MIPS Instruction Set Architecture (I)

Jinkyu Jeong (jinkyu@skku.edu)
Computer Systems Laboratory
Sungkyunkwan University
http://csl.skku.edu
Overview

Textbook: P&H 2.1-2.4 (both 4th Ed. and 5th Ed.)

• Architecture
• Operations of Computer Hardware
• Operands of Computer Hardware
• Signed and Unsigned Operands
Architecture*

* Adapted from the 2012 course slides offered by Prof. Jinsoo Kim (SKKU)
Architecture

“the attributes of a system as seen by the programmer, i.e., the conceptual structure and functional behavior, as distinct from the organization of the data flow and controls, the logical design, and the physical implementation”


• The visible interface between software and hardware
• What the user (OS, compiler, ...) needs to know to reason about how the machine behaves
• Abstracted from the details of how it may accomplish its task
Architecture

• Computer “Architecture” defines
  – Instruction set architecture (ISA)
    • Instruction set
    • Operand types
    • Data types (integers, FPs, ...)
    • Memory addressing modes, ...
  – Registers and other state
  – The interrupt/exception model
  – Memory management and protection
  – Virtual and physical address layout
  – I/O model
  – ...

EEE3050: Theory on Computer Architectures, Spring 2017, Jinkyu Jeong (jinkyu@skku.edu)
Architecture

• Microarchitecture
  – Organization of the machine below the visible level
    • Number/location of functional units
    • Pipeline/cache configurations
    • Programmer transparent techniques: prefetching, ...
  – Must provide same meaning (semantics) as the visible architecture model

• Implementation
  – Hardware realization
  – Logic circuits, VLSI technology, process, ...
Architecture vs. Microarchitecture: ARM

*Source: arm.com*
What is ISA?

Architecture
Brand Name

XScale (IXA)

IA-32

Intel 64
(IA-32e, EM64T, x86-64)

Itanium (IA-64)

Microarchitecture
Brand Name

Processor
Brand Name

P5
Pentium
Pentium MMX

Pentium Pro
Pentium II
Pentium III

NetBurst
Pentium 4
Pentium D

Mobile
Pentium M
Core Duo/Solo

Core
Core 2
Quad/Duo/Solo

Processor
Code Name

P5, P54C, P54CS – P55C, Tillamook

Pentium Pro – Klamath, Deschutes – Katmai, Coppermine, Tualatin

Willamette, Northwood, Prescott, Cedar Mill – Smithfield, Presler

Banias, Dothan

Yonah

Conroe, Allendale, Wolfdale – Merom, Penryn – Kentsfield, Yorkfield

EEE3050: Theory on Computer Architectures, Spring 2017, Jinkyu Jeong (jinkyu@skku.edu)
Operations of Computer Hardware
Instruction Set

• The repertoire of instructions of a computer
• Different computers have different instruction sets
  – But with many aspects in common
• Early computers had very simple instruction sets
  – Simplified implementation
• Many modern computers also have simple instruction sets
  – So-called “RISC” (Reduced Instruction Set Computer)
  – e.g., MIPS, ARM, PowerPC, etc.
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, …
  – Almost 100 million MIPS processors manufactured in 2002
  – Used by NEC, Nintendo, Cisco, Silicon Graphics, Sony, …
• Typical of many modern ISAs
  – See MIPS Reference Data tear-out card (“green card”), and Appendixes B and E
Arithmetic Operations

• Add and subtract, three operands
  – Two sources and one destination
    \[
    \text{add } a, b, c \quad \# \quad a \gets b + c
    \]
• All arithmetic operations have this form

• **Design Principle 1:** Simplicity favors regularity
  – Regularity makes implementation simpler
  – Simplicity enables higher performance at lower cost
Arithmetic Example

• C code:

\[ f = (g + h) - (i + j); \]

• Compiled MIPS code (assembly):

```
add t0, g, h     # temp t0 = g + h
add t1, i, j     # temp t1 = i + j
sub f, t0, t1    # f = t0 - t1
```
Operands of Computer Hardware
Register Operands

- Arithmetic instructions use register operands
- MIPS has a $32 \times 32$-bit register file
  - Use for frequently accessed data
  - Numbered 0 to 31
  - 32-bit data called a “word”
- Assembler names
  - $t0, t1, \ldots, t9$ for temporary values
  - $s0, s1, \ldots, s7$ for saved variables

- **Design Principle 2: Smaller is faster**
  - c.f. main memory: millions of locations
Register Operand Example

• C code:

\[
f = (g + h) - (i + j);
\]

– Assume \( f, g, h, i, j \) are stored in \( s0, s1, s2, s3, s4 \)

• Compiled MIPS code:
Register Operands: MIPS Registers

- 32 general-purpose registers

<table>
<thead>
<tr>
<th>#</th>
<th>Name</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$zero</td>
<td>The constant value 0</td>
</tr>
<tr>
<td>1</td>
<td>$at</td>
<td>Assembler temporary</td>
</tr>
<tr>
<td>2</td>
<td>$v0</td>
<td>Values for results and expression evaluation</td>
</tr>
<tr>
<td>3</td>
<td>$v1</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>$a0</td>
<td>Arguments</td>
</tr>
<tr>
<td>5</td>
<td>$a1</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>$a2</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>$a3</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>$t0</td>
<td>Temporaries (Caller-save registers)</td>
</tr>
<tr>
<td>9</td>
<td>$t1</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>$t2</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>$t3</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>$t4</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>$t5</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>$t6</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>$t7</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>$s0</td>
<td>Saved temporaries</td>
</tr>
<tr>
<td>17</td>
<td>$s1</td>
<td>(Callee-save registers)</td>
</tr>
<tr>
<td>18</td>
<td>$s2</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>$s3</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>$s4</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>$s5</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>$s6</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>$s7</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>$t8</td>
<td>More temporaries</td>
</tr>
<tr>
<td>25</td>
<td>$t9</td>
<td>(Caller-save registers)</td>
</tr>
<tr>
<td>26</td>
<td>$k0</td>
<td>Reserved for OS kernel</td>
</tr>
<tr>
<td>27</td>
<td>$k1</td>
<td></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>$fp</td>
<td>Frame pointer</td>
</tr>
<tr>
<td>31</td>
<td>$ra</td>
<td>Return address</td>
</tr>
</tbody>
</table>
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 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 Addr</th>
<th>Word Addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>32 bits of data 0</td>
</tr>
<tr>
<td>4</td>
<td>32 bits of data 1</td>
</tr>
<tr>
<td>8</td>
<td>32 bits of data 2</td>
</tr>
<tr>
<td>12</td>
<td>32 bits of data 3</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Byte Addresses

- Since 8-bit bytes are so useful, most architectures address individual bytes in memory
  - Alignment restriction - the memory address of a word must be on natural word boundaries (a multiple of 4 in MIPS-32)
- Big Endian: leftmost byte is word address
  - IBM 360/370, Motorola 68k, MIPS, Sparc, HP PA
- Little Endian: rightmost byte is word address
  - Intel 80x86, DEC Vax, DEC Alpha (Windows NT)

[source: M. J. Irwin @ PSU]
Memory Operand Example 1

- **C code:**
  
  g = h + A[8];
  
  - g in $s1, h in $s2, base address of A in $s3

- **Compiled MIPS code:**
  
  - 4 bytes per word
  - Index 8 requires offset of 32
  
  lw $t0, 32($s3)    # load word
  add $s1, $s2, $t0
Memory Operand Example 2

- C code:
  \[
  \]
  - \(h\) in \$s2, base address of \(A\) in \$s3

- Compiled MIPS code:
  - Index 4 requires offset of 16
    (Again, 4 bytes per word)
  \[
  lw \ $t0, -16($s3) \quad \# \text{load word}
  add \ $t0, \ $s2, \ $t0
  sw \ $t0, \ ($s3) \quad \# \text{store word}
  \]
Registers vs. Memory

- Registers are faster to access than memory
- Operating on memory data requires loads and stores
  - More instructions to be executed
- Compiler must use registers for variables as much as possible
  - Only spill to memory for less frequently used variables
  - Register optimization is important!
Immediate Operands

• Constant data specified in an instruction
  \[
  \text{addi } \$s3, \$s3, 4
  \]

• No subtract immediate instruction
  – Just use a negative constant
  \[
  \text{addi } \$s2, \$s1, -1
  \]

• Design Principle 3: Make the common case fast
  – Small constants are common
  – Immediate operand avoids a load instruction
The Constant Zero

- MIPS register 0 ($zero) is the constant 0
  - Cannot be overwritten (i.e., read-only)
- Useful for common operations
  - E.g., move between registers
    add $t2, $s1, $zero
Signed and Unsigned Operands (Review)
Unsigned Binary 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: 0 to \(+2^n - 1\)

- Example
  - 0000 0000 0000 0000 0000 0000 0000 1011_2
  \[= \quad = \quad = \]

- Using 32 bits
  - 0 to \(+4,294,967,295\)
2’s-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

\[ \begin{array}{ccccccccccccccccccccccccccccccc}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & \ 1 & 1 & 0 & 0 \\
\end{array} \]

= \[ \begin{array}{ccccccccccccccccccccccccccccccc}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & \ 0 & 0 \\
\end{array} \]

= \[ \begin{array}{ccccccccccccccccccccccccccccccc}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & \ 0 & 0 \\
\end{array} \]

• Using 32 bits

\([-2,147,483,648\) to \(+2,147,483,647\)
2’s-Complement Signed Integers

• Bit 31 is sign bit
  – 1 for negative numbers
  – 0 for non-negative numbers
• \((-2^{n-1})\) can’t be represented
• Non-negative numbers have the same unsigned and 2s-complement representation
• Some specific numbers
  – 0:
  – \(-1:\)
  – Most-negative:
  – Most-positive:
Signed Negation

• Complement and add 1
  – Complement means 1 → 0, 0 → 1

\[ x + \bar{x} = 1111\ldots111_2 = -1 \]

\[ x + 1 = -x \]

• Example: negate +2
  – +2 = 0000 0000 \ldots 0010_2
  – -2 = 1111 1111 \ldots 1101_2 + 1
    = 1111 1111 \ldots 1100_2
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 =>
  – –2: 1111 1110 =>
MIPS-32 ISA

• Instruction Categories
  – Computational
  – Load/Store
  – Jump and Branch
  – Floating Point
    • coprocessor
  – Memory Management
  – Special

• 3 Instruction Formats: all 32 bits wide

We will learn a variety of instructions defined by MIPS ISA next time!