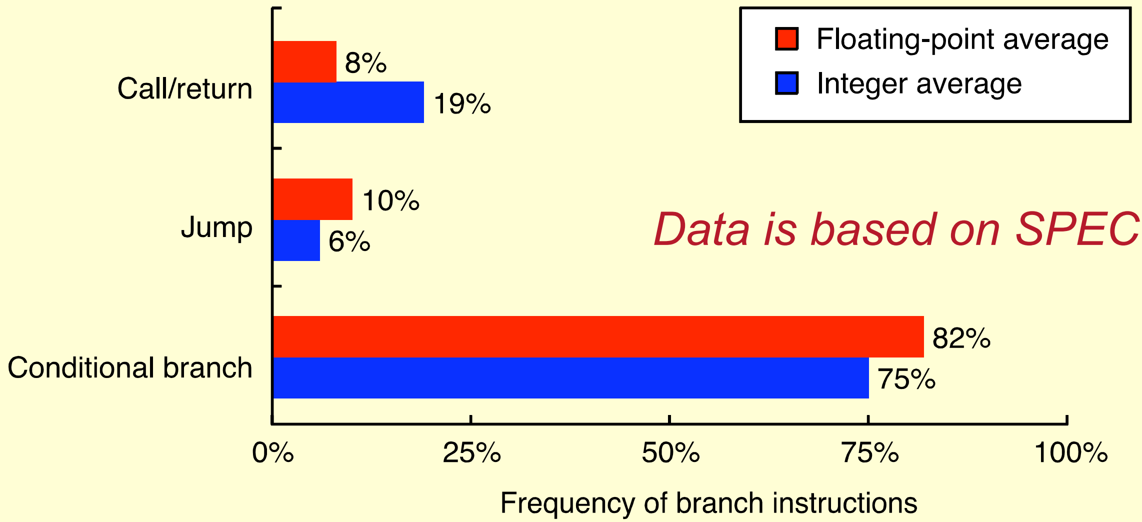
CMSC 611: Advanced Computer Architecture

Instruction Set Architecture (2)

Some material adapted from Mohamed Younis, UMBC CMSC 611 Spr 2003 course slides
Some material adapted from Hennessy & Patterson / © 2003 Elsevier Science

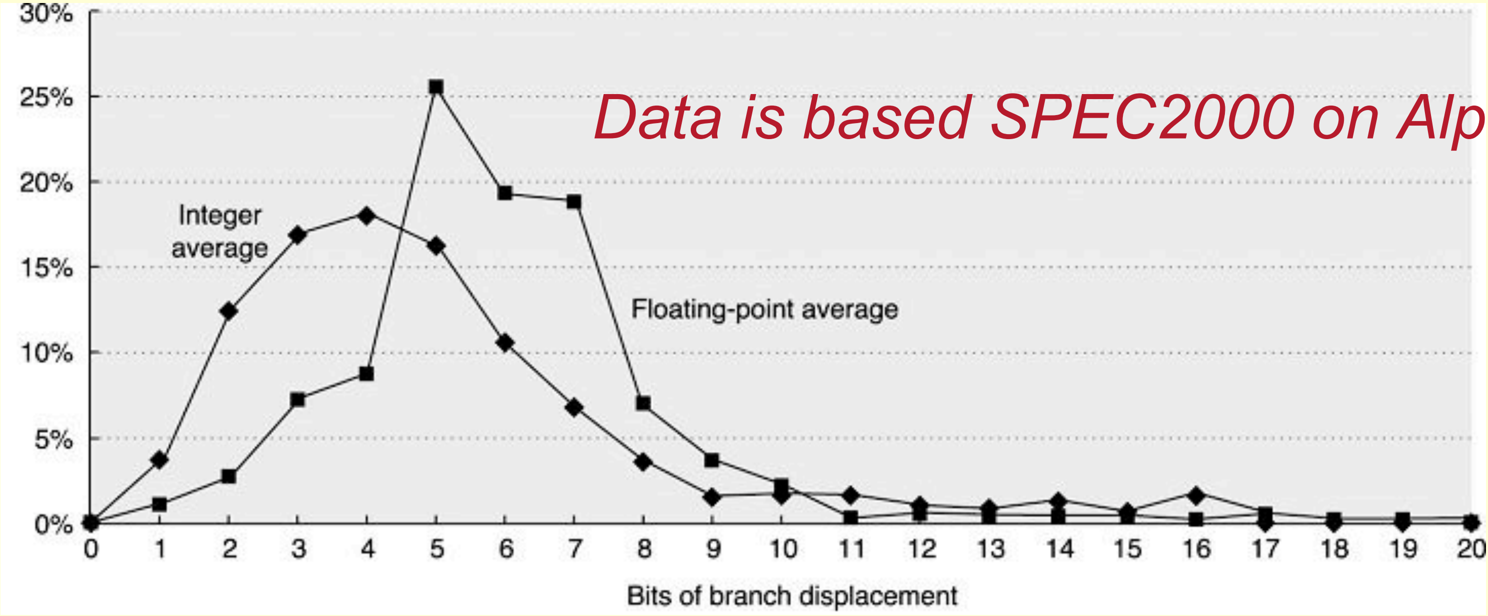
Control Flow Instructions



Data is based on SPEC2000 on Alpha

- Jump: unconditional change in the control flow
- Branch: conditional change in the control flow
- Procedure calls and returns

Destination Address Definition

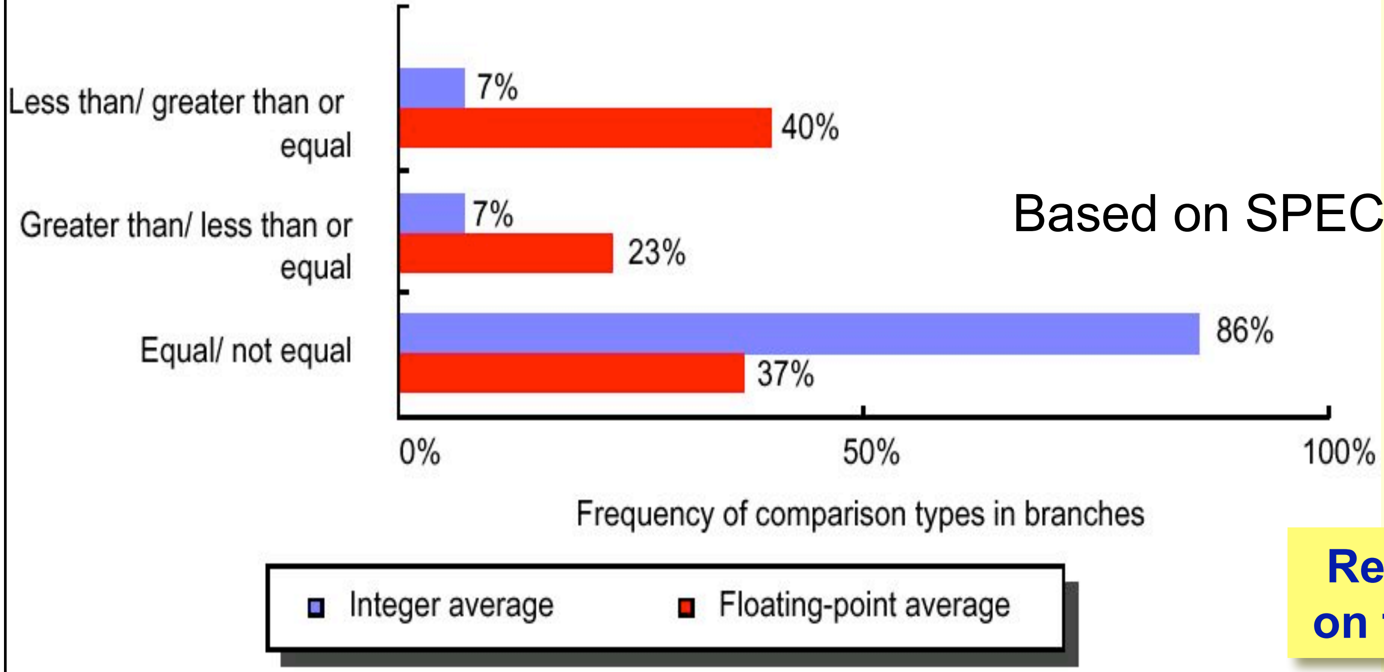


Data is based SPEC2000 on Alpha

- PC-relative addressing
  - Good for short position-independent forward & backward jumps
- Register indirect addressing
  - Good for dynamic libraries, virtual functions & packed case statements

Condition Evaluation

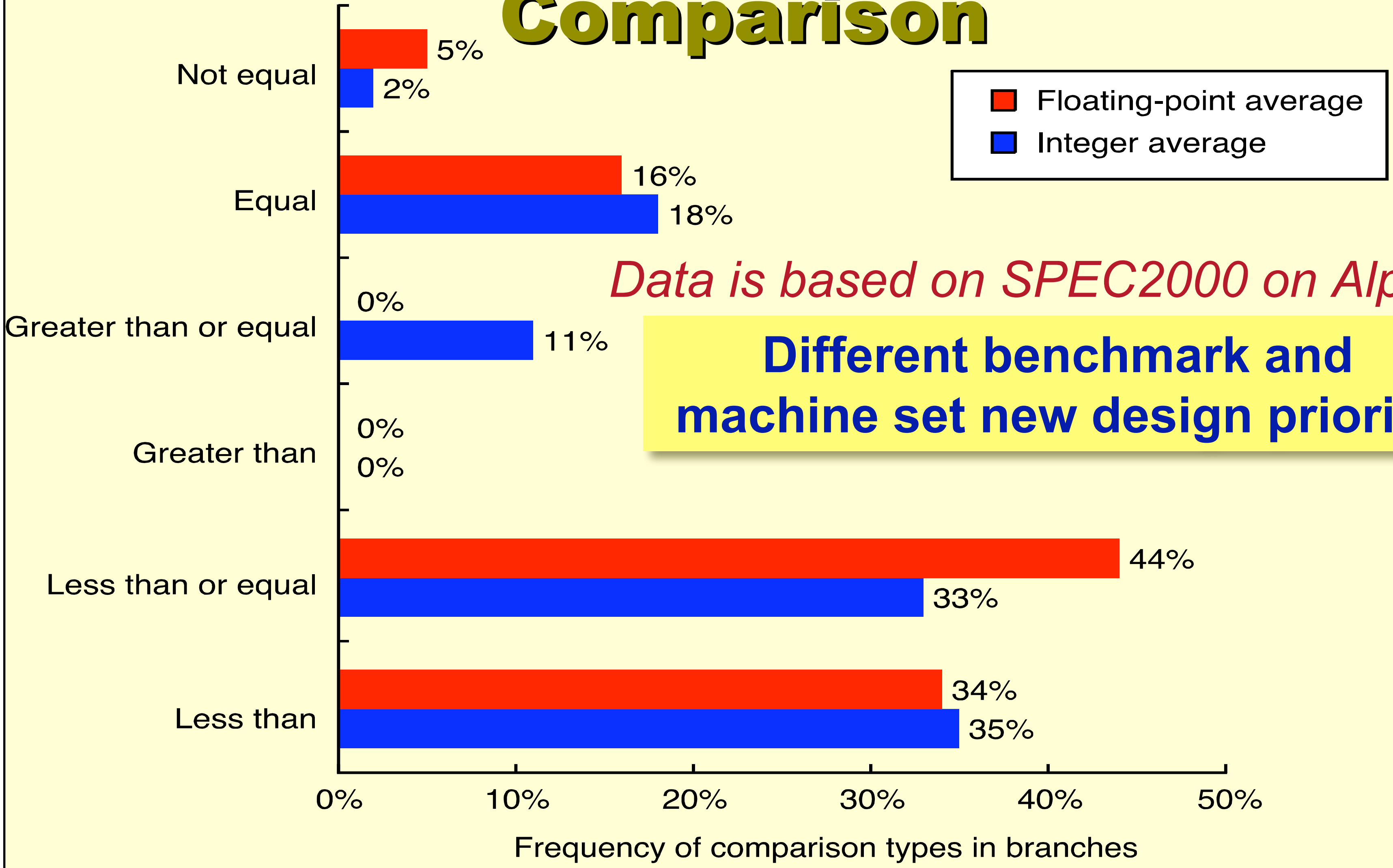
Name	How condition is tested	Advantages	Disadvantages
Condition Code (CC)	Special bits are set by ALU operations, possibly under program control	Sometimes condition is set for free	CC is extra state. Condition codes constrain instructions' ordering since they pass info. from one instruction to a branch
Condition register	Test arbitrary register with the result of a comparison	Simple	Uses up a register
Compare & branch	Compare is part of the branch.	One instruction rather than two for a branch	May be too much work per instruction



Based on SPEC92 on MIPS

Remember to focus on the common case

Frequency of Types of Comparison



Data is based on SPEC2000 on Alpha

Different benchmark and machine set new design priority

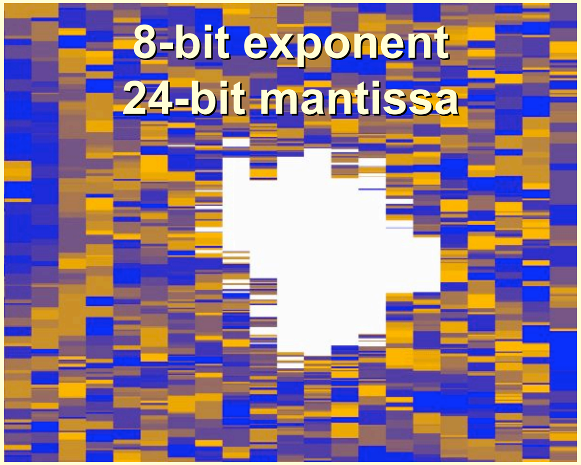
DSPs support *repeat* instruction for for loops (vectors) using 3 registers

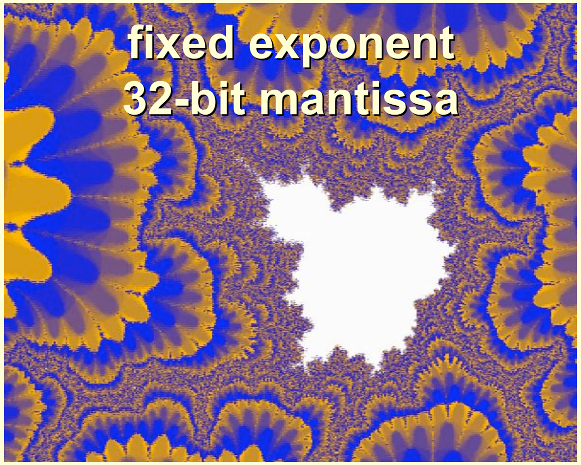
Type and Size of Operands

- Operand type encoded in instruction opcode
  - The type of an operand effectively gives its size
- Common types include character, half word and word size integer, single- and double-precision floating point
  - Characters are almost always in ASCII, though 16-bit Unicode (for international characters) is gaining popularity
  - Integers in 2’s complement
  - Floating point in IEEE 754

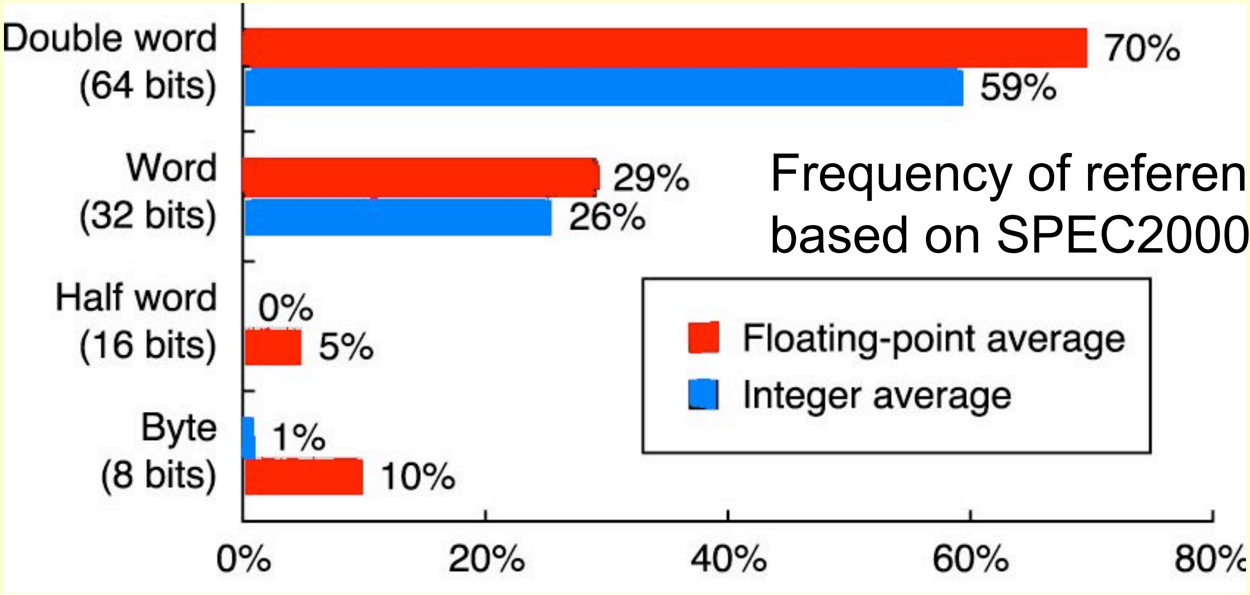
Unusual Types

- Business Applications
  - Binary Coded Decimal (BCD)
    - Exactly represents all decimal fractions (binary doesn’t!)
- DSP
  - Fixed point
    - Good for limited range numbers: more mantissa bits
  - Block floating point
    - Single shared exponent for multiple numbers
- Graphics
  - 4-element vector operations (RGBA or XYZW)
    - 8-bit, 16-bit or single-precision floating point





Size of Operands



Frequency of reference by size based on SPEC2000 on Alpha

- Double-word: double-precision floating point + addresses in 64-bit machines
- Words: most integer operations + addresses in 32-bit machines
- *For the mix in SPEC*, word and double-word data types dominates

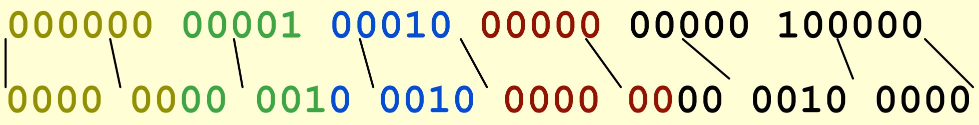
Instruction Representation

- All data in computer systems is represented in binary
- Instructions are no exception
- The program that translates the human-readable code to numeric form is called an *Assembler*
- Hence *machine-language* or *assembly-language*

Example:

Assembly: `ADD $t0, $s1, $s2`

M/C language (binary):



```
000000 00001 00010 00000 00000 100000
0000 0000 0010 0010 0000 0000 0010 0000
```

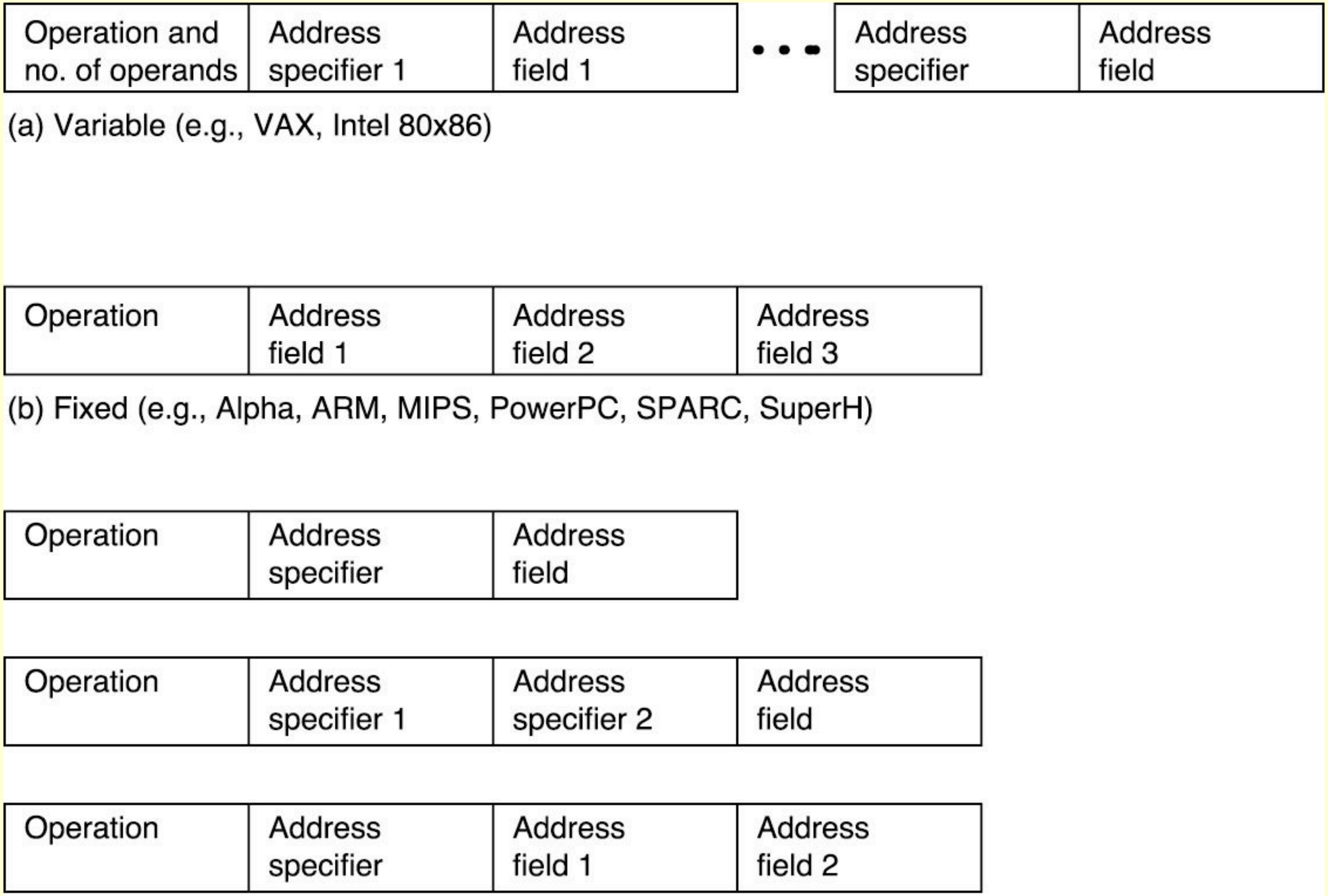
M/C language (hex): `0x00220020`

Note: MIPS compiler by default maps $s0,…,$s7 to reg. 16-23 and $t0,…,$t7 to reg. 8-15

Encoding an Instruction Set

- Affects the size of the compiled program
- Also complexity of the CPU implementation
- Operation in one field called opcode
- Addressing mode in opcode or separate field
- Must balance:
  - Desire to support as many registers and addressing modes as possible
  - Effect of operand specification on the size of the instruction (and program)
  - Desire to simplify instruction fetching and decoding during execution
- Fixed size instruction encoding simplifies CPU design but limits addressing choices

Encoding Examples



Operation and no. of operands	Address specifier 1	Address field 1	• • •	Address specifier	Address field

(a) Variable (e.g., VAX, Intel 80x86)

Operation	Address field 1	Address field 2	Address field 3

(b) Fixed (e.g., Alpha, ARM, MIPS, PowerPC, SPARC, SuperH)

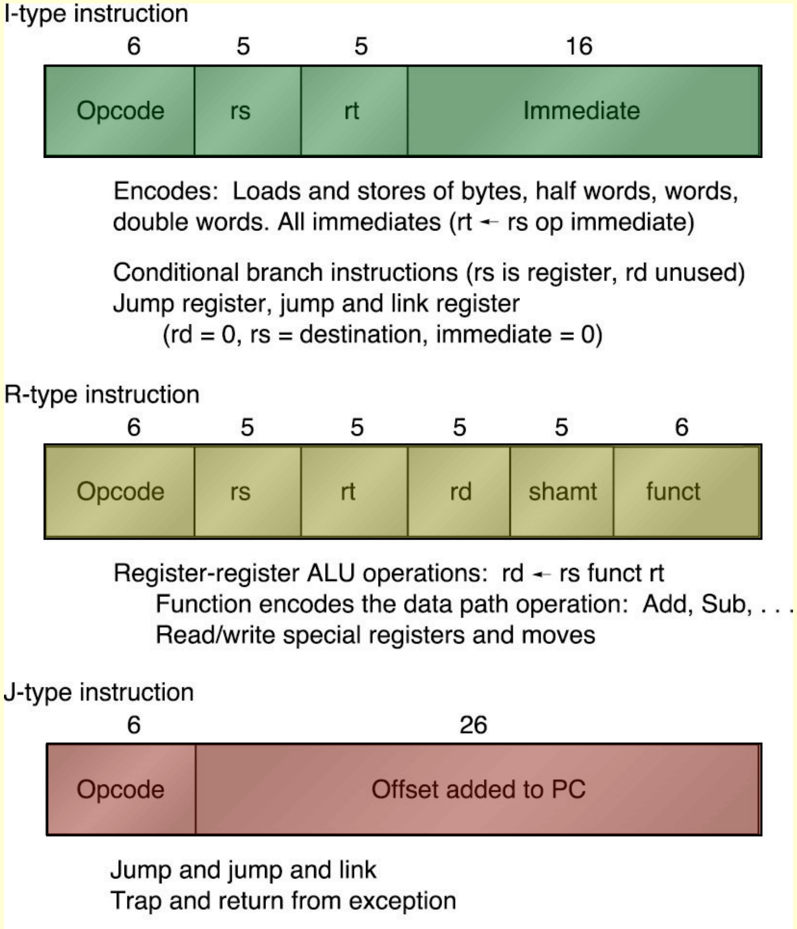
Operation	Address specifier	Address field

Operation	Address specifier 1	Address specifier 2	Address field

Operation	Address specifier	Address field 1	Address field 2

(c) Hybrid (e.g., IBM 360/70, MIPS16, Thumb, TI TMS320C54x)

MIPS Instruction Formats



I-type instruction

Encodes: Loads and stores of bytes, half words, words, double words. All immediates (rt ← rs op immediate)

Conditional branch instructions (rs is register, rd unused)
Jump register, jump and link register
(rd = 0, rs = destination, immediate = 0)

R-type instruction

Register-register ALU operations: rd ← rs funct rt
Function encodes the data path operation: Add, Sub, . . .
Read/write special registers and moves

J-type instruction

Jump and jump and link
Trap and return from exception

opcodes

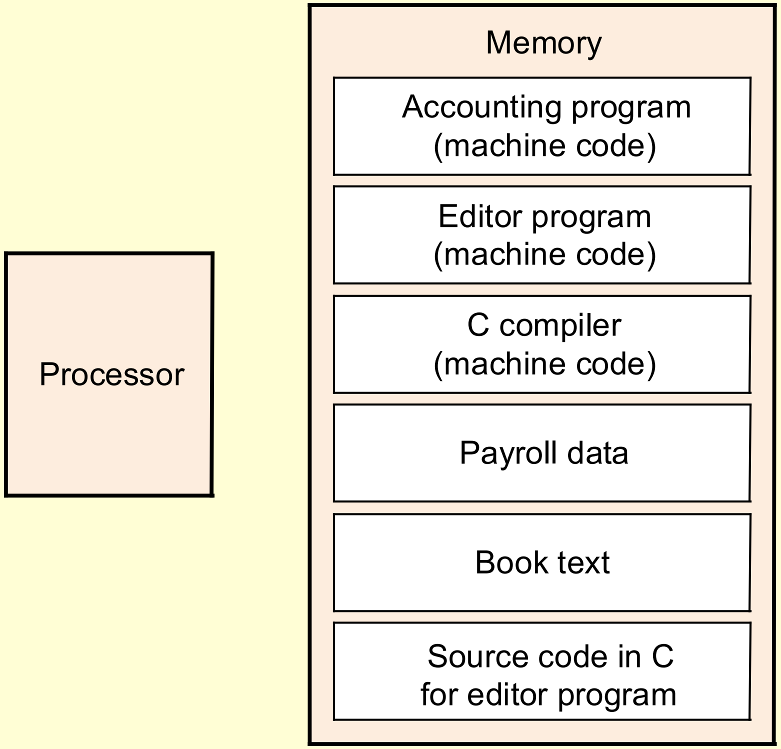
	000	001	010	011	100	101	110	111
000	R-type		j	jal	beq	bne	blez	bgtz
001	addi	addiu	slti	sltiu	andi	ori	xori	
010								
011	llo	lhi	trap					
100	lb	lh		lw	lbu	lhu		
101	sb	sh		sw				
110								
111								

funct codes

	000	001	010	011	100	101	110	111
000	sll		srl	sra	sllv		srlv	srav
001	jr	jalr						
010	mfhi	mthi	mflo	mtlo				
011	mult	multu	div	divu				
100	add	addu	sub	subu	and	or	xor	nor
101			slt	sltu				
110								
111								

The Stored Program Concept

- Today’s computers are built on two key principles :
  - Instructions are represented as numbers
  - Programs can be stored in memory to be read or written just like numbers
- Memory can contain:
  - the source code for an editor
  - the compiled m/c code for the editor
  - the text that the compiled program is using
  - the compiler that generated the code


Processor
Memory
Accounting program (machine code)
Editor program (machine code)
C compiler (machine code)
Payroll data
Book text
Source code in C for editor program

GPU Shading ISA

- Data
  - IEEE-like floating point
  - 4-element vectors
    - Most instructions perform operation on all four
- Addressing
  - No addresses
  - ATTRIB, PARAM, TEMP, OUTPUT
  - Limited arrays
  - Element selection (read & write)
    - C.xyw, C.rgba

GPU Shading ISA

- Instructions:

Instruction	Operation	Instruction	Operation
ABS r,s	r = abs(s)	MIN r,s1,s2	r = min(s1,s2)
ADD r,s1,s2	r = s1+s2	MOV r,s1	r = s1
CMP r,c,s1,s2	r = c<0 ? s1 : s2	MUL r,s1,s2	r = s1*s2
COS r,s	r = cos(s)	POW r,s1,s2	r ≈ $s1^{s2}$
DP3 r,s1,s2	r = s1.xyz • s2.xyz	RCP r,s1	r = 1/s1
DP4 r,s1,s2	r = s1 • s2	RSQ r,s1	r = 1/sqrt(s1)
DPH r,s1,s2	r = s1.xyz1 • s2	SCS r,s1	r = (cos(s),sin(s),?,?)
DST r,s1,s2	r = (1,s1.y*s2.y,s1.z,s2.w)	SGE r,s1,s2	r = s1≥s2 ? 1 : 0
EX2 r,s	r ≈ 2^{s}	SIN r,s	r = sin(s)
FLR r,s	r = floor(s)	SLT r,s1,s2	r = s1<s2 ? 1 : 0
FRC r,s	r = s - floor(s)	SUB r,s1,s2	r = s1-s2
KIL s	if (s<0) discard	SWZ r,s,cx,cy,cz,cw	r = swizzle(s)
LG2 r,s	r ≈ $\log_2(s)$	TEX r,s,name,nD	r = texture(s)
LIT r,s	r = lighting computation	TXB r,s,name,nD	r = textureLOD(s)
LRP r,t,s1,s2	r = t*s1 + (1-t)*s2	TXP r,s,name,nD	r = texture(s/s.w)
MAD r,s1,s2,s3	r = s1*s2 + s3	XPD r,s1,s2	r = s1×s2
MAX r,s1,s2	r = max(s1,s2)		

GPU Shading ISA

- Notable:
  - Many special-purpose instructions
  - No binary encoding, interface is text form
    - No ISA limits on future expansion
    - No ISA limits on registers
    - No ISA limits on immediate values
  - No branching! (exists now… added later)