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 instructions 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

- **Obtain instruction from program storage**
- **Determine required actions and instruction size**
- **Locate and obtain operand data**
- **Compute result value or status**
- **Deposit results in storage for later use**
- **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?

\[ y = x + b \]

(add r1, r2, r5)
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

• 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

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

VAX 11 instruction format
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>
<tbody>
<tr>
<td>6 bits</td>
<td>5 bits</td>
<td>5 bits</td>
<td>5 bits</td>
<td>5 bits</td>
<td>6 bits</td>
</tr>
</tbody>
</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

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] ]

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 $1, $2, $3</td>
<td></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 $1, $2, #35</td>
<td></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 $1, disp($2)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Memory Organization

- 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.
Memory Organization

- Bytes are nice, but most data items use larger "words"
- For MIPS, a word is 32 bits or 4 bytes.
- $2^{32}$ bytes with byte addresses from 0 to 232-1
- $2^{30}$ words with byte addresses 0, 4, 8, ... 232-4
- 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

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

  \[ \text{beq } r1, r2, addr \Rightarrow \text{if } (r1 == r2) \text{ goto addr} \]

  \[ \text{slt } $1, $2, $3 \Rightarrow \text{if } ($2 < $3) $1 = 1; \text{else} $1 = 0 \]

- these, combined with $0, can implement all fundamental branch conditions
  - Always, never, !=, ==, >, <=, <, >(unsigned), <= (unsigned), ...

  if (i<j)
  \[ \text{w} = \text{w}+1; \]
  else
  \[ \text{w} = 5; \]

Jumps

- need to be able to jump to an absolute address sometime
- need to be able to do procedure calls and returns

  \[ \text{jump -- j 10000} \Rightarrow \text{PC} = 10000 \]
  \[ \text{jump and link -- jal 100000} \Rightarrow \text{$31 = PC + 4}; \text{PC} = 10000 \]
  - used for procedure calls

  \[
  \begin{array}{c|c}
    \text{OP} & \text{target} \\
  \end{array}
  \]

  \[ \text{jump register -- jr$31} \Rightarrow \text{PC} = 31 \]
  - 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.

```
instruction             program counter
6                        4                                26
```

Jump destination address

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;
  }
  ```

```
    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
```
PowerPC

- Indexed addressing
  - example: \( \text{lw $t1,$a0+$s3} \) \( \#t1=\text{Memory[$a0+$s3]} \)
  - What do we have to do in MIPS?

- Update addressing
  - update a register as part of load (for marching through arrays)
  - example: \( \text{lwu $t0,4($s3)} \) \( \#t0=\text{Memory[$s3+4]}; s3=s3+4 \)
  - What do we have to do in MIPS?

- Others:
  - load multiple/store multiple
  - a special counter register “be Loop”
    \( \text{decrement counter, if not 0 goto loop} \)

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
- Some principles of IS architecture
  - Regularity produces simplicity
  - good design demands compromise
  - make the common case fast