MIPS Memory & Instruction Representation

CS 64: Computer Organization and Design Logic Lecture #6

Ziad Matni
Dept. of Computer Science, UCSB
Administrative

• Assignment #2: Due Monday 1/30
• Assignment #3: Due Friday 2/3

• Midterm #1 is on Feb. 2\textsuperscript{nd} (a week from today)
  – Material included is: all lectures, all readings (see syllabus), all assignments turned in before 2/2.
  – Will do a review on the lesson before the midterm.
  – If you need to contact DSP, please do so ASAP
**MIDTERM IS COMING!**

- **Thursday, 2/2** in this classroom
- **Starts at 3:30pm **SHARP**
  - Please start arriving 5-10 minutes before class
- **I will chose where you sit!**
  - Wait on my instruction...
- **Duration: 1 hour long**
- **Closed book:** no calculators, no phones, no computers
- Only 1 sheet (single-sided) of written notes and the MIPS R.C.
  - Must be no bigger than 8.5” x 11”
  - You have to turn it in with the exam
- **You will write your answers on the exam sheet itself.**
About the Midterm Exam

• Made up of Multiple Choice & Short Answers.

EXAMPLES:

Complete the following MIPS assembly code that is supposed to add the number in register $t0 to 15:

li $t0, 12
______________

A. add $t2, $t1, $t2
B. addu $t2, $t1, $t2
C. addi $t2, $t0, F
D. addi $t2, $t0, 0x15
E. addui $t2, $t0, 0xF

What is the 2’s complement of 0x5EC?

A. 1x5EC
B. 0x5EC
C. 0xA13
D. 0xA14
E. 0xA15
Lecture Outline

• Refresher material (for Assignment 2)
• MIPS Memory
  – Addressing
  – Allocation Map
  – Big/Little Endian
• MIPS Reference Card
• Instruction Representation
Refresher Material For Assignment 2

• Bit Level Manipulations
• Say, you’re given a 32-bit number: 0xHHHHHHHHHH

• How do you multiply it by $2^n$?
  – You shift it to the LEFT by $n$ places

• How do you set bit $x$ to a 0? to a 1?
  – You AND bit $x$ with a 0; You OR bit $x$ with a 1

• How do you set bits $x_a$ to $x_b$ (in the middle of the word) to another set of bits $y_a$ to $y_b$?
Refresher on Carry and Overflow

• The CARRY bit
  – When doing *unsigned* math, that’s the *only* one we care about

• The OVERFLOW bit
  – When doing *signed* math, that’s the *only* one we care about
  – Overflow can be calculated by the carry bit (and something else)
Addressing Memory

• If you’re not using the `.data` declarations, then you need starting *addresses* of the data in memory with `lw` and `sw` instructions

  Example: `lw $t0, 0x0000400A` (← not a real address)
  Example: `lw $t0, 0x0000400A($s0)` (← not a real address)

• 1 word = 32 bits (in MIPS)
  – So, in a 32-bit unit of memory, that’s 4 bytes
  – Represented with 8 hexadecimals

    `8 x 4 bits = 32 bits... checks out...`

• MIPS addresses sequential memory addresses, but not in “words”
  – Addresses are in Bytes instead
  – MIPS words *must* start at addresses that are multiples of 4
  – Called an *alignment restriction*
NOTE:
Not all memory addresses can be accessed by the programmer.

Although the address space is 32 bits, the top addresses from 0x80000000 to 0xFFFFFFFF are not available to user programs. They are used mostly by the OS.

This is found on your MIPS Reference Card
Mapping MIPS Memory

(say that 10 times fast!)

• Imagine computer memory like a big array of words
• Size of computer memory is:
  \[2^{32} = 4 \text{ Gbits, or 512 MBytes (MB)}\]
  - We only get to use 2 Gbits, or 256 MB
  - That’s (256 MB/ groups of 4 B) = 64 million words
### MIPS Computer Memory Addressing Conventions

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B1</th>
<th>1A</th>
<th>80</th>
<th>C5</th>
<th>29</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0000</td>
<td>52</td>
<td>0x0004</td>
<td>0x0000</td>
<td>0x0001</td>
<td>0x0002</td>
<td>0x0003</td>
</tr>
<tr>
<td>0x0001</td>
<td>00</td>
<td>0x0005</td>
<td>0x0001</td>
<td>0x0002</td>
<td>0x0003</td>
<td>0x0004</td>
</tr>
<tr>
<td>0x0002</td>
<td>37</td>
<td>0x0006</td>
<td>0x0002</td>
<td>0x0003</td>
<td>0x0004</td>
<td>0x0005</td>
</tr>
<tr>
<td>0x0003</td>
<td>EE</td>
<td>0x0007</td>
<td>0x0003</td>
<td>0x0004</td>
<td>0x0005</td>
<td>0x0006</td>
</tr>
<tr>
<td>0x0004</td>
<td>11</td>
<td>0x0008</td>
<td>0x0004</td>
<td>0x0005</td>
<td>0x0006</td>
<td>0x0007</td>
</tr>
<tr>
<td>0x0005</td>
<td>1A</td>
<td>0x0009</td>
<td>0x0005</td>
<td>0x0006</td>
<td>0x0007</td>
<td>0x0008</td>
</tr>
<tr>
<td>0x0006</td>
<td>A5</td>
<td>0x000A</td>
<td>0x0006</td>
<td>0x0007</td>
<td>0x0008</td>
<td>0x0009</td>
</tr>
<tr>
<td>0x0007</td>
<td>-</td>
<td>0x000B</td>
<td>0x0007</td>
<td>0x0008</td>
<td>0x0009</td>
<td>0x000A</td>
</tr>
</tbody>
</table>
MIPS Computer Memory Addressing Conventions

<table>
<thead>
<tr>
<th>B</th>
<th>0x000B</th>
<th>0x000A</th>
<th>0x0009</th>
<th>0x0008</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0x0007</td>
<td>0x0006</td>
<td>0x0005</td>
<td>0x0004</td>
</tr>
<tr>
<td>B1</td>
<td>0x0003</td>
<td>0x0002</td>
<td>0x0001</td>
<td>0x0000</td>
</tr>
<tr>
<td>1A</td>
<td>0x0007</td>
<td>0x0006</td>
<td>0x0005</td>
<td>0x0004</td>
</tr>
<tr>
<td>80</td>
<td>0x0002</td>
<td>0x0001</td>
<td>0x0000</td>
<td></td>
</tr>
<tr>
<td>52</td>
<td>0x0001</td>
<td>0x0000</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

or...

<table>
<thead>
<tr>
<th>B</th>
<th>0x000B</th>
<th>0x000A</th>
<th>0x0009</th>
<th>0x0008</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0x0007</td>
<td>0x0006</td>
<td>0x0005</td>
<td>0x0004</td>
</tr>
<tr>
<td>B1</td>
<td>0x0003</td>
<td>0x0002</td>
<td>0x0001</td>
<td>0x0000</td>
</tr>
<tr>
<td>1A</td>
<td>0x0007</td>
<td>0x0006</td>
<td>0x0005</td>
<td>0x0004</td>
</tr>
<tr>
<td>80</td>
<td>0x0002</td>
<td>0x0001</td>
<td>0x0000</td>
<td></td>
</tr>
<tr>
<td>C5</td>
<td>0x0001</td>
<td>0x0000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>0x0000</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# A Tale of 2 Conventions...

## BIG END (MSByte) gets addressed first

<table>
<thead>
<tr>
<th>BIG END</th>
<th>BIG END</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIG END</td>
<td>BIG END</td>
</tr>
<tr>
<td>BIG END</td>
<td>BIG END</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>0x0000</th>
<th>0x0001</th>
<th>0x0002</th>
<th>0x0003</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0004</td>
<td>0x0005</td>
<td>0x0006</td>
<td>0x0007</td>
</tr>
</tbody>
</table>

## LITTLE END (LSByte) gets addressed first

<table>
<thead>
<tr>
<th>LITTLE END</th>
<th>LITTLE END</th>
</tr>
</thead>
<tbody>
<tr>
<td>LITTLE END</td>
<td>LITTLE END</td>
</tr>
<tr>
<td>LITTLE END</td>
<td>LITTLE END</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>0x0000</th>
<th>0x0001</th>
<th>0x0002</th>
<th>0x0003</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0004</td>
<td>0x0005</td>
<td>0x0006</td>
<td>0x0007</td>
</tr>
</tbody>
</table>

1/26/17
The Use of Big Endian vs. Little Endian

Origin: Jonathan Swift (author) in “Gulliver's Travels”.
Some people preferred to eat their hard boiled eggs from the “little end” first (thus, little endians), while others prefer to eat from the “big end” (i.e. big endians).

• MIPS users typically go with Big Endian convention
  – MIPS allows you to program “endian-ness”

• Most Intel processors go with Little Endian...

• It’s just a convention – it makes no difference to a CPU!
MIPS Reference Card

• Let’s take a closer look at that card...

• Found inside front cover of your textbook
• Also found as PDF on class website
Instruction Representation

Recall: A MIPS instruction has 32 bits
32 bits are divided up into 5 fields *(aka the R-Type format)*

- **op** code 6 bits basic operation
- **rs** code 5 bits first register source operand
- **rt** code 5 bits second register source operand
- **rd** code 5 bits register destination operand
- **shamt** code 5 bits shift amount
- **funct** code 6 bits function code

Why did the designers allocate 5 bits for registers?
Instruction Representation in R-Type

- The combination of the *opcode* and the *funct* code tell the processor what it is supposed to be doing
- Example:

```
add $t0, $s1, $s2
```

<table>
<thead>
<tr>
<th>op</th>
<th>rs</th>
<th>rt</th>
<th>rd</th>
<th>shamt</th>
<th>funct</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>17</td>
<td>18</td>
<td>8</td>
<td>0</td>
<td>32</td>
</tr>
</tbody>
</table>

- op = 0, funct = 32 means “add”
- rs = 17 means “$s1”
- rt = 18 means “$s2”
- rd = 8 means “$t0”
- shamt = 0 means this field is unused in this instruction

A full list of codes can be found in your MIPS Reference Card
Exercises

• Using your MIPS Reference Card, write the 32 bit instruction for the following. Write all the fields in BOTH decimal and hex. Write the final instruction as a hexadecimal.

  add $t3, $t2, $s0
  addu $a0, $a3, $t0
  sub $t1, $t1, $t2
  sll $t0, $t0, 3
• The R-Type format is used for many, but not all instructions
  – Why?

  *Hint: how many registers are there? How bits represent a register in R-Type format?*

• What if you wanted to load/save from/to memory?
  – Why is this problematic with R-Type format?
A Second Type of Format...

32 bits are divided up into 4 fields (the I-Type format)

- **op code** 6 bits  basic operation
- **rs code** 5 bits  first register source operand
- **rt code** 5 bits  second register source operand
- **address code** 16 bits  constant or memory address

**Note:** The I-Type format uses the **address** field to access $\pm 2^{15}$ addresses from whatever value is in the **rs** field
I-Type Format

- The I-Type **address** field is a signed number
  - It can be positive or negative

- The **addi** instruction is an I-Type, example:
  
  \[ \text{addi} \quad \$t0, \quad \$t1, \quad 42 \]
  - What is the largest, most positive, number you can put as an immediate?

  **Ans:** \(2^{15} - 1\)
### Instruction Representation in I-Type

• Example:

```
addi $t0, $s0, 124
```

<table>
<thead>
<tr>
<th>op</th>
<th>rs</th>
<th>rt</th>
<th>address/const</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>8</td>
<td>16</td>
<td>124</td>
</tr>
</tbody>
</table>

- **op** = 8  mean “addi”
- **rs** = 8  means “$t0”
- **rt** = 16 means “$s0”
- **address/const** = 124 is the immediate value

A full list of codes can be found in your MIPS Reference Card
Exercises

• Using your MIPS Reference Card, write the 32 bit instruction for the following. Write all the fields in BOTH decimal and hex. Write the final instruction as a hexadecimal.

addi $t3, $t2, -42
andi $a0, $a3, 1
lw $t1, 0x10008001
YOUR TO-DOs

• Tuesday (1/31): more on MIPS memory
  – Finish reading 2.5, 2.6, 2.8

• Assignment #2: Due Monday 1/30
• Assignment #2: Due Friday 2/3