an inactive state (Table 5-1) when the LOAD command is written, another DMA control byte must precede the LOAD. Any other command, such as DISABLE DMA, serves this purpose.

Since LOAD unforces a Forced-Ready condition, the LOAD must precede a FORCE READY command when the latter is used.

CONTINUE (D3). This command clears the byte counter to zero but leaves the address counters of both ports with their current contents. Transfers or searches continue from where they left off after an ENABLE DMA command, although the byte count starts over.

The CONTINUE command is typically used to transfer several blocks into consecutive locations in memory when it is desirable to know when each block has finished transferring. Specifically, an interrupt at the end of each block may be needed. Use this command rather than a LOAD command to transfer the next block after the interrupt. A new block length can be entered in WR0 in conjunction with the CONTINUE command.

If the DMA is in an inactive state (Table 5-1) when the CONTINUE command is written, another DMA control byte must precede the CONTINUE. Any other command, such as DISABLE DMA, serves this purpose.

DISABLE INTERRUPTS (AF). The command is used in non-Z80 CPU environments to simulate the Z80 CPU's automatic interrupt acknowledge to the DMA. When the DMA interrupts a non-Z80 CPU, the CPU writes a DISABLE INTERRUPTS to the DMA early in the service routine. This allows the /INT line to go inactive but prevents the DMA from sending subsequent interrupts while the routine is being executed. Near the end of the routine, the CPU writes an ENABLE INTERRUPTS command to the DMA, which enables it to generate a new interrupt.

This command is less extensive than the RESET AND DISABLE INTERRUPTS command because it does not reset the Interrupt Pending (IP) and Interrupt Under Service (IUS) latches.

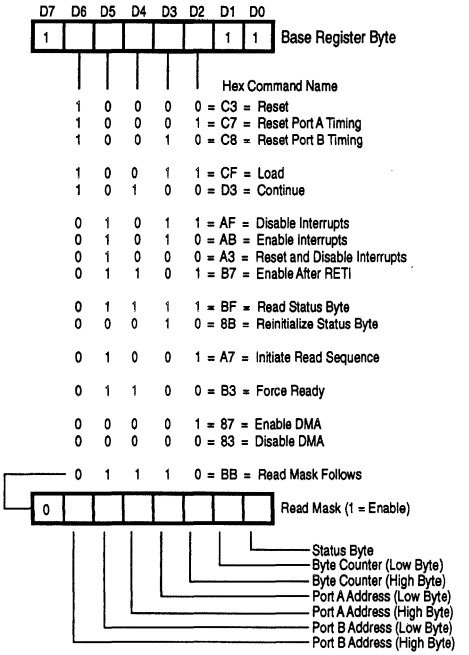


Figure 5-8. Write Register 6 Group

ENABLE INTERRUPTS (AB). See the preceding description of DISABLE INTERRUPTS. A Z80 CPU environment uses this command at power-up to enable the interrupt logic at the beginning (the DMA comes up with this logic disabled). It is not needed, however, to enable subsequent interrupts because this function is provided for by the CPU's fetching-and the DMA's decoding of the RETI instruction. The only exception to this is when the DISABLE INTERRUPTS command is used; then the ENABLE INTERRUPTS command must also be used to begin DMA operations again.

Any conditions selected to cause an interrupt are latched in the DMA even when interrupts are disabled. They can then cause a later interrupt after interrupts are reenabled.

The ENABLE INTERRUPTS command must not be written until after the DMA has been configured and the REINITIALIZE STATUS BYTE command has been written. This command has the same effect as writing a 1 to bit 5 of WR3.

RESET AND DISABLE INTERRUPTS (A3). This command is useful in CPU environments such as the 8080 and 8085 where there is an interrupt acknowledge function but no RETI instruction, as in the Z80 CPU. This command does four things:

- Resets the Interrupt Under Service (IUS) latch.
- Resets the Interrupt Pending (IP) latch.
- Unforces an internal FORCE READY condition.
- Disables further interrupts by the DMA (same as the DISABLE INTERRUPTS command).

In the non-Z80 environment just described it would be used as follows: after the DMA interrupt is received and acknowledged, the interrupt vector is sent to the CPU, which branches to the service routine. Near the end of the service routine, the CPU writes a RESET AND DISABLE INTERRUPTS command, then an ENABLE INTERRUPTS command, and then an ENABLE DMA command before executing its return-from-interrupt instruction.

This command, when followed by an ENABLE INTERRUPTS command, takes the place of the Z80 BET] instruction. It is not needed in a Z80 environment. Since RESET AND DISABLE INTERRUPTS unforces a Forced-Ready condition, the RESET AND DISABLE INTERRUPTS must precede a FORCE READY command when the latter is used.

ENABLE AFTER RETI (B7). This command is used only when the DMA is operated in the Interrupt On Ready mode (programmed in WR4). It enables the DMA to request the bus again after returning from an interrupt. Always use this command in Z80 CPU environments to get further bus requesting after an Interrupt on Ready. It is sometimes used in other environments, such as the 8080.

An Interrupt on Ready (IOR) latch is set during such an interrupt. This latch prevents the DMA from requesting the bus from the time the Ready line goes active until the time the latch is reset by the ENABLE AFTER RETI command (in Z80 and some other environments, there is an overlap in bus-request prevention by the IOR and the IUS latches).

In a Z80 CPU interrupt service routine, the order of DMA commands and CPU instructions must be:

•
•
•
ENABLE AFTER RETI command
ENABLE DMA command
•
•
•
RETI instruction

READ STATUS BYTE (BF). This command causes the next CPU read of the DMA to access the status byte, which is illustrated in the following section entitled "Read Registers."

If other read registers are being read, the sequence of reading (as defined by the read mask) should be completed before issuing this command.

REINITIALIZE STATUS BYTE (8B). This command reinitializes bits 4 and 5 of the status byte. After reinitialization, the status byte looks like this:

Bit	Value	Meaning
0	I/O	DMA operation has/hasn't occurred
1	I/O	Ready line active/inactive
2	X	Undefined bit
3	0/1	Interrupt pending/not pending
4	1	Match not found
5	1	Not end-of-block
6	X	Undefined bit
7	X	Undefined bit

The DISABLE DMA or any other command must be used before the REINITIALIZE STATUS BYTE command after having stopped on end-of-block or byte match. Due to a potential hardware race condition internal to the DMA, reinitialization of the status bits may remove the condition that stopped the DMA and the DMA might immediately request the bus again if it is not disabled. (The REINITIALIZE STATUS BYTE command in WR6 is similar in this respect to the WR0 byte when transfer direction is being changed: both of these control bytes must be preceded by some other control bytes to ensure that the DMA is disabled.)

The interrupt pending status (bit 3 of the status byte) can be reinitialized by acknowledging the interrupt, servicing it, and writing a RESET AND DISABLE INTERRUPTS command. The DMA operation status (bit 0) can be reinitialized with a LOAD command.

5.8 WRITE REGISTER 6 GROUP (Continued)

READ MASK FOLLOWS (BB). This command points to the read mask (Figure 5-8). It means that the next control byte written to the DMA goes to the read mask register. The read mask is used to set a new sequence, for reading the read registers, RR0 through RR6, and it is normally part of the power-up initialization of the DMA.

The read registers are always read in a fixed sequence beginning with RR0 and ending with RR6. However, the registers read in this sequence can be, limited by programming the read mask. The read mask is programmed with 1s in the bit positions associated with the registers to be read. For example, if the read mask contains 00011001, the following read registers are read in the following order:

Status byte (RR0)
Port A address counter, low byte (RR3)
Port A address counter, high byte (RR4)

Once the read mask is programmed it must be initialized to begin at the lowest-order register selected. Do this with the INITIATE READ SEQUENCE command.

INITIATE READ SEQUENCE (A7). This command initiates the read-sequence pointer command so that the next CPU read instruction to the DMA accesses the first (low-order) read register designated as readable by the read mask. Once started, the read sequence specified by the read mask must be completed before, for example, giving another INITIATE READ SEQUENCE or a READ STATUS BYTE command.

Registers needn't be read immediately after writing the INITIATE READ SEQUENCE command. Other commands (except INITIATE READ SEQUENCE and READ STATUS BYTE) can be written and go through bus-request/bus-release cycles before actually executing the first read and subsequent reads.

FORCE READY (B3). This command, in Burst or Continuous mode, forces an internal Ready condition to take the place of an external active Ready signal. It is used for memory-to-memory transfers and memory searches where no Ready line is necessary. Ready active High/Low (bit 3 of WR5) need not be considered when this command is used.

The FORCE READY condition is unforced by the following commands and conditions:

- RESET command
- LOAD command
- RESET AND DISABLE INTERRUPTS command
- End-of-block termination
- Byte-match termination
- Bus release by DMA

Because bus release by the DMA unforces the Ready condition, this command allows the DMA to transfer only one byte in the byte mode.

ENABLE DMA (87). This command allows the DMA to request the system bus and proceed with its operation if all other functional conditions are met (e.g., if the Ready line is active or the FORCE READY condition is present. This command, and bit 6 of WR3, are the only control bytes that do not disable the DMA; all other control bytes written to the DMA automatically disable the DMA. Therefore, the ENABLE DMA command is always required as the last command after writing or reading any other bytes to or from the DMA.

This command enables the DMA's bus request logic. It does not affect interrupt logic and it does not reset any functions or latches. This bus-request-enabling function is duplicated in bit 6 of WR3.

In an interrupt service routine, the ENABLE DMA command must be the last command to the DMA before the CPU executes its return-from-interrupt instruction.

DISABLE DMA (83). This command prevents the DMA from requesting the bus. It is used to stop DMA action for external reasons, such as a pending power-out, and in the special case of reinitializing the status byte after a stop on end-of-block or a stop on byte match (see the REINITIALIZE STATUS BYTE command).

5.9 READ REGISTERS

Read registers are read by first writing a command to the DMA, then reading either immediately thereafter or some time later. CPU reads are done by addressing the DMA as an I/O device using input instructions (such as INIR for the Z80 CPU).

The commands written to the DMA can be one of two:

READ STATUS BYTE. This command causes the next CPU read of the DMA to access the status byte, which is the first read register.

INITIATE READ SEQUENCE. This command initializes access to a repeatable series of reads that follow the sequence defined in the read mask.

These commands are described in the immediately preceding pages, and Figure 5-8 illustrates the read mask. As mentioned in the description of these commands, the reading of registers needn't be contiguous in time with these write commands or with other CPU read instructions accessing registers in the same read sequence.

Two other commands are also related to the read registers:

REINITIALIZE STATUS BYTE. This command reinitializes bits 4 and 5 of the status byte to 1s.

READ MASK FOLLOWS. This command allows the read mask to be programmed.

Figures 4-3 and 5-9 both illustrate more clearly the group of seven read registers in relation to the write registers, The read registers include:

5.9.1 Status Byte (RR0)

The status byte can be read independently of the other read registers and two of its bits can be reinitialized to identify end-of-block and match bytes. The bits in the status byte are defined as follows:

Bit 0 Indicates whether the DMA has requested the bus since the last LOAD command. A1 indicates yes, a 0 indicates no.

Bit 1 Indicates whether the DMA's RDY pin currently has a signal input that is defined as active by bit 3 of WR5. A 1 indicates an active Ready line. A 0 indicates an inactive Ready line.

Bit 2 Undefined.

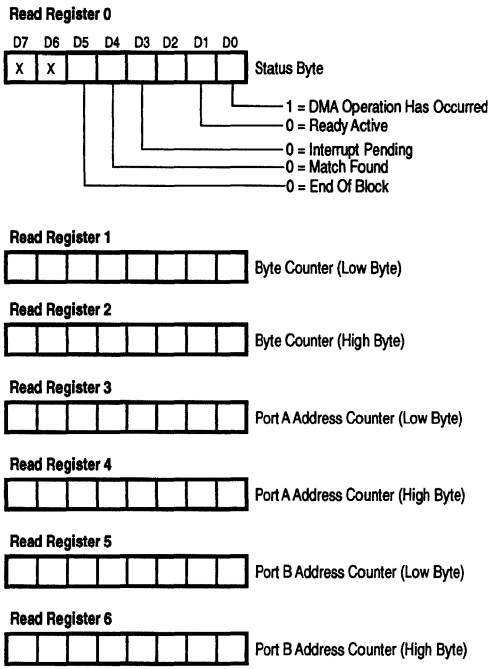


Figure 5-9. Read Register 0 through Read Register 6

Bit 3 Indicates the state of the Interrupt Pending (IP) latch. A 0 indicates either that an interrupt is pending (the DMA has its /INT line active if the interrupt has not been acknowledged). A1 indicates no interrupt pending.

Bit 4 A 0 indicates that a match has been found since the last RESET or REINITIALIZE STATUS BYTE command. A1 indicates no match was found. See Table 4-2 to determine where the match occurred.

Bit 5 A 0 indicates that an end-of-block was reached since the last RESET, LOAD, CONTINUE, or REINITIALIZE STATUS BYTE command. A 1 indicates no end-of-block was reached. See Table 4-1 to determine the contents of counters when the DMA stops.

Bit 6 Undefined.

Bit 7 Undefined.

5.9 READ REGISTERS (Continued)

5.9.2 Byte Counter (RR1, RR2)

This 16-bit counter is cleared to zero by the LOAD, CONTINUE, and RESET commands only. When the DMA begins transferring or searching, the byte counter increments by one at the end of each read cycle and the byte counter is compared with the programmed contents of the block length register to determine end-of-block. The number of bytes read in a transfer always equals the number of bytes written because the DMA completes any transfer it starts, even when stopping on byte matches in transfer/search operations.

Tables 4-1 and 4-2 illustrate how the pipelining of data affects the number of bytes transferred or searched in the various classes, modes, and circumstances of operation. In most cases, the number of bytes transferred in a transfer operation that slops at end-of-block is one more than the programmed block length.

When the pulse-generation feature. is used, the contents of the pulse control byte in WR4 are compared with the lower byte of the byte counter after each byte is transferred.

5.9.3 Port A Address Counter (RR3, RR4)

This 16-bit counter is loaded from the Port A starting address register in WR0 by the LOAD command. It increments, decrements, or remains fixed in accordance with the specifications in WR1. Tables 4-1 and 4-2 show how this counter reads under various transfer or search conditions.

5.9.4 Port B Address Counter (RR5, RR6)

This counter is exactly analogous to the Port A address counter just described. If either Port A or Port B is a fixed-address destination port it must be programmed as described under "Fixed Address Destination Ports" to function properly.

5.10 REVIEW OF PROGRAMMING SEQUENCES

This section contains a review of rules for programming the DMA in both the general case and in various application-specific cases. Also, see Figure 5-10 for a sample DMA Program.

5.10.1 DMA Initialization

All registers to be used in the DMA must be programmed at power-up. None have useful defaults. This includes the enabling of interrupts and reinitialization of the status byte as well as many other functions, including class and mode designation, port designation, address and block-length designation.

Table 5-2 suggests the order in which control bytes should be written for the, general case of either initialization or, reinitialization due to program abort. Some of these control bytes may not be relevant to a specific application. All "commands" referred to are WR6 control bytes. There is a maximum of 35 control bytes when all of the above are written.

All control bytes written to the DMA disable the DMA, except the ENABLE DMA command and possibly also the REINITIALIZE STATUS BYTE command and the WR0 control byte (when changing transfer directions). The ENABLE DMA command must always be the last command written after any communication between the CPU and DMA, if the DMA is to continue operating. Furthermore, communication with the DMA can only occur when the CPU is bus master.

Table 5-2. Control Byte Order

Initialization/Reinitialization Sequence	Maximum No. of Bytes for Z80 CPU
DISABLE DMA Command	1
RESET Command (Multiple)	6
WR0 Control Bytes	5
WR1 Control Bytes	2
WR2 Control Bytes	2
WR3 Control Bytes	3
WR4 Control Bytes	5
WR5 Control Bytes	1
RESET PORT A TIMING Command	1
RESET PORT B TIMING Command	1
LOAD Command	1
REINITIALIZE STATUS BYTE Command	1
READ MASK FOLLOWS Command	1
Read Mask Control Byte	1
INITIATE READ SEQUENCE Command	1
FORCE READY Command	1
ENABLE INTERRUPTS Command	1
ENABLE DMA Command	1
Total	35

5.10.2 Port Designation

Either Port A or Port B can be selected as the source or destination, as illustrated in Figure 2-2, since both ports have the same degree of programmability. A special case arises when the destination port is also a fixed-address port; this is dealt with under "Fixed-Address Destination Ports."

Port characteristics are specified in the following control-byte groups:

Port A	Port B
WR0	WR0
WR1	WR2
WR6	WR4
	WR6

In a transfer, if the direction of transfer (bit 2 of WR0) is being changed, the WR0 control byte must be preceded by some other control byte to insure that the DMA is disabled.

5.10.3 Address Loading

Starting addresses are written into the starting-address registers for each port through WR0 (Port A) and WR4 (Port B). They are loaded into the address counters by the LOAD command. The addresses must he written to the registers before they are loaded into the counters.

New addresses may be written to the address registers at any time when the CPU is bus master, even between transfers and even when the DMA is operating in the Auto Restart mode (e.g., in Byte mode between byte transfers). Except in the Auto Restart mode, however, the new addresses must be reloaded before they are used. If a Forced-Ready condition is used, the LOAD command must precede the FORCE READY command.

5.10.4 Fixed-Address Destination Ports

A special circumstance arises when programming a destination port to have a fixed address. The load command in WR6 only loads a fixed address to a port selected as the source, not to a port selected as the destination. Therefore, a fixed-destination address must be loaded after temporarily declaring its port as a source port. The true source port is subsequently declared as such (thereby implicitly making the other port a destination) and the true source address is then loaded.

The following example illustrates the steps in this procedure, assuming that transfers are to occur from a variable-address source (Port A) to a fixed-address destination (Port B):

1. Write Port B (fixed destination) address to WR4.
2. Temporarily declare, Port B as source in WR0 (bit 2 = 0).
3. Load Port B address with the LOAD command.
4. Write Port A (variable source) starting address to WR0.
5. Declare Port A as source in WR0 (bit 2 = 1).
6. Load Port A address with the LOAD command.
   - •
   - •
   - •
7. Enable DMA with the ENABLE DMA command.

5.10.5 Interrupts

The interrupt vector (WR4) must be written before interrupts using it can occur, and interrupts must be enabled with the ENABLE INTERRUPTS command at initialization or reinitialization. In a Z80 CPU environment, interrupt service routines after DMA initialization usually include, the following commands at the end of the routine:

Interrupt on End-of-Block or Byte Match

- •
- •
- •

ENABLE DMA command

- •
- •
- •

RETI instruction

Interrupt on Ready (before requesting the bus)

- •
- •
- •

ENABLE AFTER RETI command
ENABLE DMA command

- •
- •
- •

RETI instruction

Interrupts on end-of-block, for example, might be done in reading a floppy disk. If the disk transfers 128-byte records, the DMA can be made to interrupt at the end of each record to inform the CPU of its completion. Then the CPU can read the destination (memory) address counter to find the last memory location filled (see Table 4-2 for address-counter contents). A service routine for continuing inputs into contiguous locations of memory typically contains the CONTINUE, REINITIALIZE STATUS BYTE, and ENABLE DMA commands before the CPU's return from interrupt. A service routine for shutting the DMA down after the record arrives typically includes DISABLE DMA and REINITIALIZE STATUS BYTE commands. If the DMA transfer is started by an interrupt from some other device, the service routine for that other device would include an ENABLE DMA command written to the DMA's port address.

C

5.11 REVIEW OF PROGRAMMING SEQUENCE (Continued)

Interrupts on byte match (a search or transfer/search operation) can be implemented so that any ending byte, error indicator, or other character causes the interrupt. This frees the CPU from looking for these characters in a stream of data, it increase throughput, and it reduces CPU software complexity. For example, the DMA might search for end-of-text (EXT) characters or carriage returns in a communications environment and interrupt the CPU only when the complete message frame has arrived. The service routines for this would be very much like those for interrupts on end-of-block.

Interrupts on Ready are somewhat different. First, the DMA cannot be the bus master before the interrupt since the CPU only sees interrupts when the CPU is the bus master (the other types of interrupts are not processed until the bus has been released). Second, to enable the DMA, the ENABLE AFTER RETI command must be used in the service routine after an Interrupt on Ready. The typical purpose of interrupting when the Ready line comes active is to allow the CPU time to consider where a transfer should go (which it does in the service routine). This is ofter done in systems using dynamic memery allocation and it improves the efficiency with which memory can be allocated. For example, the CPU might write and load new starting addresses for a memory destination into the DMA in the service routine. Only at the end of the service routine is the DMA enabled to request the bus. The ENABLE AFTER RETI command, which must precede the ENABLE DMA command, resets a latch which is set when the Interrupt on Ready first occurred.

For non-Z80 CPU environments, the DISABLE INTERRUPTS, ENABLE INTERRUPTS, and RESET AND DISABLE INTERRUPTS commands are available. They can simulate the Z80 CPU's interrupt-acknowledge cycle and return-from-interrupt instruction, both of which the DMA needs to perform and return from interrupts.

5.10.6 Byte Matching (Searches)

In stopping, or stopping and interrupting on match (WR3, WR4), to perform additional operations with the DMA, write the following sequence of commands:

- LOAD or CONTINUE
- REINITIALIZE STATUS BYTE
- ENABLE DMA

Another command (any command except ENABLE DMA) must precede the REINITIALIZE STATUS BYTE command. (TABLE 4-4 shows the contents of various counters when stopping on byte match.)

5.10.7 End-Of-Block

After a stop or stop and interrupt on end-of-block (WR4 or WR5), where it is necessary to perform additional operations with the DMA, write the same sequence of commands listed immediately under "Byte Matching (Searches)." Table 4-4 shows the contents of various counters when stopping on end-of-block.

5.10.8 Auto Restart

To obtain a repetitive transfer or search using the same block length and starting addresses originally entered, initialize the DMA, including bit 5 = 1 in WR5. The loading of addresses and clearing of the byte counter is automatic.

If in Byte mode, or possibly even in Burst mode (where the Ready line is occasionally released), it is possible to alter the starting addresses during a transfer (i.e., between bus requests) without disturbing that transfer. At the end of the transfer in which this occurs, the DMA automatically loads the new addresses into the counter and continues without interruption.

5.10. 9 Force Ready Condition

The FORCE READY command is provided for operations like memory-to-memory transfer or memory search-only where no Ready line from an I/O device is used. However, there are several DMA commands that unforce the Ready condition after the FORCE READY command is written. The sequence of command entry is therefore important. This is described under the FORCE READY command in the section entitled "Write Register 6 Group."

5.10.10 Pulse Generation

To obtain pulses at 256-byte intervals, after a variable offset period, only the WR4 group need be considered. The /INT line is used for these pulses.

5.10.11 Variable Timing

The timing on the /RD, /WR, /MREQ, and /IORQ lines can be varied independently for either port by programming the WR1 and/or WR2 register groups. A special case arises in programming memory-to-I/O, I/O-to-memory, or I/O-to-I/O sequential transfers or transfer/searches. The /IORQ line must be programmed in a specific way. See the "Variable Cycle (Port A)" discussion under WR1.

5.10.12 Enabling the DMA

The last command written to the DMA before an operation is to occur must be, the ENABLE DMA command, or WR3 with bit 6 = 1, which is equivalent. Only this command makes the DMA operate. If all other conditions for operation are satisfied at the time of enabling (e.g., the Ready line is active) the DMA will begin immediately. In an interrupt service routine, the ENABLE DMA command must be the last DMA command written before the return-

from-interrupt instruction. Other instructions may, and usually do, follow the ENABLE DMA command in the service routine before the RETI instruction is executed, but none of these commands are for the DMA.

5.10.13 Reading Status

Two commands can be used to allow the CPU to read DMA status:

READ STATUS BYTE. Causes the next CPU read of the DMA to access the status byte. Every time the status byte is to be read, the READ STATUS BYTE must first be written.

INITIATE READ SEQUENCE. Causes the next CPU read of the DMA to access the first status register specified as readable by the read mask. Subsequent reads of the DMA, which must complete the sequence of all designated readable registers, do not require write commands. Reading of the sequence of registers must be completed before the next READ STATUS BYTE or INITIATE READ SEQUENCE command.

Figure 5-10 illustrates a program to transfer data from memory (Port A) to a peripheral device (Port B). In this example, the Port A memory starting address is 1050H and the Port B peripheral fixed address is 05H. The number of data bytes to be transferred is 1001H bytes (one more than specified by the block length). The table of DMA commands may be stored in consecutive memory locations and transferred to the DMA with an output instruction such as the Z80 CPU's OTIR instruction.

	D7		D5	D4	D3	D2	D1	D0	HEX
WR0 sets DMA to receive block length, Port A starting address and temporarity sets Port B as source.	0	1 Block Length Upper Follows	1 Block Length Upper Follows	1 Port A Upper Address Follows	1 Port A Upper Address Follows	0 B⟶A Temporary for Loading B Address	0 Transfer. No Search		
Port A address (lower)	0	1	0	1	0	0	0	0	50
Port A address (upper)	0	0	0	1	0	0	0	0	10
Block length (lower)	0	0	0	0	0	0	0	0	00
Block length (upper)	0	0	0	1	0	0	0	0	10
WR1 defines Port A as memory with fixed incrementing address.	0	0 No Timing Follows	0 Address Changes	1 Address Changes	0 Port is Memory	1	0	0	14
WR4 defines Port A as memory with fixed incrementing address.	0	0 No Timing Follows	1 Fixed Address	0	1 Port is I/O	0	1	0	28
WR4 sets mode to Burst, sets DMA to expect Port B address.	1	1	0	0 No Interrupt Control Byte Follows	0 No Upper Address	1 Port B Lower Address Follows	0	1	C5
Port B address (lower)	0	0	0	0	0	1	0	1	05
WR5 sets Ready active High.	1	0	0 No Auto Restart	0 No Wait Status	1 RDY Active High	0	1	0	8A
WR6 loads Port B address and resets block counter.*	1	1	0	0	1	1	1	1	CF
WR0 sets Port A as source.*	0	0	0 No Address or Block Length Bytes	0	0	1 B⟶A	0 Transfer. No Search	1	05
WR6 loads Port A address and resets block counter.	1	1	0	0	1	1	1	1	CF
WR6 enables DMA to start operation.	1	0	0	0	0	1	1	1	87

NOTE: The actual number of bytes transferred is one more than specified by the block length.
* These entries are necessary only in the case of a fixed destination address.

Figure 5-10. Sample DMA Program

CHAPTER 6
APPLICATIONS

6.0 Z80 DMA AND CPU

As a member of the Z80 Family, the Z80 DMA's signals and timing are directly compatible with those of the Z80 CPU. When it is bus master, the DMA can exhibit read- and write-cycle characteristics identical to those of the Z80 CPU, thus simplifying system design. In addition, variable timing features allow the system designer to interface memories and I/O devices more easily with non-standard capabilities or requirements. The DMA can shorten its read- or write-cycle timings for higher performance or lengthen and tailor control signals to accommodate slower devices. Because these features are under programmed control, the hardware configuration is invariant to changes in cycle and control signal timings.

6.0.1 Interconnection

In small systems, or where the Z80 DMA shares a board with the CPU, most of the pins on the DMA may be connected directly to the corresponding CPU pins. These include the address bus (A15-A0), the data bus (D7-D0), and the control signals /MREQ, /IORQ, /RD, and /WR. The interrupt request and bus request signals, /INT and /BUSREQ, may also be connected directly to the CPU, in common with corresponding open-drain outputs from other devices. The priority daisy chains for these functions are described in an earlier chapter and are illustrated in Figures 4-4 and 4-10.

Power, ground, and clock signals are also common to the CPU and DMA, but extra care must be taken to provide low-impedance paths and adequate decoupling. The 30 ns clock transition time requirement for Z80 Family parts merits consideration, too. A 300 Ohms pullup from a TTL clock driver output may suffice for small systems operating at the 2.5 MHz rate, but the increased loadings and speeds in larger high-performance systems require active pullup. A complemetary-transistor driver for Z80/Z8000 systems is shown in Figure 6-1.

6.0.2 Chip Selection and Enabling

Z80 peripherals are normally addressed in the 256 address I/O space. Each peripheral Chip is enabled by an active-low Chip Enable (/CE) input. The /CE input becomes active when an active /IORQ signal coincides with the peripheral's address on the low order byte of the address bus. Small systems may dedicate address lines to their few peripherals, obviating decoder hardware. A system using DMA, however, is likely to have more peripherals, so that address decoding by means of PROM or MSI TTL decoder is normally provided.

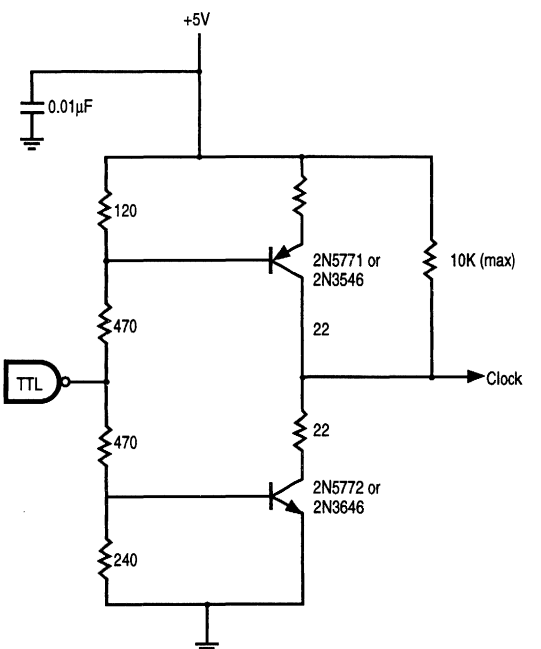


Figure 6-1. Z80/Z8000 Clock Driver

6.0 Z80 DMA AND CPU (Continued)

Figure 6-2 illustrates three chip enable arrangements. In Figure 6-2a, for a small system, the DMA responds to half of the 256 possible I/O addresses. In part (Figure 6-2b), a 256 x 4 PROM has been programmed to provide a low output on its 01 pin only when the DMA's address is present. The PROM must access quickly enough to meet the DMA's /CE setup time requirement.

Figure 6-2c shows a one-of-eight TTL decoder used to provide chip enable signals for eight different peripheral devices. Address bits A0 and A1 are often used directly by peripherals such as the Z80 SIO, PIO, and CTC, and so are not decoded here. Additional decoders can be added when there are more peripheral devices.

/IORQ and /M1 are internally gated with /CE in Z80 peripheral devices and need not be terms in /CE. However, gating chip-enable signals explicitly with these control lines does no harm and may produce less-ambiguous logic sequences for circuit-level debugging. Figure 6-2c shows this.

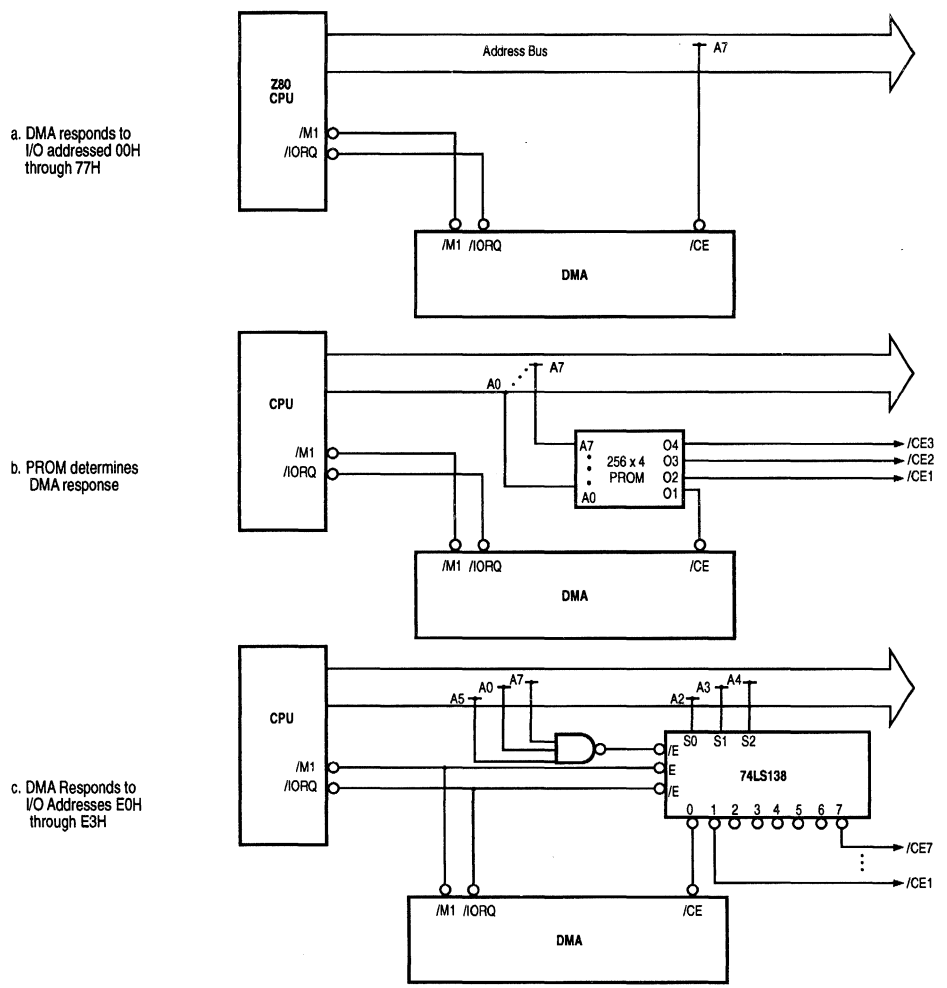


Figure 6-2. Chip Enable Decoding with Z80 CPU

6.0.3 Use of /WAIT Input

When the DMA is bus master, the /CE//WAIT pin functions as an input from memory or I/O logic which may extend read or write cycles by requesting Waits states. An active /BUSACK output from the CPU signals that is has relinquished the bus; thus, if this DMA is bus master, it samples the /WAIT signal for these requests. A simple 2-input multiplexer is used to steer the /CE//WAIT signals as shown in Figure 6-3. (Using /BUSACK assumes there is only one DMA. In systems with three or more possible bus masters, /BAI active and /BAO inactive identify the master.)

6.0.4 Simultaneous Transfers

The highest-speed DMA method is the simultaneous-transfer, or "flyby" arrangement. This requires some external hardware to generate simultaneous read- and write-control signals to the source and destination ports.

Since the address bus is used for memory address, only transfers between I/O and memory can be implemented straightforwardly when the I/O port selection is done by hardwiring. The DMA is put into search mode, and a circuit like that in Figure 6-4 generates separate. simultaneous read- and write-control signals which may he ORed into the read- and write-control paths at memory and I/O. Figure 6-5 shows such an arrangement. This arrangement allows both the CPU and DMA access to the I/O peripheral. (If the peripheral communicates only through DMA, it only needs to use the /IORD and /IOWR signals.)

Careful attention must be paid to access, setup, and hold times in this mode. Since the DMA is programmed to do searching, the /MWR and /IOWR signals are derived from the DMA /RD signal and mimic its timing. This does not cause a problem for write operations, which are trailing edge-activated. To make /MWR look more like a CPU or DMA write cycle signal, the circuit of Figure 6-6 may be used to delay the leading edge of /MWR until after the falling edge in T2. The programmable variable timing features of the DMA may be helpful, too.

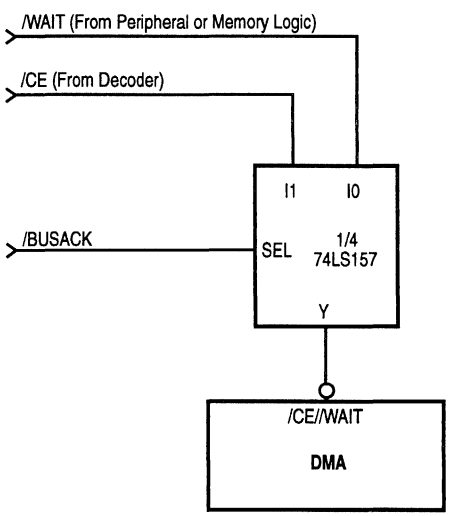


Figure 6-3. /CE//WAIT Multiplexer

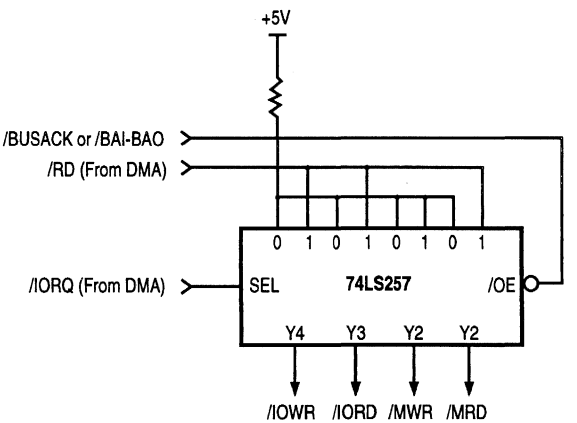


Figure 6-4. Simultaneous Transfer Multiplexer

6.0 Z80 DMA AND CPU (Continued)

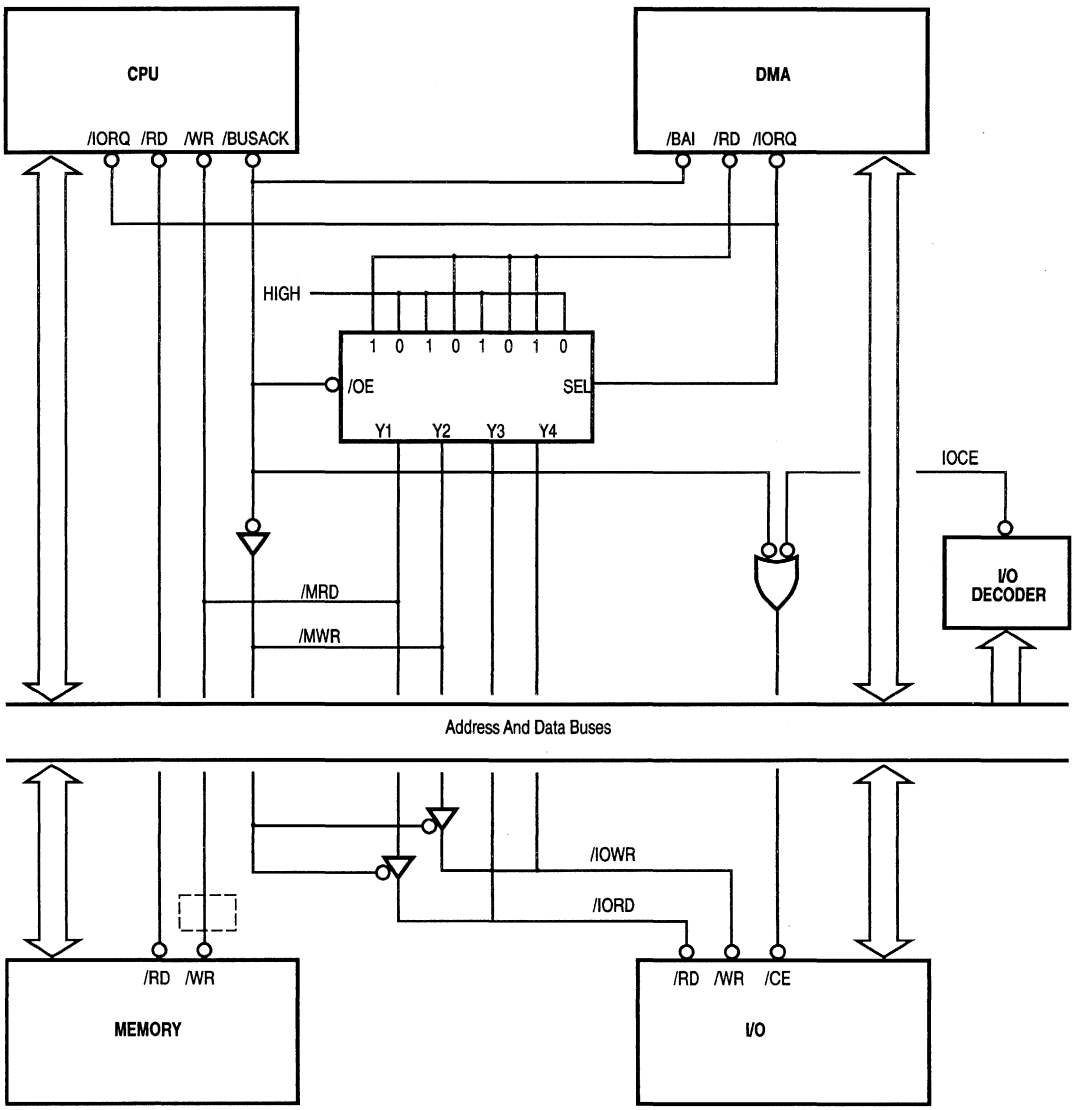


Figure 6-5. Simultaneous Transfer

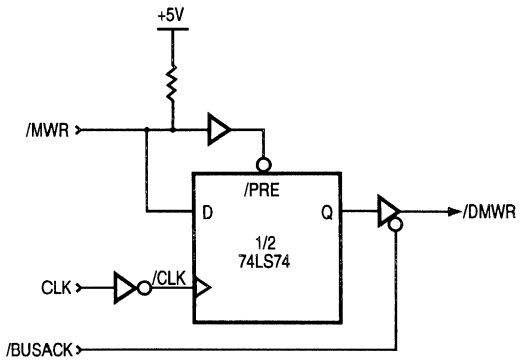


Figure 6-6. Delaying the Leading Edge of /MWR

6.0.5 Bus Buffering

Microcomputer systems using DMA often include large memories, many peripheral devices, or occupy several circuit cards. In these cases, the system buses and control signals need buffering to increase drive capability and noise margin and to decrease delay times.

The need for buffering within a single circuit card can be estimated by comparing drive capabilities of bus master devices (CPU and DMA) to loadings presented by all inputs and outputs connected to the buses. Both static (DC current) and dynamic (capacitive drive) requirements must be considered. When driving a motherboard or other cards, buffering is a practical necessity.

If the bus master devices (CPU and DMAs) are on the same card, they can share buffers for address, data, and control buses to other cards. Otherwise, each card's bus interfaces require buffering.

Address lines are, unidirectional and can be buffered by many common devices such as 74LS244 and 74LS367 (non-inverting tri-state buffer/drivers) or 74LS240 and 74LS366 (inverting tri-state buffer/drivers). The tri-state enable inputs on buffers such as these allow the bus to be isolated (floated) in a manner similar to the CPU and DMA address pins. For example, in a system with one CPU and one DMA, the /BUSACK signal can disable CPU buffers and enable DMA buffers when it is active. Where there can be three or more potential bus masters, only those buffers associated with the actual bus master must be active at any time. Thus, each DMA, if its /BAI signal is active (Low) and its /BAO signal is inactive (High), has control of the bus and can enable its drivers.

Data bus lines are bidirectional, making their buffer control more complicated. Any device from which the CPU can read drives the data bus when it is selected and the /RD control signal is active. In this sense, the /RD signal is the principal directional control. Non-CPU devices also drive the data bus during interrupt-acknowledge cycles (in which the device puts its vector on the bus) and during DMA write cycles. Figure 6-7 illustrates a bidirectional data bus buffer and its control. Here, Z80 SIO, PIO, CTC, and DMA peripherals share a circuit card. Their common on-card data bus is buffered to and from the system (motherboard or backplane) bus. Each of the three conditions mentioned causes the buffers to drive data out onto the system bus; otherwise, data is buffered into the card. Suitable devices for bidirectional buffering include the 74LS241 (tri-state bus drivers) and 74LS245 (transceivers).

The control signals /MREQ, /IORQ, /RD, and /WR should be unidirectionally buffered in large- or multi-card systems. These signal buffers are, again, enabled when their associated device or card has bus control and are forced into high-impedance states when another master takes control of these bus lines. Since there are short intervals during transfer of the bus when the bus is not driven by any master, /MREQ, /IORQ, /RD, and /WR should be pulled up to + 5V with 2.7 kohms to 4.7 kohms resistors so that they remain inactive. Other control. signals on CPU and DMA may be permanently driven. This usually includes /M1, /RFSH, and /HALT from the CPU, and /BAO from a DMA.

The /BUSREQ line is bidirectional and cannot easily he externally buffered. However, the DMA can sink 3.2 mA on /BUSREQ, more than on other signals. To maximize current, the system's /BUSREQ pullup resistor can be as low as 1. 8 kohms.

An interesting and somewhat unfortunate situation exists with respect to ratings of the ability of TTL buffers to drive capacitive loads. While the DC output ratings of standard buffers like the 74LS367 are usually ample, propagation times through these buffers are rated at capacitive loadings of only 30 pF, a value easily exceeded in practice. Capacitive loading thus usually dominates bus driving requirements (Z80 Family parts are specified over ranges of capacitive loading). The load seen by a device driving a bus line has components due to wiring and printed-circuit land capacitance, connector capacitance, and capacitances of inputs and outputs connected to the signal. A standard low-power Schottky (LS) TTI input presents about 6 pF of capacitive load, an LS output of about 8 pF. Most other input and output capacitances can be estimated from device data sheets, but capacitance associated with interconnection may vary markedly. Sometimes, propagation delays and allowable capacitive loading for buffered lines must be, determined by measurement or by trial and error.

C

6.0 Z80 DMA AND CPU (Continued)

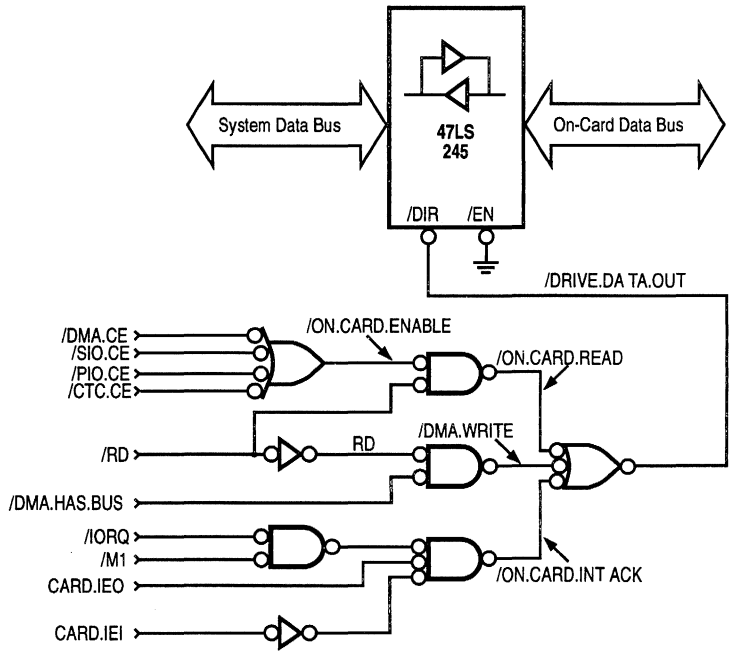


Figure 6-7. Data Bus Buffer Control Example

6.1 Z80 DMA AND Z80 SIO EXAMPLE

A common application of the DMA is to handle data transfers over a serial data link. The Z80 SIO peripheral is Used to interface to the link, providing conversion between serial and parallel data formats, synchronization, and other functions.

In this case, comparing the efficiency of interrupt-driven and DMA data transfers requires examination of the, event sequences during the brief time, intervals when the SIO needs a character (byte) transfer. Most of the time, of course, the SIO is busy transmitting or receiving message bits and requires no service.

The SIO must be programmed to drive its /WAIT//RDY line as a /RDY signal to the DMA, which is programmed for active-Low /RDY in Byte mode.

The event sequences for SIO-DMA transfers are shown in Tables 6-1 and 6-2.

Table 6-1. Receive Event Sequence

Event	Inter-event delay (clock periods)	
SIO receives last bit of character	10-13	latency
SIO /RDY becomes active	2	latency
DMA asserts /BUSREQ	1-5	latency
Current CPU machine cycle ends	1	latency, bus occupancy
CPU asserts /BUSACK	4	latency, bus occupancy
DMA I/O read cycle begins	4	latency, bus occupancy
DMA memory write cycle begins	2	bus occupancy
DMA terminates /BUSREQ	1	bus occupancy
DMA memory write cycle ends	1	bus occupancy
CPU terminates /BUSACK and regains control of bus	1	bus occpuancy

Note: Latency (delay from reception of final data bit to reading of received data) is 22 to 29 clock periods. The system bus is occupied by the DMA for 13 clock periods per byte transferred.

C

Table 6-2. Transmit Event Sequence

Event	Inter-event delay (clock periods)	
SIO transmits last bit of character	5-6	latency
SIO /RDY becomes true	2	latency
DMA asserts /BUSREQ	1-5	latency
Current CPU machine cycle ends	1	latency, bus occupancy
CPU asserts /BUSACK	4	latency, bus occupancy
DMA memory read cycle begins	3	latency, bus occupancy
DMA I/O write cycle begins	3	latency, bus occupancy
DMA terminates /BUSREQ	1	latency, bus occupancy
DMA I/O write cycle ends	1	latency, bus occupancy
CPU terminates /BUSACK and regains control of bus	1	bus occupancy

Note: Latency (delay from transmission of final data bit to loading of another character) is 20 to 28 clock periods. The system bus is occupied by the DMA for 13 clock periods per byte transferred.

In an interrupt-driven CPU transfer scheme, the SIO must interrupt the CPU whenever it has received a character or needs another character to transmit. A very short benchmark service routine, which presumes the exclusive use of the Z80 CPU's alternate register set for SIO interrupt handling, is provided (numbers in parentheses are clock periods per instruction):

```
SIOSVC:
        EXX             ; get transfer parameters       (4)
        OUTI            ; transfer a byte,
                        ; update parameters             (16)
        JRZ,BLKEND      ; test for end-of-block         (7)
        EXX             ; save parameters               (4)
        EI              ; reenable interrupts           (4)
        RETI                                            (14)
```

6.1. Z80 DMA AND SIO EXAMPLE (Continued)

Before the service routine can be executed, the CPU must have its interrupts enabled, finish its current instruction, and execute an interrupt acknowledge cycle (19 clock periods). This optimistic benchmark takes at least 68 clock periods per byte transferred, and severely restricts CPU activity by permanently occupying the alternate register set.

To compare these transfer methods, the ratios of clock cycles used per Kbaud to clock cycles available per second can be calculated. These represent the fractional reductions in CPU throughput per Kbaud transferred.

	Z80 (2.5 MHz)	Z80A (4 MHz)
DMA sequential transfer DMA sequential transfer/search	0.065%	0.041%
Interrupt-driven transfer	0.340%	0.213%

Thus, DMA has a shorter and more predictable latency period and decreases system overhead by at least a factor of five in this conservative example.

A diagram of a typical Z80 system using a Z80 CPU, a Z80 CTC for asynchronous baud rate generation, both channels of a Z80 SIO, and two Z80 DMAs (one for each serial channel) appears in Figure 6-8. The diagram omits the system memory (ROM and RAM), bus buffers (as required), and chip enable decoders, which are described above.

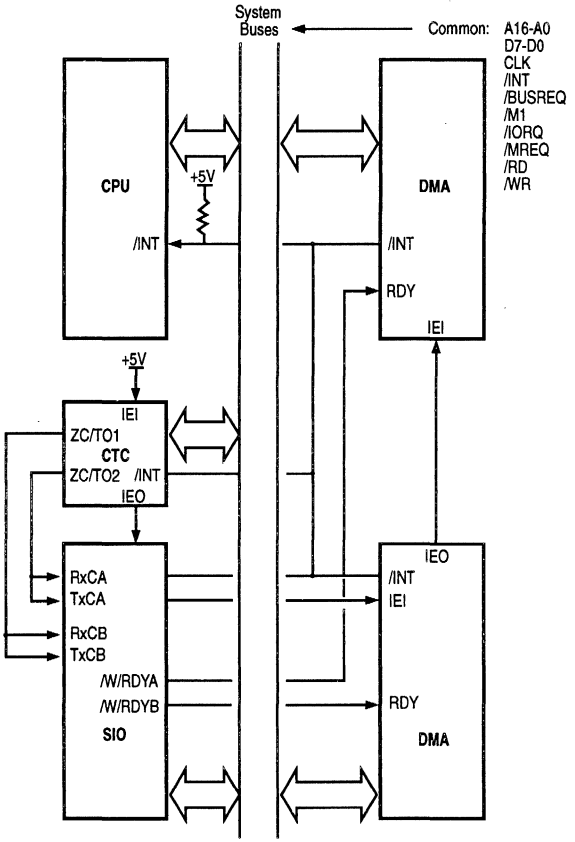


Figure 6-8. Z80 DMA-SIO Environment

6.2 USING THE Z80 DMA WITH OTHER PROCESSORS

Because it is so versatile, designers of computer systems using other CPUs may want to use the Z80 DMA in their applications Since the DMA was designed as a member of the Z80 Family, it requires certain signals and bus characteristics like those of the Z80 bus in order to function well. Three main groups of requirements are distinguished as follows:

- Bus request/release mechanisms
- Bus characteristics
- Interrupt request, acknowledge, and return

These topics are described in this section, and suggestions for design are given. It is, of course, impractical to describe all the possible combinations in detail, so each designer must invoke some creativity to come up with a complete, workable design.

6.2.1 Bus Request/Release Mechanisms

Probably the most fundamental characteristic that distinguishes the Z80 DMA from other monolithic DMACs is its full control of the system bus during its active state. An immediate consequence is that processors using the DMA must be able to give up control of the system bus, including address, data, and the control lines /MREQ, /IORQ, /RD, and /WR (or their equivalents). Some processors have no mechanism for freeing the bus. Others, including the 6800 and its relatives, have rudimentary bus control facilities, but due to their internal dynamic logic implementations, cannot relinquish control for indefinite periods of time. This makes them difficult to interface to the DMA.

Many popular microprocessor CPUs, however, do have adequate bus control facilities-some are very similar to the Z80 /BUSREQ and /BUSACK signals. For instance, the 8080, 8085, and 8086 signals HOLD and HLDA are very close approximations.

The active levels of HOLD and HLDA are positive rather than negative, and variations exist in timing, but the use of HOLD and HLDA does allow the address and data bus drivers to be put into their high-impedance states. In 8080 systems using an 8238 to demultiplex commands, the /MEMW, /MEMR, /IOW, and /IOR control lines can be floated using the /BUSEN input. With the 8085, a tri-state decoder can be used to decode or disable corresponding signals. The 8086 and its support chips also tri-state their control signals when HLDA is active.

The Zilog Z8000 Family will include a completely compatible DMAC, but until it is available, the Z80 peripheral chip family can be used with the Z8000. Generation of Z80-like control signals for Z80 peripherals is discussed in the Zilog Application Note, "A Small Z8000 System," although the Z80 DMA requires special consideration beyond that of other Z80 peripherals.

6.2.2 Bus Characteristics

Like the Z80, the 8080 and 8085 have 8-bit data paths and 16-bit addresses. The DMA is matched well to these numbers—it can search whole data words and directly address any byte in the memory.

The 8086 and the Z8000 CPUs have 16-bit data paths and larger address spaces, thus making it somewhat harder to use the Z80 DMA. Searching can be done for match bytes in either half of the data word, but not for a whole unique word. Often this is not a problem since byte matches suffice, for example, in detecting special ASCII characters in a data block, The problem of handling larger address spaces can be handled by using an external segment or page register, latched to the appropriate high-order addresses before the DMA becomes bus master, or by other schemes such as indexing. This, of course, requires some external hardware.

In order to conserve pins, the 8085, 8086, Z8001, and Z8002 multiplex addresses and data. Strobes allowing demultiplexing then become part of the bus structure and must be accounted for in DMA interface. In such cases, the DMA should be connected to the demultiplexed address and data lines rather than closer to the processor itself. Figure 6-9 gives a simplified diagram of this idea.

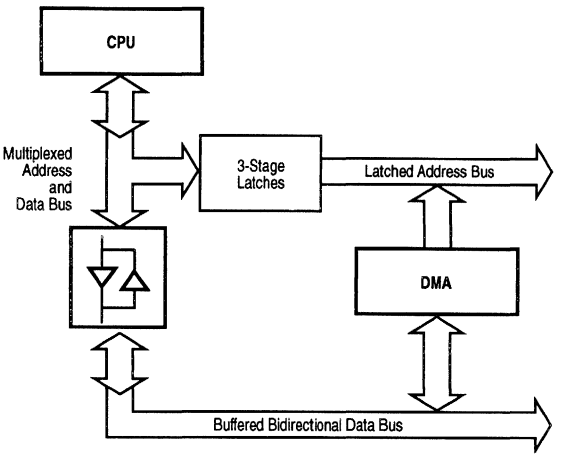


Figure 6-9. Connecting DMA to Demultiplexed Address/Data Buses

Many processors encode their control signals, analogous to the Z80's /M1, /MREQ, /IORQ, /RD, and /WR, into status words which are often demultiplexed before they are distributed to memory, peripherals, etc. Again, it is better to link the DMA to these demultiplexed signals, taking advantage of tri-state decoders to float the outputs when the DMA is master.

The DMA's Z80-like control signals probably need to be retimed to meet the requirements of the foreign buses. But the programmable timing feature of the DMA may well reduce the hardware costs incurred.

6.2.3 Interrupt Request, Acknowledge, and Return

This is, in may ways, the thorniest issue faced in using the DMA with other processors. The ways of signaling, prioritizing, identifying, responding to, and returning from interrupts are multitudinous in their profusion. Non-Z80 interrupt environments do not use the IEI and IEO signals, often use separate interrupt controllers to generate vectors, and handle acknowledgement and return in different ways (or not at all).

Interrupt request is usually easy enough: active levels typically are low voltage, and there may he one or more separate interrupt request pins. Timing requirements for interrupt requests vary (including pulse widths, latching, etc.) and should be examined for each case.

6.2 USING THE Z80 DMA WITH OTHER PROCESSORS (Continued)

Priority of simultaneous or overlapping requests is handled in several ways: some processors (e.g., the 8085) have multiple interrupt-request pins, some use daisy-chained priority schemes (as in the Z80), and there are several kinds of interrupt control ICs available.

Acknowledgement and identification methods vary, too. Sometimes, several fixed memory locations correspond to different interrupt pins' service routines. In other cases, the interrupting device is responsible for identifying itself by putting a vector or instruction on the data bus for the CPU to read. Interrupt controllers often provide appropriate vectors to the CPU and distinguish between and prioritize multiple requests. The DMA has the built in capability to supply an arbitrary vector byte when it detects a Z80 interrupt acknowledge (/IORQ and /M1 both active) and its IEI input is active (no higher-priority device is interrupting). Often, then, gating the /M1, /IORQ, and IEI pins appropriately can obviate use of a separate interrupt controller. /IORQ serves another function, too, so it must appear during CPU-DMA transfers and be available to signal I/O reads or writes in the active state.

At the end of its service routine, the DMA expects to see the CPU fetch the RETI instruction (ED, 4D appear on the data bus accompanied by /M1). The DMA command, RESET AND DISABLE INTERRUPTS, is designed for this purpose in non-Z80 CPU environments. Alternatively, the RETI instruction might be simulated by regating /M1 and programming the CPU to write to a phantom peripheral the bytes ED, 4D. The "chip select" for this nonexistent peripheral is used to simulate /M1 at these times (Figure 6-10).

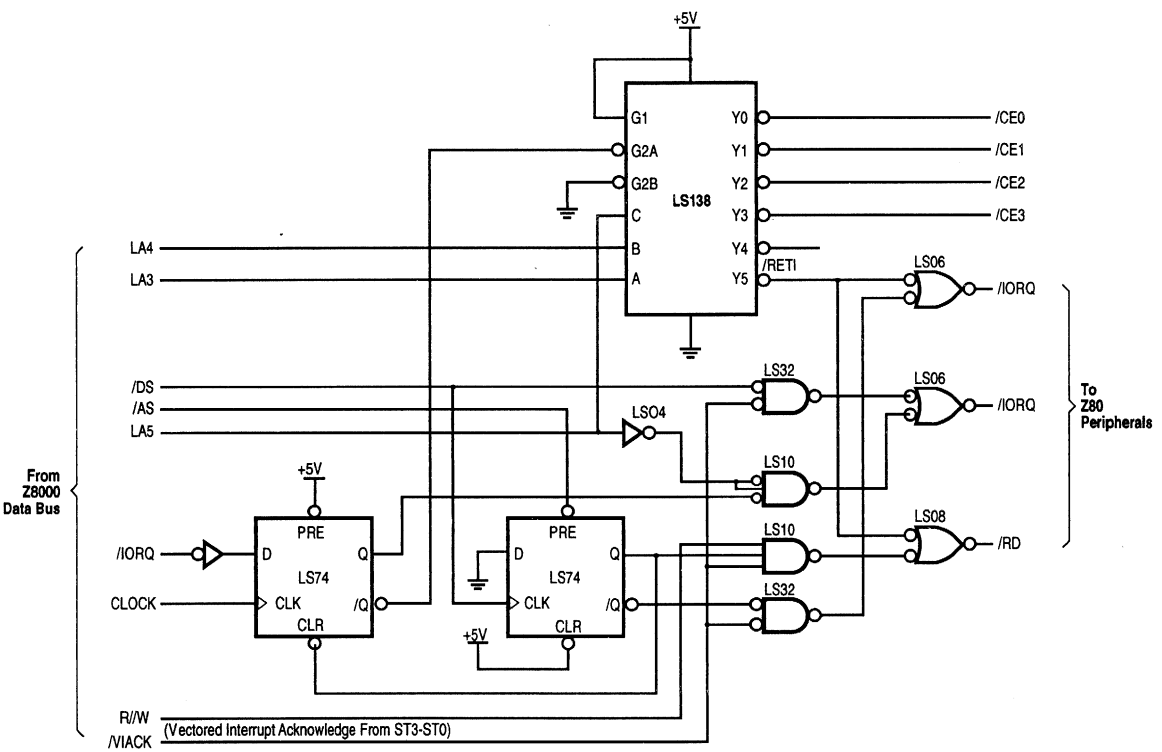


Figure 6-10. Z8000/Z80 Peripheral Interface

CHAPTER 7
PERFORMANCE LIMITATIONS

7.0 BUS CONTENTION

The principal limitation to the use of the DMA is its impact on CPU activity. When the DMA operates, it is bus master, thereby preventing the CPU from fetching and executing instructions. Bringing the CPU to a hall in this manner can create several problems, including:

- No interrupt servicing (including nonmaskable CPU interrupts)
- No refresh for dynamic memory (if accomplished by the CPU)
- No polling

The degree to which time-critical functions of the CPU are affected when the DMA is operating varies with the DMA's operating mode.

7.0.1 Byte Mode

This is the most desirable mode when bus contention is a problem, since it allows interleaving of CPU functions and DMA functions for each byte of data transferred. The disadvantage is slower transfer speed.

7.0.2 Burst Mode

This may be useful if the data to be transferred is distributed over time in a manner that causes the DMA to release the bus back to the CPU before other CPU-dependent functions are endangered. The Burst mode has the merit of using the bus only when it is needed and of maximizing transfer speed during that time. It may not be usable, however, if very long bursts of data (long periods when the Ready line is active) are possible.

7.0.3 Continuous Mode

This mode is the ultimate bus hog since it holds onto the bus until an end-of-block or byte match, no matter what the state of the Ready line. While it achieves the fastest transfer speeds, it can only be used when there are no time-critical functions dependent upon the CPU or when the blocks are relatively short.

The Byte mode is normally safe for most applications. What must be known to calculate the effect of using the Burst or Continuous modes is the following:

- Maximum block length
- Maximum DMA transfer rate (see Table 2-1)
- Maximum time Ready line will remain active

There is a method of forcing the DMA off the bus in Byte or Burst mode. This method uses an external gate to remove the RDY input to the DMA. Figure 7-1 illustrates this. The negative consequences of forcing the DMA to stop in the middle of a transfer must be considered when contemplating such a scheme. This method cannot he used if the DMA is operating in the Continuous mode; only a power-down or normal termination with end-of-block or byte match can make the DMA release the bus.

Figure 7-1. DMA Bus-Master Gate (Byte or Burst Modes Only)

7.1 CONTROL OVERHEAD

The software overhead incurred by the CPU to initialize and update the DMA's program may also limit the degree to which the DMA contributes to overall system efficiency. Table 5-2 shows that a maximum of about 35 control bytes would be required to initialize the DMA if all functions of the DMA were used to their fullest extent. In addition, use of the Interrupt mode requires servicing by the CPU and this normally includes additional control bytes written to the DMA.

So, the increase in system throughput is not as great for applications which require frequent reprogramming of the DMA or extensive interrupt service of data-independent DMA functions. The ratio of overhead incurred to number of bytes transferred is minimized for repetitive transfers of large blocks.

CHAPTER 8
TIMING

8.0 WHEN THE CPU IS BUS MASTER

8.0.1 Writing Control Bytes

The DMA can be programmed with control bytes whenever the CPU is the bus master. Table 5-1 describes this is the "disabled," "enabled/inactive," or "enabled/stopped" States (the latter two are equivalent).

The DMA is programmed by addressing it as an I/O peripheral in a CPU output instruction (it can be addressed in the full 64K I/O space). To do this, three lines must simultaneously be active-low on the rising edge of the clock:

/CE	Chip Enable
/IORQ	Input/Output Request
/WR	Write

Figure 8-1 illustrates the timing required for this to happen. In a Z80 CPU environment, this timing happens automatically when the CPU and DMA are on the same board and have no buffers, drivers, or other external gates ill series with the common CPU and DMA pins. This applies to the sequential transfer, sequential transfer/search, and search-only classes of operation. It may or may not apply to the simultaneous transfer or simultaneous transfer/search operations, depending on the speed of the external devices used (see the "Applications" chapter).

The essential characteristics of gaining the DMA's attention for writing control bytes to it are the following:

- The DMA's /CE line must be Low (normally done by decoding the lower byte of the address bus).
- The /IORQ and /WR lines must be Low at this time.
- The control byte must be placed on the data bus so that it is stabilized at a rising clock edge which occurs one clock period after the /CE, /IORQ, and /WR lines have stabilized.

8.0.2 Reading Status Bytes.

Figure 8-2 illustrates the timing needed for the CPU to read any of the DMA's read registers, RR6 through RR0, while the CPU is bus master. The following condition must be met to read a register:

- The /CE, /IORQ, and /RD lines must be active and stabilized over two rising edges of the clock.

Status data becomes available on the data bus at the time of the second clock rising edge. It remains on the bus for as long as the /CE, /IORQ, and /RD lines remain simultaneously active.

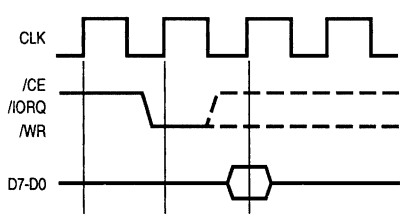


Figure 8-1. CPU-to-DMA Write Cycle Requirements

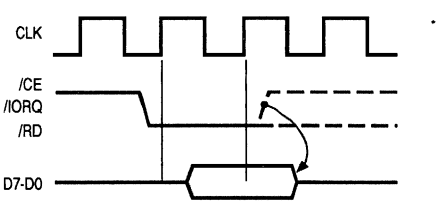


Figure 8-2. CPU-to-DMA Read Cycle Requirements

8.1 WHEN THE DMA IS BUS MASTER

8.1.1 Sequential Transfers

In sequential transfer and transfer/search operations (both have the same timing), data is latched onto the bus by the rising edge of the /RD signal (with standard timing this is the falling edge of T3). Data is held on the data bus across the boundary between read and write cycles, through the end of the following write cycle. The DMA data bus drivers become active after /RD has become inactive.

Figure 8-3 illustrates the timing for memory-to-I/O port transfers, and Figure 8-4 illustrates I/O-to-memory transfers. Memory-to-memory and I/O-to-I/O transfer timings are simply permutations of these diagrams.

The default timing uses three clock cycles for memory transactions and four clock cycles for I/O transactions, which include one automatically inserted wait cycle between T2 and T3. If the /CE/WAIT line is programmed to act as a /WAIT line during the DMA's active state, it is sampled on the falling edge of T2 for memory transactions and the falling edge of TW for I/O transactions. If /CE/WAIT is Low during this time another T-cycle is added, during which the /CE/WAIT line is again sampled. The duration of transactions can thus be indefinitely extended.

8.1.2 Simultaneous Transfers

The timing for simultaneous transfers and simultaneous transfer/ searches is the same. The DMA is programmed in the Search-Only mode, and both read and write cycles happen simultaneously in the time that a source-port read would occur in search-only. Only one address is generated on the address bus; the I/O port is hardwire-selected during this operation as shown in the "Applications" chapter. The /IORQ, /MREQ, /RD, and /WR lines are gated into two new signals by external logic. These signals are either:

/MEMWR (Memory write)
/IORD (I/O read)
or:
/MEMRD (Memory read)
/IOWR (I/O write)

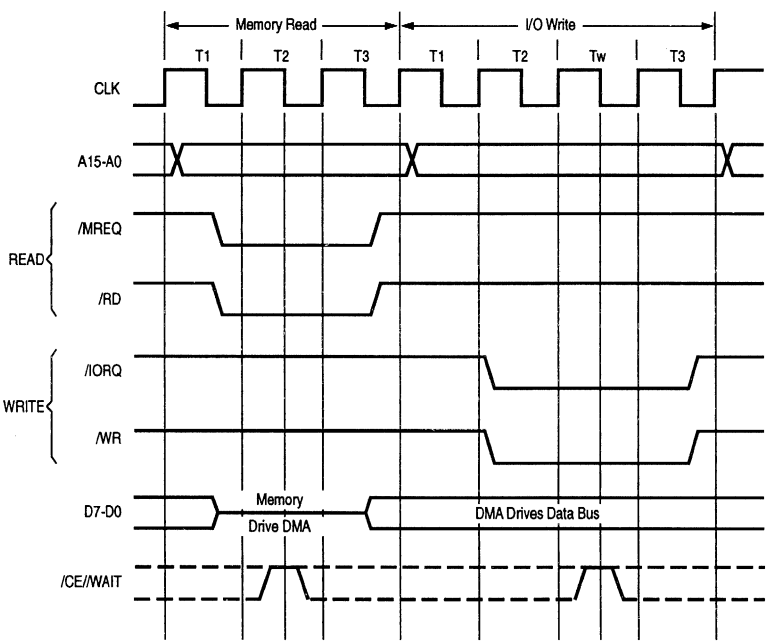


Figure 8-3. Sequential Memory-to-I/O Transfer, Standard Timing (Searching Is Optional)

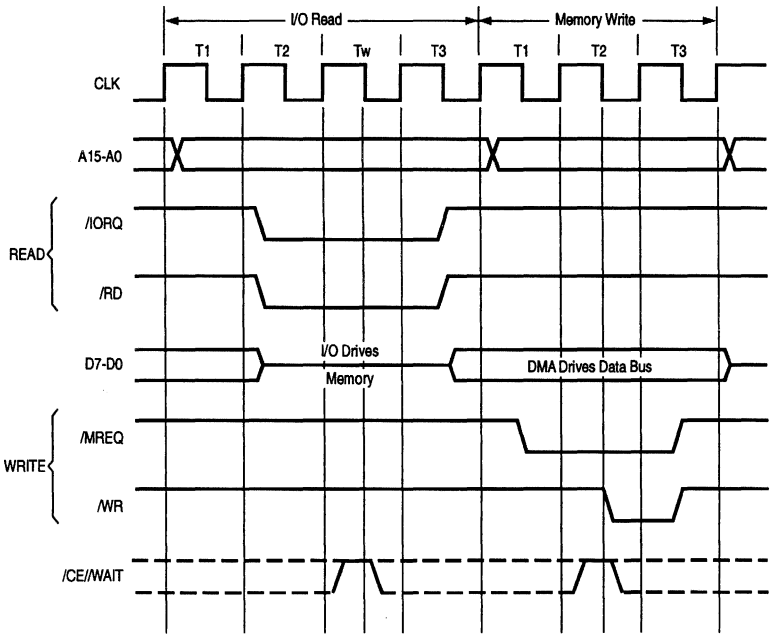


Figure 8-4. Sequential I/O-to-Memory Transfer, Standard Timing (Searching Is Optional)

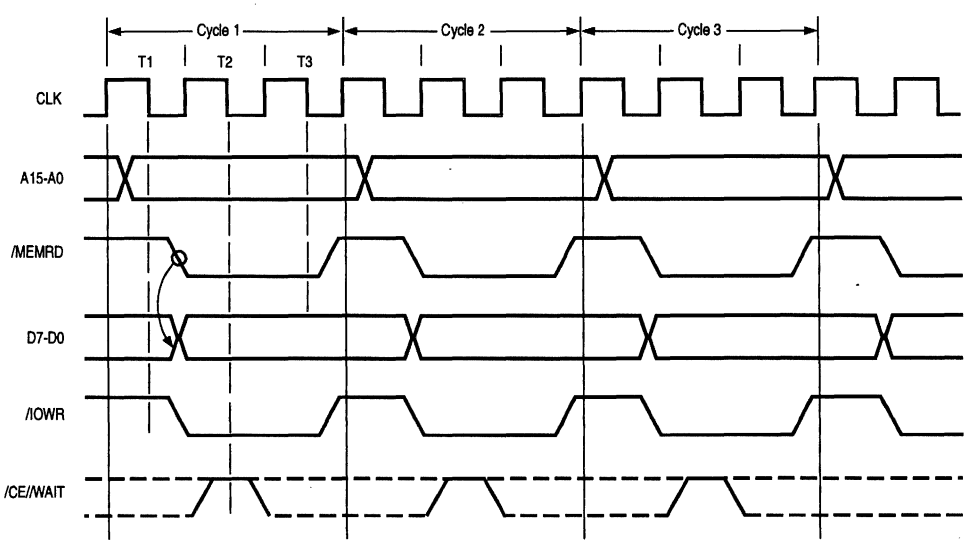


Figure 8-5. Simultaneous Memory-to-I/O Transfer (Burst and Continuous Mode)

8.1 WHEN DMA IS BUS MASTER (Continued)

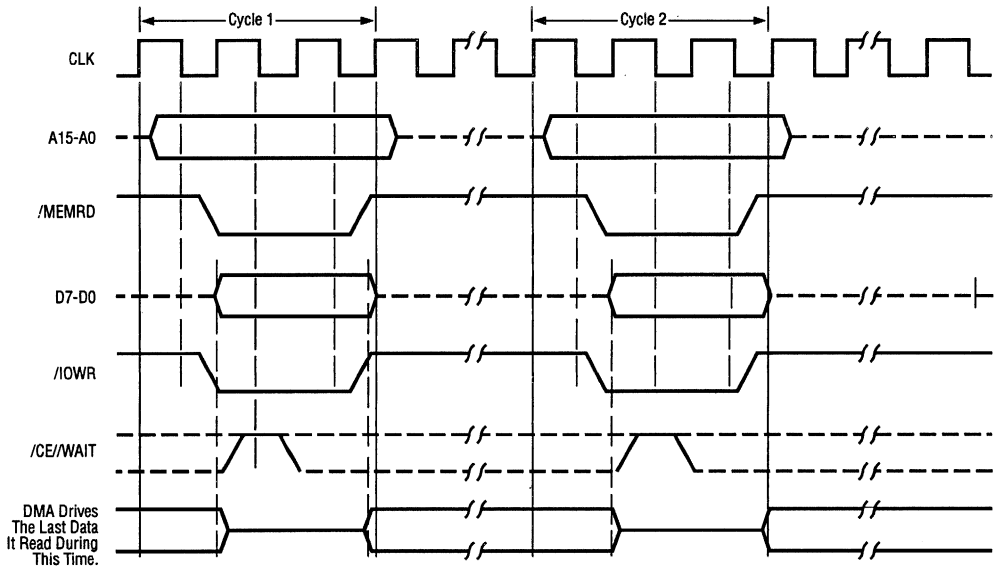


Figure 8-6. Simultaneous Memory-to-I/O Transfer (Byte Mode)

Figure 8-5 shows the timing for simultaneous transfers in Burst and Continuous modes between memory and I/O, using standard Z80 timing. The timing within each cycle is similar to the "memory read" cycle shown in Figure 8-3. The address bus activity is the same, and the cycle length is the same. However, the/MREQ, /RD, /IORQ, and /WR lines in Figure 8-3 have been changed to /MEMRD and /IOWR lines in Figure 8-5. In addition, the data bus comes active earlier in Figure 8-5, in response to the /MEMRD line coming active. Data is clocked into the I/O port on the rising edge of /IOWR.

Figure 8-6 shows the timing for Byte mode. It is the same as Figure 8-5 within each cycle. The breaks between each cycle, where the address and data bus are tri-stated and the /MEMRD and /IOWR lines remain inactive, are caused by the activity on the /BUSREQ and /BAI lines that is explained later.

8.1.3 Search-Only

The standard timing for search-only operations is identical to the read cycles of Figures 8-3 and 8-4. Search-only is equivalent to read-only; data is simply being read into a DMA register for comparison with the match byte.

8.1.4 Bus Requests.

Figure 8-7 illustrates the bus request and acceptance timing. The RDY line, which may be programmed active High or Low, is sampled on every rising edge of CLK.

If it is found to be active, and if the bus is not in use by any other device, the following rising edge of CLK drives /BUSREQ Low. After receiving /BUSREQ the CPU acknowledges on its /BUSACK (which is connected to the DMA's /BAI input either directly or through a multiple-DMA daisy chain).

The CPU looks at its /BUSREQ input one clock cycle before the end of each CPU machine cycle. If it sees a request, it releases the bus at the end of that same machine cycle. The maximum time delay from the CPU receiving /BUSREQ to the response on its /BUSACK line is, therefore, one machine cycle plus slightly less than one clock cycle. The CPU tri-states all of its bus control lines (/M1 is not tri-stated) when it acknowledges on the /BUSACK line.

The RDY line, which has a specified setup time with respect to a rising edge of CLK, must remain active until after the DMA becomes bus master in Byte or Burst modes.

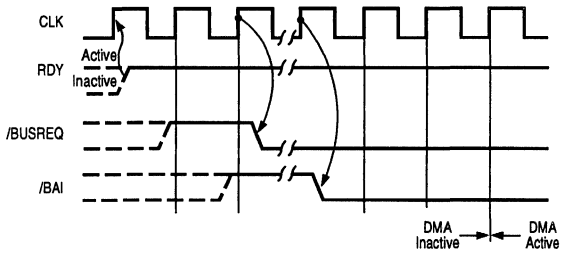


Figure 8-7. Bus Request and Acceptance Timing

(RDY is detected as a level, not an edge.) The only situation in which a pulse on RDY can be used to allow the DMA to become bus master is in the Continuous mode; in this event, the DMA becomes bus master but does not begin operations.

When the DMA detects a Low on /BAI for two consecutive rising edges of CLK, the DMA begins transferring data on the next rising edge of CLK.

In Byte mode, after each byte is transferred, the DMA waits until its /BAI line goes inactive before requesting the bus again on /BUSREQ for the next byte transfer. This allows a minimum of one CPU machine cycle to occur between each byte transferred.

8.1.4 Bus Release Byte-at-a-Time

In Byte mode, /BUSREQ is brought High on the rising edge of CLK prior to the end of each read cycle (search-only) or write cycle (transfer and transfer/search) as illustrated in Figure 8-8. This occurs regardless of the state of RDY. There is no possibility of confusion when a Z80 CPU is used since the CPU cannot begin an operation until the following clock cycle. Nor does this bother most other CPUs, although note should be taken of it. The effect of this is to decrease the time needed for a byte transfer by one clock cycle.

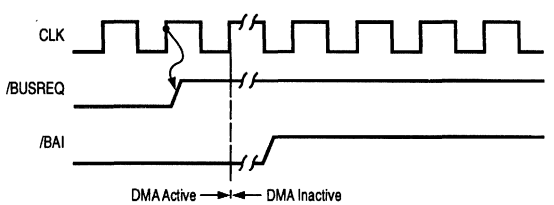


Figure 8-8. Bus Release in Byte Mode

The next bus request for the next byte comes after both /BUSREQ and /BAI have returned High. In a Z80 environment, /BAI returns High one clock cycle after /BUSREQ returns High.

8.1.5 Bus Release on End-of-Block.

If the DMA is programmed to stop on end-of-block in Burst or Continuous modes, an end-of-block causes to go High (inactive) on the same rising edge of CLK in which the DMA completes the transfer of the data block (see Figure 8-9). The last byte in the block is transferred even if RDY goes inactive before completion of the last byte operation.

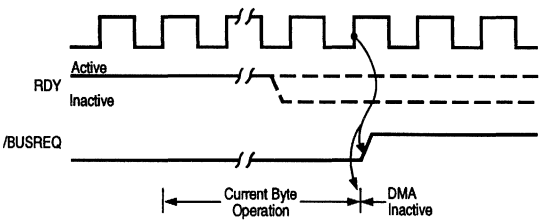


Figure 8-9. Bus Release on End-of-Block (Burst and Continuous Modes)

8.1.6 Bus Release on Match

If the DMA is programmed to stop (release the bus) on match in Burst or Continuous modes, a match causes /BUSREQ to go inactive on the next DMA operation, i.e., at the end of the next read in search-only or simultaneous transfer/searches or at the end of the following write in sequential transfer or transfer/searches (Figure 8-10).

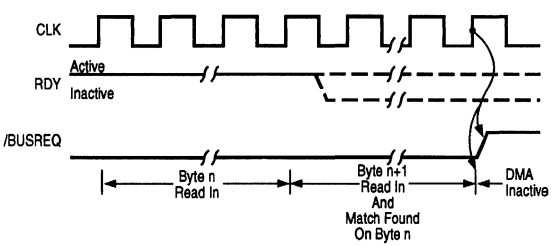


Figure 8-10. Bus Release on Match (Burst and Continuous Modes)

Due to the pipelining scheme, matches are determined while the next DMA read or write is being performed. Table 4-2 contains a complete reference to the number of bytes transferred in any class or mode.

C

8.1 WHEN DMA IS BUS MASTER (Continued)

The RDY line can go inactive after the matching operation begins without affecting this bus-release timing. However, the time at which RDY goes inactive can affect the number of bytes transferred, as shown in Table 4-1 and Figure 4-5.

8.1.7 Bus Release on Not Ready

In Burst mode, when RDY goes inactive it causes /BUSREQ to go High on the next rising edge of CLK after the completion of its current byte operation, i.e., at the end of the current read in search-only or simultaneous transfer/search or at the end of the following write in sequential transfer/search (Figure 8-11). The action on /BUSREQ is thus somewhat delayed from action on the RDY line. The DMA always completes it current byte operation in an orderly fashion before releasing the bus.

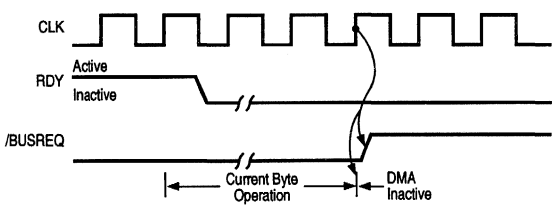


Figure 8-11. Bus Release on Not Ready (Burst Mode)

By contrast, /BUSREQ is not released in Continuous mode when RDY goes inactive. Instead, the DMA idles after completing the current byte operation, awaiting an active RDY again.

Figures 8-12, 8-13, and 8-14 review the relationship between the Ready line going inactive and the state of the other lines for each mode of operation, assuming a search-only of memory using standard Z80 timing. (The timing for Ready coming active is discussed under "Bus Request.") RDY is sampled on the rising edge of CLK in the last clock cycle of each read or write cycle. It is a level-sample, not an edge-sample. RDY can go inactive prior to the completion of the last byte operation without disturbing that operation. At the end of that operation, the /BUSREQ and /BAI lines go High in Byte or Burst mode according to Figures 8-10 and 8-13. The bus control lines /MREQ, /IORQ, /RD, /WR) also remain High in Byte and Burst mode during an inactive RDY, with both the address and data buses tri-stated.

The Continuous mode (Figure 8-14) differs in that the address bus holds the preincremented address for the next byte throughout the time that RDY is inactive. This address is immediately available when RDY comes active again.

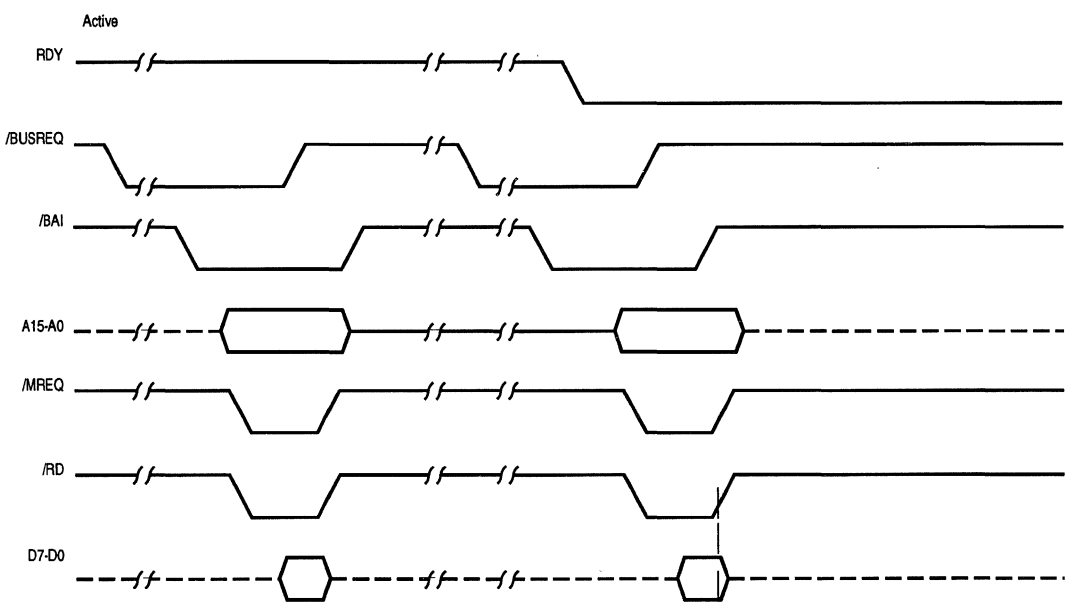


Figure 8-12. RDY Line in Byte Mode

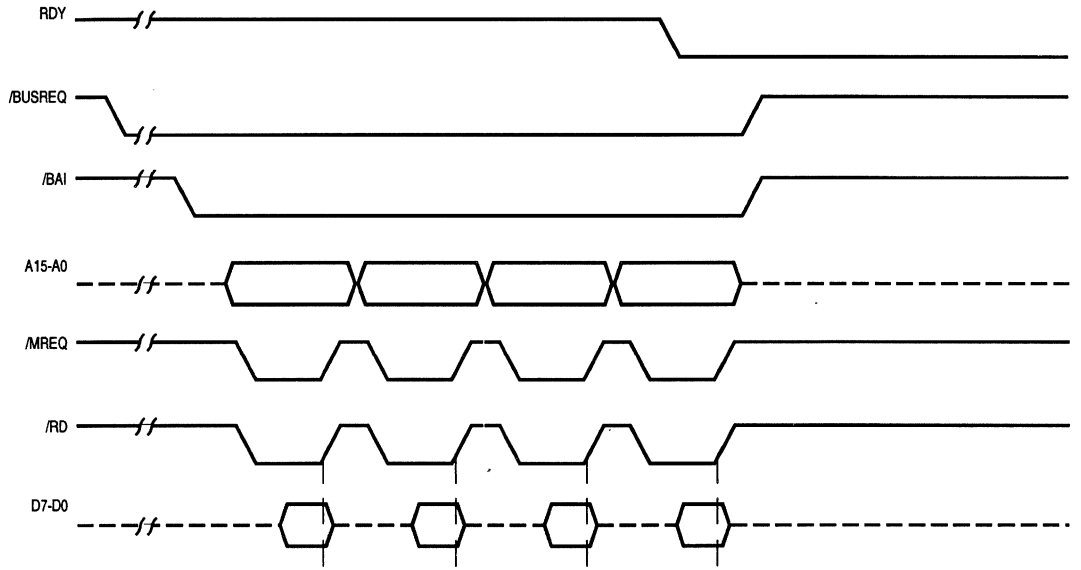


Figure 8-13. RDY Line in Burst Mode

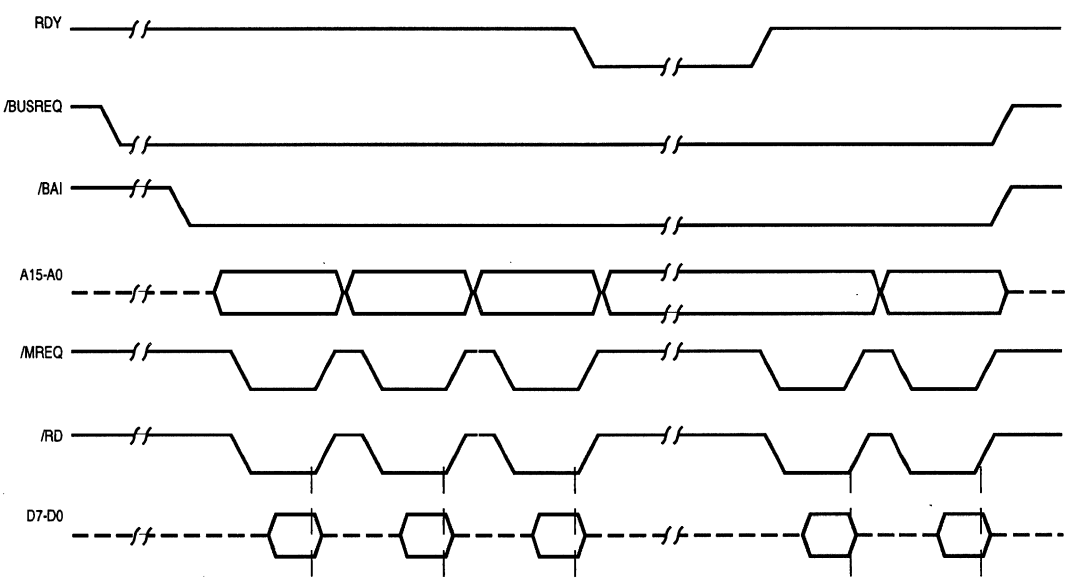


Figure 8-14. RDY Line in Continuous Mode

8.1 WHEN DMA IS BUS MASTER (Continued)

8.1.8 Variable Cycle and Edge Timing

The Z80 DMA's operation-cycle length, without Wait states, for the source (read) port and destination (write) port can be independently programmed. This variable-cycle feature allows read or write cycles consisting of two, three, or four clock cycles (more if Wait cycles are inserted), thereby increasing or decreasing the pulse widths of all signals generated by the DMA. In addition, the trailing edges of the /IORQ, /MREQ, /RD, and /WR signals can be independently terminated one-half cycle early. Figure 8-15 illustrates this.

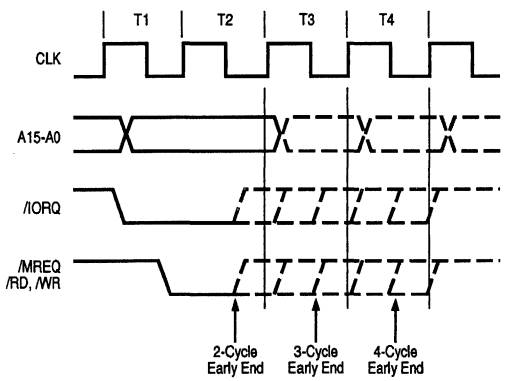


Figure 8-15. Variable-Cycle and Edge Timing

In the Variable-Cycle mode, unlike default timing, /IORQ comes active one-half cycle before /MREQ, /RD, and /WR. /CE/WAIT can be used to extend only the 3 or 4 clock-cycle variable memory cycles and only the 4-cycle variable I/O cycle (see Figure 8-16). The /CE/WAIT line is sampled at the falling edge of T2 for 3- or-4-cycle memory operations, and at the falling edge of T3 for 4-cycle I/O operations. The line is not sampled for 2-cycle operations. During transfers, data is latched on the clock edge causing the rising edge of /RD and held until the end of the write cycle.

A special case arises when using variable timing on an I/O-search or a simultaneous transfer or transfer/search with I/O as the source port. (The simultaneous transfers are actually programmed in the DMA as searches and only distinguished from searches by the manner in which external logic handles the bus control signals.) In these applications, the /IORQ line must be programmed to have an early ending (refer to WR1 description on page 5-4).

Figure 8-14 shows the bus control lines (/MREQ and /RD) remaining inactive when the RDY line goes inactive in Continuous mode. The same is not true of the /IORQ line when variable timing is used. In this case, /IORQ and any functions created from it by external logic in simultaneous transfer operations (such as /IOWR and /IORD) remain active during an inactive RDY line before stopping on end-of-block or byte match.

8.1.9 Interrupts

Timings for interrupt acknowledge and return from interrupt are the same as timings for these in other Z80 peripherals. Figure 8-17 illustrates this timing. The interrupt signal (/INT) is sampled by the CPU on the rising edge of the final clock cycle of any instruction. The signal is not accepted if the internal CPU software-controlled interrupt-enable flip-flop is not set or if the /BUSREQ signal is active. When the /INT signal is accepted, a special /M1 cycle is generated.

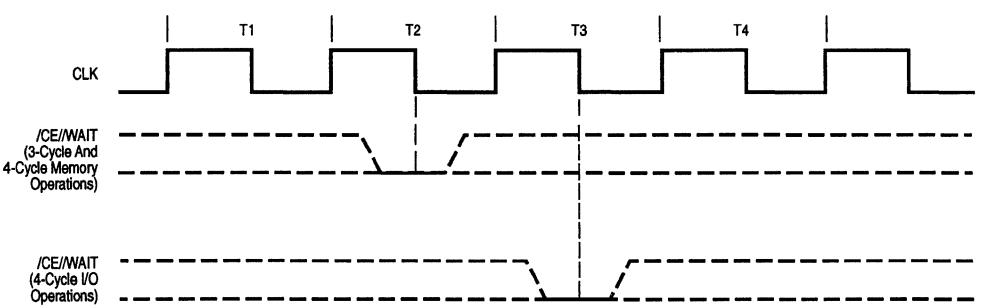


Figure 8-16. /WAIT Line Sampling in Variable-Cycle Timing

During this special /M1 cycle, the /IORQ signal becomes simultaneously active (instead of the normal /MREQ) to indicate that the interrupting device can place its 8-bit vector on the data bus. Two wait states are automatically added to this cycle. These states are added so that a ripple priority interrupt scheme can be easily implemented. The two wait states allow time for the ripple signals to stabilize and identify which I/O device must respond. Refer to Zilog Application Note 03-0041-01 (The Z80 Family Program Interrupt Structure) and to the Z80 CPU Technical Manual for more details.

Interrupt on RDY (interrupt before requesting the bus) does not directly affect the /BUSREQ line. Instead, the interrupt service routine may handle this by issuing the following commands to WR6:

- Enable after Return From Interrupt (RETI) Command—Hex B7
- An RETI instruction that resets the Interrupt Under Service (IUS) latch in the Z80 DMA—Hex ED, 4D

8.1.10 Pulse Generation

When the pulse generation option is selected, the /INT line is driven Low every 256 bytes after the offset value. The line goes Low during the DMA cycle in which the pulse-control byte matches the lower byte of the byte counter, and it remains Low for one complete "transfer cycle." A transfer cycle is here defined as either a read cycle (search-only or simultaneous transfer operations) or a read plus a write cycle, where read and write cycles can be independently programmed for length through the variable-cycle option.

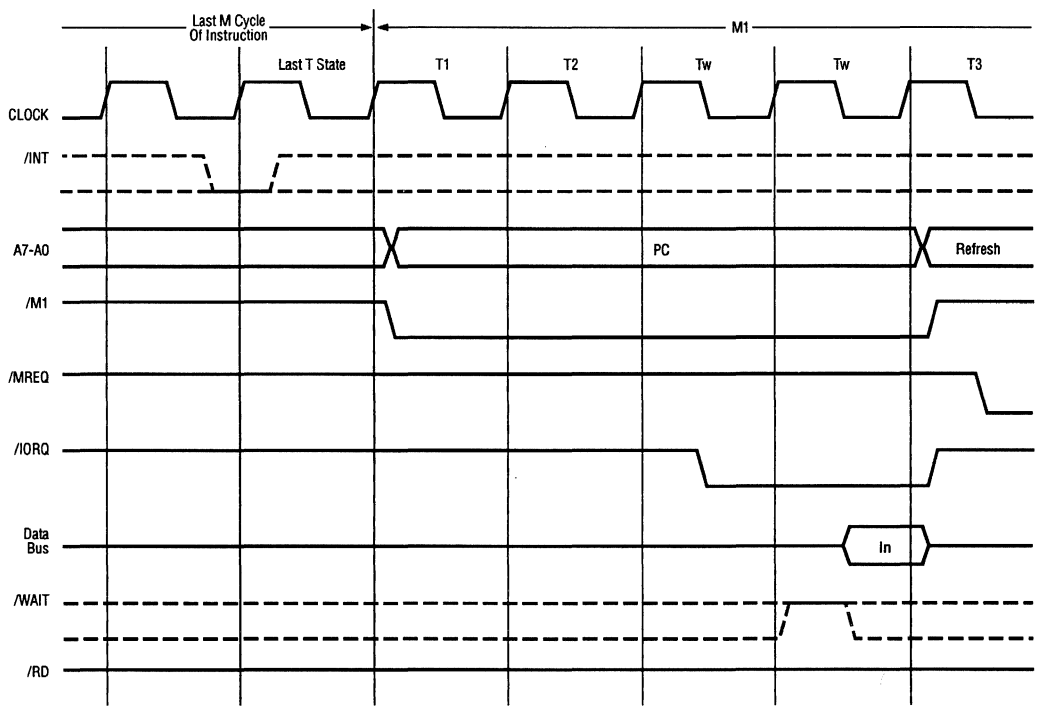


Figure 8-17. Interrupt Acknowledge

C

APPENDIX A

WRITE REGISTER BIT FUNCTIONS

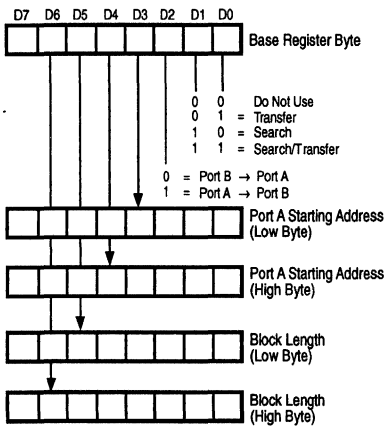


Figure A1. Write Register 0 Group

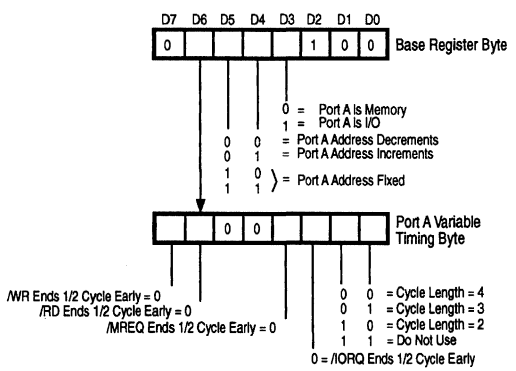


Figure A2. Write Register 1 Group

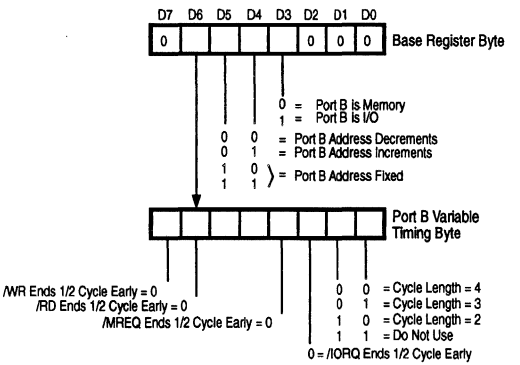


Figure A3. Write Register 2 Group

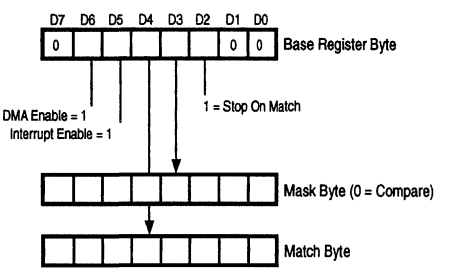


Figure A4. Write Register 3 Group

WRITE REGISTER BIT FUNCTIONS (Continued)

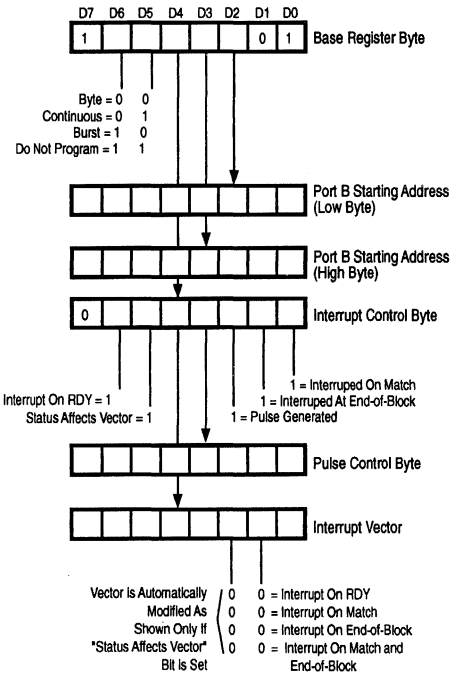


Figure A5. Write Register 4 Group

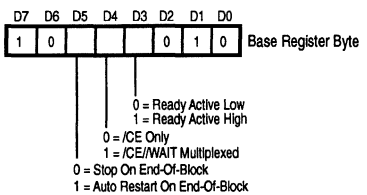


Figure A6. Write Register 5 Group

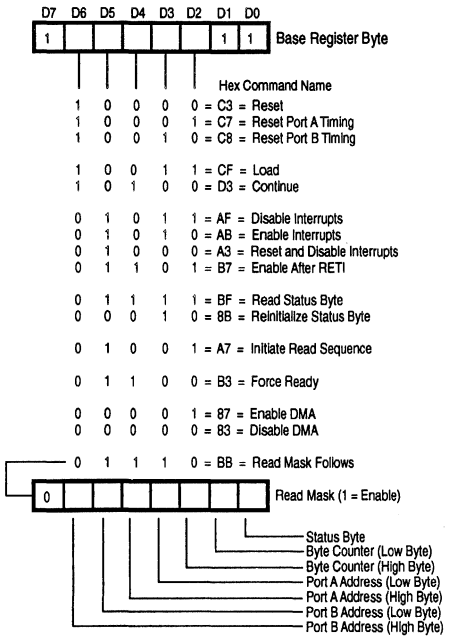


Figure A7. Write Register 6 Group

READ REGISTER BIT FUNCTIONS

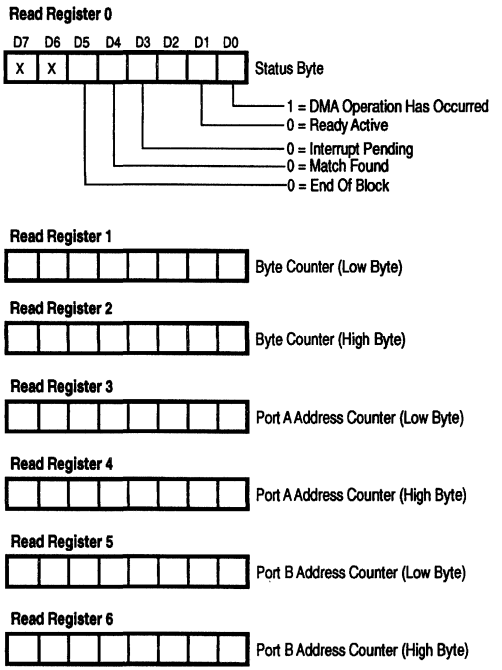


Figure A8. Read Register 0 Through 6 Bit Functions

GLOSSARY

DEFINITION OF TERMS

Active. (1) When the DMA is active, it is the bus master and is either operating or is suspended. (2) Signal lines are active when the circuitry recognizes them as such, i.e., when their voltage levels are either at logic 1 (High) or at logic 0 (Low) and the hardware. or software recognizes them as active in one or the other state. An active-High signal is written without a bar as in RDY; an active-Low signal is written with a bar as in /RDY.

Address Counters. Counts the source or destination addresses (two separate registers are provided). When a source address is variable, it increments or decrements immediately before the counted byte is read in. When a destination address is variable, it increments or decrements immediately before the counted byte is written out. Incrementing or decrementing does not occur for the first byte in an operation; the programmed starting address is used on the first byte.

Buffer. A means of (1) storing a logic state, (2) amplifying a signal, or (3) isolating a signal.

Burst Mode. A transfer or search mode ill which the DMA operates continuously as long as the I/O device's Ready line (or an internal Forced Ready condition) is active. If Ready becomes inactive, the DMA releases the bus.

Bus. The address bus, the data bus, the control bus, or all three buses (system bus). To say that the DMA can become "bus master" means that the DMA controls the address bus, the data bus, and the following bus control lines: /RD, /WR, /IORQ, /MREQ.

Bus Control. The DMA has control of the bus when its Bus Request (/BUSREQ) and Bus Acknowledge In (/BAI) lines are active simultaneously.

Byte Counter. Counts the number of bytes read. The counter begins at zero and increments after the counted byte has been read in. An end-of-block condition occurs when the contents of the byte counter equal the contents of the block-length register.

Byte Match. A match (or compare) between a data byte and the masked match byte.

Byte Mode. A transfer or search mode in which the DMA operates on only one byte before releasing the bus. The bus is then requested again for the operation on the next byte.

Channel. A controlled link between two ports that keeps track of the flow of data. The channel includes the address counters, byte counters and bus control logic.

Classes of Operation. Transfer, search, and transfer-while-searching (or transfer/search) classes. See also "Modes of Operation" and "Methods of Operation."

Clear. Set to logic 0.

CLK. System clock.

Clock Cycle. One cycle of the system clock (CLK), conventionally defined to begin and end on the positive-going (rising) edge. There is one T-cycle (time cycle) per clock cycle. In a 2.5 MHz clock, the clock cycle is 400 ns long (the inverse of the clock rate). In a 4 MHz clock, the clock cycle is 250 ns long.

Command. A control byte written to the DMA by the CPU. It usually causes more immediate action than other control bytes, which tend to be mode-setting for future activity. In some cases, mode setting is combined with immediate action in a single control byte.

Continuous Mode. A transfer or search mode in which the DMA completes a block transfer before releasing the system bus. If the I/O port's Ready line goes inactive during the transfer, the DMA pauses but retains control of the bus.

Control Byte. A byte written to the DMA while the CPU is bus master and is addressing the DMA as an I/O peripheral via a decoded Chip Enable (/CE) signal.

Destination Port. The port to which data is written in a transfer. Either Port A or Port B can be programmed as the destination port. Also see "Port."

DEFINITION OF TERMS (CONTINUED)

Disabled. The DMA is not able to request the system bus when it is in the disabled state. Any control byte written to the DMA, except the ENABLE DMA byte, disables the DMA.

DMAC. Direct Memory Access Controller (or chip).

Enabled. The DMA is able to request the system bus when it is enabled. It may also currently be the bus master in this state. The enabled state includes both the active and Inactive states

Flowthrough. See "Sequential 'Transfer."

Flyby. See ""Simultaneous Transfer."

High. Logic one (high voltage potential).

Inactive. (1) A subset of the DMA's enabled state. The DMA can request the bus when it is inactive. It becomes active when it becomes the bus master. (2) Signal lines are inactive when they are at the opposite logic level as their "active " state.

Interrupt Vector. An 8-bit byte passed to the CPU by a peripheral device after the CPU acknowledges the device's interrupt. In a Z80 CPU environment, the vector identifies the interrupting device and forms the low byte of the interrupt service routine's starting address. The CPU supplies the high byte of that address.

Land Capacitance. The capacitance, with respect to signal ground, of any part of the printed circuitry on a PC board.

Low. Logic 0 (low voltage potential).

M-Cycle. See "Machine Cycle."

Machine Cycle. A DMA machine cycle is the time required to do a read or write operation in sequential transfers. In simultaneous transfers, it is the time required to do both a read and write. A CPU machine cycle is a basic operation such as an an opcode fetch, memory read, or memory write (one instruction can achieve multiple machine cycles). DMA machine cycles may be 2, 3, or 4 clock cycles (T-cycles) in length, without wait states added. Z80 CPU machine cycles are between 3 and 10 clock cycles, unless wait states are added.

Methods of Operation. Sequential and simultaneous methods. These are also called flowthrough and flyby, respectively. See also "Modes Operation" and "Classes of Operation."

Modes of Operation. Byte, Burst, or Continuous modes. These are also called Single, Demand, and Block modes, respectively. See also "Classes of Operation" and "Methods of Operation."

Operating. When the DMA is operating, it is transferring and/or searching bytes of data. Control of status transfers between the CPU and the DMA do not constitute operation in this sense.

Port. A source or destination of data. Ports may be I/O peripherals or memory. The Z80 DMA generates addresses for both the source and destination port every time a byte of data is transferred. When data is only searched (no transfer), only a source port is used.

Race Condition. Multiple logic signal transitions which may give rise to different states of a machine depending upon their relative timing. The outcome of the race cannot always be predicted accurately.

Reset. Reinitialize to a default starting condition. This may contain logic 1s or 0s. See "Clear."

RR. Read Register. There are seven read registers that the CPU can read status bytes from when the DMA has relinquished the bus.

Sequential Transfer. A transfer in which bytes are read from the source port in one read cycle and then written to the destination port in a separate write cycle. Searches can also be performed concurrently in this class of operation, and transfers can be between any two I/O or memory ports. No external logic is needed but speed is only half as fast as in simultaneous transfers.

Set. Set to a starting condition. Also, often used to indicate setting to logic 1.

Simultaneous Transfer. A transfer in which bytes are simultaneously read from the source port and written to the destination port. Searches can also be performed concurrently in this class of operation, but at least one external logic gate is needed, and transfers are limited to memory-to-I/O or I/O-to-memory (no memory-to-memory or I/O-to-I/O). Speeds are twice as fast as in sequential transfers. Simultaneous transfers are implemented by programming the DMA for a search-only class of operation and using external logic to generate the appropriate control signals.

Source Port. The port from which data is read. Either Port A or Port B can be programmed as the source port. See also "Port."

Status Byte. Read register 0 (RR0).

Stop. The DMA releases the bus when it stops. This state terminates a DMA transfer and/or search.

Stop on Compare. Stop on byte match.

Suspended. In the suspended state, the DMA is the bus master but it is not currently operating (transferring and/or searching data).

System Bus. The combined address, data, and control buses.

T-Cycle. See "Clock Cycle."

WR. Write Register or Write line (/WR). There are 21 write registers that the CPU can write control bytes into, but access to them is gained through a subset of seven, which are named WR0 through WR6.

SYMBOLIC NOTATION

In addition to the terms defined in this glossary, the following symbolic notation is used in the manual:

Address Bus *A15-A0.* Parallel address lines 0 through 15.

Bar Notation */RDY.* Active-Low signal (i.e., active at Low voltage or logic 0). RDY: Active-High signal (i.e., active at High voltage or logic 1).

Bits in Data Byte *D7-D0.* Bits in a byte that are transmitted over the data bus.

Data Bus *D7-D0.* Parallel data lines 7 through 0.

C

TABLE OF CONTENTS

List of Figures

CHAPTER 1
INTRODUCTION

1.0 INTRODUCTION

The Z80 Parallel I/O (PIO) Circuit is a programmable, two port device which provides a TTL compatible interface between peripheral devices and the Z80-CPU. The CPU can configure the Z80-PIO to interface with a wide range of peripheral devices with no other external logic required. Typical peripheral devices that are fully compatible with the Z80-PIO include most keyboards, paper tape readers and punches, printers, PROM programmers, etc. The Z80-PIO is packaged in a 40-pin DIP, or a 44-pin PLCC, or a 44-pin QFP. NMOS and CMOS versions are also available. Major features of the Z80-PIO include:

One of the unique features of the Z80-PIO that separates it from other interface controllers is that all data transfer between the peripheral device and the CPU is accomplished under total interrupt control. The interrupt logic of the PIO permits full usage of the efficient interrupt capabilities of the Z80-CPU during I/O transfers. All logic necessary to implement a fully nested interrupt structure is included in the PIO so that additional circuits are not required. Another unique feature of the PIO is that it can be programmed to interrupt the CPU on the occurrence of specified status conditions in the peripheral device. For example, the PIO can be programmed to interrupt if any specified peripheral alarm conditions should occur. This interrupt capability reduces the amount of time that the processor must spend in polling peripheral status.

1.1 FEATURES

- Two Independent 8-Bit Bidirectional Peripheral Interface Ports with 'Handshake' Data Transfer Control
- Interrupt Driven 'Handshake' for Fast Response
- Any One of Four Distinct Modes of Operation May be Selected for a Port Including:
  - — Byte Output
  - — Byte Input
  - — Byte Bidirectional Bus (Available on Port A Only)
  - — Bit Control Mode
  - — All with Interrupt Controlled Handshake
- Daisy Chain Priority Interrupt Logic Included to Provide for Automatic Interrupt Vectoring Without External Logic
- Eight Outputs are Capable of Driving Darlington Transistors
- All Inputs and Outputs Fully TTL Compatible
- Single 5V Supply and Single Phase Clock are Required.

CHAPTER 2
PIO ARCHITECTURE

2.0 OVERVIEW

A block diagram of the Z80-PIO is shown in Figure 2-1. The internal structure of the Z80-PIO consists of a Z80-CPU bus interface, internal control logic, Port A I/O logic, Port B I/O logic, and interrupt control logic. The CPU bus interface logic allows the PIO to interface directly to the Z80-CPU with no other external logic. However, address decoders and/or line buffers may be required for large systems. The internal control logic synchronizes the CPU data bus to the peripheral device interfaces (Port A and Port B). The two I/O ports (A and B) are virtually identical and are used to interface directly to peripheral devices.

The Port I/O logic is composed of 6 registers with "handshake" control logic as shown in Figure 2-2. The registers include: an 8-bit data input register, an 8-bit data output register, a 2-bit mode control register, an 8-bit mask register, an 8-bit input/output select register, and a 2-bit mask control register.

The 2-bit mode control register is loaded by the CPU to select the desired operating mode (byte output, byte input, byte bidirectional bus, or bit control mode). All data transfer between the peripheral device and the CPU is achieved through the data input and data output registers. Data may be written into the output register by the CPU or read back to the CPU from the input register at any time. The handshake lines associated with each port are used to control the data transfer between the PIO and the peripheral device.

D

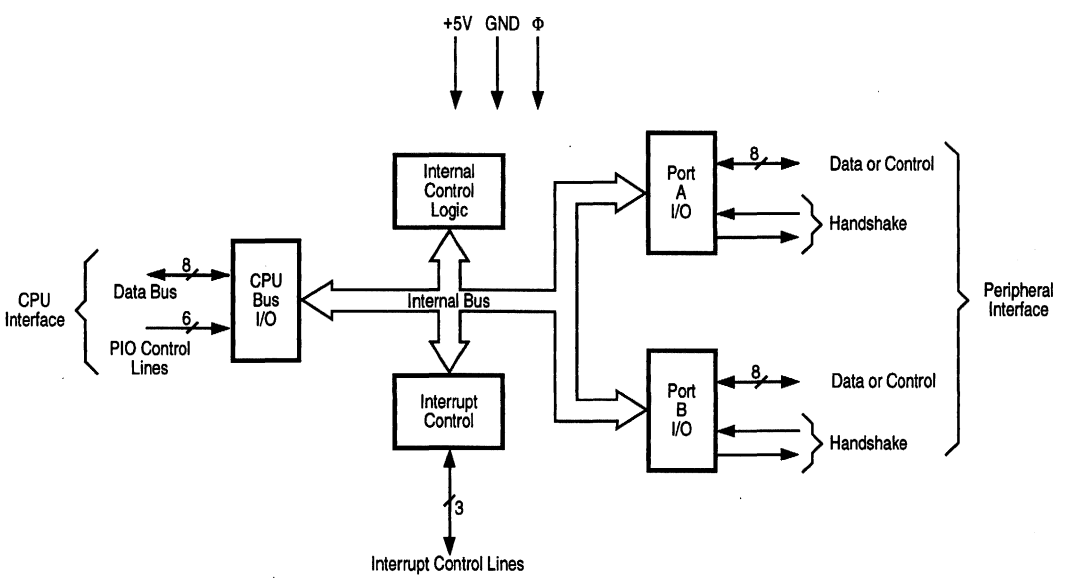


Figure 2-1. PIO Block Diagram

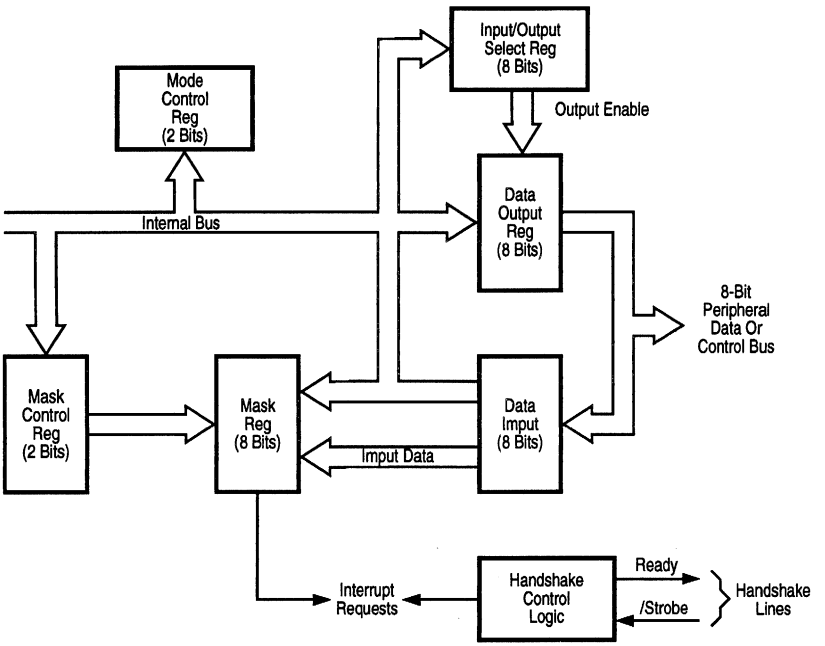


Figure 2-2. Port I/O Block Diagram

The 8-bit mask register and the 8-bit input/output select register are used only in the bit control mode. In this mode, any of the eight peripheral data or control bus pins can be programmed to be an input or an output as specified by the select register. The mask register is used in this mode in conjunction with a special interrupt feature. This feature allows an interrupt to be generated when any or all of the unmasked pins reach a specified state (either High or Low). The 2-bit mask control register specifies the active state desired High or Low) and if the interrupt should be generated when all unmasked pins are active (AND condition) or when any unmasked pin is active (OR condition). This feature reduces the requirement for CPU status checking of the peripheral by allowing an interrupt to be automatically generated on specific peripheral status conditions. For example, in a system with three alarm conditions, an interrupt may be generated if any one occurs or if all three occur.

The interrupt control logic section handles all CPU interrupt protocol for nested priority interrupt structures. The priority of any device is determined by its physical location in a daisy chain configuration. Two lines are provided in each PIO to form this daisy chain. The device closest to the CPU has the highest priority. Within a PIO, Port A interrupts have higher priority than those of Port B. In the byte input, byte output or bidirectional modes, an interrupt can be generated whenever a new byte transfer is requested by the peripheral. In the bit control mode an interrupt can be generated when the peripheral status matches a programmed value. The PIO provides for complete control of nested interrupts. That is, lower priority devices may not interrupt higher priority devices that have not had their interrupt service routine completed by the CPU. Higher priority devices may interrupt the servicing of lower priority devices.

When an interrupt is accepted by the CPU in Mode 2, the interrupting device must provide an 8-bit interrupt vector for the CPU. This vector is used to form a pointer to a location in the computer memory where the address of the interrupt service routine is located. The 8-bit vector from the interrupting device forms the least significant eight bits of the indirect pointer while the I Register in the CPU provides the most significant eight bits of the pointer. Each port (A and B) has an independent interrupt vector. The least significant bit of the vector is automatically set to a 0 within the PIO since the pointer must point to two adjacent memory locations for a complete 16-bit address.

The PIO decodes the RETI (Return from interrupt) instruction directly from the CPU data bus so that each PIO in the system knows at all times whether it is being serviced by the CPU interrupt service routine without any other communication with the CPU.

CHAPTER 3
PIN DESCRIPTION

3.0 PIN DESCRIPTION

A diagram of the Z80-PIO pin configuration is shown in Figure 3-1. This section describes the function of each pin.

D7-D0 *Z80-CPU Data Bus* (bidirectional, tri-state). This bus is used to transfer all data and commands between the Z80-CPU and the Z80-PIO. D0 is the least significant bit of the bus.

B/A Sel *Port B or A Select* (input, active High). This pin defines which port will be accessed during a data transfer between the Z80-CPU and the Z80-PIO. A Low level on this pin selects Port A while a High level selects Port B. Often, Address bit A0 from the CPU will be used for this selection function.

C/D Sel *Control or Data Select* (input, active High). This pin defines the type of data transfer to be performed between the CPU and the PIO. A High level on this pin during a CPU write to the PIO causes the Z80 data bus to be interpreted as a command for the port selected by the B/A Select line. A Low level on this pin means that he Z80 data bus is being used to transfer data between the CPU and the PIO. Often Address bit A1 from the CPU will be used for this function.

/CE *Chip Enable* (input, active Low). A Low level on this pin enables the PIO to accept command or data inputs from the CPU during a write cycle or to transmit data to the CPU during a read cycle. This signal is generally a decode of four I/O port numbers that encompass Ports A and B, data, and control.

Φ *System Clock* (input). The Z80-PIO uses the standard Z80 system clock to synchronize certain signals internally. This is a single phase clock.

/M1 *Machine Cycle One Signal from CPU* (input, active Low). This signal from the CPU is used as a sync pulse to control several internal PIO operations. When /M1 is active and the /RD signal is active, the Z80-CPU is fetching an instruction from memory. Conversely, when /M1 is active and /IORQ is active, the CPU is acknowledging an interrupt. In addition, the /M1 signal has two other functions within the Z80-PIO.

1. /M1 synchronizes the PIO interrupt logic.
2. When /M1 occurs without an active /RD or /IORQ signal, the PIO logic enters a reset state.

/IORQ *Input/Output Request from Z80-CPU* (input, active Low). The /IORQ signal is used in conjunction with the B/A Select, C/D Select, /CE, and /RD signals to transfer commands and data between the Z80-CPU and the Z80-PIO. When /CE, /RD, and /IORQ are active, the port addressed by B/A will transfer data to the CPU (a read operation). Conversely, when /CE and /IORQ are active but /RD is not active, then the port addressed by B/A will be written into from the CPU with either data or control information as specified by the C/D Select signal. Also, if /IORQ and /M1 are active simultaneously, the CPU is acknowledging an interrupt and the interrupting port will automatically place its interrupt vector on the CPU data bus if it is the highest priority device requesting an interrupt.

/RD *Read Cycle Status from the Z80-CPU* (input, active Low). If /RD is active a MEMORY READ or I/O READ operation is in progress. The /RD signal is used with B/A Select, C/D Select, /CE, and /IORQ signals to transfer data from the Z80-PIO to the Z80-CPU.

IEI *Interrupt Enable In* (input, active High). This signal is used to form a priority interrupt daisy chain when more than one interrupt driven device is being used. A High level on this pin indicates that no other devices of higher priority are being serviced by a CPU interrupt service routine.

IEO *Interrupt Enable Out* (output, active High). The IEO signal is the other signal requited to form a daisy chain priority scheme. It is High only if IEI is High and the CPU is not servicing an interrupt from this PIO. Thus, this signal blocks lower priority devices from interrupting while a higher priority device is being serviced by its CPU interrupt service routine.

/INT *Interrupt Request* (output, open-drain, active Low). When /INT is active, the Z80-PIO is requesting an interrupt from the Z80-CPU.

A7-A0 *Port A Bus* (bidirectional, tri-state). This 8-bit bus is used to transfer data and/or status or control information between Port A of the Z80-PIO and a peripheral device. A0 is the least significant bit of the Port A data bus.

/ASTB *Port A Strobe Pulse from Peripheral Device* (input, active Low). The meaning of this signal depends on the mode of operation selected for Port A as follows:

1. Output mode: The positive edge of this strobe is issued by the peripheral to acknowledge the receipt of data made available by the PIO.

2. Input mode: The strobe is issued by the peripheral to load data from the peripheral into the Port A input register. Data is loaded into the PIO when this signal is active.

3. Bidirectional mode: When this signal is active, data from the Port A output register is gated onto Port A bidirectional data bus. The positive edge of the strobe acknowledges the receipt of the data.

4. Control mode: The strobe is inhibited internally.

ARDY *Register A Ready* (output, active High). The meaning of this signal depends on the mode of operation selected for Port A as follows:

1. Output mode: This signal goes active to indicate that the Port A output register has been loaded and the peripheral data bus is stable and ready for transfer to the peripheral device.

2. Input mode: This signal is active when the Port A input register is empty and is ready to accept data from the peripheral device.

3. Bidirectional mode: This signal is active when data is available in the Port A output register for transfer to the peripheral device. In this mode, data is slot placed on the Port A data bus unless A STB is active.

4. Control mode: This signal is disabled and forced to a Low state.

B7-B0 *Port B Bus* (bidirectional, tri-state). This 8-bit bus is used to transfer data and/or status or control information between Port B of the PIO and a peripheral device. The Port B data bus is capable of supplying 1.5 mA @ 1.5V to drive Darlington transistors. B0 is the least significant bit of the bus.

/BSTB *Port B Strobe Pulse from Peripheral Device* (input, active Low). The meaning of this signal is similar to that of /ASTB with the following exception:

In the Port A bidirectional mode, this signal strobes data from the peripheral device into the Port A input register.

BRDY *Register B Ready* (output, active High). The meaning of this signal is similar to that of A Ready with the following exception:

In the Port A bidirectional mode this signal is High when the Port A input register is empty and ready to accept data from the peripheral device.

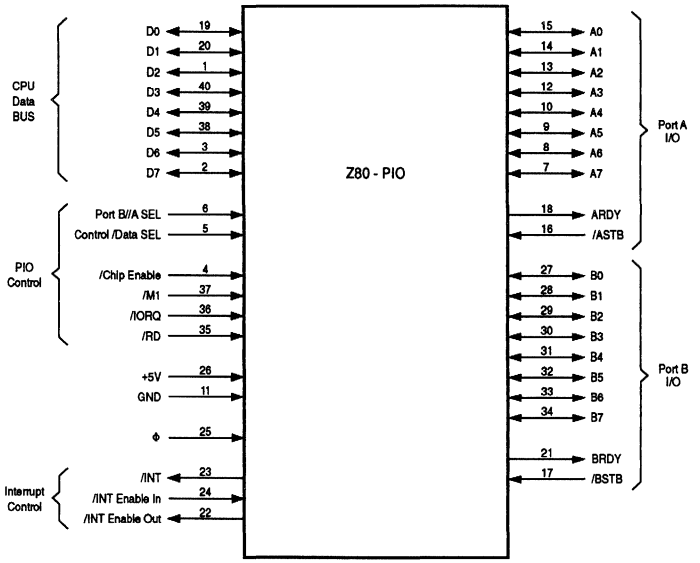


Figure 3-1. PIO Pin Functions

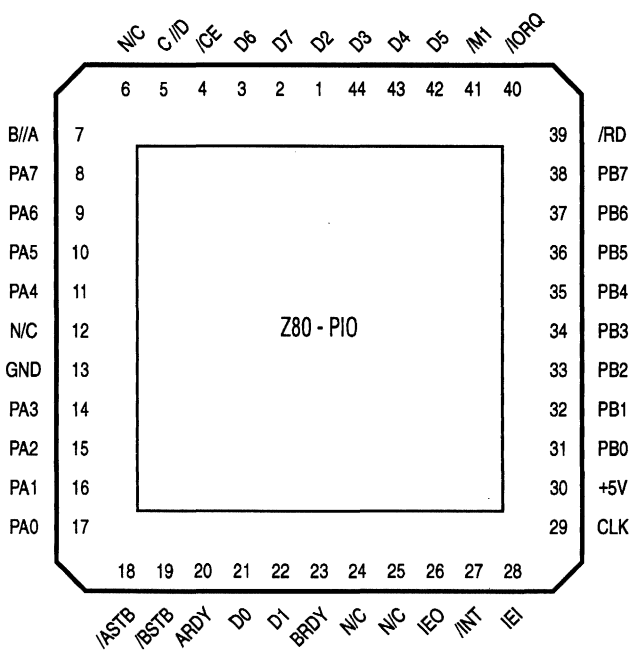


Figure 3-2. PIO 44-Pin PLCC Pin Assignments

CHAPTER 4
PROGRAMMING THE PIO

4.0 RESET

The Z80-PIO automatically enters a reset state when power is applied. The reset state performs the following functions:

1. Both port mask registers are reset to inhibit all port data bits.
2. Port data bus lines are set to a high-impedance state and the Ready "handshake" signals are inactive (Low). Mode 1 is automatically selected.
3. The vector address registers are not reset.
4. Both port interrupt enable flip-flops are reset.
5. Both port output registers are reset.

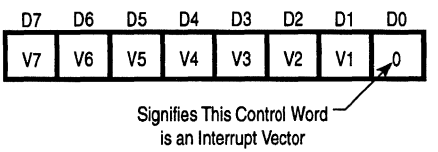
In addition to the automatic power-on reset, the PIO can be reset by applying an /M1 signal without the presence of a /RD or /IORQ signal. If no /RD or /IORQ is detected during /M1, the PIO will enter the reset state immediately after the /M1 signal goes inactive. The purpose of this reset is to allow a single external gate to generate a reset without a power down sequence. This approach was required due to the 40-pin packaging limitation.

Once the PIO has entered the internal reset state, it is held there until the PIO receives a control word from the CPU.

D

4.1 LOADING THE INTERRUPT VECTOR

The PIO has been designed to operate with the Z80-CPU using the Mode 2 interrupt response. This mode requires that an interrupt vector be supplied by the interrupting device. This vector is used by the CPU to form the address for the interrupt service routine of that port. This vector is placed on the Z80 data bus during an interrupt acknowledge cycle by the highest priority device requesting service at that time. (Refer to the Z80-CPU User's Manual Section for details on how an interrupt is serviced by the CPU). The desired interrupt vector is loaded into the PIO by writing a control word to the desired port of the PIO with the following format:

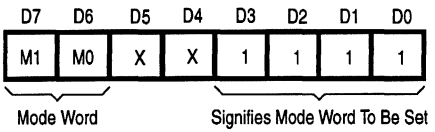


D0 is used in this case as a flag bit which when Low, causes V7 through V1 to be loaded into the vector register. At interrupt acknowledge time, the vector of the interrupting port will appear on the Z80 data bus exactly as shown in the format above.

4.2 SELECTING AN OPERATING MODE

Port A of the PIO may be operated in any of four distinct modes: Mode 0 (output mode), Mode 1 (input mode), Mode 2 (bidirectional mode), and Mode 3 (control mode). Note that the mode numbers have been selected for mnemonic significance; i.e., 0 = Out, 1 = In, 2 = Bidirectional. Port B can operate in any of these modes except Mode 2.

The mode of operation must be established by writing a control word to the PIO in the following format:



X=Unused Bit

Bits D7 and D6 from the binary code for the desired mode according to the following table:

D7	D6	Mode
0	0	0 (output)
0	1	1 (input)
1	0	2 (bidirectional)
1	1	3 (control)

Bits D5 and D4 are ignored. Bits D3-D0 must be set to 1111 to indicate "Set Mode".

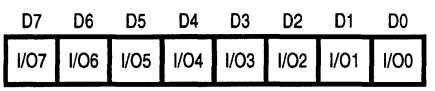
Selecting Mode 0 enables any data written to the port output register by the CPU to be enabled onto the port data bus. The contents of the output register may be changed at any time by the CPU simply by writing a new data word to the port. Also the current contents of the output register may be read back to the Z80-CPU at any time through the execution of an input instruction.

With Mode 0 active, a data write from the CPU causes the Ready handshake line of that port to go High to notify the peripheral that data is available. This signal remains High until a strobe is received from the peripheral. The rising edge of the strobe generates an interrupt (if it has been enabled) and causes the Ready line to go inactive. This very simple handshake is similar to that used in many peripheral devices.

Selecting Mode 1 puts the port into the input mode. To start handshake operation, the CPU merely performs an input read operation from the port. This activates the Ready line to the peripheral to signify that data should be loaded into the empty input register. The peripheral device then strobes data into the port input register using the strobe line. Again, the rising edge of the strobe causes an interrupt request (if it has been enabled) and deactivates the Ready signal. Data may be strobed into the input register regardless of the state of the Ready signal if care is taken to prevent a data overrun condition.

Mode 2 is a bidirectional data transfer mode which uses all four handshake lines. Therefore, only Port A may be used for Mode 2 operation. Mode 2 operation uses the Port A handshake signals for output control and the Port B handshake signals for input control. Thus, both ARDY and BRDY may be active simultaneously. The only operational difference between Mode 0 and the output portion of Mode 2 is that data from the Port. A output register is allowed on to the port data bus only when /ASTB is active in order to achieve a bidirectional capability.

Mode 3 operation is intended for status and control applications and does not utilize the handshake signals. When Mode 3 is selected, the next control word sent to the PIO must define which of the port data bus lines are to be inputs and which are outputs. The format of the control word is shown below:

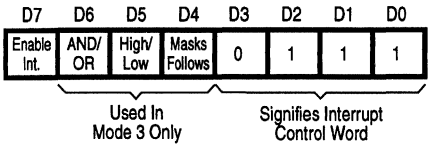


If any bit is set to a one, then the corresponding data bus line will be used as an input. Conversely, if the bit is reset, the line will be used as an output.

During Mode 3 operation, the strobe signal is ignored and the Ready line is held Low. Data may be written to a port or read from a port by the Z80-CPU at any time during Mode 3 operation. When reading a port, the data returned to the CPU will be composed of input data from port data bus lines assigned as inputs plus port output register data from those lines assigned as outputs.

4.3 SETTING THE INTERRUPT CONTROL WORD

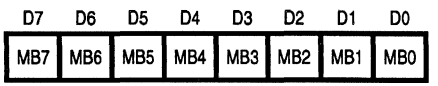
The interrupt control word for each port has the following format:



If bit D7 = 1, the interrupt enable flip-flop of the port is set and the port may generate an interrupt. If bit D7 = 0, the enable flag is reset and interrupts may not be generated. If an interrupt is pending when the enable flag is set, it will then be enabled onto the CPU interrupt request line. bits D6, D5, and D4 are used only with Mode 3 operation. However, setting bit D4 of the interrupt control word during any mode of operation will cause any pending interrupt to be reset. These three bits are used to allow for interrupt operation in Mode 3 when any group of the I/O lines go to certain defined states. Bit D6 (AND/OR) defines the logical operation to be performed in port monitoring. If bit D6 = 1, an AND function is specified and if D6 = 0, an OR function is specified. For example, if the AND function is specified, all bits must go to a specified state before an interrupt will be generated while the OR function will generate an interrupt if any specified bit goes to the active state.

Bit D5 defines the active polarity of the port data bus line to be monitored. If bit D5 = 1, the port data lines are monitored for a high state while if D5 = 0, they will be monitored for a low state.

If bit D4 = 1, the next control word sent to the PIO must define a mask as follows:



Only those port lines whose mask bit is zero will be monitored for generating an interrupt is high, the forced state of Ready will prevent input register data from changing while the CPU is reading the PIO. Ready will go High again after the trailing edge of the /IORQ as previously described.

D

CHAPTER 5
TIMING

5.0 OUTPUT MODE (MODE 0)

Figure 5-1 illustrates the timing associated with Mode 0 operation. An output cycle is always started by the execution of an output instruction by the CPU. A /WR* pulse is generated by the PIO during a CPU I/O write operation and is used to latch the data from the CPU data bus into the addressed ports (A or B) output register. The rising edge of the /WR* pulse then raises the Ready flag after the next falling edge of Φ to indicate that data is available for the peripheral device. In most systems, the rising edge of the Ready signal can be used as a latching signal in the peripheral device if desired. The Ready signal will remain active until: (1) a positive edge is received from the strobe line indicating that the peripheral has taken the data, or (2) if already active, Ready will be forced low 1one and one-half Φ cycles after the leading edge of /IORQ if the port's output register is written into. Ready will return High on the first falling edge of Φ after the trailing edge of /IORQ. This guarantees that Ready is low when port data is changing. The Ready signal will not go inactive until a falling edge occurs on the clock (Φ) line. The purpose of delaying the negative transition of the Ready signal until after a negative clock transition is that it allows for a very simple generation scheme for the strobe pulse. By merely connecting the Ready line to the Strobe line, a strobe with a duration of one clock period will be generated with no other logic required. The positive edge of the strobe pulse automatically generates an /INT request if the interrupt enable flip-flop has been set and this device is the highest priority device requesting an interrupt.

If the PIO is not in a reset state, the output register may be loaded before Mode 0 is selected. This allows the port output lines to become active in a user defined state.

D

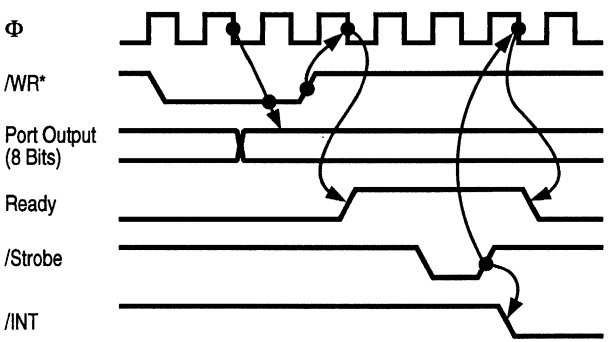


Figure 5-1. Mode 0 (Output) Timing

5.1 INPUT MODE (MODE 1)

Figure 5-2 illustrates the timing of an input cycle. The peripheral initiates this cycle using the strobe line after the CPU has performed a data read. A low level on this line loads data into the port input register and the rising edge of the strobe line activates the interrupt request line (/INT) if the interrupt enable is set and this is the highest priority requesting device. The next filling edge of the clock line (Φ) will then reset the Ready line to an inactive state signifying that the input register is full and further loading must be inhibited until the CPU reads the data. The CPU will in the course of its interrupt service routine, read the data from the interrupting port. When this occurs, the positive edge from the CPU /RD signal will raise the Ready line with the next low going transition of Φ indicating that new data can be loaded into the PIO. If already active, Ready will be forced low one and one-half Φ periods following the leading edge of /IORQ during a read of a PIO port. If the user strobes data into the PIO only when Ready

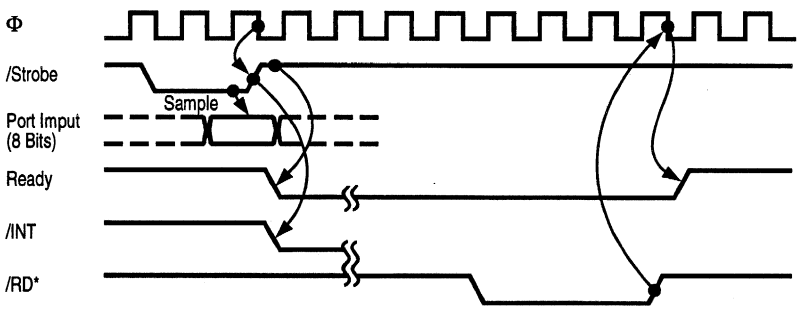


Figure 5-2. Mode 1 (Input) Timing

5.2 BIDIRECTIONAL MODE (MODE 2)

This mode is merely a combination of Mode 0 and Mode 1 using all four handshake lines. Since it requires all four lines, it is available only on Port A. When this mode is used on Port A, Port B must be set to the Bit Control Mode. The same interrupt vector will be returned for a Mode 3 interrupt on Port B and an input transfer interrupt during Mode 2 operation of Port A. Ambiguity is avoided if Port B is operated in a polled mode and the Port B mask register is set to inhibit all bits.

Figure 5-3 illustrates the timing for this mode. It is almost identical to that previously described for Mode 0 and Mode 1 with the Port A handshake lines used for output control and the Port B lines used for input control. The difference between the two modes is that, in Mode 2, data is allowed out onto the bus only when the A strobe is Low. The rising edge of this strobe can be used to latch the data into the peripheral since the data will remain stable until after this edge. The input portion of Mode 2 operates identically to Mode 1. Note that both Port A and Port B must have their interrupts enabled to achieve an interrupt driven bidirectional transfer.

The peripheral must not gate data onto a port data bus while /ASTB is active. Bus contention is avoided if the peripheral uses /BSTB to gate input data onto the bus. The PIO uses the /BSTB low level to latch this data. The PIO has been designed with a zero hold time requirement for the data when latching in this mode so that this simple gating structure can be used by the peripheral. That is, the data can be disabled from the bus immediately after the strobe rising edge.

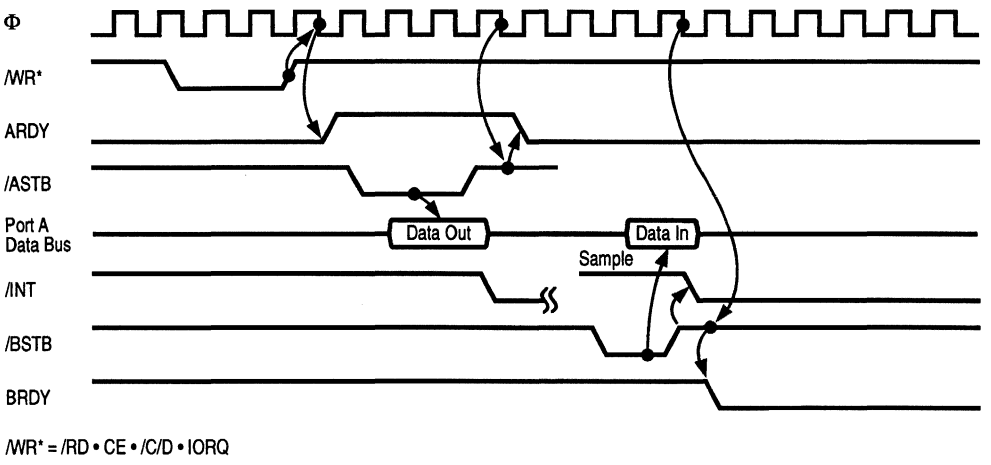


Figure 5-3. Port A, Mode 2 (Bidirectional) Timing

5.3 CONTROL MODE (MODE 3)

The control mode does not utilize the handshake signals and a normal port write or port read can be executed at any time. When writing, the data will be latched into output registers with the same timing as Mode 0. ARDY will be forced low whenever Port A is operated in Mode 3. BRDY will be held low whenever Port B is operated in Mode 3 unless Port A is in Mode 2. In the latter case, the state of BRDY will not be affected.

When reading the PIO, the data returned to the CPU will be composed of output register data from those port data lines assigned as outputs and input register data from those port data lines assigned as inputs. The input register will contain data which was present immediately prior to the falling edge of /RD. See Figure 5-4.

An interrupt will be generated if interrupts from the port are enabled and the data on the port data lines satisfies the logical equation defined by the 8-bit mask and 2-bit mask control registers. Another interrupt will not be generated until a change occurs in the status of the logical equation. A Mode 3 interrupt will be generated only if the result of a Mode 3 logical operation changes from false to true. For example, assume that the Mode 3 logical equation is an OR function. An unmasked port data line becomes active and an interrupt is requested. If a second unmasked port data line becomes active concurrently with the first, a new interrupt will not be requested since a change in the result of the Mode 3 logical operation has not occurred.

If the result of a logical operation becomes true immediately prior to or during /M1 an interrupt will be requested after the trailing edge of /M1.

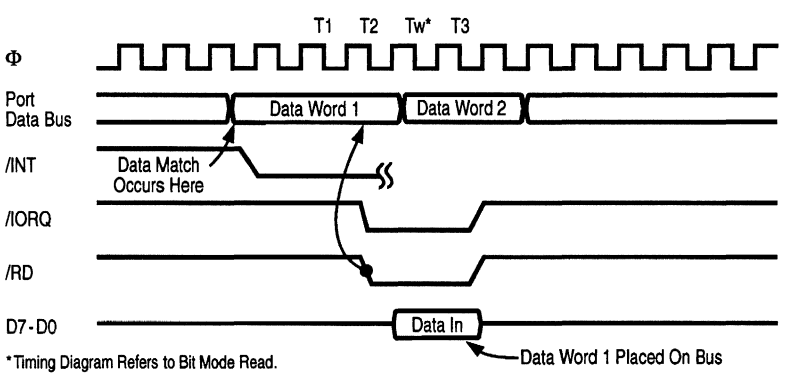


Figure 5-4. Control Mode (Mode 3) Timing

CHAPTER 6
INTERRUPT SERVICING

6.0 INTERRUPT SERVICING

Some time after an interrupt is requested by the PIO, the CPU will send out an interrupt acknowledge (/M1 and /IORQ). During this time the interrupt logic of the PIO will determine the highest priority port which is requesting an interrupt. (This is simply the device with its Interrupt Enable Input high and its Interrupt Enable Output low). To insure that the daisy chain enable lines stabilize, devices are inhibited from changing their interrupt request status when /M1 is active. The highest priority device places the contents of its interrupt vector register onto the Z80 data bus during interrupt acknowledge.

Figure 6-1 illustrates the timing associated with interrupt requests. During /M1 time, no new interrupt requests can be generated. This gives time for the Int Enable signals to ripple through up to four PIO circuits. The PIO, with IEI High and IEO Low during /INTA, will place the 8-bit interrupt vector of the appropriate port on the data bus at this time.

If an interrupt requested by the PIO is acknowledged, the requesting port is 'under service'. IEO of this port will remain low until a return from interrupt instruction (RETI) is executed while IEI of the port is high. If an interrupt request is not acknowledged, IEO will be forced high for one /M1 cycle after the PIO decodes the opcode 'ED'. This action guarantees that the 2-byte RETI instruction is decoded by the proper PIO port (Figure 6-2).

Figure 6-3 illustrates a typical nested interrupt sequence that could occur with four ports connected in the daisy chain. In this sequence Port 2A requests and is granted an interrupt. While this port is being serviced, a higher priority port (1B) requests and is granted an interrupt. The service routine for the higher priority port is completed and a RETI instruction is executed to indicate to the port that its routine is complete. At this time the service routine of the lower priority port is completed.

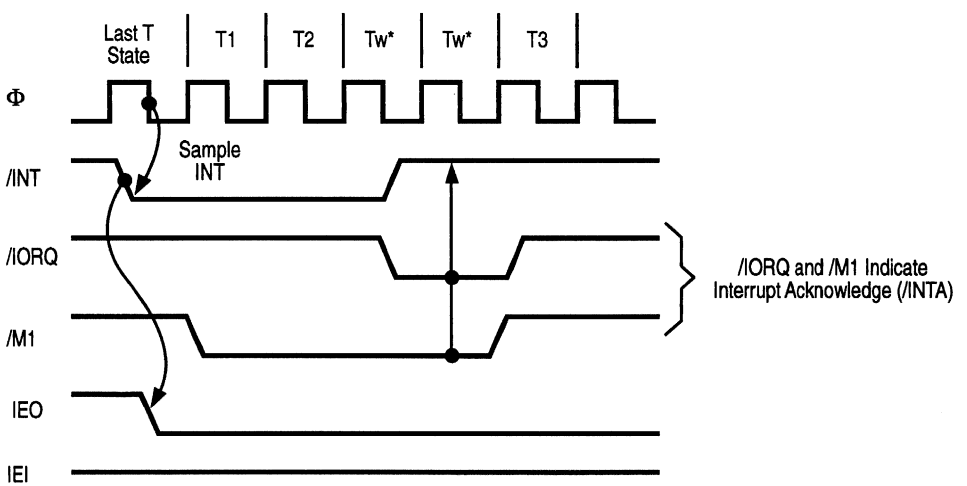


Figure 6-1. Interrupt Acknowledge Timing

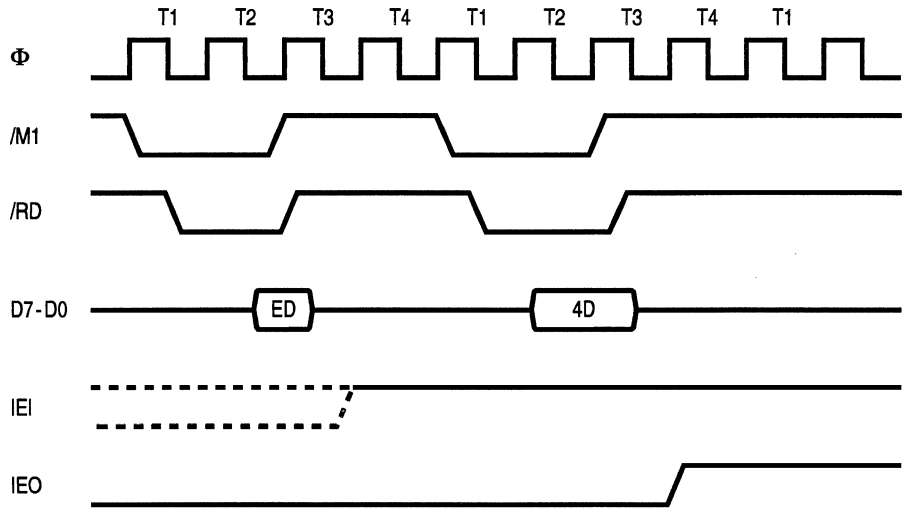


Figure 6-2. Return From Interrupt Cycle

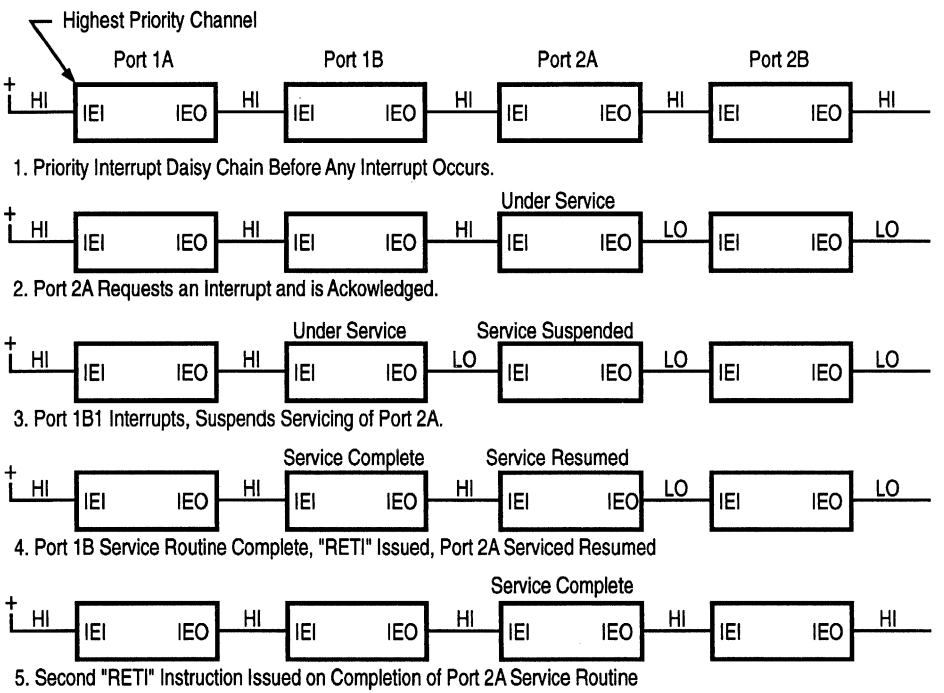


Figure 6-3. Daisy-Chain Interrupt Servicing

CHAPTER 7
APPLICATIONS

7.0 EXTENDING THE INTERRUPT DAISY CHAIN

Without any external logic, a maximum of four Z80-PIO devices may be daisy chained into a priority interrupt structure. This limitation is required so that the interrupt enable status (IEO) ripples through the entire chain between the beginning of /M1, and the beginning of /IORQ during an interrupt acknowledge cycle. Since the interrupt enable status cannot change during /M1, the vector address returned to the CPU is assured to be from the highest priority device which requested an interrupt.

If more than four PIO devices must be accommodated, a "look-ahead" structure may be used as shown in Figure 7-1. With this technique, more than thirty PIO's may be chained together using standard TTL logic.

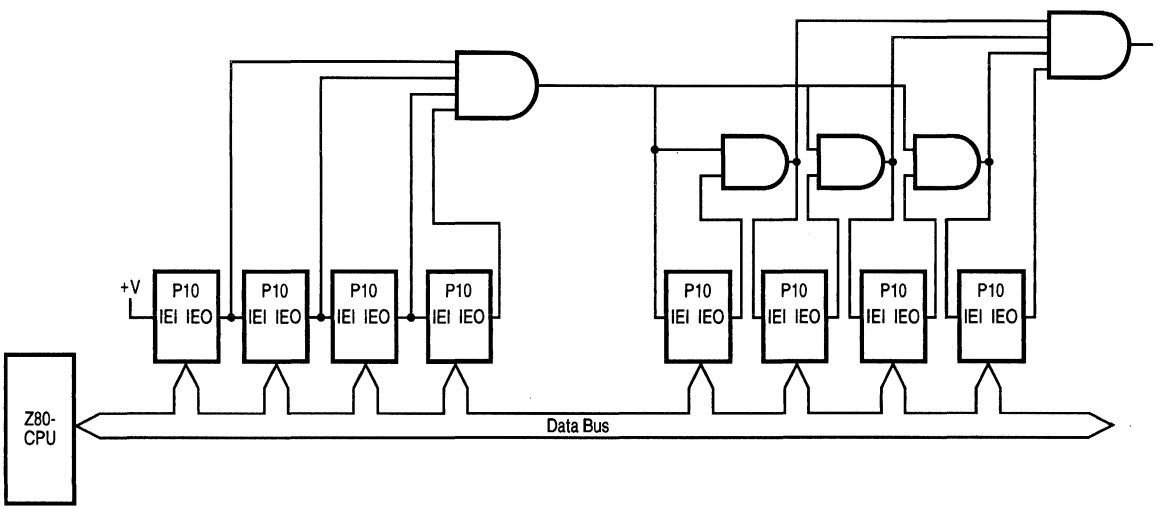


Figure 7-1. A Method of Extending the Interrupt Priority Daisy Chain

7.1 I/O DEVICE INTERFACE

In this example, the Z80-PIO is connected to an I/O terminal device which communicates over an 8-bit parallel bidirectional data bus as illustrated in Figure 7-2. Mode 2 operation (bidirectional) is selected by sending the following control word to Port A:

D7	D6	D5	D4	D3	D2	D1	D0
1	0	X	X	1	1	1	1

Mode Control

Next, the proper interrupt vector is loaded (refer to CPU Manual for details on the operation of the interrupt).

D7	D6	D5	D4	D3	D2	D1	D0
V7	V6	V5	V4	V3	V2	V1	0

Interrupts are then enabled by the rising edge of the first /M1 after the interrupt mode word is set unless that /M1 defines an interrupt acknowledge cycle. If a mask follows the interrupt mode word, interrupts are enabled by the rising edge of the first /M1 following the setting of the mask.

Data can now be transferred between the peripheral and the CPU. The timing for this transfer is as described in Section 5.0.

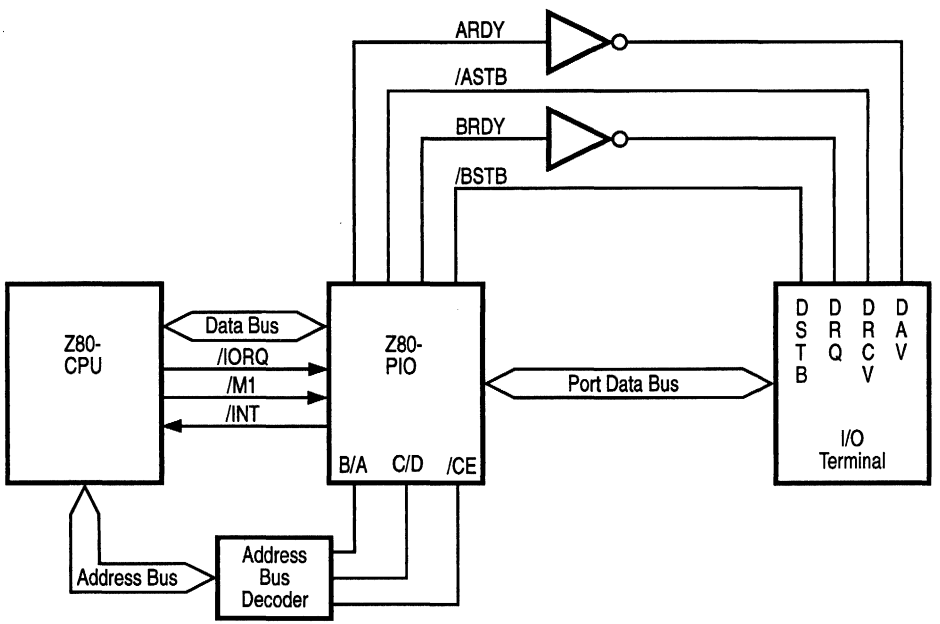


Figure 7-2. Example of I/O Interface

7.2 CONTROL INTERFACE

A typical control mode application is illustrated in Figure 7-3. Suppose an industrial process is to be monitored. The occurrence of any abnormal operating condition is to be reported to a Z80-CPU based control system. The process control and status word has the following format:

D7	D6	D5	D4	D3	D2	D1	D0
Special Test	Turn On Power	Power Failure Alarm	Halt Processing	Temp. Alarm	Turn Heaters On	Pressurize System	Pressure Alarm

The PIO may be used as follows. First Port A is set for Mode 3 operation by writing the following control word to Port A.

D7	D6	D5	D4	D3	D2	D1	D0
1	1	X	X	1	1	1	1

Whenever Mode 3 is selected, the next control word sent to the port must be an I/O select word. In this example we wish to select port data lines A5, A3, and A0 as inputs and so the following control word is written:

D7	D6	D5	D4	D3	D2	D1	D0
0	0	1	0	1	0	0	1

Next the desired interrupt vector must be loaded (refer to the CPU manual for details):

D7	D6	D5	D4	D3	D2	D1	D0
V7	V6	V5	V4	V3	V2	V1	0

An interrupt control word is next sent to the port:

D7	D6	D5	D4	D3	D2	D1	D0
1	0	1	1	0	1	1	1
Enable Interrupts	OR Logic	Active High	Mask Follows	Interrupt Control			

The mask word following the interrupt mode word is:

D7	D6	D5	D4	D3	D2	D1	D0
1	1	0	1	0	1	1	0

Selects A5, A3, and A0 to be Monitored

Now, if a sensor puts a high level on line A5, A3, or A0, an interrupt request will be generated. The mask word may select any combination of inputs or outputs to cause an interrupt. For example, if the mask word above had been:

D7	D6	D5	D4	D3	D2	D1	D0
0	1	0	1	0	1	1	0

then an interrupt request would also occur if bit A7 (Special Test) of the output register was set.

Assume that the following port assignments are to be used:

E0H = Port A Data
E1H = Port B Data
E2H = Port A Control
E3H = Port B Control

All port numbers are in hexadecimal notation. This particular assignment of port numbers is convenient since A0 of the address bus can be used as the Port B/A Select and A1 of the address bus can be used as the Control/Data Select. The Chip Enable would be the decode of CPU address bits A7 thru A2 (1110 00). Note that if only a few peripheral devices are being used, a Chip Enable decode may not be required since a higher order address bit could be used directly.

D

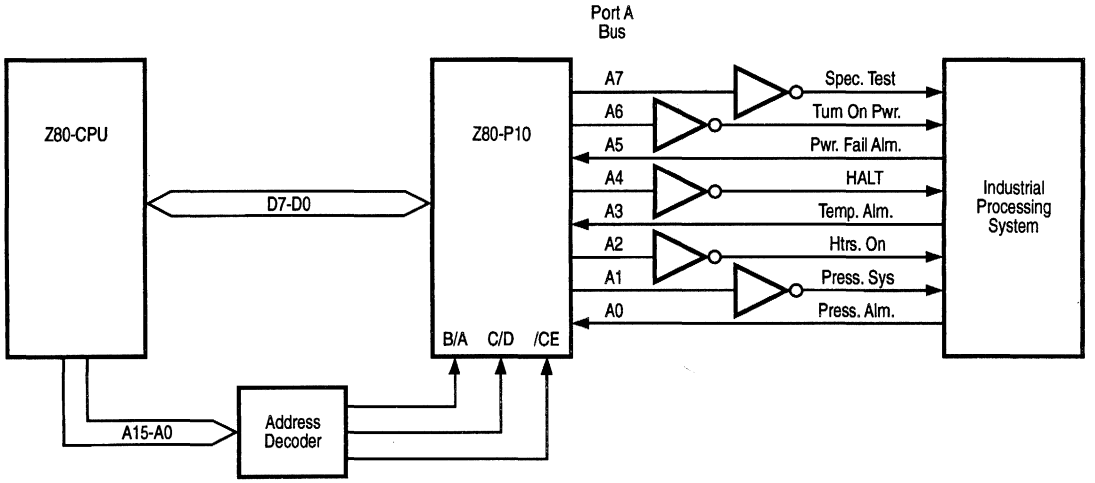


Figure 7-3. Control Mode Application

CHAPTER 8
PROGRAMMING SUMMARY

8.0 LOAD INTERRUPT VECTOR

V7	V6	V5	V4	V3	V2	V1	0

8.1 SET MODE

M1	M0	X	X	1	1	1	1

M1	M0	Mode
0	0	Output
0	1	Input
1	0	Bidirectional
1	1	Bit Control

When selecting Mode 3, the next word must set the I/O Register:

8.2 SET INTERRUPT CONTROL

Enable Int.	AND/ OR	High/ Low	Masks Follows	0	1	1	1

Used In Mode 3 Only

If the "mask follows" bit is high, the next control word written to the port must be the mask:

MB7	MB6	MB5	MB4	MB3	MB2	MB1	MB0

MB = 0, Monitor bit

MB = 1, Mask bit from being monitored

Also, the interrupt enable flip-flop of a port may be set or reset without modifying the rest of the interrupt control word by using the following command:

Enable Int.	X	X	X	0	0	1	1

ZILOG

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

List of Figures

List of Tables

CHAPTER 1
GENERAL INFORMATION

1.0 FEATURES

- CMOS and NMOS Version
- 40-Pin DIP, 44-Pin PLCC/QFP Packages
- Single 5V Power Supply
- Single-Phase 5V Clock
- All Inputs and Outputs TTL Compatible
- Two Independent Full-Duplex Channels
- Data Rates in Synchronous or Isosynchronous Modes:
  - 0-800K Bits/Second with 4 MHz System Clock Rate
  - 0-1.2M Bits/Second with 6 MHz System Clock Rate
  - 0-2.5M Bits/Second with 10 MHz System Clock Rate
- Receiver Data Registers Quadruply Buffered; Transmitter Doubly Buffered
- Asynchronous Features:
  - 5, 6, 7, or 8 Bits/Character
  - 1, 1 1/2, or 2 Stop Bits
  - Even, Odd, or No Parity
  - x1, x16, x32, and x64 Clock Modes
  - Break Generation and Detection
  - Parity, Overrun, and Framing Error Detection
- Binary Synchronous Features:
  - Internal or External Character Synchronization
  - One or Two Sync Characters in Separate Registers
  - Automatic Cync Character Insertion
  - CRC Generation and Checking
- HDLC and IBM SDLC Features:
  - Abort Sequence Generation and Detection
  - Automatic Zero Insertion and Deletion
  - Automatic Flag insertion Between Messages
  - Address Field Recognition
  - 1-Field Residue Handling
  - Valid Receive Messages Protected from Overrun
- CRC generation and checking
- Separate Modem Control Inputs and Outputs for Both Channels
- CRC-16 or CRC-CCITT Block Check
- Daisy-Chain Priority Interrupt Logic Provides Automatic Interrupt Vectoring Without External Logic
- Modem Status can be Monitored

1.1 INTRODUCTION

The Z80-SIO (Serial Input/Output) is a dual-channel multi-function peripheral component designed to satisfy a wide variety of serial data communications requirements in microcomputer systems. Its basic function is a serial-to-parallel, parallel-to-serial converter/controller, but, within that role, it is configurable by systems software so its "personality" can be optimized for a given serial data communications application.

The Z80-SIO is capable of handling asynchronous and synchronous byte-oriented protocols such as IBM Bisync, and synchronous bit-oriented protocols such as HDLC and IBM SDLC. This versatile device can also be used to support virtually any other serial protocol for applications other than data communications (cassette or floppy disk interfaces, for example).

The Z80-SIO can generate and check CRC codes in any synchronous mode and can be programmed to cheek data integrity in various modes. The device also has facilities for modem controls in both channels. In applications where these controls are not needed, the modem controls can be used for general-purpose I/O.

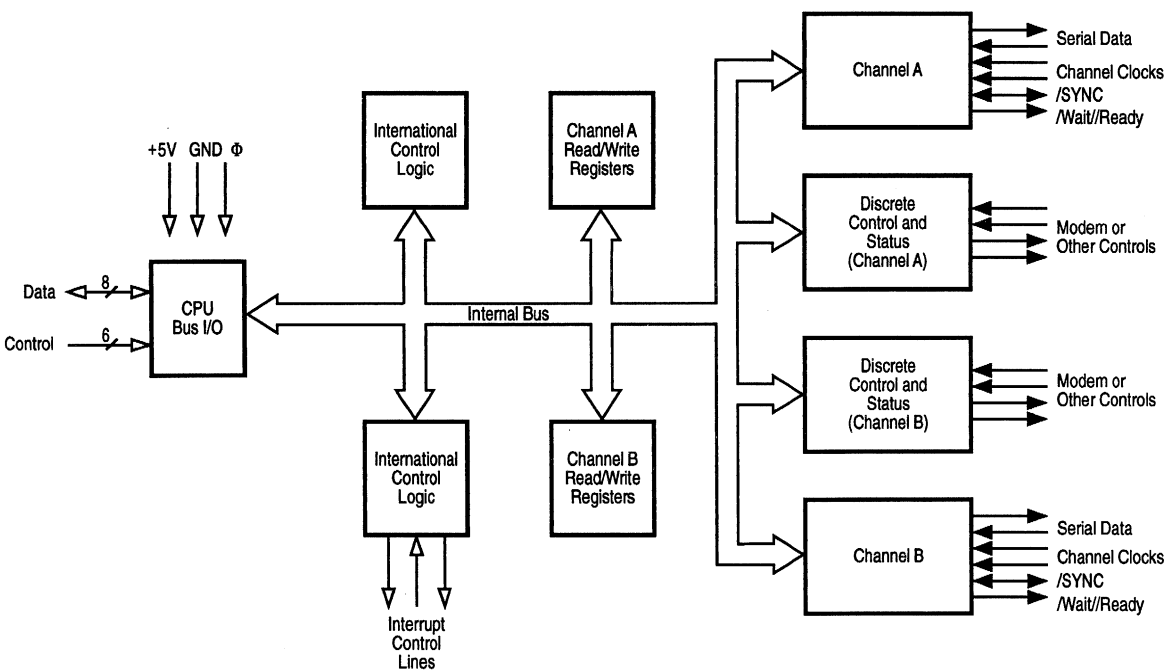


Figure 1-1. Z80-SIO Block Diagram

CHAPTER 2
PIN DESCRIPTION

2.0 PIN FUNCTIONS

D7-D0 *System Data Bus* (bidirectional, tri-state). The system data bus transfers data and commands between the CPU and the Z80-SIO. D0 is the least significant bit.

B//A *Channel A or B Select* (input, High selects Channel B). This input defines which channel is accessed during a data transfer between the CPU and the Z80-SIO. Address bit A0 from the CPU is often used for the selection function.

C//D *Control Or Data Select* (input, High selects Control). This input defines the type of information transfer performed between the CPU and (be Z80-SIO. A High at this input during a CPU write to the Z80-SIO causes the information on the data bus to be interpreted as a command for the channel selected by B//A. A Low at C//D means that the information on the data bus is data. Address bit A1 is often used for this function.

/CE *Chip Enable* (input, active Low). A Low level at this input enables the Z80-SIO to accept command or data inputs from the CPU during a write cycle, or to transmit data to the CPU during a read cycle.

Φ *System Clock* (input). The Z80-SIO uses the standard Z80A System Clock to synchronize internal signals. This is a single-phase clock.

/M1 *Machine Cycle One* (input from Z80-CPU, active Low). When /M1 is active and /RD is also active, the Z80-CPU is fetching an instruction from memory; when /M1 is active while /IORQ is active, the Z80-SIO accepts /M1 and /IORQ as an interrupt acknowledge if the Z80-SIO is the highest priority device that has interrupted the Z80-CPU.

/IORQ *Input/Output Request* (input from CPU, active Low). /IORQ is used in conjunction with B//A, C//D, /CE, and /RD to transfer commands and data between the CPU and the Z80-SIO. When /CE, /RD, and /IORQ are all active, the channel selected by B//A transfers data to the CPU (a read operation). When /CE and /IORQ are active, but /RD is inactive, the channel selected by B//A is written to by the CPU with either data or control information as specified by C//D. As mentioned previously, if /IORQ and /M1 are active simultaneously, the CPU is acknowledging an interrupt and the Z80-SIO automatically places its interrupt vector on the CPU data bus if it is the highest priority device requesting an interrupt.

/RD *Read Cycle Status* (input from CPU, active Low). If /RD is active, a memory or I/O read operation is in progress. /RD is used with B//A, /CE, and /IORQ to transfer data from the Z80-SIO to the CPU.

/RESET *Reset* (input, active Low). A Low /RESET disables both /RESET and transmitters, forces TxDA and TxDB marking, forces the modem controls High and disables all interrupts. The control registers must be rewritten after the Z80-SIO is reset and before data is transmitted or received.

IEI *Interrupt Enable In* (input, active High). This signal is used with IEO to form a priority daisy-chain when there is more than one interrupt-driven device. A High on this line indicates that no other device of higher priority is being serviced by a CPU interrupt service routine.

IEO *Interrupt Enable Out* (output, active High). IEO is High only if IEI is High and the CPU is not servicing an interrupt from this Z80-SIO. Thus, this signal blocks lower priority devices from interrupting while a higher priority device is being serviced by its CPU interrupt service routine.

/INT *Interrupt Request* (output, open-drain, active Low). When the Z80-SIO is requesting an interrupt, it pulls /INT Low.

W//RDYA, W//RDYB *Wait/Ready A, Wait/Ready B* (outputs, open-drain when programmed for Wait function, driven High and Low when programmed for Ready function). These dual-purpose outputs may be programmed as Ready lines for a DMA controller or as Wait lines that synchronize the CPU to the Z80-SIO data rate. The reset state is open-drain.

/CSTA, /CSTB *Clear To Send* (inputs, active Low). When programmed as Auto Enables, a Low on these inputs enables the respective transmitter. If not programmed as Auto Enables, these inputs may be programmed as general-purpose inputs. Both inputs are Schmitt-trigger buffered to accommodate slow-risetime inputs. The Z80-SIO detects pulses on these inputs and interrupts the CPU on both logic level transitions. The Schmitt-trigger inputs do not guarantee a specified noise-level margin.

/DCDA, /DCDB *Data Carrier Detect* (inputs, active Low). These signals are similar to the /CTS inputs, except they can be used as receiver enables.

RxDA, RxDB *Receive Data* (inputs, active High).

TxDA, TxDB *Transmit Data* (outputs, active High).

/RxCA, /RxCB* *Receiver Clocks* (inputs). See the following section on bonding options. The Receive Clocks may be 1, 16, 32, or 64 times the data rate in asynchronous modes. Receive data is sampled on the rising edge of /RxC.

/TxCA, /TxCB* *Transmitter Clocks* (inputs). See section on bonding options. In asynchronous modes, the Transmitter clocks may be 1, 16, 32 or 64 times the data rate. The multiplier for the transmitter and the receiver must be the same. Both the /TxC and /RxC inputs are Schmitt-trigger buffered for relaxed rise- and fall-time requirements (no noise margin is specified). TxD Changes on the falling edge of /TxC.

***Note:** These clocks may be directly driven by the Z80-CTC (Counter Timer Circuit) for fully programmable baud rate generation.

/RTSA, /RTSB *Request To Send* (outputs, active Low). When the /RTS bit is set, the /RTS output goes Low. When the /RTS bit is reset in the Asynchronous mode, the output goes High after the transmitter is empty. In Synchronous modes, the /RTS pin strictly follows the state of the /RTS bit. Both pins can be used as general-purpose outputs.

/DTRA, /DTRB *Data Terminal, Ready* (outputs, active Low). See note on bonding options. These outputs follow the state programmed into the /DTR bit. They can also be programmed as general-purpose outputs.

/SYNCA, /SYNCB *Synchronization* (inputs/outputs, active Low). These pins can act either as inputs or outputs. In the Asynchronous Receive mode, they are inputs similar to /CTS and /DCD. In this mode, the transitions on these lines affect the state of the Sync/Hunt status bits in RR0. In the External Sync mode, these lines also act as inputs. When external synchronization is achieved, /SYNC must be driven Low on the second rising edge of /RxC after that rising edge of /RxC on which the last bit of the sync character was received. In other logic must wait for two full Receive Clock cycles to activate the /SYNC input. Once SYNC is forced Low, it is wise to keep it Low until the CPU informs the external sync logic that synchronization has been lost or a new message is about to start. Character assembly begins on the rising edge of /RxC that immediately precedes the falling edge of /SYNC in the External Sync mode.

In the Internal Synchronization mode (Monosync and Bisync), these pins act as outputs that are active during the part of the receive clock (/RxC) cycle in which sync characters are recognized. The sync condition is not latched, so these outputs are active each time a sync pattern is recognized, regardless of character boundaries.

2.1 BONDING OPTIONS

The constraints of a 40-pin package make it impossible to bring out the Receive Clock, Transmit Clock, Data Terminal Ready, and Sync signals for both channels. Therefore, Channel B must sacrifice a signal or have two signals bonded together. Since user requirements vary, three bonding options are offered:

- Z80-SIO/O has all four signals, but /TxCB and /RxCD are bonded together (Figure 2-1).
- Z80-SIO/1 sacrifices /DTRB and keeps /TxCB, /RxCD and /SYNCB (Figure 2-3).
- Z80-SIO/2 sacrifices /SYNCB and keeps /TxCB, /RxCB and /DTRB (Figure 2-5).
- The 44-pin package version SIO/3 (QFP) and SIO/4 (PLCC) have all signals (Figures 2-7 and 2-8).

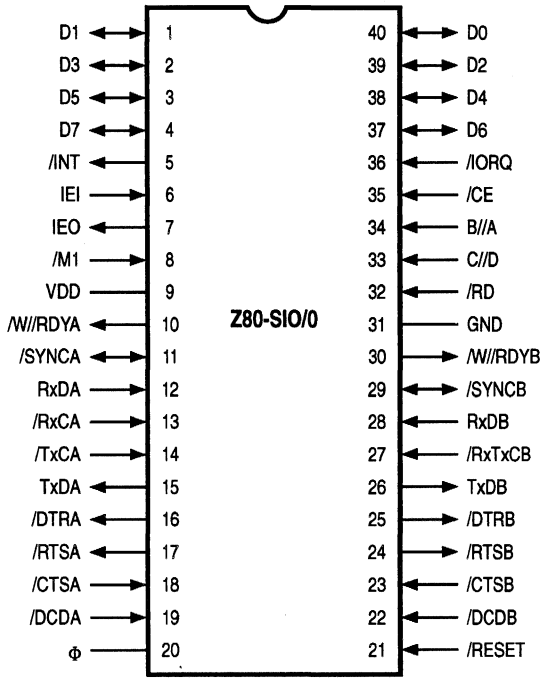


Figure 2-1. Z80-SIO/0 Pin Functions

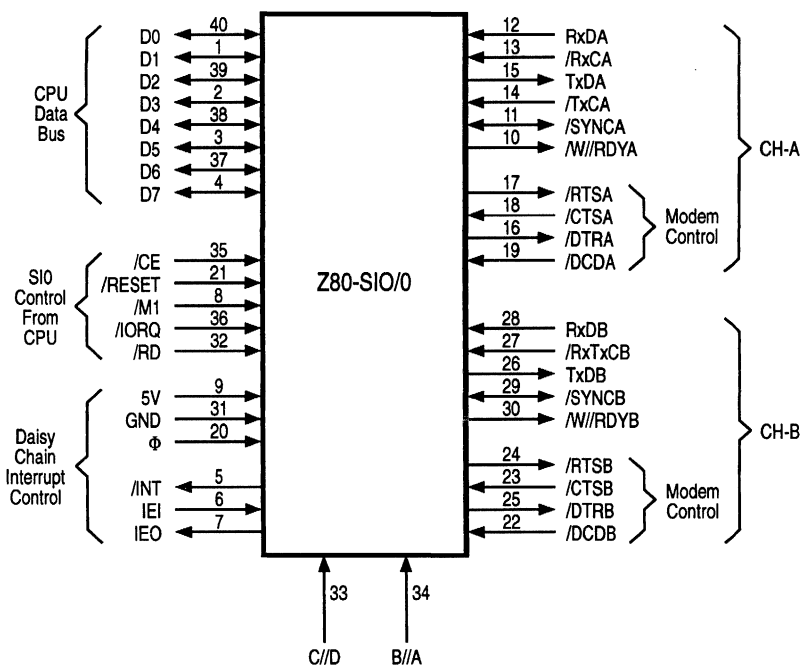
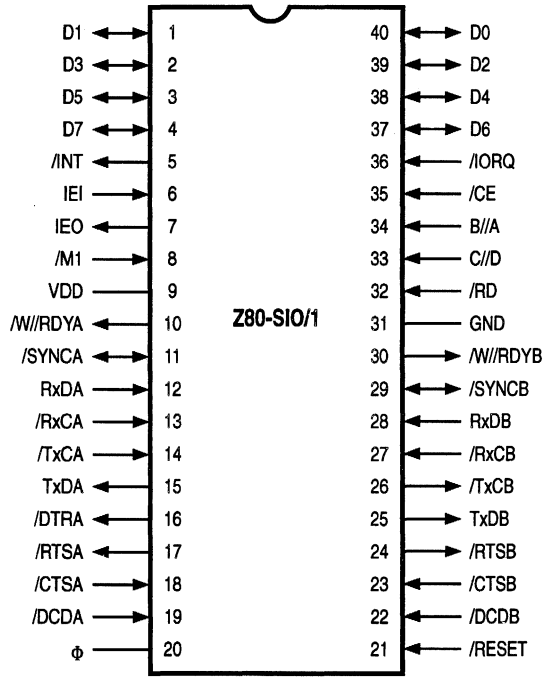


Figure 2-2. Z80-ZIO/0 Pin Assignments



Fiugre 2-3. Z80-SIO/1 Pin Functions

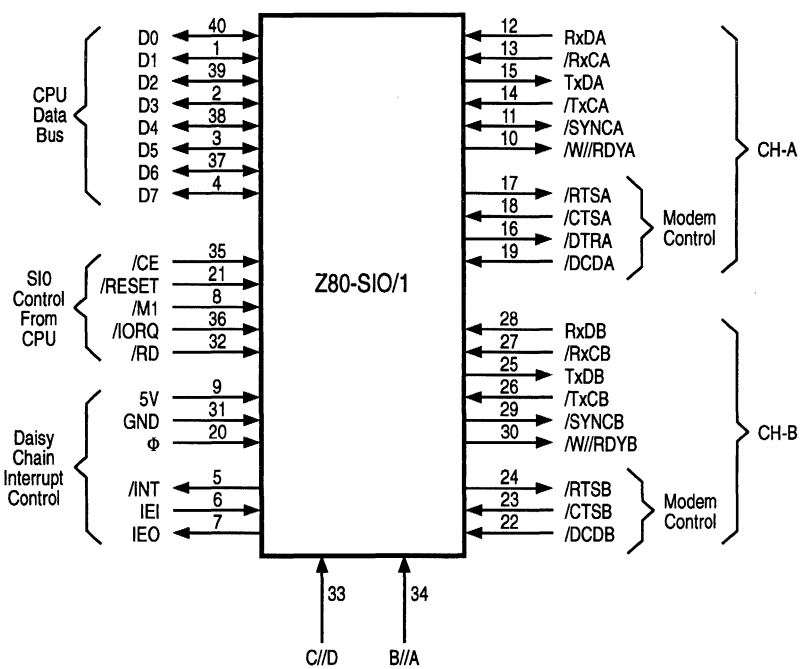
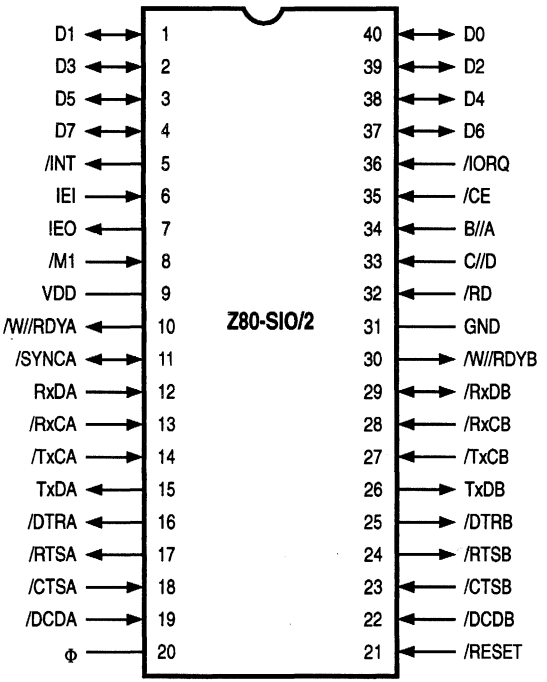


Figure 2-4. Z80-ZIO/1 Pin Assignments



Fiugre 2-5. Z80-SIO/2 Pin Functions

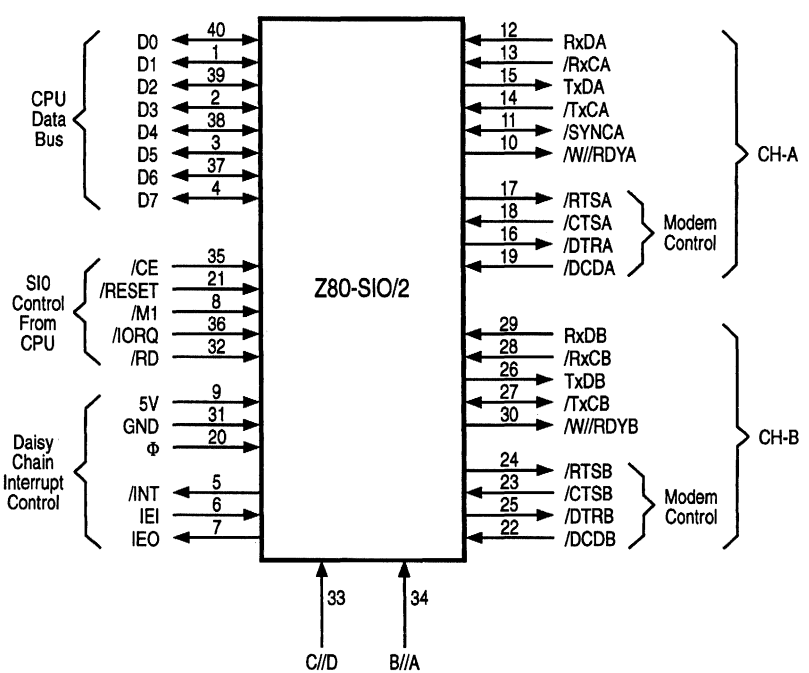
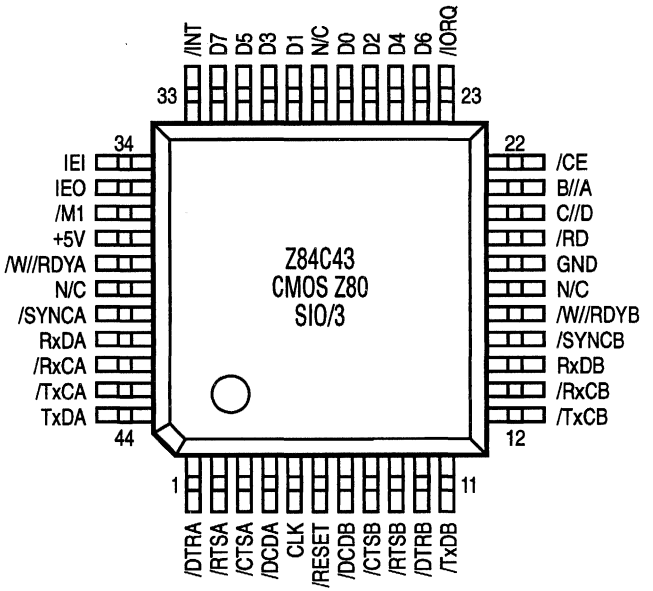


Figure 2-6. Z80-ZIO/2 Pin Assignments

E



Fiugre 2-7. Z80-SIO/3 Pin Assignments

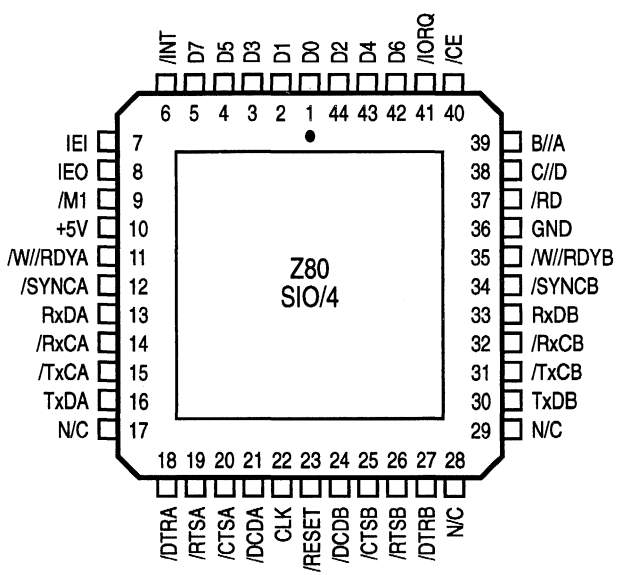


Figure 2-8. Z80-ZIO/4 Pin Assignments

CHAPTER 3
ARCHITECTURE

3.0 INTRODUCTION

The device internal structure includes a Z80-CPU interface, internal control and interrupt logic, and two full-duplex channels. Associated with each channel are read and write registers, and discrete control and status logic that provides the interface to modems or other external devices.

The read and write register group includes five 8-bit control registers, two sync-character registers and two status registers. The interrupt vector is written into an additional 8-bit register (Write Register 2) in Channel B that may be read through Read Register 2 in Channel B. The registers for both channels are designated in the text as follows:

WR7-WR0 — Write Registers 0 through 7

RR2-RR0 — Read Registers 0 through 2

The bit assignment and functional grouping of each register is configured to simplify and organize the programming process. Tables 3-1 and 3-2 illustrate the functions assigned to each read or write register.

The logic for both channels provides formats, synchronization and validation for data transferred to and from the channel interface. The modem control inputs Clear to Send (/CTS) and Data Carrier Detect (/DCD) are monitored by the discrete control logic under program control. All the modem control signals are general purpose in nature and can be used for functions other than modem control.

For automatic interrupt vectoring, the interrupt control logic determines which channel and which device within the channel has the highest priority. Priority is fixed with Channel A assigned a higher priority than Channel B; Receive, Transmit and External/ Status interrupts are prioritized in that order within each channel.

Table 3-1. Write Register Functions

Bit	Function
WR0	Register pointers, CRC initialize, initialization commands for the various modes, etc.
WR1	Transmit/Receive interrupt and data transfer mode definition.
WR2	Interrupt vector (Channel 8 only)
WR3	Receive parameters and controls
WR4	Transmit/Receive miscellaneous parameters and modes
WR5	Transmit parameters and controls
WR6	Sync character or SDLC address field
WR7	Sync character or SDLC flag

Table 3-2. Read Register Functions

Bit	Function
RR0	Transmit/Receive buffer status, interrupt status, and external status
RR1	Special Receive Condition status
RR2	Modified interrupt vector (Channel B only)

3.1 DATA PATH

The transmit and receive data path for each channel is shown in Figure 3-1. The receiver has three 8-bit buffer registers in a FIFO arrangement (to provide a 3-byte delay) in addition to the 8-bit receive shift register. This arrangement creates additional time for the CPU to service an interrupt at the beginning of a block of high-speed data. The receive error FIFO stores parity and framing errors and other types of status information for each of the three bytes in the receive data FIFO.

Incoming data is routed through one of several paths depending on the mode and character length. In the Asynchronous mode, serial data is entered in the 3-bit buffer if it has a character length of seven or eight bits, or is entered in the 8-bit receive shift register if it has a length of five or six bits.

In the Synchronous mode, however, the data path is determined by the phase of the receive process currently in operation. A Synchronous Receive operation begins with the receiver in the Hunt phase, during which the receiver searches the incoming data stream for a bit pattern that matches the preprogrammed sync characters (or flags in the SDLC mode). If the device is programmed for Monosync Hunt, a match is made with a single sync character stored in WR7. In Bisync Hunt, a match is made with dual sync characters stored in WR6 and WR7.

In either case (he incoming data passes through the receive sync register, and is compared against the programmed sync character in WR6 or WR7. In the Monosync mode, a match between the sync character programmed into WR7 and the character assembled in the receive sync register establishes synchronization.

In the Bisync mode, however, incoming data is shifted to the receive shift register while the next eight bits of the message are assembled in the receive sync register. The match between the assembled character in the receive sync registers with the programmed sync character in WR6 and WR7 establishes synchronization. Once synchronization is established, incoming data bypasses the receive sync register and directly enters the 3-bit buffer.

In the SDLC mode, incoming data first passes through the receive sync register, which continuously monitors the receive data stream and performs zero deletion when indicated. Upon receiving five contiguous 1's, the sixth bit is inspected. If the sixth bit is a 0, it is deleted from the data stream. If the sixth bit is a 1, the seventh bit is inspected. If that bit is a 0, a Flag sequence has been received; if it is a 1, an Abort sequence has been received.

The reformatted data enters the 3-bit buffer and is transferred to the receive shift register. Note that the SDLC receive operation also begins in the Hunt phase, during which the Z80-SIO tries to match the assembled character in the receive shift register with the nag pattern in WR7. Once the first flag character is recognized, all subsequent data is routed through the same path, regardless of character length.

Although the same CRC checker is used for both SDLC and synchronous data, the data path taken for each mode is different. In Bisync protocol, a byte-oriented operation requires that the CPU decide to include the data character in CRC. To allow the CPU ample time to make this decision, the Z80-SIO provides an 8-bit delay for synchronous data. In the SDLC mode, no delay is provided since the Z80-SIO contains logic that determines the bytes on which CRC is calculated.

The transmitter has an 8-bit transmit data register that is loaded from the internal data bus and a 20-bit transmit shift register that can be loaded from WR6, WR7 and the transmit data register. WR6 and WR7 contain sync characters in the Monosync or Bisync modes, or address field (one character long) and flag respectively in the SDLC mode. During Synchronous modes, information contained in WR6 and WR7 is loaded into the transmit shift register at the beginning of the message and, as a time filler, in the middle of the message if a Transmit Underrun condition occurs. In the SDLC mode, the flags are loaded into the transmit shift register at the beginning and end of message.

Asynchronous data in the transmit shift register is formatted with start and stop bits and is shifted out to the transmit multiplexer at the selected clock rate. Synchronous (Monosync or Bisync) data is shifted out to the transmit multiplexer and also to the CRC generator at the x1 clock rate.

SDLC/HDLC data is shifted out through the zero insertion logic, which is disabled while the flags are being sent. For all other fields (address, control, and frame check) a 0 is inserted following five contiguous 1's in the data stream. The CRC generator result for SDLC data is also routed through the zero insertion logic.

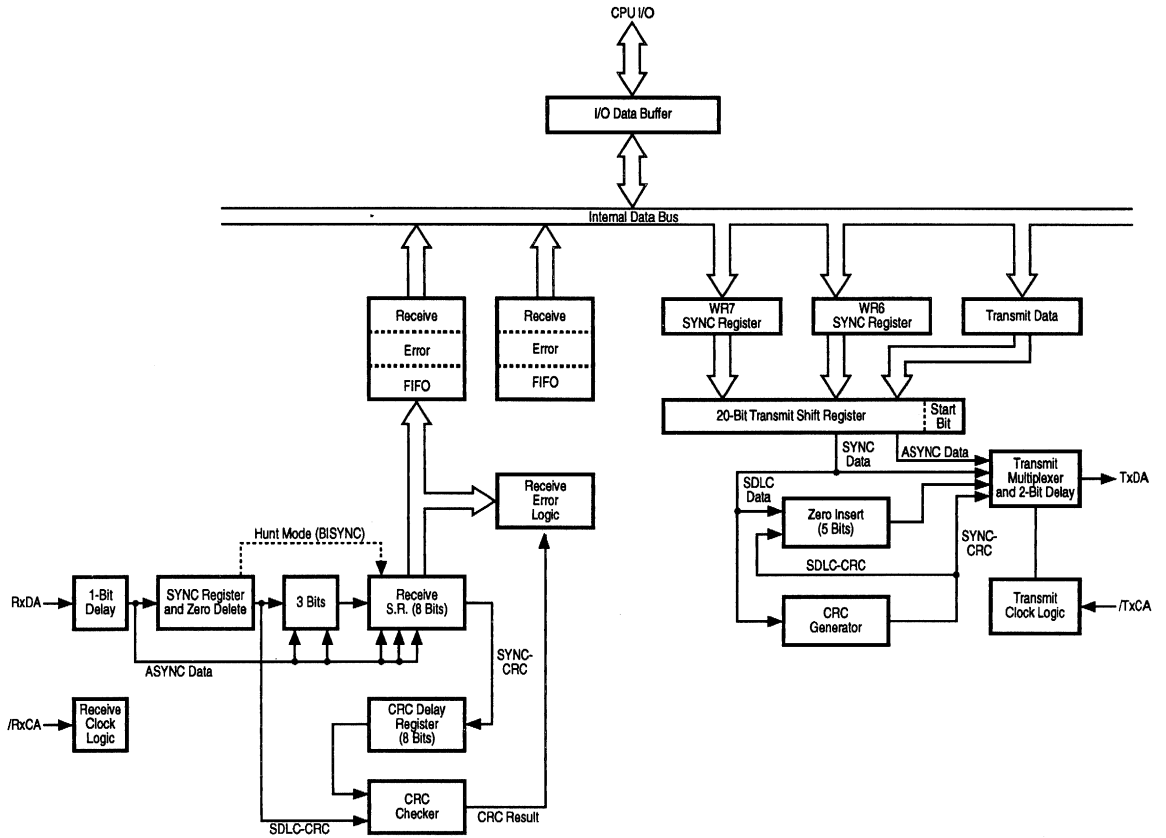


Figure 3-1. Transmit and Receive Data Path

E

3.2 FUNCTIONAL DESCRIPTION

The functional capabilities of the Z80-SIO can be described from two different points of view: as a data communications device, it transmits and receives serial data, and meets the requirements of various data communications protocols; as a Z80 family peripheral, it interacts with the Z80-CPU and other Z80 peripheral, circuits, and shares their data, address and control busses, as well as being a part of the Z80 interrupt structure. As a peripheral to other microprocessors, the Z80-SIO offers valuable features such as non-vectored interrupts, polling and simple handshake capabilities.

The first part of the following functional description describes the interaction between the CPU and Z80-SIO; the second part introduces its data communications capabilities.

3.2.1 I/O Capabilities

The Z80-SIO offers the choice of Polling, Interrupt (vectored or non-vectored) and Block Transfer modes to transfer data, status and control information to and from the CPU. The Block Transfer mode can be implemented under CPU or DMA control.

Polling. The Polled mode avoids interrupts. Status registers RR0 and RR1 are updated at appropriate times for each function being performed (for example, CRC Error status valid at the end of the message). All the interrupt modes of the Z80-SIO must be disabled to operate the device in a polled environment.

While in its Polling sequence, the CPU examines the status contained in RR0 for each channel; the RR0 status bits serve as an acknowledge to the Poll inquiry. The two RR0 status bits D0 and D2 indicate that a receive or transmit data transfer is needed. The status also indicates Error or other special status conditions (see "Z80-SIO Programming"). The Special Receive Condition status contained in RR1 does not have to be read in a Polling sequence because the status bits in RR1 are accompanied by a Receive Character Available status in RR0.

Interrupts. The Z80-SIO offers an elaborate interrupt scheme to provide fast interrupt response in real-time applications. As mentioned earlier, Channel B registers WR2 and RR2 contain the interrupt vector that points to an interrupt service routine in the memory. To service operations in both channels and to eliminate the necessity of writing a status analysis routine, the Z80-SIO can modify the interrupt vector in RR2 so it points directly to one of eight interrupt service routines. This is done under program control by setting a program bit (WR1, D2) in Channel B called "Status Affects Vector." When this bit is set, the interrupt vector in WR2 is modified according to the assigned priority of the various interrupting conditions. The table in the Write Register 1 description (Z80-SIO Programming section) shows the modification details.

Transmit interrupts, Receive interrupts, and External/Status interrupts are the main sources of interrupts (Figure 3-2). Each interrupt source is enabled under program control with Channel A having a higher priority than Channel B, and with Receiver, Transmit, and External/Status interrupts prioritized in that order within each channel. When the Transmit interrupt is enabled, the CPU is interrupted by the transmit buffer becoming empty. (This implies that the transmitter must have had a data character written into it so it can become empty.) When enabled, the receiver can interrupt the CPU in one of three ways:

- Interrupt on first receive character
- Interrupt on all receive characters
- Interrupt on a Special Receive condition

Interrupt On First Character is typically used with the Block Transfer mode. Interrupt On All Receive Characters has the option of modifying the interrupt vector in the event of a parity error. The Special Receive Condition interrupt can occur on a character or message basis (End-of-Frame interrupt in SDLC, for example). The Special Receive condition can cause an interrupt only if the Interrupt On First Receive Character or Interrupt On All Receive Characters mode is selected. In Interrupt On First Receive Character, an interrupt can occur from Special Receive conditions (except Parity Error) after the first receive character interrupt (example: Receive Overrun interrupt).

The main function of the External/Status interrupt is to monitor the signal transitions of the /CTS, /DCD, and /SYNC pins; however, an External/Status interrupt is also caused by a Transmit Underrun condition or by the detection of a Break (Asynchronous mode) or Abort (SDLC mode) sequence in the data stream. The interrupt caused by the Break/Abort sequence has a special feature that allows the Z80-SIO to interrupt when the Break/Abort sequence is detected or terminated. This feature facilitates the proper termination of the current message, correct initialization of the next message, and the accurate timing of the Break/Abort condition in external logic.

CPU/DMA Block Transfer. The Z80-SIO provides a Block Transfer mode to accommodate CPU block transfer functions and DMA controllers (Z80-DMA or other designs). The Block Transfer mode uses the /WAIT//READY output in conjunction with the Wait/Ready bits of Write Register 1. The /WAIT//READY output can be defined under software control as a /WAIT line in the CPU Block Transfer mode or as a /READY line in the DMA Block Transfer mode.

To a DMA controller, the Z80-SIO /READY output indicates that the Z80-SIO is ready to transfer data to or from memory. To the CPU, the /WAIT output indicates that the Z80-SIO is not ready to transfer data, thereby requesting the CPU to extend the I/O cycle. The programming of bits 5, 6, and 7 of Write Register 1 and the logic states of the /WAIT//READY line are defined in the Write Register 1 description (Z80-SIO Programming section).

3.2.2 Data Communications Capabilities

In addition to the I/O capabilities previously discussed, the Z80-SIO provides two independent full-duplex channels as well as Asynchronous, Synchronous, and SDLC (HDLC) operational modes. These modes facilitate the implementation of commonly used data communications protocols.

The specific features of these modes are described in the following sections. To preserve the independence and completeness of each section, some information common to all modes is repeated.

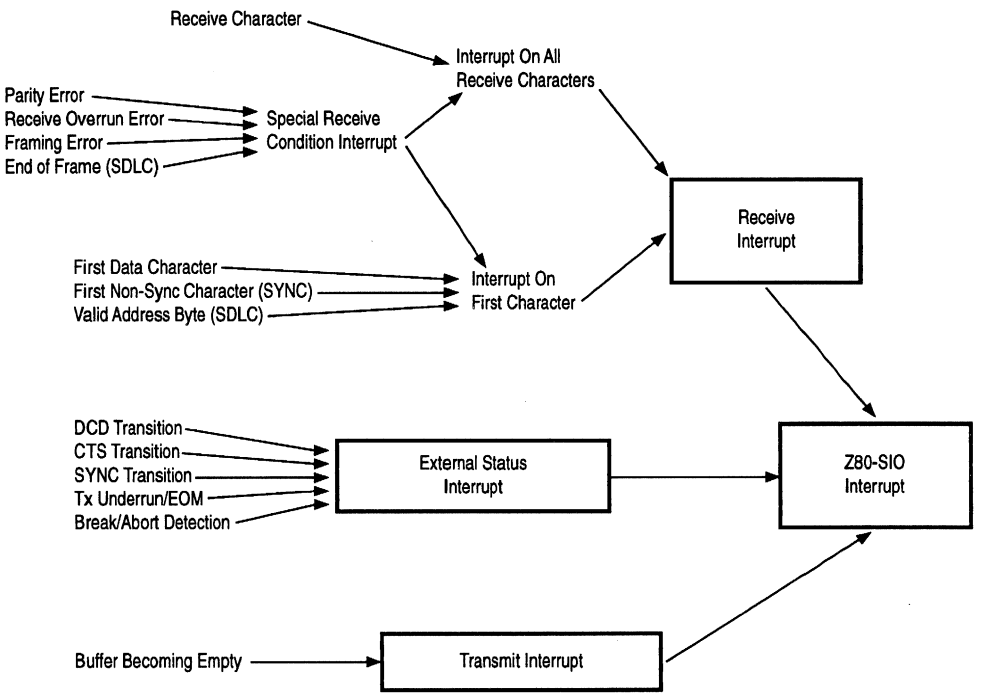


Figure 3-2. Interrupt Structure

E

CHAPTER 4
ASYNCHRONOUS OPERATION

4.0 INTRODUCTION

To receive or transmit data in the Asynchronous mode, the Z80-SIO must be initialized with the following parameters: character length, clock rate, number of stop bits, even or odd parity, interrupt mode, and receiver or transmitter enable. The parameters are loaded into the appropriate write registers by the system program. WR4 parameters must be issued before WR1, WR3, and WR5 parameters or commands.

If the data is transmitted over a modem or RS232C interface, the REQUEST TO SEND (/RTS) and DATA TERMINAL READY (/DTR) outputs must be set along with the Transmit Enable bit. Transmission cannot begin until the Transmit Enable bit is set.

The Auto Enables feature allows the programmer to send the first data character of the message to the Z80-SIO without waiting for /CTS. If the Auto Enables bit is set, the Z80-SIO will wait for the /CTS pin to go Low before it begins data transmission. /CTS, /DCD, and /SYNC are general-purpose I/O lines that may be used for functions other than their labeled purposes. If /CTS is used for another purpose, the Auto Enables Bit must be programmed to 0.

Figure 4-1 illustrates asynchronous message formats; Table 4-1 shows WR3, WR4, and WR5 with bits set to indicate the applicable modes, parameters and commands in asynchronous modes. WR2 (Channel B only) stores the interrupt vector; WR1 defines the interrupt modes and data transfer modes. WR6 and WR7 are not used in asynchronous modes. Table 4-2 shows the typical program steps that implement a full-duplex receive/transmit operation in either channel.

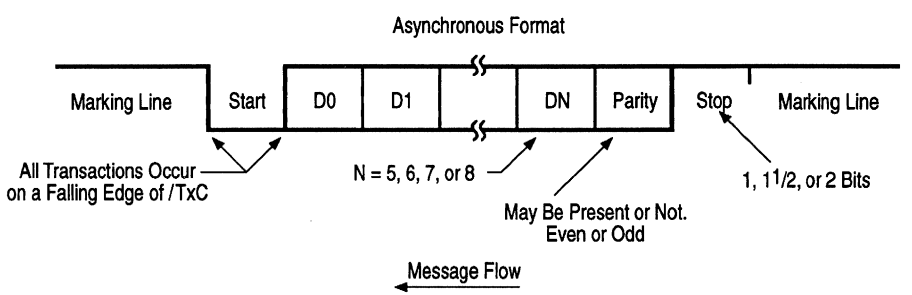


Figure 4-1. Asynchronous Message Format

4.1 ASYNCHRONOUS TRANSMIT

The Transmit Data output (TxD) is held marking (High) when the transmitter has no data to send. Under program control, the Send Break (WR5, D4) command can be issued to hold TxD spacing (Low) until the command is cleared.

The Z80-SIO automatically adds the start bit, the programmed parity bit (odd, even, or no parity) and the programmed number of stop bits to the data character to be transmitted. When the character length is six or seven bits, the unused bits are automatically ignored by the Z80-SIO. If the character length is five bits or less, refer to the table in the Write Register 5 description (Z80-SIO Programming section) for the data format.

Serial data is shifted from TxD at a rate equal to 1, 1/16th, 1/32nd, or 1/64th of the clock rate supplied to the Transmit Clock input /TxC. Serial data is shifted out on the falling edge of /TxC.

If set, the External/Status Interrupt mode monitors the status of /DCD, /CTS, and /SYNC throughout the transmission of the message. If these inputs change for a period of time greater than the minimum specified pulse width, the interrupt is generated. In a transmit operation, this feature is used to monitor the modem control signal /CTS.

Table 4-1. Contents of Write Registers 3, 4, and 5 in Asynchronous Modes

	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
WR3	00 = Rx 5 BITS/CHAR 10 = Rx 6 BITS/CHAR 01 = Rx 7 BITS/CHAR 11 = Rx 8 BITS/CHAR		AUTO ENABLES	0	0	0	0	Rx ENABLE
WR4	00 = x1 CLOCK MODE 01 = x16 CLOCK MODE 10 = 32 CLOCK MODE 11 x64 CLOCK MODE		0	0	00 = NOT USED 01 = 1 STOP BIT/CHAR 10 = 1-1/2 STOP BITS/CHAR 11 = 2 STOP BITS/CHAR		EVEN//ODD PARITY	PARITY ENABLE
WR5	DTR	00 = Tx 5 BITS (OR LESS)/CHAR 10 = Tx 6 BITS/CHAR 01 = Tx 7 BITS/CHAR 11 = Tx 8 BITS/CHAR		SEND BREAK	Tx ENABLE	0	RTS	0

Table 4-2. Asynchronous Mode

Function		Typical Program Steps	Comments
	REGISTER:	INFORMATION LOADED:	
INITIALIZE	WR0	CHANNEL RESET	Reset SIO
	WR0	POINTER 2	
	WR2	INTERRUPT VECTOR	Channel B only
	WR0	POINTER 4, RESET EXTERNAL/STATUS INTERRUPT	
	WR4	ASYNCHRONOUS MODE, PARITY INFORMATION, STOP BITS INFORMATION, CLOCK RATE INFORMATION	Issue parameters
	WR0	POINTER 3	
	WR3	RECEIVE ENABLE, AUTO ENABLES, RECEIVE CHARACTER LENGTH	
	WR0	POINTER 5	
	WR5	REQUEST TO SEND, TRANSMIT ENABLE, TRANSMIT CHARACTER LENGTH, DATA TERMINAL READY	Receive and Transmit both fully initialized. Auto Enables will enable Transmitter if /CTS is active and Receiver if /DCD is active.
	WR0	POINTER 1, RESET EXTERNAL/STATUS INTERRUPT	
	WR1	TRANSMIT INTERRUPT ENABLE, STATUS AFFECTS VECTOR, INTERRUPT ON ALL RECEIVE CHARACTERS. DISABLE WAIT/ READY FUNCTION, EXTERNAL INTERRUPT ENABLE	Transmit/Receive interrupt mode selected. External Interrupt monitors the status of the /CTS, /DCD, and /SYNC inputs and detects the Break sequence. Status Affects Vector in Channel B only. This data byte must be transferred or no transmit interrupts will occur.
		TRANSFER FIRST DATA BYTE TO SIO	
IDLE MODE		EXECUTE HALT INSTRUCTION OR SOME OTHER PROGRAM	Program is waiting for an interrupt from the SIO.
DATA TRANSFER AND ERROR MONITORING		Z80 INTERRUPT ACKNOWLEDGE CYCLE TRANSFERS RR2 TO CPU	When the Interrupt occurs, the interrupt vector is modified by: 1. Receive Character Available; 2. Transmit Buffer Empty; 3. External/Status change; and 4. Special Receive condition.
		IF A CHARACTER IS RECEIVED: ■ TRANSFER DATA CHARACTER TO CPU ■ UPDATE POINTERS AND PARAMETERS ■ RETURN FROM INTERRUPT	
		IF TRANSMITTER BUFFER IS EMPTY,- TRANSFER DATA CHARACTER TO SIO UPDATE POINTERS AND PARAMETERS RETURN FROM INTERRUPT	Program control is transferred to one of the eight interrupt Service routines.
		IF EXTERNAL STATUS CHANGES: ■ TRANSFER RR0 TO CPU ■ PERFORM ERROR ROUTINES (INCLUDE BREAK DETECTION) ■ RETURN FROM INTERRUPT	If used with processors other than the Z80. the modified interrupt vector (RR2) should be returned to the CPU in the Interrupt Acknowledge sequence.
		IF SPECIAL RECEIVE CONDITION OCCURS. ■ TRANSFER RR1 TO CPU ■ D6 SPECIAL ERROR (E.G. FRAMING ERROR) ROUTINE ■ RETURN FROM INTERRUPT	
TERMINATION		REDEFINE RECEIVE/TRANSMIT INTERRUPT MODES	When transmit or receive data transfer is complete.
		DISABLE TRANSMITIRECEIVE MODES UPDATE MODEM CONTROL OUTPUTS (E.G. RTS OFF)	In Transmit, the All Sent status bit indicates transmission is complete.

E

4.2 ASYNCHRONOUS RECEIVE

An Asynchronous Receive operation begins when the Receive Enable bit is set. If the Auto Enables option is selected, /DCD must be Low as well. A Low (spacing) condition on the Receive Data input (RxD) indicates a start bit. If this Low persists for at least one-half of a bit time, the start bit is assumed to be valid and the data input is then sampled at mid-bit time until the entire character is assembled. This method of detecting a start bit improves error rejection when noise spikes exist on an otherwise marking line.

If the x1 clock mode is selected, bit synchronization must be accomplished externally. Receive data is sampled on the rising edge of RxC. The receiver inserts I's when a character length of other than eight bits is used. If parity is enabled, the parity bit is not stripped from the assembled character for character lengths other than eight bits. For lengths other than eight bits, the receiver assembles a character length of the required number of data bits, plus a parity bit and I's for any unused bits. For example, the receiver assembles a 5-bit character with the following format: 11 P D4 D3 D2 D1 D0.

Since the receiver is buffered by three 8-bit registers in addition to the receive shift register, the CPU has enough time to service an interrupt and to accept the data character assembled by the Z80-SIO. The receiver also has three buffers that store error nags for each data character in the receive buffer. These error flags arc loaded at the same time as the data characters.

After a character is received, it is checked for the following error conditions:

- When parity is enabled, the Parity Error bit (RR1, D4) is set whenever the parity bit of the character does not match with the programmed parity. Once this bit is set, it remains set until the Error Reset Command (WR0) is given.
- The Framing Error bit (RR1, D6) is set if the character is assembled without any stop bits (that is, a Low level detected for a stop bit). Unlike the Parity Error bit, this bit is set (and not latched) only for the character on which it occurred. Detection of framing error adds an additional one-half of a bit time to the character time so the framing error is not interpreted as a new start bit.
- If the CPU fails to read a data character while more than three characters have been received, the Receive Overrun bit (RR1, D5) is set. When this occurs, the fourth character assembled replaces the third character in the receive buffers. With this arrangement, only the character that has been written over is flagged with the Receive Overrun Error bit. Like Parity Error, this bit can only be reset by the Error Reset command from the CPU. Both the Framing Error and Receive Overrun Error cause an interrupt with the interrupt vector indicating a Special Receive condition (if Status Affects Vector is selected).

Since the Parity Error and Receive Overrun Error flags are latched, the error status that is read reflects an error in the current word in the receive buffer plus any Parity or Overrun Errors received since the last Error Reset command. To keep correspondence between the state of the error buffers and the contents of the receive data buffers, the error status register must be read before the data. This is easily accomplished if vectored interrupts are used, because a special interrupt vector is generated for these conditions.

While the External/Status interrupt is enabled, break detection causes an interrupt and the Break Detected status bit (RR0, D7) is set. The Break Detected interrupt should be handled by issuing the Reset External/Status Interrupt command to the Z80-SIO in response to the first Break Detected interrupt that has a Break status of 1 (RR0, D7). The Z80-SIO monitors the Receive Data input and waits for the Break sequence to terminate, at which point the Z80-SIO interrupts the CPU with the Break status set to 0. The CPU must again issue the Reset External/Status Interrupt command in its interrupt service routine to reinitialize the break detection logic.

The External/Status interrupt also monitors the status of /DCD. If the /DCD pin becomes inactive for a period greater than the minimum specified pulse width, an interrupt is generated with the /DCD status bit (RR0, D3) set to 1. Note that the /DCD input is inverted in the RR0 status register.

If the status is read after the data, the error data for the next word is also included if it has been stacked in the buffer. If operations are performed rapidly enough so the next character is not yet received, the status register remains valid. An exception occurs when the Interrupt On First Character Only mode is selected. A special interrupt in this mode holds the error data and the character itself (even if read from the buffer) until the Error Reset command is issued. This prevents further data from becoming available in the receiver until the Reset command is issued, and allows CPU intervention on the character with the error even if DMA or block transfer techniques are being used.

If Interrupt On Every Character is selected, the interrupt vector is different if there is an error status in RR1. If a Receiver Overrun occurs, the most recent character received is loaded into the buffer; the character preceding it is lost. When the character that has been written over the other characters is read, the Receive Overrun bit is set and the Special Receive Condition vector is returned if Status Affects Vector is enabled.

In a polled environment, the Receive Character Available bit (RR0, D0) must be monitored so the Z80-CPU can know when to read a character. This bit is automatically reset when the receive buffers are read. To prevent overwriting data in polled operations, the transmit buffer status must be cheeked before writing into the transmitter. The Transmit Buffer Empty bit is set to 1 whenever the transmit buffer is empty.

The External/Status interrupt also monitors the status of /DCD. If the /DCD pin becomes inactive for a period greater than the minimum specified pulse width, an interrupt is generated with the /DCD status bit (RR0, D3) set to 1. Note that the /DCD input is inverted in the RR0 status register.

If the status is read after the data, the error data for the next word is also included if it has been stacked in the buffer. If operations are performed rapidly enough so the next character is not yet received, the status register remains valid. An exception occurs when the Interrupt On First Character Only mode is selected. A special interrupt in this mode holds the error data and the character itself (even if read from the buffer) until the Error Reset command is issued. This prevents further data from becoming available in the receiver until the Reset command is issued, and allows CPU intervention on the character with the error even if DMA or block transfer techniques are being used.

If Interrupt On Every Character is selected, the interrupt vector is different if there is an error status in RR1. If a Receiver Overrun occurs, the most recent character received is loaded into the buffer; the character preceding it is lost. When the character that has been written over the other characters is read, the Receive Overrun bit is set and the Special Receive Condition vector is returned if Status Affects Vector is enabled.

In a polled environment, the Receive Character Available bit (RR0, D0) must be monitored so the Z80-CPU can know when to read a character. This bit is automatically reset when the receive buffers are read. To prevent overwriting data in polled operations, the transmit buffer status must be cheeked before writing into the transmitter. The Transmit Buffer Empty bit is set to 1 whenever the transmit buffer is empty.

CHAPTER 5
SYNCHRONOUS OPERATION

5.0 INTRODUCTION

Before describing synchronous transmission and reception, the three types of character synchronization, Monosync, Bisync, and External Sync, require some explanation. These modes use the x1 clock for both Transmit and Receive operations. Data is sampled on the rising edge of the Receive Clock input (/RxC). Transmitter data transitions occur on the falling edge of the Transmit Clock input (/TxC).

The differences between Monosync, Bisync, and External Sync are in the manner in which initial character synchronization is achieved. The mode of operation must be selected before sync characters are loaded, because the registers are used differently in the various modes. Figure 5-1 shows the formats for all three of these synchronous modes.

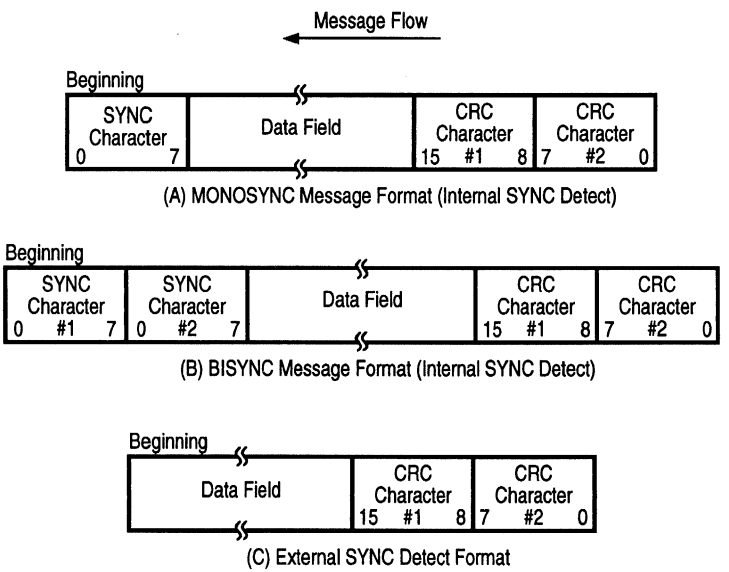


Figure 5-1. Synchronous Formats

5.1 SYNCHRONOUS MODES OF OPERATION

Monosync. In a Receive operation, matching a single sync character (8-bit sync mode) with the programmed sync character stored in WR7 implies character synchronization and enables data transfer.

Bisync. Matching two contiguous sync characters (16-bit sync mode) with the programmed sync characters stored in WR6 and WR7 implies character synchronization. In both the Monosync and Bisync modes, /SYNC is used as an output, and is active for the part of the receive clock that detects the sync character.

External Sync. In this mode, character synchronization is established externally; /SYNC is an input that indicates external character synchronization has been achieved. After the sync pattern is detected, the external logic must wait for two full Receive Clock cycles to activate the /SYNC input. The /SYNC input must be held Low until character synchronization is lost. Character assembly begins on the rising edge of /RxC that precedes the falling edge of /SYNC.

In all cases after a reset, the receiver is in the Hunt phase, during which the Z80-SIO looks for character synchronization. The hunt can begin only when the receiver is enabled, and data transfer can begin only when character synchronization has been achieved. If character synchronization is lost, the Hunt phase can be re-entered by writing a control word with the Enter Hunt Phase bit set (WR3, D4). In the Transmit mode, the transmitter always sends the programmed number of sync bits (8 or 16). In the Monosync mode, the transmitter transmits from WR6; the receiver compares against WR7.

In the Monosync, Bisync, and External Sync modes, assembly of received data continues until the Z80-SIO is reset, or until the receiver is disabled (by command or by DCD in the Auto Enables mode), or until the CPU sets the Enter Hunt Phase bit.

After initial synchronization has been achieved, the operation of the Monosync, Bisync, and External Sync modes is quite similar. Any differences are specified in the following text.

Table 5-1 shows how WR3, WR4, and WR5 are used in synchronous receive and transmit operations. WR0 points to other registers and issues various commands, WR1 defines the interrupt modes, WR2 stores the interrupt vector, and WR6 and WR7 store sync characters. Table 5-2 illustrates the typical program steps that implement a half-duplex Bisync transmit operation.

Table 5-1. Contents of Write Registers 3, 4, and 5 In Synchronous Modes

	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT1	BIT 0
WR3	00 = Ax 5 BITS/CHAR 10 = Rx 6 BITS/CHAR 01 = Rx 7 BITS/CHAR 11 = Rx 8 BITS/CHAR		AUTO ENABLES	ENTER HUNT MODE	Rx CRC ENABLE	0	SYNC CHAR LOAD INHIBIT	RX ENABLE
WR4	0	0	00 = 8-BIT SYNC CHAR 01 = 16-BIT SYNC CHAR 10 = SDLC MODE 11 = EXT SYNC MODE		0	0	EVEN//ODD PARITY	PARITY ENABLE
					SELECTS SYNC MODES			
WR5	DTR	00 = Tx 5 BITS (OR LESS)/CHAR 10 = Tx 6 BITS/CHAR 01 = Tx 7 BITS/CHAR 11 = Tx 8 BITS/CHAR		SEND BREAK	Tx ENABLE	1 SELECTS CRC-16	RTS	TX CRC ENABLE

Table 5-2. Bisync Transmit Mode

FUNCTION		TYPICAL PROGRAM STEPS	COMMENTS
	REGISTER:	INFORMATION LOADED.	
INITIALIZE	WR0	CHANNEL RESET, RESET TRANSMIT CRC GENERATOR	Reset SIO, initilize CRC generator,
	WR0	POINTER 2	
	WR2	INTERRUPT VECTOR	Channel B only
	WR0	POINTER 3	
	WR3	AUTO ENABLES	Transmission begins only after /CTS is detected.
	WR0	POINTER 4	
	WR4	PARITY INFORMATION, SYNC MODES INFORMATION, x1 CLOCK MODE	Issue transmit parameters.
	WR0	POINTER 6	
	WR6	SYNC CHARACTER 1	
	WR0	POINTER 7, RESET EXTERNAL/STATUS INTERRUPTS	
	WR7	SYNC CHARACTER 2	
	WR0	POINTER 1, RESET EXTERNAL/STATUS INTERRUPTS	
	WR1	STATUS AFFECTS VECTOR, EXTERNAL INTERRUPT ENABLE, TRANSMIT INTERRUPT ENABLE OR WAIT/READY MODE ENABLE	External Interrupt mode monitors the status of /CTS and /DCD input pins as well as the status of Tx Underrun/EOM latch. Transmit Interrupt Enable interrupts when the Transmit buffer becomes empty; the Wait/Ready mode can be used to transfer data using DMA or CPu Block Transfer.
	WR0	POINTER 5	Status Affects Vector (Channel B only).
	WR5	REQUEST TO SEND, TRANSMIT ENABLE, BSYNC CRC, TRANSMIT CHARACTER LENGTH FIRST SYNC BYTE TO SIO	Transmit CRC Enable should be set when first non-sync data is sent to Z80-SIO. Need several sync characters in the beginning of message. Transmitter is fully initialized.
IDLE MODE		EXECUTE HALT INSTRUCTION OR SOME OTHER PROGRAM	Waiting for interrupt or Wait/Ready output to transfer data.
DATA TRANSFER AND STATUS MONITORING		WHEN INTERRUPT (WAIT/READY) OCCURS: ■ INCLUDE/EXECLUDE DATA BYTE FROM CRC ACCUMULATION (IN SIO). ■ TRANSFER DATA BYTE FROM CPU (OR MEMORY) TO SIO. ■ DETECT AND SET APPROPRIATE FLAGS FOR CONTROL CHARACTERS (IN CPU) ■ RESET Tx UNDERRUN/EOM LATCH (WR0) IF LAST CHARACTER OF MESSAGE IS DETECTED. ■ UPDATE POINTERS AND PARAMETERs (CPU). RETURN FROM INTERRUPT. IF ERROR CONDITION OR STATUS CHANGE OCCURS: ■ TRANSFER RR0 TO CPU. ■ EXECUTE ERROR ROUTINE. ■ RETURN FROM INTERRUPT.	Interrupt occurs (Wait/Ready becomes active) when first data byte is being sent, Wail mode allows CPU block transfer from memory to sio; Ready mode allows DMA block transfer from memory to sio. The DMA Chip can be programmed to capture special control characters (by examining only the bits that specify ASCII or EBCDIC control characters), and interrupt CPU. Tx Underrun/EOM indicates either transmit underrun (sync character being sent) or end of message (CRC-16 being sent).
TERMINATION		REDEFINE INTERRUPT MODES, UPDATE MODEM CONTROL OUTPUTS (E.G., TURN OFF /RTS). DISABLE TRANSMIT MODE	Program should gracefully terminate message.

E

5.2 SYNCHRONOUS TRANSMIT

5.2.1 Initialization

The system program must initialize the transmitter with the following parameters: odd or even parity, x1 clock mode, 8-bit or 16-bit sync character(s), CRC polynomial, Transmitter Enables, Request To Send, Data Terminal Ready, interrupt modes and transmit character length., WR4 parameters must be issued before WR1, WR3, WR5, WR6, and WR7 parameters or commands.

One of two polynomials, CRC – 16 ($X^{16} + X^{15} + X^2 + 1$) or SDLC ($X^{16} + X^{12} + X^5 + 1$), may be used with synchronous modes. In either case (SDLC mode not selected), the CRC generator and checker are reset to all 0's. In the transmit initialization process, the CRC generator is initialized by setting the Reset Transmit CRC Generator command bits (WR0). Both the transmitter and the receiver use the same polynomial.

Transmit Interrupt Enable or Wait/Ready Enable can be selected to transfer the data. The External/Status interrupt mode is used to monitor the status of the CLEAR TO SEND (/CTS) input as well as the Transmit Underrun/EOM latch. Optionally, the Auto Enables feature can be used to enable the transmitter when /CTS is active. The first data transfer to the Z80-SIO can begin when the External/Status interrupt occurs (/CTS status bit set) or immediately following the Transmit Enable command (if the Auto Enables modes is set).

Transmit data is held marking after reset or if the transmitter is not enabled. Break may be programmed to generate a spacing line that begins as soon as the Send Break bit is set. With the transmitter fully initialized and enabled, the default condition is continuous transmission of the 8-bit or 16-bit sync character.

5.2.2 Data Transfer and Status Monitoring

In this phase, there are several combinations of interrupts and Wait/Ready.

Data Transfer Using Interrupts. If the Transmit Interrupt Enable bit (WR1, D1) is Set, an interrupt is generated each time the transmit buffer becomes empty. The interrupt can be satisfied either by writing another character into the transmitter or by resetting the Transmitter Interrupt Pending latch with a Reset Transmitter Pending command (WR0, CMD5). If the interrupt is satisfied with this command and nothing more is written into the transmitter, there can be no further Transmit Buffer Empty interrupts, because it is the process of the buffer becoming empty that causes the interrupts and the buffer cannot become empty when it is already empty. This situation does cause a Transmit Underrun condition, which is explained in the "Bisync Transmit Underrun" section.

Data Transfer Using /WAIT//READY. To the CPU, the activation of /WAIT indicates that the Z80-SIO is not ready to accept data and that the CPU must extend the output cycle. To a DMA controller, /READY indicates that the transmit buffer is empty and that the Z80-SIO is ready to accept the next data character. If the data character is not loaded into the Z80-SIO by the time the transmit shift register is empty, the Z80-SIO enters the Transmit Underrun condition.

Bisync Transmit Underrun. In Bisync protocol, filler characters are inserted to maintain synchronization when the transmitter has no data to send (Transmit Underrun condition). The Z80-SIO has two programmable options for solving this situation: it can insert sync characters, or it can send the CRC characters generated so far, followed by sync characters.

These options are under the control of tile Reset Transmit Underrun/EOM Command in WR0. Following a chip or channel reset, the Transmit Underrun/EOM status bit (RR0, D6) is in a set condition and allows the insertion of sync characters when there is no data to send. CRC is not calculated on the automatically inserted sync characters. When the CPU detects the end of message, a Reset Transmit Underrun/EOM command can be issued. This allows CRC to be sent when the transmitter has no data. In this case, the Z80-SIO sends CRC, followed by sync characters, to terminate the message.

There is no restriction as to when in the message the Transmit Underrun/EOM bit can be reset. If Reset is issued after the first data character has been loaded the 16-bit CRC is Sent and followed by sync characters the first time the transmitter has no data to send. Because of the Transmit Underrun condition, an External/Status interrupt is generated whenever the Transmit Underrun/EOM bit becomes set.

In the case of sync insertion, an interrupt is generated only after the first automatically inserted sync character has been loaded. The status indicates the Transmit Underrun/EOM bit and the Transmit Buffer Empty bit are set.

In the case of CRC insertion, the Transmit Underrun/ EOM bit is set and the Transmit Buffer Empty bit is reset while CRC is being sent. When CRC has been completely sent,

the Transmit Buffer Empty status bit is set and an interrupt is generated to indicate to the CPU that another message can begin (this interrupt occurs because CRC has been sent and sync has been loaded). If no more messages are to be sent, the program can terminate transmission by resetting RTS, and disabling the transmitter (WR5, D3).

Pad characters may be sent by setting the Z80-SIO to eight bits/transmit character and writing FF to the transmitter while CRC is being sent. Alternatively, the sync characters can be redefined as pad characters during this time. The following example is included to clarify this point.

- The Z80-SIO interrupts with the Transmit Buffer Empty bit set.
- The CPU recognizes that the last character (ETX) of the message has already been sent to the Z80-SIO by examining the internal program status.
- To force the Z80-SIO to send CRC, the CPU issues the Reset Transmit Underrun/EOM Latch command (WR0) and satisfies the interrupt with the Reset Transmit Interrupt Pending command. (This command prevents the Z80-SIO from requesting more data.) Because of the transmit underrun caused by this command, the Z80-SIO starts sending CRC. The Z80-SIO also causes an External/Status interrupt with the Transmit Underrun/EOM latch set.
- The CPU satisfies this interrupt by loading pad characters into the transmit buffer and issuing the Reset External/Status Interrupt command.
- With this sequence, CRC is followed by a pad character instead of a sync character. Note that the Z80-SIO will interrupt with a Transmit Buffer Empty interrupt when CRC is completely sent and that the pad character is loaded into the transmit shift register.
- From this point on the CPU can send more pad characters or sync characters.

Bisync CRC Generation. Setting the Transmit CRC enable bit (WR5, D0) initiates CRC accumulation when the program sends the first data character to the Z80-SIO. Although the Z80-SIO automatically transmits up to two sync characters (16-bit sync), it is wise to send a few more sync characters ahead of the message (before enabling Transmit CRC) to ensure synchronization at the receiving end.

The transmit CRC Enable bit can be changed on the fly any time in the message to include or exclude a particular data character from CRC accumulation. The Transmit CRC Enable bit should be in the desired state when the data character is loaded from the transmit data buffer into the transmit shift register. To ensure this bit is in the proper state, the Transmit CRC Enable bit must be issued before sending the data character to the Z80-SIO.

Transmit Transparent Mode. Transparent mode (Bisync protocol) operation is made possible by the ability to change Transmit CRC Enable on the fly and by the additional capability of inserting 16-bit sync characters. Exclusion of DLE characters from CRC calculation can be achieved by disabling CRC calculation immediately preceding the DLE character transfer to the Z80-SIO.

In the case of a Transmit Underrun condition in the Transparent mode, a pair of DLE-SYN characters are sent. The Z80-SIO can be programmed to send the DLE-SYN sequence by loading a DLE character into WR6 and a sync character into WR7.

Transmit Termination. The Z80-SIO is equipped with a special termination feature that maintains data integrity and validity. If the transmitter is disabled while a data or sync character is being sent, that character is sent as usual, but is followed by a marking line rather than CRC or sync characters. When the transmitter is disabled, a character in the buffer remains in the buffer. If the transmitter is disabled while CRC is being sent, the 16-bit transmission is completed, but sync is sent instead of CRC.

A programmed break is effective as soon as it is written into the control register; characters in the transmit buffer and shift register are lost.

In all modes, characters are sent with the least significant bits first. This requires right-hand justification of transmitted data if the word length is less than eight bits. If the word length, is five bits or less, the special technique described in the Write Register 5 discussion (Z80-SIO Programming section) must be used for the data format. The states of any unused bits in a data character are irrelevant, except when in the Five Bits Or Less mode.

If the External/Status Interrupt Enable bit is set, transmitter conditions such as "starting to send CRC characters," "starting to send sync characters," and CTS changing state cause interrupts that have a unique vector if Status Affects Vector is set. This interrupt mode may be used during block transfers.

All interrupts may be disabled for operation in a Polled mode, or to avoid interrupts at inappropriate times during the execution of a program.

5.3 SYNCHRONOUS RECEIVE

5.3.1 Initialization

The system program initiates the Synchronous Receive operation with the following parameters: odd or even parity, 8-bit or 16-bit sync characters, x 1 clock mode, CRC polynomial, receive character length, etc. Sync characters must be loaded into registers WR6 and WR7. The receivers can be enabled only after all receive parameters are set. WR4 parameters must be issued before WR1, WR3, WR5, WR6, and WR7 parameters or commands.

After this is done, the receiver is in the Hunt phase. It remains in this phase until character synchronization is achieved. Note that, under program control, all the leading sync characters of the message can be inhibited from loading the receive buffers by setting the Sync Character Load Inhibit bit in WR3.

5.3.2 Data Transfer and Status Monitoring

After character synchronization is achieved, the assembled characters are transferred to the receive data FIFO. The following four interrupt modes are available to transfer the data and its associated status to the CPU.

No Interrupts Enabled. This mode is used for a purely polled operation or for off-line conditions.

Interrupt On First Character Only. This mode is normally used to start a polling loop or a Block Transfer instruction using /WAIT//READY to synchronize the CPU or the DMA device to the incoming data rate. In this mode, the Z80-SIO interrupts on the first character and thereafter interrupts only if Special Receive conditions are detected. The mode is reinitialized with the Enable Interrupt On Next Receive Character command to allow the next character received to generate an interrupt. Parity errors do not cause interrupts in this mode, but End-of-Frame (SDLC mode) and Receive Overrun do.

If External/Status interrupts are enabled, they may interrupt any time DCD changes state.

Interrupt On Every Character. Whenever a character enters the receive buffer, an interrupt is generated. Error and Special Receive conditions generate a special vector if Status Affects Vector is selected. Optionally, a Parity Error may be directed not to generate the special interrupt vector.

Special Receive Condition Interrupts. The Special Receive Condition interrupt can occur only if either the Receive Interrupt On First Character Only or Interrupt On Every Receive Character modes is also set. The Special Receive Condition interrupt is caused by the Receive Overrun error condition. Since the Receive Overrun and Parity error status bits are latched, the error status-when read-reflects an error in the current word in the receive buffer in addition to any Parity or Overrun errors received since the last Error Reset command. These status bits can only be reset by the Error reset command.

CRC Error Checking and Termination. A CRC error check on the receive message can be performed on a per character basis under program control. The Receive CRC Enable bit (WR3, D3) must be set/reset by the program before the next character is transferred from the receive shift register into the receive buffer register. This ensures proper inclusion or exclusion of data characters in the CRC check.

To allow the CPU ample time to enable or disable the CRC check on a particular character, the Z80-SIO calculates CRC eight bit times after the character has been transferred to the receive buffer. If CRC is enabled before the next character is transferred, CRC is calculated on the transferred character. If CRC is disabled before the time of the next transfer, calculation proceeds on the word in progress, but the word just transferred to the buffer is not included. When these requirements are satisfied, the 3-byte receive data buffer is, in effect, unusable in Bisync operation. CRC may be enabled and disabled as many times as necessary for a given calculation.

In the Monosync, Bisync and External Sync modes, the CRC/Framing Error bit (RR1, D6) contains the comparison result of the CRC checker 16-bit times (eight bits delay and eight shifts for CRC) after the character has been transferred from the receive shift register to the buffer. The result should be zero, indicating an error-free transmission. (Note that the result is valid only at the end of CRC calculation. If the result is examined before this time, it usually indicates an error.) The comparison is made with each transfer and is valid only as long as the character remains in the receive FIFO.

Following is an example of the CRC checking operation when four characters (A, B, C, and D) are received in that order.

Character A loaded into buffer
Character B loaded into buffer

If CRC is disabled before C is in the buffer, CRC is not calculated on B.

Character C loaded into buffer

After C is loaded, the CRC/Framing Error bit shows the result of the comparison through character A.

Character D loaded into buffer

After D is in the buffer, the CRC Error bit shows the result of the comparison through character B whether or not B was included in the CRC calculations.

Due to the serial nature of CRC calculation, the Receive Clock (/RxC) must cycle 16 times (8-bit delay plus 8-bit CRC shift) after the second CRC character has been loaded into the receive buffer, or 20 times (the previous 16 plus 3-bit buffer delay and 1-bit input delay) after the last bit is at the RxD input, before CRC calculation is complete. A faster external clock can be gated into the Receive Clock input to supply the required 16 cycles. The Transmit and Receive Data Path diagram (Figure 3-1) illustrates the various points of delay in the CRC path.

The typical program steps that implement a half-duplex Bisync Receive mode are illustrated in Table 5-3. The complete set of command and status bit definitions are explained under "Z80-SIO Programming."

Table 5-3. Bisync Receive Mode

FUNCTION		TYPICAL PROGRAM STEPS	COMMENTS
	REGISTER:	INFORMATION LOADED	
INITIALIZE	WR0	CHANNEL RESET, RESET RECEIVE CRC CHECKER	Reset SIO; initialize Receive CRC checker.
	WR0	POINTER 2	
	WR2	INTERRUPT VECTOR	Channel B only
	WR0	POINTER 4	
	WR4	PARITY INFORMATION, SYNC MODES INFORMATION, CLOCK MODE	Issue receive parameters.
	WR0	POINTER 5, RESET EXTERNAL STATUS INTERRUPT	
	WR5	BISYNC CRC-16, DATA TERMINAL READY	
	WR0	POINTER 3	
	WR3	SYNC CHARACTER LOAD INHIBIT, RECEIVE CRC ENABLE; ENTER HUNT MODE, AUTO ENABLES, RECEIVE CHARACTER LENGTH	Sync character load inhibit strips all the leading sync characters at the beginning of the message. Auto Enables enables the receiver to accept data only after the /DCD input is active.
	WR0	POINTER 6	
	WR6	SYNC CHARACTER 1	
	WR0	POINTER 7	
	WR7	SYNC CHARACTER 2	
	WR0	POINTER 1, RESET EXTERNAL/STATUS INTERRUPT	
	WR1	STATUS AFFECTS VECTOR, EXTERNAL INTERRUPT ENABLE, RECEIVE INTERRUPT ON FIRST CHARACTER ONLY	In this interrupt mode, only the first non-sync data character is transferred to the CPU. All subsequent data is transferred on a DMA basis; however Special Receive Condition interrupts will interrupt the CPU. Status Affects Vector used in Channel B only.

E

Table 5-3. Bisync Receive Mode (Continued)

FUNCTION		TYPICAL PROGRAM STEPS	COMMENTS
INITIALIZE (Continued)	WR0	POINTER 3, ENABLE INTERRUPT ON NEXT RECEIVE CHARACTER	Resetting this interrupt mode provides simple program loopback entry for the next transaction.
	WR3	RECEIVE ENABLE, SYNC CHARACTER LOAD INHIBIT, ENTER HUNT MODE. AUTO ENABLE, RECEIVE WORD LENGTH	WR3 is reissued to enable receiver, Receive CRC Enable must be set after receiving SOH or STX character. Receive mode is fully initialized and the
IDLE MODE		EXECUTE HALT INSTRUCTION OR SOME OTHER PROGRAM	system is waiting for interrupt on first character.
DATA TRANSFER AND STATUS MONITORING		WHEN INTERRUPT ON FIRST CHARACTER OCCURS, THE CPU DOES THE FOLLOWING: ■ TRANSFERS DATA BYTE TO CPU ■ DETECTS AND SETS APPROPRIATE FLAGS FOR CONTROL CHARACTERS (IN CPU) ■ INCLUDES/EXCLUDES DATA BYTE IN CRC CHECKER ■ UPDATES POINTERS AND OTHER PARAMETERS ■ ENABLES WAIT/READY FOR DMA OPERATION ■ ENABLES DMA CONTROLLER ■ RETURNS FROM INTERRUPT	During the Hunt mode, the SIO detects two contiguous characters to establish synchronization. The CPU establishes the DMA mode and all subsequent data characters are transferred by the DMA controller. The controller is also programmed to capture special characters (by examining only the bits that specify ASCII or EBCDIC control characters) and interrupt the CPU upon detection, In response, the CPU examines the status or control characters and takes appropriate action (e.g. CRC Enable Update).
		WHEN WAIT/READY BECOMES ACTIVE, THE DMA CONTROLLER DOES THE FOLLOWING: ■ TRANSFERS DATA BYTE TO MEMORY ■ INTERRUPTS CPU IF A SPECIAL CHARACTER IS CAPTURED BY THE DMA CONTROLLER ■ INTERRUPTS THE CPU IF THE LAST CHARACTER OF THE MESSAGE IS DETECTED	
		FOR MESSAGE TERMINATION, THE CPU DOES THE FOLLOWING: ■ TRANSFERS RR1 TO THE CPU ■ SETS ACK/NAK REPLY FLAG BASED ON CRC RESULT ■ UPDATES POINTERS AND PARAMETERS ■ RETURNS FROM INTERRUPT	The SIO interrupts the CPU for error condition, and the error routine aborts the present message, clears the error condition, and repeats the operation.
TERMINATION		REDEFINE INTERRUPT MODES AND SYNC MODES UPDATE MODEM CONTROLS DISABLES RECEIVE MODE	

CHAPTER 6
SDLC (HDLC) OPERATION

6.0 INTRODUCTION

The Z80-SIO is capable of handling both High-level Synchronous Data Link Control (HDLC) and IBM Synchronous Data Link Control (SDLC) protocols. In the following text, only SDLC is referred to because of the high degree of similarity between SDLC and HDLC.

The SDLC mode is considerably different than Synchronous Bisync protocol because it is bit oriented rather than character oriented and, therefore, can naturally handle transparent operation. Bit orientation makes SDLC a flexible protocol in terms of message length and bit patterns. The Z80-SIO has several built-in features to handle variable message length. Detailed information concerning SDLC protocol can be found in literature published on this subject, such as IBM document GA27-3093.

The SDLC message, called the frame (Figure 6-1), is opened and closed by flags that are similar to the sync characters in Bisync protocol. The Z80-SIO handles the transmission and recognition of the flag characters that mark the beginning and end of the frame. Note that the Z80-SIO can receive shared-zero flags, but cannot transmit them. The 8-bit address field of an SDLC frame contains the secondary station address. The Z80-SIO has an Address Search mode that recognizes the secondary station address so it can accept or reject the frame.

Since the control field of the SDLC frame is transparent to the Z80-SIO, it is simply transferred to the CPU. The Z80-SIO handles the Frame Check sequence in a manner that simplifies the program by incorporating features such as initializing the CRC generator to all 1's, resetting the CRC checker when the opening flag is detected iii the Receive mode, and sending the Frame Cheek/Flag sequence in the Transmit mode. Controller hardware is simplified by automatic zero insertion and deletion logic contained in the Z80-SIO.

Table 6-1 shows the contents of WR3, WR4, and WR5 during SDLC Receive and Transmit modes. WR0 points to other registers and issues various commands, WR1 defines the interrupt modes. WR2 stores the interrupt vector. WR7 stores the flag character and WR6 the secondary address.

E

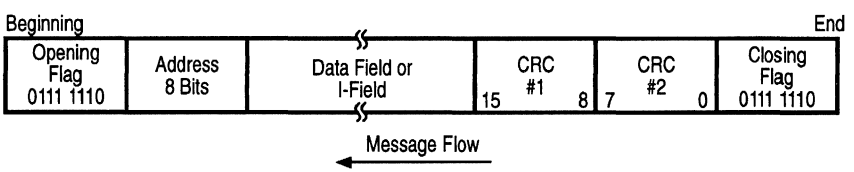


Figure 6-1. Transmit/Receive SDLC/HDLC Message Format

6.1 SDLC TRANSMIT

6.1.1 Initialization

Like Synchronous operation, the SDLC Transmit mode must be initialized with the following parameters: SDLC mode, SDLC polynomial, Request To Send, Data Terminal Ready, transmit character length, transmit interrupt modes (or Wait/Ready function), Transmit Enable, Auto Enables, and External/Status interrupt.

Selecting the SDLC mode and the SDLC polynomial enables the Z80-SIO to initialize the CRC Generator to all 1's. This is accomplished by issuing the Reset Transmit CRC Generator command (WR0). Refer to the Synchronous Operation section for more details on the interrupt modes.

After reset, or when the transmitter is not enabled, the Transmit Data output is held marking. Break may be programmed to generate a spacing line. With the transmitter fully initialized and enabled, continuous flags are transmitted on the Transmit Data output.

An abort sequence may be sent by issuing the Send Abort command (WR0, CMD1). This causes at least eight, but less than 14, 1's to be sent before the line reverts to continuous flags. It is possible that the Abort sequence (eight 1's) could follow up to five continuous 1 bits (allowed by the zero insertion logic) and thus cause up to thirteen 1's to be sent. Any data being transmitted and any data in the transmit buffer is lost when an abort is issued.

When required, an extra 0 is automatically inserted when there are five contiguous 1's in the data stream. This does not apply to flags or aborts.

6.1.2 Data Transfer and Status Monitoring

There are several combinations of interrupts and the Wait/Ready function in the SDLC mode.

Data Transfer Using Interrupts. If the Transmit Interrupt Enable bit is set, an interrupt is generated each time the buffer becomes empty. The interrupt may be satisfied either by writing another character into the transmitter or by resetting the Transmit Interrupt Pending latch with a Reset Transmitter Pending command (WR0, CMD5). If the interrupt is satisfied with this command and nothing more is written into the transmitter, there are no further transmitter interrupts. The result is a Transmit Underrun condition. When another character is written and sent out, the transmitter can again become empty and interrupt the CPU. Following the flags in an SDLC operation, the 8-bit address field, control field and information field may be sent to the Z80-SIO using the Transmit Interrupt mode. The Z80-SIO transmits the Frame Check sequence using the Transmit Underrun feature.

When the transmitter is first enabled, it is already empty and obviously cannot then become empty. Therefore, no Transmit Buffer Empty interrupts can occur until after the first data character is written.

When the transmitter is first enabled, it is already empty and cannot then become empty, Therefore, no Transmit Buffer Empty interrupts can occur until after the first data character is written.

Table 6-1. Contents of Write Registers 3, 4, and 5 in SDLC Modes

	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
WR3	00 = Rx 5 BITS CHAR 10 = Rx 6 BITS CHAR 01 = Rx 7 BITS CHAR 11 = Rx 8 BITS CHAR		AUTO ENABLES	ENTER HUNT MODE (IF INCOMING DATA NOT NEEDED)	Rx CRC ENABLE	ADDRESS SEARCH MODE	0	Rx ENABLE
WR4	0	0	1	0	0	0	0	0
			SELECTS SDLC MODE					
WR5	DTR	00 = Tx 5 BITS (OR LESS) CHAR 10 = Tx 6 BITS, CHAR 01 = Tx 7 BITS CHAR 11 = Tx 8 BITS CHAR		0	TX ENABLE	0 SELECTS SDLC CRC	RTS	TX CRC ENABLE

Table 6-2. SDLC Transmit Mode

FUNCTION		TYPICAL PROGRAM STEPS	COMMENTS
	REGISTER:	INFORMATION LOADED:	
INITIALIZE	WR0	CHANNEL RESET	Reset SIO.
	WR0	POINTER 2	
	WR2	INTERRUPT VECTOR	Channel B only
	WR0	POINTER 3	
	WR3	AUTO ENABLES	Transmitter sends data only after /CTS is detected.
	WR0	POINTER 4, RESET EXTERNAL/STATUS INTERRUPTS	
	WR4	PARITY INFORMATION, SDLC MODE, x1 CLOCK MODE	
	WR0	POINTER 1. RESET EXTERNAL/STATUS INTERRUPTS	
	WR1	EXTERNAL INTERRUPT ENABLE, STATUS AFFECTS VECTOR, TRANSMIT INTERRUPT ENABLE OR WAIT/READY MODE ENABLE	The External Interrupt mode monitors the status of the /CTS and DCD inputs, as well as the status of Tx Underrun/EOM latch. Transmit Interrupt interrupts when the Transmit butter becomes empty; the Wait/Ready mode can be used to transfer data on a DMA or Block Transfer basis. The first interrupt occurs when /CTS becomes active, at which point flags are transmitted by the Z80-SIO. The first data byte (address field) can be loaded in the Z80-SIO after this interrupt. Flags cannot be sent to the Z80-SIO as data. Status Affects Vector used in Channel B only,
	WR0	POINTER 5	
	WR5	TRANSMIT CRC ENABLE, REQUEST TO SEND, SDLC-CRC. TRANSMIT ENABLE, TRANSMIT WORD LENGTH, DATA TERMINAL READY	SDLC-CRC mode must be defined before initializing transmit CRC generator.
	WR0	RESET TRANSMIT CRC GENERATOR	Initialize CRC generator to all 1's.
IDLE MODE		EXECUTE HALT INSTRUCTION OR SOME OTHER PROGRAM	Waiting for Interrupt or Wait/Ready output to transfer data.
DATA TRANSFER AND STATUS MONITORING		WHEN INTERRUPT (WAIT READY) OCCURS, THE CPU DOES THE FOLLOWING: ■ CHANGES TRANSMIT WORD LENGTH (IF NECESSARY) ■ TRANSFERS DATA BYTE FROM CPU (MEMORY) TO SIO RESETS Tx UNDERRUN/EOM LATCH (WR0) IF LAST CHARACTER OF THE I-FIELD IS SENT, THE SIO DOES THE FOLLOWING: ■ SENDS CRC ■ SENDS CLOSING FLAG ■ INTERRUPTS CPU WITH BUFFER EMPTY STATUS CPU DOES THE FOLLOWING: ■ ISSUES RESET Tx INTERRUPT PENDING COMMAND TO THE Z80-S10 ■ UPDATES NS COUNT ■ REPEATS THE PROCESS FOR NEXT MESSAGE, ETC. IF THE VECTOR INDICATES AN ERROR. THE CPU DOES THE FOLLOWING: ■ SENDS ABORT ■ EXECUTES ERROR ROUTINE ■ UPDATES PARAMETERS, MODES. ETC. ■ RETURNS FROM INTERRUPT	Flags are transmitted by the SIO as Soon as Transmit Enable is set and /CTS becomes active, The /CTS status change is the first interrupt that occurs and is followed by transmit buffer empty for subsequent transfers. Word length can be changed "on the fly" for variable I-field length. The data byte can contain address, control, or I-Field information (never a flag). It is a good practice to reset Tx Underrun/EOM latch in the beginning of the message to avoid a false end-of-frame detection at the receiving end. This ensures that, when underrun occurs, CRC is transmitted and underrun interrupt (Tx Underrun/EOM latch active) occurs. Note that "Send Abort" can be issued to the SIO in response to any interrupting continuing to abort the transmission,
TERMINATION		REDEFINE INTERRUPT MODES UPDATE MODEM CONTROL OUTPUTS DISABLE TRANSMIT MODE	Terminate gracefully.

E

Data Transfer Using Wait/Ready. If the Wait/Ready function has been Selected, /WAIT indicates to the CPU that the Z80-SIO is not ready to accept the data and the CPU must extend the I/O cycle. To a DMA controller, /READY indicates that the transmitter buffer is empty and that the Z80-SIO is ready to accept the next character. If the data character is not loaded into the Z80-SIO by the time the transmit shift register is empty, the Z80-SIO enters the Transmit Underrun condition. Address, control, and information fields may be transferred to the Z80-SIO with this mode using the Wait/Ready function. The Z80-SIO transmits the Frame Check sequence using the Transmit Underrun feature.

SDLC Transmit Underrun/End of Message. SDLC-like protocols do not have provisions for fill characters within a message. The Z80-SIO therefore automatically terminates an SDLC frame when the transmit data buffer and output shift register have no more bits to send. It does this by first sending the two bytes of CRC and following these with one or more flags. This technique allows very high-speed transmissions under DMA or CPU block I/O control without requiring the CPU to respond quickly to the end of message situation.

The action that the Z80-SIO takes in the underrun situation depends on the state of the Transmit Underrun/EOM command. Following a reset, the Transmit Underrun/EOM status bit is in the set state and prevents the insertion of CRC characters during the time there is no data to send. Consequently, flag characters are sent. The Z80-SIO begins to send the frame as data is written into the transmit buffer. Between the time the first data byte is written and the end of the message, the Reset Transmit Underrun/EOM Command must be issued, Thus the Transmit Underrun/EOM status bit is in the reset state at the end of the message (when underrun occurs), which automatically sends the CRC characters, The sending of CRC again sets the Transmit/Underrun/ EOM status bit.

Although there is no restriction as to when the Transmit Underrun/EOM bit can be reset within a message, it is usually reset after the first data character (secondary address) is sent to the Z80-SIO. Resetting this bit allows CRC and flags to be sent when there is no data to send which gives additional time to the CPU for recognizing the fault and responding with an abort command. By reselling it early in the message, the entire message has the maximum amount of CPU response time in an unintentional transmit underrun situation.

When the External/Status interrupt is set and while CRC is being sent, the Transmit Underrun/EOM bit is set and the Transmit Buffer Empty bit is reset to indicate that the transmit register is full of CRC data. When CRC has been completely sent, the Transmit Buffer Empty status bit is set and an interrupt is generated to indicate to the CPU that another message can begin. This interrupt occurs because CRC has been sent and the flag has been loaded. If no more messages are to be sent, the program can terminate transmission by resetting /RTS, and disabling the transmitter.

In the SDLC mode, it is good practice to reset the Transmit Underrun/EOM status bit immediately after the first character is sent to the Z80-SIO. When the Transmit Underrun is detected, this ensures that the transmission time is filled by CRC characters, giving the CPU enough time to issue the Send Abort command. This also stops the flags from going on the line prematurely and eliminates the possibility of the receiver accepting the frame as valid data. The situation can happen because it is possible that, at the receiving end, the data pattern immediately preceding the automatic flag insertion could match the CRC checker, giving a false CRC check result. The External/Status interrupt is generated whenever the Transmit Underrun/EOM bit is set because of the Transmit Underrun condition.

The transmit underrun logic provides additional protection against premature flag insertion if the proper response is given to the Z80-SIO by the CPU interrupt service routine. The following example is given to clarify this point:

- The Z80-SIO raises an interrupt with the Transmit Buffer Empty status bit set.
- The CPU does not respond in time and causes a Transmit Underrun condition.
- The Z80-SIO starts sending CRC characters (two bytes).
- The CPU eventually satisfies the Transmit Buffer Empty interrupt with a data character that follows the CRC character being transmitted.
- The Z80-SIO sets the External/Status interrupt with the Transmit Underrun/EOM status bit set
- The CPU recognizes the Transmit Underrun/EOM status and determines from its internal program status that the interrupt is not for "end of message".
- The CPU immediately issues a Send Abort Command (WR0) to the Z80-SIO.
- The Z80-SIO sends the Abort sequence by destroying whatever data (CRC, data, or flag) is being sent.

This sequence illustrates that the CPU has a protection of 22 minimum and 30 maximum transmit clock cycles.

SDLC CRC Generation. The CRC generator must be reset to all 1's at the beginning of each frame before CRC accumulation can begin. Actual accumulation begins when the program sends the address field (eight bits) to the Z80-SIO. Although the Z80-SIO automatically transmits one flag character following the Transmit Enable, it may be wise to send a few more flag characters ahead of the message to ensure character synchronization at the receiving end. This can be done by externally timing out after enabling the transmitter, and before loading the first character.

The Transmit CRC Enable (WR5, D0) should be enabled prior to sending the address field. In the SDLC mode all the characters between the opening and closing flags are included in CRC accumulation, and the CRC generated in the Z80-SIO transmitter is inverted before it is sent on the line.

Transmit Termination. If the transmitter is disabled while a character is being sent, that Character (data or flag) is sent in the normal fashion, but is followed by a marking line rather than CRC or flag characters.

A character in the buffer when the transmitter is disabled remains in the buffer; however, a programmed Abort sequence is effective as soon as it is written into the control register. Characters being transmitted, if any, are lost. In the case of CRC, the 16-bit transmission is completed if the transmitter is disabled; however, flags are sent in place of CRC.

In all modes, characters are sent with the least-significant bits first. This requires right-hand justification of data to be transmitted if the word length is less than eight bits. If the word length is five bits or less, the special technique described in the Write Register 5 section ("Z80-SIO Programming" chapter; "Write Registers" section) must be used.

Since the number of bits/character can be changed on the fly, the data field can be filled with any number of bits. When used in conjunction with the Receiver Residue codes, the Z80-SIO can receive a message that has a variable I-field and retransmit it exactly as received with no previous information about the character structure of the I-field (if any). A change in the number of bits does not affect the character in the process of being shifted out. Characters are sent with the number of bits programmed at the time that the character is loaded from the transmit buffer to the transmitter.

If the External/Status Interrupt Enable is set, transmitter conditions such as "starting to send CRC characters," "starting to send flag characters," and /CTS changing state cause interrupts that have a unique vector if Status Affects Vector is set. All interrupts can be disabled for operation in a polled mode.

Table 6-2 shows the typical program steps that implement the half-duplex SDLC Transmit mode.

6.2 SDLC RECEIVE

6.2.1 Initialization

The SDLC Receive mode is initialized by the system with the following parameters: SDLC mode, x1 clock mode, SDLC polynomial, receive word length, etc. The flag characters must also be loaded in WR7 and the secondary address field loaded in WR6. The receiver is enabled only after all the receive parameters have been set. After all this has been done, the receiver is in the Hunt phase and remains in this phase until the first flag is received. While in the SDLC mode, the receiver never re-enters the Hunt phase, unless specifically instructed to do so by the program. The WR4 parameters must be issued prior to the WR1, WR3, WR5, WR6, and WR7 parameters.

Under program control, the receiver can enter the Address Search mode. If the Address Search bit (WR1, D2) is set, a character following the flag (first non-flag character) is compared against the programmed address in WR6 and the hardwired global address (1111 1111). If the SDLC frame address field matches either address, data transfer begins.

Since the Z80-SIO is capable of matching only one address character, extended address field recognition must be done by the CPU. In this case, the Z80-SIO simply transfers the additional address bytes to the CPU as if they were data characters. If the CPU determines that the frame does not have the correct address field, it can set the Hunt bit, and the Z80-SIO suspends reception and searches for a new message headed by a flag. Since the control field of the frame is transparent to the Z80-SIO, it is transferred to the CPU as a data character. Extra zeros inserted in the data stream are automatically deleted; flags are not transferred to the CPU.

6.2.2 Data Transfer and Status Monitoring

After receipt of a valid flag, the assembled characters are transferred to the receive data FIFO. The following four interrupt modes are available to transfer this data and its associated status.

No Interrupts Enabled. This mode is used for purely polled operations or for off-line conditions.

Interrupt On First Character Only. This mode is normally used to start a software polling loop or a Block Transfer instruction using /WAIT//READY to synchronize the CPU or DMA device to the incoming data rate. In this mode, the Z80-SIO interrupts on the first character and thereafter only interrupts if Special Receive conditions are detected. The mode is reinitialized with the Enable Interrupt On Next Receive Character Command.

The first character received after this command is issued causes an interrupt. If External/Status interrupts are enabled, they may interrupt any time the DCD input changes state. Special Receive conditions such as End-of-Frame and Receiver Overrun also cause interrupts. The End-of-Frame interrupt can be used to exit the Block Transfer mode.

Interrupt On Every Character. An interrupt is generated whenever the receive FIFO contains a character. Error and Special Receive conditions generate a special vector if Status Affects Vector is selected.

Special Receive Condition Interrupts. The Special Receive Condition interrupt is not, as such, a separate interrupt mode. Before the Special Receive condition can cause an interrupt, either Interrupt On First Receive Character Only or Interrupt On Every Character must be selected. The Special Receive Condition interrupt is caused by a Receive Overrun or End-of-Frame detection. Since the Receive Overrun status bit is latched, the error status read reflects an error in the current word in the receive buffer in addition to any errors received since the last Error Reset command. The Receive Overrun status bit can only be reset by the Error Reset command. The End-of-Frame status bit indicates that a valid ending flag has been received and that the CRC Error and Residue codes are also valid.

Character length may be changed on the fly. If the address and control bytes are processed as 8-bit characters, the receiver may be switched to a shorter character length during the time that the first information character is being assembled. This change must be made fast enough so it is effective before the number of bits specified for the character length have been assembled. For example, if the change is to be from the 8-bit control field to a 7-bit information field, the change must be made before the first seven bits of the I-Field are assembled.

SDLC Receive CRC Checking. Control of the receive CRC checker is automatic. It is reset by the leading flag and CRC is calculated up to the final flag. The byte that has the End-of-Frame bit set is the byte that contains the result of the CRC check. If the CRC/Framing Error bit is not set, the CRC indicates a valid message. A special check sequence is used for the SDLC cheek because the transmitted CRC check is inverted. The final check must be 0001 1101 0000 1111. The 2-byte CRC check characters must be read by the CPU and discarded because the Z80-SIO, while using them for CRC checking, treats them as ordinary data.

SDLC Receive Termination. If enabled, a special vector is generated when the closing flag is received. This signals that the byte with the End-of-Frame bit set has been received. In addition to the results of the CRC check, RR1 has three bits of Residue code valid at this time. For those cases in which the number of bits in the I-Field is not an integral multiple of the character length used, these bits indicate the boundary between the CRC check bits and the I-Field bits. For a detailed description of the meaning of these bits, see the description of the residue codes in RR1 under "Z80-SIO Programming."

Any frame can be prematurely aborted by an Abort sequence. Aborts are detected if seven or more 1's occur and cause an External/Status interrupt (if enabled) with the Break/Abort bit in RR0 set. After the Reset External/Status interrupts command has been issued a second interrupt occurs when the continuous 1's condition has been cleared. This can be used to distinguish between the Abort and Idle line conditions.

Unlike the synchronous mode, CRC calculation in SDLC does not have an 8-bit delay since all the characters are included in CRC calculation. When the second CRC character is loaded into the receive buffer, CRC calculation is complete.

Table 6-3 shows the typical steps required to implement a half-duplex SDLC receive mode. The complete set of command and status bit definitions is found in the next section.

Table 6-3. SDLC Receive Mode

FUNCTION		TYPICAL PROGRAM STEPS	COMMENTS
	REGISTER:	INFORMATION LOADED.	
INITIALIZE	WR0	CHANNEL 2	Reset SIO
	WR0	POINTER 2	
	WR2	INTERRUPT VECTOR	Channel B only
	WR0	POINTER 4	
	WR4	PARITY INFORMATION, SYNC MODE, SDLC MODE, x1 CLOCK MODE	
	WR0	POINTER 5, RESET EXTERNAL/STATUS INTERRUPTS	
	WR5	SDLC-CRC, DATA TERMINAL READY	
	WR0	POINTER 3	
	WR3	RECEIVE CRC ENABLE, ENTER HUNT MODE, AUTO ENABLES, RECEIVE CHARACTER LENGTH, ADDRESS SEARCH MODE	'Auto Enables' enables the receiver to accept data only after Mb becomes active. Address Search Mode enables SIO to match the message address with the programmed address or the global address.
	WR0	POINTER 6	
	WR6	SECONDARY ADDRESS FIELD	This address is matched against the message address in an SDLC poll operation.
	WR0	POINTER 7	
	WR7	SDLC FLAG 0111 1110	This flag detects the start and end-of-frame in an SDLC operation.
	WR0	POINTER 1, RESET EXTERNAL/STATUS INTERRUPTS	In this interrupt mode, only the Address Field (1 character only) is transferred to
	WR1	STATUS AFFECTS VECTOR, EXTERNAL INTERRUPT ENABLE, RECEIVE INTERRUPT ON FIRST CHARACTER ONLY.	the CPU. All subsequent fields (Control, information, etc.) are transferred on a DMA basis, Status Affects Vector in Channel B only.
	WR0	POINTER 3, ENABLE INTERRUPT ON NEXT RECEIVE CHARACTER	Used to provide simple loop-back entry point for next transaction.
	WR3	RECEIVE ENABLE, RECEIVE CRC ENABLE, ENTER HUNT MODE, AUTO ENABLES, RECEIVER CHARACTER LENGTH, ADDRESS SEARCH MODE	WR3 reissued to enable receiver.
IDLE MODE		EXECUTE HALT INSTRUCTION OR SOME OTHER PROGRAM	SDLC Receive Mode is fully initialized and SIO is waiting for the opening flag followed by a matching address field to interrupt the CPU.
DATA TRANSFER AND STATUS MONITORING		WHEN INTERRUPT ON FIRST CHARACTER OCCURS, THE CPU DOES THE FOLLOWING: ■ TRANSFERS DATA BYTE (ADDRESS BYTE) TO CPU ■ DETECTS AND SETS APPROPRIATE FLAG FOR EXTENDED ADDRESS FIELD ■ UPDATES POINTERS AND PARAMETERS ■ ENABLES DMA CONTROLLER ■ ENABLES WAIT/READY FUNCTION IN SIO ■ RETURNS FROM INTERRUPT	During the Hunt phase, the SIO interrupts when the programmed address matches the message address. The CPU establishes the DMA mode and all subsequent data characters are transferred by the DMA controller to memory.
		WHEN THE READY OUTPUT BECOMES ACTIVE, THE DMA CONTROLLER DOES THE FOLLOWING: ■ TRANSFERS THE DATA BYTE TO MEMORY ■ UPDATES THE POINTERS	During the DMA operation, the SIO monitors the DCD input and the Abort sequence in the data stream to interrupt the CPU with External Status error. The Special Receive condition interrupt is caused by Receive Overrun error.

E

Table 6-3. SDLC Receive Mode (Continued)

FUNCTION	TYPICAL PROGRAM STEPS	COMMENTS
DATA TRANSFER AND STATUS MONTIORING (Continued)	WHEN END OF FRAME INTERRUFT OCCURS, THE CPU DOES THE FOLLOWING: ■ EXITS DMA MODE (DISABLES WAIT/READY) ■ TRANSFERS RR1 TO THE CPU ■ CHECKS THE CRC ERROR BIT STATUS AND RESIDUE CODES ■ UPDATES NR COUNT ■ ISSUES ERROR RESET COMMAND TO SIO	Detection of End of Frame (Flag) causes interrupt and deactivates the Wait/Ready function. Residue codes indicate the bit structure of the last two bytes of the message, which were transferred to memory under DMA. 'Error Reset' is issued to clear the special condition.
	WHEN ABORT SEQUENCE DETECTED INTERRUPT OCCURS, THE CPU DOES THE FOLLOWING: ■ TRANSFERS RR0 TO THE CPU ■ EXITS DMA MODE ■ ISSUES THE RESET EXTERNAL STATUS INTERRUPT COMMAND TO THE SIO	Abort sequence is detected when seven or more 1's are found in the data stream.
	■ ENTERS THE IDLE MODE	CPU is waiting for Abort Sequence to terminate. Termination clears the Break/ Abort status bit and causes interrupt.
	WHEN THE SECOND ABORT SEQUENCE INTERRUPT OCCURS, THE CPU DOES THE FOLLOWING: ■ ISSUES THE RESET EXTERNAL STATUS INTERRUPT COMMAND TO THE SIO	At this point, the program proceeds to terminate this message.
TERMINATION	REDEFINE INTERRUPT MODES, SYNC MODE AND SDLC MODES DISABLE RECEIVE MODE	

CHAPTER 7
PROGRAMMING

7.0 INTRODUCTION

To program the Z80-SIO, the system program first issues a series of commands that initialize the basic mode of operation and then other commands that qualify conditions within the selected mode. For example, the Asynchronous mode, character length, clock rate, number of stop bits, even or odd parity are first set, then the interrupt mode and, finally, receiver or transmitter enable. The WR4 parameters must be issued before any other parameters are issued in the initialization routine.

Both channels contain command registers that must be programmed via the system program prior to operation.

The Channel Select input (B//A) and the Control/Data input (C//D) are the command structure addressing controls, and are normally controlled by the CPU address bus. Figures 9-1 through 9-4 illustrate the timing relationships for programming the write registers, and transferring data and status.

Table 7-1. Channel Select Functions

C//D	B//A	Function
0	0	Channel A Data
0	1	Channel B Data
1	0	Channel A Commands/Status
1	1	Channel B Commands/Status

7.1 WRITE REGISTERS

The Z80-SIO contains eight registers (WR7-WR0) in each channel that are programmed separately by the system program to configure the functional personality of the channels. With the exception of WR0, programming the write registers requires two bytes. The first byte contains three bits (D2-D0) that point to the selected register; the second byte is the actual control word that is written into the register to configure the Z80-SIO.

Note that the programmer has complete freedom, after pointing to the selected register, of either reading to test the read register or writing to initialize the write register. By designing software to initialize the Z80-SIO in a modular and structured fashion, the programmer can use powerful block I/O instructions.

WR0 is a special case in that all the basic commands (CMD2-CMD0) can be accessed with a single byte. Reset (internal or external) initializes the pointer bits D0-D2 to point to WR0.

The basic commands (CMD2-CMD0) and the CRC controls (CRC0, CRC1) are contained in the first byte of any write register access. This maintains maximum flexibility and system control. Each channel contains the following control registers. These registers are addressed as commands (not data).

7.1.1 Write Register 0

WR0 (Figure 7-1) is the command register; however, it is also used for CRC reset codes and to point to the other registers.

D7	D6	D5	D4	D3	D2	D1	D0
CRC Reset Code 1	CRC Reset Code 0	CMD 2	CMD 1	CMD 0	PTR 2	PTR 1	PTR 0

E

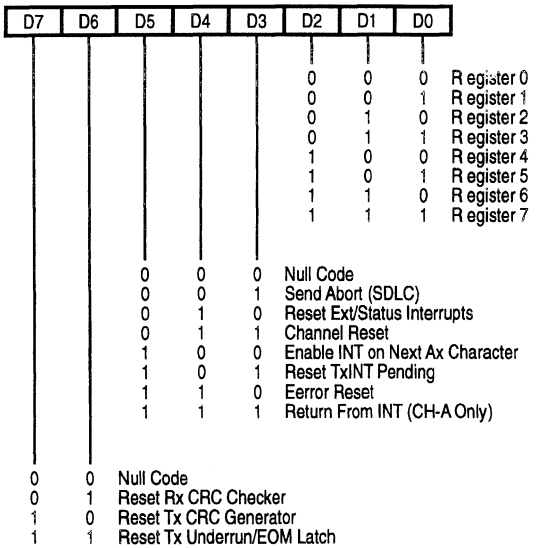


Figure 7-1. Write Register 0

Pointer Bits (D2-D0). Bits D2-D0 are pointer bits that determine which other write register the next byte is to be written into or which read register the next byte is to be read from. The first byte written into each channel after a reset (either by a Reset command or by the external reset input) goes into WR0. Following a read or write to any register (except WR0), the pointer will, point to WR0.

Command Bits (D5-D3). Three bits, D5-D3, are encoded to issue the seven basic Z80-SIO commands (Table 7-2).

Table 7-2. Z80-SIO Commands

Command	CMD2	CMD1	CMD0	
0	0	0	0	Null Command (no effect)
1	0	0	1	Send Abort (SDLC Mode)
2	0	1	0	Reset External/Status Interrupts
3	0	1	1	Channel Reset
4	1	0	0	Enable Interrupt on next Rx Character
5	1	0	1	Reset Transmitter Interrupt Pending
6	1	1	0	Error Reset (latches)
7	1	1	1	Return from Interrupt (Channel A)

Command 0 (Null). The Null command has no effect. Its normal use is to cause the Z80-SIO to do nothing while the pointers are set for the following byte.

Command 1 (Send Abort). This command is used only with the SDLC mode to generate a sequence of eight to thirteen 1's.

Command 2 (Reset External/Status Interrupts). After an External/Status interrupt (a change on a modem line or a break condition, for example), the status bits of RR0 are latched. This command re-enables them and allows interrupts to occur again. Latching the status bits captures short pulses until the CPU has time to read the change.

Command 3 (Channel Reset). This command performs the same function as an External Reset, but only on a single channel. Channel A Reset also resets the interrupt prioritization logic. All control registers for the channel must be rewritten after a Channel Reset command.

After a Channel Reset, four extra system clock cycles should be allowed for Z80-SIO reset time before any additional commands or controls are written into that channel. This can normally be the time used by the CPU to fetch the next opcode.

Command 4 (Enable Interrupt On Next Receive Character). If the Interrupt On First Receive Character mode is selected, this command reactivates that mode after each complete message is received to prepare the Z80-SIO for the next message.

Command 5 (Reset Transmitter Interrupt Pending). The transmitter interrupts when the transmit buffer becomes empty if the Transmit Interrupt Enable mode is selected. In those cases where there are no more characters to be sent (at the end of message, for example), issuing this command prevents further transmitter interrupts until after the next character has been loaded into the transmit buffer or until CRC has been completely sent.

Command 6 (Error Reset). This command resets the error latches. Parity and Overrun errors are latched in RR1 until they are reset with this command. With this scheme, parity errors occurring in block transfers can be examined at the end of the block.

Command 7 (Return From Interrupt). This command must be issued in Channel A and is interpreted by the Z80-SIO in exactly the same way it would interpret an RETI command on the data bus. It resets the interrupt under-service latch of the highest priority internal device under service and thus allows lower priority devices to interrupt through the daisy chain. This command allows use of the internal daisy chain even in systems with no external daisy chain or RETI command.

CRC Reset Codes 0 and 1 (D6 and D7). Together, these bits select one of the three following reset commands:

CRC Reset Code 1	CRC Reset Code 0	
0	0	Null Code (no affect)
0	1	Reset Receive CRC Checker
1	0	Reset Transmit CRC Generator
1	1	Reset Tx Underrun/End of Message latch

The Reset Transmit CRC Generator command normally initializes the CRC generator to all 0's. If the SDLC mode is selected, this command initializes the CRC generator to all 1's. The Receive CRC checker is also initialized to all 1's for the SDLC mode.

7.1.2 Write Register 1

WR1 (Figure 7-2) contains the control bits for the various interrupt and Wait/Ready modes.

D7	D6	D5	D4
Wait/Ready Enable	Wait or Ready Function	Wait/Ready On Receive/ Transmit	Receive Interrupt Mode 1

D3	D2	D1	D0
Receive Interrupt Mode 0	Status Affects Vector	Transmit Interrupt Enable	External Interrupts Enable

External/Status Interrupt Enable (D0). The External/ Status Interrupt Enable allows interrupts to occur as a result of transitions on the /DCD, /CTS, or /SYNC inputs, as a result of a Break/Abort detection and termination, or at the beginning of CRC or sync character transmission when the Transmit Underrun/EOM latch becomes set.

Transmitter Interrupt Enable (D1). If enabled, interrupts occur whenever the transmitter buffer becomes empty.

Status Affects Vector (D2). This bit is active in Channel B only. If this bit is not set, the fixed vector programmed in WR2 is returned from an interrupt acknowledge sequence. If this bit is set, the vector returned from an interrupt acknowledge is variable according to the following interrupt conditions:

	V3	V2	V1	
Ch B	0	0	0	Ch B Transmit Buffer Empty
	0	0	1	Ch B External/Status Change
	0	1	0	Ch B Receive Character Available
	1	1	1	Ch B Special Receive Condition*
Ch A	1	0	0	Ch A Transmit Buffer Empty
	1	0	1	Ch A External/Status Change
	1	1	0	Ch A Receive Character Available
	1	1	1	Ch A Special Receive Condition*

*Special Receive Conditions: Parity Error, Rx Overrun Error, Framing Error, End-of-Frame (SDLC).

Receive Interrupt Modes 0 and 1 (D3 and D4). Together these two bits specify the various character-available conditions. In Receive Interrupt modes 1, 2, and 3, a Special Receive Condition can cause an interrupt and modify the interrupt vector.

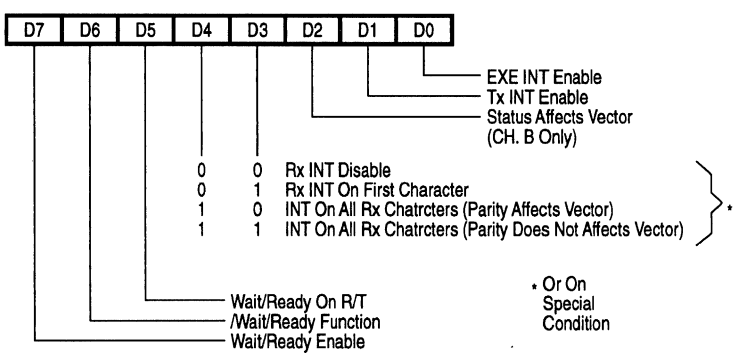


Figure 7-2. Write Register 1

D4 Receive Interrupt Mode 1	D3 Receive Interrupt Mode 0	
0	0	Receive Interrupts Disabled
0	1	Receive Interrupt On First Character Only
1	0	Interrupt On All Receive Characters—parity error is a Special Receive condition
1	1	Interrupt On All Receive Characters—parity error is not a Special Receive condition

Wait/Ready Function Selection (D7-D5). The Wait and Ready functions are selected by controlling D5, D6, and D7. Wait/Ready function is enabled by setting Wait/Ready Enable (WR1, D7) to 1. The Ready function is selected by setting D6 (Wait/Ready function) to 1. If this bit is 1, the /WAIT//READY output switches from High to Low when the Z80-SIO is ready to transfer data. The Wait function is selected by setting D6 to 0. If this bit is 0, the /WAIT//READY output is in the open-drain state and goes Low when active.

Both the Wait and Ready functions can be used in either the Transmit or Receive modes, but not both simultaneously. If D5 (Wait/Ready on Receive/Transmit) is set to 1, the Wait/Ready function responds to the condition of the receive buffer (empty or full). If D5 is set to 0, the Wait/Ready function responds to the condition of the transmit buffer (empty or full).

The logic states of the /WAIT//READY output when active or inactive depend on the combination of modes selected. Following is a summary of these combinations:

If D7 = 0	
and D6 = 1	**and D6 = 0**
/READY is High	/WAIT is floating

If D7 = 0			
and D5 = 0		**and D5 = 1**	
/READY	is High when transmit buffer is full.	/READY	is High when receive buffer is empty.
/WAIT	is Low when transmit buffer is full and SIO data port is an selected.	/WAIT	is Low when receive an buffer is empty and SIO data port is selected.
/READY	is Low when transmit buffer is empty.	/READY	is Low when receive buffer is full.
/WAIT	is floating when transmit buffer is empty.	/WAIT	is floating when receive buffer is full.

The /WAIT output High-to-Low transition occurs with the delay time tDIC(WR) after the I/O request. The Low-to-High transition occurs with the delay tDHΦ(WR) from the falling edge of Φ. The /READY output High-to-Low transition occurs with the delay tDLΦ(WR) from the rising edge of Φ. The /READY output Low-to-High transition occurs with the delay tDIC(WR) after /IORQ falls.

The Ready function can occur any time the Z80-SIO is not selected. When the /READY output becomes active Low, the DMA controller issues /IORQ and the corresponding B//A and C//D inputs to the Z80-SIO to transfer data. The /READY output becomes inactive as soon as /IORQ and /CS become active. Since the Ready function can occur internally in the Z80-SIO whether it is addressed or not, the /READY output becomes inactive when any CPU data or command transfer takes place. This does not cause problems because the DMA controller is not enabled when the CPU transfer takes place.

The Wait function, on the other hand, is active only if the CPU attempts to read Z80-SIO data that has not yet been received, which occurs frequently when block transfer instructions are used. The Wait function can also become active (under program control) if the CPU tries to write data while the transmit buffer is still full. The fact that the /WAIT output for either channel can become active when the opposite channel is addressed (because the Z80-SIO is addressed) does not affect operation of software loops or block move instructions.

7.1.3 Write Register 2

WR2 is the interrupt vector register; it exists in Channel B only. V7-V4 and V0 are always returned exactly as written; V3-V1 are returned as written if the Status Affects Vector (WR1, D2) control bit is 0. If this bit is 1, they are modified as explained in the previous section.

D7	D6	D5	D4	D3	D2	D1	D0
V7	V6	V5	V4	V3	V2	V1	V0

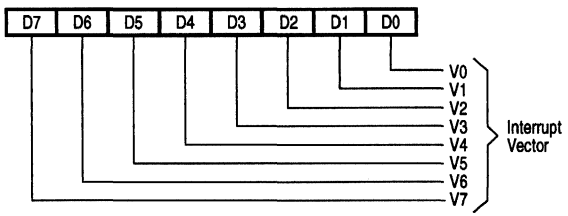


Figure 7-3. Write Register 2

7.1.4 Write Register 3

WR3 contains receiver logic control bits and parameters.

D7	D6	D5	D4
Receiver Bits/ Char 1	Receiver Bits/ Char 0	Auto Enables	Enter Hunt Phase

D3	D2	D 1	D0
Receiver CRC Enable	Address Search Mode	Sync Char Load Inhibit	Receiver Enable

Receiver Enable (D0). A 1 programmed into this bit allows receive operations to begin. This bit should be set only after all other receive parameters are set and receiver is completely initialized.

Sync Character Load Inhibit (D1). Sync characters preceding the message (leading sync characters) are not loaded into the receive buffers if this option is selected. Because CRC calculations are not stopped by sync character stripping, this feature should be enabled only at the beginning of the message.

Address Search Mode (D2). If SDLC is selected, setting this mode causes messages with addresses not matching the programmed address in WR6 or the global (1111 1111) address to be rejected. In other words, no receive interrupts can occur in the Address Search mode unless there is an address match.

Receiver CRC Enable (D3). If this bit is set, CRC calculation starts (or restarts) at the beginning of the last character transferred from the receive shift register to the buffer stack, regardless of the number of characters in the stack. See "SDLC Receive CRC Checking" (SDLC Receive section) and "CRC Error Checking" (Synchronous Receive section) for details regarding when this bit should be set.

Enter Hunt Phase (D4). The Z80-SIO automatically enters the Hunt phase after a reset; however, it can be re-entered if character synchronization is lost for any reason (Synchronous mode) or if the contents of an incoming message are not needed (SDLC mode). The Hunt phase is re-entered by writing a 1 into bit D4. This sets the Sync/Hunt bit (D4) in RR0.

Auto Enables (D5). If this mode is selected, /DCD and /CTS become the receiver and transmitter enables, respectively. If this bit is not set, /DCD and /CTS are simply inputs to their corresponding status bits in RR0.

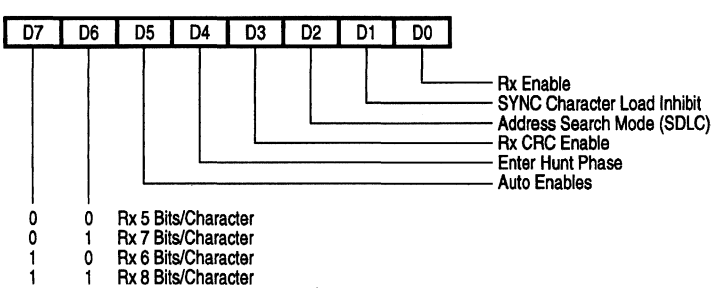


Figure 7-4. Write Register 3

Receiver Bits/Characters 1 and 0 (D7 and D6). Together, these bits determine the number of serial receive bits assembled to form a character. Both bits may be changed during the time that a character is being assembled, but they must be changed before the number of bits currently programmed is reached.

D7	D6	Bits/Character
0	0	5
0	1	7
1	0	6
1	1	8

7.1.5 Write Register 4

WR4 contains the control bits that affect both the receiver and transmitter. In the transmit and receive initialization routine, these bits should be set before issuing WR1, WR3, WR5, WR6, and WR7.

D7	D6	D5	D4	D3	D2	D1	D0
Clock Rate 1	Clock Rate 0	Sync Modes 1	Sync Modes 0	Stop Bits 1	Stop Bits 0	Parity Even/ /Odd	Parity

Parity (D0). If this bit is set, an additional bit position (in addition to those specified in the bits/character control) is added to transmitted data and is expected in receive data. In the Receive mode, the parity bit received is transferred to the CPU as part of the character, unless eight bits/character is selected.

Parity Even//Odd (D1). If parity is specified, this bit determines whether it is sent and checked as even or odd (1 = even).

Stop Bits 0 and 1 (D2 and D3). These bits determine the number of stop bits added to each asynchronous character sent. The receiver always checks for one stop bit. A special mode (00) signifies that a synchronous mode is to be selected.

D3 Stop Bits 1	D2 Stop Bits 0	
0	0	Sync modes
0	1	1 stop bit per character
1	0	1-1/2 stop bits per character
1	1	2 stop bits per character

Sync Modes 0 and 1 (D4 and D5). These bits select the various options for character synchronization.

Sync Mode 1	Sync Mode 0	
0	0	8-bit programmed sync
0	1	16-bit programmed sync
1	0	SDLC mode (0111 1110 flag pattern)
1	1	External Sync mode

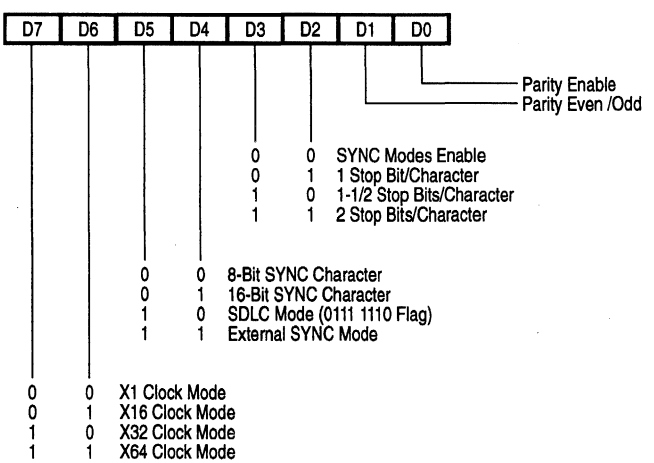


Figure 7-5. Write Register 4

Clock Rate 0 and 1 (D6 and D7). These bits specify the multiplier between the clock (/TxC and /RxC) and data rates. For synchronous modes, the x1 clock rate must be specified. Any rate may be specified for asynchronous modes; however, the same rate must be used for both the receiver and transmitter. The system clock in all modes must be at least 4.5 times the data rate. If the x1 clock rate is selected, bit synchronization must be accomplished externally.

Clock Rate 1	Clock Rate 0	
0	0	Data Rate x1 = Clock Rate
0	1	Data Rate x16 = Clock Rate
1	0	Data Rate x32 = Clock Rate
1	1	Data Rate x64 = Clock Rate

7.1.6 Write Register 5

WR5 contains control bits that affect the operation of transmitter, with the exception of D2, which affects the transmitter and receiver.

D7	D6	D5	D4	D3	D2	D1	D0
DTR	Tx Bits/ Char 1	Tx Bits/ Char 0	Send Break	Tx Enable	CRC-16/ /SDLC	RTS	Tx CRC Enable

Transmit CRC Enable (D0). This bit determines if CRC is calculated on a particular transmit character. If it is set at the time the character is loaded from the transmit buffer into the transmit shift register, CRC is calculated on the character. CRC is not automatically sent unless this bit is set when the Transmit Underrun condition exists.

Request To Send (D1). This is the control bit for The /RTS pin. When the /RTS bit is set, the /RTS pin goes Low; when reset, /RTS goes High. In the Asynchronous mode, /RTS goes High only after all the bits of the character are transmitted and the transmitter buffer is empty. In Synchronous modes, the pin directly follows the state of the bit.

CRC-16/SDLC (D2). This bit selects the CRC polynomial used by both the transmitter and receiver. When set, the CRC-16 polynomial ($X^{16}+X^{15}+X^2+1$) is used; when reset the SDLC polynomial ($X^{16}+X^{12}+X^5+1$) is used. If the SDLC mode is selected, the CRC generator and checker are preset to all 1's and a special check sequence is used. The SDLC CRC polynomial must be selected when the SDLC mode is selected. If the SDLC mode is not selected, the CRC generator and checker are preset to all 0's (for both polynomials).

Transmit Enable (D3). Data is not transmitted until this bit is set, and the Transmit Data output is held marking. Data or sync characters in the process of being transmitted are completely sent if this bit is reset after transmission has started. If the transmitter is disabled during the transmission of a CRC character, sync, or flag characters are sent instead of CRC.

Send Break (D4). When set, this bit immediately forces the Transmit Data output to the spacing condition, regardless of any data being transmitted. When reset, TxD returns to marking.

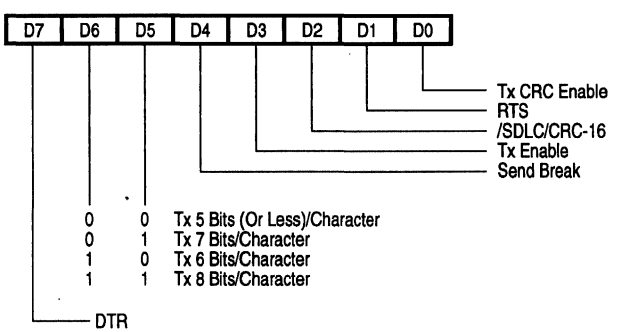


Figure 7-6. Write Register 5

Transmit Bits/Characters 0 and 1 (D5 and D6). Together, D6 and D5 control the number of bits in each byte transferred to the transmit buffer.

D6 Transmit Bits/ Character 1	D5 Transmit Bits/ Character 0	Bits/Character
0	0	Five or less
0	1	7
1	0	6
1	1	8

Bits to be sent must be right justified, least significant bits first. The Five Or Less mode allows transmission of one to five bits per character; however, the CPU should format the data character as shown in the following table.

D7	D6	D5	D4	D3	D2	D1	D0	
1	1	1	1	0	0	0	D	Sends one data bit
1	1	1	0	0	0	D	D	Sends two data bits
1	1	0	0	0	D	D	D	Sends three data bits
1	0	0	0	D	D	D	D	Sends four data bits
0	0	0	D	D	D	D	D	Sends five data bits

Data Terminal Ready (D7). This is the control bit for the DTR pin. When set, DTR is active (Low); when reset, DTR is inactive (High).

7.1.7 Write Register 6

This register is programmed to contain the transmit sync character in the Monosync mode, the first eight bits of a 16-bit sync character in the Bisync mode, or a transmit sync character in the External Sync mode. In the SDLC mode, it is programmed to contain the secondary address field used to compare against the address field of the SDLC frame.

D7	D6	D5	D4	D3	D2	D1	D0
Sync 7	Sync 6	Sync 5	Sync 4	Sync 3	Sync 2	Sync 1	Sync 0

7.1.8 Write Register 7

This register is programmed to contain the receive sync character in the Monosync mode, a second byte (last eight bits) of a 16-bit sync character in the Bisync mode, or a flag character (0111 1110) in the SDLC mode. WR7 is not used in the External Sync mode.

D7	D6	D5	D4	D3	D2	D1	D0
Sync 15	Sync 14	Sync 13	Sync 12	Sync 11	Sync 10	Sync 9	Sync 8

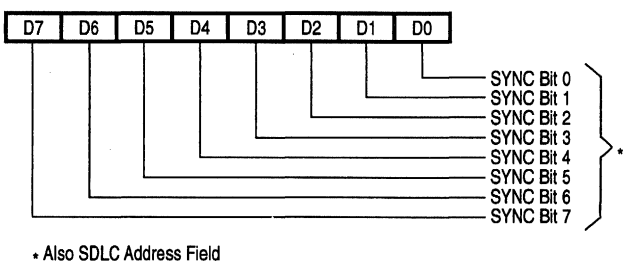


Figure 7-7. Write Register 6

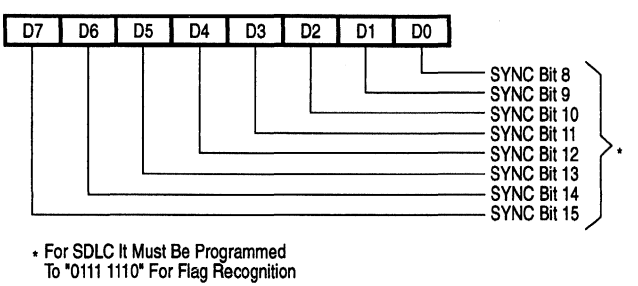


Figure 7-8. Write Register 7

7.2 READ REGISTERS

The Z80-SIO contains three registers, RR2-RR0 (Figures 7-9 through 7-11), that can be read to obtain the status information for each channel (except for RR2-Channel B only). The status information includes error conditions, interrupt vector and standard communications-interface signals.

To read the contents of a selected read register other than RR0, the system program must first write the pointer byte to WR0 in exactly the same way as a write register operation. Then, by executing an input instruction, the contents of the addressed read register can be read by the CPU.

The status bits of RR0 and RR1 are carefully grouped to simplify status monitoring. For example, when the interrupt vector indicates that a Special Receive Condition interrupt has occurred, all the appropriate error bits can be read from a single register (RR1).

7.2.1 Read Register 0

This register contains the status of the receive and transmit buffers; the /DCD, /CTS, and /SYNC inputs; the Transmit Underrun/EOM latch; and the Break/Abort latch.

D7	D6	D5	D4	D3	D2	D1	D0
Break/ Abort	Trans- mit Under- runt EOM	/CTS	Sync/ Hunt	/DCD	Trans- mit Buffer Empty	Inter- rupt Pend- ing (Ch. A only)	Receive Charac- ter Avail- able

Receive Character Available (D0). This bit is set when at least one character is available in the receive buffer; it is reset when the receive FIFO is completely empty.

Interrupt Pending (D1). Any interrupting condition in the Z80-SIO causes this bit to be set; however, it is readable only in Channel A. This bit is mainly used in applications that do not have vectored interrupts available. During the interrupt service routine in these applications, this bit indicates if any interrupt conditions are present in the Z80-SIO. This eliminates the need for analyzing all the bits of RR0 in both Channels A and B. Bit D1 is reset when all the interrupting conditions are satisfied. This bit is always 0 in Channel B.

Transmit Buffer Empty (D2). This bit is set whenever the transmit buffer becomes empty, except when a CRC character is being sent in a synchronous or SDLC mode. The bit is reset when a character is loaded into the transmit buffer. This bit is in the set condition after a reset.

Data Carrier Detect (D3). The /DCD bit shows the state of the /DCD input at the time of the last change of any of the five External/Status bits (/DCD, /CTS, Sync/Hunt, Break/ Abort or Transmit Underrun/EOM). Any transition of the /DCD input causes the /DCD bit to be latched and causes an External/Status interrupt. To read the current state of the /DCD bit, this bit must be read immediately following a Reset External/Status Interrupt command.

Sync/Hunt (D4). Since this bit is controlled differently in the Asynchronous, Synchronous and SDLC modes, its operation is somewhat more complex than that of the other bits and therefore requires more explanation.

In asynchronous modes, the operation of this bit is similar to the /DCD status bit, except that Sync/Hunt shows the state of the /SYNC input. Any High-to-Low transition on the /SYNC pin sets this bit and causes an External/Status interrupt (if enabled). The Reset External/Status Interrupt command is issued to clear the interrupt. A Low-to-High transition clears this bit and sets the External/Status inter-

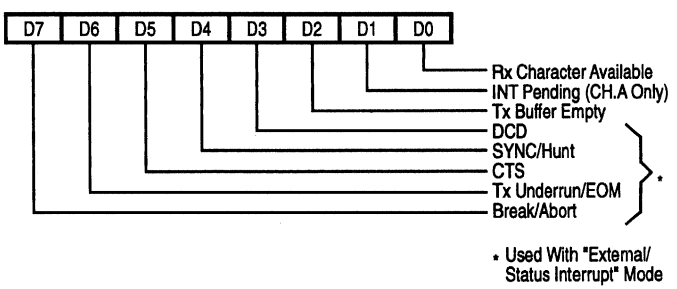


Figure 7-9. Read Register 0

rupt. When the External/Status interrupt is set by the change in state of any other input or condition, this bit shows the inverted state of the /SYNC pin at the time of the change. This bit must be read immediately following a Reset External/Status Interrupt command to read the current state of tile /SYNC input.

In the External Sync mode, the Sync/Hunt bit operates in a fashion similar to the Asynchronous mode, except the Enter Hunt Mode control bit enables the external sync detection logic. When the External Sync Mode and Enter Hunt Mode bits are set (for example, when the receiver is enabled following a reset), the /SYNC input must be held High by the external logic until external character synchronization is achieved. A High at the /SYNC input holds the Sync/Hunt status bit in the reset condition.

When external synchronization is achieved, /SYNC must be driven Low on the second rising edge of /RxC after that rising edge of /RxC on which the last bit of the sync character was received. In other words, after the sync pattern is detected, the external logic must wait for two full Receive Clock cycles to activate the /SYNC input, Once /SYNC is forced Low, it is a good practice to keep it Low until the CPU informs the external sync logic that synchronization has been lost or a new message is about to start. Refer to Figure 9-8 for timing details. The High-to-Low transition of the /SYNC input sets the Sync/Hunt bit, which, in turn, sets the External/Status interrupt. The CPU must clear the interrupt by issuing the Reset External/Status Interrupt command.

When the /SYNC input goes High again, another External/Status interrupt is generated that must also be cleared. The Enter Hunt Mode control bit is set whenever character synchronization is lost or the end of message is detected. In this case, the Z80-SIO again looks for a High-to-Low transition of the /SYNC input and the operation repeats as explained previously. This implies the CPU should also inform the external logic that character synchronization has been lost and that the Z80-SIO is wailing for /SYNC to become active.

In the Monosync and Bisync Receive modes, the Sync/Hunt status bit is initially set to 1 by the Enter Hunt Mode bit. The Sync/Hunt bit is reset when the Z80-SIO establishes character synchronization. The High-to-Low transition of the Sync/Hunt bit causes an External/Status interrupt that must be cleared by the CPU issuing the Reset External/Status Interrupt command. This enables the Z80-SIO to detect the next transition of other External/Status bits.

When the CPU detects the end of message or that character synchronization is lost, it sets the Enter Hunt Mode control bit, which-in turn-sets the Sync/Hunt bit to 1. The Low-to-High transition of the Sync/Hunt bit sets the External/Status interrupt, which must also be cleared by the Reset External/Status Interrupt command. Note that the /SYNC pin acts as an output in this mode and goes Low every time a sync pattern is detected in the data stream.

In the SDLC mode, the Sync/Hunt bit is initially set by the Enter Hunt mode bit, or when the receiver is disabled. In any case, it is reset to 0 when the opening flag of the first frame is detected by the Z80-SIO. The External/Status interrupt is also generated, and should be handled as discussed previously.

Unlike the Monosync and Bisync modes, once the Sync/Hunt bit is reset in the SDLC mode, it does not need to be set when the end of message is detected. The Z80-SIO automatically maintains synchronization. The only way the Sync/Hunt bit can be set again is by the Enter Hunt Mode bit, or by disabling the receiver.

Clear To Send (D5). This bit is similar to the /DCD bit, except that it shows the inverted state of the /CTS pin.

Transmit Underrun/End of Message (D6). This bit is in a set condition following a reset (internal or external). The only command that can reset this bit is the Reset Transmit Underrun/EOM Latch command (WR0, D6 and D7). When the Transmit Underrun condition occurs, this bit is set; its becoming set causes the External/ Status interrupt, which must be reset by issuing the Reset External/Status Interrupt command bits (WR0). This status bit plays an important role in conjunction with other control bits in controlling a transmit operation. Refer to "Bisync Transmit Underrun" and "SDLC Transmit Underrun" for additional details.

Break/Abort (D7). In the Asynchronous Receive mode, this bit is set when a Break sequence (null character plus framing error) is detected in the data stream. The External/Status interrupt, if enabled, is set when Break is detected. The interrupt service routine must issue the Reset External/Status Interrupt command (WR0, CMD2) to the break detection logic so the Break sequence termination can be recognized.

The Break/Abort bit is reset when the termination of the Break sequence is detected in the incoming data stream. The termination of the Break sequence also causes the External/Status interrupt to be set. The Reset External/Status Interrupt command must be issued to enable the break detection logic to look for the next Break sequence. A single extraneous null character is present in the receiver after the termination of a break; it should be read and discarded.

In the SDLC Receive mode, this status bit is set by the detection of an Abort sequence (seven or more 1's). The External/Status interrupt is handled the same way as in the case of a Break. The Break/Abort bit is not used in the Synchronous Receive mode.

7.2.2 Read Register 1

This register contains the Special Receive condition status bits and Residue codes for the I-Field in the SDLC Receive Mode.

D7	D6	D5	D4	D3	D2	D1	D0
End of Frame (SDLC)	CRC/ Framing Error	Receiver Overrun Error	Parity Error	Residue Code 2	Residue Code 1	Residue Code 0	All Sent

All Sent (D0). In asynchronous modes, this bit is set when all the characters have completely cleared the transmitter. Transitions of this bit do not cause interrupts. It is always set in synchronous modes.

Residue Codes 0, 1, and 2 (D3-D1). In those cases of the SDLC receive mode where the I-field is not an integral multiple of the character length, these three bits indicate the length of the I-field. These codes are meaningful only for the transfer in which the End-of-Frame bit is set (SDLC). For a receive character length of eight bits per character, the codes signify the following:

Residue Code 2	Residue Code 1	Residue Code 0	I-Field Bits in Previous Byte	I-Field Bits in Second Previous Byte
1	0	0	0	3
0	1	0	0	4
1	1	0	0	5
0	0	1	0	6
1	0	1	0	7
0	1	1	0	8
1	1	1	1	8
0	0	0	2	8

I-Field bits are right-justified in all cases.

If a receive character length different from eight bits is used for the 1-field, a table similar to the previous one may be constructed for each different character length. For no residue (that is, the last character boundary coincides with the boundary of the I-field and CRC field), the Residue codes are:

Bits per Character	Residue Code 2	Residue Code 1	Residue Code 0
8 Bits per Character	0	1	1
7 Bits per Character	0	0	0
6 Bits per Character	0	1	0
5 Bits per Character	0	0	1

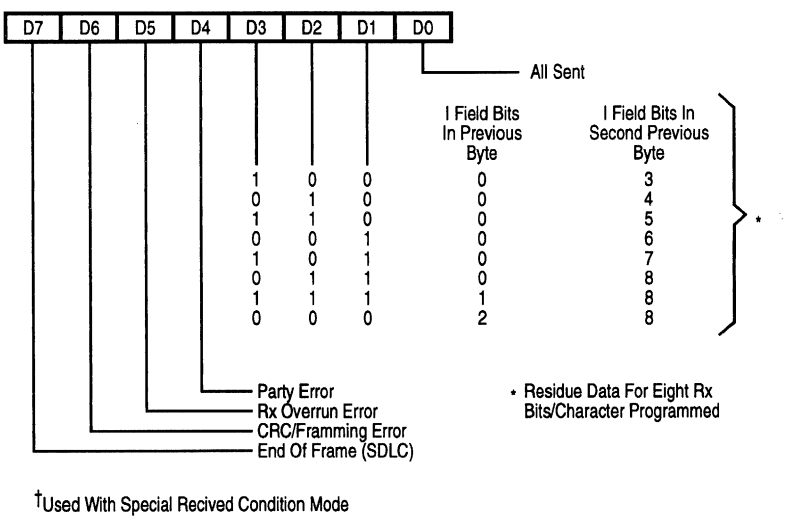


Figure 7-10. Read Register 1

Parity Error (D4). When parity is enabled, this bit is set for those characters whose parity does not match the programmed sense (even/odd). The bit is latched, so once an error occurs, it remains set until the Error Reset command (WR0) is given.

Receive Overrun Error (D5). This bit indicates that more than three characters have been received without a read from the CPU. Only the character that has been written over is flagged with this error, but when this character is read, the error condition is latched until reset by the Error Reset command. If Status Affects Vector is enabled, the character that has been overrun interrupts with a Special Receive Condition vector.

CRC/Framing Error (D6). If a Framing Error occurs (asynchronous modes), this bit is set (and not latched) for the receive character in which the Framing Error occurred. Detection of a Framing Error adds an additional one-half of a bit time to the character time so the Framing Error is not interpreted as a new start bit. In synchronous and SDLC modes, this bit indicates the result of comparing the CRC checker to the appropriate check value. This bit is reset by issuing an Error Reset command. The bit is not latched, so it is always updated when the next character is received. When used for CRC error and status in synchronous modes, it is usually set since most bit combinations result in a non-zero CRC except for a correctly completed message.

End-of-Frame (D7). This bit is used only with the SDLC mode and indicates that a valid ending flag has been received and that the CRC Error and Residue codes are also valid. This bit can be reset by issuing the Error Reset command. It is also updated by the first character of the following frame.

7.2.3 Read Register (Channel B Only)

This register contains the interrupt vector written into WR2 if the Status Affects Vector control bit is not set. If the control bit is set, it contains the modified vector shown in the Status Affects Vector paragraph of the Write Register 1 section. When this register is read, the vector returned is modified by the highest priority interrupting condition at the time of the read. If no interrupts are pending, the vector is modified with V3 = 0, V2 = 1, and V1 = 1. This register may be read only through Channel B.

D7	D6	D5	D4	D3	D2	D1	D0
V7	V6	V5	V4	V3	V2	V1	V0
				Variable if Status Affects Vector is enabled			

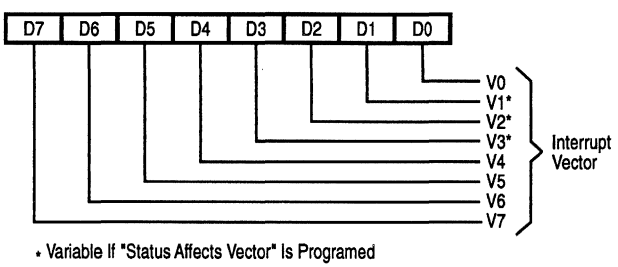


Figure 7-11. Read Register 2 (Channel B Only)

CHAPTER 8
APPLICATIONS

8.0 INTRODUCTION

The flexibility and versatility of the Z80-SIO make it useful for numerous applications, a few of which are included here. These examples show several applications that combine the Z80-SIO with other members of the Z80 family.

Figure 8-1 shows simple processor-to-processor communication over a direct line. Both remote processors in this system can communicate to the Z80-CPU with different protocols and data rates. Depending on the complexity of the application, other Z80 peripheral circuits (Z80-CTC, for example) may be required. The unused channel of the Z80-SIO can be used to control other peripherals or they can be connected to other remote processors.

Figure 8-2 illustrates how both channels of a single Z80-SIO are used with modems that have primary and secondary, or reverse channel options. Alternatively, two modems without these options can be connected to the Z80-SIO. A suitable baud-rate generator (Z80-CTC) must be used for asynchronous modems.

Figure 8-3 shows the Z80-SIO in a data concentrator, a relatively complex application that uses two Z80-SIOs to perform a variety of functions. The data concentrator can be used to collect data from many terminals over low-speed lines and transmit it over a single high-speed line after editing and reformatting.

The Z80-DMA controller circuit is used with Z80-SIO/2 to transmit the reformatted data at high speed with the required protocol. The high-speed modem provides the transmit clock for this channel. The Z80-CTC counter-timer circuit supplies the transmit and receive clocks for the low-speed lines and is also used as a time-out counter for various functions.

Z80-SIO/1 controls local or remote terminals. A single intelligent terminal is shown within the dashed lines. The terminal employs a Z80-SIO to communicate to the data concentrator on one channel while providing the interface to a line printer over its second channel. The intelligent terminal shown could be designed to operate interactively with the operator.

Depending on the software and hardware capabilities built into this system, the data concentrator can employ store-and-forward or hold-and-forward methods for regulating information traffic between slow terminals and the high-speed remote processor, If the high-speed channel is provided with a dial-out option, the channel can be connected to a number of remote processors over a switched line.

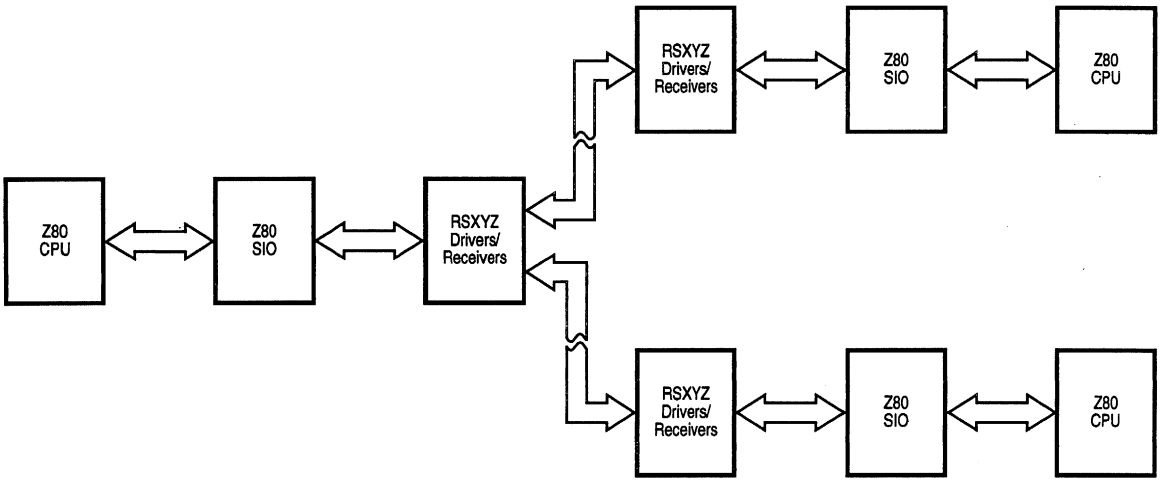


Figure 8-1. Synchronous/Asynchronous Processor-to-Processor Communication (Direct Wire to Two Remote Locations)

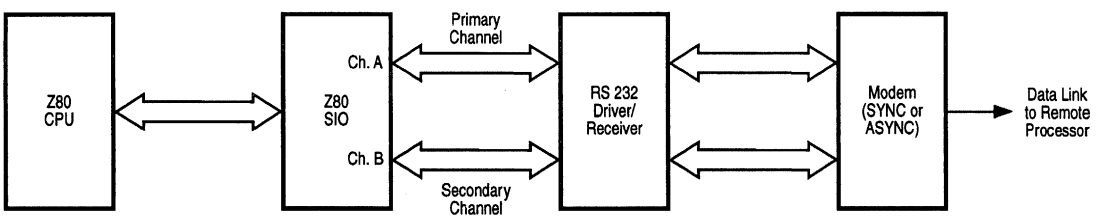


Figure 8-2. Synchronous/Asynchronous Processor-to-Processor Communication (Using Telephone Line)

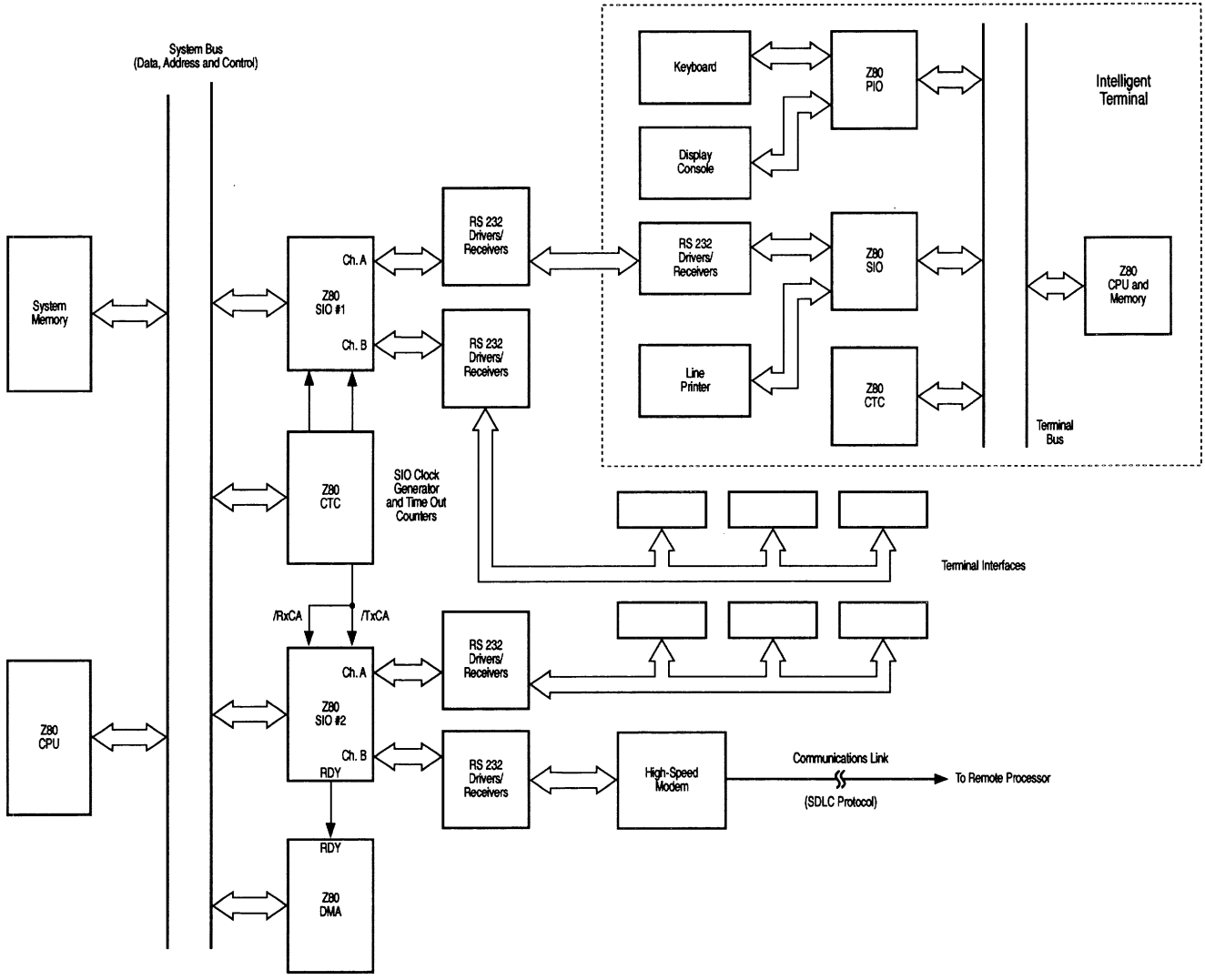


Figure 8-3. Data Concentrator

CHAPTER 9
TIMING

9.0 READ CYCLE

The timing signals generated by a Z80-CPU input instruction to read a Data or Status byte from the Z80-SIO are illustrated in Figure 9-1.

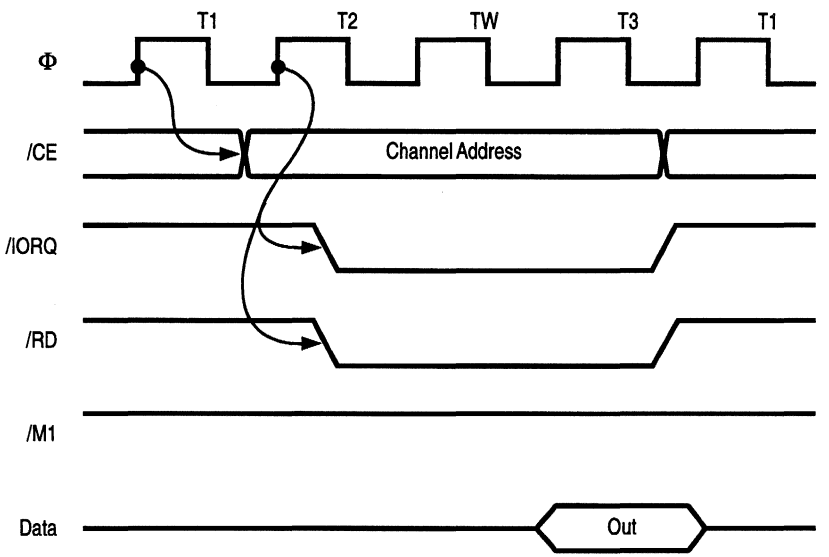


Figure 9-1. Read Cycle Timing

9.1 WRITE CYCLE

Figure 9-2 illustrates the timing and data signals generated by a Z80-CPU output instruction to write a Data or Control byte into the Z80-SIO.

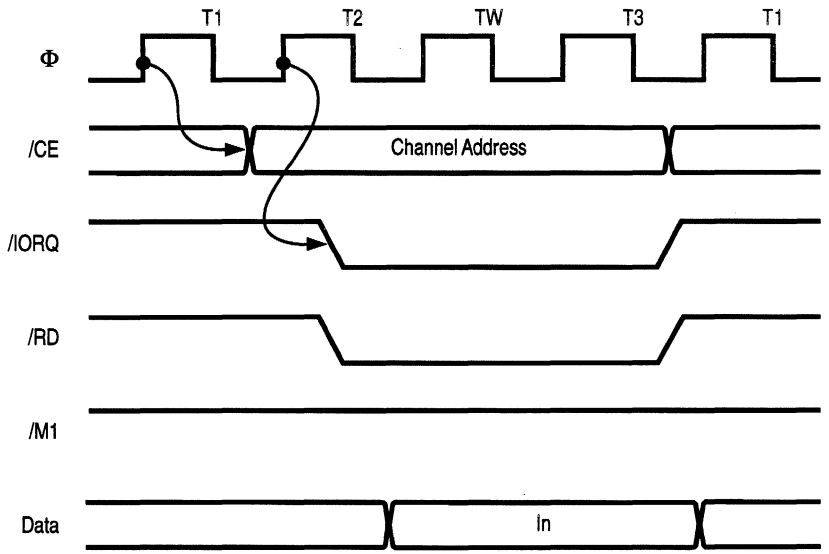


Figure 9-2. Write Cycle Timing

9.2 INTERRUPT ACKNOWLEDGE CYCLE

After receiving an Interrupt Request signal (/INT pulled Low), the Z80-CPU sends an Interrupt Acknowledge signal (/M1 and /IORQ both Low). The daisy-chained interrupt circuits determine the highest priority intertupt requestor. The IEI of the highest priority peripheral is terminated High. For any peripheral that has no interrupt pending or under service, IEO = IEI. Any peripheral that does have an interrupt pending or under service forces its IEO Low.

To insure stable conditions in the daisy chain, all interrupt status signals are prevented from changing while mi is Low. When /IORQ is Low, the highest priority interrupt requestor (the one with IEI High) places its interrupt vector on the data bus and sets its internal interrupt-underservice latch (Figure 9-3).

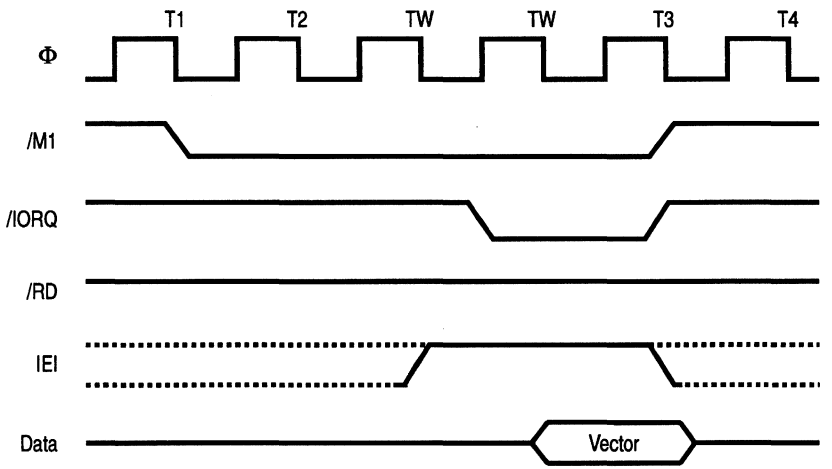


Figure 9-3. Interrupt Acknowledge Cycle Timing

E

9.3 RETURN FROM INTERRUPT CYCLE

Normally, the Z80-CPU issues a RETI (RETurn from Interrupt) instruction at the end of an interrupt service routine. RETI is a 2-byte opcode (ED-4D) that resets the interrupt-underservice latch to terminate the interrupt that has just been processed. This is accomplished by manipulating the daisy chain in the following way.

The normal daisy chain opetation can be used to detect a pending interrupt; however, it cannot distinguish between an interrupt under service and a pending unacknowledged interrupt of a higher priority. Whenever "ED" is decoded, the daisy chain is modified by forcing High the IEO of any interrupt that has not yet been acknowledged. Thus the daisy chain identifies the device presently under service as the only one with an IEI High and an IEO Low. If the next opcode byte is "4D," the interrupt-underservice latch is reset (Figure 9-4).

The Tipple time of the interrupt daisy chain (both the High-to-Low and the Low-to-High transitions) limits the number of devices that can be placed in the daisy chain. Ripple time can be improved with carry-lookahead, or by extending the interrupt acknowledge cycle. For further information about techniques for increasing the number of daisy-chained devices, refer to Zilog Application Note 03-0041 -01 (The Z80 Family Program Interrupt Structure).

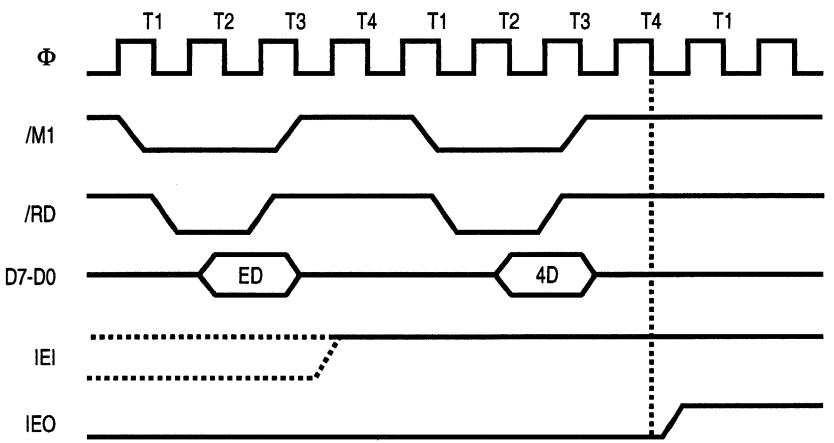


Figure 9-4. Return From Interrupt Cycle Timing

9.4 DAISY CHAIN INTERRUPT NESTING

Figure 9-5 illustrates the daisy chain configuration of interrupt circuits and their behavior with nested interrupts (an interrupt that is interrupted by another with a higher priority).

Each box in the illustration could be a separate external Z80 peripheral circuit with a user-defined order of interrupt priorities. However, a similar daisy chain structure also exists inside the Z80-SIO, which has six interrupt levels with a fixed order of priorities.

The case illustrated occurs when the transmitter of Channel B interrupts and is granted service. While this interrupt is being serviced, it is interrupted by a higher priority interrupt from Channel A. The second interrupt is serviced and-upon completion-a RETI instruction is executed or a RETI command is written into the Z80-SIO, resetting the interrupt-under-service latch of the Channel A interrupt. At this time, the service routine for Channel B is resumed. When it is completed, another RETI instruction is executed to complete the interrupt service.

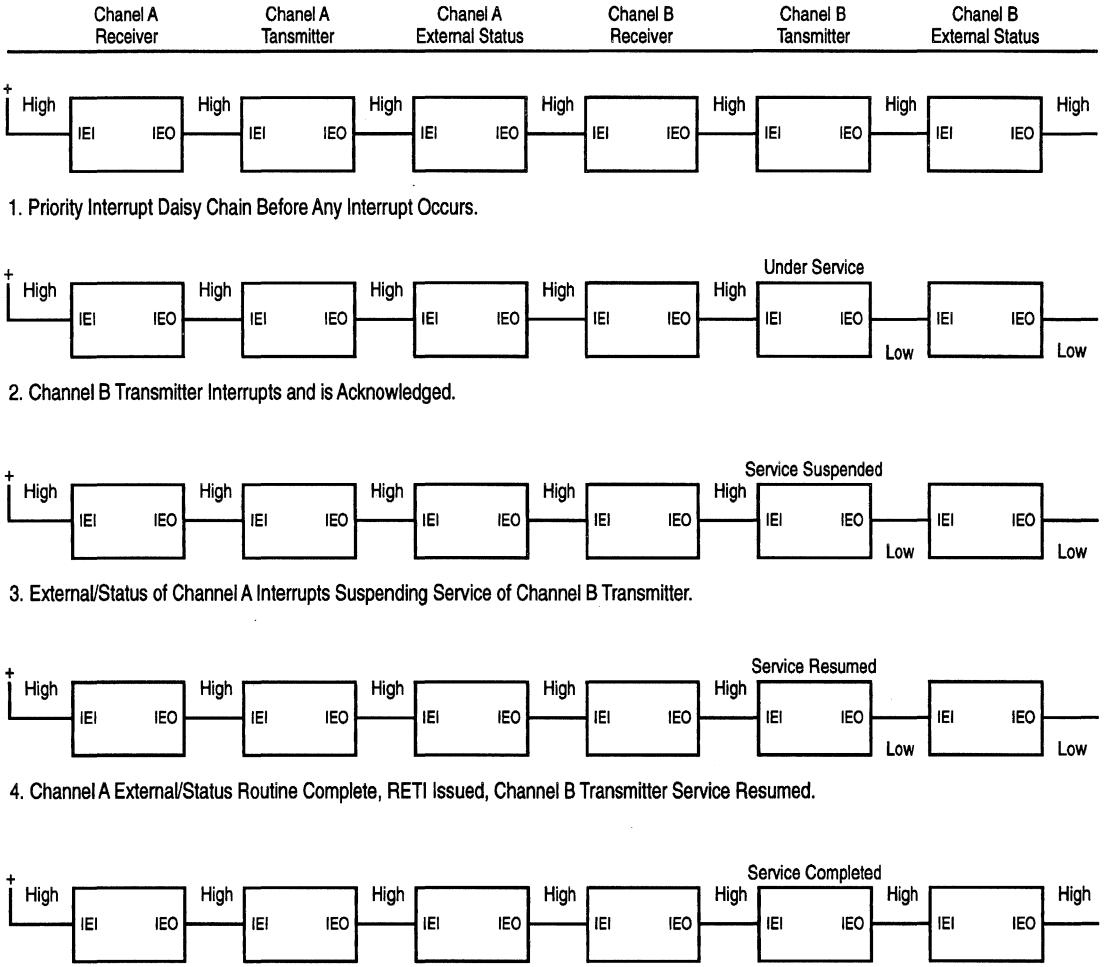
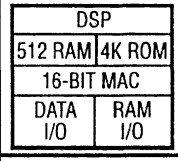
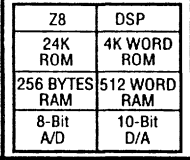
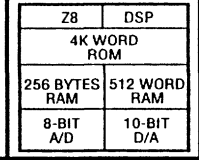
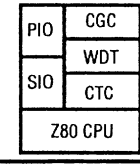
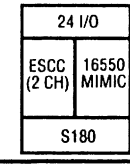
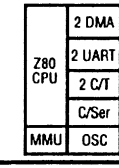
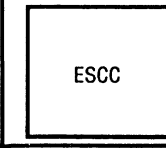


Figure 9-5. Typical Interrupt Service

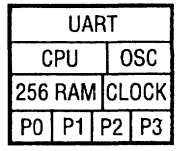
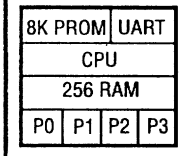
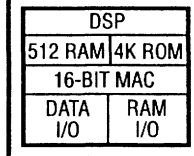
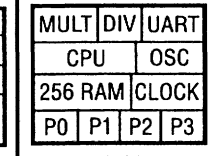
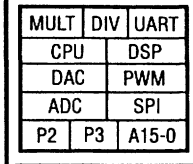
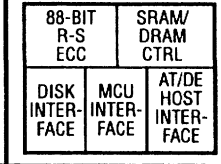
E

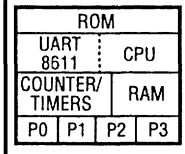
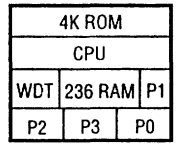
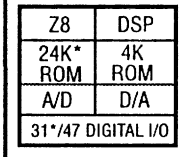
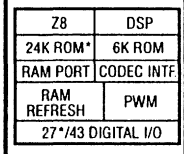
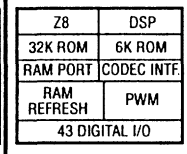
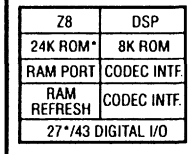
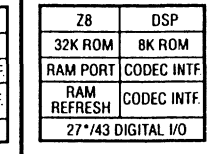
ZiLOG Fax/Modem Superintegration™ Products Guide

	Data Pump	Single Chip		Controllers			
Block Diagram							
Part #	**Z89C00**	**Z89120**	**Z89920**	**Z84C15**	**Z80182**	**Z80180**	**Z85230**
Description	16-Bit Digital Signal Processor	Zilog Modem/Fax Controller (ZMFC)	Zilog Modem/Fax Controller (ZMFC)	IPC/EIPC Controller	Zilog Intelligent Peripheral (ZIP™)	High-performance Z80® CPU with peripherals	Enhanced Serial Com. Controller
Process/Speed	CMOS 10, 15 MHz	CMOS 20 MHz	CMOS 20 MHz	CMOS 6, 10,16 MHz	CMOS 16, 20 MHz	6, 8, 10, 16*, 20* *Z8S180 only	CMOS 8, 10,16, 20 MHz
Features	16-bit Mac 75 ns 2 data RAMs (256 words each) 4K word ROM 64Kx16 Ext. ROM 16-bit I/O Port 74 instructions Most single cycle Two conditional branch inputs, two user outputs Library of software macros available zero overhead pointers	Z8® controller with 24 Kbyte ROM 16-bit DSP with 4K word ROM 8-bit A/D 10-bit D/A (PWM) Library of software macros available 47 I/O pins Two comparators Independent Z8® and DSP Operations Power-Down Mode	Z8 w/64K external memory DSP w/4K word ROM 8-bit A/D 10-bit D/A Library of macros 47 I/O pins Two comparators Independent Z8® and DSP Operations Power-Down Mode	Z80® CPU, SIO, CTC WDT, CGC The Z80 Family in one device Power-On Reset Two chip selects 32-bit CRC WSG EV mode[1] 3 and 5 Volt Version	Complete Static Version of Z180™ plus ESCC (2 channels of Z85230) 16550 MIMIC 24 Parallel I/O Emulation Modes[1]	Enhanced Z80® CPU MMU 1 Mbyte 2 DMAs 2 UARTs with BRGs C/Serial I/O Port Oscillator Z8S180 includes; Pwr dwn, Prgmble EMI, divide-by-one clock option	Full dual-channel SCC plus deeper FIFOs: 4 bytes on Tx 8 bytes on Rx DPLL counter per channel Software compatible to SCC
Package	68-pin PLCC 60-pin VQFP	68-pin PLCC	68-pin PLCC	100-pin QFP 100-pin VQFP	100-pin QFP 100-pin VQFP	64-pin DIP 68-pin PLCC 80-pin QFP	40-pin DIP 44-pin PLCC
Other Applications	16-bit General-Purpose DSP TMS 32010/20/25 applications	Multimedia-Audio Voicemail Speech Storage and Transmission Modems FAXes, Sonabouys	Multimedia-Audio Voicemail Speech Storage and Transmission Modems FAXes, Sonabouys	Intelligent peripheral controllers Modems	General-Purpose Embedded Control Modem, Fax, Data Communications	Embedded Control	General-Purpose datacom. High performance SCC software compatible upgrade

S

ZiLOG Mass Storage Superintegration™ Products Guide

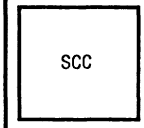
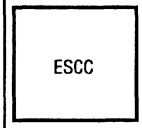
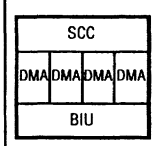
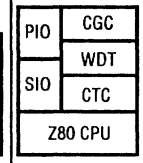
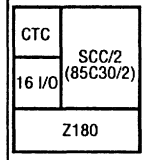
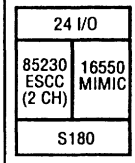
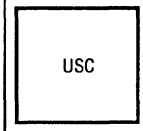
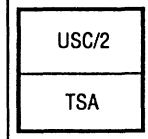
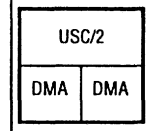
Block Diagram						
Part #	**Z86C91/Z8691**	**Z86E21**	**Z89C00**	**Z86C93**	**Z86C95**	**Z86018**
Description	ROMless Z8®	Z8® 8K OTP	16-Bit Digital Signal Processor	Enhanced Z8®	Enhanced Z8® with DSP	Zilog Datapath Controller (ZDPC)
Process/Speed	CMOS 16 MHz (C91) NMOS 12 MHz (91)	CMOS 12, 16 MHz	CMOS 10, 15 MHz	CMOS 20, 25 MHz	CMOS 24 MHz	CMOS 40 MHz
Features	Full duplex UART 2 Standby Modes (STOP and HALT) 2x8 bit Counter/Timer	8K OTP ROM 256 Byte RAM Full-duplex UART 2 Standby Modes (STOP and HALT) 2 Counter/Timers ROM Protect option RAM Protect option Low EMI option	16-bit Mac 75 ns 2 data RAMs (256 words each) 4K word ROM 64Kx16 Ext. ROM 16-bit I/O Port 74 instructions Most single cycle Two conditional branch inputs, two user outputs Library of software macros available zero overhead pointers	16x16 Multiply 1.7 µs 32x16 Divide 2.0 µs Full duplex UART 2 Standby Modes (STOP and HALT) 3 16-bit Counter/Timers Pin compatible to Z86C91 (PDIP)	8 channel 8-bit ADC, 8-bit DAC 16-bit Multiply/Divide Full duplex UART SPI (Serial Peripheral Interface) 3 Standby Modes (STOP/HALT/PAUSE) Pulse Width Modulator 3x16-bit timer 16-bit DSP slave processor 83 ns Mult./Accum.	Full track read Automatic data transfer (Point & Go®) 88-bit Reed Solomon ECC "on the fly" Full AT/IDE bus interface 64 KB SRAM buffer 1 MB DRAM buffer Split data field support 100-pin VQFP package JTAG boundary scan option Up to 8 KB buffer RAM reserved for MCU
Package	40-pin DIP 44-pin PLCC 44-pin QFP	40-pin DIP 44-pin PLCC 44-pin QFP	68-pin PLCC 60-pin VQFP	40-pin DIP 44-pin PLCC 44-pin QFP 48-pin VQFP	80-pin QFP 84-pin PLCC 100-pin VQFP	100-pin VQFP 100-pin QFP
Application	Disk Drives Modems Tape Drives	Software Debug Z8® prototyping Z8® production runs Card Reader	Disk Drives Tape Drives Servo Control Motor Control	Disk Drives Tape Drives Modems	Disk Drives Tape Drives Servo Control Motor Control	Hard Disk Drives

Block Diagram							
Part #	**Z08600/Z08611**	**Z86C30/E30 Z86C40/E40**	**Z89C65/C66**	**Z89C67/C68**	**Z89C69**	**Z89167/168**	**Z89169**
Description	Z8® NMOS (CCP™) 8600 = 2K ROM 8611 = 4K ROM	Z8® Consumer Controller Processor (CCP™) with 4K ROM C30 = 28-pin C40 = 40-pin E30/E40 = OTP version	Telephone Answering Controller with DSP LPC voice synthesis and DTMF detection, External ROM /RAM Interface (C66)	Telephone Answering Controller with digital voice encode and decode DTMF detection and full memory control interface. Ext. ROM/RAM Inttc. (C68)	Telephone Answering Controller with digital voice encode and decode DTMF detection and full memory control interface	Enhanced telephone answering controller with digital voice encode and decode DTMF detection and full memory controller inttc. ext. ROM/RAM inttc. (168)	Enhanced telephone answering controller with digital voice encode and decode DTMF detection and full memory controller interface
Process/Speed	NMOS 8,12 MHz	CMOS 12 MHz	CMOS 20 MHz	CMOS 20 MHz	CMOS 20 MHz	CMOS 24 MHz	CMOS 24 MHz
Features	2K/4K ROM 128 Bytes RAM 22/32 I/O lines On-chip oscillator 2 Counter/Timers 6 vectored, priority interrupts UART (Z8611)	4K ROM, 236 RAM 2 Standby Modes 2 Counter/Timers ROM Protect RAM Protect 4 Ports (86C40/E40) 3 Ports (86C30/E30) Brown-Out Protection 2 Analog Comparators Low EMI Watch-Dog Timer Auto Power-On Reset Low Power option	Z8® Controller 24K ROM (C65) 16-bit DSP 4K Word ROM 8-bit A/D with AGC DTMF macro available LPC macro available 10-bit PWM D/A Other DSP software options available 47 I/O Pins (C65) * = Note Z89C66 is ROMless (Z8) with 31 I/O pins.	Z8® Controller 24K ROM (C67) 16-bit DSP 6K Word ROM DTMF macro available LPC macro available 10-bit PWM D/A Other DSP S/W opt. avail. ARAM/DRAM/ROM Controller & Interface Dual Codec Interface 43 I/O (C67) * = Note Z89C68 is ROMless (Z8) with 27 I/O pins.	Z8® Controller 32K ROM 16-bit DSP 6K Word ROM DTMF macro available LPC macro available 10-bit PWM D/A Other DSP software options available ARAM/DRAM/ROM Controller & Interface Dual Codec Interface 43 I/O	Z8® Controller 24K ROM (167) 16-bit DSP 8K Word ROM DTMF Macro available LPC Macro available 10-bit PWM D/A Other DSP software options available ARAM/DRAM/ROM Dual Codec Interface 43 I/O (167) * = Note Z89168 is ROMless (Z8) with 27 I/O pins.	Z8® Controller 32K ROM 16-bit DSP 8K Word ROM DTMF Macro available LPC Macro available 10-bit PWM D/A Other DSP software options available ARAM/DRAM/ROM Dual Codec Interface 43 I/O
Package	28-pin DIP 40-pin DIP 44-pin PLCC	28-pin DIP 40-pin DIP 44-pin PLCC, QFP	68-pin PLCC	84-pin PLCC	84-pin PLCC	84-pin PLCC 80-pin QFP	84-pin PLCC 80-pin QFP
Application	Low cost tape board TAD	Cordless Phone TAD	Fully featured cassette answering machines with voice prompts and DTMF signaling Digital OGM available	Voice Processing, DSP applications in tapeless TAD and other high-performance voice processors	Voice Processing, DSP applications in tapeless TAD and other high-performance voice processors	Voice Processing, DSP applications in tapeless TAD and other high-performance voice processors	Voice Processing, DSP applications in tapeless TAD and other high-performance voice processors

ZiLOG Video Products Superintegration™ Products Guide

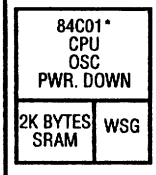
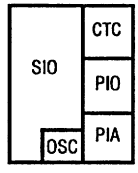
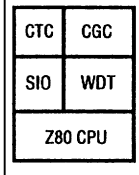
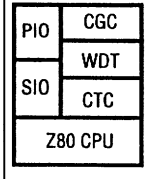
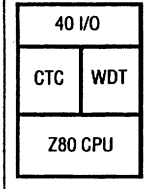
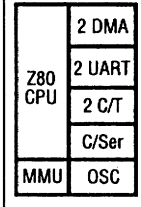
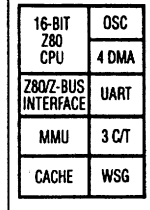
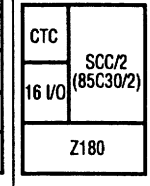
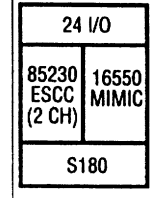
	TV Controller			IR Controller		Cable TV	
Block Diagram	8K ROM 4K CHAR ROM Z8 CPU | RAM OSD 13 PWM | TIMER WDT | 5 PORTS	6K ROM 3K CHAR ROM Z8 CPU | RAM OSD 7 PWM | TIMER WDT | 3 PORTS	CHAR ROM COMMAND INTERPRETER ANALOG SYNC/DATA SLICER | OSD CTRL	1K/6K ROM Z8 CPU WDT | 124 RAM P2 | P3	2K/8K/16K ROM Z8 CPU WDT | 128,256, 768 RAM P0 | P1 | P2 | P3	4K ROM CPU WDT | 236 RAM | P1 P2 | P3 | P0	16K ROM | UART CPU | 236 RAM P0 | P1 | P2 P3 | P4 | P5 | P6
Part #	**Z86C27/127/97**	**Z86227**	**Z86128**	**Z86L06/L29**	**Z86L70/71/72** (Q193)	**Z86C40/E40**	**Z86C61/62**
Description	Z8® Digital Television Controller MCU with logic functions needed for Television Controller, VCRs and Cable	Standard DTC features with reduced ROM, RAM, PWM outputs for greater economy	Line 21 Controller (L21C™) for Closed Caption Television	18-pin Z8® Consumer Controller Processor (CCP™) low-voltage and low-current battery operation 1K-6K ROM	Z8® (CCP™) low-voltage parts that have more ROM, RAM and special Counter/Timers for automated output drive capabilities	Z8® Consumer Controller Processor (CCP™) with 4K ROM (C40) E40 = OTP version	Z8® MCU with Expanded I/O's and 16K ROM
Process/Speed	CMOS 4 MHz	CMOS 4 MHz	CMOS 12 MHz	Low Voltage CMOS 8 MHz	Low Voltage CMOS 8 MHz	CMOS 12 MHz	CMOS 16, 20 MHz
Features	Z8/DTC Architecture 8K ROM, 256-byte RAM 160x7-bit video RAM On-Screen Display (OSD) video controller Programmable color, size, position attributes 13 PWMs for D/A conversion 128-character set 4Kx6-bit char. Gen. ROM Watch-Dog Timer (WDT) Brown-Out Protection 5 Ports/36 pins 2 Standby Modes Low EMI Mode	Z8/DTC Architecture 6K ROM, 256-byte RAM 120x7-bit video RAM OSD on board Programmable color, size, position attributes 7 PWMs 96-character set 3Kx6-bit character generator ROM Watch-Dog Timer (WDT) Brown-Out Protection 3 Ports/20 pins 2 Standby Modes Low EMI Mode	Conforms to FCC Line 21 format Parallel or serial modes Stand-alone operation On-board data sync and slicer On-board character generator - Color - Blinking - Italic - Underline	Z8® Architecture 1K ROM & 6K ROM Watch-Dog Timer 2 Analog Comparators with output option 2 Standby Modes 2 Counter/Timers Auto Power-On Reset 2 volt operation RC OSC option Low Noise option Brown-Out Protection High current drivers (2, 4)	Z8® Architecture 2K/8K/16K ROM Watch-Dog Timer 2 Analog Comparators with output option 2 Standby Modes 2 Enhanced Counter/Timers, Auto Pulse Reception/Generation Auto Power-On Reset 2 volt operation RC OSC option Brown-Out Protection High current drivers (4)	4K ROM, 236 RAM 2 Standby Modes 2 Counter/Timers ROM Protect RAM Protect 4 Ports Brown-Out Protection 2 Analog Comparators Low EMI Watch-Dog Timer Auto Power-On Reset Low Power option	16K ROM Full duplex UART 2 Standby Modes (STOP and HALT) 2 Counter/Timers ROM Protect option RAM Protect option Pin compatible to Z86C21 C61 = 4 Ports C62 = 7 Ports
Package	64-pin DIP 52-pin active (127)	40-pin DIP	18-pin DIP	18-pin DIP 18-pin SOIC	20-pin DIP (L71), 18-pin DIP, SOIC (L70) 40,44-pin DIP, PLCC, QFP (L72)	40-pin DIP	40-pin DIP (C61) 44-pin PLCC,QFP (C61) 68-pin PLCC (C62)
Application	Low-end Television Cable/Satellite Receiver	Low-end Television Cable/Satellite Receiver	TVs, VCRs, Decoders	I.R. Controller Portable battery operations	I.R. Controller Portable battery operations	Window Control Wiper Control Sunroof Control Security Systems TAD	Cable Television Remote Control Security

Zilog Datacommunications Superintegration™ Products Guide

Block Diagram									
Part #	**Z8030/Z80C30 Z8530/Z85C30**	**Z85230/Z80230 Z85233***	**Z16C35**	**Z84C15**	**Z80181**	**Z80182**	**Z16C30**	**Z16C33**	**Z16C32**
Description	Serial Com. Controller	Enhanced Serial Com. Controller	Integrated Serial Com. Controller	Intelligent Peripheral Controller	Smart Access Controller	Zilog Intelligent Peripheral	Universal Serial Controller	Mono-channel Universal Serial Controller	Integrated Universal Serial Controller
Process/ Speed/ Clock Data Rate	NMOS: 4, 6, 8 MHZ CMOS: 8,10 16 MHz 2, 2.5, 4 Mb/s	CMOS: 10, 16 20 MHz 2.5, 4.0, 5.0 Mb/s	CMOS: 10, 16 MHz 2.5, 4.0 Mb/s	CMOS 6, 10,16 MHz	10, 12.5	CMOS 16, 20 MHz	CMOS: 20 MHz CPU Bus 10 Mb/s 20 Mb/s	CMOS: 10 MHz CPU Bus 10 Mb/s	CMOS:20 MHz CPU Bus 16 Mb/s 20 Mb/s
Features	Two independent full-duplex channels Enhanced DMA support: 10x19 status FIFO 14-bit byte counter NRZ/NRZI/FM	Full dual-channel SCC plus deeper FIFOs: 4 bytes on Tx 8 bytes on Rx DPLL counter per channel Software compatible to SCC *One channel of Z85230	Full dual-channel SCC plus 4 DMA controllers and a bus interface unit	Z80® CPU, SIO, CTC WDT, CGC The Z80 Family in one device Power-On Reset Two chip selects 32-bit CRC WSG EV mode[1] 3 and 5 Volt Version	Complete Z180™ plus SCC/2 CTC 16 I/O lines Emulation Mode[1]	Complete Static version of Z180 plus ESCC (2 channels of 85230) 16550 MIMIC 24 Parallel I/O Emulation Mode[1]	Two dual-channel 32-byte receive & transmit FIFOs 16-bit bus B/W: 18.2 Mb/s 2 BRGs per channel Flexible 8/16-bit bus interface	Single-channel (half of USC™) plus Time Slot Assigner functions for ISDN	Single-channel (half of USC) plus two DMA controllers Array chained and linked-list modes with ring buffer support
Package	40-pin DIP 44-pin CERDIP 44-pin PLCC	40-pin DIP 44-pin PLCC *44-pin QFP (85233)	68-pin PLCC	100-pin QFP 100-pin VQFP	100-pin QFP	100-pin QFP 100-pin VQFP	68-pin PLCC	68-pin PLCC	68-pin PLCC
Application	General-Purpose datacom.	General-Purpose datacom. High performance SCC software	High performance datacom. SCC upgrades	Intelligent peripheral controllers Modems	Intelligent peripheral controllers Printers, Faxes, Modems, Terminals	General-Purpose Embedded Control Modem, Fax, Data Communications	General-Purpose high-end datacom. Ethernet HDLC X.25 Frame Relay	General-Purpose high-end datacom. Ethernet HDLC X.25 Frame Relay	General-Purpose high-end datacom. Ethernet HDLC X.25 Frame Relay

AppleTalk® A Registered Trademark of Apple Computer, Inc

ZiLOG Z80® Embedded Controllers Superintegration™ Products Guide

Block Diagram									
Part #	**Z84C50**	**Z84C90**	**Z84013/C13**	**Z84015/C15**	**Z84011/C11**	**Z80180/S180**	**Z80280**	**Z80181**	**Z80182**
Description	Z80/84C01 with 2K SRAM	Killer I/O (3 Z80 peripherals)	Intelligent Peripheral Controller	Intelligent Peripheral Controller	Parallel I/O Controller	High-performance Z80® CPU with peripherals	16-bit Z80® code compatible CPU with peripherals	Smart Access Controller	Zilog Intelligent Peripheral
Speed MHz	10	8, 10, 12.5	6, 10	6, 10, 16	6, 10	6, 8, 10, 16*, 20* *Z8S180 only	10, 12	10, 12.5	16, 20
Features	Z80® CPU 2 Kbytes SRAM WSG Oscillator Pin compatible with Z84C00 DIP & PLCC EV mode[1] *84C01 is available as a separate part	SIO, PIO, CTC plus 8 I/O lines	Z80® CPU, SIO, CTC WDT, CGC, WSG, Power-On Reset 2 chip selects EV mode[1]	Z80® CPU, SIO, CTC WDT, CGC The Z80 Family in one device Power-On Reset Two chip selects 32-bit CRC WSG EV mode[1]	Z80® CPU, CTC, WDT 40 I/O lines bit programmable Power-On Reset EV mode[1]	Enhanced Z80 CPU MMU 1 Mbyte 2 DMAs 2 UARTs with BRGs C/Serial I/O Port Oscillator Z8S180 includes; Pwr dwn, Prgmble EMI, divide-by-one clock option	16-bit code compatible Z80® CPU Three stage pipeline MMU 16 Mbyte CACHE 256 byte Inst. & Data Peripherals 4 DMAs, UART, 3 16-bit C/T, WSG Z80/Z-BUS® interface	Complete Z180 plus SCC/2 CTC 16 I/O lines Emulation Mode[1]	Complete Static Version of Z180™ plus ESCC (2 channels of Z85230) 16550 MIMIC 24 Parallel I/O Emulation Modes[1]
Package	40-pin DIP 44-pin PLCC 44-pin QFP	84-pin PLCC	84-pin PLCC	100-pin QFP 100-pin VQFP	100-pin QFP	64-pin DIP 68-pin PLCC 80-pin QFP	68-pin PLCC	100-pin QFP	100-pin QFP 100-pin VQFP
Application	Embedded Controllers	General-purpose peripheral that can be used with Z80 and other CPU's	Intelligent datacom controllers	Intelligent peripheral controllers Modems	Intelligent parallel-I/O controllers Industrial display terminals	Embedded Control	Embedded Control Terminals Printers	Intelligent peripheral controllers Printers, Faxes, Modems, Terminals	General-Purpose Embedded Control Modem, Fax, Data Communications

[1] Allows use of existing development systems.

	Z8036 Z8536	Z32H00	Z5380 Z53C80	Z85C80
Description	Counter/Timer & parallel I/O Unit (CIO)	Hyperstone Enhanced Fast Instruction Set Computer (EFISC) Embedded (RISC) Processor	Small Computer System Interface (SCSI)	Serial Communication Controller and Small Computer System Interface
Process/ Speed	NMOS 4,6 MHz	CMOS 25 MHz	CMOS Z5380: 1.5 MB/s Z53C80: 3.0 MB/s	CMOS SCC - 10, 16 MHz SCSI - 3.0 MB/s
Features	Three 16-bit Counter/Timers, Three I/O ports with bit catching, pattern matching interrupts and handshake I/O	32-bit MPU 4 Gbytes address space 19 global and 64 local registers of 32 bits each 128 bytes instruction cache 1.2µ CMOS 42 mm² die	ANSI X3.131-1986 Direct SCSI bus interface On-board 48 mA drivers Normal or Block mode DMA transfers Bus interface, target and initiator	Full dual-channel SCC plus SCSI sharing databus and read/write functions
Package	40-pin PDIP 44-pin PLCC	144-pin PGA 132-pin QFP	Z5380: 40-pin DIP 44-pin PLCC Z53C80: 48-pin DIP 44-pin PLCC	68-pin PLCC
Application	General-Purpose Counter/Timers and I/O system designs	Embedded high-performance industrial controller Workstations	Bus host adapters, formatters, host ports	AppleTalk® networking SCSI disk drives

[1] Software and hardware compatible with discrete devices.

ZiLOG

Z8®/SUPER8™ MICROCONTROLLER FAMILY

Databooks	Part No	Unit Cost
Z8 Microcontrollers Databook (Includes the following documents)	**DC-8275-04**	**5.00**

Z8 CMOS Microcontrollers

Z86C00/C10/C20 MCU OTP Product Specification
Z86C06 Z8 CCP™ Preliminary Product Specification
Z86C08 8-Bit MCU Product Specification
Z86E08 Z8 OTP MCU Product Specification
Z86C09/19 Z8 CCP Product Specification
Z86E19 Z8 OTP MCU Advance Information Specification
Z86C11 Z8 MCU Product Specification
Z86C12 Z8 ICE Product Specification
Z86C21 Z8 MCU Product Specification
Z86E21/Z86E22 OTP Product Specification
Z86C30 Z8 CCP Product Specification
Z86E30 Z8 OTP CCP Product Specification
Z86C40 Z8 CCP Product Specification
Z86E40 Z8 OTP CCP Product Specification
Z86C27/97 Z8 DTC™ Product Specification
Z86127 Low-Cost Digital Television Controller Adv. Info. Spec.
Z86C50 Z8 CCP ICE Advance Information Specification
Z86C61 Z8 MCU Advance Information Specification
Z86C62 Z8 MCU Advance Information Specification
Z86C89/C90 CMOS Z8 CCP Product Specification
Z86C91 Z8 ROMless MCU Product Specification
Z86C93 Z8 ROMless MCU Preliminary Product Specification
Z86C94 Z8 ROMless MCU Product Specification
Z86C96 Z8 ROMless MCU Advance Information Specification
Z88C00 CMOS Super8 MCU Advance Information Specification

Z8 NMOS Microcontrollers

Z8600 Z8 MCU Product Specification
Z8601/03/11/13 Z8 MCU Product Specification
Z8602 8-Bit Keyboard Controller Preliminary Product Spec.
Z8604 8-Bit MCU Product Specification
Z8612 Z8 ICE Product Specification
Z8671 Z8 MCU With BASIC/Debug Interpreter Product Spec.
Z8681/82 Z8 MCU ROMless Product Specification
Z8691 Z8 MCU ROMless Product Specification
Z8800/01/20/22 Super8 ROMless/ROM Product Specification

Peripheral Products

Z86128 Closed-Captioned Controller Adv. Info. Specification
Z765A Floppy Disk Controller Product Specification
Z5380 SCSI Product Specification
Z53C80 SCSI Advance Information Specification

Z8 Application Notes and Technical Articles

Zilog Family On-Chip Oscillator Design
Z86E21 Z8 Low Cost Thermal Printer
Z8 Applications for I/O Port Expansions
Z86C09/19 Low Cost Z8 MCU Emulator
Z8602 Controls A 101/102 PC/Keyboard
The Z8 MCU Dual Analog Comparator
The Z8 MCU In Telephone Answering Systems
Z8 Subroutine Library
A Comparison of MCU Units
Z86xx Interrupt Request Registers
Z8 Family Framing
A Programmer's Guide to the Z8 MCU
Memory Space and Register Organization

Super8 Application Notes and Technical Articles

Getting Started with the Zilog Super8
Polled Async Serial Operations with the Super8
Using the Super8 Interrupt Driven Communications
Using the Super8 Serial Port with DMA
Generating Sine Waves with Super8
Generating DTMF Tones with Super8
A Simple Serial Parallel Converter Using the Super8

Additional Information

Z8 Support Products
Zilog Quality and Reliability Report
Literature List
Package Information
Ordering Information

L

Z8®/SUPER8™ MICROCONTROLLER FAMILY (Continued)

Databooks By Market Niche	Part No	Unit Cost
Digital Signal Processor Databook (includes the following documents) Z86C95 Z8® Digital Signal Processor Preliminary Product Specification Z89C00 16-Bit Digital Signal Processor Preliminary Product Specification Z89C00 DSP Application Note "Understanding Q15 Two's Complement Fractional Multiplication" Z89120, Z89920 (ROMless) 16-Bit Mixed Signal Processor Preliminary Product Specification Z89121, Z89921 (ROMless) 16-Bit Mixed Signal Processor Preliminary Product Specification Z89320 16-Bit Digital Signal Processor Preliminary Product Specification Z89321 16-Bit Digital Signal Processor Advance Information Specification	DC-8299-03	3.00
Telephone Answering Device Databook (includes the following documents) Z89C65, Z89C66 (ROMless) Dual Processor T.A.M. Controller Preliminary Product Specification Z89C67, Z89C68/C69 (ROMless) Dual Processor Tapeless T.A.M. Controller Preliminary Product Specification Z89C65 Software Development Guide Z89C67/C69 Software Development Guide	DC-8300-02	3.00
Infrared Remote (IR) Control Databook (includes the following documents) Z86L06 Low Voltage CMOS Consumer Controller Processor Preliminary Product Specification Z86L29 6K Infrared (IR) Remote (ZIRC™) Controller Advance Information Specification Z86L70/L71/L72, Z86E72 Zilog IR (ZIRC™) CCP™ Controller Family Preliminary Product Specification	DC-8301-03	3.00
Z8 Microcontrollers (includes the following documents) Z86C07 CMOS Z8 8-Bit Microcontroller Product Specification Z86C08 CMOS Z8 8-Bit Microcontroller Product Specification Z86E08 CMOS Z8 8-Bit OTP Microcontroller Product Specification Z86C11 CMOS Z8 Microcontroller Product Specification Z86C12 CMOS Z8 In-Circuit Microcontroller Emulator Product Specification Z86C21 8K ROM Z8 CMOS Microcontroller Product Specification Z86E21 CMOS Z8 8K OTP Microcontroller Product Specification Z86C61/62/96 CMOS Z8 Microcontroller Product Specification Z86C63/64 32K ROM Z8 CMOS Microcontroller Product Specification Z86C91 CMOS Z8 ROMless Microcontroller Product Specification Z86C93 CMOS Z8 Multiply/Divide Microcontroller Product Specification	DC-8305-02	3.00
Mass Storage (includes the following documents) Z86C21 8K ROM Z8 CMOS Microcontroller Product Specification Z86E21 CMOS Z8 8K OTP Microcontroller Product Specification Z86C91 CMOS Z8 ROMless Microcontroller Product Specification Z86C93 CMOS Z8 Multiply/Divide Microcontroller Product Specification Z86C95 Z8 Digital Signal Processor Product Specification Z89C00 16-Bit Digital Signal Processor Product Specification Z89C00 DSP Application Note - "Understanding Q15 Two's Complement Fractional Multiplication"	DC-8303-00	3.00

Z8®/SUPER8™ MICROCONTROLLER FAMILY (Continued)

Databooks By Market Niche	Part No	Unit Cost
Digital Television Controllers (includes the following documents) Z86C27/97 CMOS Z8® Digital Signal Processor Product Specification Z86C61/62/96 CMOS Z8 Microcontroller Product Specification Z86C63/64 32K ROM CMOS Z8 Microcontroller Product Specification Z86127 Low Cost Digital Television Controller Product Specification Z86128 Line 21 Closed-Caption Controller (L21C™) Digital Television Controller Product Specification Z86227 40-Pin Low Cost (4LDTC™) Digital Television Controller Product Specification	DC-8308-00	3.00
Keyboard/Mouse/Pointing Devices Databook (includes the following documents) Z8602 NMOS Z8® 8-Bit Keyboard Controller Product Specification Z8614 NMOS Z8® 8-Bit Keyboard Controller Product Specification Z8615 NMOS Z8® 8-Bit Keyboard Controller Product Specification Z86E23 Z8® 8-Bit Keyboard Controller with 8K OTP Product Specification Z86C04 CMOS Z8® 8-Bit Microcontroller Product Specification Z86C08 CMOS Z8® 8-Bit Microcontroller Product Specification Z88C17 CMOS Z8® 8-Bit Microcontroller Product Specification	DC-8304-00	3.00
PC Audio Databook (includes the following documents) Z86321 Digital Audio Processor Preliminary Product Specification Z89320 16-Bit Digital Signal Processor Preliminary Product Specification Z89321/371 16-Bit Digital Signal Processor Preliminary Product Specification Z89331 16-Bit PC ISA Bus Interface Advance Information Specification Z89341/42/43 Wave Synthesis Chip Set Advance Information Specification Z5380 Small Computer System Interface Product Specification	DC-8317-00	3.00
PCMCIA/SCSI Interface Controllers (includes the following documents) Z5380 Small Computer System Interface Product Specification Z53C80 Small Computer System Interface Product Specification Z85C80 SCSCI™ Serial Communications and Small Computer Interface Product Specification Z86017 PCMCIA Interface Preliminary Product Specification Z86015 PCMCIA Interface with DMA Support Advance Product Specification Z86020 CardBus/PCI Interface Advance Product Specification	DC 8313-00	3.00
Discrete Z8® Microcontrollers (Includes the following documents) Z86C04 CMOS Z8® 8-Bit Low Cost 1K ROM Microcontroller Product Specification Z86E04 OTP CMOS Z8® 8-Bit Microcontroller Product Specification Z86C08 CMOS Z8® 8-Bit Microcontroller Product Specification Z86E08 CMOS Z8® 8-Bit Microcontroller Product Specification Z86C30 CMOS Z8® 8-Bit Microcontroller Product Specification Z86E30 CMOS Z8® OTP CCP™ Microcontroller Product Specification Z86C31 CMOS Z8® 8-Bit Microcontroller Product Specification Z86E31 CMOS Z8® OTP CCP™ Microcontroller Product Specification	DC 8318-00	3.00

Z8®/SUPER8™ MICROCONTROLLER FAMILY (Continued)

Z8 Product Specifications, Technical Manuals and Users Guides	Part No	Unit Cost
Z8® Microcontrollers Technical Manual	DC-8291-02	5.00
Z86018 Preliminary User's Manual	DC-8296-00	N/C
Digital TV Controller User's Manual	DC-8284-01	3.00
Z89C00 16-Bit Digital Signal Processor User's Manual/DSP Software Manual	DC-8294-02	3.00
Z86C95 16-Bit Digital Signal Processor User Manual	DC-8595-00	3.00
Z86017 PCMCIA Adaptor Chip User's Manual	DC-8298-02	3.00
PLC Z89C00 Cross Development Tools Brochure	DC-5538-01	N/C

Z8 Application Notes	Part No	Unit Cost
The Z8 MCU Dual Analog Comparator	DC-2516-01	N/C
Z8 Applications for I/O Port Expansions	DC-2539-01	N/C
Z86E21 Z8 Low Cost Thermal Printer	DC-2541-01	N/C
Zilog Family On-Chip Oscillator Design	DC-2496-01	N/C
Using the Zilog Z86C06 SPI Bus	DC-2584-01	N/C
Interfacing LCDs to the Z8	DC-2592-01	N/C
X-10 Compatible Infrared (IR) Remote Control	DC-2591-01	N/C
Z86C17 In-Mouse Applications	DC-3001-01	N/C
Z86C40/E40 MCU Applications Evaluation Board	DC-2604-01	N/C
Z86C08/C17 Controls A Scrolling LED Message Display	DC-2605-01	N/C
Z86C95 Hard Disk Controller Flash EPROM Interface	DC-2639-01	N/C
Timekeeping with Z8; DTMF Tone Generation; Serial Communication Using the CCP Software UART	DC-2645-01	N/C

Z80®/Z8000® CLASSIC FAMILY OF PRODUCTS

Databooks By Market Niche	Part No	Unit Cost
High-Speed Serial Communication Controllers Z16C30 CMOS Universal Serial Controller (USC™) Preliminary Product Specification Z16C32 Integrated Universal Serial Controller (IUSC™) Preliminary Product Specification Application Notes and Support Products	**DC-8314-00**	**3.00**
Serial Communication Controllers Z8030/Z8530 Z-Bus® SCC Serial Communication Controller Product Specification Z80C30/Z85C30 CMOS Z-Bus® SCC Serial Communication Controller Product Specification Z80230 Z-Bus® ESCC™ Enhanced Serial Communication Controller Preliminary Product Specification Z85230 ESCC™ Enhanced Serial Communication Controller Product Specification Z85233 EMSCC™ Enhanced Mono Serial Communication Controller Product Specification Z85C80 SCSCI™ Serial Communications and Small Computer Interface Product Specification Z16C35/Z85C35 CMOS ISCC™ Integrated Serial Communications Controller Product Specification Application Notes and Support Products	**DC-8316-00**	**3.00**

Z80/Z180/Z280/Z380 Product Specifications, Technical Manuals and Users Guides	Part No	Unit Cost
Z80 Family Technical Manual	DC-8309-00	3.00
Z80180 Z180 MPU Microprocessor Unit Technical Manual	DC-8276-04	3.00
Z280 MPU Microprocessor Unit Technical Manual	DC-8224-03	3.00
Z80182 Zilog Intelligent Peripheral (ZIP™)	DC-2616-03	N/C
Z380™ Preliminary Product Specification	DC-6003-03	N/C
Z380™ User's Manual	DC-8297-02	3.00
Z80 Family Programmer's Reference Guide	DC-0012-04	N/C
ZNW2000 User's Manual for PC WAN Adaptor Board Development Kit	DC-8315-00	N/C

Z8000 Family of Products (Includes the following) Z8000 Family Databook Zilog's Z8000 Family Architecture Z8001/Z8002 Z8000 CPU Product Specification Z8016 Z8000 Z-DTC Product Specification Z8036 Z8000 Z-CIO Product Specification Z8536 CIO Counter/Timer and Parallel I/O Unit Product Specification Z8038/Z8538 FIO FIFO Input/Output Interface Unit Product Specification Z8060/Z8560 FIFO Buffer Unit Z8581 Clock Generator and Controller Product Specification Z8000 CPU Central Processing Unit User's Manual Z8010 Memory Management Unit (MMU) User's Manual Z8036 Z-CIO/Z8536 CIO Counter/Timer and Parallel Input/Output User's Manual Z8038 Z8000 Z-FIO FIFO Input/Output Interface User's Manual Z8000 Application Notes and Military Products Zilog's Superintegration™ Products Guide Literature Guide and Ordering Information Zilog's Sales Offices, Representatives and Distributors	**DC-8319-00**	**5.00**

Z80®/Z8000® CLASSIC FAMILY OF PRODUCTS (Continued)

Z8000 User's Manuals	Part No	Unit Cost
SCC Serial Communication Controller User's Manual	DC-8293-02	3.00
High-Speed SCC, Z16C30 USC/Z16C32 IUSC, User's Manual	DC-8320-00	5.00
Z16C35 ISCC Integrated Serial Communication Controller Technical Manual	DC-8286-01	3.00
Z16C35 ISCC Integrated Serial Communication Controller Addendum	DC-8286-01A	N/C

Z80/Z180/Z280 Application Notes	Part No	Unit Cost
Z180/SCC™ Serial Communications Controller Interface at 10 MHz	DC-2521-02	N/C
Z80 Using the 84C11/C13/C15 in place of the 84011/013/015	DC-2499-02	N/C
A Fast Z80 Embedded Controller	DC-2578-01	N/C

Z8000 Application Notes	Part No	Unit Cost
Z16C30 Using the USC in Military Applications	DC-2536-01	N/C
Datacom IUSC/MUSC Time Slot Assigner	DC-2497-02	N/C
Datacom Evaluation Board Using The Zilog Family With The 80186 CPU	DC-2560-03	N/C
Boost Your System Performance Using the Zilog ESCC Controller	DC-2555-02	N/C
Z16C30 USC - Design a Serial Board for Multiple Protocols	DC-2554-01	N/C
Using a SCSI Port for Generalized I/O	DC-2608-01	N/C

MILITARY COMPONENTS FAMILY

Military Specifications	Part No	Unit Cost
Z8681 ROMless Microcomputer Military Product Specification	DC-2392-02	N/C
Z8001/8002 Military Z8000 CPU Central Processing Unit Military Product Specification	DC-2342-03	N/C
Z8581 Military CGC Clock Generator and Controller Military Product Specification	DC-2346-01	N/C
Z8030 Military Z8000 Z-SCC Serial Communications Controller Military Product Specification	DC-2388-02	N/C
Z8530 Military SCC Serial Communications Controller Military Product Specification	DC-2397-02	N/C
Z8036 Military Z8000 Z-CIO Counter/Timer Controller and Parallel I/O Military Electrical Specification	DC-2389-01	N/C
Z8038/8538 Military FIO FIFO Input/Output Interface Unit Military Product Specification	DC-2463-02	N/C
Z8536 Military CIO Counter/Timer Controller and Parallel I/O Military Electrical Specification	DC-2396-01	N/C
Z8400 Military Z80 CPU Central Processing Unit Military Electrical Specification	DC-2351-02	N/C
Z8420 Military PIO Parallel Input/Output Controller Military Product Specification	DC-2384-02	N/C
Z8430 Military CTC Counter/Timer Circuit Military Electrical Specification	DC-2385-01	N/C
Z8440/1/2/4 Z80 SIO Serial Input/Output Controller Military Product Specification	DC-2386-02	N/C
Z80C30/85C30 Military CMOS SCC Serial Communications Controller Military Product Specification	DC-2478-02	N/C
Z84C00 CMOS Z80 CPU Central Processing Unit Military Product Specification	DC-2441-02	N/C
Z84C20 CMOS Z80 PIO Parallel Input/Output Military Product Specification	DC-2384-02	N/C
Z84C30 CMOS Z80 CTC Counter/Timer Circuit Military Product Specification	DC-2481-01	N/C
Z84C40/1/2/4 CMOS Z80 SIO Serial Input/Output Military Product Specification	DC-2482-01	N/C
Z16C30 CMOS USC Universal Serial Controller Military Preliminary Product Specification	DC-2531-01	N/C
Z80180 Z180 MPU Microprocessor Unit Military Product Specification	DC-2538-01	N/C
Z84C90 CMOS KIO Serial/Parallel/Counter Timer Preliminary Military Product Specification	DC-2502-00	N/C
Z85230 ESCC Enhanced Serial Communication Controller Military Product Specification	DC-2595-00	N/C

GENERAL LITERATURE

Catalogs, Handbooks, Product Flyers and Users Guides	Part No	Unit Cost
Superintegration Master Selection Guide 1994-1995	DC-5634-00	N/C
Superintegration Products Guide	DC-5499-07	N/C
Quality and Reliability Report	DC-2475-12	N/C
ZIA™ 3.3-5.5V Matched Chip Set for AT Hard Disk Drives Datasheet	DC-5556-01	N/C
ZIA ZIA00ZC0 Disk Drive Development Kit Datasheet	DC-5593-01	N/C
Zilog Hard Disk Controllers - Z86C93/C95 Datasheet	DC-5560-01	N/C
Zilog Infrared (IR) Controllers - ZIRC™ Datasheet	DC-5558-01	N/C
Zilog Intelligent Peripheral Controller - ZIP™ Z80182 Datasheet	DC-5525-01	N/C
Zilog Digital Signal Processing - Z89320 Datasheet	DC-5547-01	N/C
Zilog Keyboard Controllers Datasheet	DC-5600-01	N/C
Z380™ - Next Generation Z80®/Z180™ Datasheet	DC-5580-02	N/C
Fault Tolerant Z8® Microcontroller Datasheet	DC-5603-01	N/C
32K ROM Z8® Microcontrollers Datasheet	DC-5601-01	N/C
Zilog Datacommunications Brochure	DC-5519-00	N/C
Z89300 DTC Controller Family Brochure	DC-5608-01	N/C
Zilog Digital Signal Processing Brochure	DC-5536-02	N/C
Zilog ASSPs - Partnering With You Product Brochure	DC-5553-01	N/C
Zilog Wireless Products Datasheet	DC-5630-00	N/C
Zilog Z8604 Cost Cruncher Datasheet	DC-5623-00	N/C
Zilog Databast of IR Codes Datasheet	DC-5631-00	N/C
Zilog PCMCIA Adaptor Chip Z86017 Datasheet	DC-5585-01	N/C
Zilog Television/Video Controllers Datasheet	DC-5567-01	N/C
Zilog TAD Controllers - Z89C65/C67/C69 Datasheet	DC-5561-02	N/C
Zilog Z87000 Z-Phone Datasheet	DC-5632-00	D/C
Zilog 1991 Annual Report	DC-1991-AR	N/C
Zilog 1992 Annual Report	DC-1992-AR	N/C
Zilog 1993 Second Quarter Financial Report	DC-1993-Q2	N/C
Zilog 1993 Third Quarter Financial Report	DC-1993-Q3	N/C
Microcontroller Quick Reference Folder	DC-5508-01	N/C

ZiLOG LITERATURE GUIDE

ORDERING INFORMATION

Complete the attached literature order form. Be sure to enclose the proper payment or supply a purchase order. Please reference specific order requirements.

MINIMUM ORDER REQUIREMENTS

Orders under $300.00 must be prepaid by check, money order or credit card. Canadian and foreign orders must be accompanied by a cashier's check in U.S. dollars, drawn on a correspondent U.S. bank only.
Orders over $300.00 may be submitted with a Purchase Order.

SHIPMENT

Orders will be shipped after your check is cashed or credit is checked via the most economical method. Please allow four weeks for delivery.

RETURNS ARE NOT ACCEPTED.

PLEASE PRINT OR TYPE

NAME			PHONE ( ) -
COMPANY			**Method of Payment (Check One)** ☐ Check ☐ Money Order
ADDRESS			Credit Card ☐ VISA ☐ M/C ☐ P.O. (over $300.00)
CITY	STATE	ZIP	COUNTRY

PART NUMBER	DOCUMENT TITLE	UNIT COST	QTY.	TOTAL
– –		$		$
– –		$		$
– –		$		$
– –		$		$
– –		$		$
– –		$		$
– –		$		$
– –		$		$
– –		$		$
– –		$		$
– –		$		$
– –		$		$
– –		$		$
– –		$		$
– –		$		$

Mail To:

ZiLOG
210 E. HACIENDA AVE. M/S C1-0
CAMPBELL, CA 95008-6600

Phone: (408)370-8016
Fax: (408)370-8056

Credit Card or Purchase Order # _____ SUBTOTAL

Expiration Date _____ ADD APPLICABLE SALES TAX (CA ONLY)

Signature _____ ADD 10% SHIPPING AND HANDLING

TOTAL

ZILOG DOMESTIC SALES OFFICES AND TECHNICAL CENTERS

CALIFORNIA
Agoura 818-707-2160
Campbell 408-370-8120
Irvine 714-453-9701
San Diego 619-658-0391

COLORADO
Boulder 303-494-2905

FLORIDA
Clearwater 813-725-8400

GEORGIA
Duluth 404-931-4022

ILLINOIS
Schaumburg 708-517-8080

MINNESOTA
Minneapolis 612-944-0737

NEW HAMPSHIRE
Nashua 603-888-8590

OHIO
Independence 216-447-1480

OREGON
Portland 503-274-6250

PENNSYLVANIA
Horsham 215-784-0805

TEXAS
Austin 512-343-8976
Dallas 214-987-9987

INTERNATIONAL SALES OFFICES

CANADA
Toronto 905-850-2377

CHINA
Shenzhen 86-755-2236089

GERMANY
Munich 49-8967-2045
Sömmerda 49-3634-23906

JAPAN
Tokyo 81-3-3587-0528

HONG KONG
Kowloon 852-7238979

KOREA
Seoul 82-2-577-3272

SINGAPORE
Singapore 65-2357155

TAIWAN
Taipei 886-2-741-3125

UNITED KINGDOM
Maidenhead 44-628-392-00

Zilog, Inc. 210 East Hacienda Ave.
Campbell, CA 95008-6600
Telephone (408) 370-8000
Telex 910-338-7621
FAX 408 370-8056

Q2/94 DC 8309-00

Zilog, Inc.
210 East Hacienda Ave.
Campbell, CA 95008-6600
408-370-8000