**ARM CPU SIMULATOR**

This project is the design of a single cycle (non-pipelined) processor, which is capable of performing basic arithmetic, logic and data operations. It is based on ARM 64-bit architecture, with 32 registers of 64-bits wide with instruction lengths of 32-bits each. The project is designed in such a way that the instruction memory, data memory and register memory are located outside the CPU as separate functional units and it is possible to verify each functional units separately. In the fully functional operation all the individual modules will the co-operated. The instructions needs to be stored in the instruction memory and Program Counter (PC) will be incremented by CPU with the clock. Depend on the instruction the CPU will generate corresponding control signals to control different operations and to access other modules

The assembly instructions supported by the CPU are

LDUR: Load RAM into Register

STUR: Store Register content into RAM

ADD: Add Registers

SUB: Subtract Registers

ORR: Bit-wise OR Register

AND: Bit-wise AND Register

CBZ: Conditional Jump (when the value in Register is zero)

B: Unconditional (arbitrary) Jump

The pattern of generation of machine instructions from assembly instructions is shown in the below table.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Instruction** | **Example** | **Op-code (31- 21)** | **READ\_REG\_2**  **(20 to 16)** | **15 to 10** | **READ\_REG\_1**  **(9 to 5)** | **WRITE\_REG (4 to 0)** |
| LDUR | LDUR r2, [r10] | 1111 1000 010 | 0 0000 | 0000 00 | 0 1010- R10 | 0 0010 -R2 |
| STUR | STUR r4, [r7] | 1111 1000 010 | 0 0000 | 0000 00 | 0 0111 - R7 | 0 0100 -R4 |
| ADD | ADD r5, r3, r2 | 1000 1011 000 | 0 0010 -R2 | 0000 00 | 0 0011 -R3 | 0 0101- R5 |
| SUB | SUB r4, r3, r2 | 1100 1011 000 | 0 0010 -R2 | 0000 00 | 0 0011 -R3 | 0 0100 -R4 |
| AND | AND r7, r2, r3 | 1000 1010 000 | 0 0011 -R3 | 0000 00 | 0 0010 -R2 | 0 0111 -R7 |
| ORR | ORR r6, r2, r3 | 1010 1010 000 | 0 0011- R3 | 0000 00 | 0 0010 -R2 | 0 0110- R6 |
| STUR | STUR r4, [r7,] | 1111 1000 000 | 0 0000 | 0000 00 | 0 0111- R7 | 0 0100-R4 |
| LDUR | LDUR r2, [r10, #1] | 1111 1000 010 | 0 0000 | 0000 01 | 0 1010- R10 | 0 0010 -R2 |
| STUR | STUR r4, [r7, #1 ] | 1111 1000 000 | 0 0000 | 0000 01 | 0 0111 - R7 | 0 0100 -R4 |

**TEST BENCH MODULES DESIGNED**

1. **Data Memory module (TEST2.v)**

This module will exclusively test the functionality of the memory module. The data memory module is taken as an independent functional unit and its operation is only tested. The steps in which the test bench is designed is described below

* 1. 64 bit random values are generated and written into the data memory by setting the proper control signals. At the same time the written values and its address are stored in a reference array. Here the writing address is generated sequentially from 0 to 31.
  2. Next the control signals are set to read the contents from the memory. The address to be read from is generated randomly and this code runs in a loop to cover until the generated random values cover all the addresses. The value red is compared with one stored in the reference array and validated.

1. **Register module (TEST3.v)**

This module will test the functionality of the register module only. The register memory module is taken as an independent functional unit and its operation is only tested. The steps in which the test bench is designed is described below

* 1. The register is 64 bit wide and this data is generated using random function. The register name is also generated randomly. The register and the values written to a reference file and it is stored in disk.
  2. Then the registers are red sequentially and the red data is compared with the reference file data and validated.

1. **Instruction Memory module (TEST1.v)**

This module tests the functionality of the instruction memory module. The instruction memory is alone is considered as a separate functional unit and its operation is verified.

* 1. Here some sample instructions are stored directly into the instruction memory module and the same instructions are stored in a file as well for reference.
  2. The instruction memory is red sequentially by incrementing PC (Program Counter) manually in the test bench and the value is compared with reference file and validation is done.

1. **CPU Testing- Instructions from file (CPU\_TEST.v)**

Here the overall functionality of the processor is validated. This is done by loading instructions from file and checking whether it is getting executed. Here the results of this instructions are not automatically validated, only the end to end flow is evaluated. The results can be manually validated using GUI.

1. **CPU Testing- ALU (CPU\_TEST1.v)**

In this techbench the ALU of the processor is tested. All the ALU instructions such as ADD, SUB, ORR & AND are tested regressively. A reference file is created using MATLAB which contain different operands and the results of this operands with each of the ALU instructions. Then in the testbench a pair of operands are red and all the ALU operations are applied. The results of all the ALU instructions are validated against the results stored in the reference file. The processor is designed in such a way that all the operands should be in registers. So after reading operands it is stored in the registers, and the selection of the registers is done randomly. Verilog constructors such as tasks, file reading etc are used in this testbench.

1. **CPU Testing- End to End Testing (CPU\_TEST2.v)**

In this module the end to end functionality of the processor is tested. An interactive test environment is developed in which the assembly instructions, operands and expected results are typed in a file on the go, and testbench is executed and the results are validated. A C++ utility will convert the assembly instructions into machine code and this will be written into a file. The test bench will read this file with the operands and instructions are executed. Then the results are cross checked with the one stored in the file.

This module adds the flexibility to the testing process as the verifying engineer can dynamically add the instructions to be tested. Also he can create a series of instructions and can check for the program flow.

1. **CPU Testing- LOAD instruction (CPU\_TEST3.v)**

Similar to the previous case, the test bench for the LOAD instructions is done separately. The memory locations are preloaded with random values and the same is stored in a reference array. Here as well the engineer can type different load instructions in a file and all those instructions executed and results are validated from the reference file.

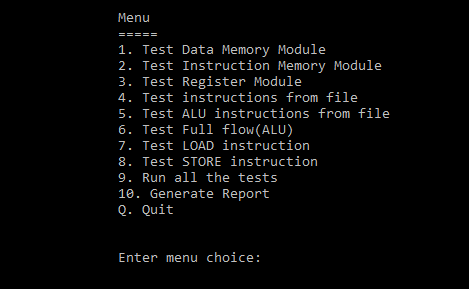
1. **CPU Testing- STORE instruction (CPU\_TEST4.v)**

Here test bench for the STORE instructions is done. The engineer can type different store instructions in a file and all those instructions executed and results are validated.

The testbench codes are added in the APPENDIX-A

**SCRIPTING DETAILS**

All the above described testbenches can be executed from command line, and the flow is automated using Python Scripting. A menu driven utility is developed in which the user can select which testbench needs to be executed. The menu interface is shown below



The python file (automate.py) is added in APPENDIX-B

**SUMMARY OF SUPPORTING MODULES DESIGNED**

1. process.cpp

A utility written in C++ to convert the assembly language into machine codes- A simple implementation of a compiler (without syntax checking).

1. MATLAB code which generates reference file.
2. Report.cpp

A C++ utility to summarize the test results and generate report

**REFERENCE**

The project source code link

<https://github.com/nextseto/Verilog-Projects/tree/master/Project%205%20-%20ARM%20LEGv8%20Simulator>

**APPENDIX A**

**TEST BENCH CODES**

TEST1.v

`timescale 1ns / 1ps

module TEST1;

// Inputs

reg [63:0] PC;

// Outputs

wire [31:0] CPU\_Instruction;

integer i;

integer file;

integer file1;

reg [31:0]ref\_inst;

integer flag;

integer status;

// Instantiate the Unit Under Test (UUT)

Instruction\_Memory uut (

.PC(PC),

.CPU\_Instruction(CPU\_Instruction)

);

initial begin

// Initialize Inputs

PC = 0;

flag=0;

file = $fopen("Result.txt") ;

file1 = $fopen("Reference\_Instruction.txt","r") ;

#10

$display("Verifying instruction stored in the memory");

for(i=0;i<13;i=i+1)

begin

#10;

status = $fscanf(file1,"%b\n",ref\_inst);

if(ref\_inst==CPU\_Instruction)

begin

$display("PP\_Valid");

end

else

begin

$display("FF\_Invalid instruction found: Actual-%d--Obtained-%b",ref\_inst,CPU\_Instruction);

flag=1;

end

//$display(" pass");

PC=PC+4;

end

if(flag==0)

$fdisplay(file, "1.Pass") ;

else

$fdisplay(file, "1.Fail") ;

$fclose(file);

$fclose(file1);

// Wait 200 ns for global reset to finish

#200 $finish;

end

endmodule

TEST2.v

`timescale 1ps / 1ps

module TEST2;

// Inputs

reg [63:0] inputAddress;

reg [63:0] inputData;

reg CONTROL\_MemRead;

reg CONTROL\_MemWrite;

// Outputs

wire [63:0] outputData;

//Other variables

integer i,k;

reg [63:0]ref\_data[31:0];

reg [63:0]ref\_address[31:0];

reg [9:0]value;

reg [31:0]verify;

integer count;

integer flag;

integer file;

integer fcount;

// Instantiate the Unit Under Test (UUT)

Data\_Memory uut (

.inputAddress(inputAddress),

.inputData(inputData),

.CONTROL\_MemRead(CONTROL\_MemRead),

.CONTROL\_MemWrite(CONTROL\_MemWrite),

.outputData(outputData)

);

initial

begin

// Initialize Inputs

inputAddress = 0;

inputData = 0;

CONTROL\_MemRead = 0;

CONTROL\_MemWrite = 0;

verify=0;

count=0;

flag=0;

fcount=0;

//To open file

file = $fopen("Result.txt","a") ;

#5;

//To display the values stored in the memory using CONTROL\_MemRead and inputAddress

CONTROL\_MemRead=1'b1;

#1;

$display("The data stored currently in Data Memory:");

for(i=0;i<32;i=i+1)

begin

inputAddress = i;

#1;

$display("Address-%d===Data-%d",i,outputData);

end

CONTROL\_MemRead=1'b0;

inputAddress=0;

//To write values into the memory using CONTROL\_MemWrite and inputData

CONTROL\_MemWrite=1'b1;

#2;

//$display("Writing random values into memory:");

//====================================

for(k=0;k<10;k=k+1)

begin

$display("Started");

verify=0;

#1;

CONTROL\_MemWrite=1'b1;

for(i=0;i<32;i=i+1)

begin

inputData = $random();

inputAddress=i;

ref\_data[i]=inputData;

ref\_address[i]=inputAddress;

#2;

end

CONTROL\_MemWrite=1'b0;

//To display the values written into the memory using CONTROL\_MemRead and random inputAddress

CONTROL\_MemRead=1'b1;

#2;

//$display("Memory Values");

//Loop continuously till all memory locations are covered

forever

begin

count=count+1;

value={$random};

value=(0+value%32);

#1;

inputAddress = value;

if(verify==4294967295)

begin

$display("Memory write verified");

if(flag==1)

begin

//$display("Fail");

$fdisplay(file, "2.Fail") ;

end

else

begin

//$display("Pass");

$fdisplay(file, "2.Pass") ;

end

$fclose(file) ;

//$display("Count value==>%d",count);

//$display("Number of memory reads failed=%d",fcount);

$finish;

end

if(verify[value]!=1)

begin

verify[value]=1;

#5;

if(outputData!=ref\_data[inputAddress])

begin

flag=1;

fcount=fcount+1;

$display("FF\_Invalid value found @ address-%d",inputAddress);

$display("Memory Address-%d Actual Data-%d Data retrieved-%d",i,ref\_data[inputAddress],outputData);

end

else

$display("PP\_Valid--Data Memory%d--%d",inputAddress,outputData);

end

end

CONTROL\_MemRead=1'b0;

// Wait 1000 ns for global reset to finish

end

$fclose(file);

#2000 $finish;

end

endmodule

TEST3.v

`timescale 1ns / 1ps

module TEST3;

// Inputs

reg [4:0] read1;

reg [4:0] read2;

reg [4:0] writeReg;

reg [63:0] writeData;

reg CONTROL\_REGWRITE;

// Outputs

wire [63:0] data1;

wire [63:0] data2;

//Other variables

integer i;

reg [4:0]regnum;

integer file1;

integer file2;

reg [31:0]verify;

reg [9:0]value;

integer limit;

reg [4:0]refreg;

reg [63:0]refvalue;

integer status;

integer flag;

// Instantiate the Unit Under Test (UUT)

Registers uut (

.read1(read1),

.read2(read2),

.writeReg(writeReg),

.writeData(writeData),

.CONTROL\_REGWRITE(CONTROL\_REGWRITE),

.data1(data1),

.data2(data2)

);

initial begin

// Initialize Inputs

read1 = 0;

read2 = 0;

writeReg = 0;

writeData = 0;

CONTROL\_REGWRITE = 0;

verify=0;

limit=1;

flag=0;

#5;

//To open file

file1 = $fopen("Ref.txt") ;

file2 = $fopen("Result.txt","a") ;

//To display the values of registers using read1 and data1

$display("Register values currently stored");

for(i=0;i<32;i=i+1)

begin

read1 = i;

#2;

//$display("Register-%d==Value-%d",i,data1);

end

#5;

//To write data into registers using CONTROL\_REGWRITE,writeReg and writeData

$display("Writing random values into registers:");

CONTROL\_REGWRITE=1'b1;

for(i=0;i<limit;i=i+1)

begin

//$display("KERI-%d",i);

if(verify==4294967295)

begin

$display("Values written into registers");

end

else

limit=limit+1;

value={$random};

value=(0+value%32);

#1;

if(verify[value]!=1)

begin

verify[value]=1;

writeReg = value;

regnum=value;

writeData=$random();

$fdisplay(file1,"%b %b",regnum,writeData);

#3;

end

end

$fclose(file1);

file1 = $fopen("Ref.txt","r") ;

CONTROL\_REGWRITE=1'b0;

#5

//To display the values written into registers using read2 and data2

$display("Verifying Register Values");

for(i=0;i<32;i=i+1)

begin

status = $fscanf(file1,"%b %b\n",refreg,refvalue);

//$display("Ref reg--%d==Ref value--%d",refreg,refvalue);

read2 = refreg;

#5;

if(data2!=refvalue)

begin

flag=1;

$display("FF\_Invalid value found in register:");

$display("Register-%d==Actual value-%d==Data retrieved-%d",refreg,refvalue,data2);

end

else

$display("PP\_Vaild==Register-%d==Value-%d",refreg,data2);

end

if(flag==1)

$fdisplay(file2, "3.Fail") ;

else

$fdisplay(file2, "3.Pass") ;

$fclose(file1);

$fclose(file2);

// Wait 300 ns for global reset to finish

#300 $finish;

end

endmodule

CPU\_TEST.v

`timescale 1ns / 1ps

/\*

Group Members: Akhil and Jinson

Lab Name: ARM LEG v8 CPU Testbench

\*/

module CPU\_TEST;

/\* Clock Signal \*/

reg CLOCK;

/\* Wires to connect instruction memory to CPU \*/

wire [63:0] instructionPC;

reg [31:0] instructionOut;

/\* Wires to connect registers to CPU \*/

wire [4:0] READ\_REG\_1;

wire [4:0] READ\_REG\_2;

wire [4:0] WRITE\_REG;

wire [63:0] WRITE\_DATA;

wire [63:0] DATA\_OUT\_1;

wire [63:0] DATA\_OUT\_2;

/\* Wires to connect Data Memory to CPU \*/

wire [63:0] data\_memory\_out;

wire [63:0] ALU\_Result\_Out;

/\* Wires to connect CPU Control Lines to Memories \*/

wire CONTROL\_REG2LOC;

wire CONTROL\_REGWRITE;

wire CONTROL\_MEMREAD;

wire CONTROL\_MEMWRITE;

wire CONTROL\_BRANCH;

//Other variables

//reg [63:0] temp\_instructionPC;

reg [31:0]temp\_instructionOut;

integer i;

integer file;

integer status;

/\* Instruction Memory Module \*/

Instruction\_Memory mem1 (

instructionPC,

instructionOut

);

/\* Registers Module \*/

Registers mem2 (

READ\_REG\_1,

READ\_REG\_2,

WRITE\_REG,

WRITE\_DATA,

CONTROL\_REGWRITE,

DATA\_OUT\_1,

DATA\_OUT\_2

);

/\* Data Memory Module \*/

Data\_Memory mem3 (

ALU\_Result\_Out,

DATA\_OUT\_2,

CONTROL\_MEMREAD,

CONTROL\_MEMWRITE,

data\_memory\_out

);

/\* CPU Module \*/

ARM\_CPU core (

.CLOCK(CLOCK),

.INSTRUCTION(instructionOut),

.PC(instructionPC),

.CONTROL\_REG2LOC(CONTROL\_REG2LOC),

.CONTROL\_REGWRITE(CONTROL\_REGWRITE),

.CONTROL\_MEMREAD(CONTROL\_MEMREAD),

.CONTROL\_MEMWRITE(CONTROL\_MEMWRITE),

.CONTROL\_BRANCH(CONTROL\_BRANCH),

.READ\_REG\_1(READ\_REG\_1),

.READ\_REG\_2(READ\_REG\_2),

.WRITE\_REG(WRITE\_REG),

.REG\_DATA1(DATA\_OUT\_1),

.REG\_DATA2(DATA\_OUT\_2),

.ALU\_Result\_Out(ALU\_Result\_Out),

.data\_memory\_out(data\_memory\_out),

.WRITE\_REG\_DATA(WRITE\_DATA)

);

/\* Setup the clock \*/

initial

begin

CLOCK = 1'b0;

i=0;

file = $fopen("instruction.txt","r") ;

while ( ! $feof(file))

begin

status = $fscanf(file,"%b\n",temp\_instructionOut);

if(status==1)

begin

i=i+1;

@(posedge CLOCK)

instructionOut=temp\_instructionOut;

$display("PP\_Instruction%d executed",i);

end

else

$display("FF\_Reading failed");

end

$fclose(file);

#100 $finish;

end

/\* Toggle the clock \*/

always begin

#1 CLOCK = ~CLOCK;

end

endmodule

CPU\_TEST1.v

`timescale 1ps / 1ps

/\*

Group Members: Akhil and Jinson

Lab Name: ARM LEG v8 CPU Testbench

\*/

module CPU\_TEST1;

/\* Clock Signal \*/

reg CLOCK;

/\* Wires to connect instruction memory to CPU \*/

wire [63:0] instructionPC;

reg[31:0] instructionOut;

/\* Wires to connect registers to CPU \*/

wire [4:0] READ\_REG\_1;

wire [4:0] READ\_REG\_2;

wire [4:0] WRITE\_REG;

wire [63:0] WRITE\_DATA;

wire [63:0] DATA\_OUT\_1;

wire [63:0] DATA\_OUT\_2;

/\* Wires to connect Data Memory to CPU \*/

wire [63:0] data\_memory\_out;

wire [63:0] ALU\_Result\_Out;

/\* Wires to connect CPU Control Lines to Memories \*/

wire CONTROL\_REG2LOC;

wire CONTROL\_REGWRITE;

wire CONTROL\_MEMREAD;

wire CONTROL\_MEMWRITE;

wire CONTROL\_BRANCH;

//Other variables

integer file1,status1,i;

reg [63:0]op1,op2,o1,o2,o3,o4;

//reg [63:0]op3,op4;

reg [10:0]i1;

reg [4:0]r1;

reg [4:0]r2;

reg [4:0]r3;

reg [5:0]i2;

//reg [63:0]result;

//integer file,status;

//reg [1:0]select;

/\* Instruction Memory Module \*/

/\*Instruction\_Memory mem1 (

instructionPC,

instructionOut

);\*/

/\* Registers Module \*/

Registers mem2 (

READ\_REG\_1,

READ\_REG\_2,

WRITE\_REG,

WRITE\_DATA,

CONTROL\_REGWRITE,

DATA\_OUT\_1,

DATA\_OUT\_2

);

/\* Data Memory Module \*/

Data\_Memory mem3 (

ALU\_Result\_Out,

DATA\_OUT\_2,

CONTROL\_MEMREAD,

CONTROL\_MEMWRITE,

data\_memory\_out

);

/\* CPU Module \*/

ARM\_CPU core (

.CLOCK(CLOCK),

.INSTRUCTION(instructionOut),

.PC(instructionPC),

.CONTROL\_REG2LOC(CONTROL\_REG2LOC),

.CONTROL\_REGWRITE(CONTROL\_REGWRITE),

.CONTROL\_MEMREAD(CONTROL\_MEMREAD),

.CONTROL\_MEMWRITE(CONTROL\_MEMWRITE),

.CONTROL\_BRANCH(CONTROL\_BRANCH),

.READ\_REG\_1(READ\_REG\_1),

.READ\_REG\_2(READ\_REG\_2),

.WRITE\_REG(WRITE\_REG),

.REG\_DATA1(DATA\_OUT\_1),

.REG\_DATA2(DATA\_OUT\_2),

.ALU\_Result\_Out(ALU\_Result\_Out),

.data\_memory\_out(data\_memory\_out),

.WRITE\_REG\_DATA(WRITE\_DATA)

);

/\* Setup the clock \*/

initial

begin

CLOCK = 1'b0;

i1=11'b10001011000;

i2=6'b000000;

//$display("Started");

file1 = $fopen("inputfile.txt","r");

for(i=0;i<100;i=i+1)

begin

status1 = $fscanf(file1,"%b %b %b %b %b %b\n",op1,op2,o1,o2,o3,o4);

//$display("OPERAND1--%d==OPERAND2--%d==SUM--%d==DIFFERENCE--%d==AND--%d==OR--%d",op1,op2,o1,o2,o3,o4);

//Randomly selecting the registers

r1=0+{$random}%10;

r2=10+{$random}%10;

r3=20+{$random}%12;

//$display("Register==>%d %d %d",r1,r2,r3);

//Wwriting the values into the registers using force and release

force CONTROL\_REGWRITE=1'b1;

force WRITE\_REG=r2;

force WRITE\_DATA=op1;

#1;

force WRITE\_REG=r3;

force WRITE\_DATA=op2;

#1;

release WRITE\_REG;

release WRITE\_DATA;

release CONTROL\_REGWRITE;

#1;

//Values placed in registers

//Verifying ADD instruction

i1=11'b10001011000;

executeinst(i1,i2,r3,r2,r1,o1,2'b00);

#2;

//Verifying SUB instruction

i1=11'b11001011000;

executeinst(i1,i2,r3,r2,r1,o2,2'b01);

#2;

//Verifying AND instruction

i1=11'b10001010000;

executeinst(i1,i2,r3,r2,r1,o3,2'b10);

#2;

//Verifying OR instruction

i1=11'b10101010000;

executeinst(i1,i2,r3,r2,r1,o4,2'b11);

#2;

// CLOCK = ~CLOCK;

#1;

end

$fclose(file1);

#500 $finish;

end

/\* Toggle the clock \*/

/\*always begin

#1 CLOCK = ~CLOCK;

end\*/

task executeinst;

input [10:0]i1;

input [5:0]i2;

input [4:0]r3;

input [4:0]r2;

input [4:0]r1;

input [63:0]res;

input [2:0]mode;

begin

//$display("In task");

instructionOut={i1,r3,i2,r2,r1};

//$display("%h %b",instructionOut,mode);

#1;

//$display("Output---ALU\_Result\_Out=%d",ALU\_Result\_Out);

if(ALU\_Result\_Out==res)

$display("PP\_Valid");

else

begin

//$display("Failed");

//$display("Register==>%d %d %d",r1,r2,r3);

case(mode)

2'b00:

$display("FF\_Failed Addition==Operand1-%d, Operand2-%d, Actual Result-%d, Result Obtained-%d",op1,op2,res,ALU\_Result\_Out);

2'b01:

$display("FF\_Failed Subtraction==Operand1-%d, Operand2-%d, Actual Result-%d, Result Obtained-%d",op1,op2,res,ALU\_Result\_Out);

2'b10:

$display("FF\_Failed AND==Operand1-%d, Operand2-%d, Actual Result-%d, Result Obtained-%d",op1,op2,res,ALU\_Result\_Out);

2'b11:

$display("FF\_Failed OR==Operand1-%d, Operand2-%d, Actual Result-%d, Result Obtained-%d",op1,op2,res,ALU\_Result\_Out);

endcase

end

end

endtask

endmodule

CPU\_TEST2.v

`timescale 1ps / 1ps

/\*

Group Members: Akhil and Jinson

Lab Name: ARM LEG v8 CPU Testbench

\*/

module CPU\_TEST2;

/\* Clock Signal \*/

reg CLOCK;

/\* Wires to connect instruction memory to CPU \*/

wire [63:0] instructionPC;

reg[31:0] instructionOut;

/\* Wires to connect registers to CPU \*/

wire [4:0] READ\_REG\_1;

wire [4:0] READ\_REG\_2;

wire [4:0] WRITE\_REG;

wire [63:0] WRITE\_DATA;

wire [63:0] DATA\_OUT\_1;

wire [63:0] DATA\_OUT\_2;

/\* Wires to connect Data Memory to CPU \*/

wire [63:0] data\_memory\_out;

wire [63:0] ALU\_Result\_Out;

/\* Wires to connect CPU Control Lines to Memories \*/

wire CONTROL\_REG2LOC;

wire CONTROL\_REGWRITE;

wire CONTROL\_MEMREAD;

wire CONTROL\_MEMWRITE;

wire CONTROL\_BRANCH;

//Other variables

integer file1,status1,i;

integer file,status;

integer n1,n2,n3;

reg [63:0]inst;

reg [4:0]r1;

reg [4:0]r2;

reg [4:0]r3;

reg [63:0]result;

/\* Instruction Memory Module \*/

/\*Instruction\_Memory mem1 (

instructionPC,

instructionOut

);\*/

/\* Registers Module \*/

Registers mem2 (

READ\_REG\_1,

READ\_REG\_2,

WRITE\_REG,

WRITE\_DATA,

CONTROL\_REGWRITE,

DATA\_OUT\_1,

DATA\_OUT\_2

);

/\* Data Memory Module \*/

Data\_Memory mem3 (

ALU\_Result\_Out,

DATA\_OUT\_2,

CONTROL\_MEMREAD,

CONTROL\_MEMWRITE,

data\_memory\_out

);

/\* CPU Module \*/

ARM\_CPU core (

.CLOCK(CLOCK),

.INSTRUCTION(instructionOut),

.PC(instructionPC),

.CONTROL\_REG2LOC(CONTROL\_REG2LOC),

.CONTROL\_REGWRITE(CONTROL\_REGWRITE),

.CONTROL\_MEMREAD(CONTROL\_MEMREAD),

.CONTROL\_MEMWRITE(CONTROL\_MEMWRITE),

.CONTROL\_BRANCH(CONTROL\_BRANCH),

.READ\_REG\_1(READ\_REG\_1),

.READ\_REG\_2(READ\_REG\_2),

.WRITE\_REG(WRITE\_REG),

.REG\_DATA1(DATA\_OUT\_1),

.REG\_DATA2(DATA\_OUT\_2),

.ALU\_Result\_Out(ALU\_Result\_Out),

.data\_memory\_out(data\_memory\_out),

.WRITE\_REG\_DATA(WRITE\_DATA)

);

/\* Setup the clock \*/

initial

begin

CLOCK = 1'b0;

//$display("Started");

file1 = $fopen("inputfile1.txt","r") ;

file = $fopen("MachineCodeOutput.txt","r") ;

while (!$feof(file))

begin

status1 = $fscanf(file1,"%d %d %d\n",n1,n2,n3);

status = $fscanf(file,"%b\n",inst);

//$display("Value1--%d==Value2--%d==Sum--%d==Inst--%b",n1,n2,n3,inst);

r2=inst[20:16];

r3=inst[9:5];

r1=inst[4:0];

$display("Register==>%d %d %d",r2,r3,r1);

force CONTROL\_REGWRITE=1'b1;

force WRITE\_REG=r2;

force WRITE\_DATA=n1;

#1;

force WRITE\_REG=r3;

force WRITE\_DATA=n2;

#1;

release WRITE\_REG;

release WRITE\_DATA;

release CONTROL\_REGWRITE;

#2;

instructionOut=inst;

#2;

result=ALU\_Result\_Out;

#2;

$display("Output---ALU\_Result\_Out=%d",ALU\_Result\_Out);

//$display("Output---data\_memory\_out=%d",data\_memory\_out);

if(result==n3)

$display("PP\_Valid");

else

begin

//$display("Register==>%d %d %d",r1,r2,r3);

$display("FF\_Failed: Operands are: %d & %d: Actual Result=%d, Result obtained=%d",n1,n2,n3,result);

end

force CONTROL\_REGWRITE=1'b0;

//$display("%d",r1);

force READ\_REG\_1=r1;

#1;

if(DATA\_OUT\_1==n3)

$display("PP\_Valid:Written into proper register");

else

$display("FF\_Data not written into destination");

release CONTROL\_REGWRITE;

release READ\_REG\_1;

end

#1;

$fclose(file1);

$fclose(file);

#700 $finish;

end

/\* Toggle the clock \*/

/\*always begin

#1 CLOCK = ~CLOCK;

end\*/

endmodule

CPU\_TEST3.v

`timescale 1ns / 1ps

/\*

Group Members: Akhil and Jinson

Lab Name: ARM LEG v8 CPU Testbench

\*/

module CPU\_TEST3;

/\* Clock Signal \*/

reg CLOCK;

/\* Wires to connect instruction memory to CPU \*/

wire [63:0] instructionPC;

reg[31:0] instructionOut;

/\* Wires to connect registers to CPU \*/

wire [4:0] READ\_REG\_1;

wire [4:0] READ\_REG\_2;

wire [4:0] WRITE\_REG;

wire [63:0] WRITE\_DATA;

wire [63:0] DATA\_OUT\_1;

wire [63:0] DATA\_OUT\_2;

/\* Wires to connect Data Memory to CPU \*/

wire [63:0] data\_memory\_out;

wire [63:0] ALU\_Result\_Out;

/\* Wires to connect CPU Control Lines to Memories \*/

wire CONTROL\_REG2LOC;

wire CONTROL\_REGWRITE;

wire CONTROL\_MEMREAD;

wire CONTROL\_MEMWRITE;

wire CONTROL\_BRANCH;

//Other variables

integer file1,status1,i;

integer file,status;

integer n1,n2;

reg [63:0]inst;

reg [4:0]r1;

integer r2;

reg [63:0]value;

reg [63:0]reference[0:31];

/\* Instruction Memory Module \*/

/\*Instruction\_Memory mem1 (

instructionPC,

instructionOut

);\*/

/\* Registers Module \*/

Registers mem2 (

READ\_REG\_1,

READ\_REG\_2,

WRITE\_REG,

WRITE\_DATA,

CONTROL\_REGWRITE,

DATA\_OUT\_1,

DATA\_OUT\_2

);

/\* Data Memory Module \*/

Data\_Memory mem3 (

ALU\_Result\_Out,

DATA\_OUT\_2,

CONTROL\_MEMREAD,

CONTROL\_MEMWRITE,

data\_memory\_out

);

/\* CPU Module \*/

ARM\_CPU core (

.CLOCK(CLOCK),

.INSTRUCTION(instructionOut),

.PC(instructionPC),

.CONTROL\_REG2LOC(CONTROL\_REG2LOC),

.CONTROL\_REGWRITE(CONTROL\_REGWRITE),

.CONTROL\_MEMREAD(CONTROL\_MEMREAD),

.CONTROL\_MEMWRITE(CONTROL\_MEMWRITE),

.CONTROL\_BRANCH(CONTROL\_BRANCH),

.READ\_REG\_1(READ\_REG\_1),

.READ\_REG\_2(READ\_REG\_2),

.WRITE\_REG(WRITE\_REG),

.REG\_DATA1(DATA\_OUT\_1),

.REG\_DATA2(DATA\_OUT\_2),

.ALU\_Result\_Out(ALU\_Result\_Out),

.data\_memory\_out(data\_memory\_out),

.WRITE\_REG\_DATA(WRITE\_DATA)

);

/\* Setup the clock \*/

initial

begin

CLOCK = 1'b0;

file = $fopen("MachineCodeOutput1.txt","r") ;

for(i=0;i<32;i=i+1)

begin

value=$random();

force ALU\_Result\_Out=i;

force DATA\_OUT\_2=value;

force CONTROL\_MEMWRITE =1'b1;

reference[i]=value;

#1;

release ALU\_Result\_Out;

release DATA\_OUT\_2;

release CONTROL\_MEMWRITE;

end

for(i=0;i<32;i=i+1)

begin

//$display("%d--%d",i,reference[i]);

end

while (!$feof(file))

begin

status = $fscanf(file,"%b\n",inst);

r1=inst[4:0];

r2=inst[9:5];

//$display("Value-%d",reference[r2]);

//$display("Register==>%d %d",r2,r3);

instructionOut=inst;

#1;

force CONTROL\_REGWRITE=1'b0;

force READ\_REG\_1=r1;

#2;

//$display("output%d",data\_memory\_out);

if(DATA\_OUT\_1!=reference[r2])

$display("FF\_Failed--Actual value-%d..Obtained value-%d",reference[r2],DATA\_OUT\_1);

else

$display("PP\_Valid--Actual value-%d..Obtained value-%d",reference[r2],DATA\_OUT\_1);

#1;

release CONTROL\_REGWRITE;

release READ\_REG\_1;

#2;

end

#1;

// $fclose(file1);

$fclose(file);

#700 $finish;

end

/\* Toggle the clock \*/

/\*always begin

#1 CLOCK = ~CLOCK;

end\*/

endmodule

CPU\_TEST4.v

`timescale 1ns / 1ps

/\*

Group Members: Akhil and Jinson

Lab Name: ARM LEG v8 CPU Testbench

\*/

module CPU\_TEST4;

/\* Clock Signal \*/

reg CLOCK;

/\* Wires to connect instruction memory to CPU \*/

wire [63:0] instructionPC;

reg[31:0] instructionOut;

/\* Wires to connect registers to CPU \*/

wire [4:0] READ\_REG\_1;

wire [4:0] READ\_REG\_2;

wire [4:0] WRITE\_REG;

wire [63:0] WRITE\_DATA;

wire [63:0] DATA\_OUT\_1;

wire [63:0] DATA\_OUT\_2;

/\* Wires to connect Data Memory to CPU \*/

wire [63:0] data\_memory\_out;

wire [63:0] ALU\_Result\_Out;

/\* Wires to connect CPU Control Lines to Memories \*/

wire CONTROL\_REG2LOC;

wire CONTROL\_REGWRITE;

wire CONTROL\_MEMREAD;

wire CONTROL\_MEMWRITE;

wire CONTROL\_BRANCH;

//Other variables

integer file1,status1,i;

integer file,status;

integer n1,n2;

reg [63:0]inst;

reg [4:0]r1;

integer r2;

reg [63:0]value;

reg [63:0]reference[0:31];

/\* Instruction Memory Module \*/

/\*Instruction\_Memory mem1 (

instructionPC,

instructionOut

);\*/

/\* Registers Module \*/

Registers mem2 (

READ\_REG\_1,

READ\_REG\_2,

WRITE\_REG,

WRITE\_DATA,

CONTROL\_REGWRITE,

DATA\_OUT\_1,

DATA\_OUT\_2

);

/\* Data Memory Module \*/

Data\_Memory mem3 (

ALU\_Result\_Out,

DATA\_OUT\_2,

CONTROL\_MEMREAD,

CONTROL\_MEMWRITE,

data\_memory\_out

);

/\* CPU Module \*/

ARM\_CPU core (

.CLOCK(CLOCK),

.INSTRUCTION(instructionOut),

.PC(instructionPC),

.CONTROL\_REG2LOC(CONTROL\_REG2LOC),

.CONTROL\_REGWRITE(CONTROL\_REGWRITE),

.CONTROL\_MEMREAD(CONTROL\_MEMREAD),

.CONTROL\_MEMWRITE(CONTROL\_MEMWRITE),

.CONTROL\_BRANCH(CONTROL\_BRANCH),

.READ\_REG\_1(READ\_REG\_1),

.READ\_REG\_2(READ\_REG\_2),

.WRITE\_REG(WRITE\_REG),

.REG\_DATA1(DATA\_OUT\_1),

.REG\_DATA2(DATA\_OUT\_2),

.ALU\_Result\_Out(ALU\_Result\_Out),

.data\_memory\_out(data\_memory\_out),

.WRITE\_REG\_DATA(WRITE\_DATA)

);

/\* Setup the clock \*/

initial

begin

CLOCK = 1'b0;

file = $fopen("MachineCodeOutput2.txt","r") ;

while (!$feof(file))

begin

status = $fscanf(file,"%b\n",inst);

r1=inst[4:0];

r2=inst[9:5];

instructionOut=inst;

#1;

force ALU\_Result\_Out=r2;

force CONTROL\_MEMREAD =1'b1;

#1;

value=data\_memory\_out;

#1;

release ALU\_Result\_Out;

release CONTROL\_MEMREAD;

force CONTROL\_REGWRITE=1'b0;

force READ\_REG\_1=r1;

#2;

//$display("output%d",data\_memory\_out);

if(DATA\_OUT\_1!=value)

$display("FF\_Failed--Actual value-%d..Obtained value-%d",DATA\_OUT\_1,value);

else

$display("PP\_Valid--Actual value-%d..Obtained value-%d",DATA\_OUT\_1,value);

#1;

release CONTROL\_REGWRITE;

release READ\_REG\_1;

#2;

end

$fclose(file);

#700 $finish;

end

/\* Toggle the clock \*/

/\*always begin

#1 CLOCK = ~CLOCK;

end\*/

endmodule

**APPENDIX B**

**SCRIPTS AND SUPPORTING MODULES**

AUTOMATE.PY

import msvcrt as m

import os

os.system('cls')

def wait():

m.getch()

def getChoice():

print("\n\t\t\tMenu\n\t\t\t=====\n\t\t\t1. Test Data Memory Module\n\t\t\t2. Test Instruction Memory Module\n\t\t\t3. Test Register Module\n\t\t\t4. Test instructions from file\n\t\t\t5. Test ALU instructions from file\n\t\t\t6. Test Full flow(ALU)")

print("\t\t\t7. Test LOAD instruction \n\t\t\t8. Test STORE instruction \n\t\t\t9. Run all the tests \n\t\t\t10. Generate Report\t\t\t\n\t\t\tQ. Quit")

choice= input("\n\n\t\t\tEnter menu choice:")

return choice

def testDataMemory():

os.system('vlogcomp -work isim\_temp -intstyle ise -prj TEST2\_stx\_beh.prj')

os.system('fuse -intstyle ise -incremental -lib unisims\_ver -lib unimacro\_ver -lib xilinxcorelib\_ver -o TEST2.exe -prj TEST1\_stx\_beh.prj isim\_temp.TEST2 isim\_temp.glbl')

os.system('TEST2.exe -intstyle ise -tclbatch isim.cmd -log test2.txt')

def testInsMemory():

os.system('vlogcomp -work isim\_temp -intstyle ise -prj TEST1\_stx\_beh.prj')

os.system('fuse -intstyle ise -incremental -lib unisims\_ver -lib unimacro\_ver -lib xilinxcorelib\_ver -o TEST1.exe -prj TEST1\_stx\_beh.prj isim\_temp.TEST1 isim\_temp.glbl')

os.system('TEST1.exe -intstyle ise -tclbatch isim.cmd -log test1.txt')

def testRegisters():

os.system('vlogcomp -work isim\_temp -intstyle ise -prj TEST3\_stx\_beh.prj')

os.system('fuse -intstyle ise -incremental -lib unisims\_ver -lib unimacro\_ver -lib xilinxcorelib\_ver -o TEST3.exe -prj TEST3\_stx\_beh.prj isim\_temp.TEST3 isim\_temp.glbl')

os.system('TEST3.exe -intstyle ise -tclbatch isim.cmd -log test3.txt')

def testInstructionFile():

os.system('vlogcomp -work isim\_temp -intstyle ise -prj CPU\_TEST\_stx\_beh.prj')

os.system('fuse -intstyle ise -incremental -lib unisims\_ver -lib unimacro\_ver -lib xilinxcorelib\_ver -o CPU\_TEST.exe -prj CPU\_TEST\_stx\_beh.prj isim\_temp.CPU\_TEST isim\_temp.glbl')

os.system('CPU\_TEST.exe -intstyle ise -tclbatch isim.cmd -log test4.txt')

def testAluIns():

os.system('vlogcomp -work isim\_temp -intstyle ise -prj CPU\_TEST1\_stx\_beh.prj')

os.system('fuse -intstyle ise -incremental -lib unisims\_ver -lib unimacro\_ver -lib xilinxcorelib\_ver -o CPU\_TEST1.exe -prj CPU\_TEST1\_stx\_beh.prj isim\_temp.CPU\_TEST1 isim\_temp.glbl')

os.system('CPU\_TEST1.exe -intstyle ise -tclbatch isim.cmd -log test5.txt')

def testComplete():

os.system('start notepad AssemblyInstructionInput.txt')

os.system('start notepad inputfile1.txt')

print("Press any key to resume")

wait();

os.system('process.exe')

os.system('vlogcomp -work isim\_temp -intstyle ise -prj CPU\_TEST2\_stx\_beh.prj')

os.system('fuse -intstyle ise -incremental -lib unisims\_ver -lib unimacro\_ver -lib xilinxcorelib\_ver -o CPU\_TEST2.exe -prj CPU\_TEST2\_stx\_beh.prj isim\_temp.CPU\_TEST2 isim\_temp.glbl')

os.system('CPU\_TEST2.exe -intstyle ise -tclbatch isim.cmd -log test6.txt')

def testCompleteLoad():

os.system('start notepad AssemblyInstructionInput1.txt')

print("Press any key to resume")

wait();

os.system('process\_load.exe')

os.system('vlogcomp -work isim\_temp -intstyle ise -prj CPU\_TEST3\_stx\_beh.prj')

os.system('fuse -intstyle ise -incremental -lib unisims\_ver -lib unimacro\_ver -lib xilinxcorelib\_ver -o CPU\_TEST3.exe -prj CPU\_TEST3\_stx\_beh.prj isim\_temp.CPU\_TEST3 isim\_temp.glbl')

os.system('CPU\_TEST3.exe -intstyle ise -tclbatch isim.cmd -log test7.txt')

def testCompleteStore():

os.system('start notepad AssemblyInstructionInput2.txt')

print("Press any key to resume")

wait();

os.system('process\_store.exe')

os.system('vlogcomp -work isim\_temp -intstyle ise -prj CPU\_TEST4\_stx\_beh.prj')

os.system('fuse -intstyle ise -incremental -lib unisims\_ver -lib unimacro\_ver -lib xilinxcorelib\_ver -o CPU\_TEST4.exe -prj CPU\_TEST3\_stx\_beh.prj isim\_temp.CPU\_TEST4 isim\_temp.glbl')

os.system('CPU\_TEST4.exe -intstyle ise -tclbatch isim.cmd -log test8.txt')

def runAll():

os.system('run.cmd')

def generateReport():

os.system('report.exe')

choice = getChoice()

while choice!="q":

if choice=="1":

testDataMemory()

elif choice=="2":

testInsMemory()

elif choice=="3":

testRegisters()

elif choice=="4":

testInstructionFile()

elif choice=="5":

testAluIns()

elif choice=="6":

testComplete()

elif choice=="7":

testCompleteLoad()

elif choice=="8":

testCompleteStore()

elif choice=="9":

runAll()

elif choice=="10":

generateReport()

choice = getChoice()

print("Exited")

PROCESS.CPP

/\*

ARM ASSEMBLER

=============

Supported Instruction and formats

ADD R5,R3,R2

SUB R4,R3,R2

AND R7,R2,R3

ORR R6,R2,R3

LDUR R2,[R10]

\*/

#include <iostream>

#include <fstream>

#include <string>

#include <cstring>

#include <sstream>

#include <fstream>

using namespace std;

int getIndex(string reg);

int main ()

{

string STRING;

string opcode[100];

string registers[32];

string writeReg="";

string readReg1="";

string readReg2="";

ifstream infile;

infile.open ("AssemblyInstructionInput.txt");

ofstream myfile;

myfile.open ("MachineCodeOutput.txt");

string token;

string subToken;

int tokenNo=0;

int subTokenNo=0;

opcode[0]="10001011000";//ADD

opcode[1]="11001011000";//SUB

opcode[2]="10001010000";//AND

opcode[3]="10101010000"; //ORR

opcode[4]="11111000010"; //LDUR

opcode[5]="11111000000"; //STUR

registers[0]="00000";

registers[1]="00001";

registers[2]="00010";

registers[3]="00011";

registers[4]="00100";

registers[5]="00101";

registers[6]="00110";

registers[7]="00111";

registers[8]="01000";

registers[9]="01001";

registers[10]="01010";

registers[11]="01011";

registers[12]="01100";

registers[13]="01101";

registers[14]="01110";

int opcodeIndex=0;

bool isLoad=false;

bool isStore=false;

string instruction="";

string fullInstruction="";

if (infile.is\_open())

{

while ( getline (infile,STRING) )

{

isLoad=false;

isStore=false;

instruction="";

opcodeIndex=0;

writeReg="00000";

readReg1="00000";

readReg2="00000";

tokenNo=0;

//cout<<STRING<<endl; // Prints our STRING.

std::istringstream iss(STRING);

while (std::getline(iss, token, ' '))

{

if(tokenNo==0)

{

if(token=="ADD")

opcodeIndex=0;

else if(token=="SUB")

opcodeIndex=1;

else if(token=="AND")

opcodeIndex=2;

else if(token=="ORR")

opcodeIndex=3;

else if(token=="LDUR")

{

opcodeIndex=4;

isLoad=true;

}

else if(token=="STUR")

{

opcodeIndex=5;

isStore=true;

}

//std::cout << token<<":"<<opcodeIndex<<endl ;

instruction.append(opcode[opcodeIndex]) ;

//instruction.append("000000");

}

else

{

subTokenNo=0;

//std::cout <<"second :"<< token<<endl;

std::istringstream subiss(token);

while (std::getline(subiss, subToken, ','))

{

//std::cout << "sub token: "<<subToken<<endl;

if(!isLoad&&!isStore)

{

if(subTokenNo==0)

{

writeReg= registers[getIndex(subToken)];

//std::cout << "writeReg "<<writeReg<<endl;

}

else if(subTokenNo==1)

{

readReg1= registers[getIndex(subToken)];

//std::cout << "readReg1 "<<readReg1<<endl;

}

else if(subTokenNo==2)

{

readReg2= registers[getIndex(subToken)];

// std::cout << "readReg2 "<<readReg2<<endl;

}

}

else if(isLoad||isStore)

{

//cout<<"===========LOAD/STORE==========="<<endl;

if(subTokenNo==0)

{

writeReg= registers[getIndex(subToken)];

//std::cout << "writeReg "<<writeReg<<endl;

}

else if(subTokenNo==1)

{

readReg1= registers[getIndex(subToken)];

//std::cout << "readReg1 "<<readReg1<<endl;

}

}

subTokenNo++;

}

}

tokenNo++;

}

instruction.append(readReg2).append("000000").append(readReg1).append(writeReg);

//cout<<"============="<<endl;

// cout<<"instruction: "<<instruction<<endl;

// cout<<"============="<<endl;

fullInstruction.append(instruction).append("\n");

}

}

cout<<"fullInstruction"<<endl<<fullInstruction;

myfile<<fullInstruction;

myfile.close();

infile.close();

return 0;

}

int getIndex(string reg)

{

int regNo;

if(reg=="R0"||reg=="[R0]")

regNo=0;

if(reg=="R1"||reg=="[R1]")

regNo=1;

if(reg=="R2"||reg=="[R2]")

regNo=2;

if(reg=="R3"||reg=="[R3]")

regNo=3;

if(reg=="R4"||reg=="[R4]")

regNo=4;

if(reg=="R5"||reg=="[R5]")

regNo=5;

if(reg=="R6"||reg=="[R6]")

regNo=6;

if(reg=="R7"||reg=="[R7]")

regNo=7;

if(reg=="R8"||reg=="[R8]")

regNo=8;

if(reg=="R9"||reg=="[R9]")

regNo=9;

if(reg=="R10"||reg=="[R10]")

regNo=10;

if(reg=="R11"||reg=="[R11]")

regNo=11;

if(reg=="R12"||reg=="[R12]")

regNo=12;

if(reg=="R13"||reg=="[R13]")

regNo=13;

if(reg=="R14"||reg=="[R14]")

regNo=14;

if(reg=="R15"||reg=="[R15]")

regNo=15;

if(reg=="R16"||reg=="[R16]")

regNo=16;

if(reg=="R17"||reg=="[R17]")

regNo=17;

if(reg=="R18"||reg=="[R18]")

regNo=18;

if(reg=="R19"||reg=="[R19]")

regNo=19;

if(reg=="R20"||reg=="[R20]")

regNo=20;

if(reg=="R21"||reg=="[R21]")

regNo=21;

if(reg=="R22"||reg=="[R22]")

regNo=22;

if(reg=="R23"||reg=="[R23]")

regNo=23;

if(reg=="R24"||reg=="[R24]")

regNo=24;

if(reg=="R25"||reg=="[R25]")

regNo=25;

if(reg=="R26"||reg=="[R26]")

regNo=26;

if(reg=="R27"||reg=="[R27]")

regNo=27;

if(reg=="R28"||reg=="[R28]")

regNo=28;

if(reg=="R29"||reg=="[R29]")

regNo=29;

if(reg=="R30"||reg=="[R30]")

regNo=30;

if(reg=="R31"||reg=="[R31]")

regNo=31;

if(reg=="R32"||reg=="[R32]")

regNo=32;

return regNo;

}

REPORT.CPP

#include <iostream>

#include <fstream>

#include <string>

#include <cstring>

#include <sstream>

#include <fstream>

using namespace std;

int main ()

{

string inFiles[10];

ifstream infile;

ofstream myfile;

inFiles[0]="test1.txt";

inFiles[1]="test2.txt";

inFiles[2]="test3.txt";

inFiles[3]="test4.txt";

inFiles[4]="test5.txt";

inFiles[5]="test6.txt";

inFiles[6]="test7.txt";

inFiles[7]="test8.txt";

int length=8;

string STRING;

int noPass=0;

int noFail=0;

int noCases=0;

int noTbs=0;

myfile.open ("example.txt");

for(int i=0;i<length;i++)

{

int l\_noCase=0,l\_noFail=0,l\_noPass=0;

noTbs++;

//cout<<inFiles[i]<<endl;

cout<<"TEST BENCH :"<<i<<endl;

myfile<<"TEST BENCH :"<<i<<endl;

cout<<"=============="<<endl;

myfile<<"=============="<<endl;

infile.open (inFiles[i]);

if (infile.is\_open())

{

while ( getline (infile,STRING) )

{

//cout<<STRING<<endl;

//if(STRING.at(0)=='F')

if (STRING.compare(0,3,"FF\_") == 0)

{

noFail++;

l\_noFail++;

noCases++;

l\_noCase++;

}

//if(STRING.at(0)=='P')

if (STRING.compare(0,3,"PP\_") == 0)

{

noPass++;

l\_noPass++;

noCases++;

l\_noCase++;

}

}

}

infile.close();

cout<<"Number of Test Cases Executed: "<<l\_noCase<<endl;

cout<<"Number of Test Cases Passed: "<<l\_noPass<<endl;

cout<<"Number of Test Cases Failed: "<<l\_noFail<<endl;

myfile<<"Number of Test Cases Executed: "<<l\_noCase<<endl;

myfile<<"Number of Test Cases Passed: "<<l\_noPass<<endl;

myfile<<"Number of Test Cases Failed: "<<l\_noFail<<endl;

cout<<endl<<endl;

}

cout<<"TEST SUMMARY"<<endl;

cout<<"=============="<<endl;

cout<<"Number of Test Benches: "<<noTbs<<endl;

cout<<"Number of Test Cases Executed: "<<noCases<<endl;

cout<<"Number of Test Cases Passed: "<<noPass<<endl;

cout<<"Number of Test Cases Failed: "<<noFail<<endl;

myfile<<"TEST SUMMARY"<<endl;

myfile<<"=============="<<endl;

myfile<<"Number of Test Benches: "<<noTbs<<endl;

myfile<<"Number of Test Cases Executed: "<<noCases<<endl;

myfile<<"Number of Test Cases Passed: "<<noPass<<endl;

myfile<<"Number of Test Cases Failed: "<<noFail<<endl;

// myfile<<"test"<<endl<<"check";

myfile.close();

return 0;

}