INFORMAL DESCRIPTION OF THE SCC 6700

Butler W. Lampson

University of California, Berkeley

Document No. W-41

Issued March 6, 1968

Contract SD-185

Office of Secretary of Defense

Advanced Research Projects Agency

Washington, D. C. 20325

INFORMAL DESCRIPTION OF THE SCC 6700

The Scientific Control Corporation 6700 is a large-scale general purpose computer specifically designed for time-sharing. It is the product of a joint effort of Project Genie and SCC personnel. One of these machines, together with accompanying drums, disks and memory, has been ordered by the project. It is hoped that the system will be operational by the first quarter of 1969.

This document is an informal reference manual for the machine. It describes the overall system organization, the CPU, the memory system and the input/output at a level of detail which is intended to be the maximum of what a systems programmer might want to know and what is currently decided. It is not to be taken as certain truth.

Overall design, Main memory

The 6700 is a memory-centered system. It consists of a large number of more or less independent devices which communicate with each other primarily through memory. There are a few other channels of communication: the ACT bus, the PRO bus and wakeup lines.

The main memory of the system is 128K (expandable to 512K) of core organized into 8 modules of 16K each. The cycle time of the memory is 1000 ns; the read access time is 400 ns. The memory is 52 bits wide; there is room for 2 of the machine's 24-bit words and 4 parity bits.

Each module of the memory is connected to a fast memory which consists of six address registers and six double word data registers, one for each address register. All communication between the outside world and the core takes place through the fast memory.

Each fast memory has four ports, i.e., four independent mechanisms for presenting addresses to the memory and transmitting

25-bit words and control information. A device references the memory by presenting an address and six control bits, which are used in the following way:

One bit is a memory access priority bit. If it is on, the request has high priority for access to the fast memory. This means that if another fast memory request is presented to the module in the same cycle (100 ns) which does not have the memory access priority bit set, the request which does have the bit set will win. The other request can be made again in the next 100 ns cycle. If two requests have the same memory access priority, the one is chosen which comes in at the port having the highest priority. The priorities of the ports are fixed.

Two bits specify the core access priority as Low, Warning, Medium or High. If the request requires a core access, these bits determine how it fares in competition with other requests requiring core accesses. W is equivalent to L except that it causes any pending W requests to be converted to H; it is used by the drum transfer unit.

Three bits describe the nature of the request. They are called Fetch, Store and Hold. Five combinations of these bits are normally used:

F	S	H	
0	0	1	Pre-store. Obtains a register in fast memory for the address and sets its hold bit. The register will not be released, unless its hold bit is cleared by another request with the same address, until 10 μs has elapsed. No core reference is initiated by this request.
0	1	0	Store. Takes 24 bits from the data bus and puts them into a fast memory register. The data will be stored into core as soon as the core access priority gets it a core access. The hold bit on the register is cleared.

0 1 1 Store and hold. Same as store except that the hold bit is set.

1 0 1 Pre-fetch. Obtains a register in fast memory for the address and sets its hold bit. The memory will make a core access if necessary to make the data portion of the register agree with the contents of core.

1 0 0 Fetch. If data is ready in the fast memory, it is transferred to the bus. The hold bit is cleared.

Each module of the memory listens to requests at its four ports in every 100 ns cycle. It accepts at most one request; the others are rejected within the same cycle. If a request is accepted, data and addresses are transferred in the following cycle. Thus, an accepted request takes 200 ns, a rejected one 100 ns. This is not a complete description of memory bus timing.

The fast memory attempts to keep its contents in agreement with core, doing fetches or stores from core as necessary. If the memory is left alone for six cycles, it will be able to make all the core references it needs. There will then be 48 double words in fast memory. They will stay there until displaced by later requests. A program operating in a tight loop on a small volume of data may be able to approximate this situation quite closely.

The following devices are attached to the main memory:

The CPU

The drum and disk transfer units

The drum and disk controller (AMC or auxiliary memory controller)

The process control unit (PCU)

The character input/output controller (CHIOC)

Any other I/O devices which may be attached to the system.

There is an ACTivate bus which connects the CPU to the two controllers, the PCU and any other I/O equipment which may be attached. There is also a PROtect bus which connects the CPU, the PCU, the controllers and any other device which needs to set the protect signal.

The CPU

We will discuss the capabilities of the CPU under the following headings:

addressing and mapping

instruction set

input/output

Because of the highly interdependent nature of the system it will be necessary to use certain terms which are not defined until later in the discussion.

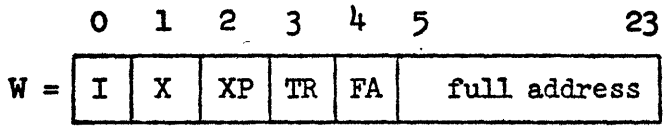
Addressing

Each process has available to it a 2↑19 word virtual memory which is divided into 32 units of 16K words each called chunks. A full address is therefore 19 bits, divided into a 5-bit chunk number and a 14-bit word address within the chunk. A 19-bit field can be interpreted as a full address in the following contexts:

1) As part of an indirect address word.
2) As part of the X register during indirect addressing or in executing LFB or SFB.
3) As part of a word in a pop transfer vector.
4) As the operand of BRR or XCI or BSR.

Since case (1) is by far the most important we will consider it in detail and then point out the ways in which the other cases differ.

An indirect address word has the following format:



The I bit specifies another level of indirection. The X bit specifies indexing. The TR bit specifies that a trap should occur. The FA bit specifies that this level of indirection provides a full address. The XP bit specifies that the X register is used as a pointer, i.e., that it provides the chunk number.

Indirection proceeds in the following manner. If W has been fetched as an indirect word from address Q:

Trap if TR = 1 (see below)

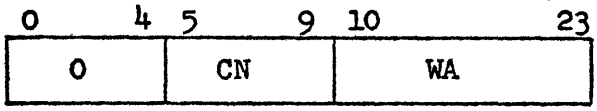
Set WA = W_{10-23} if X = 0, $W_{10-23} + X_{10-23}$ if X = 1

Set CN = Q_{5-9} if FA = 0

W_{5-9} if FA = 1 and XP = 0

X_{5-9} if FA = 1 and XP = 1

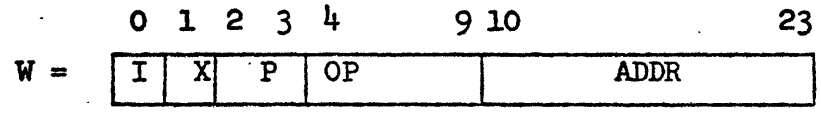
The effective address at this level is (CN,WA), by which we mean the 24-bit quantity:



If I = 1, we use this word as the address of another indirect word. Otherwise, we return it as the effective address.

The effect of all this is that if FA = 0 indirection proceeds exactly as in the 940, except for the different arrangement of the bits and the presence of the trap bit. The address produced is in the same chunk as the indirect word. If FA = 1, the address produced is in a different chunk, either the one specified by the indirect word or the one specified by X, depending on the setting of XP.

An instruction word has the format:



If P = 0, the OP field specifies one of 64 instructions to be executed. Every instruction <u>without exception</u> generates an effective address in the same way:

1) Let CN = L_{5-9} (chunk number of the location counter)

2) Let WA = ADDR if X = 0, ADDR+X_{10-23} if X = 1

3) Go indirect through the address (CN,WA) if I = 1. Otherwise, take (CN,WA) as the effective address.

Note that this is equivalent to treating the instruction word with bits 2-9 cleared as an indirect word.

Once the effective address Q has been generated, execution of the instruction depends on the P and OP fields. This subject is discussed in the next section.

Addressing from one chunk to another is controlled by a 32 X 32 matrix called the inter-chunk protection (ICP) matrix. An entry in this matrix contains two bits and determines the legality of an inter-chunk reference as follows:

0 no access is allowed
1 read access only
2 read and indirect access
3 any kind of access

A reference made to obtain a word used in indirection is an indirect reference. A reference made to obtain an instruction is an execute reference. One made to fetch an operand is a read reference, and one made to store a data word is a write reference. The entry of the ICP matrix to be used in checking the legality of a reference is determined by the chunk containing the word which provide the address (source chunk) and the chunk containing the word addressed (target chunk). Thus, in the sequence

chunk M1	M	LDA*	N	
	N	IA*	O1, O2	this is an indirect address directive. The operands are chunk number and word address respectively.
chunk O1	0	IA	P1, P2	
chunk P1	P	DATA	24	

execution of the instruction at M loads 24 into A provided

M1 has indirect access to O1, or ICP[M1,O1] $\geq$ 2

O1 has read access to P1, or ICP[O1,P1] $\geq$ 1

Note that the value of ICP[M1,P1] is not in question.

A chunk is paged. The nominal page size of the chunk may vary from 128 words to 2048 words. Each virtual page may have one of three values:

empty - no real memory corresponds to this virtual memory

a real page reference

an indirect page table reference

If the value is a real page reference, it has two parts: a drum address which specifies the page and three protection bits: W, X, P.

W allows writing into the page if it is on.

X allows instruction words to be fetched from the page if it is on.

P allows privileged instructions to be fetched if it is on.

Note that the protection bits are associated with the entry for the page in the map and not with the physical page itself.

Converting a full address (CN,WA) into a physical address requires the following steps:

Obtain the CNth entry from the chunk table for the process. This tells where the page table for chunk CN is and what its nominal page size is. Call the NPS P.

Obtain the (WA/P)th word from the page table. If its value is a real page, this gives us a drum address for the page.

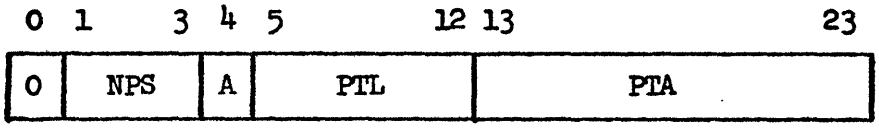
Look up D in a hash table called DHT. If it is there, the associated information tells where the page is in core and what state it is in.

This process is described in complete detail below.

If the value of a virtual page is an indirect page table reference, it also has two parts: an actual page size < the NPS and an address which tells where to find a small page table containing real page pointers. If the NPS is 1024 and the APS is 256, the small table contains four entries for four 256 word real pages which make up the 1024 word nominal page. Each of these entries is 0 or a real page reference.

A precise understanding of how the map works can be obtained from the bit-by-bit description which follows. The above general description should suffice for most purposes.

Each process has a 32-word chunk table; one word of this table either contains 0, indicating that no memory is assigned to the chunk, or points to the page table for the chunk. The format of a page table pointer is



NPS = nominal page size for this chunk. This number determines the division of the word address into page number and word in page. The nominal page size is $32 * 2^{NPS}$, and NPS ranges between 1 (64 words, not implemented) and 6 (2048 words). NPS = 7 is reserved.

A = absolute address bit. PTA is interpreted as an absolute address if this bit is set, as an address in the context block if it is clear.

PTL = length of page table (in words) -1.

PTA = address of page table, modified by A.

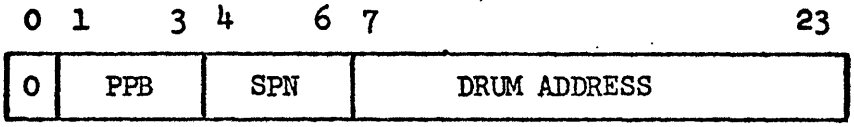
A page table starts with four words which contain one row of the ICP matrix, packed eight entries per word, two bits per entry in the bottom 16 bits. The interpretation of an entry is:

00 - no access
01 - read only
10 - read and indirect
11 - free access

} describing access <u>from</u> this chunk <u>to</u> another one

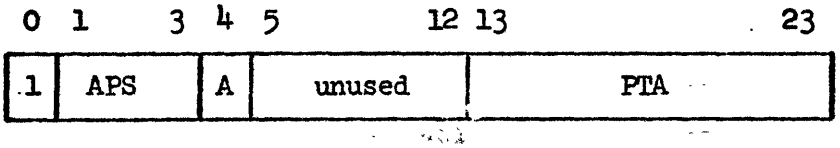
The remainder of the page table, PTL words of it, contains entries of three types: empty, real page reference, or indirect page table reference. The pointer from the chunk table is to the word following the four ICP words.

An empty entry is 0. A real page reference has the form



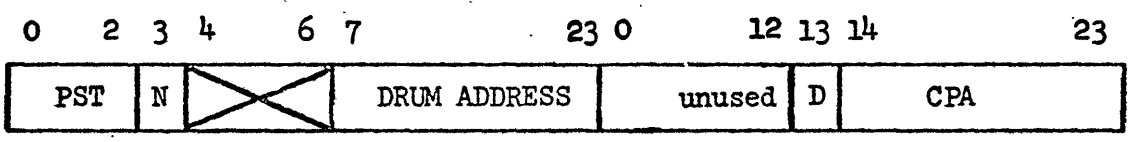
PPB = page protection bits
bit 1 = P: privileged instruction authorization
bit 2 = X: instruction fetch authorization
bit 3 = W: write authorization
Reading is always allowed
SPN = sub-page number. See the discussion of real address formation below.

An indirect page table reference has the form



APS = actual page size
A, PTA have the meanings ascribed above
This entry points to a small page table with page size specified by APS. This table has only empty or real page entries. It contain NPS/APS words. We require APS < NPS.

There is a hash table called DHT which keeps track of the drum pages which are in core. The key is the drum address. A linear collision doctrine is used. An entry in this table occupies two words and has the form:



PST = page status

0 - on drum

1 - read scheduled

2 - read in progress

3 - read error

4 - in core

5 - write scheduled

6 - write in progress

7 - write error

Note that a page is in core and readable if and only if the first bit of PST is on.

N = no write. A write into this page will not be allowed if this bit is set. The primary purpose is to prevent a swapper decision to write the page from being subverted.

D = dirty. Set if the page in core may differ from the drum version.

CPA = core page address. The top 10 bits of a 19 bit real core address.

A reference to the map in core proceeds as follows: starting with a 24-bit full address word A, use A_{5-9} to select a chunk table entry. If it is 0, generate trap M1. Otherwise, let PS = NPS and P = $A_{10-(18-PS)}$. If P > LPT, generate trap M2. Otherwise, examine the Pth entry in the page table addressed by PTA+4. If the entry is empty, generate trap M3. If it is an indirect page table reference, let Q = $A_{(18-NPS+1)-(18-APS)}$ and PS = APS, and examine the Qth entry in small page table addressed by the indirect reference.

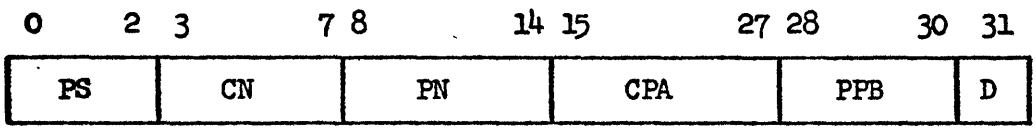
When a real page reference is found, look up the drum address in DHT. If it is not found, generate trap M4. If the first bit of PST in the entry found is 0, generate trap M5. Otherwise, compute the real address as

$$CPA*2^9+SPN*2^6+A_{(18-PS+1)-23}$$

Note that the + signs may be taken as merge rather than add because of the way in which these quantities are obtained.

A reference to the map must specify whether the address is to be used for writing. If so, D is turned on in the DHT entry found unless N is set, in which case trap M5 is generated.

Since it is not acceptable to subject every memory reference to this time-consuming process, the machine is equipped with eight associative registers to hold information about the most recently used core pages. One of these registers has the form:



PS = page size. Same code as for NPS above.
CN = chunk number.
PN = page number.
CPA = core page address/64. Bit 27 is set to 0 permanently.
PPB = page protection bits
D = dirty bit from DHT

When an address is presented to the map unit, an association is done on the CN and PN fields of each register. The bits of the address used in this association are determined by the PS field. If the association is successful, the real address is computed as CPA*64 merged with the bits of the address not used for the association. The PPB field is also returned so that the processor can check the protection.

With the address must come a bit which specifies whether a load or store is being done. If the latter, D must be set when the association succeeds. Otherwise, that register is cleared and the map unit proceeds as though association had failed. This is to ensure that the D bit in DHT is set whenever a store is done. The final state of D is returned.

If the association fails, the core map is referenced using the algorithm described above. If it is successful, an associative register is chosen and its former contents erased, and its fields are set from the results of the reference to the core map:

PS $\leftarrow$ PS computed during map reference

CN $\leftarrow A_{5-9}$

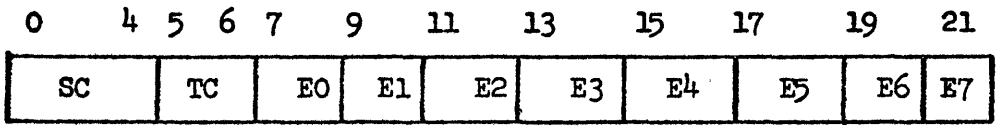
PN $\leftarrow A_{10-(18-PS)}$

CPA ← CPA in the DHT entry * 8 + SPN

PPB ← PPB in the page table entry

D ← D in the DHT entry

In parallel with the association for core address is one for ICP bits. There is another set of registers of the form:



SC = source chunk number

TC = top two bits of target chunk number

Ei = ICP [SC,TC*8+i]

I.e., if association on SC and TC succeeds, the last three bits of the target chunk number are used to select the proper Ei.

If association fails, a reference to the core map must be made. The page table for chunk SC is found as before, and PTA+TC is fetched.

The overall function of the map is described by the following table:

Input	Output
Source chunk SC	ICP bits (2)
Target chunk TC	PPB (3)
Target word address WA	Real core address (19)
Load/store indication	D bit (1)

There is an ACT instruction for the map-loader called

CVRA Convert Virtual to Real address

It accepts one argument, namely a 19 bit virtual address, and returns either:

a) A noskip if the page containing this address is not in the map, or

b) a skip and

1) The drum address of the page

2) The real address of its DHT entry, or 0 if it is not

in DHT

3) The real address corresponding to the virtual address if the page is in core, or 0 otherwise.

In other words, this instruction makes available most of the results of the mapping operation, so that the processor does not have to duplicate the computation made by the map loader. Under no circumstances does it cause a trap.

Instructions

We will describe the 6700 instruction set in terms of its differences from that of the 940. Notation: L = 24-bit program counter, Q = 24-bit effective address, (Z) means contents of memory location addressed by Z. Needless to say, all addressing is mapped. Assume L ← L+1 in every instruction description unless otherwise stated. Any add to L or Q is a 14-bit add; the chunk number is unaffected.

General

1. The format of an instruction word is different. See above.
2. The effective address is computed in exactly the same way for every instruction.
3. Indirect addressing is handled quite differently. See above.

Symbols

L = 24 bit location (program) counter

A,B,X = A,B,X registers

Q = 24 bit effective address

OV = overflow bit

CO = carry bit from 24th bit of adder. Also used by floating point instructions.

CI = Carry into bit 0 of adder.

PRO = protect signal

Loads and Stores

New instructions:

XMX Exchange memory and X

$X \rightarrow (Q)$, $(Q) \rightarrow X$

STM Store masked

$(Q) \wedge \overline{B} \vee A \wedge B \rightarrow (Q)$

I.e., store the bits of A selected by 1 bits in B.

LDD Load double

$(Q) \rightarrow A$; $(Q+1) \rightarrow B$

STD Store double

$A \rightarrow (Q)$; $B \rightarrow (Q+1)$

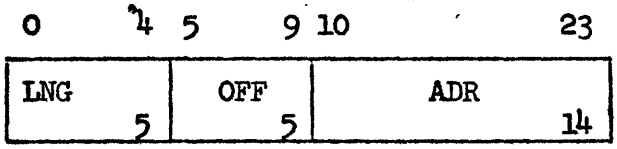
STZ Store zero

$0 \rightarrow (Q)$

Field Instructions

There are six new instructions used to load and store parts of words in memory. They allow convenient handling of fields from zero to twenty-four bits in length arbitrarily positioned in either a single word in memory or two adjacent words. These instructions all make use of a common "field descriptor" or pointer word to control the field to be loaded or stored.

The field instructions expect the word at the effective address to be a word of the following format:



This word is called a "field descriptor" or "FD" and defines a contiguous field in memory from zero to twenty-four bits in length.

LNG - A five bit integer which defines the length of the field. LNG must be equal to or less than 24 in the standard case. Specification of a length greater than this will cause a trap G4 whenever the FD is referenced. An LNG value of 31 (37 octal) is a special case to be described later.

OFF - A five bit integer which defines the offset of the field from the left side of the word addressed. The value of OFF must lie between 0 and 23 inclusive or trap G4 will occur. Bit 0 is the left (high order) bit of the word and bit 23 is the right (low order) bit of the word. An OFF value of 31 (37 octal) is a special case to be described later.

ADR - A fourteen bit integer which is the memory address of the word containing the left most bit of the defined field. The field is assumed to lie in the same chunk as the FD, except in the LDFB and STFB instructions.

Although these instructions will most frequently be used to handle eight bit (or six bit) characters packed three (or four) to a word, they are explicitly intended to handle arbitrary fields which may overlap word boundaries in any way.

In order to avoid repetition in the following instruction descriptions, the setup common to all six field handling instructions will be described here.

1. $(Q)_{10-23} \rightarrow$ ADR, $(Q)_{5-9} \rightarrow$ OFF, $(Q)_{0-4} \rightarrow$ LNG
2. If OFF = 37_8, $(X)_{5-9} \rightarrow$ OFF
3. If LNG = 37_8, $(X)_{0-4} \rightarrow$ LNG

   } suppress this for LDFI and STFI
4. If OFF > 23, generate trap G4
5. If LNG > 24, generate trap G4
6. OFF $\rightarrow$ I, OFF + LNG -1 $\rightarrow$ J, 23 - LNG+1 $\rightarrow$ K

That is, the contents of the effective address are separated into their component pieces. If either the offset or length is 31 (37_8), the value is taken from the corresponding part of the index register. The offset and length are then checked to be sure they are within limits.

Finally, the left and right bit numbers of the field in memory and the left bit number of the field in the A register are computed. In the following descriptions $(ADR)_{I-J}$ means $(ADR, ADRH)_{I-J}$ if J > 23.

LDF Load Field

$0 \rightarrow A$, $(ADR)_{I-J} \rightarrow A_{K-23}$

The field described is right aligned in the A register. The remainder of the A register is cleared.

STF Store Field

$A_{K-23} \rightarrow (ADR)_{I-J}$

The field described by the FD at the effective address is replaced by the right most LNG bits in the A register. A is not affected by this instruction.

LDFB — Load Field Based

$ADR + X_{10-23} \rightarrow ADR$

if $X_4 = 1$ use X_{5-9} as chunk number for data

$0 \rightarrow A,\ (ADR)_{I-J} \rightarrow A_{K-23}$

This instruction is the same as LDF except that the contents of the address field of the index register are added to the FD address before the field is loaded, and the chunk number for the field may be taken from X.

STFB — Store Field Based

$ADR + X_{10-23} \rightarrow ADR$

if $X_4 = 1$ use X_{5-9} as chunk number for data

$A_{K-23} \rightarrow (ADR)_{I-J}$

This instruction is the same as STF except that the contents of the address field of the index register are added to the FD address before the field is stored, and the chunk number for the field may be taken from X.

LDFI — Load Field and Increment

If $[(Q+1)_{10-23} - ADR] * 24 + (Q+1)_{5-9} - OFF < LNG$ no action,

Otherwise $(Q)_{5-9} + LNG_2 \rightarrow (Q)_{5-9}$ (if ≤ 23)

else $(Q)_{5-9} + LNG_2 - 24 \rightarrow (Q)_{5-9}$, $(Q)_{10-23} + 1 \rightarrow (Q)_{10-23}$

$LC + 2 \rightarrow LC$

$0 \rightarrow A,\ (ADR)_{I-J} \rightarrow A_{K-23}$

The contents of the effective address and contents of the next location are both considered to be FD's. The second of these words is a limit. If there is insufficient space for the field before the limit, no action occurs. Otherwise, the FD at the

effective address is adjusted by the length of the field defined, the field is loaded into A and a skip occurs to signify that the field was loaded. This special test for LNG or OFF fields equal to 37_8 is suppressed.

STFI — Store Field and Increment

If $[(Q+1)_{10-23} - ADR] * 24 + (Q+1)_{5-9} - OFF < LNG$, no action

Otherwise, $(Q)_{5-9} + LNG_2 \rightarrow (Q)_{5-9}$ (if ≤ 23)

else $(Q)_{5-9} + LNG_2 - 24 \rightarrow (Q)_{5-9}$, $(Q)_{10-23} + 1 \rightarrow (Q)_{10-23}$

$LC + 2 \rightarrow LC$

$A_{K-23} \rightarrow (ADR)_{I-J}$

The contents of the effective address and the contents of the next location are both considered to be FD's. The second of these words is a limit. If there is insufficient room for the field, no action occurs. Otherwise, the FD at the effective address is adjusted by the length of the field defined, the right most LNG bits in A are stored in the designated field, and a skip occurs to signify that the field was stored.

IFD — Increment Field Descriptor

If $[(Q+1)_{10-23} - ADR] * 24 + (Q+1)_{5-9} - OFF < LNG$, no action

Otherwise, $(Q)_{5-9} + LNG_2 \rightarrow (Q)_{5-9}$ (if ≤ 23)

else $(Q)_{5-9} + LNG_2 - 24 \rightarrow (Q)_{5-9}$, $(Q)_{10-23} + 1 \rightarrow (Q)_{10-23}$

$LC + 2 \rightarrow (ADR)_{I-J}$

This instruction works exactly like LDFI except that it does not do the load.

Arithmetic

1. ADD and SUB do not affect X_0. Instead there is a carry bit CO which is set or reset by the carry from bit 24 of the adder on ADD or SUB. Both instructions take CI as a carry into bit 0 and reset it. CI is set only by the CCB instruction in the perform group and by BTO. When CCB is executed, CO is copied into CI. Multiple-precision arithmetic is done by adding (or subtracting) the least significant words and then doing CCB before adding the most significant words. This causes the carry from the sum of the least significant words to be added into the sum of the most significant ones. Unless CCB is executed, CI will normally remain 0 and will not disturb the operation of the machine.

2. MUL leaves the sign and most significant part in B, the least significant part in A. I.e.

LDA = 2
MUL = 3

leaves 6 in A.

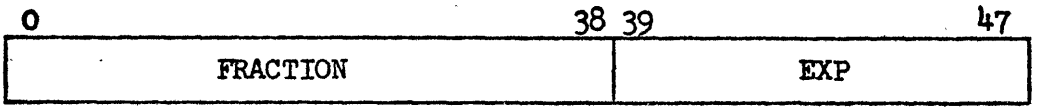
3. DIV takes the 48-bit AB register as an integer dividend, (Q) as an integer divisor. The integer quotient appears in A, the remainder in B. On overflow ($|(Q)| \leq |A|$) OV is set and AB are unchanged.

New instructions:

MDE	Memory decrement	
	(Q) - 1 → Q	Overflow and carry unaffected
ADX	X + (Q) → X	Overflow and carry unaffected

Floating Point Arithmetic

All these instructions are new. They take two-word floating point numbers as arguments. The format of a floating point number is



FRACTION is a two's complement fraction with the binary point between bits 0 and 1.
EXP is a two's complement exponent in the range (-400_8, 377_8).

All floating point instructions except UFAD and UFSB expect normalized operands and produce normalized results. FAD and FSB will correctly post-normalize a result produced from unnormalized operands. FMP and FDV will not work properly on unnormalized operands.

If an exponent overflow occurs in any floating point operation, trap G1 is generated. The result left in AB is correct except that the sign bit of the exponent is wrong; the correct sign bit is the complement of B_{15}.

If an exponent underflow occurs in any floating point operation, trap G2 is generated. The result is correct with the exception stated.

All floating point operations in this section set the carry bit C0. None sets the overflow bit.

All arithmetic is unrounded. The most significant fraction bit not included in the result is saved in CARRY. An instruction is provided to do rounding.

In addition to the instructions listed here, FLT, FIX, FRND, and FNA in the perform group operate on floating point numbers.

FAD Floating add

$$AB_F + (Q,Q+1)_F \rightarrow AB_F$$

The result is always normalized. Exponent overflow or underflow may occur. Arithmetic is done in a 48-bit adder, and 48 bits are kept until the result is packed, at which point the first bit discarded is saved in CO for use in rounding.

FSB Floating subtract

$$AB_F - (Q,Q+1)_F \rightarrow AB_F$$

See FAD

UFAD Unnormalized floating add

$AB_F + (Q,Q+1)_F \rightarrow AB_F$ (unnormalized)

Exactly as FAD except that normalization is omitted. Exponent underflow cannot occur.

UFSB Unnormalized floating subtract

$AB_F - (Q,Q+1)_F \rightarrow AB_F$ (unnormalized)

See FSB and UFAD

FMP Floating multiply

$$AB_F * (Q,Q+1)_F \rightarrow AB_F$$

See FAD. Only one bit of post-normalization will be done.

FDV Floating divide

$$AB_F / (Q,Q+1)_F \rightarrow AB_F$$

If the divisor is 0 or unnormalized, trap G3 will be taken with A and B unchanged. See FAD.

Skips

1. SKD is abolished.
2. SKE, SKG, SKA, and SKN have their inverse operations.

New instructions:

SKNE Skip on A unequal to memory

If $A \neq (Q)$, $L+2 \rightarrow L$

SKNG Skip on A not greater than memory

If $A \leq (Q)$, $L+2 \rightarrow L$

SKNA Skip on A and memory not zero

If $A \wedge (Q) \neq 0$, $L+2 \rightarrow L$

SKP Skip if memory positive

If $(Q) \geq 0$, $L+2 \rightarrow L$

Branches

1. BRM is abolished.
2. BRI is abolished.
3. BRX tests the sign bit of X, not bit 9.
4. BRR is completely redefined.

New instructions:

BDX Branch and decrement index.

X-1 → X; if X ≥ 0, Q → L

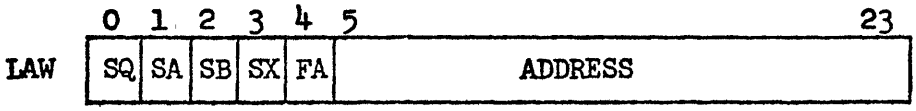
BSR Branch and save registers

Treat $(Q)_{\wedge}$ 3777777B as an indirect word and let Z be the resulting effective address.

$(L+1) \vee ((Q)_{\wedge} 74B6) \vee 2B6 \rightarrow (Z)$ Save p-counter as full address and register save bits

If $(Q)_0 = 1$, → Z+1, Z Q → (Z)
If $(Q)_1 = 1$, → Z+1, Z A → (Z)
If $(Q)_2 = 1$, → Z+1, Z B → (Z)
If $(Q)_3 = 1$, → Z+1, Z X → (Z)
Q+1 → L

This is the subroutine call instruction. It addresses the link address word of the subroutine, which has the form



FA = full address bit. This and the address determine where to store the return link, which consists of the location counter +1 and LAW_{0-4}.

SQ = save effective address. If this bit is set, the effective address of the BSR is saved following the return link. It exists primarily for compatibility with traps and pops.

SA,SB,SX cause A,B and X to be saved in successive locations following the return link if set.

All the 5 bits are saved in the return link so that BRR will know what to restore.

```
Thus    2000 BSR     3000    assume this is A in chunk 5
                :
        3000 DATA    30001000B
```

will cause locations 1000-1002 in chunk 5 to be set up as follows:

```
1000    32242001
1001    contents of A
1002    contents of B
```

Control will go to 3001

BRR Branch and restore registers

Let Z = (Q)

If $Z_0 = 1$, $Q+1 \rightarrow Q$

If $Z_1 = 1$, $Q+1 \rightarrow Q$, $(Q) \rightarrow A$

If $Z_2 = 1$, $Q+1 \rightarrow Q$, $(Q) \rightarrow B$

If $Z_3 = 1$, $Q+1 \rightarrow Q$, $(Q) \rightarrow X$

Treat $Z \wedge 37777777B$ as an indirect word and transfer to the resulting effective address.

This instruction is designed to be used for exiting from a routine called by BSR. It restores the registers saved by that instruction and transfers control to the following location.

To continue the earlier example

```
        BRR    1000
```

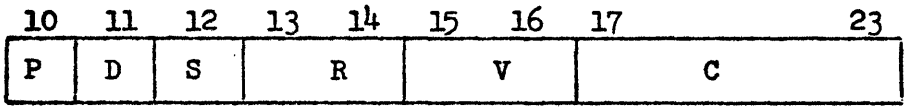
loads A and B from 1001 and 1002 respectively and transfers to 2001.

Shifts

The shift instructions on the 930 are abolished in favor of one new instruction which provides almost all the power of the old ones and a number of new features. Its differences are:

a) Different arrangement of bits in the address field.
b) Effective address is computed the same as for all other instructions.
c) Shift is by -64 to +63 bits, not max of 48.
d) NOD is abolished. But see LLT in the perform group.

The bits of the effective address field are interpreted in the following way:



Where

D Specifies the shift direction
0 - Left
1 - Right

S Specifies the type logical or arithmetic
0 - Logical. The overflow indicator is unaffected by this instruction.
1 - Arithmetic. On right shifts the sign bit is not shifted but is copied into vacated bit positions. Bits shifted out of the right bit of each active register are lost. Overflow is set if the sign bit of the A register changes during a left shift.

R Specifies the active registers
0 - A and B are taken as a single 48 bit register
1 - A only is specified
2 - B only is specified
3 - A and B are both shifted but are treated as two independent 24 bit registers.

V Specifies the action to be taken on vacated bit positions

0 - Shift in 0's

1 - Shift in 1's

2 - Shift in bits shifted out from other end of register (cycle). Extend the sign bit on arithmetic right shift.

3 - Shift in complement of bits shifted out from other end of register. Shift in the complement of the sign bit on arithmetic right shift.

C Shift count. The shift count is a seven bit two's complement number, $-64 \leq C \leq 63$. If C is negative, the direction of the shift indicated by the D field is reversed.

P (post-index). If this bit is set, X_{17-23} is added to C. The resulting signed 7-bit shift count is used to determine the direction and extent of the shift exactly as C is when P = 0.

Miscellaneous

1. EAX puts the 24 bit effective address + 2B6 (the FA bit) into X. X_{0-3} will therefore always be 0.

New instruction:

XCI Execute indirect

Take ((Q)) as the instruction I to be executed. Before executing it (Q)+1 → (Q).

If I causes a skip, L+2 → L. If it tries to cause a branch, L+3 → L and the branch is suppressed. If it tries to cause a trap, L+4 → L and the trap is suppressed. Otherwise, L+1 → L. Any kind of pop is regarded as a branch.

Non-addressable Instructions

1. RCH is abolished. Some OPR instruction can do anything an RCH can do provided it does not

a) use the E bit, or

b) use the N bit and specify any other operation, or

c) specify an or of two registers and some other operation.

2. The overflow test instructions are abolished, but see BTO.

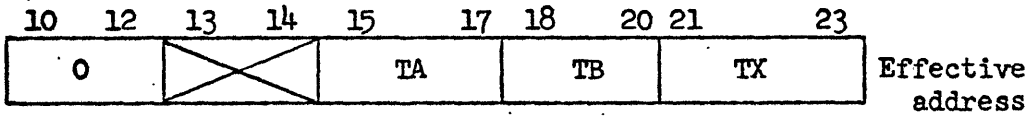
New instructions:

OPR Operate

The effective address is computed according to the ordinary rules. Then Q_{10-12} are used to select a sub-instruction as follows:

Q_{10-12}		sub-instruction
0	SWP	swap registers
1	LRO	logical register operate
2	ARO	arithmetic register operate
3	RIN	register increment
4	BTO	bit test and operate
5	unused	
6	unused	
7	PFM	perform (miscellaneous operations)

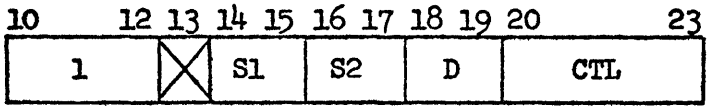
SWP Swap registers



TA =	0	0 → A
	1	A → A
	2	B → A
	3	X → A
	4	-1 → A
	5	A → A
	6	B → A
	7	X → A

TB, TX specify the final contents of B and X in the same way. To leave a register unchanged, it is necessary to specify that it should be transferred to itself.

LRO Logical register operate



Effective address

S1 = source register 1: 0 = 0
1 = A
2 = B
3 = X

S2 = source register 2, same code

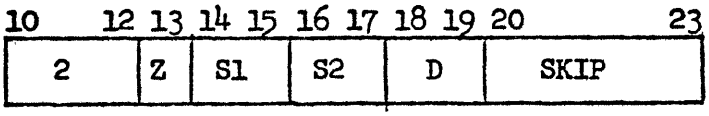
D = destination register, same code. 0 means discard result. In this case the instruction is a NOP.

CTL = Control. These four bits specify how bits from the source registers determine corresponding bits in the destination:

S1 bit	S2 bit	D bit
1	1	CTL_0
1	0	CTL_1
0	1	CTL_2
0	0	CTL_3

Thus, S1$\wedge$ S2 is specified by CTL = 1000, S1$\vee$ S2 by CTL = 1110, etc.

ARO Arithmetic register operate



Effective address

If Z = 0 S1 + S2 → D

Z = 1 S1 - S2 → D

Skip causes the instruction to skip if any of the four conditions which may be tested are satisfied.

$SKIP_0$	skip if D < -1
$SKIP_1$	skip if D = -1
$SKIP_2$	skip if D = 0
$SKIP_3$	skip if D > 0

Sample skip conditions

No skip	0000
Skip on 0 result	0010
Skip on positive result	0011
Skip on result $\leq$ 0	1110
Skip on mixed ones and zeroes	1001

This instruction can also be used, for example, to double X. Overflows and carries are ignored.

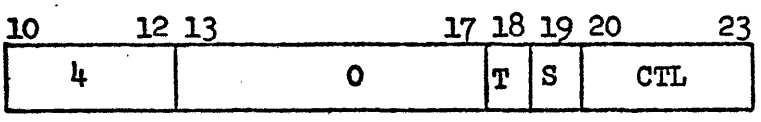
RIN Register increment

This instruction is identical to ARO except that S2 is interpreted as a constant which may be 0, 1, 2 or 3.

Thus, to decrement X by 2 and skip if the result is negative, use an OPR address

01111110111100

BTO Bit test and operate



If T = 0:

S = 0 selects overflow (OV)

1 selects carry (CO)

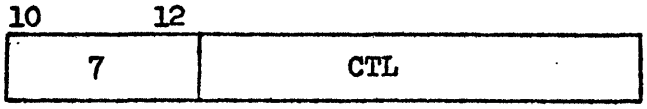
The bits of CTL select four operations:

CTL_0	skip if bit is 0
CTL_1	skip if bit is 1
CTL_2	set bit to 1 if it is 0
CTL_3	set bit to 0 if it is 1

If T = 1:

S = 0	means copy OV, CI, CO into A_{21-23}
1	means copy A_{21-23} into OV, CI, CO

PFM Perform



The control field selects one of 20 operations to be performed. Its precise format is not yet specified. The operations are

FLT Convert to floating point

$$AB_I \rightarrow AB_F$$

The contents of the A and B registers are assumed to be a 48 bit integer which is converted to a normalized floating point number. The resulting floating point number replaces the contents of A and B. Overflow cannot occur. The most significant of any discarded bits is saved in the carry flip-flop.

Example:

	A	B
Before Execution	00000345	76325410
After Execution	34576325	41000040

FIX Convert to fixed point

$$AB_F \rightarrow AB_I$$

The normalized floating point number in A and B is converted to a 48 bit integer in A and B. If the exponent of the floating point number is less than or equal to 0, A and B are cleared. If the exponent is greater than 47, the overflow indicator is turned on and exit is made with A and B unchanged. The most significant fraction bit is stored in the carry bit, CO.

Example:

	A	B	CO
Before Execution	24500000	00000005	X
After Execution	00000000	00000024	1

FRND Floating round

Add 1000B * (CO B_{14}) to AB (integer arithmetic). If overflow, shift fraction right and add 1 to exponent. If exponent overflow, generate trap G1. This operation is intended for use after a floating point instruction which leaves the first non-significant bit of the result in CO. It rounds the magnitude to the nearest even number. Thus, if .5 is the first non-significant bit, we have the following results of rounding according to this algorithm:

ORIGINAL NUMBER		ROUNDED RESULT	
Decimal	Binary	Binary	Decimal
2.5	010.1	010.0	2.0
3.5	011.1	100.0	4.0
-2.0	110.0	110.0	-2.0
-2.5	101.1	110.0	-2.0
-3.0	101.0	101.0	-3.0
-3.5	100.1	100.0	-4.0

FNEG Floating negate

$-AB_F \rightarrow AB_F$ Normalize result one bit if necessary

Exponent overflow may occur if AB contain

40000000 00000377

In this case trap G1 occurs.

Exponent underflow may occur if AB contain

20000000 00000400

In this case trap G2 occurs.

CFDC Convert field descriptor to bit count

$A_{10-23} * 24 + A_{5-9} \rightarrow A$

This instruction converts a field descriptor (Words + Offset) into a bit count in A. It can be used to convert a string length pointer into the length of the string in bits. The length of the string in characters may then be obtained by shifting A right 3 bits (divide by 8).

CCFD Convert bit count to field descriptor

$A/24 \rightarrow A_{10-23}$, remainder $\rightarrow A_{5-9}$

This instruction converts a bit count in A into a FD with zero length field in A. (i.e. $0 \leq A_{5-9} \leq 23$)

NRM Normalize field descriptor

1. If $A_0 = 1$, $A + 01337777_8 \rightarrow A$

   If $A_{15-9} \geq 30_8$, $A - 01337777_8 \rightarrow A$

2. If $0 \leq A_{0-9} \leq 23$, $0 \rightarrow OV$

   Otherwise $1 \rightarrow OV$

This instruction is used to restore a field descriptor in A to normalized form after two descriptors with zero LNG fields are added or subtracted. One normalization step is taken. If the result is normalized the overflow indicator is turned off. If the result is not normalized, the overflow indicator is turned on.

LLO Locate leading one

The bit position of the first (left most) one bit in the A and B registers is placed in the X register. The sign bit of the A register is bit 0 and the least significant bit of the B register is bit 47. If there are no one bits in A or B, X is set to -1.

LLZ Locate leading zero

The bit position of the first (left most) zero bit in the A and B registers is placed in the X register. If no bits are zero, X is set to -1.

LLT Locate leading transition

The bit position of the first zero bit followed by a one, or the first one bit followed by a zero, is placed in the X register. If all bits of both A and B are 0, X is set to -1. Since bit 23 of B is always assumed to be followed by a 0, all 1's set to 47.

CNT Count bits

The number of bits in A and B which are 1 is placed in the X register. The result in X will therefore lie between 0 and 48 inclusive.

CCB Copy carry bit

Copies CO into CI and resets OV. For the use of this instruction see ADD and SUB.

RCC Read calendar clock

The processor is equipped with a 48-bit calendar clock which is incremented once every 100 μs whenever power is on. This instruction reads the current value of the clock into AB. The system will ensure that the clock reading when added to a number available from the system, reflects the amount of time which has elapsed since January 1, 1969 to the nearest second, and that the difference of two readings of the clock will measure the real time which elapsed between the two readings with an accuracy of at least 200 μs, provided no system crash has intervened.

RIT Read interval timer

The processor is also equipped with a 24-bit interval timer which is incremented once every 10 μs. This timer is part of the state of a process and increments only when the process is running. When the timer becomes 0 trap S1 is generated. This instruction reads the current value of the timer into A. The system may reset the timer at any time, so that it cannot be used for measuring real time. There is a privileged instruction to set the timer.

PRO_i Protect

i = 4, 8, 16, 32

This instruction causes the processor to obtain control of the PRO bus. Once it does so it raises the PRO signal and holds it up until:

a) The program has made i successful memory fetches. All fetches, whether for instructions, indirect words or data are counted.

b) or any floating point instruction or PFM or MUL or DIV is executed.

c) or ACT is executed. See the description of ACT for a specification of the action taken.

d) or a trap occurs.

Privileged Instructions

1. EOM, SKS, BRI, POT and PIN are abolished.
2. There is one privileged instruction, called ACT. Its effective address is computed in the usual way. The processor then attempts to gain control of the ACT bus. When it does, it puts the 14 address bits on the bus and raises the ACT line. It then hangs, looking at the return lines on the ACT bus, which are:

   ACON — ACT considered. Raised when the target device accepts the ACT.

   AACK — ACT acknowledged. Raised when the target device is willing to let the CPU proceed.

   ASKP — Raised if the target device wants the CPU to skip.

   The processer holds up ACT and hangs until it sees ACON or until 10 μs have elapsed. In the latter case it lowers ACT and waits another .5 μs. If ACON is still not raised, the ACT instruction terminates with no skip. This exit should be interpreted by the program as an indication that the device has not accepted the signal.

   Once ACON has been raised, the processor hangs until it sees AACK. It then exits with L+2 → L or L+3 → L depending on whether ASKP is low or high. This allows the device to signal success or failure, or some other piece of information, to the processor. If the processor has PRO raised when ACT is executed, the PRO stays up until the third successful memory fetch after the ACT is completed, regardless of how soon it would normally have been terminated.

Pops

If the P field of an instruction word is non-zero, the instruction is interpreted as a programmed operator. There are three kinds:

P = 1	user pop
P = 2	system pop
P = 3	process pop

They differ only in the location of the transfer vector and in the treatment of protection. For user pops the transfer vector occupies 100_8-177_8 in the chunk in which the pop is located. For system pops it occupies 100_8-177_8 in chunk 31. For process pops it resides in the context block in locations not yet decided.

When a pop is executed, the machine performs a BSR* through T+OP, where T is the origin of the transfer vector and OP is the opcode field of the instruction. If the LAW of the BSR specifies saving Q, the effective address of the pop is saved. If P > 1, the ICP mechanism is turned off. I.e., a user or process pop may allow control to be transferred from any chunk to any other.

Traps

The machine has a large variety of traps, or forced transfers of control. Each trap has a core location in chunk 31 assigned to it. When the condition for a trap arises, a BSR* through this core location is performed. If the LAW of the BSR specifies saving of Q, the quantity actually saved depends on the trap. Note that traps have nothing to do with interrupts or wakeup signals, which are not handled by the CPU at all.

The traps are classified as follows:

General traps

	Condition	Quantity saved as Q
G1	Floating point overflow. The result in AB is correct except that a 0 sign bit must be supplied to the left of the exponent.	0
G2	Floating point underflow. The result in AB is correct except that a 1 sign bit must be supplied to the left of the exponent.	0
G3	Floating point divide check: 0 or unnormalized divisor. AB are unchanged.	0
G4	Field descriptor check: OFF or LNG fields > 23. AB are unchanged.	0
G5	Indirect address trap.	Address of the indirect word in which the trap bit was set.

Memory traps

M1	Missing chunk (chunk table entry = 0)	Virtual address causing trouble.
M2	Page number > LPT	Virtual address causing trouble.
M3	Missing page (page table entry = 0)	Virtual address causing trouble.
M4	Page not in DHT	Virtual address causing trouble.
M5	Page in DHT, but PST_0 = 0, or N = 1 on write.	Virtual address causing trouble.

System traps

S1	Interval timer passes through 0	0
S2	Undefined opcode	0
S3	Parity error	Real address causing error
S4	Non-existent real memory	Real address causing error

Protection traps

P1	ICP violation	0
P2	Write into page with PPB_w not set	0
P3	Execute from page with PPB_x not set	0
P4	Privileged instruction from page with PPB_p not set	0

Processes and the Context Block

A process is defined by a page of virtual memory called its context block, which holds the entire state of the process while it is not running. When the process starts to run, its state is copied from the context block into the registers of the CPU. When the process stops running, the current contents of the CPU registers are copied back into the context block. This can in general happen between the execution of any two CPU instructions.

The device which controls which process is to run is called the process control unit (PCU) or the scheduler. Associated with each processor is a fixed core location NP and a line from the PCU called SWITCH. The processor also has a register which contains the real address of the context block for the process which is currently running. When SWITCH is raised, the processor stores its state into the context block of the process it is executing, picks up the contents of NP and treats it as the real address of a new context block, loads its state from this new block and continues to execute. This operation is called process switching. It takes place only on command from the PCU. It is inhibited by PRO.

The context block has the following format:

Word(s)	Bits	Contents
0-31	0	chunk table
32	0	OV
	1	CI
	2	CO
	5-23	L
33		A
34		B
35		X
36		interval timer

Note that the contents of the context block defines the map, among other things. The remainder of the block holds page tables and storage for the routines of the core monitor. The context block will always appear in a fixed place in the map of every process.

The PCU is responsible for scheduling the execution of processes on whatever processors happen to be available. It does this by maintaining tables in main memory which indicate what processes are candidates to run and with what requirements (priority, memory, deadlines, etc.). The formats of these tables and the algorithms to be used in scheduling have not been fully defined.

The PCU also accepts interrupt signals from the outside world. Associated with each interrupt line is a fixed area in core which contains information about the process to be activated when the line is raised. The PCU adds the process to its tables in accordance with this information whenever it sees the line raised.

As far as the PCU is concerned, a process can be in one of three states:

blocked
ready
running

A blocked process is not a candidate to run. A process becomes blocked when some processor tells the PCU to block it. Of course, a process may block itself. Block, like all instructions to the PCU from a processor, is a privileged instruction.

When a process receives a wakeup signal, it becomes ready, unless it is already ready or running. This means that it is a candidate to run. A wakeup signal may be an interrupt line (see above) or an instruction from a processor. The signal carries assorted information about the process:

the drum address of its context block
possibly priorities and deadlines

which allow the PCU to make a processor run it and to decide when it should run in relation to other processes.

When the algorithms to be used by the PCU have been decided, this section of the manual will be greatly expanded.

Summary of 6700 Instructions

Loads and stores:

LDA	Load A
LDB	Load B
LDX	Load X
LDD	Load double
STA	Store A
STB	Store B
STX	Store X
STD	Store double
STM	Store masked
STZ	Store zero
XMA	Exchange memory and A
XMX	Exchange memory and X

Field instructions

LDF	Load field
STF	Store field
LDFB	Load field based
STFB	Store field based
LDFI	Load field and increment
STFI	Store field and increment
IFD	Increment field descriptor

Arithmetic

ADD	Add to A
SUB	Subtract from A
MUL	Multiply
DIV	Divide
MIN	Memory increment
MDE	Memory decrement
ADM	Add to memory
ADX	Add to X

Floating point

FAD	Floating add
FSB	Floating subtract
UFAD	Unnormalized floating add
UFSB	Unnormalized floating subtract
FMP	Floating multiply
FDV	Floating divide

Skips

SKE	Skip on A equal to memory
SKNE	Skip on A not equal to memory
SKG	Skip on A greater than memory
SKNG	Skip on A not greater than memory
SKN	Skip on negative
SKP	Skip on postive
SKA	Skip on A and memory zero
SKNA	Skip on A and memory not zero
SKB	Skip on B and memory not zero
SKR	Decrement memory and skip if negative

Branches

BRU	Branch
BRV	Branch and increment X
BDX	Branch and decrement X
BSR	Branch and save registers
BRR	Branch and restore registers

Miscellaneous

SHFT	Shift
EAX	Effective address to X
EXU	Execute
XCI	Excute indirect

Operate

SWP	Swap registers
LRD	Logical register operate
ARD	Arithmetic register operate
RIN	Register increment
BTO	Bit test and operate
PFM	Perform
FLT	Convert to floating point
FIX	Convert to fixed point
FRND	Floating round
FNEG	Floating negate
CFDC	Convert field descriptor to bit count
CCFD	Convert bit count to field descriptor
NRM	Normalize field descriptor
LLO	Locate leading one
LLZ	Locate leading zero
LLT	Locate leading transition
CNT	Count bits
CCB	Copy carry bit
RCC	Read calendar clock
RIT	Read interval timer
PRO	Protect

Privileged

ACT	