

CSE 3038 COMPUTER ORGANIZATION

Programming Project 2
Report

04.06.2022

Prepared By

İSMAİL ÖKSÜZ - 150119516

BARIŞ HAZAR - 150118019

ULAŞ DENİZ IŞIK - 150118887

EMRE SAĞIROĞLU - 150119766



TABLE OF CONTENTS

1. INTRODUCTION	3
2. TABLES FOR SIGNALS	4
2.1. Table 1	4
2.2 Table 2	5
3. ADDITIONAL VERILOG FILES	6
3.1. 'mult8_to_1_32.v'	6
3.2. 'new_component.v'	7
3.3. 'flipflop.v'	7
4. INSTRUCTIONS	8
4.1. balmn	8
4.2. brz	9
4.3. jmadd	10
4.4. bn	11
4.5. sllv	12
4.6.. jspal	13
5. SINGLE CYCLE DATAPATH	14
6. CONCLUSION & SUMMARY	15
7. REFERENCES	15

1.Introduction:

Tables for each signals that were used, extra Verilog files that were added, instructions that we used (represented in this section either), new components and single-cycle datapath are represented in this document.

Before we begin, the instructions that were used to complete this project are balmn, brz, jmadd, bn, sllv and jspal. In short:

Instr	Type	Code	Syntax	Meaning
balmn	I-type	opcode=24	balmn \$rt, imm16(\$rs)	if Status [N]=1, branches to address found in memory, link address is stored in \$rt
brz	R-type	funct=20	brz \$rs	if Status [Z]=1, branches to address found in register \$rs
jmadd	R-type	funct=32	jmadd \$rs,\$rt	jumps to address found in memory [\$rs+\$rt], link address is stored in \$31
bn	J-type	opcode=25	bn Target	if Status [N]=1, branches to pseude-direct address (formed as j does)
sllv	R-type	func=4	sllv \$rd, \$rt, \$rs	shift register \$rt to left by the value in register \$rs, and store the result in register \$rd
jspal	I-type	opcode=19	jspal	jumps to address found in memory where the memory address is written in register 29 (\$sp) and link address is stored in memory (DataMemory[Register[29]])

2.Tables:

2.1.Table 1:

Table 1 shows the values of all instructions for each control signals.

	RegDst	Jump	Branch	MemRead	MemToReg
(5) balmn	0	0	0	1	X
(7) brz	X	0	0	0	0
(11) jmadd	X	0	0	1	X
(15) bn	X	0	0	0	X
(24) sllv	1	0	0	0	0
(28) jspal	X	0	0	1	0

	AluOP	MemWrite	AluSRC	RegWrite
(5) balmn	00	0	1	1
(7) brz	X	0	X	0
(11) jmadd	00	0	0	1
(15) bn	X	0	X	0
(24) sllv	10	0	0	1
(28) jspal	X	1	X	0

2.2.Table 2:

Table 2 shows the values of all instructions for each signals that were added by us. Name of the signals and the values are represented as:

	MUX3 Signal	31 Signal	linkAdr Signal	29 Signal	PC+4 Signal	AddrMem Signal
(5) balmn	001	0	1	0	0	0
(7) brz	010	0	0	0	0	0
(11) jmadd	011	1	1	0	0	0
(15) bn	100	0	0	0	0	0
(24) sllv	000	0	0	0	0	0
(28) jspal	101	0	0	1	1	1

3.Additional Verilog Files:

To complete the design accurately, some extra Verilog files were required. Consequently, three extra Verilog files were added.

3.1.'mult8_to_1_32.v'

The first extra Verilog file that is created is 'mult8_to_1_32.v'. That file represents a 8 to 1 multiplexer.

```
1  module mult8_to_1_32(out, i0,i1,i2,i3,i4,i5,i6,i7,s0);
2  output [31:0] out;
3  input [31:0]i0,i1,i2,i3,i4,i5,i6,i7;
4  input [2:0]s0;
5  assign out = (s0 == 3'b000) ? i0 :
6               (s0 == 3'b001) ? i1 :
7               (s0 == 3'b010) ? i2 :
8               (s0 == 3'b011) ? i3 :
9               (s0 == 3'b100) ? i4 :
10              (s0 == 3'b101) ? i5 :
11              (s0 == 3'b110) ? i6 : i7;
12  endmodule
```

3.2.'new_component.v'

The other extra Verilog file that is created is 'new_component.v'. That file represents a new component which processes ALU operation according to the ALU control line values. This file created for using the values of n and z in the previous cycle.

```
1  module new_component(aluResult,z,n);
2  input [31:0] aluResult;
3  output z,n;
4  assign z = ~(|aluResult);
5  assign n = aluResult[31];
6  endmodule
```

3.3.'flipflop.v'

The last extra Verilog file that is created is 'flipflop.v'. That file stands for flip flop circuit. In this file, N stands for negative results and Z stands for zero results.

```
1  module flipflop (input d,
2  |               |      input clk,
3  |               |      output reg q);
4  |
5  |       always @ (negedge clk)
6  |               q <= d;
7  endmodule
8  |
```

4. Instructions:

4.1. balmn

opcode	rs	rt	immediate
6 bits	5 bits	5 bits	16 bits

Balmn is an I-type instruction

/processor/pc	4	8	16	20
/processor/dk	-1			
/processor/datmem	0 0 0 1 6 0 0 0 5 0 0 0 1 6 0 0 1 0 0 0 1 1 6 0 0 0 3 7 0 0 1 3 6 x x x x	0 0 0 1 6 0 0 0 5 0 0 0 1 6 0 0 1 0 0 0 1 1 6 0 0 0 3 7 0 0 1 3 6		
/processor/mem	0 0 0 0 0 1 1 6 3 2 9 6 6 7 0 9 7 6 0 0 0 9 6 6 8 0 1 3 0 0 0 3 2 0 - 9 6 0 2 0 - 1 - 1 ...	0 0 0 0 0 1 1 6 3 2 9 6 6 7 0 9 7 6 0 0 0 9 6 6 8 0 1 3 0 0 0 3 2 0 - 9 6 6 8 0		
/processor/dataa	0	-1		
/processor/datab	-1	0	12	35
/processor/out2	-1	9		13
/processor/out3	-1	8		12
/processor/out4	00000008	0000000d		00000014
/processor/sum	-1	8		12
/processor/extad	4128	9		13
/processor/adder1out	8	12		20
/processor/adder2out	16520	48		72
/processor/sextad	16512	36		52
/processor/writeData	-1	12		20
/processor/outPc	8	16		20
/processor/output5	X	16		20
/processor/output7	X	12		20
/processor/output15	8	12		20
/processor/inst31_26	0	24		
/processor/inst25_21	0	2		
/processor/inst20_16	1	3		4
/processor/inst15_11	2	0		
/processor/out1	2	3		4
/processor/out31	2	3		4
/processor/inst15_0	4128	9		13
/processor/instruc	00011020	60430009		6044000d
/processor/dpack	x	16		256
/processor/gout	2	2		
/processor/zout	0			

Instruction: 011000 00010 00011
0000000000001001 [if n is 1, branch to memory[\$r2 + 9] (r3 holds pc+4)] 0x60430009
\$r2 holds -1. And in memory[8], the value is 0x20. So
pc branches to 32

4.2.brz

opcode	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

Brz is a R-type instruction

/processor/pc	24	24	32
/processor/dk	-1		
/processor/datmem	0 0 0 16 0 0 5 0 0 0 16 0 0 1 0 0 0 1 16 0 0 0 37 0 0 1 36 x x x x	0 0 0 16 0 0 5 0 0 0 16 0 0 1 0 0 0 1 16 0	
/processor/mem	0 0 0 0 1 16 32 96 67 0 9 76 0 0 0 96 68 0 13 0 0 0 32 0 -96 0 20 -1 -1 -1..	0 0 0 0 1 16 32 96 67 0 9 76 0 0 0 96 68 0	
/processor/dataa	32	32	0
/processor/datab	0	0	
/processor/out2	0	0	
/processor/out3	0	0	
/processor/out4	0000001c	0000001c	00000024
/processor/sum	0	0	
/processor/extad	20	20	0
/processor/adder1out	28	28	36
/processor/adder2out	108	108	36
/processor/sextad	80	80	0
/processor/writeData	0	0	
/processor/outPc	32	32	36
/processor/output5	28	28	36
/processor/output7	32	32	0
/processor/output15	28	28	36
/processor/inst31_26	0	0	
/processor/inst25_21	5	5	0
/processor/inst20_16	0	0	
/processor/inst15_11	0	0	
/processor/out1	0	0	
/processor/out31	0	0	
/processor/inst15_0	20	20	0
/processor/instruc	00a00014	00a00014	00000000
/processor/dpack	16	16	
/processor/gout	0	0	2
/processor/zout	1		

Instruction: 000000 00101 00000 00000 00000 010100

(if z flag is 1 branch to address in \$r5) 0x00A00014

Register \$r5 holds the value 32, so pc becomes 32 instead of 28

4.3.jmadd

opcode	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

Jmadd is a R-type instruction

/processor/pc	4	4	12	16
/processor/clk	-1			
/processor/datmem	0 0 0 16 0 0 0 5 0 0 0 12 0 0 1 0 0 0 0 36 0 0 0 37 0 ...	0 0 0 16 0 0 0 5 0 0 0 12 0 0 1 0 0 0 0 36 0 0 0		
/processor/mem	0 0 0 0 0 34 0 63 0 0 0 0 0 1 16 32 100 0 0 6 0 0 0 0 ...	0 0 0 0 0 34 0 63 0 0 0 0 0 1 16 32 100 0 0 6 0 0		
/processor/dataa	-1	-1	0	
/processor/datab	9	9	-1	0
/processor/out2	9	9	-1	0
/processor/out3	8	8	-1	0
/processor/out4	00000008	00000008	00000010	000
/processor/sum	8	8	-1	0
/processor/extad	63	63	4128	6
/processor/adder1out	8	8	16	20
/processor/adder2out	260	260	16528	44
/processor/sextad	252	252	16512	24
/processor/writeData	8	8	-1	0
/processor/outPc	12	12	16	24
/processor/output5	X	X	16	16
/processor/output7	X	X	16	20
/processor/output15	8	8	16	24
/processor/inst31_26	0	0		25
/processor/inst25_21	1	1	0	
/processor/inst20_16	2	2	1	0
/processor/inst15_11	0	0	2	0
/processor/out1	0	0	2	0
/processor/out31	-1	-1	2	0
/processor/inst15_0	63	63	4128	6
/processor/instruc	0022003f	0022003f	00011020	640
/processor/dpack	12	12		36
/processor/gout	2	2		
/processor/zout	0			

Instruction: 000000 00001 00010 00000 00000 111111
 0x0022003F (Jump to address in memory [\$r1+\$r2]
 and in \$31 store the link address))

4.4.bn

opcode	address
6 bits	26 bits

Bn is a J-type instruction

/processor/pc	16	16	24	28
/processor/dk	-1			
/processor/datmem	0 0 0 16 0 0 0 5 0 0 0 12 0 0 1 0 0 0 0 36 0 0 0 37 0 ...	0 0 0 16 0 0 0 5 0 0 0 12 0 0 1 0 0 0 0 36 0 0 0 37 0 ...		
/processor/mem	0 0 0 0 0 34 0 63 0 0 0 0 0 1 16 32 100 0 0 6 0 0 0 0 ...	0 0 0 0 0 34 0 63 0 0 0 0 0 1 16 32 100 0 0 6 0 0 0 0 ...		
/processor/dataa	0	0	8	0
/processor/datab	0	0	35	0
/processor/out2	0	0	35	0
/processor/out3	0	0	0	
/processor/out4	00000014	00000014	0000001c	00
/processor/sum	0	0	0	
/processor/extad	6	6	18436	0
/processor/adder1out	20	20	28	32
/processor/adder2out	44	44	73772	32
/processor/sextad	24	24	73744	0
/processor/writeData	0	0	0	
/processor/outPc	24	24	28	36
/processor/output5	16	16	28	32
/processor/output7	20	20	8	0
/processor/output15	24	24	28	32
/processor/inst31_26	25	25	0	19
/processor/inst25_21	0	0	8	0
/processor/inst20_16	0	0	7	0
/processor/inst15_11	0	0	9	0
/processor/out1	0	0	9	0
/processor/out31	0	0	9	0
/processor/inst15_0	6	6	18436	0
/processor/instruc	64000006	64000006	01074804	4c
/processor/dpack	36	36	16	
/processor/gout	2	2	0	2
/processor/zout	1			

Instruction: 011001 000000000000000000000000110
 0x64000006 (If n flag is 1 branch to pseudo direct
 address with the value 6))

4.5.sllv

opcode	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

Sllv is a R-type instruction

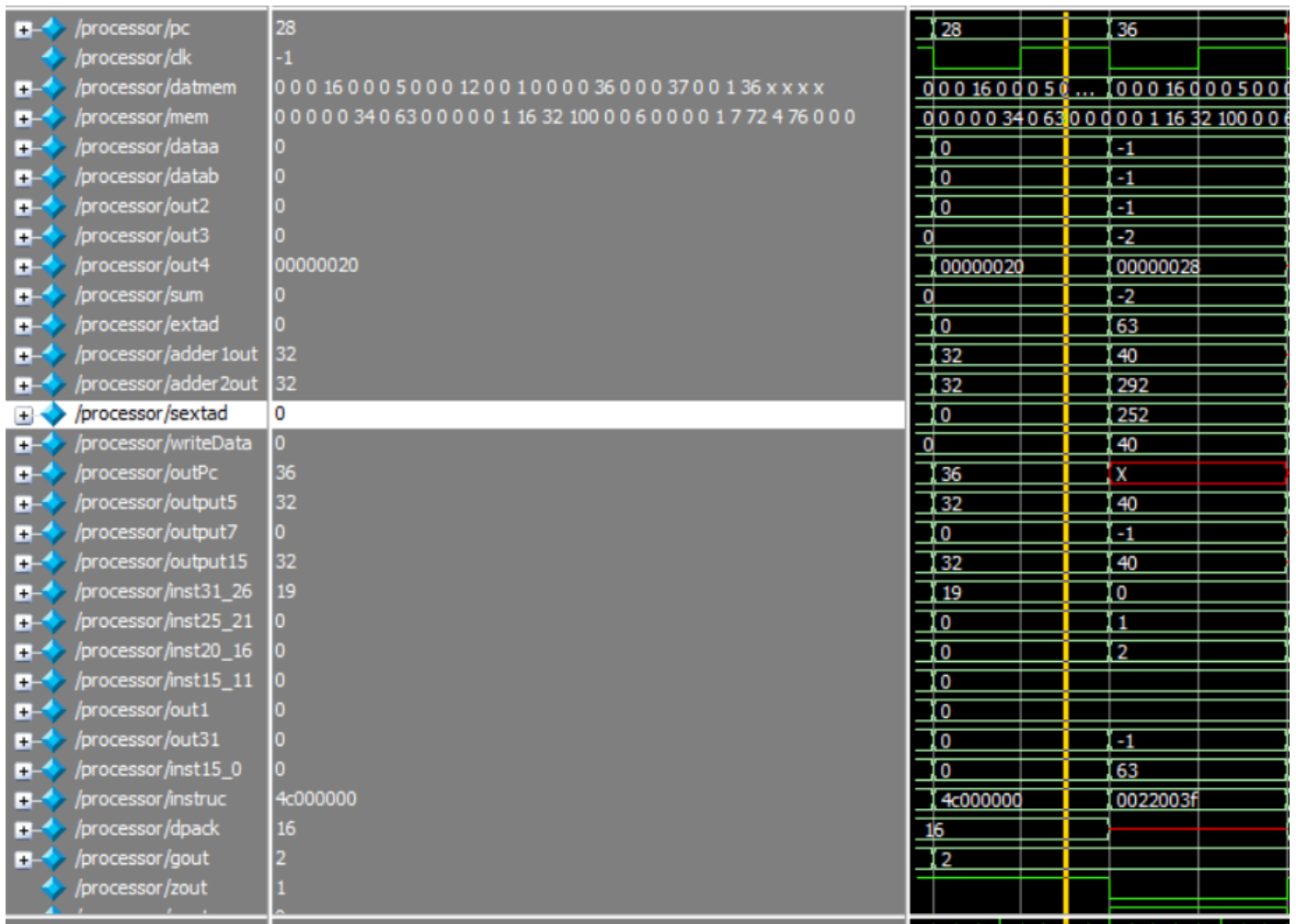
Register	Value (Hex)	Value (Binary)
0	0	00000000000000000000000000000000
1	1	00000000000000000000000000000001
2	1	00000000000000000000000000000001
3	0	00000000000000000000000000000000
4	0	00000000000000000000000000000000
5	0	00000000000000000000000000000000
6	0	00000000000000000000000000000000
7	0	00000000000000000000000000000000
8	0	00000000000000000000000000000000
9	0	00000000000000000000000000000000
10	0	00000000000000000000000000000000
11	0	00000000000000000000000000000000
12	0	00000000000000000000000000000000
13	0	00000000000000000000000000000000
14	0	00000000000000000000000000000000
15	0	00000000000000000000000000000000
16	0	00000000000000000000000000000000
17	0	00000000000000000000000000000000
18	0	00000000000000000000000000000000
19	0	00000000000000000000000000000000
20	0	00000000000000000000000000000000
21	0	00000000000000000000000000000000
22	0	00000000000000000000000000000000

Instruction: 000000 01000 00111 01001 00000 000100
[shift \$r7 to left by \$r8 and store result in \$r9]
0x01074804

4.6.jspal

opcode	rs	rt	immediate
6 bits	5 bits	5 bits	16 bits

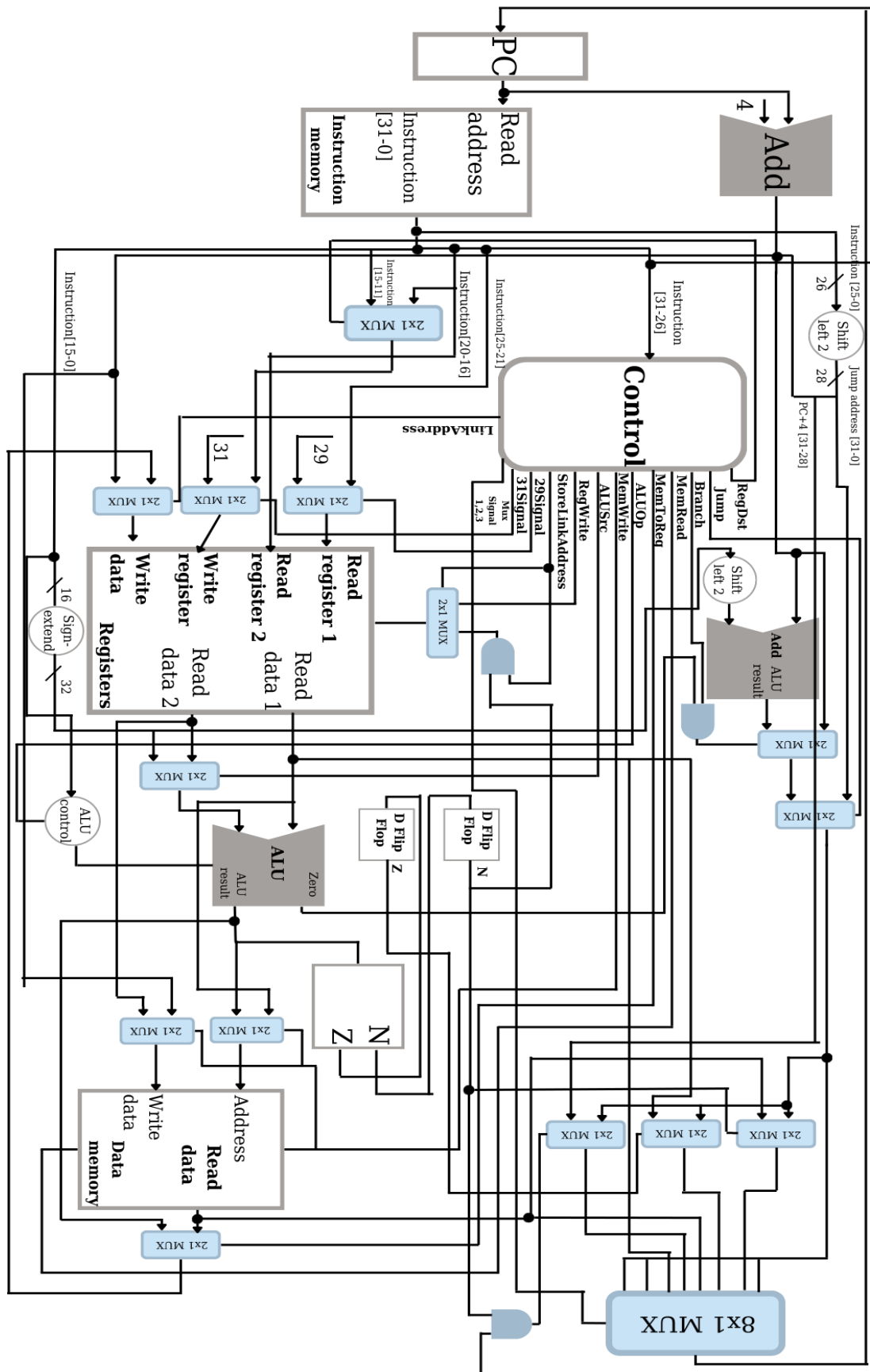
Jspal is an I-type instruction



\$r 29 holds the value 16, and in data memory[16], it has the value 36. Thus, our new pc becomes 36 and link address (32) is stored in data memory[16].

Instruction: 010011 00000 00000 00000 00000
 000000 0x4C000000 [branch to the address in mem[\$r29] and link address to that place in memory]

5. Single Cycle Datapath:



6.Conclusion & Summary:

In Conclusion; Every requirement that is necessary to complete this assignment, including; tables for each signal, additional Verilog files, instructions that were used for this assignment and a single-cycle datapath has been outlined in this document.

7.References:

- Template for Single Cycle Datapath
<https://i.stack.imgur.com/5d5XB.png>
- Commented MIPS files from Lokman ALTIN
- <https://www.chipverify.com/verilog/>