L04: Instruction Set Architecture 2

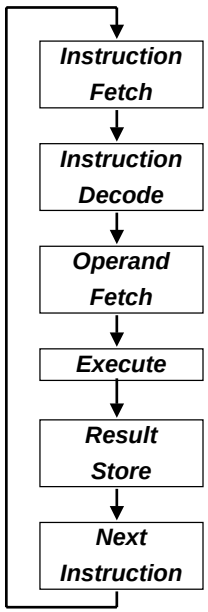
Adapted from:

CS152: Computer Architecture and Engineering
Dave Patterson (www.cs.berkeley.edu/~pattrsn)

Overview

- Quick recap
- Addressing modes
- Instruction formats
- Supported operations

Recap: Instruction Set Architecture: What Must be Specified?



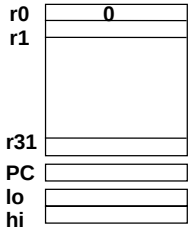
- ° Instruction Format or Encoding
  - – how is it decoded?
- ° Location of operands and result
  - – where other than memory?
  - – how many explicit operands?
  - – how are memory operands located?
  - – which can or cannot be in memory?
- ° Data type and Size
- ° Operations
  - – what are supported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

Instruction set design

- Goal:
  - The instruction set should be easy to implement and it should give good performance
- Four (instruction set) design principles:
  - Simplicity favours regularity
  - Smaller is faster
  - Good design demands a compromise
  - Make the common case fast

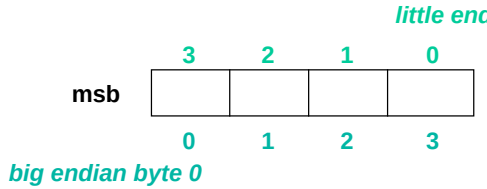
MIPS I Registers

- Programmable storage
  - 2^32 x bytes of memory
  - 31 x 32-bit GPRs (R0 = 0)
  - 32 x 32-bit FP regs (paired DP)
  - HI, LO, PC

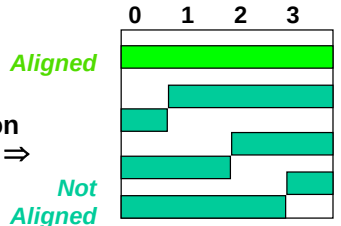


Addressing Objects: Endianness and Alignment

- Big Endian: address of most significant byte = word address
  - IBM 360/370, Motorola 68k, MIPS, Sparc, HP PA
- Little Endian: address of least significant byte = word address
  - Intel 80x86, DEC Vax, DEC Alpha (Windows NT)



Alignment: require that objects fall on address that is multiple of their size ⇒ the address of each MIPS instruction is a multiple of 4



Possible 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? (ignore register mode)

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

---	Displacement:	42% avg, 32% to 55%
---	Immediate:	33% avg, 17% to 43%
---	Register 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%

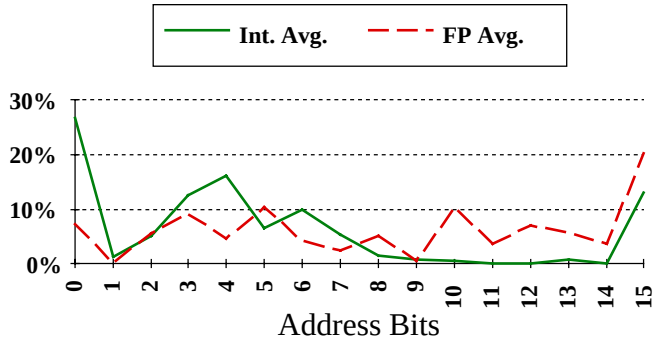
75%

88%

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 > 2^3 (8) and ≤ 2^4 (16)

° 1% of addresses > 16-bits

° 12 - 16 bits of displacement needed

Immediate Size?

- 50% to 60% fit within 8 bits
- 75% to 80% fit within 16 bits

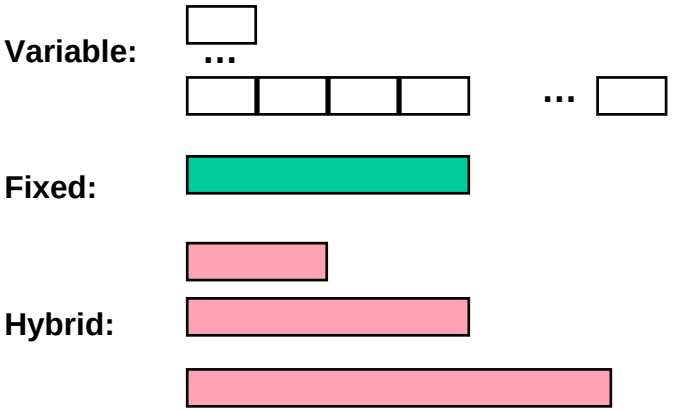
Addressing Summary

•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

Generic Examples of Instruction Format Widths



Summary of Instruction Formats

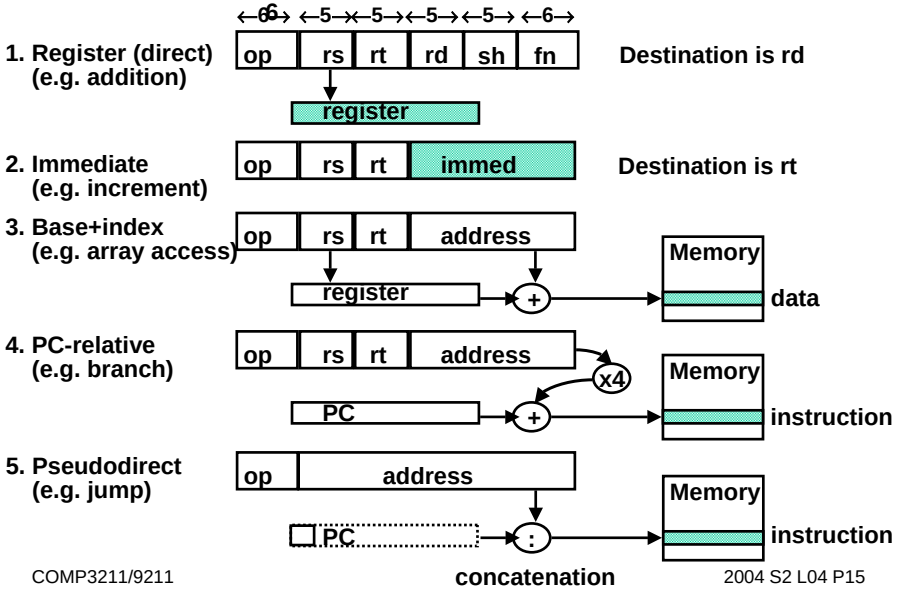
- If code size is most important, use variable length instructions
- If performance is most important, use fixed length instructions
- Embedded machines (ARM, MIPS) added optional mode to execute subset of 16-bit wide instructions (Thumb, MIPS16); per procedure decide performance or density

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 instr. and one or two addressing modes,
  => encode addressing mode in the opcode

MIPS Addressing Modes/Instruction Formats

- All instructions 32 bits wide – 5 modes



Typical Operations (little change since 1960)

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 x 16-bit add)

Top 10 80x86 Instructions

Rank	Instruction	Integer Average Percent total 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

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;

Compilers and Instruction Set Architectures

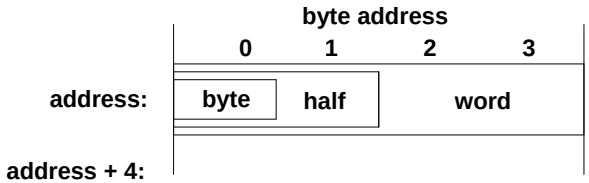
- Ease of compilation
  - ° orthogonality: no special registers, few special cases, all operand modes available with every data and instruction type
  - ° completeness: support for a wide range of operations and target applications
  - ° regularity: no overloading for the meanings of instruction fields
  - ° streamlined: resource needs easily determined
- Register Assignment is critical too
  - ° Easier if lots of registers

Summary of Compiler Considerations

- 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.

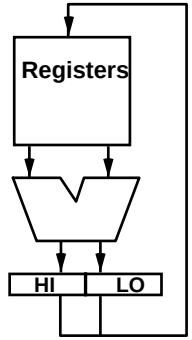
MIPS I Operation Overview

- Arithmetic logical
  - Add, AddU, Sub, SubU, And, Or, Xor, Nor, SLT, SLTU
  - AddI, AddIU, SLTI, SLTIU, AndI, OrI, XorI, LUI = Load Upper Immediate
  - SLL, SRA, SLLV, SRAV = Shift Right Arithmetic Variable
- Memory Access
  - LB, LBU, LH, LHU, LW
  - SB, SH, SW



Multiply / Divide

- Start multiply, divide
  - MULT rs, rt
  - MULTU rs, rt
  - DIV rs, rt
  - DIVU rs, rt
- Move result from multiply, divide
  - MFHI rd
  - MFLO rd
- Move to HI or LO
  - MTHI rd
  - MTLO rd
- Why not Third field for destination?



Data Types

Bit: 0, 1

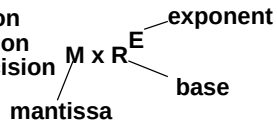
Bit String: sequence of bits of a particular length
- 4 bits is a nibble
- 8 bits is a byte
- 16 bits is a half-word
- 32 bits is a word
- 64 bits is a double-word

Character:
- ASCII 7 bit code

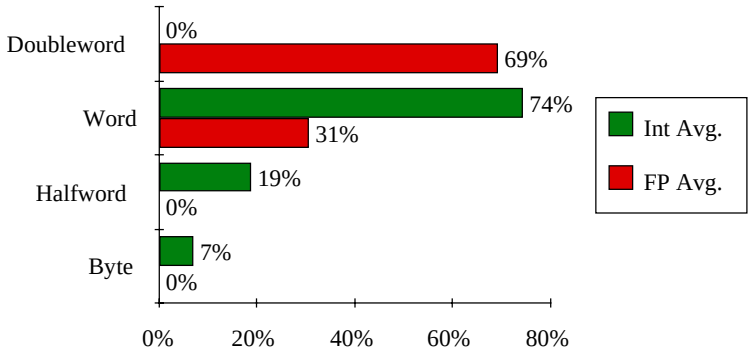
Decimal:
- digits 0-9 encoded as 0000b thru 1001b
- two decimal digits packed per 8 bit byte

Integers:
- 2's Complement

Floating Point:
- Single Precision
- Double Precision
- Extended Precision



Operand Size Usage



Frequency of reference by size

Support these data sizes and types:
8-bit, 16-bit, 32-bit integers and
32-bit and 64-bit IEEE 754 floating point numbers

MIPS arithmetic instructions

Instruction	*Example*	*Meaning*	*Comments*
add	add $1,$2,$3	$1 = $2 + $3	3 operands; exception possible
subtract	sub $1,$2,$3	$1 = $2 – $3	3 operands; exception possible
add immediate	addi $1,$2,100	$1 = $2 + 100	+ constant; exception possible
add unsigned	addu $1,$2,$3	$1 = $2 + $3	3 operands; no exceptions
subtract unsign.	subu $1,$2,$3	$1 = $2 – $3	3 operands; no exceptions
add imm. unsign.	addiu $1,$2,100	$1 = $2 + 100	+ constant; no exceptions
multiply	mult $2,$3	Hi, Lo = $2 x $3	64-bit signed product
multiply unsign.	multu$2,$3	Hi, Lo = $2 x $3	64-bit unsigned product
divide	div $2,$3	Lo = $2 ÷ $3, Hi = $2 mod $3	Lo = quotient, Hi = remainder
divide unsigned	divu $2,$3	Lo = $2 ÷ $3, Hi = $2 mod $3	Unsigned quotient & remainder
Move from Hi	mfhi $1	$1 = Hi	Used to get copy of Hi
Move from Lo	mflo $1	$1 = Lo	Used to get copy of Lo

Which add for address arithmetic? Which add for integers?

MIPS logical instructions

Instruction	*Example*	*Meaning*	*Comment*
and	and $1,$2,$3	$1 = $2 & $3	3 reg. operands; Logical AND
or	or $1,$2,$3	$1 = $2 | $3	3 reg. operands; Logical OR
xor	xor $1,$2,$3	$1 = $2 ⊕ $3	3 reg. operands; Logical XOR
nor	nor $1,$2,$3	$1 = ~($2 |$3)	3 reg. operands; Logical NOR
and immediate	andi $1,$2,10	$1 = $2 & 10	Logical AND reg, constant
or immediate	ori $1,$2,10	$1 = $2 | 10	Logical OR reg, constant
xor immediate	xori $1, $2,10	$1 = ~$2 &~10	Logical XOR reg, constant
shift left logical	sll $1,$2,10	$1 = $2 << 10	Shift left by constant
shift right logical	srl $1,$2,10	$1 = $2 >> 10	Shift right by constant
shift right arithm.	sra $1,$2,10	$1 = $2 >> 10	Shift right (sign extend)
shift left logical	sllv $1,$2,$3	$1 = $2 << $3	Shift left by variable
shift right logical	srlv $1,$2, $3	$1 = $2 >> $3	Shift right by variable
shift right arithm.	srav $1,$2, $3	$1 = $2 >> $3	Shift right arith. by variable

MIPS data transfer instructions

Instruction	*Comment*	*Meaning*
SW R3, 500(R4)	Store word	Mem[R4 + 500] ← R3
SH R3, 502(R2)	Store half	
SB R2, 41(R3)	Store byte	
LW R1, 30(R2)	Load word	R1 ← Mem[R2 + 30]
LH R1, 40(R3)	Load halfword	
LHU R1, 40(R3)	Load halfword unsigned	
LB R1, 40(R3)	Load byte	
LBU R1, 40(R3)	Load byte unsigned	
LUI R1, 40	Load Upper Immediate (16 bits shifted left by 16)	

Why need LUI?