MIPS Instruction Representation
More Assembly Instruction Use

CS 64: Computer Organization and Design Logic
Lecture #7

Ziad Matni
Dept. of Computer Science, UCSB
• Assignment #3: Due Friday 2/3

• Midterm #1 is on this Thursday Feb. 2\textsuperscript{nd}
  – Material included is: all lectures, all readings (see syllabus), all assignments turned in before 2/2.
  – Review material is on the course website!

• Office Hours you can use THIS WEEK for Midterm
  – Prof. Matni  W 1 – 3 PM  SSMS 4409
  – TA Xiao  W 1 – 2 PM  Trailer 936
  – TA Ozten  W 2 – 3 PM  Trailer 936
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.
What’s on the Midterm??  1/2

• Data Representation
  – Convert bin ↔ hex ↔ decimal ↔ bin
  – Signed and unsigned binaries

• Logic and Arithmetic
  – Binary addition, subtraction
    • Carry and Overflow
  – Bitwise AND, OR, NOT, XOR
  – General rules of AND, OR, XOR, using NOR as NOT

• All demos done in class
• Lab assignments 1 and 2
  – Lab 3 contains stuff that’s included on this list
What’s on the Midterm?? 2/2

Assembly

• Core components of a CPU
  – How instructions work
• Registers ($t, $s, $a, $v)
• Arithmetic in assembly (add, subtract, multiply, divide)
  – What’s the difference between add, addi, addu, addui, etc…
• Conditionals and loops in assembly
• Conversion to and from Assembly and C/C++
• syscall and its various uses (printing output, taking input, ending program)
• .data and .text declarations
• Memory in MIPS
• Big Endian vs Little Endian
• R-type and I-type instructions
• Pseudo instructions
About the Midterm Exam

• Made up of Multiple Choice & Short Answers.
  – To clarify: short answers also means coding

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
Sample Questions

Translate this C-style code into MIPS assembly code:

```c
int a = 10, b = 3;
int c = a + 2 * b;
```

What is the result of these operations?

- $0xF2 \& \neg(0x55)$
- $0x6701 \mid 0x1076$
- $0x102A99D8 \mathbin{\^} 0xAABA11CAB$
Sample Questions

Translate this MIPS assembly code into pseudo-code (C or C++ accepted):

```
li $s0, 2
li $s1, 6
li $t0, 2
add $s2, $s1, $s0
sll $s2, 3
mult $s2, $t0
mflo $s3
```

Translate this C code into MIPS:

```c
int t0 = 3;
int t1 = 2;
if ((t0 + t1) < 7) {
    t1 = 3;
    printf (t1);
} else {
    t1 = t0 + t0;
```
# Midterm Study Guide

- See the Class Website

## Week 1
- **Lecture Slides**
  - Lecture 1 (PDF)
  - Lecture 2 (PDF)

## Week 2
- **Lecture Slides**
  - Lecture 3 (PDF)
  - Lecture 4 (PDF)
- **Example Assembly Code used in class**
  - Basic arithmetic examples
  - Conditional and loop logic (control logic)

## Week 3
- **Lecture Slides**
  - Lecture 5 (PDF)
  - Lecture 6 (PDF)

## Week 4
- **MIDTERM #1 on 2/2**
  - Review: Questions
  - Review: Solutions
- **Lecture Slides**
  - Lecture 7 (PDF)

---

SAMPLE EXAM!
Lecture Outline

• Instruction Representation
  – Review R-Types
  – I-Types

• Bitwise Shift Right

• Pseudo-instructions

• Using syscall for inputs
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* mean “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

Answers:

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Hexadecimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>add $t3, $t2, $s0</td>
<td>0x01505820</td>
</tr>
<tr>
<td>addu $a0, $a3, $t0</td>
<td>0x00E82021</td>
</tr>
<tr>
<td>sub $t1, $t1, $t2</td>
<td>0x012A4822</td>
</tr>
<tr>
<td>sll $t0, $t0, 3</td>
<td>0x000840C0</td>
</tr>
</tbody>
</table>
Instruction Representation

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

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

- Example:

  \texttt{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>

- \( \text{op} = 8 \): mean “addi”
- \( \text{rs} = 8 \): means “\$t0”
- \( \text{rt} = 16 \): means “\$s0”
- \( \text{address/const} = 124 \): 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

Answers:
0x214BFFD6
0x30E40001
A Review of Bitwise Shifting

- Recall: you can bitwise shift a number to the LEFT or to the RIGHT
  - Shifting left: MIPS instruction sll
  - Shifting right: MIPS instruction srl and sra

- Why 2 different ones for shifting right??
  - One is called shift right logical and the other shift right arithmetic
srl vs sra

- srl replaces the “lost” MSBs with 0s
- sra replaces the “lost” MSBs with *either* 0s (if number is +ve) *or* 1s (if number is –ve)

**EXAMPLE:**

```assembly
addi $t0, $zero, 12
addi $t1, $zero, -12
srl $s0, $t0, 1
sra $s1, $t0, 1
srl $s0, $t1, 1
sra $s1, $t1, 1
```

*shiftDemo.asm*
**srl vs sra**

- **srl** replaces the “lost” MSBs with 0s
- **sra** replaces the “lost” MSBs with *either* 0s (if number is +ve) *or* 1s (if number is –ve)

**IMPLICATIONS:**
- **srl** should NOT be used for negative numbers
- **sra** use for positive numbers is redundant
- When using negative numbers, use **sra**
MIPS Pseudo Instructions

• **Definition:** assembler macros that are used to:
  – Do more complicated tasks (because they won’t fit into one 32-bit instruction)
  – Replace an equivalent existing instruction (because they are easier to “understand”)

• Example: load immediate instruction (**li**) and the move instruction (**move**)
Consider that $li \ $t0, 0x0123ABCD cannot fit inside a single 32-bit instruction
— Why?

So it’s actually a macro for something like:

```
li $t0, 0x0123 #load upper immediate
ori $t0, $t0, 0xABCD #OR immediate
```
move

• Consider that \texttt{move $t0, $t1} is a pseudo-instruction for a different reason
  – It’s just easier to read

• So it’s actually a macro for:
  \texttt{add $t0, $t1, $zero}
More on I/O Instructions

• Recall: to call an output, we use syscall which looks at what’s in $v0
  – If $v0 = 1, syscall will print (output) an integer
  – If $v0 = 4, syscall will print (output) a string

• Recall: we can also execute inputs with the same method
  – If $v0 = 5, syscall will get (input) an integer
  – The integer is returned in $v0
Using the $v0 = 5 syscall

- If $v0 = 5, syscall will get (input) an integer
- The integer is returned in $v0

**EXAMPLE:**

```asm
li $v0, 5
syscall
move $t0, $v0
...
li $v0, 1
la $a0, $t0
syscall
```

`inputDemo.asm`
YOUR TO-DOs

• Study for MIDTERM #1 on Thursday 2/2!!!

• Assignment #3: Due Friday 2/3
</LECTURE>