컴퓨터 구조

1차 프로젝트

학　　과: 컴퓨터정보공학부

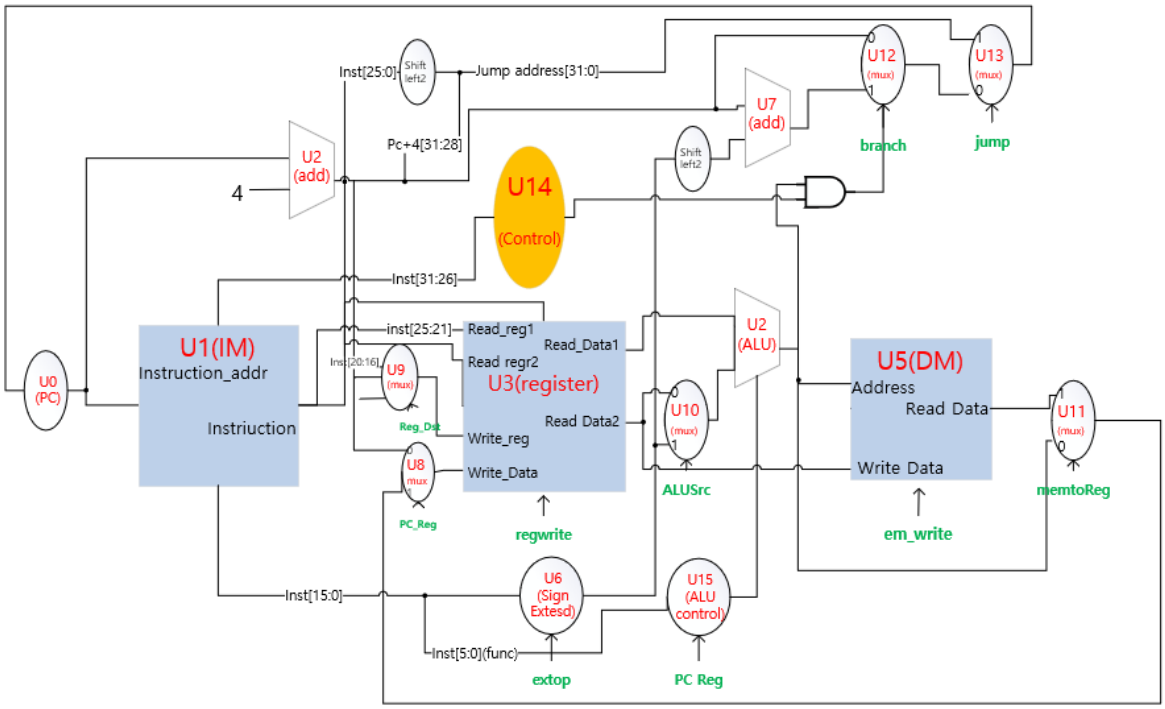
담당교수: 이성원교수님

학　　번: 2017202087

성　　명: 홍 세 정

1. 문제의 해석 및 해결 방향

문제의 해석: 이번과제에서는 single cycle인 명령어를 구현해야 한다. 구현해야 할 명령어는 SLL, SRA, ANDI, BNE, BEQ, MULTU, JALR, XORI, SLTI, JAL이다. Mycontrol에서 각 구현해야 할 명령어에 맞게 input값을 넣어줄 수 있다.



다음과 같이 singlecycle CPU datapath and control path를 그려볼 수 있다. 여기서 초록색 input은 control에서나오는 output으로, module에 들어가는 input을 우리가 조정할 수 있는 control들이다. 우리가 조정해야 할 것들은 총 10개이다. Regwrite(U3), memwrite(U5), extmode(U6), pcToReg(U8), regDst(U9), AluSrc(U10), memtoReg(U11), branch(U12), jump(U13), ALUOp(U15) 10개를 op에 따라서 조정할 수 있다. 다음 10개의 신호는 1bit신호이다.

각 하는 역할을 보면 다음과 같다.

Signal name	effect
Regwrite(U3)	레지스터 파일에 값이 쓰여질 지 말지를 정하는 신호이다. R-type일 때와 load일 때만 input이 1값이 들어간다.
memwrite(U5)	Data를 저장할지 말지를 결정한다. Input이 1이면 저장, input이 0이면 저장하지 않는다. instruction에서 저장하는 경우는 sw의 명령어 밖에 없으므로 sw일 때를 제외하고는 모드 0값을 가지게 된다.
extmode(U6)	Extend를 해준다. Input이 1일 때 sign E, input이 0일 때 zero E를 해

	줄 수 있다. 우리 코드에서 halfword[15]가 0이면 0으로 1이면 1로 확장해 주는게 signE이다.
pcToReg(U8)	Input이 0일 때는 pc에서 나온 주소값(U2의 output), input이 1일 때는 (U2의 output)이 출력된다.
regDst(U9)	레지스터 rd(1)와 rt(0)를 결정한다. R-type일때만 rd의 값이 필요하기 때문에 R-type일 때만 input이 1이 들어가는 것을 알 수 있다. load할 때는 0이, 다른 경우에는 어느 값이 들어가도 상관없다.
AluSrc(U10)	1이면 signextend에서 확장한 immediate값이 들어가고 0이면 레지스터에서 rt의 값이 들어가게 된다. immediate값이 있는 명령어에서 실행된다.
memtoReg(U11)	0일때는 R-type, ALU가 실행된 값을 출력, 1일 때는 데이터 메모리에서 나온 값을 출력하게 된다(load). 나머지 경우에는 어느 값이 들어와도 상관없다.
branch(U12)	Branch결정하는 신호로 branch할 때 만 1을 input으로 받는다. 이때 output은 pc+4+offset이 된다. 0일때는 pc+4를 출력한다.
jump(U13)	Jump를 결정하는 신호로 jump할 때 만 1을 input으로 받는다. Input이 1일 때는 pc+4 +[26:0]shift2을 출력으로 가진다.
ALUOp(U15)	ALU를 선택하는 신호이다. 보통 +와 -가 사용된다.

위의 신호를 각 명령어에 맞게 잘 사용하면 문제를 해결할 수 있다.

우리가 구현해야 할 assembly코드를 machine 코드로 바꿀 수 있다.

Assembly code	Machine code
1. sll $s0, $s1, 2	00000000 00010001 10001000 10000000 Means: s1을 왼쪽으로 2번 shift해서 s0에 값을 넣는다. R-type, function(000000)
2. sra $s0, $s1, 2	00000000 00010001 10001000 10000011 Means: s1을 오른쪽으로 2번 shift해서 s0에 값을 넣는다. R-type, function(000011)

3. andi $s1, $s2, 2	00110010 01010001 00000000 0000010
	rt = rd & 2 //rd와 2를 and한 값을 rt에 넣는다.
	I-type Opcode:(000101001100)

4. bne $s1,$s2, 25	00010110 01010001 00000000 00011001

	Rs와 rt가 같지 않을 때 pc+4 I-type opcode(000101)
5. slti $t0,$t5,4	00101001 10101000 00000000 00000100 t5<4 이면 t0가 1, 아니면 t0가 0이다. I-type opcode(001010)
6. multu $s1,$s1	00000010 00110001 00000000 00011001 S1값과 s1의 값을 곱하여 hi:lo의 출력을 낸다. R-type, function(011001)
7. jalr $s0, $s1	00000010 00100000 10000000 00001001 $s0 = rd, $s1 = rs , rd = pc; pc = rs 이다. R-type, function(001001), rs와 rd를 사용하므로 R-type이다.
8. xori &s1, $s2, 4	00111010 01010001 00000000 0000100 &s1 = s2, 4 xor한 값이다. I-type, opcode (001110)
9. jal 22	00001100 00000000 00000000 00010110 Pc = pc+4 + (22<<2) ra = pc+4 J-type, Opcode (000011)
10. R-type	R-type들은 opcode가 모두 0이다. 그래서 하위 [5:0]의 function 6bit로 instruction을 결정할 수 있다.

위와 같은 machine code를 singlecycle의 module에 넣어 값을 구할 수 있다.

a) Sll

Single name	Reg write	Mem write	Ext mode	pcTo Reg	Reg Dst	Alu Src	Memto Reg	branch	jump	ALUOp
bit	0	0	x	1	1	x	0	0	0	0010

b) Sra

Single name	Reg write	Mem write	Ext mode	pcTo Reg	Reg Dst	Alu Src	Memto Reg	branch	jump	ALUOp
bit	0	0	x	1	1	x	0	0	0	0010

c) Andi

Single name	Reg write	Mem write	Ext mode	pcTo Reg	Reg Dst	Alu Src	Memto Reg	branch	jump	ALUOp
bit	0	0	0	1	0	1	x	0	0	0000

d) Bne

Single name	Reg write	Mem write	Ext mode	pcTo Reg	Reg Dst	Alu Src	Memto Reg	branch	jump	ALUOp
bit	0	0	x	1	x	x	x	1	0	0001

e) Slti

Single name	Reg write	Mem write	Ext mode	pcTo Reg	Reg Dst	Alu Src	Memto Reg	branch	jump	ALUOp
bit	0	0	1	1	0	1	0	0	0	0100

f) Multu

Single name	Reg write	Mem write	Ext mode	pcTo Reg	Reg Dst	Alu Src	Memto Reg	branch	jump	ALUOp
bit	0	0	x	1	1	x	0	0	0	0010

g) Jalr

Single name	Reg write	Mem write	Ext mode	pcTo Reg	Reg Dst	Alu Src	Memto Reg	branch	jump	ALUOp
bit	0	0	x	1	x	x	x	0	1	?

h) xori

Single name	Reg write	Mem write	Ext mode	pcTo Reg	Reg Dst	Alu Src	Memto Reg	branch	jump	ALUOp
00bit	0	0	0	1	0	1	0	0	0	0011

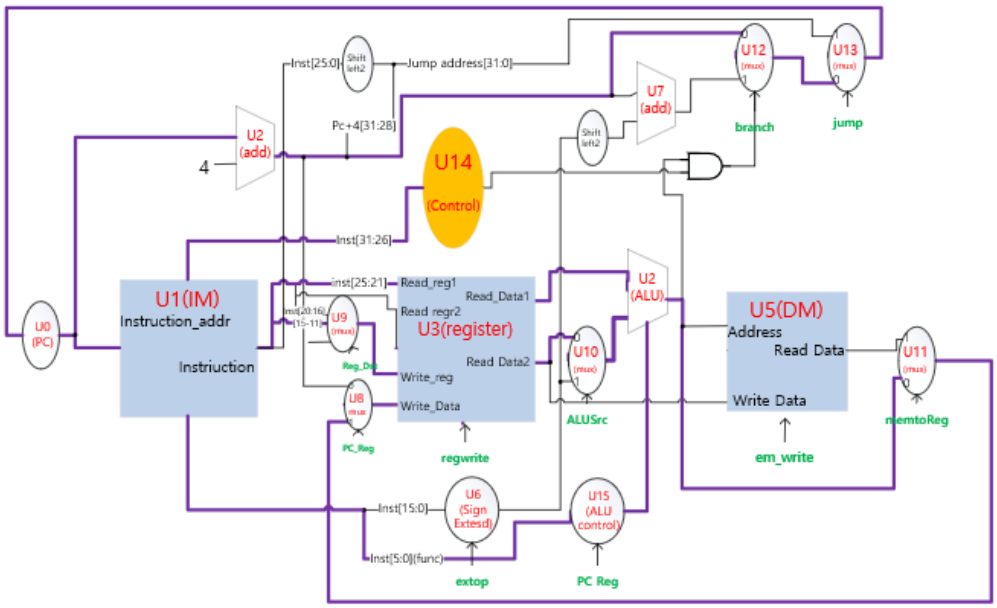
i) Jal

Single name	Reg write	Mem write	Ext mode	pcTo Reg	Reg Dst	Alu Src	Memto Reg	branch	jump	ALUOp
bit	0	0	x	1	1	x	x	0	1	x

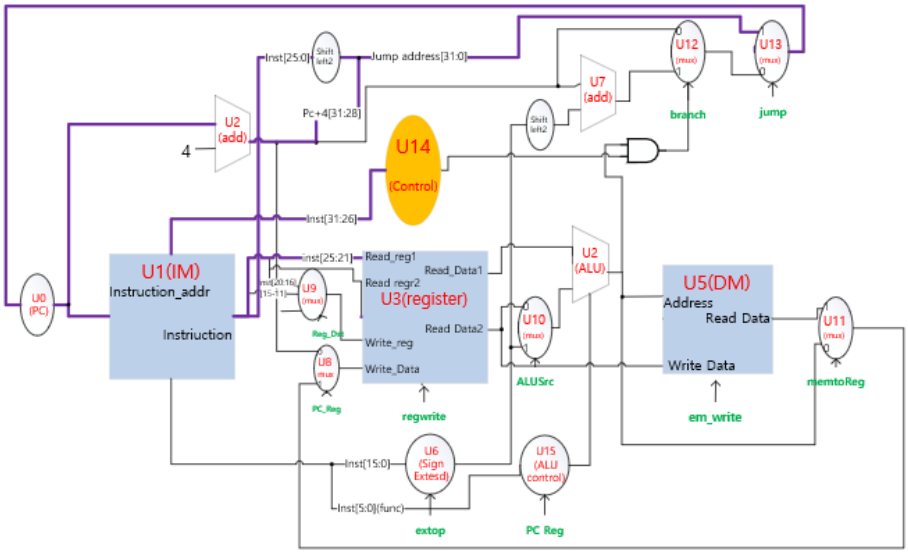
j) R type

Single name	Reg write	Mem write	Ext mode	pcTo Reg	Reg Dst	Alu Src	Memto Reg	branch	jump	ALUOp
bit	0	0	x	1	01	x	0	0	0	0010

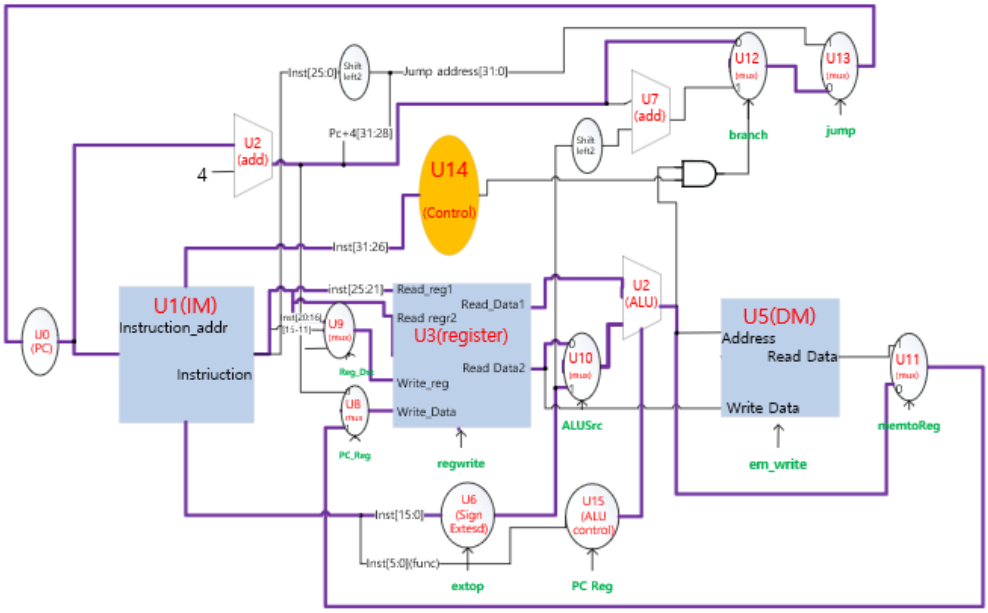
Signal control에 다음과 같이 input을 넣어주면 된다.



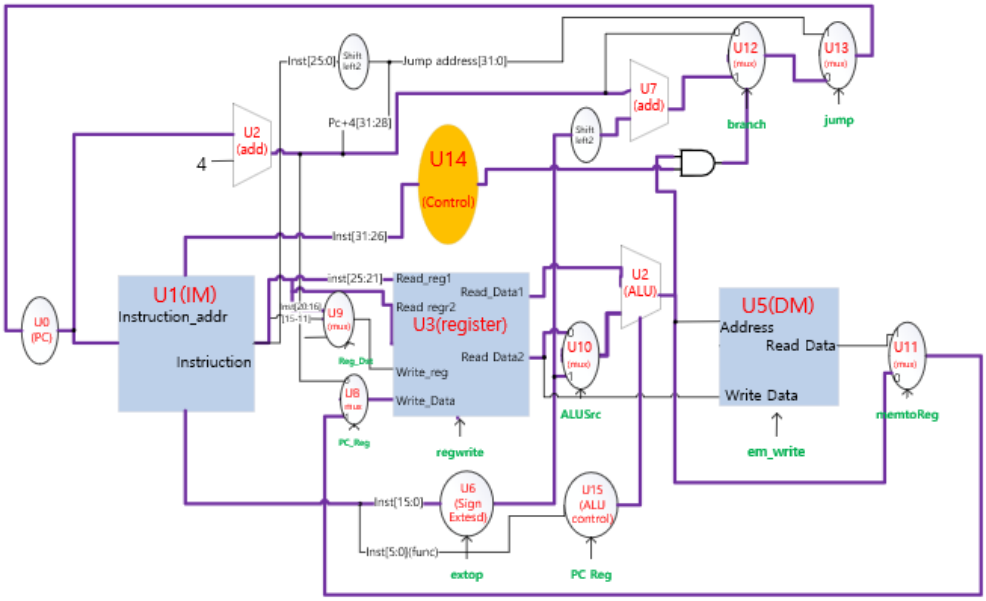
R-type의 datapath는 다음과 같다. (sll, sra, multu, jalr)



j-type의 datapath이다. (jal)



I-type의 datapath이다. 다음과 같이 extend E or extent 0 immediate값이 들어가는 것을 볼 수 있다. (xori, andi, slti)



branch의 datatpath이다. U12에서 blanch signal을 1로 받아 출력하는 것을 확인할 수 있다.

초록색으로 넣어준 input은 control signal이다. 명령어마다 입력을 각각 받을 수 있고, 모든 명령어에 다 들어가게 된다.

2. 설계 의도와 방법

R-type을 제외하고 나머지 opcode를 보고 control signal을 만들어 줄 수 있다. R-type은 opcode가 0이기 때문에 ALUcontrol에서 따로 설정해주기로 한다.

a) Sll, sra, Jalr, Multu

Sll과 sra, Jalr, Multu는 R-type으로 opcode가 모두 0이다. 따라서 mycontrol에서는 따로 signal control을 추가하지 않고 ALU control에서 opcode가 000000이고 functtion을 비교하여 alu를 선택하는 것을 볼 수 있다.

여기서 multu는 곱셈을 하는 명령어이다. 보통 곱셈의 연산은 a = b * c 가 있어야 하는데 multu의 명령어에서는 a의 값이 나오는 것이 아니라, high와 low로 연산하게 된다. 하지만 우리의 코드에서는 연산하여 high와 low연산으로 구분하는 코드가 없다. 그래서 high와 low를 판단해주는 코드가 있으면 multu의 코드가 동작할 수 있다.

b) Andi

Andi는 I-type으로 위에 머신 코드가 부족하다. ALU control에서 and 연산을 할 수 있는 코드가 부족하다(mycontrol에서의 input 값 ALUop의 값에 해당하는 4'b0000이 없어서 동작 할 수가 없다.). Andi 명령어를 실행하기 위해서는 control에서 and연산을 할 수 있는 코드를 넣어주어야 할 것 같다. 만약 ALU control의 모듈(U15)에서 and연산이 작동하면 다음과 같은 코드의 control로 동작할 수 있을 것이다.

```
else if(op==6'b001100) begin // andi
    ALUOp<=4'b0000;
    RegDst<=2'b00;
    ALUSrc<=1'b1;
    RegWrite<=1'b0;
    MemtoReg<=1'bx;
    MemWrite<=1'b0;
    Jump<=1'b0;
    Branch<=1'b0;
    PCToReg<=1'b1;
    ExtMode<=1'b0;
end
```

c) Bne

bne 코드이다. ALUop에 input 4'b0001의 값을 넣어주어 ALU control에서 빼기의 연산을 한다. BNE는 같은지를 확인하는 명령어이기 때문에 빼기의 연산을 실행한다. 하지만 ALU(U4)의 모듈에서 31번째 줄assign o_zero = (o_result != 32'b0) ? 1'b1 : 1'b0;에서 result가 0일 때 zero flag가 1비트를 가져야 하는데 그렇지 않아서 결과 값이 반대로 나오게 된다. 위의 코드를 assign o_zero = (o_result != 32'b0) ? 1'b0 : 1'b1; 다음과 같이 결과를 실행하면 값이 제대로 나올 수 있을 것이다.

```
else if(op==6'b000101) begin // bne
    ALUOp<=4'b0001;
    RegDst<=2'b11;
    ALUSrc<=1'bx;
    RegWrite<=1'b0;
    MemtoReg<=1'b1;
    MemWrite<=1'b0;
    Jump<=1'b0;
    Branch<=1'b1;
    PCToReg<=1'b1;
    ExtMode<=1'bx;
end
```

d) Slti

I-type으로 opcode 6'b001010을 보고 다음과 같은 코드를 실행할 수 있다.

다음과 같이 slti의 코드를 알맞은 signal을 input으로 넣어 구현할 수 있다.

Slti는 레지스터 두개의 값을 비교하는 명령어이다.

```
else if(op==6'b001010) begin // slti
    ALUOp<=4'b0100;
    RegDst<=2'b11;
    ALUSrc<=1'b1;
    RegWrite<=1'b0;
    MemtoReg<=1'b0;
    MemWrite<=1'b0;
    Jump<=1'b0;
    Branch<=1'b0;
    PCToReg<=1'b1;
    ExtMode<=1'b1;
end
```

e) Xori

```
else if(op==6'b001110) begin // xori
    ALUOp<=4'b0011;
    RegDst<=2'b00;
    ALUSrc<=1'b1;
    RegWrite<=1'b0;
    MemtoReg<=1'b0;
    MemWrite<=1'b0;
    Jump<=1'b0;
    Branch<=1'b0;
    PCToReg<=1'b1;
    ExtMode<=1'b0;
end
```

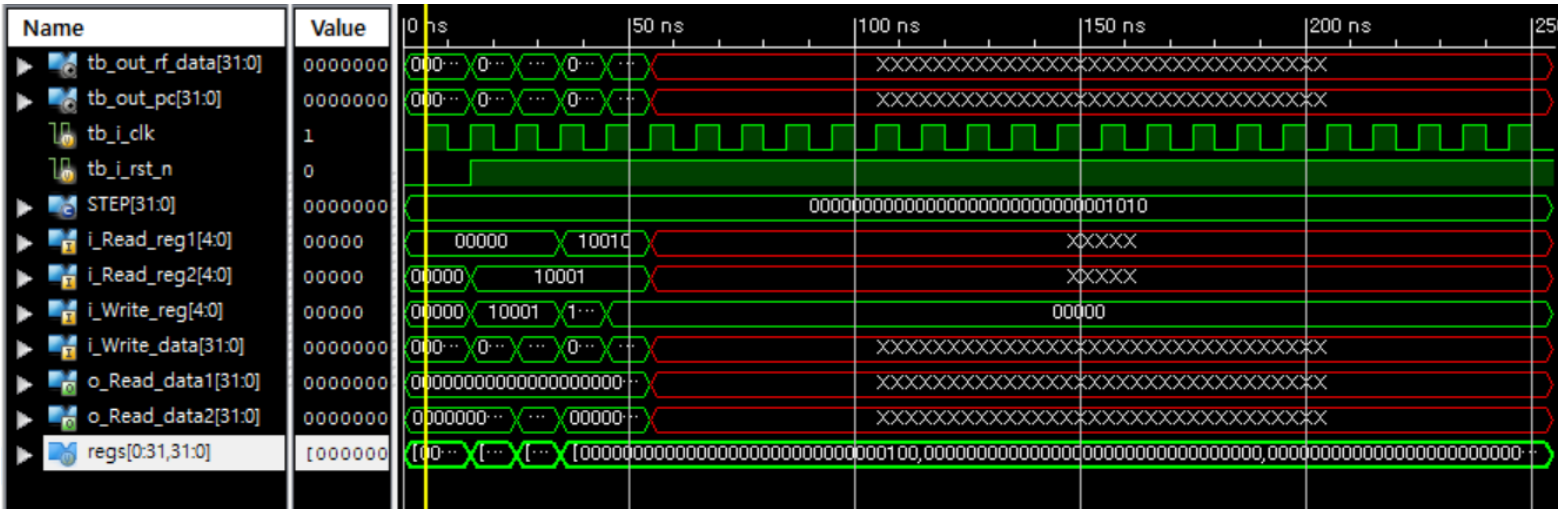
I-type으로 opcode 6'b001110을 보고 다음과 같은 코드를 실행할 수 있다. 다음과 같이 xori의 코드를 알맞은 signal을 input으로 넣어 구현할 수 있다. xori는 레지스터 두개의 값을 비교하는 명령어이다.

f) Jal

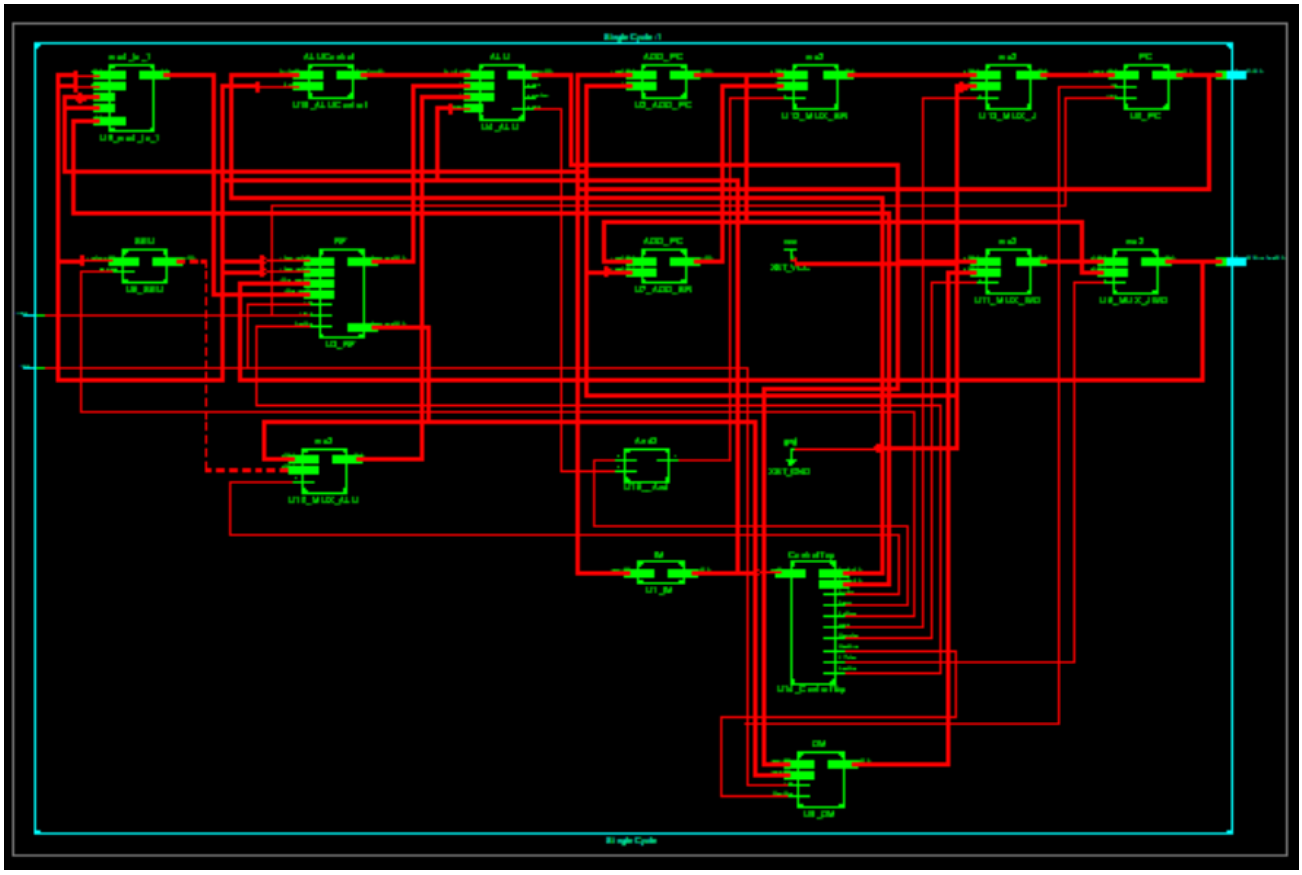
```
else if(op==6'b000011) begin // jal
    ALUOp<=4'bx;
    RegDst<=2'b11;
    ALUSrc<=1'bx;
    RegWrite<=1'b0;
    MemtoReg<=1'bx;
    MemWrite<=1'b0;
    Jump<=1'b1;
    Branch<=1'b0;
    PCToReg<=1'b1;
    ExtMode<=1'bx;
end
```

Jal은 jump하는 명령어이다. Pc = pc+4 + (22<<2) ra = pc+4을 실행하게 된다. 위에서 그린 datapath대로 이동하게 되면 j와 같은 실행만 하게 된다. 그래서 Pc = pc+4 + (22<<2) ra = pc+4을 실행할 수 있게 module을 더 만들어야 할 거 같다. Offeset을 더하기 전에 shift한 입력을 받아와서 구현할 수 있을 것 같다.

g) Testbench



다음과 같은 testbench를 나타낼 수 있다. Register file을 추가하여 reset과 clock에 맞춰 변화하는 것도 볼 수 있다.



3. 고찰

컴퓨터 구조의 첫번째 프로젝트였다. ISE의 프로그램을 어떻게 설정하는지도 모르고 심지어 처음에는 파일을 어떻게 실행시키는지도 몰랐다. 그래서 이것저것 다 눌러보고 검색해보면서 RTL viewer과 testbench 실행하는 방법을 알 수 있었다. RTL viewer를 보면서 어느 부분이 잘못되었는지 확인하면서 코드를 수정할 수 있었다.

생각보다 배웠던 코드에서 구현이 안되는 명령어가 많았다. 생각을 좀 더 하고 어떻게 동작하는지 이해하고 직접 손으로 그려보면서 따라가다 보면서 동작이 되지 않는 것도 구현할 수 있을 거 같다.