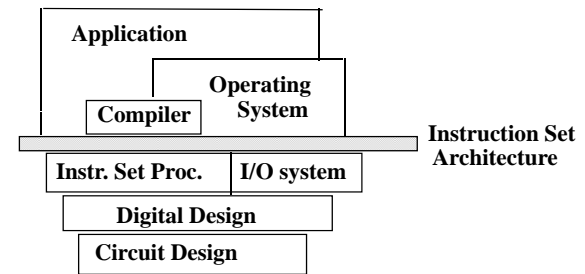


Instruction Set Architecture

or

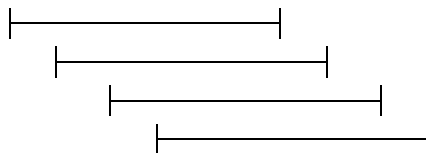
“How to talk to computers if you aren’t in Star Trek”

The Instruction Set Architecture

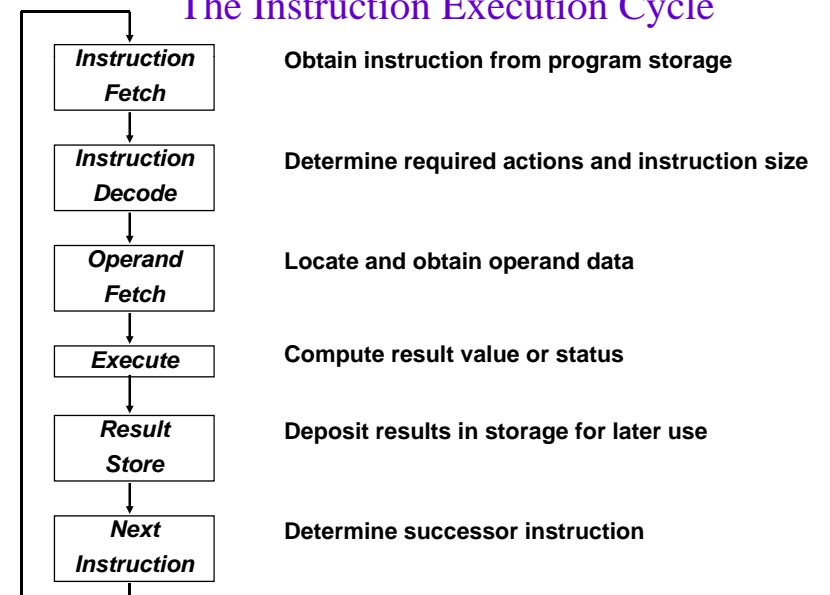


Brief Vocabulary Lesson

- *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.
- -- overlapping parts of a large task to increase throughput without decreasing latency

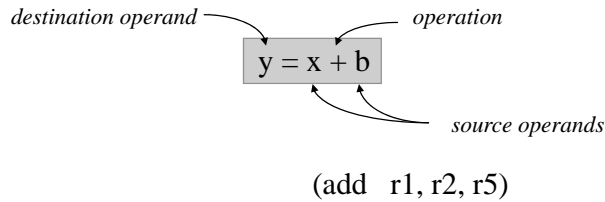


The Instruction Execution Cycle



Key ISA decisions

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

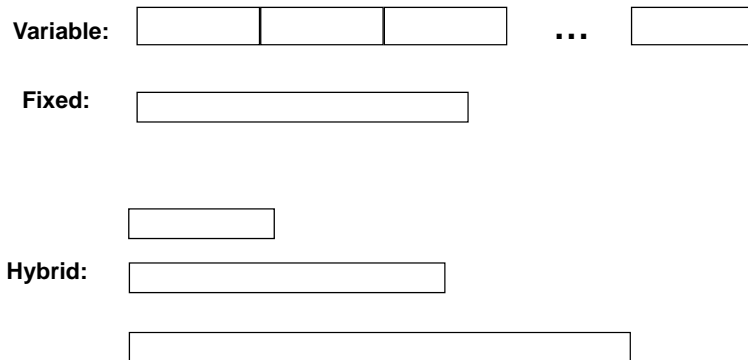


how does the computer know what
0001 0100 1101 1111
means?

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



Instruction Length

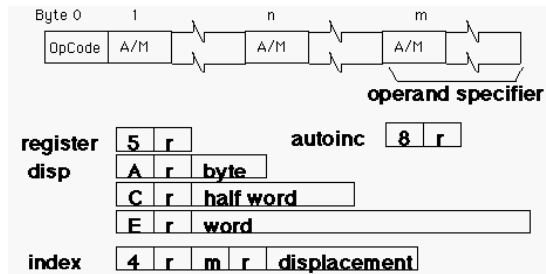
- Variable-length instructions (Intel 80x86, VAX) require multi-step fetch and decode, but allow for a much more flexible and compact instruction set.
 - Fixed-length instructions allow easy fetch and decode, and simplify pipelining and parallelism.
- ⇒ All MIPS instructions are ___ bits long.
- this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

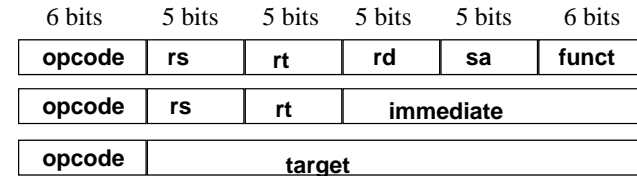
VAX II instruction format



CSE 141

Dean Tullsen

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

Dean Tullsen

Accessing the Operands

- operands are generally in one of two places:
 - (32 int, 32 fp)
 - (2^{32} locations)
- registers are
 -
 -
- 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

Dean Tullsen

Load-store architectures

- can do:
 - `add r1=r2+r3`
- and
 - `load r3, M(address)`
- ⇒ forces heavy dependence on registers, which is exactly what you want in today's CPUs
- can't do
 - `add r1 = r2 + M(address)`

CSE 141

Dean Tullsen

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.

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	$mem[Rb] \leftarrow Ra$

Comparing the Number of Instructions

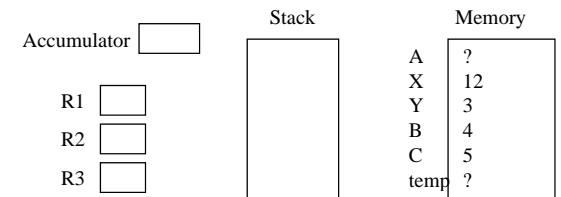
Code sequence for $C = A + B$ for four classes of instruction sets:

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

Alternate ISA's

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

Stack Architecture Accumulator GPR GPR (Load-store)



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

Dean Tullsen

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)

\swarrow *rs*
 \nwarrow *immediate*
 \uparrow *rt*

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

register indirect

\Rightarrow

absolute

\Rightarrow

CSE 141

Dean Tullsen

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 of a displacement 99% of the time.

CSE 141

Dean Tullsen

Memory Organization (digression)

- Viewed as a large, single-dimension array, with an address.
- 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

...

CSE 141

Dean Tullsen

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
i.e., what are the least 2 significant 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.
- 2 special-purpose integer registers, HI and LO, because multiply and divide produce more than 32 bits.
- registers are 32-bits wide (word)
- register, immediate, and base+displacement addressing modes

What's left

- which instructions?
- odds and ends

Which instructions?

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

Which instructions (integer)

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

Control Flow

- Jumps
- Procedure call (jump subroutine)
- Conditional Branch
 - Used to implement, for example, if-then-else logic, loops, etc.
- A conditional branch must specify two things
 - Condition under which the branch is taken
 - Location that the branch jumps to if taken (target)

Conditional branch

- How do you specify the _____ 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, $2, 100` => if ($\$1 == \2) $PC = PC + 100 * 4$
- How do we specify the _____ of the branch?

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, !=, ==, >, <=, >=, <, >(unsigned), <= (unsigned), ...

if ($i < j$)

`w = w+1;`

else

`w = 5;`



MIPS ISA Tradeoffs

6 bits	5 bits	5 bits	5 bits	5 bits	6 bits
OP	rs	rt	rd	sa	funct
OP	rs	rt	immediate		
OP	target				

What if?

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

RISC Architectures

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

Alternative Architectures

- Design alternative:
 - provide more powerful 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, 80x86, and IA64

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

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)
- 2005: dual core Pentium processors
- 2006: quad core Pentium processors
- 2009: Nehalem – eight-core multithreaded processors

CSE 141

Dean Tullsen

80x86

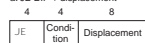
- Complexity:
 - Instructions from 1 to 17 bytes long
 - one operand must act as both a source and destination
 - one operand can come from memory
 - complex addressing modes
 - e.g., “base or scaled index with 8 or 32 bit displacement”
- Saving grace:
 - the most frequently used instructions are not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

CSE 141

Dean Tullsen

x86 Instructions

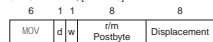
a. JE EIP + displacement



b. CALL



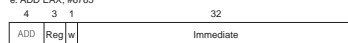
c. MOV EBX, [EDI + 45]



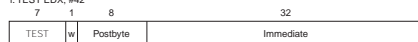
d. PUSH ESI



e. ADD EAX, #6765



f. TEST EDX, #42



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.

CSE 141

Dean Tullsen

CSE 141

Dean Tullsen