# EECS-361 COMPUTER ARCHITECTURE- I GROUP PROJECT1 - SINGLE CYCLE PROCESSOR

**GROUP5-MEMBERS** 

MATTHEW LENK YUE WANG NITHIN KRISHNAN

## **CONTENTS:**

- . Introduction
- . Design
- . Main Entities
- . Simulation
- . Results

#### Introduction:

The single cycle processor has the ability to execute one instruction in a single clock cycle. The various parts of an instruction like, instruction fetch, decode etc. are all performed during this single cycle of the clock. Hence, the duration of clock cycle required is equal to the time taken by the largest instruction of the instruction set (In our case it is the LW instruction).

#### Design:

The single cycle processor is realised with the help of ModelSim software, with the test programs loaded into the memory unit of the processor. It consists mainly of an Instruction Fetch Unit, a Register file, an Arithmetic and Logic Unit and a Memory Unit.

The instruction set for this processor contains the following instructions:

1. • arithmetic: add, addi, addu, sub, subu

2. • logical: and, or, sll

3. • data transfer: lw, sw

4. • conditional branch: beq, bne, bgtz, slt, sltu.

The different types of instructions in the MIPS architecture is

- 1. R-type instruction
- 2. I-type instruction
- 3. J-type instruction

The instruction fetch unit gets the new instruction from the Program Counter and decodes it.

The register file stores the input operands for the operation to be performed as well any data that has to be written back.

The ALU performs the required arithmetic or logic operation ,which is selected by an ALU control, and produces an output.

Finally, data is stored in the memory unit, which can then be written back to the register file depending on the instruction.

#### Main Entities:

**Instruction Fetch:** The instruction fetch unit is tasked with taking the instruction from memory and decoding the instruction into opcode, CPU control signals, type of instruction etc.

**Register File:** The register file stores the operands in its memory and forwards the data to the ALU for processing. The register file also has a 32 bit write back register for data to be written back to it from the memory depending on the instruction.

**Arithmetic and Logic Unit:** The arithmetic and logic unit takes in 2 operands and performs the required operation on them as specified by the opcode. The opcode generates the ALU control signals, which in turn control the operation performed by the ALU.

```
VHDL Implementation
      begin
```

```
--For choosing which overflow and carryOut signal to use
--The only two options for which operation it would come from
--is addition or subtraction.
--I check if its addition whose opcode is "010"
--Then I use that result as a selector for the muxes
--to choose whether to output the addOverflow and carryout or the subtraction ones
ctrl1: not gate port map(ctrl(0), o1);
ctrl2: not gate port map(ctrl(2), o2);
ctrl3: and gate port map(o1, o2, o3);
ctrl4: and_gate port map(o3, ctrl(1), addOrSub);
result<= tempResult;
--Multiplexers to choose which operation should be outputed.
muxResult: mux 4 generic map(n => 32)
             port map(
```

```
sel => ctrl.
addOp => resultAdd,
subOp => resultSub,
andOp => resultAnd,
orOp => resultOr,
sllOp => resultSll,
sltOp => resultSlt,
sltuOp => resultSltu,
Result => tempResult
);
```

```
muxCarryOut: mux
             port map(
                   sel => addOrSub,
                   src0 => carryOutSub,
                   src1 => carryOutAdd,
                   z => carryOut
                   );
muxOverflow: mux
             port map(
                   sel => addOrSub,
                   src0 => overflowSub,
                   src1 => overflowAdd.
                   z => overFlow
--calculating the zero value from the result that will be outputed of the ALU
zeroPort: zeroDetector port map(tempResult, zero);
--Each operation has its own entity
andOperation: and gate 32 port map(A, B, resultAnd);
orOperation: or gate 32 port map(A,B, resultOr);
addEntity: adder32 port map(A=>A, B=>B, carryIn=> '0', carryOut=>carryOutAdd,
overFlow=>overflowAdd, result=>resultAdd);
subEntity: sub32 port map(A=>A, B=>B, carryOut=>carryOutSub,
overFlow=>overflowSub, result=>resultSub);
resultSlt<= (0=> resultSub(31), others=>'0');
notCarry: not gate port map(carryOutSub, notCarryOutSub);
resultSltu<= (0=>notCarryOutSub, others=>'0');
sllEntity: SLLmod port map (A=>B, B=>A, result=>resultSll);
```

end architecture structural;

shamt(31 downto 5)<= (others=>'0');

**Memory:** The data from the ALU is written into the data memory. This result is stored there, or written back to the register file, depending on the instruction type. VHDL implementation:

```
begin
--clk inv: not gate port map(x => clk, z => clk n);
sram_map: sram
generic map(mem file => data mem file)
port map(
              cs => '1',
              oe => '1',
              we => we,
              addr => addr,
              din => data in,
              dout => temp data out);
data out<=temp data out;
end structural;
      Top-Level Entity: The top level entity(datapath), connects all the components
together.
VHDL IMplementation:
             begin
opcode<= inst(31 downto 26);--from IFU
Rt<=inst(20 downto 16);
Rs<=inst(25 downto 21);
Rd<=inst(15 downto 11);
Ra<=Rs;
Rb<=Rt:
immed<=inst(15 downto 0);</pre>
func<=inst(5 downto 0);
shamt(4 downto 0)<=inst(10 downto 6);</pre>
```

```
--Read data1<=opA;
opA<= Read data1;
11: instructMem generic map( mem file=>mem file) port map(address,inst);
12: main control port
map(opcode,RegDst,ALUSrc,MemtoReg,RegWr,MemWr,Branch,Jump,ExtOp,ALUop);-
-Getting main control signals from opcode
13: mux n generic map(n=> 5) port map(RegDst,Rt,Rd,Rw);-- selecting input for
register
14: register file port
map(clk,reset,RegWr,Ra,Rb,Rw,Write data,temp read,Read data2);-- output from
register file
15: signExtender port map(immed,signExtout);-- calculating extended output
16: ALU control port map(opcode, func,ALUop,ALUctr);-- calculating alu control signals
NOR0: nor gate port map(ALUctr(1),ALUctr(0),AC);
AND1: and gate port map(ALUctr(2),AC,Ensll);
MUX1: mux 32 port map(Ensll,temp_read,shamt,Read_data1);--temp_read is read
from data register and MUXd with shamt to produce read data1 which is given as opA
to ALU.
17: mux 32 port map(ALUSrc,Read data2,signExtout,opB);-- selecting operand B for
ALU operations
18: ALU port map(opA,opB,ALUctr,carryout,overflow,zero f,result);-- alu values
19: not gate port map(MemWr, MemRead); -- generating memory read signal for the
memory file
cs<='1':
110: data memory generic map(data mem file=>mem file) port
map(Read_data2,result,MemWr,Dout);--Writing data to memory
111: mux 32 port map(MemtoReg,result,Dout,Write data);-- selecting which data to
write back
112: adder32 port
Cplus4):
--I13: getJumpAddress port map(inst(25 downto 0),PCplus4(31 downto
28), Jump Address);
--I14: mux 32 port map(Jump,PCplus4,JumpAddress,nextAddress);
I13: and_gate port map(branch, zero f, branchCtrl );
114: adder32 port map(A=> PCplus4, B(31 downto 2) => signExtout(29 downto 0), B(1
downto 0) =>"00", carryln=>'0', result=> branchAddress );
```

I16: mux\_32 port map(branchCtrl, PCplus4, branchAddress, nextAddress);
I15: PC port map(nextAddress,address,clk);

-- Call PC+4 or New address based on jump.

end architecture structural;

**Main Control Signals:** The main control signals are generated with the help of the OPcode. These signals help select which data is to be provided as input and if data should be written back or read from.

The various control signals are:

- I. RegDst: Selects either a 5 bit operand or 16 bit immediate value depending on the type of instruction.
- II. ALUSrc: Selects 32 bit data from the register to perform ALU operations or selects 32 bit input to perform shift instructions.
- III. MemtoReg: Selects either the ALU output or data stored in the data memory depending on the instruction.
- IV. RegWr: If set, this signal enables writing of data on to the registers.
- V. MemWr: If set, this signals enable writing onto the data memory unit.
- VI. Branch: Helps select the address for the next instruction. If branch is taken, a new address is calculated. Otherwise, the next instruction is loaded to the Program Counter
- VII. ALUop: Controls the operation of the ALU.

VHDL Implementation:

--Find Instruction Type based on Opcode

```
--Rtype
r1: six_in_and_gate port map(nop5,nop4,nop3,nop2,nop1,nop0,inst(0));
--SLL
--OR Immediate
or1: six_in_and_gate port
map(nop5,nop4,opcode(3),opcode(2),nop1,opcode(0),inst(1));
--Load word
addi: six_in_and_gate port map(nop5, nop4, opcode(3), nop2, nop1, nop0, inst(8));
lw1: six_in_and_gate port
map(opcode(5),nop4,nop3,nop2,opcode(1),opcode(0),inst(2));
```

```
--Store Word
sw1: six in and gate port
map(opcode(5),nop4,opcode(3),nop2,opcode(1),opcode(0),inst(3));
-- Breanch Equal
beq1: six in and gate port map(nop5,nop4,nop3,opcode(2),nop1,nop0,inst(4));
bne1: six in and gate port map(nop5,nop4,nop3,opcode(2),nop1,opcode(0),inst(7));
bgtz1: six in and gate port
map(nop5,nop4,nop3,opcode(2),opcode(1),opcode(0),inst(6));
-- Jump
jmp1: six in and gate port map(nop5,nop4,nop3,nop2,opcode(1),nop0,inst(5));
--RegWrite
regwr1: or gate port map(inst(0),inst(1),temp0);
regwr2: or gate port map(inst(2),temp0,temp3);
regwr3: or gate port map(temp3, inst(8), RegWr);
--ALUSRC
alusrc1: or gate port map(inst(1),inst(2),temp1);
alusrc2: or gate port map(temp1,inst(3),temp2);
alusrc3: or gate port map(temp2, inst(8), ALUSrc);
--RegDst
RegDst<=inst(0);
--MemtoReg
MemtoReg<=inst(2);
--MemWrite
```

```
MemWr<=inst(3);</pre>
--Branch
----BEQ
--Branch(2)<=inst(4);
----BNE
--Branch(1)<=inst(7);
----BGTZ
--Branch(0)<=inst(6);
branchCheck1: or_gate port map (inst(4), inst(7), b1);
branchCheck2: or_gate port map(b1, inst(6), Branch);
--Jump
Jump<=inst(5);</pre>
--ExtOP
extop1: or_gate port map(inst(2),inst(3),ExtOp);
--ALUop
ALUop(2) \le inst(0);
ALUop(1) \le inst(1);
ALUop(0) \le inst(4);
Main Control Signal Values:
```

OPCODE	000000	001101	100011	101011	000100	000101	000110	000010
	R-TYP E	ORI	LW	SW	BEQ	BNE	BGTZ	JMP
RegDst	1	0	0	Х	Х	Х	Х	Х
ALUSrc	0	1	1	1	0	0	0	Х
MemtoReg	0	0	1	Х	Х	Х	Х	Х
MemWr	1	1	1	0	0	0	0	0
RegWr	0	0	0	1	0	0	0	0
Branch	0	0	0	0	1	1	1	0

Jump	0	0	0	0	0	0	0	1
ExtOP	Х	0	1	1	Х	X	X	X
ALUop	R-type	Or	Add	Add	Subt	Subt	Subt	XXX

**ALU Control Signals:** The ALU control signals or ALUop controls the operation of the ALu. It helps select which operation needs to be performed on the two input operands.

```
VHDL Implementation:
```

--finding bit 2 of ALUctr

```
a21: three in and gate port
map(inv ALUop(2),inv ALUop(1),ALUop(0),temp signals(0));
a22: four in and gate port
map(ALUop(2),inv func(2),func(1),inv func(0),temp signals(1));
a23: or gate port map(temp_signals(0),temp_signals(1),alsotemp(0));
a25: five in and gate port
map(ALUop(2),func(3),inv func(2),inv func(1),func(0),alsotemp(1));
a26: or gate port map(alsotemp(0),alsotemp(1),temp ALUctr(2));
-- Finding bit1 of ALUctr
a11: and gate port map(inv ALUop(2),inv ALUop(1),temp signals(2));
a12: three in and gate port map(ALUop(2),inv func(2),inv func(0),temp signals(3));
a13: or gate port map(temp_signals(2),temp_signals(3),alsotemp(2));
a14: five in and gate port
map(ALUop(2),inv func(3),func(2),func(1),inv func(0),alsotemp(3));
a15: or gate port map(alsotemp(2),alsotemp(3),temp ALUctr(1));
-- Finding bit0 of ALUctr
a01: and gate port map(inv ALUop(2),ALUop(1),temp signals(4));
a02: five in and gate port
map(ALUop(2),inv func(3),func(2),inv func(1),func(0),temp signals(5));
```

```
a03: five_in_and_gate port map(ALUop(2),func(3),inv_func(2),func(1),inv_func(0),temp_signals(6)); a04: or_gate port map(temp_signals(5),temp_signals(4),temp_signals(7)); a05: or_gate port map(temp_signals(6),temp_signals(7),alsotemp(4)); a06: five_in_and_gate port map(ALUop(2),inv_func(3),func(2),func(1),inv_func(0),alsotemp(5)); a07: or_gate port map(alsotemp(4),alsotemp(5),temp_ALUctr(0));
```

The ALUop is a three bit value. So, we can perform up to 7 distinct operations.

#### **ALU Control Signal Values:**

ALUop	1		Func				ALUctr			Operation
0	0	0	Х	Х	Х	Х	0	1	0	ADD
0	0	1	Х	Х	Х	Х	1	1	0	SUB
0	1	0	Х	Х	Х	Х	0	0	1	OR
1	Х	Х	0	0	0	0	0	1	0	ADD
1	Х	Х	0	0	1	0	1	1	0	SUB
1	Х	Х	0	1	0	0	0	0	0	AND
1	Х	Х	0	1	0	1	0	0	1	OR
1	Х	Х	1	0	1	0	1	1	1	SLT
1	Х	Х	1	0	0	0	0	1	0	ADDU
1	Х	Х	1	1	1	0	1	1	0	SUBU
1	Х	х	1	0	0	1	1	0	0	SLL
1	Х	Х	1	0	1	1	0	1	1	SLTU

## Simulation:

# **Unsigned Sum**

ſ <b>→</b>	Msgs					
// /datapath_tb_bills_branch/DP/dk	1					
/datapath_tb_bills_branch/DP/reset	0					
/datapath tb bills branch/DP/address	32'h0040003C	32'h00400020	132h00400024	132h00400028	32'h0040002C	I 32'h00400030
/datapath tb bills branch/DP/nextAddress	32'h00400040	(32h00400024	1 32 h00400028	32'h0040002C	32'h00400030	I 32'h00400034
/datapath tb bills branch/DP/PCplus4	32'h00400040	(32h00400024	32h00400028	32'h0040002C	32'h00400030	32'h00400034
/datapath tb bills branch/DP/temp read	32'h10000194	(32h00000000		32'h00001000	32'h10000000	
/datapath to bills branch/DP/inst	32'h20E70004	(32h00002820	32h20071000	32'h00E73C00	32'h00E03020	32'h20C60028
/datapath_tb_bills_branch/DP/Write_data	32'h10000198	(32h00000000	1 32'h00001000	32'h 10000000	32'h10000000	I 32'h 10000028
/datapath to bills branch/DP/Read data1	32'h 10000 194	(32h00000000		32'h00000010	32h10000000	
/datapath_tb_bills_branch/DP/Read_data2	32'h 10000 194	(32'h00000000		32'h00001000	32'h00000000	32'h10000000
/datapath_tb_bills_branch/DP/opcode	6'h08	(6'h00	I6h08	16'h00		I 6'h08
/datapath_tb_bills_branch/DP/Rw	5'h07	(5'h05	15h07		[5h06	I 5'h06
-🍫 /datapath_tb_bills_branch/DP/Ra	5'h07	(5'h00		(5h07		I 5'h06
/datapath_tb_bills_branch/DP/Rb	5'h07	(5'h00	5h07		(5h00	[5h06
/datapath_tb_bills_branch/DP/RegDst	0					
/datapath_tb_bills_branch/DP/ALUSrc	1					
/datapath_tb_bills_branch/DP/MemtoReg	0					
/datapath_tb_bills_branch/DP/RegWr	1					
/datapath_tb_bills_branch/DP/MemWr	0					
/datapath_tb_bills_branch/DP/MemRead	1	1				
/datapath_tb_bills_branch/DP/Jump	0					
/datapath_tb_bills_branch/DP/ExtOp	0					
/datapath_tb_bills_branch/DP/ALUop	3'h0	(3'h4	(3/h0	3h4		(3'h0
/datapath_tb_bills_branch/DP/Branch	0					
/datapath_tb_bills_branch/DP/ALUctr	3'h2	(3'h2		(3h4	3h2	
/datapath_tb_bills_branch/DP/opA	32'h 10000 194	(32'h00000000		32'h00000010	32h10000000	
// /datapath_tb_bills_branch/DP/opB	32'h00000004	32'h00000000	32h00001000	32'h00001000	32'h00000000	132'h00000028
/datapath_tb_bills_branch/DP/immed	16'h0004	16'h2820	16h1000	16'h3C00	16'h3020	16'h0028
/datapath_tb_bills_branch/DP/result	32'h 10000 198	32'h00000000	32h00001000	32'h10000000	32h10000000	32h10000028
/datapath_tb_bills_branch/DP/Dout	32'hXXXXXXXXXX	32'hXXXXXXXXX		32'h00	00000F	321
/datapath_tb_bills_branch/DP/carryout	0					
/datapath_tb_bills_branch/DP/overflow	0					
/datapath_tb_bills_branch/DP/zero_f	0					
√ /databath tb bills branch/DP/func	6'h04	6h20	6'h00		6h20	6h28
■ ● Now	8240 ns			40 ns		30 ns

	[32hF   32h0	32'h00000000	32h10000000	32'h10000004		32'h0000000FF	32h1000
2'h 10000000	32'h00000000	32'h0000000F	32h10000000	32'h10000028	32'h0000000F	32'h000000F0	32h1000
h08	6h23	I 6'h00	6'h08	[6'h05	6h23	(6'h00	6'h08
h06	5'h04	[5h05	5'h07	5'h06	5'h04	5'h05	5'h07
h06	5h07	15h05	5h07			5h05	5'h07
h06	[5h04		5h07	[ 5'h06	I 5'h04		5'h07
h0		3'h4	I 3'h0			[3h4	3'h0
		32'h00000000	32h10000000	I 32'h 10000004		32'h0000000F	32h1000
2'h00000028	32'h00000000	32'h0000000F	32h00000004	32'h10000028	32h00000000		32h0000
6'h0028	16'h0000	16'h2821	16h0004	16'hFFFC	I 16 h0000	16h2821	16ħ0004
2'h 10000028	32h10000000	32'h0000000F	32h10000004	32/h2000002C	32h10000004	THE RESERVE AND ADDRESS OF THE PERSON NAMED IN COLUMN TWO IS NOT THE PERSON NAMED IN COLUMN TWO IS NAMED IN COLUMN TWO IS NOT THE PERSON NAMED IN COLUMN TWO IS NAMED IN COLUMN TWO IS N	32h1000
32'hFF	FFFFFF X 32'h000	00000F 32'hXX	XXXXXXX 32h00	0000F0 32'hXX	XXXXXX 321	000000F0 32'hXX	0000000

## Bills Branch

I*	Msgs				
// /datapath tb sort correct branch/DP/clk	0				
// /datapath tb sort correct branch/DP/reset	0				
/datapath tb sort correct branch/DP/address	32'h0040003C	32'h00400020	I 32h00400024	I 32'h00400028	I 32'h0040002C
/datapath_tb_sort_correct_branch/DP/nextAddress	32'h00400048	32'h00400024	32/h00400028	32'h0040002C	32'h00400030
/datapath_tb_sort_correct_branch/DP/PCplus4	32'h00400040	32'h00400D24	32'h00400028	32'h0040002C	32'h00400030
/datapath_tb_sort_correct_branch/DP/temp_read	32'h00000000X	32'h00000000			32'h00001000
/datapath_tb_sort_correct_branch/DP/inst	32h10850002	32'h20050001	32h20060064	32'h20021000	32'h00421400
/datapath_tb_sort_correct_branch/DP/Write_data	32*hXXXXXXXXX	32'h00000001	132/h00000064	I 32'h0000 1000	32'h10000000
/datapath_tb_sort_correct_branch/DP/Read_data1	32'h00000000X	32'h00000000			32'h00000010
/datapath_tb_sort_correct_branch/DP/Read_data2	32'h00000001	32'h00000000			32'h00001000
/datapath_tb_sort_correct_branch/DP/signextout	32h00000002	32'h00000001	X 32'h00000064	32'h00001000	32'h00001400
- /datapath_tb_sort_correct_branch/DP/opcode	6'h04	6'h08			(6'h00
/datapath_tb_sort_correct_branch/DP/Rw	5'h05	5'h05	5'h06	[5h02	
/datapath_tb_sort_correct_branch/DP/Ra	5'h04	5'h00			5'h02
/datapath_tb_sort_correct_branch/DP/Rb	5'h05	5'h05	5h06	[5h02	
/datapath_tb_sort_correct_branch/DP/RegDst	0				
/datapath_tb_sort_correct_branch/DP/ALUSrc	0				
/datapath_tb_sort_correct_branch/DP/MemtoReg	0				
/datapath_tb_sort_correct_branch/DP/RegWr	0				
/datapath_tb_sort_correct_branch/DP/MemWr	0				
/datapath_tb_sort_correct_branch/DP/MemRead	1				
/datapath_tb_sort_correct_branch/DP/Jump	0				
/datapath_tb_sort_correct_branch/DP/ExtOp	0				
/datapath_tb_sort_correct_branch/DP/ALUop	3h1	3'h0			3'h4
/datapath_tb_sort_correct_branch/DP/Branch	1				
/datapath_tb_sort_correct_branch/DP/ALUctr	3'h6	3h2			3'h4
/datapath_tb_sort_correct_branch/DP/opA	32'h0000000X	32'h00000000			32'h00000010
/datapath_tb_sort_correct_branch/DP/op8	32'h00000001	32'h00000001	32h00000064	32'h00001000	32'h00001000
/datapath_tb_sort_correct_branch/DP/immed	16'h0002	16'h0001	16'h0064	16'h1000	16'h1400
/datapath_tb_sort_correct_branch/DP/result	32°hXXXXXXXXX	32'h00000001	32h00000064	32'h00001000	32h10000000
/datapath_tb_sort_correct_branch/DP/Dout	32'hXXXXXXXXX	32'hXXXXXXXXXX			32'h00
/datapath_tb_sort_correct_branch/DP/carryout	U				
/datapath_tb_sort_correct_branch/DP/overflow	0				
/datapath tb sort correct branch/DP/zero f	U				
Now	7480 ns	18	20 ns	40 ns	60 ns
Cursor 1	0 ns	0 ne			

32h0040030 32h0040034 32h0040038 32h004003C 32h0040040 32h0040044 32h00400 32h0001000 32h1000000 32h1000000 32h0000064 32h0000064 32h0000064 32h0000064 32h000421400 32h20470028 32h30000 32h0000000 32h0000000 32h0000000 32h0000000 32h0000000 32h0000000 32h10000000 32h0000000 32h00000000 32h000000000 32h00000000 32h000000000 32h00000000 32h000000000 32h00000000 32h00000000 32h00000000 32h00000000 32h0000000000	32h0040030	2'h00 <del>'4</del> 00	0.00	V poli po son		32'h004000		32'h004000		32'h00400		32'h004000			044
32h00001000	32h00001000   32h10000000   32h00000000   32h000000000   32h00000000   32h00000000   32h000000000   32h00000000   32h00000000   32h00000000   32h00000000   32h00000000   32h000000000   32h00000000   32h00000000   32h00000000   32h00000000   32h00000000   32h00000000   32h00000000   32h000000000   32h00000000   32h00000000   32h00000000   32h00000000   32h000000000   32h000000000   32h000000000   32h000000000   32h000000000000000000000000000000000000			32h004000		32'h004000		32'h004000		32'h00400		32'h00400		32'h00400	
32h00421400	32h00421400 32h20470028 32h8C430000 32h0000202 32h10850002 32h00003022 32h8C400 32h10000000 32h10000028 32h0000000 32h00000000 32h100000000 32h100000000 32h100000000 32h100000000 32h00000000 32h000000000 32h00000000 32h00000000 32h00000000 32h00000000 32h00000000 32h00000000 32h00000000 32h00000000 32h00000000 32h000000000 32h00000000 32h0000000000					32h004000	038								
32h10000000	32h10000000					201 22 121									
32h00000010	32h00000010 32h10000000 32h0000000 32h0000000 32h00000000 32h0000000 32h00000000 32h0000000000														-
32h0000100	32h0000100					32'hXX	32'h00								
32h00001400	32h00001400	*******													
6h00   6h08   6h23   6h00   6h04   6h00   6h28     5h07   5h03   5h04   5h05   5h06   5h00     5h02   5h07   5h03   5h06   5h00     5h07   5h03   5h06   5h05   5h06   5h00     5h07   5h03   5h05   5h05   5h03   5h00     3h00   3h10   3h1   3h1   3h4   3h0     3h4   3h2   3h7   3h6   3h2     3h2   3h7   3h6   3h2     3h2   3h7   3h6   3h2     3h2   3h2     3h2   3h2     3h2   3h2     3h2   3h2     3h2   3h2     3h2   3h2     3h2   3h2     3h2   3h2     3h2   3h2     3h2   3h2     3h2   3h2     3h3   3h3     3h3   3h3     3h4   3h2     3h5   3h2     3h5   3h2     3h2     3h3   3h3     3h3   3h3     3h4   3h2     3h5     3h5     3h5     3h6   3h2     3h7     3h6     3h7     3h6     3h7     3h6     3h7     3h8     3h9     3h9     3h9     3h9     3h0     3h0	6h00 6h08 6h23 6h00 6h04 6h00 6h28    Sh07   Sh03   Sh04   Sh05   Sh06   Sh00   Sh07   Sh03   Sh06   Sh04   Sh06   Sh00   Sh07   Sh03   Sh06   Sh04   Sh06   Sh00   Sh07   Sh03   Sh05   Sh03   Sh00   Sh07   Sh03   Sh05   Sh03   Sh00   Sh07   Sh03   Sh00   Sh05   Sh03   Sh00   Sh00														
	Sh07		400				000		)2A		002		)22		000
5h02         5h06         5h04         5h06         5h02           5h07         5h03         5h05         5h03         5h00           3h4         3h0         3h4         3h1         3h4         3h0           3h4         3h2         3h7         3h6         32h0000064         32h00000064         32h000000064         32h00000064         32h00000	5h02         5h06         5h04         5h06         5h02           5h07         5h03         5h05         5h03         5h00           3h4         3h0         3h4         3h1         3h4         3h0           3h4         3h2         3h7         3h6         3h2         3h10000000         32h00000064         32h00000064         32h0000000         32h00000000         32h000000000         32h000000000         32h00000000         32h0000	n00													
Sh07	Sh07			5h07		5'h03									
3h4 3h0 3h4 3h1 3h4 3h0  3h4 3h2 3h7 3h6 3h2  32h0000010 32h10000000 32h0000000 32h0000000 32h0000000 32h0000000 32h00000000 32h000000000 32h00000000 32h000000000 32h0000000000	3h4 3h0 3h4 3h1 3h4 3h0  3h4 3h2 3h7 3h6 3h2  32h0000010 32h0000000 32h00000000 32h00000000 32h00000000 32h00000000 32h000000000 32h0000000000	n02						5'h06							
3h4 3h2 3h7 3h6 3h2 32h00000010 32h00000000 32h00000000 32h00000000 32h00000000 32h00000000 32h000000000 32h000000000 32h0000000000	3h4 3h2 3h7 3h6 3h2 32h0000010 32h10000000 32h00000000 32h0000000 32h0000000 32h0000000 32h0000000 32h0000000 32h10000000 32h10000000 32h10000000 32h10000000 32h10000000 32h10000000 32h10000000 32h100000000 32h100000000 32h100000000 32h100000000 32h100000000 32h1000000000000000000000000000000000000			15h07		5'h03				15h05		I 5'h03		15h00	
3h4 3h2 3h7 3h6 3h2 32h0000010 32h0000000 32h0000000 32h0000000 32h00000000 32h00000000 32h00000000 32h00000000 32h00000000 32h00000000 32h00000000 32h000000000 32h00000000 32h00000000 32h00000000 32h00000000 32h00000000 32h00000000 32h00000000 32h00000000 32h00000000 32h00000005A 32h00000000	3h4 3h2 3h7 3h6 3h2 32h0000010 32h10000000 32h00000000 32h00000000 32h00000000 32h00000000 32h00000000 32h00000000 32h00000000 32h00000000 32h0000000 32h0000000 32h0000000 32h0000000 32h000000 32h0000000 32h0000000 32h0000000 32h0000000 32h10000000 32h10000000 32h10000000 32h10000000 32h10000000 32h100000000 32h100000000 32h100000000 32h100000000 32h100000000 32h1000000000 32h1000000000000000000000000000000000000							المراكاة							
3h4 3h2 3h7 3h6 3h2 32h0000010 32h0000000 32h0000000 32h0000000 32h00000000 32h00000000 32h00000000 32h00000000 32h00000000 32h00000000 32h00000000 32h000000000 32h00000000 32h00000000 32h00000000 32h00000000 32h00000000 32h00000000 32h00000000 32h00000000 32h00000000 32h00000005A 32h00000000	3h4 3h2 3h7 3h6 3h2 32h0000010 32h10000000 32h00000000 32h00000000 32h00000000 32h00000000 32h00000000 32h00000000 32h00000000 32h00000000 32h0000000 32h0000000 32h0000000 32h0000000 32h000000 32h0000000 32h0000000 32h0000000 32h0000000 32h10000000 32h10000000 32h10000000 32h10000000 32h10000000 32h100000000 32h100000000 32h100000000 32h100000000 32h100000000 32h1000000000 32h1000000000000000000000000000000000000														
3h4 3h2 3h7 3h6 3h2 32h00000010 32h00000000 32h00000000 32h00000000 32h00000000 32h00000000 32h000000000 32h000000000 32h0000000000	3h4 3h2 3h7 3h6 3h2 32h0000010 32h10000000 32h00000000 32h0000000 32h0000000 32h0000000 32h0000000 32h0000000 32h10000000 32h10000000 32h10000000 32h10000000 32h10000000 32h10000000 32h10000000 32h100000000 32h100000000 32h100000000 32h100000000 32h100000000 32h1000000000000000000000000000000000000														
3h4 3h2 3h7 3h6 3h2 32h00000010 32h00000000 32h00000000 32h00000000 32h00000000 32h00000000 32h000000000 32h000000000 32h0000000000	3h4 3h2 3h7 3h6 3h2 32h0000010 32h10000000 32h00000000 32h0000000 32h0000000 32h0000000 32h0000000 32h0000000 32h10000000 32h10000000 32h10000000 32h10000000 32h10000000 32h10000000 32h10000000 32h100000000 32h100000000 32h100000000 32h100000000 32h100000000 32h1000000000000000000000000000000000000														
3h4 3h2 3h7 3h6 3h2 32h0000010 32h0000000 32h0000000 32h0000000 32h00000000 32h00000000 32h00000000 32h00000000 32h00000000 32h00000000 32h00000000 32h000000000 32h00000000 32h00000000 32h00000000 32h00000000 32h00000000 32h00000000 32h00000000 32h00000000 32h00000000 32h00000005A 32h00000000	3h4 3h2 3h7 3h6 3h2 32h0000010 32h10000000 32h00000000 32h00000000 32h00000000 32h00000000 32h00000000 32h00000000 32h00000000 32h00000000 32h0000000 32h0000000 32h0000000 32h0000000 32h000000 32h0000000 32h0000000 32h0000000 32h0000000 32h10000000 32h10000000 32h10000000 32h10000000 32h10000000 32h100000000 32h100000000 32h100000000 32h100000000 32h100000000 32h1000000000 32h1000000000000000000000000000000000000														
3h4 3h2 3h7 3h6 3h2 32h00000010 32h00000000 32h00000000 32h00000000 32h00000000 32h00000000 32h000000000 32h000000000 32h0000000000	3h4 3h2 3h7 3h6 3h2 32h0000010 32h10000000 32h00000000 32h0000000 32h0000000 32h0000000 32h0000000 32h0000000 32h10000000 32h10000000 32h10000000 32h10000000 32h10000000 32h10000000 32h10000000 32h100000000 32h100000000 32h100000000 32h100000000 32h100000000 32h1000000000000000000000000000000000000														
3h4 3h2 3h7 3h6 3h2 32h00000010 32h00000000 32h00000000 32h00000000 32h00000000 32h00000000 32h000000000 32h000000000 32h0000000000	3h4 3h2 3h7 3h6 3h2 32h0000010 32h10000000 32h00000000 32h0000000 32h0000000 32h0000000 32h0000000 32h0000000 32h10000000 32h10000000 32h10000000 32h10000000 32h10000000 32h10000000 32h10000000 32h100000000 32h100000000 32h100000000 32h100000000 32h100000000 32h1000000000000000000000000000000000000														
3h4 3h2 3h7 3h6 3h2 32h00000010 32h00000000 32h00000000 32h00000000 32h00000000 32h00000000 32h000000000 32h000000000 32h0000000000	3h4 3h2 3h7 3h6 3h2 32h0000010 32h10000000 32h00000000 32h0000000 32h0000000 32h0000000 32h0000000 32h0000000 32h10000000 32h10000000 32h10000000 32h10000000 32h10000000 32h10000000 32h10000000 32h100000000 32h100000000 32h100000000 32h100000000 32h100000000 32h1000000000000000000000000000000000000														
32h00000010	32h00000010 32h10000000 32h00000064 32h00000064 32h00000000 32h000000000 32h000000000 32h000000000 32h000000000 32h0000000000	14		X3"h0				3h4		X3h1		[3h4		3'h0	
32h00000010	32h00000010 32h10000000 32h00000064 32h00000064 32h0000000 32h00000000 32h100000000 32h1000000000000000000000000000000000000														
32h00001000	32h00001000 32h00000028 32h00000000 32h0000000A 32h00000001 32h000000A 32h00000 16h1400 16h0028 16h0000 16h202A 16h0002 16h3022 16h0000 32h10000000 32h10000028 32h10000000 32h00000000 32hFFFFFFFF 32h000005A 32h10000														
16h1400         16h0002         16h0002         16h3022         16h0000           32h1000000         32h1000000         32h1000000         32h7000000         32h7000000	16h1400         16h0028         16h0000         16h202A         16h0002         16h3022         16h3000           32h1000000         32h1000000         32h1000000         32hFFFFFFFF         32h0000005A         32h10000														
32h10000000	32h10000000   32h10000028   32h10000000   32h00000000   32hFFFFFFF   32h000005A   32h10000		000				000		)0A		001		00A		
					_										
[32/h0000000A ] 32/h0000000X [32/h0000000A ] 32/h0000000X	32h000000A 32h000000X (32h000000A 32h000000X										FF	32'h000000	)5A	32h10000	
			32'h0000	000A	32'hXXXXXX	OXX	32'h0000'	000A	32'hXXXXX	XXXX					3

#### Sort Branch

<b>≨</b> 1 •	Msgs									
/tb_sortbranch/DP/mem_file	sort_correcte	sort corrected by	anch.dat							
/tb_sortbranch/DP/clk	1									
/tb_sortbranch/DP/reset	0									
■- 🍫 /tb_sortbranch/DP/address	32'h0040003C	32'h00400020	32'h00400	024	32h00400	028	32'h00400	02C	32'h00400	030
→ /tb_sortbranch/DP/nextAddress	32'h00400040	(32'h00400024	32'h00400	28	32'h00400	02C	32'h00400	030	32'h00400	034
- /tb_sortbranch/DP/PCplus4	32'h00400040	32'h00400024	32'h00400	28	32'h00400	02C	32'h00400	030	32'h00400	034
	32'h00000009	(32'h00000000	32'h00001	000	32h10000	000				
- /tb_sortbranch/DP/inst	32'h00E1302A	(32h20021000	32h00421	400	32h20440	024	32h20450	028	32'h8C470	000
+ /tb_sortbranch/DP/Write_data	32'h00000000X	(32'h00001000	32h10000	000	32'h10000	024	32'h10000	028	32'hXX	32h00.
+ /tb_sortbranch/DP/Read_data1	32'h000000009	(32h00000000	32'h00000	10	32'h10000	000				
+ /tb_sortbranch/DP/Read_data2	32"hXXXXXXXXX	(32'h00000000	132h00001	000	32'h00000	000				
	32'h0000302A	(32'h00001000	I 32'h00001	100	32'h00000	024	32'h00000	028	32'h00000	000
- /tb_sortbranch/DP/opcode	6'h00	(6'h08	16'h00		6'h08				6'h23	
	5'h06	(5'h02			5h04		5'h05		5'h07	
- /tb_sortbranch/DP/Ra	5h07	(5'h00	[5h02							
	5'h01	(5'h02			5'h04		5'h05		5'h07	
/tb_sortbranch/DP/RegDst	1									
/tb_sortbranch/DP/ALUSrc	0									
/tb_sortbranch/DP/MemtoReg	0									
/tb_sortbranch/DP/RegWr	1									
/tb_sortbranch/DP/MemWr	0									
/tb_sortbranch/DP/MemRead	1									
/tb_sortbranch/DP/Jump	0									
/tb_sortbranch/DP/ExtOp	0									
- /tb_sortbranch/DP/ALUop	3'h4	(3'h0	I 3'h4		3ħ0					
/tb_sortbranch/DP/Branch	0									
	3h7	(3h2	I 3'h4		3h2					
→ /tb sortbranch/DP/opA	32'h000000009	(32'h00000000	I 32'h00000	10	32h10000	000				
+ /tb sortbranch/DP/op8	32'hXXXXXXXXX	(32'h00001000	32'h00001	000	32'h00000	024	32'h00000	028	32'h00000	000
T- /tb_sortbranch/DP/immed	16'h302A	(16'h1000	I 16'h1400		16'h0024		16'h0028		16'h0000	
	32'h00000000X	(32/h00001000	32'h 10000	000	32'h10000		32'h10000	028	32'h10000	
	32'hxxxxxxxx	32'hxxxxxxxxx		32'h00000		32'h00000			XX	
/bb sortbranch/DP/carryout	U									
/tb sortbranch/DP/overflow	U									
¥₹ ® Now	8240 ns	75	20 ns		ns	60	ns	80	ns	11111111
Cursor 1	5545 ns		- I I I	- 10		- 00		00	-	

													H
2'h00400	030	32'h004000	34	32'h00400	038	32'h00400	03C X	32'h00400	040	32'h00400	050	32'h00400	05
2'h00400	034	32'h004000	38	32'h00400	03C	32'h00400	040	32'h00400	050	32'h00400	054	32'h00400	03
2'h00400	034	32'h004000	38	32'h00400	03C	32'h00400	040	32'h00400	044	32'h00400	054	32'h00400	Ď:
				32'h10000	004	32'h00000	009	32'h00000	001	32'h10000	004	32'h10000	įοc
2'h8C470	000	32'h204300	004	32h8C610	000	32'h00E13	02A	32'h1CC00	003	32'h20630	004	32h1465F	Ħ
2'hXX	32'h00	32'h100000	004	32'h00000	00A	32'h00000	001	32'h00000	001	32'h10000	800	32'h20000	o:
				32h10000	004	32'h00000	009	32'h00000	001	32'h10000	004	32h10000	ΙÓC
						32'h00000	00A	32'h00000	000	32'h10000	004	32h10000	Ď.
32'h00000	000	32'h000000	004	32'h00000	000	32'h00003	02A	32'h00000	003	32'h00000	004	32hFFFFF	i
h23		6°h08		6'h23		6'h00		6°h07		6'h08		6'h05	i
5'h07		5h03		5'h01		5'h06		5'h00		5'h03		5'h05	i
				5'h03		5'h07		5h06		5'h03			ī
h07		5'h03		5'h01				5'h00		5'h03		5'h05	i
													ı
													i
													ı
													П
						3'h4		3'h0					t
												90	ī
						3'h7		3'h2					t
				32h10000	004	32'h00000	009	32'h00000	001	32'h10000	004	32'h10000	Ιοc
32'h00000	000	32'h000000	004	32'h00000	000	32'h00000	00A	32'h00000	000	32'h00000		32'h10000	_
16'h0000		16'h0004		16'h0000		16'h302A		16'h0003		16'h0004		16'hFFF8	i
32'h10000	000	32'h10000	004	32'h10000	004	32'h00000	001	32'h00000	001	32'h10000	008	32'h20000	0.
Х	32'h00000	009	32'h00000	00A			32'hxxxxxx	OOX			32'h000000	008	ı
													ı

## Results:

Our three programs all appeared to be functioning all though the became stuck in an infinite loop which we were not able to figure out in time. We received the correct solutions albeit the programs would not exit at the appropriate time.