ENGR 468/568 Advanced Digital System Design

Course Project -2020

Design of a Simple 32-bit Processor

You are required to form your team (4 students preferred, maximum 5 students) and start working ASAP on this first milestone to meet the deadlines.

Each project member must contribute to **at least 20%** of the project. **You will be assessed individually.** If a group member declares that he/she contributed to all the modules, then they should be ready to answer questions about any of the modules during the live zoom presentation.

Milestone #1: 75% of Project Marks and is due on Monday November 16th @ 11:59 pm

Deliverables:

- **Design of simple ALU;**
- **Design of register banks along with decoding and multiplexing circuits;**
- **Design of memory access block along with its multiplexing (used to complete the execution of the LDR and STR instructions);**
- **Design of the RAM.**

You have to **upload to Canvas by the deadline** all your Verilog-HDL files, testbench, as well as your simulation results.

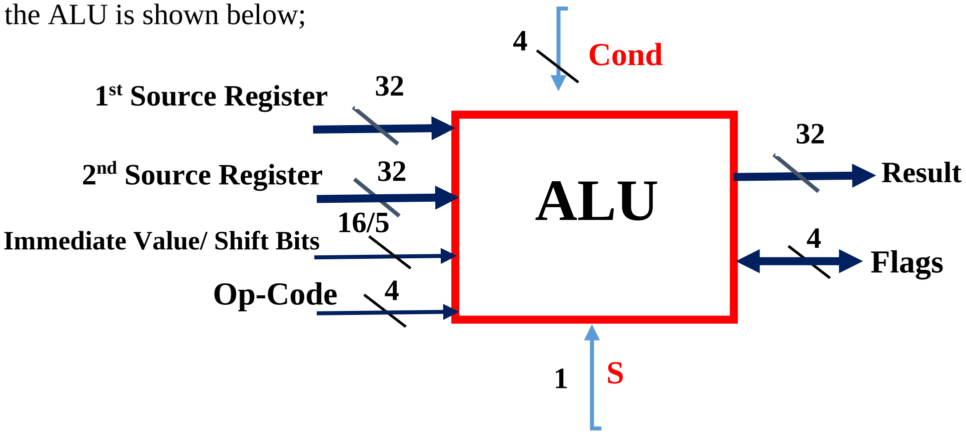
Design of Simple ALU

We would like to design a 32-bit ALU as part of a simple computer architecture. The ALU receives two inputs that are taken from the register bank (only 16 registers; R0 to R15). One output of the ALU is the 32-bit result of the required operation. This output is written back to the register bank. There is another 4-bit status flags output. The following table summarizes the available instruction set (R1, R2, and R3 are shown as examples but you can use any other register (R0 to R15):

Instruction	Description	Op Code
ADD R1, R2, R3	(R1=R2+R3)	0000
SUB R1, R2, R3	(R1=R2-R3)	0001
MUL R1, R2, R3	(R1=R2*R3) (assume the result is 32-bit maximum)	0010
ORR R1, R2, R3	(R1=R2 OR R3)	0011
AND R1, R2, R3	(R1=R2 AND R3)	0100
EOR R1, R2, R3	(R1=R2 XOR R3)	0101
MOV R1, *n*	Initialize R1 with an immediate number *n*, $0 <= n <= (2^{16} - 1)$	0110
MOV R1, R2	Copy R2 to R1	0111
MOV R1, R2, LSR #*n*	(R1=R2 >>*n*), $1 <= n <= 31$	1000
MOV R1, R2, LSL #*n*	(R1=R2 <<*n*), $1 <= n <= 31$	1001
MOV R1, R2, ROR #*n*	(R1=Rotate right R2 by *n*-bit) , $1 <= n <= 31$	1010
CMP R1, R2	Compare R1 with R2 and set the status flags	1011
ADR R1, *n*	Initialize R1 with a 16-bit address *n*, $0 <= n <= (2^{16} - 1)$	1100
LDR R2, [R1]	Load R2 with the contents at memory address R1	1101
STR R2, [R1]	Store R2 at memory address R1	1110
NOP	No Operation - Skip this instruction	1111

Assume that the instruction has been fetched and decoded. The values of *first source register*, *second source register*, *load/shift bits*, *Op-Code, and Cond* are ready as inputs to the ALU. Your task is to design an ALU that execute the instruction as per the **decoded** Op-Code and generates the associated 32-bit result as well as the 4-bit flags. Assume the shifted register is the *second source register*.

The block diagram of the ALU is shown below;



The 4-bit Flags register ({N, Z, C, V}) are set by a CMP instruction or when the S option is set to one (bit # 23 of the instruction). N is set to 1 if the result is negative, Z is set to 1 if the result is zero, C is set to 1 if the result generates a carry, and V=1 if the result generates an overflow. For the Load/Shift bits, bit [6] to bit [10] are used if the instruction includes a Shift or ROR. If the instruction involves loading an immediate value, bits 3 to 18 are used to load the 16-bit immediate value. Bits 0 to 2 are left unused for the purposes of this project.

The ALU design should be a modular one. You should write a separate module for each of the following ALU operations (sub-designs);

a) A 32-bit adder

b) A 32-bit subtractor

c) A 32-bit multiplier

d) A 32-bit bitwise ORing

e) A 32-bit bitwise ANDing

f) A 32-bit bitwise XORing

g) A parameterized 32-bit right shift register that shifts the input by *n*-bit

h) A parameterized 32-bit left shift register that shifts the input by *n*-bit

i) A parameterized 32-bit register that right rotates the input by *n*-bit

j) A 32-line *16*x1 Multiplexer (each input/output is an 32-bit wide)

k) A module that checks the S-bit /CMP instruction and generates the 4-bit flag accordingly. You may need to Google to learn how the four flags will be calculated.

l) 8-bit Counter (Program Counter (PC))

m) Other small modules that cover the remaining functions of the 15-instruction set (such as MOV and LDR).

Make sure you compile and simulate each of these sub-modules before instantiating and integrating all of them to form the ALU. The ALU design should be then compiled and simulated.

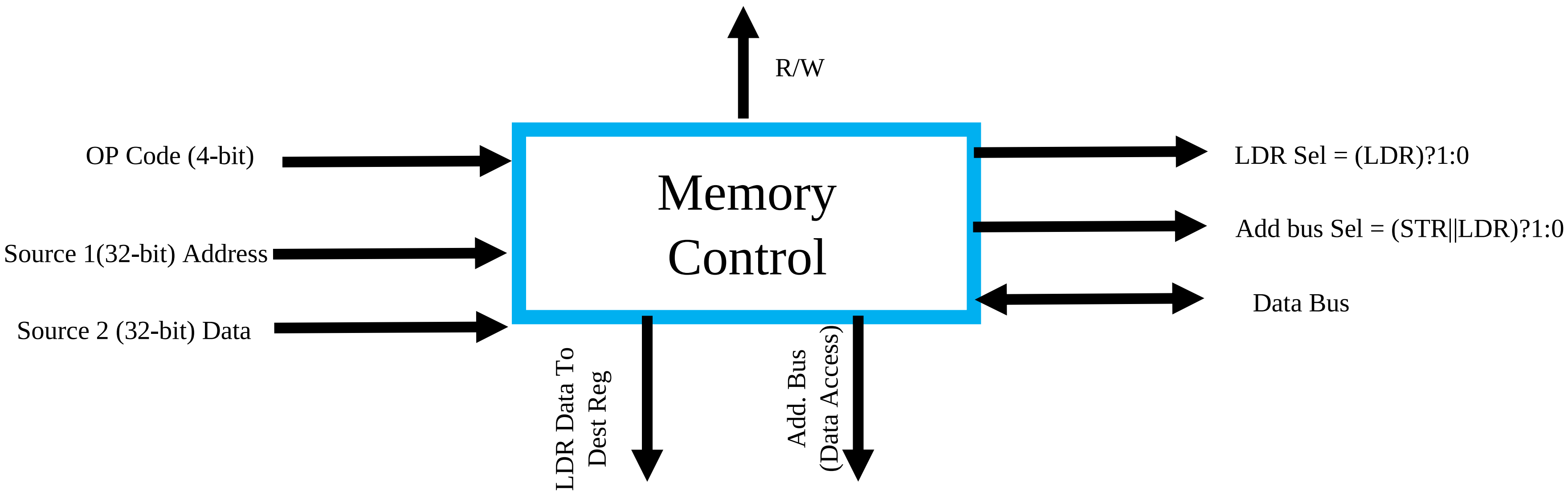
Your ALU should recognize the optional Cond bits (the four MSB of the instruction) as per the following table;

Cond bits	**Condition**
0000	No Condition
0001	EQ - Equal
0010	GT - Greater Than
0011	LT – Less Than
0100	GE – Greater Than or Equal To
0101	LE – Less Than or Equal To
0110	HI – Unsigned Higher
0111	LO – Unsigned Lower
1000	HS – Unsigned Higher or Same

We assume the PC is an 8-bit counter that always starts from 0 (first address of your RAM) and increment by 1 at the end of the fetch cycle.

Even though each block looks simple by itself, connecting the I/O and testing your codes may be a very time-consuming task. Start as early as possible and not to leave it to the last minutes. You have to **upload to Canvas by the deadline** all your Verilog-HDL files, testbench, as well as your simulation results.

Decoding of LDR and STR



During an LDR cycle, we only have one destination and one source. The source contains the address of the data in memory. For the purposes of this project, consider Source 1 to point to the address.

During an STR cycle, we consider both Sources and neglect the destination. Source 1 will point to the address and Source 2 will contain the data.

Milestone #2: 25% of project marks and is due on the week of 23th to 27th of November.

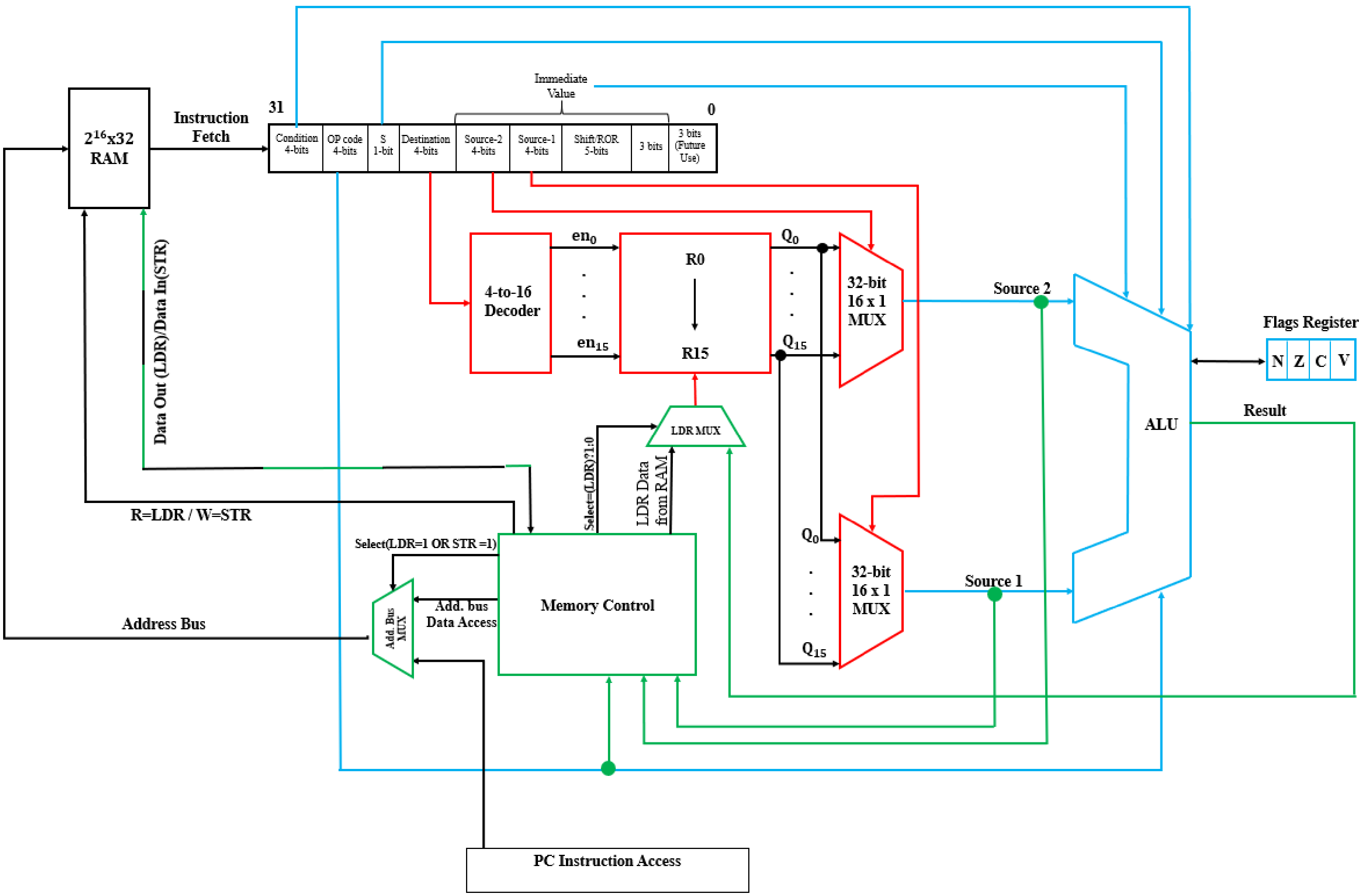
Deliverables:

- **Connecting the pieces of your processor**

  You connect the ALU to your Register Bank, RAM, Decoding, and Memory Control. During the group demo, you will be given a data file of maximum of 16 instructions, along with some data values that you will be copying to your RAM first (homework #4). Then your processor will start the standard fetch-decode-execute cycle of the 16 instructions to output the correct result. Your testbench file should display all the register values as well as the output data file (copy of your RAM).

- **Live Zoom Presentation:**

  Each group will be assessed separately over live zoom interviews. Every group member will be asked to clearly identify which section they worked on and will be assessed accordingly. The assessment will involve making tweaks/changes to the code to evaluate your understanding. Each student will be given an individual grade for this portion. If a group member declares that he/she contributed to all the modules, then they should be ready to answer questions about any of the modules in the design. Note: If a student is not capable of demonstrating sufficient contribution to the project, then their overall project score will be impacted.



Condition 4-bits	OP code 4-bits	S 1-bit	Destination 4-bits	Source-2 4-bits	Source-1 4-bits	Shift/ROR 5-bits	3 bits	3 bits (Future Use)