CSCE 611: Brief Review of MIPS Architecture

Instructor: Jason D. Bakos

UNIVERSITY OF SOUTH CAROLINA

MIPS Architecture

- Three instruction formats:
  - **R-Type**: register operands
    - Computational instructions (arithmetic, logical, shift, comparison)
    - E.g. add $1, $2, $3
  - **I-Type**: immediate operand
    - Branches, load/stores, and computational instructions that involve a constant operand (e.g. increments)
  - **J-Type**: for jumping
    - Only two instructions, J and JAL

R-Type

- *Register-type*
- Fields:
  - **rs, rt**: source registers
  - **rd**: destination register
  - **op**: the *operation code* or *opcode* (0 for R-type instructions)
  - **funct**: the *function* processor what operation to perform
  - **shamt**: the *shift amount* for shift instructions, otherwise it's 0

R-Type

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

R-Type Examples

Assembly Code

```
add $s0, $s1, $s2

sub $t0, $t3, $t5
```

Field Values

op	rs	rt	rd	shamt	funct
0	17	18	16	0	32
0	11	13	8	0	34
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

Machine Code

op	rs	rt	rd	shamt	funct	
000000	10001	10010	10000	00000	100000	(0x02328020)
000000	01011	01101	01000	00000	100010	(0x016D4022)
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	

Note the order of registers in the assembly code:

```
add rd, rs, rt
```

I-Type

- *Immediate-type*
- Fields:
  - **rs, rt**: register operands
  - **imm**: 16-bit two’s complement immediate
  - **op**: the opcode
  - Operation is completely determined by the opcode

I-Type

op	rs	rt	imm
6 bits	5 bits	5 bits	16 bits

I-Type Examples

Assembly Code / Field Values

Assembly Code	op	rs	rt	imm
`addi $s0, $s1, 5`	8	17	16	5
`addi $t0, $s3, -12`	8	19	8	-12
`lw   $t2, 32($0)`	35	0	10	32
`sw   $s1,  4($t1)`	43	9	17	4
	6 bits	5 bits	5 bits	16 bits

Machine Code

op	rs	rt	imm	
001000	10001	10000	0000 0000 0000 0101	(0x22300005)
001000	10011	01000	1111 1111 1111 0100	(0x2268FFF4)
100011	00000	01010	0000 0000 0010 0000	(0x8C0A0020)
101011	01001	10001	0000 0000 0000 0100	(0xAD310004)
6 bits	5 bits	5 bits	16 bits	

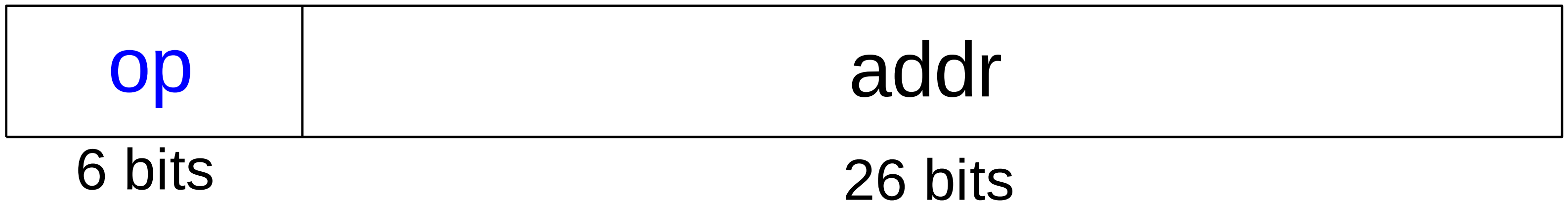
Note the differing order of registers in the assembly and machine codes:

```
addi rt, rs, imm
lw   rt, imm(rs)
sw   rt, imm(rs)
```

J-Type

- *Jump-type*
- 26-bit address operand (addr)
- Used for jump instructions (j)

J-Type



Review: Instruction Formats

R-Type

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

I-Type

op	rs	rt	imm
6 bits	5 bits	5 bits	16 bits

J-Type

op	addr
6 bits	26 bits

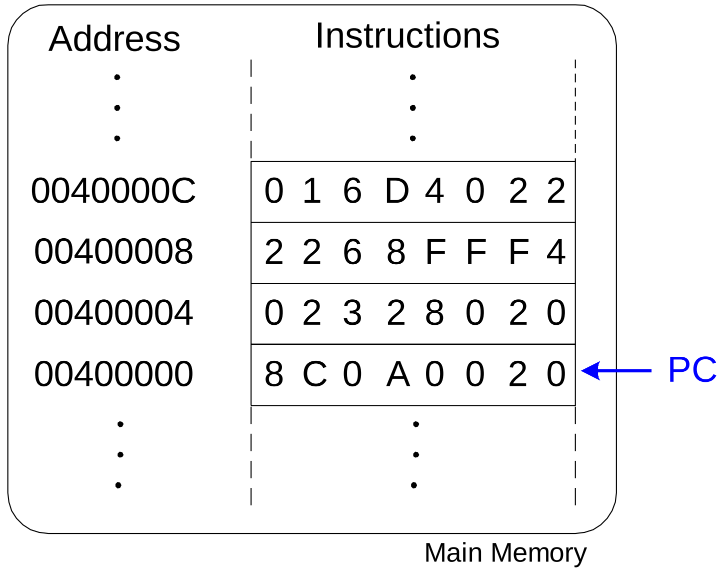
UNIVERSITY OF SOUTH CAROLINA

The Stored Program

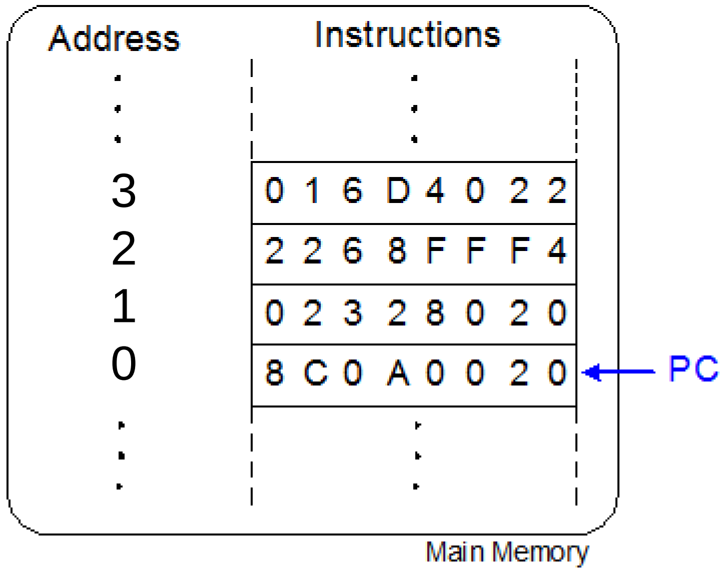
Assembly Code	Machine Code
`lw   $t2, 32($0)`	`0x8C0A0020`
`add  $s0, $s1, $s2`	`0x02328020`
`addi $t0, $s3, -12`	`0x2268FFF4`
`sub  $t0, $t3, $t5`	`0x016D4022`

- Our instruction memory is word addressed
- Program begins at address 0

Stored Program

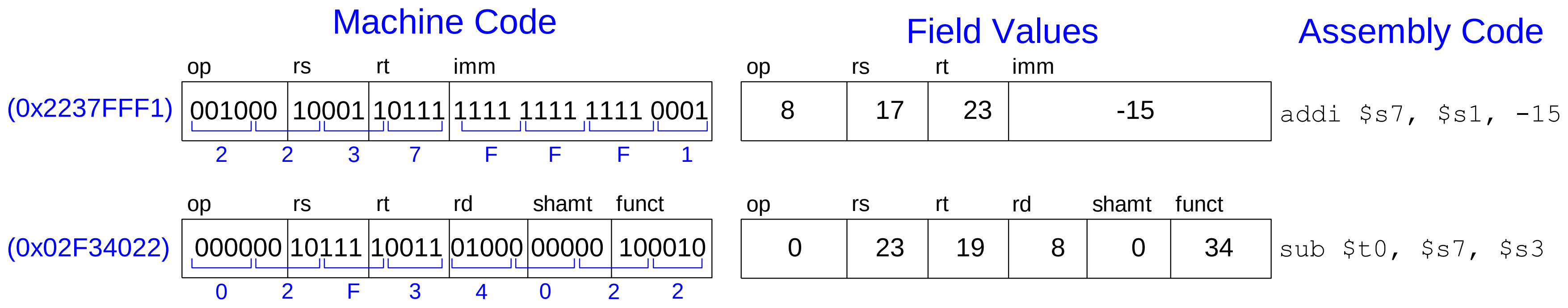


Stored Program



Interpreting Machine Language Code

- Opcode tells how to parse the remaining bits
- If opcode is all 0’s
  - R-type instruction
  - Function bits tell what instruction it is
- Otherwise
  - opcode and rt field (for some branches) tells what instruction it is



	op	rs	rt	imm			
(0x2237FFF1)	001000	10001	10111	1111 1111 1111 0001			
	op	rs	rt	rd	shamt	funct	
(0x02F34022)	000000	10111	10011	01000	00000	100010	

op	rs	rt	imm			Assembly Code
8	17	23	-15			`addi $s7, $s1, -15`
op	**rs**	**rt**	**rd**	**shamt**	**funct**	
0	23	19	8	0	34	`sub $t0, $s7, $s3`

Load/Store Addressing

Base-Offset Addressing

- Address of operand is:

  `base address + sign-extended immediate`

  - Example: `lw  $s4, 72($0)`
    - Address = `$0 + 72`
  - Example: `sw  $t2, -25($t1)`
    - Address = `$t1 - 25`

Branch Addressing

PC-Relative Addressing

```
0x10            beq   $t0, $0, else
0x14            addi  $v0, $0, 1
0x18            addi  $sp, $sp, i
0x1C            jr          $ra
0x20    else:   addi  $a0, $a0, -1
0x24            jal   factorial
```

Assembly Code | **Field Values**

`beq $t0, $0, else`
`(beq $t0, $0, 3)`

op	rs	rt	imm
4	8	0	3
6 bits	5 bits	5 bits	5 bits 5 bits 6 bits

Jump Addressing

Pseudo-direct Addressing

```
0x0040005C          jal   sum
...
0x004000A0   sum:   add   $v0, $a0, $a1
```

JTA 0000 0000 0100 0000 0000 0000 1010 0000 (0x004000A0)

26-bit addr 0000 0000 0100 0000 0000 0000 1010 0000 (0x0100028)

$\underbrace{00}_{0}\ \underbrace{00\ 01}_{1}\ \underbrace{00\ 00}_{0}\ \underbrace{00\ 00}_{0}\ \underbrace{00\ 00}_{0}\ \underbrace{10\ 10}_{2}\ \underbrace{10\ 00}_{8}$

Field Values

op	imm
3	0x0100028
6 bits	26 bits

Machine Code

op	addr	
000011	00 0001 0000 0000 0000 0010 1000	(0x0C100028)
6 bits	26 bits	

32 MIPS Instructions

• Arithmetic R-type:	add, addu, sub, subu, mult, multu
• Arithmetic I-type:	addi, addiu
• Comparison R-type:	slt, sltu
• Comparison I-type:	slti
• Logical R-type:	and, or, nor, xor
• Logical I-type:	andi, ori, xori
• Shift R-type:	sll, srl, sra
• Load/Store I-type:	lw, sw
• Branch I-type:	beq, bne, bgez
• Jump J-type:	j, jal
• Jump R-type:	jr
• Movement:	mfhi, mflo
• Miscellaneous:	nop

MIPS Registers

- 32 general purpose *integer* registers
  - Some have special purposes
  - These are the only registers the programmer can directly use
    - $0 => constant 0
    - $1 => $at (reserved for assembler)
    - $2,$3 => $v0,$v1 (function return value)
    - $4-$7 => $a0-$a3 (function arguments)
    - $8-$15 => $t0-$t7 (temporary values)
    - $16-$23 => $s0->$s7 (function local variables)
    - $24, $25 => $t8, $t9 (temporary values)
    - $26,$27 => $k0, $k1 (reserved for OS kernel)
    - $28 => $gp (pointer to global area)
    - $29 => $sp (stack pointer)
    - $30 => $fp (frame pointer)
    - $31 => $ra (function return address)
- Program counter (PC) contains address of next instruction to be executed

MARS

- Invoke MARS using:

```
java -jar /usr/local/3rdparty/csce611/MARS/Mars.jar
```

Review: MIPS Code Example

```
for (i=0;i<n;i++) a[i]=b[i]+10;
```

```
        li $s0,0              # i=0
        lw $s1,n              # load n
        sll $s1,$s1,2         # n = n * 4
loop:   bge $s0,$s1,exit      # check !(i < n)
        lw $s3,b($s0)         # load b
        addi $s3,$s3,10       # add 10
        sw $s3,a($s0)         # store to a
        addi $s0,$s0,4        # i += 4
        j loop                # loop
```