CS 322 Term Project

Name- Tarusi Mittal RollNo-1901Cs65

How to operate:

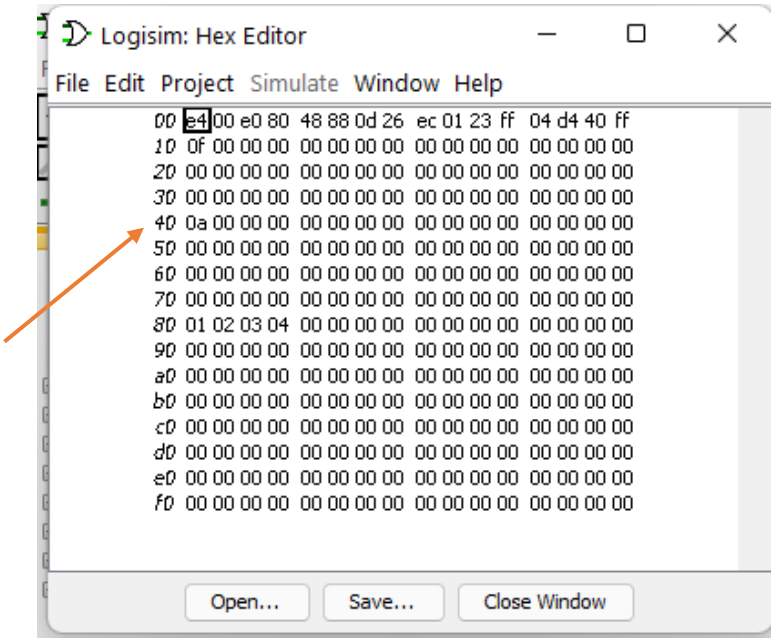
When you will open the file named 1901CS65.circ it will ask to load the Alu file so please load that file which is provided in the same zip folder.

After that right click on RAM and select the option load image. Then load the image titled sample.img which is given in the same zip file.

Then click on simulation and select ticks frequency and select that to be 16Hz. Then click on ticks enabled. Then you will see in RAM it will simulate and when you notice that the pointer is moving on ff 0f 00 again click on simulation and disable ticks enabled.

Now if you will right click on the RAM and click on edit contents you will see along 40 a number written which is basically our answer.

For an example I have used addition of 4 numbers ie 01 02 03 04 and the answer should be 10 which in hexdecimal is written as 0a so you will see alongside 40 0a written



The complete explanation is shown below:

Design

The CPU supports only 256 bytes of memory to store instructions and data due to its 8-bit data bus and 8-bit address bus.

On the inside, there are four 8-bit registers, R0 through R3, along with an Instruction Register, a Program Counter, and an 8-bit register for storing immediate values.

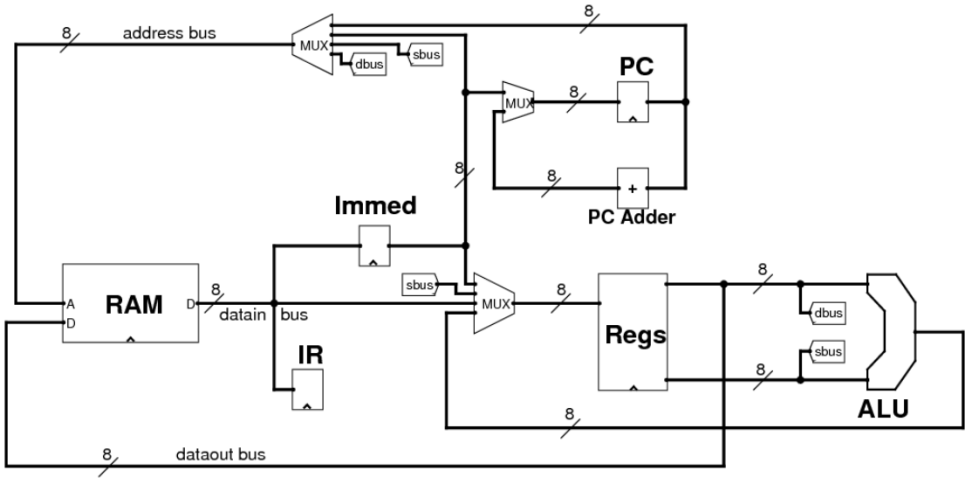
Two 8-bit values can be used as inputs for the ALU to perform AND, XOR, ADD, and SUB operations, as well as signed ADDs and SUBs.

Unlike the ALU, which only reads from and writes back to registers, the CPU is designed as a load/store architecture: data is brought into the registers before being manipulated.

There are two operands in the ALU operations: one register is a source register, and the second register is both a source register and a destination register, i.e. destination register = destination register OP source register.

There are no PC-relative branches; all jump operations are absolute jumps. The jump instruction is based on whether the destination register is zero or negative, and there is also a "jump always" instruction.

The following diagram shows the datapaths in the CPU:



Lines with the labels dbus and sbus are the lines coming out of the register file. These lines contain the values of the destination and source registers.

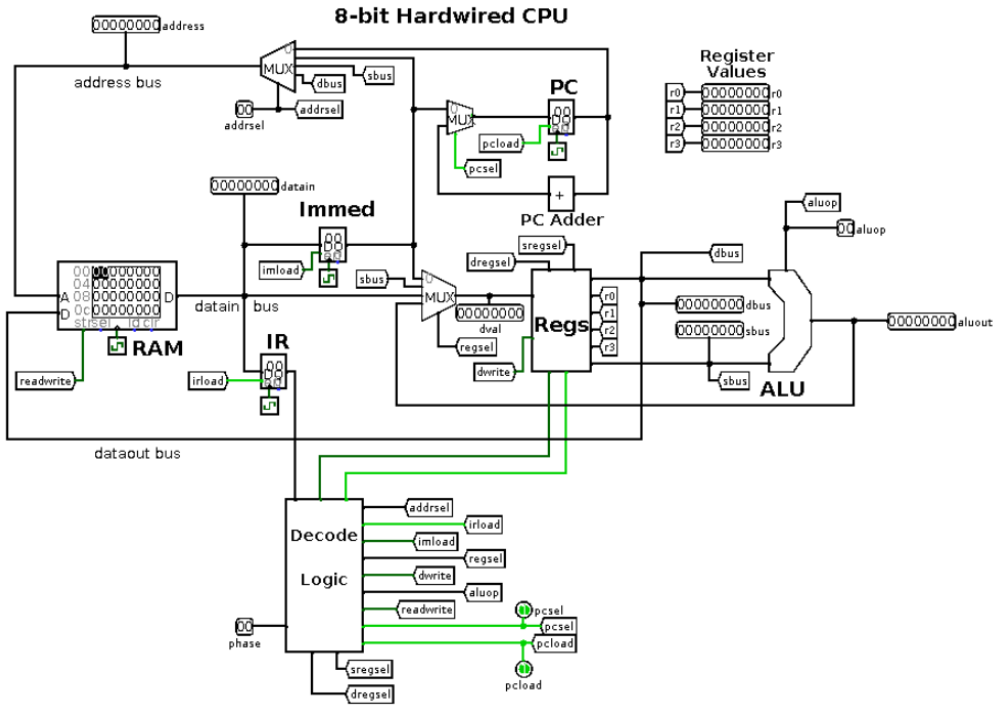
This data loop involves the registers and the ALU, whose output can only be read back into the register.

In this case, the only value that can be written to memory is the destination register because the dataout bus is connected to the dbus line only

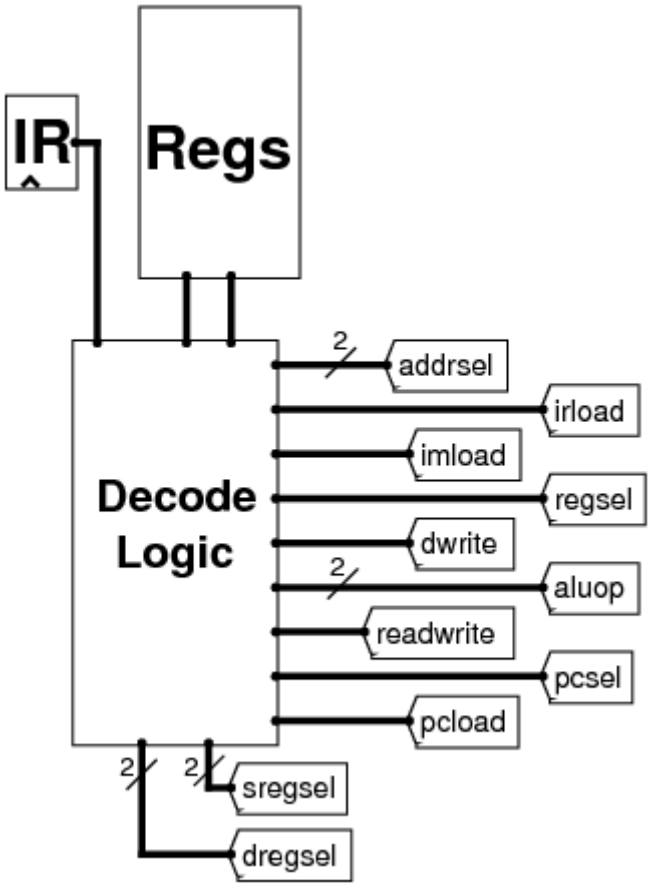
Instruction Coding

op1	op2	Mnemonic	Purpose
00	00	AND Rd, Rs	Rd = Rd AND Rs
00	01	XOR Rd, Rs	Rd = Rd XOR Rs
00	10	ADD Rd, Rs	Rd = Rd + Rs
00	11	SUB Rd, Rs	Rd = Rd - Rs
01	00	LW Rd, (Rs)	Rd = Mem[Rs]
01	01	SW Rd, (Rs)	Mem[Rs] = Rd
01	10	MOV Rd, Rs	Rd = Rs
01	11	NOP	Do nothing
10	00	JEQ Rd, immed	PC = immed if Rd == 0
10	01	JNE Rd, immed	PC = immed if Rd != 0
10	10	JGT Rd, immed	PC = immed if Rd > 0
10	11	JLT Rd, immed	PC = immed if Rd < 0
11	00	LW Rd, immed	Rd = Mem[immed]
11	01	SW Rd, immed	Mem[immed] = Rd
11	10	LI Rd, immed	Rd = immed
11	11	JMP immed	PC = immed

CPU Control Lines

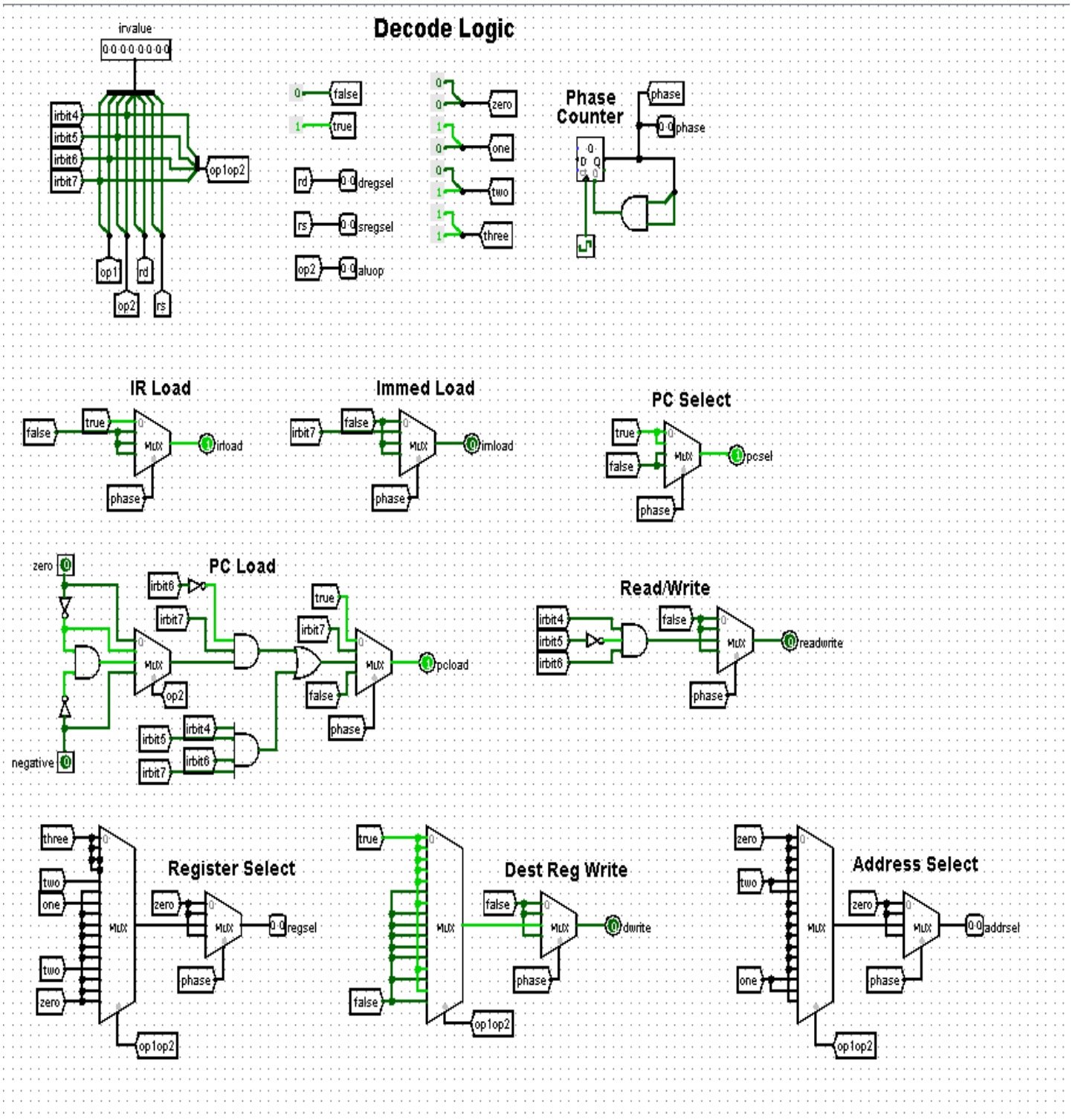


- There are several 1-bit control lines:
  - *pcsel*, increment PC or load the jump value from the Immediate Register.
  - *irload*, load the Instruction Register with a new instruction.
  - *imload*, load the Immediate Register with a new value.
  - *readwrite*, read from memory, or write to memory.
  - *dwrite*, write a value back to a register, or don't write a value.
  - *pcload*, load the PC with a new value, or don't load a new value.
- There are also several 2-bit control lines:
  - *addrsel*, select an address from the PC, the Immediate Register, the source register or the destination register.
  - *regsel*, select a value to write to a register from the Immediate Register, another register, the data bus or from the ALU.
  - *dregsel* and *sregsel*, select two registers whose values are sent to the ALU.
  - *aluop*, which are the *op2* bits that control the operation of the ALU.
- The values for all of these control lines are generated by the Decode Logic, which gets as input the value from the Instruction Register, and the zero & negative lines of the destination register.

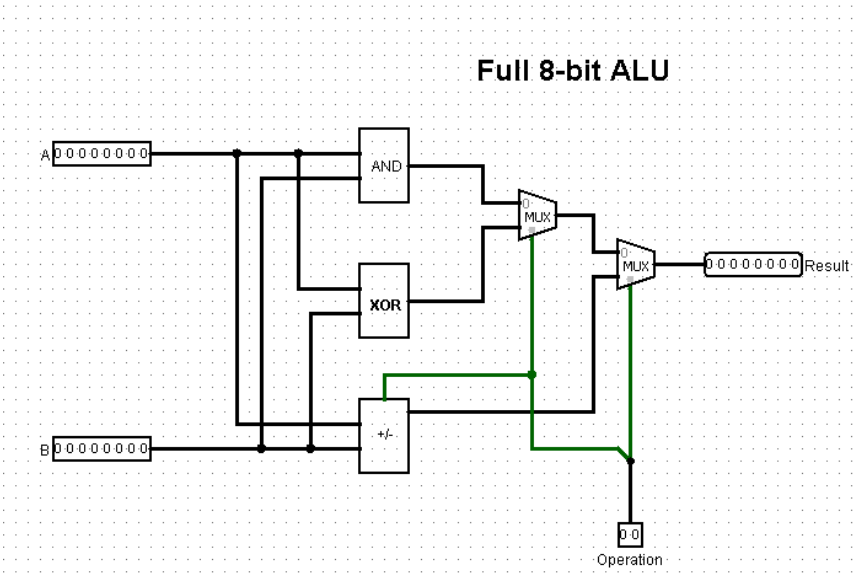


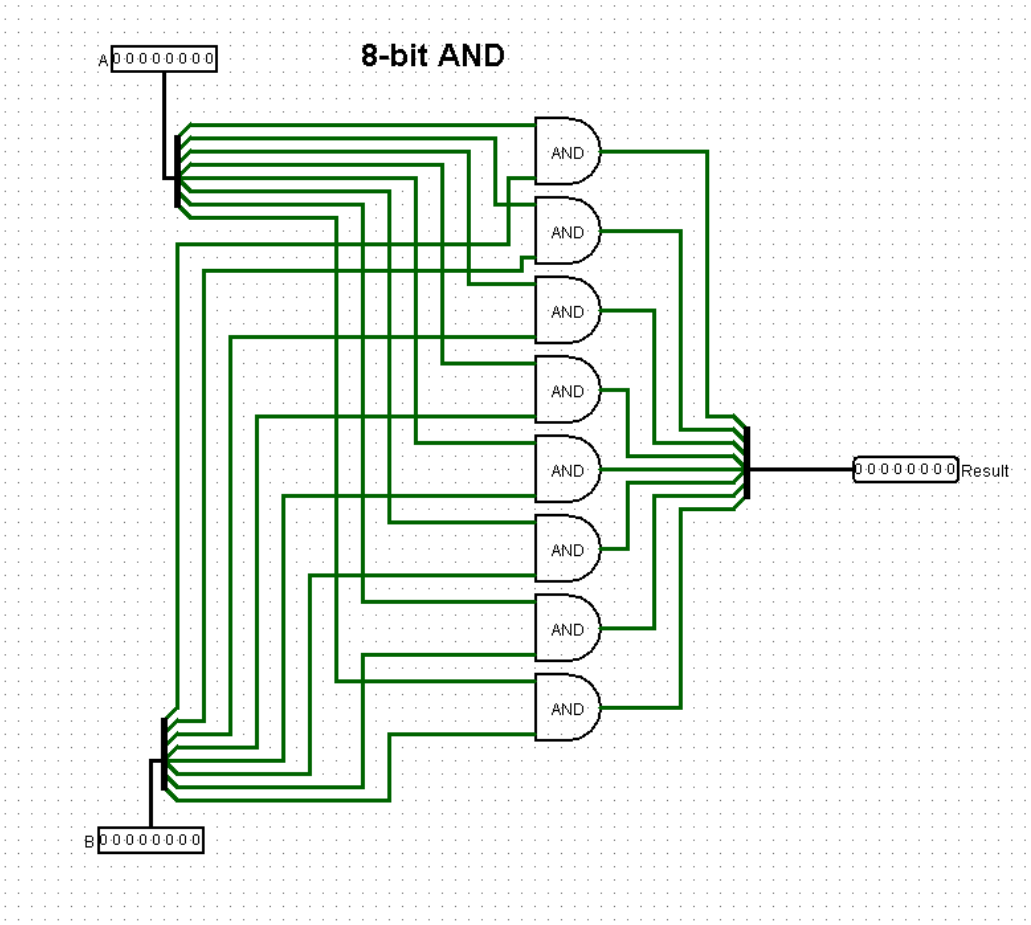
Decode Logic

Inside the Decode Logic block, the value from the Instruction Register is split into several individual lines *irbit4*, *irbit5*, *irbit6* and *irbit7*. *op1* and *op2* are split out, with *op2* exported as *aluop*. Finally, the 4 opcode bits from the instruction are split out as the *op1op2* line.



ALU


Full 8-bit ALU
A 00000000
B 00000000
AND
XOR
+/-
MUX
MUX
00000000 Result
00
Operation


8-bit AND
A 00000000
B 00000000
AND
AND
AND
AND
AND
AND
AND
AND
00000000 Result

An Example Program

- In memory starting at location 0x80 is a list of 8-bit numbers; the last number in the list is 0.
- I have written a programme to sum the numbers as an example, store the result into memory location 0x40, and loop indefinitely after that.
- We have 4 registers to use. They are allocated as follows:
  - R0 holds the pointer to the next number to add.
  - R1 holds the running sum.
  - R2 holds the next number to add to the running sum.
  - R3 is used as a temporary register.
- Here is the assembly-style code for the program.

```
        LI  R1,0x00     # Set running sum to zero
        LI  R0,0x80     # Start at beginning of list
  loop: LW  R2, (R0)    # Get the next number
        JEQ R2, end     # Exit loop if number == 0
        ADD R1, R2      # Add number to running sum
        LI  R3, 0x01    # Put 1 into R3, so we can do
        ADD R0, R3      # R0++
        JMP loop        # Loop back
  end:  SW  R1, 0x40    # Store result at address 0x40
  inf:  JMP inf         # Infinite loop
```

- Converting to machine code, here are the hex values to put into memory starting at location 0:

LI R1,0x00	e4 00
LI R0,0x80	e0 80
LW R2, (R0)	48
JEQ R2, end	88 0d
ADD R1, R2	26
LI R3, 0x01	ec 01
ADD R0, R3	23
JMP loop	ff 04
SW R1, 0x40	d4 40
JMP inf	ff 0f

- And as written in the how to operate section we can see the results.
- On the LW instruction, we will see as the *sbus* value is selected to be placed on the address bus, and the datain value is written to the destination register.
- On ALU instructions, the *sbus* and *dbus* values, the *aluop*, and the result which is written back into the destination register.
- On the JEQ instruction, the value of N and Z into the Decode Logic, and the resulting *pcsel* and *pcload* values

REFRENCES:

Books:
1. The MIPS Programmer's Handbook (The Morgan Kaufmann Series in Computer Architecture and Design)
2. MIPS RISC architecture Book by Gerry Kane
WebLinks:
http://cmosedu.com/cmos1/electric/Senior%20Design%20Report.pdf
https://github.com/yxwangcs/MIPS-CPU
https://en.wikipedia.org/wiki/MIPS_architecture
https://minnie.tuhs.org/CompArch/Tutes/week03.html
https://cs.stanford.edu/people/eroberts/courses/soco/projects/risc/mips/index.html