

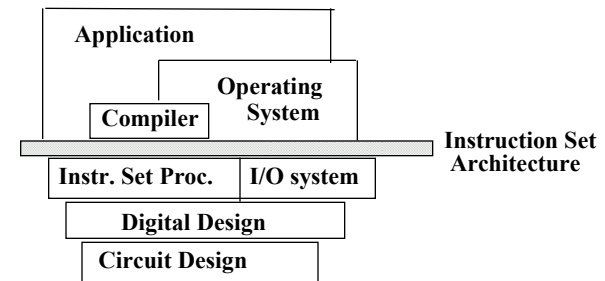
Instruction Set Architecture

"Speaking with the computer"

CSE 141, S2'06

Jeff Brown

The Instruction Set Architecture



CSE 141, S2'06

Jeff Brown

Brief Vocabulary Lesson

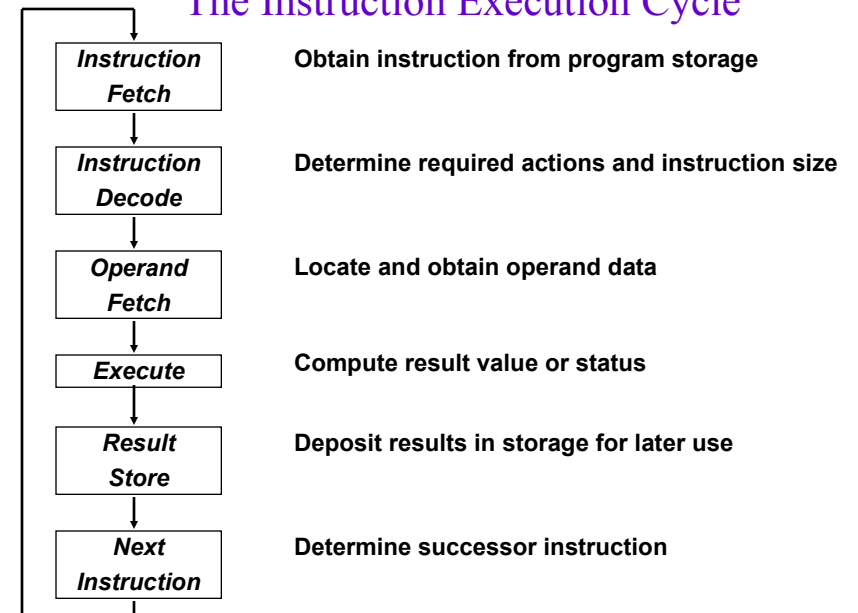
- *superscalar processor* -- can execute more than one instruction per cycle.
- *cycle* -- smallest unit of time in a processor.
- *parallelism* -- the ability to do more than one *thing* at once.
- *pipelining* -- overlapping parts of a large task to increase throughput without decreasing latency



CSE 141, S2'06

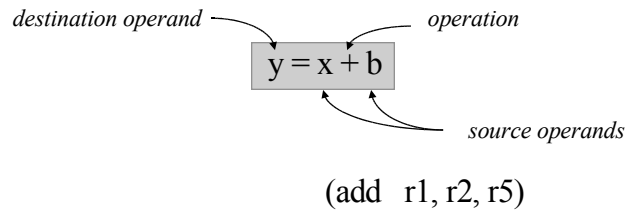
Jeff Brown

The Instruction Execution Cycle



Key ISA decisions

- operations
 - how many?
 - which ones
- operands
 - how many?
 - location
 - types
 - how to specify?
- instruction format
 - size
 - how many formats?



*how does the computer know what
0001 0100 1101 1111
means?*

Crafting an ISA

- We'll look at some of the decisions facing an instruction set architect, and
- how those decisions were made in the design of the MIPS instruction set.

Instruction Length

Variable: ...

Fixed:

Hybrid:

Instruction Length

- Variable-length instructions (Intel 80x86, VAX) require multi-step fetch and decode, but allow for a much more flexible and compact instruction set.
- Fixed-length instructions allow easy fetch and decode, and simplify pipelining and parallelism.

All MIPS instructions are 32 bits long.

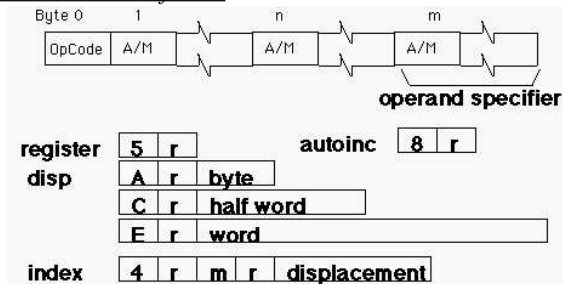
- this decision impacts every other ISA decision we make because it makes instruction bits scarce.

Instruction Formats

-what does each bit mean?

- Having many different instruction formats...
 - complicates decoding
 - uses more instruction bits (to specify the format)

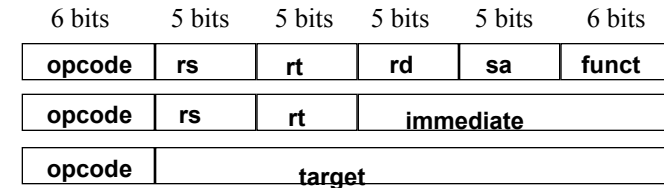
VAX 11 instruction format



CSE 141, S2'06

Jeff Brown

MIPS Instruction Formats



- the opcode tells the machine which format
- so `add r1, r2, r3` has
 - opcode=0, funct=32, rs=2, rt=3, rd=1, sa=0
 - 000000 00010 00011 00001 00000 100000

CSE 141, S2'06

Jeff Brown

Accessing the Operands

- operands are generally in one of two places:
 - registers (32 int, 32 fp)
 - memory (2^{32} locations)
- registers are
 - easy to specify
 - close to the processor (fast access)
- the idea that we want to access registers whenever possible led to *load-store architectures*.
 - normal arithmetic instructions only access registers
 - only access memory with explicit loads and stores

CSE 141, S2'06

Jeff Brown

Load-store architectures

- | | |
|--|--|
| can do: | can't do |
| <code>add r1=r2+r3</code> | <code>add r1 = r2 + M(address)</code> |
| and | |
| <code>load r3, M(address)</code> | |
| ⇒ forces heavy dependence on registers, which is exactly what you want in today's CPUs | - more instructions
+ fast implementation (e.g., easy pipelining) |

CSE 141, S2'06

Jeff Brown

How Many Operands?

- Most instructions have three operands (e.g., $z = x + y$).
- Well-known ISAs specify 0-3 (explicit) operands per instruction.
- Operands can be specified implicitly or explicitly.

CSE 141, S2'06

Jeff Brown

How Many Operands? Basic ISA Classes

Accumulator:

1 address	add A	$acc \leftarrow acc + mem[A]$
-----------	-------	-------------------------------

Stack:

0 address	add	$tos \leftarrow tos + next$
-----------	-----	-----------------------------

General Purpose Register:

2 address	add A B	$EA(A) \leftarrow EA(A) + EA(B)$
3 address	add A B C	$EA(A) \leftarrow EA(B) + EA(C)$

Load/Store:

3 address	add Ra Rb Rc	$Ra \leftarrow Rb + Rc$
	load Ra Rb	$Ra \leftarrow mem[Rb]$
	store Ra Rb	$mem[Rb] \leftarrow Ra$

CSE 141, S2'06

Jeff Brown

Comparing the Number of Instructions

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

<u>Stack</u>	<u>Accumulator</u>	<u>GP Register</u> (register-memory)	<u>GP Register</u> (load-store)
--------------	--------------------	---	------------------------------------

CSE 141, S2'06

Jeff Brown

Comparing the Number of Instructions

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

<u>Stack</u>	<u>Accumulator</u>	<u>GP Register</u> (register-memory)	<u>GP Register</u> (load-store)
Push A	Load A	ADD C, A, B	Load R1,A
Push B	Add B		Load R2,B
Add	Store C		Add R3,R1,R2
Pop C			Store C,R3

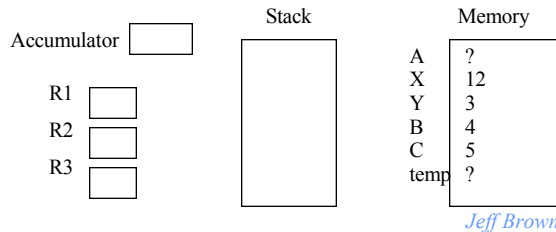
CSE 141, S2'06

Jeff Brown

Alternate ISA's

$$A = X * Y - B * C$$

Stack Architecture Accumulator GPR GPR (Load-store)



CSE 141, S2'06

Jeff Brown

Addressing Modes

how do we specify the operand we want?

- **Register direct** **R3**
- **Immediate (literal)** **#25**
- **Direct (absolute)** **M[10000]**
- **Register indirect** **M[R3]**
- **Base+Displacement** **M[R3 + 10000]**
- **Base+Index** **M[R3 + R4]**
- **Scaled Index** **M[R3 + R4*d + 10000]**
- **Autoincrement** **M[R3++]**
- **Autodecrement** **M[R3--]**
- **Memory Indirect** **M[M[R3]]**

CSE 141, S2'06

Jeff Brown

MIPS addressing modes

register direct

OP	rs	rt	rd	sa	funct
----	----	----	----	----	-------

add \$1, \$2, \$3

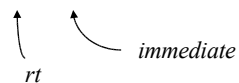
immediate

OP	rs	rt	immediate
----	----	----	-----------

add \$1, \$2, #35

base + displacement

lw \$1, disp(\$2)



$(R1 = M[R2 + disp])$

register indirect
disp = 0
absolute
(rs) = 0

CSE 141, S2'06

Jeff Brown

Is this sufficient?

- measurements on the VAX show that these addressing modes (immediate, direct, register indirect, and base+displacement) represent 88% of all addressing mode usage.
- similar measurements show that 16 bits is enough for the immediate 75 to 80% of the time
- and that 16 bits is enough for branch displacement 99% of the time.
- (so: yes, as long as we can handle all cases, somehow)

CSE 141, S2'06

Jeff Brown

Memory Organization

- Viewed as a large, single-dimension array
- A memory address is an index into the array
- "Byte addressing" means that the index points to a byte of memory.

0	8 bits of data
1	8 bits of data
2	8 bits of data
3	8 bits of data
4	8 bits of data
5	8 bits of data
6	8 bits of data

...

Memory Organization

- Bytes are nice, but most data items use larger "words"
- For MIPS, a word is 32 bits or 4 bytes.

0	32 bits of data
4	32 bits of data
8	32 bits of data
12	32 bits of data

Registers hold 32 bits of data

- 2^{32} bytes with byte addresses from 0 to $2^{32}-1$
- 2^{30} words with byte addresses 0, 4, 8, ... $2^{32}-4$
- Words are "aligned"
(what are the least-significant 2 bits of a word address?)

The MIPS ISA, so far

- fixed 32-bit instructions
- 3 instruction formats
- 3-operand, load-store architecture
- 32 general-purpose registers (integer, floating point)
 - R0 always equals 0.
- registers are 32-bits wide (word)
- 2 special-purpose integer registers, HI and LO, because multiply and divide produce more than 32 bits.
- register, immediate, and base+displacement addressing modes

What's left

- which instructions (operations)?
- odds and ends

Which instructions?

- arithmetic
- logical
- data transfer
- conditional branch
- unconditional jump

CSE 141, S2'06

Jeff Brown

Which instructions (integer)

- arithmetic
 - add, subtract, multiply, divide
- logical
 - and, or, shift left, shift right
- data transfer
 - load word, store word

CSE 141, S2'06

Jeff Brown

Control Flow

- Jump
 - Jump ("goto", "break", ...)
 - Jump subroutine (procedure or function call)
- Conditional branch
 - If-then-else logic, loops, etc.
- A conditional branch must specify two things
 - Condition: determines whether the branch is taken
 - Target: location that the branch jumps to, if taken

CSE 141, S2'06

Jeff Brown

Conditional branch

- How do you specify the destination of a branch/jump?
- studies show that almost all conditional branches go short distances from the current program counter (loops, if-then-else).
 - we can specify a relative address in much fewer bits than an absolute address
 - e.g., beq \$1, \$2, 100 $\Rightarrow$ if (\$1 == \$2) PC = PC + 100 * 4
- How do we specify the **condition** of the branch?

CSE 141, S2'06

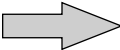
Jeff Brown

MIPS conditional branches

- beq, bne *beq r1, r2, addr* => if ($r1 == r2$) goto *addr*
- slt \$1, \$2, \$3 => if ($\$2 < \$3$) \$1 = 1; else \$1 = 0
- these, combined with \$0, can implement all fundamental branch conditions
Always, never, !=, ==, >, <=, >=, <, ...

```

if (i < j)
    w = w + 1;
else
    w = 5;
    
```



CSE 141, S2'06

Jeff Brown

Jumps

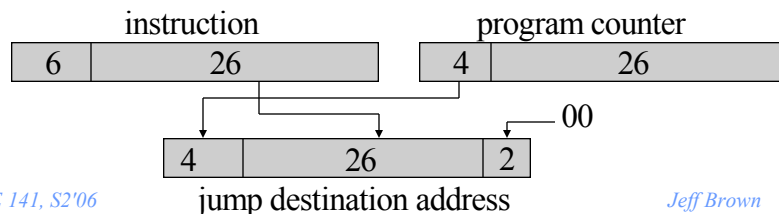
- need to be able to jump to an absolute address sometime
 - need to be able to do procedure calls and returns
 - jump -- j 10000 => PC = 10000
 - jump and link -- jal 100000 => \$31 = PC + 4; PC = 10000
 - used for procedure calls
- | OP | target (26 bits) |
|----|------------------|
|----|------------------|
- jump register -- jr \$31 => PC = \$31
 - used for returns, but can be useful for lots of other things.

CSE 141, S2'06

Jeff Brown

Branch and Jump Addressing Modes

- Branches (e.g., beq) use PC-relative addressing mode.
 - base+displacement mode, with current PC as the base
 - opcode is 6 bits, register numbers are 10 bits; how many bits are available for displacement? How far can you jump?
- Jump uses pseudo-direct addressing mode.
 - The low 26 bits of the target comes directly from the instruction; the rest is taken from the PC. (No addition.)



CSE 141, S2'06

Jeff Brown

To summarize:

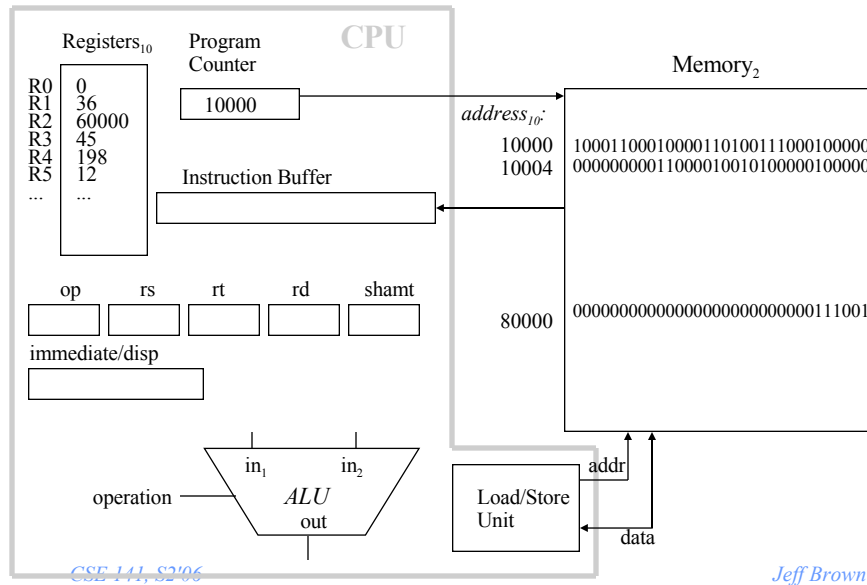
MIPS operands		
Name	Example	Comments
32 registers	\$s0-\$s7, \$t0-\$t9, \$zero, \$a0-\$a3, \$v0-\$v1, \$gp, \$fp, \$sp, \$ra, \$at	Fast locations for data. In MIPS, data must be in registers to perform arithmetic. MIPS register \$zero always equals 0. Register \$at is reserved for the assembler to handle large constants.
2 ³⁰ memory words	Memory[0], Memory[4], ..., Memory[4294967296]	Accessed only by data transfer instructions. MIPS uses byte addresses, so sequential words differ by 4. Memory holds data structures, such as arrays, and spilled registers, such as those saved on procedure calls.

MIPS assembly language				
Category	Instruction	Example	Meaning	Comments
Arithmetic	add	add \$s1, \$s2, \$s3	\$s1 = \$s2 + \$s3	Three operands; data in registers
	subtract	sub \$s1, \$s2, \$s3	\$s1 = \$s2 - \$s3	Three operands; data in registers
	add immediate	addi \$s1, \$s2, 100	\$s1 = \$s2 + 100	Used to add constants
Data transfer	load word	lw \$s1, 100(\$s2)	\$s1 = Memory[\$s2 + 100]	Word from memory to register
	store word	sw \$s1, 100(\$s2)	Memory[\$s2 + 100] = \$s1	Word from register to memory
	load byte	lb \$s1, 100(\$s2)	\$s1 = Memory[\$s2 + 100]	Byte from memory to register
	store byte	sb \$s1, 100(\$s2)	Memory[\$s2 + 100] = \$s1	Byte from register to memory
	load upper immediate	lui \$s1, 100	\$s1 = 100 * 2 ¹⁶	Loads constant in upper 16 bits
Conditional branch	branch on equal	beq \$s1, \$s2, 25	if (\$s1 == \$s2) go to PC+4+100	Equal test; PC-relative branch
	branch on not equal	bne \$s1, \$s2, 25	if (\$s1 != \$s2) go to PC+4+100	Not equal test; PC-relative
	set on less than	slt \$s1, \$s2, \$s3	if (\$s2 < \$s3) \$s1 = 1; else \$s1 = 0	Compare less than; for beq, bne
	set less than immediate	slti \$s1, \$s2, 100	if (\$s2 < 100) \$s1 = 1; else \$s1 = 0	Compare less than constant
Unconditional jump	jump	j 2500	go to 10000	Jump to target address
	jump register	jr \$ra	go to \$ra	For switch, procedure return
	jump and link	jal 2500	\$ra = PC + 4; go to 10000	For procedure call

CSE 141, S2'06

Jeff Brown

Review -- Instruction Execution in a CPU



An Example

- Can we figure out the code?

```
swap(int v[], int k)
{
    int temp;
    temp = v[k];
    v[k] = v[k+1];
    v[k+1] = temp;
}
```



```
swap:
    muli $2, $5, 4
    add $2, $4, $2
    lw $15, 0($2)
    lw $16, 4($2)
    sw $16, 0($2)
    sw $15, 4($2)
    jr $31
```

CSE 141, S2'06

Jeff Brown

MIPS ISA Tradeoffs

	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits
R	OP	rs	rt	rd	sa	funct
I	OP	rs	rt	immediate		
J	OP	target				

What if?

- 64 registers?
- 20-bit immediates
- 4 operand instruction (e.g. $Y = AX + B$)

CSE 141, S2'06

Jeff Brown

RISC Architectures

- MIPS, like SPARC, PowerPC, and Alpha AXP, is a RISC (Reduced Instruction Set Computer) ISA.
 - fixed instruction length
 - few instruction formats
 - load/store architecture
- RISC architectures worked because they enabled pipelining. They continue to thrive because they enable parallelism.

CSE 141, S2'06

Jeff Brown

Alternative Architectures

- Design alternative:
 - provide more powerful or specialized operations
 - goal is to reduce number of instructions executed
 - danger is a slower cycle time and/or a higher CPI (cycles per instruction)
- Sometimes referred to as “RISC vs. CISC”
 - **Reduced (Complex) Instruction Set Computer**
 - virtually all new instruction sets since 1982 have been RISC
 - VAX: minimize code size, make assembly language easy instructions from 1 to 54 bytes long!
- We’ll look (briefly!) at PowerPC and 80x86

CSE 141, S2’06

Jeff Brown

PowerPC

- Indexed addressing
 - example: `lw $t1,$a0+$s3 # $t1=Memory[$a0+$s3]`
 - What do we have to do in MIPS?
- Update addressing
 - update a register as part of load (for marching through arrays)
 - example: `lwu $t0,4($s3) # $t0=Memory[$s3+4]; $s3=$s3+4`
 - What do we have to do in MIPS?
- Others:
 - load multiple/store multiple
 - a special counter register “bc Loop”
decrement counter, if not 0 goto loop

CSE 141, S2’06

Jeff Brown

80x86

- 1978: The Intel 8086 is announced (16 bit architecture)
- 1980: The 8087 floating point coprocessor is added
- 1982: The 80286 increases address space to 24 bits, +instructions
- 1985: The 80386 extends to 32 bits, new addressing modes
- 1989-1995: The 80486, Pentium, Pentium Pro add a few instructions (mostly designed for higher performance)
- 1997: MMX is added
- 1999: Pentium III (same architecture)
- 2001: Pentium 4 (144 new multimedia instructions), simultaneous multithreading (hyperthreading)

CSE 141, S2’06

Jeff Brown

80x86

- See your textbook for a more detailed description
- Complexity:
 - Instructions from 1 to 17 bytes long
 - one operand must act as both a source and destination
 - one operand can come from memory
 - complex addressing modes
e.g., “base or scaled index with 8 or 32 bit displacement”
- Saving grace:
 - the most frequently used instructions are not too difficult to build
 - compilers avoid the portions of the architecture that are slow

CSE 141, S2’06

Jeff Brown

Key Points

- MIPS is a general-purpose register, load-store, fixed-instruction-length architecture.
- MIPS is optimized for fast pipelined performance, not for low instruction count
- Historic architectures favored code size over parallelism.
- MIPS most complex addressing mode, for both branches and loads/stores, is base + displacement.