Am29332
Instruction Manual
AMD

Am29332
Instruction Manual

Advanced
Micro
Devices

Advanced Micro Devices

Am29332
32-Bit Arithmetic Logic Unit

Instruction Manual

09287B

Table of Contents

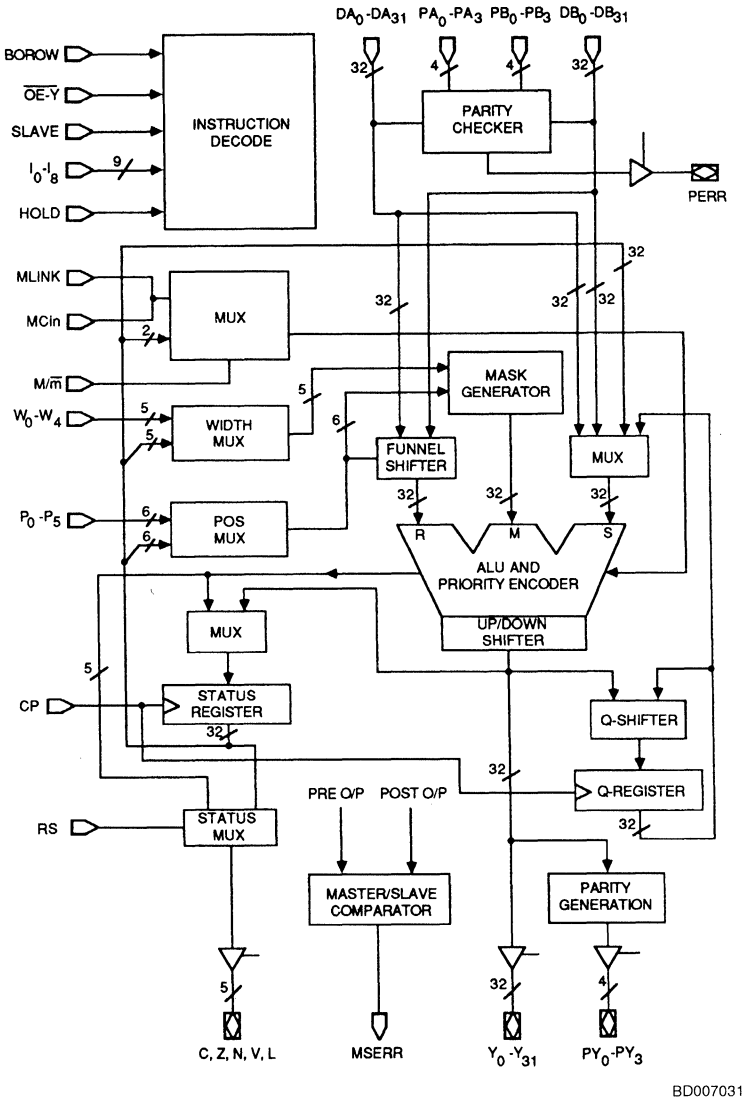


Figure 1. Detailed Block Diagram

Am29332 32-Bit Arithmetic Logic Unit

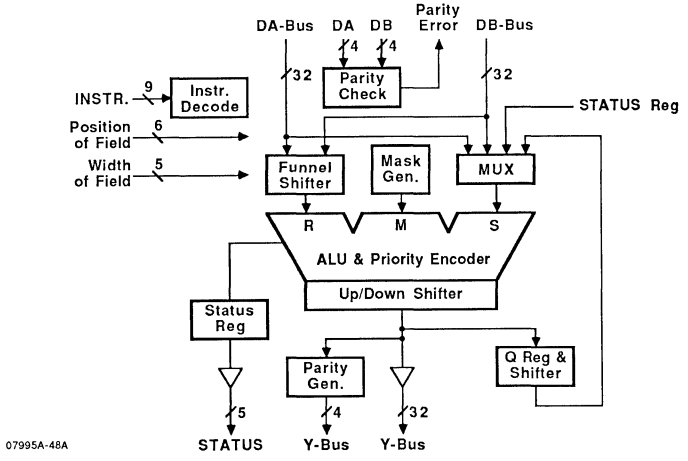


Figure 2. Simplified Block Diagram

PRODUCT OVERVIEW

The Am29332 is a 32-bit wide, high-performance, non-expandable Arithmetic Logic Unit (ALU). It has two 32-bit wide input ports (A and B) and one 32-bit wide output port (Y). These three ports provide flexibility and accessibility for high-performance processor designs. Dedicated input and output ports provide a flow-through architecture and avoid the penalty associated with switching the bus half-way through the cycle for input and output of data. The chip is designed for use with a dual-access RAM (Am29334) as a register file. In addition, the three-bus architecture facilitates the connection of other arithmetic units in parallel with the Am29332 for high-performance systems.

The Am29332 supports one-, two-, three-, and four-byte arithmetic operations. It also supports multiprecision arithmetic and multiple bit shifts. For logical operations, it can handle variable-length fields of up to 32 bits. The chip incorporates dedicated hardware to allow efficient implementation of a two bit-at-a-time (modified Booth) multiply algorithm, supporting signed and unsigned arithmetic data types. Similarly, hardware is provided to support a bit-at-a-time divide algorithm, also supporting signed and unsigned arithmetic data types. An internal 32-bit register (Q) is used by the multiply and divide hardware for double precision operands. For business applications, the Am29332 supports variable-length BCD arithmetic.

Field logical instructions operate on bit-fields taken from the A and B data inputs; they may be of variable width and starting position. A is normally the source input and B the destination input. In general, destination bits not falling within a specified field are passed by the ALU unchanged. Field width and position are specified either by direct inputs to the chip, or by entries in the status register. There are two kinds of field logical instructions – aligned and non-aligned. The first type of instruction assumes that source and destination fields are aligned and the operation is performed only for bits within the specified fields. In the second type of instruction, source and destination fields are normally non-aligned. However, it is always assumed that one field (either source or destination) is least-significant-bit (LSB) aligned.

If the destination field is LSB aligned then the source field is downshifted in order to make it LSB aligned as well. Downshifting is accomplished by making the 6-bit position input equal to the two's complement of the number of places the field is to be downshifted. If the source field is LSB aligned then it is upshifted in order to align it with the destination. Upshifting is accomplished by making the position inputs equal to the number of places the field is to be upshifted. Any other type of field operation is not allowed. Whenever the field crosses the word boundary, the portion not falling within the word boundary is ignored. This effect is useful when performing operations on fields that overlap two different words. Instructions to perform straightforward multiple-bit shifts (either up or down) are also provided. Additionally, it is possible to extract a bit-field from a word in one instruction, even if that field overlaps a word boundary.

The power and the flexibility of the processor comes partly from its ability to generate a mask to control the width of an operation for each instruction without any overhead. For all byte aligned instructions (three quarters of the instruction set), the mask is either 1, 2, 3 or 4 bytes wide and is generated from the byte width input ($I_8 - I_7$). For all field instructions the mask is of variable width and is generated from the position inputs ($P_0 - P_5$) and the width inputs ($W_0 - W_4$). Table 1 describes the position displacement from the position inputs and Table 2 the bit field from the width inputs.

TABLE 1. POSITION INPUTS AND BIT DISPLACEMENT

Inputs						Bit Displacement
P_5	P_4	P_3	P_2	P_1	P_0	p
0	0	0	0	0	0	0
0	0	0	0	0	1	1
0	0	0	0	1	0	2
⋮	⋮	⋮	⋮	⋮	⋮	⋮
0	1	1	1	1	1	31
1	0	0	0	0	0	-32
1	0	0	0	0	1	-31
⋮	⋮	⋮	⋮	⋮	⋮	⋮
1	1	1	1	1	1	-1

TABLE 2. WIDTH INPUTS AND BIT FIELD

Inputs					Bit Field
W_4	W_3	W_2	W_1	W_0	w
0	0	0	0	0	32
0	0	0	0	1	1
0	0	0	1	0	2
⋮	⋮	⋮	⋮	⋮	⋮
1	1	1	1	1	31

Whenever the width of the operand is less than 32-bits, all unselected bits from the inputs of the ALU are passed to the output without any modification. Depending upon the instruction type, unselected bits are taken from different sources. For example in all single operand instructions, bits from the source operand (from either A or B input) are passed in unselected bit positions. For two operand instructions, bits from the B input are passed in unselected bit positions. There are some exceptions which are explained in the instruction set section.

The processor has a 32-bit status register to indicate the status of different operations performed. The status register is loaded at the rising edge of the clock with new status unless the HOLD signal is HIGH. The bit position for each status bit is given in the functional description. The least significant byte of the status register holds the six position bits ($PR_0 - PR_5$). The two most significant bits of this byte may be read or loaded but are otherwise unused by the ALU. The second byte (bits 8 to 15) consists of the five width bits ($WR_0 - WR_4$) and three read-only bits that are a combinational function of other status bits, and which indicate useful branch conditions. The third byte consists of ALU status bits plus bits for high-speed multiply and divide. The most significant byte holds intermediate nibble carries for BCD operations. An extract-status instruction is provided which allows a Boolean value to be formed from any selected bit. This is particularly useful in machines employing a stack architecture. Instructions to save and restore the status register are provided. As the entire status of each instruction is stored in the status register, interrupts at any microinstruction boundary are feasible.

The processor has a 32-bit wide priority encoder to support floating-point and graphics operations. The priority encoder supports all byte aligned data types – the result is dependent upon the byte width specified. The result of a priority encode is also loaded into the position bits of the status register. The result of the prioritize operation can then be used in the following clock cycle, e.g., to normalize a floating-point number or to help detect the edge of a polygon in graphics applications.

To support system diagnostics, the Am29332 has a special "Master-Slave" mode. To use this mode, two chips are connected in parallel, and hence receive the same instructions and data. The master chip is used for the normal data path. However, in the slave chip, all outputs becomes inputs. The slave compares the outputs of the master with its own internally generated result. If the two do not match, the slave will activate an error signal.

As a further diagnostic aid, byte-wise parity checking is performed at both the A and B data inputs. The "parity" signal is activated if an error is detected. Parity bits (one per byte) are generated for the 32-bit output bus.

FUNCTIONAL DESCRIPTION

A detailed description of each functional block is given in the following paragraphs.

64-Bit Funnel Shifter

The 64-bit funnel shifter is a combinatorial network. The 64-bit input is formed from a combination of the A and B inputs. This may be left-shifted by up to 31 bits before being used by the ALU. The output of the shifter is the most significant 32 bits of the result. The 64-bit shifter can be used on either the A or B operands to perform barrel shifts (either up or down) or rotates. The operation is controlled by positioning operands properly at the input of the 64-bit up-shifter.

The number "n" by which the operand is shifted comes from two sources: the microprogram memory via the $P_0 - P_5$ pins or the internal register (byte 0 of the status register), $PR_0 - PR_5$, as selected by an instruction bit.

In general, the 6-bit position input, $P_0 - P_5$, takes a 6-bit two's complement number representing upshifts from 0 to 31 places (positive numbers) or downshifts from 1 to 32 places (negative numbers).

Mask Generator

The mask generator logic provides the ability to generate the appropriate mask for an operand of given width and position. The generation of the mask depends upon two types of instructions. The first type has byte boundary aligned operands (widths of either 1, 2, 3 or 4 bytes) with the least significant bit aligned to bit 0. The width of an operand is specified by the byte width inputs (I_8 and I_7) as shown in Table 3. The second type of instruction has operands of variable width (1 to 32 bits) and position. The operand is specified by the width inputs ($W_0 - W_4$) and the position inputs ($P_0 - P_5$)

indicating the least significant bit position of the operand. Thus, in this type of instruction the operand may or may not be least significant bit aligned. Depending upon the type of instruction, the mask generator first generates a fence of all zeros starting from the least significant bit with the width specified either by the byte width or the width input fields. This fence can be upshifted by up to 31 bits by the 32-bit mask shifter. Whenever the mask is moved up over the 32-bit boundary, it does not wrap around. Instead, ONE's are inserted from the least significant end. This configuration provides the ability to operate on a contiguous field located anywhere in a word, or across a word boundary.

The mask generator can be used as a pattern generator by allowing the mask to pass through ALU (by using the PASS-MASK instruction). For example, a single-bit wide mask can be generated and by shifting it up by different amounts can give walking ONE or walking ZERO patterns for memory tests.

TABLE 3.

I_8	I_7	Width in Bytes
0	0	4
0	1	1
1	0	2
1	1	3

Arithmetic and Logical Unit

The ALU is a three input unit which uses the mask as a second or third operand in every instruction. The mask is used to merge two operands. For all selected bits (wherever the mask is 0), the desired operation specified by the instruction input is performed, and for all unselected bits either corresponding destination bits or zeros are passed through. The status of each operation (carry, negative, zero, overflow, link) applies to the result only over the specified width. For all byte aligned arithmetic and logical operations (first three quarters of the instruction set), the status is extracted from the appropriate byte boundary. For all field operations (last quarter of the instruction set), the operand width is assumed to be 32 bits for status generation. The ZERO flag always indicates the status of all bits selected by the mask.

The actual width of the ALU is 34 bits. There are two extra bits used for the high speed signed and unsigned multiplication instructions. These two bits are automatically concatenated to the most-significant end of the ALU depending upon the width specified for the operation. Since the modified Booth algorithm requires a two-bit down-shift each cycle, these ALU bits generate the two most-significant bits of the partial product.

The ALU is capable of shifting data down by two bits for the multiplication algorithm, up by one bit for the divide algorithm and single-bit-up-shifts.

The processor is capable of performing BCD arithmetic on packed BCD operations. This logic generates nibble carries (BCD digit carry) from propagate and generates signals formed from the A and B operands. In order to simplify the hardware while maintaining throughput, the BCD add and subtract operations are performed in two cycles. In the first cycle, ordinary binary addition or subtraction is performed and BCD nibble carries are generated. These are blocked from affecting the result at this stage, but are saved in the status register to be used later for BCD correction (NC_0-NC_7). In the second cycle all BCD numbers are adjusted by examining the previously generated nibble carries. Since all the necessary information is stored in the status register, the processor can be interrupted after the first BCD cycle.

Priority Encoder

The priority encoder is provided to support floating-point arithmetic and some graphics primitives. The priority encoder takes up to 32-bits as input and generates a 5-bit wide binary code to indicate location of the most significant one in the operand. Input to the priority endcoder comes from the input multiplexer, which masks all bits that the user does not want to participate in the prioritization. The priority encoder supports 8, 16, 24, and 32-bit operations depending upon the byte width specified. For each data type the priority encoder generates the appropriate binary weighted code. For example, when a byte width of two is specified (I_8 - I_7 = 10), the output of the encoder is zero when bit 15 is HIGH. However, if byte width of four is specified (I_8 - I_7 = 00), the output of the encoder is 16 (decimal) if bit 15 is HIGH and bits 31 - 16 are LOW. Table 4 shows the output for each data type. If none of the inputs are HIGH or the most significant bit of the data type specified is HIGH, then the output is zero. The difference between these two cases is indicated by the Z-flag of the status register which is HIGH only if all inputs are zero.

Q-Register

The Q-register holds dividend and quotient bits for division, and multiplier and product bits for multiplication. During division, the contents of the Q-register are shifted left, a bit at a time, with quotient bits inserted into bit 0. During multiplication, the contents of the Q-register are shifted right, two bits at a time, with product bits inserted into the most-significant two bits (according to the selected byte width). The Q-register may be loaded from the A or B inputs and read onto the Y bus.

Master-Slave Comparator

All ALU outputs (except MSERR) employ three-state buffers. The master-slave comparator compares the input and output of each buffer. Any difference causes the MSERR signal to be made true. In Slave mode, all output buffers are disabled. Outputs from a second ALU may then be connected to the equivalent pins of the first. The comparator in the slave will then detect any difference in the results generated by the two. When the Y bus is three-stated by making Output-Enable false, the Y bus master-slave comparators are disabled.

Parity Logic

For each byte of the DA and DB inputs there is an associated parity bit (8 in all). If a parity error is detected on any byte, the Parity-Error signal is made true. Four parity signals (one per byte) are also generated for the Y bus outputs. EVEN parity is employed for the Am29332.

Status Register

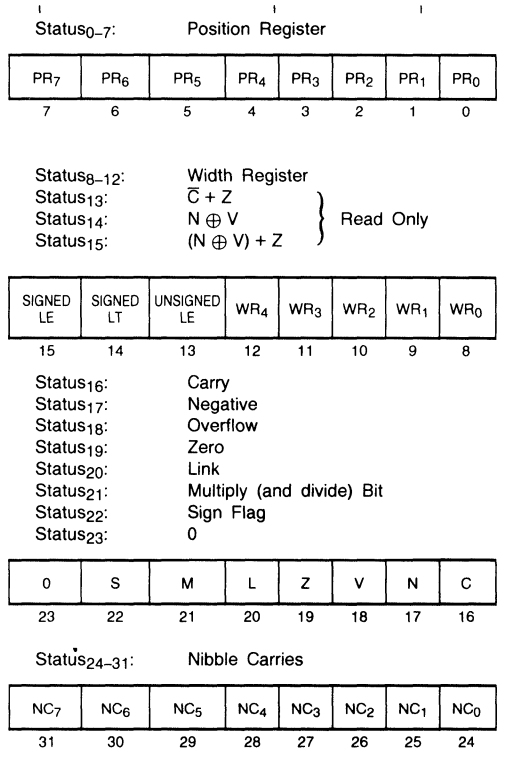
All necessary information about operations performed in the ALU is stored in the 32-bit wide status register after every microcycle. Since the register can be saved, an interrupt can occur after any cycle. The status register can be loaded from either the A or B input of the chip and can be read out on the Y bus for saving in an external register file. For loading, the byte width indicates how many bytes are to be updated. The status register is only updated if the HOLD input is inactive.

Each byte of the status register holds different types of information (see Figure 3). The least significant byte (bits 0 to 7) holds eight position bits ($PR_0 - PR_7$) for the data shifter.

The two most significant bits are not used. The next most significant byte (bits 8 to 15) holds the 5-bit width field ($WR_0 - WR_4$) for the mask generator. The three most-significant bits of that byte (bits 13 to 15) are read-only bits that represent three different conditions extracted from the other bits of the status register. They are $\overline{C} + Z$, $N \oplus V$, and $(N \oplus V) + Z$ for bits 13, 14 and 15 respectively. These bits can be read on the Y_0 pin by the extract-status instruction. The next byte contains all the necessary information generated by an ALU operation. The least-significant four bits (bits 16 to 19) hold carry, negative, overflow and zero flags. Bit 20 holds link information for single bit shifts and bits 21 and 22 are used by the multiply and divide instructions. The M flag holds the multiplier bit for the modified Booth algorithm or it holds the sign comparison result for the divide algorithm. The S flag holds the sign of the partial remainder for unsigned division. Both the flags (M and S) are provided as a part of the status register so that multiply and divide instructions can be interrupted at microinstruction boundaries. The most significant byte of the status register holds nibble carries for BCD arithmetic. Since BCD arithmetic is performed in two cycles, the nibble carries are saved in the first cycle and used in the second cycle. Since all the information is stored, BCD instructions are also interruptible at the microinstruction boundary.

TABLE 4.

Highest Priority Active Bit	Encoder Output
$I_7 - I_8 = 00$ (32-bit)	
None	0
31	0
30	1
29	2
28	3
.	.
.	.
.	.
1	30
0	31
$I_7 - I_8 = 01$ (8-bit)	
None	0
7	0
6	1
5	2
.	.
.	.
.	.
1	6
0	7
$I_7 - I_8 = 10$ (16-bit)	
None	0
15	0
14	1
13	2
12	3
.	.
.	.
.	.
1	14
0	15
$I_7 - I_8 = 11$ (24-bit)	
None	0
23	0
22	1
21	2
20	3
.	.
.	.
.	.
1	22
0	23



Note: Overflow is defined as follows:
V = (carry in to MSB) ⊕ (carry out of MSB)

Figure 3. ALU Status Register Bit Assignment

Am29332 INSTRUCTION SET

Data Types

The Am29332 supports the following data types:

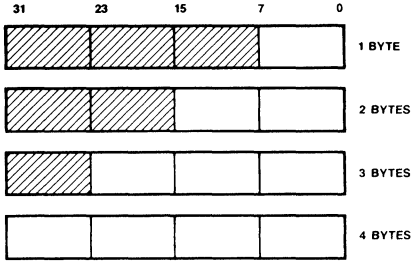
1. Integer
2. Binary-coded decimal
3. Variable-length bit field

The first two data types fall into the category of byte boundary aligned operands (Figure 4). The size of the operand could be 1 byte, 2 bytes, 3 bytes or 4 bytes. All operands are least significant bit (bit 0) aligned. The byte width is determined by bits I_8 and I_7 of the instruction as shown in Table 5.

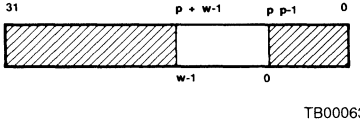
TABLE 5.

I_8	I_7	Width in Bytes
0	0	4
0	1	1
1	0	2
1	1	3

The third data type has operands of variable width (1 to 32 bits) as shown in Figure 4. The operand is specified by width inputs ($W_0 - W_4$) and position inputs ($P_0 - P_5$). The position inputs indicate the least significant bit position of the operand. Depending on bits I_8 and I_7 of the instruction, the width and position inputs can be selected from either the Status Register or the Width and Position Pins as shown in Table 6.



Byte Boundary Aligned Operands



Variable-Length Bit Field

p = Bit displacement of the least significant field with respect to bit 0.
w = Width of bit field.

Figure 4. Data Types

TABLE 6.

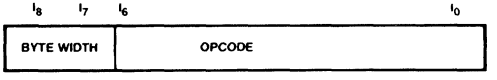
I_8	I_7	Position		Width	
		Pins	Reg	Pins	Reg
0	0	X		X	
0	1	X			X
1	0		X	X	
1	1		X		X

Data Type	Size	Range	
Integer		Signed	Unsigned
1 byte	8 bits	-128 to $+127$	0 to 255
2 bytes	16 bits	-2^{15} to $+2^{15}-1$	0 to $2^{16}-1$
3 bytes	24 bits	-2^{23} to $2^{23}-1$	0 to $2^{24}-1$
4 bytes	32 bits	-2^{31} to $2^{31}-1$	0 to $2^{32}-1$
BCD	1 to 4 bytes (8 digits)	Numeric, 2 digits per byte. Most-significant digit may be used for sign.	
Variable	1 to 32 bits	Dependent on position and width inputs.	

Instruction Format

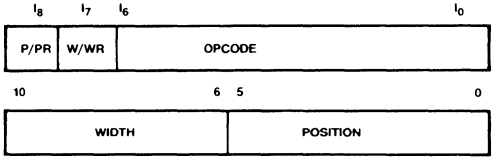
The Am29332 has two types of Instruction Formats:

1. Byte Boundary Aligned Instructions (FORMAT 1):



TB000098

2. Variable-Length Field Bit Instructions (FORMAT 2):



TB000099

For instructions that allow a field to be shifted up or down, $P_0 - P_5$ is a two's-complement number in the range -32 to $+31$ representing the direction and magnitude of the shift. For instructions that assume a fixed field position, $P_0 - P_4$ represent the position of the least-significant bit of the field and P_5 is ignored.

Instruction Classification

ALU instructions can be classified as follows:

A. Byte Boundary Aligned Operand Instructions:

1. Arithmetic
   - Binary, BCD
   - Multiply steps
   - Division steps (single and multiple precision)
2. Prioritize
3. Logical
4. Single-bit shifts
5. Data movement

B. Variable-Length Bit Field Operand Instructions:

1. N-bit shifts and rotates
2. Bit manipulations
3. Field logical operations (aligned, non-aligned, extract)
4. Mask generation

Three-fourths of the ALU instructions apply to operands that are byte boundary aligned. For these instructions, two orthogonal issues are the width of the operand (in bytes) and the contents of the high order unselected bytes on the Y bus. As mentioned earlier, the width of the operand is specified by I_8 and I_7. With the exception of a few instructions, the unselected bytes are assigned values as follows: for single operand instructions, unselected bytes are passed unchanged from the source (A or B). For two operand instructions, unselected bytes are passed unchanged from the destination (B input).

In the last quarter of the instruction set, the width of the operand is from 1 to 32 bits (based on the width input) for field operations, 32 bits for N-bit shift operations and 1-bit for bit-oriented operations. In the case of field-aligned and single-bit operands, the position bits ($P_0 - P_4$) determine the least significant bit of the operand. In the case of N-bit shifts and field non-aligned operands, the position bits $P_0 - P_5$ is a 6-bit signed integer determining the magnitude and direction of the shift.

Flags

Byte-Aligned Instructions

The zero flag always looks only at the selected bytes:

Z ← (Y and bytemask (byte width) = 0)

Similarly, N ← sign bit (Y, byte width), where the function "sign-bit" returns bit 7, 15, 23, or 31 of the first argument for byte widths 01, 10, 11, or 00 respectively.

Also, C ← carry (byte width) returns the carry from the appropriate byte boundary, and:

V ← overflow (byte width) = (carry into MSB) ⊕ (carry out of MSB)

returns the overflow from the appropriate byte boundary.

The link (L) flag is generally loaded with the bit moved out of the highest selected byte in the case of upshifts, or the bit moved out of the least significant byte for downshifts. Figure 5 shows the shift operation using link bit. Other status flags have specialized uses, explained in the following sections.

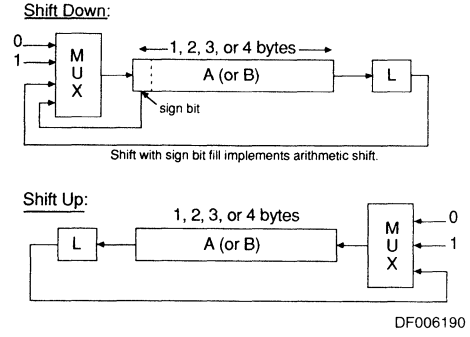


Figure 5. Upshift/Downshift Using Link Bit

Variable-Length Field Instruction:

Generally, only N and Z are affected. N takes the most-significant bit of the 32-bit result (i.e., N ← Y_{31}). Z detects zeros in the selected field of the result (i.e., Z ← (Y and bitmask (position, width) = 0)).

Output Select

The Register Status pin, RS, may be used to switch the C, Z, N, V, and L output pins between the direct output of the ALU and the outputs of the corresponding bits in the status register. If the direct status output is selected, then for instructions that do not affect a particular flag (e.g., carry for logical arithmetic) that output will reflect the state of its corresponding bit in the status register. Similarly, when the HOLD signal is made HIGH, the C, Z, N, V and L pins will be made equal to the contents of the status register, regardless of the RS input.

INSTRUCTION SET SUMMARY

Operand Size: Variable Byte Width: 1, 2, 3, 4 Bytes		
Type	**Operation**	**Data Type**
Arithmetic	• Increment by one, two, four • Decrement by one, two, four • Add, addc (carry = macro/micro) • Sub, subr • Subc, subrc (carry/borrow) • BCD sum and difference correct steps	Binary Integer and BCD
	• Negate (two's complement) • Multiply steps (modified Booth) • Divide steps (non-restoring) } (Signed and unsigned)	Binary Integer
Prioritize	• Prioritize	Binary
Logical	• Not, OR, AND, XOR, XNOR, zero, sign	Binary
Single-Bit Shifts	• Upshift with 0, 1, link fill • Downshift with 0, 1, link, sign fill } (Single and double precision)	Binary
Data Movement	• Zero extend • Sign extend • Pass-status, Q-Reg • Load-status, Q-Reg • Merge	Binary

Operand Size: 32 Bits		
Type	**Operation**	**Data Type**
N-Bit Shifts N-Bit Rotates	• Upshift by 0 to 31 bits with 0 fill • Downshift by 1 to 32 bits with 0, sign fill • Rotate by 0 to 31 bits	Binary

Operand Size: Single Bit		
Type	**Operation**	**Data Type**
Bit Manipulation	• Extract • Set • Reset	Binary

Operand Size: Variable Length Bitfield: 1 to 32 Bits		
Type	**Operation**	**Data Type**
Field Logical (aligned and non-aligned)	• Not, OR, XOR, AND, extract, insert	Binary
Mask	• Pass-mask	Binary

TABLE 6-1. DATA MOVEMENT INSTRUCTIONS

Mnemonics	Code	Description	Y Output		Status						
			Unsel	Sel	S	M	L	Z	V	N	C
ZERO-EXTA	00	Zero Extend	0	A				*		*	
ZERO-EXTB	01		0	B				*		*	
SIGN-EXTA	02	Sign Extend	Sign	A				*		*	
SIGN-EXTB	03		Sign	B				*		*	
MERGEA-B	0E	Merge A with B	B	A Merge B				*		*	
MERGEB-A	0F	Merge B with A	A	B Merge A				*		*	

TABLE 6-2. DATA MOVEMENT INSTRUCTIONS

Mnemonics	Code	Description	Y Output		Status Register	Status						
			Unsel	Sel		S	M	L	Z	V	N	C
PASS-STAT	04	Pass Status Register	B	S								
LDSTAT-A	1C	Load Status Register	S	A	A	+	+	+	+	+	+	+
LDSTAT-B	1D		S	B	B	+	+	+	+	+	+	+

TABLE 6-3. DATA MOVEMENT INSTRUCTIONS

Mnemonics	Code	Description	Y Output		Q Register	Status						
			Unsel	Sel		S	M	L	Z	V	N	C
PASS-Q	05	Pass Q Register	B	Q								
LOADQ-A	06	Load Q	Q	A	A				*		*	
LOADQ-B	07		Q	B	B				*		*	

Note: 1. These instructions use the byte aligned instruction format (FORMAT 1).

Legend: Unsel = Unselected Byte(s)
Sel = Selected Byte(s)
A = A Input
B = B Input
Q = Q Register
+ = Updated only if byte width is 3 or 4
* = Updated

Examples:

2, ZERO EXTB — Pass lower two bytes of B to Y with zero fill on upper two bytes

0, LOADQ-A — Load all four bytes of A into Q Register pass updated Q Resistor to Y

TABLE 7. LOGICAL INSTRUCTIONS

Mnemonics	Code	Description	Y Output		Status						
			Unsel	Sel	S	M	L	Z	V	N	C
NOT-A	08	One's Complement	A	$\overline{A}$				*		*	
NOT-B	09		B	$\overline{B}$				*		*	
ZERO	3C	Pass Zero	B	0				1		0	
SIGN	3D	Pass Sign	B	0(N = 0); -1(N = 1)				N			
OR	3E	OR	B	A OR B				*		*	
XOR	3F	EXOR	B	A XOR B				*		*	
AND	40	AND	B	A AND B				*		*	
XNOR	41	XNOR	B	A XNOR B				*		*	

Note: 1. These instructions use the byte aligned instruction format (FORMAT 1).

Legend: Unsel = Unselected Byte(s)
Sel = Selected Byte(s)
A = A Input
B = B Input
Q = Q Register
* = Updated

Examples:

2, NOT-A — Complement low order two bytes of A and output to Y with high order two bytes of A uncomplemented.

1, AND — AND first byte of A and B. Output to Y with high three bytes of B.

TABLE 8-1. SINGLE-BIT SHIFT INSTRUCTIONS (SINGLE PRECISION)

Mnemonics	Code	Description	Y Output		Status						
			Unsel	Sel	S	M	L	Z	V	N	C
DN1-0F-A	20	Downshift, Zero Fill	A	$Y_i = A_{i+1}$, $Y_{msb} = 0$			*	*		*	
DN1-0F-B	21		B	$Y_i = B_{i+1}$, $Y_{msb} = 0$			*	*		*	
DN1-1F-A	24	Downshift, One Fill	A	$Y_i = A_{i+1}$, $Y_{msb} = 1$			*	*		*	
DN1-1F-B	25		B	$Y_i = B_{i+1}$, $Y_{msb} = 1$			*	*		*	
DN1-LF-A	28	Downshift, Link Fill	A	$Y_i = A_{i+1}$, $Y_{msb} = L$			*	*		*	
DN1-LF-B	29		B	$Y_i = B_{i+1}$, $Y_{msb} = L$			*	*		*	
DN1-AR-A	2C	Downshift, Sign Fill	A	$Y_i = A_{i+1}$, $Y_{msb} = N$			*	*		*	
DN1-AR-B	2D		B	$Y_i = B_{i+1}$, $Y_{msb} = N$			*	*		*	
UP1-0F-A	30	Upshift, Zero Fill	A	$Y_i = A_{i-1}$, $Y_0 = 0$			*	*	*	*	
UP1-0F-B	31		B	$Y_i = B_{i-1}$, $Y_0 = 0$			*	*	*	*	
UP1-1F-A	34	Upshift, One Fill	A	$Y_i = A_{i-1}$, $Y_0 = 1$			*	*	*	*	
UP1-1F-B	35		B	$Y_i = B_{i-1}$, $Y_0 = 1$			*	*	*	*	
UP1-LF-A	38	Upshift, Link Fill	A	$Y_i = A_{i-1}$, $Y_0 = L$			*	*	*	*	
UP1-LF-B	39		B	$Y_i = B_{i-1}$, $Y_0 = L$			*	*	*	*	

Note: 1. These instructions use the byte aligned instruction format (FORMAT 1).

Example:

2, UP1-1F-A — Shift lower two bytes of A up one bit. Set LSB to 1. Fill unselected bytes to upper two bytes of A.

TABLE 8-2. SINGLE-BIT SHIFT INSTRUCTIONS (DOUBLE PRECISION)

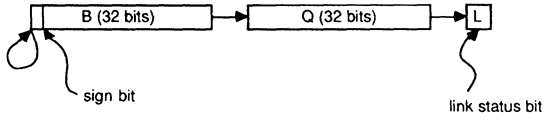
Mnemonics	Code	Description	Y Output & Q Register Selected Bytes	Status S	M	L	Z	V	N	C
DN1-0F-AQ	22	Downshift, Zero Fill	0 → A → Q 2)			*	*		*	
DN1-0F-BQ	23		0 → B → Q 3)			*	*		*	
DN1-1F-AQ	26	Downshift, One Fill	1 → A → Q 2)			*	*		*	
DN1-1F-BQ	27		1 → B → Q 3)			*	*		*	
DN1-LF-AQ	2A	Downshift, Link Fill	L → A → Q 2)			*	*		*	
DN1-LF-BQ	2B		L → B → Q 3)			*	*		*	
DN1-AR-AQ	2E	Downshift, Sign Fill	N → A → Q 2)			*	*		*	
DN1-AR-BQ	2F		N → B → Q 3)			*	*		*	
UP1-0F-AQ	32	Upshift, Zero Fill	A ← Q ← 0 2)			*	*	*	*	
UP1-0F-BQ	33		B ← Q ← 0 3)			*	*	*	*	
UP1-1F-AQ	36	Upshift, One Fill	A ← Q ← 1 2)			*	*	*	*	
UP1-1F-BQ	37		B ← Q ← 1 3)			*	*	*	*	
UP1-LF-AQ	3A	Upshift, Link Fill	A ← Q ← L 2)			*	*	*	*	
UP1-LF-BQ	3B		B ← Q ← L 3)			*	*	*	*	

Notes: 1. These instructions use the byte aligned instruction format (FORMAT 1).
2. Y Unselected byte from A, Q Unselected byte unchanged.
3. Y Unselected byte from B, Q Unselected byte unchanged.

Legend: Unsel = Unselected Byte(s)
Sel = Selected Byte(s)
A = A Input
B = B Input
Q = Q Register
* = Updated

Example:

0, DN1-AR-BQ — Shift 64 bits (all 32 bits of both B and Q) down by one bit. LSB of B fills MSB of Q. MSB of B set to sign bit (bit N of status register).



3, UP1-LF-AQ — Shift 48 bits (24-bits of A and 24-bits of Q) up by one bit. MSB of 24-bit Q fills LSB of A. MSB of 24-bit A sets link status bit. LSB of Q is filled with original link value.

DF006200

TABLE 9. PRIORITIZE INSTRUCTIONS

Mnemonics	Code	Description	Y Output	Status						
				S	M	L	Z	V	N	C
PRIOR-A	0C	Prioritization	Location of Highest 1 Bit				*			
PRIOR-B	0D						*			

Notes: 1. These instructions use the byte aligned instruction format (FORMAT 1).
2. Priority also loaded into STATUS <7:0>
3. Refer to Table 4.

Legend: A = A Input
B = B Input
Q = Q Register
* = Updated

Example:

3, PRIOR-A Value placed on Y is 2

Assume A is

01001011	00100010	00000000	00000000

TABLE 10-1. ARITHMETIC INSTRUCTIONS

Mnemonics	Code	Description	Y Output		Status						
			Unsel	Sel	S	M	L	Z	V	N	C
NEG-A	0A	Two's Complement	A	$\overline{A}$ + 1				*	*	*	*
NEG-B	0B		B	$\overline{B}$ + 1				*	*	*	*
INCR-A	12	Increment by One	A	A + 1				*	*	*	*
INCR-B	13		B	B + 1				*	*	*	*
INCR2-A	16	Increment by Two	A	A + 2				*	*	*	*
INCR2-B	17		B	B + 2				*	*	*	*
INCR4-A	1A	Increment by Four	A	A + 4				*	*	*	*
INCR4-B	1B		B	B + 4				*	*	*	*
DECR-A	10	Decrement by One	A	A – 1				*	*	*	*
DECR-B	11		B	B – 1				*	*	*	*
DECR2-A	14	Decrement by Two	A	A – 2				*	*	*	*
DECR2-B	15		B	B – 2				*	*	*	*
DECR4-A	18	Decrement by Four	A	A – 4				*	*	*	*
DECR4-B	19		B	B – 4				*	*	*	*

Notes: 1. These instructions use the byte aligned instruction format (FORMAT 1).
2. Borrow, rather than carry, is generated if BOROW is HIGH (borrow = carry).
3. Nibble bits are set by these instructions. NEG-A (or NEG-B) and DIFF-CORR may be used to form 10's complement of a BCD number. Use SUM-CORR (for increment) or DIFF-CORR (for decrement) to increment or decrement a BCD number.

Legend: Unsel = Unselected Byte(s)
Sel = Selected Byte(s)
A = A Input
B = B Input
Q = Q Register
* = Updated

Example:

2, DECR4-A Decrement lower two bytes of A by 4

TABLE 10-2. ARITHMETIC INSTRUCTIONS

Mnemonics	Code	Description	Y Output		Status						
			Unsel	Sel	S	M	L	Z	V	N	C
ADD	42	Add	B	A + B				*	*	*	*
ADDC	43	Add with Carry	B	A + B + C 6)				*	*	*	*
SUB	44	Subtract	B	A + $\overline{B}$ + 1				*	*	*	*
SUBR	46		B	B + $\overline{A}$ + 1				*	*	*	*
SUBC	45	Subtract with Carry	B	A + $\overline{B}$ + 1 + C 2) 6)				*	*	*	*
SUBRC	47		B	B + $\overline{A}$ + 1 + C 2) 6)				*	*	*	*
SUM-CORR-A	48	Correct BCD Nibbles for Addition	A	Corrected A 3)				*	*	*	*
SUM-CORR-B	49		B	Corrected B 3)				*	*	*	*
DIFF-CORR-A	4A	Correct BCD Nibbles for Subtraction	A	Corrected A 3)				*	*	*	*
DIFF-CORR-B	4B		B	Corrected B 3)				*	*	*	*

Notes: 1. These instructions use the byte aligned instruction format (FORMAT 1).
2. BOROW is LOW. For subtract operations, a borrow rather than a carry is stored in STATUS if BOROW is HIGH. Carry is always generated for ADD regardless of BOROW.
3. First, the nibble carries $NC_0 - NC_7$ are tested. Any nibble carry/borrow that is set to 1 generates "6" internally as a correction word and then the correction word is added (SUM-CORR-) or subtracted (DIFF-CORR-) from the operand. $NC_0 - NC_7$ are not affected by this operation.
4. Use SUM-CORR or DIFF-CORR to add or subtract a BCD number.
5. Use ADDC, SUBC, or SUBRC to perform operations on integers longer than 32 bits.
6. Carry bit is obtained from MCin if M/$\overline{m}$ is HIGH. Otherwise, carry is obtained from the C status bit.

Legend: Unsel = Unselected Byte(s)
Sel = Selected Byte(s)
A = A Input
B = B Input
Q = Q Register

* = Updated only if byte width is 3 or 4

Example:

0, ADD Add two 32-bit two's-complement integers

TABLE 11-1. DIVIDE INSTRUCTIONS (Aligned Format)

Name	$I_6 - I_0$ Code	Description	Source for Unselected Bytes	Output	Status						
					S	M	L	Z	V	N	C
Signed Divide Steps											
SDIVFIRST	4 E	First Instruction for Signed Divide	B	Y, Q	*	*	*	*		*	
SDIVSTEP	5 0	Iterate Step (#bits - 1 times)	B	Y, Q		*	*	*		*	*
SDIVLAST1	5 1	Last Divide Instruction Unless	B	Y, Q		*		*		*	*
SDIVLAST2	5 A	Dividend & Remainder Negative	B	Y				*			
Unsigned Divide Steps											
UDIVFIRST	4 F	First Instruction for Unsigned Divide	B	Y, Q			*	*		*	
UDIVSTEP	5 4	Iterate Step (#bits - 1 times)	B	Y, Q	*	*	*	*			*
UDIVLAST	5 5	Last Instruction	B	Y, Q	0	*		*		*	*
Multiprecision Divide Steps											
MPDIVSTEP1	5 2	First Instruction	B	Y, Q							
MPDIVSTEP2	5 6	Executed 0 Times for Double	B	Y, Q							
MPSDIVSTEP3	5 3	Last Instruction of Inner Loop	B	Y, Q							
MPUDIVSTP3	5 7	Used for Unsigned Divide	B	Y, Q							
Correction Steps											
REMCORR	5 8	Correct Remainder After Divide	B	Y							*
QUOCORR	5 9	Correct Quotient After Divide	B	Y					*		*

TABLE 11-2. EXAMPLE CODING FORM (Signed Division)

Am29331				Am29332				Am29334			Am29332 Y-Out
OP	Branch	Cond Select	Multi Sel	B/W	OP	Width	Position	A-IN	B-IN	Y-OUT	$\overline{OE}$
CONT				2	LOADQ-A			R2			1
CONT				0	SIGN					R3	0
FOR_D	15			2	SDIVFIRST			R4	R3	R3	0
DJMP_S				2	SDIVSTEP			R4	R3	R3	0
CONT				2	SDIVLAST1			R4	R3	R3	0
BRCC_D	DONE	Z									1
CONT				2	SDIVLAST2A			R4	R3	R3	0
CONT				2	PASS-Q					R1	0
CONT				2	QUOCORR				R1	R1	0
CONT				2	REMCORR			R4	R3	R3	0

Note: Divisor in A, Dividend in A
Quotient in Q, Remainder in B

Legend: A = A Input
B = B Input
S = Status Register
Q = Q Register
R1 = Quotient
R2 = Dividend
R3 = Remainder
R4 = Divisor

TABLE 12-1. MULTIPLY INSTRUCTIONS (Aligned Format)

Name	$I_6 - I_0$ Code	Description	Source for Unselected Bytes	Output	Status						
					S	M	L	Z	V	N	C
Signed Multiply Steps											
SMULFIRST	5 F	First multiply instruction	B	Y(1)							
SMULSTEP	5 E	Iterate step (#bits/2 - 1 steps)	B	Y(1)							
Unsigned Multiply Steps											
UMULFIRST	5 B	First multiply instruction	B	Y(1)		*					
UMULSTEP	5 C	Iterate step (#bits/2 - 1 steps)	B	Y(1)		*					
UMULLAST	5 D	Last multiply instruction	B	Y(1)				*			

TABLE 12-2. EXAMPLE CODING FORM (Unsigned Multiply)

Am29331				Am29332				Am29334			Am29332 Y-Out
OP	Branch	Cond Select	Multi Sel	B/W	OP	Width	Position	A-IN	B-IN	Y-OUT	$\overline{OE}$
CONT				3	ZERO				R3	R3	0
CONT				3	LOADQ-A			R1			1
FOR_D	11_{10}			3	ULMULFIRST			R2	R3	R3	0
DJMP_S				3	UMULSTEP			R2	R3	R3	0
CONT				3	UMULLAST			R2	R3	R3	0
CONT				3	PASS-Q					R4	0

Note: 1. Put ALU output in B.
2. Multiplicand in A, Multiplier in Q
Product (HIGH) in B, Product (LOW) in Q

Legend: A = A Input
B = B Input
S = Status Register
Q = Q Register
R1 = Multiplier
R2 = Multiplicand
R3 = Product (HIGH)
R4 = Product (LOW)

TABLE 13. SHIFT/ROTATE INSTRUCTIONS

Mnemonics	Code	Description	Y Output	Status S	M	L	Z	V	N	C
NB-0F-SHA	62	Field Shift, Zero Fill	$Y_{i+p} = A_i$, 0 2)				*		*	
NB-0F-SHB	63		$Y_{i+p} = B_i$, 0 2)				*		*	
NB-SN-SHA	60	Field Shift, Sign Fill	$Y_{i+p} = A_i$, N 2)				*		*	
NB-SN-SHB	61		$Y_{i+p} = B_i$, N 2)				*		*	
NBROT-A	64	Field Rotate	$Y_i = A_{(i-p) \bmod 32}$ 3)				*		*	
NBROT-B	65		$Y_i = B_{(i-p) \bmod 32}$ 3)				*		*	

Notes: 1. These instructions use the field instruction format (FORMAT 2).
2. "p" stands for bit displacement from $P_0 - P_5$ or from $PR_0 - PR_5$ ($-32 \leq p \leq 31$).
If p is positive, Y_{p-1} to Y_0 are equal to the fill bit.
If p is negative, Y_{31} to Y_{31+p+1} are equal to the fill bit.
3. The sign of the position input is ignored for this instruction and $P_0 - P_4$ are treated as a positive magnitude for a circular upshift.

Legend: A = A Input Q = Q Register
B = B Input * = Updated

Examples: *
NB-0F-SHA,,4 Shift A up 4 bits and zero fill

NB-0F-SHB,,-17 Shift B down 17 bits and sign fill

*Width field not used

TABLE 14-1. BIT-MANIPULATION INSTRUCTIONS

Mnemonics	Code	Description	Y Output Unsel	Y Output Sel	Status S	M	L	Z	V	N	C
SETBIT-A	68	Bit Set	A	$Y_i = A_i$, $Y_p = 1$				*		*	
SETBIT-B	69		B	$Y_i = B_i$, $Y_p = 1$				*		*	
RSTBIT-A	6A	Bit Reset	A	$Y_i = A_i$, $Y_p = 0$				*		*	
RSTBIT-B	6B		B	$Y_i = B_i$, $Y_p = 0$				*		*	
EXTBIT-A	66	Bit Extract	0	if $p > 0$, $Y_0 = \overline{A_p}$ 2) if $p < 0$, $Y_0 = \overline{A_p}$			*	*			
EXTBIT-B	67		0	if $p > 0$, $Y_0 = \overline{B_p}$ 2) if $p < 0$, $Y_0 = \overline{B_p}$			*	*			
EXTBIT-STAT	7E		0	if $p > 0$, $Y_0 = \overline{S_p}$ 2) if $p < 0$, $Y_0 = \overline{S_p}$			*				

Notes: 1. These instructions use the field instruction format (FORMAT 2).
2. Y_{31} to Y_1 are set to zero. "p" stands for the bit displacement from $P_0 - P_4$ or from $PR_0 - PR_5$. The sign of the position input is ignored.

TABLE 14-2. BIT-MANIPULATION INSTRUCTIONS

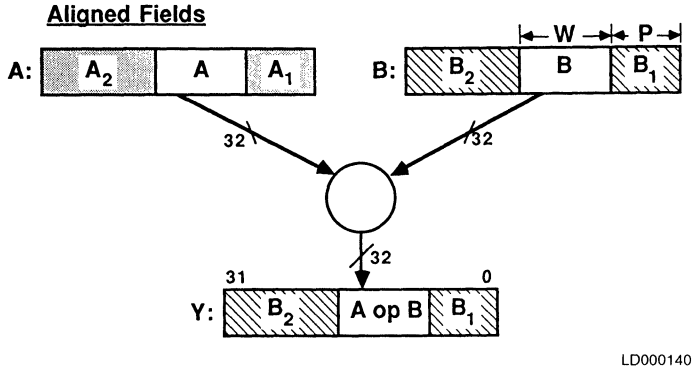
Mnemonics	Code	Description	Status Register	Y Output	Status S	M	L	Z	V	N	C
SETBIT-STAT	6C	Status Bit Set	$S_p = 1$	S	*	*	*	*	*	*	*
RSTBIT-STAT	6D		$S_p = 0$	S	*	*	*	*	*	*	*

Notes: 1. These instructions use the Field instruction format (FORMAT 2).
2. "p" stands for the bit displacement from $P_0 - P_5$ or from $PR_0 - PR_5$.

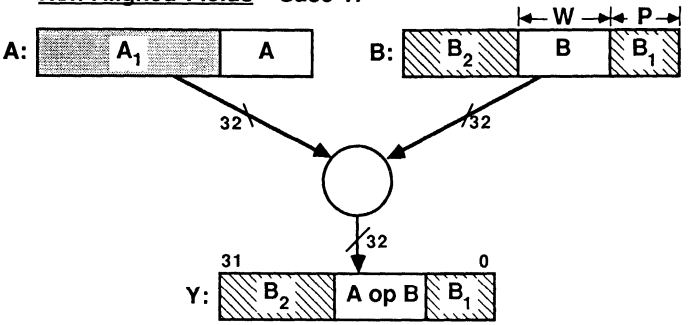
Legend: Unsel = Unselected field Q = Q Register
Sel = Selected field * = Updated
A = A Input
B = B Input

Examples:

RSTBIT-B,,3 3rd bit is set to 0 in B
EXTBIT-STAT,,-4 4th bit in status register is extracted and inverted.

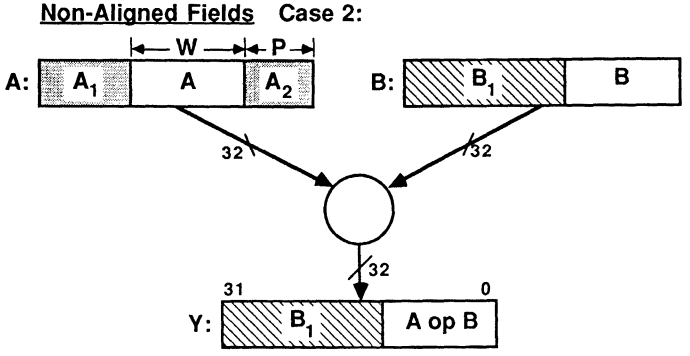


LD000140



If position (P_0-P_5) ≥ 0, A is LSB aligned
Width (W_0-W_4) = 1 to 32

LD000151



If position (P_0-P_5) < 0, B is LSB aligned
Width (W_0-W_5) = 1 to 32

LD000161

Figure 6. Field Logical Operations

TABLE 15. FIELD LOGICAL INSTRUCTIONS

Mnemonics	Code	Description	Y Output		Status					
			Unsel	Sel	S	M	L	Z	V	N	C
PASSF-AL-A	73	Field Pass 3)	B	$Y_i = A_i$				*		*	
PASSF-AL-B	6F	3)	B	$Y_i = B_i$				*		*	
PASSF-A	72	4)	B	if $p \geq 0$, $Y_i = A_{i-p}$ if $p < 0$, $Y_{i-\|p\|} = A_i$				* *		* *	
NOTF-AL-A	71	Field Complement 3)	B	$Y_i = \overline{A_i}$				*		*	
NOTF-AL-B	6E	3)	B	$Y_i = \overline{B_i}$				*		*	
NOTF-A	70	4)	B	if $p \geq 0$, $Y_i = \overline{A_{i-p}}$ if $p < 0$, $Y_{i-\|p\|} = \overline{A_i}$				* *		* *	
ORF-AL-A	75	Field OR 3)	B	$Y_i = A_i$ OR B_i				*		*	
ORF-A	74	4)	B	if $p \geq 0$, $Y_i = A_{i-p}$ OR B_i if $p < 0$, $Y_{i-\|p\|} = A_i$ OR $B_{i-\|p\|}$				* *		* *	
XORF-AL-A	77	Field XOR 3)	B	$Y_i = A_i$ XOR B_i				*		*	
XORF-A	76	4)	B	if $p \geq 0$, $Y_i = A_{i-p}$ XOR B_i if $p < 0$, $Y_{i-\|p\|} = A_i$ XOR $B_{i-\|p\|}$				* *		* *	
ANDF-AL-A	79	Field AND 3)	B	$Y_i = A_i$ AND B_i				*		*	
ANDF-A	78	4)	B	if $p \geq 0$, $Y_i = A_{i-p}$ AND B_i if $p < 0$, $Y_{i-\|p\|} = A_i$ AND $B_{i-\|p\|}$				* *		* *	
EXTF-A	7A	Field Extract 4) 5)	0	if $p \geq 0$, $Y_i = A_{i-p}$ if $p < 0$, $Y_{i-\|p\|} = A_i$				* *		* *	
EXTF-B	7B	4) 5)	0	if $p \geq 0$, $Y_i = B_{i-p}$ if $p < 0$, $Y_{i-\|p\|} = B_i$				* *		* *	
EXTF-AB	7C		0	6)				*		*	
EXTF-BA	7D		0	7)				*		*	

Notes: 1. These instructions use the field instruction format (FORMAT 2).
2. $p \leq i \leq p + w - 1$. "p" stands for position displacement from $P_0 - P_5$ or from $PR_0 - PR_5$ and "w" for the width of the bit field from $W_0 - W_4$ or $WR_0 - WR_4$. Whenever $p + w > 32$, operation takes place only over the portion of the field up to the end of the word. No wraparound occurs.
3. This instruction uses the aligned format (see Figure 6).
4. This instruction uses the unaligned field format (see Figure 6).
$p \geq 0$: Case 1
$p < 0$: Case 2
5. If p is positive, the input is LSB aligned and Y output aligned at position.
If p is negative, the input is aligned at $|p|$ and Y output at LSB.
6. Firstly, the concatenation of A(High Word) and B(Low Word) is rotated by the amount specified by the position (p). If p is positive, left-rotate is performed. If p is negative, right-rotate is performed. Secondly, the least significant bits on the Y output specified by the width (w) are extracted.
7. Same as 6) except that B input is taken as a high word and A input as a low word.

Legend: Unsel = Unselected Field
Sel = Selected Field
A = A Input
B = B Input
Q = Q Register
* = Updated

For all examples, assume STATUS (7 : 0) is −7 and STATUS (12 : 8) is 3.

1. 0,PASSF-AL-B,11,20 — Pass B to Y and test if B_{20} to B_{30} are all zero. Set Z status if so.

B: 1[00000000000]00000101011100110100

Z set to 1 in this case

2. 3,XORF-A,, — Exclusive-OR bits $A_7 - A_9$ with bits $B_0 - B_2$ and output to $Y_0 - Y_2$. Pass $B_3 - B_{31}$ to $Y_3 - Y_{31}$. Width and position values are obtained from STATUS(12 : 0).

A: 01101110001001000010111[100]1101011

B: 00011100001010001100101001001[001]

—

$A_{9-7} \oplus B_{2-0} = Y$: 00011100001010001100101001001[101]

TABLE 16. MASK INSTRUCTION

Mnemonics	Code	Description	Y Output		Status						
			Unsel	Sel	S	M	L	Z	V	N	C
PASS-MASK	7F	Generate Mask	p_5	$Y_i = \overline{p_5}$							

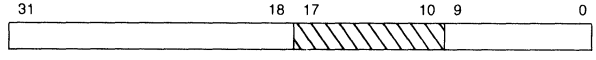
Notes: 1. This instruction uses the field instruction format (FORMAT 2).
2. $p \leqslant i \leqslant p + w - 1$. "p" stands for the position displacement and "w" for the width of bit field.

Legend: Unsel = Unselected Field
Sel = Selected Field
A = A Input
B = B Input
Q = Q Register
* = Updated

Example:

0, PASS-MASK, 8, 10

Generates an 8-bit field mask pattern starting from bit position 10.



INSTRUCTION SET GLOSSARY
(Sorted by Opcode in Hex Notation)

Opcode	Name	Opcode	Name	Opcode	Name	Opcode	Name
00	ZERO-EXTA	**20**	DN1-0F-A	**40**	AND	**60**	NB-SN-SHA
01	ZERO-EXTB	**21**	DN1-0F-B	**41**	XNOR	**61**	NB-SN-SHB
02	SIGN-EXTA	**22**	DN1-0F-AQ	**42**	ADD	**62**	NB-0F-SHA
03	SIGN-EXTB	**23**	DN1-0F-BQ	**43**	ADDC	**63**	NB-0F-SHB
04	PASS-STAT	**24**	DN1-1F-A	**44**	SUB	**64**	NBROT-A
05	PASS-Q	**25**	DN1-1F-B	**45**	SUBC	**65**	NBROT-B
06	LOADQ-A	**26**	DN1-1F-AQ	**46**	SUBR	**66**	EXTBIT-A
07	LOADQ-B	**27**	DN1-1F-BQ	**47**	SUBRC	**67**	EXTBIT-B
08	NOT-A	**28**	DN1-LF-A	**48**	SUM-CORR-A	**68**	SETBIT-A
09	NOT-B	**29**	DN1-LF-B	**49**	SUM-CORR-B	**69**	SETBIT-B
0A	NEG-A	**2A**	DN1-LF-AQ	**4A**	DIFF-CORR-A	**6A**	RSTBIT-A
0B	NEG-B	**2B**	DN1-LF-BQ	**4B**	DIFF-CORR-B	**6B**	RSTBIT-B
0C	PRIOR-A	**2C**	DN1-AR-A	**4C**	-	**6C**	SETBIT-STAT
0D	PRIOR-B	**2D**	DN1-AR-B	**4D**	-	**6D**	RSTBIT-STAT
0E	MERGEA-B	**2E**	DN1-AR-AQ	**4E**	SDIVFIRST	**6E**	NOTF-AL-B
0F	MERGEB-A	**2F**	DN1-AR-BQ	**4F**	UDIVFIRST	**6F**	PASSF-AL-B
10	DECR-A	**30**	UP1-0F-A	**50**	SDIVSTEP	**70**	NOTF-A
11	DECR-B	**31**	UP1-0F-B	**51**	SDIVLAST1	**71**	NOTF-AL-A
12	INCR-A	**32**	UP1-0F-AQ	**52**	MPDIVSTEP1	**72**	PASSF-A
13	INCR-B	**33**	UP1-0F-BQ	**53**	MPSDIVSTEP3	**73**	PASSF-AL-A
14	DECR2-A	**34**	UP1-1F-A	**54**	UDIVSTEP	**74**	ORF-A
15	DECR2-B	**35**	UP1-1F-B	**55**	UDIVLAST	**75**	ORF-AL-A
16	INCR2-A	**36**	UP1-1F-AQ	**56**	MPDIVSTEP2	**76**	XORF-A
17	INCR2-B	**37**	UP1-1F-BQ	**57**	MPUDIVSTP3	**77**	XORF-AL-A
18	DECR4-A	**38**	UP1-LF-A	**58**	REMCORR	**78**	ANDF-A
19	DECR4-B	**39**	UP1-LF-B	**59**	QUOCORR	**79**	ANDF-AL-A
1A	INCR4-A	**3A**	UP1-LF-AQ	**5A**	SDIVLAST2	**7A**	EXTF-A
1B	INCR4-B	**3B**	UP1-LF-BQ	**5B**	UMULFIRST	**7B**	EXTF-B
1C	LDSTAT-A	**3C**	ZERO	**5C**	UMULSTEP	**7C**	EXTF-AB
1D	LDSTAT-B	**3D**	SIGN	**5D**	UMULLAST	**7D**	EXTF-BA
1E	-	**3E**	OR	**5E**	SMULSTEP	**7E**	EXTBIT-STAT
1F	-	**3F**	XOR	**5F**	SMULFIRST	**7F**	PASS-MASK

CHAPTER 2

Detailed Instruction Description

In
Alphabetical
Order

ADD

ADD

Addition

Purpose: Binary addition in 2's complement.

Status Generated:

Z	N	C	V	L	M	S

Z Set if result equals zero. Cleared otherwise.

N Set if result is negative. Cleared otherwise.

C Set if carry is generated. Cleared otherwise.

V Set if an arithmetic overflow occurs. Cleared otherwise.

STATUS <31:24> ← BCD nibble carry

$Opcode_{16}$: 42

Description: Byte-width selects the number of least-significant bytes of A and B that participate in an arithmetic addition whose result is output on the Y bus. Unselected high bytes of the Y bus contain corresponding bytes of B, unaltered. Status is defined for the selected least-significant bytes only.

Notes: **1**. Carry is generated regardless of the state of borrow-mode.

2. Status bits 31 to 24 are loaded with inter-nibble BCD carries for ADD. The SUM-CORR instruction must then be used to add BCD numbers.

ADDC

ADDC

Addition With Carry

Purpose: To perform binary addition in 2's complement on integers longer than 32 bits. Also, provides the hooks for emulation of multiprecision macroinstructions via the MACRO CARRY input.

Status Generated:

Z	N	C	V	L	M	S

Z Set if result equals zero. Cleared otherwise.

N Set if result is negative. Cleared otherwise.

C Set if carry is generated. Cleared otherwise.

V Set if an arithmetic overflow occurs. Cleared otherwise.

STATUS <31:24> ← BCD nibble carry

Opcode$_{16}$: 43

Description: Byte-width selects the number of least significant bytes of A and B that participate in a multiprecision arithmetic addition whose result is output on the Y bus. Unselected high bytes of the Y bus contain corresponding bytes of B, unaltered. Status is defined for the selected least- significant bytes only.

Notes: **1**. Carry is generated regardless of the state of borrow-mode.

2. A HIGH on the MACRO input selects the MACRO-CARRY input instead of the C flag of the (micro) status register as the carry-in for ADDC. This allows an external carry flip flop to be selected for macro-instruction emulation.

3. Status bits 31 to 24 are loaded with inter-nibble BCD carries for ADDC. The SUM-CORR instruction must then be used to add BCD numbers.

4. When performing multi-precision BCD arithmetic, a SUM_CORR must be performed after each add instruction.

Logical And

Purpose: To compute a logical AND of two integers of a given byte- width.

Status Generated:

Z	N	C	V	L	M	S

Z Set if result of AND operation equals zero. Cleared otherwise.

N Set if MSB of selected bytes equals one. Cleared otherwise.

$Opcode_{16}$: 40

Description: Byte-width selects the number of least-significant bytes of A and B that participate in a logical AND operation whose result is output on the Y bus. Unselected high bytes of the Y bus contain corresponding bytes of B, unaltered. Status is defined for the selected least-significant bytes only.

ANDF-A

And Field in A with B

Purpose: To perform a logical AND operation between a field in A and a field in B.

Status Generated:

Z	N	C	V	L	M	S

Z Set if result of AND operation is zero. Cleared otherwise.

N Set if Y_{31} is a one. Cleared otherwise.

$Opcode_{16}$: 78

Description: When the position input is positive, the LSB-aligned field in A is upshifted to align it with the field in B starting at the bit specified by position. When the position input is negative, the field in A starting at the bit specified by the two's complement of position is rotated right to align it with the LSB-aligned field in B. A logical AND operation then takes place between the field in A and the field in B, up to the width specified by the width field and the resulting field appears on the Y bus, LSB-aligned in the case that position was negative, or starting at the bit specified by position, in the case that position was positive. The remaining bits of Y pass corresponding bits of B unaltered. The Z flag is set according to the contents of the selected field. However, the N flag is set according to the most significant bit of Y.

Notes: **1.** A HIGH on the I8 input selects the position inputs from the status register (STATUS <5:0>); a LOW selects the direct position <5:0> input pins.

2. A HIGH on the I7 input selects the width inputs from the status register (STATUS <12:8>); a LOW selects the direct width <4:0> input pins.

3. For position <0, A is rotated rather than shifted right. Therefore, if the sum of position and width exceeds 32, bits rotated around from the low-order end A will be selected.

And Field in A with B

Notes: **4.** For p≥ 0, a width of 0 specifies the entire field to the left of position.
e.g., p=03; w=0

```
A  = 0111 0010 0001 0101 0001 1000 0000 1111 = 7215180F
A1 = 1001 0000 1010 1000 1100 0000 0111 1xxx
B  = 1000 1111 1100 1110 0001 0011 0111 0010 = 8FCE1372
Y  = 1000 0000 1000 1000 0000 0000 0111 0010 = 80880072
```

For p<0, a width of 0 specifies the entire 32 bit field.
e.g., p=3C; w=0

```
A  = 0111 0010 0001 0101 0001 1000 0000 1111 = 7215180F
A' = 1001 0000 1010 1000 1100 0000 0111 1xxx
B  = 1000 1111 1100 1110 0001 0011 0111 0010 = 8FCE1372
Y  = 1000 0000 1000 1000 0000 0000 0111 0010 = 80880072
```

ANDF-AL-A

ANDF-AL-A

And Aligned Fields in A & B

Purpose: To perform a logical AND operation between aligned fields in A & B.

Status Generated:

Z	N	C	V	L	M	S

Z Set if result of AND operation is zero. Cleared otherwise.

N Set if Y_{31} is a one. Cleared otherwise.

$Opcode_{16}$: 79

Description: A logical AND operation is performed between the fields in A and B starting at the LSB specified by the position input and of the specified width. The resulting field appears on the Y bus at the same position. The remaining bits of Y pass corresponding bits of B unaltered. The Z flag is set according to the contents of the selected field. However, the N flag is set according to the most significant bit of Y.

Notes: **1**. A HIGH on the I-8 input selects the position inputs from the status register (STATUS <5:0>); a LOW selects the direct position <5:0> input pins.

2. A HIGH on the I-7 input selects the width inputs from the status register (STATUS <12:8>); a LOW selects the direct width <4:0> input pins.

3. A width of 0 specifies the entire field to the left of position.

4. The MSB of the position inputs is ignored.

DECR-A

DECR-A

Decrement A by 1

Purpose: To do arithmetic on pointers.

Status Generated:

Z	N	C	V	L	M	S

Z Set if result equals zero. Cleared otherwise.

N Set if result is negative. Cleared otherwise.

C EX-OR of Carryout & BORROW-MODE.

V Set if underflow occurs. Cleared otherwise.

STATUS <31:24> ← BCD nibble borrows.

$Opcode_{16}$: 10

Description: A is decremented by 1. Unselected high bytes of A are passed onto the Y bus. Status is defined for selected least- significant bytes only.

Notes: **1**. A HIGH on the BORROW-MODE pin causes a borrow to be generated, whereas a LOW causes a carry to be generated.

2. Status bits 31 to 24 are loaded with inter-nibble BCD borrows for DECR. The DIFF-CORR instruction must then be used to decrement a BCD number.

DECR2-A DECR2-A

Decrement A by 2

Purpose: To do arithmetic on pointers.

Status Generated:

Z	N	C	V	L	M	S

Z Set if result equals zero. Cleared otherwise.

N Set if result is negative. Cleared otherwise.

C EX-OR of Carryout & BORROW-MODE.

V Set if underflow occurs. Cleared otherwise.

STATUS <31:24> ← BCD nibble borrows.

$Opcode_{16}$: 14

Description: A is decremented by 2. Unselected high bytes of A are passed to the Y bus. Status is defined for selected least- significant bytes only.

Notes: **1**. A HIGH on the BORROW-MODE pin causes a borrow to be generated, whereas a LOW causes a carry to be generated.

2. Status bits 31 to 24 are loaded with inter-nibble BCD borrows for DECR. The DIFF-CORR instruction must then be used to decrement a BCD number.

DECR4-A

DECR4-A

Decrement A by 4

Purpose: To do arithmetic on pointers.

Status Generated:

Z	N	C	V	L	M	S

Z Set if result equals zero. Cleared otherwise.

N Set if result is negative. Cleared otherwise.

C EX-OR of Carryout & BORROW-MODE.

V Set if underflow occurs. Cleared otherwise.

STATUS <31:24> ← BCD nibble borrows.

$Opcode_{16}$: 18

Description: A is decremented by 4. Unselected high bytes of A are passed on to the Y bus. Status is defined for selected least- significant bytes only.

Notes: **1.** A HIGH on the BORROW-MODE pin causes a borrow to be generated, whereas a LOW causes a carry to be generated.

2. Status bits 31 to 24 are loaded with inter-nibble BCD borrows for DECR. The DIFF-CORR instructions must then be used to decrement a BCD number.

DECR-B

DECR-B

Decrement B by 1

Purpose: To do arithmetic on pointers.

Status Generated:

Z	N	C	V	L	M	S

Z Set if result equals zero. Cleared otherwise.

N Set if result is negative. Cleared otherwise.

C EX-OR of Carryout & BORROW-MODE.

V Set if underflow occurs. Cleared otherwise.

STATUS <31:24> ← BCD nibble borrows.

$Opcode_{16}$: 11

Description: B is decremented by 1. Unselected high bytes of B are passed onto the Y bus. Status is defined for selected least- significant bytes only.

Notes: **1**. A HIGH on the BORROW-MODE pin causes a borrow to be generated, whereas a LOW causes a carry to be generated.

2. Status bits 31 to 24 are loaded with inter-nibble BCD borrows for DECR. The DIFF-CORR instruction must then be used to decrement a BCD number.

DECR2-B

DECR2-B

Decrement B by 2

Purpose: To do arithmetic on pointers.

Status Generated:

Z	N	C	V	L	M	S

Z Set if result equals zero. Cleared otherwise.

N Set if result is negative. Cleared otherwise.

C EX-OR of Carryout & BORROW-MODE.

V Set if underflow occurs. Cleared otherwise.

STATUS <31:24> ← BCD nibble borrows.

$Opcode_{16}$: 15

Description: B is decremented by 2. Unselected high bytes of B are passed onto the Y bus. Status is defined for selected least- significant bytes only.

Notes: **1.** A HIGH on the BORROW-MODE pin causes a borrow to be generated, whereas a LOW causes a carry to be generated.

2. Status bits 31 to 24 and loaded with inter-nibble BCD borrows for DECR. The DIFF-CORR instruction must then be used to decrement a BCD number.

DECR4-B

DECR4-B

Decrement B by 4

Purpose: To do arithmetic on pointers.

Status Generated:

Z	N	C	V	L	M	S

Z Set if result equals zero. Cleared otherwise.

N Set if result is negative. Cleared otherwise.

C EX-OR of Carryout & BORROW-MODE.

V Set if underflow occurs. Cleared otherwise.

STATUS <31:24> ← BCD nibble borrows.

$Opcode_{16}$: 19

Description: B is decremented by 4. Unselected high bytes of B are passed onto the Y bus. Status is defined for selected least- significant bytes only.

Notes: **1**. A HIGH on the BORROW-MODE pin causes a borrow to be generated, whereas a LOW causes a carry to be generated.

2. Status bits 31 to 24 are loaded with inter-nibble BCD borrows for DECR. The DIFF-CORR instruction must then be used to decrement a BCD number.

DIFF-CORR-A DIFF-CORR-A

BCD Difference Correct A

Purpose: To correct for BCD operations after execution of NEG, DECR, or SUB(RAC) instructions.

Status Generated:

Z	N	C	V	L	M	S

Z Set if result equals zero. Cleared otherwise.

N Set if MSB of result equals one. Cleared otherwise.

C EX-NOR of most significant NIBBLE BORROW and BORROW-MODE.

V EX-OR of two most significant NIBBLE BORROW's based on byte-width.

$Opcode_{16}$: 4A

Description: Byte-width selects the number of least-significant nibble- pairs of A that are corrected before being output on the Y bus. Unselected high bytes of Y contain corresponding bytes of A unaltered.

A 32-bit correction word is formed as follows: A field of 8 nibbles (32 bits) is set to zero. The nibble borrows NC7-NC0 are then tested. Any nibble borrow that is set to 1 causes a 6 (0110) to be placed in its corresponding nibble position in the correction word. The correction word is then subtracted from the operand. The nibble borrow flip-flops themselves are not affected by this operation. Status is defined for the selected least significant bytes only.

Notes: **1.** DIFF-CORR is used after NEG, DECR, SUB, SUBR, SUBC or SUBRC.

2. The definition of negative and overflow flags assumes that one leading sign-digit is appended to odd lengths to form an even length string (the definition of overflow is not useful for odd length signed strings). A digit of 0 represents a positive sign and a digit of 9 represents a negative sign in ten's complement. The N flag is set if the most-significant bit of the operand selected by byte- width is 1. The overflow is defined as the exclusive-OR of the borrow-in and borrow-out of the most-significant nibble (available

BCD Difference Correct A

Notes: from the nibble-carry register), as defined by byte-width. The sign nibble of the result is always either 0 or 9, except when an overflow occurs, when it becomes either 1 or 8 respectively. Refer to appendix A.

3. The carry flag is loaded from the appropriate nibble- borrow flag, as defined by byte-width.

4. Subtraction can use carry/borrow arithmetic as described for SUB under control of the BORROW-MODE.

BCD Difference Correct B

Purpose: To correct for BCD operations after execution of NEG, DECR, or SUB (RAC) instructions.

Status Generated:

Z	N	C	V	L	M	S

Z Set if result equals zero. Cleared otherwise.

N Set if MSB of result equals one. Cleared otherwise.

C EX-NOR of most significant NIBBLE BORROW and BORROW-MODE.

V EX-OR of two most significant NIBBLE BORROW's based on byte-width.

$Opcode_{16}$: 4B

Description: Byte-width selects the number of least-significant nibble- pairs of B that are corrected before being output on the Y bus. Unselected high bytes of Y pass corresponding bytes of B unaltered.

A 32-bit correction word is formed as follows: A field of 8 nibbles (32 bits) is set to zero. The nibble borrows NC7-NC0 are then tested. Any nibble borrow that is set to 1 causes a 6 (0110) to be placed in its corresponding nibble position in the correction word. The correction word is then subtracted from the operand. The nibble borrow flip-flops themselves are not affected by this operation. Status is defined for the selected least significant bytes only.

Notes: 1. DIFF-CORR is used after NEG, DECR, SUB, SUBR, SUBC, or SUBRC.

2. The definition of negative and overflow flags assumes that one leading sign-digit is appended to odd length to form an even length string (the definition of overflow is not useful for odd length signed strings). A digit of 0 represents a positive sign and a digit of 9 represents a negative sign in ten's complement. The N flag is set if the most-significant bit of the operand selected by byte- width is 1. The overflow is defined as the exclusive-OR of the borrow-in and borrow-out of the most-significant nibble (available from the nibble-carry register), as defined by byte-width. The sign nibble of

BCD Difference Correct B

Notes: the result is always either 0 or 9, except when an overflow occurs, when it becomes either 1 or 8 respectively. Refer to Appendix A.

3. The carry flag is loaded from the appropriate nibble- borrow flag, as defined by byte-width.

4. Subtraction can use carry/borrow arithmetic as described for SUB under control of the BORROW-MODE.

DN1-0F-A

DN1-0F-A

Downshift A by 1 Bit, Zero Fill

Purpose: Downshift A by one bit, zero fill.

Status Generated:

Z	**N**	C	V	**L**	M	S

Z Set if result is zero. Cleared otherwise.

N Reset always.

L LSB of A moved to L

Opcode$_{16}$: 20

Description: Byte-width selects the number of least-significant bytes of A that are downshifted by one bit, with zero shifted in, and passed onto the Y bus. Unselected high bytes of Y contain corresponding bytes of A, unaltered. The L flag gets the least significant bit of A. Status is defined for selected least-significant bytes only.

DN1-0F-AQ DN1-0F-AQ

Downshift A, Q by 1 Bit, Zero Fill

Purpose: Double precision downshift A, Q by one bit. Zero fill.

Status Generated:

Z	**N**	C	V	**L**	M	S

Z Set if result is zero. Cleared otherwise.

N Reset always.

L LSB of Q moved to L.

$Opcode_{16}$: 22

Description: Byte-width selects the number of least-significant bytes of A that are downshifted by one bit, with zero shifted in, and passed onto the Y bus. Unselected high bytes of Y contain corresponding bytes of A, unaltered. Byte-width also selects the number of least-significant bytes of Q that are shifted down by one bit with the least significant bit of A as the fill bit. Unselected high bytes of Q are not affected. The L flag gets the least significant bit of Q. Status is defined for selected least-significant bytes only.

DN1-0F-B

DN1-0F-B

Downshift B by 1 Bit, Zero Fill

Purpose: Downshift B by one bit, zero fill.

Status Generated:

Z	N	C	V	L	M	S

Z Set if result is zero. Cleared otherwise.

N Reset always.

L LSB of B moved to L.

$Opcode_{16}$: 21

Description: Byte-width selects the number of least-significant bytes of B that are downshifted by one bit, with zero shifted in, and passed onto the Y bus. Unselected high bytes of Y contain corresponding bytes of B, unaltered. The L flag gets the least significant bit of B. Status is defined for selected least-significant bytes only.

DN1-0F-BQ

DN1-0F-BQ

Downshift B, Q by 1 Bit, Zero Fill

Purpose: Double precision downshift B, Q by one bit. Zero fill.

Status Generated:

Z	N	C	V	L	M	S

Z Set if result is zero. Cleared otherwise.

N Reset always.

L LSB of Q moved to L.

$Opcode_{16}$: 23

Description: Byte-width selects the number of least-significant bytes of B that are downshifted by one bit, with zero shifted in, and passed onto the Y bus. Unselected high bytes of Y contain corresponding bytes of B, unaltered. Byte-width also selects the number of least-significant bytes of Q that are shifted down by one bit with the least significant bit of B as the fill bit. Unselected high order bits of Q are not affected. The L flag gets the least significant bit of Q. Status is defined for selected least-significant bytes only.

DN1-1F-A

Downshift A by 1 Bit, One Fill

Purpose: Downshift A by one bit, one fill.

Status Generated:

Z	**N**	C	V	**L**	M	S

Z Set if result is zero. Cleared otherwise.

N Set always.

L LSB of A moved to L.

Opcode$_{16}$: 24

Description: Byte-width selects the number of least-significant bytes of A that are downshifted by one bit, with a one shifted in, and passed onto the Y bus. Unselected high bytes of Y contain corresponding bytes of A, unaltered. The L flag gets the least significant bit of A. Status is defined for selected least-significant bytes only.

DN1-1F-AQ DN1-1F-AQ

Downshift A, Q by 1 Bit, One Fill

Purpose: Double precision downshift A, Q by one bit. One fill.

Status Generated:

Z	N	C	V	L	M	S

Z Set if result is zero. Cleared otherwise.

N Set always.

L LSB of Q moved to L.

$Opcode_{16}$: 26

Description: Byte-width selects the number of least-significant bytes of A that are downshifted by one bit, with a one shifted in, and passed onto the Y bus. Unselected high bytes of Y contain corresponding bytes of A, unaltered.Byte-width also selects the number of least-significant bytes of Q that are shifted down by one bit with the least significant bit of A as the fill bit. Unselected high bytes of Q are not affected. The L flag gets the least significant bit of Q. Status is defined for selected least-significant bytes only.

DN1-1F-B

DN1-1F-B

Downshift B by 1 Bit, One Fill

Purpose: Downshift B by one bit, one fill.

Status Generated:

Z	**N**	C	V	**L**	M	S

Z Set if result is zero. Cleared otherwise.

N Set always.

L LSB of B moved to L.

Opcode$_{16}$: 25

Description: Byte-width selects the number of least-significant bytes of B that are downshifted by one bit, with a one shifted in, and passed onto the Y bus. Unselected high bytes of Y contain corresponding bytes of B, unaltered. The L flag gets the least significant bit of B. Status is defined for selected least-significant bytes only.

DN1-1F-BQ

DN1-1F-BQ

Downshift B, Q by 1 Bit, One Fill

Purpose: Double precision downshift B, Q by 1 bit. One fill.

Status Generated:

Z	N	C	V	L	M	S

Z Set if result is zero. Cleared otherwise.

N Set always.

L LSB of Q moved to L.

$Opcode_{16}$: 27

Description: Byte-width selects the number of least-significant bytes of B that are downshifted by one bit, with a one shifted in, and passed onto the Y bus. Unselected high bytes of Y contain corresponding bytes of B, unaltered. Byte-width also selects the number of least-significant bytes of Q that are shifted down by one bit with the least significant bit of B as the fill bit. Unselected high bytes of Q are not affected. The L flag gets the least significant bit of Q. Status is defined for selected least-significant bytes only.

DN1-AR-A DN1-AR-A

Downshift A by 1 Bit, Sign Bit Fill

Purpose: Downshift A by one bit. Sign bit fill.

Status Generated:

Z	**N**	C	V	**L**	M	S

Z Set if result is zero. Cleared otherwise.

N Set if result is negative. Cleared otherwise.

L LSB of A moved to L.

$Opcode_{16}$: 2C

Description: Byte-width selects the number of least-significant bytes of A that are downshifted by one bit, with the sign bit shifted in, and passed onto the Y bus. Unselected high bytes of Y contain corresponding bytes of A, unaltered. The L flag gets the least significant bit of A. Status is defined for selected least-significant bytes only.

Notes: 1. Choice of signbit as fill bit results in an arithmetic downshift.

DN1-AR-AQ DN1-AR-AQ

Downshift A, Q by 1 Bit, Sign Bit Fill

Purpose: Double precision downshift A, Q by 1 bit. Sign bit fill.

Status Generated:

Z	**N**	C	V	**L**	M	S

Z Set if result is zero. Cleared otherwise.

N Set if result is negative. Cleared otherwise.

L LSB of Q moved to L.

$Opcode_{16}$: 2E

Description: Byte-width selects the number of least-significant bytes of A that are downshifted by one bit, with the sign bit shifted in, and passed onto the Y bus. Unselected high bytes of Y contain corresponding bytes of A, unaltered. Byte-width also selects the number of least-significant bytes of Q that are shifted down by one bit with the least significant bit of A as the fill bit. Unselected high bytes of Q are not affected. The L flag gets the least significant bit of Q. Status is defined for selected least-significant bytes only.

Notes: 1. Choice of signbit as fill bit results in an arithmetic downshift.

DN1-AR-B DN1-AR-B

Downshift B by 1 Bit, Sign Bit Fill

Purpose: Downshift B by one bit. Sign bit fill.

Status Generated:

Z	N	C	V	L	M	S

Z Set if result is zero. Cleared otherwise.

N Set if result is negative. Cleared otherwise.

L LSB of B moved to L.

$Opcode_{16}$: 2D

Description: Byte-width selects the number of least-significant bytes of B that are downshifted by one bit, with the sign bit shifted in, and passed onto the Y bus. Unselected high bytes of Y contain corresponding bytes of B, unaltered. The L flag gets the least significant bit of B. Status is defined for selected least-significant bytes only.

Notes: 1. Choice of signbit as fill bit results in an arithmetic downshift.

DN1-AR-BQ DN1-AR-BQ

Downshift B, Q by 1 Bit, Sign Bit Fill

Purpose: Double precision downshift B, Q by 1 bit. Sign bit fill.

Status Generated:

Z	**N**	C	V	**L**	M	S

Z Set if result is zero. Cleared otherwise.

N Set if result is negative. Cleared otherwise.

L LSB of Q moved to L.

$Opcode_{16}$: 2F

Description: Byte-width selects the number of least-significant bytes of B that are downshifted by one bit, with the sign bit shifted in, and passed onto the Y bus. Unselected high bytes of Y contain corresponding bytes of B, unaltered. Byte-width also selects the number of least-significant bytes of Q that are shifted down by one bit with the least significant bit of B being the fill bit. Unselected high bytes of Q are not affected. The L flag gets the least significant bit of Q. Status is defined for selected least-significant bytes only.

Notes: 1. Choice of signbit as fill bit results in an arithmetic downshift.

DN1-LF-A

DN1-LF-A

Downshift A by 1 Bit, Link Bit Fill

Purpose: Downshift A by one bit. Link bit fill.

Status Generated:

Z	**N**	C	V	**L**	M	S

Z Set if result is zero. Cleared otherwise.

N Set if result is negative. Cleared otherwise.

L LSB of A moved to L.

$Opcode_{16}$: 28

Description: Byte-width selects the number of least-significant bytes of A that are downshifted by one bit, with the link bit shifted in, and passed onto the Y bus. Unselected high bytes of Y contain corresponding bytes of A, unaltered. The L flag gets the least significant bit of A. Status is defined for selected least-significant bytes only.

Notes: **1**. This instruction provides the hooks for macroinstruction emulation via the MACROLINK input. A HIGH on the MACRO/$\overline{\text{MICRO}}$ selects the MACRO-LINK input as the fill bit instead of the L bit of the status register.

DN1-LF-AQ DN1-LF-AQ

Downshift A, Q by 1 Bit, Link Bit Fill

Purpose: Double precision downshift A, Q by 1 bit. Link bit fill.

Status Generated:

Z	**N**	C	V	**L**	M	S

Z Set if result is zero. Cleared otherwise.

N Set if result is negative. Cleared otherwise.

L LSB of Q moved to L.

$Opcode_{16}$: 2A

Description: Byte-width selects the number of least-significant bytes of A that are downshifted by one bit, with the link bit shifted in, and passed onto the Y bus. Unselected high bytes of Y contain corresponding bytes of A, unaltered. Byte-width also selects the number of least-significant bytes of Q that are shifted down by one bit with the least significant bit of A as the fill bit. Unselected high bytes of Q are not affected. The L flag gets the least significant bit of Q. Status is defined for selected least-significant bytes only.

Notes: 1. This instruction provides the hooks for macroinstruction emulation via the MACROLINK input. A HIGH on the MACRO/$\overline{\text{MICRO}}$ selects the MACROLINK input as the fill bit instead of the L bit of the status register.

DN1-LF-B

DN1-LF-B

Downshift B by 1 Bit, Link Bit Fill

Purpose: Downshift B by one bit. Link bit fill.

Status Generated:

Z	**N**	C	V	**L**	M	S

Z Set if result is zero. Cleared otherwise.

N Set if result is negative. Cleared otherwise.

L LSB of B moved to L.

Opcode$_{16}$: 29

Description: Byte-width selects the number of least-significant bytes of B that are downshifted by one bit, with the link bit shifted in, and passed onto the Y bus. Unselected high bytes of Y contain corresponding bytes of B, unaltered. The L flag gets the least significant bit of B. Status is defined for selected least-significant bytes only.

Notes: 1. This instruction provides the hooks for macroinstruction emulation via the MACROLINK input. A HIGH on the MACRO/$\overline{\text{MICRO}}$ selects the MACRO-LINK input as the fill bit instead of the L bit of the status register.

DN1-LF-BQ DN1-LF-BQ

Downshift B, Q by 1 Bit, Link Bit Fill

Purpose: Double precision downshift B, Q by 1 bit. Link bit fill.

Status Generated:

Z	**N**	C	V	**L**	M	S

Z Set if result is zero. Cleared otherwise.

N Set if result is negative. Cleared otherwise.

L LSB of Q moved to L.

Opcode$_{16}$: 2B

Description: Byte-width selects the number of least-significant bytes of B that are downshifted by one bit, with the link bit shifted in, and passed onto the Y bus. Unselected high bytes of Y contain corresponding bytes of B, unaltered. Byte-width also selects the number of least-significant bytes of Q that are shifted down by one bit with the least significant bit of B as the fill bit. Unselected high bytes of Q are not affected. The L flag get the least significant bit of Q. Status is defined for selected least-significant bytes only.

Notes: 1. This instruction provides the hooks for macroinstruction emulation via the MACROLINK input. A HIGH on the MACRO/$\overline{\text{MICRO}}$ selects the MACROLINK input as the fill bit instead of the L bit of the status register.

Extract Bit A

Purpose: To test any bit of the fullword input A, or to convert any bit of the fullword input A into a Boolean variable.

Status Generated:

Z	N	C	V	L	M	S

Z Set if Y bus is zero. Cleared otherwise.

L Y0 is copied into L.

$Opcode_{16}$: 66

Description: The least significant five bits of the position input are interpreted as an unsigned magnitude giving the position of the bit in A that is to be extracted; the sign bit of position, if true, inverts the extracted bit. The extracted bit appears on bit 0 of the Y bus with the high-order bits of Y all being zero. It is also loaded into the L flag at the end of the cycle. Z is set to reflect the state of the Y bus.

Notes: 1. A HIGH on the I8 input selects the position inputs from the status register (STATUS <5:0>); a LOW selects the direct position <5:0> input pins.

EXTBIT-B

EXTBIT-B

Extract Bit B

Purpose: To test any bit of the fullword input B, or to convert any bit of the fullword input B into a Boolean variable.

Status Generated:

Z	N	C	V	L	M	S

Z Set if Y bus is zero. Cleared otherwise.

L Y_0 is copied into L.

Opcode$_{16}$: 67

Description: The least significant five bits of the position input are interpreted as an unsigned magnitude giving the position of the bit in B that is to be extracted; the sign bit of position, if true, inverts the extracted bit. The extracted bit appears on bit 0 of the Y bus with the high-order bits of Y all being zero. It is also loaded into the L flag at the end of the cycle. Z is set to reflect the state of the Y bus.

Notes: **1.** A HIGH on the I8 input selects the position inputs from the status register (STATUS <5:0>); a LOW selects the direct position <5:0> input pins.

EXTBIT-STAT EXTBIT-STAT

Extract Status Register Bit

Purpose: To test any bit of the status register, including relational conditions $\geq$, $\leq$ or to convert any of the above to Boolean variables.

Status Generated: | **Z** | N | C | V | **L** | M | S |

Z Set if Y bus is zero. Cleared otherwise.

L Y_0 is copied into L.

Opcode$_{16}$: 7E

Description: The least significant five bits of the position input are interpreted as an unsigned magnitude giving the position of the bit of the status register to be extracted. The sign bit of position, if true, inverts the extracted bit. The extracted bit is then output on bit 0 of the Y bus with the remaining bits of Y being 0. It is also loaded into the L flag at the end of the cycle. There are no side-effects on any bits of the status register other than L.

Notes: **1.** A HIGH on the I8 input selects the position inputs from the status register (STATUS <5:0>); a LOW selects the direct position <5:0> input pins.

2. STATUS <15:13> are read-only relational condition bits. Note that STATUS <13> is affected by CARRY/BORROW mode. When the BORROW-MODE input is high, the condition is C + Z; otherwise it is $\overline{C}$ + Z.

3. Note that STATUS <23> is always LOW. This can be extracted to produce the Boolean constants 0 or 1 on the Y bus.

EXTF-A

Extract Field

Purpose: To extract a field in A.

Status Generated:

Z	**N**	C	V	L	M	S

Z Set if extracted field is zero. Cleared otherwise.

N Set if Y_{31} is a one. Cleared otherwise.

Opcode$_{16}$: 7A

Description: When the position input is positive, the LSB aligned field in A is upshifted to the bit specified by the position input and output on the Y bus. When the position input is negative, the field in A starting at the bit specified by the two's complement of the position input is downshifted so as to align it with the LSB, and output on the Y bus. The width of the field is specified by the width input. The remaining bits of Y contain zeros.

Notes: **1.** A HIGH on the I8 input selects the position inputs from the status register (STATUS <5:0>); a LOW selects the direct position <5:0> input pins.

2. A HIGH on the I7 input selects the width inputs from the status register (STATUS <12:8>); a LOW selects the direct width <4:0> input pins.

3. For position <0, A is rotated rather than shifted down. Therefore, if the sum of position and width exceeds 32, bits rotated around from the low-order end A will be selected.

Extract Field

Notes: **4.** For p≥0, a width of 0 specifies the entire field to the left of position.
e.g., p=03; w=0

```
A  = 0111 0010 0001 0101 0001 1000 0000 0101 = 72151805
A' = 1001 0000 1010 1000 1100 0000 0010 1xxx
B  = 1000 1111 1100 1110 0001 0011 0111 0010 = 8FCE1372
Y  = 1001 0000 1010 1000 1100 0000 0010 1000 = 90A8C028
```

For p<0, a width of 0 specifies the entire 32 bit field.
e.g., p=3C; w=0

```
A  = 0111 0001 0111 0010 0011 1111 1110 1101 = 71723FED
A' = 1101 0111 0001 0111 0010 0011 1111 1110
B  = 1000 0010 1100 1110 0001 0101 0000 1001 = 82CE1509
Y  = 1101 0111 0001 0111 0010 0011 1111 1110 = D71723FE
```

EXTF-AB

Extract Field

Purpose: To extract a field from the concatenation of A and B where A is the high word.

Status Generated:

Z	N	C	V	L	M	S

Z Set if extracted field is zero. Cleared otherwise.

N Set if Y_{31} is a one. Cleared otherwise.

$Opcode_{16}$: 7C

Description: The concatenation of the fullword A (as the high word) and the fullword B (as the low word) is rotated by the amount specified by the position input. If position is positive (0- 31) then a left-rotate by the number of places specified is performed. If position is negative (-1 to -32) then a right- rotate by the number of places specified by position is performed. The width input specifies the number of least- significant bits of the result to appear on the Y bus LSB aligned. The rest of the Y bus contains zeroes.

Notes:

1. A HIGH on the I8 input selects the position inputs from the status register (STATUS <5:0>); a LOW selects the direct position <5:0> input pins.

2. A HIGH on the I7 input selects the width inputs from the status register (STATUS <12:8>); a LOW selects the direct width <4:0> input pins.

3. A width of 0 specifies the 32 LSBs from the 64 bit word.

EXTF-B

Extract Field

Purpose: To extract a field in B.

Status Generated:

Z	N	C	V	L	M	S

Z Set if extracted field is zero. Cleared otherwise.

N Set if Y_{31} is a one. Cleared otherwise.

$Opcode_{16}$: 7B

Description: When the position input is positive, the LSB aligned field in B is upshifted to the bit specified by position and output on the Y bus. When the position input is negative, the field in B starting at the bit specified by the two's complement of position is downshifted so as to align it with the LSB, and output on the Y bus. The width of the field is specified by the width input. The remaining bits of Y contain zeroes.

Notes: **1.** A HIGH on the I8 input selects the position inputs from the status register (STATUS <5:0>); a LOW selects the direct position <5:0> input pins.

2. A HIGH on the I7 input selects the width inputs from the status register (STATUS <12:8>); a LOW selects the direct width <4:0> input pins.

3. For position <0, B is rotated rather than shifted down. Therefore, if the sum of position and width exceeds 32, bits rotated around from the low-order end B will be selected.

Extract Field

Notes: **4.** For p≥0, a width of 0 specifies the entire field to the left of position.
e.g., p=03; w=0

```
B  = 1000 1111 1100 1110 0001 0011 0111 0010 = 8FCE1372
B1 = 0111 1110 0111 0000 1001 1011 1001 0xxx
A  = 0111 0010 0001 0101 0001 1000 0000 0101 = 72151805
Y  = 0111 1110 0111 0000 1001 1011 1001 0000 = 7E709B90
```

For p<0; a width of 0 specifies the entire 32 bit field.
e.g., p=3C; w=0

```
B  = 0111 0001 0111 0010 0011 1111 1110 1101 = 71723FED
B1 = 1101 0111 0001 0111 0010 0011 1111 1110
A  = 1000 0010 1100 1110 0001 0101 0000 1001 = 82CE1509
Y  = 1101 0111 0001 0111 0010 0011 1111 1110 = D71723FE
```

EXTF-BA

Extract Field

Purpose: To extract a field from the concatenation of B and A where B is the high word.

Status Generated:

Z	N	C	V	L	M	S

Z Set if extracted field is zero. Cleared otherwise.

N Set if Y_{31} is a one. Cleared otherwise.

$Opcode_{16}$: 7D

Description: The concatenation of the fullword B (as the high word) and the fullword A (as the low word) is rotated by the amount specified by the position input. If position is positive (0- 31) then a left-rotate by the number of places specified is performed. If position is negative (-1 to -32) then a right- rotate by the number of places specified by position is performed. The width input specifies the number of least- significant bits of the result to appear on the Y bus, LSB aligned. The rest of the Y bus contains zeroes.

Notes: **1**. A HIGH on the I8 input selects the position inputs from the status register (STATUS <5:0>); a LOW selects the direct position <5:0> input pins.

2. A HIGH on the I7 input selects the width inputs from the status register (STATUS <12:8>); a LOW selects the direct width <4:0> input pins.

3. A width of 0 specifies the 32 LSBs from the 64 bit word.

INCR-A

Increment A by 1

Purpose: To do arithmetic on pointers.

Status Generated:

Z	N	C	V	L	M	S

Z Set if result equals zero. Cleared otherwise.

N Set if result is negative. Cleared otherwise.

C EX-OR of carryout and BORROW-MODE.

V Set if overflow occurs. Cleared otherwise.

STATUS <31:24> ← BCD nibble carries.

$Opcode_{16}$: 12

Description: Byte-width selects the number of least significant bytes of A that are incremented and passed onto the Y bus. Unselected high bytes contain corresponding bytes of A, unaltered. Status is defined for the selected bytes only.

Notes: 1. Status bits 31 to 24 are loaded with inter-nibble BCD carries for BCD arithmetic. The SUM-CORR instruction must then be used to increment a BCD number.

INCR2-A

Increment A by 2

Purpose: To do arithmetic on pointers.

Status Generated:

Z	N	C	V	L	M	S

Z Set if result equals zero. Cleared otherwise.

N Set if result is negative. Cleared otherwise.

C EX-OR of carryout and BORROW-MODE.

V Set if overflow occurs. Cleared otherwise.

STATUS <31:24> ← BCD nibble carries.

$Opcode_{16}$: 16

Description: Byte-width selects the number of least significant bytes of A that are incremented by 2 and passed onto the Y bus. Unselected high bytes contain corresponding bytes of A, unaltered. Status is defined for the selected bytes only.

Notes: **1**. Status bits 31 to 24 are loaded with inter-nibble BCD carries for INCR. The SUM-CORR instruction must then be used to increment a BCD number.

INCR4-A INCR4-A

Increment A by 4

Purpose: To do arithmetic on pointers.

Status Generated:

Z	N	C	V	L	M	S

Z Set if result equals zero. Cleared otherwise.

N Set if result is negative. Cleared otherwise.

C EX-OR of carryout and BORROW-MODE.

V Set if overflow occurs. Cleared otherwise.

STATUS <31:24> ← BCD nibble carries.

Opcode$_{16}$: 1A

Description: Byte-width selects the number of least significant bytes of A that are incremented by 4 and passed onto the Y bus. Unselected high bytes contain corresponding bytes of A, unaltered. Status is defined for the selected bytes only.

Notes: **1.** Status bits 31 to 24 are loaded with inter-nibble BCD carries for INCR. The SUM-CORR instruction must then be used to increment a BCD number.

Increment B by 1

Purpose: To do arithmetic on pointers.

Status Generated:

Z	N	C	V	L	M	S

Z Set if result equals zero. Cleared otherwise.

N Set if result is negative. Cleared otherwise.

C EX-OR of carryout and BORROW-MODE.

V Set if overflow occurs. Cleared otherwise.

STATUS <31:24> ← BCD nibble carries.

$Opcode_{16}$: 13

Description: Byte-width selects the number of least significant bytes of B that are incremented and passed onto the Y bus. Unselected high bytes contain corresponding bytes of B, unaltered. Status is defined for the selected bytes only.

Notes: 1. Status bits 31 to 24 are loaded with inter-nibble BCD carries for INCR. The SUM-CORR instruction must then be used to increment a BCD number.

INCR2-B

Increment B by 2

Purpose: To do arithmetic on pointers.

Status Generated:

Z	N	C	V	L	M	S

Z Set if result equals zero. Cleared otherwise.

N Set if result is negative. Cleared otherwise.

C EX-OR of carryout and BORROW-MODE.

V Set if overflow occurs. Cleared otherwise.

STATUS <31:24> ← BCD nibble carries.

$Opcode_{16}$: 17

Description: Byte-width selects the number of least significant bytes of B that are incremented by 2 and passed onto the Y bus. Unselected high bytes contain corresponding bytes of B, unaltered. Status is defined for the selected bytes only.

Notes: 1. Status bits 31 to 24 are loaded with inter-nibble BCD carries for INCR. The SUM-CORR instruction must then be used to increment a BCD number.

INCR4-B

INCR4-B

Increment B by 4

Purpose: To do arithmetic on pointers.

Status Generated:

Z	N	C	V	L	M	S

Z Set if result equals zero. Cleared otherwise.

N Set if result is negative. Cleared otherwise.

C EX-OR of carryout and BORROW-MODE.

V Set if overflow occurs. Cleared otherwise.

STATUS <31:24> ← BCD nibble carries.

Opcode$_{16}$: 1B

Description: Byte-width selects the number of least significant bytes of B that are incremented by 4 and passed onto the Y bus. Unselected high bytes contain corresponding bytes of B, unaltered. Status is defined for the selected bytes only.

Notes: **1**. Status bits 31 to 24 are loaded with inter-nibble BCD carries for INCR. The SUM-CORR instruction must then be used to increment a BCD number.

LOADQ-A

Load A Into Q

Purpose: To set up the dividend prior to division or multiplier prior to multiplication.

Status Generated:

Z	N	C	V	L	M	S

Z Set if the selected A bytes are zero. Cleared otherwise.

N Set if the selected A bytes are negative. Cleared otherwise.

$Opcode_{16}$: 06

Description: Byte-width selects the number of least significant bytes of the A bus that are loaded into the Q register. The unselected high bytes of Q are not affected by the operation. Byte-width also selects the number of least significant bytes of A that are passed onto the Y bus. Unselected high bytes of the Y bus contain corresponding bytes of Q. Status is defined for selected least-significant bytes only.

Load B Into Q

Purpose: To set up the dividend prior to division or multiplier prior to multiplication.

Status Generated:

Z	N	C	V	L	M	S

Z Set if the selected B bytes are zero. Cleared otherwise.

N Set if the selected B bytes are negative. Cleared otherwise.

$Opcode_{16}$: 07

Description: Byte-width selects the number of least significant bytes of the B bus that are loaded into the Q register. The unselected high bytes of Q are not affected by the operation. Byte-width also selects the number of least significant bytes of B that are passed onto the Y bus. Unselected high bytes of the Y bus contain corresponding bytes of Q. Status is defined for selected least-significant bytes only.

LDSTAT-A

Load A Into Status Register

Purpose: To alter the contents of the 32-bit status register.

Status Generated:

Z	N	C	V	L	M	S

Z ← A_{19} if byte width = 00 or 11, else unchanged.

N ← A_{17} if byte width = 00 or 11, else unchanged.

C ← A_{16} if byte width = 00 or 11, else unchanged.

V ← A_{18} if byte width = 00 or 11, else unchanged.

L ← A_{20} if byte width = 00 or 11, else unchanged.

M ← A_{21} if byte width = 00 or 11, else unchanged.

S ← A_{22} if byte width = 00 or 11, else unchanged.

$Opcode_{16}$: 1C

Description: Byte-width selects the number of least-significant bytes of the A bus that are loaded into the STATUS register. Unselected high bytes of the STATUS register are not affected by the operation. Byte-width also selects the number of least-significant bytes of A that are passed onto the Y bus. Unselected high bytes of the Y bus contain corresponding bytes of the STATUS register.

Notes: 1. STATUS <13, 14, 15, 23> cannot be written.

LDSTAT-B

Load B Into Status Register

Purpose: To alter the contents of the 32-bit status register.

Status Generated: | Z | N | C | V | L | M | S |

Z ← B_{19} if byte width = 00 or 11, else unchanged.

N ← B_{17} if byte width = 00 or 11, else unchanged.

C ← B_{16} if byte width = 00 or 11, else unchanged.

V ← B_{18} if byte width = 00 or 11, else unchanged.

L ← B_{20} if byte width = 00 or 11, else unchanged.

M ← B_{21} if byte width = 00 or 11, else unchanged.

S ← B_{22} if byte width = 00 or 11, else unchanged.

$Opcode_{16}$: 1D

Description: Byte-width selects the number of least-significant bytes of the B bus that are loaded into the STATUS register. Unselected high bytes of the STATUS register are not affected by the operation. Byte-width also selects the number of least-significant bytes of B that are passed onto the Y bus. Unselected high bytes of the Y bus contain corresponding bytes of the STATUS register.

Notes: 1. STATUS <13, 14, 15, 23> cannot be written.

Merge A with B

Purpose: To pack a least-significant aligned value in A, into B.

Status Generated:

Z	**N**	C	V	L	M	S

Z Set if selected A bytes are zero. Cleared otherwise.

N Set if selected A bytes are negative. Cleared otherwise.

$Opcode_{16}$: 0E

Description: Byte-width specifies the number of least-significant bytes of A that pass onto the Y bus. The unselected most signifcant bytes contain the corresponding bytes of B. A byte-width of 00 causes all bytes of A to pass onto the Y bus. Status is defined for selected least-significant bytes only.

Merge B with A

Purpose: To pack a least-significant aligned value in B, into A.

Status Generated:

Z	N	C	V	L	M	S

Z Set if selected B bytes are zero. Cleared otherwise.

N Set if selected B bytes are negative. Cleared otherwise.

$Opcode_{16}$: 0F

Description: Byte-width specifies the number of least-significant bytes of B that pass onto the Y bus. The unselected most signifcant bytes contain the corresponding bytes of A. A byte-width of 00 causes all bytes of B to pass onto the Y bus. Status is defined for selected least-significant bytes only.

MPDIVSTEP1 MPDIVSTEP1

Multiprecision Divide Step1

Purpose: The first microcycle of the inner loop for signed or unsigned multiprecision division. A has the least significant part of the divisor (a full word), B has the least significant word of the remainder. Byte-width specifies the number of least significant-bit aligned bytes of dividend in the Q register. The result of this step replaces the least significant word of the remainder.

Status Generated: **Z** N **C** V **L** M S

Z Set if Y equals zero. Cleared otherwise.

C Set if there is an arithmetic carryout during the formation of F. red Cleared otherwise

L Bit shifted out of F moved to L.

Opcode$_{16}$: 52

Description: The full word A is added to B (M=0) or subtracted from B (M=1) to form the value F. F is upshifted one bit, with the sign bit of Q (based on byte-width) shifted in, and output on the Y bus. The bit shifted out of F gets moved to L. Least- significant bytes of Q selected by byte-width are upshifted by one bit with zero fill. Unselected high bytes of Q are unaffected. The M flag is not affected. The MPDIVSTEP1 sets up the L and C flag for use by MPDIVSTEP2 or MPDIVSTEP3.

MPDIVSTEP2 MPDIVSTEP2

Multiprecision Divide Step2

Purpose: The intermediate microcycles of the inner loop for signed or unsigned multiprecision division (not needed for double precision). A has some intermediate word of the divisor. B has the corresponding word of remainder. Byte-width is not used. The result replaces the same word of remainder.

Status Generated: | Z | N | C | V | L | M | S |

Z Set if result is zero. Cleared otherwise.

C Set if there is an arithmetic carryout during the formation of F. Cleared otherwise.

L Bit shifted out of F moved to L.

$Opcode_{16}$: 56

Description: The full word A is added to B (M=0) with carry, or subtracted from B (M=1) with carry to form the value F. F is then upshifted one bit, with the L flag shifted in, and output on the Y bus. The bit shifted out of F gets moved to L. The Q register is not affected or used in this operation. The byte-width input is ignored, and all operations are on a full word. MPDIVSTEP2 also sets up the L and C Flags for use by another MPDIVSTEP2 or by MPDIVSTEP3.

Notes: 1. MACRO/$\overline{\text{MICRO}}$ SEL = 0 selects MACRO CARRY and MACRO LINK. MACRO/$\overline{\text{MICRO}}$ SEL = 1 selects MICRO CARRY and MICRO LINK.

Multiprecision Divide Step3

Purpose: The last microcycle of the inner loop for multiprecision signed division. A has the most significant part of the remainder. Byte-width specifies the number of bytes in the most significant part of the divisor/dividend. The result of this step replaces the most significant part of the remainder.

Status Generated:

Z	N	C	V	L	M	S

Z Set if result is zero. Cleared otherwise.

C Set if there is an arithmetic carryout during the formation of F. Cleared otherwise.

L Bit shifted out of F moved to L.

M Sign A EXNOR bit shifted out of F based on byte-width.

$Opcode_{16}$: 53

Description: Byte-width selects the number of least significant bytes of A that are added to B (M=0) with carry, or subtracted from B (M=1) with carry to form the value F. F is then upshifted by one bit with the L flag shifted in and output on the Y bus. The bit shifted out of F is moved to L. Unselected high bytes of Y pass corresponding bytes of B unaltered. The least significant bit of Q and the M flag are loaded with the new quotient bit, computed as the exclusive NOR of the sign of A and the bit shifted out of F.

Notes: **1**. Q is not upshifted in this step. Upshifting of Q occurs MPDIVSTEP1, leaving Q_0 "vacant"; however, it is not until MPSDIVSTEP3 that the quotient bit is available to load into Q_0.

2. MACRO/$\overline{\text{MICRO}}$ SEL = 0 selects MACRO CARRY and MACRO LINK. MACRO/$\overline{\text{MICRO}}$ SEL = 1 selects MICRO CARRY and MICRO LINK.

Multiprecision Unsigned Divide Step3

Purpose: The last microcycle of the inner loop for multiprecision unsigned division. A has the most significant part of the divisor, B has the most significant part of the remainder. Byte-width specifies the number of bytes in the most significant part of the dividend/divisor. The result of this step replaces the most significant part of the remainder.

Status Generated: | Z | N | C | V | L | M | S |

Z Set if result is zero. Cleared otherwise.

C Set if there is an arithmetic carryout during the formation of F. Cleared otherwise.

L Bit shifted out of F moved to L.

M M EXOR S EXOR C

S Bit shifted out of F moved to S.

$Opcode_{16}$: 57

Description: Byte-width selects the number of least significant bytes of A, zero-extended by one bit, that are subtracted with carry from the same number of least-significant bytes of B, extended by S (M=1) or added with carry to the same number of least-significant bytes of B, extended by S (M=0) to form the value F. F is then upshifted by one bit with the L flag shifted in and output on the Y bus. The bit shifted out of F is moved to L and S. Unselected high bytes of Y contain corresponding bytes of B, unaltered. The least-significant bit of Q is loaded with the new quotient bit, computed as M EXOR S EXOR C. The new quotient bit is also moved to M.

Notes: **1.** Q is not upshifted in this step. Upshifting of Q occurs MPDIVSTEP1, leaving Q_0 "vacant"; however, it is not until MPUDIVSTP3 that the quotient bit is available to load into Q_0.

2. MACRO/$\overline{\text{MICRO}}$ SEL = 0 selects MACRO CARRY and MACRO LINK. MACRO/$\overline{\text{MICRO}}$ SEL = 1 selects MICRO CARRY and MICRO LINK.

NB-0F-SHA NB-0F-SHA

N-Bit Full-Word Shift with ZeroFill

Purpose: To shift the full word A by the number of bits specified by position. Position is interpreted as a signed integer.

Status Generated:

Z	N	C	V	L	M	S

Z Set if Y equals zero. Cleared otherwise.

N Set if Y is negative. Cleared otherwise.

$Opcode_{16}$: 62

Description: This shift instruction performs an upshift when the position input is positive and a downshift when the position input is negative. The shifted input appears on the Y bus. Since position is a 6-bit two's complement integer, the range of shift amounts is [-32..31]. The instruction fills with zero for up and downshift. Shifting by -32 produces zero on the Y bus. Shifting by 0 produces no shift.

Notes: 1. Position comes from the status register (STATUS <5:0>) or the direct input pins (POSITION <5:0>) depending on the I8 input. A HIGH on this input selects the status register.

N-Bit Full-Word Shift with ZeroFill

Purpose: To shift the full word B by the number of bits specified by position. Position is interpreted as a signed integer.

Status Generated:

Z	N	C	V	L	M	S

Z Set if Y equals zero. Cleared otherwise.

N Set if Y is negative. Cleared otherwise.

$Opcode_{16}$: 63

Description: This shift instruction performs an upshift when the position input is positive and a downshift when the position input is negative. The shifted input appears on the Y bus. Since position is a 6-bit two's complement integer, the range of shift amounts is [-32..31]. The instruction fills with zero for up and downshift. Shifting by -32 produces zero on the Y bus. Shifting by 0 produces no shift.

Notes: **1**. Position comes from the status register (STATUS <5:0>) or the direct input pins (POSITION <5:0>) depending on the I8 input. A HIGH on this input selects the status register.

N-Bit Full-Word Shift with Sign Fill

Purpose: To shift the full word A arithmetically by the number of bits specified by position. Position is interpreted as a signed integer.

Status Generated:

Z	N	C	V	L	M	S

Z Set if Y equals zero. Cleared otherwise.

N Set if Y is negative. Cleared otherwise.

$Opcode_{16}$: 60

Description: This shift instruction performs an upshift when the position input is positive and a downshift when the position input is negative. The shifted input appears on the Y bus. Since position is a 6-bit two's complement integer, the range of shift amounts is [-32..31]. The instruction fills with zero for an upshift and with the sign of A (A_{31}) for a downshift. Shifting by -32 produces -1 for negative input, and zero for positive input.

Notes: 1. Position comes from the status register (STATUS <5:0>) or the direct input pins (POSITION <5:0>) depending on the I8 input. A HIGH on this input selects the status register.

N-Bit Full-Word Shift with Sign Fill

Purpose: To shift the full word B arithmetically by the number of bits specified by position. Position is interpreted as a signed integer.

Status Generated:

Z	**N**	C	V	L	M	S

Z Set if Y equals zero. Cleared otherwise.

N Set if Y is negative. Cleared otherwise.

$Opcode_{16}$: 61

Description: This shift instruction performs an upshift when the position input is positive and a downshift when the position input is negative. The shifted input appears on the Y bus. Since position is a 6-bit two's complement integer, the range of shift amounts is [-32..31]. This instruction fills with zero for an upshift and with the sign of B (B_{31}) for a downshift. Shifting by -32 produces -1 for negative input, and zero for positive input.

Notes: **1**. Position comes from the status register (STATUS <5:0>) or the direct input pins (POSITION <5:0>) depending on the I8 input. A HIGH on this input selects the status register.

Rotate Full-Word A by N-Bit

Purpose: To rotate the full word A, by the number of bits specified by position. Position is interpreted as an unsigned 5-bit integer.

Status Generated:

Z	N	C	V	L	M	S

Z Set if Y equals zero. Cleared otherwise.

N Set if Y is negative. Cleared otherwise.

$Opcode_{16}$: 64

Description: The instruction ignores the sign bit of the position input and treats the least significant 5 bits as a positive magnitude for a circular upshift. Thus, a rotate by one results in A_{31} appearing on Y_0, A_0 on Y_1, etc.

Notes: 1. Position comes from the status register (STATUS <5:0>) or the direct input pins (POSITION <5:0>) depending on the I-8 input. A HIGH on this input selects the status register.

Rotate Full-Word B by N-Bit

Purpose: To rotate the full word B, by the number of bits specified by position. Position is interpreted as an unsigned 5-bit integer.

Status Generated:

Z	N	C	V	L	M	S

Z Set if Y equals zero. Cleared otherwise.

N Set if Y is negative. Cleared otherwise.

$Opcode_{16}$: 65

Description: The instruction ignores the sign bit of the position input and treats the least significant 5 bits as a positive magnitude for a circular upshift. Thus, a rotate by one results in B_{31} appearing on Y_0, B_0 on Y_1, etc.

Notes: 1. Position comes from the status register (STATUS <5:0>) or the direct input pins (POSITION <5:0>) depending on the I-8 input. A HIGH on this input selects the status register.

NEG-A NEG-A

Two's Complement of A

Purpose: To produce the two's complement of a number.

Status Generated:

Z	N	C	V	L	M	S

Z Set if result is zero. Cleared otherwise.

N Set if result is negative. Cleared otherwise.

C EX-OR of carry-out & BORROW-MODE.

V Set if overflow occurs. Cleared otherwise.

STATUS <31:24> ← BCD digit borrow.

$Opcode_{16}$: 0A

Description: Byte-width selects the number of least significant bytes of A that are two's complemented and passed onto the Y bus. Unselected high bytes contain corresponding bytes of A unaltered. Status is defined for the selected bytes only.

Notes: **1.** The ALU produces either the carry or the borrow, depending on the state of the BORROW-MODE pin. A HIGH on this pin causes the borrow to be generated (where borrow is the inverse of carry).

2. Status bits 31 to 24 are loaded with inter-nibble borrows. The DIFF-CORR instruction must then be used to form the 10's complement of a BCD number.

Two's Complement of B

Purpose: To produce the two's complement of a number.

Status Generated: | Z | N | C | V | L | M | S |

Z Set if result is zero. Cleared otherwise.

N Set if result is negative. Cleared otherwise.

C EX-OR of carry-out & BORROW-MODE.

V Set if overflow occurs. Cleared otherwise.

STATUS <31:24> ← BCD digit borrow.

$Opcode_{16}$: 0B

Description: Byte-width selects the number of least significant bytes of B that are two's complemented and passed onto the Y bus. Unselected high bytes contain corresponding bytes of B unaltered. Status is defined for the selected bytes only.

Notes: 1. The ALU produces either the carry or the borrow, depending on the state of the BORROW-MODE pin. A HIGH on this pin causes the borrow to be generated (where borrow is the inverse of carry).

2. Status bits 31 to 24 are loaded with inter-nibble borrows. The DIFF-CORR instruction must then be used to form the 10's complement of a BCD number.

NOT-A NOT-A

One's Complement of A

Purpose: To produce the one's complement of a number.

Status Generated:

Z	**N**	C	V	L	M	S

Z Set if result is zero. Cleared otherwise.
N Set if result is negative. Cleared otherwise.

Opcode$_{16}$: 08

Description: Byte-width selects the number of least significant bytes of A that are inverted and passed onto the Y bus. Unselected high bytes contain corresponding bytes of A, unaltered. Status is defined for selected least-significant bytes only.

One's Complement of B

Purpose: To produce the one's complement of a number.

Status Generated:

Z	N	C	V	L	M	S

Z Set if result is zero. Cleared otherwise.

N Set if result is negative. Cleared otherwise.

$Opcode_{16}$: 09

Description: Byte-width selects the number of least significant bytes of B that are inverted and passed onto the Y bus. Unselected high bytes contain corresponding bytes of B, unaltered. Status is defined for selected least-significant bytes only.

Complement Field in A, Merge with B

Purpose: To invert a field in A.

Status Generated: | **Z** | **N** | C | V | L | M | S |

Z Set if resulting field is zero. Cleared otherwise.
N Set if Y_{31} is a one. Cleared otherwise.

Opcode$_{16}$: 70

Description: When the position input is positive, the LSB-aligned field in A is upshifted to align it with the field in B starting at the bit specified by position. When the position input is negative, the field in A starting at the bit specified by the two's complement of position is rotated down to align it with the LSB aligned field in B. The field in A is then inverted up to the width specified by the width input, and the resulting field appears on the Y bus, LSB-aligned in the case that position was negative, or starting at the bit specified by position, in the case that position was positive. The remaining bits of Y contain corresponding bits of B unaltered. The Z flag is set according to the contents of the selected field. However, the N flag is set according to the most significant bit of Y.

Notes: **1**. A HIGH on the I-8 input selects the status register (STATUS <5:0>) as the source of position; a LOW selects the direct position <5:0> inputs pins.

2. A HIGH on the I-7 input selects the status register (STATUS <12:8>) as the source of width; a LOW selects the direct width <4:0> inputs pins.

3. For position <0, A is rotated rather than shifted down. Therefore, if the sum of position and width exceeds 32, bits rotated around from the low-order end of A will be selected.

Complement Field in A, Merge with B

Notes: **4.** For p≥0, a width of 0 specifies the entire field to the left of position.
e.g., p=02 w=0

```
A = 0001 0010 0011 0100 0101 0110 0111 1000 = 12345678
A1= 0100 1000 1101 0001 0101 1001 1110 00xx
B = 1001 1010 1011 1100 1101 1110 1111 0010 = 9ABCDEF2
Y = 1011 0111 0010 1110 1010 0110 0001 1110 = B72EA61E
```

For p<0, a width of 0 specifies the entire 32 bit field.
e.g., p=3E w=0

```
A = 0001 0010 0011 0100 0101 0110 0111 1000 = 12345678
A1= 0000 0100 1000 1101 0001 0101 1001 1110
B = 1001 1010 1011 1100 1101 1110 1111 0000 = 9ABCDEF0
Y = 1111 1011 0111 0010 1110 1010 0110 0001 = FB72EA61
```

Merge Complement of Field in A with B

Purpose: To invert a field in A starting at the bit specified by position, & having the width specified by the width inputs, & merging with B.

Status Generated:

Z	N	C	V	L	M	S

Z Set if resulting field is zero. Cleared otherwise.
N Set if Y_{31} is a one. Cleared otherwise.

$Opcode_{16}$: 71

Description: Position is treated as a 5 bit unsigned number. A logical not operation is performed on the A field starting at the bit specified by the position input and of the specified width. The resulting field appears on the Y bus at the same position. The remaining bits of Y contain corresponding bits of B unaltered. The Z flag is set according to the contents of the selected field. However, the N flag is set according to the most significant bit of Y.

Notes: 1. A HIGH on the I-8 input selects the status register (STATUS <5:0>) as the source of position; a LOW selects the direct position <5:0> inputs pins.

2. A HIGH on the I-7 input selects the status register (STATUS <12:8>) as the source of width; a LOW selects the direct width <4:0> inputs pins.

3. A width of 0 specifies the entire field to the left of position.

Invert Field in B

Purpose: To invert a field in B starting at the bit specified by position, and having the width specified by the width inputs.

Status Generated:

Z	**N**	C	V	L	M	S

Z Set if resulting field is zero. Cleared otherwise.
N Set if Y_{31} is a one. Cleared otherwise.

$Opcode_{16}$: 6E

Description: The least-significant five bits of position, treated as an unsigned magnitude, give the start bit of a field in B that is to be inverted. The width of the field is specified by the width input. Other bits on the Y bus contain corresponding bits of B unaltered. The Z flag is set depending on the state of the field. However the N flag is set according to the most-significant bit of Y.

Notes: **1.** A HIGH on the I-8 input selects the status register (STATUS <5:0>) as the source of position; a LOW selects the direct position <5:0> inputs pins.

2. A HIGH on the I-7 input selects the status register (STATUS <12:8>) as the source of width; a LOW selects the direct width <4:0> inputs pins.

3. A width of 0 specifies the entire field to the left of position.

Logical Or

Purpose: To compute a logical OR function.

Status Generated:

Z	N	C	V	L	M	S

Z Set if result of OR operation is zero. Cleared otherwise.
N Set if result is negative. Cleared otherwise.

$Opcode_{16}$: 3E

Description: Byte-width selects the number of least-significant bytes of A and B that participate in a logical OR operation whose result is output on the Y bus. Unselected high bytes of the Y bus contain corresponding bytes of B, unaltered. Status is defined for the selected least-significant bytes only.

Or Field in A with B

Purpose: To perform a logical OR operation between a field in A and a field in B.

Status Generated:

Z	N	C	V	L	M	S

Z Set if result of OR operation is zero. Cleared otherwise.
N Set if Y_{31} is a one. Cleared otherwise.

$Opcode_{16}$: 74

Description: When the position input is positive, the LSB-aligned field in A is upshifted to align it with the field in B starting at the bit specified by position. When the position input is negative, the field in A starting at the bit specified by the two's complement of position is rotated down to align it with the LSB-aligned field in B. The logical OR operation then takes place between the field in A and the field in B, up to the width specified by the width field and the resulting field appears on the Y bus, LSB-aligned in the case that position was negative, or starting at the bit specified by position, in the case that position was positive. The remaining bits of Y contain corresponding bits of B unaltered. The Z flag is set according to the contents of the selected field. However, the N flag is set according to the most significant bit of Y.

Notes: **1.** A HIGH on the I-8 input selects the status register (STATUS <5:0>) as the source of position; a LOW selects the direct position <5:0> input pins.

2. A HIGH on the I-7 input selects the status register (STATUS <12:8>) as the source of width; a LOW selects the direct width <4:0> input pins.

3. For position <0, A is rotated rather than shifted down. Therefore, if the sum of position and width exceeds 32, bits rotated around from the low-order end of A will be selected.

Or Field in A with B

Notes: 4. For p≥0, a width of 0 specifies the entire field to the left of position.
e.g., p=03; w=0

```
A = 0111 0010 0001 0101 0001 1000 0000 1111 = 7215180F
A¹= 1001 0000 1010 1000 1100 0000 0111 1xxx
B = 1000 1111 1100 1110 0001 0011 0111 0010 = 8FCE1372
Y = 1001 1111 1110 1110 1101 0011 0111 1010 = 9FEED37A
```

For p<0, a width of 0 specified the entire 32 bit field.
e.g., p=3C; w=0

```
A = 0111 0001 0111 0010 0011 1111 1110 1101 = 71723FED
A¹= 1101 0111 0001 0111 0010 0011 1111 1110
B = 1000 0010 1100 1110 0001 0101 0000 1001 = 82CE1509
Y = 1101 0111 1101 1111 0011 0111 1111 1111 = D7DF37FF
```

OR Aligned Fields in A and B

Purpose: To perform a logical OR operation between a field in A and a field in B that start at the same position.

Status Generated:

Z	N	C	V	L	M	S

Z Set if result of OR operation is zero. Cleared otherwise.

N Set if Y_{31} is a one. Cleared otherwise.

$Opcode_{16}$: 75

Description: A logical OR operation is performed between the fields in A and B starting at the bit specified by position and of the specified width. Position is treated as a 5 bit unsigned input. The MSB of position is ignored. The resulting field appears on the Y bus at the same position. The remaining bits of Y contain corresponding bits of B unaltered. The Z flag is set according to the contents of the selected field. However, the N flag is set according to the most significant bit of Y.

Notes: **1**. A HIGH on the I8 input selects the status register (STATUS <5:0>) as the source of position; a LOW selects the direct position <5:0> input pins.

2. A HIGH on the I7 input selects the status register (STATUS <12:8>) as the source of width; a LOW selects the direct width <4:0> input pins.

3. A width of 0 specifies the entire field to the left of position.

PASS-MASK PASS-MASK

Pass Mask

Purpose: To generate field mask patterns useful for masking, field extraction, exercising other logic, diagnostics, and bit- toggling in general.

Status Generated:

Z	N	C	V	L	M	S

Opcode$_{16}$: 7F

Description: The ALU puts onto the Y bus a mask of 1's in the field starting at the bit specified by the least significant five bits of the position input, and spanning a width specified by the the width input. The sign-bit of position, when true, inverts the entire mask (all 32 bits of the Y bus). There are no side-effects on the status register.

Notes: **1**. A HIGH on the I8 input selects the status register (STATUS <5:0>) as the source for position; otherwise position comes from the direct position input pins. (POSITION <5:0>).

2. A HIGH on the I7 input selects the status register (STATUS <12:8>) as the source for width; otherwise width comes from the direct width input pins. (WIDTH <4:0>).

3. A width of 0 specifies the entire field to the left of position.

Pass Q Register to Y

Purpose: To move the quotient or least-significant product word to an external destination.

Status Generated:

Z	N	C	V	L	M	S

Opcode$_{16}$: 05

Description: Byte-width selects the number of least significant bytes of the Q register that are passed onto the Y bus. Unselected high order bytes are taken from the B input. The Q and status registers are not affected by this operation.

Pass Status Register to Y

Purpose: To move the status register to an external destination.

Status Generated:

Z	N	C	V	L	M	S

Opcode$_{16}$: 04

Description: Byte-width selects the number of least significant bytes of the status register that are passed onto the Y bus. Unselected high order bytes are taken from the B input. The contents of the status register are not altered by this operation.

Notes: **1**. Status <13> is C + Z in the borrow mode. (BORROW - MODE = high)

Pass Field in A to Y, Merge with B, Unaligned

Purpose: To merge a field in A, with B and pass to Y.

Status Generated:

Z	N	C	V	L	M	S

Z Set if merged field is zero. Cleared otherwise.

N Set if Y_{31} is a one. Cleared otherwise.

$Opcode_{16}$: 72

Description: When the position input is positive, the LSB-aligned field in A is upshifted to align it with the field in B starting at the bit specified by position. When the position input is negative, the field in A starting at the bit specified by the two's complement of position is rotated down to align it with the LSB-aligned field in B. The A field is then substituted into the B word starting at the same position and for the width specified by the width input and the resulting field appears on the Y bus, LSB-aligned in the case that position was negative, or starting at the bit specified by position, in the case that position was positive. The remaining bits of Y contain corresponding bits of B unaltered. The Z flag is set according to the contents of the selected field. However, the N flag is set according to the most significant bit of Y.

Notes: **1**. A HIGH on the I8 input selects the status register (STATUS <5:0>) as the source of position; a LOW selects the direct position <5:0> input pins.

2. A HIGH on the I7 input selects the status register (STATUS <12:8>) as the source of width; a LOW selects the direct width <4:0> input pins.

3. For position <0, A is rotated rather than shifted down. Therefore, if the sum of position and width exceeds 32, bits rotated around from the low-order end of A will be selected.

Pass Field in A to Y, Merge with B, Unaligned

Notes: **4.** For p>0, a width of 0 specifies the entire field to the left of position.
e.g., p=03; w=0

```
A  = 0111 0010 0001 0101 0001 1000 0000 0101 = 7215180
A1 = 1001 0000 1010 1000 1100 0000 0010 1xxx
B  = 1000 1111 1100 1110 0001 0011 0111 0010 = 8FCE1372
Y  = 1001 0000 1010 1000 1100 0000 0010 1010 = 90A8C02A
```

For p<0, a width of 0 specifies the entire 32 bit field.
e.g., p=3C; w=0

```
A  = 0111 0001 0111 0010 0011 1111 1110 1101 = 71723FED
A1 = 1101 0111 0001 0111 0010 0011 1111 1110
B  = 1000 0010 1100 1110 0001 0101 0000 1001 = 82CE1509
Y  = 1101 0111 0001 0111 0010 0011 1111 1110 = D71723FE
```

PASSF-AL-A

Pass Field in A to Y Merged with B

Purpose: To merge a field from A into B and pass to Y.

Status Generated:

Z	N	C	V	L	M	S

Z Set if merged field is zero. Cleared otherwise.

N Set if Y_{31} is a one. Cleared otherwise.

Opcode$_{16}$: 73

Description: The least significant five bits of the position input, treated as an unsigned magnitude, give the start bit of a field in A that is to be merged into B starting at the same position. The width of the field is specified by the width inputs. Other bits on the Y bus contain corresponding bits of B unaltered. The Z flag is set depending on the state of the field.

Notes: **1**. A HIGH on the I8 input selects the status register (STATUS <5:0>) as the source for position; otherwise position comes from the direct position input pins (POSITION <5:0>).

2. A HIGH on the I7 input selects the status register (STATUS <12:8>) as the source for width; otherwise width comes from the direct width input pins (WIDTH <4:0>).

3. A width of 0 specifies the entire field to the left of position.

Pass Field

Purpose: To test a field in B starting at the bit specified by position and having the width specified by width.

Status Generated:

Z	N	C	V	L	M	S

Z Set if field being tested is zero. Cleared otherwise.

N Set if Y_{31} is a one. Cleared otherwise.

$Opcode_{16}$: 6F

Description: The least-significant five bits of the position input, treated as an unsigned magnitude, give the start bit of a field in B that is to be tested for zero. The width of the field is specified by the width input. Other bits on the Y bus contain corresponding bits of B unaltered. The Z flag is set depending on the state of the field.

Notes: **1.** A HIGH on the I8 input selects the status register (STATUS <5:0>) as the source for position; otherwise position comes from the direct position input pins (POSITION <5:0>).

2. A HIGH on the I7 input selects the status register (STATUS <12:8>) as the source for width; otherwise width comes from the direct width input pins (WIDTH <4:0>).

3. A width of 0 selects the entire field to the left of position.

Prioritize A

Purpose: To count the number of leading zeroes in an unsigned integer prior to normalization.

Status Generated:

Z	N	C	V	L	M	S

Z Set if selected bytes equal zero. Cleared otherwise.

Opcode16: 0C

Description: The prioritize operation on A for a given byte-width gives the number of bits that A must be shifted up to bring the most significant 1 in A up to the sign position of the most significant selected byte, except when A contains zero in the selected bytes. Unselected bytes of A are ignored in this operation. In the case where selected bytes of A contain zero, the priority operation generates zero on the Y bus and sets the Z flag of the status register. The priority is also loaded in STATUS <7:0> as a side-effect, for use in a subsequent shift. The most-significant 3 bytes of the Y bus and the most significant 3 bits of the position register are always forced to zero.

Prioritize B

Purpose: To count the number of leading zeroes in an unsigned integer prior to normalization.

Status Generated:

Z	N	C	V	L	M	S

Z Set if selected bytes equal zero. Cleared otherwise.

Opcode$_{16}$: 0D

Description: The prioritize operation on B for a given byte-width gives the number of bits that B must be shifted up to bring the most significant 1 in B up to the sign position of the most significant selected byte, except when B contains zero in the selected bytes. Unselected bytes of B are ignored in this operation. In the case where selected bytes of B contain zero, the priority operation generates zero on the Y bus and sets the Z flag of the status register. The priority is also loaded in STATUS <7:0> as a side-effect, for use in a subsequent shift. The most-significant 3 bytes of the Y bus and the most-significant 3 bits of the position register are always forced to zero.

Quotient Correct Step

Purpose: To correct the quotient after a signed single precision divide. B has the quotient, which has in a previous cycle been read out of the Q register. Byte-width gives the number of bytes in the quotient. The Z, N, M and S flags are setup by the prior divsteps.

Status Generated:

Z	N	**C**	**V**	L	M	S

C Set if there is a carryout during the forming of F. Cleared otherwise.

V Set if overflow/underflow occurs. Cleared otherwise.

$Opcode_{16}$: 59

Description: The value F is computed as follows:

If Z EXOR N EXOR S=1 and M=0, then F=B - 1

If Z EXOR N EXOR S=1 and M=1. then F=B + 1

If Z EXOR N EXOR S=0, then F=B;

F is output on the Y bus and replaces the quotient. Unselected high bytes of Y contain corresponding bytes of B unaltered. Status is defined according to byte-width.

Notes: **1.** N holds the partial remainder sign and S holds the initial dividend sign (0 for unsigned division).

2. N EXOR S gives the sign difference between the initial dividend and the final remainder. For unsigned division, Z will be 0 and N will always be 1. (If N was 0 at the end of the unsigned divide, the algorithm terminates). For signed division, the only case where Z=1 is the special case where the partial remainder went to zero at some intermediate divide step. In that case, N EXOR S will be zero (see SDIVLAST2). In either situation, the M flag gives the direction of the correction.

Remainder Correct Step

Purpose: To correct the remainder after a signed or unsigned single precision divide. A has the divisor, B the remainder, and byte-width gives the number of bytes in the remainder/divisor. The Z, N, M and S flags are the setup by the prior divsteps.

Status Generated:

Z	N	**C**	V	L	M	S

C Set if there is a carryout during the forming of F. Cleared otherwise.

$Opcode_{16}$: 58

Description: The value F is computed as follows:

If Z EXOR N EXOR S=1 and M=0, then F=B + A

If Z EXOR N EXOR S=1 and M=1. then F=B - A

If Z EXOR N EXOR S=0, then F=B;

F is output on the Y bus and replaces the remainder. Unselected high bytes of Y contain corresponding bytes of B unaltered. Status is defined according to byte-width.

Notes: **1**. N holds the partial remainder sign and S holds the initial dividend sign (0 for unsigned division).

2. N EXOR S gives the sign difference between the initial dividend and the final remainder. For unsigned division, Z will be 0 and N will always be 1. (If N was 0 at the end of the unsigned divide, the algorithm terminates.) For signed division, the only case where Z=1 is the special case where the partial remainder went to zero at some intermediate divide step. In that case, N EXOR S will be zero (see SDIVLAST2). In either situation, the M flag gives the direction of the correction.

RSTBIT-A

Reset Bit in A

Purpose: Bit-toggling primitives that reset any bit of the full word input A.

Status Generated:

Z	**N**	C	V	L	M	S

Z Always set.

N Set if Y_{31} is a one. Cleared otherwise.

Opcode$_{16}$: 6A

Description: The least significant five bits of position, taken as an unsigned magnitude, specify the bit position of A which is reset before the full word A is output on the Y bus. The most significant bit of position is ignored. The Z flag is set.

Notes: 1. Position comes from the status register (STATUS <5:0>) or the direct input (POSITION <5:0>) depending on the I8 input. A HIGH on this input selects the status register.

Reset Bit in B

Purpose: Bit-toggling primitives that reset any bit of the full word input B.

Status Generated: **Z** **N** C V L M S

Z Always set.

N Set if Y_{31} is a one. Cleared otherwise.

$Opcode_{16}$: 6B

Description: The least significant five bits of position, taken as an unsigned magnitude, specify the bit position of B which is reset before the full word B is output on the Y bus. The most significant bit of position is ignored. The Z flag is set.

Notes: **1**. Position comes from the status register (STATUS <5:0>) or the direct input (POSITION <5:0>) depending on the I8 input. A HIGH on this input selects the status register.

Reset Status Register Bit

Purpose: Bit-toggling primitives for the status register. The bit specified by position is reset. There are no side-effects on the ALU Status flags C, Z, N, V, etc.

Status Generated:

Z	N	C	V	L	M	S

Opcode$_{16}$: 6D

Description: The least significant five bits of position, taken as an unsigned magnitude, specify the bit of the status register that is reset. The new value of the status register also appears on the Y bus. Other bits of the status register are unaffected. The most significant bit of position is ignored.

Notes: **1**. Position comes from the status register (STATUS <5:0>) or the direct input (POSITION <5:0>) depending on the I8 input. A HIGH on this line selects the status register.

2. Bits <13,14,15,23> are read-only.

Signed Divide First Step

Purpose: To set the M and S flags of the status register in preparation for the divide step. The quotient register is assumed to contain the dividend, A has the divisor, and B has the remainder, which is the sign extension of the dividend. The result is assumed to replace the remainder.

Status Generated:

Z	N	C	V	L	M	S

Z Set if result is zero. Cleared otherwise.

N Set if result is negative. Cleared otherwise.

L Sign bit of B moved to L.

M Sign bit A EXNOR bit shifted out of B.

S Sign bit of B moved to S.

$Opcode_{16}$: 4E

Description: Byte-width selects the number of least significant bytes of B that are upshifted by one bit, with the sign bit of Q shifted into the least significant position, and output on the Y bus. Unselected high bytes contain corresponding bytes of B unaltered. Byte-width also selects the number of least significant bytes of Q that are shifted up by one bit. The exclusive NOR of the signbit of A and the bit shifted out of B is shifted into the least significant bit of Q, and the M flag. Unselected high bytes of Q are unaffected. The S flag and the L flag are loaded with the bit shifted out of B. The sign of the dividend remains in the S flag until the end of the signed divide algorithm.

Notes: 1. At this stage, the least-significant bit of Q is not useful, and will not appear in the final result.

Signed Divide Last (Conditional)

Purpose: This instruction terminates the calculation of quotient and remainder. A has the divisor, B the partial remainder. After this step, the quotient register contains the complete quotient, which is algebraically correct. That is, quotient x divisor + remainder - dividend.

Status Generated:

Z	N	C	V	L	M	S

Z Set if result is zero. Cleared otherwise.

N Set if result is negative. Cleared otherwise.

C Arithmetic carryout of F.

M Sign bit F EXNOR Sign bit A.

$Opcode_{16}$: 51

Description: Byte-width selects the number of least significant bytes of A that are added to B (M=0) or subtracted from B (M=1) to form the value F. F is then output on the Y bus. Unselected high bytes of Y pass corresponding bytes of B unaltered. Byte- width also selects the number of least significant bytes of Q that are shifted up by one bit. The exclusive NOR of the signbit of A and the signbit of F (the new quotient bit) is loaded into M. The least significant bit of Q gets loaded with a 1. Unselected high bytes of Q are unaffected.

Notes: 1. The quotient and remainder may need correction after this step. This is a consequence of the nonrestoring algorithm. At each step, the algorithm computes a new quotient bit and partial remainder while simultaneously correcting the previous partial remainder if it was wrong. Since correction lags computation of the quotient bits by one step, after the last step there is still one correction needed.

In general, division is the process of reducing the partial remainder towards zero. If the remainder is not exactly zero, it should be the same sign as the dividend. When the signs of remainder and initial dividend differ, then correction is required. For positive quotients, the correction is performed by subtracting one from the quotient and adding the divisor back to the remainder. For negative quotients, the correction is performed by adding one to the quotient, and subtracting the divisor from the remainder.

Signed Divide Last (Conditional)

Notes: See SDIVLAST2, REMCORR, and QUOCORR for further details on correction.

2. The quotient after SDIVLAST1 is always odd as a 1 is forced into the least-significant bit. This helps provide a uniform interface to the correction steps.

Signed Divide Last Step

Purpose: To check for a special case arising with negative dividends when the partial remainder goes to zero at an intermediate step of the division. Used after SDIVLAST1 and before the correction steps. A has the divisor, and B the remainder after SDIVLAST1. The result of this step is not written back to the remainder.

Status Generated:

Z	N	C	V	L	M	S

Z Set if result is zero and original S=1. Cleared otherwise.

Opcode$_{16}$: 5A

Description: The divisor sign is reconstructed by extracting the sign bit from the quotient register and exclusive ORing it with the dividend sign stored in S. Byte-width selects the number of least significant bytes of A that are subtracted from B (divisor sign=1) or added to B (divisor sign=0). The result appears on the Y bus. Unselected high bytes of Y pass corresponding bytes of B unaltered; however, this is irrelevant, since the result is not expected to be used. If the original dividend was negative (S=1) then the Z flag is updated in the usual way; otherwise the Z flag is loaded with 0. The Z flag is the only useful result of this step.

Notes: **1**. While it is always true that correction is needed when the signs of remainder and initial dividend differ, there is a special case when the signs of dividend and remainder are the same, and correction is still needed. The need for this correction is not a result of the nonrestoring algorithm, but rather a consequence of the difficulty of detecting early termination of the divide algorithm and the asymmetry of the two's complement number system, in which zero appears positive, and there is one more negative number than positive.

When the dividend is perfectly divisible by the remainder and the quotient is not odd, then the partial remainder becomes zero at an intermediate stage of the division. This situation is difficult to detect because the partial remainder spans a varying number of bits of Q and the remainder register. If this situation could be detected and the algorithm terminated early there would be no need for SDIVLAST2. Also, in the case of positive dividends, there is no harm done in not terminating the algorithm early because the zero partial remainder appears positive, and causes the desired effect (a string of 0 quotient bits, or a string of 1 quotient bits connected eventually by a resorting add of 1). However, in the case of negative dividends, the

Signed Divide Last Step

Notes: algorithms see a change of sign (zero appears positive) and attempts to compensate for it. The net result is that the final remainder has the same magnitude as the divisor, and the same sign as the dividend (negative); SDIVLAST2 tests for exactly this situation, leaving its result in the Z flag. The remainder must then be corrected by reducing it to zero, and the quotient by increasing its magnitude by one to keep the product of remainder and quotient constant.

2. If the dividend is known to be positive, SDIVLAST2 may be omitted from the microcode.

Signed Divide Step

Purpose: This instruction constitutes the inner loop of single precision signed division. The quotient register contains the dividend, A has the divisor, and B has the partial remainder. The result, Y, is assumed to replace the partial remainder.

Status Generated: **Z N C** V **L M** S

Z Set if result if zero. Cleared otherwise.

N Set if result is negative. Cleared otherwise.

C Arithmetic carryout of F.

L Bit shifted out of F moved to L.

M Sign bit A EXNOR bit shifted out of F.

$Opcode_{16}$: 50

Description: Byte-width selects the number of least significant bytes of A that are added to B (M=0) or subtracted from B (M=1) to form the value F. F is then upshifted by one bit, with the signbit of Q shifted into the least significant bit position, and output on the Y bus. Unselected high bytes of Y pass corresponding bytes of B unaltered. Byte-width also selects the number of least significant bytes of Q that are shifted up by one bit. The exclusive NOR of the signbit of A and the bit shifted out of F (the new quotient bit) is shifted into the least significant bit of Q, and loaded into M flag. Unselected high bytes of Q are unaffected. The S flag, assumed to hold the dividend sign, is not altered by the operation.

Set Bit in A

Purpose: Bit-toggling primitives that set any bit of the full word input A.

Status Generated:

Z	N	C	V	L	M	S

Z Always reset.

N Set if Y_{31} is a one. Cleared otherwise.

$Opcode_{16}$: 68

Description: The least significant five bits of position, taken as an unsigned magnitude, specify the bit position of A which is set before the full word A is output on the Y bus. The most significant bit of position is ignored. The Z flag is reset.

Notes: 1. Position comes from the status register (STATUS <5:0>) or the direct input (POSITION <5:0>) depending on the I8 input. A HIGH on this input selects the status register.

Set Bit in B

Purpose: Bit-toggling primitives that set any bit of the full word input B.

Status Generated:

Z	N	C	V	L	M	S

Z Always reset.

N Set if Y_{31} is a one. Cleared otherwise.

Opcode$_{16}$: 69

Description: The least significant five bits of position, taken as an unsigned magnitude, specify the bit position of B which is set before the full word B is output on the Y bus. The most significant bit of position is ignored. The Z flag is reset by setbit.

Notes: **1**. Position comes from the status register (STATUS <5:0>) or the direct input (POSITION <5:0>) depending on the I8 input. A HIGH on this input selects the status register.

Set Status Register Bit

Purpose: Bit-toggling primitives for the status register. The bit specified by position is set. There are no side-effects on the ALU Status flags C, Z, N, V, etc.

Status Generated:

Z	N	C	V	L	M	S

Opcode$_{16}$: 6C

Description: The least significant 5 bits of position, taken as an unsigned magnitude, specify the bit of the status register that is set. The new value of the status register also appears on the Y bus. Other bits of the status register are unaffected. The most significant bit of position is ignored.

Notes: 1. Position comes from the status register (STATUS <5:0>) or the direct input (POSITION <5:0>) depending on the state of the input I8. A HIGH on this line selects the status register.

Pass Sign of B to Y-Bus

Purpose: To perform multiple precision sign-extends.

Status Generated:

Z	N	C	V	L	M	S

Z Set if N equals 0. Cleared if N equals 1.

Opcode$_{16}$: 3D

Description: Byte-width selects the number of least-significant bytes of Y whose bits are forced to a 1 if N=1, or 0 if N=0. The state of N is checked in the status register. Unselected high- order bytes are taken from the B input. Status is defined for selected least-significant bytes only.

Sign Extend A

Purpose: Type conversion of signed integers, to single or double words; data movement.

Status Generated:

Z	N	C	V	L	M	S

Z Set if Y bus equals zero. Cleared otherwise.

N Set if Y_{31} equals one. Cleared otherwise.

$Opcode_{16}$: 02

Description: Byte-width specifies the number of least-significant bytes of A that pass onto the Y bus. The unselected most significant bytes contain the sign extension of the least significant part of the Y bus.

Notes: 1. To convert a signed byte to a double word (for example), use SIGN-EXTA on the byte with byte-width = 01 to obtain the least significant word followed by the SIGN instruction with byte-width = 00 to get the most significant word.

Sign Extend B

Purpose: Type conversion of signed integers, to single or double words; data movement.

Status Generated:

Z	N	C	V	L	M	S

Z Set if Y bus equals zero. Cleared otherwise.

N Set if Y_{31} equals one. Cleared otherwise.

$Opcode_{16}$: 03

Description: Byte-width specifies the number of least-significant bytes of B that pass onto the Y bus. The unselected most significant bytes contain the sign extension of the least significant part of the Y bus.

Notes: 1. To convert a signed byte to a double word (for example), use SIGN-EXTB on the byte with byte-width = 01 to obtain the least significant word followed by the SIGN instruction with byte-width = 00 to get the most significant word.

Signed Multiply First

Purpose: To initialize the partial product register and the multiply flag (M) prior to a single precision signed multiply, and also perform the first iteration of the multiply algorithm, yielding two new product bits. A has the multiplicand, whose width is specified by byte-width. Q has been loaded with the multiplier. The result replaces the partial product.

Status Generated:

Z	N	C	V	L	**M**	S

M Q_1 moved to M.

Opcode$_{16}$: 5F

Description: The ALU is internally extended by two bits at the appropriate byte boundary depending on byte-width. Selected bytes of B are internally zeroed. A is sign extended across the two extra bits to form A*. Next, based on the values of Q_1 and Q_0, the value F* is computed;

If $Q_1Q_0 = 00$, F* = 0;

If $Q_1Q_0 = 01$, F* = A*;

If $Q_1Q_0 = 10$, F* = 2's complement of 2A*;

If $Q_1Q_0 = 01$, F* = 2's complement of A*;

F* shifted down 2 bits is output on the Y bus. Unselected high bytes of Y contain corresponding bytes of B unaltered. Selected bytes of Q are down shifted by 2 bits with F_1F_0 being the fill bits. Unselected high bytes of Q are unaffected.

Notes: **1**. The partial product register should be read in over B if the unselected high bytes of that register are to be unaffected by the writing of the result back into the partial product register.

2. The passing property of the ALU is unaffected by the extra bits of precision.

3. If a multiply-accumulate is required, then the product register will start off with some value other than zero. In such cases, SMULFIRST should not be used. Instead, first reset M using RSTBIT-STAT(p=21) and then use SMULSTEP instead of SMULFIRST.

Signed Multiply Step

Purpose: This step contributes the inner loop for single precision signed multiplication. Each iteration of the loop yields two new products bits. A has the multiplicand, B has the partial product, and byte-width specifies the number of bytes selected. Q contains the multiplier and some product bits. The result of this step replaces the partial product.

Status Generated:

Z	N	C	V	L	**M**	S

M Q_1 moved to M.

Opcode$_{16}$: 5E

Description: The ALU is internally extended by two bits at the appropriate byte boundary depending on byte-width. A is sign extended across the two bits to form A*. B is sign extended across the two bits to form B*. Next, based on the values of Q_1 Q_0 and M, the value of F* is computed;

If $Q_1Q_0M = 000$ or 111, F* = B*;

If $Q_1Q_0M = 001$ or 010, F* = B* + A*;

If $Q_1Q_0M = 101$ or 110, F* = B* - A*

If $Q_1Q_0M = 011$, F* = B* + 2A*;

If $Q_1Q_0M = 100$, F* = B* - 2A*;

F* shifted down 2 bits is output on the Y bus. Unselected high bytes of Y contain corresponding bytes of B unaltered. Selected bytes of Q are down shifted by 2 bits with F_1F_0 being the fill bits. Unselected high bytes of Q are unaffected.

Subtract B From A

Purpose: Subtraction in 2's complement.

Status Generated: | **Z** | **N** | **C** | **V** | L | M | S |

Z Set if result is zero. Cleared otherwise.

N Set if result is negative. Cleared otherwise.

C EX-OR of carryout and BORROW-MODE.

V Set if underflow occurs. Cleared otherwise.

STATUS <31:24>← BCD nibble borrows.

Opcode$_{16}$: 44

Description: Byte-width selects the number of least-significant bytes of A and B that participate in a subtraction operation whose result is output on the Y bus. Unselected high bytes of the Y bus contain corresponding bytes of B, unaltered. Status is defined for the selected least-significant bytes only.

Notes: **1**. Both carry and borrow arithmetic are supported under control of the BORROW-MODE pin. A HIGH on this pin causes the borrow (the complement of the carry) to be stored on subtract.

2. Status bits 31 to 24 are loaded with inter-nibble BCD borrows for SUB. The DIFF-CORR instructions must then be used to subtract BCD numbers.

Subtract B From A with Carry/Borrow

Purpose: To perform arithmetic on integers longer than 32 bits. Also, provides the hooks for emmulation of multiprecision macroinstructions via the MACRO CARRY input.

Status Generated:

Z	N	C	V	L	M	S

Z Set if result is zero. Cleared otherwise.

N Set if result is negative. Cleared otherwise.

C EX-OR of carryout and BORROW-MODE.

V Set if underflow occurs. Cleared otherwise.

STATUS <31:24> ← BCD nibble borrows.

$Opcode_{16}$: 45

Description: Byte-width selects the number of least significant bytes of A and B that participate in a multiprecision subtraction operation whose result is output on the Y bus. Unselected high bytes of the Y bus contain corresponding bytes of B, unaltered. Status is defined for the selected least- significant bytes only.

Notes: **1**. Both carry and borrow arithmetic are supported under control of the BORROW-MODE pin. A HIGH on this pin causes the borrow (the complement of the carry) to be stored on subtract.

2. A HIGH on the MACRO input selects the MACRO-CARRY input instead of the C flag of the (micro) Status register as the carry-in for SUBC. This allows an external carry flip flop to be selected for macroinstruction emulation.

3. Whatever the choice of carry-in based on (2) above, for SUBC, it is inverted before being input to the adder if BORROW-MODE is HIGH. Therefore, in the borrow mode, both the internal and external carry store the borrow (not the carry).

Subtract B From A with Carry/Borrow

Notes: **4**. Status bits 31 to 24 are loaded with inter-nibble BCD borrows for SUBC. The DIFF-CORR instructions must then be used to subtract BCD numbers longer than eight digits.

5. When performing multi-precision BCD arithmetic, a DIFF-CORR must be performed after each instruction.

Subtract A From B

Purpose: Subtraction in 2's complement.

Status Generated:

Z	N	C	V	L	M	S

Z Set if result is zero. Cleared otherwise.

N Set if result is negative. Cleared otherwise.

C EX-OR of carryout and BORROW-MODE.

V Set if underflow occurs. Cleared otherwise.

STATUS <31:24> ← BCD nibble borrows.

Opcode$_{16}$: 46

Description: Byte-width selects the number of least-significant bytes of A and B that participate in a subtraction operation whose result is output on the Y bus. Unselected high bytes of the Y bus contain corresponding bytes of B, unaltered. Status is defined for the selected least-significant bytes only.

Notes: **1**. Both carry and borrow arithmetic are supported under control of the BORROW-MODE pin. A HIGH on this pin causes the borrow (the complement of the carry) to be stored on subtract.

2. Status bits 31 and 24 are loaded with inter-nibble BCD borrows for SUBR. The DIFF-CORR instructions may then be used to subtract BCD numbers.

SUBRC SUBRC

Subtract A From B with Carry/Borrow

Purpose: To perform arithmetic on integers longer than 32 bits. Also, provides the hooks for emulation of multiprecision macroinstructions via the MACRO CARRY input.

Status Generated:

Z	N	C	V	L	M	S

Z Set if result is zero. Cleared otherwise.

N Set if result is negative. Cleared otherwise.

C EX-OR of carryout and BORROW-MODE.

V Set if underflow occurs. Cleared otherwise.

STATUS <31:24> ← BCD nibble borrows.

$Opcode_{16}$: 47

Description: Byte-width selects the number of least significant bytes of A and B that participate in a multiprecision subtraction operation whose result is output on the Y bus. Unselected high bytes of the Y bus contain corresponding bytes of B, unaltered. Status is defined for the selected least- significant bytes only.

Notes: **1**. Both carry and borrow arithmetic are supported under control of the BORROW-MODE pin. A HIGH on this pin causes the borrow (the complement of the carry) to be stored on subtract.

2. A HIGH and the MACRO input selects the MACRO-CARRY input instead of the C flag of the (micro) Status register as the carry-in for SUBRC. This allows an external carry flip flop to be selected for macroinstruction emulation.

3. Whatever the choice of carry-in based on (2) above, for SUBRC, it is inverted before being input to the adder if BORROW-MODE is HIGH. Therefore, in the borrow mode, both the internal and external carry store the borrow (not the carry).

Subtract A From B with Carry/Borrow

Notes: **4**. Status bits 31 to 24 are loaded with inter-nibble BCD borrows for SUBRC. The DIFF-CORR instructions must then be used to subtract BCD numbers longer than eight digits.

5. When performing multi-precision BCD arithmetic, a DIFF-CORR must be performed after each instruction.

BCD Sum Correct A

Purpose: To correct for BCD operations after execution of ADD(C) or INCR instructions.

Status Generated:

Z	N	C	V	L	M	S

Z Set if result equals zero. Cleared otherwise.

N Set if MSB or result is one. Cleared otherwise.

C Set if most significant NIBBLE CARRY based on byte-width is set. Cleared otherwise.

V EX-OR of two most significant NIBBLE CARRIES based on byte-width.

$Opcode_{16}$: 48

Description: Byte-width selects the number of least-significant nibble- pairs of A that are corrected before being output on the Y bus. Unselected high bytes of Y pass corresponding bytes of A unaltered.

A 32-bit correction word is formed as follows:

A field of 8 nibbles (32 bits) is set to zero. The nibble-carries, NC_0–NC_7, are then tested. Any nibble carry that is set to 1 causes a 6 (0110) to be placed in its corresponding nibble position in the correction word. The correction word is then added to the operand. The nibble carry flip-flops themselves are not affected by this operation. Status is defined for selected least-significant bytes only.

Notes: **1**. Each nibble carry flip-flop stores the carry-out of the corresponding nibble in the case of ADD and INCR.

BCD Sum Correct A

Notes: **2.** SUM-CORR is used after INCR, ADD or ADDC.

3. The definitions of negative and overflow flags assume that one leading sign-digit is appended to odd length to form an even length string (the definition of overflow is not useful for odd length signed strings). A digit of 0 represents a positive sign and a digit of 9 represents a negative sign in ten's complement. The N flag is set if the most-significant bit of the operand selected by byte- width is 1. The overflow is defined as the exclusive-OR of the carry-in and carry-out of the most-significant nibble (available from the nibble-carry register), as defined by byte-width. The sign nibble of the result is always either 0 or 9, except when an overflow occurs, when it becomes either 1 or 8 respectively. Refer to Appendix A.

4. The carry flag is loaded from the appropriate nibble- carry flag, as defined by byte-width.

BCD Sum Correct B

Purpose: To correct for BCD operations after execution of ADD(C) or INCR instructions.

Status Generated:

Z	N	C	V	L	M	S

Z Set if result equals zero. Cleared otherwise.

N Set if MSB or result is one. Cleared otherwise.

C Set if most significant NIBBLE CARRY based on byte-width is set. Cleared otherwise.

V EX-OR of two most significant NIBBLE CARRIES based on byte-width.

$Opcode_{16}$: 49

Description: Byte-width selects the number of least-significant nibble- pairs of B that are corrected before being output on the Y bus. Unselected high bytes of Y pass corresponding bytes of B unaltered.

A 32-bit correction word is formed as follows:

A field of 8 nibbles (32 bits) is set to zero. The nibble- carries, NC_0-NC_7, are then tested. Any nibble carry that is set to 1 causes a 6 (0110) to be placed in its corresponding nibble position in the correction word. The correction word is then added to the operand. The nibble carry flip-flops themselves are not affected by this operation. Status is defined for selected least-significant bytes only.

Notes: 1. Each nibble carry flip-flop stores the carry-out of the corresponding nibble in the case of ADD and INCR.

BCD Sum Correct B

Notes: **2**. SUM-CORR is used after INCR, ADD or ADDC.

3. The definition of negative and overflow flags assume that one leading sign-digit is appended to odd length to form an even length string (the definition of overflow is not useful for odd length signed strings). A digit of 0 represents a positive sign and a digit of 9 represents a negative sign in ten's complement. The N flag is set if the most-significant bit of the operand selected by byte- width is 1. The overflow is defined as the exclusive-OR of the carry-in and carry-out of the most-significant nibble (available from the nibble-carry register), as defined by byte-width. The sign nibble of the result is always either 0 or 9, except when an overflow occurs, when it becomes either 1 or 8 respectively. Refer to Appendix A.

4. The carry flag is loaded from the appropriate nibble- carry flag, as defined by byte-width.

Unsigned Divide First Step

Purpose: To set up the M and S flags of the status register in preparation for the unsigned divide steps. The quotient register is assumed to hold the dividend and B has the remainder, which is assumed to be the "sign extension" of the dividend, namely all zeroes. The result is assumed to replace the remainder.

Status Generated:

Z	**N**	C	V	**L**	**M**	**S**

Z Set if result is zero. Cleared otherwise.

N Set if result is negative. Cleared otherwise.

L Bit shifted out of B moved to L.

M Always set.

S Always reset.

$Opcode_{16}$: 4F

Description: Byte-width selects the number of least-significant bytes of B that are upshifted by one bit, with the sign bit of Q shifted into the least significant position, and output on the Y bus. Unselected high bytes pass corresponding bytes of B unaltered. Byte-width also selects the number of least significant bytes of Q that are shifted up one bit with 0 fill. The M flag is loaded with a 1 (so that the first divide step performs a subtract) and the S flag is initialized with a 0. The S flag sign extends the partial remainder throughout the division.

Unsigned Divide Last

Purpose: This instruction terminates the calculation of quotient and remainder. A has the divisor, B the partial remainder. After this step the quotient register contains the complete quotient, which is algebraically correct. That is, quotient divisor and remainder = dividend.

Status Generated:

Z	N	C	V	L	M	S

Z Set if result equals zero. Cleared otherwise.

N M EXOR S EXOR C

C Arithmetic carryout of F.

M M EXOR S EXOR C

S Always reset.

$Opcode_{16}$: 55

Description: Byte-width selects the number of least-significant bytes of A that are zero-extended by one bit and subtracted from the same number of least-significant bytes of B sign extended by S (M=1) or added to the same number of least-significant bytes of B sign-extended by S (M=0) to form F. F appears on the Y bus, with unselected high bytes passing corresponding bytes of B unaltered. Least-significant bytes of Q selected by byte-width are also upshifted by one bit with the new quotient bit (M EXOR S EXOR C) shifted in. The new quotient bit is loaded into M. Unselected high bytes of Q are unaffected. S is set to zero.

Unsigned Divide Step

Purpose: This instruction constitutes the inner loop for unsigned single precision division. The quotient register contains the dividend, A has the divisor and B has the partial remainder which is extended by the S flag. The result, Y, is assumed to replace the partial remainder.

Status Generated: Z N C V L M S

Z Set if result equals zero. Cleared otherwise.

C Arithmetic carryout of F.

L Bit shifted out of F moved to L.

M M EXOR S EXOR C

S Bit shifted out of F moved to S.

$Opcode_{16}$: 54

Description: Byte-width selects the number of least significant bytes of A that are zero-extended by one bit and subtracted from the same number of least-significant bytes of B extended by S (M=1) or added to the same number of least-significant bytes of B extended by S (M=0) to form the value F. F is then upshifted by one bit, with the sign bit of Q shifted into the least significant position, and output on the Y bus. Unselected high bytes contain corresponding bytes of B, unaltered. Least-significant bytes of Q selected by byte- width are also upshifted by one bit, with the new quotient bit (M EXOR S EXOR C) shifted in. The new quotient bit is the complement of the sign of the (extended) result before the upshift. Unselected high bytes of Q are unaffected. The M flag is updated with the new quotient bit.

Notes: **1**. The ALU has one extra bit of precision for this instruction. The inputs to that extra ALU bit are 0 (divisor sign), S (dividend sign), the carryout, and the M flag which controls addition/subtraction.

2. The M flag is defined the same way as for signed division. This allows the same correction steps to be used for signed and unsigned single precision division.

3. For operands smaller than a full word, passing of unselected bytes of B is not affected by the extra bit of precision of the ALU.

Unsigned Multiply First

Purpose: To initialize the partial product register and the Multiply flag (M) prior to a single precision unsigned multiplication, and also to perform the first iteration of the Multiply algorithm, yielding two new product bits. A has the multiplicand, whose width is specified by byte-width. Q has been loaded with the multiplier. The result replaces the partial product, B.

Status Generated:

Z	N	C	V	L	**M**	S

M Q_1 moved to M.

Opcode$_{16}$: 5B

Description: The ALU is internally extended by two bits at the appropriate byte boundary depending on byte-width. Selected bytes of B are internally zeroed. A is zero extended across the two extra bits to form A*. Next, based on the values of Q_1 and Q_0, the value F* is computed;

If $Q_1Q_0 = 00$, $F^* = 0$;

If $Q_1Q_0 = 01$, $F^* = A^*$;

If $Q_1Q_0 = 10$, F^* = 2's complement of $2A^*$;

If $Q_1Q_0 = 11$, F^* = 2's complement of A^*;

F^* shifted down 2 bits is output on the Y bus. Unselected high bytes of Y contain corresponding bytes of B unaltered. Selected bytes of Q are down shifted by 2 bits with F_1F_0 being the fill bits. Unselected high bytes of Q are unaffected.

Notes: **1.** The partial product register may be brought in over the B input. This is necessary only if the unselected high bytes of that register are to remain unaltered by the write back into B since the selected low bytes of B are ignored by the ALU anyway.

2. The passing property of the ALU is not affected by the extra bits of precision.

3. If a Multiply accumulate is required then the (partial) product register will start off with some value other than zero. In such cases, UMULFIRST should not be used. Instead, first reset the M flag using RSTBIT-STAT (p=21), and the use UMULSTEP instead of UMULFIRST.

Unsigned Multiply

Purpose: This step corrects the final product in the case of unsigned single precision multiplication. A has the multiplicand and B has the product (most significant word), whose width is specified by byte-width. The result replaces the product.

Status Generated:

Z	N	C	V	L	M	S

Z Set if result equals zero. Cleared otherwise.

$Opcode_{16}$: 5D

Description: If the M flag is 1, then the low order bytes selected by byte-width of A and B are added and the result output on the Y bus. Otherwise, the B input passes unaltered to the Y bus. In either case, unselected high bytes of Y contain corresponding bytes of B unaltered, and the Z flag is set according to selected bytes only.

Notes: 1. The Booth algorithm yields the correct product for signed two's complement integers. In the case of unsigned integers, the leading bit is incorrectly interpreted as a sign bit. If the most significant bit of the multiplier is zero, then the product is correct. However, in the other case, correction is needed.

The operation of the Booth algorithm may be understood as follows: The algorithm looks for strings of ones in the multiplier. At the start of the string, it subtracts the multiplicand and at the end of the string adds back the multiplicand in the correct significance. The sign bit in a two's complement number is really the start of an infinite string of 1s. The algorithm sees the start of the string and subtracts the multiplicand, but the algorithm terminates, giving it no chance to end the string by adding back the multiplicand. This in fact yields the correct result for signed multiplication.

For unsigned multiplication, all that is needed is to concatenate two high order zero bits to the multiplier, and then leave the algorithm to recognize the end of the string of 1s and compensate. This is the role of UMULLAST. Of course, if the last iteration of the UMULSTEP has already recognized the end of string (Q_1Q_0M- 001 or 010) then UMULLAST has nothing to do. In such cases, M has a 0 and, as indicated above, UMULLAST indeed has no effect on the high word of the product.

Unsigned Multiply Step (A, B, Byte-Width)

Purpose: This step constitutes the inner loop for single precision unsigned multiplication. Each iteration of the loop yields two new product bits. A has the multiplicand, B has the partial product, and byte-width specifies the number of bytes selected. Q contains the multiplier and some product bits. The result of this step replaces the partial product.

Status Generated:

Z	N	C	V	L	**M**	S

M Q_1 moved to M.

$Opcode_{16}$: 5C

Description: The ALU is internally extended by two bits at the appropriate byte boundary depending on byte-width. A is zero extended across the two bits to form A*.

B is sign extended across the two bits to form B*. Next, based on the values of Q_1 Q_0 and M, the value F* is computed.

If $Q_1Q_0M = 000$ or 111, $F^* = B^*$;

If $Q_1Q_0M = 001$ or 010, $F^* = B^* + A^*$;

If $Q_1Q_0M = 101$ or 110, $F^* = B^* - A^*$;

If $Q_1Q_0M = 011$, $F^* = B^* + 2A^*$;

If $Q_1Q_0M = 100$, $F^* = B^* - 2A^*$;

F* shifted down 2 bits is output on the Y bus. Unselected high bytes of Y contain corresponding bytes of B unaltered. Selected bytes of Q are down shifted by 2 bits with F1F0 being the fill bits. Unselected high bytes of Q are unaffected.

UP1-0F-A

Upshift A by 1 Bit, Zero Fill

Purpose: Upshift A by one bit zero fill.

Status Generated: | **Z** | **N** | C | **V** | **L** | M | S |

Z Set if result is zero. Cleared otherwise.

N Set if result is negative. Cleared otherwise.

V Set if A_{sign} EXOR Y_{sign} equals one. Cleared otherwise.

L Sign bit of selected bytes of A moved to L.

$Opcode_{16}$: 30

Description: Byte-width selects the number of least significant bytes of A that are upshifted by one bit, with zero shifted in and passed onto the Y bus. Unselected high bytes of Y contain corresponding bytes of A, unaltered. The L flag gets the sign bit of A. The V flag gets the exclusive or of the sign bit of A and the sign bit of Y. Status is defined for selected least-significant bytes only.

Notes: 1. V is useful for detecting overflow when a signed integer is upshifted. When overflow occurs on a single bit upshift, the sign of the result is wrong. The correct sign is given by N EXOR V (STATUS <14>) which is a read-only bit of the status register. This bit can be extracted into L using EXTBIT-STAT (14) and shifted back using DN1-LF-A to recover the original value.

UP1-0F-AQ

Upshift A, Q by 1 Bit, Zero Fill

Purpose: Double precision upshift A, Q by one bit. Fill A with sign bit of Q. 0 fill Q.

Status Generated: | **Z** | **N** | C | **V** | **L** | M | S |

Z Set if result is zero. Cleared otherwise.

N Set if result is negative. Cleared otherwise.

V Set if A_{sign} EXOR Y_{sign} equals one. Cleared otherwise.

L Sign bit of selected bytes of A moved to L.

$Opcode_{16}$: 32

Description: Byte-width selects the number of least significant bytes of A that are upshifted by one bit, with the sign bit of Q shifted in and passed onto the Y bus. Unselected high bytes of Y contain corresponding bytes of A, unaltered. Byte-width also selects the number of least significant bytes of Q that are shifted up by one bit with zero as a fill bit. Unselected bytes of Q are not affected. The L flag gets the sign bit of A. The V flag gets the exclusive or of the sign bit of A and the sign bit of Y. Status is defined for selected least significant bytes only.

Notes: 1. V is useful for detecting overflow when a signed integer is upshifted. When overflow occurs on a single bit upshift, the sign of the result is wrong. The correct sign is given by N EXOR V (STATUS <14>) which is a read-only bit of the status register. This bit can be extracted into L using EXBIT-STAT(14) and shifted back using DN1-LF-A to recover the original value.

UP1-0F-B

Upshift B by 1 Bit, Zero Fill

Purpose: Upshift B by one bit zero fill.

Status Generated:

Z	N	C	V	L	M	S

Z Set if result equals zero. Cleared otherwise.

N Set if result is negative. Cleared otherwise.

V Set if B_{sign} EXOR Y_{sign} equals one. Cleared otherwise.

L Sign bit of selected bytes of B moved to L.

$Opcode_{16}$: 31

Description: Byte-width selects the number of least significant bytes of B that are upshifted by one bit, with zero shifted in and passed onto the Y bus. Unselected high bytes of Y contain corresponding bytes of B, unaltered. The L flag gets the sign bit of B. The V flag gets the exclusive or of the sign bit of B and the sign bit of Y. Status is defined for selected least-significant bytes only.

Notes: 1. V is useful for detecting overflow when a signed integer is upshifted. When overflow occurs on a single bit upshift, the sign of the result is wrong. The correct sign is given by N EXOR V (STATUS <14>) which is a read-only bit of the status register. This bit can be extracted into L using EXTBIT-STAT(14) and shifted back using DN1-LF-B to recover the original value.

UP1-0F-BQ

Upshift B, Q by 1 Bit, Zero Fill

Purpose: Double precision upshift B, Q by one bit. Fill B with sign bit of Q. 0 fill Q.

Status Generated: | **Z** | **N** | C | **V** | **L** | M | S |

- Z Set if result equals zero. Cleared otherwise.
- N Set if result is negative. Cleared otherwise.
- V Set if B_{sign} EXOR Y_{sign} equals one. Cleared otherwise.
- L Sign bit of selected bytes of B moved to L.

$Opcode_{16}$: 33

Description: Byte-width selects the number of least significant bytes of B that are upshifted by one bit, with the sign bit of Q shifted in and passed onto the Y bus. Unselected high bytes of Y contain corresponding bytes of B, unaltered. Byte-width also selects the number of least significant bytes of Q that are shifted up by one bit with zero as a fill bit. Unselected bytes of Q are not affected. The L flag gets the sign bit of B. The V flag gets the exclusive or of the sign bit of B and the sign bit of Y. Status is defined for selected least significant bytes only.

Notes: 1. V is useful for detecting overflow when a integer is upshifted. When overflow occurs on a single bit upshift, the sign of the result is wrong. The correct sign is given by N EXOR V (STATUS <14>) which is a read-only bit of the status register. This bit can be extracted into L using EXTBIT-STAT(14) and shifted back using DN1-LF-B to recover the original value.

UP1-1F-A

Upshift A by 1 Bit, One

Purpose: Upshift A by one bit one fill.

Status Generated: **Z** **N** C **V** **L** M S

Z Set if result is zero. Cleared otherwise.

N Set if result is negative. Cleared otherwise.

V Set if A_{sign} EXOR Y_{sign} equals one. Cleared otherwise.

L Sign bit of selected bytes of A moved to L.

$Opcode_{16}$: 34

Description: Byte-width selects the number of least significant bytes of A that are upshifted by one bit, with a "one" shifted in and passed onto the Y bus. Unselected high bytes of Y contain corresponding bytes of A, unaltered. The L flag gets the sign bit of A. The V flag gets the exclusive or of the sign bit of A and the sign bit of Y. Status is defined for selected least-significant bytes only.

Notes: **1**. V is useful for detecting overflow when a signed integer is upshifted. When overflow occurs on a single bit upshift, the sign of the result is wrong. The correct sign is given by N EXOR V (STATUS <14>) which is a read-only bit of the status register. This bit can be extracted into L using EXTBIT-STAT(14) and shifted back using DN1-LF-A to recover the original value.

UP1-1F-AQ

Upshift A, Q by 1 Bit, One

Purpose: Double precision upshift A, Q by one bit. Fill A with sign bit of Q. 1 fill Q.

Status Generated: | **Z** | **N** | C | **V** | **L** | M | S |

Z Set if result is zero. Cleared otherwise.

N Set if result is negative. Cleared otherwise.

V Set if A_{sign} EXOR Y_{sign} equals one. Cleared otherwise.

L Sign bit of selected bytes of A moved to L.

$Opcode_{16}$: 36

Description: Byte-width selects the number of least significant bytes of A that are upshifted by one bit, with the sign bit of Q shifted in and passed onto the Y bus. Unselected high bytes of Y contain corresponding bytes of A, unaltered. Byte-width also selects the number of least significant bytes of Q that are shifted up by one bit with a one as a fill bit. Unselected bytes of Q are not affected. The L flag gets the sign bit of A. The V flag gets the exclusive or of the sign bit of A and the sign bit of Y. Status is defined for selected least- significant bytes only.

Notes: 1. V is useful for detecting overflow when a signed integer is upshifted. When overflow occurs on a single bit upshift, the sign of the result is wrong. The correct sign is given by N EXOR V (STATUS <14>) which is a read-only bit of the status register. This bit can be extracted into L using EXTBIT-STAT(14) and shifted back using DN1-LF-A to recover the original value.

UP1-1F-B

Upshift B by 1 Bit, One Fill

Purpose: Upshift B by one bit one fill.

Status Generated: **Z** **N** C **V** **L** M S

Z Set if result equals zero. Cleared otherwise.

N Set if result is negative. Cleared otherwise.

V Set if B_{sign} EXOR Y_{sign} equals one. Cleared otherwise.

L Sign bit of selected bytes of B moved to L.

$Opcode_{16}$: 35

Description: Byte-width selects the number of least significant bytes of B that are upshifted by one bit, with a one shifted in and passed onto the Y bus. Unselected high bytes of Y contain corresponding bytes of B, unaltered. The L flag gets the sign bit of B. The V flag gets the exclusive or of the sign bit of B and the sign bit of Y. Status is defined for selected least-significant bytes only.

Notes: 1. V is useful for detecting overflow when a signed integer is upshifted. When overflow occurs on a single bit upshift, the sign of the result is wrong. The correct sign is given by N EXOR V (STATUS <14>) which is a read-only bit of the status register. This bit can be extracted into L using EXTBIT-STAT(14) and shifted back using DN1-LF-B to recover the original value.

UP1-1F-BQ

Upshift B, Q by 1 Bit, One Fill

Purpose: Double precision upshift B, Q by one bit. Fill B with sign bit of Q. 1 fill Q.

Status Generated: | **Z** | **N** | C | **V** | **L** | M | S |

Z Set if result equals zero. Cleared otherwise.

N Set if result is negative. Cleared otherwise.

V Set if B_{sign} EXOR Y_{sign} equals one. Cleared otherwise.

L Sign bit of selected bytes of B moved to L.

$Opcode_{16}$: 37

Description: Byte-width selects the number of least significant bytes of B that are upshifted by one bit, with the sign bit of Q shifted in and passed onto the Y bus. Unselected high bytes of Y contain corresponding bytes of B, unaltered. Byte-width also selects the number of least significant bytes of Q that are shifted up by one bit with a one as a fill bit. Unselected bytes of Q are not affected. The L flag gets the sign bit of B. The V flag gets the exclusive or of the sign bit of B and the sign bit of Y. Status is defined for selected least- significant bytes only.

Notes: **1**. V is useful for detecting overflow when a signed integer is upshifted. When overflow occurs on a single bit upshift, the sign of the result is wrong. The correct sign is given by N EXOR V (STATUS <14>) which is a read-only bit of the status register. This bit can be extracted into L using EXTBIT-STAT(14) and shifted back using DN1-LF-B to recover the original value.

UP1-LF-A

Upshift A by 1 Bit, Line Bit Fill

Purpose: Upshift A by one bit link bit fill.

Status Generated: | **Z** | **N** | C | **V** | **L** | M | S |

Z Set if result is zero. Cleared otherwise.

N Set if result is negative. Cleared otherwise.

V Set if A_{sign} EXOR Y_{sign} equals one. Cleared otherwise.

L Sign bit of selected bytes of A moved to L.

$Opcode_{16}$: 38

Description: Byte-width selects the number of least significant bytes of A that are upshifted by one bit, with the link bit shifted in and passed onto the Y bus. Unselected high bytes of Y contain corresponding bytes of A, unaltered. The L flag gets the sign bit of A. The V flag gets the exclusive or of the sign bit of A and the sign bit of Y. Status is defined for selected least-significant bytes only.

Notes: 1. V is useful for detecting overflow when a signed integer is upshifted. When overflow occurs on a single bit upshift, the sign of the result is wrong. The correct sign is given by N EXOR V (STATUS <14>) which is a read-only bit of the status register. This bit can be extracted into L using EXTBIT-STAT(14) and shifted back using DN1-LF-A to recover the original value.

UP1-LF-AQ

Upshift A, Q by 1 Bit, Link Bit Fill

Purpose: Double precision upshift A, Q by one bit. Fill A with sign bit of Q. Fill Q with link bit.

Status Generated:

Z	N	C	V	L	M	S

Z Set is result is zero. Cleared otherwise.

N Set if result is negative. Cleared otherwise.

V Set if A_{sign} EXOR Y_{sign} equals one. Cleared otherwise.

L Sign bit of selected bytes of A moved to L.

$Opcode_{16}$: 3A

Description: Byte-width selects the number of least significant bytes of A that are upshifted by one bit, with the sign bit of Q shifted in and passed onto the Y bus. Unselected high bytes of Y contain corresponding bytes of A, unaltered. Byte-width also selects the number of least significant bytes of Q that are shifted up by one bit with the L flag as a fill bit. Unselected bytes of Q are not affected. The L flag gets the sign bit of A. The V flag gets the exclusive or of the sign bit of A and the sign bit of Y. Status is defined for selected least-significant bytes only.

Notes: **1**. V is useful for detecting overflow when a signed integer is upshifted. When overflow occurs on a single bit upshift, the sign of the result is wrong. The correct sign is given by N EXOR V (STATUS <14>) which is a read-only bit of the status register. This bit can be extracted into L using EXTBIT-STAT(14) and shifted back using DN1-LF-A to recover the original value.

UP1-LF-B

Upshift B by 1 Bit, Link Bit Fill

Purpose: Upshift B by one bit link bit fill.

Status Generated: **Z** **N** C **V** **L** M S

Z Set if result equals zero. Cleared otherwise.

N Set if result is negative. Cleared otherwise.

V Set if B_{sign} EXOR Y_{sign} equals one. Cleared otherwise.

L Sign bit of selected bytes of B moved to L.

$Opcode_{16}$: 39

Description: Byte-width selects the number of least significant bytes of B that are upshifted by one bit, with the link bit shifted in and passed onto the Y bus. Unselected high bytes of Y contain corresponding bytes of B, unaltered. The L flag gets the sign bit of B. The V flag gets the exclusive or of the sign bit of B and the sign bit of Y. Status is defined for selected least-significant bytes only.

Notes: 1. V is useful for detecting overflow when a signed integer is upshifted. When overflow occurs on a single bit upshift, the sign of the result is wrong. The correct sign is given by N EXOR V (STATUS <14>) which is a read-only bit of the status register. This bit can be extracted into L using EXTBIT-STAT(14) and shifted back using DN1-LF-B to recover the original value.

UP1-LF-BQ

Upshift B, Q by 1 Bit, Link Bit Fill

Purpose: Double precision upshift B, Q by one bit. Fill B with sign bit of Q. Fill Q with link bit.

Status Generated:

Z	N	C	V	L	M	S

Z Set if result equals zero. Cleared otherwise.

N Set if result is negative. Cleared otherwise.

V Set if B_{sign} EXOR Y_{sign} equals one. Cleared otherwise.

L Sign bit of selected bytes of B moved to L.

$Opcode_{16}$: 3B

Description: Byte-width selects the number of least significant bytes of B that are upshifted by one bit, with the sign bit of Q shifted in and passed onto the Y bus. Unselected high bytes of Y contain corresponding bytes of B, unaltered. Byte-width also selects the number of least significant bytes of Q that are shifted up by one bit with the L flag as a fill bit. Unselected bytes of Q are not affected. The L flag gets the sign bit of B. The V flag gets the exclusive or of the sign bit of B and the sign bit of Y. Status is defined for selected least-significant bytes only.

Notes: **1**. V is useful for detecting overflow when a signed integer is upshifted. When overflow occurs on a single bit upshift, the sign of the result is wrong. The correct sign if given by N EXOR V (STATUS <14>) which is a read- only bit of the status register. This bit can be extracted into L using EXTBIT-STAT(14) and shifted back using DN1-LF-B to recover the original value.

Logical Exclusive-NOR

Purpose: To compute a logical XNOR of two integers of a given byte- width.

Status Generated:

Z	**N**	C	V	L	M	S

Z Set if result is zero. Cleared otherwise.

N Set if result is negative. Cleared otherwise.

Opcode$_{16}$: 41

Description: Byte-width selects the number of least-significant bytes of A and B that participate in a logical XNOR operation whose result is output on the Y bus. Unselected high bytes of the Y bus contain corresponding bytes of B, unaltered. Status is defined for the selected least-significant bytes only.

Logical Exclusive-OR

Purpose: To compute a logical XOR of two integers of a given byte- width.

Status Generated:

Z	**N**	C	V	L	M	S

Z Set if result is zero. Cleared otherwise.

N Set if result is negative. Cleared otherwise.

Opcode$_{16}$: 3F

Description: Byte-width selects the number of least-significant bytes of A and B that participate in a logical XOR operation whose result is output on the Y bus. Unselected high bytes of the Y bus contain corresponding bytes of B, unaltered. Status is defined for the selected least-significant bytes only.

XOR Field in A with B

Purpose: To perform a logical XOR operation between a field in A and a field in B.

Status Generated:

Z	N	C	V	L	M	S

Z Set if result of XOR operation is zero. Cleared otherwise.

N Set if Y_{31} is a one. Cleared otherwise.

$Opcode_{16}$: 76

Description: When the position input is positive, the LSB-aligned field in A is upshifted to align it with the field in B starting at the bit specified by position. When the position input is negative, the field in A starting at the bit specified by the two's complement of position is rotated right to align it with the LSB- aligned field in B. A logical XOR operation then takes place between the field in A and the field in B, up to the width specified by the width field and the resulting field appears on the Y bus, LSB-aligned in the case that position was negative, or starting at the bit specified by position in the case that position was positive. The remaining bits of Y pass corresponding bits of B unaltered. The Z flag is set according to the contents of the selected field. However, the N flag is set according to the most significant bit of Y.

Notes: **1.** A HIGH on the I8 input selects the position inputs from the status register (STATUS <5:0>); a LOW selects the direct position <5:0> input pins.

2. A HIGH on the I7 input selects the width inputs from the status register (STATUS <12:8>); a LOW selects the direct width <4:0> input pins.

3. For position <0, A is rotated rather than shifted right. Therefore, if the sum of position and width exceeds 32, bits rotated around from the low-order end A will be selected.

XOR Field in A with B

Notes: **4.** For p> 0, a width of 0 specifies the entire field to the left of position.
e.g., p=03; w=0

```
A  = 0111 0010 0001 0101 0001 1000 0000 1111 = 7215180F
A1 = 1001 0000 1010 1000 1100 0000 0111 1xxx
B  = 1000 1111 1100 1110 0001 0011 0111 0010 = 8FCE1372
Y  = 0001 1111 0110 0110 1101 0011 0000 1010 = 1F66D30A
```

For p<0, a width of 0 specifies the entire 32 bit field.
e.g., p=3C; w=0

```
A  = 0111 0001 0111 0010 0011 1111 1110 1101 = 71723FED
A1 = 1101 0111 0001 0111 0010 0011 1111 1110
B  = 1000 0010 1100 1110 0001 0101 0000 1001 = 82CE1509
Y  = 0101 0101 1101 1001 0011 0110 1111 0111 = 55D936F7
```

XORF-AL-A

XOR Aligned Fields in A & B

Purpose: To perform a logical XOR operation between a field in A and a field in B that start at the same position.

Status Generated:

Z	N	C	V	L	M	S

Z Set if resulting field is zero. Cleared otherwise.

N Set if Y_{31} equals one. Cleared otherwise.

$Opcode_{16}$: 77

Description: The position input is treated as a 5 bit unsigned number. A logical XOR operation is performed between the fields in A and B starting at the bit specified by position and of the specified width. The resulting field appears on the Y bus at the same position. The remaining bits of Y contain corresponding bits of B unaltered. The Z flag is set according to the contents of the selected field. However, the N flag is set according to the most significant bit of Y.

Notes: **1.** A HIGH on the I-8 input selects the status register (STATUS <5:0>) as the source of position; a LOW selects the direct position <5:0> input pins.

2. A HIGH on the I-7 input selects the status register (STATUS <12:8>) as the source of width; a LOW selects the direct width <4:0> input pins.

3. A width of 0 specifies the entire field to the left of position.

Pass 0'S to Y-Bus

Purpose: Type conversion of unsigned integers to multiply words.

Status Generated:

Z	N	C	V	L	M	S

Z Always set.

N Always reset.

Opcode$_{16}$: 3C

Description: Byte-width selects the number of least-significant bytes of Y that contain zeroes. Unselected high-order bytes are taken from the B input, unaltered.

Zero Extend A

Purpose: Type conversion of unsigned integers to full words; data movement.

Status Generated: | **Z** | **N** | C | V | L | M | S |

Z Set if Y equals 0. Cleared otherwise.

N Set if selected least significant bytes of Y are negative. Cleared otherwise.

Opcode$_{16}$: 00

Description: Byte-width specifies the number of least-significant bytes of A that pass onto the Y bus. The unselected most significant bytes contain zeroes. Status is defined for selected least- significant bytes only.

Notes: 1. To convert an unsigned byte to a double word (for example) use Zero-EXTA on the byte with byte-width = 01 to obtain the least significant word followed by the Zero instruction with byte-width = 00 to get the most significant word.

Zero Extend B

Purpose: Type conversion of unsigned integers to full words; data movement.

Status Generated:

Z	N	C	V	L	M	S

Z Set if Y equals 0. Cleared otherwise.

N Set if selected least significant bytes of Y are negative. Cleared otherwise.

$Opcode_{16}$: 01

Description: Byte-width specifies the number of least-significant bytes of B that pass onto the Y bus. The unselected most significant bytes contain zeroes. Status is defined for selected least- significant bytes only.

Notes: 1. To convert an unsigned byte to a double word (for example) use ZERO-EXTB on the byte with byte-width = 01 to obtain the least significant word followed by the Zero instruction with byte-width = 00 to get the most significant word.

APPENDIX A

HOW BCD ARITHMETIC WORKS ON THE Am29332 ALU

The Am29332 ALU performs BCD addition and subtraction in a two stage process. The first stage is a normal binary operation (performed on valid BCD operands), and the status information that results (C,N,V,Z) is the expected status for a binary operation. For all add and subtract class instructions, including:

ADD, ADDC
SUB, SUBC
SUBR, SUBRC
NEG
INCR, INCR2, INCR4
DECR, DECR2, DECR4

the 8 nibble-carry/borrow flag bits are also updated. These flags are treated as nibble-carries for the ADD and INCR class instructions, and as nibble-borrows for the SUB, DECR, and NEG class instructions. The borrow control signal into the chip does not affect the meaning of the nibble-borrows, it only controls the polarity of the main carry flag, the macro carry in, and macro carry out.

The second stage is the correction instruction. For the ADD and INCR instructions, the SUMCORR instruction is used. For SUB, DECR, and NEG, the DIFFCORR instruction is used. These instructions use the eight nibble-carry/borrow bits, and the result of the previous binary operation (on BCD operands) to create a correct BCD result.

Negative numbers are represented in 10's complement format. This means that if the most-significant digit is in the range 5 through 9, the number is negative. For example, if we were working with 16 bit BCD numbers, there would be four BCD digits. They could have values as follows:

10's complement number	signed value
5000	–5000
5001	–4999
5002	–4998
8000	–2000
9000	–1000
9998	–0002
9999	–0001
0000	0000
0001	+0001
0002	+0002
4000	+4000
4998	+4998
4999	+4999

Negative numbers are formed by subtracting the required number from 10000. For example, to find –952, do the following:

10000 – 952 = 9048 (9048 is the 10's complement of 952 in a 16-bit BCD system)

For 32-bit numbers (8 digits), the value is subtracted from 100,000,000 to form the 10's complement of the number.

How Addition is Performed:

Two BCD numbers are applied to the ALU, and a binary addition is performed. The result is the expected result of adding the two numbers as if they were both binary values. As a side effect, the eight BCD nibble-carries are also set to either 0 or 1, depending on the result for each of the eight nibbles being added. If the result of this add is applied to the ALU and a SUMCORR instruction is executed, the result is the correct BCD result of adding the original two operands.

How the BCD Nibble-Carries are Formed During Addition

The BCD nibble-carries are set to a 1 under either of the following conditions:

1) The result of adding the two nibbles of the operands plus the binary carry from the lower order addition gives a nibble sum greater than 9.

2) The result of adding the two nibbles of the operands gives a nibble sum of 9, the nibble-carry for the digit to the right of the current digit is set, and there is no binary carry from the digit to the right of the current digit.

All other results cause the respective nibble-carry to be set to 0. The resultant eight nibble-carries are stored in the status register. Here are some examples, in a 16-bit format (assuming that the instruction being executed is the ADD instruction) :

1) 44 + 44

```
  0044
+ 0044
  0088      nibble-carries: 0 0 0 0
```

2) 45 + 45

```
  0045
+ 0045
  008A      nibble-carries: 0 0 0 1
```

3) 48 + 48

```
  0048
+ 0048
  0090      nibble-carries: 0 0 0 1
```

4) 48 + 58

```
  0048
+ 0058
  00A0      nibble-carries: 0 0 1 1
```

5) 45 + 45

```
  0045
+ 0045
  009A      nibble-carries: 0 0 1 1
```

6) 428 + 568

```
  0428
+ 0568
  0990      nibble-carries: 0 0 0 1
```

7) 428 + 578

```
  0428
+ 0578
  09A0      nibble-carries: 0 1 1 1
```

How Borrow Mode Affects Addition

Borrow mode does not affect addition in any way.

How Carry-In Affects Addition

Carry-in (from either the macro carry input or the carry bit in the status register, depending on the select signal) causes an extra 1 to be added, if this signal is asserted (1), and the instruction is the ADDC instruction. For the nibble-carry generation, this will affect it the same way as a binary carry from the previous lower order bits. For example, if the instruction is ADDC, and the carry-in is 1, then

8) 35 + 64

```
  0035
+ 0064      plus carry-in
  009A      nibble-carries: 0 0 1 1
```

9) 37 + 68

```
  0037
+ 0068      plus carry in
  00A0      nibble-carries: 0 0 1 1
```

10) 37 + 58

```
  0037
+ 0058      plus carry-in
  0090      nibble-carries: 0 0 0 1
```

How Carry-Out is Formed During Addition

Carry-out from the ALU is the standard binary carry output that results from binary addition. Unless the ALU hold signal is asserted, the status register is updated with this carry information. Carry-out is asserted for a 2's complement ALU during addition if the sum of the most-significant bit of the two operands plus the binary carry into the most-significant bit is greater than 1.

How Summation Correct Works

The summation correct instruction expects to be preceded by an addition class instruction. The addition class instruction will set up the nibble-carries, as well as produce a result of doing a binary add on two BCD numbers (or adding 1, 2, or 4 to a number in the case of the INCR instructions). The addition result is applied to the ALU as the operand to be corrected, and the SUMCORR instruction uses the BCD nibble-carries to create a correction number. For each nibble-carry that is a 1, the correction nibble is 6. For each nibble-carry that is a 0, the correction nibble is 0. The correction word (made up of these 0s and 6s) is added to the number to be corrected, *using a binary add operation within the ALU.* For the above examples, the correction cycles are as follows:

The first four examples had carry-in = 0 for the ADD

1) 44 + 44 = 0 0 8 8 nibble-carries: 0 0 0 0

correction value 0 0 0 0
corrected result 0 0 8 8

2) 45 + 45 = 0 0 8 A nibble-carries: 0 0 0 1

correction value 0 0 0 6
corrected result 0 0 9 0

3) 48 + 48 = 0 0 9 0 nibble-carries: 0 0 0 1

correction value 0 0 0 6
corrected result 0 0 9 6

4) 48 + 58 = 0 0 A 0 nibble-carries: 0 0 1 1

correction value 0 0 6 6
corrected result 0 1 0 6

5) 45 + 45 = 0 0 9 A nibble-carries: 0 0 0 1

correction value 0 0 0 6
corrected result 0 0 9 0

6) 428 + 568 = 0 9 9 0 nibble-carries: 0 0 0 1

correction value 0 0 0 6
corrected result 0 9 9 6

7) 428 + 578 = 0 9 A 0 nibble-carries: 0 1 1 1

correction value0 6 6 6
corrected result 1 0 0 6

The next three examples had carry-in = 1 for the ADDC

8) 35 + 64 = 0 0 9 A nibble-carries: 0 0 1 1

correction value0 0 6 6
corrected result 0 1 0 0

9) 37 + 68 = 0 0 A 0 nibble-carries: 0 0 1 1

correction value0 0 6 6
corrected result 0 1 0 6

10) 37 + 58 = 0 0 9 0 nibble-carries: 0 0 0 1

correction value0 0 0 6
corrected result 0 0 9 6

How Carry-In Affects Summation Correct

Carry-in is ignored during SUMCORR.

How Carry-Out is Formed During Summation Correct

Carry out is the value of the nibble carry status bit of the most significant digit of the operation being performed. For 32 bit mode it is NC7, for 24 bit mode it is NC5, for 16 bit mode it is NC3, and for 8 bit mode it is NC1.

What Happens to Addition in Non 32–Bit Mode

For ALU sizes other than 32 bits, the output to the Y bus is the expected 24, 16, or 8-bit result, merged with the unmodified high order bytes from one of the two operands. The nibble-carries are set up as if the ALU was in 32-bit mode.

For the SUMCORR instruction, only the active 8, 16, or 24 bits are corrected. The carry-out is selected from the appropriate position, as described in the previous section. The unselected high order bytes are passed through unmodified.

How Subtraction is Performed

Two BCD numbers are applied to the ALU, after the operand to be subtracted has been inverted (1's complement) and a binary addition is performed (with carry-in set to 1 for SUB and NEG). The result is the expected result of subtracting one of the numbers from the other as if they were both binary values. As a side effect, the eight BCD nibble-borrows are also set to either 0 or 1, depending on the result for each of the eight nibbles being subtracted. If the result of this subtract is applied to the ALU, and a DIFFCORR instruction is executed, the result is the correct BCD result of subtracting the original two operands.

How the BCD Nibble-Borrows are Formed During Subtraction

A nibble-borrow occurs if the result of subtracting one nibble from the other gives a result less than 0. This is implemented in the ALU by storing the complement of the inter-nibble carry while the ALU is doing an add (remember the ALU is actually doing an add on the two operands after the one to be subtracted has been complemented). The ALU always stores nibble-borrows in the status register, regardless of the state of the borrow control pin on the chip.

Here are some examples in an 8-bit format (assuming that the instruction being executed is the SUB instruction):

11) 5 – 3

```
  0 5
– 0 3
  0 2        nibble-borrows: 0 0
```

in binary, this is:

```
 0000 0101
– 0000 0011
```

This becomes (after complementing, setting carry into ALU to 1, and doing an add):

```
  0000  0101
+ 0000  0001
+ 1111  1100   ← complement of 0000 0011
1 0000  0010   ← the answer, 2
  ↑     ↑      binary carries occurred at both boundaries so both nibble-
               borrows are zero.
```

12) 5 – 5

```
  0 5
– 0 5
  0 0        nibble-borrows: 0 0
```

in binary, this is:

```
 0000 0101
– 0000 0101
```

This becomes (after complementing, setting carry into ALU to 1, and doing an add):

```
  0000  0101
+ 0000  0001
+ 1111  1010   ← complement of 0000 0101
1 0000  0000   ← the answer, 0
  ↑     ↑      binary carries occurred at both boundaries so both nibble-
               borrows are zero.
```

13) 3 – 5

```
  0 3
– 0 5
– 0 2        nibble-borrows: 1 1
```

in binary, this is:

```
 0000 0011
– 0000 0101
```

This becomes (after complementing, setting carry into ALU to 1, and doing an add):

```
  0000  0011
+ 0000  0001
+ 1111  1010   ← complement of 0000 0101
0 1111  1110   ← the answer, FE needs correction
  ↑     ↑      No binary carry occurred at either boundary so both nibble-
               borrows are one.
```

How Borrow Mode Affects Subtraction

Borrow mode when asserted inverts the value for the carry-in and carry-out of the chip. It also inverts the data to be stored into the internal micro-carry status flag i.e. the data in the micro-carry register is the same sense as the chips carry-out and carry-in. In this mode, all three are better referred to as borrow-in, borrow-out, and micro-borrow status flag. These meanings only apply when borrow mode is asserted, and a subtraction operation is being performed.

Note that subtraction operations occur with SUB, SUBR, SUBC, SUBRC, DECR, MULTIPLY instructions, and DIVIDE instructions.

How Carry-In Affects Subtraction

When borrow mode is not asserted, carry-in set to 1 will make a SUBC execute like a SUB, and perform A – B. If carry-in is 0, then the operation performed will be A – B – 1.

How Borrow-In Affects Subtraction

When borrow mode is asserted, borrow-in set to 0 will make a SUBC execute like a SUB, and perform A – B. If borrow-in is 1, then the operation performed will be A – B – 1.

How Carry-Out is Formed During Subtraction

When borrow mode is not asserted, carry-out is the normal binary carry output of the ALU performing an add of the two operands after one of them has been complemented. Of course, the carry into the ALU can also affect the carry-out in the normal way, for SUBC and SUBRC.

How Borrow-Out is Formed During Subtraction

When borrow mode is asserted, the borrow output is the normal binary borrow-out of the ALU performing an add of the two operands after one of them has been complemented. Of course, the borrow into the ALU can also affect the borrow-out in the normal way, for SUBC and SUBRC.

How Difference Correct Works

The DIFFCORR instruction expects to be preceded by a subtraction class instruction. The subtraction class instruction will set up the nibble borrows, as well as produce a result of doing a binary subtract on two BCD numbers (or subtracting 1, 2, or 4 from a number in the case of the DECR instructions). The subtraction result is applied to the ALU as the operand to be corrected, and the DIFFCORR instruction uses the BCD nibble borrows to create a correction number. For each nibble borrow that is a 1, the correction nibble is 6. For each nibble borrow that is a 0, the correction nibble is 0. The correction word (made up of these 0s and 6s) is subtracted from the number to be corrected, using a binary subtract operation within the ALU. The subtraction is performed by complementing the correction word, and adding it to the word to be corrected. The carry into the ALU is set to 1 for this add. For the above examples, the correction cycles are as follows:

11) 5 – 3 = 0000 0010 (nibble-borrows are 0 0)

so correction value is 0000 0000
when complemented become 1111 1111

```
  0000 0010
+ 0000 0001
+ 1111 1111   ← (the complemented correction word)
  0000 0010   ← the correct answer, 2
```

12) 5 – 5 = 0000 0000 (nibble-borrows are 0 0)

so correction value is 0000 0000
when complemented become 1111 1111

```
  0000 0000
+ 0000 0001
+ 1111 1111   ← (the complemented correction word)
  0000 0000   ← the correct answer, 0
```

13) 3 – 5 = 1111 1110 (nibble-borrows are 1 1)

so correction value is 0110 0110
when complemented become 1001 1001

```
  1111 1110
+ 0000 0001
+ 1001 1001   ← (the complemented correction word)
  1001 1000   ← the correct answer, 98, which is – 2 in 10's complement
                notation.
```

How Carry-In Affects Difference Correct

Carry-in is ignored during DIFFCORR.

How Borrow-In Affects Difference Correct

Borrow-in is ignored during DIFFCORR.

How Carry-Out is Formed in Difference Correct

In carry mode (borrow mode signal not asserted) the carry-out is the complement value of the nibble-borrow status bit of the most-significant digit of the operation being performed. For 32-bit mode it is NC7, for 24-bit mode it is NC5, for 16-bit mode it is NC3, and for 8- bit mode it is NC1.

How Borrow Out is Formed in Difference Correct

In borrow mode (borrow mode signal is asserted) the borrow-out is the value of the nibble-borrow status bit of the most-significant digit of the operation being performed. For 32-bit mode it is NC7, for 24-bit mode it is NC5, for 16-bit mode it is NC3, and for 8-bit mode, it is NC1.

What happens to Subtraction in Non 32-Bit Mode

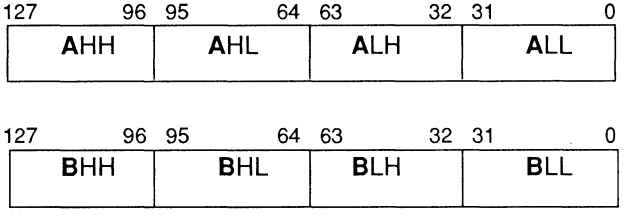
For ALU sizes other than 32 bits, the output to the Y bus is the expected 24, 16, or 8-bit result, merged with the unmodified high order bytes from one of the two operands. The nibble-borrows are set up correctly only for the enabled bytes. The nibble-borrows for the higher order bytes are also modified, but their value is undefined.

For the DIFFCORR instruction, only the active 8, 16, or 24 bits are corrected. The carry-out or borrow-out is selected from the appropriate position, as described in the previous section. The unselected high order bytes are passed through unmodified.

How to Do Multiprecision Addition and Subtraction

To do multiprecision BCD addition, these steps must be followed:

Assume two operands each of 128 bits, labeled as A and B. The 4 words that make up each operand are identified from most-significant to least-significant as HH, HL, LH, LL. thus the operands are:



The result will be called CHH, CHL, CLH, CLL.

```
ADD       TEMP = ALL,BLL
SUMCORR   CLL = TEMP
ADDC      TEMP = ALH,BLH,micro-carry
SUMCORR   CLH = TEMP
ADDC      TEMP = AHL,BHL,micro-carry
SUMCORR   CHL = TEMP
ADDC      TEMP = AHH,BHH,micro-carry
SUMCORR   CHH = TEMP
```

Note that each SUMCORR instruction generates a carry-out that is the value of the most-significant nibble-carry generated in the previous ADD or ADDC. The SUMCORR instruction stores this in the micro-carry flag of the ALU status register, which must then be selected as the source of the carry-for the following ADDC instruction. This assumes that the ALU width control is the same for both the ADD, ADDC, and SUMCORR instructions. There is no reasonable case where this would not be true.

Multiprecision subtract works exactly the same way except that SUB, SUBC, and DIFFCORR instructions are used.

Status During BCD Operations

The ALU status signals of C (carry/borrow), N (negative), Z (zero), and V (overflow), will correctly report the status of a BCD operation after the correction instruction, provided the BCD operands used have either a leading 0 (for positive numbers) or 9 (for negative numbers). This will mean that for 32-bit BCD numbers, there will only be sevev rather than eight useful digits. The rest of the number is still in 10's complement form. This reduces the range of numbers that can be represented. The tables below show the range of numbers and their representations for different byte-widths.

Byte-width = 1;

Number	Representation
–10	90
–9	91
–1	99
0	00
+1	01
+8	08
+9	09

Byte-width = 2;

Number	Representation
–1000	9000
–999	9001
–500	9500
–1	9999
0	0000
+1	0001
+500	0500
+998	0998
+999	0999

Byte-width = 3;

Number	Representation
–100000	900000
–99999	900001
–5000	995000
–1	999999
0	000000
+1	000001
+5000	005000
+99998	099998
+99999	099999

Byte–width = 0;

Number	Representation
–10000000	90000000
–9999999	90000001
–400000	99600000
–399999	99600001
–300	99999700
–1	99999999
0	00000000
+1	00000001
+300	00000300
+399999	00399999
+400000	00400000
+9999998	09999998
+9999999	09999999

The carry/borrow will be generated as described previously for SUMCORR and DIFFCORR.

The negative status is derived from the most-significant bit of the result. The most-significant result digit will be either 0 (for positive results), or 8 or 9 (for negative results). The most-significant bit of 0 is clear, so positive is reported. Both 8 and 9 have their most-significant bit set, so negative is reported.

The overflow indication (V) is the exclusive-or of the most-significant and next- most-significant nibble-carry/borrow flag. For 32-bit mode it is "NC7 xor NC6", for 24-bit mode it is "NC5 xor NC4", for 16-bit mode it is "NC3 xor NC2", and for 8-bit mode it is "NC1 xor NC0".

The zero (Z) indication is given when all enabled bytes of the result are zero. For other than 32-bit mode, only the specified width is tested for zero.

APPENDIX B

Multiplication

The ALU has a number of steps to support signed and unsigned single precision multiplication. Multiprecision multiplication is not directly supported since it is somewhat easily implemented in microcode using the single precision steps.

The algorithm used for both signed and unsigned multiplication is a modified Booth algorithm which yields two product bits per microcycle. The ALU uses two extra bits of precision appended to the most-significant byte of the operation. The algorithm operates under the control of the M flag and the two least significant bits of the Q register. Flow charts 1 through 5 outline the multiplication algorithm.

Unsigned Multiplication Example

The following steps show an example for a 1-byte unsigned multiplication. The multiplier is 65_H. The multiplicand is 48_H. Refer to Figure B-1.

Instruction	Opcode	A	B	Y	Comment
1. LOADQ-A	06	65	XX	65	Load multiplier into Q
2. UMULFIRST	5B	48	00	12	First multiplication step
3. UMULSTEP	5C	48	12	16	Inner multiplication loop, iteration 1
4. UMULSTEP	5C	48	16	E1	Inner multiplication loop, iteration 2
5. UMULSTEP	5C	48	E1	1C	Inner multiplication loop, iteration 3
6. UMULLAST	5D	48	1C	1C	Last multiplication step yields high byte
7. PASS-Q	05	XX	XX	68	Low product byte from Q register

The product is $1C68_H$. The high byte is available on the Y bus during the last multiplication step in step 6. The low byte is available from the Q reg. in step 7.

Division

The ALU has a number of steps to support division for single and multiprecision signed and unsigned integers. Divisors and dividends of any number of bytes are supported.

The algorithm used for signed division is a four quadrant nonrestoring divide algorithm which yields one quotient bit per microcycle for single precision division. Unsigned division is implemented using the same algorithm except that the sign of the partial remainder is stored in a separate bit of the status register (the S flag). In either case, the algorithm operates under control of the M bit of the status register. Flow charts B-6 and B-7 outline the divide algorithm.

Unsigned Division Example

The following steps show an example for a 1-byte unsigned division. The dividend is 75_H. The divisor is 02_H. Refer to flow chart 6.

Instruction	Opcode	A	B	Y	N Flag	Comment
1. LOADQ-A	06	75	XX	75	X	Load dividend into Q
2. UDIVFIRST	4F	02	00	00	X	First division step
3. UDIVSTEP	54	02	00	FD	X	Inner division loop, iteration 1
4. UDIVSTEP	54	02	FD	FF	X	Inner division loop, iteration 2
5. UDIVSTEP	54	02	FF	03	X	Inner division loop, iteration 3
6. UDIVSTEP	54	02	03	02	X	Inner division loop, iteration 4
7. UDIVSTEP	54	02	02	01	X	Inner division loop, iteration 5
8. UDIVSTEP	54	02	01	FE	X	Inner division loop, iteration 6
9. UDIVSTEP	54	02	FE	01	X	Inner division loop, iteration 7
10. UDIVLAST	55	02	01	FF	1	Last division step
11. REMCORR	58	02	FF	01	X	Remainder correction
12. PASS-Q	05	XX	XX	3A	X	Quotient from Q register

The quotient is $3A_H$ obtained from the Q register in step 12 and the remainder is 01_H obtained on the Y bus in step 11.

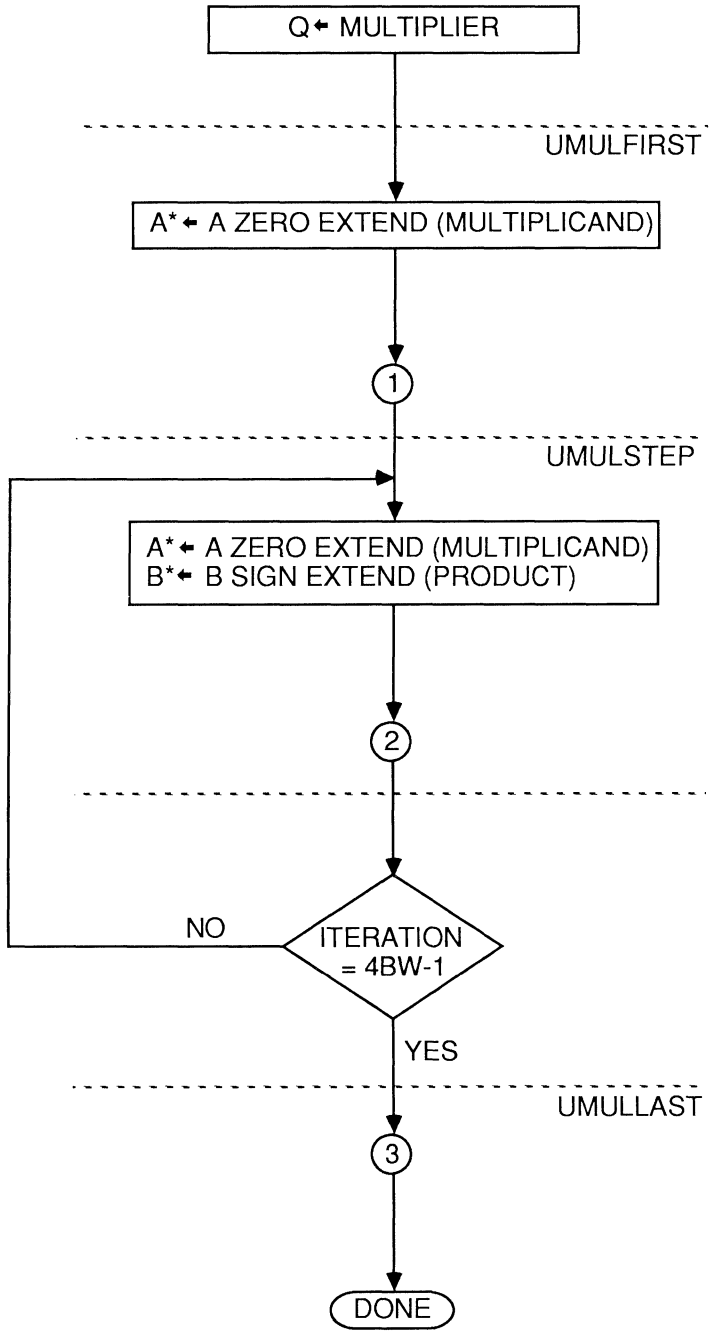


Figure B-1. Algorithm for Unsigned Multiplication

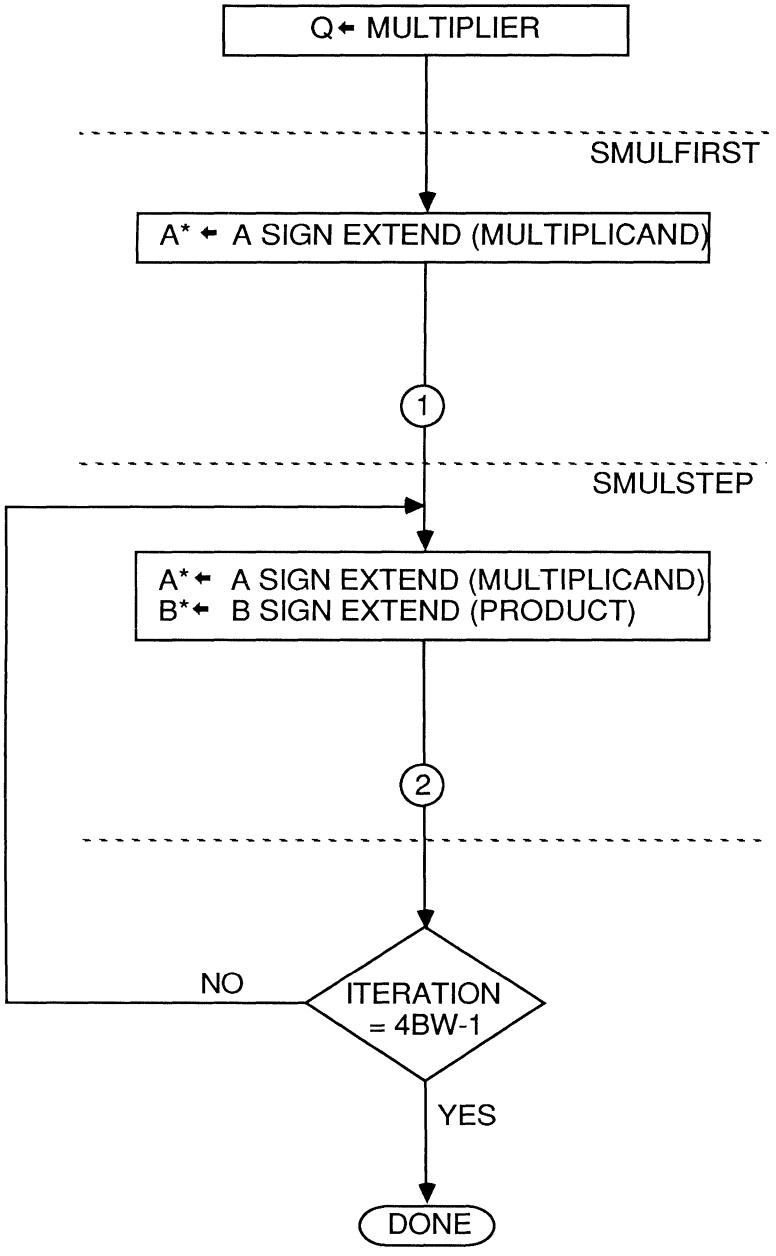


Figure B-2. Algorithm for Signed Multiplication

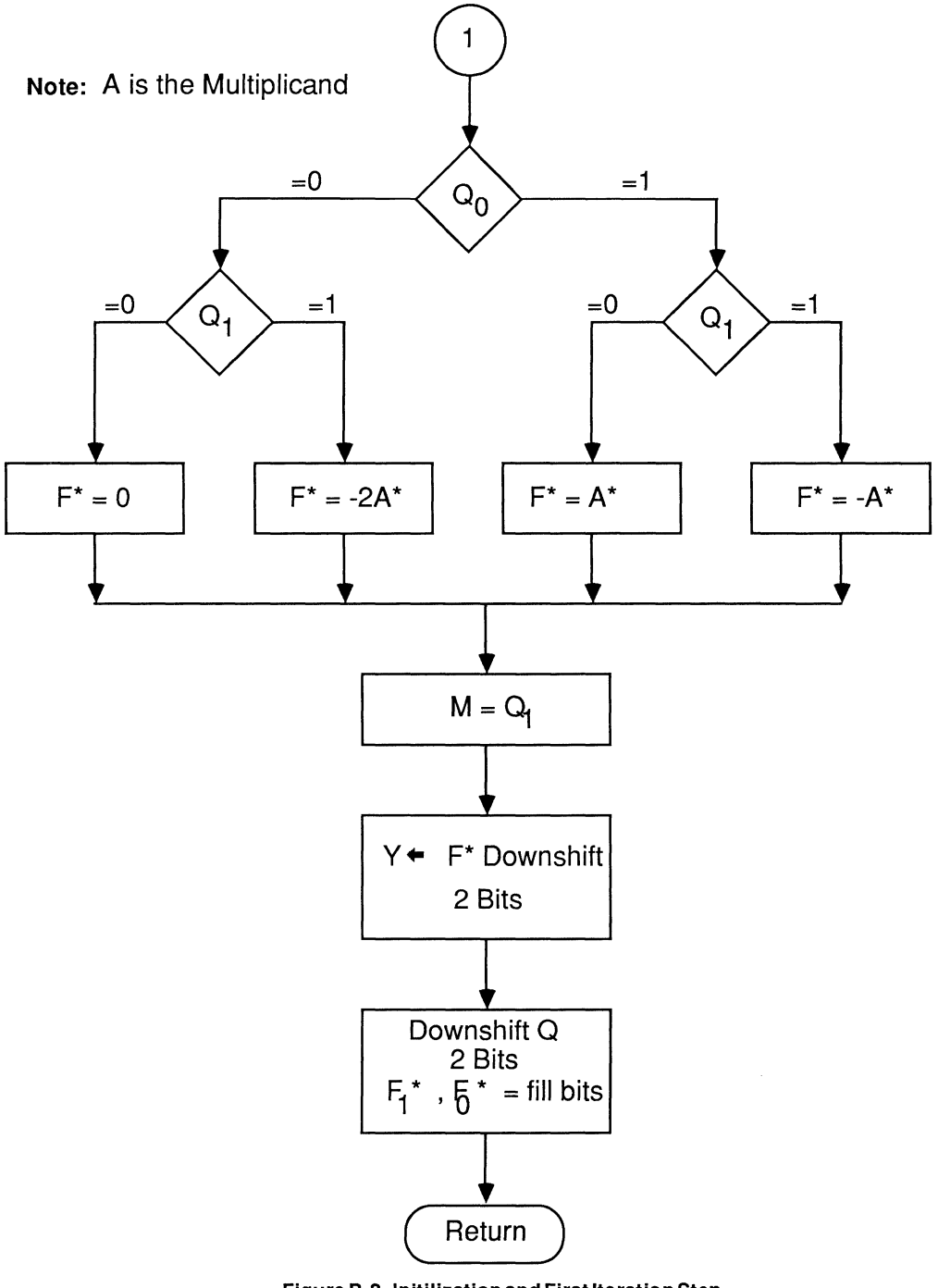


Figure B-3. Initilization and First Iteration Step

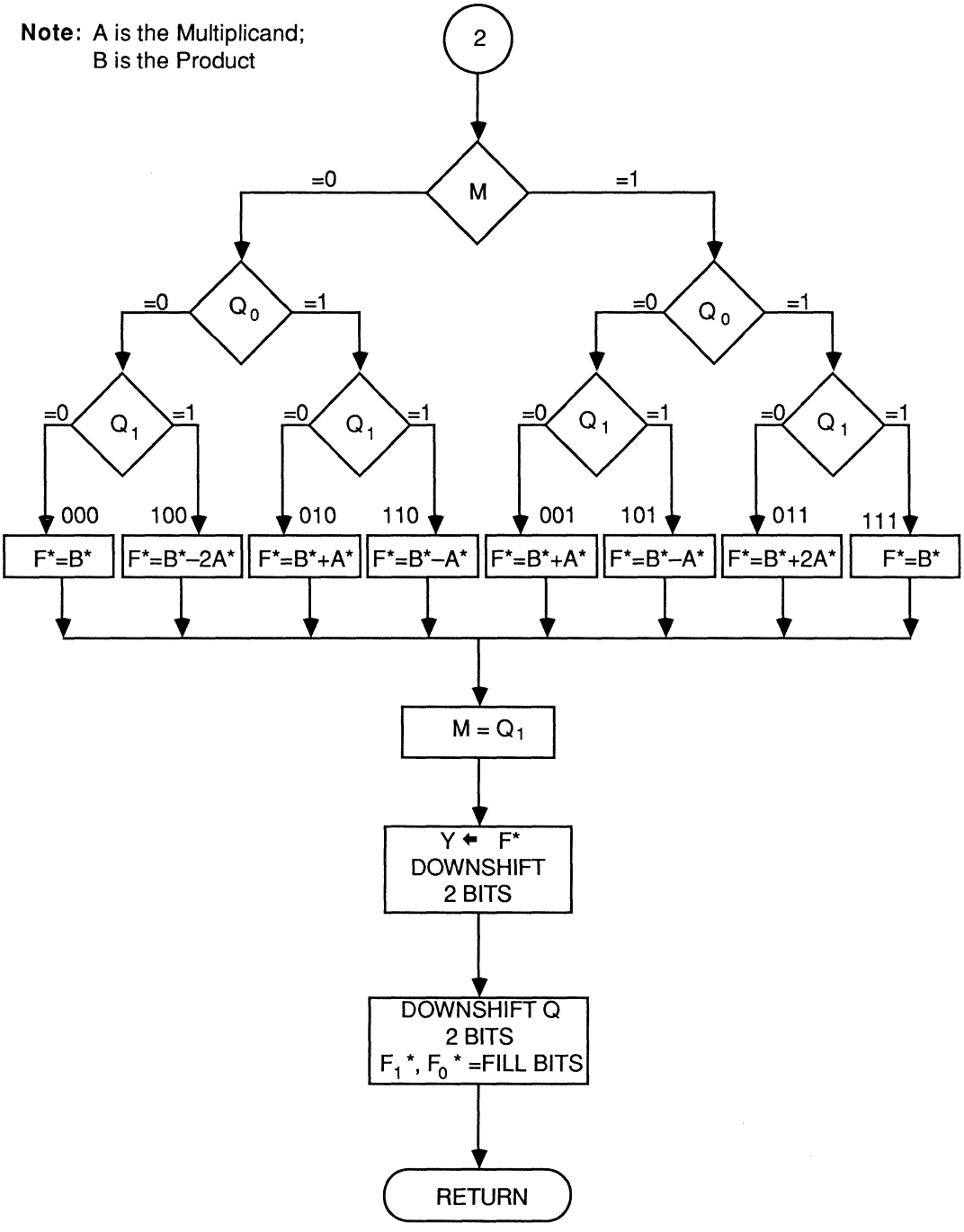


Figure B-4. Iteration Step

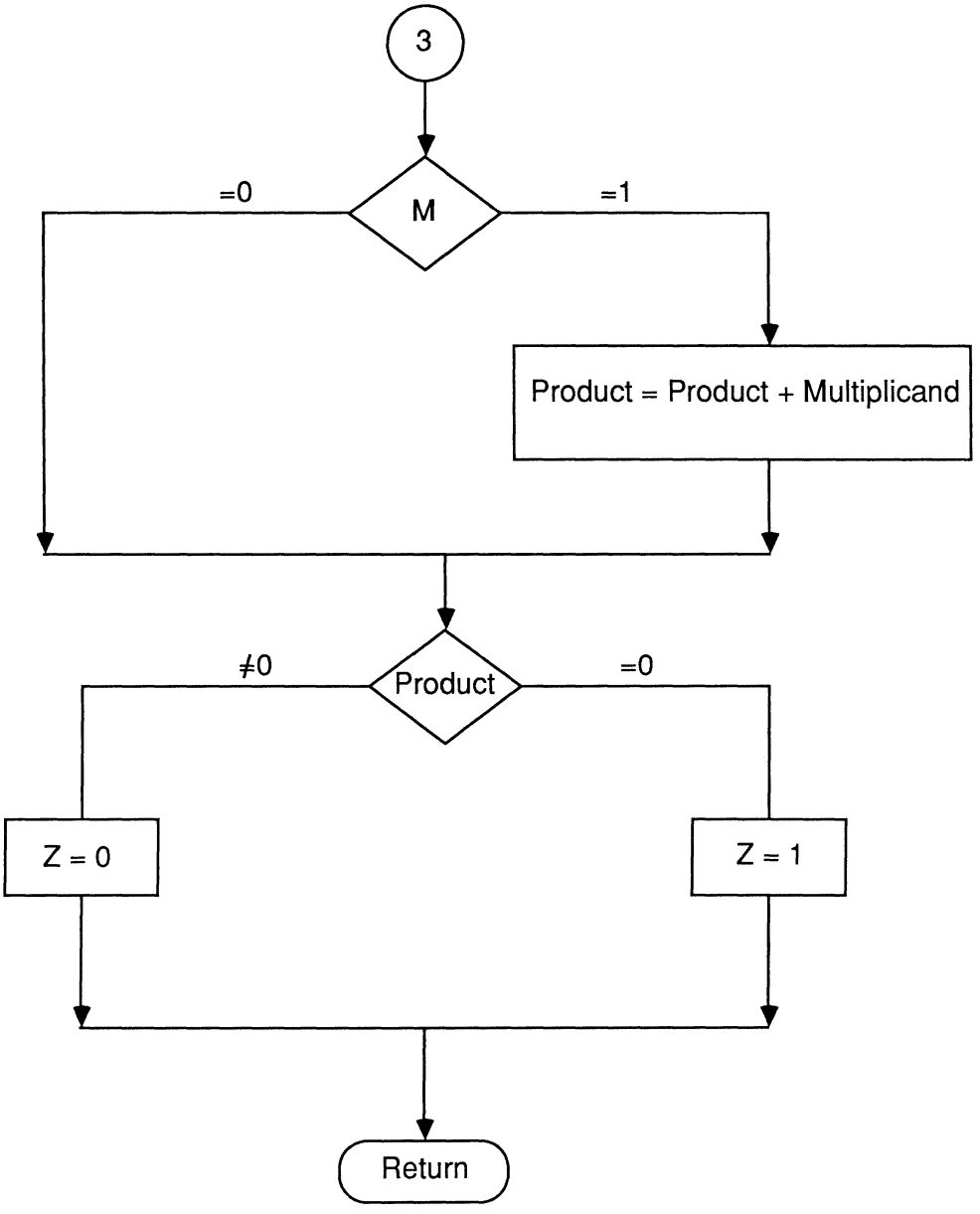


Figure B-5. Final Correction Step for Unsigned Multiply

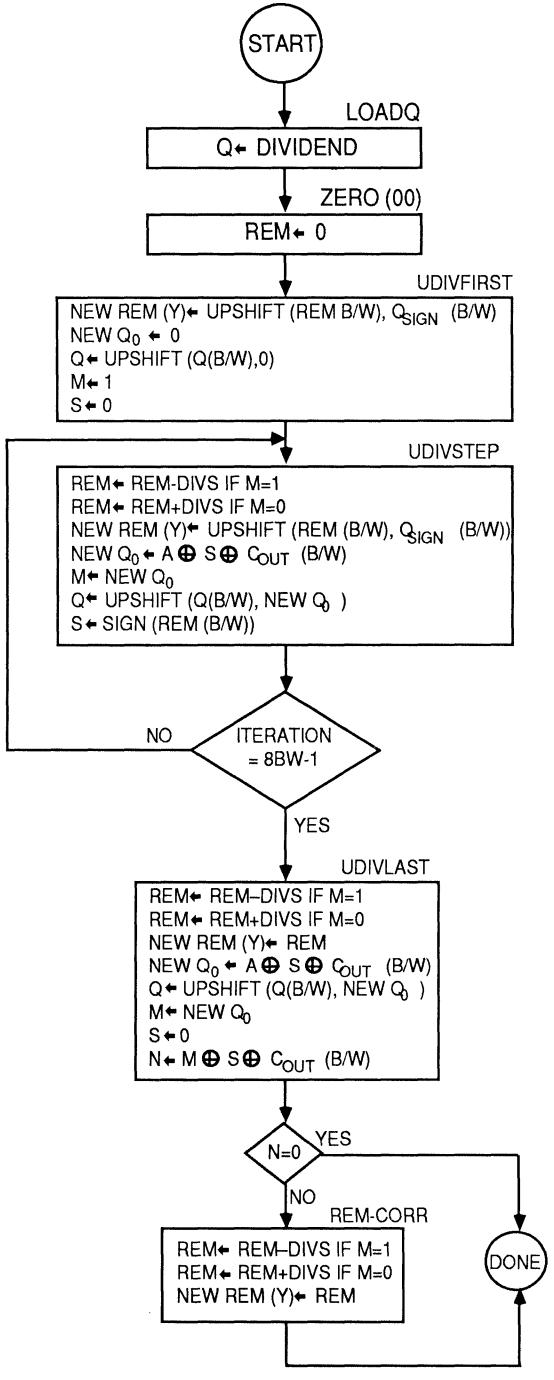


Figure B-6. Unsigned Division

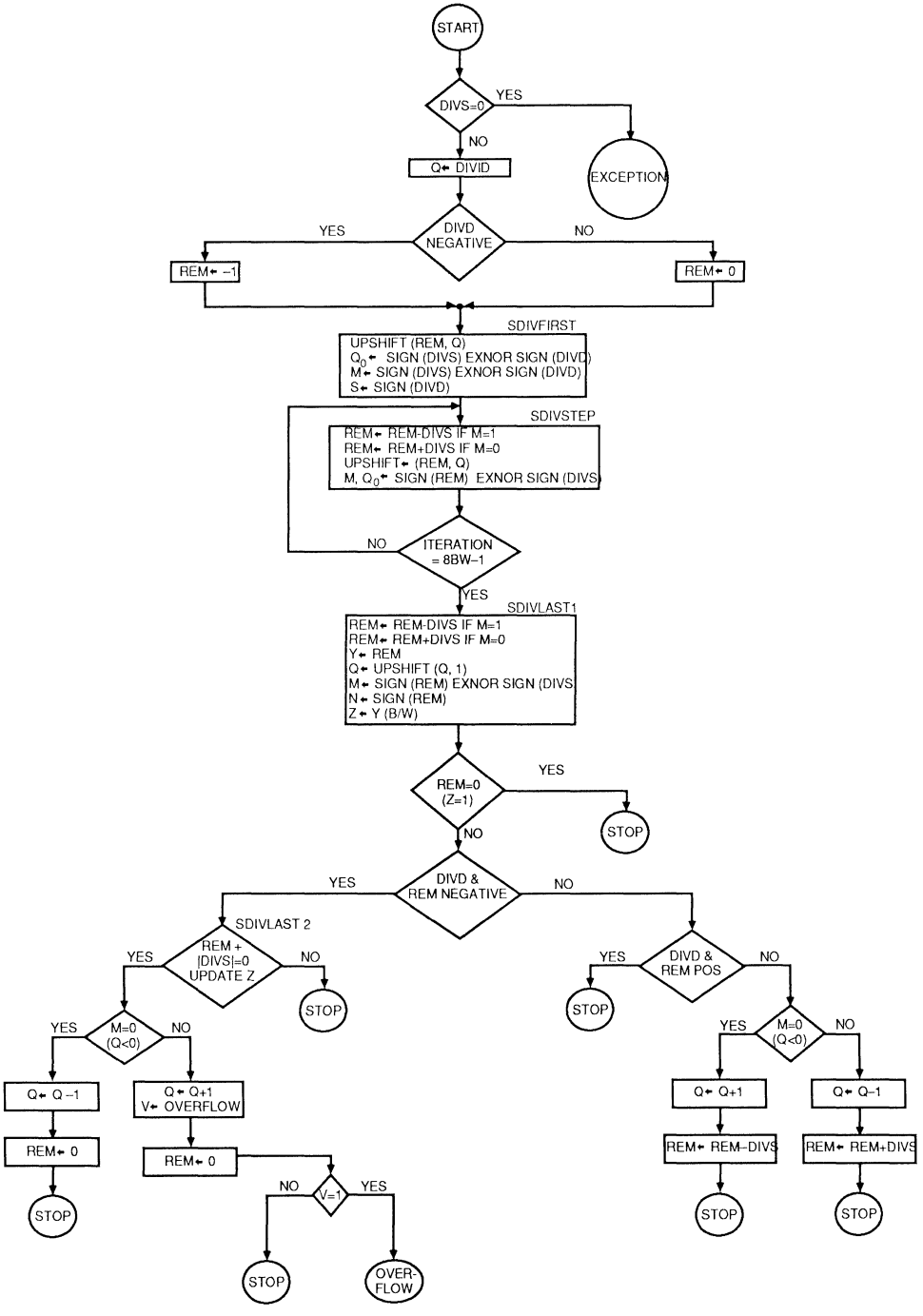


Figure B-7. Single-Precision Unsigned Division

NOTES

NOTES

ADVANCED MICRO DEVICES' NORTH AMERICAN SALES OFFICES

ALABAMA ..... (205) 882-9122
ARIZONA ..... (602) 242-4400
CALIFORNIA,
Culver City ..... (213) 645-1524
Newport Beach ..... (714) 752-6262
San Diego ..... (619) 560-7030
San Jose ..... (408) 452-0500
Woodland Hills ..... (818) 992-4155
CANADA, Ontario,
Kanata ..... (613) 592-0060
Willowdale ..... (416) 224-5193
COLORADO ..... (303) 741-2900
CONNECTICUT ..... (203) 264-7800
FLORIDA,
Clearwater ..... (813) 530-9971
Ft Lauderdale ..... (305) 776-2001
Melbourne ..... (305) 729-0496
Orlando ..... (305) 859-0831
GEORGIA ..... (404) 449-7920
ILLINOIS,
Chicago ..... (312) 773-4422
Naperville ..... (312) 505-9517
INDIANA ..... (317) 244-7207
KANSAS ..... (913) 451-3115
MARYLAND ..... (301) 796-9310
MASSACHUSETTS ..... (617) 273-3970
MINNESOTA ..... (612) 938-0001
MISSOURI ..... (913) 451-3115
NEW JERSEY ..... (201) 299-0002
NEW YORK,
Liverpool ..... (315) 457-5400
Poughkeepsie ..... (914) 471-8180
Woodbury ..... (516) 364-8020
NORTH CAROLINA ..... (919) 878-8111
OHIO ..... (614) 891-6455
Columbus ..... (614) 891-6455
Dayton ..... (513) 439-0470
OREGON ..... (503) 245-0080
PENNSYLVANIA,
Allentown ..... (215) 398-8006
Willow Grove ..... (609) 662-2900
TEXAS,
Austin ..... (512) 346-7830
Dallas ..... (214) 934-9099
Houston ..... (713) 785-9001
WASHINGTON ..... (206) 455-3600
WISCONSIN ..... (414) 792-0590

ADVANCED MICRO DEVICES' INTERNATIONAL SALES OFFICES

BELGIUM,
Bruxelles ..... TEL ..... (02) 771 91 42
FAX ..... (02) 762 37 12
TLX 61028
FRANCE,
Paris ..... TEL ..... (1) 49-75-10-10
FAX ..... (1) 49-75-10-13
TLX 263282
WEST GERMANY,
Hannover area ..... TEL ..... (0511) 736085
FAX ..... (0511) 721254
TLX 922850
München ..... TEL ..... (089) 41 14-0
FAX ..... (089) 406490
TLX 523883
Stuttgart ..... TEL ..... (0711) 62 33 77
FAX ..... (0711) 625187
TLX 721882
HONG KONG ..... TEL 852-5-8654525
FAX 852-5-8654335
TLX ... 67955AMDAPHX
ITALY, Milano ..... TEL ..... (02) 3390541
(02) 3533241
FAX ..... (02) 3498000
TLX 315286
JAPAN,
Kanagawa ..... TEL 462-47-2911
FAX 462-47-1729
Tokyo ..... TEL ..... (03) 345-8241
FAX ..... (03) 342-5196
TLX ... J24064AMDTKOJ
Osaka ..... TEL 06-243-3250
FAX 06-243-3253
KOREA, Seoul ..... TEL 82-2-784-7598
FAX 82-2-784-8014
LATIN AMERICA,
Ft. Lauderdale ..... TEL ..... (305) 484-8600
FAX ..... (305) 485-9736
TLX .. 5109554261 AMDFTL
NORWAY,
Hovik ..... TEL ..... (02) 537810
FAX ..... (02) 591959
TLX 79079
SINGAPORE ..... TEL 65-2257544
FAX 65-2246113
TLX ..... RS55650 MMI RS
SWEDEN, Stockholm ..... TEL ..... (08) 733 03 50
FAX ..... (08) 733 22 85
TLX 11602
TAIWAN ..... TLX 886-2-7122066
FAX 886-2-7122017
UNITED KINGDOM,
Manchester area ..... TEL ..... (0925) 828008
FAX ..... (0925) 827693
TLX 628524
London area ..... TEL ..... (04862) 22121
FAX ..... (0483) 756196
TLX 859103

NORTH AMERICAN REPRESENTATIVES

CALIFORNIA
I^2 INC ..... OEM (408) 988-3400
DISTI (408) 498-6868
CANADA
Burnaby, B.C.
DAVETEK MARKETING ..... (604) 430-3680
Calgary, Alberta
DAVETEK MARKETING ..... (604) 430-3680
Kanata, Ontario
VITEL ELECTRONICS ..... (613) 592-0090
Mississauga, Ontario
VITAL ELECTRONICS ..... (416) 676-9720
Quebec
VITEL ELECTRONICS ..... (514) 636-5951
IDAHO
INTERMOUNTAIN TECH MKGT ..... (208) 888-6071
INDIANA
ELECTRONIC MARKETING
CONSULTANTS, INC. ..... (317) 253-1668
IOWA
LORENZ SALES ..... (319) 377-4666
KANSAS
LORENZ SALES ..... (913) 384-6556
KENTUCKY
ELECTRONIC MARKETING
CONSULTANTS, INC. ..... (317) 253-1668
MICHIGAN
SAI MARKETING CORP ..... (313) 750-1922
MISSOURI
LORENZ SALES ..... (314) 997-4558
NEBRASKA
LORENZ SALES ..... (402) 475-4660
NEW MEXICO
THORSON DESERT STATES ..... (505) 293-8555
NEW YORK
NYCOM, INC ..... (315) 437-8343
OHIO
Centerville
DOLFUSS ROOT & CO ..... (513) 433-6776
Columbus
DOLFUSS ROOT & CO ..... (614) 885-4844
Strongsville
DOLFUSS ROOT & CO ..... (216) 238-0300
PENNSYLVANIA
DOLFUSS ROOT & CO ..... (412) 221-4420
UTAH
R^2 MARKETING ..... (801) 595-0631

ADVANCED MICRO DEVICES **901 Thompson Pl., P.O. Box 3453, Sunnyvale, CA 94088, USA**
TEL: (408) 732-2400 • TWX: 910-339-9280 • TELEX: 34-6306 • TOLL FREE: (800) 538-8450
APPLICATIONS HOTLINE **TOLL FREE: (800) 222-9323 • (408) 749-5703**

WCP-2500-6/88-1
Printed in U.S.A.

ADVANCED MICRO DEVICES, INC.
901 Thompson Place
P.O. Box 453
Sunnyvale,
California 94086
(408) 732-2400
TWX: 910-339-9280
TELEX: 34-6306
TOLL FREE
(800) 538-8450

09287B