1. **Please submit your assignment in class**
2. **NOTE 1**: Full name on each page. Number each page. Single sided. Stapled.
3. Failure to follow (2) precisely will result in your assignment not being graded.
4. **** NOTE 2 **** A reminder that all submissions must be your own work. **The academic code of integrity will be strictly enforced.**
5. **** NOTE 3 **** The code you submit should be crystal clear, well documented, with the use of proper, self-explanatory identifiers. The plots should be similarly clear, properly labeled and should be accompanied by the corresponding table of values.
6. **DUE DATE: 27th March 2014**

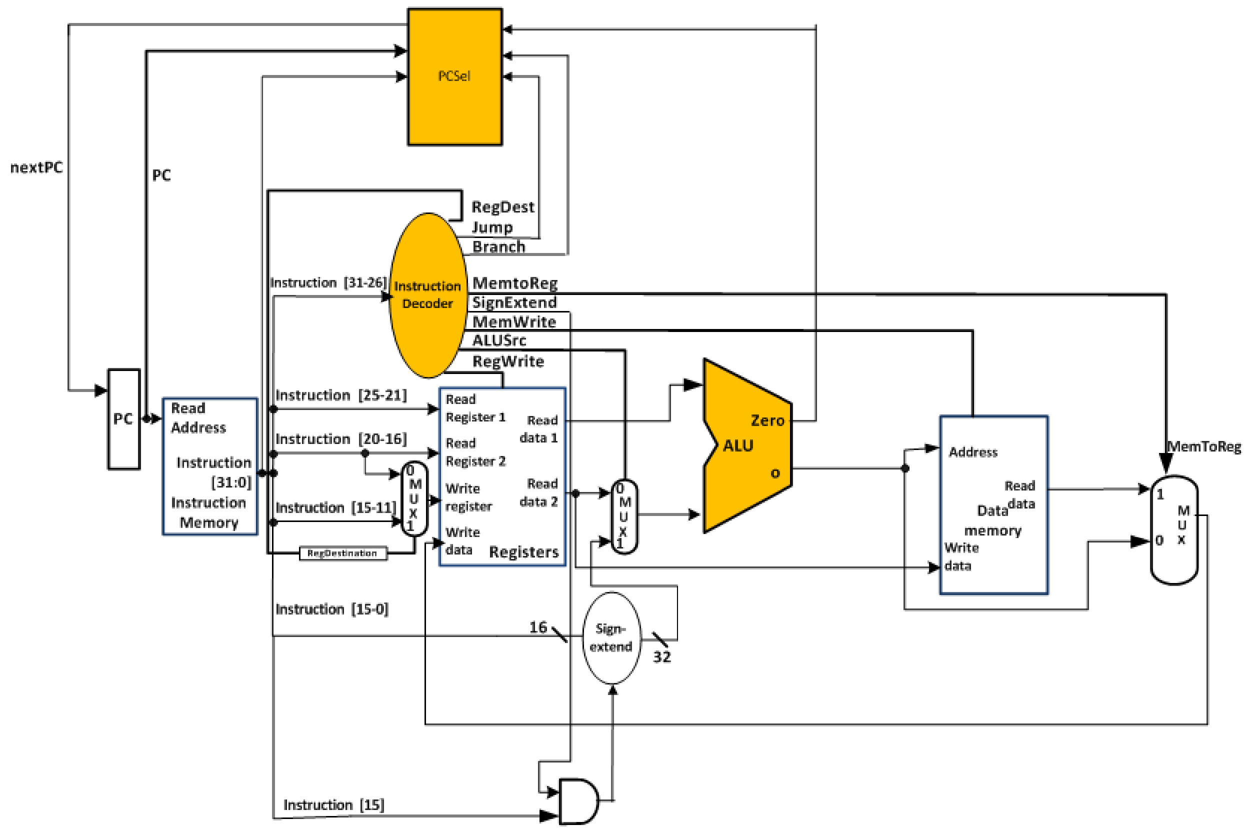
= =

OVERVIEW:

For Lab 2, you will design a 32 bit ALU (Arithmetic and Logic Unit) of the MIPS architecture for a small subset of instructions. You will be using the fastest adder that you built in Lab 1 to implement the design of the ALU. The design is expected to clean and efficient while being able to meet all the desired results.

= =

MIPS ARCHITECTURE:



This is how the MIPS processor works. On initializing, the Program Counter is reset to 0. The PC value is passed on to the Instruction Memory File wherein it decides which Instruction is to be carried out based on the value of the Program Counter. The Instruction Memory gives the 32-bit instruction to the Register File which takes in the value of the RS, RT, and RD blocks and gives data stored in those addresses to the ALU for computation and other evaluations. The ALU output is either given to the Data Memory or it is returned back to the Register File based on the instruction it is carrying out. The Program Counter gets increased by

4 for every subsequent instructions except in the case of the Branch and Jump Instructions. Please refer to the materials mentioned in Additional References for more understanding of the MIPS Architecture. You needn't worry about grasping all the concepts since we are only concentrating on the ALU part of the architecture.

Example of Instruction decoding in MIPS ISA:

Consider an R-type Instruction = 0000_0000_0010_0010_0001_1000_0010_0001
When split in sections as shown in table:

000000	00001	00010	00011	000000	100001
Opcode(6) [31-26]	**RS(5) [25-21]**	**RT(5) [20-16]**	**RD(5) [15:11]**	**Shamt [10:6]**	**funct (6) [5:0]**

What it means :
1. Take the value in **Register no. 1** in "register file".
2. **Add** to this, value in **Register no. 2** in "register file".
3. Save the resulting value in **Register no. 3** in "register file".

Note that, In this lab we have not implemented Register File, Data Memory and Instruction Memory. Hence we have to provide this data in testbench.

Above example shows encoding of R-type Instruction. Encoding for R, I and J type instructions is shown below.
We can find out which encoding applies by looking at opcode.

Format	**6 bits**	**5 bits**	**5 bits**	**5 bits**	**5 bits**	**6 bits**	**Comments**
R	op	rs	rt	rd	shamt	funct	Arithmatic
I	op	rs	rt	immediate			immediate/branch
J	op	Target					Jump

<u>Only the following instructions have to be implemented in this project.</u>

Table 1: List of instructions for your processor to be implemented

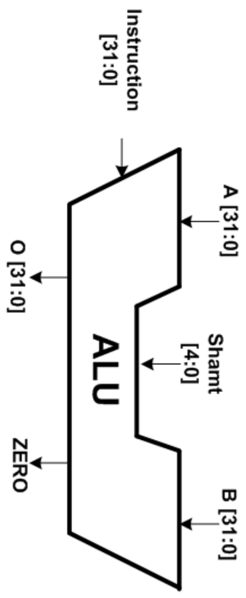
R-Type instructions						
Instruction	**Opcode(6) [31-26]**	**RS(5) [25-21]**	**RT(5) [20-16]**	**RD(5) [15:11]**	**Shamt(5) [10:6]**	**funct (6) [5:0]**
addu $RD, $RS, $RT **$RD = $RS+ $RT**	000000	RS	RT	RD	00000	100001
subu $RD, $RS, $RT **$RD = $RS- $RT**	000000	RS	RT	RD	00000	100011
nor $RD, $RS, $RT **$RD = ~($RS | $RT)**	000000	RS	RT	RD	00000	101111
Sll $RD,$RT, shamt **$RD = $RT << shamt**	000000	0 (don't care)	RT	RD	shamt	000000
Srl $RD,$RT, shamt **$RD = $RT >> shamt**	000000	0 (don't care)	RT	RD	shamt	000010

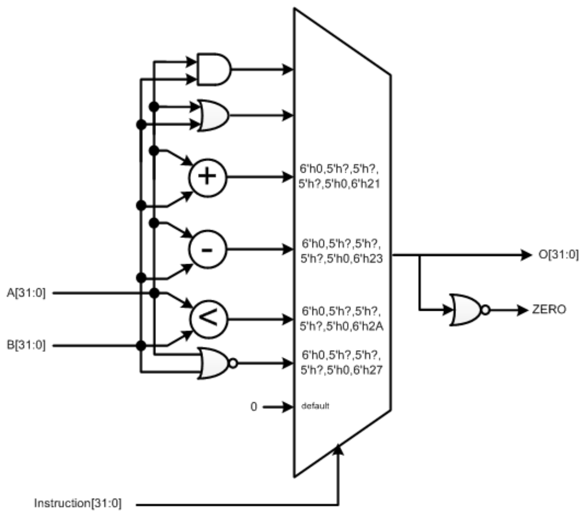
I-Type Instructions				
Instruction	**Opcode (6) [31-26]**	**RS(5) [25-21]**	**RT(5) [20-16]**	**Immediate (16) [15-0]**
addiu $RT, $RS, imm **$RT = $RS + signext(imm)**	001001	RS	RT	imm
andi $RT, $RS, imm **$RT = $RS & imm**	001100	RS	RT	imm
beq $RS, $RT, imm **if ($RS = $RT) PC = PC + 4 + (signext(imm) << 2)**	000100	RS	RT	imm
bne $RS, $RT, imm **if !($RS = $RT) PC = PC + 4 + (signext(imm) << 2)**	000101	RS	RT	imm
lw $RT, imm($RS) **$RT = mem[$RS + signext(imm)**	100011	RS	RT	imm
sw $RT, imm($RS) **mem[$RS + signext(imm)] = $RT**	101011	RS	RT	imm

Table 2: Nomenclature

Terminology	Description
R-Type inst.	Register type are instructions that operate on registers of the processor
I-Type inst.	Immediate value type instructions
RS	Source register
RT	Target register
RD	Destination register
Imm	Immediate Value
target	Target address
Shamt	Shift amount
funct	Function
addu	Add unsigned, no overflow
subu	Subtract unsigned
addiu	Add immediate unsigned, no overflow
Lw	Load word
Sw	Store word
beq	Branch if equal
bne	Branch if not equal
Sll	Shift left logical
Srl	Shift right logical

<u>32 BIT ARTHIMETIC AND LOGIC UNIT</u>




6'h0,5'h?,5'h?,5'h?,5'h0,6'h21
6'h0,5'h?,5'h?,5'h?,5'h0,6'h23
6'h0,5'h?,5'h?,5'h?,5'h0,6'h2A
6'h0,5'h?,5'h?,5'h?,5'h0,6'h27
0
default
A[31:0]
B[31:0]
O[31:0]
ZERO
Instruction[31:0]

ALU is a simple MUX. **The select signal is Instruction[31:0]**. Depending upon instruction, different operations will be performed on operand(s).
MUX in fig 2 shows execution of few of the instructions, using last 6 bits of instruction. There are many more operations that can be performed in ALU, using other bits in Instruction.

"O" is the output of this operation. "Zero" tells if all bits in output are zero.

<u>QUESTION</u>

1) Write the Verilog code to Implement the 32 bit ALU with suitable comments.
2) Create a Verilog Testbench, commenting as to what is done and why, to validate the design for all the instructions mentioned above.
3) Obtain the simulation waveforms for the same and put it in the report along with the Verilog design and the testbench with suitable comments.

The document must contain the Verilog codes mentioned above with simulation waveforms with clear labels. Please make some kind of distinction between general document text and the code. Please put effort in creating a good, structured and concise document, quality of this document will also be considered in grading.

<u>ADDITIONAL REFERENCES</u>

- http://people.cs.pitt.edu/~xujie/cs447/MIPS_Instruction.htm
- http://www.mrc.uidaho.edu/mrc/people/jff/digital/MIPSir.html