# ***OUR VISION***

Our architecture will be a 32-bit system. The 32-bits will be divided into 2 parts of 16-bits . The first 16-bits will contain a single instruction and the other 16-bits will contain another instruction.

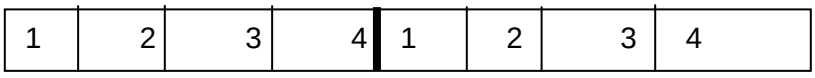
However, the execution of branch, jump and multiplication instructions will be executed by both the 1st 16-bits and the 2nd 16-bits by concatenating them together to provide a space of 32-bits divided into 4-bits Op-code, 4-bits for a register and the rest of the 20-bits for the immediate value. Note that for the special case of the jump instructions the 8-bits will be ignored. In the case of multiplication immediate 4-bits will be ignored.

The 1st 16-bits will not be allowed to execute read and write instructions because they do not have access to the data memory. This approach was inspired by the Harvard architecture approach along with VLIW (very long instruction word).

The advantages of using the Harvard architecture system is the ability to fetch data and instructions at the same time and the advantage of using VLIW is the ability to take 2 instructions and execute them at the same time. This approach will maximize the throughput of executing instructions using the least amount of power.

Instead of having a $zero register and an ADDI instruction like MIPS, we will remove the $zero register and the ADDI instruction and replace them with a $one register and a MULI (multiply immediate).

Moreover, our architecture will be word addressable not byte addressable so when we increment the PC to jump to the next instruction we will increment it by 1 not by 4.
Here is a roadmap of our 2 formats of instructions:
DEUX Format:




Instruction 1 Instruction 2

1. Opcode ,4-bits
2. Destination register,4-bits
3. Operand register , 4-bits
4. Operand register, 4-bits

Here are the instructions that will use the DEUX Format:-

- Opcode:0000 [ADD #register1,#register2,#register3] :- register1 will contain the result/destination and the other 2 registers will contain the operands.

- Opcode:0001 [SUB #register1,#register2,#register3] :- register1 will contain the result/destination and the other 2 registers will contain the operands.

- Opcode:0010 [DIV #register1,#register2,#register3] :- register1 will contain the result/destination and the other 2 registers will contain the operands.

- Opcode:0011 [MOD #register1,#register2,#register3] :- register1 will contain the result/destination and the other 2 registers will contain the operands.

- Opcode:0100 [SLT #register1,#register2,#register3] :- register 1 will be set to one if register 2 is less than register 3 and will be set to zero otherwise

- Opcode:0101 [LW #register1,#register2] :- register 1 will receive the information you want to load from the memory and register2 will contain the address of the memory location that contains the requested data.

- Opcode:0110 [SW #register1,#register2] :- register 1 will contain the data that you want to store and register2 will contain the memory address that you want to write the data in.

- Opcode:0111 [SLL #register1,#register2,#register3] :- register 1 will contain the result , register2 will contain the value and register 3 will contain the value by which you want to shift the value of register2.(shift logical left)

- Opcode:1000 [SLR #register1,#register2,#register3] :- register 1 will contain the result , register2 will contain the value and register 3 will contain the value by which you want to shift the value of register2.(shift logical right)

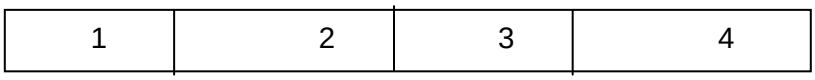
- Opcode:1001 [AND #register1, #register2,#register3] :- register 1 will contain the result, register2 and register3 will contain the operands

- Opcode:1010 [OR #register1, #register2,#register3] :- register 1 will contain the result, register2 and register3 will contain the operands

- Opcode:1011 [NOT #register1, #register2,#register3] :- register 1 will contain the result, register2 and register3 will contain the operands

- Opcode:1100 [XOR #register1, #register2, #register3] :- register 1 will contain the result, register 2 and register 3 will contain the operands.

UN Format :




1. Opcode ,4-bits
2. Destination register , 4-bits
3. Operand register, 4-bits

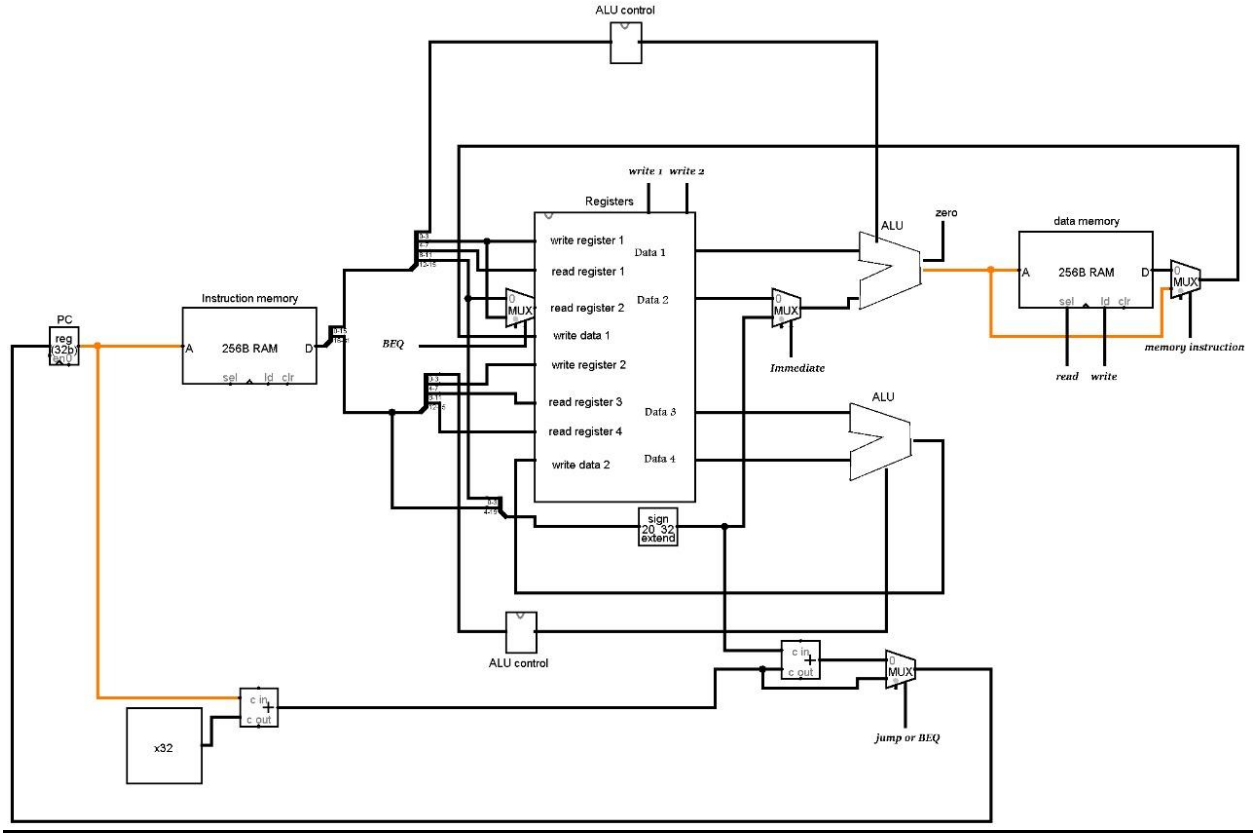
4. Immediate Value, 20-bits

- Opcode:1101 [MULI #register1 #register2 #immediate value]:- register 1 will contain the Destination register, register 2 will be ignored
- Opcode:1110 [BEQ #register1 #register2 #immediate value]:- register 1 will contain the first operand, register 2 will contain the second operand
- Opcode:1111 [j #register1 #register2 #immediate value]:- register 1 and register 2 will be ignored

The names of our registers will be as follows:

- #PC, not included in the 16 registers and not accessible by any instruction and is only changed indirectly
- #1 , will be a register with a 1 that is hardcoded in it
- #sp , will be the stack pointer
- #gp, will be the global pointer
- #i0 , #i1 ,#i2 , will be input registers (arguments)
- #s0 ,#s1, #s2, #s3 , will be the saved registers
- #t0, #t1, #t2, #t3 will be temporary registers
- #r0,#r1 , will be return registers

<u>And here is our model:</u>


ALU control
write 1
write 2
Registers
write register 1
read register 1
read register 2
write data 1
write register 2
read register 3
read register 4
write data 2
Data 1
Data 2
Data 3
Data 4
ALU
zero
data memory
A
256B RAM
D
sel
ld
clr
MUX
PC
reg
(32b)
Instruction memory
BEQ
Immediate
memory instruction
read
write
sign
20_32
extend
c in
c out
x32
jump or BEQ