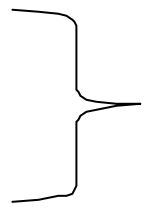


DEC pdp-11 family

- Performance range : 10:1
 - architectural goals: (1970)
 - address space: 64K (boo!)
 - dense code (Mp expensive)
 - easy interrupt processing
 - re-entrant code
 - easy peripheral interfacing
- 
- stacks

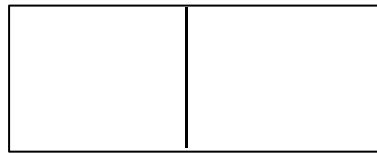
pdp-11 ISP

- 8 General Purpose Registers (GPRs)
- instructions taking 0,1 or 2 operands
- symmetric instruction set
- nearly-orthogonal instruction set
 - easier for compilers (and humans)

pdp-11 cpu cycles

- Fetch instruction cycle
- source operand cycle
- destination operand cycle
- Execute
- honour interrupts

Address generation



MODE



INDIRECTION



REGISTER R_n

CF:



4

4

12

CS 350

360

pdp-11 best feature: addressing modes

- 00 Rn contains operand (register mode)
- 01 Rn contains ptr to operand; (autoincrement)
[increment Rn AFTER operand fetch]
- 10 Rn contains ptr to operand; (autodecrement)
[decrement Rn BEFORE operand fetch]
- 11 Add $c(Rn)$ to $c(\text{nextword})$ to get operand address

Modes: single operand instruction

INC	R3	00 0	$R3 \leftarrow C(R3) + 1$
INC	(R3)	00 1	$C(R3) \leftarrow C(C(R3)) + 1$
INC	(R3)+	01 0	As above, then bump R3
INC	@(R3)+	01 1	register points to address, then increment
INC	GEO(R3)	11 0	$GEO + c(R3)$ is address
INC	@GEO(R3)	11 1	$GEO + c(R3)$ points to address

Double operand:

MOV R1,R2	00 0 00 0	R2 <- C(R1)
(R1), R2	00 1 00 0	R2 <- C(C(R1)) “LDA”
R2, (R1)	00 0 00 1	C(R1) <- C(R2) “STA”
(R1), (R2)	00 1 00 1	C(R2) <- C(C(R1)) “MOV”

Double operand

MOV R1, ARRAY(R2) 00 0 11 0
ARRAY + C(R2) <- C(R1)
INDEXED STORE

ARRAY(R2), R1 11 0 00 0
INDEXED LOAD

ARRAY(R2), VEC(R1) 11 0 11 0
DOUBLY INDEXED

TOM, GEORGE 11 0 11 0
INDEXED *RELATIVE TO PC*

@TOM, @GEORGE 11 1 11 1
AS ABOVE, AND INDEXED

TERRIFYING TIMING

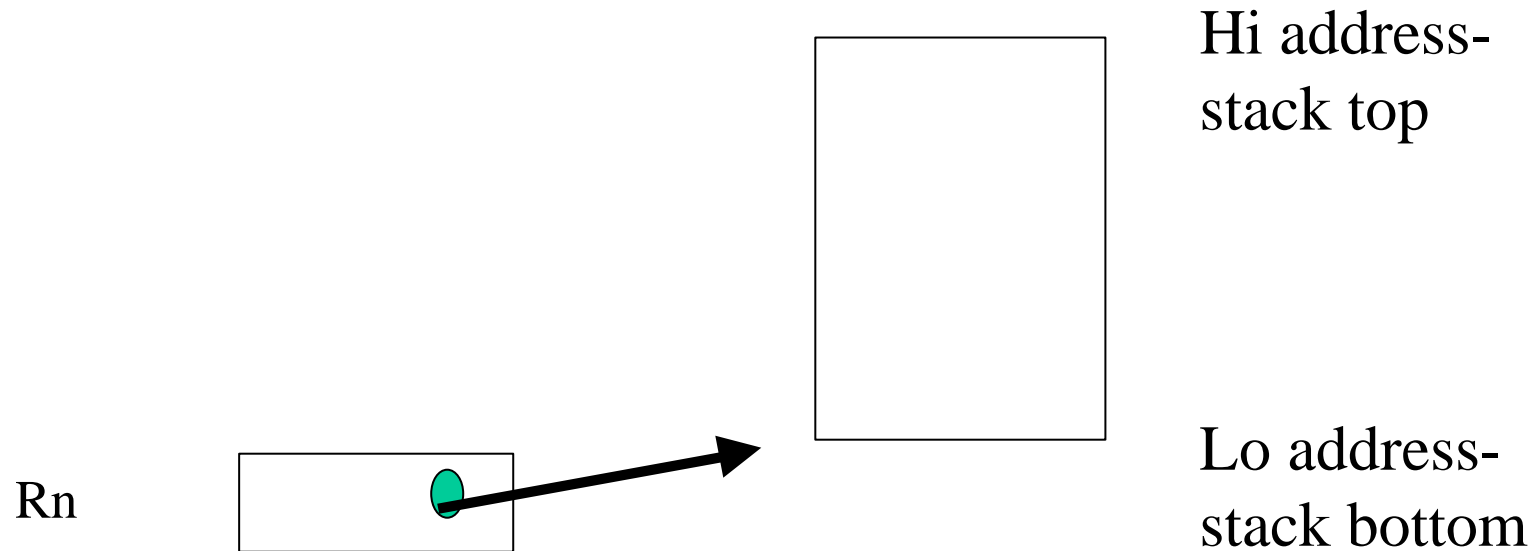
```
MOV @TOM, @GEORGE
```

REQUIRES

14 MICROSECONDS 11/20

5.6 MICROSECONDS 11/40

Stacks!



MOV ITEM, -(Rn)	PUSH ITEM ON STACK
MOV (Rn)+, ITEM	POP STACK TO ITEM

CONDITIONAL BRANCHES

ON A 4-BIT CONDITION CODE

RANGE: (-128, + 127)

DATA TYPES

WORD OR BYTE

NEARLY EVERY INSTRUCTION HAS TWO VERSIONS

MOV MOVB
INC INCB

NEARLY EVERY ADDRESSING MODE WORKS WITH EVERY
INSTRUCTION

R7 IS THE PROGRAM COUNTER

WORKS FOR ALL VALUES OF MODE & INDIRECT BITS

BEST ONES ARE:

01 0 R7 CONTAINS POINTER TO OPERAND

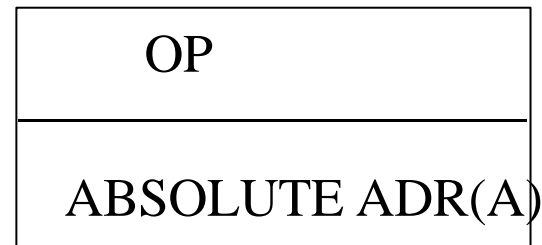
OP #N ASSEMBLES TO

OP
N

R7 IS THE PROGRAM COUNTER

01 1 R7 CONTAINS PTR TO PTR TO OPERAND:

OP @#A



PHBREAK:

**RISC ISP architecture
the MIPS ISP**

you read:

text Chapter 3

Summary of main points:

Two objectives;

1] Describe the MIPS ISP architecture

2] expose the Reduced Instruction Set Computer (RISC)
approach to architecture

RISC approach: what it is not:

CISC a la S/360, VAX (1970s)

M_p is slow

(no caches,

cycle times of 1-6 microseconds

[vs. 100 nsec = 0.1 microsec today])

so instruction fetches are expensive,
so let's make every instruction do a lot

let's mimic higher-level constructs, eg

¶ loop control (S/360 BXLE)

¶ stack push/pop (Burroughs B-5000, VAX)

¶ procedure call instruction (VAX)

"wired macroinstructions"

in general, lots of side-effects per instruction\

{ we can implement these easily (for free?),
by writing long microroutines in vertical microstore }

What happened?

seemed OK thru the 1970s, but in the 80s

¶ M_p got a lot faster, esp. with caches

RISC

- Microstore became as slow as Mp
- People needed to use compilers
 - compilers couldn't always generate efficient CISC code

RISC

- Programmers spent pages setting up a killer effect so
- code was hard to understand or modify
- solution: a form of KISS:
- Reduced Instruction Set Computer

RISC

RISC approach: what it is: Rationale

Reduced (small) set of simple instructions

¶ able to be used effectively by compilers

get rid of the slow microprogram store

i.e. instructions implemented by wired-logic controls

RISC

wired-logic decoders will be feasible and fast,
as the instructions are simple and few in number

programs will have more instructions, but
 M_p is now big (>1 Mbyte) and fast (<100 nsec)

RISC

RISC Empirical result:

In executing (e.g.) compiled C code

the product

(# of instrs executed) * (mean execution time per instruction)

is usually smaller for RISC than for CISC

RISC

the simpler control design was amenable to VLSI
(single-chip cpus) so

the microprocessor world (MIPS, SPARC, PowerPC)
is now all RISC

except Intel and Motorola 68X00

but it could all change tomorrow.

MIPS architecture

(note simplicity w r to S/360, VAX)

ALL instrs have exactly 3 operands (KISS)

there are just 32 fast registers, \$0 - \$31.

$c(\$0) = 0$, always.

