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 op $r1,$r2,$r3 is $r1 = $r2 op $r3
4. I-type instruction op $r1,$r2,imm is $r1 = $r2 op imm
5. I-type is used for arithmetic, branches, load & store, so the roles of the fields change
6. Moving data to/from memory uses imm($rs) for the *effective address,* ea = imm + $rs, to reference M[ea]

1

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

2

Instruction Format Review

Register-to-register arithmetic instructions are R-type

op	rs	rt	rd	shamt	func
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

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

op	rs	rt	address
6 bits	5 bits	5 bits	16 bits

The jump instruction uses the J-type instruction format

op	address
6 bits	26 bits

Consider the assembler for a moment

3

Recall Assembling

Add: add $4, $3, $2

000000 00011 00010 00100 00000 10 0000

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

100011 00110 00101 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

4

Decoding Machine Language

How do we convert 1s and 0s to assembly language and to C code?

Machine language ⇒ assembly ⇒ C?

For each 32 bits:

1. Look at opcode to distinquish 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?

5

Decoding (1/7)

Here are six machine language instructions in hexadecimal:

00001025_{hex}

$0005402A_{hex}$

11000003_{hex}

00441020_{hex}

$20A5FFFF_{hex}$

08100001_{hex}

Let the first instruction be at address $4,194,304_{ten}$ ($0x00400000_{hex}$)

Next step: convert hex to binary

6

Decoding (2/7)

The six machine language instructions in binary:

```
00000000000000000001000000100101
00000000000001010100000000101010
00010001000000000000000000000011
00000000010001000001000000100000
00100000101001011111111111111111
00001000000100000000000000000001
```

Next step: identify opcode and format

R	0	rs	rt	rd	shamt	funct
I	1, 4-62	rs	rt	immediate		
J	2 or 3	target address				

7

Decoding (3/7)

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

R	000000	00000	00000	00010	00000	100101
R	000000	00000	00101	01000	00000	101010
I	000100	01000	00000	0000000000000011		
R	000000	00010	00100	00010	00000	100000
I	001000	00101	00101	1111111111111111		
J	000010	00000100000000000000000001				

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:

R	0	rs	rt	rd	shamt	funct
I	1, 4-62	rs	rt	immediate		
J	2 or 3	target address				

8

Decoding (4/7)

Fields separated based on format/opcode:

Format:						
R	0	0	0	2	0	37
R	0	0	5	8	0	42
I	4	8	0	+3		
R	0	2	4	2	0	32
I	8	5	5	-1		
J	2	1,048,577				

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

9

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)

10

Decoding (6/7)

MIPS Assembly (Part 2):

```
        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!)

11

Decoding (7/7)

After C code

```
$v0: var1
$a0: var2
$a1: var3
var1 = 0;
while (var3 > 0) {
      var1 += var2;
      var3 -= 1;
}
```

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

12