

Instruction Set Principles

(Appendix B)

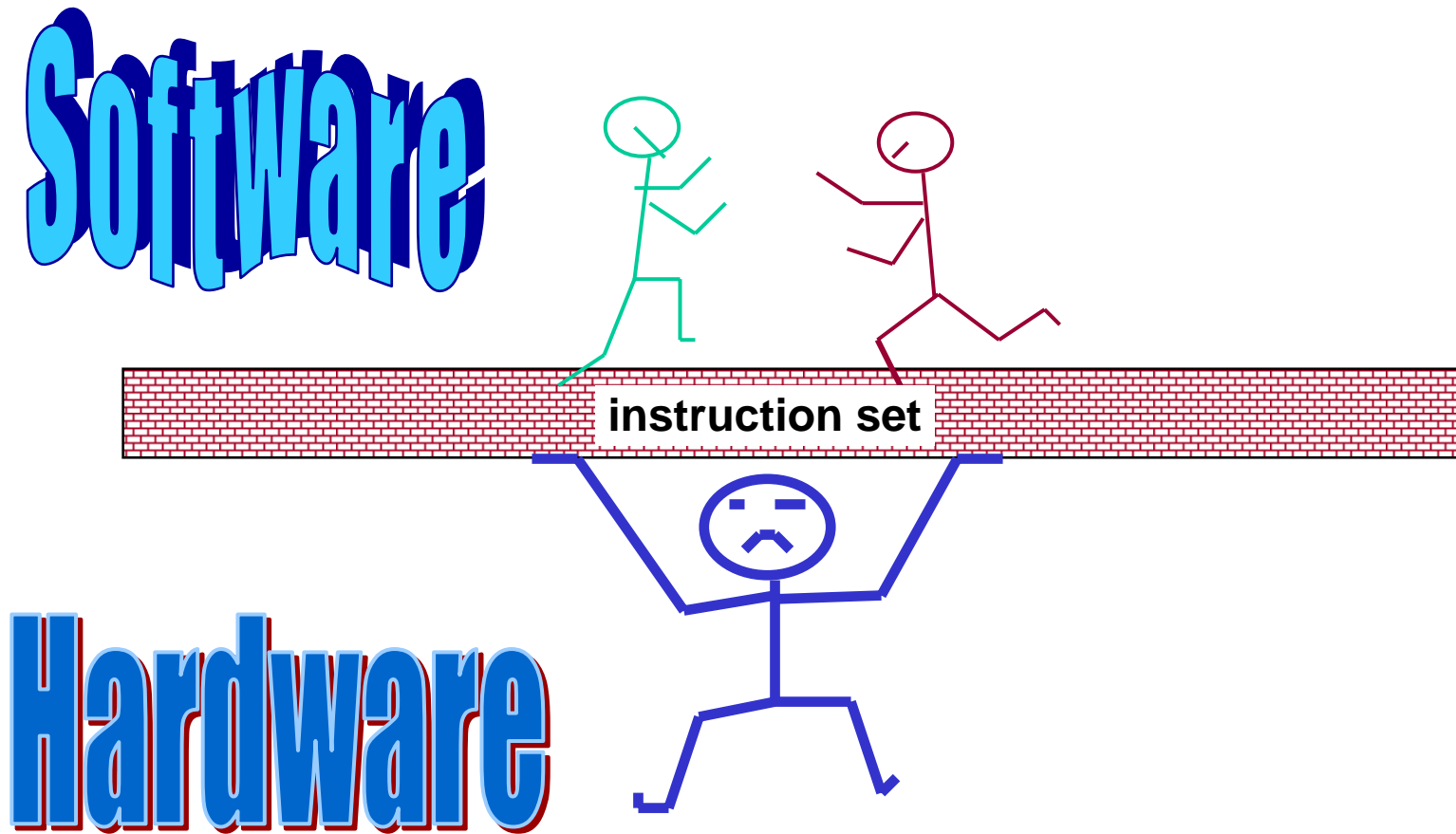
Outline

- **Introduction**
- **Classification of Instruction Set Architectures**
- **Addressing Modes**
- **Instruction Set Operations**
- **Type & Size of Operands**
- **Instruction Set Encoding**
- **Size of Register File**
- **CISC vs. RISC**
- **DLX Architecture**

Computer Architecture's Changing Definition

- **1950s to 1960s:** Computer Architecture Course
Computer Arithmetic
- **1970s to mid 1980s:** Computer Architecture Course
Instruction Set Design, especially ISA appropriate for compilers
- **1990s:** Computer Architecture Course
Design of CPU, memory system, I/O system, Multiprocessors
- **2000s:** Computer Architecture Course
Design of CPU (Performance & Power), Memory system, I/O, embedded systems, wireless, multi-core systems.

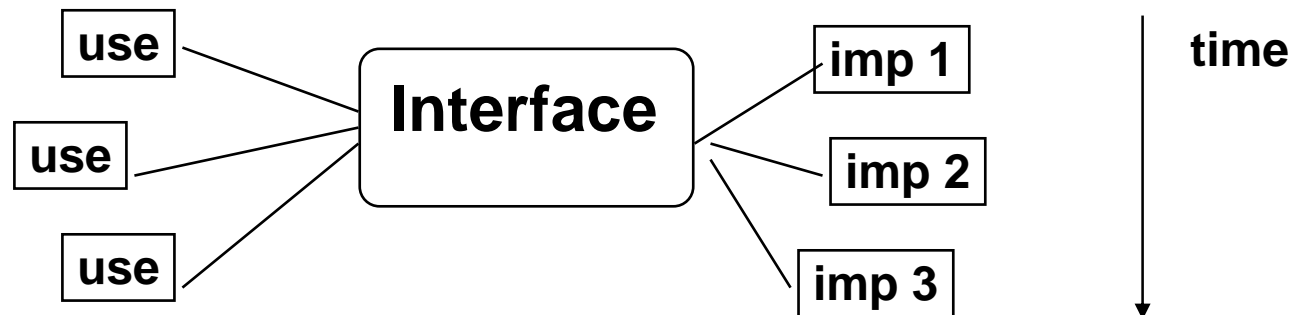
Instruction Set Architecture (ISA)



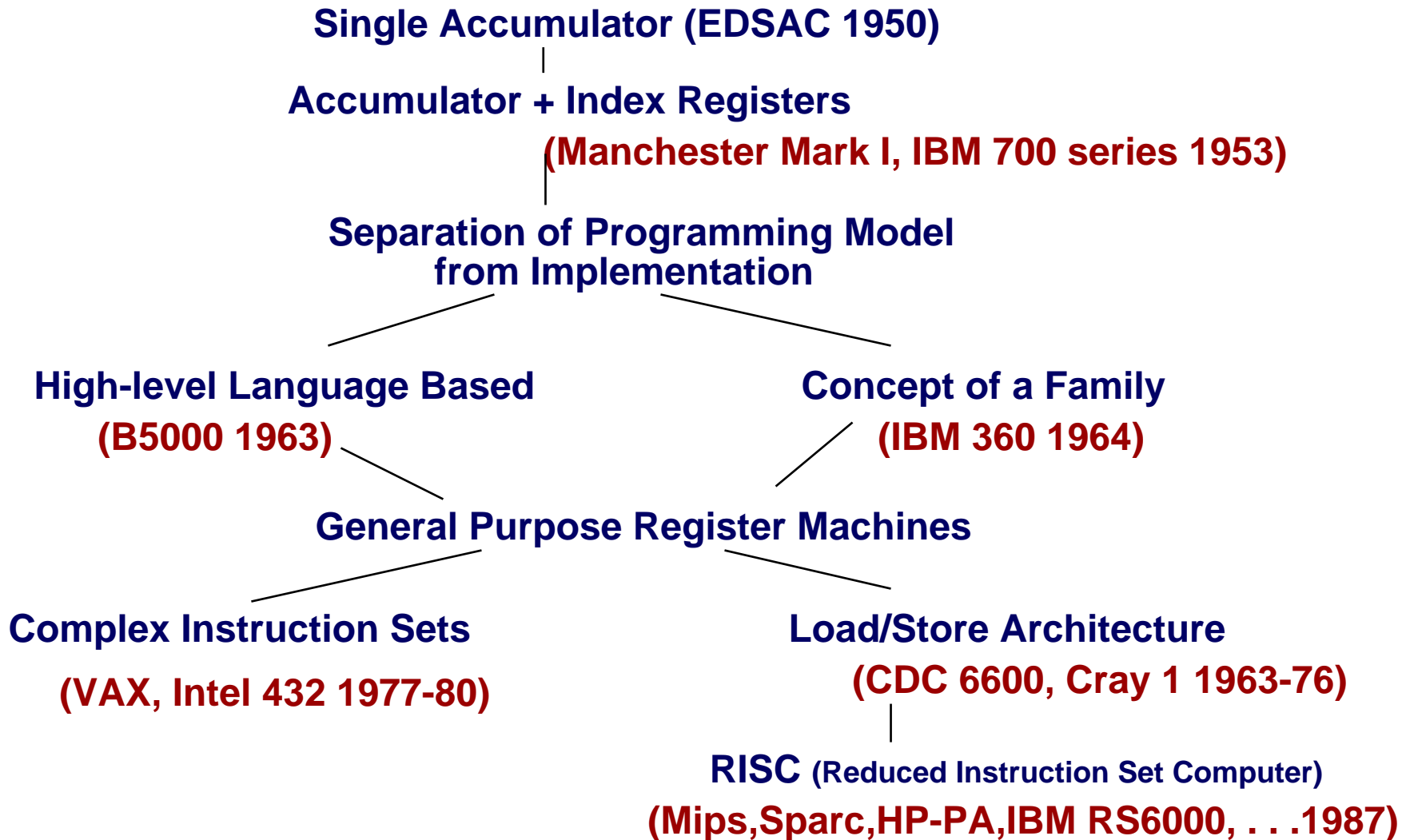
Interface Design (ISA)

A good interface:

- Lasts through many implementations (**portability, compatibility**)
- Is used in many different ways (**generality**)
- Provides **convenient** functionality to higher levels
- Permits an **efficient** implementation at lower levels



Evolution of Instruction Sets



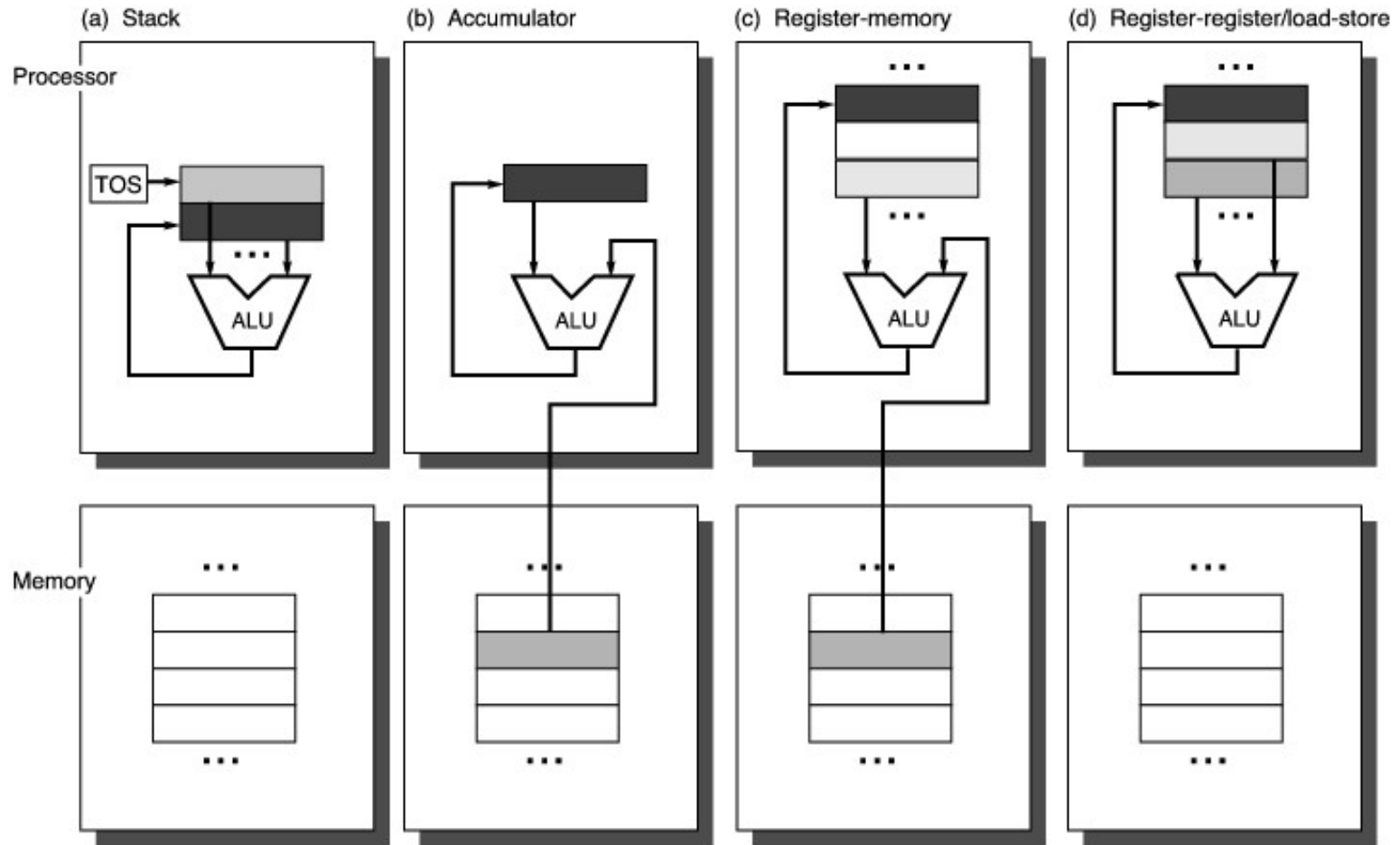
Evolution of Instruction Sets

- Major advances in computer architecture are typically associated with landmark instruction set designs
 - Ex: Stack vs GPR (System 360)
- Design decisions must take into account:
 - technology
 - machine organization
 - programming languages
 - compiler technology
 - operating systems
- And they in turn influence these

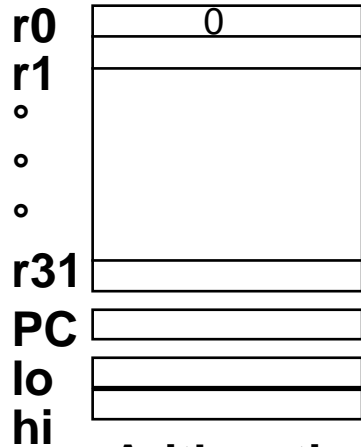
Classification of ISAs

- **Stack Architecture**
 - Only data structure: **Stack**
- **Accumulator Architecture**
 - Only one register: **Accumulator**
- **General Purpose Register (GPR) Architecture**
 - **Set or Registers (Register File)**
 - **3 sub-architectures:**
 - **Reg-Reg (Load/Store)**
 - **Reg-Mem or Mem-Reg**
 - **Mem-Mem**

Operand location for ISAs



Example: MIPS



Programmable storage

2^{32} x bytes

31 x 32-bit GPRs (R0=0)

32 x 32-bit FP regs (paired DP)

HI, LO, PC

Data types ?

Format ?

Addressing Modes?

Arithmetic logical

Add, AddU, Sub, SubU, And, Or, Xor, Nor, SLT, SLTU,
AddI, AddIU, SLTI, SLTIU, AndI, OrI, XorI, *LUI*
SLL, SRL, SRA, SLLV, SRLV, SRAV

Memory Access

LB, LBU, LH, LHU, LW, LWL, LWR
SB, SH, SW, SWL, SWR

Control

J, JAL, JR, JALR
BEq, BNE, BLEZ, BGTZ, BLTZ, BGEZ, BLTZAL, BGEZAL

32-bit instructions on word boundary

Data Format / Alignment

Data formats:

Bytes, Half words, words and double words

Some issues

- *Byte addressing*

Big Endian
vs. Little Endian

0	1	2	3
3	2	1	0

- *Word alignment*

Suppose the memory is organized in 32-bit words.

Can a word address begin only at 0, 4, 8, ?



Instruction Types

- Need to decode type before rest of fields
- Can decode all types in parallel and select after decoding type, but this is costly if more than a few types
- If type specifies execution unit, then can pipeline (dispatch then decode)

How many types?

- 2 (1 bit) forces unrelated operations into the same type
- 4 (2 bits) corresponds to arithmetic, load/store, jump, other (w/ subtypes)
- 8 (3 bits) can add FP/ integer, jump/ branch, immediate types
- More bits in type = fewer other bits

Intel Operands / Instructions

- 8, 16, 32, 48, 64, or 80 bits
- Strings
- One byte to 12 bytes
- First byte(s) contains opcode, mode, register fields
- Later bytes contain addresses or immediate values

Instruction Set Encoding

- Instructions generally contain:
 - Instruction type
 - Address(es) of operand(s) or target
 - Operation code
- One type may have immediate data

Variety of Instruction Formats

- **Two address formats:** the destination is same as
 - one of the operand sources

(Reg x Reg) to Reg

$$R I \leftarrow (R I) + (R J)$$

(Reg x Mem) to Reg

$$R I \leftarrow (R I) + M[x]$$

x could be specified directly or via a register;
effective address calculation for x could include index,
indirect, ...

- **Three operand formats:** One destination and up to two operand sources per instruction

(Reg x Reg) to Reg

$$R I \leftarrow (R J) + (R K)$$

(Reg x Mem) to Reg

$$R I \leftarrow (R J) + M[x]$$

- **many different possible formats !**

Some Problems

- **Should all addressing modes be provided for every operand?**
 - ⇒ *regular vs. irregular instruction formats*
- **Separate instructions to manipulate**
 - Accumulators**
 - Index registers**
 - Base registers**
 - ⇒ *A large number of instructions*
- **Instructions contained implicit memory references**
 - several contained more than one**
 - ⇒ *very complex control*

Variable Length Instructions

- Make it difficult to decouple memory fetches
- Fetch part, then decide whether to fetch more, and maybe miss in cache before instruction is completed.
- Fixed length allows full instruction to be fetched in one access

Impact of Increasing Register Count

- Reduces memory references
- Increases instruction word size
- Greater demand on compiler to schedule registers
- Increases cost (time) for context switching

Necessary Registers

- Stack machines (0- address) use none
- Accumulator machines (1- address) use 1
- Register bank machines (2- address or 3- address) use N
- Memory- memory machines (3- address) need 0, use N

Sufficient Registers

- Most arithmetic expressions can fit in 8 registers
- 16 are almost always enough for a basic block or function body
- More than 16 enables interprocedural register scheduling

Evolution of Register Sets

- Early machines could afford just one register (accumulator)
- Added specialized registers (accumulator extension, index, etc.)
- Specialized registers cost as much as any register, but limited in use
- More efficient to make registers general purpose, simpler scheduling

Scalability of General Registers

- Adding more alters instruction set in a straightforward manner
- Much harder to scale up specialized registers
- General registers lead to orthogonal and regular instruction sets

Inevitable Special Cases

- Program counter
- Configuration
- Program status
- etc.
- Sometimes accessed by special instructions
- Sometimes mapped to GP registers

Intel x86 (IA32)

- EAX, EBX, ECX, EDX, ESI, EDI, EBP,
- ESP (general purpose)
- First four can be 16 or 32 bits, lower 16 bits
- are divided into named bytes
- CS, SS, DS, ES, FS, GS (segment)
- Instruction pointer (16/ 32)
- Flags

Intel x86 (IA32) Other

- Control registers CR0 - CR4
- Floating point R0 - R7 (80 bits)
- FP Control, Status (16 bits each)
- Tag word (2 bits/ FP reg: 16 total)
- FP Instruction pointer (48 bits)
- FP Data pointer (48 bits)
- Debug D0 - D8, first 5 are breakpoint, other 4 control, status

PowerPC

- General Purpose GPR0 - GPR31
- Link (32 bits)
- Condition (32 bits)
- Count (32 bits)
- Exception (32 bits)
- Real- time clock (64 bits)
- Floating point FPR0 - FPR31 (64)
- FP status and control (32 bits)

CISC vs. RISC

- First computers were RISC
- Cray supercomputers were RISC
- Complexity was added in the 70s and 80s
- Return to simplicity in 80's and 90's

Why Return to RISC?

- CISC provides too many possibilities
- Compilers can't choose optimal encoding
- 70 instructions = 99% of code
- 50 instructions = 95% of code

Why Did CISC Happen?

- Fetch- Execute cycle
- Limited memory
- Reduce fetches and memory by packing multiple ops into each instruction
- Observe common sequences of ops and turn them into instructions

CISC Features

- Memory accesses and address arithmetic are tightly bound to instructions
- Rely on few registers, more memory references
- Note that memory hasn't kept pace with processor clock rate

Reduced Instruction Set Computers

(Cocke, IBM; Patterson, UC Berkeley; Hennessy, Stanford)

- **Compilers have difficulty using complex instructions**
VAX: 60% of microcode for 20% of instructions, only responsible for 0.2% execution time
IBM retargets 370 compiler to use ISA subset - generated code faster!
- **Simple instruction sets do not need microcode**
Use fast memory near processor as cache, not microcode storage
- **Design ISA for simple pipelined implementation**
 - **Fixed length, fixed format instructions**
 - **Load/store architecture with up to one memory access/instruction**
 - **Few addressing modes, synthesize others with code sequence**
 - **Register-register ALU operations**
 - **Delayed branch**

Benefits of RISC

- Reduced CPI
- Reduced decoding delay
- Simpler core design enables more chip area to be used for performance
- But today, CISC architectures like Intel use a RISC core for a subset of instructions
- Why are RISC designs still faster?

Code Expansion

- RISC does less with each instruction
- Large code size (1.3 to 1.6 X)
- Larger number of memory fetches
- Partly alleviated by larger cache
- Still generates more memory traffic than CISC
- Compressed instruction blocks?

Common RISC Features

- Load/ Store designs
- Few addressing modes
- Fixed instruction size
- Few instruction formats
- Few operand sizes
- Use more registers, separate memory operations

MIPS R2000

(One of first commercial RISCs, 1986)

- **Load/Store architecture**
 - **32x32-bit GPR (R0 is wired), HI & LO SPR (for multiply/divide)**
 - **74 instructions**
 - **Fixed instruction size (32 bits), only 3 formats**
 - **PC-relative branches, register indirect jumps**
 - **Only base+displacement addressing mode**
 - **No condition bits, compares write GPRs, branches test GPRs**
 - **Delayed loads and branches**
- **Five-stage instruction pipeline**
 - **Fetch, Decode, Execute, Memory, Write Back**
 - **CPI of 1 for register-to-register ALU instructions**
 - **8 MHz clock**
 - **Tightly-coupled off-chip FP accelerator (R2010)**

RISC/CISC Comparisons

- **R2000 vs VAX 8700 [Bhandarkar and Clark, '91]**
R2000 has ~2.7x advantage with equivalent technology
- **Intel 80486 vs Intel i860 (both 1989)**
Same company, same CAD tools, same process
i860 2-4x faster - even more on some floating-point tasks
- **DEC nVAX vs Alpha 21064 (both 1992)**
Same company, same CAD tools, same process
Alpha 2-4x faster

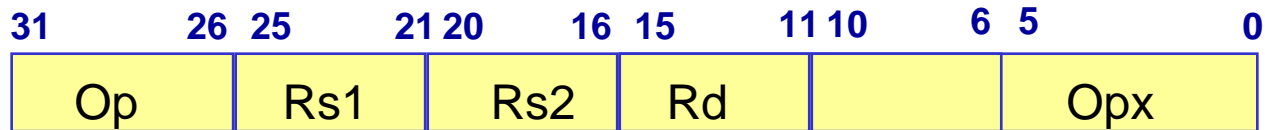
A "Typical" RISC

- 32-bit fixed format instruction (3 formats)
- 32 32-bit GPR (R0 contains zero, DP take pair)
- 3-address, reg-reg arithmetic instruction
- Single address mode for load/store:
base + displacement
 - no indirection
- Simple branch conditions
- Delayed branch

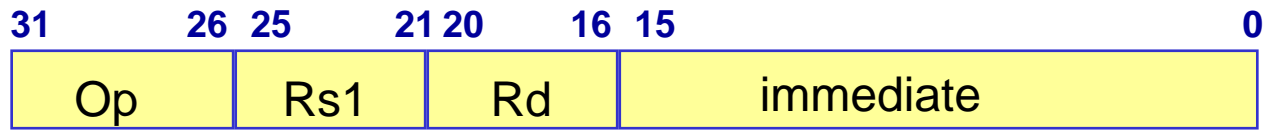
**See: SPARC, MIPS, HP PA-Risc, DEC Alpha, IBM PowerPC,
CDC 6600, CDC 7600, Cray-1, Cray-2, Cray-3**

Example: MIPS

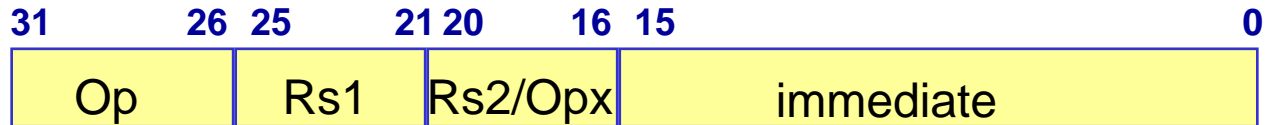
Register-Register



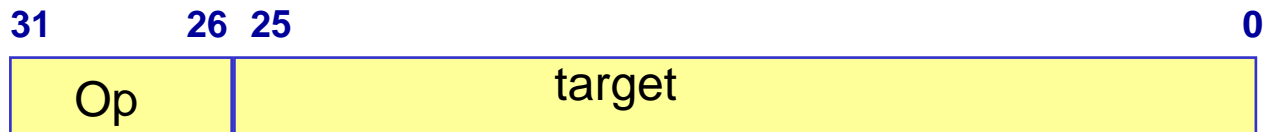
Register-Immediate



Branch



Jump / Call



Concluding Remarks

- **ISA: GPR with Load/Store**
- **Addressing modes:**
 - Displacement (12 to 16 bits)
 - Immediate (8 to 16 bits)
 - Register deferred
- **Instruction Set Operations:**
 - Load, store
 - Arithmetic, logic, shift, compare
 - Branch (PC-relative 8 bit), jump, call, return
- **Type & Size of Operands:**
 - Integer 8, 16 and 32 bit
 - Floating-point (IEEE 754) 32 and 64 bit

Concluding Remarks (cont'd)

- **Instruction Set Encoding:**
 - Fixed encoding
 - Hybrid encoding
- **Size of Register File:**
 - At least 16 GP registers
 - Separate integer (32 bit) and FP (64 bit) register files
- **CISC vs. RISC:**
 - Pros. and cons
 - Intel: typical CISC
 - Alpha: typical RISC
- **DLX Architecture**