COEN6741

Computer Architecture and Design

Chapter 2

Instruction Set Principles

(Dr. Sofiène Tahar)

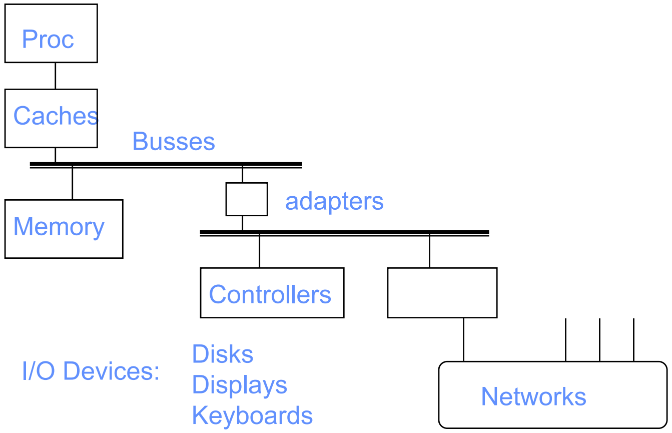
Outline

- Introduction
- ISA Classes
- Addressing Modes
- Operands Type and Size
- Instruction Operations
- Instructions Formats
- Compiler Considerations
- MIPS R3000 Case Study

Review: Organization

- All computers consist of five components
  - Processor: (1) datapath and (2) control
  - (3) Memory
  - (4) Input devices and (5) Output devices
- Not all "memory" are created equally
  - Cache: fast (expensive) memory are placed closer to the processor
  - Main memory: less expensive memory--we can have more
- Input and output (I/O) devices have the messiest organization
  - Wide range of speed: graphics vs. keyboard
  - Wide range of requirements: speed, standard, cost ...
  - Least amount of research (so far)

Review: Computer System Components


Proc

Caches

Busses

Memory

adapters

Controllers

I/O Devices:

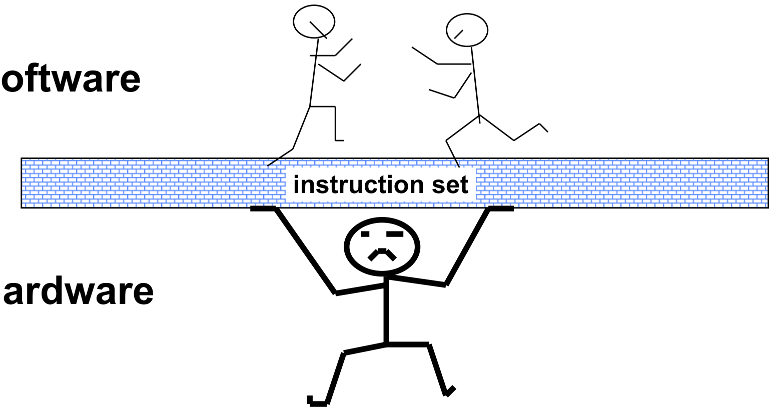
Disks
Displays
Keyboards

Networks

- All have interfaces & organizations

Review: Instruction Set Design

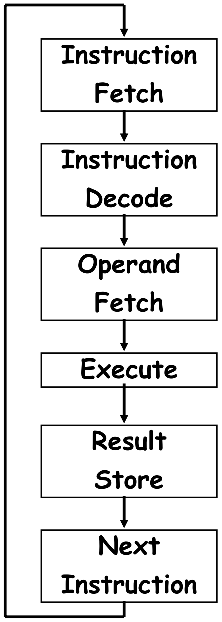
software



hardware

Which part is easier to change?

Review: Execution Cycle



Instruction Fetch — Obtain instruction from program storage

Instruction Decode — Determine required actions and instruction size

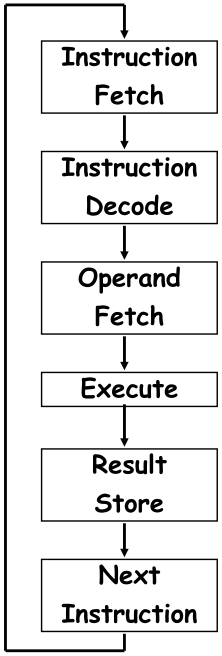
Operand Fetch — Locate and obtain operand data

Execute — Compute result value or status

Result Store — Deposit results in storage for later use

Next Instruction — Determine successor instruction

Instruction Set Architecture: What must be specified?



Instruction Fetch

Instruction Decode

Operand Fetch

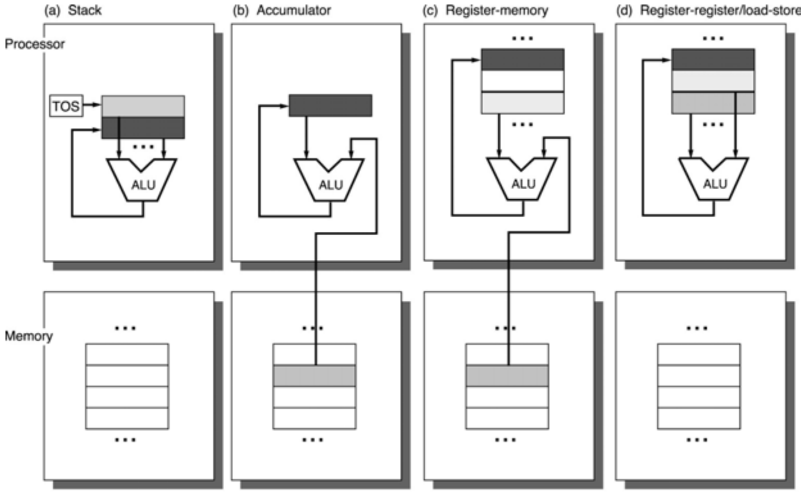
Execute

Result Store

Next Instruction

- Instruction Format or Encoding
  - how is it decoded?
- Location of operands and result
  - where other than memory?
  - how many explicit operands?
  - how are memory operands located?
  - which can or cannot be in memory?
- Data type and Size
- Operations
  - what are supported
- Successor instruction
  - jumps, conditions, branches
- fetch-decode-execute is implicit!

Basic ISA Classes



Comparison:

Bytes per instruction? Number of Instructions? Cycles per instruction?

Basic ISA Classes

Accumulator:

1 address	add A	acc ← acc + mem[A]
1+x address	addx A	acc ← acc + mem[A + x]

Stack:

0 address	add	tos ← tos + next

General Purpose Register:

2 address	add A B	EA(A) ← EA(A) + EA(B)
3 address	add A B C	EA(A) ← EA(B) + EA(C)

Load/Store:

3 address	add Ra Rb Rc	Ra ← Rb + Rc
	load Ra Rb	Ra ← mem[Rb]
	store Ra Rb	mem[Rb] ← Ra

Comparing Number of Instructions

Code sequence for C = A + B for four classes of instruction sets:

Stack	Accumulator	Register (register-memory)	Register (load-store)
Push A	Load A	Load R1,A	Load R1,A
Push B	Add B	Add R1,B	Load R2,B
Add	Store C	Store C, R1	Add R3,R1,R2
Pop C			Store C,R3

General Purpose Registers Dominate

- 1975-1995 all machines use general purpose registers
- Advantages of registers
  - registers are faster than memory
  - registers are easier for a compiler to use
    - e.g., (A*B) – (C*B) – (A*D) can do multiplies in any order vs. stack
  - registers can hold variables
    - memory traffic is reduced, so program is sped up (since registers are faster than memory)
    - code density improves (since register named with fewer bit than memory location)

General Purpose Registers Dominate

- 1975-1995 all machines use general purpose registers
- Advantages of registers
  - registers are faster than memory
  - registers are easier for a compiler to use
  - e.g., (A*B) – (C*B) – (A*D) can do multiplies in any order vs. stack
  - registers can hold variables
  - memory traffic is reduced, so program is sped up (since registers are faster than memory)
  - code density improves (since register named with fewer bits than memory location)

Classification of GPR Architectures

# Memory Addresses	Max. # Operands	Type of Architecture	Examples
0	3	Register-Register	Alpha,ARM, MIPS, PowerPC, Sparc
1	2	Register-Memory	IBM360/370,Intel80x86, MC68000, TI TMS320C54x
2	2	Memory-Memory	VAX
3	3	Memory-Memory	VAX

Instruction Classes (Summary)

- Expect new instruction set architecture to use general purpose register
- Pipelining (performance) expect it to use Load/Store variant of GPR ISA

Memory Addressing

- Since 1980 almost every machine uses addresses to level of 8-bits (byte)
- Two questions for design of ISA:
  1. Since we read a 32-bit word as four loads of bytes from sequential byte addresses or as one load word from a single byte address, how do byte addresses map onto words?
  2. Can a word be placed on any byte boundary?

Addressing Objects: Endianess and Alignment

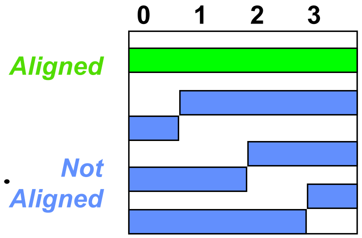
- Big Endian: address of most significant byte = word address (xx00 = Big End of word)
  - IBM 360/370, Motorola 68k, MIPS, Sparc, HP PA

big endian byte 0	0	1	2	3
	msb			lsb

- Little Endian: address of least significant byte = word address (xx00 = Little End of word)
  - Intel 80x86, DEC Vax, DEC Alpha (Windows NT)

little endian byte 0	3	2	1	0
	msb			lsb

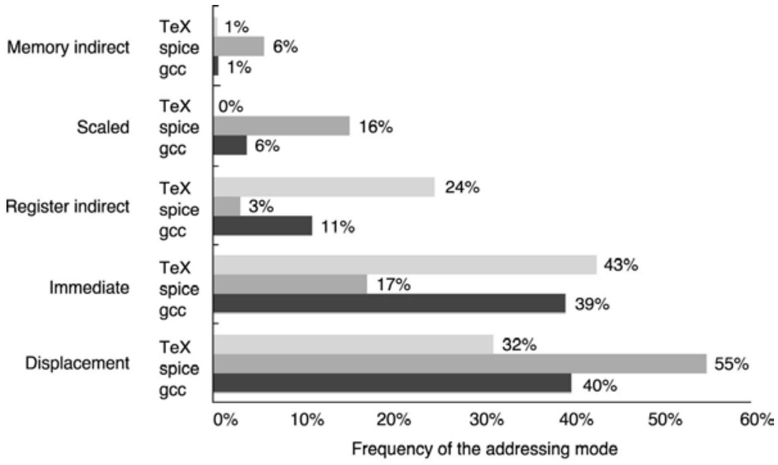
Alignment: require objects fall on Address that is multiple of their size.



Addressing Modes

Addressing mode	Example	Meaning
Register	Add R4,R3	R4← R4+R3
Immediate	Add R4,#3	R4 ← R4+3
Displacement	Add R4,100(R1)	R4 ← R4+Mem[100+R1]
Register indirect	Add R4,(R1)	R4 ← R4+Mem[R1]
Indexed / Base	Add R3,(R1+R2)	R3 ← R3+Mem[R1+R2]
Direct or absolute	Add R1,(1001)	R1 ← R1+Mem[1001]
Memory indirect	Add R1,@(R3)	R1 ← R1+Mem[Mem[R3]]
Auto-increment	Add R1,(R2)+	R1 ← R1+Mem[R2]; R2 ← R2+d
Auto-decrement	Add R1,-(R2)	R2 ← R2–d; R1 ← R1+Mem[R2]
Scaled	Add R1,100(R2)[R3]	R1← R1+Mem[100+R2+R3*d]

Addressing Mode Usage



Addressing Mode Usage (Summary)

3 programs measured on machine with all address modes (VAX)

- Displacement: 42% avg, 32% to 55
- Immediate: 33% avg, 17% to 43%
- Register deferred (indirect): 13% avg, 3% to 24%
- Scaled: 7% avg, 0% to 16%
- Memory indirect: 3% avg, 1% to 6%
- Misc: 2% avg, 0% to 3%

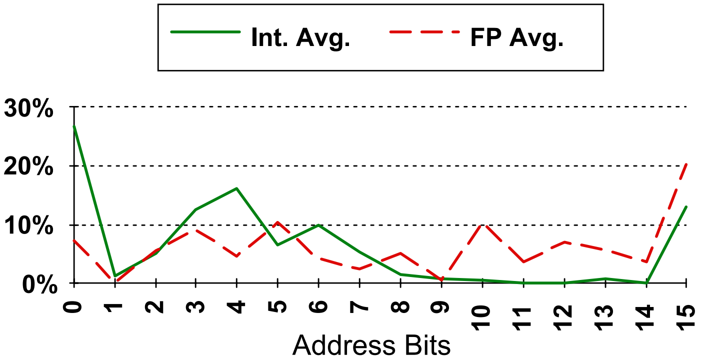
75%

85%

75% displacement & immediate

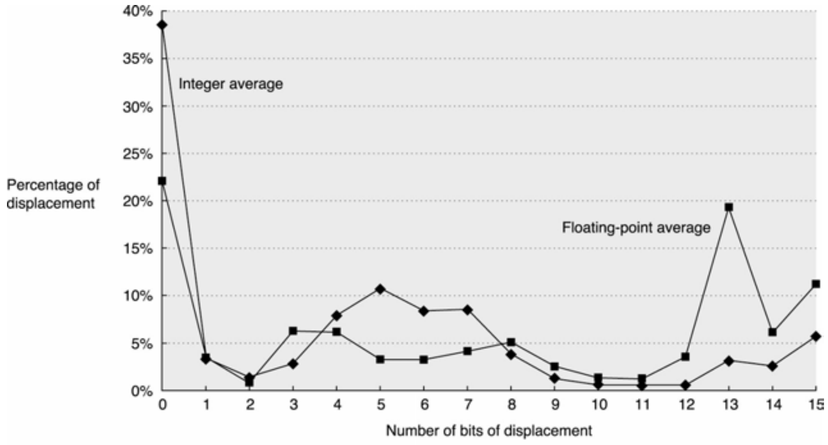
88% displacement, immediate & register indirect

Displacement Address Size?



- Avg. of 5 SPECint92 programs v. avg. 5 SPECfp92 programs
- X-axis is in powers of 2: 4 => addresses > 23 (8) and < 2 4 (16)
- 1% of addresses > 16-bits
- 12 - 16 bits of displacement needed

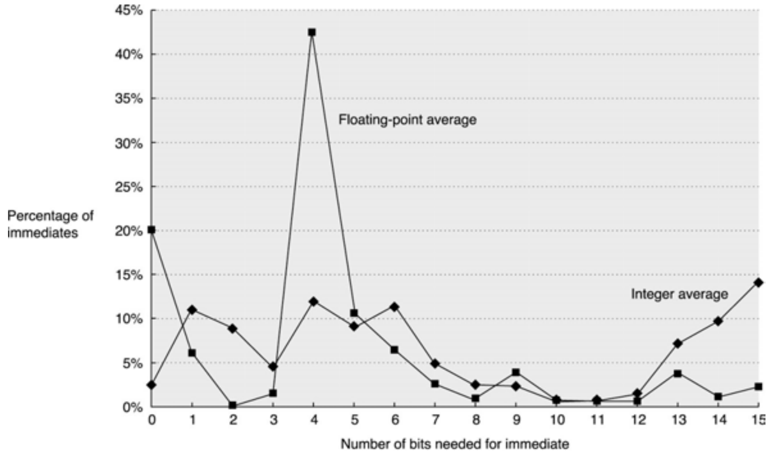
Displacement Address Size?



Alpha architecture.
SPEC CPU2000 and CINT2000 and CFP2000

© 2003 Elsevier Science (USA). All rights reserved.

Immediate Addressing



Alpha architecture.
SPEC CPU2000 and CINT2000 and CFP2000

© 2003 Elsevier Science (USA). All rights reserved.

Addressing Modes (Summary)

- Data Addressing modes that are important:
  - Displacement
  - Immediate
  - Register Indirect
- Displacement size should be 12 to 16 bits
- Immediate size should be 8 to 16 bits

A similar measurement on the Vax showed that 20% to 25% of immediates were longer than 16 bits.

Data Types

Bit: 0, 1

Bit String: sequence of bits of a particular length
4 bits is a nibble
8 bits is a byte
16 bits is a half-word
32 bits is a word
64 bits is a double-word

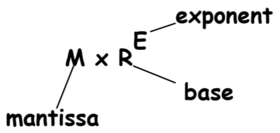
Character:
ASCII 7 bit code
16 bit unicode (used in Java) is gaining popularity

Decimal:
digits 0-9 encoded as 0000b thru 1001b
two decimal digits packed per 8 bit byte

IEEE Standard 754

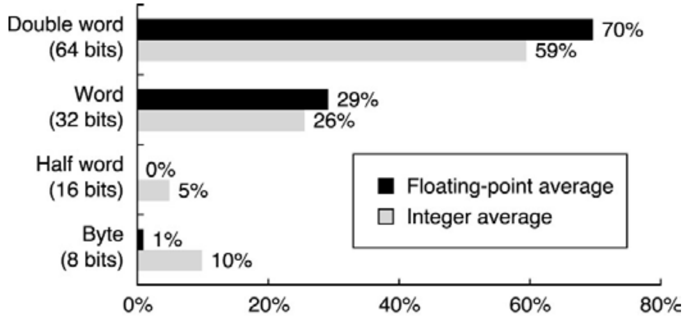
Integers:
2's Complement

Floating Point:
Single Precision
Double Precision
Extended Precision



How many +/- #'s?
Where is decimal pt?
How are +/- exponents represented?

Operand Size Usage



© 2003 Elsevier Science (USA). All rights reserved.

Support these data sizes and types:

- 8-bit, 16-bit, 32-bit integers and
- 32-bit and 64-bit IEEE 754 floating-point

Typical Operations

Data Movement	Load (from memory) Store (to memory) memory-to-memory move register-to-register move input (from I/O device) output (to I/O device) push, pop (to/from stack)
Arithmetic	integer (binary + decimal) or FP Add, Subtract, Multiply, Divide
Shift	shift left/right, rotate left/right
Logical	not, and, or, set, clear
Control (Jump/Branch)	unconditional, conditional
Subroutine Linkage	call, return
Interrupt	trap, return
Synchronization	test & set (atomic r-m-w)
String	search, translate
Graphics (MMX)	parallel subword ops (4 16bit add)

Top 10 80x86 Instructions

Rank	Instruction	Integer average percent executed
1	load	22%
2	conditional branch	20%
3	compare	16%
4	store	12%
5	add	8%
6	and	6%
7	sub	5%
8	move register-register	4%
9	call	1%
10	return	1%
	Total	96%

Simple instructions dominate instruction frequency

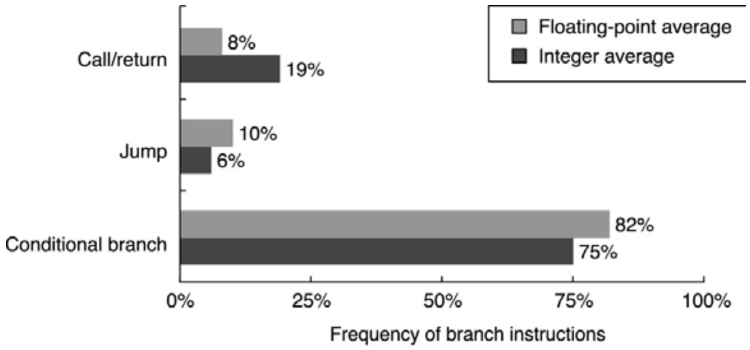
Instruction Operation (Summary)

Support these simple instructions, since they will dominate the number of instructions executed:

- load
- store
- add, subtract, move register-register, and, shift
- compare equal, compare not equal
- branch, jump, call, return;

Instruction for Control Flow

- Conditional branches
- Jumps
- Procedure calls
- Procedure returns



Methods of Testing Condition

- Condition Codes (80x86,ARM,PowerPC,Sparc)

  Processor status bits are set as a side-effect of arithmetic instructions (possibly on Moves) or explicitly by compare or test instructions.

  e.g.: add r1, r2, r3

        bz label

- Condition Register (Alpha, MIPS)

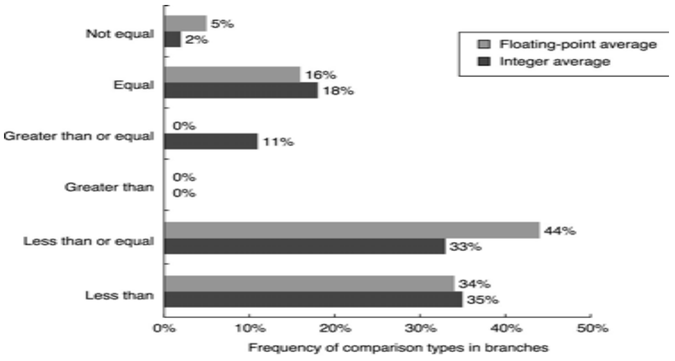
  e.g.: cmp r1, r2, r3

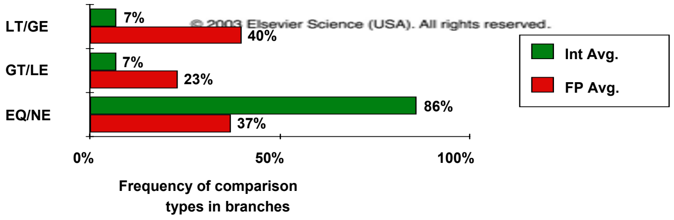
        bgt r1, label

- Compare and Branch (PA-RISC, VAX)

  e.g.: bgt r1, r2, label

Frequency of Types of Compares





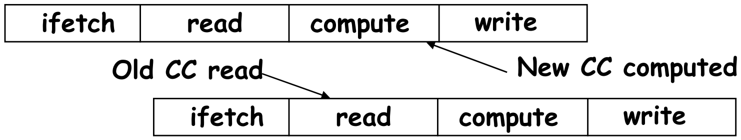
Condition Codes

Setting CC as side effect can reduce the # of instructions

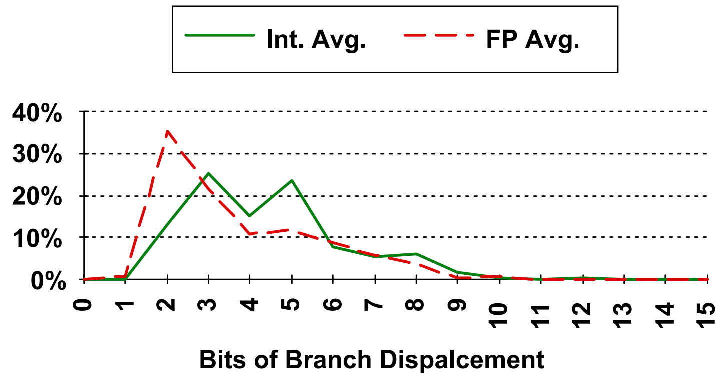
```
X:     .                        X:     .
       .                               .
       .            vs.                .
   SUB  r0, #1, r0                 SUB  r0, #1, r0
   BRP  X                          CMP  r0, #0
                                   BRP  X
```

But also has disadvantages:

- not all instructions set the condition codes; which do and which do not often confusing! *e.g., shift instruction sets the carry bit*
- dependency between the instruction that sets the CC and the one that tests it: to overlap their execution, may need to separate them with an instruction that does not change the CC

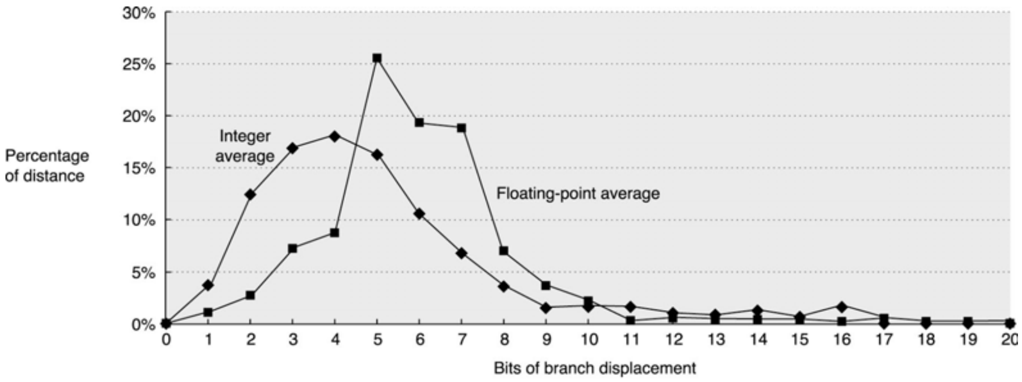


Conditional Branch Distance



- Distance from branch in instructions 2i => < ±2^{i-1} & > 2^{i-2}
- 25% of integer branches are > 2 to < 4 or -2 to -4 instructions

How far are branch targets?



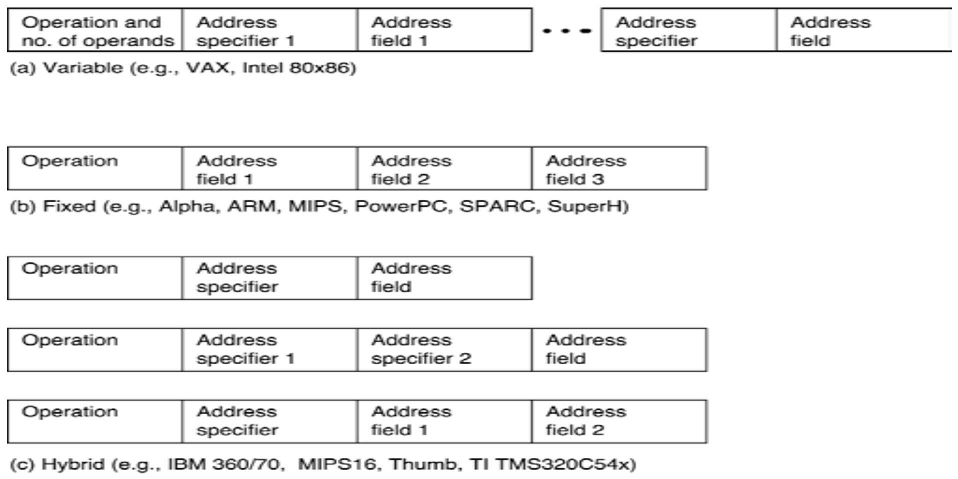
Conditional Branch (Summary)

- PC-relative since most branches are relatively close to the current PC address
- At least 8 bits suggested (± 128 instructions)
- Less than or Equal most important for integer programs (76%)

Instruction Format

- If have many memory operands per instructions and many addressing modes,
  - Address Specifier per operand
- If have load-store machine with 1 address per instruction and one or two addressing modes
  - Encode addressing mode in the opcode

Generic Examples of Instruction Format Widths

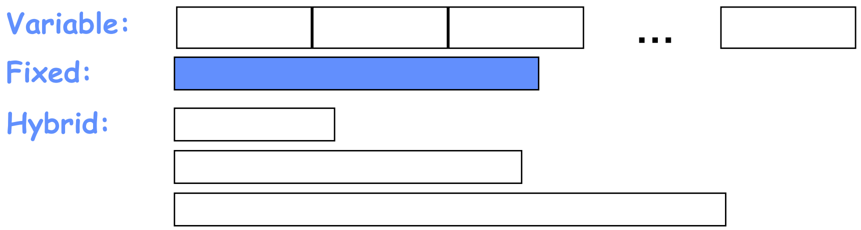


(a) Variable (e.g., VAX, Intel 80x86)

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

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

Instruction Formats



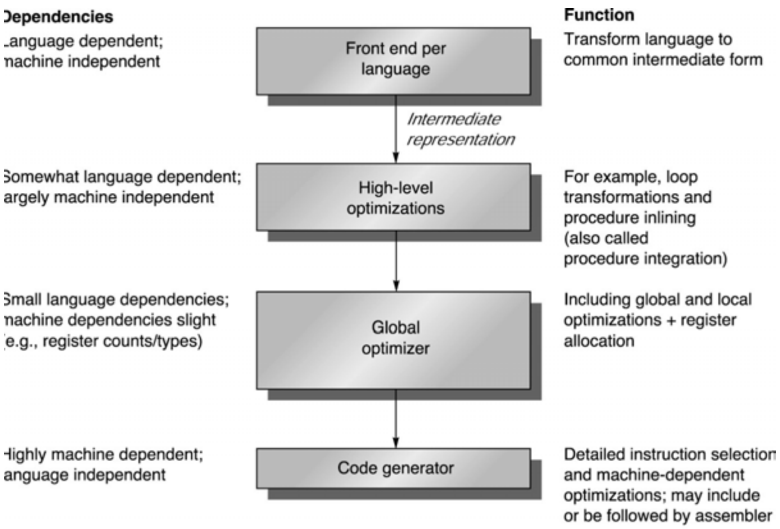
- Addressing modes
  - each operand requires address specifier => variable format
- code size => variable length instructions
- performance => fixed length instructions
  - simple decoding, predictable operations
- With load/store ISA, only one memory address and few addressing modes

➢ simple format, address mode given by opcode

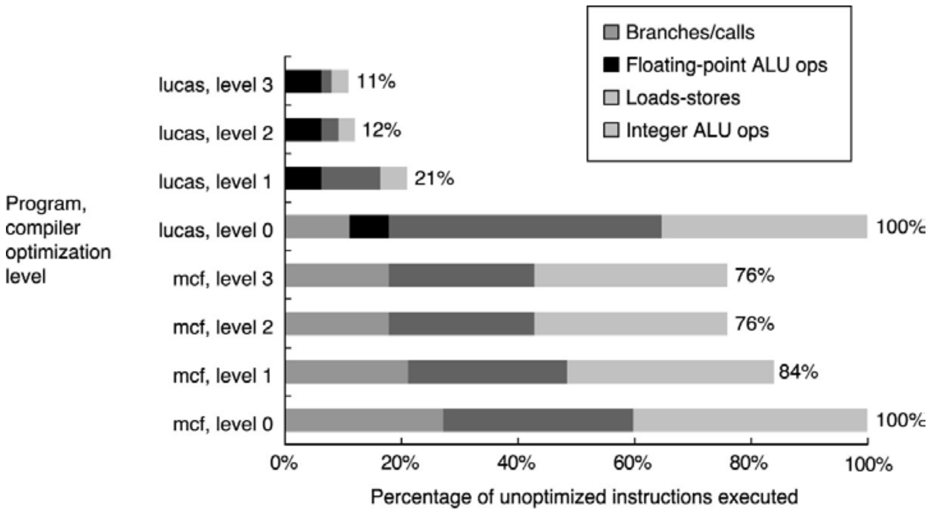
Instruction Formats (Summary)

- If code size is most important, use variable length instructions
- If performance is over is most important, use fixed length instructions
- Discuss the different architectures for different machines, see Appendix D (Intel 80x86), E (VAX), F (IBM 360/370)

Compiler Considerations


Dependencies
Function
anguage dependent; nachine independent
Front end per language
Transform language to common intermediate form
Intermediate representation
Somewhat language dependent; argely machine independent
High-level optimizations
For example, loop transformations and procedure inlining (also called procedure integration)
Small language dependencies; nachine dependencies slight e.g., register counts/types)
Global optimizer
Including global and local optimizations + register allocation
Highly machine dependent; anguage independent
Code generator
Detailed instruction selection and machine-dependent optimizations; may include or be followed by assembler

Compiler Considerations



Compiler Considerations

- Ease of compilation
  - Orthogonality: no special registers, few special cases, all operand modes available with any data type or instruction type
  - Completeness: support for a wide range of operations and target applications
  - Regularity: no overloading for the meanings of instruction fields
  - Streamlined: resource needs easily determined
- Register Assignment is critical too
- Easier if lots of registers

Compiler Considerations (Summary)

- Provide at least 16 general purpose registers plus separate floating-point registers
- Be sure all addressing modes apply to all data transfer instructions
- Aim for a minimalist instruction set

Case Study: MIPS

- **32-bit fixed format instructions** (3 formats: R, I, J)
- **32 64-bit GPR** (R0 contains zero) and 32 FP registers (and HI LO)
  - partitioned by software convention
- **3-address, reg-reg arithmetic instructions**
- **Single address mode for load/store:** base+displacement
  - no indirection, scaled
- **16-bit immediate plus LUI**
- **Simple branch conditions**
  - compare against zero or two registers for equal zero
  - no integer condition codes
- **Delayed branch**
  - execute instruction after the branch (or jump) even if the branch is taken (Compiler can fill a delayed branch with useful work about 50% of the time)

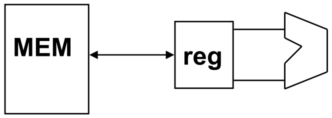
Case Study: MIPS

- Use general purpose registers with a load-store architecture: YES
- Provide at least 16 general purpose registers plus separate floating-point registers: 31 GPR & 32 FPR
- Support basic addressing modes: displacement (with an address offset size of 12 to 16 bits), immediate (size 8 to 16 bits), and register deferred; : YES: 16 bits for immediate, displacement (disp=0 => register deferred)
- All addressing modes apply to all data transfer instructions : YES

Case Study: MIPS

- Use fixed instruction encoding if interested in performance and use variable instruction encoding if interested in code size : Fixed
- Support these data sizes and types: 8-bit, 16-bit, 32-bit integers and 32-bit and 64-bit IEEE 754 floating point numbers: YES
- Support these simple instructions, since they will dominate the number of instructions executed: load, store, add, subtract, move register-register, and, shift, compare equal, compare not equal, branch (with a PC-relative address at least 8-bits long), jump, call, and return: YES
- Aim for a minimalist instruction set: YES

MIPS: Load/Store Architecture

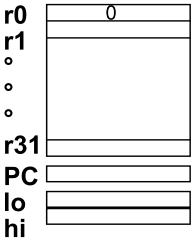


- 3 address GPR
- Register to register arithmetic
- Load and store with simple addressing modes (reg + immediate)
- Simple conditionals
  - compare ops + branch z
  - compare & branch
  - condition code + branch on condition
- Simple fixed-format encoding

R	op	r	r	r	▨
I	op	r	r	immed	
J	op	offset			

➢ Substantial increase in instructions
➢ Decrease in data BW (due to many registers)
➢ Even more significant decrease in CPI (pipelining)
➢ Cycle time, Real estate, Design time, Design complexity

MIPS: Instruction Set



Programmable storage

2^32 x bytes

31 x 32-bit GPRs (R0=0)

32 x 32-bit FP regs (paired DP)

HI, LO, PC

32-bit instructions on word boundary

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

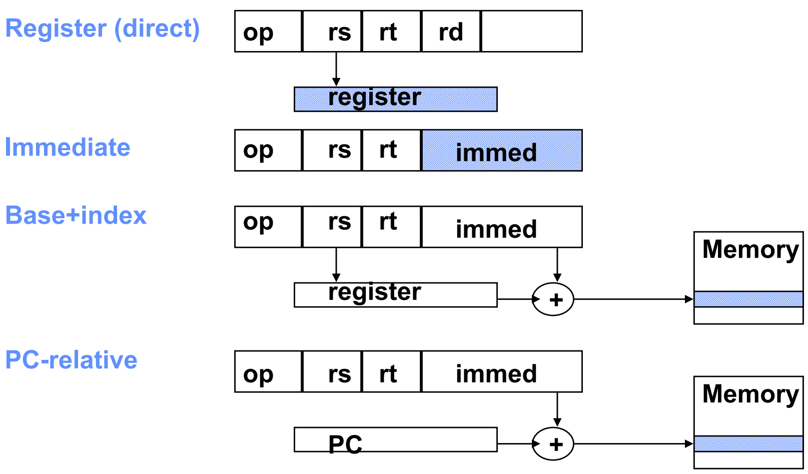
LB, LBU, LH, LHU, LW, LWL, LWR
SB, SH, SW, SWL, SWR

Control

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

MIPS: Addressing Modes & Formats

- Simple addressing modes
- All instructions 32 bits wide



MIPS: Instruction Formats

Register-Register

31 26 25 21 20 16 15 11 10 6 5 0

Op	Rs1	Rs2	Rd		Opx

Register-Immediate

31 26 25 21 20 16 15 0

Op	Rs1	Rd	immediate

Branch

31 26 25 21 20 16 15 0

Op	Rs1	Rs2/Opx	immediate

Jump / Call

31 26 25 0

Op	target

MIPS: Instruction Formats

I-type instruction

6	5	5	16
Opcode	rs	rt	Immediate

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

6	5	5	5	5	6
Opcode	rs	rt	rd	shamt	funct

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

6	26
Opcode	Offset added to PC

Jump and jump and link
Trap and return from exception

MIPS ISA (Summary)

- Instruction Categories
  - Load/Store
  - Computational
  - Jump and Branch
  - Floating Point
    - » coprocessor
  - Memory Management
  - Special

Registers

R0 - R31
PC
HI
LO

- 3 Instruction Formats: all 32 bits wide

OP	rs	rt	rd	sa	funct
OP	rs	rt	immediate		
OP	jump target				

Cray-1: The Original RISC

Register-Register

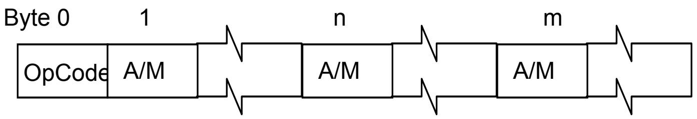
15 9	8 6	5 3	2 0
Op	Rd	Rs1	R2

Load, Store and Branch

15 9	8 6	5 3	2 0	15 0
Op	Rd	Rs1		Immediate

VAX-11: The Canonical CISC

- Variable format, 2 and 3 address instruction



- Rich set of orthogonal address modes
  - immediate, offset, indexed, autoinc/dec, indirect, indirect+offset
  - applied to any operand
- Simple and complex instructions
  - synchronization instructions
  - data structure operations (queues)
  - polynomial evaluation

Chapter 2: Summary#1

- 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

Chapter 2: Summary#2

- 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
- MIPS R3000 Architecture