Lab 4 Report

Single Cycle Design

BAOTUNG C. TRAN

EEL4713C

Added Hardware :

Andi and Ori : For this instruction, I had to add a zero extender into my design. Which therefore required me to add a mux that controlled whether the immediate value from the instruction was zero extended or sign extended.

BNE and BEQ : These instructions required me quite a bit of extra logic. The problem was that before, we were using the zero signal from the alu to determine whether to branch or not. But the problem was, there was no way to differentiate between BNE and BEQ. Therefore I added two and gates and an or gate, and also added two more control signals called BEQ and BNE to the controller to differentiate when they would branch.

Jump and Jump and Link : For these instructions I added a mux to the address of the PC counter. I also added a control signal named JJAL, which controlled the mux. When it was set to 1, it would take the value of the jump address instead of just PC+4. For jump and link, I added an additional mux at the "Write Register" port of the register file, which was also controlled by JJAL. When JJAL was true, the mux would output 11111, which would select register (31) and then write the PC+4 to that address.

Jump Register : For this instruction I added a mux after the Jump and Link mux. The mux select was controlled by a signal JR. The JR signal came from the alucontrol. This is because JR is an r-type instruction meaning, the control signal could not come from the main controller. When JR is true, the mux would switch to inputting R[rs] into the PC.

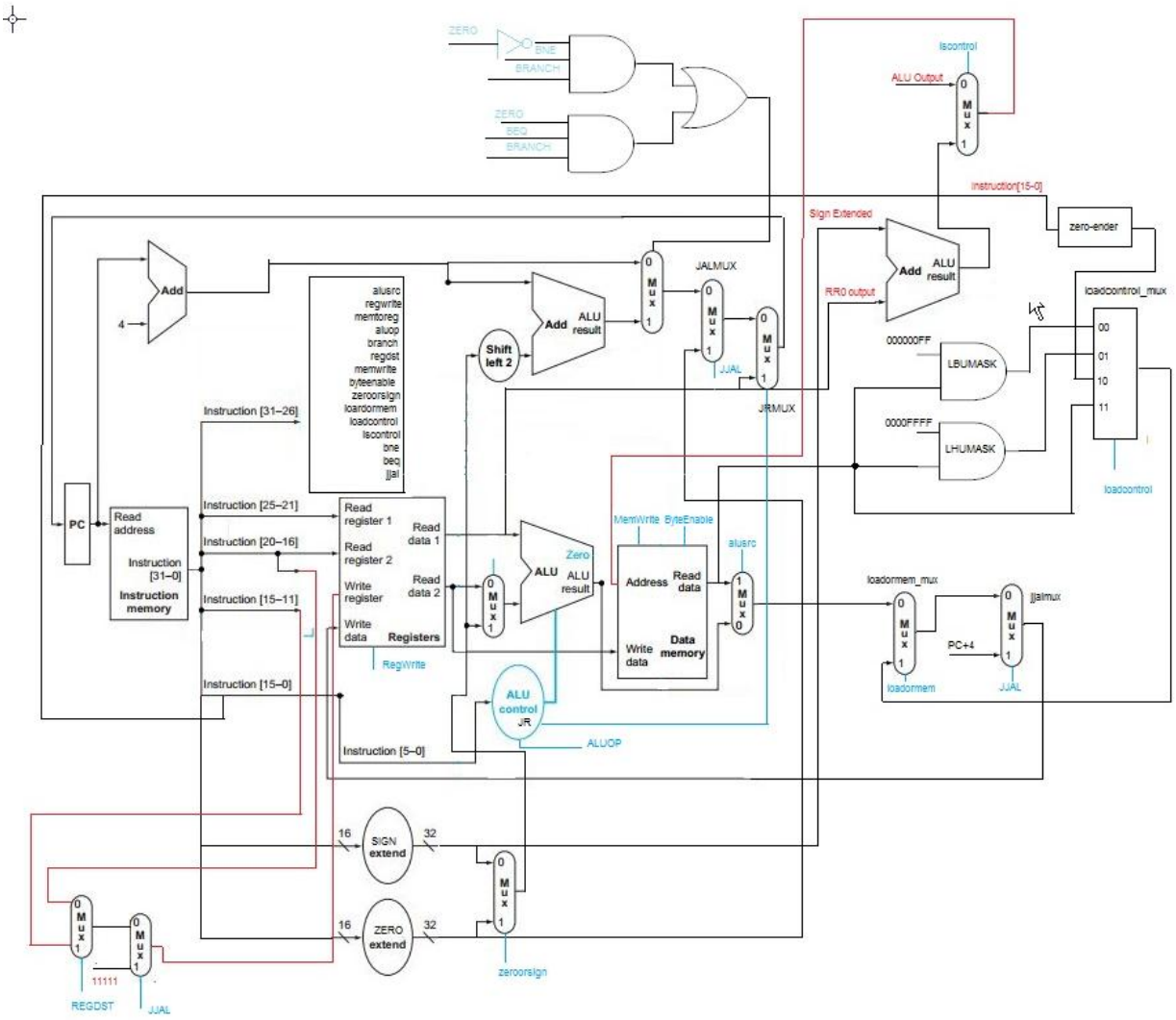
LBU, LHU, LUI, LW : These instructions were quite tricky. I firstly had to use and gates to mask M[R[rs]+SignExtImm for LBU and LHU. I then added a four-to-two mux to choose between LBU, LHU, LUI, and LW. This would go into a mux called loadormem_mux. This mux would then choose whether to output what came from the alusrc_mux(chooses between data memory or alu output) or one of the load instruction values to write to the register file.

SLL and SRL : Since these were R-Type instructions, I had to add a shdir signal to the alucontrol to tell them which way to shift.

SB, SW, SH : For these instructions, I added an adder that summed the values of the SignExtImm + R[rs]. This value would then go into a mux that took the inputs of SignExtImm+R[rs] and the ALU output. This mux was controlled by "lscontrol" which I added to the controller. By use the ByteEnable signal on the data memory, I could control to either write a byte, half word, or word.

**** My current bdf and vhd file increment the pc by just 1. During my own test file named "lab4baotung_test.mif", I incremented the pc counter by +4.

Below is the diagram of the previous hardware and the hardware that I had to include to create the entire single cycle processor. To start my Instruction Memory at x00400000, I manually my register file to reset to x00400000.



Control Signals Added:

Zeroorsign : Controls whether immediate value is zero-ext or sign-ext. I had to use this for andi, sltiu, addiu and ori, as these instructions were zero extended. If it is 1, it is zero extended, if its 0, it is sign extended

Loadormem : Chooses whether to output whats coming from the alusrc mux or, either half-word, byte, word, or immediate value that is zero extended(lui).

Loadcontrol : Select line of mux that chooses between half-word, byte, word, or upper immediate.

Lscontrol : Select line of mux that chooses either the output of the ALU or R[rs] + SignExtImm to go into address of data memory.

BNE and BEQ : Signals used to differentiate when its BEQ and BNE.

JJAL : Signal used to control two mux select lines when using the Jump or Jump and Link instruction.

JR: Placed in alucontrol since its an R-Type instruction. Used to control mux select line that goes into the "Write Register" input of register file. When it's set to 1, it will automatically set "Write Register" to 11111 or register(31).

Control Signals For New Instructions :

The complete values of the signals can be found in my controller and alu32control, which is located at the end of the report.
The following are ONLY the signals that were configured for the new instructions which required the new control signals.
These signals are directly from my controller:

```
when "001001" => -- addiu
zeroorsign <= '1';
aluop <= "001";
regwrite <= '1';
regdst <= '0';
ALUSRC <= '1';

when "001100" => -- andi
zeroorsign <= '1';
aluop <= "101";
regwrite <= '1';
regdst <= '0';
ALUSRC <= '1';

when "000100" => --BEQ
branch <= '1';
BEQ <= '1';
aluop <= "110"; -- subtraction

when "000101" => --BNE
branch <= '1';
BNE <= '1';
aluop <= "110";

when "001101" => -- ORI
zeroorsign <= '1';
aluop <= "010";
regwrite <= '1';
regdst <= '0'
```

```
ALUSRC <= '1';



when "001011" => -- sltiu
zeroorsign <= '1';
aluop <= "100"; -- tells alu control its an i instruction
regwrite <='1'; -- enables register file to write
regdst <='0'; -- writes to rt
alusrc <='1'; -- takes in immediate value taht is sign extended.
memtoreg <= '0'; --takes value, either 0 or 1 from aluout, and then will put it into rt.

when "100100" => -- lbu
 aluop <= "001"; -- immediate instruction
 memtoreg <= '1';
 alusrc <= '1';
 regwrite <='1';
 loadcontrol <= "00"; --chooses mask lbu
 loadormem <= '1';
 lscontrol <= '1';

when "100101" => --lhu
 aluop <= "001"; -- immediate instruction
 memtoreg <= '1';
 alusrc <= '1';
 regwrite <='1';
 loadcontrol <= "01"; -- chooses mask lhu
 loadormem <= '1';
 lscontrol <= '1';

when "100011" => --lw
 aluop <= "001"; --immediate instruction
 memtoreg <= '1';
 alusrc <= '1';
 regwrite <='1';
 loadcontrol <= "11"; --choses entire rs+signext
 loadormem <= '1';
 lscontrol <= '1';

when "101000" => --sb
  BYTEENABLE <= "0001";
 aluop <= "001"; --immediate instruction
 memtoreg <= '1';
 aluop <= "001";
 lscontrol <='1';
 memwrite <= '1';
```

```
 when "101001" => --sh
 BYTEENABLE <= "0011";
 aluop <= "001"; --immediate instruction
 memtoreg <= '1';
 aluop <= "001";
 lscontrol <='1';
 memwrite <= '1';

 when "101011" => --sW
 BYTEENABLE <= "1111";
 aluop <= "001"; --immediate instruction
 memtoreg <= '1';
 aluop <= "001";
 lscontrol <='1';
 memwrite <= '1';

when "001111" => --lui
 aluop <="001";
 regwrite <= '1';
 loadcontrol <="10";
 loadormem <= '1';
 lscontrol <='1';

when "001111" => --lui
 aluop <="001";
 regwrite <= '1';
 loadcontrol <="10";
 loadormem <= '1';
 lscontrol <='1';

when "000010" => --jump
 jjal <= '1';

when "000011" => --jal
 jjal <= '1';
 regwrite <= '1';
```

There were also some R-Type instructions which I had to change. Since the instruction is thrown to the alu32control, I had to change some instructions there as well. They will be listed below.
SRL, SLL, and JR are all R-Type instructions which required extra signals such as shdir for the shift direction and JR for the JR instruction.

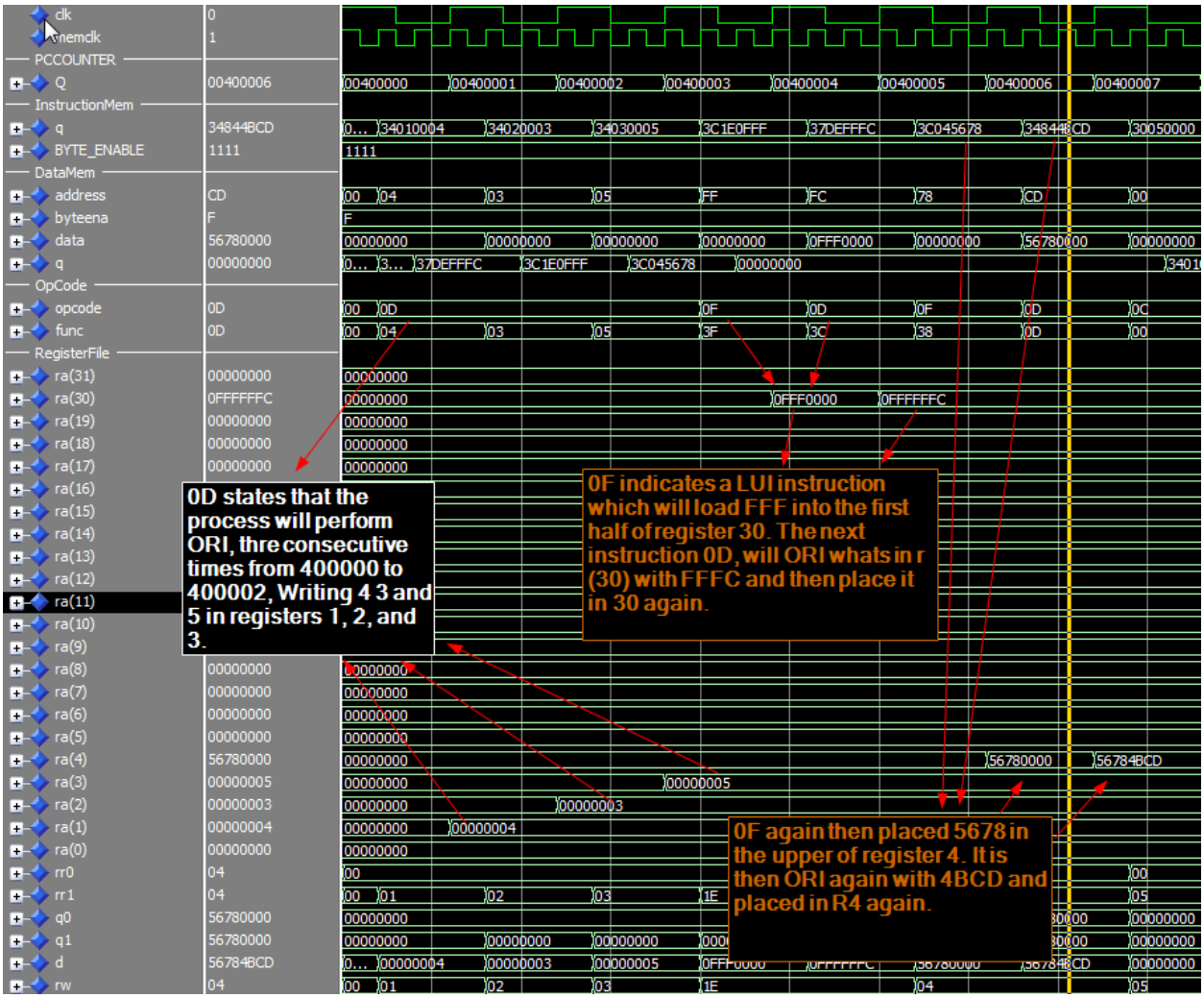
```
elsif (func = "000010") then – SRL
shdir <= '0';
control <= "0011"; -- srl control

elsif (func = "000000") then -- sll
control <= "0011"; --sll control
shdir <= '1';

elsif (func ="001000") then – jr
control <= "1110";
 jr <= '1';
```

Simulation Testing:

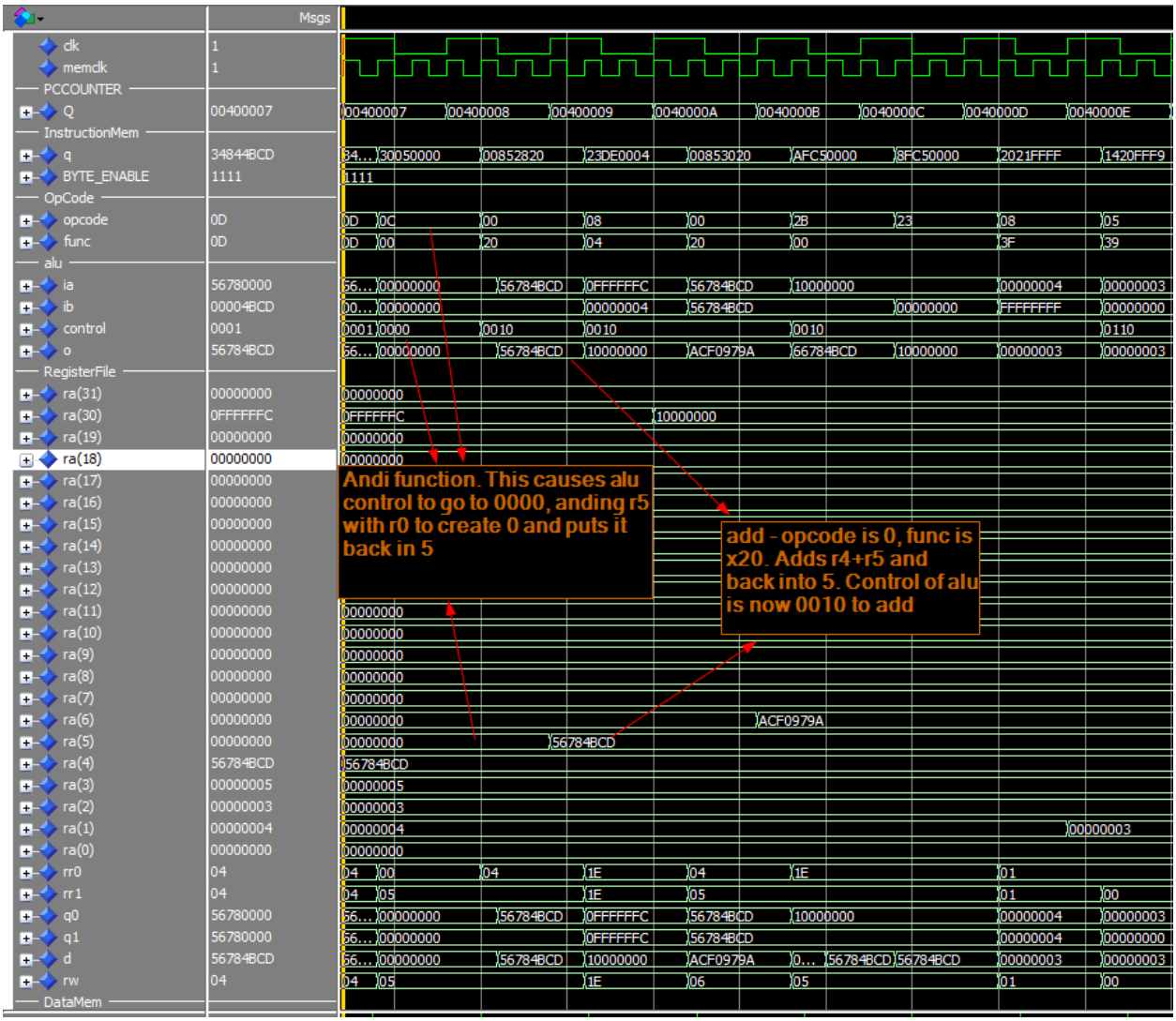
Disclaimer : I will only annotate each instruction ATLEAST once. If I have annotated it already in a previous slide, I might not annotate it again to save space and from you reading redundant information.

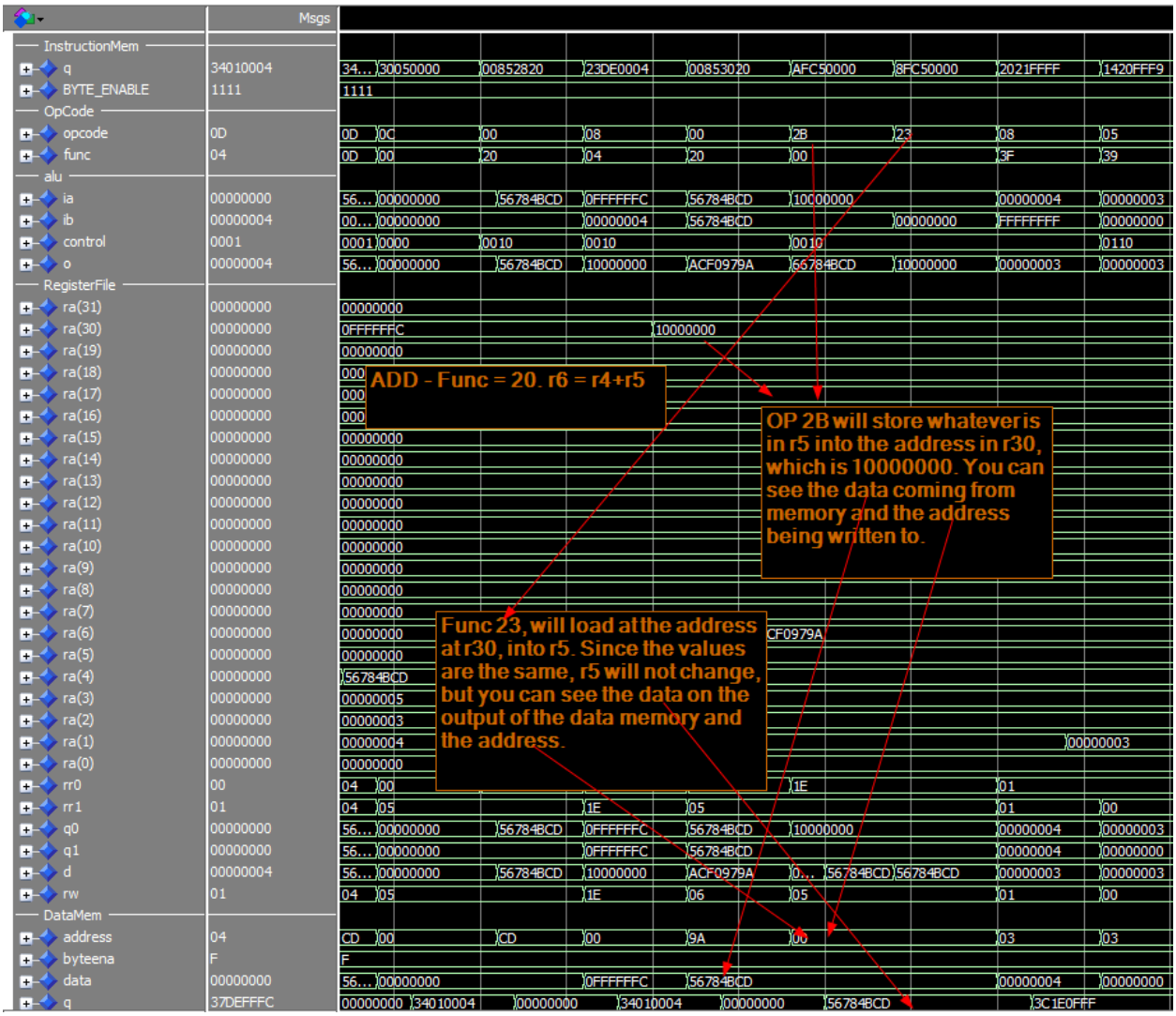
Lab4_demo.mif

```
000   :   04000134;          ori     1,0,$4
001   :   03000234;          ori     2,0,$3
002   :   05000334;          ori     3,0,$5
003   :   FF0F1E3C;          lui     30,$0FFF
004   :   FCFFDE37;          ori     30,30,$FFFC
005   :   7856043C;          lui     4,$5678
006   :   CD4B8434;          ori     4,4,$4BCD
```

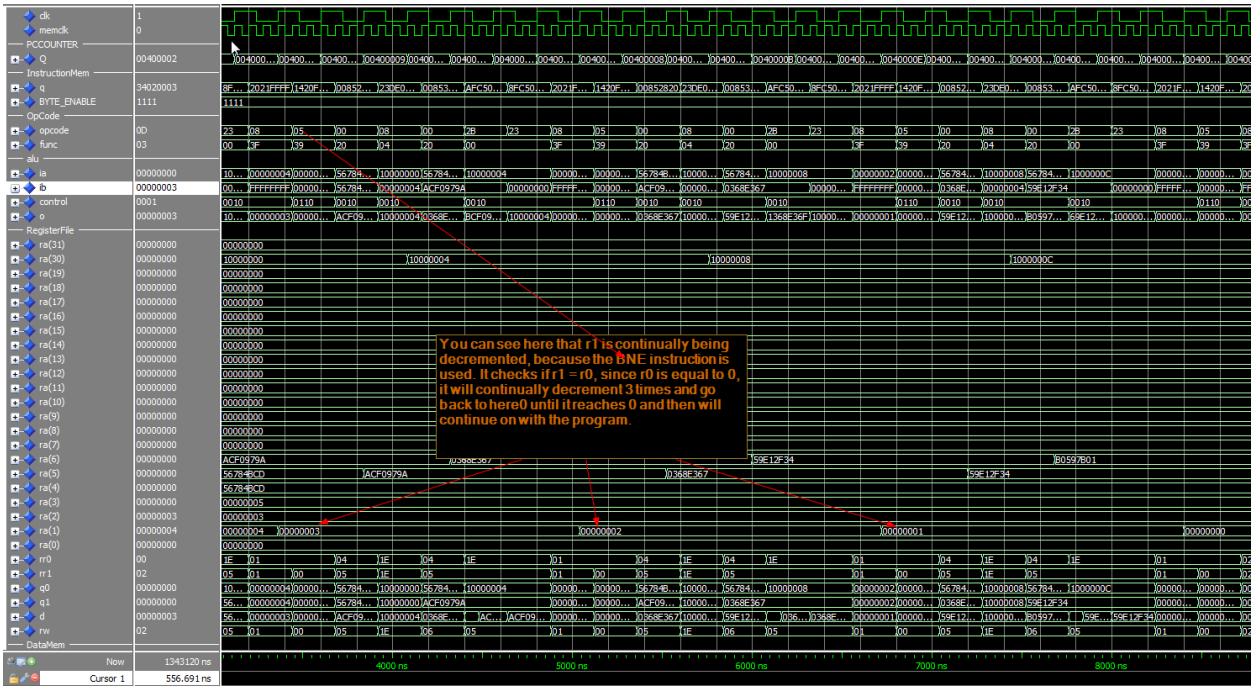


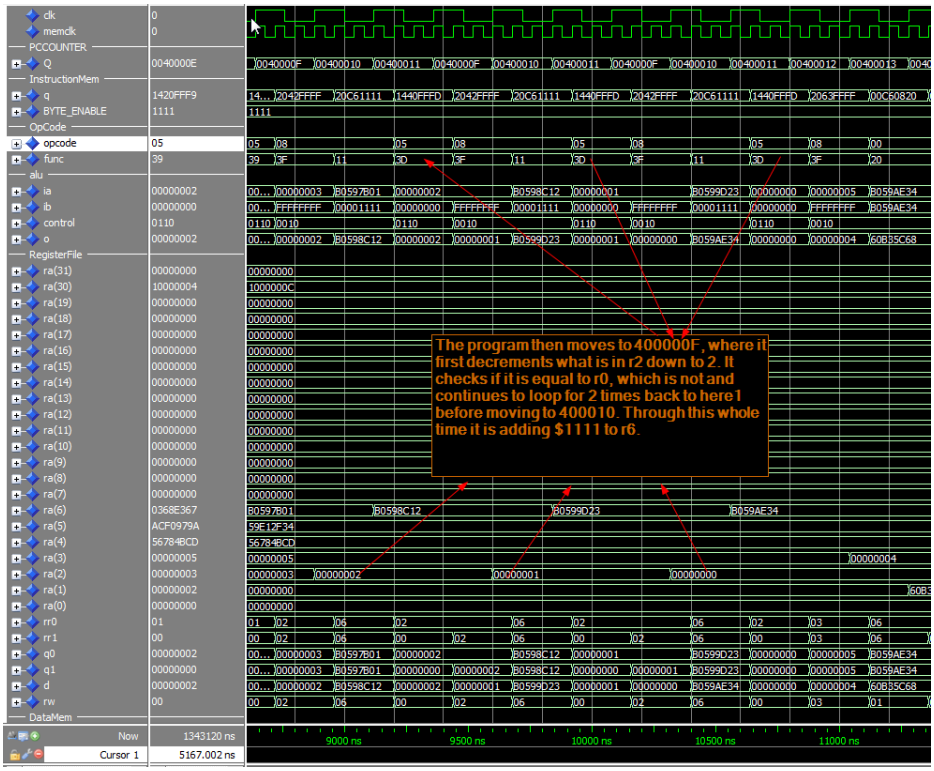
```
007  :   00000530;          andi    5,0,$0
008  :   20288500;          add     5,4,5          here0:
009  :   0400DE23;          addi    30,30,$4
00a  :   20308500;          add     6,4,5
00b  :   0000C5AF;          sw      5,30,$0
00c  :   0000C58F;          lw      5,30,$0
00d  :   FFFF2120;          addi    1,1,$FFFF
```

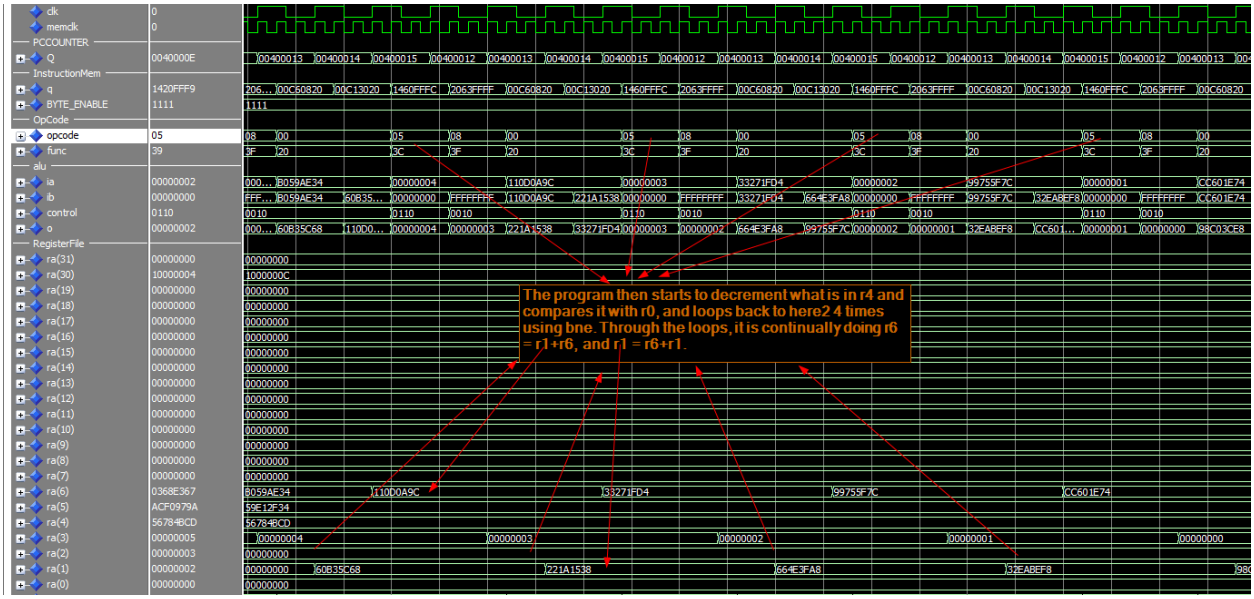



ADD - Func = 20. r6 = r4+r5
OP 2B will store whatever is in r5 into the address in r30, which is 10000000. You can see the data coming from memory and the address being written to.
Func 23, will load at the address at r30, into r5. Since the values are the same, r5 will not change, but you can see the data on the output of the data memory and the address.

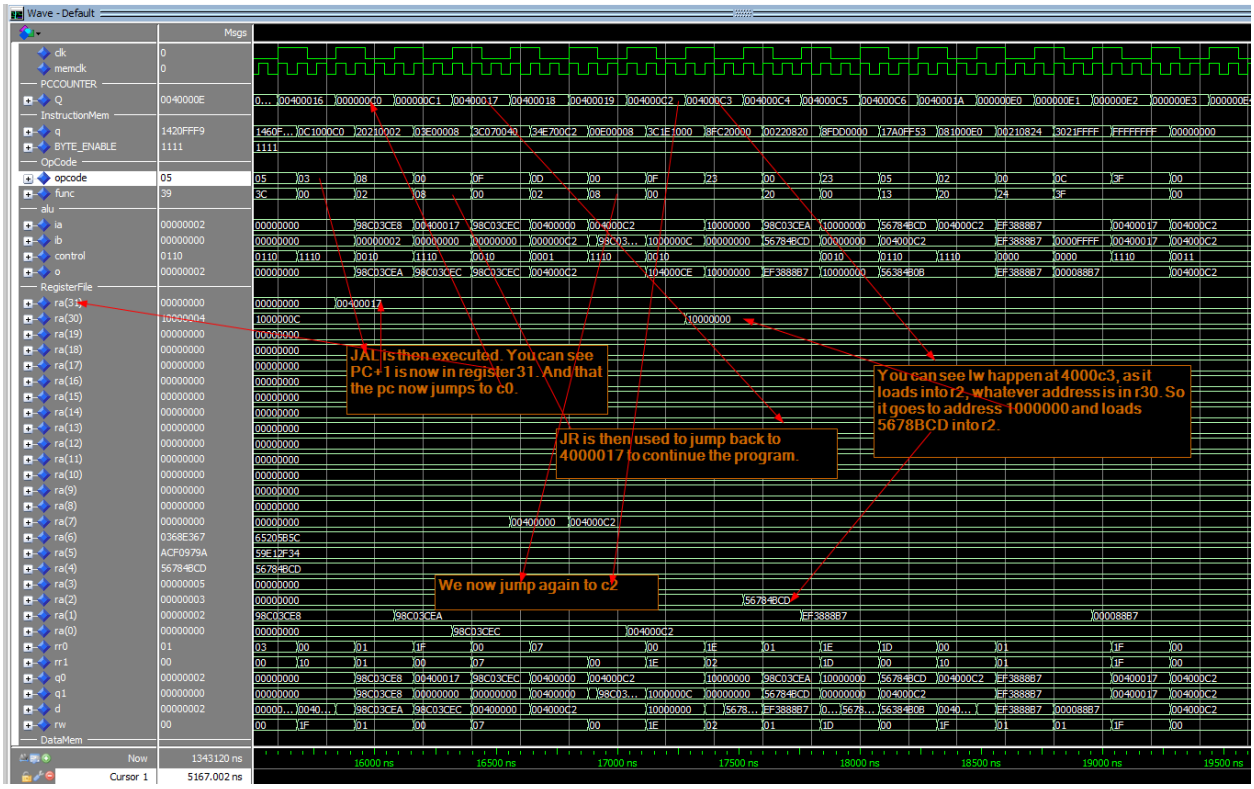
```
00e  :   F9FF2014;          bne     1,0,here0
00f  :   FFFF4220;          addi    2,2,$FFFF      here1:
010  :   1111C620;          addi    6,6,$1111
011  :   FDFF4014;          bne     2,0,here1
012  :   FFFF6320;          addi    3,3,$FFFF      here2
```



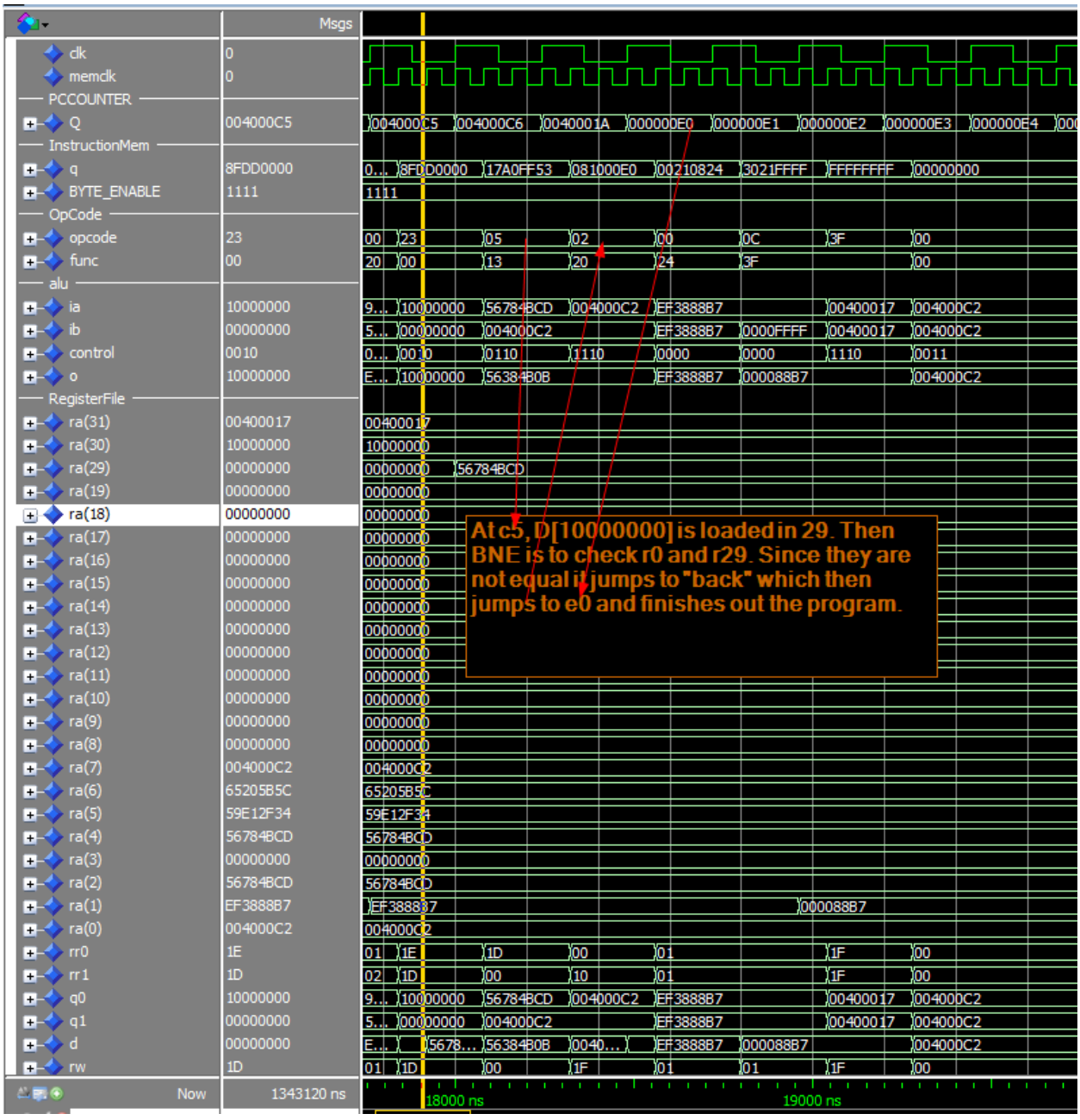



clk
memclk
PCCOUNTER
Q
InstructionMem
q
BYTE_ENABLE
OpCode
opcode
func
alu
ia
ib
control
o
RegisterFile
ra(31)
ra(30)
ra(19)
ra(18)
ra(17)
ra(16)
ra(15)
ra(14)
ra(13)
ra(12)
ra(11)
ra(10)
ra(9)
ra(8)
ra(7)
ra(6)
ra(5)
ra(4)
ra(3)
ra(2)
ra(1)
ra(0)
The program then starts to decrement what is in r4 and compares it with r0, and loops back to here2 4 times using bne. Through the loops, it is continually doing r6 = r1+r6, and r1 = r6+r1.

```
016  :   C000100C;            jal     $00400300
017  :   4000073C;            lui     7,$0040
018  :   0803E734;            ori     7,7,$308
019  :   0800E000;            jr      7
01a  :   E0001008;            j       $00400380       back:
[01b..0bf]  :   00000000;
0c0  :   02002120;            addi    1,1,$2
0c1  :   0800E003;            jr      31
0c2  :   00101E3C;            lui     30,$1000
0c3  :   0000C28F;            lw      2,30,$0
0c4  :   20082200;            add     1,1,2
```



```
0c5  :   0000DD8F;              lw      29,30,$0
0c6  :   53FFA017;              bne     29,0,back
[0c7..0df]  :   00000000;
0e0  :   24082100;              and     1,1,1
0e1  :   FFFF2130;              andi    1,1,$FFFF
0e2  :   FFFFFFFF;              GFO
[0e3..0ff]  :   00000000;
```

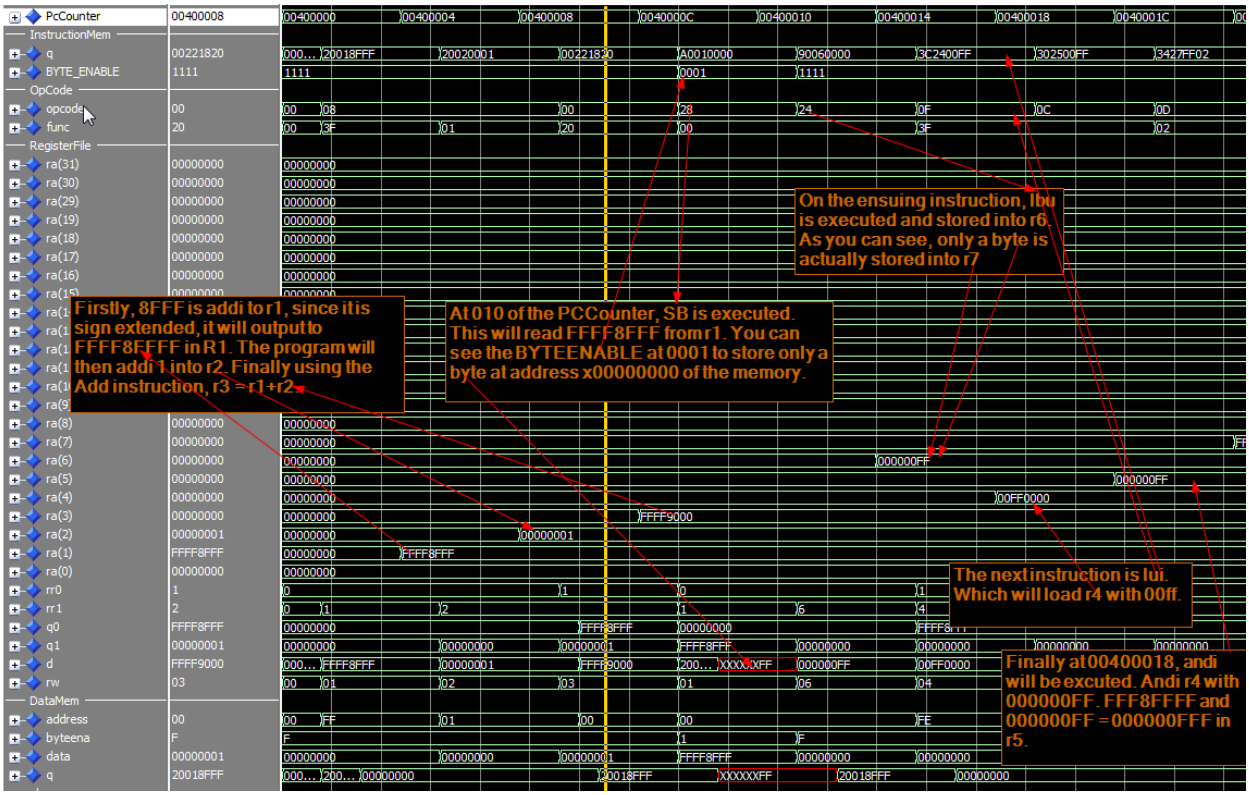


Lab4baotung_test.mif

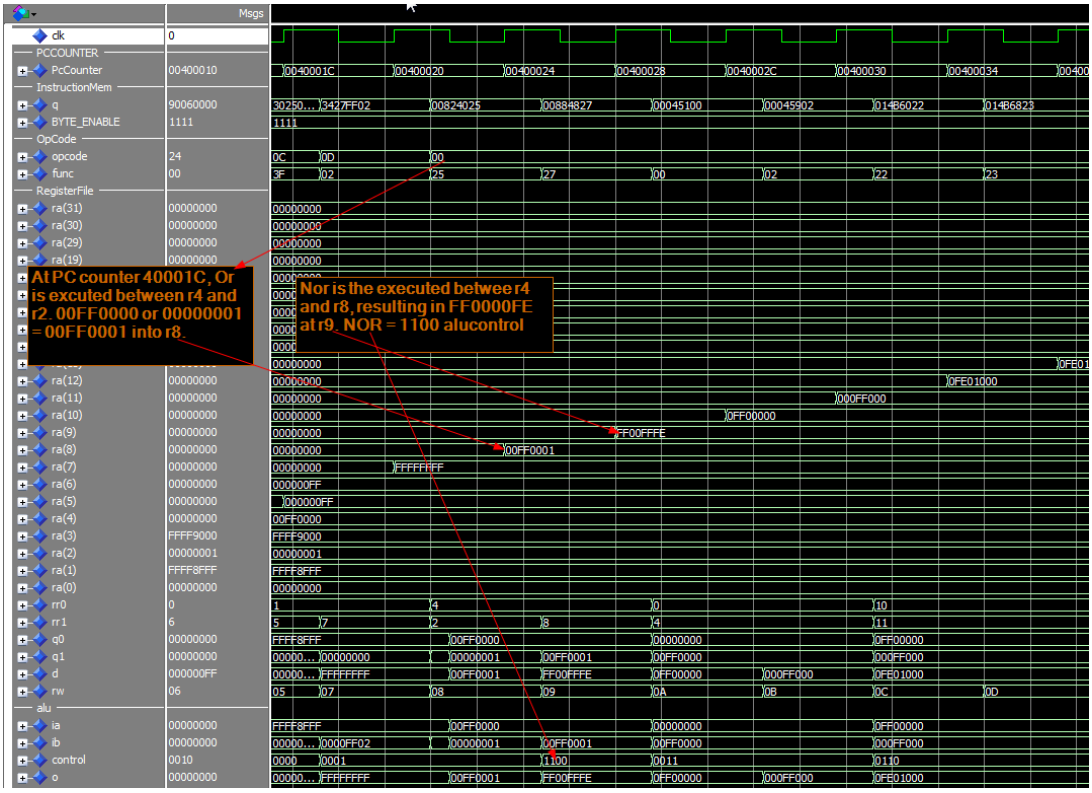
***** The annotated mif file will be included in the report**

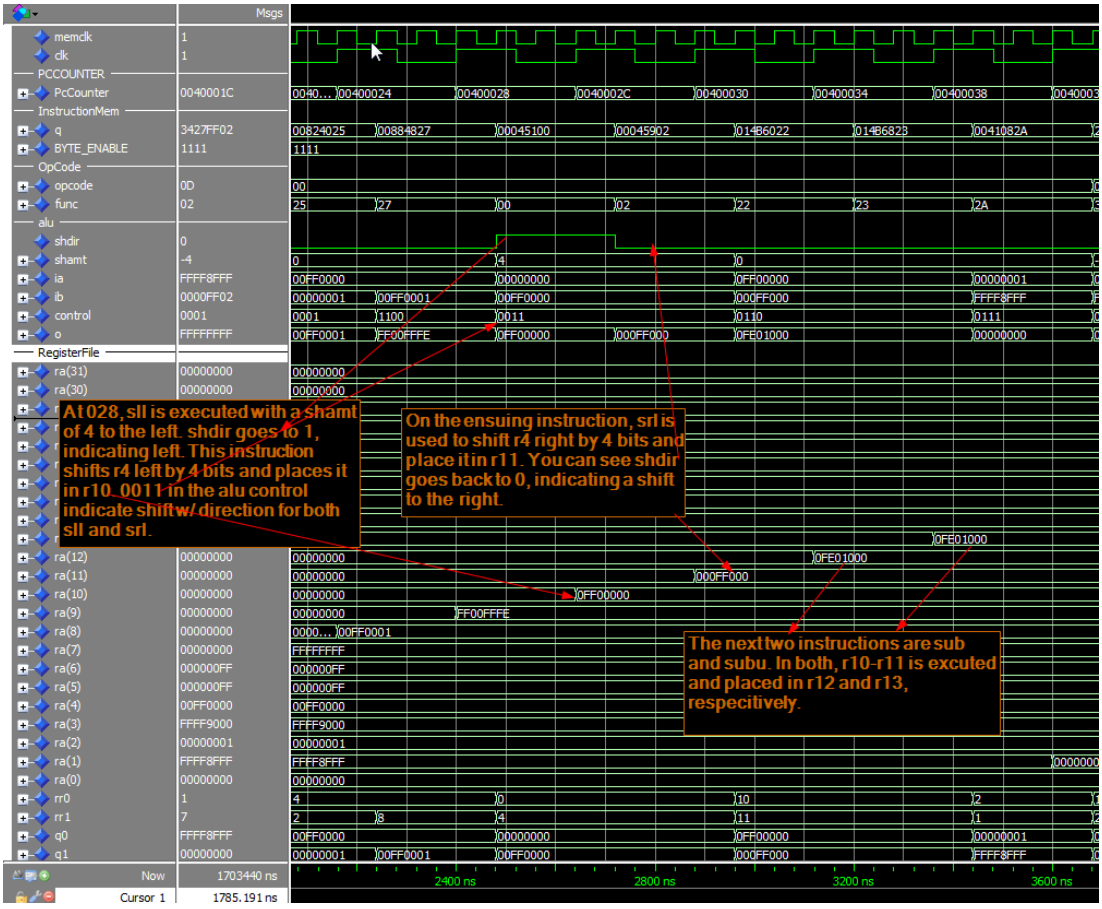
I created my own program that ran through every single besides for addiu. Addiu will be annotated when looking at the program flow of lab4_test.mif. Also, the pc is incremented by +4, and not +1.

000 : 20018fff;
004 : 20020001;
008 : 00221820;
00C : A0010000;
010 : 90060000;
014 : 3C2400FF;
018 : 302500FF;

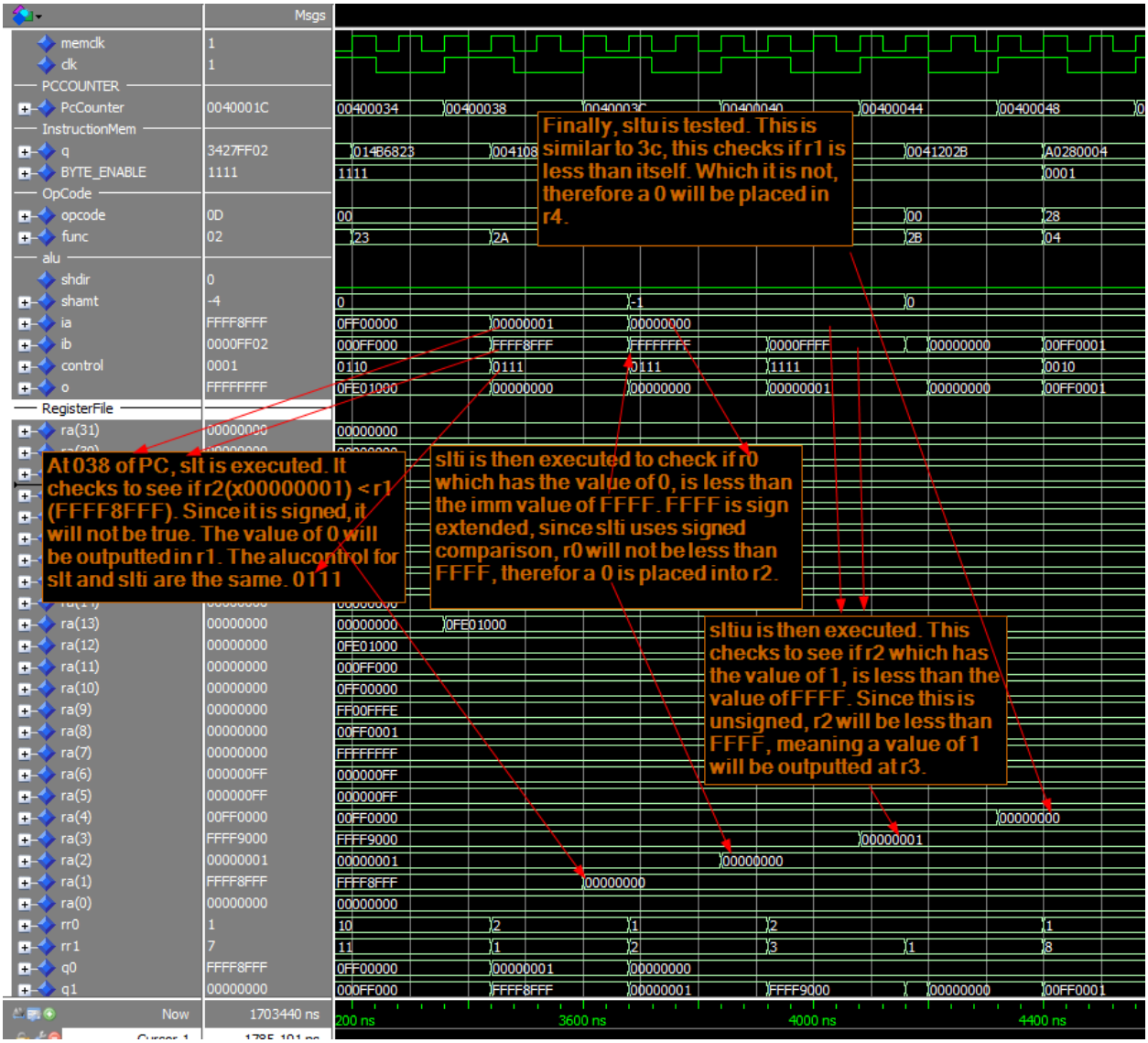


01C : 3427FF02;
020 : 00824025;
024 : 00884827;
028 : 00045100;
02C : 00045902;
030 : 014B6022;
034 : 014B6823;

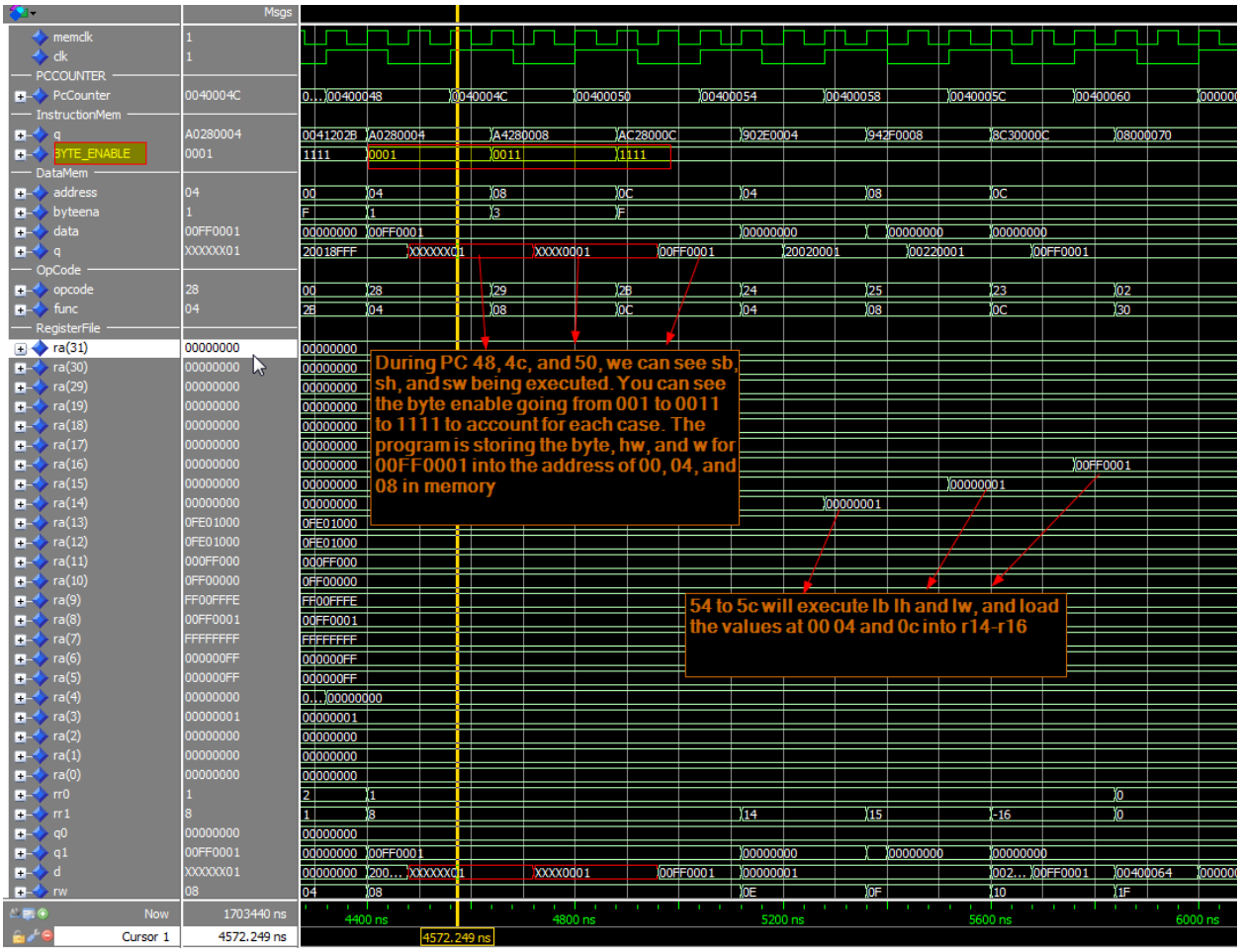

At PC counter 40001C, Or is excuted between r4 and r2. 00FF0000 or 00000001 = 00FF0001 into r8.
Nor is the executed betwee r4 and r8, resulting in FF0000FE at r9. NOR = 1100 alucontrol


At 028, sll is executed with a shamt of 4 to the left. shdir goes to 1, indicating left. This instruction shifts r4 left by 4 bits and places it in r10. 0011 in the alu control indicate shift w/ direction for both sll and srl.
On the ensuing instruction, srl is used to shift r4 right by 4 bits and place it in r11. You can see shdir goes back to 0, indicating a shift to the right.
The next two instructions are sub and subu. In both, r10-r11 is excuted and placed in r12 and r13, respecitively.

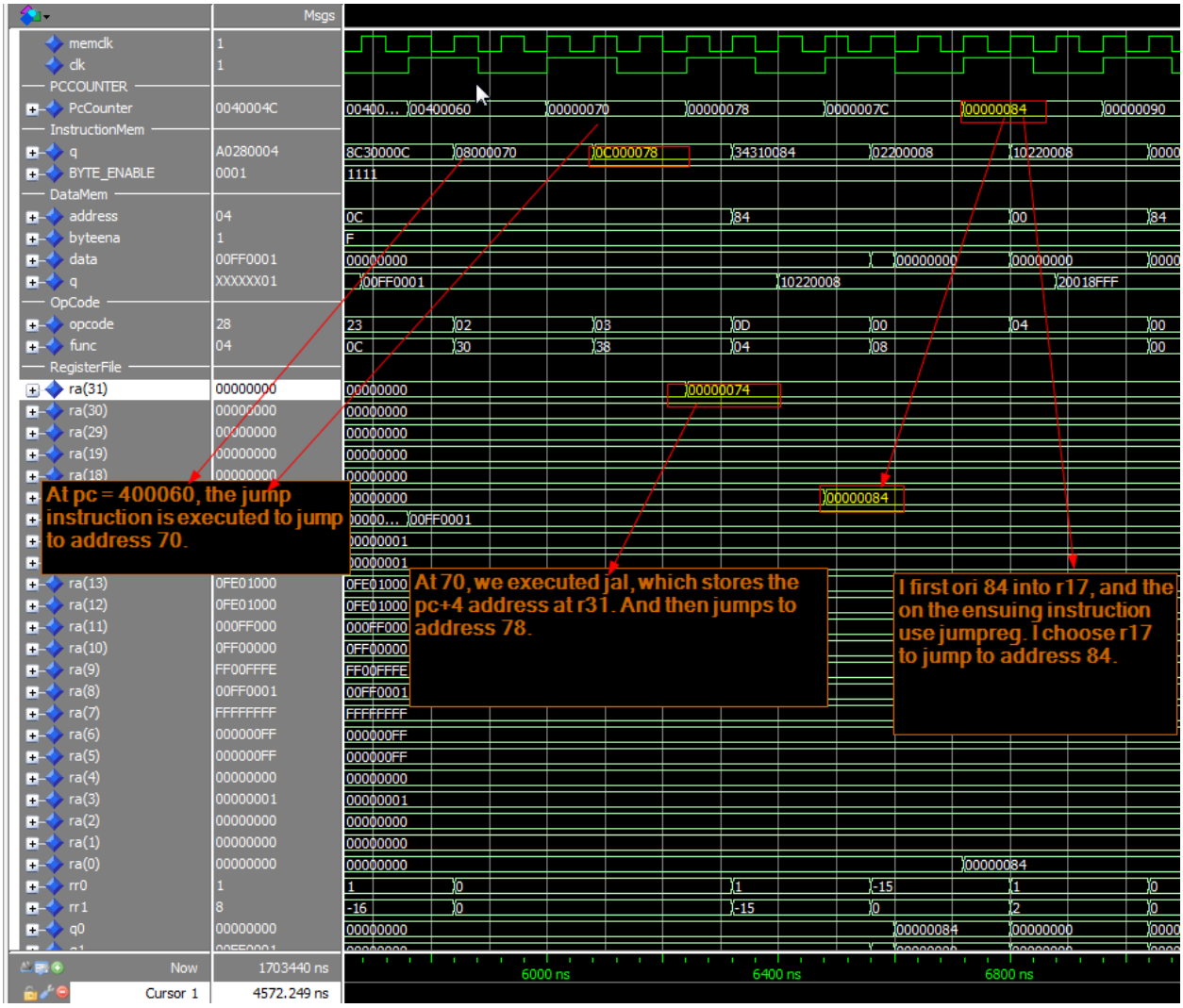
038 : 0041082A;
03C : 2822ffff;
040 : 2C43ffff;
044 : 0041202b;



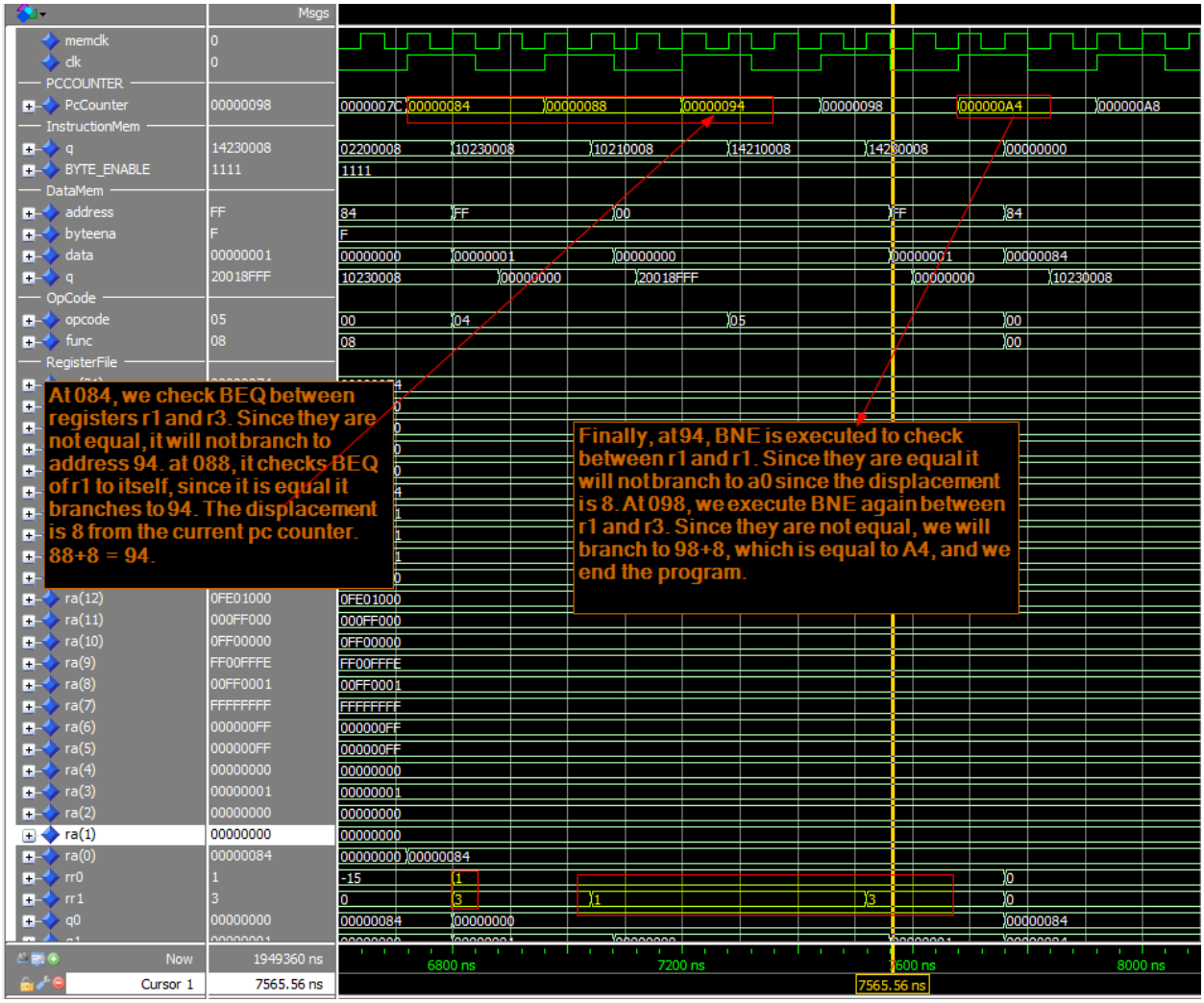
048 : A0280004;
04C : A4280008;
050 : AC28000C;
054 : 902E0004;
058 : 942F0008;
05C : 8C30000C;



060 : 08000070;
070 : 0C000078;
078 : 34310084;
07C : 02200008;

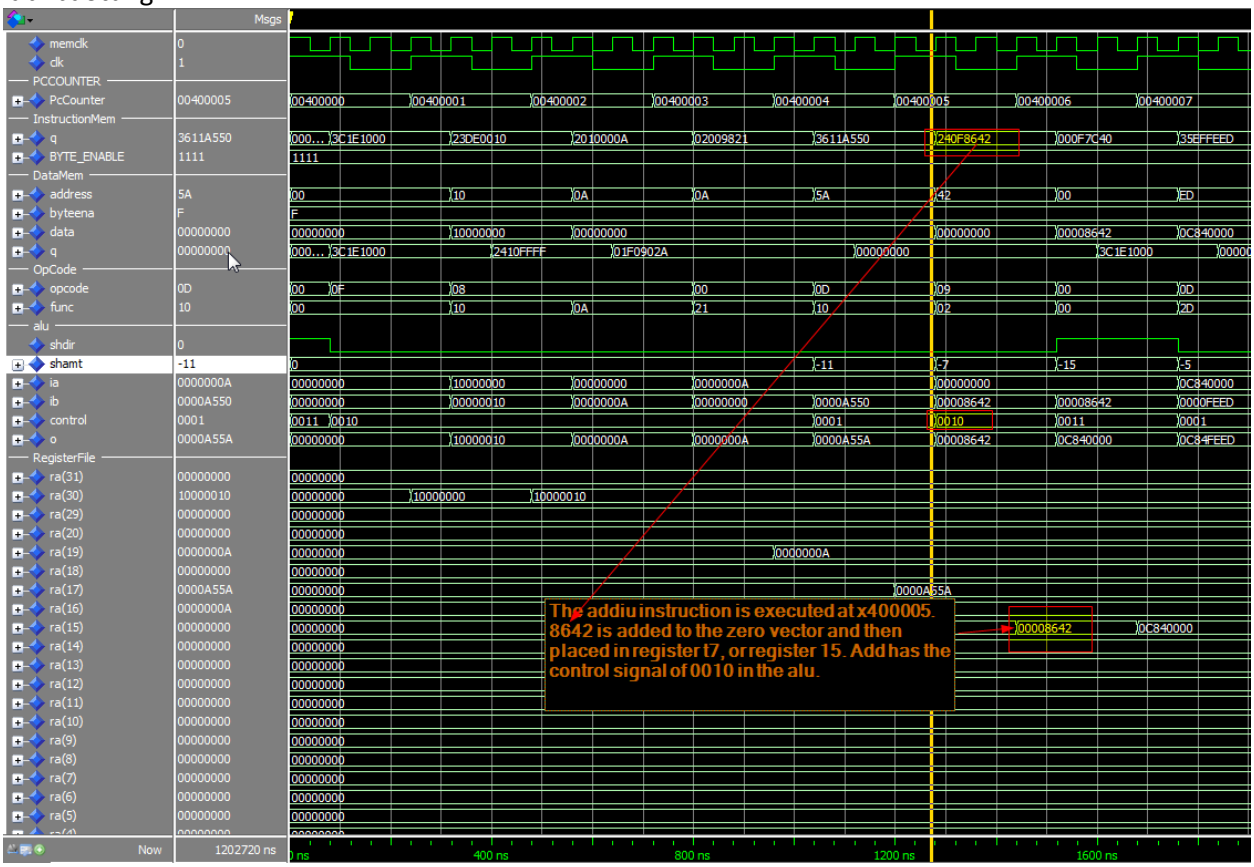


084 : 10220008;
088 : 10210008;
094 : 14210008;
098 : 14220008;

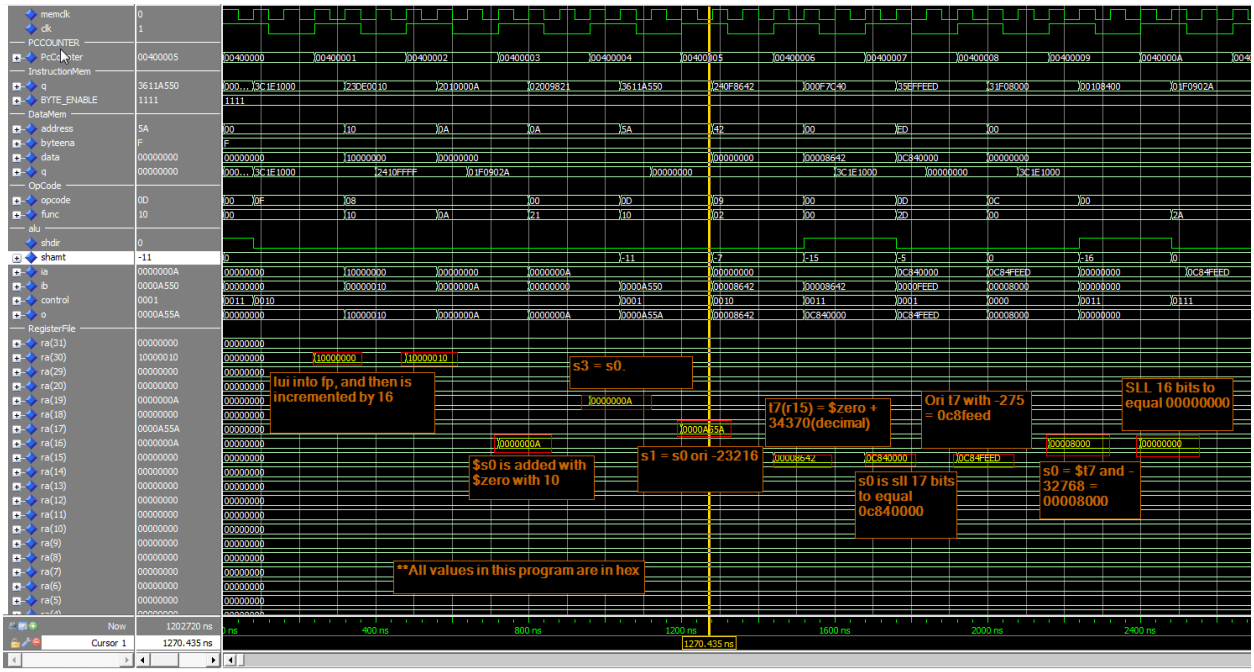


LAB4_TEST.MIF :

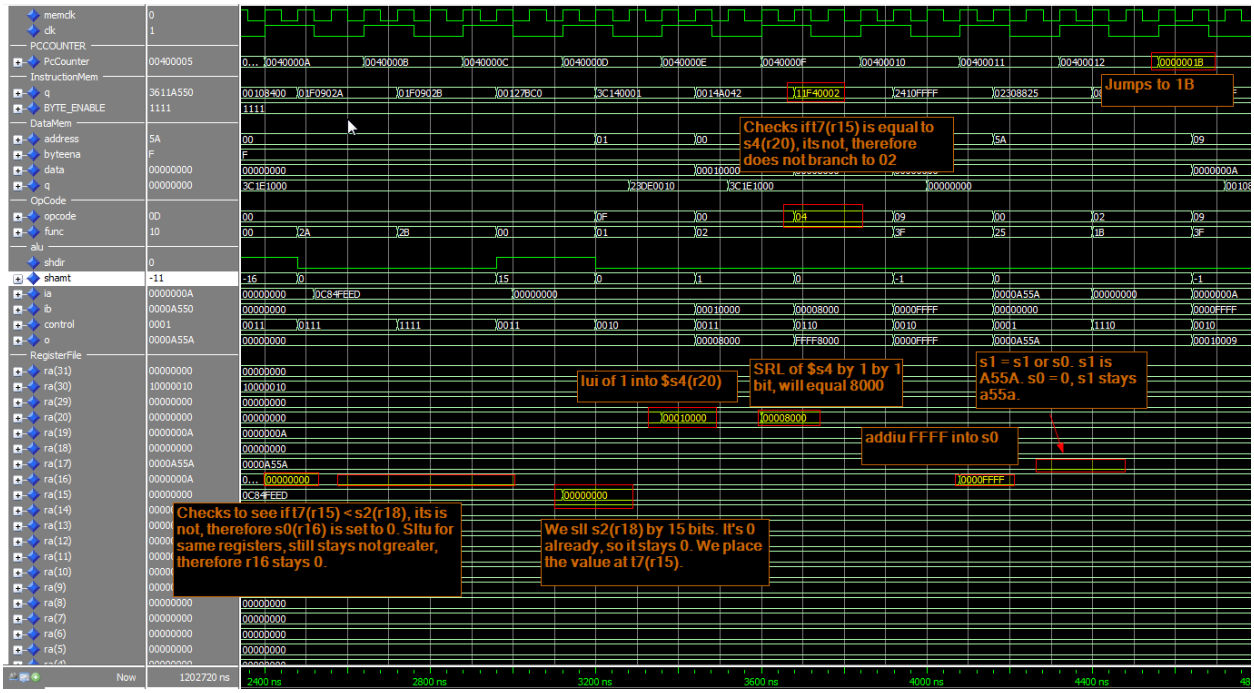
For this program, I will only explain the control flow of the program. I will also annotate the “addiu” instruction as I did not annotate that instruction in the previous simulations of lab4demo and lab4baotung.



```
[0x000000]      0x3C1E1000      # lui $fp, 4096 ($fp = 4096 << 16)
[0x000001]      0x23DE0010      # addi $fp, $fp, 16 ($fp = $fp + 16)
[0x000002]      0x2010000A      # addi $s0, $zero, 10 ($s0 = 10)
[0x000003]      0x02009821      # addu $s3, $s0, $zero ($s3 = $s0)
[0x000004]      0x3611A550      # ori $s1, $s0, -23216 ($s1 = $s0 | -23216)
[0x000005]      0x240F8642      # addiu $t7, $zero, 34370 ($t7 = 34370)
[0x000006]      0x000F7C40      # sll $t7, $t7, 17 ($t7 = $t7 << 17)
[0x000007]      0x35EFFEED      # ori $t7, $t7, -275 ($t7 = $t7 | -275)
[0x000008]      0x31F08000      # andi $s0, $t7, -32768 ($s0 = $t7 & -32768)
[0x000009]      0x00108400      # sll $s0, $s0, 16 ($s0 = $s0 << 16)
```



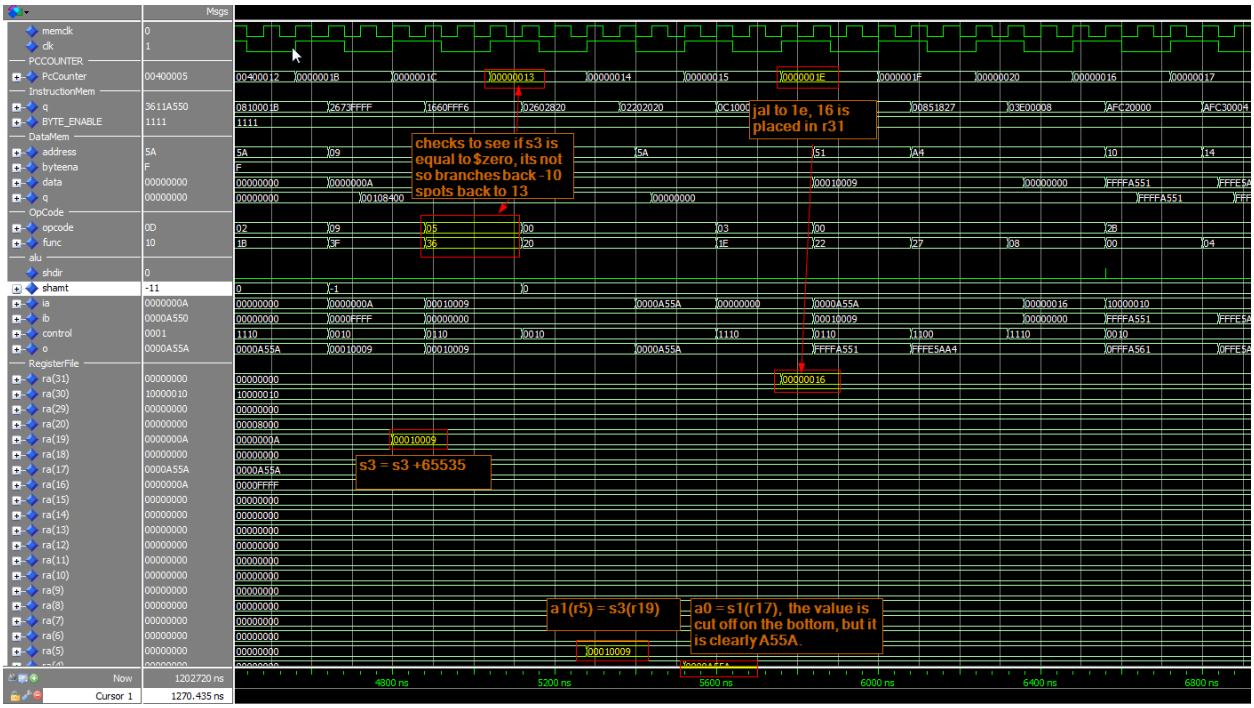
```
[0x00000a]      0x01F0902A      # slt $s2, $t7, $s0 (if ($t7 < $s0) $s2 = 1
else $s2 = 0)
[0x00000b]      0x01F0902B      # sltu $s2, $t7, $s0 (if ($t7 < $s0) $s2 = 1
else $s2 = 0)
[0x00000c]      0x00127BC0      # sll $t7, $s2, 15 ($t7 = $s2 << 15)
[0x00000d]      0x3C140001      # lui $s4, 1 ($s4 = 1 << 16)
[0x00000e]      0x0014A042      # srl $s4, $s4, 1 ($s4 = $s4 >> 1)
[0x00000f]      0x11F40002      # beq $t7, $s4, 2 (if ($t7 == $s4) goto 2)
[0x000010]      0x2410FFFF      # addiu $s0, $zero, 65535 ($s0 = 65535)
[0x000011]      0x02308825      # or $s1, $s1, $s0 ($s1 = $s1 | $s0)
[0x000012]      0x0800001B      # j 0x001B (jump to addr 0x006C)
```

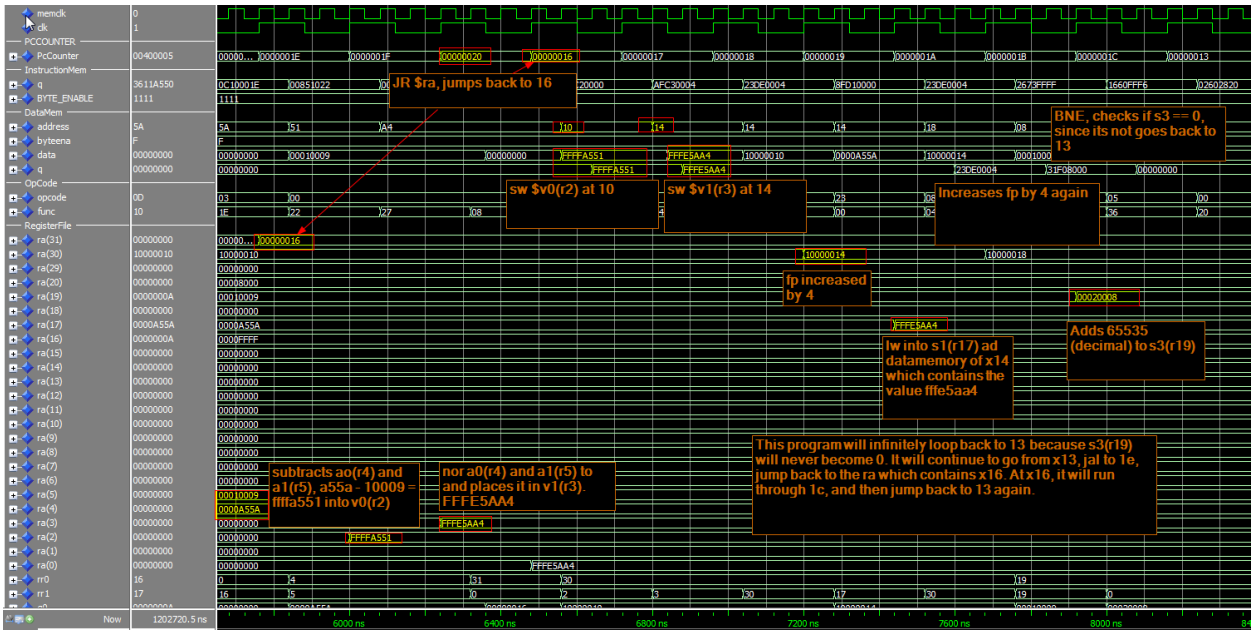


```
[0x000013]      0x02602820      # add $a1, $s3, $zero ($a1 = $s3)
[0x000014]      0x02202020      # add $a0, $s1, $zero ($a0 = $s1)
[0x000015]      0x0C00001E      # jal 0x001E (jump & link to addr 0x0078)
[0x000016]      0xAFC20000      # sw $v0, 0($fp) (mem[$fp + 0] = $v0)
[0x000017]      0xAFC30004      # sw $v1, 4($fp) (mem[$fp + 4] = $v1)
[0x000018]      0x23DE0004      # addi $fp, $fp, 4 ($fp = $fp + 4)
[0x000019]      0x8FD10000      # lw $s1, 0($fp) ($s1 = mem[$fp + 0])
[0x00001a]      0x23DE0004      # addi $fp, $fp, 4 ($fp = $fp + 4)
[0x00001b]      0x2673FFFF      # addiu $s3, $s3, 65535 ($s3 = $s3 + 65535)
[0x00001c]      0x1660FFF6      # bne $zero, $s3, -10 (if ($zero != $s3) goto -
10)
[0x00001d]      0x0800001D      # j 0x001D (jump to addr 0x0074)
[0x00001e]      0x00851022      # sub $v0, $a0, $a1 ($v0 = $a0 - $a1)
[0x00001f]      0x00851827      # nor $v1, $a0, $a1 ($v1 = ?($a0 | $a1))
[0x000020]      0x03E00008      # jr $ra (jump $ra)
```

This program will infinitely loop forever. What happens is that it will continually jump back to x13 because it will check is s3(r19) is equal to 0. S3 will never equal to 0, because we are constantly incrementing it by 65535. It will go back to 13, execute until it JAL to x1E. It will execute to 20, where it will jump back to the address at the return address register which is x16. At x16, it will execute to 1c, where it executes the BNE instruction to check if s3 is equal to 0 again, and then jump back to x13 again.

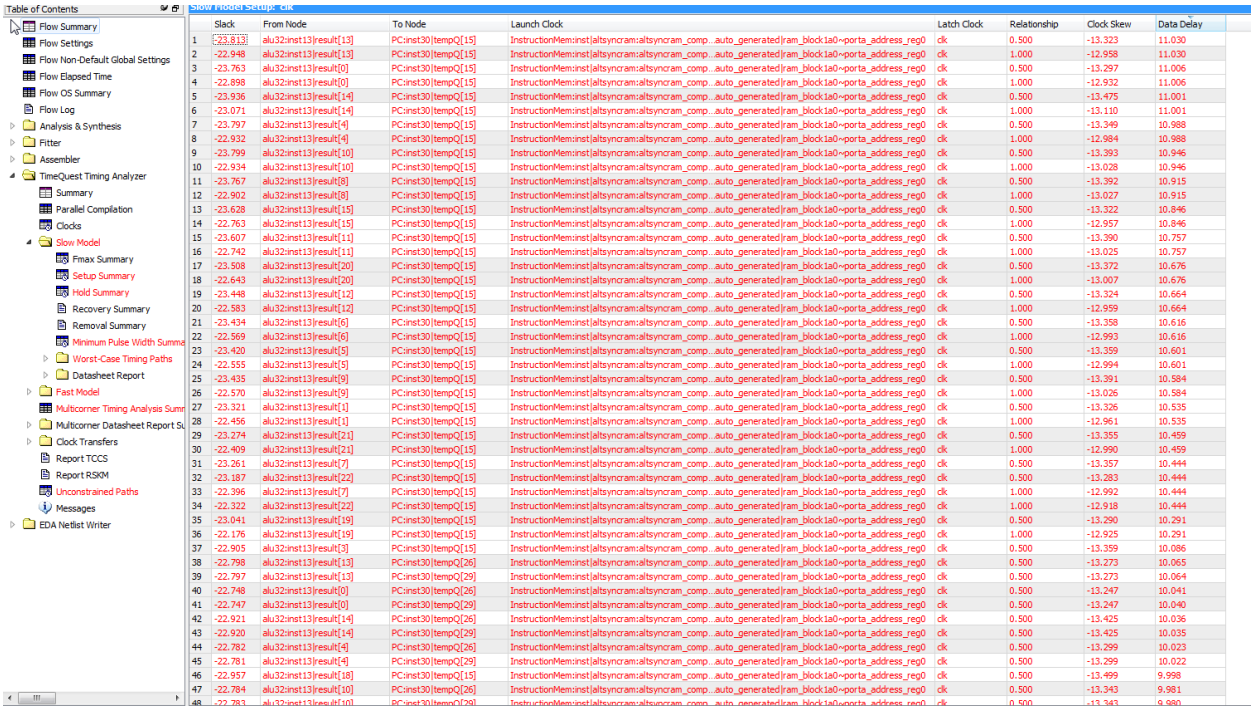
The annotations of the rest of the program flow are on the following page.


jal to 1e, 16 is placed in r31
checks to see if s3 is equal to $zero, its not so branches back -10 spots back to 13
s3 = s3 +65535
a1(r5) = s3(r19)
a0 = s1(r17), the value is cut off on the bottom, but it is clearly A55A.


JR $ra, jumps back to 16
BNE, checks if s3 == 0, since its not goes back to 13
sw $v0(r2) at 10
sw $v1(r3) at 14
increases fp by 4 again
fp increased by 4
Adds 65535 (decimal) to s3(r19)
lw into s1(r17) ad datamemory of x14 which contains the value fffe5aa4
subtracts ao(r4) and a1(r5), a55a - 10009 = ffffa551 into v0(r2)
nor a0(r4) and a1(r5) to and places it in v1(r3). FFFE5AA4
This program will infinitely loop back to 13 because s3(r19) will never become 0. It will continue to go from x13, jal to 1e, jump back to the ra which contains x16. At x16, it will run through 1c, and then jump back to 13 again.

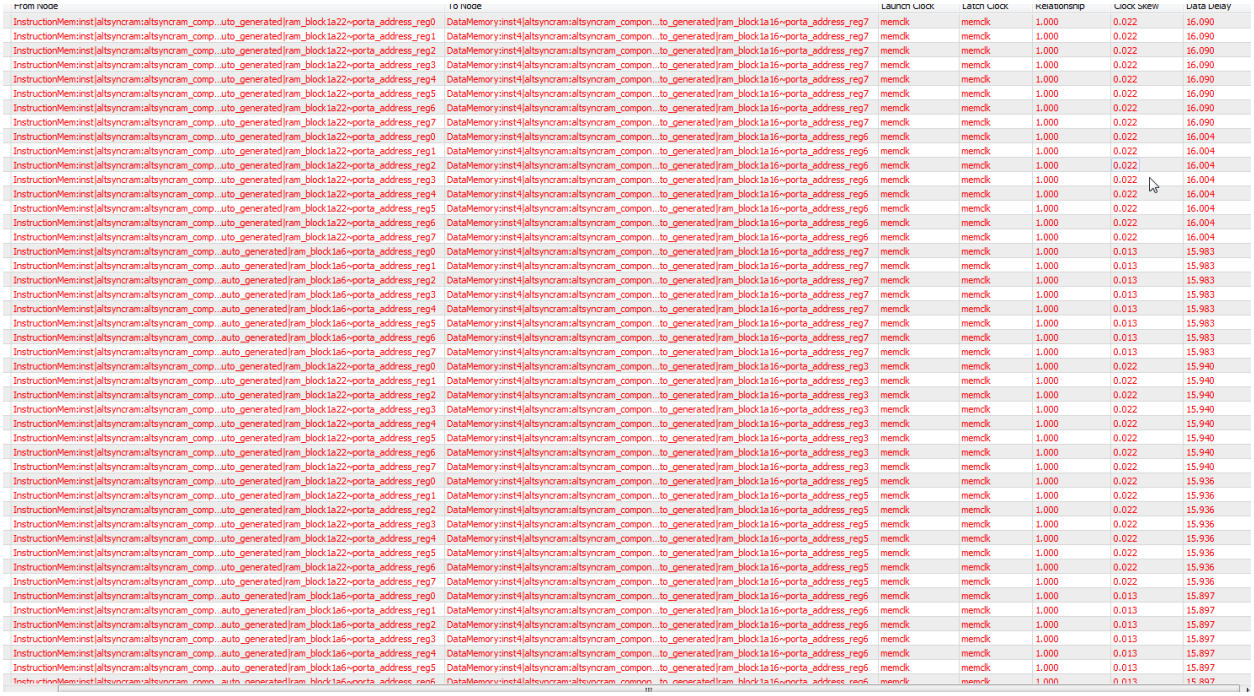
I had a problem with the timing report on Quartus for the slow model, showing that my regular clock was up to around 92 MHz, and my memclk only being around 48 Mhz. Aside from that, logically, each component that I would add would obviously decrease the max frequency of the process. I believe that the lw, lbu, and lhu would decrease the fmax the most because of the access to memory. There are many things you could do speed up your design though. The other instructions only needed extra control signals, which should not have decreased the speed as much since it is mostly sequential logic. I believe only the instructions that require us to access and load from memory will increase the critical path.

As you can see below, the longest delay path is 11.030 ns, which is ironically from the instruction memory and not the data memory.

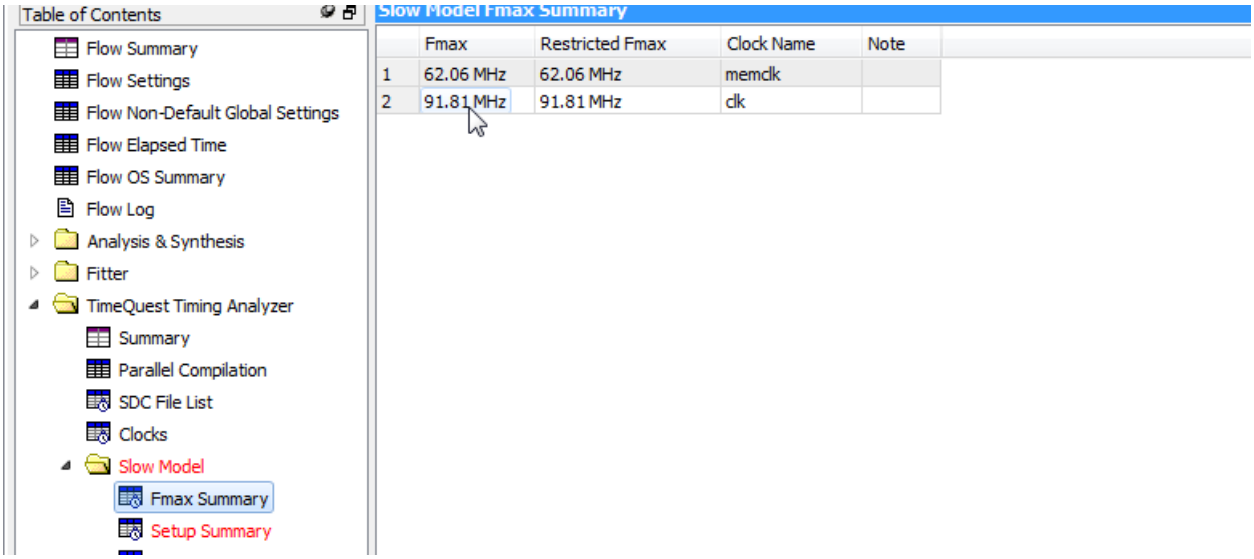


	Slack	From Node	To Node	Launch Clock	Latch Clock	Relationship	Clock Skew	Data Delay
1	-23.813	alu32:inst13|result[13]	PC:inst30|tempQ[15]	InstructionMem:inst|altsyncram:altsyncram_comp...auto_generated|ram_block1a0~porta_address_reg0	clk	0.500	-13.323	11.030
2	-22.948	alu32:inst13|result[13]	PC:inst30|tempQ[15]	InstructionMem:inst|altsyncram:altsyncram_comp...auto_generated|ram_block1a0~porta_address_reg0	clk	1.000	-12.958	11.030
3	-23.763	alu32:inst13|result[0]	PC:inst30|tempQ[15]	InstructionMem:inst|altsyncram:altsyncram_comp...auto_generated|ram_block1a0~porta_address_reg0	clk	0.500	-13.297	11.006
4	-22.898	alu32:inst13|result[0]	PC:inst30|tempQ[15]	InstructionMem:inst|altsyncram:altsyncram_comp...auto_generated|ram_block1a0~porta_address_reg0	clk	1.000	-12.932	11.006
5	-23.936	alu32:inst13|result[14]	PC:inst30|tempQ[15]	InstructionMem:inst|altsyncram:altsyncram_comp...auto_generated|ram_block1a0~porta_address_reg0	clk	0.500	-13.475	11.001
6	-23.071	alu32:inst13|result[14]	PC:inst30|tempQ[15]	InstructionMem:inst|altsyncram:altsyncram_comp...auto_generated|ram_block1a0~porta_address_reg0	clk	1.000	-13.110	11.001
7	-23.797	alu32:inst13|result[4]	PC:inst30|tempQ[15]	InstructionMem:inst|altsyncram:altsyncram_comp...auto_generated|ram_block1a0~porta_address_reg0	clk	0.500	-13.349	10.988
8	-22.932	alu32:inst13|result[4]	PC:inst30|tempQ[15]	InstructionMem:inst|altsyncram:altsyncram_comp...auto_generated|ram_block1a0~porta_address_reg0	clk	1.000	-12.984	10.988
9	-23.799	alu32:inst13|result[10]	PC:inst30|tempQ[15]	InstructionMem:inst|altsyncram:altsyncram_comp...auto_generated|ram_block1a0~porta_address_reg0	clk	0.500	-13.393	10.946
10	-22.934	alu32:inst13|result[10]	PC:inst30|tempQ[15]	InstructionMem:inst|altsyncram:altsyncram_comp...auto_generated|ram_block1a0~porta_address_reg0	clk	1.000	-13.028	10.946
11	-23.767	alu32:inst13|result[8]	PC:inst30|tempQ[15]	InstructionMem:inst|altsyncram:altsyncram_comp...auto_generated|ram_block1a0~porta_address_reg0	clk	0.500	-13.392	10.915
12	-22.902	alu32:inst13|result[8]	PC:inst30|tempQ[15]	InstructionMem:inst|altsyncram:altsyncram_comp...auto_generated|ram_block1a0~porta_address_reg0	clk	1.000	-13.027	10.915
13	-23.628	alu32:inst13|result[15]	PC:inst30|tempQ[15]	InstructionMem:inst|altsyncram:altsyncram_comp...auto_generated|ram_block1a0~porta_address_reg0	clk	0.500	-13.322	10.846
14	-22.763	alu32:inst13|result[15]	PC:inst30|tempQ[15]	InstructionMem:inst|altsyncram:altsyncram_comp...auto_generated|ram_block1a0~porta_address_reg0	clk	1.000	-12.957	10.846
15	-23.607	alu32:inst13|result[11]	PC:inst30|tempQ[15]	InstructionMem:inst|altsyncram:altsyncram_comp...auto_generated|ram_block1a0~porta_address_reg0	clk	0.500	-13.390	10.757
16	-22.742	alu32:inst13|result[11]	PC:inst30|tempQ[15]	InstructionMem:inst|altsyncram:altsyncram_comp...auto_generated|ram_block1a0~porta_address_reg0	clk	1.000	-13.025	10.757
17	-23.508	alu32:inst13|result[20]	PC:inst30|tempQ[15]	InstructionMem:inst|altsyncram:altsyncram_comp...auto_generated|ram_block1a0~porta_address_reg0	clk	0.500	-13.372	10.676
18	-22.643	alu32:inst13|result[20]	PC:inst30|tempQ[15]	InstructionMem:inst|altsyncram:altsyncram_comp...auto_generated|ram_block1a0~porta_address_reg0	clk	1.000	-13.007	10.676
19	-23.448	alu32:inst13|result[12]	PC:inst30|tempQ[15]	InstructionMem:inst|altsyncram:altsyncram_comp...auto_generated|ram_block1a0~porta_address_reg0	clk	0.500	-13.324	10.664
20	-22.583	alu32:inst13|result[12]	PC:inst30|tempQ[15]	InstructionMem:inst|altsyncram:altsyncram_comp...auto_generated|ram_block1a0~porta_address_reg0	clk	1.000	-12.959	10.664
21	-23.434	alu32:inst13|result[6]	PC:inst30|tempQ[15]	InstructionMem:inst|altsyncram:altsyncram_comp...auto_generated|ram_block1a0~porta_address_reg0	clk	0.500	-13.358	10.616
22	-22.569	alu32:inst13|result[6]	PC:inst30|tempQ[15]	InstructionMem:inst|altsyncram:altsyncram_comp...auto_generated|ram_block1a0~porta_address_reg0	clk	1.000	-12.993	10.616
23	-23.420	alu32:inst13|result[5]	PC:inst30|tempQ[15]	InstructionMem:inst|altsyncram:altsyncram_comp...auto_generated|ram_block1a0~porta_address_reg0	clk	0.500	-13.359	10.601
24	-22.555	alu32:inst13|result[5]	PC:inst30|tempQ[15]	InstructionMem:inst|altsyncram:altsyncram_comp...auto_generated|ram_block1a0~porta_address_reg0	clk	1.000	-12.994	10.601
25	-23.435	alu32:inst13|result[9]	PC:inst30|tempQ[15]	InstructionMem:inst|altsyncram:altsyncram_comp...auto_generated|ram_block1a0~porta_address_reg0	clk	0.500	-13.391	10.584
26	-22.570	alu32:inst13|result[9]	PC:inst30|tempQ[15]	InstructionMem:inst|altsyncram:altsyncram_comp...auto_generated|ram_block1a0~porta_address_reg0	clk	1.000	-13.026	10.584
27	-23.321	alu32:inst13|result[1]	PC:inst30|tempQ[15]	InstructionMem:inst|altsyncram:altsyncram_comp...auto_generated|ram_block1a0~porta_address_reg0	clk	0.500	-13.326	10.535
28	-22.456	alu32:inst13|result[1]	PC:inst30|tempQ[15]	InstructionMem:inst|altsyncram:altsyncram_comp...auto_generated|ram_block1a0~porta_address_reg0	clk	1.000	-12.961	10.535
29	-23.274	alu32:inst13|result[21]	PC:inst30|tempQ[15]	InstructionMem:inst|altsyncram:altsyncram_comp...auto_generated|ram_block1a0~porta_address_reg0	clk	0.500	-13.355	10.459
30	-22.409	alu32:inst13|result[21]	PC:inst30|tempQ[15]	InstructionMem:inst|altsyncram:altsyncram_comp...auto_generated|ram_block1a0~porta_address_reg0	clk	1.000	-12.990	10.459
31	-23.261	alu32:inst13|result[7]	PC:inst30|tempQ[15]	InstructionMem:inst|altsyncram:altsyncram_comp...auto_generated|ram_block1a0~porta_address_reg0	clk	0.500	-13.357	10.444
32	-23.187	alu32:inst13|result[22]	PC:inst30|tempQ[15]	InstructionMem:inst|altsyncram:altsyncram_comp...auto_generated|ram_block1a0~porta_address_reg0	clk	0.500	-13.283	10.444
33	-22.396	alu32:inst13|result[7]	PC:inst30|tempQ[15]	InstructionMem:inst|altsyncram:altsyncram_comp...auto_generated|ram_block1a0~porta_address_reg0	clk	1.000	-12.992	10.444
34	-22.322	alu32:inst13|result[22]	PC:inst30|tempQ[15]	InstructionMem:inst|altsyncram:altsyncram_comp...auto_generated|ram_block1a0~porta_address_reg0	clk	1.000	-12.918	10.444
35	-23.041	alu32:inst13|result[19]	PC:inst30|tempQ[15]	InstructionMem:inst|altsyncram:altsyncram_comp...auto_generated|ram_block1a0~porta_address_reg0	clk	0.500	-13.290	10.291
36	-22.176	alu32:inst13|result[19]	PC:inst30|tempQ[15]	InstructionMem:inst|altsyncram:altsyncram_comp...auto_generated|ram_block1a0~porta_address_reg0	clk	1.000	-12.925	10.291
37	-22.905	alu32:inst13|result[3]	PC:inst30|tempQ[15]	InstructionMem:inst|altsyncram:altsyncram_comp...auto_generated|ram_block1a0~porta_address_reg0	clk	0.500	-13.359	10.086
38	-22.798	alu32:inst13|result[13]	PC:inst30|tempQ[26]	InstructionMem:inst|altsyncram:altsyncram_comp...auto_generated|ram_block1a0~porta_address_reg0	clk	0.500	-13.273	10.065
39	-22.797	alu32:inst13|result[13]	PC:inst30|tempQ[29]	InstructionMem:inst|altsyncram:altsyncram_comp...auto_generated|ram_block1a0~porta_address_reg0	clk	0.500	-13.273	10.064
40	-22.748	alu32:inst13|result[0]	PC:inst30|tempQ[26]	InstructionMem:inst|altsyncram:altsyncram_comp...auto_generated|ram_block1a0~porta_address_reg0	clk	0.500	-13.247	10.041
41	-22.747	alu32:inst13|result[0]	PC:inst30|tempQ[29]	InstructionMem:inst|altsyncram:altsyncram_comp...auto_generated|ram_block1a0~porta_address_reg0	clk	0.500	-13.247	10.040
42	-22.921	alu32:inst13|result[14]	PC:inst30|tempQ[26]	InstructionMem:inst|altsyncram:altsyncram_comp...auto_generated|ram_block1a0~porta_address_reg0	clk	0.500	-13.425	10.036
43	-22.920	alu32:inst13|result[14]	PC:inst30|tempQ[29]	InstructionMem:inst|altsyncram:altsyncram_comp...auto_generated|ram_block1a0~porta_address_reg0	clk	0.500	-13.425	10.035
44	-22.782	alu32:inst13|result[4]	PC:inst30|tempQ[26]	InstructionMem:inst|altsyncram:altsyncram_comp...auto_generated|ram_block1a0~porta_address_reg0	clk	0.500	-13.299	10.023
45	-22.781	alu32:inst13|result[4]	PC:inst30|tempQ[29]	InstructionMem:inst|altsyncram:altsyncram_comp...auto_generated|ram_block1a0~porta_address_reg0	clk	0.500	-13.299	10.022
46	-22.957	alu32:inst13|result[18]	PC:inst30|tempQ[15]	InstructionMem:inst|altsyncram:altsyncram_comp...auto_generated|ram_block1a0~porta_address_reg0	clk	0.500	-13.499	9.998
47	-22.784	alu32:inst13|result[10]	PC:inst30|tempQ[26]	InstructionMem:inst|altsyncram:altsyncram_comp...auto_generated|ram_block1a0~porta_address_reg0	clk	0.500	-13.343	9.981

After adding all the new instructions. I recompiled and got a new delay report, which is found below. It increased the delay by about 5.0ns.



From Node	To Node	Launch Clock	Latch Clock	Relationship	Clock Skew	Data Delay
InstructionMem:inst|altsyncram:altsyncram_comp...uto_generated|ram_block1a22~porta_address_reg0	DataMemory:inst4|altsyncram:altsyncram_compon...to_generated|ram_block1a16~porta_address_reg7	memclk	memclk	1.000	0.022	16.090
InstructionMem:inst|altsyncram:altsyncram_comp...uto_generated|ram_block1a22~porta_address_reg1	DataMemory:inst4|altsyncram:altsyncram_compon...to_generated|ram_block1a16~porta_address_reg7	memclk	memclk	1.000	0.022	16.090
InstructionMem:inst|altsyncram:altsyncram_comp...uto_generated|ram_block1a22~porta_address_reg2	DataMemory:inst4|altsyncram:altsyncram_compon...to_generated|ram_block1a16~porta_address_reg7	memclk	memclk	1.000	0.022	16.090
InstructionMem:inst|altsyncram:altsyncram_comp...uto_generated|ram_block1a22~porta_address_reg3	DataMemory:inst4|altsyncram:altsyncram_compon...to_generated|ram_block1a16~porta_address_reg7	memclk	memclk	1.000	0.022	16.090
InstructionMem:inst|altsyncram:altsyncram_comp...uto_generated|ram_block1a22~porta_address_reg4	DataMemory:inst4|altsyncram:altsyncram_compon...to_generated|ram_block1a16~porta_address_reg7	memclk	memclk	1.000	0.022	16.090
InstructionMem:inst|altsyncram:altsyncram_comp...uto_generated|ram_block1a22~porta_address_reg5	DataMemory:inst4|altsyncram:altsyncram_compon...to_generated|ram_block1a16~porta_address_reg7	memclk	memclk	1.000	0.022	16.090
InstructionMem:inst|altsyncram:altsyncram_comp...uto_generated|ram_block1a22~porta_address_reg6	DataMemory:inst4|altsyncram:altsyncram_compon...to_generated|ram_block1a16~porta_address_reg7	memclk	memclk	1.000	0.022	16.090
InstructionMem:inst|altsyncram:altsyncram_comp...uto_generated|ram_block1a22~porta_address_reg7	DataMemory:inst4|altsyncram:altsyncram_compon...to_generated|ram_block1a16~porta_address_reg7	memclk	memclk	1.000	0.022	16.090
InstructionMem:inst|altsyncram:altsyncram_comp...uto_generated|ram_block1a22~porta_address_reg0	DataMemory:inst4|altsyncram:altsyncram_compon...to_generated|ram_block1a16~porta_address_reg6	memclk	memclk	1.000	0.022	16.004
InstructionMem:inst|altsyncram:altsyncram_comp...uto_generated|ram_block1a22~porta_address_reg1	DataMemory:inst4|altsyncram:altsyncram_compon...to_generated|ram_block1a16~porta_address_reg6	memclk	memclk	1.000	0.022	16.004
InstructionMem:inst|altsyncram:altsyncram_comp...uto_generated|ram_block1a22~porta_address_reg2	DataMemory:inst4|altsyncram:altsyncram_compon...to_generated|ram_block1a16~porta_address_reg6	memclk	memclk	1.000	0.022	16.004
InstructionMem:inst|altsyncram:altsyncram_comp...uto_generated|ram_block1a22~porta_address_reg3	DataMemory:inst4|altsyncram:altsyncram_compon...to_generated|ram_block1a16~porta_address_reg6	memclk	memclk	1.000	0.022	16.004
InstructionMem:inst|altsyncram:altsyncram_comp...uto_generated|ram_block1a22~porta_address_reg4	DataMemory:inst4|altsyncram:altsyncram_compon...to_generated|ram_block1a16~porta_address_reg6	memclk	memclk	1.000	0.022	16.004
InstructionMem:inst|altsyncram:altsyncram_comp...uto_generated|ram_block1a22~porta_address_reg5	DataMemory:inst4|altsyncram:altsyncram_compon...to_generated|ram_block1a16~porta_address_reg6	memclk	memclk	1.000	0.022	16.004
InstructionMem:inst|altsyncram:altsyncram_comp...uto_generated|ram_block1a22~porta_address_reg6	DataMemory:inst4|altsyncram:altsyncram_compon...to_generated|ram_block1a16~porta_address_reg6	memclk	memclk	1.000	0.022	16.004
InstructionMem:inst|altsyncram:altsyncram_comp...uto_generated|ram_block1a22~porta_address_reg7	DataMemory:inst4|altsyncram:altsyncram_compon...to_generated|ram_block1a16~porta_address_reg6	memclk	memclk	1.000	0.022	16.004
InstructionMem:inst|altsyncram:altsyncram_comp...auto_generated|ram_block1a6~porta_address_reg0	DataMemory:inst4|altsyncram:altsyncram_compon...to_generated|ram_block1a16~porta_address_reg7	memclk	memclk	1.000	0.013	15.983
InstructionMem:inst|altsyncram:altsyncram_comp...auto_generated|ram_block1a6~porta_address_reg1	DataMemory:inst4|altsyncram:altsyncram_compon...to_generated|ram_block1a16~porta_address_reg7	memclk	memclk	1.000	0.013	15.983
InstructionMem:inst|altsyncram:altsyncram_comp...auto_generated|ram_block1a6~porta_address_reg2	DataMemory:inst4|altsyncram:altsyncram_compon...to_generated|ram_block1a16~porta_address_reg7	memclk	memclk	1.000	0.013	15.983
InstructionMem:inst|altsyncram:altsyncram_comp...auto_generated|ram_block1a6~porta_address_reg3	DataMemory:inst4|altsyncram:altsyncram_compon...to_generated|ram_block1a16~porta_address_reg7	memclk	memclk	1.000	0.013	15.983
InstructionMem:inst|altsyncram:altsyncram_comp...auto_generated|ram_block1a6~porta_address_reg4	DataMemory:inst4|altsyncram:altsyncram_compon...to_generated|ram_block1a16~porta_address_reg7	memclk	memclk	1.000	0.013	15.983
InstructionMem:inst|altsyncram:altsyncram_comp...auto_generated|ram_block1a6~porta_address_reg5	DataMemory:inst4|altsyncram:altsyncram_compon...to_generated|ram_block1a16~porta_address_reg7	memclk	memclk	1.000	0.013	15.983
InstructionMem:inst|altsyncram:altsyncram_comp...auto_generated|ram_block1a6~porta_address_reg6	DataMemory:inst4|altsyncram:altsyncram_compon...to_generated|ram_block1a16~porta_address_reg7	memclk	memclk	1.000	0.013	15.983
InstructionMem:inst|altsyncram:altsyncram_comp...auto_generated|ram_block1a6~porta_address_reg7	DataMemory:inst4|altsyncram:altsyncram_compon...to_generated|ram_block1a16~porta_address_reg7	memclk	memclk	1.000	0.013	15.983
InstructionMem:inst|altsyncram:altsyncram_comp...uto_generated|ram_block1a22~porta_address_reg0	DataMemory:inst4|altsyncram:altsyncram_compon...to_generated|ram_block1a16~porta_address_reg3	memclk	memclk	1.000	0.022	15.940
InstructionMem:inst|altsyncram:altsyncram_comp...uto_generated|ram_block1a22~porta_address_reg1	DataMemory:inst4|altsyncram:altsyncram_compon...to_generated|ram_block1a16~porta_address_reg3	memclk	memclk	1.000	0.022	15.940
InstructionMem:inst|altsyncram:altsyncram_comp...uto_generated|ram_block1a22~porta_address_reg2	DataMemory:inst4|altsyncram:altsyncram_compon...to_generated|ram_block1a16~porta_address_reg3	memclk	memclk	1.000	0.022	15.940
InstructionMem:inst|altsyncram:altsyncram_comp...uto_generated|ram_block1a22~porta_address_reg3	DataMemory:inst4|altsyncram:altsyncram_compon...to_generated|ram_block1a16~porta_address_reg3	memclk	memclk	1.000	0.022	15.940
InstructionMem:inst|altsyncram:altsyncram_comp...uto_generated|ram_block1a22~porta_address_reg4	DataMemory:inst4|altsyncram:altsyncram_compon...to_generated|ram_block1a16~porta_address_reg3	memclk	memclk	1.000	0.022	15.940
InstructionMem:inst|altsyncram:altsyncram_comp...uto_generated|ram_block1a22~porta_address_reg5	DataMemory:inst4|altsyncram:altsyncram_compon...to_generated|ram_block1a16~porta_address_reg3	memclk	memclk	1.000	0.022	15.940
InstructionMem:inst|altsyncram:altsyncram_comp...uto_generated|ram_block1a22~porta_address_reg6	DataMemory:inst4|altsyncram:altsyncram_compon...to_generated|ram_block1a16~porta_address_reg3	memclk	memclk	1.000	0.022	15.940
InstructionMem:inst|altsyncram:altsyncram_comp...uto_generated|ram_block1a22~porta_address_reg7	DataMemory:inst4|altsyncram:altsyncram_compon...to_generated|ram_block1a16~porta_address_reg3	memclk	memclk	1.000	0.022	15.940
InstructionMem:inst|altsyncram:altsyncram_comp...uto_generated|ram_block1a22~porta_address_reg0	DataMemory:inst4|altsyncram:altsyncram_compon...to_generated|ram_block1a16~porta_address_reg5	memclk	memclk	1.000	0.022	15.936
InstructionMem:inst|altsyncram:altsyncram_comp...uto_generated|ram_block1a22~porta_address_reg1	DataMemory:inst4|altsyncram:altsyncram_compon...to_generated|ram_block1a16~porta_address_reg5	memclk	memclk	1.000	0.022	15.936
InstructionMem:inst|altsyncram:altsyncram_comp...uto_generated|ram_block1a22~porta_address_reg2	DataMemory:inst4|altsyncram:altsyncram_compon...to_generated|ram_block1a16~porta_address_reg5	memclk	memclk	1.000	0.022	15.936
InstructionMem:inst|altsyncram:altsyncram_comp...uto_generated|ram_block1a22~porta_address_reg3	DataMemory:inst4|altsyncram:altsyncram_compon...to_generated|ram_block1a16~porta_address_reg5	memclk	memclk	1.000	0.022	15.936
InstructionMem:inst|altsyncram:altsyncram_comp...uto_generated|ram_block1a22~porta_address_reg4	DataMemory:inst4|altsyncram:altsyncram_compon...to_generated|ram_block1a16~porta_address_reg5	memclk	memclk	1.000	0.022	15.936
InstructionMem:inst|altsyncram:altsyncram_comp...uto_generated|ram_block1a22~porta_address_reg5	DataMemory:inst4|altsyncram:altsyncram_compon...to_generated|ram_block1a16~porta_address_reg5	memclk	memclk	1.000	0.022	15.936
InstructionMem:inst|altsyncram:altsyncram_comp...uto_generated|ram_block1a22~porta_address_reg6	DataMemory:inst4|altsyncram:altsyncram_compon...to_generated|ram_block1a16~porta_address_reg5	memclk	memclk	1.000	0.022	15.936
InstructionMem:inst|altsyncram:altsyncram_comp...uto_generated|ram_block1a22~porta_address_reg7	DataMemory:inst4|altsyncram:altsyncram_compon...to_generated|ram_block1a16~porta_address_reg5	memclk	memclk	1.000	0.022	15.936
InstructionMem:inst|altsyncram:altsyncram_comp...auto_generated|ram_block1a6~porta_address_reg0	DataMemory:inst4|altsyncram:altsyncram_compon...to_generated|ram_block1a16~porta_address_reg6	memclk	memclk	1.000	0.013	15.897
InstructionMem:inst|altsyncram:altsyncram_comp...auto_generated|ram_block1a6~porta_address_reg1	DataMemory:inst4|altsyncram:altsyncram_compon...to_generated|ram_block1a16~porta_address_reg6	memclk	memclk	1.000	0.013	15.897
InstructionMem:inst|altsyncram:altsyncram_comp...auto_generated|ram_block1a6~porta_address_reg2	DataMemory:inst4|altsyncram:altsyncram_compon...to_generated|ram_block1a16~porta_address_reg6	memclk	memclk	1.000	0.013	15.897
InstructionMem:inst|altsyncram:altsyncram_comp...auto_generated|ram_block1a6~porta_address_reg3	DataMemory:inst4|altsyncram:altsyncram_compon...to_generated|ram_block1a16~porta_address_reg6	memclk	memclk	1.000	0.013	15.897
InstructionMem:inst|altsyncram:altsyncram_comp...auto_generated|ram_block1a6~porta_address_reg4	DataMemory:inst4|altsyncram:altsyncram_compon...to_generated|ram_block1a16~porta_address_reg6	memclk	memclk	1.000	0.013	15.897
InstructionMem:inst|altsyncram:altsyncram_comp...auto_generated|ram_block1a6~porta_address_reg5	DataMemory:inst4|altsyncram:altsyncram_compon...to_generated|ram_block1a16~porta_address_reg6	memclk	memclk	1.000	0.013	15.897



I added a lot of multiplexors to my design. What I could have done was look directly at the instruction memory and look at the function code. Some of the instructions change only by one bit, and I could have used that one bit as a control signal to when to active a select line for a multiplexor instead of adding a control signal to my controller or alu32control. This would have greatly reduced the logic from the controller as well sped up the processor as well, as the controller would have no need as much sequential logic.

Controller signals :

```
case opcode is -- rtypes first
        when "000000" => -- this will include ALL R TYPE INSTRCUTIONS besides for JR

                aluop <= "000";
                regwrite <= '1';
                regdst <= '1'; -- changes regdst to write to d, and reads from s and t.
                alusrc <='0'; -- takes in value from rs to alu

                when "001000" => -- addi

                aluop <= "001";
                regwrite <= '1';
                regdst <= '0'; -- changes write destination to t.
                ALUSRC <= '1'; -- takes in immediate value to alu

                when "001001" => -- addiu

                zeroorsign <= '1';
                aluop <= "001";
                regwrite <= '1'; -- enables register file to write
                regdst <= '0'; -- changes write destination to t.
                ALUSRC <= '1';

                when "001100" => -- andi
                zeroorsign <= '1';
                aluop <= "101"; -- tells alucontrol its an i instruction
                regwrite <= '1';  -- enables register file to write
                regdst <= '0';  -- changes write destination to rt
                ALUSRC <= '1'; -- takes in immediate value

                when "000100" => --BEQ
                branch <= '1';
                BEQ <= '1';
                aluop <= "110"; -- subtraction

                when "000101" => --BNE
                branch <= '1';
                BNE <= '1';
                aluop <= "110";

                when "001101" => -- ORI
                zeroorsign <= '1';
```

```
aluop <= "010"; -- tells alucontrol its an i instruction
regwrite <= '1';  -- enables register file to write
regdst <= '0';  -- changes write destination to rt
ALUSRC <= '1'; -- takes in immediate value


when "001010" => -- slti
aluop <= "011"; -- tells alu control its an i instruction
regwrite <='1'; -- enables register file to write
regdst <='0';
alusrc <='1'; -- takes in immediate value taht is sign extended.
memtoreg <= '0'; --takes value, either 0 or 1 from aluout, and then will put it into rt.

when "001011" => -- sltiu b hex
zeroorsign <= '1';
aluop <= "100"; -- tells alu control its an i instruction
regwrite <='1'; -- enables register file to write
regdst <='0'; -- writes to rt
alusrc <='1'; -- takes in immediate value taht is sign extended.
memtoreg <= '0'; --takes value, either 0 or 1 from aluout, and then will put it into rt.

when "100100" => -- lbu
 aluop <= "001"; -- immediate instruction
 memtoreg <= '1';
 alusrc <= '1';
 regwrite <='1';
 loadcontrol <= "00"; --chooses mask lbu
 loadormem <= '1';
 lscontrol <= '1';

when "100101" => --lhu
 aluop <= "001"; -- immediate instruction
 memtoreg <= '1';
 alusrc <= '1';
 regwrite <='1';
 loadcontrol <= "01"; -- chooses mask lhu
 loadormem <= '1';
 lscontrol <= '1';


when "100011" => --lw
 aluop <= "001"; --immediate instruction
 memtoreg <= '1';
 alusrc <= '1';
 regwrite <='1';
 loadcontrol <= "11"; --choses entire rs+signext
 loadormem <= '1';
 lscontrol <= '1';
```

```
when "101000" => --sb

  --regdst <= '0';
  BYTEENABLE <= "0001";
  aluop <= "001"; --immediate instruction
  memtoreg <= '1';
  aluop <= "001";
  lscontrol <='1';
  memwrite <= '1';

when "101001" => --sh

  --regdst <= '0';
  BYTEENABLE <= "0011";
  aluop <= "001"; --immediate instruction
  memtoreg <= '1';
  aluop <= "001";
  lscontrol <='1';
  memwrite <= '1';

 when "101011" => --sW

  --regdst <= '0';
  BYTEENABLE <= "1111";
  aluop <= "001"; --immediate instruction
  memtoreg <= '1';
  aluop <= "001";
  lscontrol <='1';
  memwrite <= '1';

when "001111" => --lui
  aluop <="001";
  regwrite <= '1';
  loadcontrol <="10";
  loadormem <= '1';
  lscontrol <='1';

when "000010" => --jump
  jjal <= '1';

when "000011" => --jal
  jjal <= '1';
  regwrite <= '1';

when others =>
aluop <= "111";
end case;
```

```
        --end if;
```

Alu32control :

```
begin
        process(ALUop, func)
        begin
         jr <= '0';
         shdir <= '0';

                if (ALUop = "000") then  -- R TYPE

                        if (func = "100000") then -- add
                                control <= "0010"; -- add
                        elsif (func = "100001") then -- addu
                         control <= "0010";

                         shdir <= '0';
                        elsif (func = "100100") then -- and func number // func code same as lbu
                                control <= "0000"; -- and

                        elsif (func = "100101") then -- or
                                control <= "0001"; -- or control

                        elsif (func = "100111") then -- nor
                                control <= "1100"; -- nor control

                        elsif (func = "101010") then -- slt
                                control <= "0111"; -- slt control

                        elsif (func = "101011") then -- sltu
                                control <= "1111"; -- sltu control

                        elsif (func = "000000") then -- sll
                                control <= "0011"; --sll control
                                shdir <= '1';

                        elsif (func = "000010") then-- srl
                         shdir <= '0';
                                control <= "0011"; -- srl control

                        elsif (func = "100010") then --sub
                                control <= "0110"; -- sub control

                        elsif (func ="100011") then --subu
```

```
                    control <= "0110"; -- subucontrol

            elsif (func ="001000") then -- jr
              control <= "1110";
              jr <= '1';
else
control <= "1110"; --VALUE THATS NOT USED

end if;

elsif (ALUop = "001")then -- I TYPE - -- this includes addi, addiu, beq, bne, lbu, lhu, ll, lw,
                                                    sb, sc, sh, sw
control <= "0010";


elsif (ALUop = "010") then -- I TYPE --- this includes ori
 control <= "0001";

elsif (ALUop = "011") then -- I TYPE -- this includes slti
 control <= "0111";

elsif (ALUop = "100") then -- I TYPE -- this includes sltiu
control <=  "1111";

elsif (AlUop = "101") then -- I Type - -- this includes ANDI
control <= "0000"; -- aND

elsif (ALUop = "110") then -- subtraction immediate-- this includes JAL
control <= "0110"; -- substraction

elsif (AlUop = "111") then -- operations which outputs 1110,
                                -- which does not doing anything -- this inlcudes JR, LUI, J
control <= "1110"; --VALUE THATS NOT USED

end if;
```