**Question 1:Why is the instruction and data memory separated in MIPS SS v2 CPU?**

It is because we can only read the instruction memory, but for data memory, we can read and write data.

**Explain the advantage and disadvantage of separate instruction and data memory**

**Advantages:**

1. When we separate DM and IM into two different parts, we can read them at the same time to increase efficiency
2. We can save power if we separate DM and IM

**Disadvantages:**

(1)Separate memory into two different parts will cause the datapath more complex. And can end up in more errors when we build them

(2)When we separate data memory and instruction memory, some instructions will have to allocate some bits to control the data memory,causing the control unit do more works. And may generate more heats, which is not good to circuits.

**Question 2: Understanding Simple CPU (mips ss v2) is important to complete the rest of the midterm project. In this part you will write an assembly programs in binary and execute it in mips ss v2 CPU running in DE0-Nano. Triangular Number Generator: (20) 1 Write a program that generates Triangular numbers up to n=20 and count backwards to 0. 2 Your program should display 8 least significant bits of each number you generate in the above part through the LEDs. 3 While counting back you should blink the LEDs whenever the count reaches 0 or 20**

**Solution :**

R1 = i , R2 = sum , R3 = 21, R4 = 1

add R1 R1 R4 \*/ i = i+1

add R2 R1 R1 \*/ sum = sum+i

out R2 \*/print R2 on LED

beq R1 R3 002 \*/ i = 21 branch

beq R2 R2 -4 \*/ i not equal 21 loop

out R0

sub R2 R2 R3 \*/ sum = sum-21

add R2 R2 R4 \*/ sum = sum-20

out R2

sub R1 R1 R4 \*/ I = i-1

sub R2 R1 R2 \*/ sum=sum-i

out R2

beq R1 R0 -12 \*/ i=0 branch

beq R2 R2 -4 \*/ i not equal to 0 loop

**OP code:**

-- NYU CS6133 Computer Architecture

-- Instructor: Vikram Padman

-- Student: Huai Lin

--Purpose of this program: Write a program that generates Triangular numbers up to

n=20 and count backwards to 0.

2 Display 8 least significant bits of each number you generate in the above part through the LEDs.

3 While counting back you should blink the LEDs whenever the count reaches 0 or 20

-- Explanation: in c code type the program is as follow:

--int i=0 ,sum=0;

--for(i=0; i<=20; i++)

--sum=sum+i;

--end module

-- R-Type: <6-bit Opcode>,<5-bit rs>,<5-bit rt>,<5-bit rd>,<5-bit shamt>,<6-bit funct>

-- bits (31-26) (25-21) (20-16) (15-11) (10-6) (5-0)

-- I-Type: <6-bit Opcode>,<5-bit rs>,<5-bit rt>,<16-bit Address>

-- bits (31-26) (25-21) (20-16) (15-0)

--File format:

-- Hex Address 3 hex nibbles (12 bits) : bit31 ...... bit0;

WIDTH=32;

DEPTH=1024;

ADDRESS\_RADIX=HEX;

DATA\_RADIX=BIN;

CONTENT BEGIN

--Hex Address : bit31..........................bit0;

000 : 10001100000000110000000000000100;

-- |\_\_\_\_||\_\_\_||\_\_\_||\_\_\_\_\_\_\_\_\_\_\_\_\_\_|

-- | | | |

-- lw, rs=0, rt=3, offset=4

-- This is the first instruction that get’s executed

-- in mips\_ss CPU in DE0-Nano.

-- This is a lw instructioni. It loads r3 with data from

-- data memory location 4. Data memory location 4 is

-- preloaded with 21, see DRAM.mif.

001 : 10001100000001000000000000000000;

-- |\_\_\_\_||\_\_\_||\_\_\_||\_\_\_\_\_\_\_\_\_\_\_\_\_\_|

-- | | | |

-- lw, rs=0, rt=4, offset=8

-- This is the first instruction that get’s executed

-- in mips\_ss CPU in DE0-Nano.

-- This is a lw instructioni. It loads r4 with data from

-- data memory location 8. Data memory location 8 is

-- preloaded with 1, see DRAM.mif.

002 : 10110000000000000000000000000000;

-- |\_\_\_\_||\_\_\_||\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|

-- | | |

-- out, rs=0, XXXXXXXXXXXXXXXXX

003 : 00000000001001000000100000100000;

-- |\_\_\_\_||\_\_\_||\_\_\_||\_\_\_||\_\_\_||\_\_\_\_|

-- | | | | | |

-- R-type,rs=1,rt=4,rd=1,---,f=add

-- add instructions (r-type, opcode=0, funct=100000)

-- add => rd = rs + rt

-- Therefore => r1 = r1 + r4

--So, right now i = i+1

004 : 00000000001000100001000000100000;

-- |\_\_\_\_||\_\_\_||\_\_\_||\_\_\_||\_\_\_||\_\_\_\_|

-- | | | | | |

-- R-type,rs=1,rt=2,rd=2,---,f=add

-- add instructions (r-type, opcode=0, funct=100000)

-- add => rd = rs + rt

-- Therefore => r2 = r1 + r2

--So, right now sum = sum+i

005 : 10110000010000000000000000000000;

-- |\_\_\_\_||\_\_\_||\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|

-- | | |

-- out, rs=2, XXXXXXXXXXXXXXXXXXX

006 : 00010000001000110000000000000011;

-- |\_\_\_\_||\_\_\_||\_\_\_||\_\_\_\_\_\_\_\_\_\_\_\_\_\_|

-- | | | |

-- beq, rs=1, rt=3, offset=3

-- This is a beq, branch if equal instruction

007 : 000100000100001011111111111111011;

-- |\_\_\_\_||\_\_\_||\_\_\_||\_\_\_\_\_\_\_\_\_\_\_\_\_\_|

-- | | | |

-- beq, rs=2, rt=2, offset=-4

-- This is a beq, branch if equal instruction

008 : 10110000000000000000000000000000;

-- |\_\_\_\_||\_\_\_||\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|

-- | | |

-- out, rs=0, XXXXXXXXXXXXXXXXX

009 : 00000000010000110001000000100010;

-- |\_\_\_\_||\_\_\_||\_\_\_||\_\_\_||\_\_\_||\_\_\_\_|

-- | | | | | |

-- R-type,rs=2,rt=3,rd=2,---,f=sub

-- sub instructions (r-type, opcode=0, funct=100010)

-- sub => rd = rs - rt

--So, right now Sum=Sum-21

00A : 00000000010001000001000000100000;

-- |\_\_\_\_||\_\_\_||\_\_\_||\_\_\_||\_\_\_||\_\_\_\_|

-- | | | | | |

-- R-type,rs=2,rt=4,rd=2,---,f=add

-- add instructions (r-type, opcode=0, funct=100000)

-- add => rd = rs + rt

-- Therefore => r2 = r4 + r2

--So, right now Sum=Sum-20

011 : 10110000010000000000000000000000;

-- |\_\_\_\_||\_\_\_||\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|

-- | | |

-- out, rs=2, XXXXXXXXXXXXXXXXXXX

012 : 00000000001001000000100000100010;

-- |\_\_\_\_||\_\_\_||\_\_\_||\_\_\_||\_\_\_||\_\_\_\_|

-- | | | | | |

-- R-type,rs=1,rt=4,rd=1,---,f=sub

-- sub instructions (r-type, opcode=0, funct=100010)

-- sub => rd = rs - rt

--So, right now i = i-1

013 : 00000000001000100001000000100010;

-- |\_\_\_\_||\_\_\_||\_\_\_||\_\_\_||\_\_\_||\_\_\_\_|

-- | | | | | |

-- R-type,rs=1,rt=2,rd=2,---,f=sub

-- sub instructions (r-type, opcode=0, funct=100010)

-- sub => rd = rs - rt

--So, right now sum = sum-i

014 : 10110000010000000000000000000000;

-- |\_\_\_\_||\_\_\_||\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|

-- | | |

-- out, rs=2, XXXXXXXXXXXXXXXXXXX

015 : 00010000001000001111111111110100;

-- |\_\_\_\_||\_\_\_||\_\_\_||\_\_\_\_\_\_\_\_\_\_\_\_\_\_|

-- | | | |

-- beq, rs=1, rt=0, offset=-12

-- This is a beq, branch if equal instruction

016 : 000100000100001011111111111111011;

-- |\_\_\_\_||\_\_\_||\_\_\_||\_\_\_\_\_\_\_\_\_\_\_\_\_\_|

-- | | | |

-- beq, rs=2, rt=2, offset=-4

-- This is a beq, branch if equal instruction

**As implemented, mips ss v2 only supports register and immediate addressing modes. For this part add the following addressing modes to mips 22 v2:**

**1 Displacement (30)**

**2 Register indirect (40)**

**Solution:**

I will first demonstrate Register indirect. The format for register indirect is “add R4 (R1). Which means Reg[R4] <－ Reg[4]+Mem[Reg[R1]]

* **Register Indirect**

**Instruction formats:**

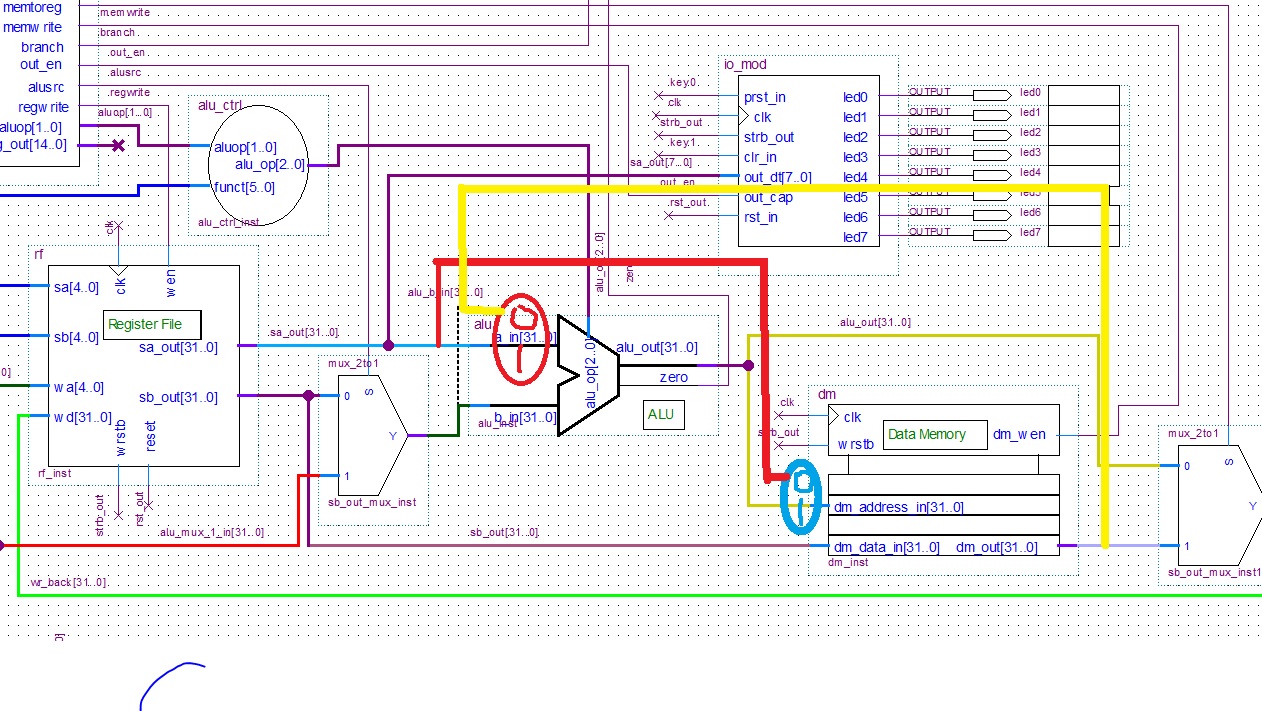
**op rs rt rd function**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |

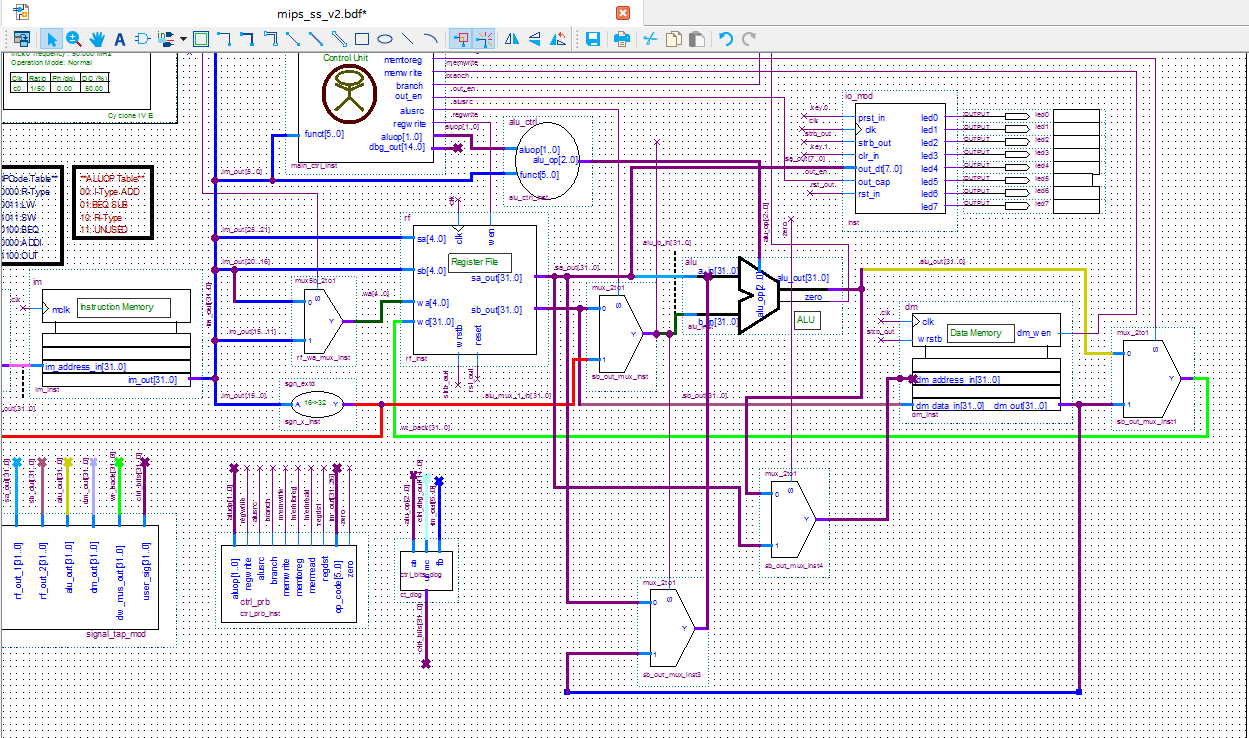
**6bits 5bits 5bits 5bits 11bits**

In my design, op code for indirect is 111111, and function is all 0s.

**Implementation details:**

****

***Figure 1. Indirect datapath***

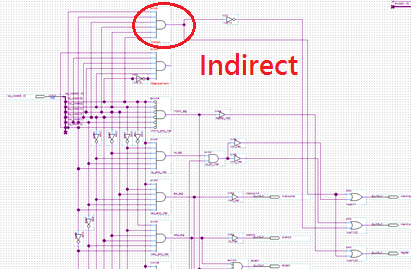


***Figure 2. Modification in quartus***

**Step1**. For step 1, we have to let machine know, data for register one is for address in Data memory. So, I add a red line. And we need a Mux to determine which address we should access（Whether from ALU or register one）. That is why I put a blue Mux.

**Step2**. After we read data out from data memory. We send data back to ALU. That is why I draw the yellow line.

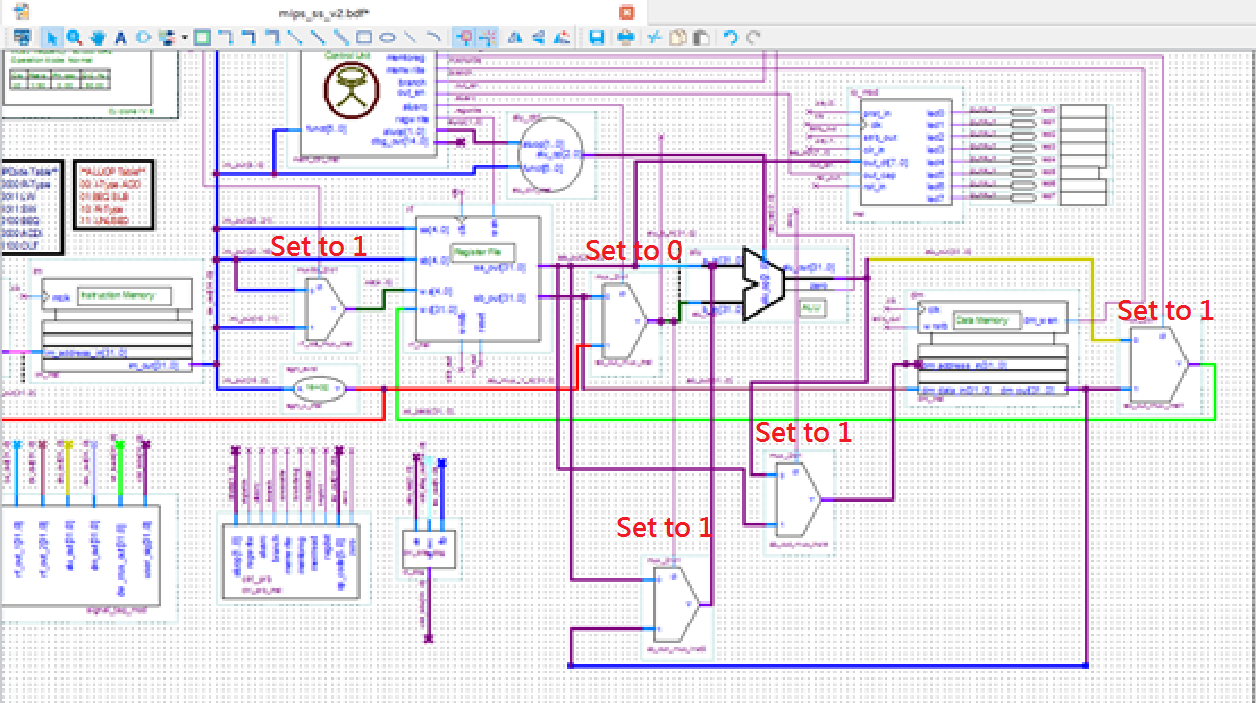
**Step3**. Finally, we need a mux to distinguish. Whether the operand for ALU is from register one or data memory we just read



***Figure 3. Modification in Control Unit***

**Explanation for modification in control unit:**

In my design, the op code for indirect is 111111. So, first I connect every line to an and gate without using a not gate. And for the complete datapath for indirect. I have to using four Mux. The two Mux which I add must be 1 and I connect them to “mem read”. The other two Mux are connect to “mem to reg” and “red dst”. “mem to reg” must be set to 0. “red dst” must be set to 1

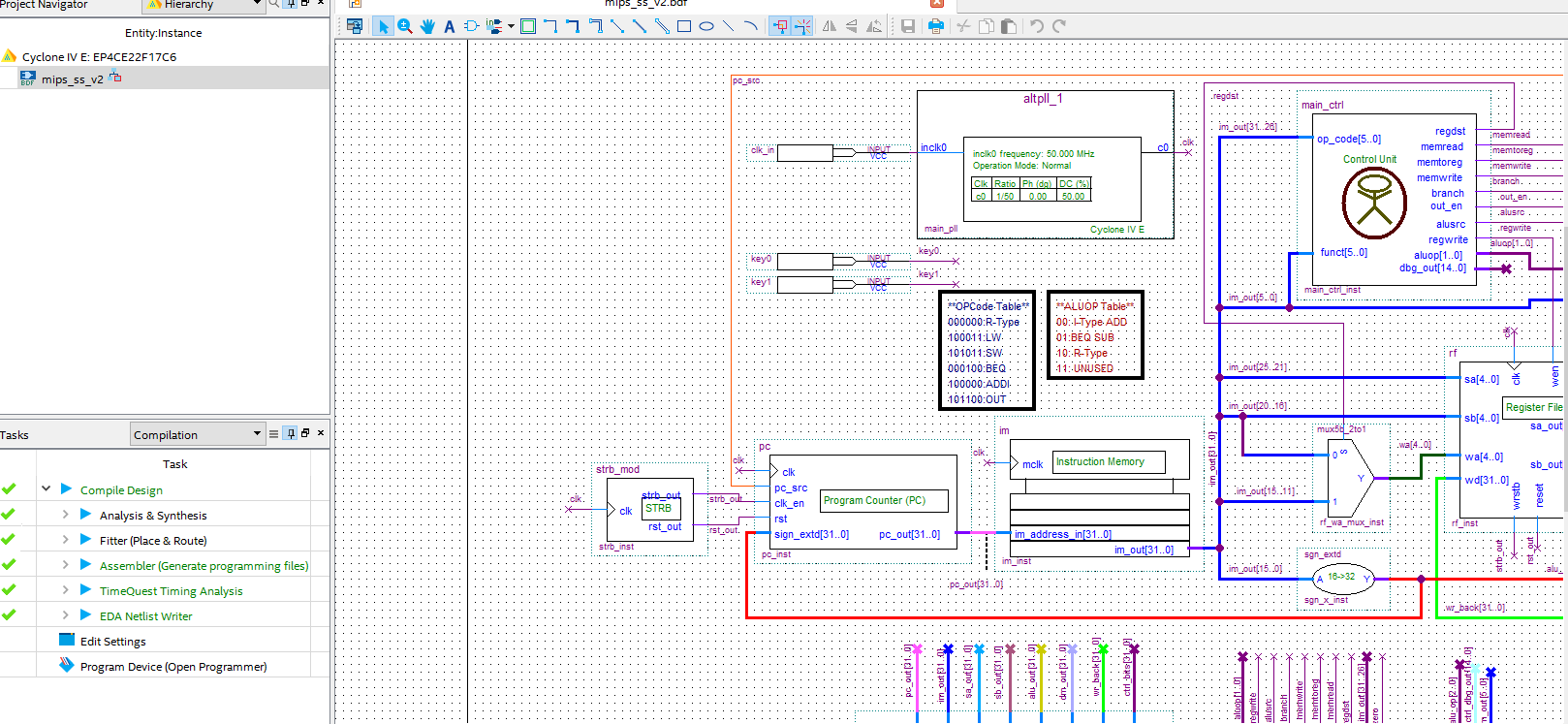
****

***Figure 4. Control signal to every Mux***

**Detailed description of any new modules:**

I have add two new mux to perform “Register Indirect”

* The first one is add in front of the Data memory (Which is draw in blue in figure one). Its purpose is to distinguish whether the address for data memory is from ALU(Such as lw sw instructions) or from register one
* The second Mux is use to distinguish whether the first operand for ALU is from Register one (Such as add sub instructions) or from data memory (Such as Indirect and displacement instructions). This Mux is draw in red in figure one.



***Figure 5. My compile result in quartus(Indirect)***

* **Displacement**

**Solution:**

For displacement instruction. Add R4 100(R1) which means Reg[R4] <－ Reg[4]+Mem[100+Reg[R1]]. In this case 100 is binary number and is equal to 4 in decimal. So, we have to add a 4 into address of data memory

**Instruction formats:**

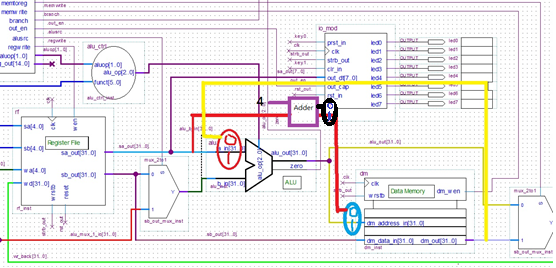
**op rs rt rd function**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |

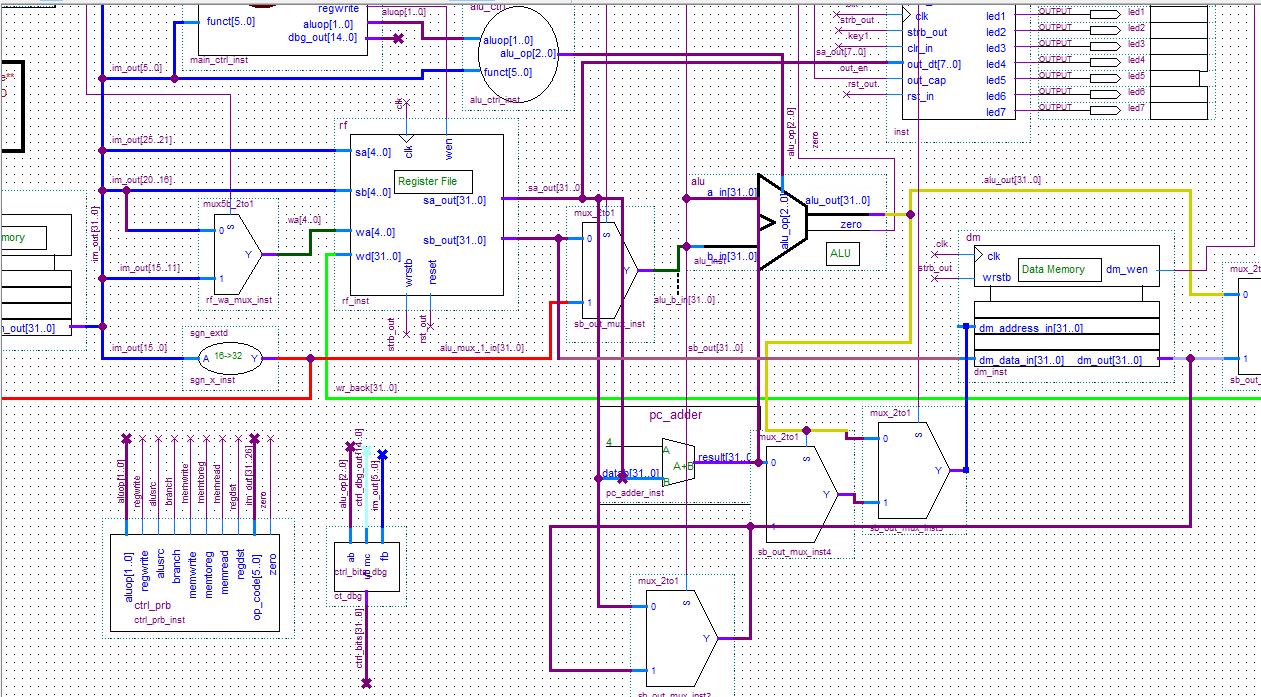
**6bits 5bits 5bits 5bits 11bits**

In my design, op code for indirect is 111110, and function is all 0s.

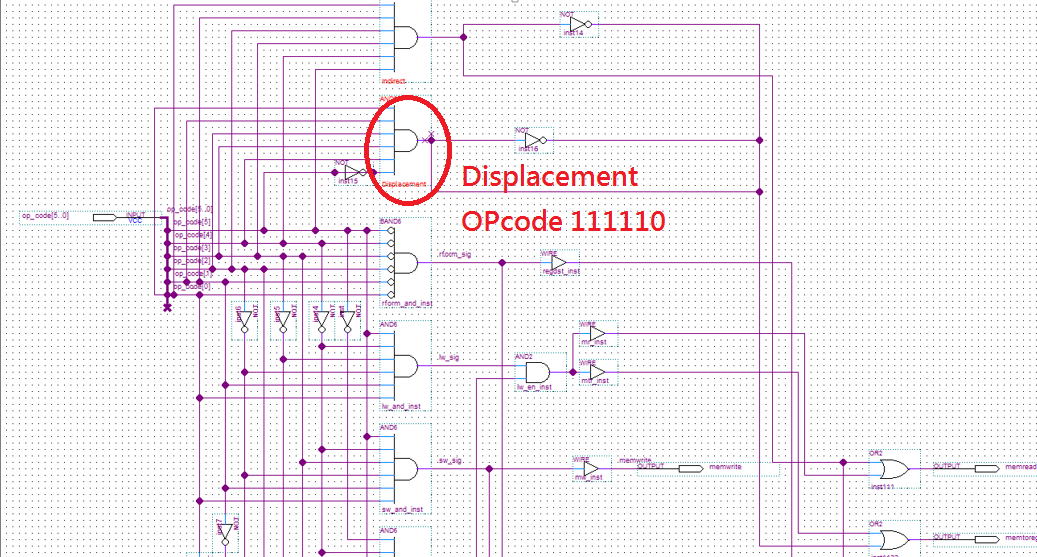
**Implementation details:**



***Figure 6. Displacement datapath***



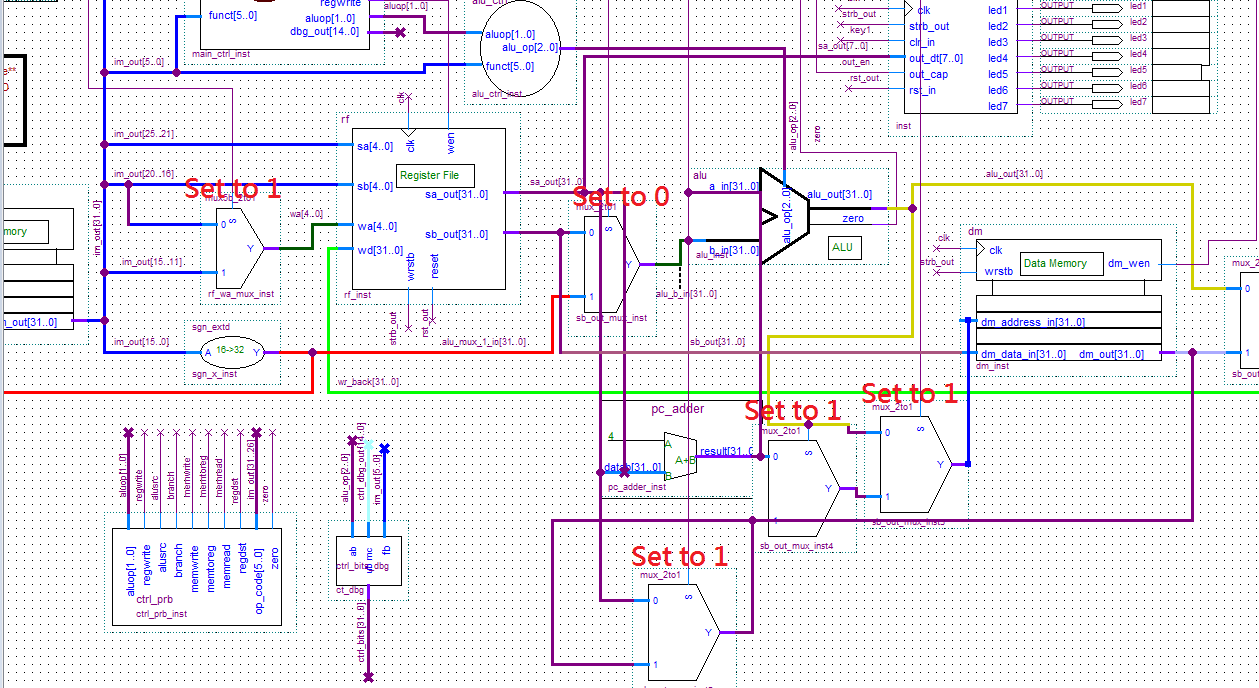
***Figure 7. Modification in quartus***

****

***Figure 8. Modification in Control Unit***

**Explanation for modification in control unit:**

In my design, the op code for indirect is 111110. So, first I connect every line to an and gate and a not gate to the last bit. And for the complete datapath for indirect. I have to using five Mux. The three Mux which I add must be 1 and I connect them to “mem read”. The other two Mux are connect to “mem to reg” and “red dst”. “mem to reg” must be set to 0. “red dst” must be set to 1



***Figure 9. Control signal to every Mux and an adder***

**Solution:**

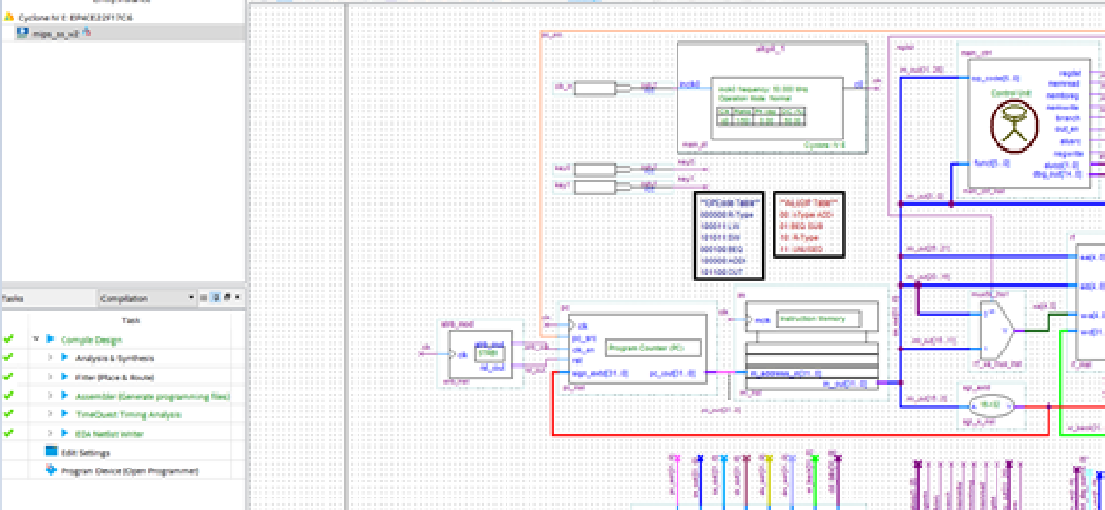
1. For displacement, we first have to read data from register one and use it as address for data memory. After that, we have to add that address with 4. By doing that, we can use an adder accomplish that task. (In the above figure, the adder is draw in purple)
2. Then, we need a Mux to decide whether the address have to add 4(Displacement instruction) or not to add 4(Indirect instruction). So, we have to add a Mux after the adder.
3. After that. We have to put a Mux in front of the data memory to determine whether the address for data memory is from ALU (Such as lw or sw instructions) or from register one(Such as Displacement instruction or Indirect instruction )

4. When we get data from data memory, we send it back to ALU to finish the job. But we need to add another Mux in front of ALU to decide whether the first operand for ALU is from register one (Such as add sub instructions) or from data memory (Such as Indirect and displacement instructions).

**Detailed description of any new modules:**

I have add three new mux and an adder to perform “Register Indirect”

* The adder is used to add four with the data in register to find the correct address in data memory.
* The second one is a Mux after the adder have added 4 to the address. We need a Mux to decide whether the address have to add 4(For example: Displacement instruction) or not (For example: Indirect instruction)
* The third one is a mux add in front of the Data memory (Which is draw in blue in figure one). Its purpose is to distinguish whether the address for data memory is from ALU(Such as lw sw instructions) or from register one
* The last Mux is use to distinguish whether the first operand for ALU is from Register one (Such as add sub instructions) or from data memory (Such as Indirect and displacement instructions). This Mux is draw in red in figure one.



***Figure 10. My compile result in quartus(Displacement)***