The MIPS Instruction Set

- Used as the example throughout the book
- Stanford MIPS commercialized by MIPS Technologies (www.mips.com)
- Large share of embedded core market
  - Applications in consumer electronics, network/storage equipment, cameras, printers, …
  - Typical of many modern ISAs
    - See MIPS Reference Data tear-out card, and Appendixes B and E

Arithmetic Operations

- All arithmetic instructions are 3-address
  - Two sources and one destination
  - \texttt{add a, b, c} \# \texttt{a \leftarrow b + c}

Register Operands

- Arithmetic instructions use register operands and destination
- MIPS has a $32 \times 32$-bit register file
  - Numbered 0 to 31
  - 32-bit data called a “word”
- Assembler names
  - $t0, t1, \ldots, t9$
  - $s0, s1, \ldots, s7$
  - Compilers typically use $tx$ for temps, $sx$ for saved values – there’s no difference and we can use them for anything
Register Operand Example

- C code:
  \[ f = (g + h) - (i + j); \]
  - \( f, \ldots, j \) in \$s0, \ldots, \$s4

- Compiled MIPS code:
  - add \$t0, \$s1, \$s2 \# \$t0 \leftarrow \$s1+\$s2
  - add \$t1, \$s3, \$s4 \# \$t1 \leftarrow \$s3+\$s4
  - sub \$s0, \$t0, \$t1 \# \$s0 \leftarrow \$t0-\$t1

Memory Operands

- Main memory used for composite data
  - Arrays, structures, dynamic data
- To apply arithmetic operations
  - Load values from memory into registers
  - Store result from register to memory

Memory Addressing

- Memory is byte addressed
  - Each address identifies an 8-bit byte
- Words are aligned in memory
  - Address must be a multiple of 4
- MIPS is Big Endian
  - Most-significant byte at least address of a word
  - c.f. Little Endian: least-significant byte at least address

<table>
<thead>
<tr>
<th>Byte Address</th>
<th>Word/Instruction Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>8</td>
</tr>
</tbody>
</table>

Load and Store Instructions

- Use Base Addressing Mode (aka “offset” mode)
  - lw \ rd, offset (rs) \ # load word from memory
    - where:
      - \ rd \ is the destination register
      - \ rs \ contains a base address
      - offset such that base address + offset = address you want
  - sw \ rd, offset (rs) \ # store word to memory
Memory Operand Example 1

- C code:
  
  ```c
  g = h + A[8];
  ```
  
  - Assume the value of g is in $s1, h in $s2, the base address of A in $s3
  
  - Compiled MIPS code:
    ```mips
    lw $t0, 32($s3)   # load word
    add $s1, $s2, $t0
    ```

  Offset = 8 words x 4 bytes/word

Memory Operand Example 2

- C code:
  ```c
  ```

  - h in $s2, base address of A in $s3
  
  - Compiled MIPS code:
    ```mips
    lw $t0, 32($s3)   # load word
    add $t0, $s2, $t0
    sw $t0, 48($s3)   # store word
    ```

Immediate Operands

- Constant data specified in an instruction
  ```mips
  addi $s3, $s3, 4
  ```

- No subtract immediate instruction
  - use a negative constant:
    ```mips
    addi $s2, $s1, -1
    ```
  - Reg $zero gives the constant 0

Unsigned Binary Integers

- Given an n-bit number
  ```latex
  x = \sum_{i=0}^{n-1} x_i 2^i
  ```

- Range: 0 to $2^n - 1$

- Example
  ```latex
  0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 1011_2
  = 0 + ... + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0
  = 0 + ... + 8 + 0 + 2 + 1 = 11_{10}
  ```

- Using 32 bits
  ```latex
  0 to +4,294,967,295```
2s-Complement Signed Integers

- Given an n-bit number
  \[ x = -x_{n-1}2^{n-1} + x_{n-2}2^{n-2} + \cdots + x_12^1 + x_02^0 \]
- Range: \(-2^{n-1}\) to \(+2^{n-1} - 1\)
- Example
  - 1111 1111 1111 1111 1111 1111 1111 1100
  - \(-2 \times 2^{31} + 1 \times 2^{30} + \cdots + 1 \times 2^2 + 0 \times 2^1 + 0 \times 2^0\)
  - \(-2,147,483,648 + 2,147,483,644 = -4_{10}\)
- Using 32 bits
  - \(-2,147,483,648\) to \(+2,147,483,647\)

Adapted from Patterson and Hennessy (Morgan Kaufman Pub)

Sign Extension

- Representing a number using more bits
  - Preserve the numeric value
- In MIPS instruction set
  - addi: extend immediate value
  - lb, lh: extend loaded byte/halfword
  - beq, bne: extend the displacement
  - Replicate the sign bit to the left
    - c.f. unsigned values: extend with 0s
- Examples: 8-bit to 16-bit
  - \(+2: 0000\ 0010\ => 0000\ 0000\ 0000\ 0010\)
  - \(-2: 1111\ 1110\ => 1111\ 1111\ 1111\ 1110\)

Adapted from Patterson and Hennessy (Morgan Kaufman Pub)