COMP311: Logic Circuit Design

Spring 2022, Prof. Taigon Song

Final Project. Due: June 22 8:59am [Total: 150 points]
(Early submission: June 15 8:59am)

1. Goal:

Design a 16-bit CPU based on MIPS architecture. We will define “1 word = 16 bit” in our project.

2. Some warm-up information:

A 16-bit MIPS processor is very similar to a 32-bit MIPS processor that is covered in our COMP311 class. Based on the lectures provided, your job is to fully understand the functionality of a 32-bit MIPS processor, then apply your knowledge to design a 16-bit MIPS processor. This project will provide full details of the instruction set of a 16-bit MIPS processor, but it will be your job to design the actual hardware out of it.

3. Specifications:

- You will be provided with the instructions stored inside the instruction memory.
- You will be provided with the data stored inside the data memory.
- You will NOT be provided with any data inside the register file (RF). You will assume that the RF is initialized. Therefore, RF should save its data via any necessary instructions.
- You will need to initialize your CPU with the values provided for the instruction memory and the data memory. Once the data are loaded to the memories, the program counter will automatically increase its counter number so that the instructions are automatically processed.

4. Provided instructions/data

Data memory

Addr.	Data	Addr.	Data	Addr.	Data
[0]	0000_0000_0000_0000	[8]	1110_0001_0010_1000	[16]	0110_1000_0011_1001
[1]	0000_0000_0010_0011	[9]	0101_0011_1100_0101	[17]	1111_1000_0010_1011
[2]	0000_0000_0000_1001	[10]	1001_0111_1000_1001	[18]	0011_1101_0111_1001
[3]	0000_0000_0011_0001	[11]	1101_0011_1101_1111	[19]	1011_0000_0111_0001
[4]	0000_0000_1100_1001	[12]	1011_1001_1001_0001	[20]	0010_0001_1110_0110
[5]	0000_0000_0011_1100	[13]	0000_0100_0110_0101	[21]	1101_0000_1100_1010
[6]	0000_0000_1101_1011	[14]	0001_1100_0101_0110	[22]	0111_0111_0000_1110
[7]	0000_0000_0000_0111	[15]	0001_0010_1110_0100	[23]	1111_1101_1011_1001

Instruction memory

Addr.	Data	Addr.	Data	Addr.	Data
[0]	1001_1001_0000_0001	[8]	0000_1001_1100_0011	[16]	0000_0000_0000_0000
[1]	1001_1001_1000_0010	[9]	0110_0000_0000_1101	[17]	0000_0000_0000_0000
[2]	0000_1001_1100_0000	[10]	1111_0010_0000_0010	[18]	0000_0000_0000_0000
[3]	0000_1001_1001_0100	[11]	1011_1010_0000_0011	[19]	0000_0000_0000_0000
[4]	1100_0100_0000_0010	[12]	0100_0000_0000_1111	[20]	0000_0000_0000_0000
[5]	0000_1001_1100_0010	[13]	000_010_011_100_0110	[21]	0000_0000_0000_0000
[6]	0100_0000_0000_1000	[14]	0001_1100_0000_1000	[22]	0000_0000_0000_0000
[7]	0000_1001_1100_0001	[15]	0000_0000_0000_0000	[23]	0000_0000_0000_0000

5. 16-bit MIPS instruction set

Below are the types of instructions supported in a 16-bit MIPS processor

Name	Fields					Comments
Field size	3-bits	3-bits	3-bits	3-bits	4-bist	
R-format	op	rs	rt	rd	funct	Arithmetic instructions format
I-format	op	rs	rt	func/addr		Transfer, branch, immediate format
J-format	op	target addr				Jump instruction format

Name	Format	Example					Comments
		3 bits	3 bits	3 bits	3 bits	4 bits	
add	R	0	2	3	1	0	add $1, $2, $3
sub	R	0	2	3	1	1	sub $1, $2, $3
and	R	0	2	3	1	2	and $1, $2, $3
or	R	0	2	3	1	3	or 1, $2, $3
slt	R	0	2	3	1	4	slt $1, $2, $3
mul	R	0	2	3	1	5	mul $1, $2, $3
div	R	0	2	3	1	6	div $1, $2, $3
jr	R	0	7	0	0	8	jr $7
lw	I	4	2	1	5		lw $1, 5 ($2)
sw	I	5	2	1	5		sw $1, 5 ($2)
beq	I	6	1	2	5		beq $1, $2, 5
addi	I	7	2	1	5		addi $1, $2, 5
slti	I	1	2	1	5		slti $1, $2, 5
j	J	2	100				j 100
jal	J	3	100				jal 100

For detailed information on a 32-bit MIPS instruction set, you can refer to the “” or see the link below:

http://www.dsi.unive.it/~gasparetto/materials/MIPS_Instruction_Set.pdf

6. 16-bit MIPS architecture datapath

Below is a figure of 32-bit MIPS datapath. Your 16-bit MIPS architecture data path should look quite similar to the figure below.

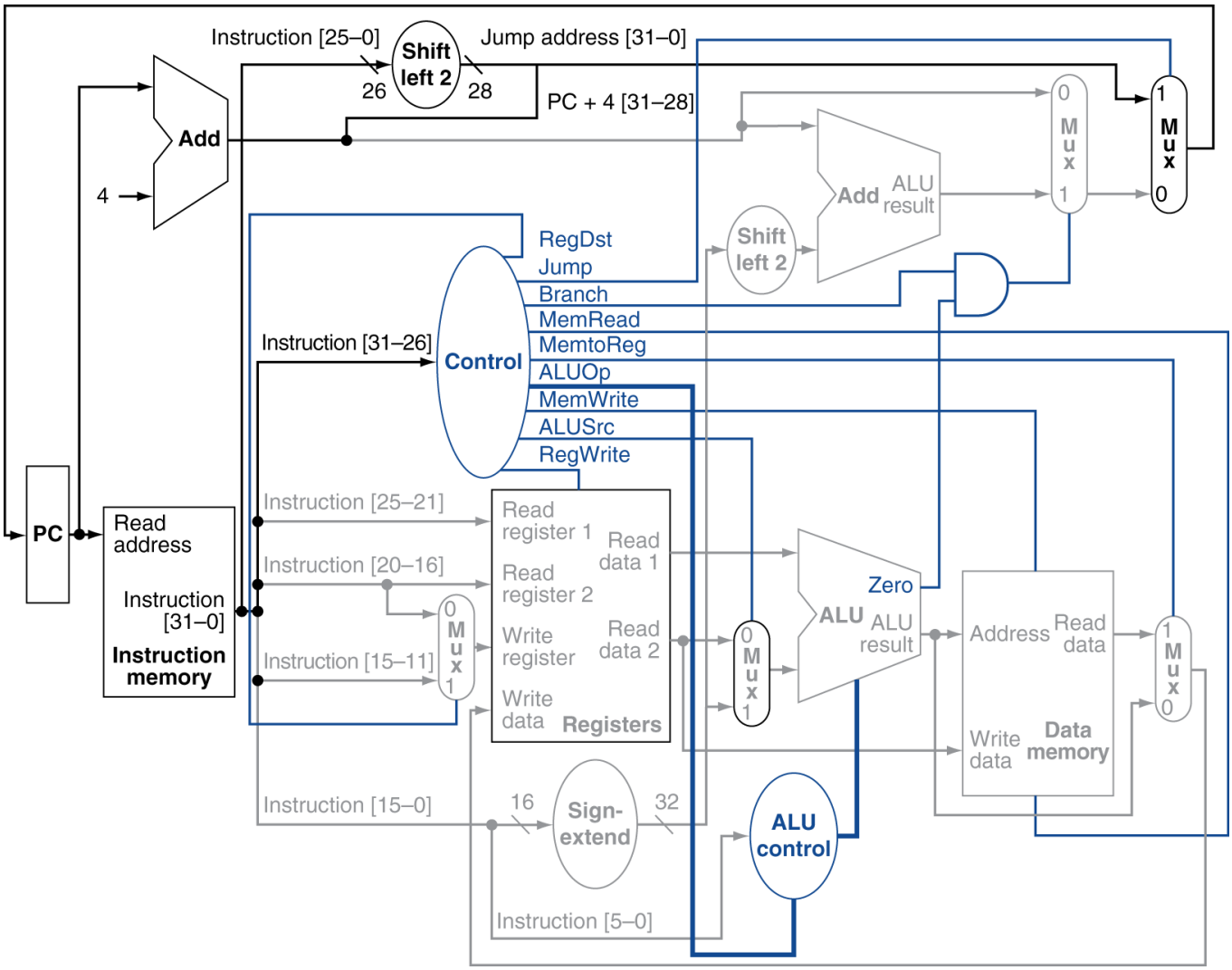

Figure 1 – A 32-bit MIPS processor datapath

7. Control Signals

Table 1 – Control signals

Instruction	regDst	aluSrc	memtoReg	regWrite	memRead	memWrite	Branch	aluOp	Jump
R-type	1		0					0	0
lw	0		1					3	0
sw	0		0					3	0
addi	0		0					3	0
beq	0		0					1	0
j	0		0					0	1
jal	2		2					0	1
slti	0		0					2	0

Keep in mind that Figure 1 is not complete for your implementation. Still, Figure 1 will provide a good idea of how your 16-bit MIPS processor should be implemented. For example, regarding the "Control" unit (CTU) in Figure 1, we see that CTU connects many control signals to other modules. For your convenience, Table 1 provides a hint on how these control signals should be managed.

Table 2 – ALU Control signals

ALU op	Function	ALU ctrl output	ALU operation	Instruction
11	xxxx	000	add	addi, lw, sw
01	xxxx	001	sub	beq
00	00	000	add	add
00	01	001	sub	sub
00	02	010	and	and
00	03	011	or	or
00	04	100	slt	slt
00	05	101	mul	mul
00	06	110	div	div
10	xxxx	100	slt	slti

You should also develop an "alu_control" module that generates the control for the ALU. For your convenience, Table 2 provides the full functionality of the alu_control module.

8. Extra modules not defined in Figure 1

Figure 1 misses some modules that should be there for your implementation. Also, there are two MUXes that should be different from Figure 1. If you can figure out what is missing and can submit your project report by 6/15 8:59am, you will get an extra 20 points. Otherwise, the details of the final module will be released at 6/15 9:00am.

9. Design conditions

Below are some criteria that you should satisfy in order to receive full credit:

- Your top module should be consisting of sub modules only. No assign/always statements for functionality. (assign statements for wire definition and connecting OK)
- Adders and Muxes should be designed using parameters
- Initialize your memories in the most convenient way you think it can be done
- Should be synthesizable (Synthesis report for the 'top module' required only).
- Program counter bits are 13 bits, and PC increases by 1 each time.
- $ra is $7, and $0 is $zero. In addition, $0, $6 should be constant 0.
- All registers/FFs/Memories are positive edge triggered
- All sequential elements reset at rst=0;
- Must need Inputs: rst, clk; outputs: current_instr,

- At the beginning: clk = 0; rst = 0; #2 rst = 1; , run for 20ns
- clk toggles every 1ns

10. Some hints for approach

1. 1 word is 16-bits. Think what this means in terms of PC address.
2. Below is the register table you will refer to

$register table		
No.	**Fuction**	**Note**
000	$zero	0
001	$s0	
010	$s1	
011	$s2	
100	$s3	
101	stack pointer	
110	frame pointer	
111	$ra	

3. You should fill in something like this for your own reference.

PC No.	Op [15:13]	Rs [12:10]	rt [9:7]	rd [6:4]	Func [3:0]	Note

10. Report submission/grading

To verify your functionality, please screen capture the below in your iVerilog command prompt.
(Need the full list till the end of your testbench runtime)

```
ID:1234, at time                2000 ps, PC =    0, RF[0, 1, 2, 3, 4, 7] is:     0,     0,     0,     0,     0,     0
```

Also, prove your final result by printing a $display statement after all instructions are executed.

$display("The final result of $s3 in memory is: %d", tgcpu.data_memory.internal_mem[?]);

Based on your judgement, justify what the number would be to replace the question mark (?).

For your final report, you should submit the below compressed in a .zip file online.

- Your report file (doc, hwp, or pdf)
- .v and _tb.v
- Synthesis report

Grading policy:

- No cover sheet on your report file: -10 points
- No synthesis report: -10 points

A sample of the cover page is shown in the next page.

2022 Spring COMP311: Logic Circuit Design

Final Project

Student ID / Name:

Final Project check list (each 10 points)

	Answers
1. First value of $s1, $s2	
2. First value of $s3	
3. Final $s3 value	
4. What are the instructions implemented, but not used?	
5. What is the final PC # processed?	
6. What are the PC #s that have not been processed?	
7. Parameterized MUX design	(Y/N)
8. load memory with instructions	(Y/N)
9. Implemented result stored in the register after PC 0000	(Y/N)
10. Implemented result after beq	(Y/N)
11. Implemented result after first sw is done	(Y/N)
12. When run is complete: (1) final $s3 value,	(Y/N)
13. When run is complete: (2) registers have intended results	(Y/N)