PDP-11 instruction reference

CSC 258

Overall notes

Most numbers in this document are in octal (base 8).

"msb" means "most significant bit" (sometimes "byte", but always "bit" in this document).

The "bit scheme" for condition codes is:

N = msb of result
Z = whether the result is zero
V is reset (i.e. V becomes 0)
C is not affected

The "value scheme" for condition codes is:

N,Z are determined by the final result as above
V is determined by whether arithmetic overflow occurred
C is not affected unless specified

The symbol "$^{0}_{1}$" in an opcode refers to a 0 for the word version of the instruction and a 1 for the byte version of the instruction.

Addressing modes

The six bits indicating addressing type are divided into:

- three bits for "mode"
- three bits to indicate a register (yielding a register number from 0 to 7, inclusive)

The top two of the three mode bits work as follows:

top two bits	octal	name	assembly syntax	EA and other semantics
00_	0 or 1	register	Ri	Ri
01_	2 or 3	autoincrement	(Ri)+	[Ri], then Ri←[Ri]+2 or 1
10_	4 or 5	autodecrement	−(Ri)	first Ri←[Ri]−2 or 1, then EA=new [Ri]
11_	6 or 7	index	n(Ri)	[[R7]]+[Ri], then inc PC by 2 (n follows in next memory word)

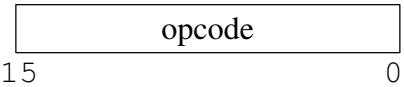
The third mode bit is "indirect". It yields one extra indirection. In the assembly syntax, we add an "@".

Some modes have a special assembly syntax when used with R7:

mode bits	octal	name when used with R7	assembly syntax	EA and other semantics
010	2	immediate	#n	EA=[R7], then R7←[R7]+2
011	3	absolute direct	@#n	EA=[[R7]], then R7←[R7]+2
110	6	relative direct	n	X=[[R7]], then inc R7, then EA=X+[R7]
111	7	relative indirect	@n	X=[[R7]], then inc R7, then EA=[X+[R7]]

Zero-operand instructions

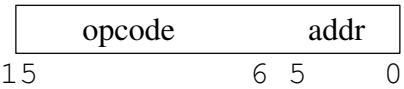
Format:



Mnemonic	Opcode	Semantics
HALT	000000	Halt CPU until restarted; abort i/o
WAIT	000001	Halt CPU until restarted or interrupted
RESET	000005	Reset all i/o devices
NOP	000240	No operation

One-operand instructions

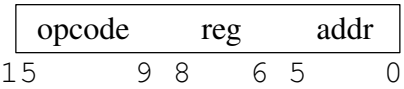
Format:



Mnemonic	Opcode	Semantics	Condition code notes
CLR{B}	%1050	*addr* ← 0	N,V,C reset; Z set
INC{B}	%1052	*addr* ← [*addr*] + 1	"value" scheme
DEC{B}	%1053	*addr* ← [*addr*] − 1	"value" scheme
ADC{B}	%1055	*addr* ← [*addr*] + [C] i.e. add 1 if C=1, add 0 if C=0	"value" scheme; C is carry from addition
SBC{B}	%1056	*addr* ← [*addr*] − [C]	"value" scheme; C is "borrow" from subt.
TST{B}	%1057	set condition codes by [*addr*]	N,Z by [*addr*]; V,C reset
NEG{B}	%1054	*addr* ← −[*addr*]	"value" scheme (overflows iff was 100000); C reset if result=0, C set if result≠0
COM{B}	%1051	*addr* ← one's complement of [*addr*]	N,Z by result; V reset; C set
ROR{B}	%1060	*addr* ← rotate [*addr*] right one bit through carry	N,Z by result; C by rotation; V = new N ⊕ new C
ROL{B}	%1061	*addr* ← rotate [*addr*] left one bit through carry	as ROR{B}
ASR{B}	%1062	*addr* ← arithmetic-shift [*addr*] right one bit	as ROR{B}; C = old low bit
ASL{B}	%1063	*addr* ← shift [*addr*] left one bit	as ROR{B}; C = old high bit
SWAB (swap bytes)	0003	*addr* ← [*addr*] with high/low bytes swapped (it's a word instruction, despite the "B")	C,V reset; N,V by *low* byte of result (i.e. by source!!)
SXT (sign-extend)	0067	addr ← all bits [N] (the condition code)	N,C,V not affected; Z set by result

One-and-a-half-operand instructions

Format:

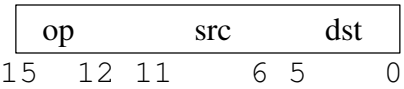


Note! The "addr" field always comes last in the machine-language format *but* the target field (which may be "addr" *or* "reg", according to the semantics) is what comes last in the assembly-language format.

Mnemonic	Opcode	Semantics	Condition code notes
MUL	070	*reg*;*reg*∨1 ← [*reg*] × [*addr*] If *reg* is odd, only the low-order word is stored.	N, Z describe entire product; V reset; C set iff low word overflows
DIV	071	*reg* ← [*reg*;*reg*∨1] `div` [*addr*] *reg*∨1 ← [*reg*;*reg*∨1] `mod` [*addr*] *reg* must be even.	C,V both set if divide by 0; else V set iff `div` result would not fit in *reg*
ASH	072	*reg* ← [*reg*] shifted by low 6 bits of [*addr*]; shift value is interpreted as signed, where negative means right	V set iff msb of [*reg*] changed during shift; C = last bit shifted out
ASHC (combined)	073	*reg*;*reg*∨1 ← [*reg*;*reg*∨1] shifted by low 6 bits of [*addr*], as with ASH	V set iff msb changed; C = last bit shifted out
XOR	074	*addr* ← [*reg*] ⊕ [*addr*]	"bit" scheme

Two-operand instructions

Format:

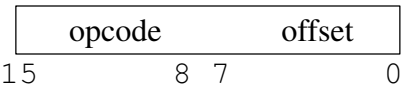


Mnemonic	Opcode	Semantics	Condition code notes
MOV{B}	$^{0}_{1}$1	*dst* ← [*src*] In MOVB, if *dst* is reg, is sign-extended.	"value" scheme; V reset
ADD	06	*dst* ← [*dst*] + [*src*]	"value" scheme; C is carry from addition
SUB	16	*dst* ← [*dst*] − [*src*]	"value" scheme; C is "borrow" from subt.
CMP{B}	$^{0}_{1}$2	compute [*src*] − [*dst*] (not stored) N.B. different order from SUB!	"value" scheme; C is "borrow" from subt.
BIS{B} (bit set)	$^{0}_{1}$5	*dst* ← [*dst*] ∨ [*src*] (bitwise)	"bit" scheme
BIC{B} (bit clear)	$^{0}_{1}$4	*dst* ← [*dst*] ∧ $\overline{[src]}$ (bitwise)	"bit" scheme
BIT{B} (bit test)	$^{0}_{1}$3	compute [*src*] ∧ [*dst*] (bitwise) (not stored)	"bit" scheme

Branch instructions

(also see SOB in the following section)

Format:



(octal coding doesn't work out too nicely here)

Mnemonic	Name	Binary opcode	Branch condition
BR	Branch	00000001	unconditional
BNE	Branch on not equal	00000010	Z = 0
BEQ	Branch on equal	00000011	Z = 1
BPL	Branch on plus	10000000	N = 0
BMI	Branch on minus	10000001	N = 1
BVC	Branch on overflow clear	10000100	V = 0
BVS	Branch on overflow set	10000101	V = 1
BHIS	Branch on higher than or same as	10000110	C = 0
BCC	Branch on carry clear	10000110	C = 0
BLO	Branch on lower	10000111	C = 1
BCS	Branch on carry set	10000111	C = 1
BGE	Branch on greater than or equal to	00000100	$N \oplus V = 0$
BLT	Branch on less than	00000101	$N \oplus V = 1$
BGT	Branch on greater than	00000110	$Z \vee (N \oplus V) = 0$
BLE	Branch on less than or equal to	00000111	$Z \vee (N \oplus V) = 1$
BHI	Branch on higher than	10000010	$C \vee Z = 0$
BLOS	Branch on lower than or same as	10000011	$C \vee Z = 1$

The eight bits of the offset are interpreted as the high eight bits of a nine-bit signed offset whose last bit is zero. Thus if the condition is met, we do R7 ← [R7] + 2 × offset.

BHI, BHIS (aka BCC), BLO (aka BCS), and BLOS do what they say if the data is interpreted as unsigned; the use of BLT, BLE, BGT, and BGE tends to interpret the data as two's-complement.

Other instructions involving transfer of control

Format: according to schema below
Where R is a register, and AA is an address in any of the standard addressing modes yielding an EA
(condition codes are not affected, except by setting the entire PSW where indicated)

Mnemonic	Name	Octal schema	Semantics
JMP	Jump	0001AA	R7 ← EA (use @ to get [EA]) (can't use plain register mode if not '@')
SOB	Subtract One and Branch	077Rnn	R ← [R] − 1 then if [R] ≠ 0, R7 ← [R7] − 2×nn Note: nn is treated as unsigned; can only jump backward. Cond codes unchanged.

(table continues)

Other instructions involving transfer of control, continued

Mnemonic	Name	Octal schema	Semantics
JSR	Jump to subroutine	004RAA	*temp* ← EA R6 ← [R6] – 2 [R6] ← [R] R ← [R7] R7 ← [*temp*]
RTS	Return from subroutine	00020R	R7 ← [R] R ← [[R6]] R6 ← [R6] + 2
RTI	Return from interrupt (or trap)	000002	R7 ← [[R6]] R6 ← [R6] + 2 PSW ← [[R6]] R6 ← [R6] + 2
TRAP	Trap	104xyz, x≥4	R6 ← [R6] – 2 [R6] ← [PSW] R6 ← [R6] – 2 [R6] ← [R7] R7 ← [34] PSW ← [36] This seq. is hereby called “trap from 34”.
BPT	Breakpoint trap. Used by debuggers.	000003	R6 ← [R6] – 2 [R6] ← [PSW] R6 ← [R6] – 2 [R6] ← [R7] R7 ← [14] PSW ← [16] i.e. “trap from 14”.
IOT	I/O trap. Used by OS for I/O calls.	000004	trap from 20 (see above)
EMT	Emulator trap. Used by OS to implement fake ops.	104xyz, x≤3	trap from 30 (see above)
RTT	Return from trace trap	000002	same as RTI, but suppresses the immediately-following trace trap. Typically, [14] will have the T bit clear; after returning, we want to trap after the next instruction, but not after the RTT itself when the previous PSW is restored.

Note that the opcode in the TRAP and EMT ops is the high byte, and the low byte can be anything (intended to be interpreted by the ISR).

Processor status word (PSW)

The PSW's high byte contains 11/45-specific stuff about kernel and supervisor modes. The low byte contains the priority (three highest bits), followed by T, N, Z, V, and C, in that order.

An external interrupt which is not precluded by the CPU priority causes a trap from the vector address appropriate to the interrupting device (see trap sequence, previous section). Zero is the normal priority. Seven is the highest priority; a higher-number interrupt interrupts a lower priority but not vice versa. (This is the PDP-11 ordering; it differs among CPUs.)

When T is set, the execution of every instruction except for RTT causes a subsequent trace trap. A trace trap is a trap from 14 (i.e. it's the same as executing a BPT).

The processor status word is also affected by the following ops:

Mnemonic	Name	Octal schema	Semantics
SPL	Set priority level	00023n	bits 7-5 of PSW ← n
CLC	Clear C	000241	C ← 0
CLV	Clear V	000242	V ← 0
CLZ	Clear Z	000244	Z ← 0
CLN	Clear N	000250	N ← 0
SEC	Set C	000261	C ← 1
SEV	Set V	000262	V ← 1
SEZ	Set Z	000264	Z ← 1
SEN	Set N	000270	N ← 1
CCC	Clear condition codes	000257	C,V,Z,N ← 0
SCC	Set condition codes	000277	C,V,Z,N ← 1

In fact, you can make up your own "clear" and "set" ops in the implied pattern; the last four bits of the word indicate whether N, Z, V, and/or C are being referred to, in that order. This also gives us our NOP op ("clear nothing").

Alternatively, the PSW can be addressed as location −2, but this should only be done by the OS.

Pseudo-ops and other syntax

Pseudo-ops are lines in your assembly-language program which are instructions to the assembler. Some of them do not directly correspond to generated code.
They all begin with a period to distinguish them from ops.

Some of the pseudo-ops we will be using are:

.ORG — start assembling at the given address. Example: the next word following a ".ORG 1000" line will be placed at memory address 1000.
.WORD — emit the given value as one word. Example: ".WORD 264" is the same as "SEZ".
.BLKW — emit the stated number of zero words. Example: ".BLKW 3" emits 6 zero bytes.
.BLKB — emit the stated number of zero bytes.
.END — this line must appear as the last line of your assembly language file. It is entirely unrelated to "RTS". It serves no function and we won't use it in this course.

A *label* is an identifier at the beginning of a line, followed by a colon. This causes the address of the instruction following the label to be entered into the assembler's symbol table. There is also a syntax "*var* = *value*" (example: "N = 5"). The symbol "." is always the current instruction's address. These symbols can be used anywhere an integer can be used.

A semi-colon begins a comment, which extends to the end of the line.