PDP-1 COMPUTER
ELECTRICAL ENGINEERING DEPARTMENT
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
CAMBRIDGE, MASSACHUSETTS
02139

- SAVE -

PDP-35

INSTRUCTION MANUAL

PART 1 -- BASIC INSTRUCTIONS

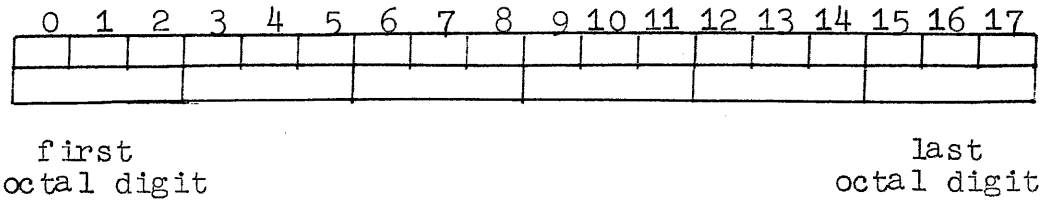
USE FOR "ONE" MOLE

21 March 1972

1. Basic Instructions

1.1 Word Formats

Each PDP-1 word is 18 bits long. The bits are numbered, in decimal, 0 to 17 from left to right. In this manual, all numbers are octal, unless otherwise specified. The numbering of bit positions in words is always decimal.



1.1.1 Number Formats

The entire word may be regarded as a signed 18-bit number. Bit 0 is the sign bit. It is on (i.e., a "1") if the number is negative. There are two different ways (called one's complement and two's complement) in which the PDP-1 hardware can interpret signed numbers. Positive numbers are the same in both representations. Positive numbers are represented in the ordinary binary notation. The range of positive numbers that can be represented is 0 to 377777 (131071 decimal).

In ones's complement, the negative of a number is formed by complementing all of its bits. Hence -1 is represented as 777776 in one's complement. 400000 (-131071 decimal) is the most negative number that can be represented in one's complement. Note that there is a representation for -0, 777777. Since -0 and +0 are similar in their arithmetic properties, this dual representation is an ambiguity which can be the source of difficulties.

In two's complement, the negative of a number is formed by complementing all bits of the number and then adding 1. Hence -1 is represented as 777777. 400000 (-131072 decimal) is the most negative number that can be represented in two's complement. Note that the negative of 0 is 777777+1 which is 0. In two's complement notation, 0 is represented uniquely.

Number (base 10)	One's Complement	Two's Complement
0	0	0
-0	777777	no representation
5	5	5
-5	777772	777773
131071	377777	377777
-131071	400000	400001
-131072	no representation	400000

By ignoring the sign convention, a program could deal with data words as unsigned numbers ranging between 0 and 262143 (decimal) or 0 and 777777 (octal).

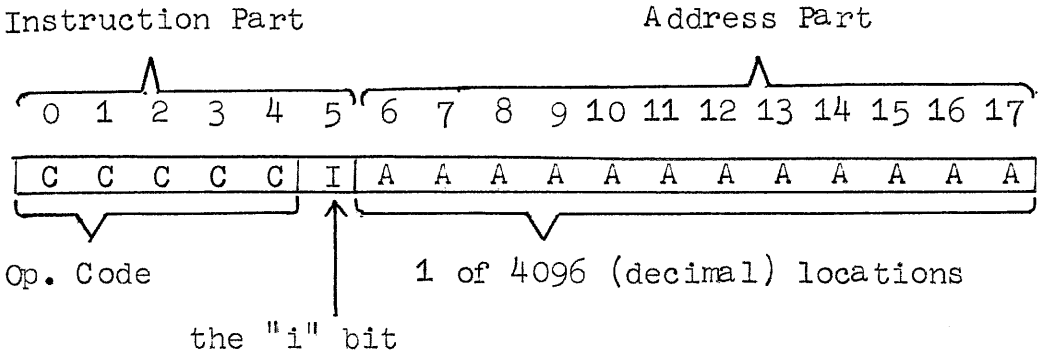
The one's complement addition rule for 2 18-bit numbers is as follows. Add the 2 numbers in the normal fashion, propagating carries to the left. If there is a carry out of the last bit (bit 0) add this carry in at the right hand side of the word (bit 17). The addition is said to overflow if the result does not correctly represent the algebraically correct sum, i.e., the magnitude of the correct result is greater than 377777.

The two's complement addition rule is similar. Add the 2 numbers in the normal fashion, propagating carries to the left. Ignore any carry out of bit 0. Overflow is said to occur if both operands are of the same sign and the sign of the result differs from the signs of the operands.

In both one's and two's complement, the overflow condition is equivalent to the condition that a carry occurred from bit 1 but not from bit 0 or vice versa.

Example	One's Complement	Two's Complement
7 + -100 -61	7 + 777677 777706	7 + 777700 777707
123456 + 666666 1012344	123456 + 666666 1012344 = 012345	123456 + 666666 1012344 = 012344

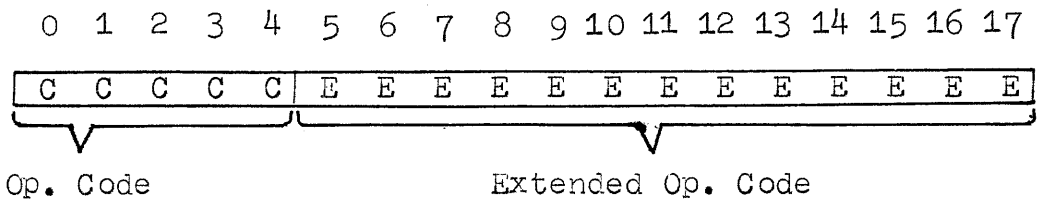
1.1.2 Addressable Instruction Format



The "instruction part" consists of 5 bits (the "op. code") that tell which instruction the computer will do if it executes this word, and the "i-bit" which has to do with address modification.

Except for jmp and jsp, addressable instructions take a minimum of two memory cycles -- one to fetch the actual instruction and another to fetch the operand. jmp and jsp do not take an operand and consequently, require a minimum of one cycle. The amount of time required for the execution of an addressable instruction depends on what addressing is done (see PDP-35, INSTRUCTION MANUAL, Part 2 for complete information). Under the most common conditions, the actual time required for instruction execution is the minimum time given above.

1.1.6 Non-addressable Instruction Format



The low 13 bits are actually an extension of the op. code, further specifying what the computer is to do. Non-addressable instructions never require more than one cycle because they do not reference memory, except for the ivk instruction with PRL on (See PDP-35, INSTRUCTION MANUAL, Part 5).

1.2 Registers

The PDP-1 contains 6 18-bit registers which define the state of the user's process. They are the A (accumulator), I (input/output register), X (index register), G, F (flag register), and W. When the user first logs in, all of these registers contain 0 (all bits off). This defines the initial state of the user's process. Each of these registers has its own properties.

1.2.1 Accumulator

The accumulator is the major register for use in processing data. Most arithmetic and logical instructions operate on data in A and leave results in A.

1.2.2 Input/Output Register

The input/output register was originally used primarily for input/output operations. This is no longer true. I is now a secondary accumulator. Many arithmetic and logical operations can be performed on data contained in I.

1.2.3 Index Register

The index register has two functions. First, it is used in addressing memory (See PDP-35, INSTRUCTION MANUAL, Part 2). Second, it is, like I, a secondary accumulator.

1.2.4 G Register

The G register contains the 15-bit program counter, the overflow bit, the extend mode bit, and the arithmetic mode bit.

The program counter (PC) is bits 3-17 of the G register. The purpose of the program counter is to tell the processor where in memory the instruction that is to be executed next lies. The program counter is incremented after each instruction so that instructions are executed sequentially, according to their locations in memory. When the program counter is incremented, carries out of bit 6 of G are lost. Thus, the instruction executed after the instruction in location 7777 is the instruction in location 0. Certain testing instructions increment the PC an extra time, causing an instruction to be skipped.

Bit 0 of G is the overflow bit (OVF). It is set to 1 by certain arithmetic instructions when overflow occurs, and cleared by the szo instruction. Bit 1 of G is the extend mode bit (EXD). This bit is used in addressing (See PDP-35, INSTRUCTION MANUAL, Part 2). Bit 2 of G is the arithmetic mode bit (TWOS). If this bit is off, the processor is in one's complement mode; if this bit it on, the processor is in two's complement mode.

The format of the G register is shown below.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
OVF	EXD	TWOS	PC3	PC4	PC5	PC6	PC7	PC8	PC9	PC10	PC11	PC12	PC13	PC14	PC15	PC16	PC17

1.2.5 Flag Register

The flag register contains the 6 program flags and several bits which define various states of the processor.

The program flags are 6 1-bit registers that may be quickly and conveniently set and tested by programs (see Operate and Skip class instructions). Also see part 3.2.6, the Light Pen

The 3 bits AMD, AEF, and AAL determine the mode of addressing which the processor will use on memory referencing instructions (see PDP-35, INSTRUCTION MANUAL, Part 2).

The 2 bits SBH (Sequence Break Hold) and SBM (Sequence Break Mode) determine the state of the sequence break system (see PDP-35, INSTRUCTION MANUAL, Part 3).

The ESI (Execute Single Instruction) bit causes the processor to trap after each instruction is executed (see PDP-35, INSTRUCTION MANUAL, Part 5).

The PRL (Program Reference List) bit affects the way in which the ivk (invoke) instruction works (see PDP-35, INSTRUCTION MANUAL, Part 5). When the PRL bit is 1, references to memory locations 0-77 are illegal.

The format of the F register is as follows:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
AMD	AEF	AAL	SBM	SBH	PRL	ESI						PF1	PF2	PF3	PF4	PF5	PF6

1.2.6 W Register

The W register, a software register maintained by the time-sharing supervisor, is used solely for communication with the supervisor. Certain mta and ivk instructions (both are supervisor calls) use the W register (see PDP-35, INSTRUCTION MANUAL, Part 5).

1.3 Instructions to Set the Arithmetic Mode

The PDP-1 processor may operate in either of two arithmetic modes, one's complement and two's complement. The arithmetic mode in which an instruction is executed is determined by the state of the TWOS bit in the G register. When the TWOS bit is off (contains a 0), the processor is in one's complement mode. One's complement mode (TWOS off) is the default mode. The following instructions change the state of TWOS.

Mnemonic	Op.Code	Name	Function
e2m	770060	Enter two's mode	Set TWOS to 1
e1m	770061	Enter one's mode	Set TWOS to 0

The instructions add, adm, sub, mul, div, idx, isp, sft, opr, and opr i (the micro-program instruction) behave differently in one's and two's mode. Address arithmetic is always done in the current arithmetic mode.

1.4 Addressable Instructions

In this section, the symbol "y" used in the context of "ins y" means the memory location referenced by the instruction "ins". The notation "(y)" means the contents of location y. For information on how addresses are computed, see PDP-35, INSTRUCTION MANUAL, Part 2.

1.4.1 Data Moving Instructions

These instructions serve to move data between memory locations and the A, I, and X registers. These instructions copy data words (or parts of words) from one place to the other and never destroy information at the source.

Mnemonic	Op.Code	Name	Function
lac y	20	load A	Copy (y) into A
lio y	22	load I	Copy (y) into I
lxr y	12	load X	Copy (y) into X
dac y	24	deposit A	Copy A into y
dio y	32	deposit I	Copy I into y
dap y	26	deposit address part of A	Copy the low 12 bits of A into y. The high 6 bits of y are unchanged
dip y	30	deposit instruction part (of A)	Copy the high 6 bits of A into y. The low 12 bits of y are unchanged
dzm y	34	deposit zero in memory	Makes location y contain 0

1.4.2 Logical Instructions

These instructions take one operand in A and the other from a memory location. The result is left in A. Each bit of the result depends only on the corresponding bits of A and the memory word before the operation.

Mnemonic	Op.Code	Name	Each bit of A will be a 1 if and only if the corresponding bits in A (before the instruction) and (y) were --
and y	02	and	both one
ior y	04	inclusive or	not both zero
xor y	06	exclusive or	different

1.4.3 Arithmetic Instructions

The following instructions are used to compute sums and differences. The "left" operand is in A and the "right" operand is taken from memory. The result is left in A and in the case of adm (add to memory), it replaces the contents of the memory location as well.

In one's mode, arithmetic is done in one's complement. If the result of an operation is -0, it is changed to +0 with the exception that (-0) - (+0) results in -0.

In two's mode, arithmetic is done in two's complement.

The overflow bit OVF will be set by the add, adm, or sub instructions if the signed result cannot be correctly represented in 18 bits. This is the case if and only if a carry occurred from bit one and no carry occurred from bit 0, or vice versa. See the szo instruction.

Mnemonic	Op.Code	Name	Function
add y	40	add	Sum of A and (y) to A
adm y	36	add to memory	Sum of A and (y) to A and y
sub y	42	subtract	A minus (y) to A

1.4.4 Multiply and Divide

Multiply and divide behave very differently in one's mode and two's mode. Hence the actions of these these instructions will be explained separately for each of the arithmetic modes.

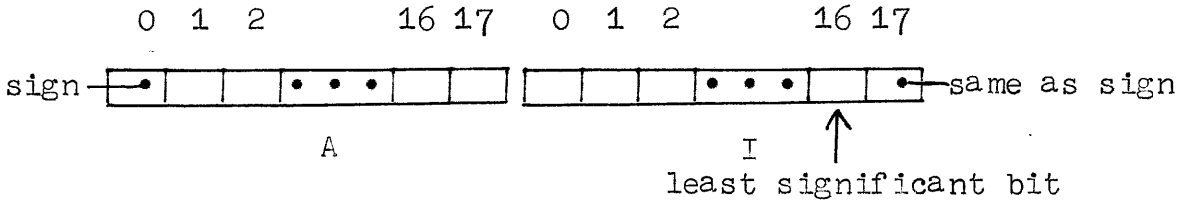
Mnemonic	Op.Code	Name	Function
mul y	54	multiply	A times (y) to A,I (A,I)
div y	56	divide	Quotient of (A)/(y) to A Remainder to I Skip if the quotient does not overflow

Multiply takes a minimum of two memory cycles plus from 3 to 15 microseconds, depending on the number of one's in A. Divide instructions which skip take two cycles plus 20 microseconds. In one's mode, divide instructions which do not skip take 2 cycles. In two's mode, divide instructions which restore A and I and do not skip take 2 cycles; other non-skipping divides take 2 cycles plus 20 microseconds. ???

1.4.4.1 Multiply (One's Complement Mode)

Multiply and divide deal with double-length numbers. mul may be viewed as multiplying two 17-bit integers plus signs to produce a 34 bit integer plus two signs. A result of -0 is changed to +0.

Result of Multiply (One's Mode)



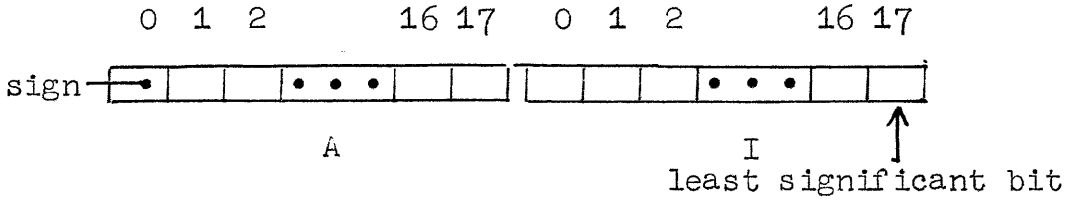
Quite often, two integers are multiplied such that the result can be held in one register. In this case, A will be +0 or -0, and the result will be in I, but it will appear to be shifted left one bit position. A rir 1s instruction will convert I back to normal single-length integer format.

Examples --

Before mul		After mul	
A	y	A	I
3	200000	1	400000
-3	200000	777776	377777
3	2	0	14
-0	anything	0	0

1.4.4.2 Multiply (Two's Complement Mode)

Multiply and divide deal with double-length numbers. mul may be viewed as multiplying two 17 bit integers plus signs to produce a 35 bit integer plus sign in the combined A and I registers.



When two integers are multiplied such that the result can be held in one register, the entire result will be in I in the conventional signed integer format.

Examples --

Before mul		After mul	
A	y	A	I
3	200000	0	600000
-3	200000	777777	200000
3	2	0	6
-3	2	777777	777772
400000	2	777777	0
400000	400000	200000	0

1.4.4.3 Divide (One's and Two's Mode)

Divide (div) takes a double-length integer in A and I (in the format produced by mul in the same arithmetic mode) and divides this by a single length integer in the addressed memory location. The result of a div is a single-length integer quotient in A, and a single-length integer remainder in I. The sign of the remainder will be the same as that of the dividend.

If a quotient overflow occurs, that is, if the divisor goes into the dividend more times than can be represented in the accumulator (A), the div instruction will not skip. In one's mode, a divide will skip if the absolute value of the original A is less than the absolute value of the divisor. In one's mode if a div does not skip, the original A and I are preserved, except that if A and I were both -0, A is changed to +0. In two's mode, if a div does not skip, the contents of the A and I registers are preserved if the divisor is 0, otherwise they are usually destroyed. div never sets the overflow bit.

Examples

Before divide			After divide			
			One's Mode		Two's Mode	
A	I	C(Y)	A	I	A	I
0	16	2	3	1	7	0
0	7	3	1	0	2	1
0	11	777773	-1	0	-1	4
0	25	777774	-3	1	-5	1
1	400000	200000	3	0	6	0
777776	377777	777774	200000	0	300000	-1
777777	0	400000	1	0	2	0
200000	0	400000	577777	200000	400000	0
6	777776	0	no skip, A and I unchanged		no skip, A and I unchanged	
100000	222222	100000	no skip, A and I unchanged		no skip, A and I destroyed	

1.4.5 Counting Instructions

The two instructions idx and isp add one (in the current arithmetic mode) to the contents of the specified memory location and leave the result both in memory and A. isp skips the next instruction if the result is positive, including 0. Neither instruction will set overflow. In one's mode, a -0 result is corrected to +0, i.e., indexing -1 results in +0, and indexing -0 results in +1.

Mnemonic	Op.Code	Name	Function
idx y	44	index	(y) + 1 to A and y
isp y	46	index and skip if result positive	(y) + 1 to A and y skip if result $\geq$ 0

1.4.6 Compare Instructions

The following two instructions are used to compare A with the contents of a memory loction. The comparison is done bit-by-bit, therefore, the contents of A are the same as the contents of memory if and only if every bit of A is the same as the corresponding bit in memory.

Mnemonic	Op.Code	Name and Function
sas y	52	skip if A is the same as (equal to) the contents of y
sad y	50	skip if A is different from (y)

1.4.7 Transfer of Control

All of the following instructions have the effect of changing the program counter (PC) so that the PDP-1 will begin executing instructions ~~out of the normal~~ in a new sequence.

Mnemonic	Op.Code	Name	Function
jmp y	60	jump	transfer control to location y
jdp y	14	jump and deposit program counter	store G in y, jump to y+1
jsp y	62	jump and save program counter	jump to y and save G in A
jda y	17	jump and deposit accumulator	store A in y, save G in A, jump to y+1 (dac y, jsp y+1)
cal y	16	call	store A in 00100, save G in A, jump to 00101, y ignored (similar to jda 100)

The instructions jdp, jsp, and jda are used chiefly for calling subroutines. The saved G register is the linkage mechanism which allows the subroutine to return to the place from which it was called.

Simple Examples

```
        jdp subr                jsp subr                jda subr
          ...                     ...                     ...

subr,   0               subr,   dap subx        subr,   0
          ...                     ...                     dap subx
          ...                     ...                       ...
        jmp i subr      subx,   jmp             subx,   jmp
```

These examples show three ways in which a subroutine may be called. In each example the method by which the subroutine returns to the calling program is illustrated. In each example the return is to the location immediately following the subroutine call.

1.4.8 Execute

The xct instruction causes the contents of the specified memory location to be executed as an instruction. xct's may execute other xct's. In all cases the effect is the same as if the xct were replaced by the instruction it addresses. xct takes a minimum of one cycle plus the time to do the addressed instruction.

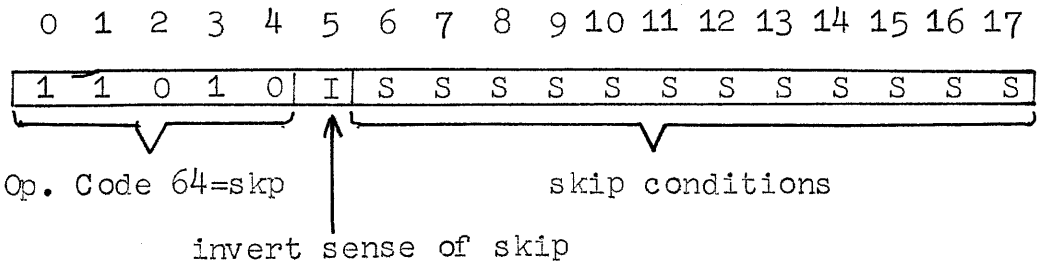
Mnemonic	Op.Code	Name	Function
xct y	10	execute	execute the contents of y as an instruction

1.5 Non-Memory Referencing Instructions

1.5.1 Skip Class

Instructions from the skip class will cause the PDP-1 to jump over one instruction in the normal sequence if the skip condition described by the low 13 bits of the instruction is true.

A skip class instruction has the following format --



Each of the low 12 bits enables a different skip condition.

Mnemonic	Op.Code	Name	Function
skp	640000	skip	never skips
szf n	64000n	skip on zero flag n ($1 \leq n \leq 7$)	skip if program flag n is off. szf 7 skips if all flags are off.
szs n0	6400n0	skip on zero switch n ($1 \leq n \leq 7$)	skip if sense switch n is off. szs 70 skips if all switches are off.
sza	640100	skip on zero A	skip if A(0-17) = 0
spa	640200	skip on positive A	skip if A(0) = 0
sma	640400	skip on minus A	skip if A(0) = 1
szo	641000	skip on zero overflow	skip if the overflow bit (OVF) is off. A szo instruction always clears OVF
spi	642000	skip on positive I	skip if I(0) = 0
sni	644000	skip on nonzero I	skip if I(0-17) $\neq$ 0

If more than one skip condition is enabled, the instruction skips if any (i.e. the logical OR) of the skip conditions is true.

The "i-bit" reverses the sense of the skip, i.e., a skip instruction with the "i-bit" on will skip if and only if the corresponding instruction with the "i-bit" off would not skip. For example, skp 4200 will skip if either I $\neq$ 0 or A(0) = 0. skp i 4200 will not skip if either of these conditions is true. Thus, skp i 4200 will skip only if I = 0 and A(0) = 1.

The following mnemonics define several useful compound skip instructions, i.e, each has several skip conditions enabled.

Mnemonic	Op.Code	Name	Function
szm	640500	skip on zero or minus accumulator	skip if A(0-17) = 0 or if A(0) = 1 (szm = smaVsza)
spq	650500	skip on positive quantity	skip if A > 0 (spq = smaVsza i)
clo	651600	clear overflow	this never skips; it is used to clear overflow (clo = spaVsmaVszo i)

1.5.2 Shift/Rotate Class

The shift/rotate instruction class is markedly different in one's mode and two's mode. A separate explanation of the shift/rotate instructions is given for each arithmetic mode.

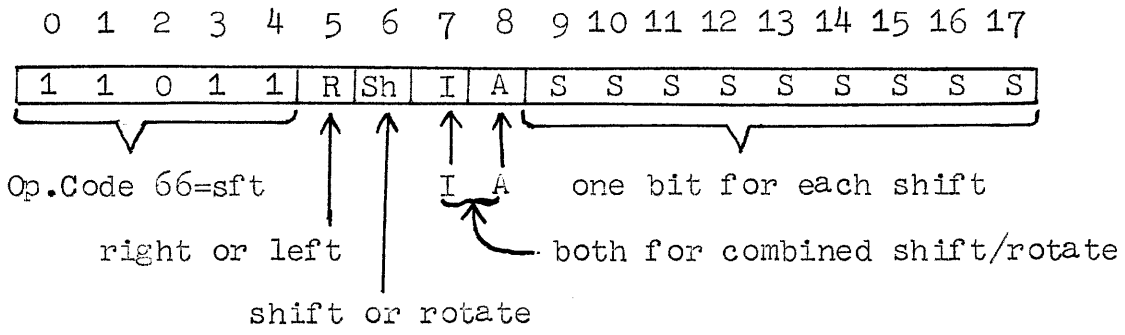
1.5.2.1 One's Complement Mode

Shift is an arithmetic operation. Shifting a number left one bit position is equivalent to multiplying it by two. A single right shift divides a number by two. The sign bit does not change during a shift because the sign of the result must be the same as that of the operand. The sign bit is copied into the bit vacated by the shift.

Rotate is a logical operation in which the register being rotated is considered to be a ring with bits leaving one end and coming back into the other.

The shift/rotate class may operate on either A or I or both combined as a double-length register. In the case of a combined (AI) shift, the sign of the accumulator is taken as the sign of the whole 36-bit number. No shift/rotate instruction will set overflow.

The number of bit positions of shift or rotate is determined by how many of the low nine bits of the instruction are on. Thus, ral 1, ral 2, and ral 400 are equivalent.

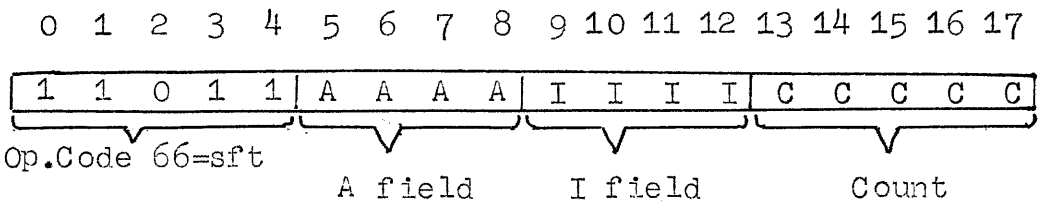


Mnemonic	Op.Code	Name and Function
sft	660000	
sal	665000	shift A left
sar	675000	shift A right
ral	661000	rotate A left
rar	671000	rotate A right
sil	666000	shift I left
sir	676000	shift I right
ril	662000	rotate I left
rir	672000	rotate I right
scl	667000	shift combined AI left
scr	677000	shift combined AI right
rcl	663000	rotate combined AI left
rcr	673000	rotate combined AI right

1s	1	Used in conjunction with
2s	3	the above to shift or
3s	7	rotate the indicated
4s	17	number of places
5s	37	
6s	77	
7s	177	
8s	377	
9s	777	

1.5.2.2 Two's Complement Mode

The two's complement mode shift instruction offers a very powerful set of operations including rotates, logical and arithmetic shifts, normalization, and bit counting. The register operated upon may be the A, I, AI (bit 17 of A joined to bit 0 of I), or IA (bit 17 of I joined to bit 0 of A). The A and I registers may be operated upon independently.



The 4 bits of the A and I fields are interpreted as follows:

D = direction: 0 => right 1 => left
RRR = operation: decoded as follows:

0:

If direction = right, do nothing to this register. If direction = left, zeroes are shifted into bit 17, bit 0 is unchanged; if bit lost from bit 1 ≠ bit 0, set overflow (arithmetic shift left).

1:

If direction = right, see below.
If direction = left, bit 0 of opposite register is shifted into bit 17, bit 0 is unchanged; if bit lost from bit 1 ≠ bit 0, set overflow (arithmetic shift left combined).

2:

Shift zeroes into vacated bit (logical shift).

3:

Shift ones into vacated bit.

4:

Shift bit 0 of this register into vacated bit.

5:

Shift bit 17 of this register into vacated bit.

6:

Shift bit 0 of opposite register into vacated bit.

7:

Shift bit 17 of opposite register into vacated bit.

The mnemonics? for 2's mode shift instructions are:

$\alpha\beta\gamma$

where:	*Mnemonic* for	Meaning
$\alpha \in \{L, R\}$	Left / Right	Specify direction

$\beta \in$		Code =
F	with overflow Stop	0
G	Gross	1
Z	Zeroes	2
O	Ones	3
P	Propagate	4 if right, 5 if left
R	Rotate	5 " , 4 "
V	reVerse bits	6 " , 7 "
C	part of Combined	7 " , 6 "
S	Shift	4 " , 0 "

$\gamma \in$	
A	AC
I	IO
C	Combined
R	Reverse combined
N	Normalize

~~$\alpha RC \Rightarrow RCA \vee RCI$; $\alpha RC \Rightarrow RCA \vee RCI$~~

$LSC \Rightarrow LGA \vee LZI$; $\alpha RC \Rightarrow \alpha CA \vee \alpha CI$

$L\beta C \Rightarrow LCA \vee L\beta I$; $R\beta C \Rightarrow R\beta A \vee RCI$

$L\beta R \Rightarrow L\beta A \vee LCI$; $R\beta R \Rightarrow RCA \vee R\beta I$

$LSR \Rightarrow LZA \vee LGI$; $\alpha RR \Rightarrow \alpha CA \vee \alpha CI$

$\alpha\beta N \Rightarrow RGA \vee \alpha\beta I$

If A field = 01, then:

If the I field is 00, 10, or 11, then shift I until either I(0) ≠ I(1) or the count runs out. A will contain the number of places shifted (Normalize).

If the I field is not 00, 10, or 11, then I will shift as usual. A will contain the number of ones shifted out of I(17) (Count Bits).

Note: A is first cleared and then counts during the operation, hence I field = 11 will shift in zeroes just as I field = 10 does.

I field = 01 is reserved for future expansion.

The number of positions to move the selected register(s) is determined by the number in the 5 bit count field. If the count field is 0, the count is taken from X(13-17).

Execution time = 1 cycle + (count-5)×0.2 microsecond

USAGE

ID and the assembler consider certain symbols consisting entirely of upper case letters to be two's mode shift instructions. The entire symbol must be in upper case, and may appear in any expression, e.g., storage word, constant, etc. When typing into ID, the instruction must be preceeded by a single quote (').

All of the common operations possible with this instruction can be specified by a 2, 3, or 4 letter symbol.

The symbols specifying shift/rotate operations are formed as follows:

<direction field>×~~<register field>~~function×~~<shift-in field>~~register

<direction field>

The direction field indicates the direction in which the selected register is to be moved. The only permissable contents of this field are R or L, specifying right and left motion, respectively.

CHANGED

CHANGED

<register field>
The register field selects the register which is to be operated upon. The register field may contain either an A, I, C, R, or N.

A indicates the A register.

I indicates the I register.

C indicates the combined A and I registers (AI). In this mode, bit 17 of A is joined to bit 0 of I.

R indicates the reverse combined I and A registers (IA). In this mode, bit 17 of I is joined to bit 0 of A.

N indicates that the operation to be performed is a normalize or count bits operation (A field = 01). LNS indicates that I is to be normalized, i.e., I is to be shifted left until I(0) ≠ I(1) or the count runs out. The number of positions shifted will be placed in A. If the symbol is not LNS, then the operation is count bits. I is the selected register and is moved as specified by the count, <direction field>, and <shift-in field>. The number of ones shifted out of I(17) is placed in A.

<shift-in field>
The shift-in field determines what is to be shifted into the bit position that is vacated as the selected register is moved. This field may be blank or it may contain one of the symbols Z, O, L, H, OL, OH, or S.

If the shift-in field is empty, a rotate is assumed (a rotate is an operation where the bits shifted out of one end of the register are shifted into the other end of the register).

Z indicates that zeroes are to be shifted in.

O indicates that ones are to be shifted in.

L indicates that the low bit of this register is to be shifted in. In the case of double register operations (register field = C or R), the low bit is the least significant bit of the entire 36-bit register.

H indicates that the high bit of the selected register is to be shifted in. In the case of double register operations, the high bit is the most significant bit of the entire 36-bit register.

CHANGED

OL indicates the the least significant bit of the non-selected register is to be shifted in. In the case of double register operations, this bit is taken to be the least significant bit of the most significant register.

OH indicates that the most significant bit of the non-selected register is to be shifted in. In the case of double register operations, this bit is taken to be the most significant bit of the least significant register.

S indicates that the operation is an arithmetic shift. The selected register is shifted arithmetically. Note that an arithmetic shift is equivalent to a multiplication or division by a power of 2. During an arithmetic right shift, sign bits are moved into the vacated bit position; during an arithmetic left shift, zeroes are moved into the vacated bit position. If any significant bits are shifted out during a left shift, overflow is set (shifting out a significant bit will change the sign bit).

EXAMPLES

OPERATION	A	I	AI	IA
shift left	LAS	LIS	LCS	LRS
shift right	RAS	RIS	RCS	RRS
rotate left	LA or LAH	LI or LIH	LC or LCH	LR or LRH
rotate right	RA or RAL	RI or RIL	RC or RCL	RR or RRL
logical left	LAZ	LIZ	LCZ	LRZ
logical right	RAZ	RIZ	RCZ	RRZ

count bits in I	RN 18.
normalize I	LNS 17.

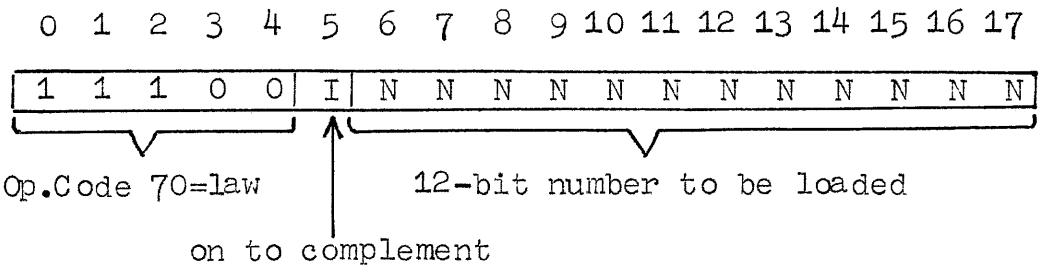
Two rotate/shift operations, one moving only A and the other only I may be combined into one instruction by logically ORing together the instruction moving A and that moving I. Both ORed instructions must share the same count.

EXAMPLES

shift A right 3; shift I left 3	RASVLIS 3
rotate A left 7; shift I right 7	LAVRIS 7
copy reversed bits of I into A	LAOLVRI 18.
clear A and I	RIZVRAZ 18.
reverse 36 bits of AI	LAOLVRIOH 18.

1.5.3 The law Instruction

law and law i ("load accumulator with") make it possible to load the accumulator with 12 bit positive or negative numbers in one cycle.



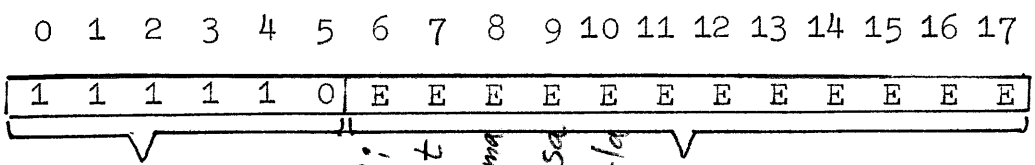
law first loads A from the address part of the instruction and then, if the i-bit is on, complements A. Thus, law 3 puts three into A and law i 3 puts 777774 into A.

In one's complement mode, law i n (where $0 \leq n \leq 7777$) will put -n (one's complement) into A.

For convenience when programming for two's complement mode, lan ("load accumulator with negative") is defined as law 7777. Thus, lan n ($1 \leq n \leq 10000$) will load A with -n (two's complement).

1.5.4 The Operate Class

Each bit of the address part of an opr instruction enables an operation in the processor.



Several operations can be enabled by turning on more than one bit in the address part. The order in which the various operations on A and I occur is -- first, clear A and clear I; third ~~second~~, complement and step A; ~~third~~, OR in the test word; fourth, complement A and complement I; and last, switch data between A and I.

second

Mnemonic	Op.Code	Name	Function
opr	760000		
nop	760000	no operation	one cycle time delay
stf n	76001n	set flag n ($1 \leq n \leq 7$)	set one of the six program flags. Set all if n=7. Set none if n=0
clf n	76000n	clear flag n ($1 \leq n \leq 7$)	clear selected flag. Clear all if n=7. Clear none if n=0
cla	760200	clear A	put 0 into A
	762000	OR test word to A	
lat	762200	load test word	copy test word switches into A
csa	761400	complement & step A	
cma	761000	complement A	
clc	761200	clear and complement A	put 777777 in A
csc	761400	complement, step, and complement A	Subtract 1 from A.
cli	764000	clear I	put 0 into I
cmi	760100	complement I	
lia	760020	load I from A	
lai	760040	load A from I	
swp	760060	swap A & I	

lia	760020	load I from A	
swp	760060	swap A and I	
csa	760400	complement and step A	complement A, then add one to A. In one's mode, if A contains 1, csa will result in 777777. In 2's mode, negate. If A contained 400000, OVF is set.

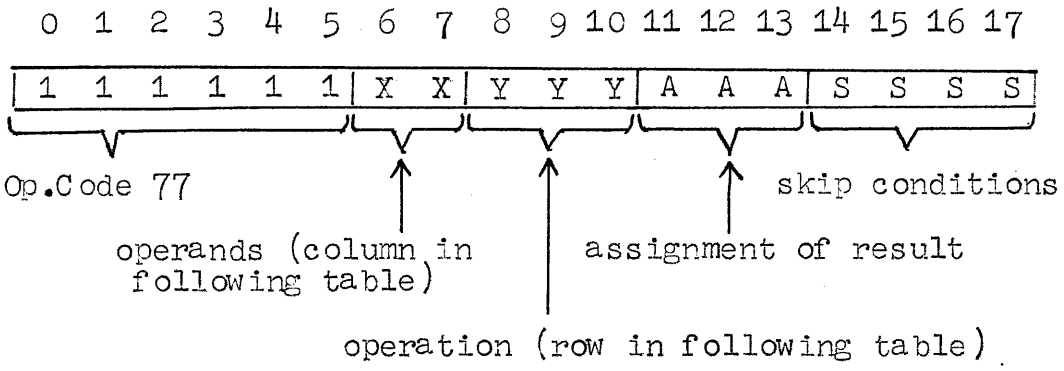
~~Note: csa/cma subtracts 1 from A.~~

Although the stf and clf instructions are usually used to manipulate the program flags, there are also the following two instructions --

Mnemonic	Op.code	Name	Function
lpf	770051	load program flags	Top 2 bits of I to Address mode bits. Bottom 6 bits of I to program flags.
rpf	770050	read program flags	Address mode to top 3 bits of I. Flags to low 6 bits of I.

1.5.7 The Micro-program Class

The micro-program instruction class has the following instruction format --



Symbolic Specification of Micro-program Operations

YYY \ XX	00	01	10	11	
000	SPECIAL INSTRUCTIONS	TI	TA	TX	T (Test, or transfer)
001		NI	NA	NX	N (Negate)
010		A→I (I)	X→A (A)	X→I (I)	exchange (see explanation below)
011		AMI	XMA	XMI	M (arithmetic minus)
100	Z (zero)	AVI	XVA	XVI	V (inclusive or)
101	SA (A)+1	A∧I	X∧A	X∧I	∧ (bitwise and)
110	SI (I)+1	A~I	X~A	X~I	~ (exclusive or)
111	SX (X)+1	A+I	X+A	X+I	+ (arithmetic plus)

S (Step, i.e., add one)

The functions of the result assignment and skip condition fields are as follows --

AAA	Bit 11=1 will put the result in A	
	Bit 12=1 " " " " " I	
	Bit 13=1 " " " " " X	

		Mode	One's	Two's
SSSS	Bit 14=1:	Skip if the result is	> 0	> 0
	Bit 15=1:	" " " " "	< -0	< -1
	Bit 16=1:	" " " " "	= 0	= 0
	Bit 17=1:	" " " " "	= -0	= -1

The execution of a micro-program instruction computes the result specified by the XX and YYY bits (i.e. from the Table on the previous page), puts this result in the specified register(s) if any, and skips if any of the specified conditions are true. Note that the skip condition is evaluated on the result of the micro-program, not the final contents of any given register, and the result need not be assigned to any register. Thus, it is possible to test the sum of A and I to see if it is greater than 0 without destroying the contents of these registers. The instruction to do this is 773610, or symbolically, A+I>. All micro-program instructions require one cycle.

The "T" operation clears the transmitted register after the transfer. This may be circumvented by assigning the result back to the transmitted register. See the examples.

The exchange operation is a special case. The result A→I (I) means that the result of the function is the old I register, but in addition the accumulator is placed in I. This secondary assignment is done before the assignment given by the AAA bits in the instruction. Thus, the instruction A→I (772400) will move A to I, A→IA (772500) will swap A and I, and A→II (772440) does nothing (because the primary assignment is to I and the result is the old I).

The add, subtract, step, and negate operations vary according to the arithmetic mode. In one's mode, negate produces the complement of the specified register. In two's mode, it produces the complement plus one.

In one's mode, arithmetic results (from add, subtract, negate, and step) of -0 are converted to +0, except in the following cases:

$$(-0) + (-0) = (-0)$$

$$(-0) - (+0) = (-0)$$

$$- (+0) = (-0)$$

Micro-program instructions executed in one's mode never turn on the overflow bit. In two's mode, the instructions add and subtract will turn on the overflow bit if the addition or subtraction overflows. Stepping 377777 and negating 400000 also set overflow in two's mode.

USAGE

The assembler considers certain symbols consisting of capital letters as micro-program instructions. The entire instruction must be in upper case, and may appear in any expression, e.g., storage word, constant, etc. When typing into ID, the instruction must be preceded by a single quote (').

Micro-program instructions are specified by concatenating three "fields" -- the result field, the assignment field, and the skip field. The characters in all of these must be in upper case and there must be no separator between the fields.

<result field><assignment field><skip field>

<result field>

The result field must be one of the twenty-eight results given in the table on a previous page. "C" may be used in place of "N" (negate).

<assignment field>

The assignment field may be null (no characters) or any combination of A, I, and X to specify in which registers the result will be placed.

<skip field>

The skip field may be null or contain any combination of <, >, P, M, |, _, and =. Note that the <, =, and _ are treated differently depending on the arithmetic mode.

Symbol	Meaning:	One's	Two's
>	skip if result	>0	>0
P	skip if result	=+0	=0
M	skip if result	=-0	=-1
<	skip if result	<-0	<0
=	skip if result	=+0,=-0	=0
_	skip if result	=+0,=-0	=0
<M	skip if result	<-0	<-1
|	invert the specified skip conditions		

Since the octal representation of a micro-program instruction is computed by exclusive-or'ing all of the specifications within each field, redundant specifications may lead to unexpected results. For example, TAIII and TAI>> are the same as TAI.

SAMPLE MICRO-PROGRAM INSTRUCTIONS

The following are the same in One's and Two's Mode.

symbolic	octal	action
A+I	773600	computes sum of A and I and does nothing at all with it.
A+IA	773700	the sum of A and I is put into A.
A+IA IX	773760	the sum of A and I is put into A, I, and X.
ZA IX	771160	A, I, and X are cleared.
SA >	771210	skip if A plus one is > 0.
SAAP	771302	add one to (step) A and skip if it is 0.
TXM	776001	skip if X is 777777 and clear X.
TXXM	776021	skip if X is 777777.
TAXI	774060	transfer the contents of A into I and X, then clear A.
TAAXI	774160	same as TAXI, but A is not cleared.
A→IA	772500	exchange the contents of A and I.
X→A >P	774412	transfer the contents of X into A, skip if the previous A is positive.
SAM	771201	skip if A plus one is ~~777777~~. Note that ~~in two's complement mode,~~ this will skip only if A contains -2. ~~In one's mode, this instruction never skips~~.

Oops. I ignore crossing out. Typed text is correct.

The following are different in One's and Two's Mode.

One's Mode

symbolic	octal	action
A+I<P	773606	skip if the sum of A and I is < -0 or =+0.
A+I<P|	773611	skip if the sum of A and I is not (< -0 or =+0).
A+IX≥	773633	the sum of A and I is put into X. If the sum was ≥ -0, the instruction will skip.
NXP	776202	skip if X is 777777.

Two's Mode

A+I<P or A+I ≤	773607	skip if the sum of A and I is ≤ 0.
A+I<P| or A+I ≠	773610	skip if the sum of A and I is > 0.
A+IX≥	773632	the sum of A and I is put into X. If the sum is ≥ 0, the instruction will skip.
NXP or NX=	776202	skip is X is 0.