Ramiro Gonzalez **Homework/Lab 01** CSE 140 February 18, 2020

Using Ubuntu Linux

Activity 1

1. Record the name(s) of your partner(s): Ramiro Gonzalez, Nikko Solon, Nathan
2. MIPS_Reference_Sheet
3. Load proc1.s
   (a) Download and Install Mars. Link
   (b) Running Mars
   ```
           $ sudo chmod 777 Mars4_5.jar
           $ java -jar Mars4_5.jar
   ```
   (c) Open proc1.s using mars GUI
4. After assembling the program, study the **Text Segment** window and see how your source code is translated into True Assembly Language (Basic) as well as machine code (Code).

Line	MIPS	Basic	Machine Code
7	addi $s0, $zero, 25	addi $16, $0, 0x000000019	0x20100019

   - Conventional Name for the save registers are $s0-$s7 and in True Assembly Language (Basic) they are register $16-$23
   - Numbers are converted to hexadecimal
   - Machine code is represented in Hex
5. In true assembly language, every single instruction can be translated into a machine instruction. How many bits does a machine instruction contain? A machine instruction contains 32 bits. R-Format, I-Format, J-Format contain 32 bits.
6. To utilize the limited number of bits efficiently, all machine instructions are categorized into different types (or formats). How many types are there? What are they? Give 2 operations for each type as examples.
   - There are 3 formats.
     (a) R-Format (Arithmetic Logical)
     (b) I-Format (Immediate format)
     (c) J-Format (Jump Format)
   - For R-Format

```
        add $t0, $s1, $s2
        sll $t2, $s0, 4
```

- I-Format (Immediate Format)

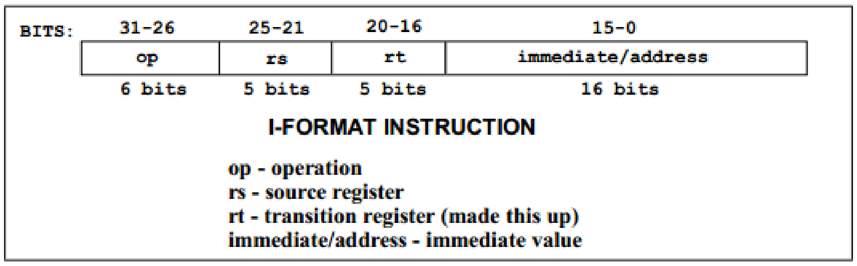
```
        addi $t0, $s1, 5
        lw $t0, 8($s3)
```

- J-Format

```
        j Exit
        jal $ra
```

7. Now, locate the instruction in line #7 of proc1.s. Let's translate this instruction into machine code.

   (a) What instruction type is this? How many fields does this type of instruction have? What are the names of these fields?



BITS:	31-26	25-21	20-16	15-0
	op	rs	rt	immediate/address
	6 bits	5 bits	5 bits	16 bits

I-FORMAT INSTRUCTION

op - operation
rs - source register
rt - transition register (made this up)
immediate/address - immediate value

   - This is type I-format (immediate)
   - This type of instruction has 4 fields.
   - The name of the fields are **op, rs, rt, immediate/address**

   (b) Refer to the **MIPS sheet**, what is the value of the **opcode** of this instruction in **Hex?** What register is **rs**? What is the value of this in **Hex?** What is rt? What is the value of this register in **Hex?**. What is the value of **immediate** in **Hex?**

   - 8_{hex}
   - \$s0 or register 16 is **rs**?
   - The value in hex 0x10

   (c) Construct the machine code of line #7 using the values obtained from **part b.** Write your answer in both **binary** and **Hex** formats. You can verify your answer with the Code column in Text Segment window.

```
    0x20100019
```

```
0010 0000 0001 0000 0000 0000 0001 1001
```

8. Now, let's convert a machine code to MIPS instruction. Locate address 0x00400010 from the *Text Segment* window.

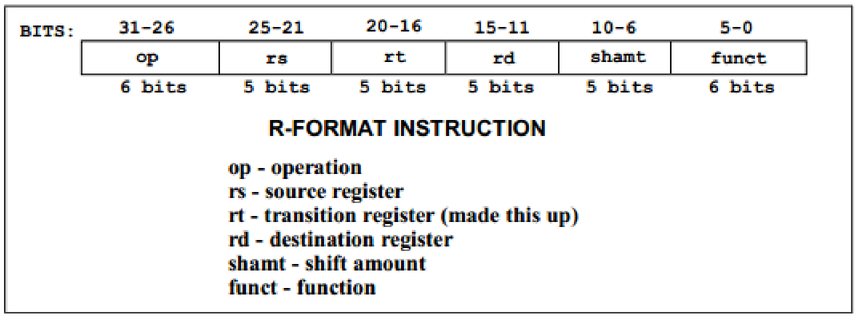
   (a) What is the machine code at this adress in **Hex**? Convert this code into **binary**.

   - The machine code is

   ```
   0x0230402a
   ```

   ```
   0000 0010 0011 0000 0100 0000 0010 1010
   ```

   (b) From the binary version of this machine code. What is the instruction type? How can you tell? How many fields are there in this instruction type? What are the names of these fields?



   - 0 is the opcode and 2a is the funct. This instruction is
   - This instruction is R-format because the opcode is `0`, (000000)
   - There are 6 fields
   - **op, rs, rt,rd shamt, funct**

   (c) According to the binary machine code, what is the value of each field in Hex?

   - The value in hex

   ```
   [0000 00] [10 001] [1 0000] [0100 0] [000 00] [10 1010] — Binary
   ```

   ```
   [0] [17] [16] [8] [0] [42] — Decimal
   ```

   ```
   [0x0] [0x11] [0x10] [0x8] [0x0] [02a] — Hex
   ```

   (d) Refer to the MIPS sheet, what operation is this instruction? How can you tell? What is the mapping of the registers being used in this instruction?

- It is an arithmetic logical (R-Format), the operation is SLT (Set Less Than).
- You can tell by the opcode (0x0) and function (0x2a)
- The opcode for all arithmetic logical (R-Type) instructions are `0`
- The mapping of the registers is as follows **rs** register 17 (\$s1), **rt** is register 16 (\$s0), and **rd** is register 8 (\$t0).

(e) What is the final MIPS instruction? Is it the same as the Source column in the Text Segment window?

- Yes, the instructions are similar to the *Text Segment window*

```
            SLT \$t0, $s1, $s0
```

9. Now, let's take a look at line #17 proc1.s

```
    bne $t0, $zero, LESS
```

(a) What format is this instruction

- I-Format (Immediate)

(b) What are the values of **opcode**, **rs**, and **rt** of this instruction in **hex**

- **opcode**: 5_{hex}, **rs**: 8_{hex}, **rt**: 0_{hex}

(c) What is the name of the target label if it takes the branch? What is the address of this label in **hex**? (Hint: you can find it in the Text Segment window.)

- The name of the target label is `LESS`
- The address of the label `LESS` is 0x0004001c

(d) So, do we put this address as the value of the immediate field of the instruction? Why?

- No, the immediate field contains the offset

(e) How do we find the value of the **immediate field?** What is this value?

- Address of *bne* instruction is 0x00400014
- The offset (immediate field is) 0x00000001
- The address of `LESS` is 0x0004001c

i. Compute Byte Offset
(pc + 4) - Branch Destination Address = (0x00400014 + 4) - 0x004001c = -0x4

ii. Computer Word Offset
-0x4/4 = -0x1

iii. Extend it to 16 bits

iv. Compute 2's complement
We start at a lower address and end at a higher address. (lower address to higher is positive offset, higher to lower address is negative address (requires 2's complement))

v. 0x00000001

(f) What is the machine code of this instruction in **binary** and **hex** formats? Does your answer match the Code column in the *Text Segment windows?*

- Fields for I-format: [5_{hex} | 8_{hex} | 0_{hex} | 1_{hex}]
- Binary: 0001 0101 0000 0000 0000 0000 0000 0001
- Hex: 0x15000001
- Yes it is the same as the *Text Segment window*

10. Finally, let's convert the j instruction in

```
    j GREQ
```

line #**20**.

(a) What format is this instruction? How many fields are there in this format?

- I-Format
- There are two fields, the **opcode, 26 bit address of the destination**

(b) What is the opcode of this instruction in **hex?**

- **opcode** 2_{hex}

(c) What label and **address** does this instruction jump to?

- The instruction jumps to the address is 0x00400030 with the label `GREQ`

(d) How many bits can you use in the address field of the instruction? How can we `squeeze` the address into this field? What are the reasons behind this approach? What is the value of the address field in **binary?**

- The address field for the J instruction is 26 bits.
- There is a shift, the lower two bits are `00` , and the high order four bits come rom the PC (program counter)
- The value of the address field in binary is

```
            0000 0000 0100 0000 0000 0000 0011 0000
```

(e) What is the machine code of this instruction in **binary** and **hex**? Is it the same as what's in the *code* column of the *Text Segment window?*

```
        j 0x00400030
```

- **opcode**: $2_{hex}, 000010_2$
- **Target Field**: $10000C_{hex}$
  - The 32 bit address needs to be converted to 26-bit
  - Unshift lower to bits, and remove the high order four bits

```
        0x00400030 — needs to be converted to 26—bit
        0000 0000 0100 0000 0000 0000 0011 0000
        0000 0100 0000 0000 0000 0011 00
```

- *code*: $0x0810000c$. Yes it is the same

```
        [opcode] [target]
        [000010] [0000 0100 0000 0000 0000 0011 00]
        [0000][1000][0001][0000][0000][0000][0000][1100]
        0x0810000c
```

Assignment 1

Convert the following line in proc2.s to machine code and then back to MIPS instructions:

1. Line #7

```
    addi $s0, $zero, —15
```

- I-format with fields **opcode, rs, rt, immediate**.
  -15 is represented using 2's complement

```
        [opcode: 8] [rs: 0] [rt: 16] [imm: —15]
        [001000] [00000] [10000] [0000 0000 0000 1111]
        [001000] [00000] [10000] [1111 1111 1111 0001]
        [0010] [0000] 0001 0000 1111 1111 1111 0001
        0x2010FFF1 — Machine code
```

- $0x2010FFF1$ - Machine Code
  (a) Find the fields, **opcode, rs, rt, immediate**
  (b) opcode 8: corresponds to I-format instruction addi
  (c) rs: 0, *zero*
  (d) rt: $16_{10}, 0x10$. And register 16 corresponds to the saved temporary S$s0
  (e) 1111 1111 1111 0001 converted to decimal using 2's complement is -15

```
0010 0000 0001 0000 1111 1111 1111 0001
[001000] [00000] [10000] [1111 1111 1111 0001]
[opcode: 8] [rs: 0] [rt: 16]
addi $s0,$zero, −15
```

2. Line #14

```
slt $t0, $s0, $s1
```

- set less than is R-Format instruction. **opcode:** 0, and **funct**: 2a . The instruction contains 6 fields **opcode, rs, rt, rd, shamt, funct**
  rs: $s0, rt: $s1, rd:$t0 map to registers 16,17, 8

```
[000000 10000 10001 01000 00000 101010]
[0000| 0010| 0001| 0001| 0100| 0000| 0010| 1010]
0x0211402a
```

- 0x0211402a. slt because opcode/func is 0/2a

```
[0000 0010 0001 0001 0100 0000 0010 1010]
[000000| 10000| 10001| 01000| 00000| 101010]
slt $t0, s0, $s1
```

3. Line #17

```
beq $t0, $zero, LEEQ
```

- The instruction *beq* is I-format with opcode 4_{hex}. This has four fields, **opcode, rs, rt, immediate**
- $t0 maps to register 8, and $zero to 0, LEEQ is branches to address (0x00400030) and is 16-bit immediate value.
  (a) Branch target address is calculated by incrementing program counter by 4 and adding it to the 16-bit immediate.

$$(pc + 4) + (16 - bit \text{ immediate})$$

  i. To form the 16 bit immediate
     – (current pc + 4) - branch destination address

$$(0x00400014 + 0x4) - (0x00400030) = -0x24$$

- Divide result by 4 to get word address

$$-24/4 = -0x6$$

- If it is not 16-bit extend it

```
                0000 0000 0000 0110 = 0x6
```

- If the branch offset is negative, use 2's complement.

(b) 0x1100000A

```
        [000100] [01000] [00000] [0000 0000 0000 1010]
        [0001| 0001| 0000| 0000| 0000| 0000| 0000| 1010]
```

- 0x1100000A is converted to binary to find the fields to find the MIPS instruction.

```
        [0001| 0001| 0000| 0000| 0000| 0000| 0000| 1010]
        [000100] [01000] [00000] [0000 0000 0000 1010]
        [0x4] [0x8] [0x0]  [0x6]
        bne $t0, $zero, 0x00000006
```

4. Line #20

```
    j GRT
```

(a) The instruction j is J-format. It has two fields **opcode, 26-bit address**

(b) The address of the label GRT is 0x0040001c

- Convert GRT 32-bit address to 26-bit

```
        0000 0000 0100 0000 0000 0000 0001 1100
        1. Remove high order 4-bits
        0000 0100 0000 0000 0000 0001 1100
        2. Remove 2 low order bits
        0000 0100 0000 0000 0000 0001 11
        [00|0001|0000|0000|0000|0000|0111]
       3. 26-bit immediate
```

- Add opcode and 26-bit immediate value to fields

```
    [000010] [00|0001|0000|0000|0000|0000|0111]
    [0000|1000|0001|0000|0000|0000|0000|0111]
    0x08100007
```

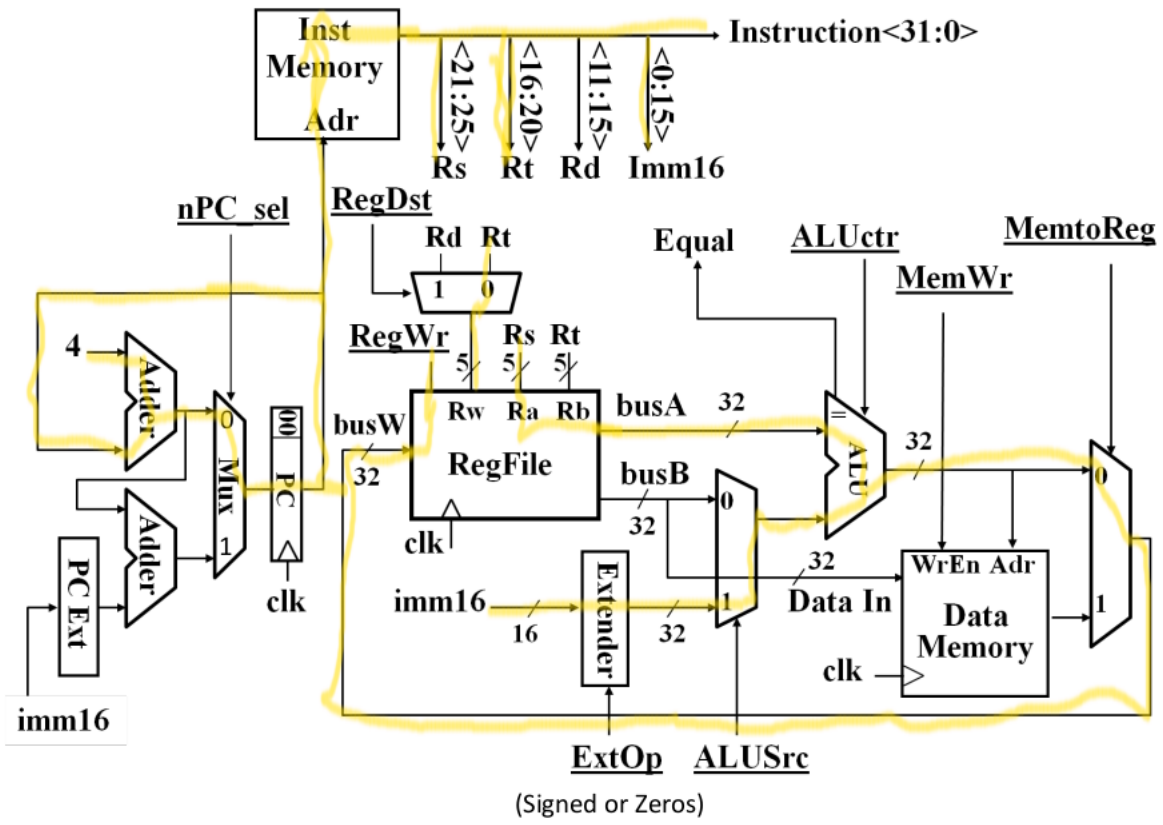
5. Convert 0x08100007 to MIPS instruction

```
 %1. Convert to Binary
 [0000|1000|0001|0000|0000|0000|0000|0111]
 %2. Reorder opcode, 26-bit immediate
 [000010] [00|0001|0000|0000|0000|0000|0111]
 %3. opcode is 0x2
%4. Convert 26-bit immediate to 32 bit. Add lower 2 bit from PC
 [0000|0100| 0000| 0000| 0000| 0001| 1100]
 %5. The PC is (0x00400018) increment by 4 and add the 4 high order bits to
     the 26-bit immediate
 [0000| 0000|0100| 0000| 0000| 0000| 0001| 1100]
 [0x2] [0x0040001C]
 %5. From the MIPS cheat sheet 0x2 opcode is for J instruction
 J 0x0040001C
```

Assignment 2

1. Trace through the paths the execution of SLTIU instruction in this single cycle datapath design. Fill in the appropriate control signal values in the table. If there are any modifications needs to make it work, draw them in the diagram and create appropriate control values.

   - SLTIU is I-format. 4 Fields (**opcode, rs, rt, immediate/addresss**
     (a) Stage 1: [Instruction Fetch] Fetch instruction and increase PC (program counter)
     (b) Stage 2: [Instruction Decode & Register Read]
     (c) Stage 3: [ALU (Arithmetic Logic Unit) Execute]
     (d) Stage 4: [Memory Access] idle
     (e) Stage 5: [Register Write]



nPc_sel	ExtOp	ALUsrc	ALUctr	MemWr	MemtoReg	RegDst	RegWr
0	Zeros	1	Sub	0	0	0	1