T.C.

MARMARA UNIVERSITY

FACULTY of ENGINEERING

COMPUTER ENGINEERING DEPARTMENT

CSE3038 - Computer Organization

Project-2 Report

Group Members

150118073 - Serkan KOÇ

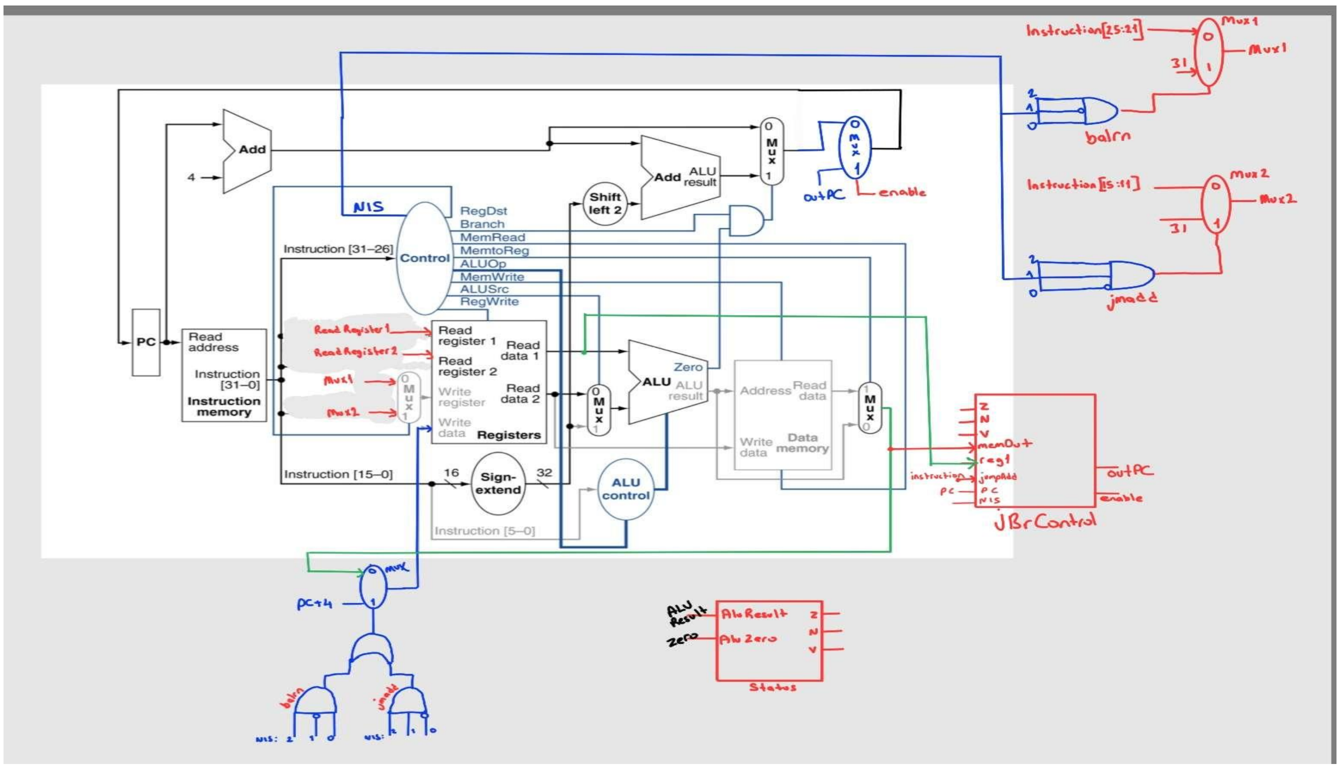
150119639 - Erdem PEHLİVANLAR

150120056 - Haldun Halil OLCAY

150119657- Abdulkerim Talha TİMUR

1. General Description and Datapath

In this project, we simply implement a MIPS single-cycle datapath which includes additional instructions. We have designed additional two components which are Status and JBrControl and an additional control signal **"NIS"** for branch and jump instructions. The detailed datapath with the diagram is below. Also, we used two additional multiplexers while implementing the balrn and jmadd instructions to combine the new data paths into the default design.



2-New Components

I-Status Component

The new status module has 2 inputs (ALUResult and Zero) and 3 outputs (z, n, and v values). ALUResult is the default value that comes from the alu module and Zero is the input that checks whether the output of the alu module is zero or not.

It is a 3-bit input and it is used in jBrControl. Status input is the main input that selects which instruction will be executed.

If alu result is negative, then the n flag will be 1 and others are 0. If alu zero input is 1, (means that alu result is 0) then z flag will be 1 and others are 0. Last, if there is an overflow in the alu result, the v flag will be 1. These condition flags will be used in the jBrControl module to determine which instruction will be executed.

II- JBrControl

JBrControl module is the main module that executes the newly implemented instruction set. It has 7 inputs (Z, N, V, jumpAdd, memOut, PC, NIS,reg1) and 2 inputs (outPc, enable). Z, N, and V inputs come from the status module based on the alu module output. Status is the input that comes from the control unit.Here is the list of status bits and corresponding instructions:

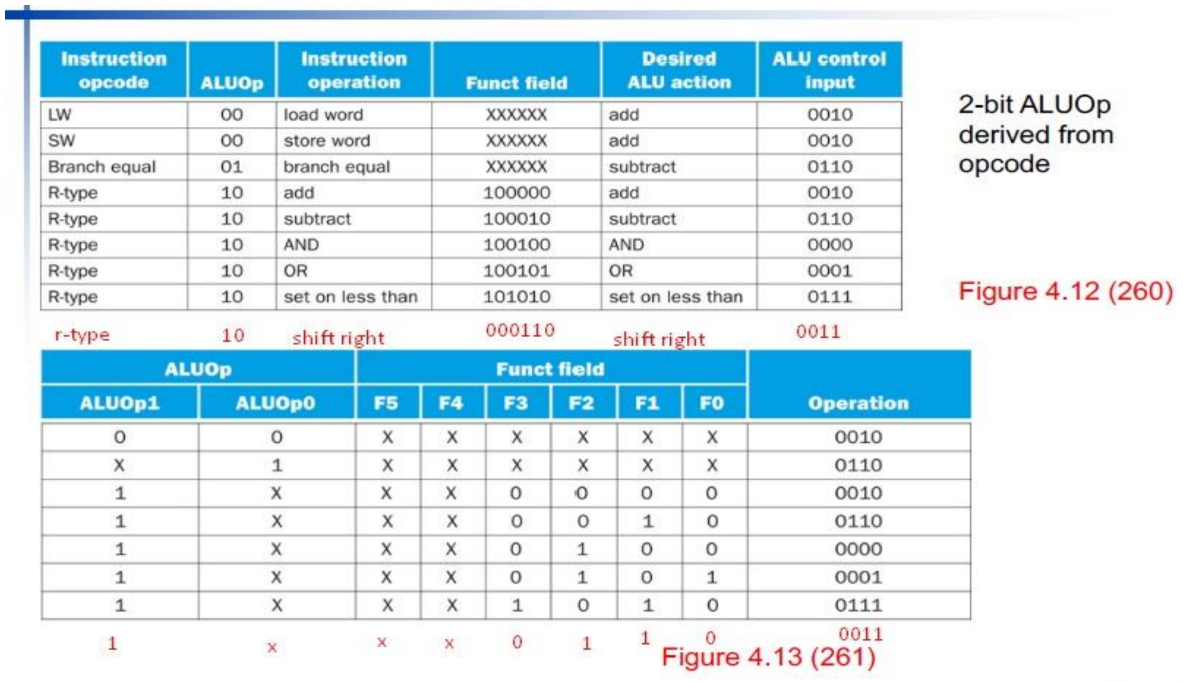
- 000: None of the new instructions will be executed.
- 001: bmv
- 010: bz
- 011: srlv
- 110: jsp
- 101: balrn
- 110: jmadd

memOut is the result that comes from data memory. PC is the pc + 4 value and jumpAdd is the direct address to jump(Instruction [25:0]). outPc is the program counter that will be the next address of the program and enable bit is used to activate the jBrControl module. If status and enable are not 0, new instructions will be executed.

3. New Instructions

1- **balrn**, if Status [N] = 1, branches to address found in register, $rs link address is stored in $rd (which defaults to 31)
2- **jmadd** jumps to address found in memory [$rs+ $rt], link address is stored in $31
3- bmv, if Status [V] = 1, branches to address found in memory
4- bz if Status [Z] = 1, branches to pseudo-direct address (formed as j does)
5- jsp jumps to address found in memory where the memory address is written in register ($sp).
6- srlv, shift register $rt to right by the value in register $rs, and store the result in register $rd.

The new modified signal instruction is below. We made changes regarding the **srlv** instruction for the right shift operation.



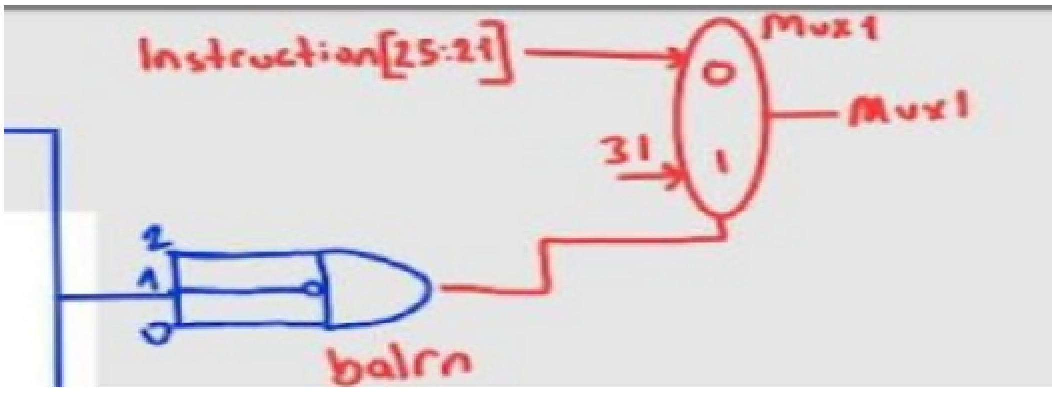
Instruction opcode	ALUOp	Instruction operation	Funct field	Desired ALU action	ALU control input
LW	00	load word	XXXXXX	add	0010
SW	00	store word	XXXXXX	add	0010
Branch equal	01	branch equal	XXXXXX	subtract	0110
R-type	10	add	100000	add	0010
R-type	10	subtract	100010	subtract	0110
R-type	10	AND	100100	AND	0000
R-type	10	OR	100101	OR	0001
R-type	10	set on less than	101010	set on less than	0111
r-type	10	shift right	000110	shift right	0011

2-bit ALUOp derived from opcode

Figure 4.12 (260)

ALUOp		Funct field						
ALUOp1	ALUOp0	F5	F4	F3	F2	F1	F0	Operation
0	0	X	X	X	X	X	X	0010
X	1	X	X	X	X	X	X	0110
1	X	X	X	0	0	0	0	0010
1	X	X	X	0	0	1	0	0110
1	X	X	X	0	1	0	0	0000
1	X	X	X	0	1	0	1	0001
1	X	X	X	1	0	1	0	0111
1	x	x	x	0	1	1	0	0011

Figure 4.13 (261)

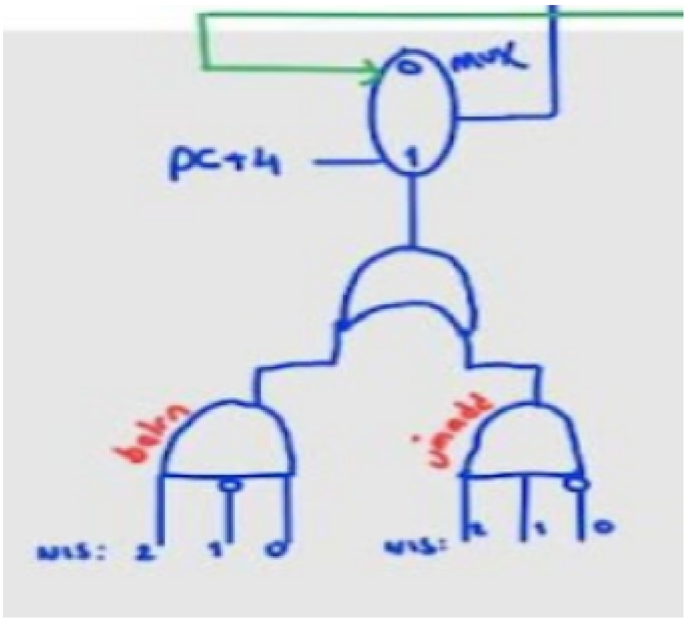


Instruction 1: balrn

For this instruction NIS control signal is 101. We implemented an AND gate according to these status bits. It takes register 31 as input into the multiplexer and gives output . Also write data signal is activated with NIS signal and conducts necessary actions. It writes the link address to the specified register .

Relevant Verilog code block:

```
3'b101: // balrn instruction is active
                        begin
                        outPC = n ? reg1 : PC4;
                        enable = 1;
                        end
```



Instruction 2: bz

bz 48 => (if status[Z] ==1, branch to address 48)

In this instruction if the Z signal from the status component is 1 it jumps directly to the target address. For this instruction we did not modify the datapath. It happens in the JBrControl component. If the status NIS code is 010 then bz instruction is active. if the z signal is one it assigns jumpAdd to the outPC. If the z signal is zero then it assigns the PC address to outPC and finally makes enable bit 1.

The following code block shows the general procedure.

Relevant Verilog code block:

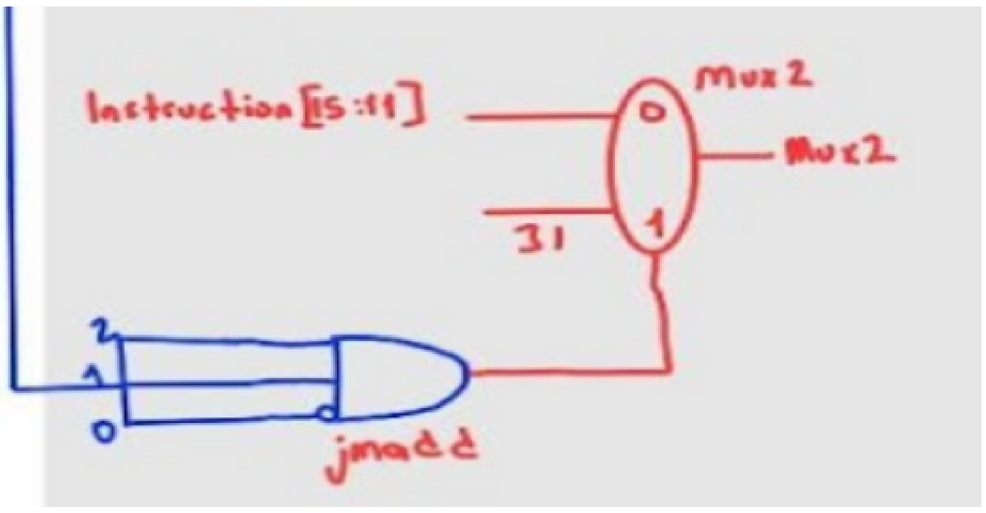
```
module jBrControl(outPC, enable, PC4, memOut, reg1, jmpAddr, nis0, nis1, nis2, n, z, v);
…..

                3'b010: // bz instruction is active
                        begin
                        outPC= z ? jmpAddr : PC4;
                        enable = 1;
                        end
                …..
```

Instruction 3:jmadd

Relevant Verilog code block:

```
        3'b110: // jmadd instruction is active
                begin
                outPC = memOut;
                enable = 1;
                end
```

If the NIS signal bits are 110 then Jmadd instruction is active. It assigns memOut value to the outPC and makes enable bit 1.



Instruction 4: srlv

e.g, (srlv $rd, $rt, $rs)

srlv=> (shift register $rt to right by the value in register $rs, and store the result in register $rd)

It is not added as an additional component for the R-Type srlv instruction. It is modified only for control signals. **NIS Signal** of srlv instruction is **011**.

We made some changes in the **ALU** and **ALU CONTROL** components to handle this instruction.

ALUCONT

```
……

if(aluop1)//R-type
begin
……

if (~(f3)&f2&f1&~(f0))gout=3'b011;//function code=0110,ALU control=011 (shift right)
```

ALU

```
module alu32(sum,a,b,zout,gin);//ALU operation according to the ALU control line values
……….
begin
        case(gin)
        ………….
        3'b011: sum=a>>>b;              //ALU control line=011, arithmetic shift right
        ………….
zout=~(|sum);
end
endmodule
```

Instruction 5: bmv

Relevant Verilog code block:

```
3'b001: // bmv instruction is active
                        begin
                        outPC = v ? memOut : PC4;
                        enable = 1;
                        end
```

For this instruction, if the v signal is 1memOut is assigned to the outPC. If the v signal is zero, the next program counter value PC + 4 is assigned to the outPC. Finally the enable signal is converted to 1. Necessary NIS signal for this instruction is 001. There is no additional component for this instruction. The JBrControl component handles this procedure.

Instruction 6: jsp

jsp=> (jumps to address found in memory where the memory address is written in register ($sp).)

This instruction is controlled by the NIS control signal. **NIS signal** of jsp instruction is **100**.

Relevant Verilog code block:

```
always @ (*)
begin
        case(nis_check)
                ….

                3'b100: // jsp instruction is active

                        begin
                        outPC = memOut;
                        enable = 1;
                        end
                ….
```

Other Component Modifications:

We made some changes in the control module. We modified the module by adding the new variables as input and output. Below we presented the modified code blocks.

```
          module control(in, func, regdest, alusrc, memtoreg, regwrite, memread, memwrite,
branch, aluop1, aluop2, nis0, nis1, nis2);

………..

assign bmv=~in[5]& in[4]&(~in[3])&in[2]&in[1]&(~in[0]); // 010110 opcode = 22

assign balrn=~in[5]&
(~in[4])&(~in[3])&(~in[2])&(~in[1])&(~in[0])&(~func[5])&(func[4])&(~func[3])&(func[2
])&(func[1])&(func[0]); // 010111  func code = 23

assign jmadd=~in[5]&
(~in[4])&(~in[3])&(~in[2])&(~in[1])&(~in[0])&(func[5])&(~func[4])&(~func[3])&(~func[
2])&(~func[1])&(~func[0]); // 100000  func code = 32

assign bz=~in[5]& in[4]&in[3]&(~in[2])&(~in[1])&(~in[0]); // 011000 opcode = 24

assign jmadd=~in[5]&
(~in[4])&(~in[3])&(~in[2])&(~in[1])&(~in[0])&(~func[5])&(~func[4])&(~func[3])&(func[
2])&(func[1])&(~func[0]); // 000110  func code = 6

assign jsp=~in[5]& in[4]&(~in[3])&(~in[2])&in[1]&(~in[0]); // 010010 opcode = 18

…….

assign alusrc=lw|sw|bmv|jsp;

assign memtoreg=lw|bmv|jmadd|jsp;

assign regwrite=rformat|lw|balrn|jmadd|srlv;

assign memread=lw|bmv|jmadd|jsp;

……

assign nis0 = beq|bmv|srlv|balrn;

assign nis1 = beq|bz|srlv|jsp|jmadd;

assign nis2 = beq|jsp|balrn|jmadd;

……
```