PDP-11 CPU

a documentation

Contents

Chapter 0

PDP-11 Instruction Set

> We are to admit no more causes of natural things than such as are both true and sufficient to explain their appearances.
>
> Isaac Newton, Principles: A System of the World

This paper describes the PDP-11/40 Instruction Set, including the Extended Instruction Set (EIS) and the Memory Management Unit (MMU). This particular configuration is similar to the machine John Lions used during his studies of Unix Version 6 at the University of New South Wales in 1976/1977. Features of the more powerful models PDP-11/45 and PDP-11/70 are appended since on these machines Ken Thompson and Dennis Ritchie developed Unix V6 and V7 in the 1970s.

Notation: The contents of registers, memory and addresses are represented as nonnegative integers. Leading zero indicates octal and leading "0x" indicates hexadecimal notation. The (horizontal or vertical) sequence of digits starts always with the most significant one. "Low memory" are cells with low addresses. "K" equals 2^{10}. Integer ranges "[i, k)" include i and exclude k if i ¡ k and are empty otherwise. The C language is employed for expressions.

0.1 Registers

All registers are 16-bit wide.

0.1.1 General Registers R0, R1, R2, R3, R4, R5 (FP), R6 (SP), R7 (PC)

The PDP-11 has eight general registers. They are addressed explicitly by a 3-bit register number which is encoded in instruction words. In addition to this, three registers are used implicitly, namely:

R7, the program counter (PC), points to the instruction to be executed next. After each fetch of a word addressed implicitly by the PC, the PC is incremented by 2, before being used in address calculations.

R6, the stack pointer (SP), supports subroutine, trap and interrupt linkage by employing a "last in/first out" memory allocation scheme. It is used implicitly among others by jsr, rts, mark, sys and rtt. There are two SPs, one active in user mode (USP), the other active in kernel mode (KSP). "Push x" allocates memory and saves the value x, "pop y" restores y and frees the memory allocated for y. The stack builds toward low memory. If the SP was initalized to i, the stack occupies the memory at [SP, i). Thus the SP addresses the item allocated last.

R5, the frame pointer (FP), is used implicitly solely by the mark instruction to support subroutine linkage.

0.1.2 Processor Status Word PSW (at 0777776)

The bits of the PSW are assigned as:

name	width	meaning
CM	2	current operation mode, 00 is kernel mode, 11 user mode
PM	2	previous operation mode
***	4	not used
IPL	3	interrupt priority level
T	1	enable trace trap
NZVC	4	condition codes

A pending interrupt will be served if the hardwired bus request number (BR) of the interrupting device is greater than the IPL.

If the T-bit is set, a trap at 14 is taken after execution of an instruction. The T-bit enables debuggers to single-step a program.

Arithmetical and boolean instructions modify the condition codes (CCs) to be sensed later by conditional branch instructions.

N indicates "result is negative"

Z indicates "result is zero"

V indicates "signed overflow"

C indicates the carry bit, i.e., unsigned additive overflow

0.1.3 Panel Registers (at 0777570)

There are two panel registers: a read only switch register (SR), which mirrors the state of the panel switches, and a write only display register (DR) to control the panel lights.

After power on, all but two registers are set to zero. The IPL in the PSW is set to 7 and the switch register is left as it is.

0.2 Addresses

16-bit (virtual) addresses are translated to 18-bit (physical) bus addresses. The first 58K addresses are mapped to memory, the remaining 8K addressses are used to access device registers and are mapped to the last 8K bus addresses. The last 8K block is called "I/O page".

```
This, this mapping is defined as:
        [0,    56K) --> [0,      56K),     x --> x
        [56K, 64K) --> [248K, 256K),     x --> x + 192K (== x | 0600000)
```

The memory management unit (MMU) can be enabled to set this mapping under software control. The MMU handles two mappings (aka address spaces)–one is active in user mode, the other in kernel mode. After power on, the MMU is disabled.

0.3 Operands and Operations

Operands are stored in 8-bit bytes, 16-bit words and 32-bit longwords. The multiplicative instructions multiply, divide, and "arithmetic shift concatenated" (i.e., mul, div, ashc) are the only ones that operate on longwords.

The instruction set provides signed integer arithmetic (byte, word and longword), operations on boolean arrays of length 8 or 16, and additive artithmetic on unsigned integers.

Most binary and unary operations are implemented for byte and word sized operands. Addition and subtraction operate on words only.

Words are stored in memory or registers, bytes in memory only. The address of a word is even and equals the address of the word's low byte ("Little Endian"). If a register is the destination of a byte operation, the signed value of the byte is stored in the register, i.e., the sign bit is extended to the high byte of the register. A register is truncated to its low byte if it is the source of a byte operation.

A longword operand is stored in a pair of general registers RS, with r, the number of R, being even and the number of S being r+1. R holds the high word ("Big Endian").

0.4 Instruction Formats and Address Modes

The parts of an instruction word are:

name	width	meaning
w	1	operand width, 0 specifies word, 1 specifies byte
ss	6	source operand specification
dd	6	destination operand specification
a	3	address mode
r	3	register number
o	varying	operation code
nn	8	signed word offset (in branches) ; system call number (in sys)
nn	6	unsigned number
m	4	mask to select condition flags NZVC in CC operations

The operand specifications ss and dd consist of the address mode and the register number.

Notation: Instructions containing 4-bit or 8-bit parts are notated hexadecimal, the others octal. If octal, the first character stands for 1 bit and the others for 3 bits each; if hexadecimal each character stands for 4 bits. Hexadecimal notation is prefixed by "0x".

name	format	example
double operand	wossdd	mov src,dst
register and operand	woordd	jsr r5,subr
register and operand	woorss	ash 2,r0
single operand	wooodd	inc a
single register	ooooor	rts r5
zero operand	oooooo	halt
branch	0xoonn	br label
CC operation	0xooom	sec

The operand is specificied depending on the address mode:

a–mode	operand	name
0	R	register
1	m[R]	register indirect
2	m[R]; R=R+w	auto–increment (pop)
3	m[m[R]]; R=R+2	auto–increment indirect (pop)
4	R=R–w; m[R]	auto–decrement (push)
5	R=R–2; m[m[R]]	auto–decrement indirect (push)
6	m[m[PC]+R]	indexed
7	m[m[m[PC]+R]]	indexed indirect

Here R is the register selected in the operand specification, m the memory, and w the operand width, i.e., w equals

1 for byte operands and 2 for word operands. If R is the SP, w == 2, even for byte operands.

0.5 Instructions

The instructions are sorted into four lists:

- Traps and other special instructions

- Arithmetic, boolean and data transfer

- Jump and subroutine linkage

- Branches

Notation: Throughout the instruction lists, s and d denote source respective destination operands. On the right side of an assignment ("=") d stands for the value before the operation is applied.

0.5.1 Traps and other Special Instructions

The memory at [0, 0400), the "yellow stack", is reserved for the interrupt vector. An entry spans two words. The first one holds an address of a trap routine respective of an interrupt service routine (ISR) and the second one the program status (PS), which is the low byte of the PSW.

When a trap or interrupt is executed, the CPU switches to kernel mode, pushes the previous PSW and the PC onto kernel stack, sets PM to the previous operation mode, and executes the trap routine with the PS set from the vector. This sequence is abbreviated by "trap at n" where n is the address of the vector entry.

Return to interrupted/trapped program (rti, rtt) pops the PC and the PSW and thus undoes the change of the state done by the trap or interrupt. In user mode, these instructions won't change CM, PM and IPL when they pop the PSW from stack.

rtt differs from rti, in that it will not be trapped even if the T-bit is set during execution. This enables debuggers to execute one instruction of the debugged program.

Accessing word operands or instructions with an odd address, accessing nonexisting memory, executing the halt instruction in user mode, or pushing into the yellow stack while in kernel mode will trap at 4, whereas executing an illegal (i.e., not implemented) instruction traps at 10.

Execution of a trap might fail because a memory error occures while saving PSW and PC. In that case, both words are saved in an emergency stack at [0, 4) and a "red stack trap" is executed at 4.

In both the emulator and system trap instruction its low byte is reserved for operating systems. In Unix it holds the system call number.

ass	read	code	operation
halt	halt	000000	stop the processor (privileged)
wait	wait	000001	wait for interrupt
rti	return to intr	000002	PC = SP++*; PSW = SP++*
bpt	breakpoint	000003	trap at 14 (for debugging)
iot	i/o trap	000004	trap at 20
reset	reset	000005	reset I/O devices (no–op in user mode)
rtt	return to trap	000006	PC = SP++*; PSW = SP++*
emt	emulator trap	0x88nn	trap at 30 (used by DEC)
sys	system trap	0x89nn	trap at 34 (used by Unix)

0.5.2 Arithmetic, Boolean and Data Transfer

Shift/rotate: ror, rot, asr, asl, ash, ashc

These instructions implement multiplicative operations with one operand being a power of two. The C flag is set to the bit shifted out.

Integer Division: div, asr, ror, ash, ashc.

div truncates towards zero (notated by "/") whereas the right shifts truncate to the smaller integer, notated by "//".

Access memory via previous address space: mfpi, mtpi

The mapping of the operand's virtual address is controlled by the previous instead of the current mode's address space. The previous mode is determined from PM. If the operand is the SP, the previous mode's SP is chosen. Mfpi pushes and mtpi pops the operand using the current stack.

CC operations: cln, clz, clv, clc, sen, sez, sev, sec

These instructions clear rsp. set condition codes. The CCs are selected by the mask m in the instruction word. More then one flag can be selected by oring instructions of either the set or the clear type. When no flag is selected, the CC operations degenerate to no-ops.

The list is sorted by instruction code. The byte instructions (codes [100000, 104000), [105000, 106400), [110000, 170000)) are left out.

ass	read	code	operation	Condition Codes affected
cl?	clear CC	0x00Am		NZVC = NZVC & ~m
se?	set CC	0x00Bm		NZVC = NZVC | m
swab	swab bytes	0003dd	d.hi <-> d.lo	N, Z from low byte; V=C=0
clr(b)	clear (byte)	w050dd	d=0	N, Z, V, C
com(b)	complement	w051dd	d=~d	N, Z; V=0; C=1
inc(b)	increment	w052dd	d=d+1	N, Z, V
dec(b)	decrement	w053dd	d=d-1	N, Z, V
neg(b)	negate	w054dd	d=0-d	N, Z, V, C
adc(b)	add carry	w055dd	d=d+C	N, Z, V, C
sbc(b)	sub carry	w056dd	d=d-C	N, Z, V, C
tst(b)	test	w057dd		N, Z; V=C=0
ror(b)	rotate right	w060dd	d=d//2 - C*2^15	N, Z, V; C=d%2
rol(b)	rotate left	w061dd	d=d*2 + C	N, Z, V, C
asr(b)	shift right	w062dd	d=d//2	N, Z, V; C=d%2
asl(b)	shift left	w063dd	d=d*2	N, Z, V, C
mfpi	mv fr prev i-sp	0065ss	--SP* = s	N, Z; V=0
mtpi	mv to prev i-sp	0066dd	d = SP++*	N, Z; V=0
sxt	sign extend	0067dd	d=-N	N, Z, V
mov(b)	move (byte)	w1ssdd	d=s	N, Z; V=0
cmp(b)	compare	w2ssdd		N, Z, V, C as if d = s-d
bit(b)	bit test	w3ssdd		N, Z as if d = s&d; V=0
bic(b)	bit clear	w4ssdd	d = d & ~s	N, Z; V=0
bis(b)	bit set	w5ssdd	d = d | s	N, Z; V=0
add	add	06ssdd	d = d + s	N, Z, V, C
sub	subtract	16ssdd	d = d - s	N, Z, V, C
mul	multiply, r evn	070rss	RS = s*R	N, Z, V (C see next line)
mul	" , r odd	070rss	R = s*R	C="s*R does not fit in word"
div	divide	071rss	R=RS/s; S=RS%s	N, Z, V; C="/ by zero"
ash	arith shift	072rss	R=R*2^s	N, Z, V, C
ashc	" conct., r evn	073rss	RS=RS*2^s	N, Z, V, C (C=RS%2 if s > 0)
ashc	" conct., r odd	073rss	R=R*2^s	N, Z, V, C (C=R%2 if s > 0)
xor	exclusive or	074rdd	d = d ^ R	N, Z; V=0

0.5.3 Subroutine Linkage and Jump

Jump subroutine (jsr) pushes R, saves the return address (i.e., PC, the address of the next instruction) in R, and jumps to the address of d. It follows that if R is the PC, jsr simply pushes the PC and jumps.

The destination of jmp and jsr must not be a register, i.e., the address mode must not equal zero. If it does, a trap at 10 occurs.

Return from subroutine (rts) restores the PC from R and pops R. It follows that if R is the PC, rts just pops it.

With the mark instruction being the word last pushed, jmp (sp) means: Free nn+1 (i.e. including mark) words from the stack, restore the caller's FP and return; nn is a 6-bit unsigned integer.

Subtract one and branch (sob) decrements R and, if R != 0, jumps up nn words, where nn is a 6-bit unsigned integer.

ass	read	code	operation
jmp	jump	0001dd	PC = &d (a–mode > 0)
rts	return fr subr	00020r	PC = R; R = *SP++
jsr	jump subroutine	004rdd	*–-SP = R; R = PC; PC = &d (a–mode > 0)
mark	mark	0064nn	SP = PC + 2*nn; PC = FP; FP = *SP++
sob	subtract one	077rnn	R = R – 1; if R != 0: PC = PC – 2*nn

0.5.4 Branches

If the condition holds, these instructions set PC = PC - 2*nn. The column labeled "s/u" indicates whether the tested condition is appropiate for signed respective unsigned artithmetic.

ass	read	code	s/u	condition	cmp a, b	tst a
br	branch	0x01nn	s/u	always		
bne	not equal	0x02nn	s/u	!Z	a != b	a != 0
beq	equal	0x03nn	s/u	Z	a == b	a == 0
bge	greater equal	0x04nn	s	N == V	a >= b	
blt	less than	0x05nn	s	N != V	a < b	
bgt	greater than	0x06nn	s	!Z && N == V	a > b	
ble	less equal	0x07nn	s	Z | N != V	a <= b	
bpl	plus	0x80nn	s	!N		a >= 0
bmi	minus	0x81nn	s	N		a < 0
bhi	high	0x82nn	u	!C & !Z	a > b	
blos	lower or same	0x83nn	a	C | Z	a <= b	
bvc	overflow clear	0x84nn	s	!V	no overflow	

bvs	overflow set	0x85nn	s	V	overflow
bcc	carry clear	0x86nn	u	!C	a >= b
bhis	higher or same		another name for bcc		
bcs	carry set	0x87nn	u	C	a < b
blo	lower		another name for bcs		

0.6 Memory Management Unit

0.6.1 The Mapping as Controlled by Paging Registers

The virtual address range [0, 64K) is partitioned into eight pages
[0*8K, 1*8K), [1*8K, 2*8K), ... , [7*8K, 8*8K) .

Each page is assigned a page description register (PDR) and a page address register (PAR). The virtual address in a page are mapped to a range of physical addresses which starts at p = PAR*64.

The meaning of the PDR bits are:

name	width	meaning
*	1	not used
S	7	with b=64*S, the lower part of page n is [n, n+b].
*	1	not used
C	1	set, when memory addressed by this page is changed cleared, when the PDR is changed
**	2	not used
U	1	upper part of the page is valid, i.e., mapped
W	1	write permission
R	1	read permission
*	1	not used

Let v be a virtual address contained in the page [n, n+8K). This page is divided into a lower part [n, n+b] and an upper part [n+b, n+8K]. U controls which part of the page is mapped:

```
If U off:       [n,   n+b]   -->[p, p+b],     v  -->p + v - n
If U on:        [n+b, n+8K) -->[p, p+8K–b), v  -->p + v - (n+b)
```

A trap at 0250 is executed if one of the following conditions holds:
R is off
v is not mapped, i.e. not in the lower respective upper part
W is off and the program tried to write to the memory

Physical addresses of the page registers:

kernel: PDR 0-7: [0772300, 0772320), PAR 0-7: [0772340, 0772360)

user: PDR 0-7: [0777600, 0777620), PAR 0-7: [0777640, 0777660)

0.6.2 Control and status registers

MMR0 register (at 0777572)

The MMU is enabled if the low bit is set. The other bits are set when the MMU traps. They indicate the reason and identify the page that caused the trap.

mask	width	meaning
010000	1	write access and !W
004000	1	address not mapped; not in valid part of the page
002000	1	read or write access, and page not readable
000140	2	operation mode to which the page belongs
000020	1	if set, page is from data space (PDP−11/45, /70 only)
000016	3	page number
000001	1	enable MMU

MMR2 (at 0777576) is a status register that holds the address of an instruction that caused an MMU trap.

0.7 PDP-11/45 and PDP-11/70

0.7.1 Extensions of the Instruction Set.

Instructions that are illegal on the PDP-11/40 might be implemented by other PDP-11 models or by optional hardware extensions.

The MMU in the PDP-11/45/70 supports two mappings per operation mode. One is used for words addressed by the PC, i.e., instructions, the other for the rest. This feature is called "separate I/D space".

Floating point instructions are offered either by the "Floating Point Instruction Set" (FIS), a small extension, or by the "FP11", a full blown floating point processor.

The mtps/mfps instructions are only needed by models that cannot access the PSW, and therefore the PS, via the I/O page.

ass	read	code	implemented in
mfpt	move from processor type	000007	other models
spl	set IPL to m	00023m	PDP−11/45, PDP−11/70
csm	call supervisor mode	0070dd	PDP−11/45, PDP−11/70
tstset	test & set	0072dd	multiprocessor machines

wrtlck	write locked	0073dd	multiprocessor machines
FIS	floating pt. instr. set	075000−075777	optional for PDP−11/40
CIS	commercial instr. set	076000−076777	optional for other models
mtps	move to PS	1064dd	other models
mfpd	move from prev d−space	1065dd	PDP−11/45, PDP−11/70
mtpd	move to prev d−space	1066dd	PDP−11/45, PDP−11/70
mfps	move from PS	1067dd	other models
FP11	floating pt. processor	170000−177777	PDP−11/70, optionial for /45

0.7.2 6.1 The MMU of the PDP-11/45 and PDP-11/70

The CPUs with separate I/D space need another four sets of paging registers, namely PDRs and PARs for the kernel data space and for the user data space. Each of these sets is located in the I/O page below the corresponding set of the instruction space. Since these PDP-11s offer a third operation mode (supervisor mode), they also have another two sets of paging registers for this mode. (at [772200, 772300))

MMR1 (at 777574) identifies the registers changed by the incrementing / decrementing address modes in an instruction that caused an MMU trap. The increment is in the range [-2, +2] and encoded as a 3-bit integer which is negative in case of decrementing address modes.

The meanings of the bits are:

width	meaning
2	not used
3	increment from source a−mode
3	register from source a−mode
2	not used
3	increment from destination a−mode
3	register from destination a−mode

MMR3 (at 777516) is a control register used to enable separate I/D space and the CSM instruction, i.e. the supervisor mode.

The bits are assigned as:

width	meaning
10	not used
1	enable 22bit I/O addresses (PDP−11/70 only)
1	enable 22bit physical addresses (PDP−11/70 only)
1	enable csm
1	enable seperate kernel data space
1	enable seperate supervisor data space
1	enable seperate user data space

0.8 Assignment of Instruction Codes

"Illegal" means illegal on Lions' PDP-11/40

```
000000-000006   traps; halt, reset ...
000007-000077   illegal; mfpt
000100-000207   jump, rts
000210-000237   illegal; spl
000240-000377   arithmetic, boolean and data trnsfr; CC operations, swab bytes
000400-003777   branches
004000-004777   jsr
005000-006377   arithmetic, boolean and data transfer; single word operand
006400-006477   mark
006500-006777   arithmetic, boolean and data transfer; single word operand
007000-007777   illegal, csm, tstset, wrtlck
010000-067777   arithmetic, boolean and data transfer; two word operands
070000-073777   extended instruction set (EIS); mul, div, ash, ashc
074000-074777   boolean; XOR
075000-076777   illegal
077000-077777   sob
100000-103777   branches
104000-104777   traps; emt, sys
105000-106377   arithmetic, boolean and data transfer; single byte operand
106400-107777   illegal; mtps, mfpd, mtpd, mfps
110000-167777   arithmetic, boolean and data transfer; two byte operands
170000-177777   illegal; FP11
```