

PDP11

CS450 : Saelee

Programming the
PDP-11/10





vital stats

(first) 16-bit computer

64KB address space

64KB-4MB physical memory
(wow!)

8 general purpose registers

CISC-y ISA

PDP11 ISA

Lions' Commentary Section 2 (page 5)

just "bit", "cmp" and "tst" (whose stated function is to set the condition codes).

adc Add the contents of the C bit to the destination;

add Add the source to the destination;

ash Shift the contents of the defined register left the number of times specified by the shift count. (A negative value implies a right shift.);

ashc Similar to "ash" except that two registers are involved;

asl Shift all bits one place to the left. Bit 0 becomes 0 and bit 15 is loaded into C;

asr Shift all bits one place to the right. Bit 15 is replicated and bit 0 is loaded into C;

beq Branch if equal, i.e. if $Z = 1$;

bge Branch if greater than or equal to, i.e. if $N = V$;

bhi Branch if higher, i.e. if $C = 0$ and $Z = 0$;

bhis Branch if higher or the same, i.e. if $C = 0$;

bic Clear each bit to zero in the destination that corresponds to a non-zero bit in the source;

bis Perform an "inclusive or" of source and destination and store the result in the destination;

bit Perform a logical "and" of the source and destination to set the condition codes;

ble Branch if greater than or equal to, i.e. if $Z = 1$ or $N = V$;

blo Branch if lower (than zero), if $C = 1$;

bne Branch if not equal (to zero), i.e. if $Z = 0$;

br Branch to a location within the range (. -128, . +127) where "." is the current location;

clc Clear C;

clr Clear destination to zero;

cmp Compare the source and destination to set the condition codes. N is set if the source value is less than the destination value;

dec Subtract one from the contents of the destination;

div The 32 bit two's complement integer stored in rn and r(n+1) (where n is even) is divided by the source operand. The quotient is left in rn, and the remainder in r(n+1);

inc Add one to the contents of the destination;

jmp Jump to the destination;

jsr Jump to subroutine. Register values are shuffled as follows:

pc, rn, -(sp) = dest., pc, rn

mfpj Push onto the current stack the value of the designated word in the "previous" address space;

mov Copy the source value to the destination;

mtpj Pop the current stack and store the value in the designated word in the "previous" address space;

mul Multiply the contents of rn and the source. If n is even, the product is left in rn and r(n+1);

reset Set the INIT line on the Unibus for 10 milliseconds. This will have the effect of reinitialising all the device controllers;

rorg Rotate all bits of the destination one place to the right. Bit 0 is loaded into C, and the previous value of C is loaded into bit 15;

rts Return from subroutine. Reload pc from rn, and reload rn from the stack;

rtt Return from interrupt or trap. Reload both pc and ps from the stack;

sbc Subtract the carry bit from the destination;

sob Subtract one from the designated register. If the result is not zero, branch back "offset" words;

sub Subtract the source from the destination;

swab Exchange the high and low order bytes in the destination;

tst Set the condition codes, N and Z, according to the contents of the destination;

wait Idle the processor and release the Unibus until a hardware interrupt occurs.

The "byte" version of the following instructions are used in the file "m40.s", as well as the "word" versions described above:

bis	inc
clr	mov
cmp	tst

complex addressing modes:

direct: sp

indirect: (sp)

indexed: $4(sp)$

autoinc/dec: $(sp)+, -(sp)$

autoinc deferred: $*(r0)+$

“UNIBUS” architecture

memory-mapped I/O

e.g., MMU regs,
device status/command regs

```

5451 rkintr()
5452 {
5453     register struct buf *bp;
5454
5455     if (rktab.d_active == 0)
5456         return;
5457     bp = rktab.d_actf;
5458     rktab.d_active = 0;
5459     if (RKADDR->rkcs < 0) {          /* error bit */
5460         deverror(bp, RKADDR->rker, RKADDR->rkds);
5461         RKADDR->rkcs = RESET|GO;
5462         while((RKADDR->rkcs&CTLRDY) == 0) ;
5463         if (++rktab.d_errcnt <= 10) {
5464             rkstart();
5465             return;
5466         }
5467         bp->b_flags |= B_ERROR;
5468     }
5469     rktab.d_errcnt = 0;
5470     rktab.d_actf = bp->av_forw;
5471     iodone(bp);
5472     rkstart();
5473 }

```

```

5363 #define      RKADDR  0177400
5364 #define      NRK      4
5365 #define      NRKBLK   4872
5366
5367 #define      RESET    0
5368 #define      GO       01
5369 #define      DRESET   014
5370 #define      IENABLE  0100
5371 #define      DRY      0200
5372 #define      ARDY     0100
5373 #define      WLO      020000
5374 #define      CTLRDY   0200
5375
5376 struct {
5377     int rkds;
5378     int rker;
5379     int rkcs;
5380     int rkwc;
5381     int rkba;
5382     int rkda;
5383 };

```

Registers (16-bit)

- R0 - R4
- R5 (Frame Pointer - FP)
- R6 (Stack Pointer - SP)
- R7 (Program Counter - PC)

Processor Status Word

processor mode
kernel / user

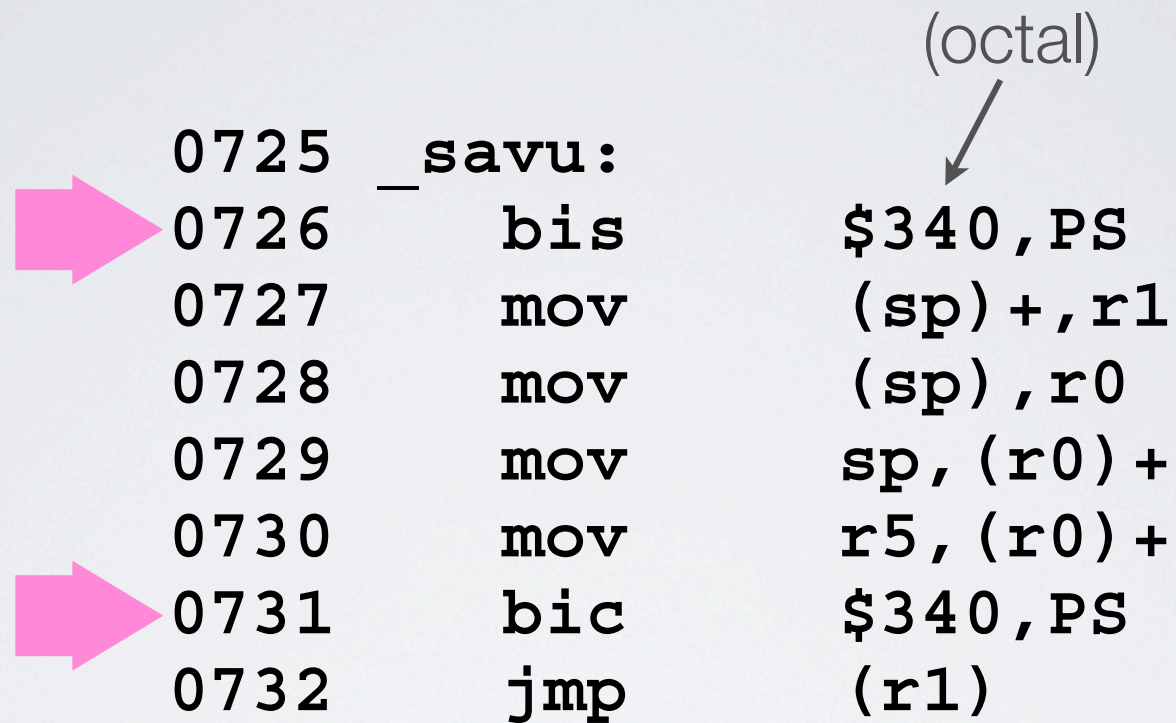
privileged instructions
separate MMU regs
separate SP reg

processor priority
0-7

inhibiting interrupts

(octal)
↓

```
0725  _savu:  
0726  _    bis    $340, PS  
0727      mov    (sp) +, r1  
0728      mov    (sp) , r0  
0729      mov    sp, (r0) +  
0730      mov    r5, (r0) +  
0731      bic    $340, PS  
0732      jmp    (r1)
```



status codes
(à la x86 FLAGS)

bits	description
14,15	current mode (00 = kernel;)
12,13	previous mode (11 = user;)
5,6,7	processor priority (range 0..7)
4	trap bit
3	N, set if the previous result was negative
2	Z, set if the previous result was zero
1	V, set if the previous result gave an overflow
0	C, set if the previous operation gave a carry

PDP11 MMU

16-bit words

16-bit “virtual” addresses

virtual memory: [0, 64KB)

8KB pages

mem-mapped I/O

160000

140000

120000

100000

060000

040000

020000

000000

physical memory limit:

PDP1 1/40: 256K (18-bit)

PDP1 1/70: 4MB (22-bit)

8KB page \rightarrow 128 \times 64-bytes

virtual page



physical block(s)

8 × { **p**age **a**ddress **r**eg
page **d**escription **r**eg

virtual address:



vpn

block #

offset



\oplus

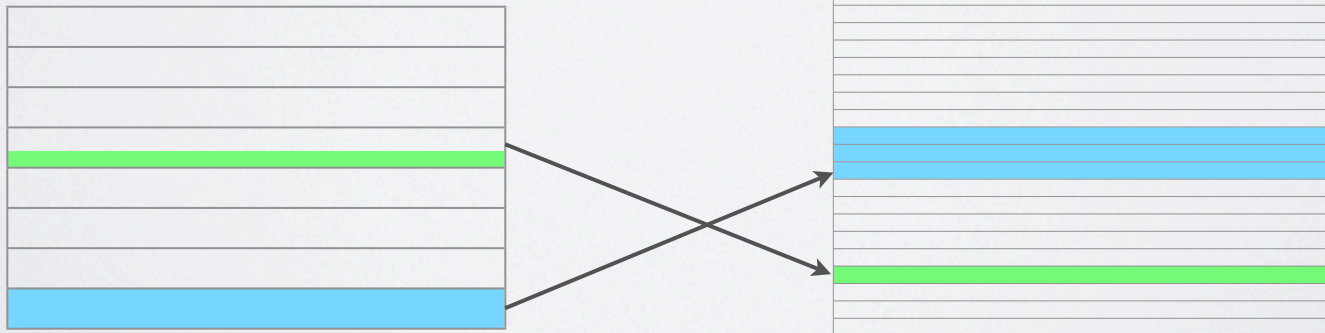
physical address:



PDR = size / direction

physical

virtual



separate PAR/PDR sets

kernel mode / user mode

KISA0 → 1st kernel PAR

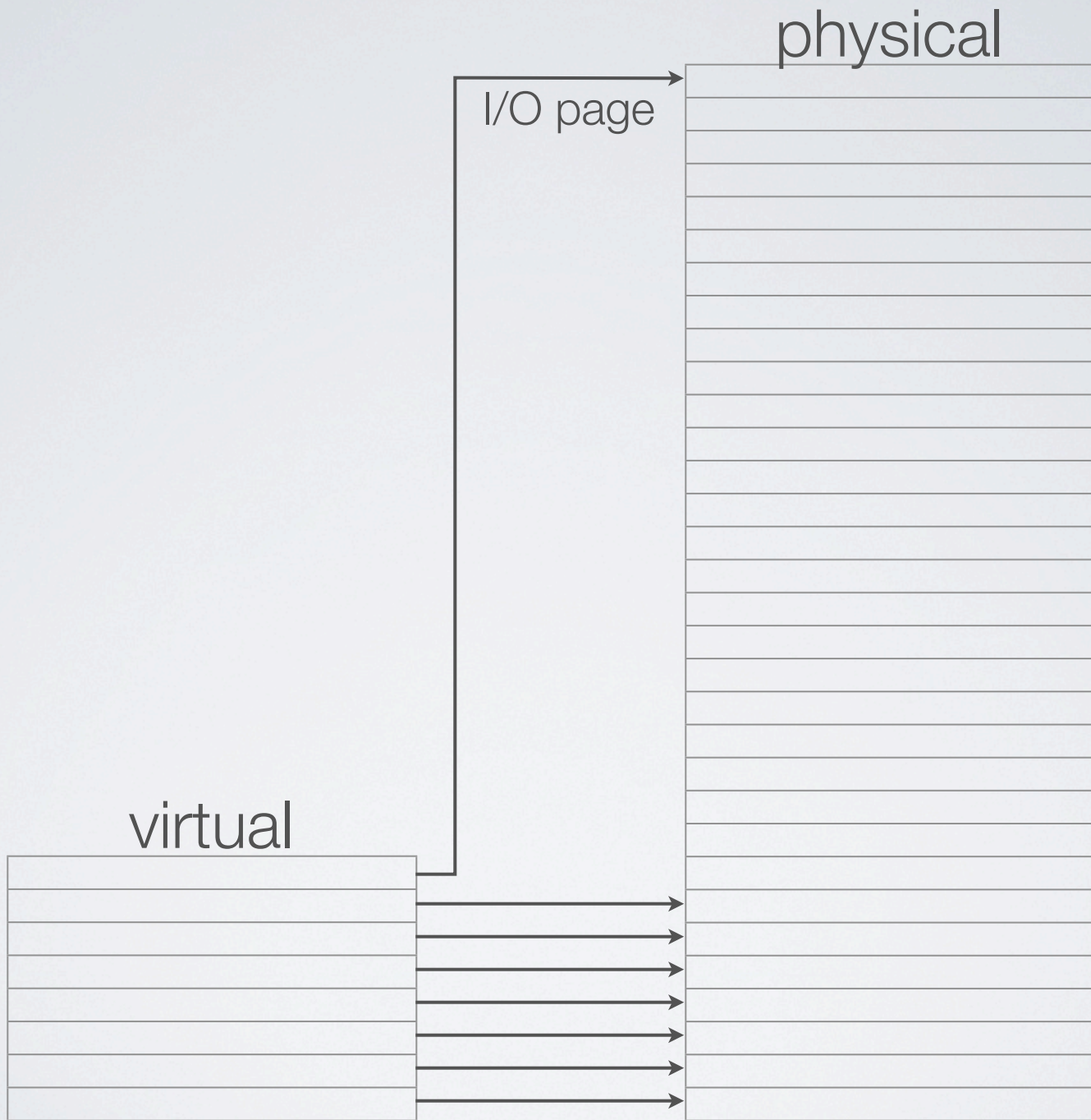
KISD0 → 1st kernel PDR

UISA0 → 1st user PAR

UISD0 → 1st user PDR

startup convention

(MM disabled)



SSR0 → MM on/off