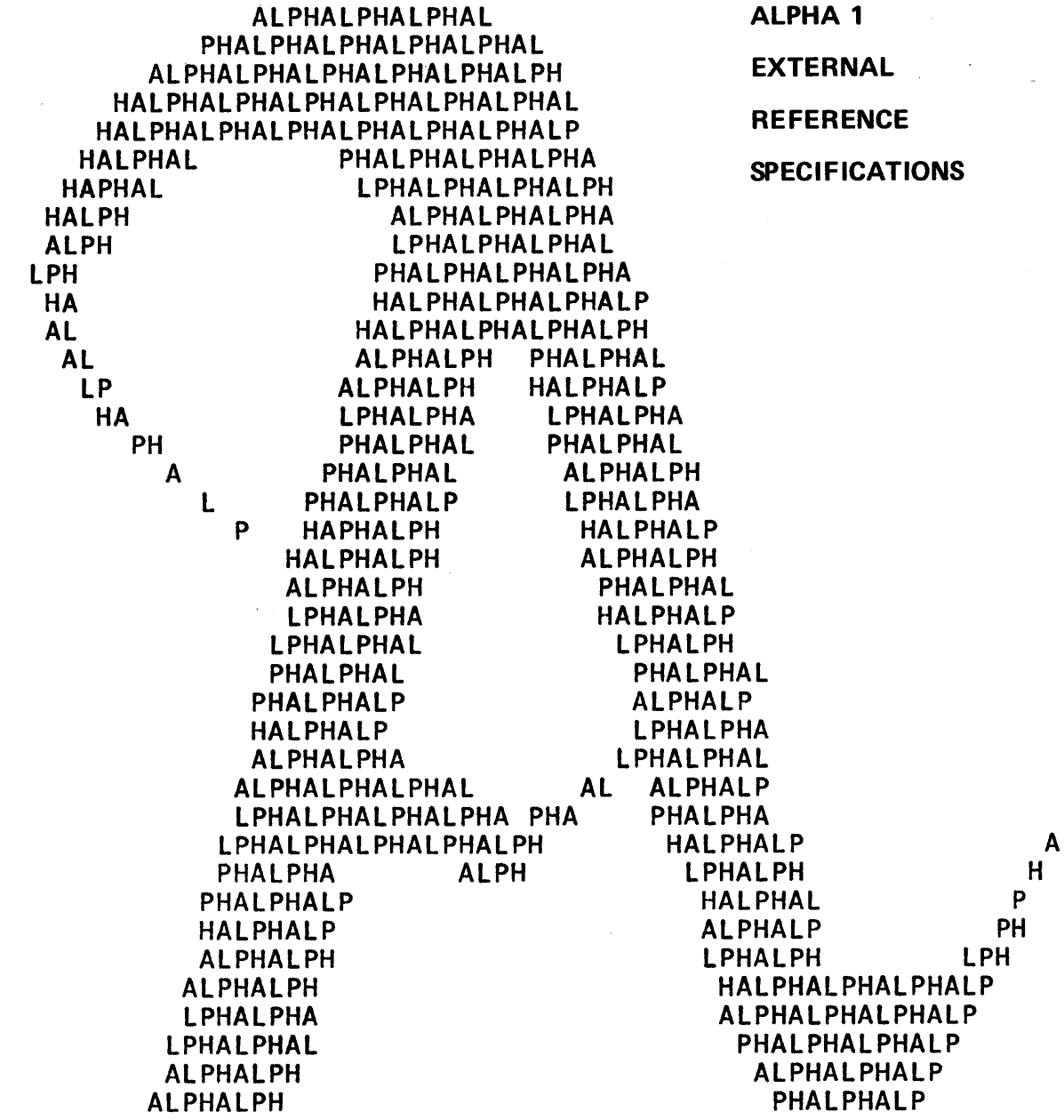


ALPHA 1

EXTERNAL

REFERENCE

SPECIFICATIONS

H E W L E T T - P A C K A R D P R I V A T E

JULY 20, 1970

DO NOT REPRODUCE----------COPY 42

T A B L E O F C O N T E N T S

SECTION VII - INSTRUCTION SET

SECTION I - SYSTEM INTRODUCTION

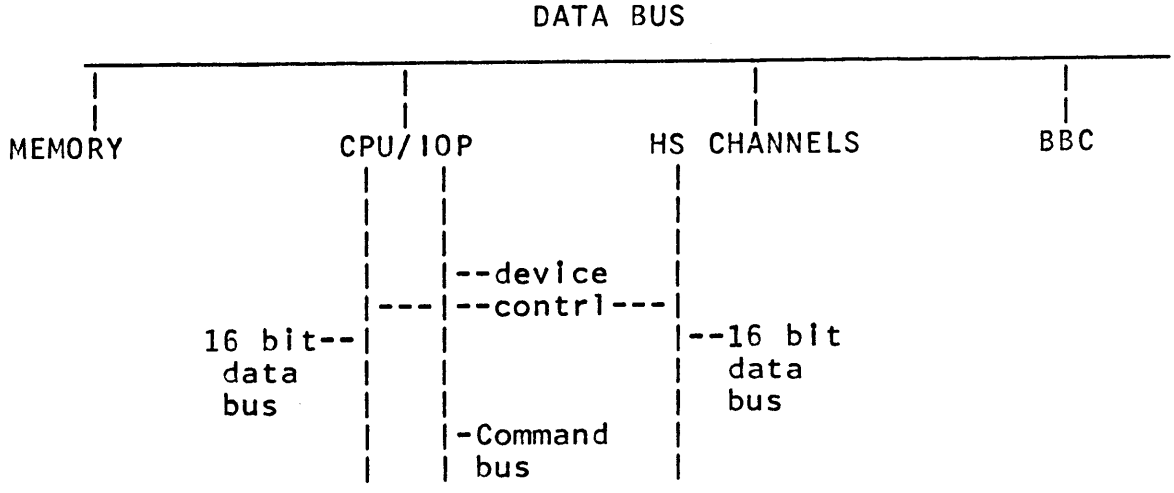
The ALPHA 1 is a computer system designed for efficient operation in a single processor/multiprogramming realtime environment. The operating domain for an executing program consists of a stack, an executing code segment and a global (or Common) data area.

The central processor contains address registers for specifying the location and limits of the currently operational address space. Direct addressing modes, with indexing, relative to these registers are provided into the stack and into the currently executing code segment.

Memory protection is provided to assure that all memory references remain within the address space assigned to the executing program. Changes to the address space may be made only when the CPU is in privileged mode under control of the operating system.

The hardware is organized on a modular basis with communication between modules occurring over an asynchronous priority demand data bus. Modules consist of a central processor, memories, high speed channels, and other equivalent higher level devices. Peripheral devices transfer data to and from memory, or the CPU through and under control of the I/O processor portion of the CPU.

The system architecture incorporates an omnibus concept. The maximum number of modules (7) and the comunication rate are set to minimize cost and maximize the I/O capabilities of the machine.

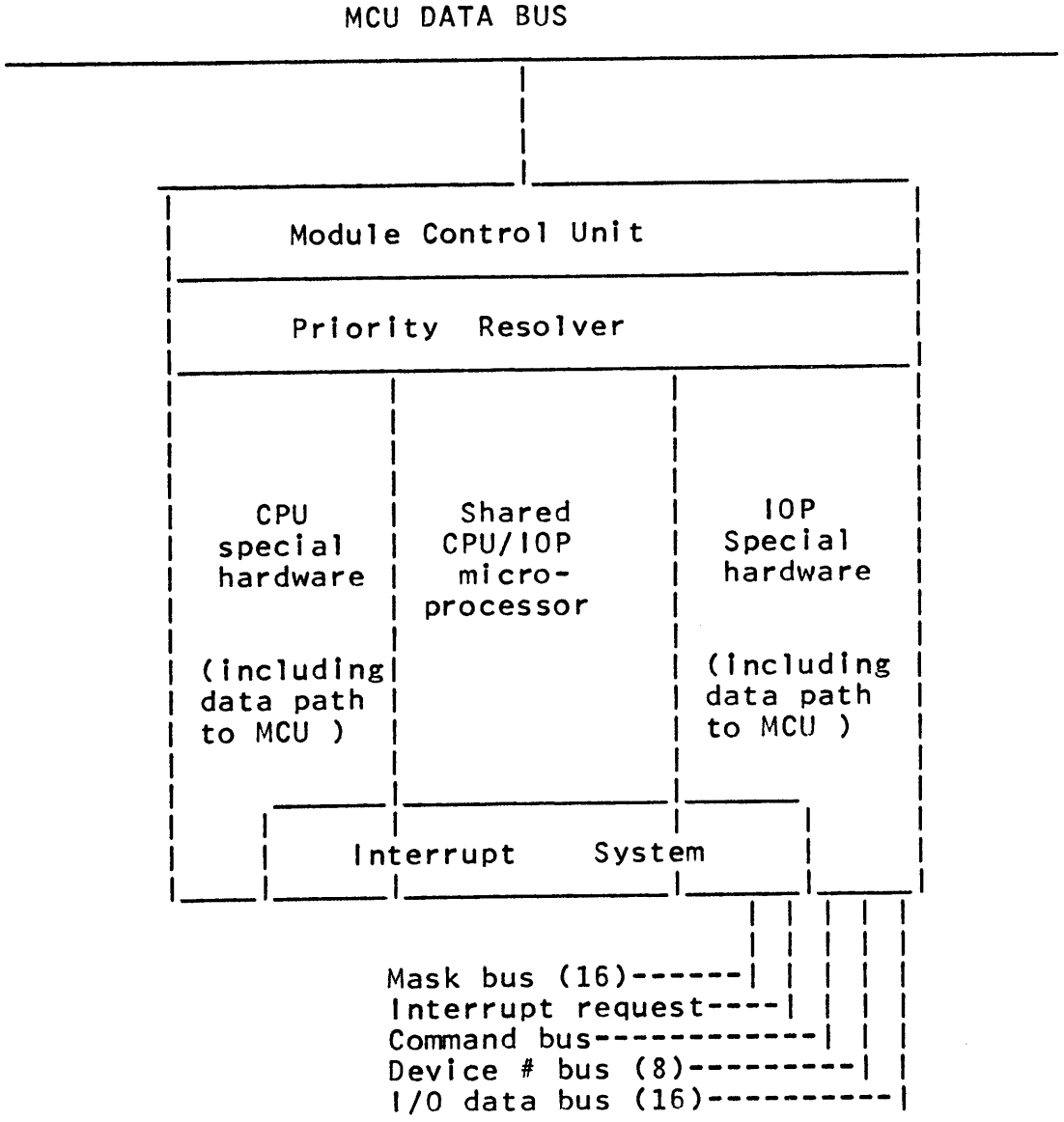


SECTION II - SYSTEM HARDWARE MODULES

A. CENTRAL PROCESSOR MODULE

The CPU is a stack oriented processor designed to give both high code density and a flexible addressing structure suitable for a real time environment and a multiprogramming environment.

The internal organization of the CPU/IOP is designed to give the maximum I/O rate and flexibility at the minimum cost. To this end a major portion of the hardware of the CPU is shared with the IOP. The following sketch gives the general configuration:



B. MEMORY MODULE

A memory module may contain from 4k to 65k words of any speed storage medium. The maximum addressable storage is 65k words.

The memory will run asyncronously to the bus execpt for communication on the bus. This will facilitate the addition of new memory technologies as they become available. All memory in a given module must be of the same type. A memory module will contain its own address and data registers.

C. HIGH-SPEED CHANNELS MODULE

This module is an optional port onto the internal data bus for high-speed input-output devices.

D. BUFFERED BUS COMMUNICATOR MODULE

This module presents a standard 16 bit interface from the bus to the outside world. It is a glorified microcircuit register.

E. MODULE CONTROL UNIT (MCU) & DATA BUS

The MCU processes requests from the various modules to use the Data Bus. It controls the assignment of the bus time slices to the requesting modules on a two level priority basis. All communication on the Data Bus is synchronous with the MCU clock, although each module may be asynchronous to the other modules.

SECTION III - DESCRIPTIONS AND FORMATS

A. DEFINITIONS

TOS ---------- Top of stack

CIR ---------- Current instruction register

Condition codes ---------- CCA sets CC = CCL if operand < 0
CC = CCE if operand = 0
CC = CCG if operand > 0

CCB sets CC = CCL if special ASCII character.
CC = CCE if alphabetic ASCII character, upper & lower case.
CC = CCG if numeric ASCII character.

CCC sets CC = CCL if opnd1 < opnd2
CC = CCE if opnd1 = opnd2
CC = CCG if opnd1 > opnd2

B. CENTRAL PROCESSOR REGISTERS

DB: Data base register - 16 bits
contains the absolute address of the first location (bottom) of the stack.

Q: Stack marker pointer - 16 bits
contains the absolute address of the current stack marker being used within the stack.

SM: Memory stack pointer register - 16 bits
contains the absolute address of the last used core location of the stack.

SR: Register stack pointer register - 3 bits
contains the number of registers in the CPU which contain elements on the TOS.

S: Logical quantity which always points to the top element of the stack.

S = SM + SR

Z: Stack limit pointer - 16 bits
contains the absolute address of the last word of memory available to the stack.

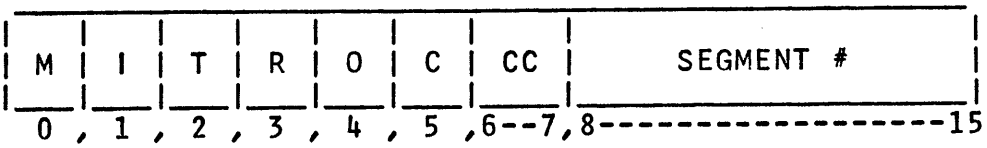
DL: Data limit register - 16 bits
contains the absolute address of the last word of memory availble to the user's data space.

PB: Program base register - 16 bits
contains the absolute address of the first location of the code segment being executed.

P: Program counter - 16 bits
contains the absolute address of the instruction being executed.

PL: Program limit register - 16 bits
contains the absolute address of the last location of the current code segment.

STA: Status register - 16 bits
includes:



where...

M = Privileged mode bit

I = Enable/Disable External Interrupts bit

T = Enable/Disable traps bit

R = Right Stack Op bit -- set to 1 if execution will proceed with Stack Op B of the instruction word

O = Overflow bit

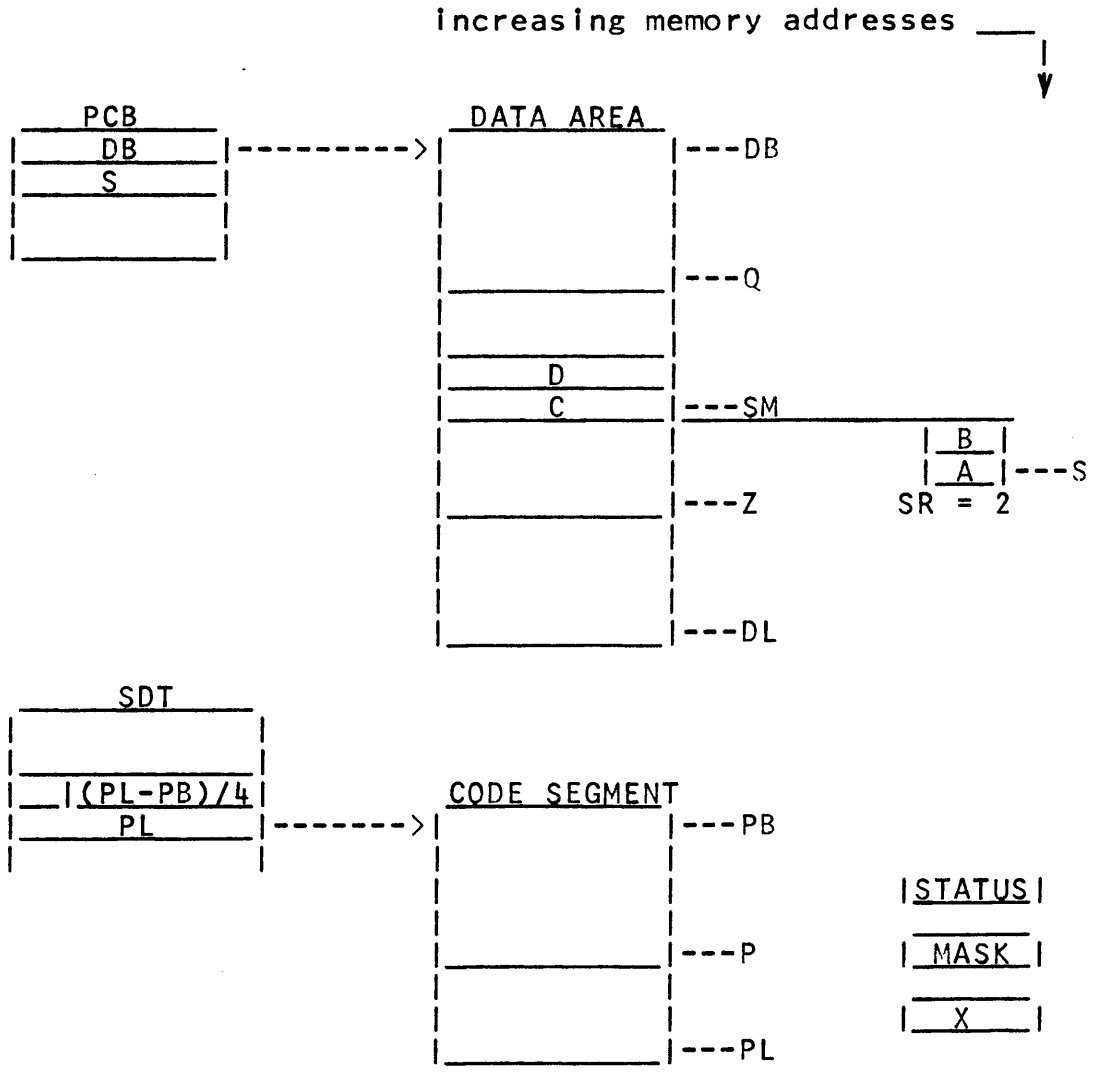
C = Carry bit

CC = Condition code
CCG = 00, CCE = 01, CCL = 10 or 11

SEGMENT # = The double word index into the Segment Descriptor Table (SDT).

C. ADDRESS SPACES (CODE SEGMENTS, STACKS)

The address space of a user will be organized as follows



D. ADDRESSING CONVENTIONS

1. S relative
   Operations use the top elements of the stack as implicit operand addresses and an address computed by subtracting the displacement from S.
   This mode will typically be used for accessing temporary variables in subroutines.

2. Q relative
   Operations use the top of the stack as implicit addresses if needed and an address computed by adding (or subtracting) the displacement field to (from) Q. This is useful for parameter passing and local variable storage in procedures.

3. DB relative
   Operations use the top of the stack as implicit addresses if needed and an address computed by adding a displacement field to DB. This is used for global variable addressing.

4. P relative
   Operations use the top of the stack as implicit addresses if needed and an address computed by adding (or subtracting) the displacement to (from) P. This is used for branches, procedure calls, literals, etc.

Address computation

The following are definitions used throughout in discussing address computation:

E = Effective address
RA = Relative address
X = If the X-bit, CIR(4), = 0 then 0 else the content of the X register
S = SM + SR, where SM and SR are the CPU registers defined previously.

The following are the different cases of address computation and how the above quantities are computed.

1. Code, direct
   E = P + D + X
   or E = P - D + X
   RA = E - PB

2. Code, indirect
   E = (P + D) + X + PB
   or E = (P - D) + X + PB
   RA = E - PB

3. Data, direct
   E = DB + D + X
   or E = Q + D + X
   or E = Q - D + X
   or E = S - D + X
   RA = E - DB

4. Data, indirect
   E = (DB + D) + X + DB
   or E = (Q + D) + X + DB
   or E = (Q - D) + X + DB
   or E = (S - D) + X + DB
   RA = E - DB

Bounds check:
If E lies outside of the following ranges, a trap will occur.

Code, direct or indirect
PB <= E <= PL

Data, direct
DB <= E <= SM (memory)
or SM+1 <= E <= S (TOS registers)

Data, indirect
DB <= E <= SM (memory)
or SM+1 <= E <= S (TOS registers)
or Z <= E <= DL (memory)

Bounds checking is applied according to the following table:

	USER MODE	PRIVILEGED MODE
Code read	Yes	No
Code execute	Yes	Yes
Data read	Yes	No
Data write	Yes	No
Stack overflow (S <= Z)	Yes	Yes

This gives the following address modes

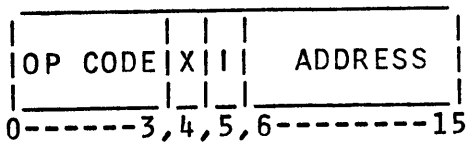
1. P + D	13. Q + D
2. P + D + X	14. Q + D + X
3. (P + D) + PB	15. (Q + D) + DB
4. (P + D) + X + PB	16. (Q + D) + X + DB
5. P - D	17. Q - D
6. P - D + X	18. Q - D + X
7. (P - D) + PB	19. (Q - D) + DB
8. (P - D) + X + PB	20. (Q - D) + X + DB
9. DB + D	21. S - D
10. DB + D + X	22. S - D + X
11. (DB + D) + DB	23. (S - D) + DB
12. (DB + D) + X + DB	24. (S - D) + X + DB

Addressing arithmetic is done modulo 65k words.
One level of indirect addressing is allowed.

E. INSTRUCTION FORMATS

Instructions may take one of eight formats. Please see attached sheets for instruction descriptions.

1. Memory reference



I Indirect bit

X Indexing to be used

If both X = 1 and I = 1 then post indexing will occur. The displacement is a positve quantity, giving sign-magnitude addressing modes.

Address decoding

P+ CIR(6:7) = 00
CIR (8:15) = Displacement
Range(P,P+255)

P- CIR(6:7) = 01
CIR (8:15) = Displacement
Range(P-255,P)

DB+ CIR (6:7) = 10
CIR (8:15) = Displacement
Range(DB,DB+255)

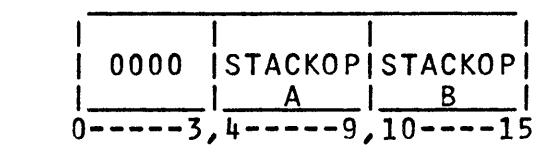
Q+ CIR (6:8) = 110
CIR (9:15) = Displacement
Range(Q,Q+127)

Q- CIR (6:9) = 1110
CIR (10:15) = Displacement
Range(Q-63,Q)

S- CIR (6:9) = 1111
CIR (10:15) = Displacement
Range(S-63,S)

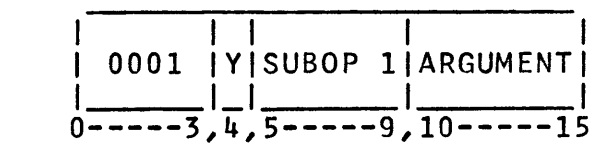
		CIR 6	7	8	9	10	11	12	13	14	15
P+	relative	0	0	....	Displace.	(0:255)					
P-	relative	0	1	....	Displace.	(0:255)					
DB+	relative	1	0	....	Displace.	(0:255)					
Q+	relative	1	1	0	..	Displace.	(0:127)				
Q-	relative	1	1	1	0	Displace.	(0: 63)				
S-	relative	1	1	1	1	Displace.	(0: 63)				

2. Stack & Control



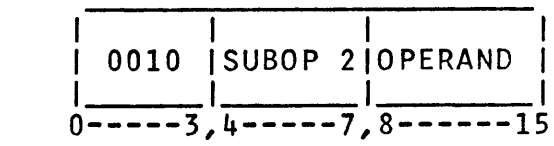
Execution is from left to right (A first, then B)

3. Shifts, bit tests, conditional branches

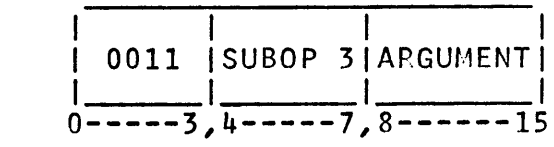


Note that in this format the Y bit, CIR(4), is a true index bit on the shift and bit test instructions, and is the indirect addressing bit for the conditional branches.

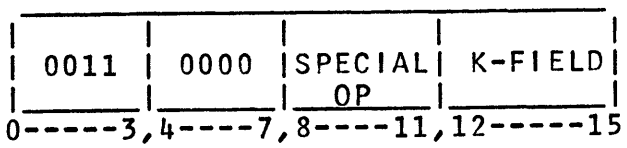
4. Immediate



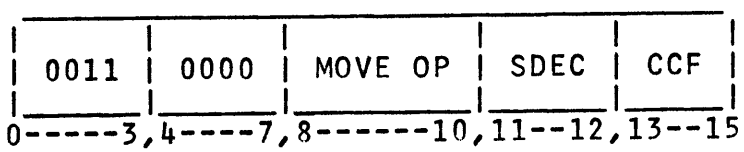
5. Linkage and control instructions



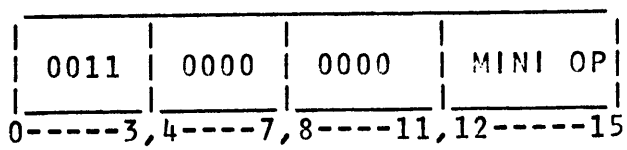
6. Special Opcodes



7. Move Opcodes

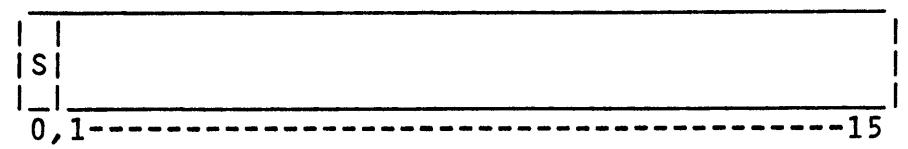


8. Mini Opcodes



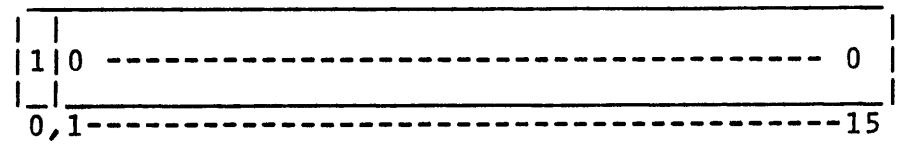
F. DATA FORMATS

Fixed point format: 16 bit, two's complement



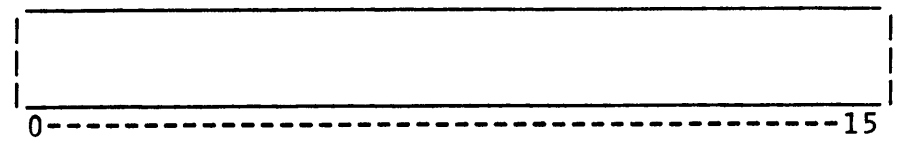
(For double-word fixed point format, 16 magnitude bits are added on the right).

A trap will be invoked if an integer arithmetic operation (not addressing or indexing) encounters the illegal integer represented by

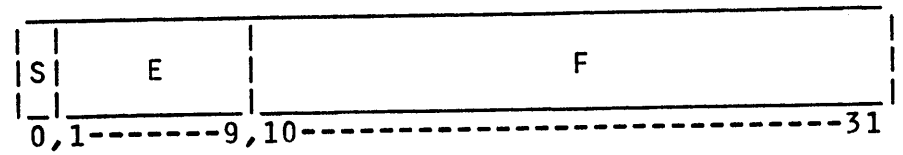


in single word integer instructions or the integer represented above followed by another word of all zeroes in double word integer format.

Logical format, 16 bit positive integer



Floating point format, sign + magnitude representation



S = Fraction sign bit (0 for positive, 1 for negative)

E = Exponent+256 (Range 0 to 511)

F = Fraction (Range 0 to (2**22) - 1)

Note: In sign + magnitude representation, the fraction is always positive with the S bit containing the sign of the number. The binary point is assumed to the left of bit 10 with an implied leading 1 to the left of the binary point. Thus all floating point numbers are stored in normalized form, but no bit is wasted on the leading 1, making all fraction bits significant. The exceptions are that 0 is a word containing all 0's, and a one in the sign position with the remainder of the word containing zeros is illegal and if encountered causes a trap.

Decimal value = (-1)**S * 2**(E-256) * (1+F*2**(-22))
Decimal value = 0 when S = E = F = 0

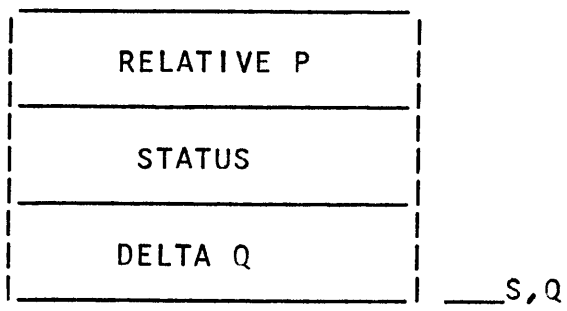
Note the following special cases:

```
|0|000000000|00000000000000000000000| =0
|0|100000000|00000000000000000000000| = 1
|1|000000000|00000000000000000000000| = ILLEGAL
```

Definitions:

```
floor (X) = largest integer <= X
sign (X) = 1, if X > 0
           0, if X = 0
          -1, if X < 0
exp (X)  = floor (logbase2 |X|), if X ¬= 0
           undefined, if X = 0
trunc (X) = sign (X) * floor (|X|)
round (X,N) = 0, if X = 0
              2 ** (exp (X) - N) * sign (X) *
                floor (2**(N-exp (X)) * |X| + 0.5), if X ¬= 0
   where N+1 is the number of significant bits
```

G. STACK MARKER FORMATS



DELTA Q = Value to be subtracted from the Q register to obtain the Q value of the caller.

STATUS = The content of the status register.

RELATIVE P = P+1 - PB
(The instruction to be returned to)

SECTION IV - TABLES

There are seven classes of tables resident in the Alpha operating system. They are classified below:

NAME	MNEMONIC	#	LENGTH	WHERE
Segment Table Pointer	SDTB	1	1	Fixed location 0
Current PCB Pointer	CPCB	1	1	Fixed location 1
Interrupt Stack Base	QI	1	1	Fixed location 2
Interrupt Stack Limit	ZI	1	1	Fixed location 3
Device Reference and Ext. Interrupt Table	DRT	1	<=255	Fixed locations 4--1777 4 word entries, 255 max.
Segment Descriptor Table	SDT	1	<=256	Starts at location SDTB 2 word entries, 256 max.
Segment Transfer Table	STT	var	var	Dynamically allocated
Process Control Block	PCB	var	var	Dynamically allocated
Privileged Data Table	PDT	var	var	Dynamically allocated

The entries in these tables are described below.

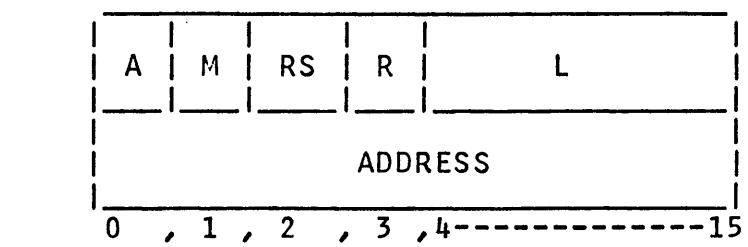
A. SEGMENT TABLE POINTER & SEGMENT DESCRIPTOR TABLE

The segment descriptor table is actually the controlling table of the system. It resides at a known location in memory (SDTB pointer) containing a known number of double word entries determined at system generation time. Segment # 0 is reserved for interrupt use, and the zeroth entry in the SDT contains the address SDTL of the last entry in the SDT. There are a maximum of 255 double word descriptors. Five distinct regions exist in the Segment Descriptor Table:

1. Monitor intrinsics
2. Library subroutines
3. Processors and shared programs
4. User programs (both segmented & non-segmented)
5. Internal Interrupts and Traps

The length of regions 1-3 are fixed at system generation time. The length of region 4 is dynamically allocated at job execution time. The length of region 5 is permanently fixed, but the descriptors may change dynamically (Note that External Interrupts are not contained in the SDT).

The entries of the SDT have the following form:



A = Absence bit, set to 1 if the code segment is absent from core.

M = Mode bit, set to 1 if segment is to be executed in privileged mode.

RS = Reserved for future use

R = Reference bit, set to 1 when this descriptor is accessed.

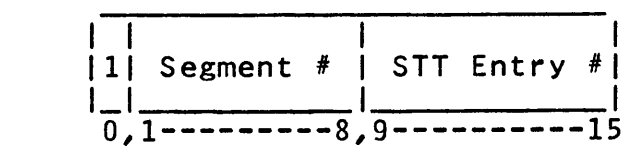
L = For an N-word segment, L = ((N-1) + 3)/4, such that PL = PB + 4*L >= PB + N - 1 and 4*L - (N-1) <= 3.

ADDRESS = Absolute address of PB if the segment is present, otherwise it indicates the absolute disc address

B. SEGMENT TRANSFER TABLE (STT)

A segment transfer table is associated with each segment and appended to each generated program segment at load time. PL-addressing is used by the CALL instructions to access the table. It contains 1 word entries which are of two basic types: external reference and local reference. The entries are:

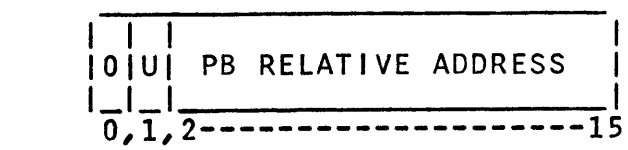
External entry



Segment # = Logical entry number in the SDT

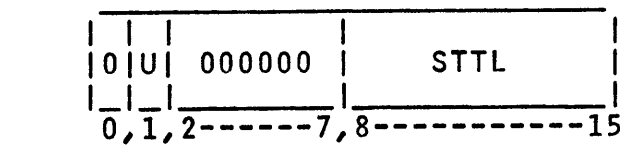
STT Entry # = Number to index from PL to obtain entry point into segment.

Local entry



U = Uncallable bit. This bit is ignored when referenced within a segment or when in privileged mode. It is set to 1 when the entry point is uncallable from a non-privileged segment.

PL entry (The last word of any program segment)



STTL = # of entries in this Segment Transfer Table (STT).

A transfer through STT entry # 0 implies that execution will begin at P = PB.

Note: Any call outside the current program segment may only reference the first 128 entries in any other segment transfer table (PL, PL-127).

C. CURRENT PCB POINTER (CPCB)

The current PCB pointer is one word of core in a dedicated location which points to the process control block which is currently executing on the machine.

D. PROCESS CONTROL BLOCK (PCB)

A process control block is associated with each user. This variable length block (entries to be defined) keeps track of the location of the user's stack area, privileged data table, etc.

E. PRIVILEGED DATA TABLE (PDT)

A privileged data table is associated with each user. The location of the privileged data table is known to the system via the process control block (PCB) associated with that user. The entries are descriptor words (to be defined) pointing to file addresses, I/O commands, directories, buffers, etc.

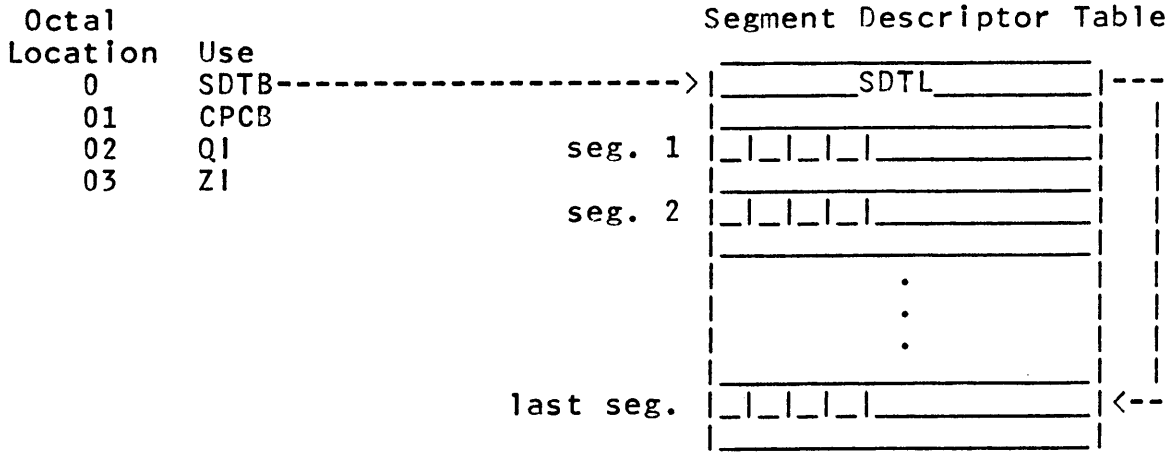
F. DEVICE REFERENCE & EXTERNAL INTERRUPT TABLE (DRT)

This dedicated table contains one 4 word entry for each device number, up to the maximum of 255 devices. The first word of each entry corresponds to a unique device and contains the address of the next I/O command instruction for that device. The second word of each entry contains the absolute program address PI at which execution will begin for that external interrupt, and the third word contains the absolute data base address DBI associated with that external interrupt. Bit 0 of the fourth word is the Interrupt Reference Flag, IRF. It may be set by the SIRF instruction to indicate the presence of an interrupt from the device. Note that device number 0 does not exist.

G. INTERRUPT STACK BASE AND LIMIT (QI & ZI)

These two dedicated locations point to the start and end of the memory area allocated to the interrupt stack.

FIXED MEMORY ALLOCATION



	DRT	
04	I/O prog. dev. 1	I/O program address = 4*device #
05	PI dev. 1	PI address = 4*device # + 1
06	DBI dev.10	DBI address = 4*device # + 2
07	IRF dev. 1	IRF address = 4*device # + 3
10	I/O prog. dev. 2	
11	PI dev. 2	Address calculation above is in octal.
12	DBI dev. 2	
13	IRF dev. 2	
.		
.		
.		
1774	I/O prog. dev. 255	Note that the DRT need contain
1775	PI dev. 255	only as many entries as there
1776	DBI dev. 255	are devices on the system.
1777	IRF dev. 255	

SECTION V - INPUT/OUTPUT

I/O programming is handled by a portion of the CPU microprogram along with special hardware which enables data transfer between devices and memory to proceed in parallel with normal CPU operation.

A. CPU INSTRUCTIONS

There are five CPU I/O instructions and addressing capability for 256 device controllers. The I/O instructions are: Start I/O (SIO), Read I/O (RIO), Write I/O (WIO), Test I/O (TIO), and Control I/O (CIO). They expect a device controller number in the stack. For TIO and for a rejected SIO, RIO or WIO a 16 bit device status word (DSW) is returned to the top of the stack.

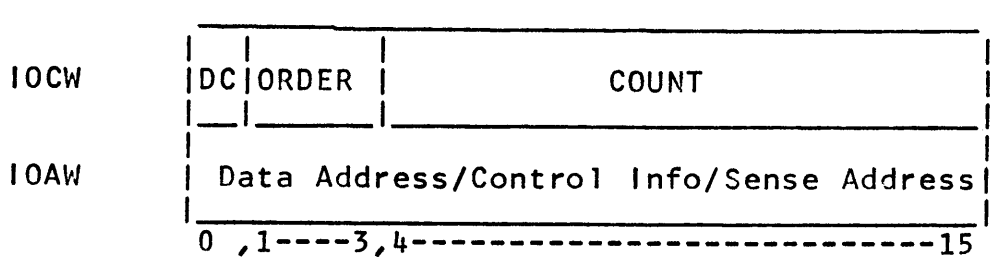
Start I/O (SIO) causes the initiation of an I/O program pointed to by an entry in the Device Reference Table (DRT). The DRT is a table beginning in memory location 4 containing at most 255 four word entries. Each table entry corresponds to a unique device number and the first word contains the address of the next I/O command instruction for that device.

The execution of an SIO causes the transfer of the DRT entry specified by the instruction to be made to the corresponding device controller. The controller then assumes control of the I/O program execution and transfers data to and from the bus. Note that once the device operation has been initiated, the CPU is free to continue processing. Both tasks run concurrently until device termination caused by the appropriate I/O command instruction.

B. SIO PROGRAMMING

An SIO type transfer is initiated by the CPU executing a Start I/O instruction for the desired device, assuming that there is an I/O Command program stored in memory. The DRT entry associated with the device must be pointing to the beginning of the I/O Command program.

The I/O command program consists of a set of doubleword instructions which controls the transfer of data between device and memory. The format of an I/O instruction doubleword is:



DC = Data Chain upon command execution complete.
DC bit should be 1 if the ORDER code of the next sequential I/O program doubleword is the same as the current ORDER. This applies only to READ and WRITE ORDER codes.
ORDER = I/O operation code
COUNT = Transfer count--may be bytes, words or records depending upon the particular device controller.
IOCW = I/O Command Word
IOAW = I/O Address Word--the Transfer address

C. I/O COMMAND CODES

CONDITIONAL JUMP (000) The count field of the IOCW is examined by the controller and a conditional jump to the address given by the IOAW is made at the discretion of the controller.

RETURN RESIDUE (001) This causes the residue of the count to be returned to the memory address given by the IOAW.

INTERRUPT (010) This causes the device controller to interrupt the CPU.

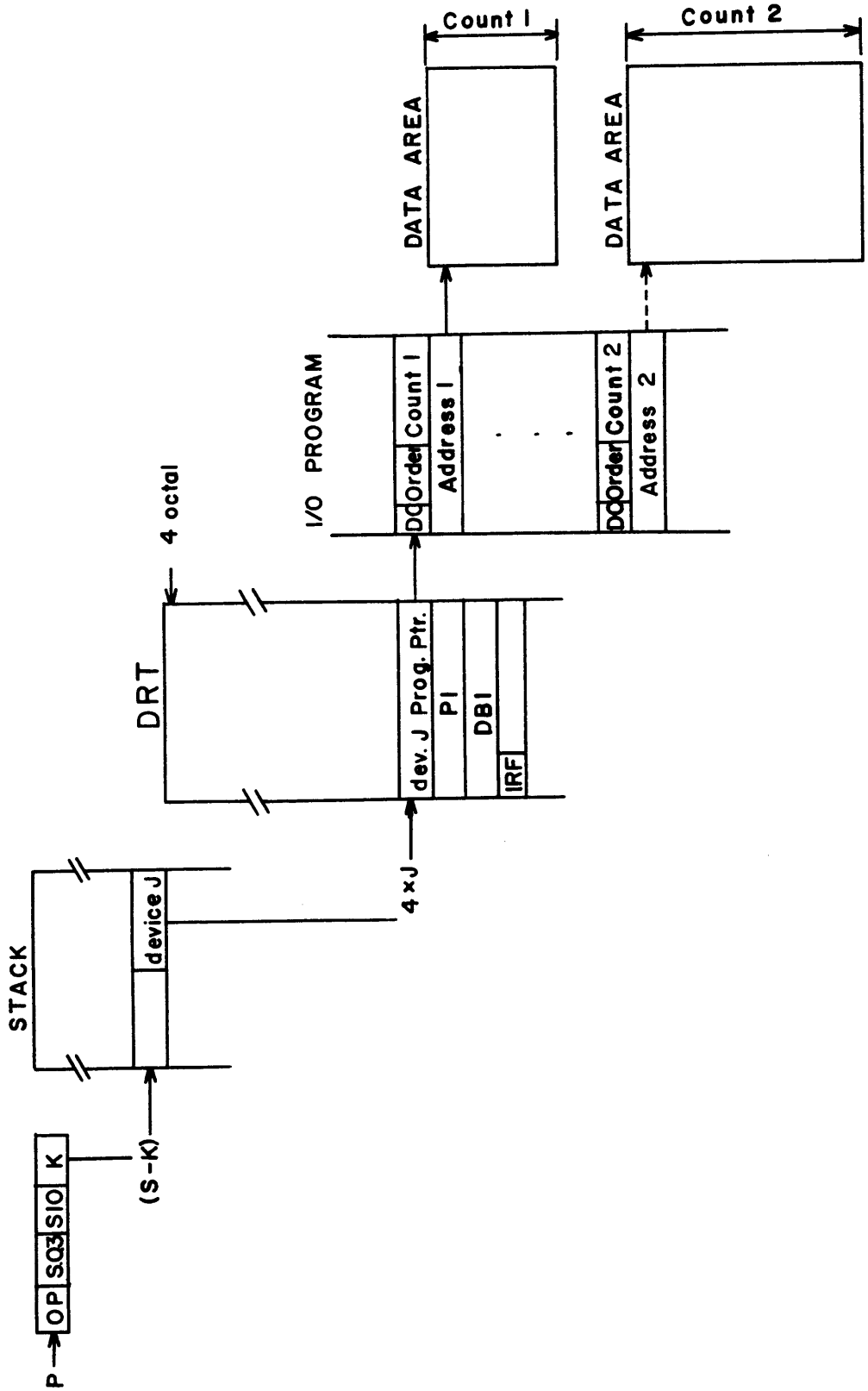
END (011) End of the I/O program.

CONTROL (100) This causes transfer of a 16 bit control word at core location specified by IOAW to the device controller.

SENSE (101) This causes transfer of a 16 bit status word from the device to memory at address given by the IOAW.

WRITE (110) This causes COUNT words of data to be transferred between core and the device starting at the address given by the IOAW.

READ (111) This causes COUNT words of data to be transferred between the device and core starting at the address given by the IOAW.



SIO PROGRAMMING

SECTION VI - INTERRUPT AND TRAP PROCESSING

A. EXTERNAL INTERRUPTS

The external interrupt structure is a "polling" structure with a maximum of 255 devices allowed on the Interrupt Poll (IP) line.

Servicing of the external interrupts is done in descending order of priority, i.e. the highest priority interrupt is serviced first. The interrupt priority of a device is determined by its logical proximity to the CPU on the IP line. The interrupt structure is nested such that a higher priority interrupt can pre-empt a lower one. A 16 bit Mask register is provided for the purpose of masking off groups of external interrupts. Each bit of the MASK may be associated with a device by connecting the device to that bit on the mask portion of the I/O bus. Up to 16 external interrupts may be assigned to a mask group.

In the ALPHA I/O system there are four characteristic numbers or values associated with an I/O device. (Note that they are fixed at hardware system configuration tive. These are Device Number Data Service Priority, Interrupt Priority and Interrupt Mask Number. These characteristic values are all independent of each other, giving the following advantages:

1. Device Numbers may be numbered consecutively, starting at 1 and proceeding to the number of devices on the system. When a new I/O device is added to the system, it is merely assigned the next highest available number, if desired.

2. Since both Data Service and Interrupt Priorities are independent of device number, a new device to the system may be placed anywhere in the priority chain, independent of physical location within the cabinet.

3. Since Data Service Priority and Interrupt Priority are independent of each other, a device which requires a high data transfer rate may be assigned a low interrupt priority, if desired (such as a disc), or a device which has a very low data rate may be configured to a high priority interrupt (such as an alarm condition).

4. Since Interrupt Masks are independent of device numbers and priorities, devices may be masked in groups related to any desired function. For example if two data terminals were on the system and each has both a high speed and a low speed device, the interrupts could be masked for each terminal, rather than on a data rate basis.

Each interrupt has four states: 1. Masked, 2. Unmasked, 3. Pending, 4. Active. An interrupt may not move to the Active state if it is masked off. The interrupt system is automatically turned off when an interrupt occurs and must be turned back on by an instruction in the interrupt routine.

The interrupt response time is defined to be the maximum time that may elapse between the setting of the active state of an interrupt and the start of the execution of the first instruction of the interrupt routine for the highest priority unmasked interrupt. This response time is approximately 25 microseconds.

Upon receipt of an external interrupt by the CPU, the currently executing instruction is terminated, the current DB pointer is pushed onto the stack, and a standard format 3 word stack marker is created on the current stack. The hardware top of stack registers are pushed into core memory such that S = SM and SR = 0. The new value of S (note that Q = S points to the stack marker) is stored into the Process Control Block (PCB) of the non-interrupt process. The stack environment is changed to the Interrupt Stack by setting

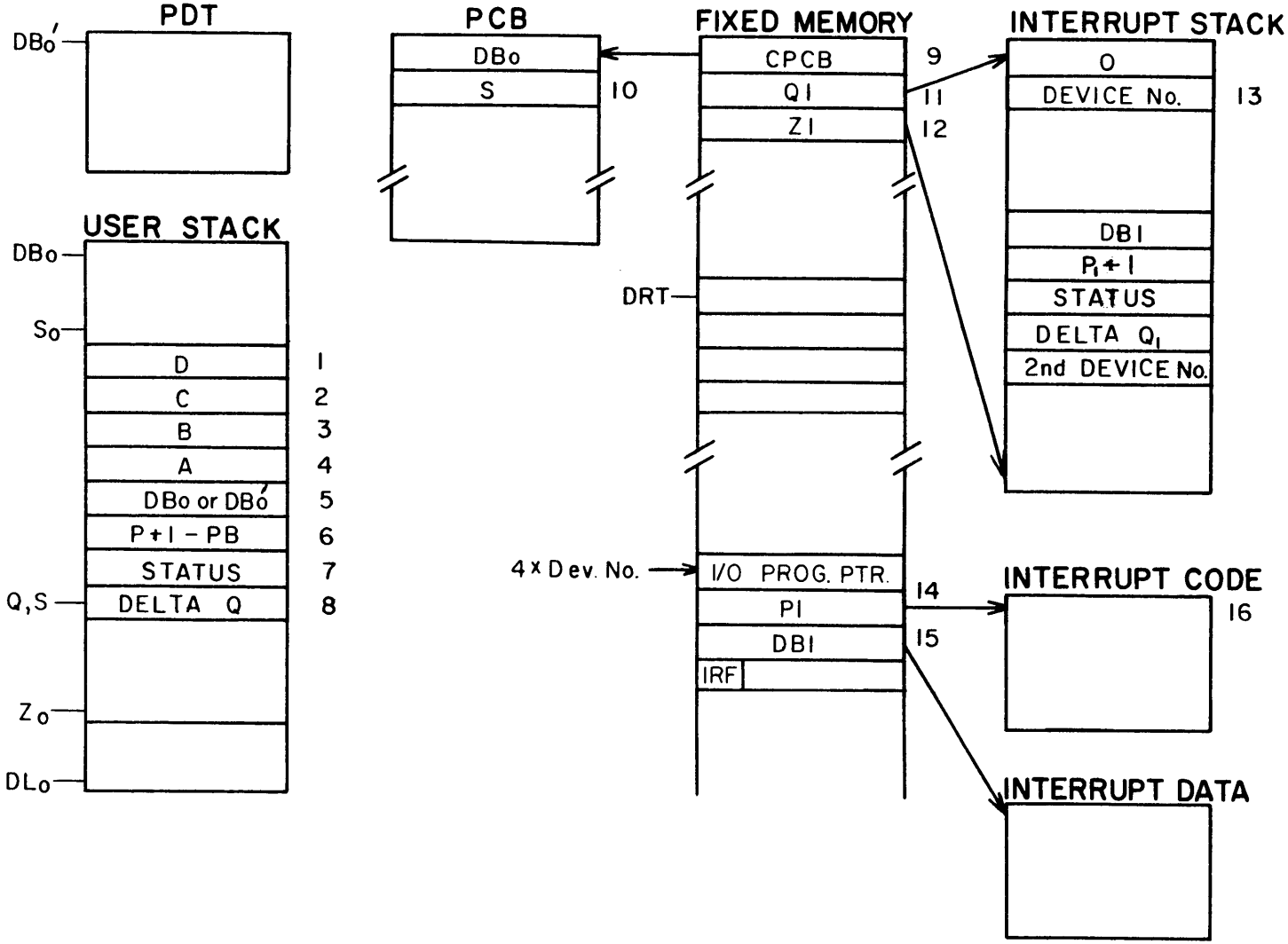
```
Q := QI  <<from memory location 2>>;
Z := ZI  <<from memory location 3>>;
S := Q + 1;
(S) := Device Number of External Interrupt;
<<note that operating system guarantees (Q) = 0>>
```

The CPU references the entry in the Device Reference Table and External Interrupt Table (DRT) for this device number, setting

```
PB := 0;
PL := 2**16 - 1     <<highest addressable memory location>>;
P := PI             <<from DRT>>;
DB := DBI           <<from DRT>>;
STATUS(8:15) := 0   <<segment # = 0 for all ext. intrps>>;
STATUS(0:1) := 10   <<priv. mode, disabled ext. interrupts>>;
```

Code execution begins in privileged mode at the instruction pointed to by the PI entry for the interrupting device. The external interrupt system is disabled.

The steps in the External Interrupt processing described above are shown in the following diagram and explanation:



EXTERNAL INTERRUPT PROCESSING

HP ALPHA 1 7-20-70 COPY 42
HEWLETT-PACKARD PRIVATE--DO NOT REPRODUCE

STEP	ACTION
0	An External Interrupt request is received by the CPU.
1	The four top of stack registers are pushed into
2	memory if they contain valid information. This
3	takes a maximum of four cycles if all four are
4	full.
5	The current value of the user's DB is pushed onto his stack in memory.
6	A normal three word stack marker is created and pushed
7	into memory, saving P + 1 - PB, STATUS and
8	Delta Q.
9	The Current PCB pointer is fetched from location 1.
10	The user's current value of S is stored in his PCB. Note that the PCB always contains the fixed current values DBo, Zo, DLo; they need not be resaved.
11	The Interrupt Stack Base QI is fetched and loaded into the Q register.
12	The Interrupt Stack Limit ZI is fetched and loaded into the Z register.
13	The devices are polled and the highest prioity interrupting device returns its device number. Note that this may not be the same device that issued the original Interrupt Request, if a higher priority device has requested an interrupt in the intervening time. S is set to QI + 1 and the device number is stored at S. The external interrupt system is turned off and no further interrupts are accepted.
14	Using the device number to generate an index into the DRT, the Program Pointer for this device is fetched and stored in the P register. PB is set to zero and PL is set to the highest addressable memory location. The segment # in STATUS is set to zero.
15	The interrupt program's data base, DBI, is fetched from the DRT and loaded into the DB register.
16	The first instruction of the interrupt receiving routine is fetched and execution begins.

The diagram also shows the items stacked when a higher priority device interrupts the current interrupt. DBI, P+1, STATUS, Delta Q are stacked and steps 13 through 16 are followed.

The approximate time for the processes are

a. 16 memory cycles for the first interrupt (steps 1-16)
b. 8 memory cycles for succesive higher priority interrupts (steps 5-8, 13-16)
c. 4 memory cycles for a pending intermediate priority interrupt upon EXIT from the higher priority routine (steps 13-16).

B. INTERNAL INTERRUPTS

The fifteen internal interrupts, including machine check conditions and entry points into the operating system, occupy segment numbers 1 through 15 in the Segment Descriptor Table (SDT); all user related trap conditions enter one code segment, number 15 (see Interrupt and Trap Table).

When an internal interrupt condition is detected by the CPU, the interrupt is serviced as follows:

1. If required, a parameter word is pushed onto the current stack.
2. A new three word standard format stack marker is placed on the stack.
3. The STATUS register is changed to Privileged mode and the segment number set to the SDT entry for the particular interrupt being serviced; the Enable/Disable external interrupts bit is unchanged.
4. An Interrupt CALL is performed through the SDT by the hardware, similar to a CALL instruction except that code execution will begin at PB in the called segment. (This is equivalent to a CALL through STT entry 0.)

C. TRAPS

User related arithmetic and stack faults are trapped into an Interrupt CALL on segment 15, if the Enable/Disable Traps bit in STATUS is enabled. The parameter will be the trap number, as shown in the Interrupt and Trap Table. If traps are disabled, no special action will occur.

INTERRUPT and TRAP TABLE

Seg. #	Priority	Type
0	5	External Interrupts (via DRT and Interrupt Stack)
1	1	Power Fail
2	1	Power On
3	2	Parity Error
4	2	Non-responding Module
5	3	Interrupt from another Module
6	4	Illegal Address (including Memory Protect Violation)
7	4	Stack Underflow (S < DB or Q < DB)
8	6	Interrupt Stack Bottom (EXIT Seg # 0 & Delta Q = 0)
9	4	Code Segment Presence (CALL or EXIT instruction)
10	4	Stack Overflow (S > Z)
11	-	Unassigned
12	-	Unassigned
13	-	Unassigned
14	7	Console Interrupt
15	8	Traps, with parameter =

1, Integer Overflow
2, Floating Point Overflow
3, Floating Point Underflow
4, Integer Divide by Zero
5, Floating Point Divide by Zero
6, Undefined Integer
7, Undefined Floating Point
8, Stack Integrity (S < Q)
9, Privileged Instruction
10, Unimplemented Instruction

D. POWER ON & COLD START PROCESSING

When power is turned on, the data control registers DB, Q, S, Z and DL are initialized from the Process Control Block pointed to by the CPCB pointer (the values were set by the Power Fail routine), and an Interrupt CALL is performed to the Power On internal interrupt, setting PB, P, PL and STATUS from the information in the Segment Descriptor Table (SDT). At this time one of two things will happen:

1. If Automatic Restart from a power failure is enabled by a panel switch, then the Power On interrupt segment will begin execution at P = PB.
2. If Automatic Restart is disabled, the computer will HALT with P = PB pointing to the Power On interrupt segment. At this point the operator must either
   a. manually push the Resume button, causing the program to start execution at location P.

b. Manually push the Cold Load button, which initializes the CPU registers as described below. An I/O program is then generated to load memory from a device specified by the operator-set switches on the panel. The Cold Load program reads an 8 bit device number (DN) from SWITCH(8:15) and a preconditioning I/O device control byte from SWITCH(0:7). The Cold Load program located at the end of the first 4K of memory is as follows:

```
DRT(DN) := 7720; <<I/O program counter>>
STATUS  := Privileged mode, Enable ext. interrupts, Seg. # 0;
PB := P := 7700;
PL := 7704;

(7700)  := SIO 0; <<CPU instruction, device number in (S)>>
(7701)  := BCC CCL or CCG, P+3; <<HALT if SIO rejected>>
(7702)  := PAUS;
(7703)  := BR  P-1;
(7704)  := HALT;
(7705)  := control I/O word = eight 0's and SWITCH(0:7);

DB := Q := SM := 7706;
Z := DL := 7717;
SR := 1;
(S) := device number = SWITCH(8:15);

(7720)  := Control ORDER; <<I/O instruction>>
(7721)  := 7705; <<address of control information>>
(7722)  := Read ORDER, word count = 44decimal; <<maximum>>
(7723)  := 7724; <<input read address>>
```

7724 through 7777 are 44decimal words of load area for the I/O program.

All addresses are given in octal.

SECTION VII - INSTRUCTION SET

MEMORY ADDRESS INSTRUCTIONS

E = Effective address
PDQS = P+, P-, DB+, Q+, Q-, S-
DQS = DB+, Q+, Q-, S-
P = P+, P-
* = Indirect addressing allowable
D = Displacement field of instruction word used in address computation. It is always a word displacement even in byte and double word instructions.
L = Displacement field of Conditional Branch instructions. The range is P - 31, P + 31.
X = Indexing allowed

A. LOAD INSTRUCTIONS

1. LOAD E (*, D, X, PDQS) <Load> Mem. OpCode=04

```
S := S + 1;
(S) := (E);
```

The content of E is pushed onto the stack.
Indicators = CCA

2. LDB E (*, D, X, DQS) <Load byte> Mem. OpCode=15
CIR(6) = 0

```
S := S + 1;
(S(0:7)) := 0;
if CIR(5) = 0 then
begin
   (S(8:15)) := if CIR(4) = 0 or X(15) = 0 then (E(0:7))
   else (E(8:15))
end else <<indirect addressing>>
begin
   BE := (base + D) + if CIR(4) = 0 then 0 else X <<byte>>;
   E := BE/2 <<E-DB < 2**15>>;
   if BE mod 2 = 0 then (S(8:15)) := E(0:7)
      else (s(8:15)) := E(8:15);
end;
```

X contains a byte index. On indirect addressing the word referenced by the direct address (base + D) contains a DB relative byte address. The byte index is added to te relative byte address, and the byte referenced is then loaded onto the stack.
Indicators = CCB

3. LDD E (*, D, X, DQS) <Load double> Mem. OpCode=15
CIR(6) = 1

```
S := S + 2;
(S-1, S) := (E, E+1);
```

X contains a double word index. The double word in E, E+1 is pushed onto the stack, most significant word first.
Indicators = CCA

4. LDPP N <Load double from program, positive> Sub OpCode3=10

```
S := S + 1;
(S) := (P + N);
S := S + 1;
(S) := (P + N + 1);
```

The doubleword contained at P + N is pushed onto the stack.
Indicators = CCA

5. LDPN N <Load double from program, negative> Sub OpCode3=11

```
S := S + 1;
(S) := (P - N);
S := S + 1;
(S) := (P - N + 1);
```

The doubleword in location P - N is pushed onto the stack.
Indicators = CCA

6. LDX E (*, D, X, PDQS) <Load Index> Mem. OpCode=13

```
X := (E);
```

The content of E is loaded into the index register.
Indicators = Unaffected.

7. LRA E (*, D, X, PDQS) <Load relative address> Mem. OpCode=17

```
S := S + 1;
if CIR(6) = 0 then (S) := E-PB else (S) := E-DB;
```

The effective address is computed in the normal manner and then the appropriate base register (PB for P+- addressing, DB for DB+, Q+-, and S- addressing) is subtracted from the absolute address E. The relative address is pushed onto the stack.
Indicators = Unaffected.

B. STORE INSTRUCTIONS

8. STOR E (*, D, X, DQS) <Store> Mem. OpCode=05

```
(E) := (S);
S := S-1;
```

The content of the TOS is stored in memory location E. The stack is popped.
Indicators = Unaffected

9. STB E (*, D, X, DQS) <Store byte> Mem. OpCode=16
CIR(6) = 0

```
if CIR(5) = 0 then
begin
   if CIR(4) = 0 or X(15) = 0 then (E(0:7)) := (S(8:15))
   else (E(8:15)) := (S(8:15));
end else <<indirect addressing>>
begin
   BE := (base + D) + if CIR(4) = 0 then 0 else X <<byte>>;
   E := BE/2 <<E-DB < 2**15>>;
   if BE mod 2 = 0 then (E(8:15)) := (S(8:15))
      else (E(0:7)) := (S(8:15));
end;
S := S - 1;
```

X contains a byte index. On indirect addressing the word referenced by the direct address (base + D) contains a DB relative byte address. The byte index is added to te relative byte address, and the right hand byte of the TOS is stored into the referenced byte. The TOS is deleted.
Indicators = Unaffected

10. STD E (*, D, X, DQS) <Store double> Mem. OpCode=16
CIR(6) = 1

```
(E, E+1) := (S-1, S);
S := S-2;
```

X contains a double word index. The top two words on the stack are stored at E, E+1.
Indicators = Unaffected

C. MEMORY REFERENCE INTEGER INSTRUCTIONS

11. CMPM E (*, D, X, PDQS) <Compare memory> Mem. OpCode=06

```
CC := (S) : (E);
S := S-1;
```

The condition code is set to pattern C as a result of the comparison of (S):(E). The stack is popped.
Indicators = CCC

12. ADDM E (*, D, X, PDQS) <Add memory> Mem. OpCode=07

```
(S) := (S) + (E);
```

The content of E is added in integer form to the TOS. The result replaces the operand on the TOS.
Indicators = CCA, Carry, Overflow

13. SUBM E (*, D, X, PDQS) <Subtract memory> Mem. OpCode=10

```
(S) := (S) - (E);
```

The content of E is subtracted in integer form from the TOS. The result replaces the operand on the TOS.
Indicators = CCA, Carry, Overflow

14. MPYM E (*, D, X, PDQS) <Multiply memory> Mem. OpCode=11

```
(S) := (S) * (E);
```

The TOS is multiplied in integer form by the content of E. The least significant word of the result replaces the operand on the TOS.
Indicators = CCA, Carry, Overflow

15. INCM E (*, D, X, DQS) <Increment memory> Mem. OpCode=12
CIR (6) = 0

```
(E) := (E) + 1;
```

The content of E is incremented by 1 in integer form.
Indicators = CCA, Carry, Overflow

16. DECM E (*, D, X, DQS) <Decrement memory> Mem. OpCode=12
CIR (6) = 1

```
(E) := (E) - 1;
```

The content of E is decremented by 1 in integer form.
Indicators = CCA, Carry, Overflow

D. BRANCH INSTRUCTION

17. BR E (*, D, X, P or indirect DQS) <Branch> Mem. OpCode=14

if CIR(6) = 0 then P := E <<as defined in section III-D>>
else P := PB + (DB or Q or S +- D) + X; <<indirect addr>>

Control is transferred to location E.
Indicators = Unaffected.

E. CONDITIONAL BRANCHES

CIR(10) is the sign of P+- displacement
Range (P-31, P+31)
CIR(4) is the indirect bit for cond. branches
L = signed range

18. BCC L (*, L, P) <Branch on Condition Code> Mem. OpCode=14
CIR(5:6) = 01

If CC AND (CCF) ¬= 0 then P := P + L;

The branch addresses are in the range P-31, P+31.
Condition code field (CCF) in instruction is CIR (7:9).
Control is transferred to location P + L under the following conditions:

If CCF = 0, Never branch
1, Branch if CC = CCL
2, Branch if CC = CCE
3, Branch if CC = CCL or CCE
4, Branch if CC = CCG
5, Branch if CC = CCG or CCL
6, Branch if CC = CCG or CCE
7, Always Branch

Indicators = Unaffected.

19. BCY L (*, L, P) <Branch on carry> Sub OpCode1=14

if STATUS(5) = 1 then P := P + L else P := P + 1;

If the carry bit of the status bit is on, control is transferred to P + L.
Indicators = Carry cleared

20. BNCY L (*, L, P) <Branch on no carry> Sub OpCode1=15

if STATUS(5) = 0 then P := P + L else P := P + 1;

If the carry bit of the status bit is off, control is transferred to P + L.
Indicators = Carry cleared

21. BOV L (*, L, P) <Branch on overflow> Sub OpCode1=30

if STATUS(4) = 1 then P := P + L else P := P + 1;

If the overflow bit in the status word is on, control is transferred to P + L.
Indicators = Overflow cleared

22. BNOV L (*, L, P) <Branch on no overflow> Sub OpCode1=31

if STATUS(4) = 0 then P := P + L else P := P + 1;

If the overflow bit in the status word is off, control is transferred to P + L.
Indicators = Overflow cleared

23. BRO L (*, L, P) <Branch on TOS odd> Sub OpCode1=36

if (S(15)) = 1 then P := P + L;

If the TOS is odd, control is transferred to P + L.
Indicators = Unaffected

24. BRE L (*, L, P) <Branch on TOS even> Sub OpCode1=37

if (S(15)) = 0 then P := P + L;

If the TOS is even, control is transferred to P + L.
Indicators = Unaffected

25. IABZ L (*, L, P) <Increment A branch if zero> Sub OpCode1=07

(S) := (S) + 1;
if (S) = 0 then P := P + L else P := P + 1;

The TOS is incremented. If the result is then zero, control is transferred to P + L.
Indicators = CCA, Overflow

26. IXBZ L (*, L, P) <Increment X, branch if zero> Sub OpCode1=12

```
X := X + 1;
if X = 0 then P := P + L else P := P + 1;
```

The index register is incremented. If the result is then zero, control is transferred to P + L.
Indicators = CCA, Overflow

27. DABZ L (*, L, P) <Decrement A, branch if zero> Sub OpCode1=27

```
(S) := (S) - 1;
if (S) = 0 then P := P + L else P := P + 1;
```

The TOS is decremented. If the result is then zero, control is transferred to P + L.
Indicators = CCA, Overflow

28. DXBZ L (*, L, P) <Decrement X, branch if zero> Sub OpCode1=13

```
X := X - 1;
if X = 0 then P := P + L else P := P + 1;
```

The index register is decremented. If the result is then zero, control is transferred to P + L.
Indicators = CCA, Overflow

29. CPRB L <Compare range and branch> Sub OpCode1=26

```
if (S-1) <= X <= (S) then
begin
   CC := CCE;
   P := P + L;
end;
if X < (S-1) then CC := CCL;
if X > (S) then CC := CCG;
S := S - 2;
```

The integer in the index register is tested to determine if it is within the interval defined by the upper bound integer on the TOS and the lower bound integer found in the second word of the stack. The condition code is set by the comparison. If the integer in X is within the range then the branch to P + L occurs.
Indicators = Condition Code

F. LOOP CONTROL BRANCH INSTRUCTIONS

30. TBA E (D, P) <Test, branch, A> Mem. OpCode=05
CIR(4:6) = 000

```
VAR := ((S-2));
STEP := (S-1);
FINAL := (S);
if STEP >= 0 then
   if VAR > FINAL then S := S - 3 else P := E;
if STEP < 0 then
   if VAR < FINAL then S := S - 3 else P := E;
```

This instruction requires that the top 3 elements of the stack are (TOS, TOS -1, TOS -2) the final value, the step size and the address (DB+ relative) of the variable respectively. VAR is tested against the limit. If the limit is exceeded, then pop the top 3 words from the stack and execution continues at P + 1. If the limit is not exceeded, then control is transferred to E. The branch addresses are in the range P-255, P+255.
Indicators = Unaffected.

31. MTBA E (D, P) <Modify, Test, Branch, A> Mem. OpCode=05
CIR(4:6) = 010

```
VAR := ((S-2));
STEP := (S-1);
FINAL := (S);
VAR := VAR + STEP;
((S-2)) := VAR;
if STEP >= 0 then
   if VAR > FINAL then S := S - 3 else P:=E;
if STEP < 0 then
   if VAR < FINAL then S := S - 3 else P:=E;
```

This instruction requires that the top 3 elements of the stack are (TOS, TOS -1, TOS -2) the final value, the step size and the address (DB+ relative) of the variable respectively. The step size is added in integer form to the variable, the sum replaces the old value of the variable, and the sum is tested against the limit. If the limit is exceeded then pop all 3 from the stack and execution continues at P + 1. IF the limit is not exceeded, then control is transfered to E. The branch addresses are in the range P-255, P+255.
Indicators = Unaffected.

32. TBX E (D, P) <Test, branch, X> Mem. OpCode=05
CIR(4:6) = 100

```
VAR := X;
STEP := (S-1);
FINAL := (S);
if STEP >= 0 then
   if VAR > FINAL then S := S - 2 else P := E;
if STEP < 0 then
   if VAR < FINAL then S := S - 2 else P := E;
```

This instruction requires that the the index register contains the variable, and that the top 2 words of the stack are (TOS, TOS - 1) the final value and the step size respectively. VAR is tested against the limit. If the limit is exceeded, then pop the top 2 words from the stack and execution continues at P + 1. If the limit is not exceeded, then control is transferred to E.
Indicators = Unaffected.

33. MTBX E (D, P) <Modify, Test, Branch, X> Mem. OpCode=05
CIR(4:6) = 110

```
VAR := X;
STEP := (S-1);
FINAL := (S);
VAR := VAR + STEP;
X := VAR;
if STEP >= 0 then
   if VAR > FINAL then S := S - 2 else P:=E;
if STEP < 0 then
   if VAR < FINAL then S := S - 2 else P:=E;
```

This instruction requires that the the index register contains the variable, and that the top 2 words of the stack are (TOS, TOS - 1) the final value and the step size respectively. The step size is added in integer form to the variable, the sum replaces the old value of the variable, and the sum is tested against the limit. If the limit is exceeded, then pop the top 2 words from the stack and execution continues at P + 1. If the limit is not exceeded, then control is transferred to E. The branch addresses are in the range P-255, P+255.
Indicators = Unaffected.

G. INTEGER INSTRUCTIONS

34. CMP <Compare> Stack OpCode=17

```
C := (S-1) : (S);
S := S - 2;
```

The condition code is set to pattern C as a result of the integer comparison of (S-1):(S). The operands are deleted.
Indicators = CCC

35. ADD <Add> Stack OpCode=20

```
(S-1) := (S-1) + (S);
S := S - 1;
```

The top two words of the stack are added in integer form and they are then popped. The resulting sum is pushed onto the stack.
Indicators = CCA, Carry, Overflow

36. SUB <Subtract> Stack OpCode=21

```
(S-1) := (S-1) - (S);
S := S - 1;
```

The top word of the stack is subtracted in integer form from the second word and they are popped. The resulting difference is pushed onto the stack.
Indicators = CCA, Carry, Overflow

37. MPY <Multiply> Stack OpCode=22

```
(S-1) := (S-1) * (S);
```

The top two words of the stack are multiplied in integer form. The two words are deleted, and the least significant word of the double length product is pushed onto the stack.

Carry is cleared if the high order 17 bits (including the sign bit of the second word) were either all zeroes or all ones, such that the low order 16 bits represent the true result. Otherwise, the carry bit is set.
Indicators = CCA, Carry, Overflow

38. DIV <Divide> Stack OpCode=23

```
TEMP := (S-1)/(S);
(S) := (S-1) mod (S);
(S-1) := TEMP;
```

The integer in the second word of the stack is divided by the integer on the TOS. The second word is replaced by the quotient, and the top word is replaced by the remainder.
Indicators = CCA, Overflow

39. NEG <Negate> Stack OpCode=24

```
(S) := -(S);
```

The TOS is replaced by its 2's complement.
Indicators = CCA, Overflow

H. LOGICAL INSTRUCTIONS

40. LCMP <Logical compare> Stack OpCode=57

```
CC := (S-1):(S);
S := S - 2;
```

The condition code is set to pattern C on the comparison of (S-1):(S). The two operands are deleted.
Indicators = CCC

41. LADD <Logical add> Stack OpCode=60

```
(S-1) := (S-1) + (S) mod 2**16;
S := S - 1;
```

The top two words of the stack are added as 16 bit positive integers and they are deleted from the stack. The resulting sum is pushed onto the stack.
Indicators = CCA, Carry

42. LSUB <Logical subtract> Stack OpCode=61

```
(S-1) := (S-1) - (S) mod 2**16;
S := S - 1;
```

The top word of the stack is subtracted in logical form from the second word and they are deleted. The resulting difference is pushed onto the stack.
Indicators = CCA, Carry

43. LMPY <Logical multiply> Stack OpCode=62

```
(S-1, S) := (S-1) * (S);
```

The top two words on the stack are multiplied as 16 bit positive integers. The words are replaced by the double length product with the most significant half on the TOS.

Carry is cleared if the high order 16 bits are all zeroes such that the low order 16 bits represent the true result. Otherwise the carry bit is set.

Indicators = CCA, Carry

44. LDIV <Logical divide> Stack OpCode=63

```
TEMP := logical((S-2, S-1)/(S));
(S-1) := (S-2, S-1) mod (S);
(S-2) := TEMP;
S := S - 1;
```

The 32 bit positive integer in the 2nd and 3rd words of the stack is divided by the 16 bit positive integer on the TOS. The top three words are deleted. The quotient is pushed onto the stack and then the remainder is pushed on.

Indicators = CCA, Overflow

45. NOT <One's complement> Stack OpCode=64

```
(S) := ¬(S);
```

Converts the top word of the stack to its one's complement.

Indicators = CCA

I. DOUBLE INTEGER INSTRUCTIONS

46. DCMP <Double compare> Stack OpCode=10

```
CC := (S-3, S-2):(S-1, S);
S := S - 4;
```

The condition code is set to pattern C as a result of the double word integer comparison of (S-3, S-2):(S-1, S). The two double words are deleted from the stack.
Indicators = CCC

47. DADD <Double add> Stack OpCode=11

```
(S-3, S-2) := (S-3, S-2) + (S-1, S);
S := S - 2;
```

The two double word integers contained in the top 4 elements of the stack are added in double length integer form and they are deleted. The double word integer sum is pushed onto the stack.
Indicators = CCA, Carry, Overflow

48. DSUB <Double subtract> Stack OpCode=12

```
(S-3, S-2) := (S-3, S-2) - (S-1, S);
S := S - 2;
```

The double word integer contained in the 1st and 2nd words of the stack is subtracted from the double word integer contained in the 3rd and 4th words of the stack. The top 4 words of the stack are deleted and the double word integer result is pushed onto the stack.
Indicators = CCA, Carry, Overflow

49. MPYL <Multiply Long> Stack OpCode=13

```
(S-1, S) := (S-1) * (S);
```

The top two words of the stack are multiplied in integer form. The words are replaced by the double length product, with the least significant half on the TOS.

Carry is cleared if the high order 17 bits (including the sign bit of the second word) are either all zeroes or all ones, such that the low order 16 bits represent the true result. Otherwise, the carry bit is set.
Indicators = CCA, Carry, Overflow

50. DIVL <Divide Long> Stack OpCode=14

```
TEMP1 := (S-2, S-1)/(S);
TEMP2 := (S-2, S-1) mod (S);
S := S - 1;
(S) := TEMP2;
(S-1) := TEMP1;
```

The double word integer in the second and third elements of the stack is divided by the integer in the TOS. The three words are deleted, and the quotient is pushed onto the stack. The remainder is pushed onto the stack.
Indicators = CCA, Overflow

51. DNEG <Double negate> Stack OpCode=15

```
(S-1, S) := - (S-1, S);
```

The double word integer contained in the top 2 words of the stack is negated and replaces the original double word integer.
Indicators = CCA, Carry

J. FLOATING POINT INSTRUCTIONS

52. FCMP <Floating compare> Stack OpCode=50

```
CC := (S-3, S-2):(S-1, S);
S := S - 4;
```

The condition code is set to pattern C as a result of the floating point comparison of (S-3, S-2):(S-1, S). The two floating point doublewords are deleted.
Indicators = CCC

53. FADD <Floating add> Stack OpCode=51

```
(S-3, S-2) := (S-3, S-2) + (S-1, S);
S := S - 2;
```

The two floating point numbers contained in the top 4 words of the stack are added in floating point form. The top 4 words of the stack are deleted and the 2 word sum is pushed onto the stack.
Note: Computed result = round (true result, 22);
Indicators = CCA, Overflow

54. FSUB <Floating subtract> Stack OpCode=52

```
(S-3, S-2) := (S-3, S-2) - (S-1, S);
S := S - 2;
```

The floating point number contained in the 1st and 2nd words of the stack is subtracted from the floating point number contained in the 3rd and 4th words of the stack in floating point form. The top 4 words of the stack are deleted and the 2 word difference is pushed onto the stack.
Note: Computed result = round (true result, 22);
Indicators = CCA, Overflow

55. FMPY <Floating multiply> Stack OpCode=53

```
(S-3, S-2) := (S-3, S-2) * (S-1, S);
S := S - 2;
```

The two floating point numbers contained in the top 4 words of the stack are multiplied in floating point form. The top 4 words of the stack are deleted and the 2 word result is pushed onto the stack.
Note: Computed result = round (true result, 22);
Indicators = CCA, Overflow

56. FDIV <Floating divide> Stack OpCode=54

```
(S-3, S-2) := (S-1, S) / (S-3, S-2);
S := S - 2;
```

The floating point number contained in the top 2 words of the stack is divided by the floating point number contained in the 3rd and 4th words of the stack. The top 4 words of the stack are deleted and the 2 word quotient is pushed onto the stack.
Note: Computed result = round (true result, 22);
Indicators = CCA, Overflow

57. FNEG <Floating negate> Stack OpCode=55

```
(S-1, S) := -(S-1, S);
```

The floating point number contained in the top 2 words of the stack is negated in floating point form.
Indicators = CCA

58. FLT <Float> Stack OpCode=47

```
S := S + 1;
(S-1, S) := float (S-1);
```

Converts the integer on the TOS to a 32 bit floating point number with rounding. The TOS is deleted and the double word floating point result is pushed onto the stack.
Note: Computed result = round (true result, 22);
Indicators = CCA

59. DFLT <Double float> Stack OpCode=30

```
(S-1, S) := dfloat((S-1, S));
```

Converts the double word integer contained in the top 2 words of the stack to a floating point number with rounding.
Note: Computed result = round (true result, 22);
Indicators = CCA

60. FIXT <Fix and truncate> Stack OpCode=71

```
(S-1, S) := trunc((S-1, S));
```

The floating point number contained in the top two words of the stack is truncated and converted to fixed point form.
Carry is cleared if the high order 17 bits (including the sign bit of the second word) are either all zeroes or all ones, such that the low order 16 bits represent the true result. Otherwise, the carry bit is set.
Indicators = CCA, Carry, Overflow

61. FIXR <Fix and round> Stack OpCode=70

```
(S-1, S) := trunc((S-1, S) + .5*sign(S-1, S));
```

The floating point number contained in the top 2 words of the stack is rounded and converted to fixed point form.
Carry is cleared if the high order 17 bits (including the sign bit of the second word) are either all zeroes or all ones, such that the low order 16 bits represent the true result. Otherwise, the carry bit is set.
Indicators = CCA, Carry, Overflow

K. BOOLEAN INSTRUCTIONS

62. OR <Or, logical> Stack OpCode=65

```
(S-1) := (S-1) or (S);
S := S-1;
```

The top two words of the stack are logically inclusive ORed together, the operands are deleted and the result is pushed onto the stack.
Indicators = CCA on the new TOS

63. XOR <Exclusive or, logical> Stack OpCode=66

```
(S-1) := ¬(S-1)and(S) or (S-1)and¬(S);
S := S-1;
```

The top two words of the stack are logically exclusive ORed together, the operands are deleted, and the result is pushed onto the stack.
Indicators = CCA on the new TOS

64. AND <And, logical> Stack OpCode=67

```
(S-1) := (S-1) and (S);
S := S - 1;
```

The top two words of the stack are logically ANDed together, the operands are deleted and the result is pushed onto the stack.
Indicators = CCA on the new TOS

L. TEST INSTRUCTIONS

65. TEST <Test TOS> Stack OpCode=25

Indicators = CCA on (S)

66. DTST <Test double word on TOS> Stack OpCode=27

Carry is cleared if the high order 17 bits (including the sign bit of the second word) are either all zeroes or all ones, such that the low order 16 bits represent the true result. Otherwise, the carry bit is set.
Indicators = CCA on double word (S-1, S)

67. BTST <Test byte on TOS> Stack OpCode=31

M. INCREMENT & DECREMENT INSTRUCTIONS

68. INCA <Increment A> Stack OpCode=33

(S) := (S) + 1;

The TOS is incremented in integer form by one.
Indicators = CCA, Carry, Overflow

69. INCB <Increment B> Stack OpCode=73

(S-1) := (S-1) + 1;

The second word of the stack is incremented in integer form by one. The TOS is unchanged.
Indicators = CCA, Carry, Overflow

70. INCX <Increment Index> Stack OpCode=04

X := X + 1;

The content of the index register is incremented by one in integer form.
Indicators = CCA, Carry, Overflow

71. DECA <Decrement A> Stack OpCode=34

(S) := (S) - 1;

The TOS is decremented in integer form by one.
Indicators = CCA, Carry, Overflow

72. DECB <Decrement B> Stack OpCode=74

(S-1) := (S-1) - 1;

The second word of the stack is decremented in integer form by one. The TOS is unaffected.
Indicators = CCA, Carry, Overflow

73. DECX <Decrement X> Stack OpCode=05

X := X - 1;

The content of the index register is decremented by one in integer form.
Indicators = CCA, Carry, Overflow

N. ZERO INSTRUCTIONS

74. ZERO <Push zero> Stack OpCode=06

```
S := S + 1;
(S) := 0;
```

A zero word is pushed onto the stack.
Indicators = Unaffected.

75. DZRO <Double push zero> Stack OpCode=07

```
S := S + 1;
(S) := 0;
S := S + 1;
(S) := 0;
```

Two words containing all zeroes are pushed onto the stack.
Indicators = Unaffected.

76. ZROB <Zero B> Stack OpCode=41

```
(S-1) := 0;
```

The second word of the stack is replaced by zero. The TOS is unaffected.
Indicators = Unaffected.

77. ZROX <Zero X> Stack OpCode=03

```
X := 0;
```

The content of the index register is replaced by zero.
Indicators = Unaffected.

0. DUPLICATE & DELETE INSTRUCTIONS

78. DEL <Delete A> Stack OpCode=40

```
S := S - 1;
```

The TOS is deleted.
Indicators = Unaffected

79. DDEL <Double delete> Stack OpCode=02

```
S := S - 2;
```

The top 2 words of the stack are deleted.
Indicators = Unaffected

80. DELB <Delete B> Stack OpCode=01

```
(S-1) := (S);
S := S - 1;
```

The second word of the stack is deleted and the stack is compressed. The TOS is unchanged.
Indicators = Unaffected.

81. DUP <Duplicate A> Stack OpCode=45

```
S := S + 1;
(S)  := (S-1);
```

The top word of the stack is duplicated by pushing a copy of the TOS onto the stack.
Indicators = CCA

82. DDUP <Double duplicate> Stack OpCode=46

```
S := S + 2;
(S-1, S) := (S-3, S-2);
```

The double word in the top two words of the stack is duplicated by pushing a copy of it onto the stack.
Indicators = CCA on the double word integer.

P. EXCHANGE INSTRUCTIONS

83. XCH <Exchange A and B> Stack OpCode=32

```
TEMP := (S-1);
(S-1) := (S);
(S) := TEMP;
```

The top 2 words of the stack are interchanged.
Indicators = CCA on the new TOS

84. DXCH <Double exchange> Stack OpCode=16

```
TEMP := (S-1, S);
(S-1, S) := (S-3, S-2);
(S-3, S-2) := TEMP;
```

The top two double word pairs are interchanged on the stack.
Indicators = CCA on the new TOS double word

85. XAX <Exchange A and X> Stack OpCode=35

```
TEMP := X;
X := (S);
(S) := TEMP;
```

The content of TOS and the index register are interchanged.
Indicators = CCA on the new TOS

86. XBX <Exchange B and X> Stack OpCode=75

```
TEMP := X;
X := (S-1);
(S-1) := TEMP;
```

The second word of the stack is interchanged with the content of the index register.
Indicators = Unaffected

87. CAB <Rotate ABC> Stack OpCode=56

```
TEMP := (S-2);
(S-2) := (S-1);
(S-1) := (S);
(S) := TEMP;
```

The third word of the stack is removed from the stack, the top two words are compressed onto the rest of the stack, and the original third word is pushed onto the stack.
Indicators = CCA on the new TOS

Q. INDEX TRANSFER INSTRUCTIONS

88. LDXA <Load X onto stack> Stack OpCode=44

```
S := S + 1;
(S) := X;
```

The content of the index register is pushed onto the stack.
Indicators = CCA on the new TOS.

89. LDXB <Load X into B> Stack OpCode=42

```
(S-1) := X;
```

The second word of the stack is replaced by the content of the index register. The TOS is unaffected.
Indicators = CCA on the new B

90. STAX <Store A into X> Stack OpCode=43

```
X := (S);
S := S-1;
```

The TOS replaces the content of the index register, and the stack is popped.
Indicators = CCA on the new X

91. STBX <Store B into X> Stack OpCode=26

```
X := (S-1);
```

The second word of the stack replaces the content of the index register. The stack is unchanged.
Indicators = CCA on the new X

R. INDEX ARITHMETIC INSTRUCTIONS

92. ADAX <Add A to X> Stack OpCode=36

```
X := X + (S);
```

The TOS is added to the content of the index register in integer form. The sum replaces the content of the index.
Indicators = CCA on the new X, Carry, Overflow

93. ADBX <Add B to X> Stack OpCode=76

```
X := X + (S-1);
```

The second word of the stack is added to the content of the content of the index register in integer form, and the result replaces the content of the index register.
Indicators = CCA on new X, Carry, Overflow

94. ADXA <Add X to A> Stack OpCode=37

```
(S) := (S) + X;
```

The content of the index register is added to the TOS, and the sum replaces the TOS.
Indicators = CCA on the new TOS, Carry, Overflow

95. ADXB <Add X to B> Stack OpCode=77

```
(S-1) := (S-1) + X;
```

The content of the index register is added to the second word of the stack in integer form, and the result replaces the second word of the stack.
Indicators = CCA on new (S-1), Carry, Overflow

S. CONTROL INSTRUCTION

96. NOP <No operation> Stack OpCode=00

```
;
```

The users program space and data space remain unchanged.
Indicators = Unaffected.

SHIFTS ---------- Memory opcode = 01
Sub OpCode = CIR(5:9)
Bit 4 is a true index bit
L = (CIR(10:15) + (if CIR(4) = 1
then X else 0)) mod 64;

T. SINGLE WORD SHIFT INSTRUCTIONS

97. ASL L <Arithmetic shift left> Sub OpCode1=00

The TOS is shifted left L bits, preserving the sign bit.
Indicators = CCA

98. ASR L <Arithmetic shift right> Sub OpCode1=01

The TOS is shifted right L bits, propagating the sign bit.
Indicators = CCA

99. LSL L <Logical shift left> Sub OpCode1=02

The TOS is shifted left L bits logically.
Indicators = CCA

100. LSR L <Logical shift right> Sub OpCode1=03

The TOS is shifted right L bits logically.
Indicators = CCA

101. CSL L <Circular shift left> Sub OpCode1=04

The TOS is shifted left L bits circularly.
Indicators = CCA

102. CSR L <Circular shift right> Sub OpCode1=05

The TOS is shifted right L bits circularly.
Indicators = CCA

103. SCAN <Scan bits> Sub OpCode1=06

```
if CIR(4) = 0 then X := 0;
if (S) ¬= 0 then
begin
   while (S(0)) = 0 then
      begin
         logical shift left  A by 1;
         X := X + 1;
      end;
   logical shift left A by 1;
end else X := X + 16;
```

The TOS is shifted left until S(0) = 1, then shifted left one more bit. The shift count is left in the index register.
Indicators = CCA on final TOS

U. DOUBLE WORD SHIFT INSTRUCTIONS

104. DASL L <Double arithmetic shift left> Sub OpCode1=20

The doubleword contained in (S-1, S) is shifted left L bits, preserving the sign bit, (S-1(0)).
Indicators = CCA

105. DASR L <Double arithmetic shift right> Sub OpCode1=21

The doubleword contained in (S-1, S) is shifted right L bits, propagating the sign bit, (S-1(0)).
Indicators = CCA

106. DLSL L <Double logical shift left> Sub OpCode1=22

The double word contained in (S-1, S) is shifted left L bits logically.
Indicators = CCA

107. DLSR L <Double logical shift right> Sub OpCode1=23

The double word contained in (S-1, S) is shifted right L bits logically.
Indicators = CCA

108. DCSL L <Double circular shift left> Sub OpCode1=24

The double word contained in (S-1, S) is shifted left L bits circularly.
Indicators = CCA

109. DCSR L <Double circular shift right> Sub OpCode1=25

The double word contained in (S-1, S) is shifted ritht L bits circularly.
Indicators = CCA

V. TRIPLE WORD SHIFT INSTRUCTIONS

110. TASL L <Triple arithmetic shift left> Sub OpCode1=10

The 3 word integer contained in (S-2, S-1, S) is shifted left L bits preserving the sign bit, (S-2(0)).
Indicators = CCA

111. TASR L <Triple arithmetic shift right> Sub OpCode1=11

The 3 word integer contained in (S-2, S-1, S) is shifted right L bits propagating the sign bit, (S-2(0)).
Indicators = CCA

112. TNSL <Triple normalizing shift left> Sub OpCode1=16

```
if CIR(4) = 0 then X := 0;
if (S-2, S-1, S) ¬= 0 then
begin
   while (S-2(0)) = 0 do
     begin
       X := X + 1;
       arithmetic left shift 1 of (S-2, S-1, S);
     end;
end else X := X + 42;
```

The top 3 words of the stack are shifted left arithmetically until (S-2(6)) = 0. The shift count is stored in the index register.
Indicators = CCA on final value of (S-2,S-1,S)

W. BIT TEST INSTRUCTIONS

113. TRBM L <Test and reset bit in memory> Sub OpCode3=14

```
TEMP := (DB + N) and ¬(S); <<clear bit>>
(S) := (DB + N) and (S); <<get bit>>
(DB + N) := TEMP;
```

All bit positions of the DB+ relative memory word that have a 1 in the corresponding position on the top of the stack are cleared; at the same time the memory word replaces the TOS. Interrupts may not occur during the execution of this instruction.
Indicators = CCA on new TOS

114. TBC L <Test bit and set condition code> Sub OpCode1=32

The bit position of the TOS to be tested is specified by (CIR(10:15) + (if CIR(4) ¬= 0 then X else 0)) mod 16.
Note that CIR(4) is a true index bit for the bit number.

Indicators = CCE if the bit was 0
CCL or CCG if the bit was 1

115. TRBC L <Test and reset bit, set condition code> Sub OpCode1=33

This instructions operation is identical to that of TBC except that the tested bit is reset to 0 after the test.
Indicators = CCE if the bit was 0
CCL or CCG if the bit was 1

116. TSBC L <Test, set bit, set condition code> Sub OpCode1=34

This instructions operation is identical to that of TBC except that the tested bit is set to 1 after the test.
Indicators = CCE if the bit was 0
CCL or CCG if the bit was 1

117. TCBC L <Test and complement bit and set CC> Sub OpCode1=35

This instructions operation is identical to that of TBC except that the tested bit is complemented after the test.
Indicators = CCE if the bit was 0
CCL or CCG if the bit was 1

X. FIELD INSTRUCTIONS

J = (CIR(8:11))

K = (CIR(12:15))

118. EXF J,K <Extract field> Sub OpCode2=15

0<=J<=15 specifies the first (leftmost) bit in the source field. 0<= K <=15 specifies the number of bits in the field.

```
(S(16-K:15)) := (S(J:J+K-1));
(S(0:15-K)) := 0;
```

Bits J, J+1, J+2, ... , J+K-1 are extracted from the TOS, and the TOS is deleted. The K+1 bit field is right justified with high order zeroes, and this result is pushed onto the stack.
Indicators = CCA on new TOS

119. DPF J,K <Deposit field> Sub OpCode2=16

0<=J<=15 specifies the first bit in the destination field. 0<= K <=15 specifies the number of bits in the field.

```
(S-1(J:J+K-1)) := (S(16-K:15));
(S-1(0:J)) := (S-1(0:J));
(S-1(J+K:15)) := (S-1(J+K:15));
S := S - 1;
```

The K least significant bits of the TOS are placed in bits J, J+1, ..., J+K-1 of the second word of the stack; the remaining bits of the second word of the stack are unchanged. The source operand is deleted from the stack.
Indicators = Unaffected.

Y. IMMEDIATE INSTRUCTIONS

N = (CIR(8:15)) 0 <= N <= 255

120. LDI =N <Load immediate> Sub OpCode2=02

```
S := S + 1;
(S) := N;
```

The immediate positive quantity N is pushed onto the stack as a positive integer.
Indicators = CCA on the new TOS

121. LDNI =N <Load negative immediate> Sub OpCode2=12

```
S := S + 1;
(S) := -N:
```

The immediate positive quantity N is 2's complemented and pushed onto the stack as a negative integer.
Indicators = CCA on the new TOS

122. CMPI =N <Compare immediate> Sub OpCode2=04

```
CC := (S):N;
S := S - 1;
```

The condition code is set to pattern C as a result of the comparison of the TOS with the positive quantity N. The TOS is deleted.
Indicators = CCC

123. CMPN =N <Compare negative immediate> Sub OpCode2=14

```
CC := (S) : -N
S := S - 1;
```

The condition code is set to pattern C as a result of the comparison of the TOS with the 2's complement of the positive quantity N. The TOS is deleted.
Indicators = CCC

124. ADDI =N <Add immediate> Sub OpCode2=05

```
(S) := (S) + N;
```

The immediate positive quantity N is added to the TOS in integer form, and the sum replaces the TOS.
Indicators = CCA on the new TOS, Carry, Overflow

125. SUBI =N <Subtract immediate> Sub OpCode2=06

(S) := (S) - N;

The immediate positive quantity N is subtracted from the TOS is integer form, and the result replaces the TOS.
Indicators = CCA, Carry, Overflow

126. MPYI =N <Multiply immediate> Sub OpCode2=07

(S) := (S) * N;

The immediate positive quantity N is multiplied with the TOS in integer form; the 16 bit integer result replaces the TOS.
Indicators = CCA on the new TOS, Overflow

127. DIVI =N <Divide immediate> Sub OpCode2=10

(S) := (S) div N;

The immediate positive quantity N is divided into the TOS in integer form; the 16 bit integer quotient replaces the TOS.
Indicators = CCA on the new TOS, Carry, Overflow

128. ORI =N <Logical OR immediate> Sub OpCode3=15

(S) := (S) or N;

The immediate positive quantity n is expanded to 16 bits with high order zeroes and inclusive ORed with the TOS; the result replaces the TOS.
Indicators = CCA

129. XORI =N <Logical Exclusive OR immediate> Sub OpCode3=16

(S) := ¬(S) and N or (S) and ¬N;

The immediate positive quantity n is expanded to 16 bits with high order zeroes and exclusive ORed with the TOS; the result replaces the TOS.
Indicators = CCA

130. ANDI =N <Logical AND immediate> Sub OpCode3=17

(S) := (S) and N;

The immediate positive quantity N is expanded to 16 bits with high order zeroes and ANDed with the TOS; the result replaces the TOS.
Indicators = CCA

Z. IMMEDIATE INDEX INSTRUCTIONS

131. LDXI =N <Load X immediate> Sub OpCode2=03

X := N;

The index register is loaded with the positive immediate operand N.
Indicators = Unaffected.

132. LDXN =N <Load X negative immediate> Sub OpCode2=13

X := -N;

The index register is loaded with the 16 bit 2's complement of the immediate operand.
Indicators = Unaffected.

133. ADXI =N <Add immediate to X> Sub OpCode2=00

X := X + N;

The immediate positive quantity N is added to the content of the index register in integer form.
Indicators = CCA on X

134. SBXI =N <Subtract immediate from X> Sub OpCode2=01

X := X - N;

The immediate positive quantity N is subtracted from the content of the index register in integer form.
Indicators = CCA on X

AA. PROGRAM CONTROL INSTRUCTIONS

135. PCAL N <Procedure call> Sub OpCode3=02

```
if N = 0 then
   begin
     TEMP := (S);
     S := S - 1;
   end else TEMP := (PL - N);
S := S + 1;
(S) := P +1 - PB;
S := S + 1;
(S) := STATUS;
S := S + 1;
(S) := S - Q;
Q := S;
P := evaluation of label in TEMP;
```

Control is transferred to the location evaluated by the label contained on the TOS if N = 0 or by the label contained in PL - N otherwise. Then a three word stack marker is placed on the stack. The labels have the two forms discussed in section IV - D. If reference is made to a local label, transfer is made to the PB relative address and the stack linkage is created ignoring the U bit.

If reference is made to an external label whose Segment descriptor table number is different than the present number and the machine is in user mode, the U bit of the target segment linkage must not be set, otherwise a trap occurs. Furthermore if an external label is referenced a local label must occur in the STT table of the called segment.

Indicators = Unaffected

136. SCAL N <Subroutine Call > Sub OpCode3=01

```
if N = 0 then
   begin
     TEMP := (S);
     S := S - 1;
   end else TEMP := (PL - N);
S := S + 1;
(S) := P + 1 - PB;
P := evaluation of label in TEMP;
```

Control is transferred to the location pointed to by the evaluation of the local label on the TOS, if N = 0, otherwise by the evaluation of the local label at PL - N. The return address is then pushed onto the stack. Labels are evaluated identically to the PCAL instruction, except only local labels are allowed.

Indicators = Unaffected

137. EXIT N <Procedure and interrupt exit > Sub OpCode3=03

```
if STATUS(8:15) = 0 then
begin <<in an external interrupt routine>>
   DN := (Q + 1); <<Q+1 contains device # that interrupted>>
   Reset active state of device DN;
end;
if External Interrupt Request then
begin <<another external interrupt waiting>>
   S := Q + 1;
   (S) := Device # of new External Interrupt;
   Remainder of normal External Interrupt CALL; <<see
      Section V-A>>
end else
begin
   if (Q) = 0 and (STATUS(8:15)) = 0 then
   begin <<bottom of interrupt stack>>
      Q := S := old S stored in current PCB;
      Generate Internal Interrupt to Segment # 8 except
         do not create a new stack marker;
   end else
   begin <<normal exit>>
      S := Q;
      Q := S - (S);
      S := S - 1;
      STATUS := (S);  Note that user mode may not EXIT to
                      privileged mode and may not change the
                      interrupt disable bit, I.
      S := S - 1;
      PB, PL := evaluation of the segment descriptor
                specified by STATUS(8:15);
                if STATUS(8:15) = 0 then PB := 0,
                PL := 2**16 - 1;
      P := PB + (S);
      S := S - N - 1; <<Note that delta Q = 0 and
                      segment # ¬= 0 causes an
                      integrity trap because S < Q.>>
   end;
end;
```

This instruction is used to return from routines called by the PCAL instruction and by an Interrupt CALL.

If an External Interrupt routine is being exited, then the active state of the device's interrupt is reset.

If an External Interrupt is waiting then its routine is CALLed without removing the old stack marker or creating a new one.

If an External Interrupt is being exited and there are no pending interrupts, then a check is made to see if this is the bottom interrupt on the interrupt stack. If it is, then move Q and S to point to the users stack given in the Current Process Control Block and cause an Internal Interrupt to Segment # 8.

Otherwise, a normal exit occurs by setting Q to Q -Delta Q, STATUS to the previous status and P to the previous P. (See Section III-G for the stack marker format.)

138. SXIT N <Subroutine exit> Sub OpCode3=04

```
P := PB + (S);
S := S - N - 1;
```

This instruction is used to return from subroutines specified by the SCAL instruction.
Indicators = Unaffected

139. HALT K <Halt> Spec. OpCode=17

This is a privileged instruction. The machine halts. This instruction is non-interruptable and manual intervention is required to restart the machine. The K field is ignored.
Indicators = Unaffected.

140. PAUS K <Pause> Spec. OpCode=01

This is a privileged instruction. The machine pauses. This instruction is interruptable. The K field is ignored.
Indicators = Unaffected.

141. XEQ K <Execute> Spec. OpCode=06

```
CIR := (S - K);
```

The content of the word in the stack at S - K is placed in the current instruction register to be executed.

Control is returned to the instruction after the XEQ unless a transfer of control was executed.
Indicators = Set by the executed instruction or StackOps.

BB. MOVE INSTRUCTIONS

```
Memory OpCode = 03
Sub OpCode3 = 07
Move OpCode = CIR(8:10)
SDEC = CIR(11:12)
CCF = CIR(13:15)
      CIR(13) = Numeric
      CIR(14) = Alphabetic
      CIR(15) = Special
```

142. MOVE <Move words> Move OpCode=00

```
while X ¬= 0 do
   begin
     ((S-1)) := ((S));
     (S-1) := (S-1) + sign(X);
     (S) := (S) + sign(X);
     X := X - sign(X);
   end;
   S := S - SDEC;
```

This instruction expects a signed word count in the index register, a source word address on the TOS and a destination word address in the second word of the stack. If the word count is positive the words are moved in a left to right manner from the source area to the destination area. A negative word count will cause a right to left move form the source area to the destination area.
S is decremented by the amount indicated in the SDEC field.
Indicators = Unaffected

143. MVB <Move bytes> Move OpCode=01

```
while X ¬= 0 do
   begin
     ((S-1)) := ((S));
     (S-1) := (S-1) + sign(X);
     (S) := (S) + sign(X);
     X := X - sign(X);
   end;
   S := S - SDEC;
```

This instruction expects a signed byte count in the index register, a source byte address on the TOS, and a destination byte address in the second word of the stack.
S is decremented by the amount indicated in the SDEC field.
Indicators = Unaffected

144. MVBW <Move bytes while> Move OpCode=04

```
while CCB on ((S)) = CCF do
   begin
     ((S-1)) := ((S));
     (S-1) := (S-1) + 1;
     (S) := (S) + 1;
   end;
   S := S - SDEC;
```

This instruction expects the TOS to contain a source byte address, the second word of the stack to contain a destination byte address. As long as the byte in the source string is one of the types specified by the CCF, it is moved. The byte count moved is kept in the TOS. S is decremented by the amount indicated in the SDEC field.
Indicators = CCB on last character scanned

145. SCW <Scan while> Move OpCode=02

```
while ((S-1)) = (S(8:15)) do (S-1) := (S-1) + 1;
if ((S-1)) = (S(0:7)) then Carry := 1 else
   Carry := 0;
CC := CCB;
S := S - SDEC;
```

This instruction expects the TOS to contain a terminal character in the left byte and a test character in the right byte. The second word of the stack contains a source byte address. Bytes are scanned until the source string presents a character different from the test character.
If the terminating character is the same as the left byte of the TOS, the carry bit is set.
S is decremented by the amount indicated in the SDEC field.
Indicators = CCB on last character scanned, Carry

146. SCU <Scan until> Move OpCode=03

```
while ((S-1)) ¬= (S(8:15)) and ((S-1)) ¬= (S(0:7))do
   (S-1) := (S-1) + 1;
Carry := if ((S-1)) = (S(8:15)) then 1 else 0;
S := S - SDEC;
```

This instruction expects the TOS to contain a terminal character in the left byte and a test character in the right byte. The second word of the stack contains a source byte address. Bytes are scanned until either the terminal or test character is encountered. The address of this character is found in the second word of the stack. S is decremented by the amount indicated in the SDEC field.
If the last character scanned = the terminal character then Carry = 1 else Carry = 0.
Indicators = Carry

147. CMPB <Compare bytes> Move OpCode=05

```
while ((S)) = ((S-1)) and X > 0 do
   begin
     (S-1) := (S-1) + 1;
     (S) := (S) + 1;
     X := X - 1;
   end;
CC := if X = 0 then CCE else if ((S-1)) > ((S)) then
       CCG else CCL.
S := S - SDEC;
```

This instruction expects a byte count in the index register, a source byte address in the TOS and a target byte address in the second word of the stack. As long as the bytes in the source string compare with the tartet string the count in the index register is decremented. The instruction terminates when either a comparison fails or the byte count in the index register reaches zero. S is decremented by the amount indicated in the SDEC field.
Indicators = if X = 0 then CCE else if the final target byte = the last source byte scanned then CCG else CCL

CC. I/O & INTERRUPT INSTRUCTIONS

For the I/O instructions assume the following definitions:

(S-K(8:15)) contains a device number.
DRT is the base address of a Device Reference Table.
DRTE is the address of a particular DRT entry.
K is a four bit displacement 0 <= K <= 15
All I/O and Interrupt control instructions are PRIVILEGED.

148. SIO K <Start I/O> Spec. OpCode=07

```
Device # := (S-K(8:15));
DRTE := DRT + 3 * (S-K(8:15));
(DRTE) is passed to device controller
(DRTE) := (DRTE) + 2;
if I/O error then
begin
   S := S + 1;
   (S) := I/O INSTRUCTION STATUS;
end;
```

The address of the I/O program is passed to the device specified in the right byte of the word in the stack at S - K. The I/O instruction status is pushed on the stack if there is an I/O error. This is a privileged instruction.
Indicators = if error then CCL or CCG else CCE

149. RIO K <Read I/O> Spec. OpCode=10

```
Device # := (S-K(8:15));
S := S + 1;
(S) := 16 BIT DATA WORD FROM DEVICE CONTROLLER;
if I/O error then
begin
   S := S + 1;
   (S) := I/O INSTRUCTION STATUS;
end;
```

The 16 bit direct data word from the device specified is pushed onto the stack. The I/O instruction status is pushed onto the stack. This is a privileged instruction.
Indicators = if error then CCL or CCG else CCE

150. WIO K <Write I/O> Spec. OpCode=11

```
Device # := (S-K(8:15));
16 BIT DIRECT DATA WORD TO DEVICE CONTROLLER := (S);
if I/O error then
begin
   S := S + 1;
   (S) := I/O INSTRUCTION STATUS;
end
else S := S - 1;
```

Assume TOS contains the data word. The 16 bit data word is obtained from the stack and transmitted to the output device specified by the right byte of the content of S - K. If no error is detected, the data word is deleted; otherwise, the I/O instruction status is pushed onto the stack. This is a privileged instruction.
Indicators = if error then CCL or CCG else CCE

151. TIO K <Test I/O> Spec. OpCode=12

```
Device # := (S-K(8:15));
S := S + 1;
(S) := 16 BIT DEVICE STATUS;
```

The 16 bit device status word is pushed onto the stack. Note that TIO does not reset the device status bits. This is a privileged instruction.
Indicators = if error then CCL or CCG else CCE

152. CIO K <Control I/O> Spec. OpCode=13

```
Device # := (S-K(8:15));
16 BIT DIRECT CONTROL WORD TO DEVICE CONTROLLER := (S);
if I/O error then
begin
   S := S + 1;
   (S) := I/O INSTRUCTION STATUS;
end
else S := S - 1;
```

Assume TOS contains the control word. The 16 bit control word is obtained from the stack and transmitted to the device specified by the right byte of the content of S - K. If no error is detected, the control word is deleted; otherwise, the I/O instruction status is pushed onto the stack. This is a privileged instruction.
Indicators = if error then CCL or CCG else CCE

153. SED K <Set enable/disable external interrupts> Spec. OpCode=02

```
STATUS(1) := CIR(15);
```

The interrupt system is enabled or disabled corresponding to the least significant bit of the instruction. K = 1 implies enable, K = 0 disable. This is a privileged instruction.
Indicators = Unaffected

154. SIN K <Set interrupt> Spec. OpCode=16

Sets the interrupt flip flop in the device specified by the content of S - K. This is a privileged instruction.
Indicators = if error then CCL or CCG else CCE

155. SIRF K <Set external interrupt reference flag> Spec. OpCode=15

```
Device # := (S - K);
E := Device # * 4;
(E) := (E) or 2 ** 15;
```

The IRF bit in the Device Reference Table (see Section IV-F) corresponding to the device number in the stack at S - K is set to 1. S remains unchanged. This is a privileged instruction.
Indicators = Unaffected

156. CMD K <Command> Spec. OpCode=14

```
MCU Module # := (S-K(12:15));
Command := (S-K(8:11));
16 BIT DATA WORD := (S);
S := S - 1;
```

The 16 bit data word in the TOS and the 4 bit command in S-K(8:11) are sent over the MCU bus to the module given in S-K(12:15); the data word is deleted from the stack. This is a privileged instruction.
Indicators = Unaffected

DD. REGISTER CONTROL INSTRUCTIONS

157. PSHR N <Push registers> Sub OpCode2=11

```
if CIR(15) = 1 then PUSH(S-DB);
if CIR(14) = 1 then PUSH(Q-DB);
if CIR(13) = 1 then PUSH(Z-DB);
if CIR(12) = 1 then PUSH(DL-DB);
if CIR(11) = 1 then PUSH(DB);
if CIR(10) = 1 then PUSH(STATUS);
if CIR( 9) = 1 then PUSH(MASK);
if CIR( 8) = 1 then PUSH(X);
```

where...

```
PUSH(REGISTER) is defined to be
procedure(REGISTER);
begin
   S := S + 1;
   (S) := REGISTER;
end;
```

Note for pushing S-DB the following occurs:

```
S := S + 1;
(S) := S - DB;
```

The registers specified by CIR(8:15) are pushed onto the stack.
Indicators = Unaffected

158. SETR N <Set registers> Sub OpCode2=17

```
if CIR(8) = 1 then POP(X);
if CIR(9);= 1 and PRIV. MODE then POP(MASK);
if CIR(10) = 1 and PRIV. MODE then POP(STATUS);
if CIR(11) = 1 and PRIV. MODE then POP(DB);
if CIR(12) = 1 and PRIV. MODE then POP(DL + DB);
if CIR(13) = 1 and PRIV. MODE then POP(Z + DB);
if CIR(14) = 1 then POP(Q + DB);
if CIR(15) = 1 then POP(S + DB);

where...
   POP(REGISTER) is defined to be
   procedure(REGISTER);
   begin
      REGISTER := (S);
      S := S - 1;
   end;
```

The registers specified by CIR(8:15) are filled by an absolute value from the TOS for PB, MASK, STATUS, and DB, and an absolute value computed by adding DB to the TOS for the others. After each register is set, the TOS is deleted.
Indicators = Unaffected

159. ADDS N <Add to S> Sub OpCode3=05

```
if N > 0 then S := S + N;
if N = 0 then S := S - 1 + (S);
```

If the eight bit operand is zero, the content of the TOS is added to S - 1, otherwise the operand is added to S.
Indicators = Unaffected

160. SUBS N <Subtract from S> Sub OpCode3=06

```
S := S - N;
```

The eight bit immediate operand is subtracted from S.
Indicators = Unaffected

161. XCHD K <Exchange DB> Spec OpCode=03

```
TEMP := DB;
DB := (S);
(S) := TEMP;
```

This instruction expects a new DB value on the TOS. The current DB repalces that value on the TOS, while the new value is placed in DB. K is ignored.
This is a privileged instruction.
Indicators = Unaffected

EE. SPECIAL CONTROL INSTRUCTIONS

162. PLDA <Privileged load from absolute address> Mini OpCode=01

```
S := S + 1;
(S) := (X);
```

The content of the index register is a 16 bit absolute address; the content of this address is pushed onto the stack.
This is a privileged instruction.
Indicators = CCA

163. PSTA <Privileged store into absolute address> Mini OpCode=03

```
(X) := (S);
S := S - 1;
```

The content of the index register is a 16 bit absolute address; the top word of the stack is stored into memory at that address, and then deleted from the stack.
This is a privileged instruction.
Indicators = Unaffected

164. RSW <Read Switch register> Mini OpCode=00

```
S := S + 1;
(S) := Switch register;
```

The content of the Switch register is pushed onto the stack.
Indicators = CCA

FF. LIST SEARCH INSTRUCTION

165. LLSH <Linked list search> Mini OpCode=02

```
while  X > 0 and ((S)) < (S - 1) do
begin
   (S) := ((S) + (S - 2));
   X := X - 1;
end;
```

The TOS contains an absolute pointer into a linked list. This pointer references a target number which is compared to a source number in the second word of the stack. If the count in the index register is zero, or if the target number is logically greater or equal to the source number, the instruction terminates. (If the target number is all ones, the instruction terminates.) Otherwise another pointer is expected a distance delta away from the target number. This delta is contained in the third word of the stack. The pointer referenced replaces the TOS, the count is decremented, and the instruction repeats again.

Indicators = CCL if terminate on X = 0
CCE if terminate on ((S)) = (S - 1)
CCG if terminate on ((S)) > (S - 1)

GG. UNASSIGNED INSTRUCTION COMBINATIONS

Stack OpCode= 72
Sub OpCode1= 17
with 6 bit L parameter, optional indexing or indirect addressing with the Y bit
Sub OpCode3= 12,13
each with 8 bit N parameter
Spec. OpCode= 04,05
each with 4 bit K parameter

APPENDIX A - ALPHABETICAL LISTING OF INSTRUCTIONS

APPENDIX B - NUMERICAL LISTING OF INSTRUCTIONS

1.	LOAD	E	(*, D, X, PDQS) <Load>	Mem. OpCode=04
2.	LDB	E	(*, D, X, DQS) <Load byte>	Mem. OpCode=15
3.	LDD	E	(*, D, X, DQS) <Load double>	Mem. OpCode=15
4.	LDPP	N	<Load double from program, positive>	Sub OpCode3=10
5.	LDPN	N	<Load double from program, negative>	Sub OpCode3=11
6.	LDX	E	(*, D, X, PDQS) <Load Index>	Mem. OpCode=13
7.	LRA	E	(*, D, X, PDQS) <Load relative address>	Mem. OpCode=17
8.	STOR	E	(*, D, X, DQS) <Store>	Mem. OpCode=05
9.	STB	E	(*, D, X, DQS) <Store byte>	Mem. OpCode=16
10.	STD	E	(*, D, X, DQS) <Store double>	Mem. OpCode=16
11.	CMPM	E	(*, D, X, PDQS) <Compare memory>	Mem. OpCode=06
12.	ADDM	E	(*, D, X, PDQS) <Add memory>	Mem. OpCode=07
13.	SUBM	E	(*, D, X, PDQS) <Subtract memory>	Mem. OpCode=10
14.	MPYM	E	(*, D, X, PDQS) <Multiply memory>	Mem. OpCode=11
15.	INCM	E	(*, D, X, DQS) <Increment memory>	Mem. OpCode=12
16.	DECM	E	(*, D, X, DQS) <Decrement memory>	Mem. OpCode=12
17.	BR	E	(*, D, X, P or indirect DQS) <Branch>	Mem. OpCode=14
18.	BCC	E	(*, L, P) <Branch on Condition Code>	Mem. OpCode=14
19.	BCY	L	(*, L, P) <Branch on carry>	Sub OpCode1=14
20.	BNCY	L	(*, L, P) <Branch on no carry>	Sub OpCode1=15
21.	BOV	L	(*, L, P) <Branch on overflow>	Sub OpCode1=30
22.	BNOV	L	(*, L, P) <Branch on no overflow>	Sub OpCode1=31
23.	BRO	L	(*, L, P) <Branch on TOS odd>	Sub OpCode1=36
24.	BRE	L	(*, L, P) <Branch on TOS even>	Sub OpCode1=37
25.	IABZ	L	(*, L, P) <Increment A branch if zero>	Sub OpCode1=07
26.	IXBZ	L	(*, L, P) <Increment X, branch if zero>	Sub OpCode1=12
27.	DABZ	L	(*, L, P) <Decrement A, branch if zero>	Sub OpCode1=27
28.	DXBZ	L	(*, L, P) <Decrement X, branch if zero>	Sub OpCode1=13
29.	CPRB	L	<Compare range and branch>	Sub OpCode1=26
30.	TBA	E	(D, P) <Test, branch, A>	Mem. OpCode=05
31.	MTBA	E	(D, P) <Modify, Test, Branch, A>	Mem. OpCode=05
32.	TBX	E	(D, P) <Test, branch, X>	Mem. OpCode=05
33.	MTBX	E	(D, P) <Modify, Test, Branch, X>	Mem. OpCode=05
34.	CMP	<Compare>		Stack OpCode=17
35.	ADD	<Add>		Stack OpCode=20
36.	SUB	<Subtract>		Stack OpCode=21
37.	MPY	<Multiply>		Stack OpCode=22
38.	DIV	<Divide>		Stack OpCode=23
39.	NEG	<Negate>		Stack OpCode=24
40.	LCMP	<Logical compare>		Stack OpCode=57
41.	LADD	<Logical add>		Stack OpCode=60
42.	LSUB	<Logical subtract>		Stack OpCode=61
43.	LMPY	<Logical multiply>		Stack OpCode=62
44.	LDIV	<Logical divide>		Stack OpCode=63
45.	NOT	<One's complement>		Stack OpCode=64
46.	DCMP	<Double compare>		Stack OpCode=10
47.	DADD	<Double add>		Stack OpCode=11
48.	DSUB	<Double subtract>		Stack OpCode=12

49.	MPYL	<Multiply Long>	Stack OpCode=13
50.	DIVL	<Divide Long>	Stack OpCode=14
51.	DNEG	<Double negate>	Stack OpCode=15
52.	FCMP	<Floating compare>	Stack OpCode=50
53.	FADD	<Floating add>	Stack OpCode=51
54.	FSUB	<Floating subtract>	Stack OpCode=52
55.	FMPY	<Floating multiply>	Stack OpCode=53
56.	FDIV	<Floating divide>	Stack OpCode=54
57.	FNEG	<Floating negate>	Stack OpCode=55
58.	FLT	<Float>	Stack OpCode=47
59.	DFLT	<Double float>	Stack OpCode=30
60.	FIXT	<Fix and truncate>	Stack OpCode=71
61.	FIXR	<Fix and round>	Stack OpCode=70
62.	OR	<Or, logical>	Stack OpCode=65
63.	XOR	<Exclusive or, logical>	Stack OpCode=66
64.	AND	<And, logical>	Stack OpCode=67
65.	TEST	<Test TOS>	Stack OpCode=25
66.	DTST	<Test double word on TOS>	Stack OpCode=27
67.	BTST	<Test byte on TOS>	Stack OpCode=31
68.	INCA	<Increment A>	Stack OpCode=33
69.	INCB	<Increment B>	Stack OpCode=73
70.	INCX	<Increment index>	Stack OpCode=04
71.	DECA	<Decrement A>	Stack OpCode=34
72.	DECB	<Decrement B>	Stack OpCode=74
73.	DECX	<Decrement X>	Stack OpCode=05
74.	ZERO	<Push zero>	Stack OpCode=06
75.	DZRO	<Double push zero>	Stack OpCode=07
76.	ZROB	<Zero B>	Stack OpCode=41
77.	ZROX	<Zero X>	Stack OpCode=03
78.	DEL	<Delete A>	Stack OpCode=40
79.	DDEL	<Double delete>	Stack OpCode=02
80.	DELB	<Delete B>	Stack OpCode=01
81.	DUP	<Duplicate A>	Stack OpCode=45
82.	DDUP	<Double duplicate>	Stack OpCode=46
83.	XCH	<Exchange A and B>	Stack OpCode=32
84.	DXCH	<Double exchange>	Stack OpCode=16
85.	XAX	<Exchange A and X>	Stack OpCode=35
86.	XBX	<Exchange B and X>	Stack OpCode=75
87.	CAB	<Rotate ABC>	Stack OpCode=56
88.	LDXA	<Load X onto stack>	Stack OpCode=44
89.	LDXB	<Load X into B>	Stack OpCode=42
90.	STAX	<Store A into X>	Stack OpCode=43
91.	STBX	<Store B into X>	Stack OpCode=26
92.	ADAX	<Add A to X>	Stack OpCode=36
93.	ADBX	<Add B to X>	Stack OpCode=76
94.	ADXA	<Add X to A>	Stack OpCode=37
95.	ADXB	<Add X to B>	Stack OpCode=77
96.	NOP	<No operation>	Stack OpCode=00
97.	ASL	L <Arithmetic shift left>	Sub OpCode1=00
98.	ASR	L <Arithmetic shift right>	Sub OpCode1=01
99.	LSL	L <Logical shift left>	Sub OpCode1=02
100.	LSR	L <Logical shift right>	Sub OpCode1=03
101.	CSL	L <Circular shift left>	Sub OpCode1=04

```
102. CSR    L  <Circular shift right>                                  Sub OpCode1=05
103. SCAN      <Scan bits>                                             Sub OpCode1=06
104. DASL   L  <Double arithmetic shift left>                         Sub OpCode1=20
105. DASR   L  <Double arithmetic shift right>                        Sub OpCode1=21
106. DLSL   L  <Double logical shift left>                            Sub OpCode1=22
107. DLSR   L  <Double logical shift right>                           Sub OpCode1=23
108. DCSL   L  <Double circular shift left>                           Sub OpCode1=24
109. DCSR   L  <Double circular shift right>                          Sub OpCode1=25
110. TASL   L  <Triple arithmetic shift left>                         Sub OpCode1=10
111. TASR   L  <Triple arithmetic shift right>                        Sub OpCode1=11
112. TNSL      <Triple normalizing shift left>                        Sub OpCode1=16
113. TRBM   N  <Test and reset bit in memory>                         Sub OpCode3=14
114. TBC    N  <Test bit and set condition code>                       Sub OpCode1=32
115. TRBC   N  <Test and reset bit, set condition code>               Sub OpCode1=33
116. TSBC   N  <Test, set bit, set condition code>                    Sub OpCode1=34
117. TCBC   N  <Test and complement bit and set CC>                   Sub OpCode1=35
118. EXF  J,K    <Extract field>                                      Sub OpCode2=15
119. DPF  J,K    <Deposit field>                                      Sub OpCode2=16
120. LDI   =N  <Load immediate>                                       Sub OpCode2=02
121. LDNI  =N  <Load negative immediate>                              Sub OpCode2=12
122. CMPI  =N  <Compare immediate>                                    Sub OpCode2=04
123. CMPN  =N  <Compare negative immediate>                           Sub OpCode2=14
124. ADDI  =N  <Add immediate>                                        Sub OpCode2=05
125. SUBI  =N  <Subtract immediate>                                   Sub OpCode2=06
126. MPYI  =N  <Multiply immediate>                                   Sub OpCode2=07
127. DIVI  =N  <Divide immediate>                                     Sub OpCode2=10
128. ORI   =N  <Logical OR immediate>                                 Sub OpCode3=15
129. XORI  =N  <Logical Exclusive OR immediate>                       Sub OpCode3=16
130. ANDI  =N  <Logical AND immediate>                                Sub OpCode3=17
131. LDXI  =N  <Load X immediate>                                     Sub OpCode2=03
132. LDXN  =N  <Load X negative immediate>                            Sub OpCode2=13
133. ADXI  =N  <Add immediate to X>                                   Sub OpCode2=00
134. SBXI  =N  <Subtract immediate from X>                            Sub OpCode2=01
135. PCAL   N  <Procedure call>                                       Sub OpCode3=02
136. SCAL   N  <Subroutine Call >                                     Sub OpCode3=01
137. EXIT   N  <Procedure and interrupt exit >                        Sub OpCode3=03
138. SXIT   N  <Subroutine exit>                                      Sub OpCode3=04
139. HALT   K  <Halt>                                                 Spec. OpCode=17
140. PAUS   K  <Pause>                                                Spec. OpCode=01
141. XEQ    K  <Execute>                                              Spec. OpCode=06
142. MOVE   <Move words>                                              Move OpCode=00
143. MVB    <Move bytes>                                              Move OpCode=01
144. MVBW   <Move bytes while>                                        Move OpCode=04
145. SCW    <Scan while>                                              Move OpCode=02
146. SCU    <Scan until>                                              Move OpCode=03
147. CMPB   <Compare bytes>                                           Move OpCode=05
148. SIO K    <Start I/O>                                             Spec. OpCode=07
149. RIO K    <Read I/O>                                              Spec. OpCode=10
150. WIO K    <Write I/O>                                             Spec. OpCode=11
151. TIO K    <Test I/O>                                              Spec. OpCode=12
152. CIO K    <Control I/O>                                           Spec. OpCode=13
153. SED K    <Set enable/disable external interrupts>                Spec. OpCode=02
154. SIN K    <Set interrupt>                                         Spec. OpCode=16
```

155.	SIRF	K	<Set external interrupt reference flag> — Spec. OpCode=15
156.	CMD	K	<Command> — Spec. OpCode=14
157.	PSHR	N	<Push registers> — Sub OpCode2=11
158.	SETR	N	<Set registers> — Sub OpCode2=17
159.	ADDS	N	<Add to S> — Sub OpCode3=05
160.	SUBS	N	<Subtract from S> — Sub OpCode3=06
161.	XCHD	K	<Exchange DB> — Spec OpCode=03
162.	PLDA		<Privileged load from absolute address> — Mini OpCode=01
163.	PSTA		<Privileged store into absolute address> — Mini OpCode=03
164.	RSW		<Read Switch register> — Mini OpCode=00
165.	LLSH		<Linked list search> — Mini OpCode=02

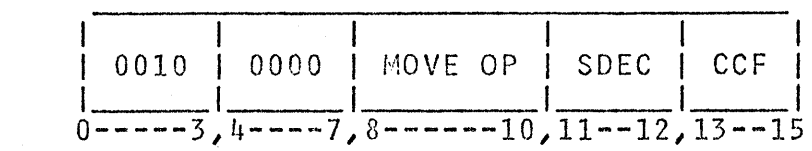
0	1 2 3	4	5 6 7	8 9	10	11 12 13	14 15
00	STACK OPCODES ALL 64 STACK OPS MAY BE USED IN EITHER POSITION (STACK OP A OR B)		00	NOP		40	DEL
			01	DELB		41	ZROB
			02	DDEL		42	LDXB
			03	ZROX		43	STAX
			04	INCX		44	LDXA
			05	DECX		45	DUP
			06	ZERO		46	DDUP
			07	DZRO		47	FLT
			10	DCMP		50	FCMP
			11	DADD		51	FADD
			12	DSUB		52	FSUB
			13	MPYL		53	FMPY
			14	DIVL		54	FDIV
			15	DNEG		55	FNEG
			16	DXCH		56	CAB
			17	CMP		57	LCMP
			20	ADD		60	LADD
			21	SUB		61	LSUB
			22	MPY		62	LMPY
			23	DIV		63	LDIV
			24	NEG		64	NOT
			25	TEST		65	OR
			26	STBX		66	XOR
			27	DTST		67	AND
			30	DFLT		70	FIXR
			31	BTST		71	FIXT
			32	XCH		72	SPARE
			33	INCA		73	INCB
			34	DECA		74	DECB
			35	XAX		75	XBX
			36	ADAX		76	ADBX
			37	ADXA		77	ADXB
01	SUB OPCODE 1	X	00	ASL		SHIFT COUNT L	
		X	01	ASR		" " "	
		X	02	LSL		" " "	
		X	03	LSR		" " "	
		X	04	CSL		" " "	
		X	05	CSR		" " "	
		X	06	SCAN			
		I	07	IABZ	+/-	P RELATIVE DISPLACEMENT	
		X	10	TASL		SHIFT COUNT L	
		X	11	TASR		" " "	
		I	12	IXBZ	+/-	P RELATIVE DISPLACEMENT	
		I	13	DXBZ	+/-	" " "	
		I	14	BCY	+/-	" " "	
		I	15	BNCY	+/-	" " "	
		X	16	TNSL		SHIFT COUNT L	
		X	17	SPARE			
		X	20	DASL		SHIFT COUNT L	
		X	21	DASR		" " "	
		X	22	DLSL		" " "	
		X	23	DLSR		" " "	
		X	24	DCSL		" " "	
		X	25	DCSR		" " "	
		I	26	CPRB	+/-	P RELATIVE DISPLACEMENT	
		I	27	DABZ	+/-	" " "	
		I	30	BOV	+/-	" " "	
		I	31	BNOV	+/-	" " "	
		X	32	TBC		BIT POSITION	
		X	33	TRBC		" "	
		X	34	TSBC		" "	
		X	35	TCBC		" "	
		I	36	BRO	+/-	P RELATIVE DISPLACEMENT	
		I	37	BRE	+/-	" " "	
02	SUB OPCODE 2	00	MOVE OPS	0	MOVE	SDEC	RESERVED
				1	MVB	"	RESERVED
				2	SCM	"	RESERVED
				3	SCU	"	RESERVED
				4	MVBW	"	CCG CCE CCL
				5	CMPB	"	RESERVED

0	1 2 3	4	5 6 7	8 9	10 11	12 13	14 15
02	SUB OPCODE 2	00	MISC OPS	14	RCW		RESERVED
				15	PLDA		RESERVED
				16	LLSH		RESERVED
				17	PSTA		RESERVED
		01	SPARE				
		02	LDI		IMMEDIATE OPERAND N		
		03	LDXI		"	" "	
		04	CMPI		"	" "	
		05	ADDI		"	" "	
		06	SUBI		"	" "	
		07	MPYI		"	" "	
		10	DIVI		"	" "	
		11	PSHR	X	MASK	STA DB DL	Z O S
		12	LDNI		IMMEDIATE OPERAND N		
		13	LDXN		"	" "	
		14	CMPN		"	" "	
		15	EXE	BEGINNING BIT #		# OF BITS	
		16	DPE	" " "		" " "	
		17	SETR	X	MASK	STA DB DL	Z O S
03	SUB OPCODE 3	00	SPECIAL OP	00	SPARE		
				01	PAUS		K FIELD
				02	SED		" "
				03	XCHD		" "
				04	SPARE		
				05	SPARE		
				06	XEQ		K FIELD
				07	SIO		" "
				10	RIO		" "
				11	WIO		" "
				12	TIO		" "
				13	CIO		" "
				14	CND		" "
				15	SIRF		" "
				16	SIN		" "
				17	HALT		" "
		01	SCAL		STT ENTRY # N		
		02	PCAL		" " " "		
		03	EXIT		N FIELD		
		04	SXIT		" "		
		05	ADDS		" "		
		06	SUBS		" "		
		07	SPARE				
		10	LDPP		P+ DISPLACEMENT N		
		11	LDPN		P- DISPLACEMENT N		
		12	ADXI		IMMEDIATE OPERAND N		
		13	SBXI		"	" "	
		14	TRBM		DB+ DISPLACEMENT N		
		15	ORI		IMMEDIATE OPERAND N		
		16	XORI		"	" "	
		17	ANDI		"	" "	
04	LOAD	X	I	PDOS ADDRESS MODE & DISPLACEMENT			
05	TBA	0	0	0 +/-	P RELATIVE DISPLACEMENT		
	MTBA	0	1	0 +/-	" " "		
	TBX	1	0	0 +/-	" " "		
	MTBX	1	1	0 +/-	" " "		
	STOB	X	I	1	DDS ADDRESS MODE & DISPLACEMENT		
06	CMPB	X	I	PDDS	" " "		
07	ADDB	X	I		" " "		
10	SUBB	X	I		" " "		
11	MPYB	X	I		" " "		
12	INCB	X	I	0	DQS		
	DECB	X	I	1	"		
13	LDX	X	I	PDOS	"		
14	BR	X	I	0 +/-	P RELATIVE DISPLACEMENT		
	BR	X	I	1 DQS	ADDRESS MODE (INDIRECT) & DISPLACEMENT		
	BCC	I	0	CCG CCE CCL +/-	P RELATIVE DISPLACEMENT		
15	LDB	X	I	0	DQS ADDRESS MODE & DISPLACEMENT		
	LDD	X	I	1	" " "		
16	STB	X	I	0	" " "		
	STD	X	I	1	" " "		
17	LRA	X	I	PDQS	" " "		

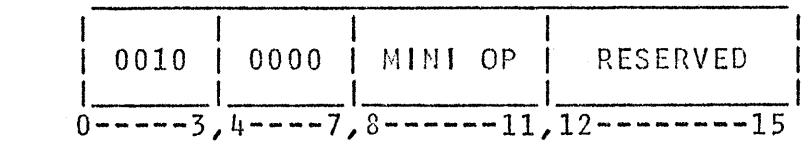
ERRATA # 1 7-26-70 PAGE 1 of 2

Page 14

Move opcodes should be



Mini opcodes should be



~~Page 32~~

~~In LDB the line beginning BE := (base + D) + if should read~~

~~BE := (base + D)*2+ if CIR(4) = 0 then 0 else X <<byte>>;~~

~~Page 34~~

~~In STB the line beginning BE := (base + D) + if should read~~

~~BE := (base + D)*2+ if CIR(4) = 0 then 0 else X <<byte>>;~~

Page 38, 79

29. CPRB should read

29. CPRB L (*, L, P) <Compare range and branch> Sub OpCode1=26

Page 62, 81

133. ADXI should read

133. ADXI =N <Add immediate to X> Sub OpCode3=12

134. SBXI should read

134. SBXI =N <Subtract immediate from X> Sub OpCode3=13

Page 66

The BB. section heading first 3 lines should read

BB. MOVE INSTRUCTIONS
Memory OpCode = 02
Sub OpCode2 = 00

Page 73, 82
162. PLDA should read

162. PLDA <Privileged load from absolute address> Mini OpCode=15

163. PSTA should read

163. PSTA <Privileged store into absolute address> Mini OpCode=17

164. RSW should read

164. RSW <Read Switch register> Mini OpCode=14

Page 74, 82
165. LLSH should read

165. LLSH <Linked list search> Mini OpCode=16

Page 74
Section GG should read

GG. UNASSIGNED INSTRUCTION COMBINATIONS

Stack OpCode= 72
Sub OpCode1= 17
with 6 bit L parameter, optional indexing
or indirect addressing with the Y bit
Sub OpCode2= 01
with 8 bit N parameter
Sub OpCode3= 07
with 8 bit N parameter
Spec. OpCode= 00,04,05
each with 4 bit K parameter

Page 75
29. CPRB should read

29. CPRB L (*, L, P) <Compare range and branch> PAGE 38

Page 22
The second line of section A. should read

for 255 device controllers. The I/O instructions are: Start I/O

Please report all other corrections to Bert Forbes.

ERATA TO ALPHA ERS

PAGE 5

Sec. B, 2nd paragraph, 1st line change "execpt" to "except"

PAGE 7

In the P definition, change "procram" to "program"

PAGE 8

I = Enable/Disable (1/0) external interrupt bit

T = Enable/Disable (1/0) user traps bit

CC = Condition Code

CCG = ØØ, CCE = 1Ø, CCL = Ø1

PAGE 9

Delete the first sentence of paragraph and substitute:

The address space of a user is divided into separate areas for program code and data. The program code is partitioned into logical code segments, one of which is active in core while the remaining segments are inactive and may reside in core or in secondary store. The data area and the currently executing code segment are directly addressable while the rest of the code segments are callable through a virtual memory structure using program labels. Each code segment has a maximum size of 16K words while the data area may be up to 65K words.

The active address space of a user (data area and executing code segment) are defined by hardware and are organized as follows:

.
.
.

Change DATA AREA in picture label to read DATA AREA (MAX 65K)

Change Code Segment label to read CODE SEGMENT (MAX 16K)

Change second SDT picture entry from PL to PB

PAGE 10

Address Computation definitions

Insert after S = ----

D = Displacement field of instruction word (See E-1 of instruction formats).

PAGE 11

Bounds check: Data, indirect

or Z + 14 < E <= DL

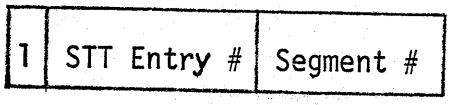
Bounds checking table	USER MODE	PRIV MODE
Code read	yes	no
Program transfers	yes	yes
- - - -		
Stack Overflow	yes	yes
Stack Underflow	yes	no

PAGE 17

ADD

H. Program Label Formats

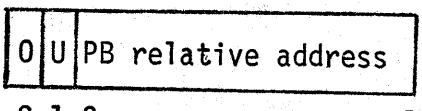
External Program Label



STT Entry # = Number to index from PL into the STT table to obtain entry point into the program segment.

Segment # = Logical entry number into the segment descriptor table (SDT).

Local Program Label

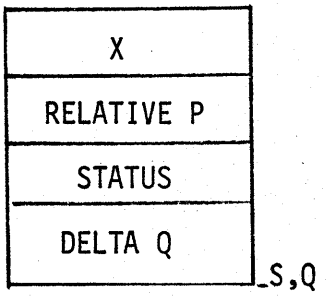


U = Uncallable bit. When set to 1 this entry point is uncallable from a non-privileged external segment. If so called, an interrupt to the system results.. This bit is ignored when referenced from within the local segment or when in privileged mode.

PAGE 17

Replace Section G with the following:

G. Stack Marker Formats



X = Contents of the Index Register

RELATIVE P = P+1 - PB
(The instruction to be returned to)

STATUS = The content of the Status register.

DELTA Q = Value to be subtracted from the Q register to obtain the Q value of the caller.

PAGE 18

A. Segment - - -

Third line . . . . . number of double word segment descriptor entries determined at system generation time. Each entry contains control information about the segment and gives the length and starting address of the segment. Segment # 0 is reserved for interrupt use, ...

PAGE 19

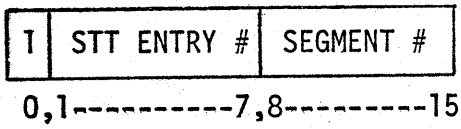
Change the definition of L to read:

L = for an N word code segment, L = N/4 and PL = PB + 4 * L - 1. Note: N is always an exact integer multiple of 4 and that the maximum L = 2^{12} - 1. The maximum segment size, therefore is 16,384 words.

Change in section STT

- - - - It contains 1 word entries of two basic types which are the program labels having the formats given in section III - H. References to external programs segments use the

External Program Label entry



STT ENTRY # = Number to index from PL to obtain entry point into segment.

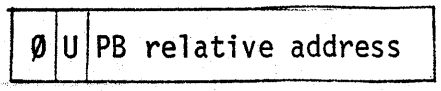
SEGMENT # = Logical entry number in SDT.

References from within the segment or calls through external labels to this segment use the

CHANGES CONTINUED ON PAGE 20 OF ERS

PAGE 20

Local Program Label entry



U = Uncallable bit prevents calls from unpriviledged external segments to this section of code.

PL Entry (The last word - - - -

PAGE 21

Under 03 ZI Add

04	Interrupt counter	
05	Unassigned	
06	Unassigned	
07	Unassigned	

DRT

10	I/O program dev. 2	(same)
11	PI dev. 2	
12	DBI dev. 2	
13	IRF dev. 2	
14	I/O prog. dev. 2	
15	PI dev. 3	
16	DBI dev. 3	
17	IRF dev. 3	
.		
.		
.		
1774	I/O prog. dev. 255	

etc.

PAGE 23

COUNT = Logical transfer count - - - may be bytes, words - - -

PAGE 25

Third paragraph change "tive" to "time"

PAGE 26

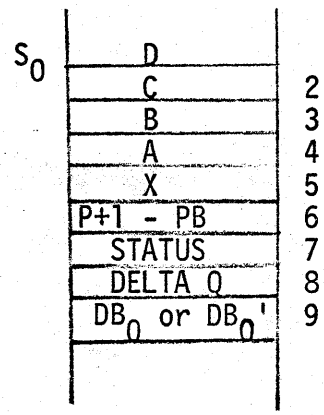
Change third paragraph to read:

- - - - is terminated, a standard 4 word stack marker is created on the current stack and then the current DB pointer is pushed onto the stack. If the interrupt occurred while operating in the user stack, the hardware top of stack registers are pushed into core memory such that S=SM and SR=Ø. The new value of S is stored into the Process Control Block (PCB) of the non-interrupt process. A hardware interrupt stack flag is set and the stack environment is changed to the Interrupt Stack by setting

Q := QI - - -
---- . The external interrupt system is left enabled.

PAGE 27

Change the user stack picture to:



PAGE 30

INTERRUPT AND TRAP TABLE

SEGMENT #	TYPE
Serviced on Interrupt Stack	
Ø	External Interrupts (via DRT)
1	Powerfail
2	Power On
3	Stack Overflow
4	Module interrupt
5	Console interrupt
6	Unassigned

CORRECTIONS TO PAGE 30 CONTINUED ON NEXT PAGE

SEGMENT #	TYPE
Serviced on Current Stack	
8	Unassigned
9	Unassigned
10	Parity Error (All three types)
11	Misc. Error

TYPE	SEGMENT #
Non-responding module	Ø
Illegal address	1
Stack underflow	2
SDT Bounds violation	3
STT Bounds violation	4
Module violation	5

12	Code Segment Absense
13	Traps

SUB SEGMENT #	TYPE
User traps controlled by trap enable bit in status	
Ø	Unassigned
1	Integer Overflow
2	Floating Point Over
3	Floating Pt. Underf
4	Integer divide by
5	Flt. Pt. divide by
System traps that are always active.	
6	Priviledged Inst.
7	Unimplemented Instr

PAGE 32

Change LDB instructions:

X contains a byte index. The byte referenced is loaded into the right half of the top of the stack. Note that byte indexing can cover only 32K of addresses. On indirect addressing the word referenced by the direct address (base + D) contains a DB relative byte address. The byte index is added to the relative byte address to obtain the effective byte address.

PAGE 35

In instruction 14, change the last line to read:
Indicators = CCA, Overflow.

PAGE 37

In instruction 23, add the line "S := S-1" after the If statement in the algol definition.

In instruction 24, add the line "S := S-1" after the If statement in the algol definition.

In instruction 25, change the last line to read:
Indicators = CCA, Carry, Overflow.

PAGE 38

In # 26, IXBZ, the first line of the English definition should read: The index register is logically incremented. . . .
The last line should read: Indicators = unaffected.

In # 27, DABZ, the last line should read: Indicators = CCA, Carry, Overflow.

In # 28, the first line of the English definition should read:
The index register is logically decremented. . . . .
The last line should read: Indicators = unaffected.

In # 29, CPRB, the algol definition should read:

```
...
  P := P + L;
end else
if X ....
```

PAGE 41

In the instruction MPY, #37, the algol defintion should read:

```
(S-1) := (S-1) * (S);
S := S - 1;
```

Also, delete the second paragraph of the English defintion and insert:

If the high order 17 bits (including the sign bit of the second word) are not all zeroes, or all ones, overflow is set.
Indicators = CCA, Overflow.

PAGE 43

In instruction #43, LMPY, change the word "most" to "least" in the third line of the English definition. It will then read:

. . . length product with the least significant . . . .

In instruction LDIV, #44, add to the English description the sentence: "If overflow occurs, the remainder will be modulo 2**16."

PAGE 45

In instruction #51, DNEG, the last line should read:

Indicators = CCA, Overflow.

PAGE 46

In instruction FDIV, #56, the algol definition should read:

```
(S-3,S-2) := (S-3,S-2)/(S-1,S)
S := S - 2
```

The English definition should read:

The floating point number contained in the 3rd and 4th words of the stack is divided by the floating point number contained in the 1st and 2nd words of the stack. . . . .

PAGE 48

Instruction #67, should appear as follows:

67. BTST Test double word on TOS

Indicators = CCB on (S(8:15)).

PAGE 49

The English definition of instruction #70, INCX, should read:
The content of the index register is incremented by one in logical form.
Indicators = CCA.

PAGE 51

The last line of the English definition in instruction # 81 should read:
Indicators = unaffected.

The last line of the English definition in instruction # 82 should read:
Indicators = unaffected.

PAGE 57

The fourth line of the Algol definition in instruction # 112, TNSL, should read:
while (S-2(6)) = Ø do

The second line of the English definition in the same instruction, TNSL, should read:
... until (S-2(6)) = 1. The shift count . . .

PAGE 58

Instruction # 113 now appears as follows:

TSBM N ⟨Test and set bits in memory⟩

```
TEMP := (DB+N) or (S);        set bits
(S) := (DB+N) and (S);        get bits
(DB+N) := TEMP;
```

All bit positions of the DB+ relative memory word that have a 1 in the corresponding position on the top of the stack are set; at the same time the memory word ANDed with the mask in the TOS replaces the TOS. Interrupts may not occur during the execution of this instruction.
Indicators = CCA on the new TOS.

In instructions #115, #116, and #117 the first three words of the English definition should read: This instruction's operation . .

PAGE 59

The following change should be made to instruction # 118, EXF:

Second line: ....TOS is deleted. The K bit field is .....

PAGE 61

The second line of the English definition in instruction #125, SUBI, should read: ...TOS in integer form, ...

PAGE 61 (CONT.)

In the DIVI instruction, #127, the last line should read:

Indicators = CCA on the new TOS.

PAGE 63

In instruction #135, PCAL, insert after line six of the Algol definition the following:

```
(S) := X;
S := S+1;
```

Change third line of the Enlish definition in the same instruction to read:

..... otherwise. Then a four word stack .....

In instruction SCAL, # 136, add then statement to the last line. ...allowed. Non local label gives illegal address trap.

PAGE 64

Replace the Algol definition of instruction #137, EXIT.

```
EXIT   N  ⟨Procedure and interrupt exit⟩

       TEMP := 4
       If in interrupt stack then   interrupt stack flag is set
       Begin   If STATUS(8:15) = Ø then
           Begin  ⟨⟨in an external interrupt routine⟩⟩
               DB := (Q+1); ⟨⟨Q+1 contains the DB of the interrupt routin
               DN := (Q+2); ⟨⟨Q+2 contains the device # that interrupted⟩⟩
               Reset active state of device DN;
           End;
           If External interrupt request then
           Begin ⟨⟨another external interrupt waiting⟩⟩
               S:=Q+1;
               (S) := Device # of new External Interrupt;
               Remainder of normal External interrupt CALL;
                ⟨⟨See section V-A⟩⟩
           End;
           If (Q) = Ø then
               Begin  Temp := -1;
                      Set Dispatches flag;
               End;
       End;
       Begin ⟨⟨normal exit⟩⟩
           TEMP := Q - TEMP;
           S := Q;
```

CONTINUED ON NEXT PAGE

CONTINUED FROM PAGE 10

```
Q := S - (S);
S := S - 1;
STATUS := (S);    Note that user mode may not EXIT to
                  privileged mode and may not change the
                  interrupt disable bit, I.
If STATUS(8:15) = Ø then
    Begin   PB := Ø;
            PL := 2**16-1;
    End
Else PB,PL := evaluation of segment descriptor specified
    By STATUS(8:15);
S := S-1;
P := PB + (S);
S := S-1:
X := (3);
S := TEMP - N     Note that in exits from interrupt routine
                  N must be Ø.
```

In the English definition of the EXIT instruction, delete the fourth paragraph.

PAGE 66

Change BB. MOVE INSTRUCTIONS to;

```
Memory Opcode = 02
Sub Opcode2 = 00
Move Opcode = CIR(8:10)
SDEC = CIR(14:15)

On some instructions optional base addresses are
      available for relative addressing where
      BASE := If CIR(11) = 1 then DB else PB;
```

Change instruction #142, MOVE to appear as follows:

```
while (S) ≠ Ø do
  begin
    (BASE + (S-2)) := (BASE) + (S-1);
    (S-2) := (S-2) + Sign(S);
    (S-1) := (S-1) + Sign(S);
    (S-1) := (S) - Sign(S);
  end;
S := S - SDEC;
```

MOVE INSTRUCTION CONTINUED ON NEXT PAGE

PAGE 66 (CONT.)

CONTINUATION of MOVE instruction(English definition):

This instruction expects a signed word count in the TOS, a source word relative address in the second word of the stack and a destination word relative address in the third word of the stack. If the word count is positive the words are moved in a lower to higher address sequence from the source area to the destination area. A negative word count will cause a higher to lower address move from the source area to the distination area.

The first six lines of the Algol definition of the MVB instruction should appear as follows:

```
While (S) ≠ Ø do
  Begin
  (BASE + (S-2)/2) := (BASE + (S-1)/2)
  (S-2) := (S-2) + Sign(S);
  (S-1) := (S-1) + Sign(S);
  (S) := (S) - Sign(S)
```

PAGE 67

The Algol definition of the MVBW instruction, #144, should read:

```
CCF = CIR(11:13)
      CIR(11) = Numeric
      CIR(12) = Alphabetic
      CIR(13) = Special
While CCB on (DB + (S)) = CCF do
  Begin
    (DB + (S-1)/2) := (DB + (S)/2);
    (S-1) := (S-1) + 1;
    (S) := (S) + 1;
  End;
```

The Algol definition of the SCW instruction, #145, should read:

```
While (DB + (S-1)/2) = (S(8:15)) do (S-1) := (S-1) + 1;
If (DB + (S-1)/2) = (S(0:7) then carry := 1 else
  Carry := Ø;
CC := CCB;
S := S - SDEC;
```

PAGE 67 (CONT.)

The Algol definition of the SCU instruction, #146, should read:

```
While (DB + (S-1)/2) ≠ (S(8:15)) and (DB + (S-1)/2) ≠ (S(0:7)) do
  (S-1) := (S-1) + 1;
Carry := if (DB + (S-1/2) = (S8:15)) then 1 else Ø;
S := S - SDEC;
```

PAGE 68

The Algol definition of instruction #147, CMPB, should appear:

```
While (BASE +(S-1)) = (BASE + (S-2)) and (S)   Ø do
  Begin
    (S-2) := (S-2) + 1;
    (S-1) := (S-1) + 1;
    (S) := (S)-1;
  End
CC := if (S) = 0 then CCE else if (BASE + (S-2)) > (Base + (S-1)
  then CCG else CCL.
S := S - SDEC;
```

The English defintion of the same instruction should appear:

This instruction expects a byte count in TOS, a service byte address in TOS-1, and a target byte address in the third word of the stack. As long as the bytes in the source string compare with the target string, TOS is decremented. The instruction terminates when either a comparison fails or the byte count in TOS reaches zero. S is decremented by the amount indirected in the SDEC field.
Indicators = if TOS = 0 then CCE else
if the final target byte the last source byte scanned
then CCG else CCL.

In the Sub-title CC. I/O Interrupt Instructions, after the last line add:

A non-responding device controller will terminate I/O instructions and set indicator to CCL.

Replace the second line of the Algol definition in instruction #148, SIO, with the following:

```
DRTE := DRT + 4 * (S-K(8:15));
```

The last line of the same instruction, SIO, should read:

```
Indicators = if I/O error then CCG else CCE
```

PAGE 69

In the RIO instruction, #149, change the fifth line of the English definition to read:

.... instruction status is pushed onto the stack if there is an error. This ....

The last line should read:

Indicators = if I/O error the CCG else CCE

In the WIO instruction, #150, the last line should read:

Indicators = if I/O error then CCG else CCE

In the TIO instruction, #151, the last line should read:

Indicators = if I/O error then CCG else CCE

PAGE 70

In the CIO instruction, #152, change the algol definition to read:

```
Device # := (S-K(8:15));
16 bit direct control word to device controller := (S);
S := S-1;
```

Change the last four lines of the English definition to read:

S - K. If no error is detected, the control word is deleted.
This is a privileged instruction.
Indicators = if non-responding module then CCL else CCE.

The second line of the algol definition in instruction #155, SIRF, should read:

```
E := Device # * 4 + 3;
(E(0)) := 0;
(Address 4) := (Address 4) + 1;
```

The third line of the English definition of the same instruction, #155, should be change to:

set to 0. S remains unchanged. This is a privileged instruction.

PAGE 71

Replace line 11 of the PSHR instruction, #157, with:

```
Procedure push (Register);
Value register; Integer register;
```

The 18th line of the same instruction should read:

```
(S) := S-DB-1;
```

PAGE 72

Line 3 of the SETR instruction should be replace with:

```
if CIR(10) = 1 and if PRIV. MODE then
               POP(STATUS(0:15)) else
               POP(STATUS(2,4:70
```

Replace the last five lines of the Algol definition with:

```
Procedure Pop(Register);
  Begin
    Register := (S);
    if Register  = "S"
        then S := S-1;
  End;
```

The second line of the English definition of the SETR instruction should read:

absolute value from the TOS for X, MASK, STATUS, and DB

PAGE 74

Change the first line of the Algol definition of the LLSH instruction, #165, to read:

while X > 0 and ((S)) < (S-1) do

Add to the end of the English definition of the same instruction:

```
This is a privileged instruction.
Indicators = CCL if terminate on X=0
             CCE if terminate on ((S)) >=  (S-1)
             CCG if terminate on ((S)) = 2**16 - 1
                    ≪all ones≫
```