EE 361
16-Bit Single Cycle MIPS Project – Last update November 13, 2014

Summary: You will implement, in verilog, a simpified version of the **single-cycle** MIPS. We will refer to this computer as MIPSL for MIPS Lite.

MIPSL Description: Its data and address buses are 16-bits wide rather than 32-bits. All instructions and operands (except ASCII characters) are 16-bits, i.e., "words" are 16 bits. Memory is byte addressable and is organized as Little Endian. Note that memory addresses of words are divisible by two.

General Purpose Registers (eight of them)

Name	**Reg #**	**Usage**	**Preserved on call?**
$zero	0	the constant 0	NA (not applicable)
$v0-$v1	1-2	values for results and expression evaluation	No
$t0-$t2	3-5	temporaries	No
$sp	6	stack pointer	Yes
$ra	7	return address	No

Instruction Formats

Name	Fields					Comments
Field Size	4 bits	3 bits	3 bits	3 bits	4 bits	ALL MIPS-L instructions 17 bits
R-format	op	rs	rt	rd	NA	Arithmetic instruction format
I-format	op	rs	rt	Address/ immediate		Transfer, branch, immediate format

The I format is similar to the 32-bit MIPS. However, the R format is different. Note that there is no function field ("NA" in the table above means "Not Applicable"). This implies that the opcode value of the R format will be different for different instructions (in the 32-bit MIPS the opcode value is zero for different instructions). This is shown in the next table.

Machine Instructions:

Name	Format	Example					Comments
		3 bits	3 bits	3 bits	3 bits	4 bits	
add	**R**	**0**	**2**	**3**	**1**	NA	**add $1,$2,$3**
sub	**R**	**1**	**2**	**3**	**1**	NA	**sub $1,$2,$3**
slt	**R**	**2**	**2**	**3**	**1**	NA	**slt $1,$2,$3**
lw	**I**	**3**	**2**	**1**	**100**		**lw $1,100($2)**
sw	**I**	**4**	**2**	**1**	**100**		**sw $1,100($2)**
beq	**I**	**5**	**1**	**2**	**(offset to 24)/2**		**beq $1,$2,12**
addi	**I**	**6**	**2**	**1**	**100**		**addi $1,$2,100**
andi	**I**	**7**	**2**	**1**	**100**		**andi $1,$2,100**

The instructions in the MIPSL are similar to the 32-bit MIPS. Notice that the opcode value completely determines what the instruction is because for R-type instructions, there is no function field.

The circuit diagram for the single cycle MIPSL is similar to the following circuit diagram for the 32-bit single cycle MIPS. However, there are differences in the bus widths and shifting. Also, there is no jump instruction.

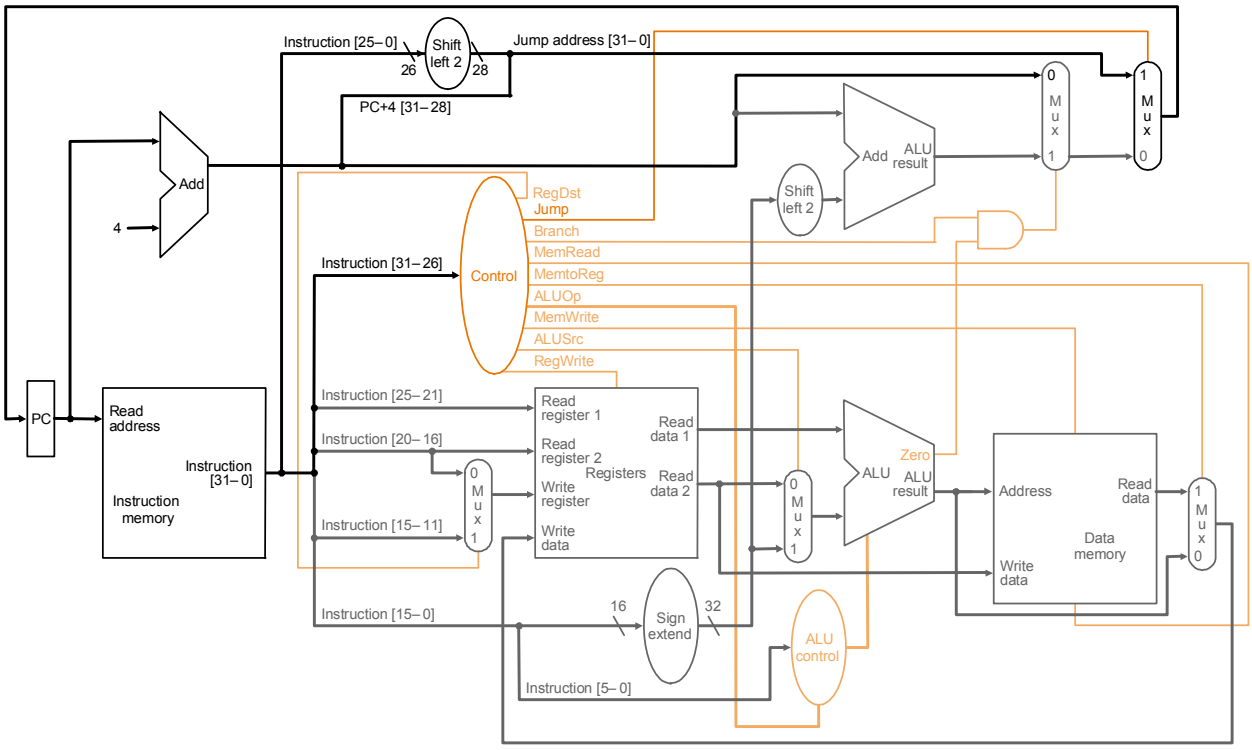

Figure 1. 32-bit MIPS Single Cycle.

Also ALU Control is not needed since there is no function field. Instead the select for the ALU should be connected to the main controller.

The components of the circuit are

- Instruction Memory that has the program
- Register File
- Data Memory and IO
- ALU
- Adders
- 2:1 Multiplexers
- Shifters
- Sign Extenders
- Control
- Program Counter (PC) and its control logic

Figure 2 explains the PC and its control logic.

Circled in the figure is the PC control logic.

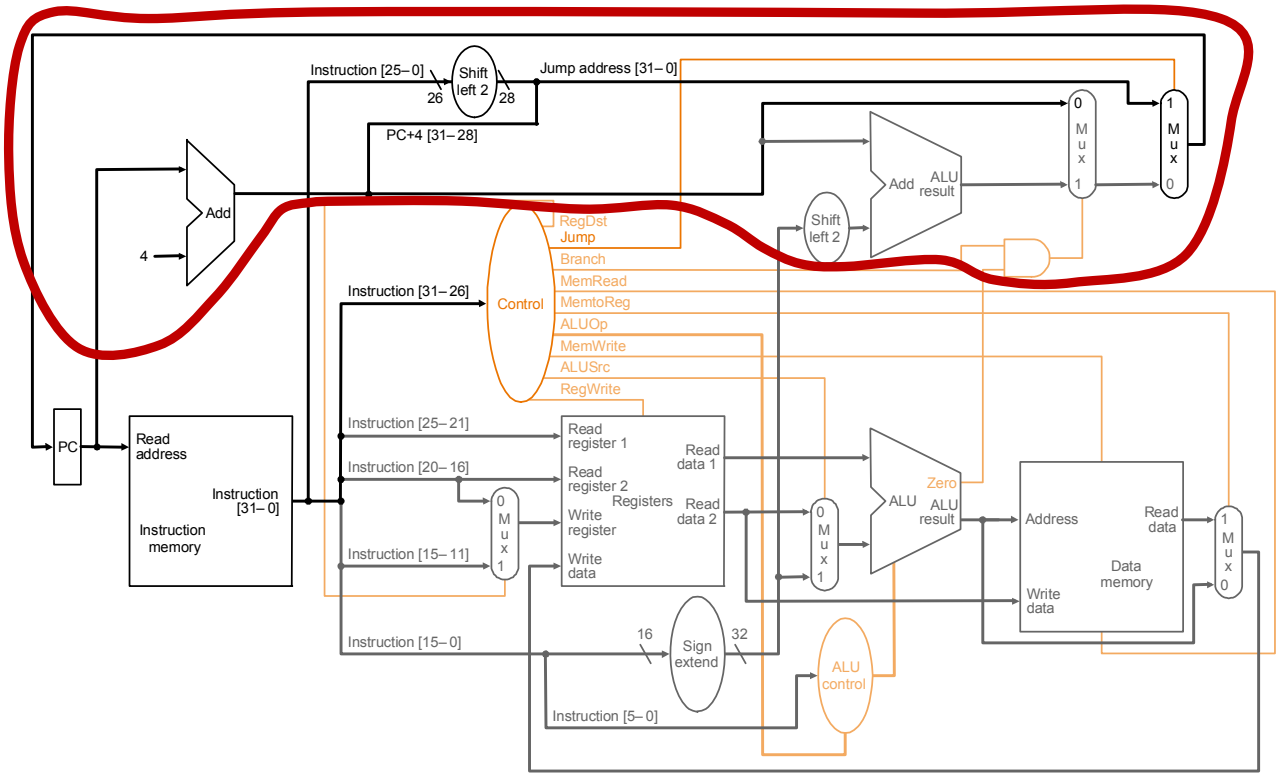

Figure 2. Schematic showing the PC control logic in the MIPS processor

Figure 3 is a block diagram of the PC control circuit. You can implement this as a single verilog module. This simplifies the design since you have less components.

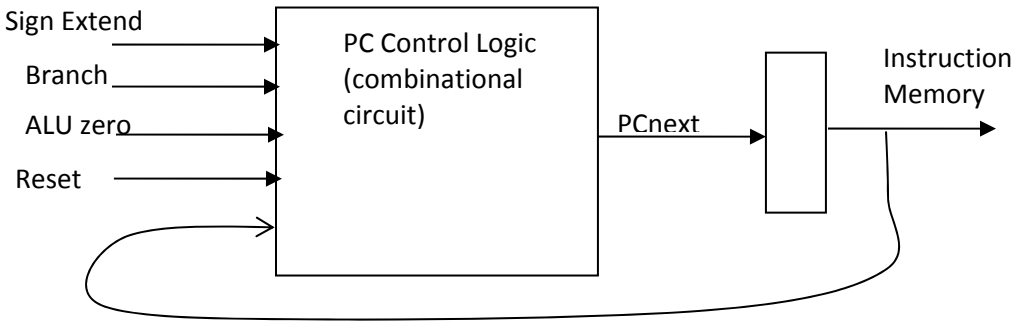

Figure 3. Block diagram of the PC control logic.

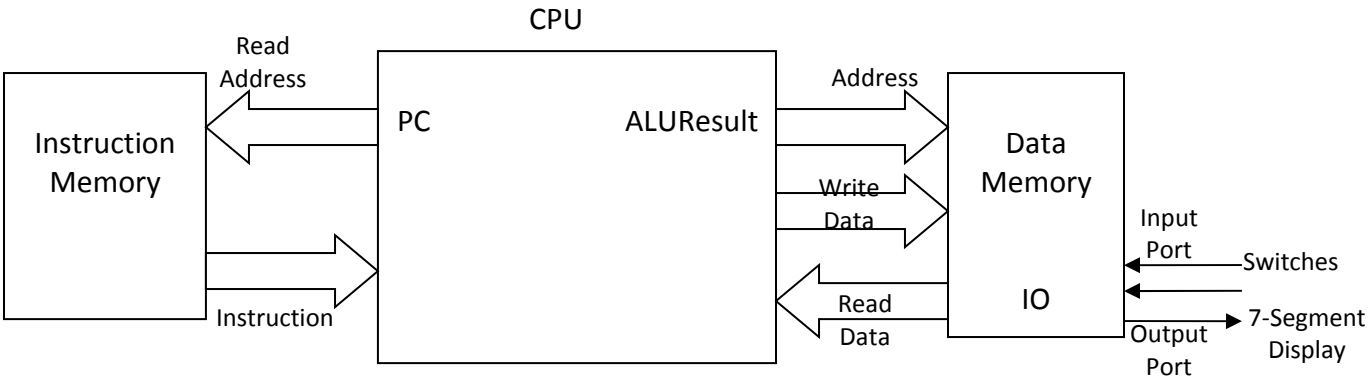

Figure 4. Block diagram of the computer, clock not shown.

Figure 4 shows a block diagram of the computer. There is the CPU, instruction memory, and data memory, which includes I/O. Notice that the Data Memory/IO block has

- Input port at address 0xfff0. It is connected to sliding switches sw0 and sw1: sw0 is bit 0 and sw1 is bit 1
- Output port at address 0xfffa. It is connected to a 7 segment display.

The Data Memory is 128 memory cells, each with 16-bits. The addresses of the memory cells are 0, 2, 4, ..., 254.

Assignment:

There are three stages to this homework. Each stage takes about a week. DON'T FALL BEHIND.

- Stage 1: This stage is to test components of the computer. There's nothing to submit. The files for this stage are in the folder Hw10-Stage1. This stage is straight forward but it will take time.
- Stage 2: You will implement a computer so that it runs a simple program, which multiplies two integers by using a loop. The computer should be able to run the instructions: add, addi, and beq. The files for this stage are in the folder Hw10-Stage2
- Stage 3: This stage improves the computer from stage 2. In particular, it can also run lw, sw, and andi. The program that it will run will access the IO ports. The files for this stage are in the folder Hw10-Stage3

For each of these stages are verilog files for component and testbenches. Read the testbenches to understand them. They have comments to guide you.

Grading: The Homework is worth 30 points according to the following

Complete	**Points**
Partially working Stage 2	15
Completely working Stage 2	24
Completely working Stage 3	30

Submission Instructions:

Submit a single Windows folder (zipped), which should have

1. All your verilog files including the testbench
2. Also include either
    a. IM2.V if you have Stage 3 completely working or
    b. IM1.V if you have Stage 2 partially or completely working but not Stage 3
3. Project file that will run the testbench
4. README file which includes
    a. Your name
    b. Description of what you have completed (and not completed). For example
        i. "I have completed Stage 3. I have included IM2.V"
        ii. "I have completed Stage 2, I didn't get Stage 3 to work completely. I have included IM1.V"
        iii. "I didn't get Stage 2 to work completely but I have it partially working. I have included IM1.V"
        iv. "I didn't get anything to work in Stage 2"
    c. Description of what you have in this folder including a list of all the files including all verilog and project files.

Zip the folder and submit it through laulima as an attachment on the due date indicated in the assignment.

Description of 3 Stages:

Stage 1: Here are the files in Hw10-Stage1 folder (note some files may need some editing)

- MIPS-Parts.V: This has
  - Register file
  - ALU
  - 16-bit 2:1 multiplexer
  - 16-bit 4:1 multiplexer
  - Sign extender
  - Data memory
    - The data memory has 128 memory cells that are 16-bits
    - Memory starts from address 0.
    - Memory is byte addressable, each 16-bit word has an address that is divisible by 2
    - The word addresses are 0, 2, 4, ...
    - There are two IO ports
      - Output port to a 7-segment display at address 0xfffa.
      - Input port from two switches at address 0xfff0.
  - There are three testbenches for this file
    - testbenchHw10-Parts-CombCirc.V: Tests the combinational circuits.
    - testbenchHw10-Parts-Rfile.V: Tests the register file
    - testbenchHw10-Parts-Dmem.V: Tests the data memory and IO

Create three projects, each having one of the testbenches and MIPS-Parts.V. Run each project and make sure MIPS-Parts.V works. Read the testbenches to understand them. There are comments to guide you. You can also write your own testbenches or modify the existing ones.

Be sure to look over the verilog files to understand the circuits.

There is nothing to turn in for this stage.

The next two stages is to build a working processor that can run some of the instructions.

Stage 2: In the Hw10-Stage 2 folder there are the following files (some files may need editing, i.e., buggy)

- MIPSL-SingleCycle.V: This has the MIPSL processer module. It's incomplete (actually it's basically empty) and you must complete it.
    - It must implement the add, addi, and beq instructions
- testbenchHw10-MIPSL.V: This is the test bench for MIPSL-SingleV. It has instantiations of the
    - MIPSL module
    - Data memory (with IO)
    - Instruction memory IM1.V. This is a program that multiplies 3 by 5 using a loop. It uses the add, addi, and beq, instructions. The following is the program

```
L0:   addi  $2,$0,3    # $2 = 3, $2 is used as a counter
      add   $4,$0,$0   # Clear $4, which will contain the final product
L1:   beq   $2,$0,L0   # If counter = 0 then we've completed the multiply and we can start all over
      addi  $4,$4,5    # Increment $4 by 5
      addi  $2,$2,-1   # Decrement counter
      beq   $0,$0,L1   # Continue looping
```

    - Note: One way we will check if the MIPS-L is working correctly is viewing the ALU output. For example, if the instruction is addi $4,$0,3 then the ALU output = 3. Then if the instruction is addi $4,$4,3 then the ALU output is 6. Also note that the ALU output is the write data output of the MIPS-L that goes to data memory. So by viewing the write data output from the MIPS-L we will view the ALU output.

- To create a project you need the following files:
    - IM1.V
    - MIPSL-SingleCycle.V
    - MIPSL-Control.V (described below)
    - MIPSL-PC.V (described below)
    - MIPS-Parts.V
    - testbenchHw10-MIPSL.V.

- MIPSLControl.V: This has the Controller module of the MIPS-L. This will be instantiated in the MIPS-L module.
    - MIPSLControl.V: This has an incomplete Controller module. Shown below is the function tables for the ALU for your reference. This is straight forward if you understand how the MIPS-L datapath works.
    - There's a sample testbench testbenchHw10-Control (it may be buggy). You can make your own testbench too.

ALU function table

alu select	**Operation**
0	add
1	subtract
2	slt
3	or
4	and

- MIPSLPC.V: This has the PC control logic described earlier (see Figures 2 and 3). This is a component of MIPS-L and will be instantiated in the MIPS-L module.
    - MIPSLPC.V: has an incomplete PCLogic module. Currently, it only increments PC by 2 and resets to 0. You should also have it operate for conditional branch beq. This is straight forward (really).
    - testbenchHw10-PC.V: this is a test bench for PCLogic module.

Stage 3: Implement MIPSL so that it can run lw and sw instructions as well as the instructions of Stage 2. It should run Instruction Memory IM2.V. The files are in the folder Stage Hw10-Stage3. It also includes the testbench.

The IM2.V program interacts with the I/O ports:

- Input switch 0 at address 0xfff0
- Output port connected to the seven segment display at address 0xfffa

It will continually check switch 0. If the switch = 0 then it outputs "0" to the seven segment display. If the switch = 1 then it outputs "1" to the seven segment display.

Description of IM2.V

The program is an infinite loop. It first does the following

- Set $3 to 0xfff0. This is a reference point to the I/O ports. For example, note that
    - lw $4,0($3) will load a value from the input switches into $4
    - sw $4,0($3) will store the value in $4 into the 7 segment display

Next it checks the input switch 0. If it's 0 then the computer will display "0" on the 7 segment display. Otherwise, it displays "1" on the display.

```
#  Initialization Phase
L0:     addi    $3,$0,fff0          # $3=0xfff0, the addr to sw0/display
        lw      $5,0($3)            # $5 = input switch value
        andi    $5,$5,1             # Mask all bits except bit 0 (sw0)
        beq     $0,$5,Disp0         # if bit = 0 then $4 = pattern "0" else $4 = pattern "1"
        addi    $4,$0,0110000       # $4 = bit pattern "1"
        beq     $0,$0,Skip
Disp0:  addi    $4,$0,1111110       # $4 = bit pattern "0"
Skip:
        sw      $4,10($3)           # 7-segment display = bit pattern
        beq     $0,$0,L0            # Repeat loop
```