IBM

System 360/370/390

Section 3.2 CISC

IBM S/360

- 370 is 360, re-implemented , plus a few new features
- 390 is 370, re-implemented , plus a few new features
- still a mainstay of IBM’s business

S/360

- Instroduced in 1964 to replace three (!) incompatible families
  - 7090 fixed-point word oriented
  - 1401 decimal character-oriented
  - 1620 decimal numeric oriented
- with one architecture

S/360

- Instruction set a UNION of instruction sets
  - fixed-point binary (7090)
  - floating binary (7090)
  - decimal character-oriented (1401)
  - string processing
- 4 hardware data types
  - byte, halfword, word, doubleword

S/360

- One OS (well, 5)
  - OS/360 big batch OS, no virtual memory
  - DOS/360 little batch OS, no virtual memory
  - DAMPS real time (360/44)
  - VM virtual memory for 360/67, 370s
  - TSS failure, virtual memory for 360/67

S/360

- 8 models with almost identical ISP
  - (the 360 ISP)
- performance range of 300:1 (!)
- microprogrammed CPUs in Models 20 - 67
- microstore times of 200 nsec - 1 μsec
- data bus widths of 1 - 8 bytes
- memory cycle times of 7.2 μsec - 0.75 μsec

Models

- 20 25 30 40 44 50 65 67 75 85 91
- Anomalies:
  - 44 for real time process control
  - 67 to compete with GE 645 in timesharing market (paging, segmentation, swap drums, unique OS)
  - 25 had user-alterable control store

Prices:

- Model 40 with 250 Kbytes Mp, 3 μsec cycle time: $40 000/month (not sold)
- model 75: $100 000 / month
- model 91 (pipelined): $10M

360 ISP design

- Previous large machines:
  - addressed 32K cells of Mp maximum
  - cell = word of length 36 bits or so
  - registers: AC, MQ, few Index Registers
  - fixed length instructions & operands

Op	IR	address

360 ISP goals

- Bigger address space (2^{24})
  - cf. Motorola 68000 of 25 years later
- General Purpose registers and more of them
  - 16
- use program store more efficiently
  - variable length instructions

360 ISP decisions

- Registers: 16 GPRS, useable as
  - accumulators
  - MQs
  - index registers
  - base registers

Instruction length

- 2 - 6 bytes

Address Formation

- How to avoid storing 24 bits of real address per instruction?

- Use locality of reference principle
  - next memory reference likely to be “close” to the last reference

Address Formation

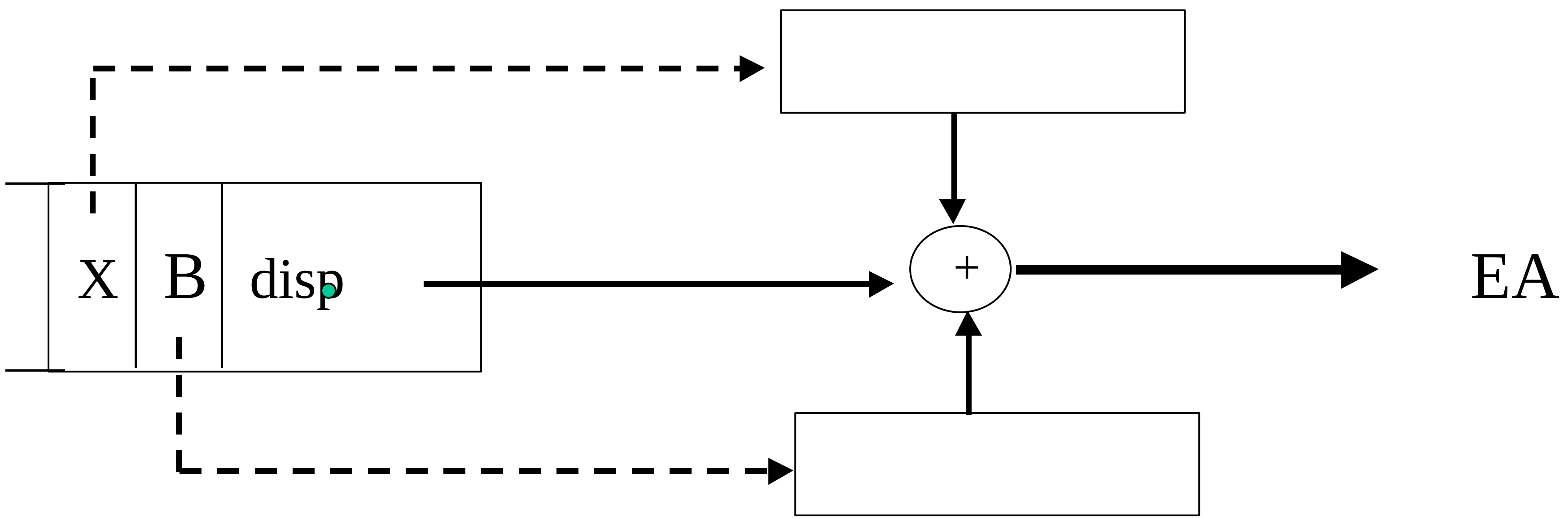
- Use a GPR (32 bits) to point to the general vicinity of the desired cell
- use a small address (12 bits) in the instruction to hit the precise location

Address Formation

- How to avoid storing 24 bits of real address per instruction?
  - Instruction holds a 12-bit field (spans 4096 bytes) called *displacement*
  - 24- bit memory address formed from displacement, c(base register [GPR]) & c(index register [GPR]) as follows:

Address formation

c(base register j) + disp + c(Xi) -> EA



Good & Bad

- What's good?
  - Uses codespace efficiently IF locality of reference is valid

- What's bad?
  - Need to make a gpr point within 4096 of address A before we can access it ("establishing addressability)

Good & Bad

- Base registers are NOT invisible to the programmer , do the OS can NOT use them for program relocation (blunder)

- tends to tie up many GPRs

Program relocation

- Newer models had programmer-invisible relocation registers (DAT box, MMU) in addition to the above
  - 360/67, all 370s, MIPS chip, . . .
- Older 360s could not relocate programs or data (!)
- BTW, MIPS uses a 64-bit address space

360 Critique (1998)

- Not enough address space!
  - 2^{24} bytes insufficient
- no virtual address spaces
  - program & data relocation impractical
  - batch throughput oscillated
- not enough GPRs (16)
- inadequate interrupt structure

However

- 360 redefined computer architecture
  - Gerritt Blauuw: “the end of architecture”

Instruction Formats

- Goals:
  - flexibility
  - efficient use of code space (Mp was expensive in 1962)

Register-Register (RR)

8	4	4	
OP	R1	R2	2 bytes

* e.g. AR R3, R4

*short

* can’t reference memory

Register - Storage, indexed RX

8	4	4	4	12
op	R1	X2	B2	D2

Storage address

* e.g   L R5, GEO(R3)
*twice as long as RR format

Register - Storage, unindexed RS

8	4	4	4	12
op	R1	R3	B2	D2

RS format

Storage address (B2, D2)

E.g LM R1, R6, SUE

no more indexing, 2nd register can be specified

Storage- Storage
SS

8	8	4	12	4	12
op	L	B1	D1	B2	D2

ADDRESS ADDRESS

E.G. MVC 12, TOM, SUE

360 INSTRUCTION FORMATS

- Note lack of orthogonality of opcode space to format space --
    - not all opcodes work with all formats

- Note lack of symmetry:
  - OP(A,B) ~=> OP(B,A)

360 INSTRUCTION DESIGN

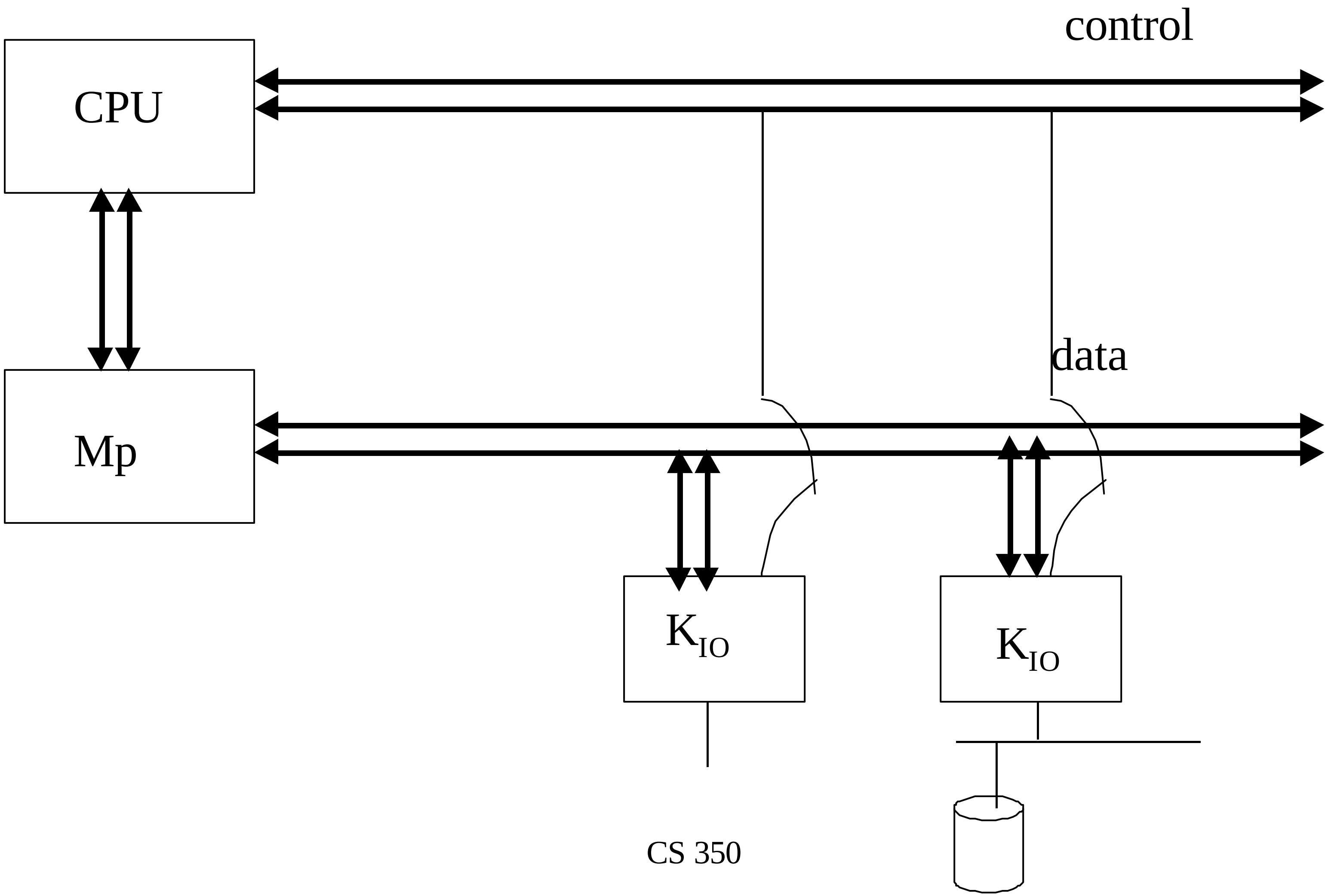
- 256 opcodes, many are spares

- *FOUR* instruction sets
  - fixed-point binary arithmetic & logic
  - floating-point binary
  - decimal
  - miscellany (protection, I/O)
  - “machine is a union of machines”

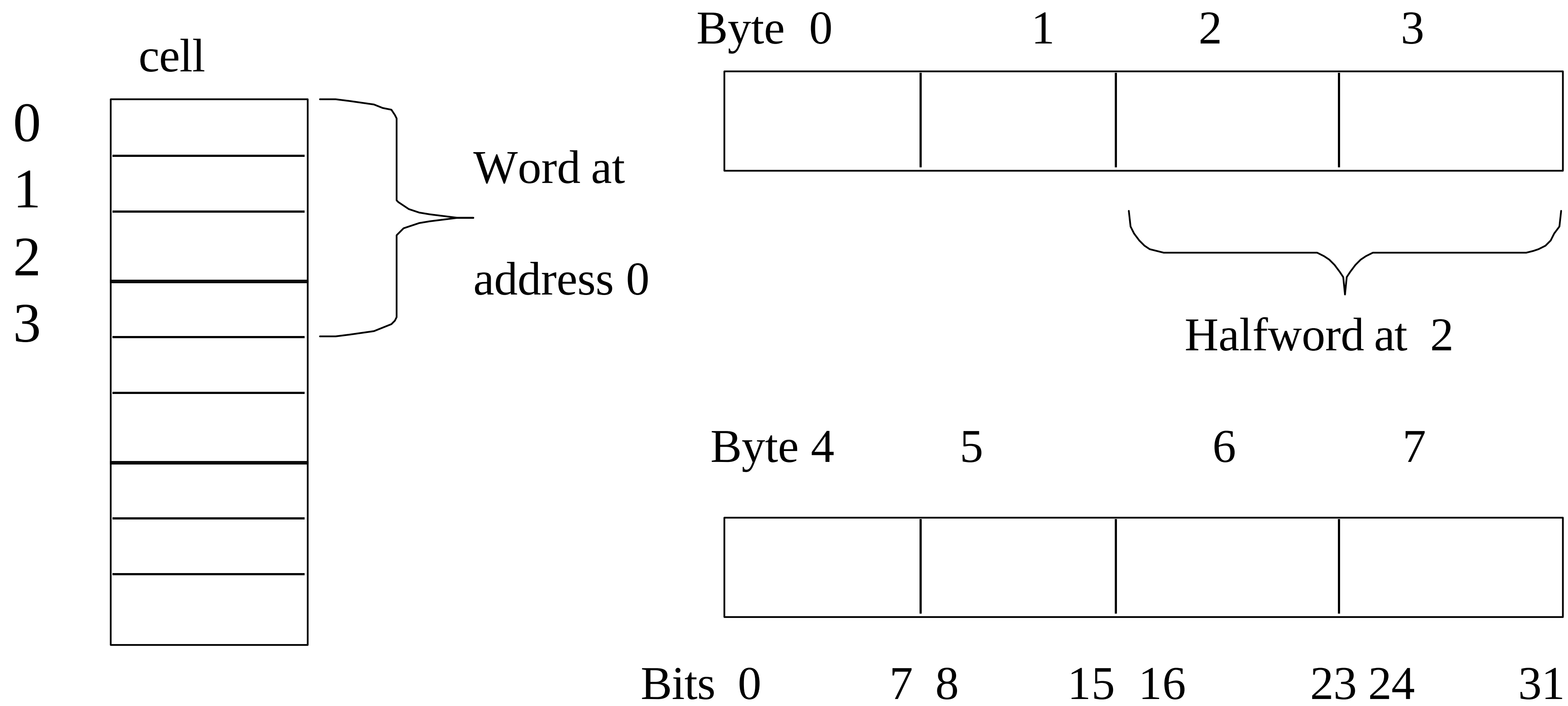
360 INSTRUCTION DESIGN

- E.G.
  - S R3, GEORGE(X2); BINARY RX
  - AP TOM, GEORGE; DECIMAL SS
  - CLI SUE, C’X’ ; COMPARE IMMEDIATE
  - SSK R1, R2 ;   PROTECTION
    - 5-bit key in PSW must match key of memory block - grossly inadequate

360 Bus Structure



360 Storage structure



Little-endian order -- is this natural?

360 Storage structure

- Halfword addresses = 0 mod 2
- word addresses = 0 mod 4
- doubleword addresses = 0 mod 8
  - Why?

360 Implementations

- How to get a performance range of 300:1 using ONE logic family?

Model 40

- ALU: 1 byte wide
- microinstruction time: 625 nsec
- Mp cycle: 2.5 μsec
- max Mp: 0.25 Mbyte
- rent: $20 000 / month

Model 50

- ALU: 32 bits wide
- microinstruction time: 500 nsec
- Mp cycle: 1.5 μsec
- max Mp: 0.5 Mbyte
- interrupt response: < 600 μsec
- $30 000 / month

- 1963

Motorola 68000

- microinstr time: 250 nsec
- Mp cycle: 0.5 μsec
- max Mp: 16 Mbyte
- $200

- 1983

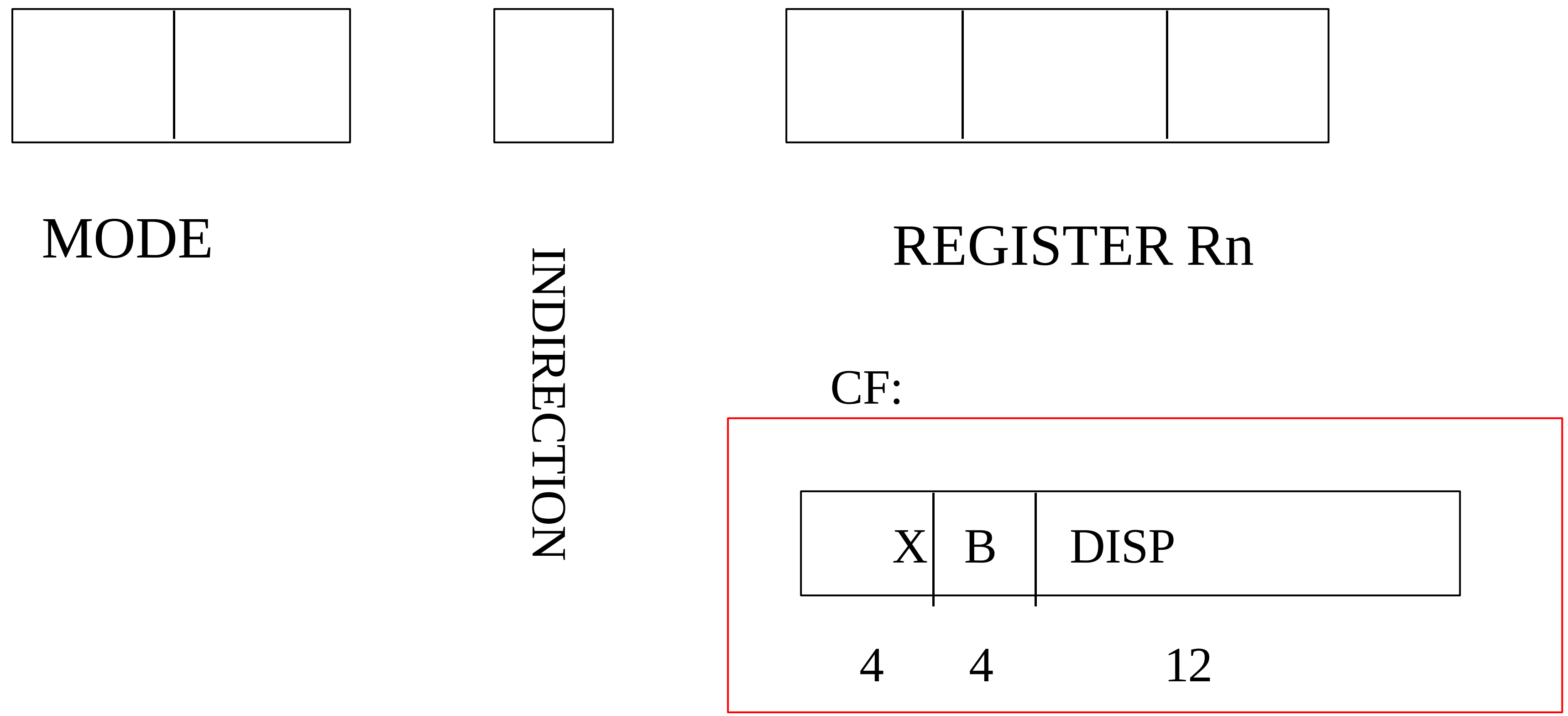
pdp-11 ISP

- 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 intruction cycle
- source operand cycle
- destination operand cycle
- Execute
- honour interrupts

Address generation



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(nextword) to get operand address

Modes: single operand instruction

INC	R3	00 0	R3 <- C(R3) + 1
INC	(R3)	00 1	C(R3) <- 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

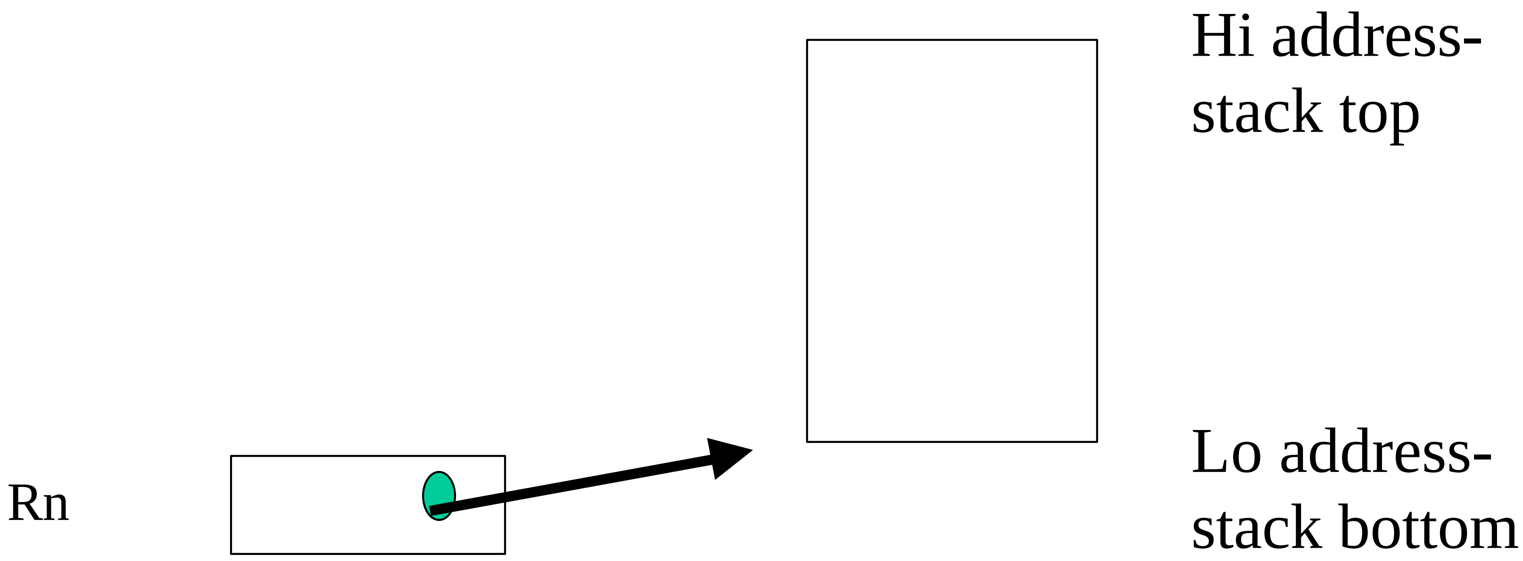
MOV @TOM, @GEORGE

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

OP
ABSOLUTE ADR(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 contructs, 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

- Microstore became as slow as Mp

- People needed to use compilers
  - compilers couldn’talsways generate efficient CISC code

- Programmers spent pages setting upa killer effect so

- code was hard to understand or modify

- solution: a form of KISS:
- Reduced Instruction Set Computer

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

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 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 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.

2^{30} memory cells,

4 bytes wide and byte addressed.

¶ *Aligned* word data begin at byte adresses of form 4n.

¶ Index registers must be incremented by 4 when addressing word data.

ALL instructions 32 bits (1 word) long