Instruction Set Architecture

or

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

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

The Instruction Execution Cycle

1. **Instruction Fetch**: Obtain instruction from program storage
2. **Instruction Decode**: Determine required actions and instruction size
3. **Operand Fetch**: Locate and obtain operand data
4. **Execute**: Compute result value or status
5. **Result Store**: Deposit results in storage for later use
6. **Next Instruction**: Determine successor instruction

Key ISA decisions

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

\[ \text{operation} \quad y = x + b \quad \text{destination operand} \quad \text{source operands} \]

(add \( r_1, r_2, r_5 \))

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.
• 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.

Instruction Length

- Variable: ...
- Fixed: 
- Hybrid: 

Instruction Formats

- what does each bit mean?

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

VAX 11 instruction format

粤港澳

⇒ All MIPS instructions are 32 bits long.
  – this decision impacts every other ISA decision we make because it makes instruction bits scarce.
MIPS Instruction Formats

<table>
<thead>
<tr>
<th>opcode</th>
<th>rs</th>
<th>rt</th>
<th>rd</th>
<th>sa</th>
<th>funct</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>opcode</th>
<th>rs</th>
<th>rt</th>
<th>immediate</th>
</tr>
</thead>
</table>

- 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

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

Load-store architectures

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

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 ← acc + mem[A]

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 the Number of Instructions

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

<table>
<thead>
<tr>
<th>Instruction Set</th>
<th>Code Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stack</td>
<td></td>
</tr>
<tr>
<td>Accumulator</td>
<td></td>
</tr>
<tr>
<td>GP Register</td>
<td></td>
</tr>
<tr>
<td>GP Register</td>
<td></td>
</tr>
</tbody>
</table>

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*4 + 10000]
- Autoincrement: M[R3++]
- Autodecrement: M[R3 - -]
- Memory Indirect: M[M[R3]]
MIPS addressing modes

- **register direct**
  
<table>
<thead>
<tr>
<th>OP</th>
<th>rs</th>
<th>rt</th>
<th>rd</th>
<th>sa</th>
<th>funct</th>
</tr>
</thead>
<tbody>
<tr>
<td>add</td>
<td>$1$, $2$, $3$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **immediate**
  
<table>
<thead>
<tr>
<th>OP</th>
<th>rs</th>
<th>rt</th>
<th>immediate</th>
</tr>
</thead>
<tbody>
<tr>
<td>add</td>
<td>$1$, $2$, $#35$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **base + displacement**
  
<table>
<thead>
<tr>
<th>OP</th>
<th>rs</th>
<th>rt</th>
<th>immediate</th>
</tr>
</thead>
<tbody>
<tr>
<td>lw</td>
<td>$1$, disp($2$)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

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.

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

Which instructions?

- arithmetic
- logical
- data transfer
- conditional branch
- unconditional jump
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 => if ($1 == $2) PC = PC + 100 * 4
- How do we specify the condition 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), ...

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

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

- Jump register -- jr $31 => PC = $31
  - Used for returns, but can be useful for lots of other things.

Branch and Jump Addressing Modes

- Branch (e.g., beq) uses PC-relative addressing mode (uses few bits if address typically close). That is, it uses base+displacement mode, with the PC being the base. If opcode is 6 bits, how many bits are available for displacement? How far can you jump?
- Jump uses pseudo-direct addressing mode. 26 bits of the address is in the instruction, the rest is taken from the PC.

```
6 26
00
```

```
        instruction      program counter
+-----------+------------------+
|          | 26               |
|          |                  |
|          |                  |
|          |                  |
+-----------+------------------+
              4          26
```

Jump destination address
Review -- Instruction Execution in a CPU

MIPS ISA Tradeoffs

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

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 and 80x86

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 “be 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)
- 2000 Pentium 4 (144 new multimedia instructions)

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

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.