Computer Instructions

CSE 410 - Computer Systems
October 3, 2001
Readings and References

• Reading
  – Sections 3.1-3.4, Patterson and Hennessy, Computer Organization & Design

• Other References
  – D Sweetman, *See MIPS Run*, Morgan Kauffman, Publishers
    • section 8.5, Instruction encoding
    • section 11.6, Endianess
A very simple organization

main memory

program counter
registers
functional units
Instructions in main memory

• Instructions are stored in main memory
• Program counter (PC) points to the next instruction
  – All MIPS instructions are 4 bytes long, and so instruction addresses are always multiples of 4
• Program addresses are 32 bits
  – \(2^{32} = 4,294,967,296 = 4\) GigaBytes (GB)
Instructions in memory

<table>
<thead>
<tr>
<th>Instruction addresses</th>
<th>:</th>
<th>:</th>
<th>:</th>
<th>:</th>
<th>:</th>
<th>:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>20</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>instruction value</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>instruction value</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Some common storage units

<table>
<thead>
<tr>
<th>unit</th>
<th># bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte</td>
<td>8</td>
</tr>
<tr>
<td>half-word</td>
<td>16</td>
</tr>
<tr>
<td>word</td>
<td>32</td>
</tr>
<tr>
<td>double word</td>
<td>64</td>
</tr>
</tbody>
</table>
Alignment

- An object in memory is “aligned” when its address is a multiple of its size
- Byte: always aligned
- Halfword: address is multiple of 2
- Word: address is multiple of 4
- Double word: address is multiple of 8
- Alignment simplifies load/store hardware
System organization so far

main memory

instructions and data

32-bit instructions

program counter

increments by 4

registers

functional units
Registers

• 32 bits wide
  – 32 bits is 4 bytes
  – same as a word in memory
  – signed values from $-2^{31}$ to $+2^{31}-1$
  – unsigned values from 0 to $2^{32}-1$

• easy to access and manipulate
  – on chip, so very fast to access
  – 32 registers, so easy to address
Register addresses

- 32 general purpose registers
- how many bits does it take to identify a register?
  - 5 bits, because $2^5 = 32$
- 32 registers is a compromise selection
  - more would require more bits to identify
  - fewer would be harder to use efficiently
## Register numbers and names

<table>
<thead>
<tr>
<th>number</th>
<th>name</th>
<th>usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>zero</td>
<td>always returns 0</td>
</tr>
<tr>
<td>1</td>
<td>at</td>
<td>reserved for use as assembler temporary</td>
</tr>
<tr>
<td>2–3</td>
<td>v0, v1</td>
<td>values returned by procedures</td>
</tr>
<tr>
<td>4–7</td>
<td>a0–a3</td>
<td>first few procedure arguments</td>
</tr>
<tr>
<td>8–15, 24, 25</td>
<td>t0–t9</td>
<td>temps - can use without saving</td>
</tr>
<tr>
<td>16–23</td>
<td>s0–s7</td>
<td>temps - must save before using</td>
</tr>
<tr>
<td>26, 27</td>
<td>k0, k1</td>
<td>reserved for kernel use - may change at any time</td>
</tr>
<tr>
<td>28</td>
<td>gp</td>
<td>global pointer</td>
</tr>
<tr>
<td>29</td>
<td>sp</td>
<td>stack pointer</td>
</tr>
<tr>
<td>30</td>
<td>fp or s8</td>
<td>frame pointer</td>
</tr>
<tr>
<td>31</td>
<td>ra</td>
<td>return address from procedure</td>
</tr>
</tbody>
</table>
How are registers used?

• Many instructions use 3 registers
  – 2 source registers
  – 1 destination register

• For example
  – `add $t1, $a0, $t0`
    • add $a0 and $t0 and put result in $t1
  – `add $t1,$zero,$a0`
    • move contents of $a0 to $t1 ($t1 = 0 + a0)
R-format instructions: 3 registers

- 32 bits available in the instruction
- 15 bits for the 5-bit register numbers
- The remaining 17 bits are available for specifying the instruction
  - 6-bit op code - basic instruction identifier
  - 5-bit shift amount
  - 6-bit function code
## R-format fields

<table>
<thead>
<tr>
<th>op code</th>
<th>source 1</th>
<th>source 2</th>
<th>dest</th>
<th>shamt</th>
<th>function</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>

- Some common R-format instructions
  - Arithmetic: `add`, `sub`, `mult`, `div`
  - Logical: `and`, `or`, `sll`, `srl`
  - Comparison: `slt` (set on less than)
  - Jump through register: `jr`
Bits are just bits

• The bits mean whatever the designer says they mean when the ISA is defined
• How many possible 3-register instructions are there?
  \[ 2^{17} = 131,072 \]
  – includes all values of op code, shamt, function
• As the ISA develops over the years, the encoding tends to become less logical
System organization again

- main memory
- instructions and data
- program counter: increments by 4
- registers: 32 bits wide, 32 in number
- functional units: implement instructions

- 32-bit instructions
Transfer from memory to register

• Load instructions
  - word: \texttt{lw} \ rt, address
  - half word: \texttt{lh} \ rt, address
    \texttt{lhu} \ rt, address
  - byte: \texttt{lb} \ rt, address
    \texttt{lbu} \ rt, address

• signed load $\Rightarrow$ sign bit is extended into the upper bits of destination register

• unsigned load $\Rightarrow$ 0 in upper bits of register
Transfer from register to memory

• Store instructions
  
  – word: sw rt, address
  
  – half word: sh rt, address
  
  – byte: sb rt, address
The “address” term

- There is one basic addressing mode:
  offset + base register value
- Offset is 16 bits (± 32 KB)
- Load word pointed to by s0, add t1, store
  \[\text{lw} \quad \$t0, 0(\$s0)\]
  \[\text{add} \quad \$t0, \$t0, \$t1\]
  \[\text{sw} \quad \$t0, 0(\$s0)\]
I-format fields

- The contents of the base register and the offset value are added together to generate the address for the memory reference.
- Can also use the 16 bits to specify an immediate value, rather than an address.
Instructions and Data flow

- Main memory
- Instructions and data
- Functional units: implement instructions
- Registers: 32 bits wide, 32 in number
- Program counter: increments by 4
The eye of the beholder

• Bit patterns have no inherent meaning
• A 32-bit word can be seen as
  – a signed integer ($\pm$ 2 Billion)
  – an unsigned integer or address pointer (0 to 4B)
  – a single precision floating point number
  – four 1-byte characters
  – an instruction
Big-endian, little-endian

• A 32-bit word in memory is 4 bytes long
• but which byte is which address?
• Consider the 32-bit number 0x01234567
  – four bytes: 01, 23, 45, 67
  – most significant bits are 0x01
  – least significant bits are 0x67
Data in memory- big endian

Big endian - **most** significant bits are in byte 0 of the word

```
<table>
<thead>
<tr>
<th>:</th>
<th>:</th>
<th>:</th>
<th>:</th>
<th>:</th>
<th>:</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>01</td>
<td>23</td>
<td>45</td>
<td>67</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

<table>
<thead>
<tr>
<th>byte #</th>
<th>contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>67</td>
</tr>
<tr>
<td>6</td>
<td>45</td>
</tr>
<tr>
<td>5</td>
<td>23</td>
</tr>
<tr>
<td>4</td>
<td>01</td>
</tr>
</tbody>
</table>

0 1 2 3 ← byte offsets
Data in memory - little endian

Little endian - **least** significant bits are in byte 0 of the word

<table>
<thead>
<tr>
<th>byte</th>
<th>contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>01</td>
</tr>
<tr>
<td>6</td>
<td>23</td>
</tr>
<tr>
<td>5</td>
<td>45</td>
</tr>
<tr>
<td>4</td>
<td>67</td>
</tr>
</tbody>
</table>

byte offsets: 3 2 1 0
Unsigned binary numbers

• Each bit represents a power of 2
• For unsigned numbers in a fixed width field
  – the minimum value is 0
  – the maximum value is $2^n - 1$, where $n$ is the number of bits in the field
• Fixed field widths determine many limits
  – 5 bits = 32 possible values ($2^5 = 32$)
  – 10 bits = 1024 possible values ($2^{10} = 1024$)
## Binary, Hex, and Decimal

<table>
<thead>
<tr>
<th>$2^8=256$</th>
<th>$2^7=128$</th>
<th>$2^6=64$</th>
<th>$2^5=32$</th>
<th>$2^4=16$</th>
<th>$2^3=8$</th>
<th>$2^2=4$</th>
<th>$2^1=2$</th>
<th>$2^0=1$</th>
<th><strong>Hex</strong>$_{16}$</th>
<th><strong>Decimal</strong>$_{10}$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td>0x3</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0x9</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0xA</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0xF</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0x10</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0x1F</td>
<td>31</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0x7F</td>
<td>127</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0xFF</td>
<td>255</td>
</tr>
</tbody>
</table>