Organization of Digital Computer Lab
EECS 112L

LAB 2

EECS Department
The Henry Samueli School of Engineering
University of California, Irvine

Winter 2020

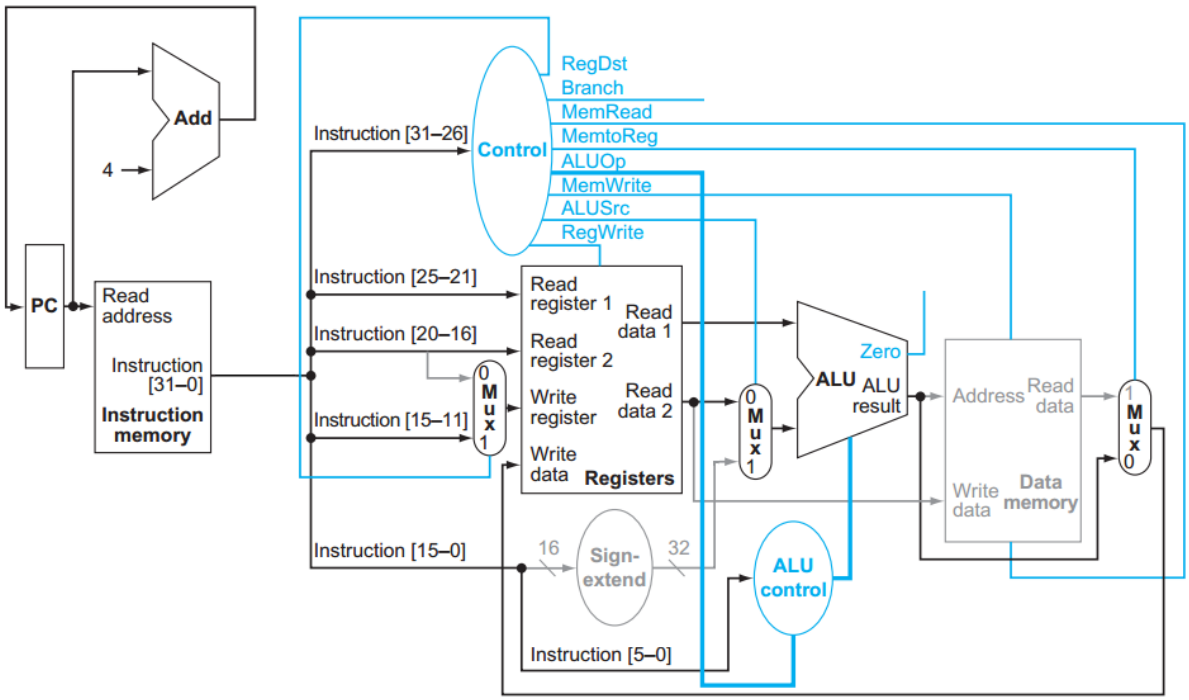
Due on Sunday 02/03/2020 11:00 pm

In this lab, we want you to complete a Single Cycle MIPS Processor.

1 Single Cycle Processor

Figure 1 shows a single cycle processor.

Figure 1 : Datapath.



The instruction execution starts by using the program counter to supply the instruction address to the instruction memory. After the instruction is fetched, the register operands used by an instruction are specified by fields of that instruction. Once the register operands have been fetched, they can be operated on to compute a memory address (for a load or store), to compute an arithmetic result (for an integer arithmetic-logical instruction), or an equality check (for a branch). If the instruction is an arithmetic-logical instruction, the result from the ALU must be written to a register. If the operation is a load or store, the ALU result is used as an address to either store a value from the registers or load a value from memory into the registers. The result from the ALU or memory is written back into the register file. Branches require the use of the ALU output to determine the next instruction address, which comes either from the ALU (where the PC and branch oset are summed) or from an adder that increments the current PC by four. The thick lines interconnecting the functional units represent buses, which consist of multiple signals. The arrows are used to guide the reader in knowing how information ows. Since signal lines may cross, we explicitly show when crossing lines are connected by the presence of a dot where the lines cross.

Create a new project in Vivado using the files listed below. Find these files under lab2 on the Canvas.
mips_32.v, datapath.v, control.v, ALUControl.v, instruction_mem.v, mux2.v, register_file.v, sign_extend.v, ALU.v, data_memory.v, and tb_mips_32.

Now you have a mips processor which is able to run these 7 instructions:

Name	Format	op	rs	rt	rd	shamt	funct	Comments
add	R	0	reg	reg	reg	0	32	add $s1,$s2,$s3
sub	R	0	reg	reg	reg	0	34	sub $s1,$s2,$s3
addi	I	8	reg	reg	n.a.	n.a.	n.a.	addi $s1,$s2,100
lw	I	35	reg	reg	n.a.	n.a.	n.a.	lw $s1,100($s2)
sw	I	43	reg	reg	n.a.	n.a.	n.a.	sw $s1,100($s2)
and	R	0	reg	reg	reg	0	36	and $s1, $s2, $s3
or	R	0	reg	reg	reg	0	37	or $s1, $s2, $s3

We want you to complete this processor and add a new set of instructions to it.

Name	Format	op	rs	rt	rd	shamt	funct	Comments
Add	R	0	reg	reg	reg	0	32	add $s1, $s2, $s3
Addi	I	8	reg	reg	n.a.	n.a.	n.a.	addi $s1, $s2, 20
and	R	0	reg	reg	reg	0	36	and $s1, $s2, $s3
andi	I	12	reg	reg	n.a.	n.a.	n.a.	andi $s1, $s2, 20
Beq	I	4	reg	reg	n.a.	n.a.	n.a.	Beq $s1, $s0, L1
Lw	I	35	reg	reg	n.a.	n.a.	n.a.	lw $s1, 20($s2)
Nor	R	0	reg	reg	reg	0	39	nor $s1, $s2, $s3
Or	R	0	reg	reg	reg	0	37	or $s1, $s2, $s3
Slt	R	0	reg	reg	reg	n.a.	42	slt $s1, $s2, $s3
Sll	R	48	reg	reg	reg	Shift amount	0	sll $s1, $s2, 10
Srl	R	48	reg	reg	reg	Shift amount	2	srl $s1, $s2, 10
sra	R	48	reg	reg	reg	Shift amount	3	sra $s1, $s2, 10
Sw	I	43	reg	reg	n.a.	n.a.	n.a.	sw $s1, 20($s2)
Sub	R	0	reg	reg	reg	0	34	sub $s1, $s2, $s3
xor	R	0	reg	reg	reg	0	38	xor $s1, $s2, $s3
Mult	R	0	reg	reg	reg	0	24	mult $s1, $s2, $s3
div	R	0	reg	reg	reg	0	26	div $s1, $s2, $s3
jump	J	2	n.a.	n.a.	n.a.	n.a.	n.a.	J 2500

2 MIPS Instructions

MIPS has 3 different instruction types. Table 1 shows these three instruction formats. In our simple implementation of MIPS we only have some R-type instructions plus load word and store word and addi instructions which are I-type.

Table 1 : Instruction Set.

Name	Fields						Comments
Field size	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	All MIPS instructions are 32 bits long
R-format	op	rs	rt	rd	shamt	funct	Arithmetic instruction format
I-format	op	rs	rt	address/immediate			Transfer, branch, imm. format
J-format	op	target address					Jump instruction format

2.1 R-Format

Below you see the R-Format instruction fields.

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

Here is the meaning of each name of the felds in MIPS instructions:

- op: Basic operation of the instruction, traditionally called the opcode.
- rs: The first register source operand.
- rt: The second register source operand.
- rd: The register destination operand. It gets the result of the operation.
- shamt: Shift amount.

- funct: Function. This field, often called the function code, selects the specific variant of the operation in the op field

Here is an example for add instruction:

```
add $s16,$s5,$s7

op        rs       rt       rd       shamt    funct
000000    00101    00111    10000    00000    100000
```

NOTE: for shift instructions we want to shift rs register shamt times and store back the result in the rd register. We are not using the rt register field for shift instructions.

```
sll $s1,$s3,2

op        rs       rt       rd       shamt    funct
110000    00011    00000    00001    00010    000000
```

2.2 I-Format

Below you see the I-Format instruction fields. They are same as R-Format except they have an address (or immediate value) field.

Here is an example for beq instruction. If register one and register three are equal we want to branch to 2 instructions after the current instruction:

```
beq    $s3, $s1, #2

op        rs       rt       constant or address
000100    00011    00001    00000000000000010
```

2.3 J-Format

Below you see the J-Format instruction fields.

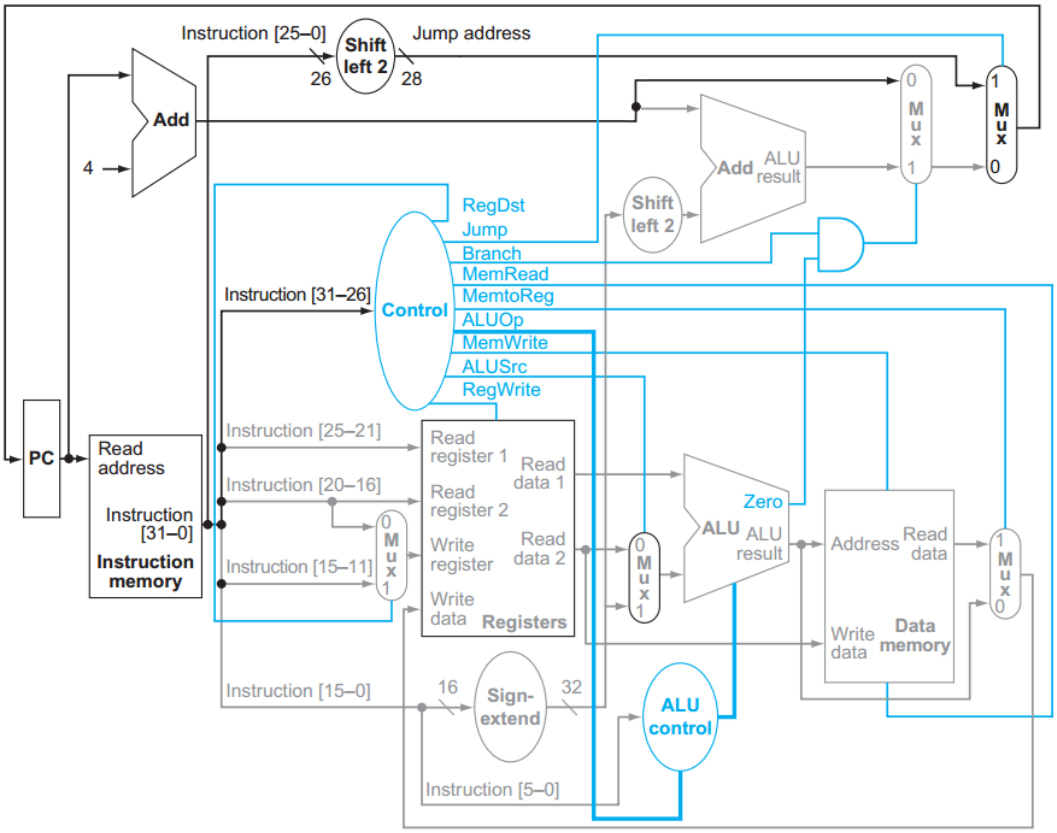
Here is an example for jump instruction. We want to jump to address 2:

```
j       #2

op                  address
000010    00000000000000000000000010
```

3 MIPS processor

After you modify the code your processor would be something like this.



use this table to modify control, ALUControl, and ALU files.

Name	format	Instruction[31:26]	Instruction[5:0]	ALUop	alu_control
Add	R	000000	100000	10	0010
Addi	I	001000	-	00	0010
and	R	000000	100100	10	0000
andi	I	001100	-	11	0000
Beq	I	000100	-	01	0110
Lw	I	100011	-	00	0010
Nor	R	000000	100111	10	1100
Or	R	000000	100101	10	0001
Slt	R	000000	101010	10	0111
Sll	R	110000	000000	10	1000
Srl	R	110000	000010	10	1001
sra	R	110000	000011	10	1010
Sw	I	101011	-	00	0010
Sub	R	000000	100010	10	0110
xor	R	000000	100110	10	0100
Mult	R	000000	011000	10	0101
div	R	000000	011010	10	1011
jump	J	000010	-	-	-

4 Test New Instructions

To test each new instruction you need to find the binary code of that instruction and add it to the instruction memory. Chapter 2 of the book will help you to write your own instructions.

4.1 Addressing

Pay attention that memories in MIPS are byte addressable. It is the reason that we increase program counter (pc) by 4. Also if you look at the instruction memory and data memory code you will see that we are not using the lower 2 bits of the address. This means the address is byte address and since we want to access words we can omit the lower 2 bits.

5 Assignment Deliverables

Your submission should include the following:

- Block designs. (mips_32.v, datapath.v, control.v, ALUControl.v, instruction_mem.v, mux2.v, register_file.v, sign_extend.v, ALU.v, data_memory.v)
- A report in **pdf** format. Follow the rules in the "sample report" under "additional resources" in the Canvas.

Note1: Compress all files (11 files : 10 .v files + report) into zip and upload to the **CANVAS** before deadline.
Note2: Use the code samples that are given. **The module names and the port names should exactly look like the code sample otherwise you lose points**.