Faculty of Engineering CEE216/CEE210

Electrical Engineering Dep year Communication

Instructor// Dr: Gihan Naguib Spring 2020

TA: Muhammad Dakheel

**Simulation and Designing 32-bit Mips processor**

Group members: -

1.عبدالله محمد أمين صادق (2820)

2.علاء محمد صديق محمد (2821)

3.جاسر جمال أحمد أحمد (2810)

Group number **(9)**

**Introduction**

It’s so clear to any engineer studies computer architecture that there is some mane steps in designing a processor specially single cycle processor and we’ll conclude them roughly as following below.

**The steps of design a single cycle processor: -**

1) There are 5 main components -functions- in single cycle processor

1. PC 2. Register file 3. ALU 4. Memory which is helpful to -be separated into Instruction\_Mem and Data\_Mem 5. Control unit

. (علاء محمد + عبدالله محمد + جاسر جمال)

2) ALU implementation which is made to execute all the required operations given in the project (عبدالله محمد + جاسر جمال).

3) Register file implementation (جاسر جمال)

4) ALU Data Path. (عبدالله محمد ).

5)Data path j-type instructions (علاء محمد).

6)Connections of the circuit (علاء محمد + عبدالله محمد + جاسر جمال ).

7) Control signals table (علاء محمد + جاسر جمال ).

8) Control unit (عبدالله محمد + علاء محمد )

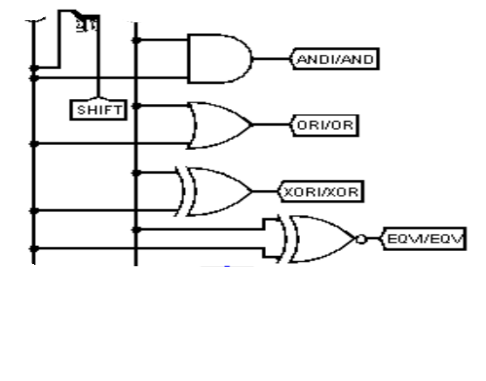
9) converting test code to machine language then entering instructions to memory (عبدالله محمد+ جاسر جمال ).

10) Repairing problems and errors in the design (عبدالله محمد + علاء محمد) .

11) Following the Execution of the program and evaluating the final values . (عبد الله محمد + علاء محمد + جاسر جمال).

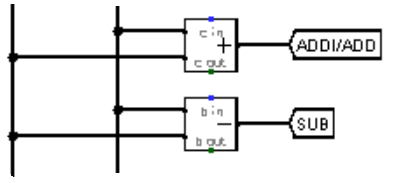
(12Making the documentation and the report ( جاسر جمال)

1. ***Design and Implementation***: -

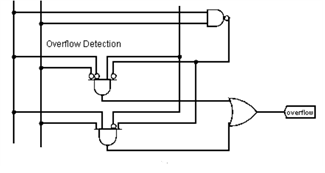
Firstly, we’ll talk about a useful component in performing most of our instructions called Arithmetic and Logical Unit, simply (ALU) which is considered below:

1. Logical operations:

AND, OR, XOR, EQV -XNOR-.

1.  Arithmetic operation:

add, subtract \_ with detecting over flow case.

1.  Comparison operation:

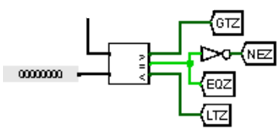
beq, bne, SLT, BEQZ, BNEZ, BLTZ, BGEZ, BGTZ, BLEZ.

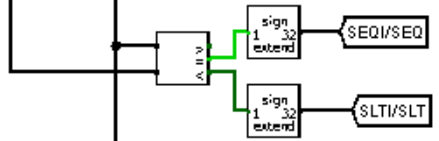
1. Shifting operation:

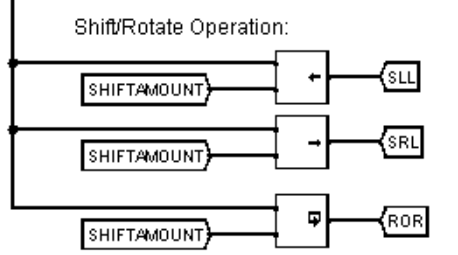
 sll, srl, RoR.

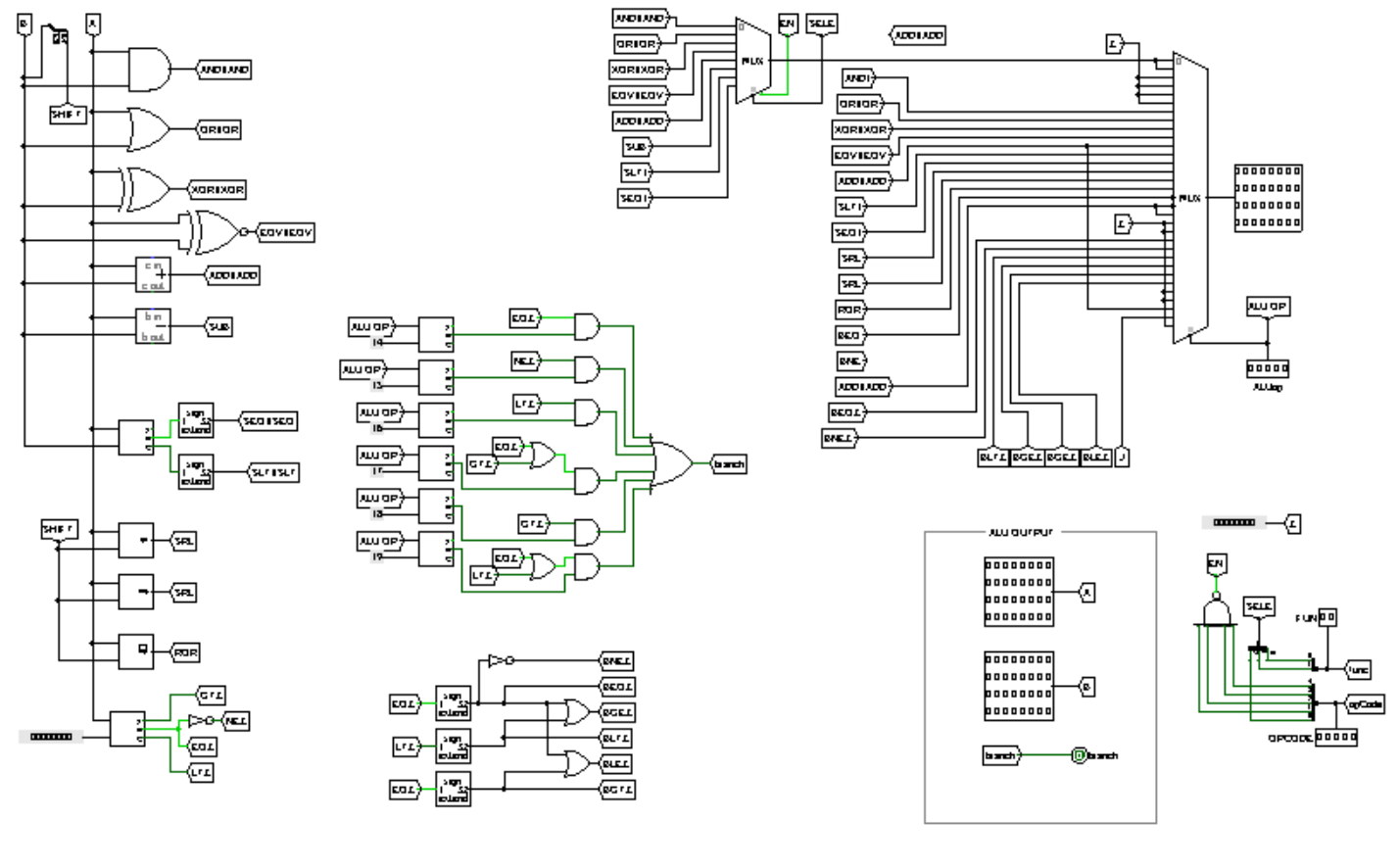
ALU has 2 outputs:

.1the result of “logical, arithmetic, shifting and Comparison operation”

 .2A flag for overflow

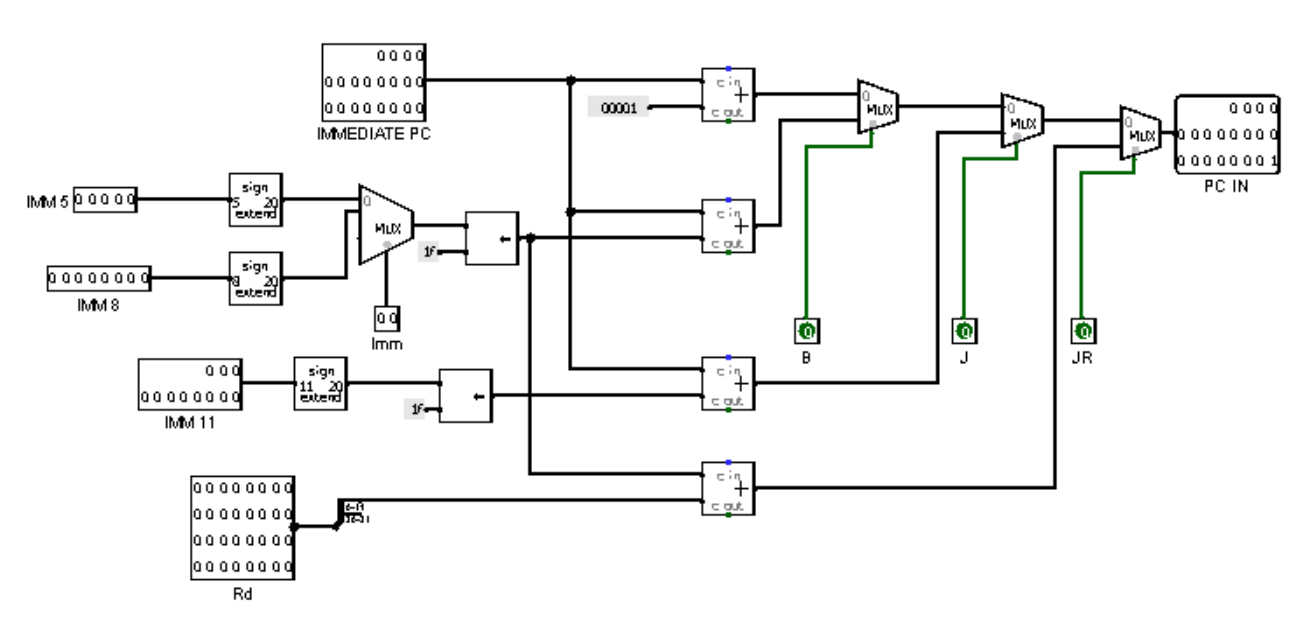




***Implement a 32-bit ALU to perform all the required operations-:***

Secondly, we’ll talk about program counter or simply (PC) that is used to fetch instruction from memory -which is separated here to instruction \_mem and data\_mem- and hold the address of the next instruction for execution.

* We have modified our (PC) to support the required given instructions as it’s considered in the following figure: -

***Implement a program counter (PC): -***

Thirdly, we’ll talk about how we can store \_ usage data in our processor specially, I mean register file that is an array of a processor, but differs from memory as it ’s so small comparing with memory, but too fast.

* There is two useful ways to implementa Register file are considered in below: -

1. Using Mux.

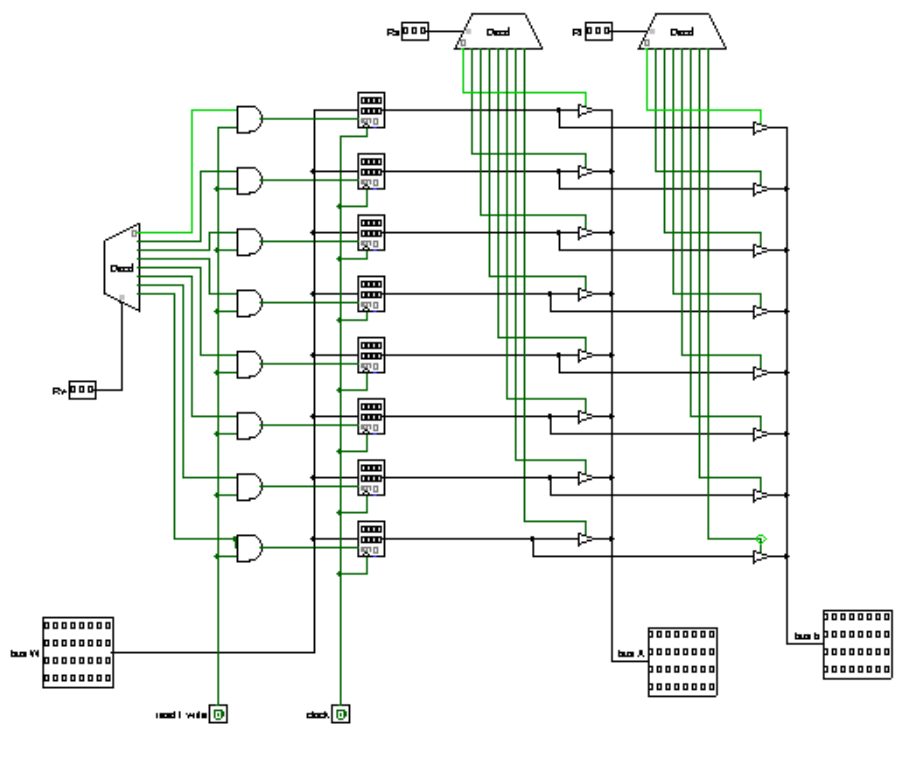
2.Using a decoder and tri-state.

* We have used the second method as it has two advantages: -

1.Reducing cost 2. Power consumption

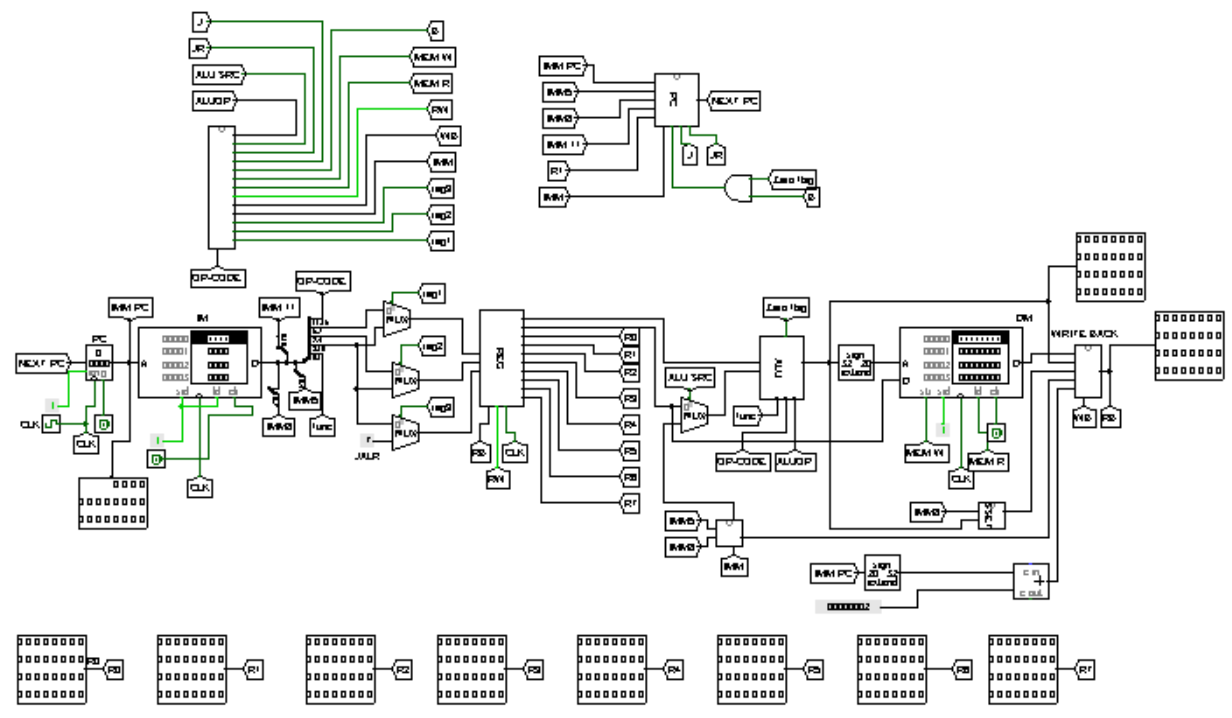
* As we only have need decoders and tri-state which are less hardware components comparing with using as we know from logic design Muxs consists of AND, OR, NOT logic gates.

***Implement a Register file containing eight 32-bit registers: -***



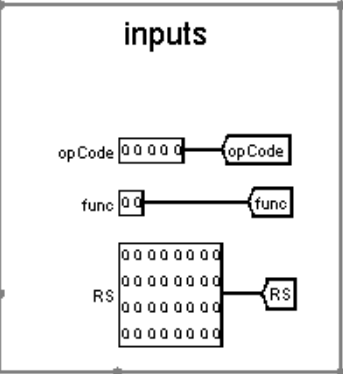
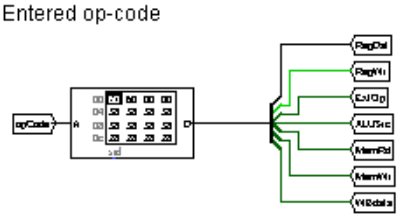
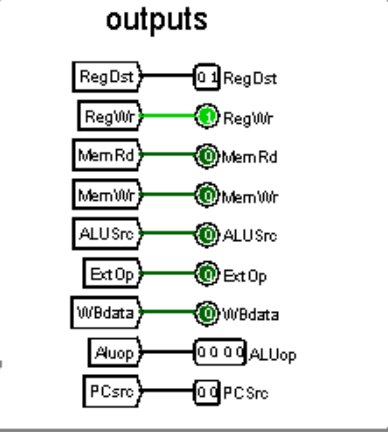
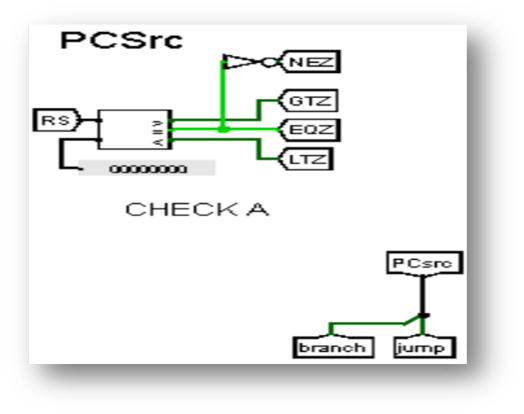
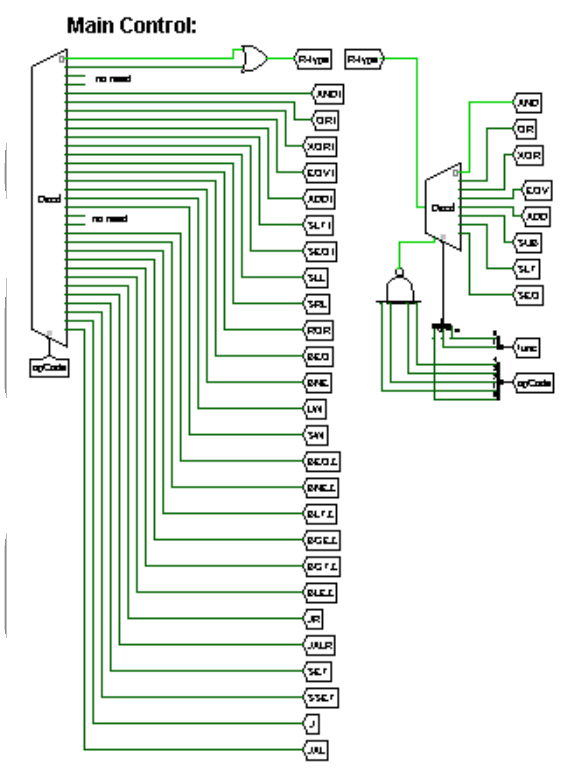
Next here we’ll talk about building a data-path that connects our devices -components – together- that we’ve talked about, earlier- to perform a certain instruction after knowing its type by the help of control unit which we’ll talk about at the end.

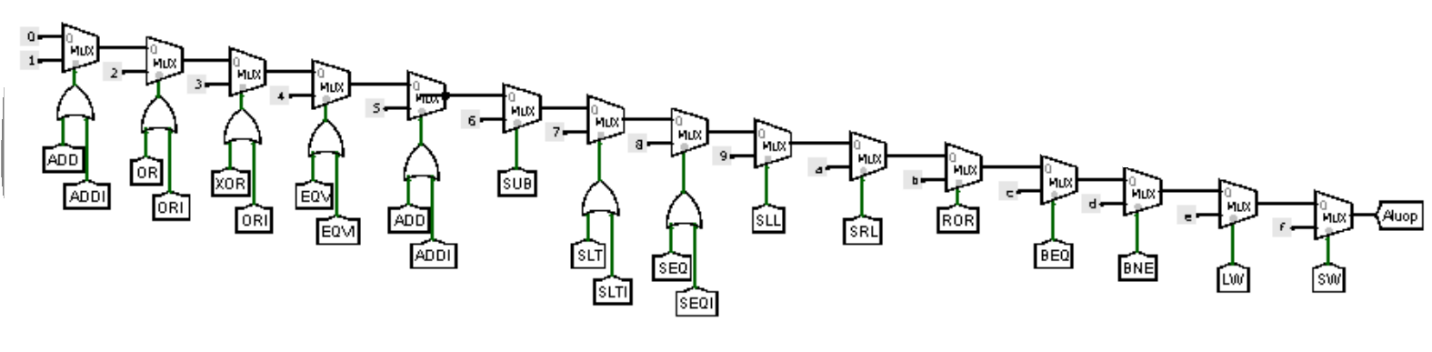
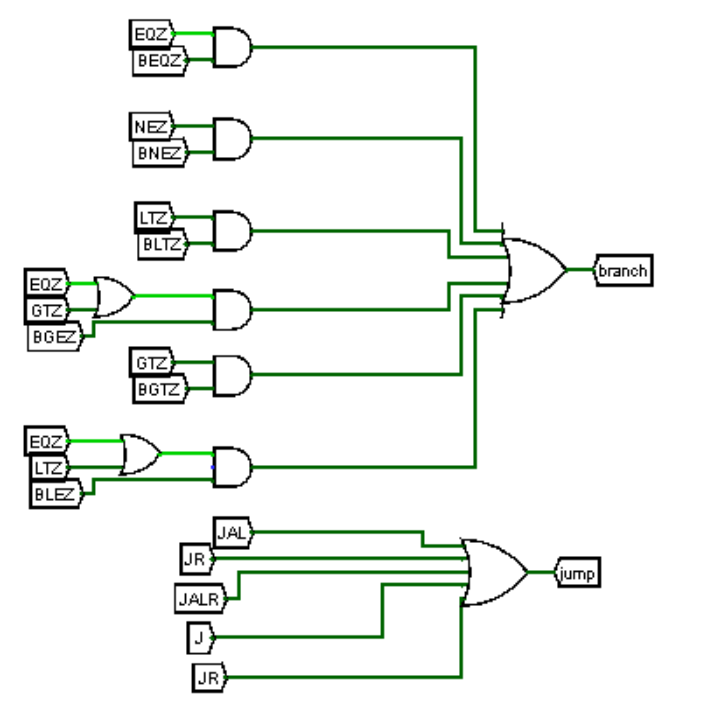
* We’ve applied connections which support the four given types of instructions R-type, I-type, B-type, J-type.

***Building a data-path: -***

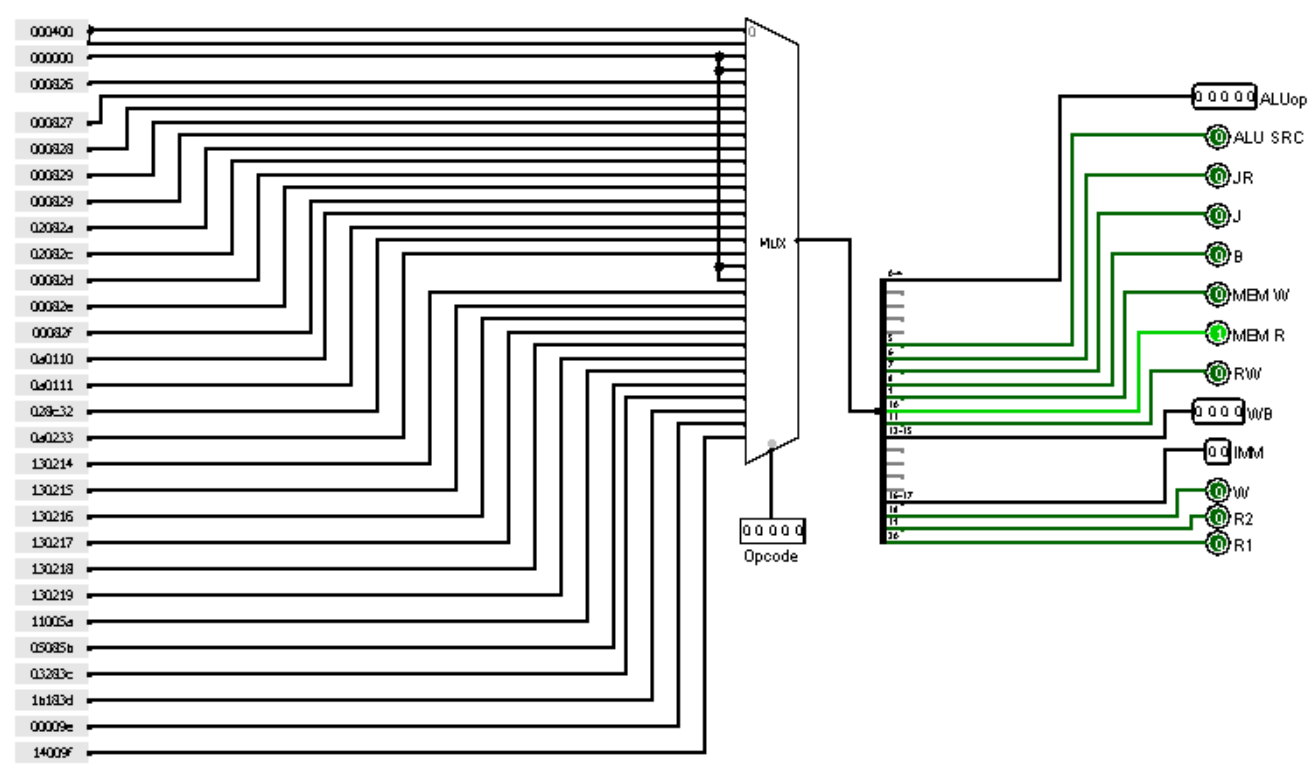
Finally, we’ll talk about the mind of the brain. I mean control unit that has a big role in generating signals to control our data-path and enabling components related to a certain instruction.

* We take inputs(I/o , MR ,WR ) from processor inter-face and fed it to a decoder to give us op-codes which is fed to our main control as a special marque which differs from types of instruction .However for arithmetic operation add , sub , addI , ….etc or specially, R-type instructions op-code doesn’t enough we should have another special marque with it which is Func (2bits) that all is fed to (ALU) through [ ALU -op ] which is a subset of op-code.

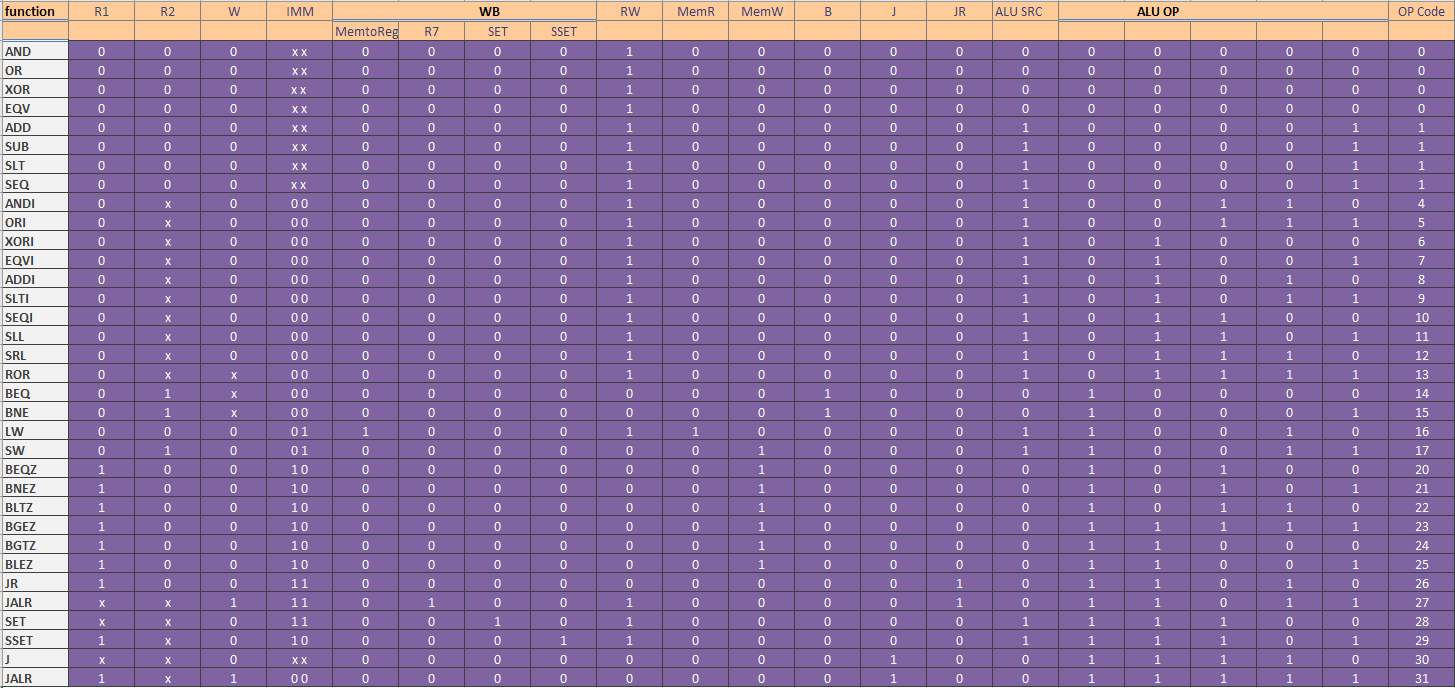


******Controlling B-type and J-type instructions: -**

The full main control circuit: -

**The full main control circuit: -**

***Table giving the control signal values for each instruction: -***

***Table giving the control signal values for each instruction: -***

***The logic equations for each control signal: -***

|  |  |
| --- | --- |
| **Groups** | |
| (20,21,22,23,28,29,30,31) | A.C |
| (24,25,26,27,28,29,30,31) | A.B |

R1 = AC + AB

|  |  |
| --- | --- |
| **Groups** | |
| (2,3,6,7,10,11,14,15,18,19,22,23,26,27,30,31) | D |
| (16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31) | A |

R2 = D + A

|  |  |
| --- | --- |
| **Groups** | |
| (18,19,22,23,26,27,30,31) | A.D |

W (Reg\_dis) = AD

|  |  |
| --- | --- |
| **Groups** | |
| (20,21,22,23,28,29,30,31) | A.C |
| (24,25,26,27,28,29,30,31) | A.B |

Imm = AC + AB

|  |  |
| --- | --- |
| **Groups** | |
| (16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31) | A |

Mem to Reg = A

|  |  |
| --- | --- |
| **Groups** | |
| (27,31) | A.B.D.E |

R7 = ABDE

|  |  |
| --- | --- |
| **Groups** | |
| (28) | A.B.C.D.E |

SET = ABCD'E'

|  |  |
| --- | --- |
| **Groups** | |
| (29) | A.B.C.D.E |

SSET = ABCD'E

|  |  |
| --- | --- |
| **Groups** | |
| (0,1,2,3,4,5,6,7) | A.B |
| (0,1,2,3,8,9,10,11) | A.C |
| (0,4,8,12) | A.D.E |
| (0,2,16,18) | B.C.E |
| (3,11,19,27) | C.D.E |
| (28,29,30,31) | A.B.C |

Reg\_W = A'B' + A'C' + A'D'E' + B'C'E' + C'DE + ABC

|  |  |
| --- | --- |
| **Groups** | |
| (16,18) | A.B.C.E |

Mem\_R = AB'C'E'

|  |  |
| --- | --- |
| **Groups** | |
| (17,19) | A.B.C.E |

Mem\_w = AB'C'E

|  |  |
| --- | --- |
| **Groups** | |
| (20,21,22,23) | A.B.C |
| (14,15) | A.B.C.D |
| (24,25) | A.B.C.D |

Branch = AB'C + A'BCD + ABC'D'

|  |  |
| --- | --- |
| **Groups** | |
| (30,31) | A.B.C.D |

J = ABCD

|  |  |
| --- | --- |
| **Groups** | |
| (18,19,26,27) | A.C.D |

JR = AC'D

|  |  |
| --- | --- |
| **Groups** | |
| (16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31) | A |
| (2,3,6,7,18,19,22,23) | B.D |
| (4,5,12,13,20,21,28,29) | C.D |
| (8,9,10,11,24,25,26,27) | B.C |

ALU Src = A + B'D + CD' + BC'

ALU Op:-

|  |  |
| --- | --- |
| **Groups** | |
| (20,21,22,23,28,29,30,31) | A.C |
| (24,25,26,27,28,29,30,31) | A.B |

Y1 = AC + AB

|  |  |
| --- | --- |
| **Groups** | |
| (12,13,14,15,28,29,30,31) | B.C |
| (10,11,14,15) | A.B.D |
| (16,17,18,19) | A.B.C |

Y2 = BC + A'BD + AB'C'

|  |  |
| --- | --- |
| **Groups** | |
| (16,17,18,19,24,25,26,27) | A.C |
| (6,7,14,15) | A.C.D |
| (8,9,24,25) | B.C.D |

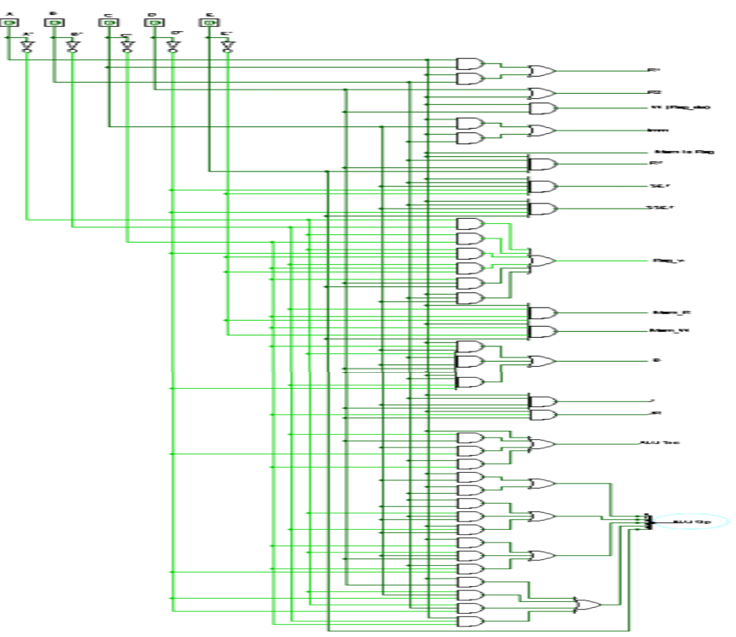
Y3 = AC' + A'CD + BC'D'

|  |  |
| --- | --- |
| **Groups** | |
| (18,19,22,23,26,27,30,31) | A.D |
| (4,5,12,13) | A.C.D |
| (8,9,12,13) | A.B.D |
| (16,17,18,19) | A.B.C |

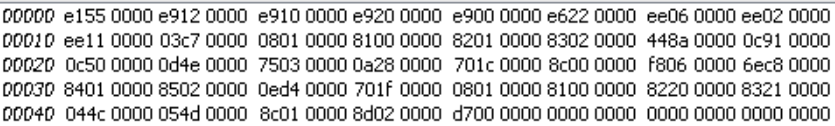
Y4 = AD + A'CD' + A'BD' + AB'C'

|  |  |
| --- | --- |
| **Groups** | |
| (1,3,5,7,9,11,13,15,17,19,21,23,25,27,29,31) | E |

Y5 = E



|  |  |  |
| --- | --- | --- |
| Instruction Bin\_num | Instruction | Hex |
| 1110000101010101 | set $r1, 0x55 | e155 |
| 1110100100010010 | sset $r1, 0x12 | e912 |
| 1110100100010000 | sset $r1, 0x10 | e910 |
| 1110100100100000 | sset $r1, 0x20 | e920 |
| 1110100100000000 | sset $r1, 0x00 | e900 |
| 1110011000100010 | set $r6, 0x22 | e622 |
| 1110111000000110 | sset $r6, 0x06 | ee06 |
| 1110111000000010 | sset $r6, 0x02 | ee02 |
| 1110111000010001 | sset $r6, 0x11 | ee11 |
| 001111000111 | eqv $r3, $r6, $r1 | 3c7 |
| 100000000001 | sub $r0, $r0, $r0 | 801 |
| 1000000100000000 | lw $r1, 0($r0) | 8100 |
| 1000001000000001 | lw $r2, 1($r0) | 8201 |
| 1000001100000010 | lw $r3, 2($r0) | 8302 |
| 0100010010001010 | addi $r4, $r4, 10 | 448a |
| 110010010001 | sub $r4, $r4, $r4 | c91 |
| 110001010000 | add $r4, $r2, $r4 | c50 |
| 110101001110 | slt $r5, $r2, $r3 | d4e |
| 0111010100000011 | beq $r5, $r0, 2 | 7503 |
| 101000101000 | add $r2, $r1, r2 | a28 |
| 0111000000011100 | beq $r0, $r0, -5 | 701c |
| 1000110000000000 | sw $r4, 0($r0) | 8c00 |
| 1111100000000110 | jal func | f806 |
| 0110111011001000 | ror $r6, $r6, 8 | 6ec8 |
| 1000010000000001 | lw $r4, 1($r0) | 8401 |
| 1000010100000010 | lw $r5, 2($r0) | 8502 |
| 111011010100 | add $r6, $r6, $r5 | ed4 |
| 011100000001 | beq $r0, $r0, -1 | 701 |
| 1111 | sub $r0, $r0, $r0 | f |
| 100000000001 | lw $r1, 0($r0) | 801 |
| 1000000100000000 | lw $r2, 0($r1) | 8100 |
| 1000001000100000 | lw $r3, 1($r1) | 8220 |
| 1000001100100001 | and $r4, $r2, $r3 | 8321 |
| 010001001100 | or $r5, $r2, $r3 | 44c |
| 010101001101 | sw $r4, 1($r0) | 54d |
| 1000110000000001 | sw $r5, 2($r0) | 8c01 |
| 1000110100000010 | jr $r7 | 8d02 |

Ro= 0x00000000/0

R1= 0x00000000/0

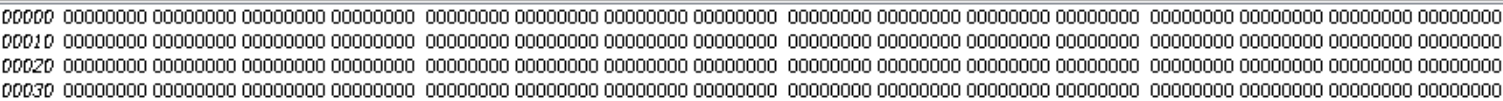
R2= 0x00000000/0

R3= 0x00000000/0

R4= 0x00000000/0

R5= 0x00000000/0

R6= 0x00000011/17

R7= 0x00000000/0

|  |  |
| --- | --- |
| Memory address | Value |
| 00000 | 0x00000000/0 |