Assembly Language Wrap-Up

We’ve introduced MIPS assembly language
Remember these ten facts about it
1. MIPS is representative of all assembly languages – you should be able to learn any other easily
2. Assembly language is machine language expressed in symbolic form, using decimal and naming
3. R-type instruction \( \text{op } \text{r1}, \text{r2}, \text{r3} \) is \( \text{r1} = \text{r2} \text{ op } \text{r3} \)
4. I-type instruction \( \text{op } \text{r1}, \text{r2}, \text{imm} \) is \( \text{r1} = \text{r2} \text{ op } \text{imm} \)
5. I-type is used for arithmetic, branches, load & store, so the roles of the fields change
6. Moving data to/from memory uses \( \text{imm}($\text{rs}$) \) for the effective address, \( \text{ea} = \text{imm} + \text{$\text{rs}$} \), to reference \( M[\text{ea}] \)

Ten Facts Continued

7. Branch and Jump destinations in instructions refer to words (instructions) not bytes
8. Branch offsets are relative to PC+4
9. By convention registers are used in a disciplined way; following it is wise!
10. “Short form” explanation is on the green card, “Long form” is in appendix B
Instruction Format Review

Register-to-register arithmetic instructions are R-type

<table>
<thead>
<tr>
<th>op</th>
<th>rs</th>
<th>rt</th>
<th>rd</th>
<th>shamt</th>
<th>func</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>

Load, store, branch, & immediate instructions are I-type

<table>
<thead>
<tr>
<th>op</th>
<th>rs</th>
<th>rt</th>
<th>address</th>
</tr>
</thead>
<tbody>
<tr>
<td>6 bits</td>
<td>5 bits</td>
<td>5 bits</td>
<td>16 bits</td>
</tr>
</tbody>
</table>

The jump instruction uses the J-type instruction format

<table>
<thead>
<tr>
<th>op</th>
<th>address</th>
</tr>
</thead>
<tbody>
<tr>
<td>6 bits</td>
<td>26 bits</td>
</tr>
</tbody>
</table>

Consider the assembler for a moment

Recall Assembling

Add: add $4, $3, $2

000000 00011 00010 00100 00000 100000

Load word: lw $5, 8($6)

100011 00110 00101 0000 0000 0000 0000 1000

Branch: bne $7, $2, skip_next_4

000100 00010 00111 0000 0000 0000 0100

Jump: j to_inst_at_memloc_32K

100000 00 0000 0000 0001 0000 0000 0000

Overall process:

C code ⇒ assembly ⇒ binary
Decoding Machine Language

How do we convert 1s and 0s to assembly language and to C code?
Machine language $\Rightarrow$ assembly $\Rightarrow$ C?
For each 32 bits:

1. Look at opcode to distinguish between R-Format, J-Format, and I-Format
2. Use instruction format to determine which fields exist
3. Write out MIPS assembly code, converting each field to name, register number/name, or decimal/hex number
4. Logically convert this MIPS code into valid C code. Always possible? Unique?

Decoding (1/7)

Here are six machine language instructions in hexadecimal:

- $00001025_{\text{hex}}$
- $0005402A_{\text{hex}}$
- $11000003_{\text{hex}}$
- $00441020_{\text{hex}}$
- $20A5FFFF_{\text{hex}}$
- $08100001_{\text{hex}}$

Let the first instruction be at address $4,194,304_{\text{ten}}$ ($0x00400000_{\text{hex}}$)
Next step: convert hex to binary
Decoding (2/7)

The six machine language instructions in binary:

00000000000000000001000000100101
00000000000001010100000000101010
00010001000000000000000000000011
00000000010001000010010101000000001
00100000100101111111111111111111
00001000001000000000000000000001

Next step: identify opcode and format

<table>
<thead>
<tr>
<th>R</th>
<th>rs</th>
<th>rt</th>
<th>rd</th>
<th>shamt</th>
<th>funct</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,4-62</td>
<td>rs</td>
<td>rt</td>
<td>immediate</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 or 3</td>
<td>target address</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Decoding (3/7)

Select the opcode (first 6 bits) to determine the format:

<table>
<thead>
<tr>
<th>R</th>
<th>rs</th>
<th>rt</th>
<th>rd</th>
<th>shamt</th>
<th>funct</th>
</tr>
</thead>
<tbody>
<tr>
<td>000000</td>
<td>000000000000000000000100101</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000000</td>
<td>000000000000000101010001010</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000100</td>
<td>010000000000000000000000011</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000000</td>
<td>00100000000000000000000001000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>001000</td>
<td>001001010101010101010101010</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000010</td>
<td>000000000000000000000000000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Look at opcode: 0 means R-Format, 2 or 3 mean J-Format, otherwise I-Format

Next step: separation of fields R R I R I J Format:

<table>
<thead>
<tr>
<th>R</th>
<th>rs</th>
<th>rt</th>
<th>rd</th>
<th>shamt</th>
<th>funct</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,4-62</td>
<td>rs</td>
<td>rt</td>
<td>immediate</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 or 3</td>
<td>target address</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Decoding (4/7)

Fields separated based on format/opcode:

<table>
<thead>
<tr>
<th>Format</th>
<th>R</th>
<th>R</th>
<th>I</th>
<th>J</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>5</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>0</td>
<td>+3</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>4</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>5</td>
<td>5</td>
<td>-1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1,048,577</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Next step: translate ("disassemble") MIPS assembly instructions R R I R I J Format:

Decoding (5/7)

MIPS Assembly (Part 1):

Address:          Assembly instructions:
0x00400000        or     $2,$0,$0
0x00400004        slt    $8,$0,$5
0x00400008        beq    $8,$0,3
0x0040000c        add    $2,$2,$4
0x00400010        addi   $5,$5,-1
0x00400014        j      0x100001

Better solution: translate to more meaningful MIPS instructions (fix the branch/jump and add labels, registers)
Decoding (6/7)

MIPS Assembly (Part 2):

```mips
or    $v0,$0,$0
Loop:  slt   $t0,$0,$a1
       beq   $t0,$0,Exit
       add   $v0,$v0,$a0
       addi  $a1,$a1,-1
       j    Loop

Exit:

Next step: translate to C code (must be creative!)
```

Decoding (7/7)

After C code

```c
$v0$: var1  
$a0$: var2  
$a1$: var3  

var1 = 0;
while (var3 > 0) {
    var1 += var2;
    var3 -= 1;
}
```

```mips
or    $v0,$0,$0
Loop:  slt   $t0,$0,$a1
       beq   $t0,$0,Exit
       add   $v0,$v0,$a0
       addi  $a1,$a1,-1
       j    Loop

Exit:
```