MATRIX MULTIPLICATION USING MULTIPLE PROCESSORS

FINAL REPORT

170180N - G.Ganeshaaraj
170637A - R.Thiruparan
170657J - S.Viththakan
160257A - J.J.Jonathan

This report is submitted as a partial fulfilment for the module
EN3030: Circuits and Systems Design
Department of Electronic and Telecommunication Engineering
University of Moratuwa
7th of July 2021

TABLE OF CONTENTS

1. Introduction

1.1. Objective

Multiplication of matrices is a time consuming process so implementing matrix multiplication in a single processor is an inefficient solution. The objective of this project is to multiply two matrices efficiently using multiple processors.

1.2. Overview of the implemented solution

- In this project we have used 4 different custom made processors to effectively carry out the matrix multiplication.
- The number of processors needed for a particular matrix multiplication is dependent on the dimensions of those matrices.
- If matrix A and matric B are the input matrices
    - Each row of matrix A will be given to each and every activated processor.
    - All the elements of matrix B will be given to all the activated processors

- All the multiplications and additions necessary for matrix multiplication will be done in a parallel manner within all the processors.
- Results of the matrix multiplication will be stored in the data memory

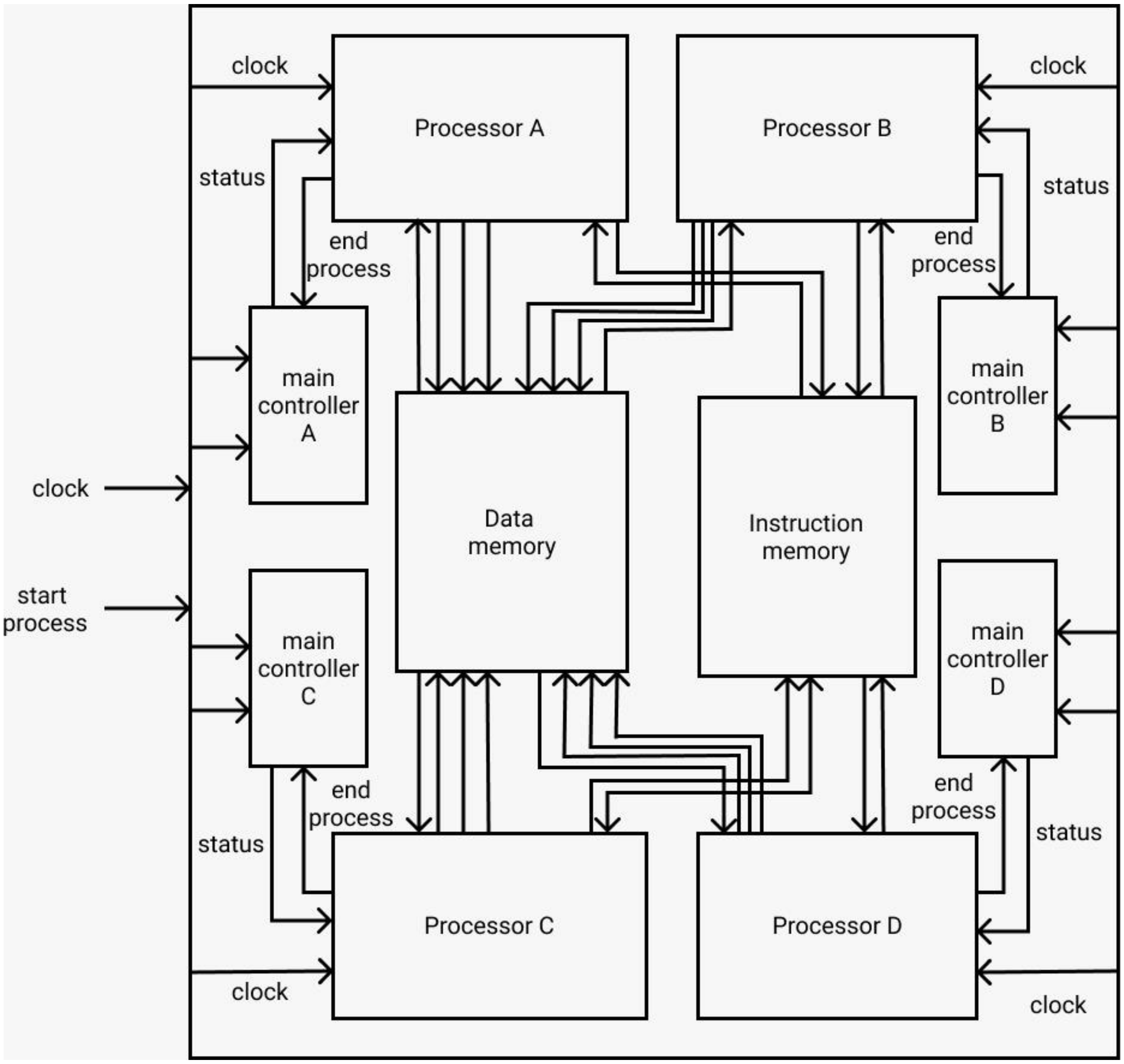
1.3 Tools / Software

- Modelsim – This is used as the simulation tool in our project
- Quartus –This is used to compile the Verilog HDL code
- Matlab – This is used to create input text files.

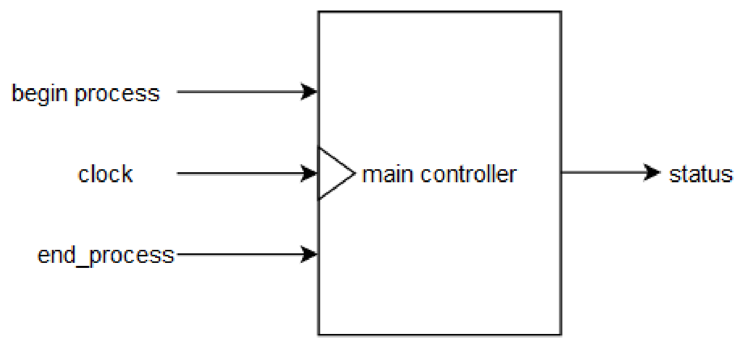
2.PROCEESOR DESIGN

2.1. Top-Module Architecture

- This below architecture explains the top hierarchy of our multi-processor design. Top Module consists of
  - Inputs – Clock, Start Process
  - 4 processors(A/B/C/D)
  - 4 main controllers for each processor
  - Data Memory
  - Instruction Memory



2.2 Main controller



The main control module is used to change the status of the process cycle. In this design there are two modes of operation. They are,

- Process – This mode is to indicate that the processor is running
- Finish – This mode indicates that the processor has finished with its operations.

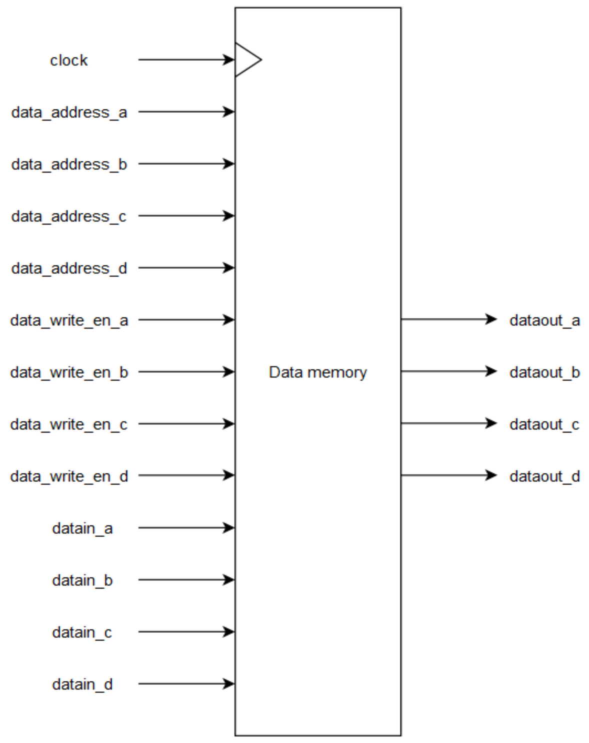
This module has inputs,

1. begin process [1 bit] – This input is based on the start_process input
2. Clock - for synchronization
3. end_process [1 bit] - Switch to notify process cycle is over ,this input is fed from the processor.

And output

1. Status – This output is connected to the processors. Processors carry out operations according to this output value.
   - If status = 01 then start the processor and perform the operations.
   - If status = 11 turn off the processor.

2.3 Data memory



This is the primary data memory which stores the matrix dimensions and the elements of the input matrix and the results of matrix multiplication.

Inputs of data memory are,

- Clock – for synchronization

- Data_address_a [16 bit] – This is fed from the processor. This 16 bit address value indicates the memory block from which the data has to be retrieved or saved.
- Data_address_b [16 bit] - This is fed from the processor. This 16 bit address value indicates the memory block from which the data has to be retrieved or saved.
- Data_address_c [16 bit] - This is fed from the processor. This 16 bit address value indicates the memory block from which the data has to be retrieved or saved.
- Data_address_d [16 bit] - This is fed from the processor. This 16 bit address value indicates the memory block from which the data has to be retrieved or saved.

- Data_write_en_a [1 bit] – This is fed from the processor. This indicates the datamemory to write the input data.
- Data_write_en_b[1 bit] – This is fed from the processor. This indicates the datamemory to write the input data.
- Data_write_en_c[1 bit] – This is fed from the processor. This indicates the datamemory to write the input data.
- Data_write_en_d[1 bit] -– This is fed from the processor. This indicates the datamemory to write the input data.

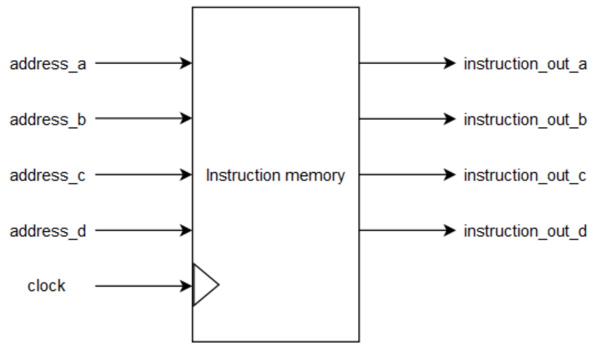
- datain_a [16 bits] - This is the data input from the processor
- datain_b [16 bits] - This is the data input from the processor
- datain_c [16 bits] - This is the data input from the processor
- datain_d [16 bits] - This is the data input from the processor

Outputs

- dataout_a[16 bits] -- This is the data output from data memory
- dataout_b[16 bits] -- This is the data output from data memory
- dataout_c[16 bits] -- This is the data output from data memory
- dataout_d[16 bits] -- This is the data output from data memory

In our design we have created 4 subunits inside the data memory. All the subunits are identical.

2.4 Instruction memory



Here, Instruction memory is implemented as a ROM where data can only be read. Data cannot be written as Instruction memory is a read only memory and embedded outside the processor. Instructions to be executed are stored in the instruction memory location and executed one after the other.

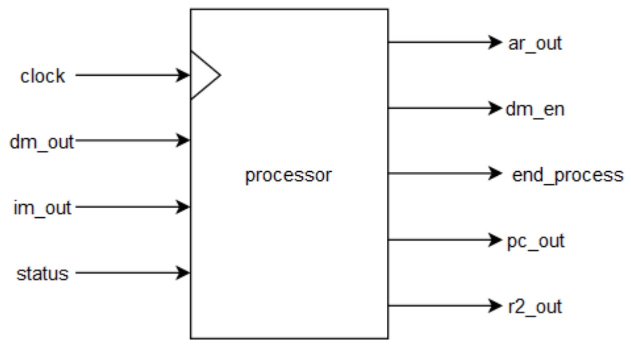
- Clock - Used for synchronization

- address_a [16 bits] – This is fed from the processor, to indicate the next instruction to be executed.
- address_b [16 bits] - This is fed from the processor, to indicate the next instruction to be executed.
- address_c [16 bits] - This is fed from the processor, to indicate the next instruction to be executed.
- address_d [16 bits] - This is fed from the processor, to indicate the next instruction to be executed.

- instruction_out_a [16 bits] – Instruction to be executed is sent through this output.
- instruction_out_b [16 bits] - Instruction to be executed is sent through this output.
- instruction_out_c [16 bits] - Instruction to be executed is sent through this output.
- instruction_out_d [16 bits] - Instruction to be executed is sent through this output.

The Instruction memory holds 90 instructions, each 16 bit wide and they are called by a 16 bit program counter (PC).

2.5 Processor



In the top module, there are altogether 4 processors.
Each Processor contains

- Instances of other modules
- Controlling instructions

Inputs of the processor

- clock – For synchronization
- dm_out [16 bits] – Data from the data memory
- im_out [16 bits] – Instructions from the instruction memory
- status [2 bits] – To indicate the status of process

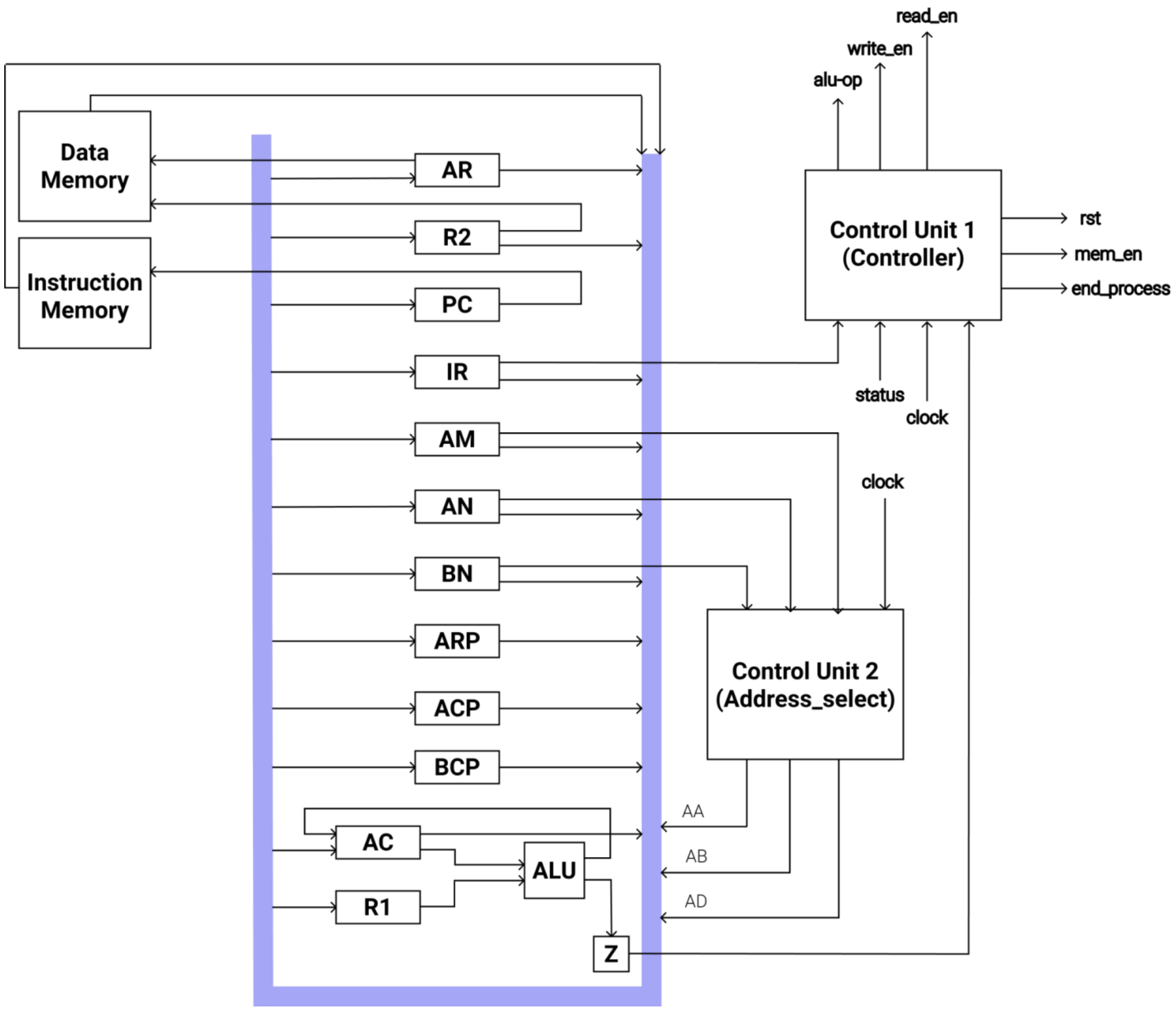
Outputs of the processor

- ar_out [16 bits] – Data memory address value from which the data to be read or written
- dm_en [1 bit] – Enable pin to write data to the memory
- end_process [1 bit] – To indicate that the allocated operations are over
- pc_out [16 bits] – Instruction memory address value which is to be read.
- r2_out [16 bits] – Data to be written in data memory

Instruction set architecture, instructions , micro-instructions architecture , modules and components of the processor are explained in the below sections.

2.6 Instruction Set Architecture

2.6.1 Data Path



2.6.2 Instructions

For the task of solving this specific problem, we implemented a custom defined set of instructions. The following table is a summary of the instructions.

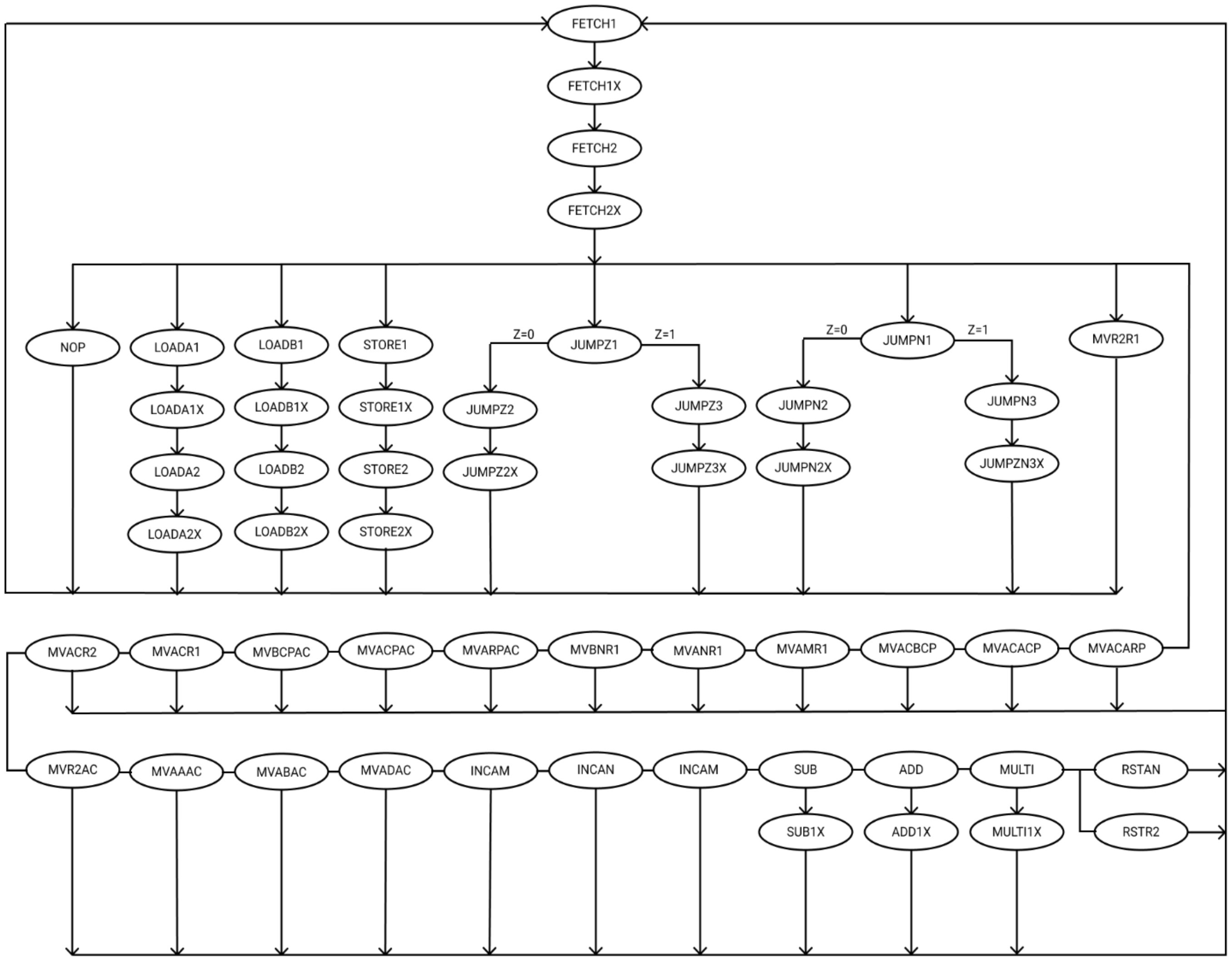
NO	Instruction	Operation	Description
1	nop	No operation	No operation
2	loada	AC ← DM	Retrieve data from memory
3	loadb	AC ← DM	Retrieve data from memory
4	store	DM ← R2	Write data to memory
5	jumpz	GO TO X IF Z=0	Control the order of instruction execution
6	jupmn	GO TO X IF Z=1	
7	mvr2r1	R1 ← R2	Move from R2 to R1
8	mvacarp	ARP ← AC	Move from AC to ARP
9	mvacacp	ACP ← AC	Move from AC to ACP
10	mvacbcp	BCP ← AC	Move from AC to BCP
11	incam	AM← AM+1	Increment AM by 1
12	incan	AN← AN+1	Increment AN by 1
13	incbn	BN← BN+1	Increment BN by 1
14	mvamr1	R1 ← AM	Move from AM to R1
15	mvanr1	R1 ← AN	Move from AN to R1
16	mvbnr1	R1 ← BN	Move from BN to R1
17	mvarpac	AC ← ARP	Move from ARP to AC
18	mvacpac	AC ← ACP	Move from ACP to AC
19	mvbcpac	AC ← BCP	Move from BCP to AC
20	mvaaac	AC ← AA	Move from AA to AC
21	mvabac	AC ← AB	Move from AB to AC
22	mvadac	AC ← AD	Move from AD to AC
23	mvacr1	R1 ← AC	Move from AC to R1
24	mvacr2	R2 ← AC	Move from AC to R2
25	mvr2ac	AC ← R2	Move from R2 to AC
26	multi	AC ← AC * R1	Multiplication
27	add	AC ← AC + R1	Addition
28	sub	AC ← AC - R1	Subtraction
29	rstan	AN ← 0	Reset AN
30	rstr2	R2 ← 0	Reset R2
31	endop	End Operation	End Operation

2.6.3 Micro Instruction Set Architecture

Instructions	Micro Instructions	
fetch	fetch	IR ← IM
	fetch1x	No operation
	fetch2	PC ← PC+1
	fetch2x	No operation
loada	loada	AR ← IM
	load2	AC ← DM
loadb	loadb	AR ← AC
	loadb1x	No operation
	loadb2	AC ← DM
store	store1	AR ← AD
	store2	DM ← R2
jumpz	jumpz1	ARP ← IM
	jumpz2	PC ← ARP
	jumpz2x	No operation
	jumpz3	PC ← PC +1
	jumpz3x	No operation
jumpn	jumpn1	ARP ← IM
	jumpn2	PC ← ARP
	jumpn2x	No operation
	jumpn3	PC ← PC +1
	jumpn3x	No operation
sub	sub	ALU ← AC – R1
	sub1x	AC ← ALU
add	add	AC ← AC + R1
	add1x	AC ← ALU
multi	multi	AC ← AC * R1
	multi1x	AC ← ALU
mvr2r1	mvr2r1	R1 ← R2
mvacarp	mvacarp	ARP ← AC
mvacacp	mvacacp	ACP ← AC
mvacbcp	mvacbcp	BCP ← AC
incam	incam	AM← AM+1
incan	incan	AN← AN+1
incbn	incbn	BN← BN+1
mvamr1	mvamr1	R1 ← AM
mvanr1	mvanr1	R1 ← AN
mvbnr1	mvbnr1	R1 ← BN
mvarpac	mvarpac	AC ← ARP
mvacpac	mvacpac	AC ← ACP
mvbcpac	mvbcpac	AC ← BCP
mvaaac	mvaaac	AC ← AA
mvabac	mvabac	AC ← AB

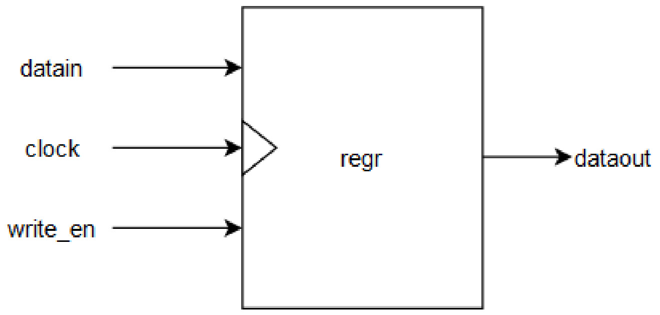
mvadac	mvadac	AC ← AD
mvacr1	mvacr1	R1 ← AC
mvacr2	mvacr2	R2 ← AC
mvr2ac	mvr2ac	AC ← R2
rstan	rstan	AN ← 0
rstr2	rstr2	R2 ← 0
endop	endop	End Operation

2.6.4. State Diagram



2.7 Components and Modules

2.7.1 Registers



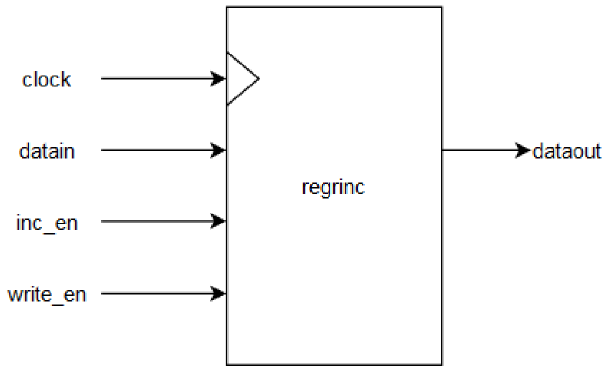
Registers are used to store the data for a short period of time during the processing cycle. Write enable pin decides whether to write the data from bus to the register.

- If write_en = 1 , then the data will be written to that particular register.

Data which was stored will available in the dataout pin. read_en decides whether to write the data to bus.

- If write_en = 1 , then the data will be written to that particular register.

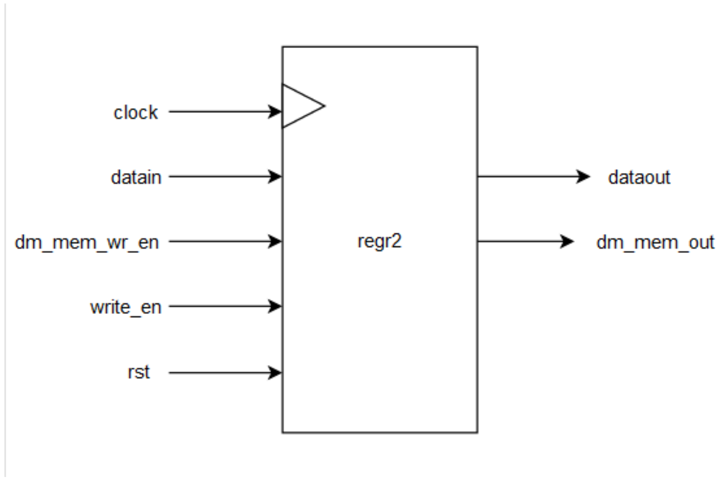
2.7.2 Registers with increment



This module does all the operations described in the above register module. In addition to those operations, this module increments the value in the register.

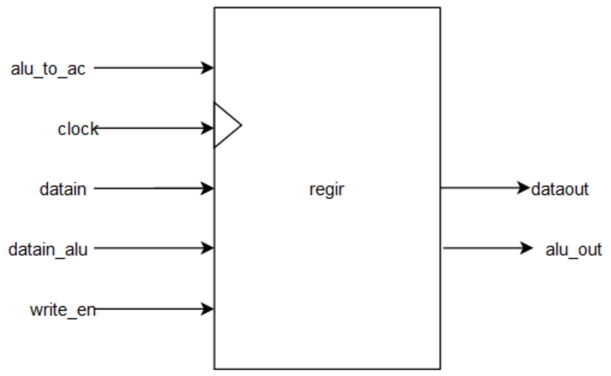
This increment procedure doesn't need ALU. inc_en has to be set to 1 during the positive edge of the clock in order to perform the increment operation.

2.7.3 Special Register R2



This is a special register. The uniqueness of this module is that data is written to the memory from this module. If the dm_mem_wr_en is set to 1, datain will be written to dm_mem_out. This dm_mem_out pin is connected to on the inputs of data memory. Apart from that, this module consists of resetting operation based on the rst pin. If the rst pin is set to 1, dataout will be reset to the value equal to zero.

2.7.4 Accumulator/AC

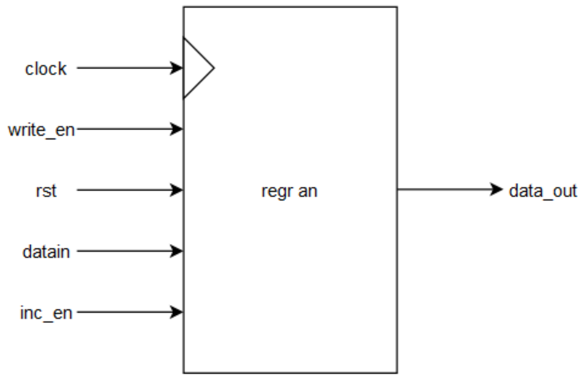


Accumulator is a special register which is similar to the register with increment. The uniqueness of this module lies in the fact that this module contains two inputs.

The one input is from bus and the other one is from ALU. Results of the ALU operation are stored temporarily in the accumulator register.

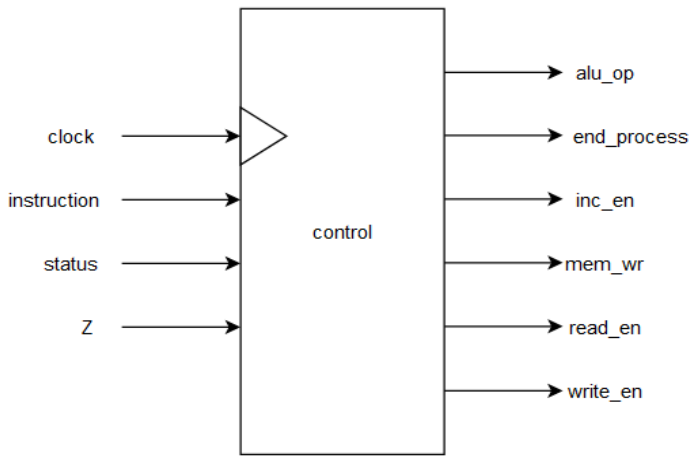
2.7.5 Special register AN

This is a special register which is similar to the register with increment. Apart from that, this module consists of resetting operation based on the rst pin. If the rst pin is set to 1, dataout will be reset to the value equal to zero



2.7.6 Control unit

This is a state machine that is used to control signal generations which controls the processing cycle of the processor.



Inputs are

- Clock – for synchronization
- Instruction [16 bits] – Instruction which is being executed
- Status [2 bits] – defines the mode of operation
- Z [16 bits] – defines whether ALU output is equal to zero

Outputs are

- alu_op[3 bits] – indicates the ALU operation
- end_process [2 bits] - indicates the end of operation
- inc_en [16 bits] - indicates the module which should perform increment operation
- mem_wr [1 bit] - indicates to write the data to memory
- read_en [4 bits] - indicates the module from which the value should be written to bus
- write_en [16 bits] - indicates the module to which the value should be written from bus
- rst [3 bits] - indicates the module which should perform reset operation

All the controlling outputs of the state machine are mentioned below

A) ALU Operation

This is a 3 bit data value which is used to control the ALU operations. The different operations of ALU are decided by the different values of the 3 bit signal. ALU operations and the corresponding signal given by the state machine are given in the following table.

Control signal	ALU operation
001	Multiplication
010	Addition
011	Subtraction

B) Write Enable

Write enable is a 16 bit signal which is connected to the registers. If the write enable of a particular register is set to 1, then the value in the bus will be written to that particular register. Multiple registers can be written at once with the same value in the bus because each bit of 16 bit write enable is connected to different registers The registers and the corresponding Write enable values are given in the following table.

IM	DM	ALU TO AC	AC	R1	BCP	ACP	ARP	BN	AN	AM	IR	PC	R2	AR
14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

C) Read Enable

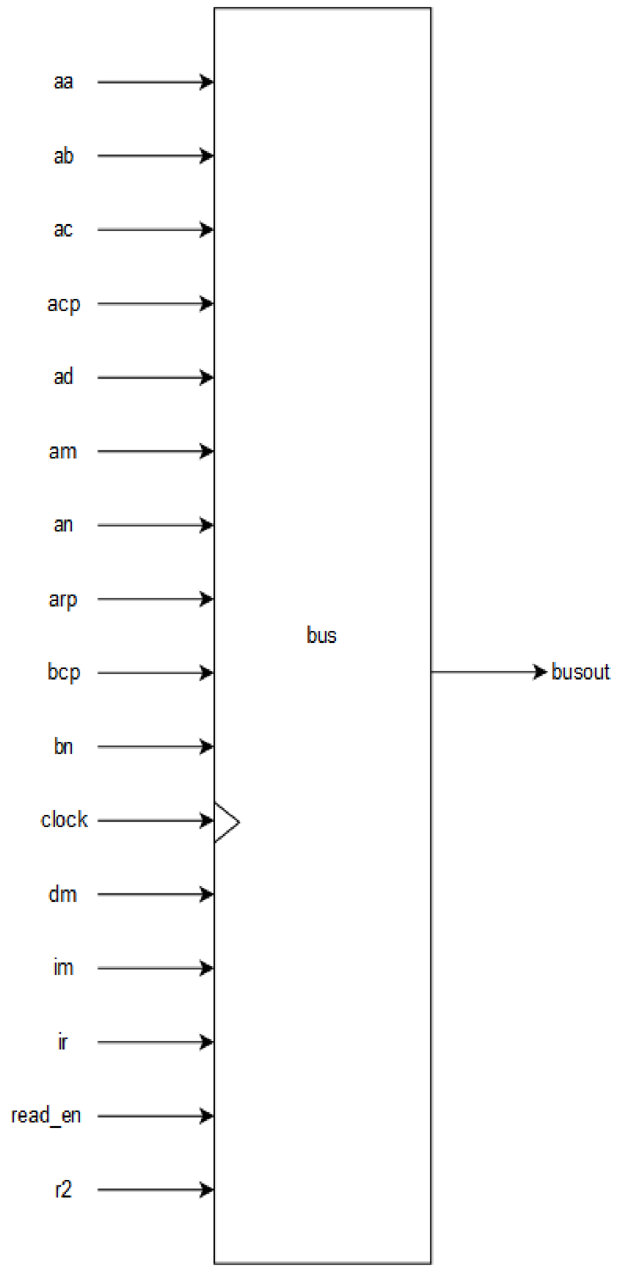
Not all but some of the register's outputs are connected to the bus. If the read enable of a particular register is set to 1, then the value in that particular register will be written to the bus. This is carried out by a 4 bit wide wire enabling 16 different unique registers to be called. This read enable pin is designed in a way such that in a given time only one register's value can be written to the bus. .The register values and corresponding values of the read enable pin is given below.

Read enable value	Register
0000	
0001	AA
0010	R2
0011	AB
0100	IR
0101	AM
0110	AN
0111	BN
1000	ARP
1001	ACP
1010	BCP
1011	R1
1100	AC
1101	AD
1110	DM
1111	IM

D) Increment Enable

Increment Enable is a 16 bit pin and each bit in the array of 16 bits represents different registers. If the increment enable pin of a particular register is set to 1, then the value in that particular register will be incremented by 1. The same allocation order as write enable above was used.

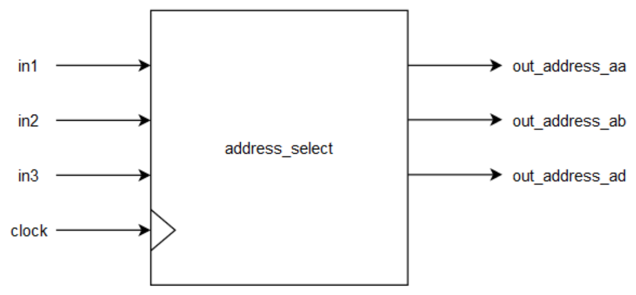
2.7.7 BUS



In our design bus is the module that is responsible for data transfer within the processor and gets data from both the data and instruction memory. In our processor design the bus is connected to registers AR, R2, PC, IR, AM, AN, BN, ARP, ACP, BCP, AC, R1 and outside the processor it is connected to instruction memory (IM) and data memory (DM).

Bus is not connected to neither of the two control units. Data in the registers is written to bus according to read_en pin from control unit. Data in the bus is written to registers according the write_en pin from control unit.

2.7.8 Address select



This module is created to supply memory addresses during data retrieval and writing processes. This module supplies suitable memory addresses with the help of matrix indexes which are present in a particular time of process.

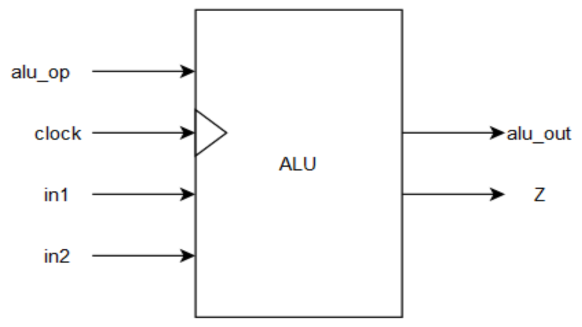
Inputs for this module are (consider the input matrices to be A and B)

- clock
- In1 – index of matrix A's row
- In2 - index of matrix A's column
- In3 - index of matrix A's column

Outputs of this module are

- Out_address_aa - memory address which holds an element of matrix A
- Out_address_ab - memory address which holds an element of matrix b
- Out_address_ad- memory address to which the data should be written

2.7.9 ARITHMETIC LOGIC UNIT (ALU)



This module performs all the arithmetic operations.
Inputs of this module are

- Alu_op [3 bits] - decides the operation to be done
- clock - for synchronization
- In1 [16 bits] - input from R1 register
- In2 [16 bits] - input from AC

Outputs of this module are

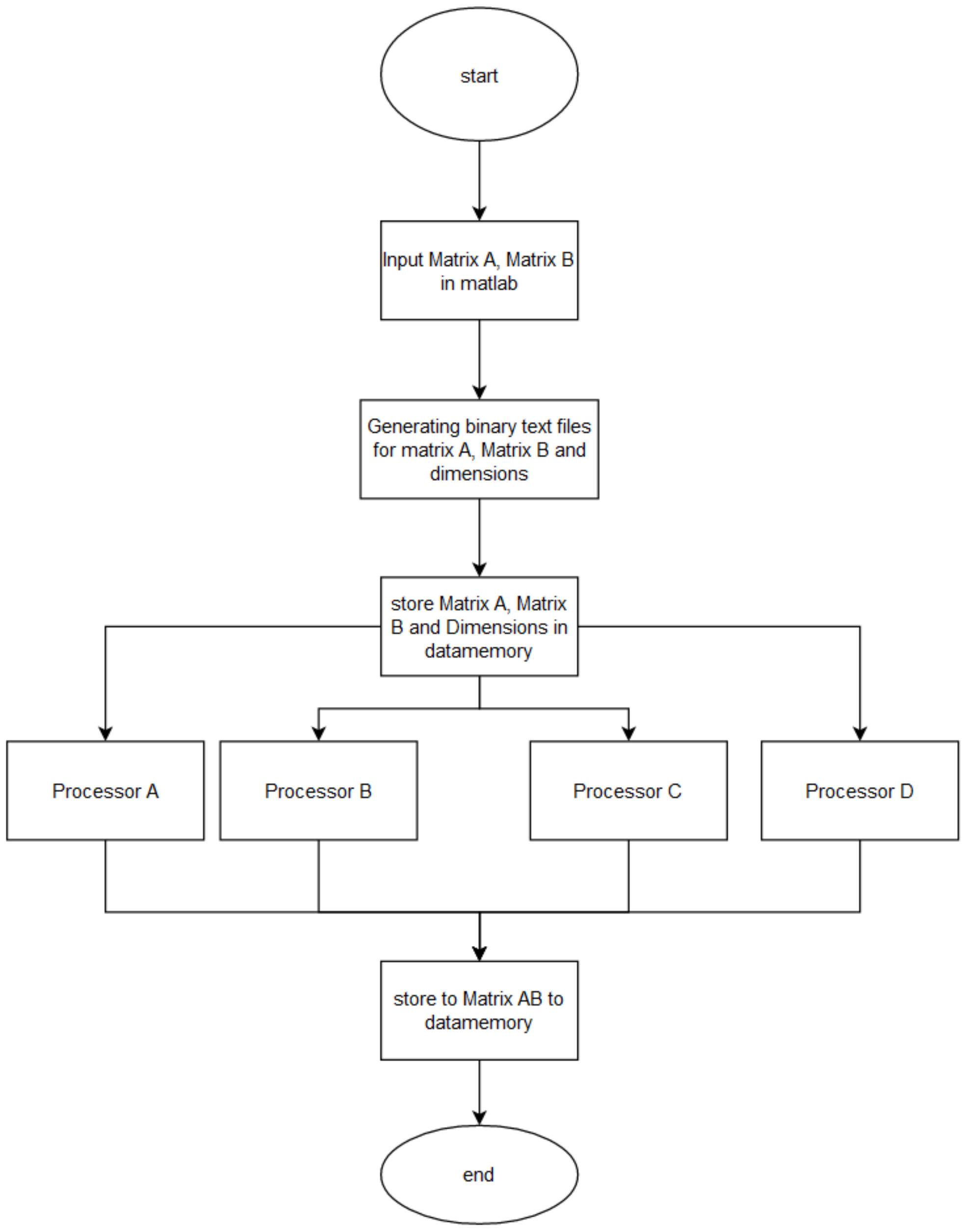
- alu_out [16 bits] - result of ALU operation
- Z – Flag [16 bits] - if alu_out is zero then z flag is set to zero or else z flag will be set to 1.

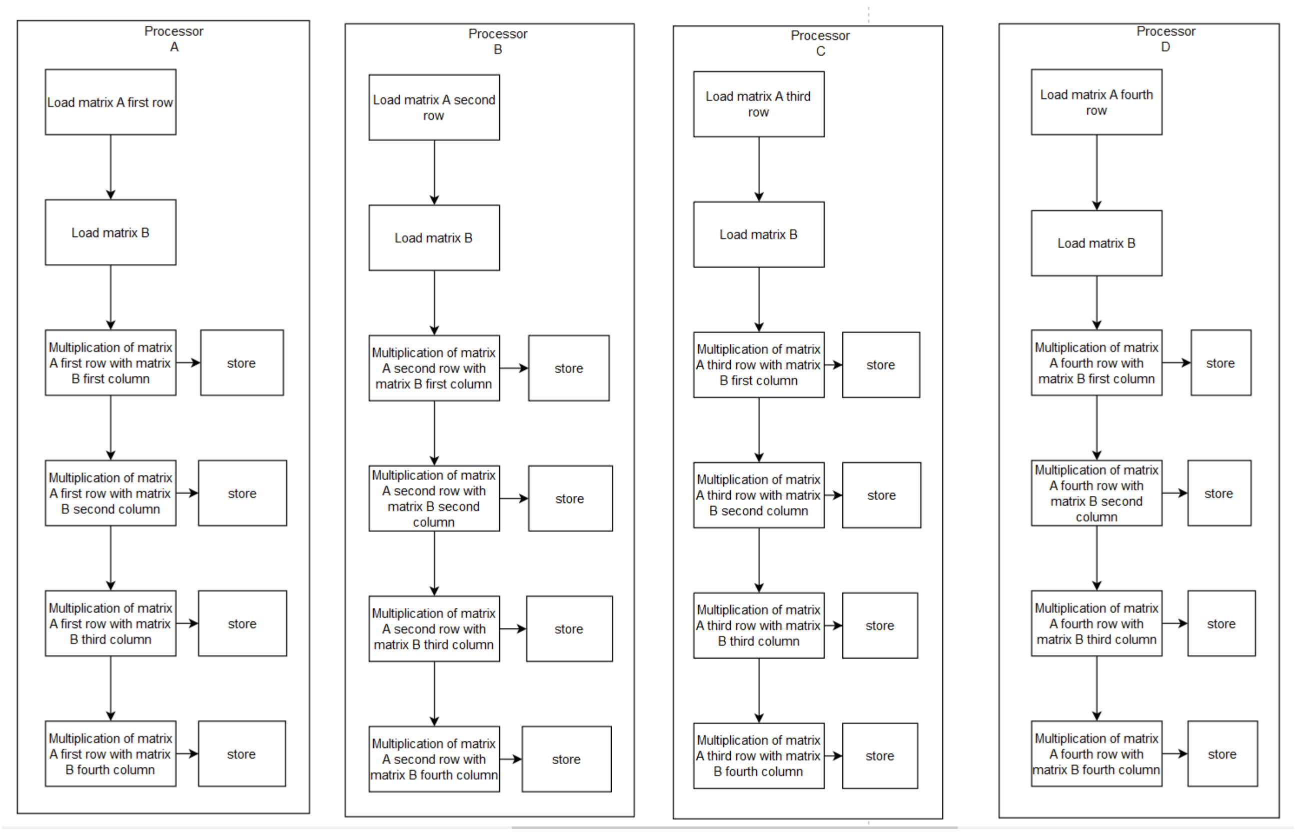
alu_out is connected back to AC.

The alu_op decides the operations of the ALU as follows

Control signal	Operation	ALU operation
001	Multiplication	alu_out = in1 * in2
010	Addition	alu_out = in1 + in2
011	Subtraction	alu_out = in1 - in2

3. Algorithm


start
Input Matrix A, Matrix B in matlab
Generating binary text files for matrix A, Matrix B and dimensions
store Matrix A, Matrix B and Dimensions in datamemory
Processor A
Processor B
Processor C
Processor D
store to Matrix AB to datamemory
end


Processor A
Load matrix A first row
Load matrix B
Multiplication of matrix A first row with matrix B first column
store
Multiplication of matrix A first row with matrix B second column
store
Multiplication of matrix A first row with matrix B third column
store
Multiplication of matrix A first row with matrix B fourth column
store
Processor B
Load matrix A second row
Load matrix B
Multiplication of matrix A second row with matrix B first column
store
Multiplication of matrix A second row with matrix B second column
store
Multiplication of matrix A second row with matrix B third column
store
Multiplication of matrix A second row with matrix B fourth column
store
Processor C
Load matrix A third row
Load matrix B
Multiplication of matrix A third row with matrix B first column
store
Multiplication of matrix A third row with matrix B second column
store
Multiplication of matrix A third row with matrix B third column
store
Multiplication of matrix A third row with matrix B fourth column
store
Processor D
Load matrix A fourth row
Load matrix B
Multiplication of matrix A fourth row with matrix B first column
store
Multiplication of matrix A fourth row with matrix B second column
store
Multiplication of matrix A fourth row with matrix B third column
store
Multiplication of matrix A fourth row with matrix B fourth column
store

Matrix A row 1

A[0,0]	A[0,1]	A[0,2]	A[0,3]
A[1,0]	A[1,1]	A[1,2]	A[1,3]
A[2,0]	A[2,1]	A[2,2]	A[2,3]
A[3,0]	A[3,1]	A[3,2]	A[3,3]

X

Matrix B (4 x4)

B[0,0]	B[0,1]	B[0,2]	B[0,3]
B[1,0]	B[1,1]	B[1,2]	B[1,3]
B[2,0]	B[2,1]	B[2,2]	B[2,3]
B[3,0]	B[3,1]	B[3,2]	B[3,3]

=

Matrix AB(4 x4)

AB[0,0]	AB[0,1]	AB[0,2]	AB[0,3]
AB[1,0]	AB[1,1]	AB[1,2]	AB[1,3]
AB[2,0]	AB[2,1]	AB[2,2]	AB[2,3]
AB[3,0]	AB[3,1]	AB[3,2]	AB[3,3]

At Processor A ,multiplication of matrix A first row with matrix B

A[0,0]	A[0,1]	A[0,2]	A[0,3]

X

Matrix B (4 x4)

B[0,0]	B[0,1]	B[0,2]	B[0,3]
B[1,0]	B[1,1]	B[1,2]	B[1,3]
B[2,0]	B[2,1]	B[2,2]	B[2,3]
B[3,0]	B[3,1]	B[3,2]	B[3,3]

At Processor B ,Multiplication of matrix A second row with matrixB

A[1,0]	A[1,1]	A[1,2]	A[1,3]

X

Matrix B (4 x4)

B[0,0]	B[0,1]	B[0,2]	B[0,3]
B[1,0]	B[1,1]	B[1,2]	B[1,3]
B[2,0]	B[2,1]	B[2,2]	B[2,3]
B[3,0]	B[3,1]	B[3,2]	B[3,3]

At Processor C , Multiplication of matrix A third row with matrix B

A[2,0]	A[2,1]	A[2,2]	A[2,3]

X

Matrix B (4 x4)

B[0,0]	B[0,1]	B[0,2]	B[0,3]
B[1,0]	B[1,1]	B[1,2]	B[1,3]
B[2,0]	B[2,1]	B[2,2]	B[2,3]
B[3,0]	B[3,1]	B[3,2]	B[3,3]

At Processor D ,Multiplication of matrix A fourth row with matrix B

A[3,0]	A[3,1]	A[3,2]	A[3,3]

X

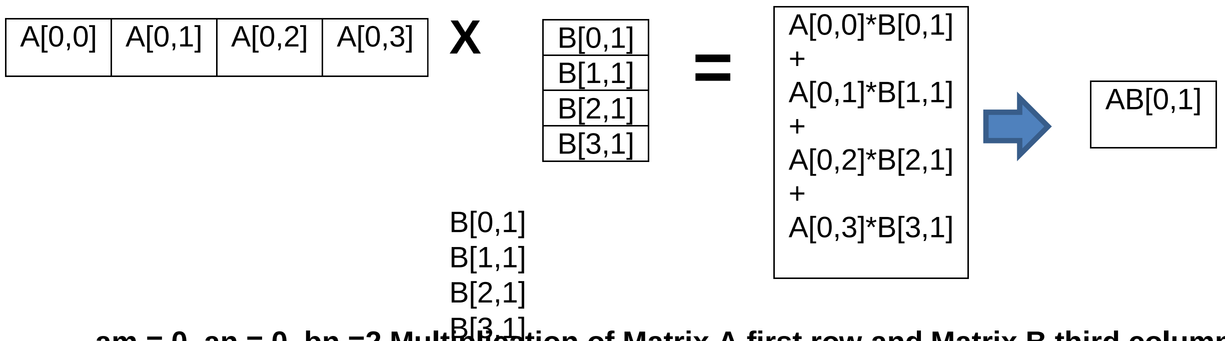
Matrix B (4 x4)

B[0,0]	B[0,1]	B[0,2]	B[0,3]
B[1,0]	B[1,1]	B[1,2]	B[1,3]
B[2,0]	B[2,1]	B[2,2]	B[2,3]
B[3,0]	B[3,1]	B[3,2]	B[3,3]

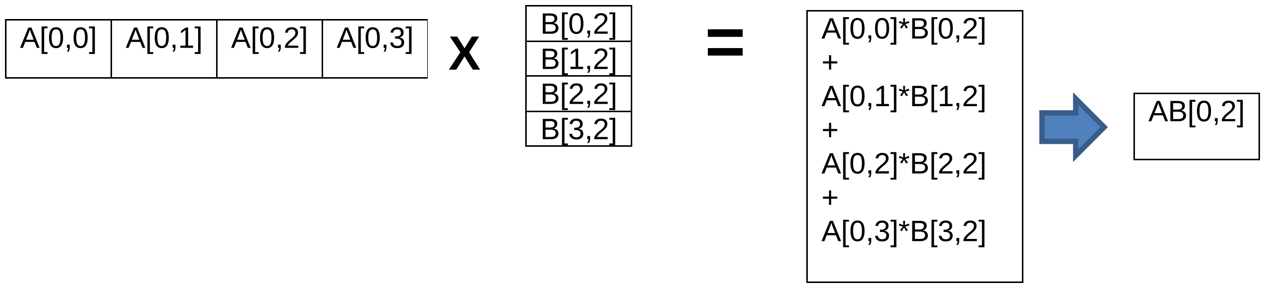
At processor A (am = 0, an = 0, bn = 0) Multiplication of Matrix A first row and Matrix B first column

$$\begin{array}{|c|c|c|c|}\hline A[0,0] & A[0,1] & A[0,2] & A[0,3] \\ \hline\end{array} \times \begin{array}{|c|}\hline B[0,0] \\ \hline B[1,0] \\ \hline B[2,0] \\ \hline B[3,0] \\ \hline\end{array} = \boxed{\begin{array}{l} A[0,0]*B[0,0] + \\ A[0,1]*B[1,0] + \\ A[0,2]*B[2,0] + \\ A[0,3]*B[3,0] \end{array}} \Rightarrow \boxed{AB[0,0]}$$

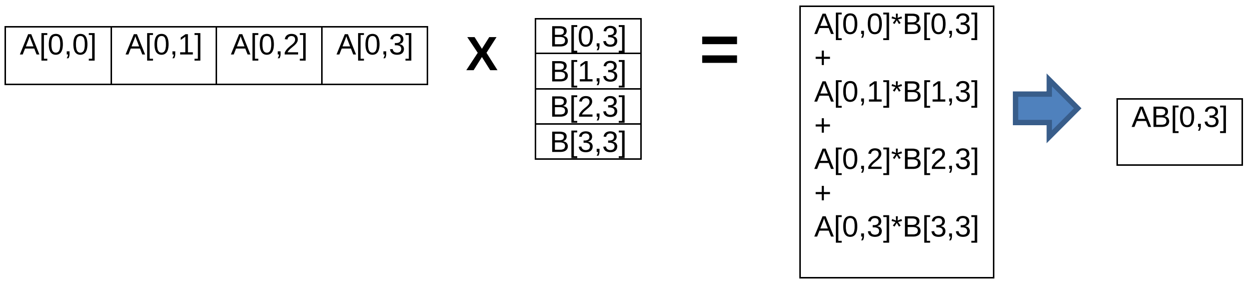
am = 0, an = 0, bn = 1 Multiplication of Matrix A first row and Matrix B second column


A[0,0] A[0,1] A[0,2] A[0,3]
X
B[0,1] B[1,1] B[2,1] B[3,1]
=
A[0,0]*B[0,1] + A[0,1]*B[1,1] + A[0,2]*B[2,1] + A[0,3]*B[3,1]
AB[0,1]
B[0,1]
B[1,1]
B[2,1]
B[3,1]

am = 0, an = 0, bn =2 Multiplication of Matrix A first row and Matrix B third column


A[0,0] A[0,1] A[0,2] A[0,3]
X
B[0,2] B[1,2] B[2,2] B[3,2]
=
A[0,0]*B[0,2] + A[0,1]*B[1,2] + A[0,2]*B[2,2] + A[0,3]*B[3,2]
AB[0,2]

am = 0, an = 0, bn =3 Multiplication of Matrix A first row and Matrix B fourth column


A[0,0] A[0,1] A[0,2] A[0,3]
X
B[0,3] B[1,3] B[2,3] B[3,3]
=
A[0,0]*B[0,3] + A[0,1]*B[1,3] + A[0,2]*B[2,3] + A[0,3]*B[3,3]
AB[0,3]

At processor A for instant [am = 0, an = 0, bn = 0]

am = 0, an = 0, bn = 0

A[0,0]*B[0,0]

am = 0, an = 1, bn = 0

A[0,1]*B[1,0]

am = 0, an = 2, bn = 0

A[0,2]*B[2,0]

am = 0, an = 3, bn = 0

A[0,3]*B[3,0]

4. Verilog Implementation

4.1 Modelsim Simulation

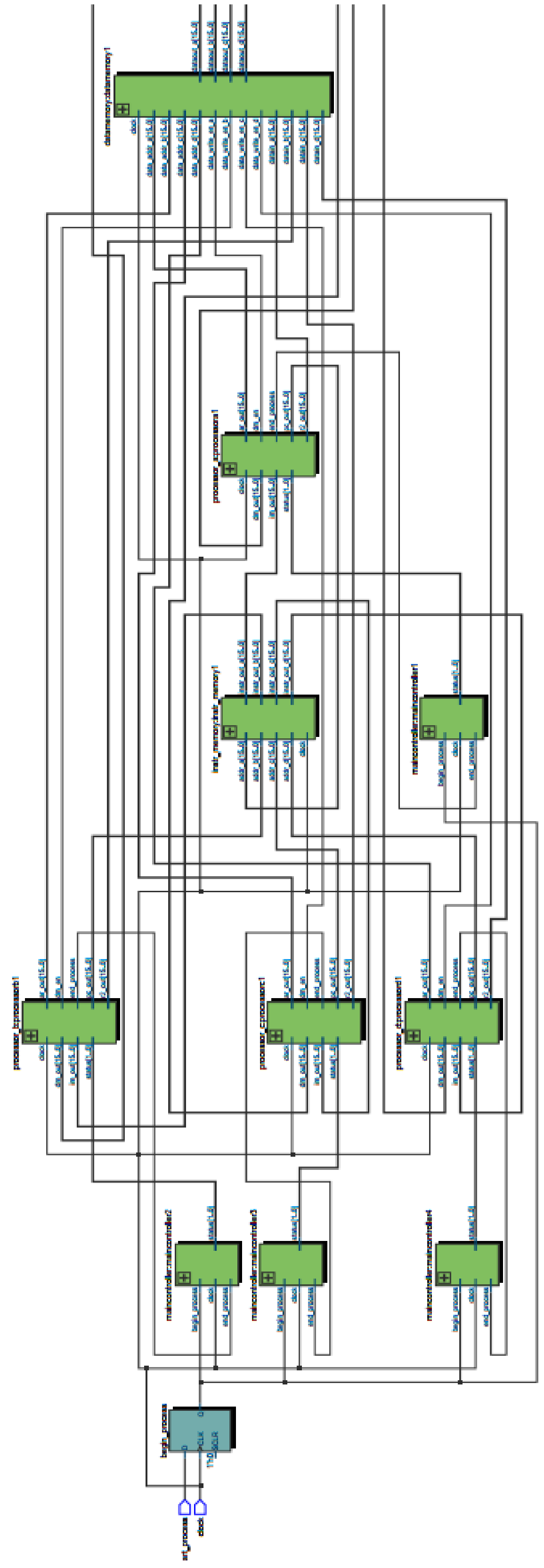
We used Modelsim Altera software as the simulation tool for the designing process. With this help of software,

- We independently tested each module with its respective test benche.
- Modules were combined in a hierarchical manner and tested to ensure the expected functionality.
- We efficiently used the waveform window in the Modelsim software to analyse the outputs from test benches.
- We efficiently debugged most of the issues.

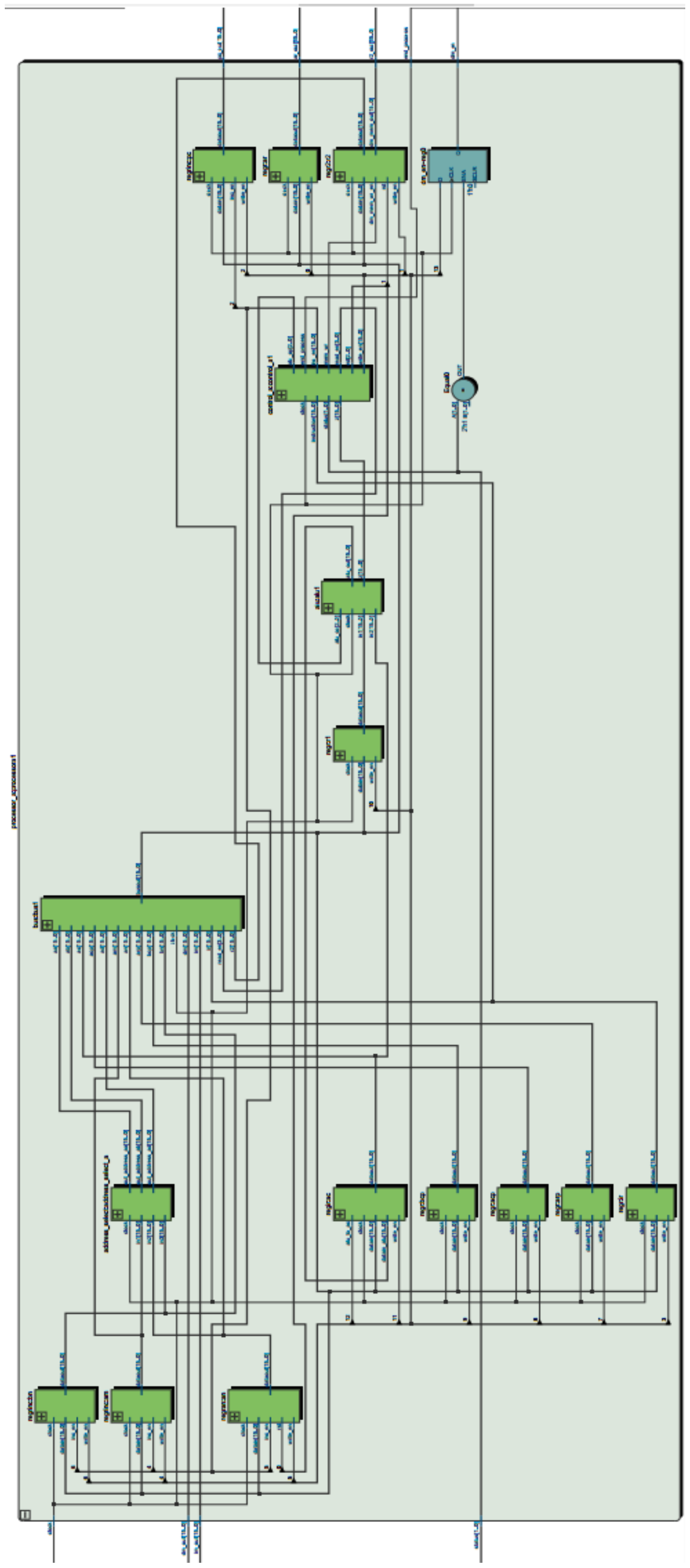
4.2 Quartus Implementation

We chose Verilog HDL for developing the code and Quartus Software to compile the developed Verilog code.

4.3 RTL Design Visualization – Top module



4.4 RTL Design Visualization – Proccesor



5 . Results

MATRIX		TIME (ps)	CLOCK CYCLES
Dimension of Matrix A	Dimension of Matrix B		
[4,4]	[4,4]	168850	1689
[3,3]	[3,3]	109150	1092
[2,2]	[2,2]	65250	653

APPENDIX-A

I. ASSEMBLY LANGUAGE INSTRUCTION

```
ram[0] = 16'd4096+loada;//////////// 1st core
ram[1] = 16'd0;
ram[2] = 16'd4096+mvacarp;
ram[3] = 16'd4096+loada;
ram[4] = 16'd2;
ram[5] = 16'd4096+mvacacp;
ram[6] = 16'd4096+loada;
ram[7] = 16'd2;
ram[8] = 16'd4096+mvacbcp;
ram[9] = 16'd4096+nop;
ram[10] = 16'd4096+mvamr1;
ram[11] = 16'd4096+mvarpac;
ram[12] = 16'd4096+sub;
ram[13] = 16'd4096+jumpz;
ram[14] = 16'd87;
ram[15] = 16'd8192+loada;///////////2nd core
ram[16] = 16'd0;
ram[17] = 16'd8192+mvacarp;
ram[18] = 16'd8192+loada;
ram[19] = 16'd1;
ram[20] = 16'd8192+mvacacp;
ram[21] = 16'd8192+loada;
ram[22] = 16'd2;
ram[23] = 16'd8192+mvacbcp;
ram[24] = 16'd8192+incam;
ram[25] = 16'd8192+mvamr1;
ram[26] = 16'd8192+mvarpac;
ram[27] = 16'd8192+sub;
```

```
ram[28] = 16'd8192+jumpz;
ram[29] = 16'd88;
ram[30] = 16'd16384+loada;///////////3rd core
ram[31] = 16'd0;
ram[32] = 16'd16384+mvacarp;
ram[33] = 16'd16384+loada;
ram[34] = 16'd1;
ram[35] = 16'd16384+mvacacp;
ram[36] = 16'd16384+loada;
ram[37] = 16'd2;
ram[38] = 16'd16384+mvacbcp;
ram[39] = 16'd16384+incam;
ram[40] = 16'd16384+incam;
ram[41] = 16'd16384+mvamr1;
ram[42] = 16'd16384+mvarpac;
ram[43] = 16'd16384+sub;
ram[44] = 16'd16384+jumpz;
ram[45] = 16'd89;
ram[46] = 16'd32768+loada;//////////4th core
ram[47] = 16'd0;
ram[48] = 16'd32768+mvacarp;
ram[49] = 16'd32768+loada;
ram[50] = 16'd1;
ram[51] = 16'd32768+mvacacp;
ram[52] = 16'd32768+loada;
ram[53] = 16'd2;
ram[54] = 16'd32768+mvacbcp;
ram[55] = 16'd32768+incam;
ram[56] = 16'd32768+incam;
ram[57] = 16'd32768+incam;
ram[58] = 16'd32768+mvamr1;
ram[59] = 16'd32768+mvarpac;
ram[60] = 16'd32768+sub;
ram[61] = 16'd32768+jumpz;
ram[62] = 16'd90;
ram[63] = 16'd61440+mvaaac;///////// all core
ram[64] = 16'd61440+loadb;
ram[65] = 16'd61440+mvacr1;
ram[66] = 16'd61440+mvabac;
ram[67] = 16'd61440+loadb;
ram[68] = 16'd61440+multi;
ram[69] = 16'd61440+mvr2r1;
ram[70] = 16'd61440+add;
ram[71] = 16'd61440+mvacr2;
```

```
ram[72] = 16'd61440+incan;
ram[73] = 16'd61440+mvanr1;
ram[74] = 16'd61440+mvacpac;
ram[75] = 16'd61440+sub ;
ram[76] = 16'd61440+jumpn;
ram[77] = 16'd63;
ram[78] = 16'd61440+store;
ram[79] = 16'd61440+incbn;
ram[80] = 16'd61440+rstan;
ram[81] = 16'd61440+rstr2;
ram[82] = 16'd61440+mvbnr1;
ram[83] = 16'd61440+mvbcpac;
ram[84] = 16'd61440+sub;
ram[85] = 16'd61440+jumpn;
ram[86] = 16'd63;
ram[87] = 16'd4096+endop;
ram[88] = 16'd8192+endop;
ram[89] = 16'd16384+endop;
ram[90] = 16'd32768+endop;
```

APPENDIX-B

I. Verilog Codes for the Processor Design

1. Ultimate

```
module ultimate(clock,start_process);

                    wire [ 15: 0] dm_out_a;
                    wire [ 15: 0] dm_out_b;
                    wire [ 15: 0] dm_out_c;
                    wire [ 15: 0] dm_out_d;

                    wire [ 15: 0] im_out_a;
                    wire [ 15: 0] im_out_b;
                    wire [ 15: 0] im_out_c;
                    wire [ 15: 0] im_out_d;

                    wire [ 15: 0] r2_out_a;
                    wire [ 15: 0] r2_out_b;
                    wire [ 15: 0] r2_out_c;
                    wire [ 15: 0] r2_out_d;

                    wire dm_en_a;
                    wire dm_en_b;
                    wire dm_en_c;
                    wire dm_en_d;

                    wire end_process_a;
                    wire end_process_b;
                    wire end_process_c;
                    wire end_process_d;

                    wire [ 15: 0] ar_out_a;
                    wire [ 15: 0] ar_out_b;
                    wire [ 15: 0] ar_out_c;
                    wire [ 15: 0] ar_out_d;

                    wire [ 15: 0] pc_out_a;
                    wire [ 15: 0] pc_out_b;
                    wire [ 15: 0] pc_out_c;
                    wire [ 15: 0] pc_out_d;
                    wire end_process;

                    wire [ 1: 0] status_a;
                    wire [ 1: 0] status_b;
                    wire [ 1: 0] status_c;
                    wire [ 1: 0] status_d;

                    input wire start_process;
                    input wire clock;
                    reg begin_process;

                    wire [ 15: 0] datain_a;
                    wire data_write_en_a;
                    wire [ 15: 0] data_addr_a;

                    wire [ 15: 0] datain_b;
                    wire data_write_en_b;
                    wire [ 15: 0] data_addr_b;

                    wire [ 15: 0] datain_c;
                    wire data_write_en_c;
                    wire [ 15: 0] data_addr_c;

                    wire [ 15: 0] datain_d;
                    wire data_write_en_d;
                    wire [ 15: 0] data_addr_d;

                    wire [ 15: 0] instr_addr_a;
                    wire [ 15: 0] instr_addr_b;
                    wire [ 15: 0] instr_addr_c;
                    wire [ 15: 0] instr_addr_d;


                    always @(posedge clock) begin


                                        if (start_process == 1) begin
                                          begin_process <=1;
                                        end

                         else begin
                                          begin_process <=0;
                                        end
                    end

                    instr_memory instr_memory1(. clock(clock),. addr_a(pc_out_a),.
addr_b(pc_out_b),. addr_c(pc_out_c),. addr_d(pc_out_d),. instr_out_a(instr_addr_a),.
instr_out_b(instr_addr_b),. instr_out_c(instr_addr_c),. instr_out_d(instr_addr_d));

                    maincontroller maincontroller1(. clock(clock),.
begin_process(begin_process),. end_process(end_process_a),. status(status_a));

                    maincontroller maincontroller2(. clock(clock),.
begin_process(begin_process),. end_process(end_process_b),. status(status_b));

                    maincontroller maincontroller3(. clock(clock),.
begin_process(begin_process),. end_process(end_process_c),. status(status_c));

                    maincontroller maincontroller4(. clock(clock),.
begin_process(begin_process),. end_process(end_process_d),. status(status_d));

                    datamemory datamemory1(. clock(clock),. datain_a(r2_out_a),.
data_write_en_a(dm_en_a),. data_addr_a(ar_out_a),. datain_b(r2_out_b),.
data_write_en_b(dm_en_b),. data_addr_b(ar_out_b),. datain_c(r2_out_c),.
data_write_en_c(dm_en_c),. data_addr_c(ar_out_c),. datain_d(r2_out_d),.
data_write_en_d(dm_en_d),. data_addr_d(ar_out_d),. dataout_a(dm_out_a),.
dataout_b(dm_out_b),. dataout_c(dm_out_c),. dataout_d(dm_out_d));

                    processor_a processora1 (. clock(clock),. dm_out(dm_out_a),.
im_out(instr_addr_a),. dm_en(dm_en_a),. pc_out(pc_out_a),. ar_out(ar_out_a),.
status(status_a),. r2_out(r2_out_a),. end_process(end_process_a));

                    processor_b processorb1 (. clock(clock),. dm_out(dm_out_b),.
im_out(instr_addr_b),. dm_en(dm_en_b),. pc_out(pc_out_b),. ar_out(ar_out_b),.
status(status_b),. r2_out(r2_out_b),. end_process(end_process_b));

                    processor_c processorc1 (. clock(clock),. dm_out(dm_out_c),.
im_out(instr_addr_c),. dm_en(dm_en_c),. pc_out(pc_out_c),. ar_out(ar_out_c),.
status(status_c),. r2_out(r2_out_c),. end_process(end_process_c));

                    processor_d processord1 (. clock(clock),. dm_out(dm_out_d),.
im_out(instr_addr_d),. dm_en(dm_en_d),. pc_out(pc_out_d),. ar_out(ar_out_d),.
status(status_d),. r2_out(r2_out_d),. end_process(end_process_d));

endmodule
```

2. Main controller

```
module maincontroller(

                        input clock,
                        input begin_process,
                        input end_process,
                        output reg [1:0] status);

                        reg [ 1: 0] present = 2'b01;
                        reg [ 1: 0] next = 2'b01;
                        parameter
                        process = 2'b01,
                        alldone = 2'b11;

                        always @(posedge clock)
                        present <= next;

                        always @(present or begin_process or end_process)

                        case(present)

                                    process: begin

                                    status <= 2'b01;

                                    if (end_process)

                                        next<=alldone;
                                    else
                                        next<=process;
                                    end


                                    alldone: begin

                                    status <= 2'b11;

                                    next<=alldone;

                                    end

                                    endcase
endmodule
```

3. Instruction memory

```
module instr_memory(
                              input clock,
                              input [ 15: 0] addr_a,
                              input [ 15: 0] addr_b,
                              input [ 15: 0] addr_c,
                              input [ 15: 0] addr_d,
                              output reg [ 15: 0] instr_out_a,
                              output reg [ 15: 0] instr_out_b,
                              output reg [ 15: 0] instr_out_c,
                              output reg [ 15: 0] instr_out_d);
                              reg [ 15: 0] ram [ 94: 0];
                              parameter
                              loada = 16'd3,
                              loadb = 16'd5,
                              store = 16'd7,
                              jumpz = 16'd9,
                              jumpn = 16'd13,
                              mvr2r1 = 16'd20,
                              mvacarp = 16'd23,
                              mvacacp = 16'd24,
                              mvacbcp = 16'd25,
                              incam = 16'd26,
                              incan = 16'd27,
                              incbn = 16'd28,
                              mvamr1 = 16'd29,
                              mvanr1 = 16'd30,
                              mvbnr1 = 16'd31,
                              mvarpac = 16'd32,
                              mvacpac = 16'd33,
                              mvbcpac = 16'd34,
                              mvaaac = 16'd35,
                              mvabac = 16'd36,
                              mvadac = 16'd37,
                              mvacr1 = 16'd38,
                              mvacr2 = 16'd39,
                              mvr2ac = 16'd40,
                              multi = 16'd41,
                              add = 16'd42,
                              sub = 16'd43,
                              rstan = 16'd45,
                              rstr2 = 16'd46,
                              nop = 16'd47,
endop = 16'd48;

initial begin

                ram[0] = 16'd4096+loada;//////////// 1st core
                ram[1] = 16'd0;
                ram[2] = 16'd4096+mvacarp;
                ram[3] = 16'd4096+loada;
                ram[4] = 16'd2;
                ram[5] = 16'd4096+mvacacp;
                ram[6] = 16'd4096+loada;
                ram[7] = 16'd2;
                ram[8] = 16'd4096+mvacbcp;
    ram[9] = 16'd4096+nop;
                ram[10] = 16'd4096+mvamr1;
                ram[11] = 16'd4096+mvarpac;
                ram[12] = 16'd4096+sub;
                ram[13] = 16'd4096+jumpz;
                ram[14] = 16'd87;
                ram[15] = 16'd8192+loada;///////////2nd core
                ram[16] = 16'd0;
                ram[17] = 16'd8192+mvacarp;
                ram[18] = 16'd8192+loada;
                ram[19] = 16'd1;
                ram[20] = 16'd8192+mvacacp;
                ram[21] = 16'd8192+loada;
                ram[22] = 16'd2;
                ram[23] = 16'd8192+mvacbcp;
                ram[24] = 16'd8192+incam;
                ram[25] = 16'd8192+mvamr1;
                ram[26] = 16'd8192+mvarpac;
                ram[27] = 16'd8192+sub;
                ram[28] = 16'd8192+jumpz;
                ram[29] = 16'd88;
                ram[30] = 16'd16384+loada;///////////3rd core
                ram[31] = 16'd0;
                ram[32] = 16'd16384+mvacarp;
                ram[33] = 16'd16384+loada;
                ram[34] = 16'd1;
                ram[35] = 16'd16384+mvacacp;
                ram[36] = 16'd16384+loada;
                ram[37] = 16'd2;
                ram[38] = 16'd16384+mvacbcp;
                ram[39] = 16'd16384+incam;
```

```
ram[40] = 16'd16384+incam;
ram[41] = 16'd16384+mvamr1;
ram[42] = 16'd16384+mvarpac;
ram[43] = 16'd16384+sub;
ram[44] = 16'd16384+jumpz;
ram[45] = 16'd89;
ram[46] = 16'd32768+loada;//////////4th core
ram[47] = 16'd0;
ram[48] = 16'd32768+mvacarp;
ram[49] = 16'd32768+loada;
ram[50] = 16'd1;
ram[51] = 16'd32768+mvacacp;
ram[52] = 16'd32768+loada;
ram[53] = 16'd2;
ram[54] = 16'd32768+mvacbcp;
ram[55] = 16'd32768+incam;
ram[56] = 16'd32768+incam;
ram[57] = 16'd32768+incam;
ram[58] = 16'd32768+mvamr1;
ram[59] = 16'd32768+mvarpac;
ram[60] = 16'd32768+sub;
ram[61] = 16'd32768+jumpz;
ram[62] = 16'd90;
ram[63] = 16'd61440+mvaaac;///////// all core
ram[64] = 16'd61440+loadb;
ram[65] = 16'd61440+mvacr1;
ram[66] = 16'd61440+mvabac;
ram[67] = 16'd61440+loadb;
ram[68] = 16'd61440+multi;
ram[69] = 16'd61440+mvr2r1;
ram[70] = 16'd61440+add;
ram[71] = 16'd61440+mvacr2;
ram[72] = 16'd61440+incan;
ram[73] = 16'd61440+mvanr1;
ram[74] = 16'd61440+mvacpac;
ram[75] = 16'd61440+sub ;
ram[76] = 16'd61440+jumpn;
ram[77] = 16'd63;
ram[78] = 16'd61440+store;
ram[79] = 16'd61440+incbn;
ram[80] = 16'd61440+rstan;
ram[81] = 16'd61440+rstr2;
ram[82] = 16'd61440+mvbnr1;
ram[83] = 16'd61440+mvbcpac;
```

```
                    ram[84] = 16'd61440+sub;
                    ram[85] = 16'd61440+jumpn;
                    ram[86] = 16'd63;
                    ram[87] = 16'd4096+endop;
                    ram[88] = 16'd8192+endop;
                    ram[89] = 16'd16384+endop;
                    ram[90] = 16'd32768+endop;

            end
            always @(posedge clock)
                    begin
                            instr_out_a <= ram[addr_a] ;
                            instr_out_b <= ram[addr_b] ;
                            instr_out_c <= ram[addr_c] ;
                            instr_out_d <= ram[addr_d] ;
                 end
endmodule
```

4. Data memory

```
module
datamemory(clock,datain_a,data_write_en_a,data_addr_a,datain_b,data_write_en_b,data_addr
_b,datain_c,data_write_en_c,data_addr_c,datain_d,data_write_en_d,data_addr_d,dataout_a,da
taout_b,dataout_c,dataout_d);

    input clock;
    input data_write_en_a;
    input [15:0] data_addr_a;
    input [15:0] datain_a;

  input data_write_en_b;
    input [15:0] data_addr_b;
    input [15:0] datain_b;

  input data_write_en_c;
    input [15:0] data_addr_c;
    input [15:0] datain_c;

  input data_write_en_d;
    input [15:0] data_addr_d;
    input [15:0] datain_d;

    output reg [15:0] dataout_a;
    output reg [15:0] dataout_b;
```

```
        output reg [15:0] dataout_c;
        output reg [15:0] dataout_d;

        reg [7:0] ram_in_1[0:15];
        reg [7:0] ram_in_2[0:15];
        reg [7:0] dim[0:15];
        reg [15:0] ram_A[0:100];
        reg [15:0] ram_B[0:100];
        reg [15:0] ram_C[0:100];
        reg [15:0] ram_D[0:100];

        integer i;
        integer k;
        integer f;

        initial
        begin

                $readmemb("F:/Semester 5/processor_final/processor/Matrix_A.txt", ram_in_1);
                $readmemb("F:/Semester 5/processor_final/processor/Matrix_B.txt", ram_in_2);
                $readmemb("F:/Semester   5/processor_final/processor/Dimensions_matrix.txt",
dim);


    ram_A[0] = dim[0];
                ram_B[0] = dim[0];
                ram_C[0] = dim[0];
                ram_D[0] = dim[0];
                ram_A[1] = dim[1];
                ram_B[1] = dim[1];
                ram_C[1] = dim[1];
                ram_D[1] = dim[1];
                ram_A[2] = dim[2];
                ram_B[2] = dim[2];
                ram_C[2] = dim[2];
                ram_D[2] = dim[2];

                for(k=0; k<dim[1]; k=k+1)
                        begin
                                ram_A[3+k]=ram_in_1[0*dim[1]+k];
                                ram_B[3+k]=ram_in_1[1*dim[1]+k];
                                ram_C[3+k]=ram_in_1[2*dim[1]+k];
                                ram_D[3+k]=ram_in_1[3*dim[1]+k];
```

```
                    end

            k = 0;
            for(i=0; i<dim[1]*dim[2]; i=i+1)
                        begin
                                ram_A[8+f]=ram_in_2[i];
                                ram_B[8+f]=ram_in_2[i];
                                ram_C[8+f]=ram_in_2[i];
                                ram_D[8+f]=ram_in_2[i];
                                f = f+1;
                                k = k+1;
                                if (k == dim[1])begin
                                   f = f+4-dim[1];
                                           k = 0;
                                           end
                    end
end


always @(posedge clock)
            begin
                    if (data_write_en_a == 1)begin
                            ram_A[data_addr_a] <= datain_a[15:0];
                    end

                    else begin
                            dataout_a <= ram_A[data_addr_a];
                    end
            end


always @(posedge clock)
            begin
                    if (data_write_en_b == 1)begin
                            ram_B[data_addr_b] <= datain_b[15:0];
                    end

                    else begin
                            dataout_b <= ram_B[data_addr_b];
                    end
            end

always @(posedge clock)
            begin
```

```
                if (data_write_en_c == 1)begin
                        ram_C[data_addr_c] <= datain_c[15:0];
                end

                else begin
                        dataout_c <= ram_C[data_addr_c];
                end
        end


    always @(posedge clock)
        begin
                if (data_write_en_d == 1)begin
                        ram_D[data_addr_d] <= datain_d[15:0];
                end
                else begin
                        dataout_d <= ram_D[data_addr_d];
                end
        end

endmodule
```

5. Processor_a

```
module processor_a(clock, status,dm_out, im_out,dm_en, pc_out, ar_out,end_process,r2_out);
        input clock;
        input [ 1: 0] status;
        input [ 15: 0] dm_out;
        input [ 15: 0] im_out;

        output reg dm_en;
        output [ 15: 0] pc_out;
        output [ 15: 0] ar_out;
        output  end_process;
        output [ 15: 0] r2_out;

        wire mem_write;
        wire [ 2: 0] alu_op;
        wire [ 15:0 ] bus_out;
        wire [ 15:0] alu_out;
        wire [ 15:0] r2_out;
        wire [ 15:0] r2_bus_out;
        wire [ 15:0] pc_out;
        wire [ 15: 0] ir_out;
        wire [ 15: 0] am_out;
        wire [ 15: 0] an_out;
        wire [ 15: 0] bn_out;
        wire [ 15: 0] arp_out;
        wire [ 15: 0] acp_out;
        wire [ 15: 0] bcp_out;
        wire [ 15: 0] aa_out;
        wire [ 15: 0] ab_out;
        wire [ 15: 0] ad_out;
        wire [ 15: 0] out_aa_out;
        wire [ 15: 0] out_ab_out;
        wire [ 15: 0] out_ad_out;
        wire [ 15: 0] alu_ac_out;
        wire [ 15: 0] r1_out;
        wire [ 15: 0] ac_out;
        wire [ 15: 0] write_en;
        wire [ 3: 0] read_en;
        wire [ 2: 0] rst;
        wire [ 15: 0] inc_en;
        wire [ 15: 0] z ;

        regr ar(. clock(clock),. write_en(write_en[0] ),. datain(bus_out),. dataout(ar_out)) ;

        regr2 r2(. clock(clock),. write_en(write_en[1] ),. rst(write_en[15]),. datain(bus_out),. dataout(r2_bus_out),. dm_mem_out(r2_out),. dm_mem_wr_en(mem_write)) ;

        regrinc pc(. clock(clock),. write_en(write_en[2] ),. datain(bus_out),. dataout(pc_out),. inc_en(inc_en[2])) ;

        regr ir(. clock(clock),. write_en(write_en[3] ),. datain(bus_out),. dataout(ir_out)) ;

        regrinc am(. clock(clock),. write_en(write_en[4] ),. datain(bus_out),. dataout(am_out),. inc_en(inc_en[4])) ;

        regran an(. clock(clock),. write_en(write_en[5] ),. rst(rst[0]),. datain(bus_out),. dataout(an_out),. inc_en(inc_en[5])) ;

        regrinc bn(. clock(clock),. write_en(write_en[6] ),. datain(bus_out),. dataout(bn_out),. inc_en(inc_en[6])) ;

        regr arp(. clock(clock),. write_en(write_en[7] ),. datain(bus_out),. dataout(arp_out)) ;
```

```
            regr acp(. clock(clock),. write_en(write_en[8] ),. datain(bus_out),.
dataout(acp_out)) ;

            regr bcp(. clock(clock),. write_en(write_en[9] ),. datain(bus_out),.
dataout(bcp_out)) ;

            regr r1(. clock(clock),. write_en(write_en[10] ),. datain(bus_out),. dataout(r1_out))
;

            bus bus1(. aa(out_aa_out) ,. r2(r2_bus_out),. ir(ir_out),. ab(out_ab_out),.
am(am_out),. an(an_out),. bn(bn_out),. arp(arp_out),. acp(acp_out),. bcp(bcp_out),. ac(ac_out),.
ad(out_ad_out),. dm(dm_out),. im(im_out),. read_en(read_en),. clock(clock),. busout(bus_out)) ;

            regir ac(. clock(clock),. write_en(write_en[11] ),.alu_to_ac(write_en[12]),.
datain(bus_out),. datain_alu(alu_out),. dataout(ac_out),. alu_data(alu_ac_out));

            alu alu1(. clock(clock),. in1(r1_out) ,. in2(ac_out),. alu_op(alu_op),.
alu_out(alu_out),. z(z) ) ;

            control_a control_a1(. clock(clock),
            . z(z),
            . status(status),
            . instruction(ir_out),
            . alu_op(alu_op),
            . rst(rst),
            . write_en(write_en),
            . read_en(read_en),
            . inc_en(inc_en),
            . mem_wr(mem_write),
            . end_process(end_process));

            address_select address_select_a (. clock(clock),. in1(am_out) ,. in2(an_out),.
in3(bn_out),. out_address_aa(out_aa_out),. out_address_ab(out_ab_out),.
out_address_ad(out_ad_out) );

            always @ (posedge clock)
                        if (status == 2'b01) begin
                                    dm_en <= write_en[13];
                        end
endmodule
```

6.Processor_b

```
module processor_b(clock, status,dm_out, im_out,dm_en, pc_out, ar_out,end_process,r2_out);
                input clock;
                input [ 1: 0] status;
                input [ 15: 0] dm_out;
                input [ 15: 0] im_out;

                output reg dm_en;
                output [ 15: 0] pc_out;
                output [ 15: 0] ar_out;
                output  end_process;
                output [ 15: 0] r2_out;

                wire mem_write;
                wire [ 2: 0] alu_op;
                wire [ 15:0 ] bus_out;
                wire [ 15:0] alu_out;

                wire [ 15:0] r2_out;
                wire [ 15:0] r2_bus_out;
                wire [ 15:0] pc_out;
                wire [ 15: 0] ir_out;

                wire [ 15: 0] am_out;
                wire [ 15: 0] an_out;
                wire [ 15: 0] bn_out;
                wire [ 15: 0] arp_out;
                wire [ 15: 0] acp_out;
                wire [ 15: 0] bcp_out;
                wire [ 15: 0] aa_out;
                wire [ 15: 0] ab_out;
                wire [ 15: 0] ad_out;
                wire [ 15: 0] out_aa_out;
                wire [ 15: 0] out_ab_out;
                wire [ 15: 0] out_ad_out;
                wire [ 15: 0] alu_ac_out;
                wire [ 15: 0] r1_out;
                wire [ 15: 0] ac_out;
                wire [ 15: 0] write_en;
                wire [ 3: 0] read_en;
                wire [ 2: 0] rst;
                wire [ 15: 0] inc_en;
                wire [ 15: 0] z ;
```

```
          regr ar(. clock(clock),. write_en(write_en[0] ),. datain(bus_out),. dataout(ar_out)) ;

          regr2 r2(. clock(clock),. write_en(write_en[1] ),. rst(write_en[15]),.
datain(bus_out),. dataout(r2_bus_out),. dm_mem_out(r2_out),. dm_mem_wr_en(mem_write)) ;

          regrinc pc(. clock(clock),. write_en(write_en[2] ),. datain(bus_out),.
dataout(pc_out),. inc_en(inc_en[2])) ;

          regr ir(. clock(clock),. write_en(write_en[3] ),. datain(bus_out),. dataout(ir_out)) ;

          regrinc am(. clock(clock),. write_en(write_en[4] ),. datain(bus_out),.
dataout(am_out),. inc_en(inc_en[4])) ;

          regran an(. clock(clock),. write_en(write_en[5] ),. rst(rst[0]),. datain(bus_out),.
dataout(an_out),. inc_en(inc_en[5])) ;

          regrinc bn(. clock(clock),. write_en(write_en[6] ),. datain(bus_out),.
dataout(bn_out),. inc_en(inc_en[6])) ;

          regr arp(. clock(clock),. write_en(write_en[7] ),. datain(bus_out),.
dataout(arp_out)) ;

          regr acp(. clock(clock),. write_en(write_en[8] ),. datain(bus_out),.
dataout(acp_out)) ;

          regr bcp(. clock(clock),. write_en(write_en[9] ),. datain(bus_out),.
dataout(bcp_out)) ;

          regr r1(. clock(clock),. write_en(write_en[10] ),. datain(bus_out),. dataout(r1_out))
;

          bus bus1(. aa(out_aa_out) ,. r2(r2_bus_out),. ir(ir_out),. ab(out_ab_out),.
am(am_out),. an(an_out),. bn(bn_out),. arp(arp_out),. acp(acp_out),. bcp(bcp_out),. ac(ac_out),.
ad(out_ad_out),. dm(dm_out),. im(im_out),. read_en(read_en),. clock(clock),. busout(bus_out)) ;

          regir ac(. clock(clock),. write_en(write_en[11] ),.alu_to_ac(write_en[12]),.
datain(bus_out),. datain_alu(alu_out),. dataout(ac_out),. alu_data(alu_ac_out));

          alu alu1(. clock(clock),. in1(r1_out) ,. in2(ac_out),. alu_op(alu_op),.
alu_out(alu_out),. z(z) ) ;

          control_b control_b1(. clock(clock),
          . z(z),
          . status(status),
          . instruction(ir_out),
          . alu_op(alu_op),
          . rst(rst),
          . write_en(write_en),
          . read_en(read_en),
          . inc_en(inc_en),
          . mem_wr(mem_write),
          . end_process(end_process));

          address_select  address_select_a  (.  clock(clock),.  in1(am_out)  ,.  in2(an_out),.
in3(bn_out),.              out_address_aa(out_aa_out),.              out_address_ab(out_ab_out),.
out_address_ad(out_ad_out) );

          always @ (posedge clock)
                              if (status == 2'b01) begin
                                          dm_en <= write_en[13];
                              end
endmodule
```

7. Processor_c

```
module processor_c(clock, status,dm_out, im_out,dm_en, pc_out, ar_out,end_process,r2_out);
                    input clock;
                    input [ 1: 0] status;
                    input [ 15: 0] dm_out;
                    input [ 15: 0] im_out;

                    output reg dm_en;
                    output [ 15: 0] pc_out;
                    output [ 15: 0] ar_out;
                    output  end_process;
                    output [ 15: 0] r2_out;

                    wire mem_write;
                    wire [ 2: 0] alu_op;
                    wire [ 15:0 ] bus_out;
                    wire [ 15:0] alu_out;
                    wire [ 15:0] r2_out;
                    wire [ 15:0] r2_bus_out;
                    wire [ 15:0] pc_out;
                    wire [ 15: 0] ir_out;
                    wire [ 15: 0] am_out;
                    wire [ 15: 0] an_out;
                    wire [ 15: 0] bn_out;
                    wire [ 15: 0] arp_out;
                    wire [ 15: 0] acp_out;
                    wire [ 15: 0] bcp_out;
                    wire [ 15: 0] aa_out;
                    wire [ 15: 0] ab_out;
                    wire [ 15: 0] ad_out;
                    wire [ 15: 0] out_aa_out;
                    wire [ 15: 0] out_ab_out;
                    wire [ 15: 0] out_ad_out;
                    wire [ 15: 0] alu_ac_out;
                    wire [ 15: 0] r1_out;
                    wire [ 15: 0] ac_out;
                    wire [ 15: 0] write_en;
                    wire [ 3: 0] read_en;
                    wire [ 2: 0] rst;
                    wire [ 15: 0] inc_en;
                    wire [ 15: 0] z ;

                    regr ar(. clock(clock),. write_en(write_en[0] ),. datain(bus_out),. dataout(ar_out)) ;

                    regr2 r2(. clock(clock),. write_en(write_en[1] ),. rst(write_en[15]),.
datain(bus_out),. dataout(r2_bus_out),. dm_mem_out(r2_out),. dm_mem_wr_en(mem_write)) ;

                    regrinc pc(. clock(clock),. write_en(write_en[2] ),. datain(bus_out),.
dataout(pc_out),. inc_en(inc_en[2])) ;

                    regr ir(. clock(clock),. write_en(write_en[3] ),. datain(bus_out),. dataout(ir_out)) ;

                    regrinc am(. clock(clock),. write_en(write_en[4] ),. datain(bus_out),.
dataout(am_out),. inc_en(inc_en[4])) ;

                    regran an(. clock(clock),. write_en(write_en[5] ),. rst(rst[0]),. datain(bus_out),.
dataout(an_out),. inc_en(inc_en[5])) ;

                    regrinc bn(. clock(clock),. write_en(write_en[6] ),. datain(bus_out),.
dataout(bn_out),. inc_en(inc_en[6])) ;

                    regr arp(. clock(clock),. write_en(write_en[7] ),. datain(bus_out),.
dataout(arp_out)) ;

                    regr acp(. clock(clock),. write_en(write_en[8] ),. datain(bus_out),.
dataout(acp_out)) ;
```

```
            regr bcp(. clock(clock),. write_en(write_en[9] ),. datain(bus_out),. dataout(bcp_out)) ;

            regr r1(. clock(clock),. write_en(write_en[10] ),. datain(bus_out),. dataout(r1_out)) ;

            bus bus1(. aa(out_aa_out) ,. r2(r2_bus_out),. ir(ir_out),. ab(out_ab_out),. am(am_out),. an(an_out),. bn(bn_out),. arp(arp_out),. acp(acp_out),. bcp(bcp_out),. ac(ac_out),. ad(out_ad_out),. dm(dm_out),. im(im_out),. read_en(read_en),. clock(clock),. busout(bus_out)) ;

            regir ac(. clock(clock),. write_en(write_en[11] ),.alu_to_ac(write_en[12]),. datain(bus_out),. datain_alu(alu_out),. dataout(ac_out),. alu_data(alu_ac_out));

            alu alu1(. clock(clock),. in1(r1_out) ,. in2(ac_out),. alu_op(alu_op),. alu_out(alu_out),. z(z) ) ;

            control_c control_c1(. clock(clock),
            . z(z),
            . status(status),
            . instruction(ir_out),
            . alu_op(alu_op),
            . rst(rst),
            . write_en(write_en),
            . read_en(read_en),
            . inc_en(inc_en),
            . mem_wr(mem_write),
            . end_process(end_process));

            address_select address_select_a (. clock(clock),. in1(am_out) ,. in2(an_out),. in3(bn_out),. out_address_aa(out_aa_out),. out_address_ab(out_ab_out),. out_address_ad(out_ad_out) );

            always @ (posedge clock)
                        if (status == 2'b01) begin
                                    dm_en <= write_en[13] ;
                        end
endmodule
```

8. Processor_d

```
module processor_d(clock, status,dm_out, im_out,dm_en, pc_out, ar_out,end_process,r2_out);
                input clock;
                input [ 1: 0] status;
                input [ 15: 0] dm_out;
                input [ 15: 0] im_out;

                output reg dm_en;
                output [ 15: 0] pc_out;
                output [ 15: 0] ar_out;
                output  end_process;
                output [ 15: 0] r2_out;

                wire mem_write;
                wire [ 2: 0] alu_op;
                wire [ 15:0 ] bus_out;
                wire [ 15:0] alu_out;
                wire [ 15:0] r2_out;
                wire [ 15:0] r2_bus_out;
                wire [ 15:0] pc_out;
                wire [ 15: 0] ir_out;
                wire [ 15: 0] am_out;
                wire [ 15: 0] an_out;
                wire [ 15: 0] bn_out;
                wire [ 15: 0] arp_out;
                wire [ 15: 0] acp_out;
                wire [ 15: 0] bcp_out;
                wire [ 15: 0] aa_out;
                wire [ 15: 0] ab_out;
                wire [ 15: 0] ad_out;
                wire [ 15: 0] out_aa_out;
                wire [ 15: 0] out_ab_out;
                wire [ 15: 0] out_ad_out;
                wire [ 15: 0] alu_ac_out;
                wire [ 15: 0] r1_out;
                wire [ 15: 0] ac_out;
                wire [ 15: 0] write_en;
                wire [ 3: 0] read_en;
                wire [ 2: 0] rst;
                wire [ 15: 0] inc_en;
                wire [ 15: 0] z ;
```

```
        regr ar(. clock(clock),. write_en(write_en[0] ),. datain(bus_out),. dataout(ar_out)) ;

        regr2 r2(. clock(clock),. write_en(write_en[1] ),. rst(write_en[15]),.
datain(bus_out),. dataout(r2_bus_out),. dm_mem_out(r2_out),. dm_mem_wr_en(mem_write)) ;

        regrinc pc(. clock(clock),. write_en(write_en[2] ),. datain(bus_out),.
dataout(pc_out),. inc_en(inc_en[2])) ;

        regr ir(. clock(clock),. write_en(write_en[3] ),. datain(bus_out),. dataout(ir_out)) ;

        regrinc am(. clock(clock),. write_en(write_en[4] ),. datain(bus_out),.
dataout(am_out),. inc_en(inc_en[4])) ;

        regran an(. clock(clock),. write_en(write_en[5] ),. rst(rst[0]),. datain(bus_out),.
dataout(an_out),. inc_en(inc_en[5])) ;

        regrinc bn(. clock(clock),. write_en(write_en[6] ),. datain(bus_out),.
dataout(bn_out),. inc_en(inc_en[6])) ;

        regr arp(. clock(clock),. write_en(write_en[7] ),. datain(bus_out),.
dataout(arp_out)) ;

        regr acp(. clock(clock),. write_en(write_en[8] ),. datain(bus_out),.
dataout(acp_out)) ;

        regr bcp(. clock(clock),. write_en(write_en[9] ),. datain(bus_out),.
dataout(bcp_out)) ;

        regr r1(. clock(clock),. write_en(write_en[10] ),. datain(bus_out),. dataout(r1_out))
;

        bus bus1(. aa(out_aa_out) ,. r2(r2_bus_out),. ir(ir_out),. ab(out_ab_out),.
am(am_out),. an(an_out),. bn(bn_out),. arp(arp_out),. acp(acp_out),. bcp(bcp_out),. ac(ac_out),.
ad(out_ad_out),. dm(dm_out),. im(im_out),. read_en(read_en),. clock(clock),. busout(bus_out)) ;

        regir ac(. clock(clock),. write_en(write_en[11] ),.alu_to_ac(write_en[12]),.
datain(bus_out),. datain_alu(alu_out),. dataout(ac_out),. alu_data(alu_ac_out));

        alu alu1(. clock(clock),. in1(r1_out) ,. in2(ac_out),. alu_op(alu_op),.
alu_out(alu_out),. z(z) ) ;

        control_d control_d1(. clock(clock),
        . z(z),
        . status(status),
        . instruction(ir_out),
        . alu_op(alu_op),
        . rst(rst),
        . write_en(write_en),
        . read_en(read_en),
        . inc_en(inc_en),
        . mem_wr(mem_write),
        . end_process(end_process));

        address_select address_select_a (. clock(clock),. in1(am_out) ,. in2(an_out),.
in3(bn_out),.     out_address_aa(out_aa_out),.     out_address_ab(out_ab_out),.
out_address_ad(out_ad_out) );

        always @ (posedge clock)
                        if (status == 2'b01) begin
                                  dm_en <= write_en[13] ;
                        end
endmodule
```

9. control_a

```
module control_a(
          input clock,
          input [15:0] z,
          input [ 1: 0] status,
          input [ 15: 0] instruction,
          output reg [ 2: 0] alu_op,
          output reg [ 15: 0] write_en,
          output reg [ 3: 0] read_en,
          output reg [ 2: 0] rst,
          output reg [ 15: 0] inc_en,
          output reg mem_wr,
          output reg end_process);

          reg [ 5: 0] present = 6'd0;
          reg [ 5: 0] next = 6'd0;
          parameter fetch1 = 6'd1,
          fetch2 = 6'd2,
          loada1 = 6'd3,
          loada2 = 6'd4,
          loadb1 = 6'd5,
          loadb2 = 6'd6,
          store1 = 6'd7,
          store2 = 6'd8,
          jumpz1 = 6'd9,
          jumpz2 = 6'd10,
          jumpz3 = 6'd11,

          fetch1x = 6'd50,
          fetch2x = 6'd51,
          loada1x = 6'd52,
          loada2x = 6'd53,
          loadb1x = 6'd54,
          loadb2x = 6'd55,
          loadb3x = 6'd56,
          jumpz3x = 6'd57,
          sub1x = 6'd58,
          sub2x = 6'd59,
          jumpz2x = 6'd60,
          multi1x = 6'd61,
          add1x = 6'd62,

          jumpn1 = 6'd13,
          jumpn2 = 6'd14,
          jumpn3 = 6'd15,
          jumpn2x = 6'd16,
          jumpn3x = 6'd17,
          store1x = 6'd18,
          store2x = 6'd19,
          mvr2r1 = 6'd20,
          mvr2r1x = 6'd21,
          mvacarp = 6'd23,
          mvacacp = 6'd24,
          mvacbcp = 6'd25,
          incam = 6'd26,
          incan = 6'd27,
          incbn = 6'd28,
          mvamr1 = 6'd29,
          mvanr1 = 6'd30,
          mvbnr1 = 6'd31,
          mvarpac = 6'd32,
          mvacpac = 6'd33,
          mvbcpac = 6'd34,
          mvaaac = 6'd35,
          mvabac = 6'd36,
          mvadac = 6'd37,
          mvacr1 = 6'd38,
          mvacr2 = 6'd39,
          mvr2ac = 6'd40,
          multi = 6'd41,
          add = 6'd42,
          sub = 6'd43,
          rstan = 6'd45,
          rstr2 = 6'd46,
          nop = 6'd47,
          endop = 6'd48,
          idle = 6'd0;

          always @(posedge clock)
          present <= next;
          always @(posedge clock)
                          begin
                          if (present == endop)
                          end_process <= 1'd1;
                          else
                          end_process <= 1'd0;
                          end
          always @(present or instruction or status or z)
                  if (instruction[15:12] == 4'd1 || instruction[15:12] == 4'd15)
                          begin
                                  case(present)

                                          idle: begin
                                          mem_wr <= 1'd0;
                                          read_en <= 4'd0;
                                          rst <=3'b000;
                                          write_en <= 16'b0000000000000000;
                                          inc_en <= 16'b0000000000000000;
                                          alu_op <= 3'd0;
                                          if (status == 2'b01)
                                          next <= fetch1;
                                          else
                                          next <= idle;
                                          end

                                          fetch1: begin
                                          mem_wr <= 1'd0;
                                          read_en <= 4'd15;
                                          rst <=3'b000;
                                          write_en <= 16'b0000000000001000;
```

```
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
next <= fetch1x;
end

fetch1x: begin
mem_wr <= 1'd0;
read_en <= 4'd0;
rst <=3'b000;
write_en <= 16'b0000000000000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
next <= fetch2;
end

fetch2: begin
mem_wr = 1'd0;
read_en <= 4'd0;
rst <=3'b000;
write_en <= 16'b0000000000000000;
inc_en <= 16'b0000000000000100;
alu_op <= 3'd0;
next <= fetch2x;
end

fetch2x: begin
mem_wr <= 1'd0;
read_en <= 4'd0;
rst <=3'b000;
write_en <= 16'b0000000000000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
next <= instruction[5:0];
end

loada1: begin
mem_wr <= 1'd0;
read_en <= 4'd15;
rst <=3'b000;
write_en <= 16'b00000000000000001;
inc_en <= 16'b0000000000000100;
alu_op <= 3'd0;
next <= loada2;
end
```

```
loada2: begin // ac to bus bus to
mem_wr = 1'd0;
read_en <= 4'd14;
rst <=3'b000;
write_en <= 16'b0000100000000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
next <= fetch1;
end

mvr2r1: begin
mem_wr = 1'd0;
read_en <= 4'd2;
rst <=3'b000;
write_en <= 16'b0000010000000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
next <= fetch1;
end

loadb1: begin
mem_wr = 1'd0;
read_en <= 4'd12;
rst <=3'b000;
write_en <= 16'b0000000000000001;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
next <= loadb1x;
end

loadb1x: begin
mem_wr <= 1'd0;
read_en <= 4'd0;
rst <=3'b000;
write_en <= 16'b0000000000000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
next <= loadb2;
end

loadb2: begin
mem_wr = 1'd0;
read_en <= 4'd14;
```

```
rst <=3'b000;
write_en <= 16'b0000100000000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
next <= fetch1;
end

store1: begin
mem_wr = 1'd0;
read_en <= 4'd13;
rst <=3'b000;
write_en <= 16'b0000000000000001;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
next <= store2;
end

store2: begin
mem_wr = 1'd1;
read_en <= 4'd2;
rst <=3'b000;
write_en <= 16'b0010000000000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
next <= fetch1;
end

jumpz1: begin
mem_wr <= 1'd0;
read_en <= 4'd15;
rst <=3'b000;
write_en <= 16'b0000000010000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
if (z == 0)
next <= jumpz2;
else
next <= jumpz3;
end

jumpz2: begin
mem_wr = 1'd0;
read_en <= 4'd8;
rst <=3'b000;
```

```
write_en <= 16'b0000000000000100;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
next <= jumpz2x;
end

jumpz2x: begin
mem_wr = 1'd0;
read_en <= 4'd0;
rst <=3'b000;
write_en <= 16'b0000000000000000;
inc_en <= 16'b0000000000000000;//pc =pc+1
alu_op <= 3'd0;
next <= fetch1;
end

jumpz3: begin
mem_wr = 1'd0;
read_en <= 4'd0;
rst <=3'b000;
write_en <= 16'b0000000000000000;
inc_en <= 16'b0000000000000100;//pc =pc+1
alu_op <= 3'd0;
next <= jumpz3x;
end

jumpz3x: begin
mem_wr = 1'd0;
read_en <= 4'd0;
rst <=3'b000;
write_en <= 16'b0000000000000000;
inc_en <= 16'b0000000000000000;//pc =pc+1
alu_op <= 3'd0;
next <= fetch1;
end

jumpn1: begin
mem_wr <= 1'd0;
read_en <= 4'd15;
rst <=3'b000;
write_en <= 16'b0000000010000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
if (z == 1)
```

```
next <= jumpn2;
else
next <= jumpn3;
end

jumpn2: begin
mem_wr = 1'd0;
read_en <= 4'd8;
rst <=3'b000;
write_en <= 16'b0000000000000100;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
next <= jumpn2x;
end

jumpn2x: begin
mem_wr = 1'd0;
read_en <= 4'd0;
rst <=3'b000;
write_en <= 16'b0000000000000000;
inc_en <= 16'b0000000000000000;//pc =pc+1
alu_op <= 3'd0;
next <= fetch1;
end

jumpn3: begin
mem_wr = 1'd0;
read_en <= 4'd0;
rst <=3'b000;
write_en <= 16'b0000000000000000;
inc_en <= 16'b0000000000000100;//pc =pc+1
alu_op <= 3'd0;
next <= jumpn3x;
end

jumpn3x: begin
mem_wr = 1'd0;
read_en <= 4'd0;
rst <=3'b000;
write_en <= 16'b0000000000000000;
inc_en <= 16'b0000000000000000;//pc =pc+1
alu_op <= 3'd0;
next <= fetch1;
end
```

```
mvacarp: begin
mem_wr = 1'd0;
read_en <= 4'd12;
rst <=3'b000;
write_en <= 16'b0000000010000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
next <= fetch1;
end

mvacacp: begin
mem_wr = 1'd0;
read_en <= 4'd12;
rst <=3'b000;
write_en <= 16'b0000000100000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
next <= fetch1;
end

mvacbcp: begin
mem_wr = 1'd0;
read_en <= 4'd12;
rst <=3'b000;
write_en <= 16'b0000001000000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
next <= fetch1;
end

incam: begin
mem_wr = 1'd0;
read_en <= 4'd0;
rst <=3'b000;
write_en <= 16'b0000000000000000;
inc_en <= 16'b0000000000010000;
alu_op <= 3'd0;
next <= fetch1;
end

incan: begin
mem_wr = 1'd0;
read_en <= 4'd0;
```

```
rst <=3'b000;
write_en <= 16'b0000000000000000;
inc_en <= 16'b0000000000100000;
alu_op <= 3'd0;
next <= fetch1;
end

incbn: begin
mem_wr = 1'd0;
read_en <= 4'd0;
rst <=3'b000;
write_en <= 16'b0000000000000000;
inc_en <= 16'b0000000001000000;
alu_op <= 3'd0;
next <= fetch1;
end

mvamr1: begin
mem_wr = 1'd0;
read_en <= 4'd5;
rst <=3'b000;
write_en <= 16'b0000010000000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
next <= fetch1;
end

mvanr1: begin
mem_wr = 1'd0;
read_en <= 4'd6;
rst <=3'b000;
write_en <= 16'b0000010000000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
next <= fetch1;
end

mvbnr1: begin
mem_wr = 1'd0;
read_en <= 4'd7;
rst <=3'b000;
write_en <= 16'b0000010000000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
```

```
next <= fetch1;
end

mvarpac: begin
mem_wr = 1'd0;
read_en <= 4'd8;
rst <=3'b000;
write_en <= 16'b0000100000000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
next <= fetch1;
end

mvacpac: begin
mem_wr = 1'd0;
read_en <= 4'd9;
rst <=3'b000;
write_en <= 16'b0000100000000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
next <= fetch1;
end

mvbcpac: begin
mem_wr = 1'd0;
read_en <= 4'd10;
rst <=3'b000;
write_en <= 16'b0000100000000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
next <= fetch1;
end

mvaaac: begin
mem_wr = 1'd0;
read_en <= 4'd1;
rst <=3'b000;
write_en <= 16'b0000100000000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
next <= fetch1;
end

mvabac: begin
```

```
mem_wr = 1'd0;
read_en <= 4'd3;
rst <=3'b000;
write_en <= 16'b0000100000000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
next <= fetch1;
end

mvadac: begin
mem_wr = 1'd0;
read_en <= 4'd13;
rst <=3'b000;
write_en <= 16'b0000100000000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
next <= fetch1;
end

mvacr1: begin
mem_wr = 1'd0;
read_en <= 4'd12;
rst <=3'b000;
write_en <= 16'b0000010000000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
next <= fetch1;
end

mvacr2: begin
mem_wr = 1'd0;
read_en <= 4'd12;
rst <=3'b000;
write_en <= 16'b0000000000000010;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
next <= fetch1;
end

multi: begin
mem_wr = 1'd0;
read_en <= 4'd0;
rst <=3'b000;
write_en <= 16'b0000000000000000;
```

```
inc_en <= 16'b0000000000000000;
alu_op <= 3'd1;
next <= multi1x;
end

multi1x: begin
mem_wr = 1'd0;
read_en <= 4'd0;
rst <=3'b000;
write_en <= 16'b0001000000000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd1;
next <= fetch1;
end

add: begin
mem_wr = 1'd0;
read_en <= 4'd0;
rst <=3'b000;
write_en <= 16'b0000000000000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd2;
next <= add1x;
end

add1x: begin
mem_wr = 1'd0;
read_en <= 4'd0;
rst <=3'b000;
write_en <= 16'b0001000000000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd2;
next <= fetch1;
end

sub: begin
mem_wr = 1'd0;
read_en <= 4'd0;
rst <=3'b000;
write_en <= 16'b0000000000000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd3;
next <= sub1x;
end
```

```
sub1x: begin
mem_wr = 1'd0;
read_en <= 4'd0;
rst <=3'b000;
write_en <= 16'b0001000000000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd3;
next <= fetch1;
end

rstan: begin
mem_wr = 1'd0;
read_en <= 4'd0;
rst <=3'b001;
write_en <= 16'b0000000000000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
next <= fetch1;
end

rstr2: begin
mem_wr = 1'd0;
read_en <= 4'd0;
rst <=3'b000;
write_en <= 16'b1000000000000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
next <= fetch1;
end

nop: begin
mem_wr = 1'd0;
read_en <= 4'd0;
rst <=3'b000;
write_en <= 16'b0000000000000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
next <= fetch1;
end

endop: begin
mem_wr = 1'd0;
read_en <= 4'd14;
```

```
                                        rst <=3'b000;
                                        write_en <= 16'b0000000000000000;
                                        inc_en <= 16'b0000000000000000;
                                        alu_op <= 3'd0;
                                        next <= endop;
                                        end

                                        default: begin
                                        mem_wr = 1'd0;
                                        read_en <= 4'd0;
                                        rst <=3'b000;
                                        write_en <= 16'b0000000000000000;
                                        inc_en <= 16'b0000000000000000;
                                        alu_op <= 3'd0;
                                        next <= fetch1;
                                        end
                endcase
        end
          else
                begin
                        case(present)

                                        idle: begin
                                        mem_wr <= 1'd0;
                                        read_en <= 4'd0;
                                        rst <=3'b000;
                                        write_en <= 16'b0000000000000000;
                                        inc_en <= 16'b0000000000000000;
                                        alu_op <= 3'd0;
                                        if (status == 2'b01)
                                        next <= fetch1;
                                        else
                                        next <= idle;
                                        end

                                        fetch1: begin
                                        mem_wr <= 1'd0;
                                        read_en <= 4'd15;
                                        rst <=3'b000;
                                        write_en <= 16'b0000000000001000;
                                        inc_en <= 16'b0000000000000000;
                                        alu_op <= 3'd0;
                                        next <= fetch1x;
                                        end
                                        fetch1x: begin
                                        mem_wr <= 1'd0;
                                        read_en <= 4'd0;
                                        rst <=3'b000;
                                        write_en <= 16'b0000000000000000;
                                        inc_en <= 16'b0000000000000000;
                                        alu_op <= 3'd0;
                                        next <= fetch2;
                                        end

                                        fetch2: begin
                                        mem_wr = 1'd0;
                                        read_en <= 4'd0;
                                        rst <=3'b000;
                                        write_en <= 16'b0000000000000000;
                                        inc_en <= 16'b0000000000000100;
                                        alu_op <= 3'd0;
                                        next <= nop;
                                        end

                                        nop: begin
                                        mem_wr = 1'd1;
                                        read_en <= 4'd0;
                                        rst <=3'b000;
                                        write_en <= 16'b0000000000000000;
                                        inc_en <= 16'b0000000000000000;
                                        alu_op <= 3'd0;
                                        next <= fetch1;
                                        end

                                        default: begin
                                        mem_wr = 1'd0;
                                        read_en <= 4'd0;
                                        rst <=3'b000;
                                        write_en <= 16'b0000000000000000;
                                        inc_en <= 16'b0000000000000000;
                                        alu_op <= 3'd0;
                                        next <= fetch1;
                                        end
                endcase
end
```

```
endmodule
```

10. control_b

```
module control_b(
	input clock,
	input [15:0] z,
	input [ 1: 0] status,
	input [ 15: 0] instruction,
	output reg [ 2: 0] alu_op,
	output reg [ 15: 0] write_en,
	output reg [ 3: 0] read_en,
	output reg [ 2: 0] rst,
	output reg [ 15: 0] inc_en,
	output reg mem_wr,
	output reg end_process);

	reg [ 5: 0] present = 6'd0;
	reg [ 5: 0] next = 6'd0;
	parameter fetch1 = 6'd1,
	fetch2 = 6'd2,
	loada1 = 6'd3,
	loada2 = 6'd4,
	loadb1 = 6'd5,
	loadb2 = 6'd6,
	store1 = 6'd7,
	store2 = 6'd8,
	jumpz1 = 6'd9,
	jumpz2 = 6'd10,
	jumpz3 = 6'd11,
	fetch1x = 6'd50,
	fetch2x = 6'd51,
	loada1x = 6'd52,
	loada2x = 6'd53,
	loadb1x = 6'd54,
	loadb2x = 6'd55,
	loadb3x = 6'd56,
	jumpz3x = 6'd57,
	sub1x = 6'd58,
	sub2x = 6'd59,
	jumpz2x = 6'd60,
	multi1x = 6'd61,
	add1x = 6'd62,
jumpn1 = 6'd13,
jumpn2 = 6'd14,
jumpn3 = 6'd15,
jumpn2x = 6'd16,
jumpn3x = 6'd17,
store1x = 6'd18,
store2x = 6'd19,
mvr2r1 = 6'd20,
mvr2r1x = 6'd21,

mvacarp = 6'd23,
mvacacp = 6'd24,
mvacbcp = 6'd25,
incam = 6'd26,
incan = 6'd27,
incbn = 6'd28,
mvamr1 = 6'd29,
mvanr1 = 6'd30,
mvbnr1 = 6'd31,
mvarpac = 6'd32,
mvacpac = 6'd33,
mvbcpac = 6'd34,
mvaaac = 6'd35,
mvabac = 6'd36,
mvadac = 6'd37,
mvacr1 = 6'd38,
mvacr2 = 6'd39,
mvr2ac = 6'd40,
multi = 6'd41,
add = 6'd42,
sub = 6'd43,
rstan = 6'd45,
rstr2 = 6'd46,
nop = 6'd47,
endop = 6'd48,
idle = 6'd0;

always @(posedge clock)
present <= next;
always @(posedge clock)
            begin
            if (present == endop)
```

```
				end_process <= 1'd1;
				else
				end_process <= 1'd0;
				end
always @(present or instruction or status or z)
		if (instruction[15:12] == 4'd2 || instruction[15:12] == 4'd15)
				begin
						case(present)

								idle: begin
								mem_wr <= 1'd0;
								read_en <= 4'd0;
								rst <=3'b000;
								write_en <= 16'b0000000000000000;
								inc_en <= 16'b0000000000000000;
								alu_op <= 3'd0;
								if (status == 2'b01)
								next <= fetch1;
								else
								next <= idle;
								end

								fetch1: begin
								mem_wr <= 1'd0;
								read_en <= 4'd15;
								rst <=3'b000;
								write_en <= 16'b0000000000001000;
								inc_en <= 16'b0000000000000000;
								alu_op <= 3'd0;
								next <= fetch1x;
								end

								fetch1x: begin
								mem_wr <= 1'd0;
								read_en <= 4'd0;
								rst <=3'b000;
								write_en <= 16'b0000000000000000;
								inc_en <= 16'b0000000000000000;
								alu_op <= 3'd0;
								next <= fetch2;
								end

								fetch2: begin
								mem_wr = 1'd0;
```

```
read_en <= 4'd0;
rst <=3'b000;
write_en <= 16'b0000000000000000;
inc_en <= 16'b0000000000000100;
alu_op <= 3'd0;
next <= fetch2x;
end

fetch2x: begin
mem_wr <= 1'd0;
read_en <= 4'd0;
rst <=3'b000;
write_en <= 16'b0000000000000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
next <= instruction[5:0];
end

loada1: begin
mem_wr <= 1'd0;
read_en <= 4'd15;
rst <=3'b000;
write_en <= 16'b00000000000000001;
inc_en <= 16'b0000000000000100;
alu_op <= 3'd0;
next <= loada2;
end


loada2: begin
mem_wr = 1'd0;
read_en <= 4'd14;
rst <=3'b000;
write_en <= 16'b0000100000000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
next <= fetch1;
end

mvr2r1: begin
mem_wr = 1'd0;
read_en <= 4'd2;
rst <=3'b000;
write_en <= 16'b0000010000000000;
```

```
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
next <= fetch1;
end


loadb1: begin
mem_wr = 1'd0;
read_en <= 4'd12;
rst <=3'b000;
write_en <= 16'b0000000000000001;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
next <= loadb1x;
end

loadb1x: begin
mem_wr <= 1'd0;
read_en <= 4'd0;
rst <=3'b000;
write_en <= 16'b0000000000000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
next <= loadb2;
end

loadb2: begin
mem_wr = 1'd0;
read_en <= 4'd14;
rst <=3'b000;
write_en <= 16'b0000100000000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
next <= fetch1;
end

store1: begin
mem_wr = 1'd0;
read_en <= 4'd13;
rst <=3'b000;
write_en <= 16'b0000000000000001;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
next <= store2;
```

```
end

store2: begin
mem_wr = 1'd1;
read_en <= 4'd2;
rst <=3'b000;
write_en <= 16'b0010000000000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
next <= fetch1;
end

jumpz1: begin
mem_wr <= 1'd0;
read_en <= 4'd15;
rst <=3'b000;
write_en <= 16'b0000000010000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
if (z == 0)
next <= jumpz2;
else
next <= jumpz3;
end

jumpz2: begin
mem_wr = 1'd0;
read_en <= 4'd8;
rst <=3'b000;
write_en <= 16'b0000000000000100;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
next <= jumpz2x;
end

jumpz2x: begin
mem_wr = 1'd0;
read_en <= 4'd0;
rst <=3'b000;
write_en <= 16'b0000000000000000;
inc_en <= 16'b0000000000000000;//pc =pc+1
alu_op <= 3'd0;
next <= fetch1;
end
```

```
jumpz3: begin
mem_wr = 1'd0;
read_en <= 4'd0;
rst <=3'b000;
write_en <= 16'b0000000000000000;
inc_en <= 16'b0000000000000100;//pc =pc+1
alu_op <= 3'd0;
next <= jumpz3x;
end

jumpz3x: begin
mem_wr = 1'd0;
read_en <= 4'd0;
rst <=3'b000;
write_en <= 16'b0000000000000000;
inc_en <= 16'b0000000000000000;//pc =pc+1
alu_op <= 3'd0;
next <= fetch1;
end

jumpn1: begin
mem_wr <= 1'd0;
read_en <= 4'd15;
rst <=3'b000;
write_en <= 16'b0000000010000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
if (z == 1)
next <= jumpn2;
else
next <= jumpn3;
end

jumpn2: begin
mem_wr = 1'd0;
read_en <= 4'd8;
rst <=3'b000;
write_en <= 16'b0000000000000100;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
next <= jumpn2x;
end
```

```
jumpn2x: begin
mem_wr = 1'd0;
read_en <= 4'd0;
rst <=3'b000;
write_en <= 16'b0000000000000000;
inc_en <= 16'b0000000000000000;//pc =pc+1
alu_op <= 3'd0;
next <= fetch1;
end

jumpn3: begin
mem_wr = 1'd0;
read_en <= 4'd0;
rst <=3'b000;
write_en <= 16'b0000000000000000;
inc_en <= 16'b0000000000000100;//pc =pc+1
alu_op <= 3'd0;
next <= jumpn3x;
end

jumpn3x: begin
mem_wr = 1'd0;
read_en <= 4'd0;
rst <=3'b000;
write_en <= 16'b0000000000000000;
inc_en <= 16'b0000000000000000;//pc =pc+1
alu_op <= 3'd0;
next <= fetch1;
end


mvacarp: begin
mem_wr = 1'd0;
read_en <= 4'd12;
rst <=3'b000;
write_en <= 16'b0000000010000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
next <= fetch1;
end

mvacacp: begin
mem_wr = 1'd0;
read_en <= 4'd12;
```

```
rst <=3'b000;
write_en <= 16'b0000000100000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
next <= fetch1;
end

mvacbcp: begin
mem_wr = 1'd0;
read_en <= 4'd12;
rst <=3'b000;
write_en <= 16'b0000001000000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
next <= fetch1;
end

incam: begin
mem_wr = 1'd0;
read_en <= 4'd0;
rst <=3'b000;
write_en <= 16'b0000000000000000;
inc_en <= 16'b0000000000010000;
alu_op <= 3'd0;
next <= fetch1;
end

incan: begin
mem_wr = 1'd0;
read_en <= 4'd0;
rst <=3'b000;
write_en <= 16'b0000000000000000;
inc_en <= 16'b0000000000100000;
alu_op <= 3'd0;
next <= fetch1;
end

incbn: begin
mem_wr = 1'd0;
read_en <= 4'd0;
rst <=3'b000;
write_en <= 16'b0000000000000000;
inc_en <= 16'b0000000001000000;
alu_op <= 3'd0;
```

```
next <= fetch1;
end

mvamr1: begin
mem_wr = 1'd0;
read_en <= 4'd5;
rst <=3'b000;
write_en <= 16'b0000010000000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
next <= fetch1;
end

mvanr1: begin
mem_wr = 1'd0;
read_en <= 4'd6;
rst <=3'b000;
write_en <= 16'b0000010000000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
next <= fetch1;
end

mvbnr1: begin
mem_wr = 1'd0;
read_en <= 4'd7;
rst <=3'b000;
write_en <= 16'b0000010000000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
next <= fetch1;
end

mvarpac: begin
mem_wr = 1'd0;
read_en <= 4'd8;
rst <=3'b000;
write_en <= 16'b0000100000000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
next <= fetch1;
end

mvacpac: begin
```

```
mem_wr = 1'd0;
read_en <= 4'd9;
rst <=3'b000;
write_en <= 16'b0000100000000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
next <= fetch1;
end

mvbcpac: begin
mem_wr = 1'd0;
read_en <= 4'd10;
rst <=3'b000;
write_en <= 16'b0000100000000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
next <= fetch1;
end

mvaaac: begin
mem_wr = 1'd0;
read_en <= 4'd1;
rst <=3'b000;
write_en <= 16'b0000100000000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
next <= fetch1;
end

mvabac: begin
mem_wr = 1'd0;
read_en <= 4'd3;
rst <=3'b000;
write_en <= 16'b0000100000000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
next <= fetch1;
end

mvadac: begin
mem_wr = 1'd0;
read_en <= 4'd13;
rst <=3'b000;
write_en <= 16'b0000100000000000;
```

```
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
next <= fetch1;
end

mvacr1: begin
mem_wr = 1'd0;
read_en <= 4'd12;
rst <=3'b000;
write_en <= 16'b0000010000000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
next <= fetch1;
end

mvacr2: begin
mem_wr = 1'd0;
read_en <= 4'd12;
rst <=3'b000;
write_en <= 16'b0000000000000010;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
next <= fetch1;
end

multi: begin
mem_wr = 1'd0;
read_en <= 4'd0;
rst <=3'b000;
write_en <= 16'b0000000000000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd1;
next <= multi1x;
end

multi1x: begin
mem_wr = 1'd0;
read_en <= 4'd0;
rst <=3'b000;
write_en <= 16'b0001000000000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd1;
next <= fetch1;
end
```

```
add: begin
mem_wr = 1'd0;
read_en <= 4'd0;
rst <=3'b000;
write_en <= 16'b0000000000000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd2;
next <= add1x;
end

add1x: begin
mem_wr = 1'd0;
read_en <= 4'd0;
rst <=3'b000;
write_en <= 16'b0001000000000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd2;
next <= fetch1;
end


sub: begin
mem_wr = 1'd0;
read_en <= 4'd0;
rst <=3'b000;
write_en <= 16'b0000000000000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd3;
next <= sub1x;
end

sub1x: begin
mem_wr = 1'd0;
read_en <= 4'd0;
rst <=3'b000;
write_en <= 16'b0001000000000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd3;
next <= fetch1;
end

rstan: begin
mem_wr = 1'd0;
```

```
read_en <= 4'd0;
rst <=3'b001;
write_en <= 16'b0000000000000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
next <= fetch1;
end

rstr2: begin
mem_wr = 1'd0;
read_en <= 4'd0;
rst <=3'b000;
write_en <= 16'b1000000000000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
next <= fetch1;
end

nop: begin
mem_wr = 1'd0;
read_en <= 4'd0;
rst <=3'b000;
write_en <= 16'b0000000000000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
next <= fetch1;
end

endop: begin
mem_wr = 1'd0;
read_en <= 4'd14;
rst <=3'b000;
write_en <= 16'b0000000000000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
next <= endop;
end

default: begin
mem_wr = 1'd0;
read_en <= 4'd0;
rst <=3'b000;
write_en <= 16'b0000000000000000;
inc_en <= 16'b0000000000000000;
```

```
                                                    alu_op <= 3'd0;
                                                    next <= fetch1;
                                                    end
                        endcase
            end
              else begin

                        case(present)

                                                    idle: begin
                                                    mem_wr <= 1'd0;
                                                    read_en <= 4'd0;
                                                    rst <=3'b000;
                                                    write_en <= 16'b0000000000000000;
                                                    inc_en <= 16'b0000000000000000;
                                                    alu_op <= 3'd0;
                                                    if (status == 2'b01)
                                                    next <= fetch1;
                                                    else
                                                    next <= idle;
                                                    end

                                                    fetch1: begin
                                                    mem_wr <= 1'd0;
                                                    read_en <= 4'd15;
                                                    rst <=3'b000;
                                                    write_en <= 16'b0000000000001000;
                                                    inc_en <= 16'b0000000000000000;
                                                    alu_op <= 3'd0;
                                                    next <= fetch1x;
                                                    end

                                                    fetch1x: begin
                                                    mem_wr <= 1'd0;
                                                    read_en <= 4'd0;
                                                    rst <=3'b000;
                                                    write_en <= 16'b0000000000000000;
                                                    inc_en <= 16'b0000000000000000;
                                                    alu_op <= 3'd0;
                                                    next <= fetch2;
                                                    end

                                                    fetch2: begin
                                                    mem_wr = 1'd0;
```

```
                                        read_en <= 4'd0;
                                        rst <=3'b000;
                                        write_en <= 16'b0000000000000000;
                                        inc_en <= 16'b0000000000000100;
                                        alu_op <= 3'd0;
                                        next <= nop;
                                        end

                                        nop: begin
                                        mem_wr = 1'd1;
                                        read_en <= 4'd0;
                                        rst <=3'b000;
                                        write_en <= 16'b0000000000000000;
                                        inc_en <= 16'b0000000000000000;
                                        alu_op <= 3'd0;
                                        next <= fetch1;
                                        end

                                        default: begin
                                        mem_wr = 1'd0;
                                        read_en <= 4'd0;
                                        rst <=3'b000;
                                        write_en <= 16'b0000000000000000;
                                        inc_en <= 16'b0000000000000000;
                                        alu_op <= 3'd0;
                                        next <= fetch1;
                                        end
                        endcase
  end

endmodule
```

11. control_c

```
module control_c(
      input clock,
      input [15:0] z,
      input [ 1: 0] status,
      input [ 15: 0] instruction,
      output reg [ 2: 0] alu_op,
      output reg [ 15: 0] write_en,
      output reg [ 3: 0] read_en,
      output reg [ 2: 0] rst,
output reg [ 15: 0] inc_en,
output reg mem_wr,
output reg end_process);

reg [ 5: 0] present = 6'd0;
reg [ 5: 0] next = 6'd0;
parameter fetch1 = 6'd1,
fetch2 = 6'd2,
loada1 = 6'd3,
loada2 = 6'd4,
loadb1 = 6'd5,
loadb2 = 6'd6,
store1 = 6'd7,
store2 = 6'd8,
jumpz1 = 6'd9,
jumpz2 = 6'd10,
jumpz3 = 6'd11,
fetch1x = 6'd50,
fetch2x = 6'd51,
loada1x = 6'd52,
loada2x = 6'd53,
loadb1x = 6'd54,
loadb2x = 6'd55,
loadb3x = 6'd56,
jumpz3x = 6'd57,
sub1x = 6'd58,
sub2x = 6'd59,
jumpz2x = 6'd60,
multi1x = 6'd61,
add1x = 6'd62,



jumpn1 = 6'd13,
jumpn2 = 6'd14,
jumpn3 = 6'd15,
jumpn2x = 6'd16,
jumpn3x = 6'd17,
store1x = 6'd18,
store2x = 6'd19,
mvr2r1 = 6'd20,
mvr2r1x = 6'd21,

mvacarp = 6'd23,
mvacacp = 6'd24,
mvacbcp = 6'd25,
incam = 6'd26,
incan = 6'd27,
incbn = 6'd28,
mvamr1 = 6'd29,
mvanr1 = 6'd30,
mvbnr1 = 6'd31,
mvarpac = 6'd32,
mvacpac = 6'd33,
mvbcpac = 6'd34,
mvaaac = 6'd35,
mvabac = 6'd36,
mvadac = 6'd37,
mvacr1 = 6'd38,
mvacr2 = 6'd39,
mvr2ac = 6'd40,
multi = 6'd41,
add = 6'd42,
sub = 6'd43,
rstan = 6'd45,
rstr2 = 6'd46,
nop = 6'd47,
endop = 6'd48,
idle = 6'd0;
always @(posedge clock)
present <= next;
always @(posedge clock)
                begin
                if (present == endop)
                end_process <= 1'd1;
                else
                end_process <= 1'd0;
                end
always @(present or instruction or status or z)
        if (instruction[15:12] == 4'd4 || instruction[15:12] == 4'd15)
                begin
                        case(present)

                                idle: begin
                                mem_wr <= 1'd0;
                                read_en <= 4'd0;
                                rst <=3'b000;
                                write_en <= 16'b0000000000000000;
```

```
inc_en <= 16'b00000000000000000;
alu_op <= 3'd0;
if (status == 2'b01)
next <= fetch1;
else
next <= idle;
end

fetch1: begin
mem_wr <= 1'd0;
read_en <= 4'd15;
rst <=3'b000;
write_en <= 16'b000000000000001000;
inc_en <= 16'b00000000000000000;
alu_op <= 3'd0;
next <= fetch1x;
end

fetch1x: begin
mem_wr <= 1'd0;
read_en <= 4'd0;
rst <=3'b000;
write_en <= 16'b00000000000000000;
inc_en <= 16'b00000000000000000;
alu_op <= 3'd0;
next <= fetch2;
end

fetch2: begin
mem_wr = 1'd0;
read_en <= 4'd0;
rst <=3'b000;
write_en <= 16'b00000000000000000;
inc_en <= 16'b00000000000000100;
alu_op <= 3'd0;
next <= fetch2x;
end

fetch2x: begin
mem_wr <= 1'd0;
read_en <= 4'd0;
rst <=3'b000;
write_en <= 16'b00000000000000000;
inc_en <= 16'b00000000000000000;
```

```
alu_op <= 3'd0;
next <= instruction[5:0];
end

loada1: begin
mem_wr <= 1'd0;
read_en <= 4'd15;
rst <=3'b000;
write_en <= 16'b00000000000000001;
inc_en <= 16'b0000000000000100;
alu_op <= 3'd0;
next <= loada2;
end

loada2: begin
mem_wr = 1'd0;
read_en <= 4'd14;
rst <=3'b000;
write_en <= 16'b0000100000000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
next <= fetch1;
end

mvr2r1: begin
mem_wr = 1'd0;
read_en <= 4'd2;
rst <=3'b000;
write_en <= 16'b0000010000000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
next <= fetch1;
end

loadb1: begin
mem_wr = 1'd0;
read_en <= 4'd12;
rst <=3'b000;
write_en <= 16'b0000000000000001;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
next <= loadb1x;
end
```

```
loadb1x: begin
mem_wr <= 1'd0;
read_en <= 4'd0;
rst <=3'b000;
write_en <= 16'b0000000000000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
next <= loadb2;
end

loadb2: begin
mem_wr = 1'd0;
read_en <= 4'd14;
rst <=3'b000;
write_en <= 16'b0000100000000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
next <= fetch1;
end

store1: begin
mem_wr = 1'd0;
read_en <= 4'd13;
rst <=3'b000;
write_en <= 16'b0000000000000001;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
next <= store2;
end

store2: begin
mem_wr = 1'd1;
read_en <= 4'd2;
rst <=3'b000;
write_en <= 16'b0010000000000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
next <= fetch1;
end

jumpz1: begin
mem_wr <= 1'd0;
read_en <= 4'd15;
rst <=3'b000;
```

```
write_en <= 16'b0000000010000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
if (z == 0)
next <= jumpz2;
else
next <= jumpz3;
end

jumpz2: begin
mem_wr = 1'd0;
read_en <= 4'd8;
rst <=3'b000;
write_en <= 16'b0000000000000100;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
next <= jumpz2x;
end

jumpz2x: begin
mem_wr = 1'd0;
read_en <= 4'd0;
rst <=3'b000;
write_en <= 16'b0000000000000000;
inc_en <= 16'b0000000000000000;//pc =pc+1
alu_op <= 3'd0;
next <= fetch1;
end

jumpz3: begin
mem_wr = 1'd0;
read_en <= 4'd0;
rst <=3'b000;
write_en <= 16'b0000000000000000;
inc_en <= 16'b0000000000000100;//pc =pc+1
alu_op <= 3'd0;
next <= jumpz3x;
end

jumpz3x: begin
mem_wr = 1'd0;
read_en <= 4'd0;
rst <=3'b000;
write_en <= 16'b0000000000000000;
```

```
inc_en <= 16'b0000000000000000;//pc =pc+1
alu_op <= 3'd0;
next <= fetch1;
end

jumpn1: begin
mem_wr <= 1'd0;
read_en <= 4'd15;
rst <=3'b000;
write_en <= 16'b0000000010000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
if (z == 1)
next <= jumpn2;
else
next <= jumpn3;
end

jumpn2: begin
mem_wr = 1'd0;
read_en <= 4'd8;
rst <=3'b000;
write_en <= 16'b0000000000000100;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
next <= jumpn2x;
end

jumpn2x: begin
mem_wr = 1'd0;
read_en <= 4'd0;
rst <=3'b000;
write_en <= 16'b0000000000000000;
inc_en <= 16'b0000000000000000;//pc =pc+1
alu_op <= 3'd0;
next <= fetch1;
end

jumpn3: begin
mem_wr = 1'd0;
read_en <= 4'd0;
rst <=3'b000;
write_en <= 16'b0000000000000000;
inc_en <= 16'b0000000000000100;//pc =pc+1
```

```
alu_op <= 3'd0;
next <= jumpn3x;
end

jumpn3x: begin
mem_wr = 1'd0;
read_en <= 4'd0;
rst <=3'b000;
write_en <= 16'b0000000000000000;
inc_en <= 16'b0000000000000000;//pc =pc+1
alu_op <= 3'd0;
next <= fetch1;
end


mvacarp: begin
mem_wr = 1'd0;
read_en <= 4'd12;
rst <=3'b000;
write_en <= 16'b0000000010000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
next <= fetch1;
end

mvacacp: begin
mem_wr = 1'd0;
read_en <= 4'd12;
rst <=3'b000;
write_en <= 16'b0000000100000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
next <= fetch1;
end

mvacbcp: begin
mem_wr = 1'd0;
read_en <= 4'd12;
rst <=3'b000;
write_en <= 16'b0000001000000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
next <= fetch1;
end
```

```
incam: begin
mem_wr = 1'd0;
read_en <= 4'd0;
rst <=3'b000;
write_en <= 16'b0000000000000000;
inc_en <= 16'b0000000000010000;
alu_op <= 3'd0;
next <= fetch1;
end

incan: begin
mem_wr = 1'd0;
read_en <= 4'd0;
rst <=3'b000;
write_en <= 16'b0000000000000000;
inc_en <= 16'b0000000000100000;
alu_op <= 3'd0;
next <= fetch1;
end

incbn: begin
mem_wr = 1'd0;
read_en <= 4'd0;
rst <=3'b000;
write_en <= 16'b0000000000000000;
inc_en <= 16'b0000000001000000;
alu_op <= 3'd0;
next <= fetch1;
end

mvamr1: begin
mem_wr = 1'd0;
read_en <= 4'd5;
rst <=3'b000;
write_en <= 16'b0000010000000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
next <= fetch1;
end

mvanr1: begin
mem_wr = 1'd0;
read_en <= 4'd6;
```

```
rst <=3'b000;
write_en <= 16'b0000010000000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
next <= fetch1;
end

mvbnr1: begin
mem_wr = 1'd0;
read_en <= 4'd7;
rst <=3'b000;
write_en <= 16'b0000010000000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
next <= fetch1;
end

mvarpac: begin
mem_wr = 1'd0;
read_en <= 4'd8;
rst <=3'b000;
write_en <= 16'b0000100000000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
next <= fetch1;
end

mvacpac: begin
mem_wr = 1'd0;
read_en <= 4'd9;
rst <=3'b000;
write_en <= 16'b0000100000000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
next <= fetch1;
end

mvbcpac: begin
mem_wr = 1'd0;
read_en <= 4'd10;
rst <=3'b000;
write_en <= 16'b0000100000000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
```

```
next <= fetch1;
end

mvaaac: begin
mem_wr = 1'd0;
read_en <= 4'd1;
rst <=3'b000;
write_en <= 16'b0000100000000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
next <= fetch1;
end

mvabac: begin
mem_wr = 1'd0;
read_en <= 4'd3;
rst <=3'b000;
write_en <= 16'b0000100000000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
next <= fetch1;
end

mvadac: begin
mem_wr = 1'd0;
read_en <= 4'd13;
rst <=3'b000;
write_en <= 16'b0000100000000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
next <= fetch1;
end

mvacr1: begin
mem_wr = 1'd0;
read_en <= 4'd12;
rst <=3'b000;
write_en <= 16'b0000010000000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
next <= fetch1;
end

mvacr2: begin
```

```
mem_wr = 1'd0;
read_en <= 4'd12;
rst <=3'b000;
write_en <= 16'b0000000000000010;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
next <= fetch1;
end

multi: begin
mem_wr = 1'd0;
read_en <= 4'd0;
rst <=3'b000;
write_en <= 16'b0000000000000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd1;
next <= multi1x;
end

multi1x: begin
mem_wr = 1'd0;
read_en <= 4'd0;
rst <=3'b000;
write_en <= 16'b0001000000000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd1;
next <= fetch1;
end

add: begin
mem_wr = 1'd0;
read_en <= 4'd0;
rst <=3'b000;
write_en <= 16'b0000000000000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd2;
next <= add1x;
end

add1x: begin
mem_wr = 1'd0;
read_en <= 4'd0;
rst <=3'b000;
write_en <= 16'b0001000000000000;
```

```
inc_en <= 16'b0000000000000000;
alu_op <= 3'd2;
next <= fetch1;
end

sub: begin
mem_wr = 1'd0;
read_en <= 4'd0;
rst <=3'b000;
write_en <= 16'b0000000000000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd3;
next <= sub1x;
end

sub1x: begin
mem_wr = 1'd0;
read_en <= 4'd0;
rst <=3'b000;
write_en <= 16'b0001000000000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd3;
next <= fetch1;
end

rstan: begin
mem_wr = 1'd0;
read_en <= 4'd0;
rst <=3'b001;
write_en <= 16'b0000000000000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
next <= fetch1;
end

rstr2: begin
mem_wr = 1'd0;
read_en <= 4'd0;
rst <=3'b000;
write_en <= 16'b1000000000000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
next <= fetch1;
end
```

```
                                        nop: begin
                                        mem_wr = 1'd0;
                                        read_en <= 4'd0;
                                        rst <=3'b000;
                                        write_en <= 16'b0000000000000000;
                                        inc_en <= 16'b0000000000000000;
                                        alu_op <= 3'd0;
                                        next <= fetch1;
                                        end

                                        endop: begin
                                        mem_wr = 1'd0;
                                        read_en <= 4'd14;
                                        rst <=3'b000;
                                        write_en <= 16'b0000000000000000;
                                        inc_en <= 16'b0000000000000000;
                                        alu_op <= 3'd0;
                                        next <= endop;
                                        end

                                        default: begin
                                        mem_wr = 1'd0;
                                        read_en <= 4'd0;
                                        rst <=3'b000;
                                        write_en <= 16'b0000000000000000;
                                        inc_en <= 16'b0000000000000000;
                                        alu_op <= 3'd0;
                                        next <= fetch1;
                                        end
                    endcase
          end
        else begin
                    case(present)

                                        idle: begin
                                        mem_wr <= 1'd0;
                                        read_en <= 4'd0;
                                        rst <=3'b000;
                                        write_en <= 16'b0000000000000000;
                                        inc_en <= 16'b0000000000000000;
                                        alu_op <= 3'd0;
                                        if (status == 2'b01)
                                        next <= fetch1;
```

```
else
next <= idle;
end

fetch1: begin
mem_wr <= 1'd0;
read_en <= 4'd15;
rst <=3'b000;
write_en <= 16'b00000000000001000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
next <= fetch1x;
end

fetch1x: begin
mem_wr <= 1'd0;
read_en <= 4'd0;
rst <=3'b000;
write_en <= 16'b0000000000000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
next <= fetch2;
end

fetch2: begin
mem_wr = 1'd0;
read_en <= 4'd0;
rst <=3'b000;
write_en <= 16'b0000000000000000;
inc_en <= 16'b0000000000000100;
alu_op <= 3'd0;
next <= nop;
end

nop: begin
mem_wr = 1'd1;
read_en <= 4'd0;
rst <=3'b000;
write_en <= 16'b0000000000000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
next <= fetch1;
end
```

```
                                        default: begin
                                        mem_wr = 1'd0;
                                        read_en <= 4'd0;
                                        rst <=3'b000;
                                        write_en <= 16'b0000000000000000;
                                        inc_en <= 16'b0000000000000000;
                                        alu_op <= 3'd0;
                                        next <= fetch1;
                                        end
                    endcase
  end


endmodule
```

12. control_d

```
module control_d(
      input clock,
      input [15:0] z,
      input [ 1: 0] status,
      input [ 15: 0] instruction,
      output reg [ 2: 0] alu_op,
      output reg [ 15: 0] write_en,
      output reg [ 3: 0] read_en,
      output reg [ 2: 0] rst,
      output reg [ 15: 0] inc_en,
      output reg mem_wr,
      output reg end_process);

      reg [ 5: 0] present = 6'd0;
      reg [ 5: 0] next = 6'd0;
      parameter fetch1 = 6'd1,
      fetch2 = 6'd2,
      loada1 = 6'd3,
      loada2 = 6'd4,
      loadb1 = 6'd5,
      loadb2 = 6'd6,
      store1 = 6'd7,
      store2 = 6'd8,
      jumpz1 = 6'd9,
      jumpz2 = 6'd10,
      jumpz3 = 6'd11,
      fetch1x = 6'd50,
fetch2x = 6'd51,
loada1x = 6'd52,
loada2x = 6'd53,
loadb1x = 6'd54,
loadb2x = 6'd55,
loadb3x = 6'd56,
jumpz3x = 6'd57,
sub1x = 6'd58,
sub2x = 6'd59,
jumpz2x = 6'd60,
multi1x = 6'd61,
add1x = 6'd62,



jumpn1 = 6'd13,
jumpn2 = 6'd14,
jumpn3 = 6'd15,
jumpn2x = 6'd16,
jumpn3x = 6'd17,
store1x = 6'd18,
store2x = 6'd19,
mvr2r1 = 6'd20,
mvr2r1x = 6'd21,

mvacarp = 6'd23,
mvacacp = 6'd24,
mvacbcp = 6'd25,
incam = 6'd26,
incan = 6'd27,
incbn = 6'd28,
mvamr1 = 6'd29,
mvanr1 = 6'd30,
mvbnr1 = 6'd31,
mvarpac = 6'd32,
mvacpac = 6'd33,
mvbcpac = 6'd34,
mvaaac = 6'd35,
mvabac = 6'd36,
mvadac = 6'd37,
mvacr1 = 6'd38,
mvacr2 = 6'd39,
mvr2ac = 6'd40,
multi = 6'd41,
add = 6'd42,
sub = 6'd43,
rstan = 6'd45,
rstr2 = 6'd46,
nop = 6'd47,
endop = 6'd48,
idle = 6'd0;
always @(posedge clock)
present <= next;
always @(posedge clock)
                begin
                if (present == endop)
                end_process <= 1'd1;
                else
                end_process <= 1'd0;
                end
always @(present or instruction or status or z)

        if (instruction[15:12] == 4'd8 || instruction[15:12] == 4'd15)

                begin
                        case(present)

                                idle: begin
                                mem_wr <= 1'd0;
                                read_en <= 4'd0;
                                rst <=3'b000;
                                write_en <= 16'b0000000000000000;
                                inc_en <= 16'b0000000000000000;
                                alu_op <= 3'd0;
                                if (status == 2'b01)
                                next <= fetch1;
                                else
                                next <= idle;
                                end

                                fetch1: begin
                                mem_wr <= 1'd0;
                                read_en <= 4'd15;
                                rst <=3'b000;
                                write_en <= 16'b0000000000001000;
                                inc_en <= 16'b0000000000000000;
                                alu_op <= 3'd0;
                                next <= fetch1x;
```

```
end

fetch1x: begin
mem_wr <= 1'd0;
read_en <= 4'd0;
rst <=3'b000;
write_en <= 16'b0000000000000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
next <= fetch2;
end

fetch2: begin
mem_wr = 1'd0;
read_en <= 4'd0;
rst <=3'b000;
write_en <= 16'b0000000000000000;
inc_en <= 16'b0000000000000100;
alu_op <= 3'd0;
next <= fetch2x;
end

fetch2x: begin
mem_wr <= 1'd0;
read_en <= 4'd0;
rst <=3'b000;
write_en <= 16'b0000000000000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
next <= instruction[5:0];
end

loada1: begin
mem_wr <= 1'd0;
read_en <= 4'd15;
rst <=3'b000;
write_en <= 16'b00000000000000001;
inc_en <= 16'b0000000000000100;
alu_op <= 3'd0;
next <= loada2;
end

loada2: begin
mem_wr = 1'd0;
```

```
read_en <= 4'd14;
rst <=3'b000;
write_en <= 16'b0000100000000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
next <= fetch1;
end

mvr2r1: begin
mem_wr = 1'd0;
read_en <= 4'd2;
rst <=3'b000;
write_en <= 16'b0000010000000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
next <= fetch1;
end

loadb1: begin
mem_wr = 1'd0;
read_en <= 4'd12;
rst <=3'b000;
write_en <= 16'b0000000000000001;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
next <= loadb1x;
end

loadb1x: begin
mem_wr <= 1'd0;
read_en <= 4'd0;
rst <=3'b000;
write_en <= 16'b0000000000000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
next <= loadb2;
end

loadb2: begin
mem_wr = 1'd0;
read_en <= 4'd14;
rst <=3'b000;
write_en <= 16'b0000100000000000;
inc_en <= 16'b0000000000000000;
```

```
alu_op <= 3'd0;
next <= fetch1;
end

store1: begin
mem_wr = 1'd0;
read_en <= 4'd13;
rst <=3'b000;
write_en <= 16'b0000000000000001;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
next <= store2;
end


store2: begin
mem_wr = 1'd1;
read_en <= 4'd2;
rst <=3'b000;
write_en <= 16'b0010000000000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
next <= fetch1;
end


jumpz1: begin
mem_wr <= 1'd0;
read_en <= 4'd15;
rst <=3'b000;
write_en <= 16'b0000000010000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
if (z == 0)
next <= jumpz2;
else
next <= jumpz3;
end

jumpz2: begin
mem_wr = 1'd0;
read_en <= 4'd8;
rst <=3'b000;
write_en <= 16'b0000000000000100;
```

```
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
next <= jumpz2x;
end

jumpz2x: begin
mem_wr = 1'd0;
read_en <= 4'd0;
rst <=3'b000;
write_en <= 16'b0000000000000000;
inc_en <= 16'b0000000000000000;//pc =pc+1
alu_op <= 3'd0;
next <= fetch1;
end

jumpz3: begin
mem_wr = 1'd0;
read_en <= 4'd0;
rst <=3'b000;
write_en <= 16'b0000000000000000;
inc_en <= 16'b0000000000000100;//pc =pc+1
alu_op <= 3'd0;
next <= jumpz3x;
end

jumpz3x: begin
mem_wr = 1'd0;
read_en <= 4'd0;
rst <=3'b000;
write_en <= 16'b0000000000000000;
inc_en <= 16'b0000000000000000;//pc =pc+1
alu_op <= 3'd0;
next <= fetch1;
end

jumpn1: begin
mem_wr <= 1'd0;
read_en <= 4'd15;
rst <=3'b000;
write_en <= 16'b0000000010000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
if (z == 1)
next <= jumpn2;
```

```
else
next <= jumpn3;
end

jumpn2: begin
mem_wr = 1'd0;
read_en <= 4'd8;
rst <=3'b000;
write_en <= 16'b0000000000000100;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
next <= jumpn2x;
end

jumpn2x: begin
mem_wr = 1'd0;
read_en <= 4'd0;
rst <=3'b000;
write_en <= 16'b0000000000000000;
inc_en <= 16'b0000000000000000;//pc =pc+1
alu_op <= 3'd0;
next <= fetch1;
end

jumpn3: begin
mem_wr = 1'd0;
read_en <= 4'd0;
rst <=3'b000;
write_en <= 16'b0000000000000000;
inc_en <= 16'b0000000000000100;//pc =pc+1
alu_op <= 3'd0;
next <= jumpn3x;
end

jumpn3x: begin
mem_wr = 1'd0;
read_en <= 4'd0;
rst <=3'b000;
write_en <= 16'b0000000000000000;
inc_en <= 16'b0000000000000000;//pc =pc+1
alu_op <= 3'd0;
next <= fetch1;
end
```

```
mvacarp: begin
mem_wr = 1'd0;
read_en <= 4'd12;
rst <=3'b000;
write_en <= 16'b0000000010000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
next <= fetch1;
end

mvacacp: begin
mem_wr = 1'd0;
read_en <= 4'd12;
rst <=3'b000;
write_en <= 16'b0000000100000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
next <= fetch1;
end

mvacbcp: begin
mem_wr = 1'd0;
read_en <= 4'd12;
rst <=3'b000;
write_en <= 16'b0000001000000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
next <= fetch1;
end

incam: begin
mem_wr = 1'd0;
read_en <= 4'd0;
rst <=3'b000;
write_en <= 16'b0000000000000000;
inc_en <= 16'b0000000000010000;
alu_op <= 3'd0;
next <= fetch1;
end

incan: begin
mem_wr = 1'd0;
read_en <= 4'd0;
rst <=3'b000;
```

```
write_en <= 16'b0000000000000000;
inc_en <= 16'b0000000000100000;
alu_op <= 3'd0;
next <= fetch1;
end

incbn: begin
mem_wr = 1'd0;
read_en <= 4'd0;
rst <=3'b000;
write_en <= 16'b0000000000000000;
inc_en <= 16'b0000000001000000;
alu_op <= 3'd0;
next <= fetch1;
end

mvamr1: begin
mem_wr = 1'd0;
read_en <= 4'd5;
rst <=3'b000;
write_en <= 16'b0000010000000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
next <= fetch1;
end

mvanr1: begin
mem_wr = 1'd0;
read_en <= 4'd6;
rst <=3'b000;
write_en <= 16'b0000010000000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
next <= fetch1;
end

mvbnr1: begin
mem_wr = 1'd0;
read_en <= 4'd7;
rst <=3'b000;
write_en <= 16'b0000010000000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
next <= fetch1;
```

```
end

mvarpac: begin
mem_wr = 1'd0;
read_en <= 4'd8;
rst <=3'b000;
write_en <= 16'b0000100000000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
next <= fetch1;
end

mvacpac: begin
mem_wr = 1'd0;
read_en <= 4'd9;
rst <=3'b000;
write_en <= 16'b0000100000000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
next <= fetch1;
end

mvbcpac: begin
mem_wr = 1'd0;
read_en <= 4'd10;
rst <=3'b000;
write_en <= 16'b0000100000000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
next <= fetch1;
end

mvaaac: begin
mem_wr = 1'd0;
read_en <= 4'd1;
rst <=3'b000;
write_en <= 16'b0000100000000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
next <= fetch1;
end

mvabac: begin
mem_wr = 1'd0;
```

```
read_en <= 4'd3;
rst <=3'b000;
write_en <= 16'b0000100000000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
next <= fetch1;
end

mvadac: begin
mem_wr = 1'd0;
read_en <= 4'd13;
rst <=3'b000;
write_en <= 16'b0000100000000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
next <= fetch1;
end

mvacr1: begin
mem_wr = 1'd0;
read_en <= 4'd12;
rst <=3'b000;
write_en <= 16'b0000010000000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
next <= fetch1;
end

mvacr2: begin
mem_wr = 1'd0;
read_en <= 4'd12;
rst <=3'b000;
write_en <= 16'b0000000000000010;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
next <= fetch1;
end

multi: begin
mem_wr = 1'd0;
read_en <= 4'd0;
rst <=3'b000;
write_en <= 16'b0000000000000000;
inc_en <= 16'b0000000000000000;
```

```
alu_op <= 3'd1;
next <= multi1x;
end

multi1x: begin
mem_wr = 1'd0;
read_en <= 4'd0;
rst <=3'b000;
write_en <= 16'b0001000000000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd1;
next <= fetch1;
end

add: begin
mem_wr = 1'd0;
read_en <= 4'd0;
rst <=3'b000;
write_en <= 16'b0000000000000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd2;
next <= add1x;
end

add1x: begin
mem_wr = 1'd0;
read_en <= 4'd0;
rst <=3'b000;
write_en <= 16'b0001000000000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd2;
next <= fetch1;
end


sub: begin
mem_wr = 1'd0;
read_en <= 4'd0;
rst <=3'b000;
write_en <= 16'b0000000000000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd3;
next <= sub1x;
end
```

```
sub1x: begin
mem_wr = 1'd0;
read_en <= 4'd0;
rst <=3'b000;
write_en <= 16'b0001000000000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd3;
next <= fetch1;
end

rstan: begin
mem_wr = 1'd0;
read_en <= 4'd0;
rst <=3'b001;
write_en <= 16'b0000000000000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
next <= fetch1;
end

rstr2: begin
mem_wr = 1'd0;
read_en <= 4'd0;
rst <=3'b000;
write_en <= 16'b1000000000000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
next <= fetch1;
end

nop: begin
mem_wr = 1'd0;
read_en <= 4'd0;
rst <=3'b000;
write_en <= 16'b0000000000000000;
inc_en <= 16'b0000000000000000;
alu_op <= 3'd0;
next <= fetch1;
end

endop: begin
mem_wr = 1'd0;
read_en <= 4'd14;
```

```
                                        rst <=3'b000;
                                        write_en <= 16'b0000000000000000;
                                        inc_en <= 16'b0000000000000000;
                                        alu_op <= 3'd0;
                                        next <= endop;
                                        end

                                        default: begin
                                        mem_wr = 1'd0;
                                        read_en <= 4'd0;
                                        rst <=3'b000;
                                        write_en <= 16'b0000000000000000;
                                        inc_en <= 16'b0000000000000000;
                                        alu_op <= 3'd0;
                                        next <= fetch1;
                                        end
                      endcase
   end
else begin
                      case(present)

                                        idle: begin
                                        mem_wr <= 1'd0;
                                        read_en <= 4'd0;
                                        rst <=3'b000;
                                        write_en <= 16'b0000000000000000;
                                        inc_en <= 16'b0000000000000000;
                                        alu_op <= 3'd0;
                                        if (status == 2'b01)
                                        next <= fetch1;
                                        else
                                        next <= idle;
                                        end

                                        fetch1: begin
                                        mem_wr <= 1'd0;
                                        read_en <= 4'd15;
                                        rst <=3'b000;
                                        write_en <= 16'b0000000000001000;
                                        inc_en <= 16'b0000000000000000;
                                        alu_op <= 3'd0;
                                        next <= fetch1x;
                                        end
                                        fetch1x: begin
                                        mem_wr <= 1'd0;
                                        read_en <= 4'd0;
                                        rst <=3'b000;
                                        write_en <= 16'b0000000000000000;
                                        inc_en <= 16'b0000000000000000;
                                        alu_op <= 3'd0;
                                        next <= fetch2;
                                        end

                                        fetch2: begin
                                        mem_wr = 1'd0;
                                        read_en <= 4'd0;
                                        rst <=3'b000;
                                        write_en <= 16'b0000000000000000;
                                        inc_en <= 16'b0000000000000100;
                                        alu_op <= 3'd0;
                                        next <= nop;
                                        end

                                        nop: begin
                                        mem_wr = 1'd1;
                                        read_en <= 4'd0;
                                        rst <=3'b000;
                                        write_en <= 16'b0000000000000000;
                                        inc_en <= 16'b0000000000000000;
                                        alu_op <= 3'd0;
                                        next <= fetch1;
                                        end

                                        default: begin
                                        mem_wr = 1'd0;
                                        read_en <= 4'd0;
                                        rst <=3'b000;
                                        write_en <= 16'b0000000000000000;
                                        inc_en <= 16'b0000000000000000;
                                        alu_op <= 3'd0;
                                        next <= fetch1;
                                        end
                      endcase
  end


endmodule
```

13. alu

```
module alu(clock,in1,in2,alu_op,alu_out,z);

  input clock;
  input [15:0] in1;
  input [15:0] in2;
  input [2:0] alu_op;
  output reg [15:0] alu_out;
  output reg [15:0] z;

  always @(posedge clock)
    begin

                if (alu_op == 3'd1) begin
                      alu_out <= in1 * in2;

                end else if (alu_op == 3'd2) begin
               alu_out <= in1 + in2;

                end else if (alu_op == 3'd3) begin
                      alu_out <= in2 - in1;

                end else begin
                      alu_out <=alu_out;
                end

    if (alu_out == 0) begin
          z <= 0;
                end else begin
          z <= 1;
                end

         end


endmodule
```

14. address_select

```
module address_select(clock,in1 ,in2 ,in3 ,out_address_aa,out_address_ab,out_address_ad);

        input clock;
        input [15:0] in1;
        input [15:0] in2;
        input [15:0] in3;
        output[15:0] out_address_aa;
        output[15:0] out_address_ab;
        output[15:0] out_address_ad;

        integer number_1;
        integer number_2;
        integer number_3;
        reg [15:0] out_address_aa;
        reg [15:0] out_address_ab;
        reg [15:0] out_address_ad;

        always @(in1)
                number_1 = in1;
        always @(in2)
                number_2 = in2;
        always @(in3)
                number_3 = in3;

        always @(posedge clock)
         begin
                if (number_1 == 0 && number_2 == 0 && number_3 == 0)
                                begin
                                        out_address_aa = 16'd3;
                                        out_address_ab = 16'd8;
                                        out_address_ad = 16'd24;
                                end
                else if (number_1 == 0 && number_2 == 1 && number_3 == 0)
                                begin
                                        out_address_aa = 16'd4;
                                        out_address_ab = 16'd9;
                                        out_address_ad = 16'd24;
                                end
                else if (number_1 == 0 && number_2 == 2 && number_3 == 0)
```

```
                              begin
                                        out_address_aa = 16'd5;
                                        out_address_ab = 16'd10;
                                        out_address_ad = 16'd24;
                              end
else if (number_1 == 0 && number_2 == 3 && number_3 == 0)
                              begin
                                        out_address_aa = 16'd6;
                                        out_address_ab = 16'd11;
                                        out_address_ad = 16'd24;
                              end
else if (number_1 == 0 && number_2 == 0 && number_3 == 1)
                              begin
                                        out_address_aa = 16'd3;
                                        out_address_ab = 16'd12;
                                        out_address_ad = 16'd25;
                              end
else if (number_1 == 0 && number_2 == 1 && number_3 == 1)
                              begin
                                        out_address_aa = 16'd4;
                                        out_address_ab = 16'd13;
                                        out_address_ad = 16'd25;
                              end
else if (number_1 == 0 && number_2 == 2 && number_3 == 1)
                              begin
                                        out_address_aa = 16'd5;
                                        out_address_ab = 16'd14;
                                        out_address_ad = 16'd25;
                              end
else if (number_1 == 0 && number_2 == 3 && number_3 == 1)
                              begin
                                        out_address_aa = 16'd6;
                                        out_address_ab = 16'd15;
                                        out_address_ad = 16'd25;
                              end
else if (number_1 == 0 && number_2 == 0 && number_3 == 2)
                              begin
                                        out_address_aa = 16'd3;
                                        out_address_ab = 16'd16;
                                        out_address_ad = 16'd26;
                              end
else if (number_1 == 0 && number_2 == 1 && number_3 == 2)
                              begin
                                        out_address_aa = 16'd4;
```

```
                        out_address_ab = 16'd17;
                        out_address_ad = 16'd26;
                end
else if (number_1 == 0 && number_2 == 2 && number_3 == 2)
                begin
                        out_address_aa = 16'd5;
                        out_address_ab = 16'd18;
                        out_address_ad = 16'd26;
                end
else if (number_1 == 0 && number_2 == 3 && number_3 == 2)
                begin
                        out_address_aa = 16'd6;
                        out_address_ab = 16'd19;
                        out_address_ad = 16'd26;
                end
else if (number_1 == 0 && number_2 == 0 && number_3 == 3)
                begin
                        out_address_aa = 16'd3;
                        out_address_ab = 16'd20;
                        out_address_ad = 16'd27;
                end
else if (number_1 == 0 && number_2 == 1 && number_3 == 3)
                begin
                        out_address_aa = 16'd4;
                        out_address_ab = 16'd21;
                        out_address_ad = 16'd27;
                end
else if (number_1 == 0 && number_2 == 2 && number_3 == 3)
                begin
                        out_address_aa = 16'd5;
                        out_address_ab = 16'd22;
                        out_address_ad = 16'd27;
                end
else if (number_1 == 0 && number_2 == 3 && number_3 == 3)
                begin
                        out_address_aa = 16'd6;
                        out_address_ab = 16'd23;
                        out_address_ad = 16'd27;
                end
else if (number_1 == 1 && number_2 == 0 && number_3 == 0)
                begin
                        out_address_aa = 16'd3;
                        out_address_ab = 16'd8;
                        out_address_ad = 16'd24;
                end
else if (number_1 == 1 && number_2 == 1 && number_3 == 0)
                begin
                        out_address_aa = 16'd4;
                        out_address_ab = 16'd9;
                        out_address_ad = 16'd24;
                end
else if (number_1 == 1 && number_2 == 2 && number_3 == 0)
                begin
                        out_address_aa = 16'd5;
                        out_address_ab = 16'd10;
                        out_address_ad = 16'd24;
                end
else if (number_1 == 1 && number_2 == 3 && number_3 == 0)
                begin
                        out_address_aa = 16'd6;
                        out_address_ab = 16'd11;
                        out_address_ad = 16'd24;
                end
else if (number_1 == 1 && number_2 == 0 && number_3 == 1)
                begin
                        out_address_aa = 16'd3;
                        out_address_ab = 16'd12;
                        out_address_ad = 16'd25;
                end
else if (number_1 == 1 && number_2 == 1 && number_3 == 1)
                begin
                        out_address_aa = 16'd4;
                        out_address_ab = 16'd13;
                        out_address_ad = 16'd25;
                end
else if (number_1 == 1 && number_2 == 2 && number_3 == 1)
                begin
                        out_address_aa = 16'd5;
                        out_address_ab = 16'd14;
                        out_address_ad = 16'd25;
                end
else if (number_1 == 1 && number_2 == 3 && number_3 == 1)
                begin
                        out_address_aa = 16'd6;
                        out_address_ab = 16'd15;
                        out_address_ad = 16'd25;
                end
else if (number_1 == 1 && number_2 == 0 && number_3 == 2)
```

```
                        begin
                                out_address_aa = 16'd3;
                                out_address_ab = 16'd16;
                                out_address_ad = 16'd26;
                        end
else if (number_1 == 1 && number_2 == 1 && number_3 == 2)
                        begin
                                out_address_aa = 16'd4;
                                out_address_ab = 16'd17;
                                out_address_ad = 16'd26;
                        end
else if (number_1 == 1 && number_2 == 2 && number_3 == 2)
                        begin
                                out_address_aa = 16'd5;
                                out_address_ab = 16'd18;
                                out_address_ad = 16'd26;
                        end
else if (number_1 == 1 && number_2 == 3 && number_3 == 2)
                        begin
                                out_address_aa = 16'd6;
                                out_address_ab = 16'd19;
                                out_address_ad = 16'd26;
                        end
else if (number_1 == 1 && number_2 == 0 && number_3 == 3)
                        begin
                                out_address_aa = 16'd3;
                                out_address_ab = 16'd20;
                                out_address_ad = 16'd27;
                        end
else if (number_1 == 1 && number_2 == 1 && number_3 == 3)
                        begin
                                out_address_aa = 16'd4;
                                out_address_ab = 16'd21;
                                out_address_ad = 16'd27;
                        end
else if (number_1 == 1 && number_2 == 2 && number_3 == 3)
                        begin
                                out_address_aa = 16'd5;
                                out_address_ab = 16'd22;
                                out_address_ad = 16'd27;
                        end
else if (number_1 == 1 && number_2 == 3 && number_3 == 3)
                        begin
                                out_address_aa = 16'd6;
```

```
                                        out_address_ab = 16'd23;
                                        out_address_ad = 16'd27;
                              end
else if (number_1 == 2 && number_2 == 0 && number_3 == 0)
                              begin
                                        out_address_aa = 16'd3;
                                        out_address_ab = 16'd8;
                                        out_address_ad = 16'd24;
                              end
else if (number_1 == 2 && number_2 == 1 && number_3 == 0)
                              begin
                                        out_address_aa = 16'd4;
                                        out_address_ab = 16'd9;
                                        out_address_ad = 16'd24;
                              end
else if (number_1 == 2 && number_2 == 2 && number_3 == 0)
                              begin
                                        out_address_aa = 16'd5;
                                        out_address_ab = 16'd10;
                                        out_address_ad = 16'd24;
                              end
else if (number_1 == 2 && number_2 == 3 && number_3 == 0)
                              begin
                                        out_address_aa = 16'd6;
                                        out_address_ab = 16'd11;
                                        out_address_ad = 16'd24;
                              end
else if (number_1 == 2 && number_2 == 0 && number_3 == 1)
                              begin
                                        out_address_aa = 16'd3;
                                        out_address_ab = 16'd12;
                                        out_address_ad = 16'd25;
                              end
else if (number_1 == 2 && number_2 == 1 && number_3 == 1)
                              begin
                                        out_address_aa = 16'd4;
                                        out_address_ab = 16'd13;
                                        out_address_ad = 16'd25;
                              end
else if (number_1 == 2 && number_2 == 2 && number_3 == 1)
                              begin
                                        out_address_aa = 16'd5;
                                        out_address_ab = 16'd14;
                                        out_address_ad = 16'd25;
```

```
                                end
else if (number_1 == 2 && number_2 == 3 && number_3 == 1)
                                begin
                                        out_address_aa = 16'd6;
                                        out_address_ab = 16'd15;
                                        out_address_ad = 16'd25;
                                end
else if (number_1 == 2 && number_2 == 0 && number_3 == 2)
                                begin
                                        out_address_aa = 16'd3;
                                        out_address_ab = 16'd16;
                                        out_address_ad = 16'd26;
                                end
else if (number_1 == 2 && number_2 == 1 && number_3 == 2)
                                begin
                                        out_address_aa = 16'd4;
                                        out_address_ab = 16'd17;
                                        out_address_ad = 16'd26;
                                end
else if (number_1 == 2 && number_2 == 2 && number_3 == 2)
                                begin
                                        out_address_aa = 16'd5;
                                        out_address_ab = 16'd18;
                                        out_address_ad = 16'd26;
                                end
else if (number_1 == 2 && number_2 == 3 && number_3 == 2)
                                begin
                                        out_address_aa = 16'd6;
                                        out_address_ab = 16'd19;
                                        out_address_ad = 16'd26;
                                end
else if (number_1 == 2 && number_2 == 0 && number_3 == 3)
                                begin
                                        out_address_aa = 16'd3;
                                        out_address_ab = 16'd20;
                                        out_address_ad = 16'd27;
                                end
else if (number_1 == 2 && number_2 == 1 && number_3 == 3)
                                begin
                                        out_address_aa = 16'd4;
                                        out_address_ab = 16'd21;
                                        out_address_ad = 16'd27;
                                end
else if (number_1 == 2 && number_2 == 2 && number_3 == 3)
```

```
                        begin
                                out_address_aa = 16'd5;
                                out_address_ab = 16'd22;
                                out_address_ad = 16'd27;
                        end
else if (number_1 == 2 && number_2 == 3 && number_3 == 3)
                        begin
                                out_address_aa = 16'd6;
                                out_address_ab = 16'd23;
                                out_address_ad = 16'd27;
                        end
else if (number_1 == 3 && number_2 == 0 && number_3 == 0)
                        begin
                                out_address_aa = 16'd3;
                                out_address_ab = 16'd8;
                                out_address_ad = 16'd24;
                        end
else if (number_1 == 3 && number_2 == 1 && number_3 == 0)
                        begin
                                out_address_aa = 16'd4;
                                out_address_ab = 16'd9;
                                out_address_ad = 16'd24;
                        end
else if (number_1 == 3 && number_2 == 2 && number_3 == 0)
                        begin
                                out_address_aa = 16'd5;
                                out_address_ab = 16'd10;
                                out_address_ad = 16'd24;
                        end
else if (number_1 == 3 && number_2 == 3 && number_3 == 0)
                        begin
                                out_address_aa = 16'd6;
                                out_address_ab = 16'd11;
                                out_address_ad = 16'd24;
                        end
else if (number_1 == 3 && number_2 == 0 && number_3 == 1)
                        begin
                                out_address_aa = 16'd3;
                                out_address_ab = 16'd12;
                                out_address_ad = 16'd25;
                        end
else if (number_1 == 3 && number_2 == 1 && number_3 == 1)
                        begin
                                out_address_aa = 16'd4;
```

```
                                        out_address_ab = 16'd13;
                                        out_address_ad = 16'd25;
                              end
else if (number_1 == 3 && number_2 == 2 && number_3 == 1)
                              begin
                                        out_address_aa = 16'd5;
                                        out_address_ab = 16'd14;
                                        out_address_ad = 16'd25;
                              end
else if (number_1 == 3 && number_2 == 3 && number_3 == 1)
                              begin
                                        out_address_aa = 16'd6;
                                        out_address_ab = 16'd15;
                                        out_address_ad = 16'd25;
                              end
else if (number_1 == 3 && number_2 == 0 && number_3 == 2)
                              begin
                                        out_address_aa = 16'd3;
                                        out_address_ab = 16'd16;
                                        out_address_ad = 16'd26;
                              end
else if (number_1 == 3 && number_2 == 1 && number_3 == 2)
                              begin
                                        out_address_aa = 16'd4;
                                        out_address_ab = 16'd17;
                                        out_address_ad = 16'd26;
                              end
else if (number_1 == 3 && number_2 == 2 && number_3 == 2)
                              begin
                                        out_address_aa = 16'd5;
                                        out_address_ab = 16'd18;
                                        out_address_ad = 16'd26;
                              end
else if (number_1 == 3 && number_2 == 3 && number_3 == 2)
                              begin
                                        out_address_aa = 16'd6;
                                        out_address_ab = 16'd19;
                                        out_address_ad = 16'd26;
                              end
else if (number_1 == 3 && number_2 == 0 && number_3 == 3)
                              begin
                                        out_address_aa = 16'd3;
                                        out_address_ab = 16'd20;
                                        out_address_ad = 16'd27;
                              end
            else if (number_1 == 3 && number_2 == 1 && number_3 == 3)
                              begin
                                    out_address_aa = 16'd4;
                                    out_address_ab = 16'd21;
                                    out_address_ad = 16'd27;
                              end
            else if (number_1 == 3 && number_2 == 2 && number_3 == 3)
                              begin
                                    out_address_aa = 16'd5;
                                    out_address_ab = 16'd22;
                                    out_address_ad = 16'd27;
                              end
            else if (number_1 == 3 && number_2 == 3 && number_3 == 3)
                              begin
                                    out_address_aa = 16'd6;
                                    out_address_ab = 16'd23;
                                    out_address_ad = 16'd27;
                              end
            end
endmodule
```

15. bus

```
module bus(aa, r2, ab, ir, am, an, bn, arp, acp, bcp,ac, ad, dm, im ,read_en,clock,busout);
      input clock;
      input [ 3: 0] read_en;
      input [ 15: 0] aa;
      input [ 15: 0] r2;
      input [ 15: 0] ab;
      input [ 15: 0] ir;
      input [ 15: 0] am;
      input [ 15: 0] an;
      input [ 15: 0] bn;
      input [ 15: 0] arp;
      input [ 15: 0] acp;
      input [ 15: 0] bcp;
      input [ 15: 0] ac;
      input [ 15: 0] ad;
      input [ 15: 0] dm;
      input [ 15: 0] im;
      output reg [ 15: 0] busout;
```

```
        always @(aa or r2 or ab or ir or am or an or bn or arp or acp or bcp or ac or ad or im or
read_en or dm)
                begin
                        case(read_en)
                                                        4'd1: busout <= aa;
                                                        4'd2: busout <= r2;
                                                        4'd3: busout <= ab;
                                                        4'd4: busout <= ir;
                                                        4'd5: busout <= am;
                                                        4'd6: busout <= an;
                                                        4'd7: busout <= bn;
                                                        4'd8: busout <= arp;
                                                        4'd9: busout <= acp;
                                                        4'd10: busout <= bcp;
                                                        4'd12: busout <= ac;
                                                        4'd13: busout <= ad;
                                                        4'd14: busout <= dm ;
                                                        4'd15: busout <= im ;
                                                        default: busout <= 16'd0;
                        endcase
                end
endmodule
```

16. regr

```
module regr(clock,write_en,datain,dataout);
        input clock,write_en;
        input [15:0] datain;
        output reg [15:0] dataout;

        always @(posedge clock)
        begin
        if (write_en == 1)
                dataout <= datain;

        end
endmodule
```

17. regrinc

```
module regrinc(clock,write_en,datain,dataout,inc_en);
```

```
        input clock,write_en,inc_en;

        input [15:0] datain;

        output reg [15:0] dataout=16'd0;

        always @(posedge clock)

                begin

                if (write_en == 1)
                        dataout <= datain;
                if (inc_en == 1)
                        dataout <= dataout + 16'd1;
        end
endmodule
```

18. regr2

```
module regr2(clock,write_en,rst,dm_mem_wr_en,datain,dataout,dm_mem_out);
        input clock,write_en,dm_mem_wr_en,rst;
        input [15:0] datain;
        output reg [15:0] dataout = 16'd0;
        output reg [15:0] dm_mem_out;


        always @(posedge clock)
        begin

        if (write_en == 1)begin
                dataout <= datain;
        end

        if (dm_mem_wr_en == 1)begin
                dm_mem_out <= datain;
        end

        if (rst == 1)begin
           dataout <= 16'd0;
        end

        end
```

```
endmodule
```

19. regir

```
module regir(clock,write_en,alu_to_ac,datain,datain_alu,dataout,alu_data);

        input clock,write_en,alu_to_ac;

        input [15:0] datain;
        input [15:0] datain_alu;

        output reg [15:0] dataout;

        output reg [15:0] alu_data;
        always @(posedge clock)

                begin
                if (write_en == 1)
                        dataout <= datain;

                if (alu_to_ac == 1)
                        dataout <= datain_alu;
          end

endmodule
```

20. regran

```
module regran(clock,write_en,rst,datain,dataout,inc_en);

        input clock,write_en,inc_en,rst;

        input [15:0] datain;

        output reg [15:0] dataout=16'd0;

        always @(posedge clock)

                begin

                if (write_en == 1)
                        dataout <= datain;
                if (inc_en == 1)
                        dataout <= dataout + 16'd1;

    if(rst == 1) begin
                  dataout <= 16'd0;
                        end
        end

endmodule
```

II. Matlab code

```
A = [1,2,3,4; 5,6,7,8; 9,10,11,12; 13,14,15,16;];
B = [10,20,30,40; 50,60,70,80; 90,100,110,120; 130,140,150,160;];

[rows_A, columns_A]  = size(A);
[rows_B,columns_B] = size(B);
dimensions = [rows_A;columns_A;columns_B];

X = dec2bin(transpose(A));
fid = fopen('Matrix_A.txt','wt');
fprintf(fid, '%s\n', string(X));
fclose(fid);

Y = dec2bin(B);
fid = fopen('Matrix_B.txt','wt');
fprintf(fid, '%s\n', string(Y));
fclose(fid);

z = dec2bin(dimensions);
fid = fopen('Dimensions_matrix.txt','wt');
fprintf(fid, '%s\n', string(z));
fclose(fid);

C = A*B
```

III. Test Bench for the modules.

1. Ultimate_tb

```
module ultimate_tb();

        reg clock;
        reg start_process;

        integer k;

  ultimate dut (.clock(clock), .start_process(start_process));

        initial
                begin
                        clock = 0;
                        start_process=1;


                        for (k=0; k<4000; k=k+1)
                                begin
                                        #50  clock =  ! clock;
                                end
                end

endmodule
```

2. Instruction_memory_tb

```
module instr_tb();
        reg clock;
        reg [ 15: 0] addr_a;
        reg [ 15: 0] addr_b;
        reg [ 15: 0] addr_c;
        reg [ 15: 0] addr_d;
        wire [15:0] instr_out_a;
        wire [15:0] instr_out_b;
        wire [15:0] instr_out_c;
        wire [15:0] instr_out_d;

 instr_memory dut (.clock(clock), .addr_a(addr_a), .addr_b(addr_b), .addr_c(addr_c),
.addr_d(addr_d), .instr_out_a(instr_out_a), .instr_out_b(instr_out_b), .instr_out_c(instr_out_c),
.instr_out_d(instr_out_d));

 initial begin
 #50;

 clock = 1'd1; addr_a   = 16'd1; addr_b = 16'd2; addr_c = 16'd3; addr_d = 16'd4; #100;
 clock = 1'd0; addr_a   = 16'd1; addr_b = 16'd2; addr_c = 16'd3; addr_d = 16'd4; #100;
 clock = 1'd1; addr_a   = 16'd5; addr_b = 16'd6; addr_c = 16'd7; addr_d = 16'd8; #100;
 clock = 1'd0; addr_a   = 16'd5; addr_b = 16'd6; addr_c = 16'd7; addr_d = 16'd8; #100;
 clock = 1'd1; addr_a   = 16'd9; addr_b = 16'd10; addr_c = 16'd11;addr_d = 16'd12;#100;
 #100;
 clock = 1'd0; addr_a   = 16'd9; addr_b = 16'd10; addr_c = 16'd11; addr_d = 16'd12;#100;
 #100;
 clock = 1'd1; addr_a   = 16'd13; addr_b = 16'd14; addr_c = 16'd15; addr_d = 16'd16; #100;
 clock = 1'd0; addr_a   = 16'd13; addr_b = 16'd14; addr_c = 16'd15; addr_d = 16'd16; #100;
 clock = 1'd1; addr_a   = 16'd17; addr_b = 16'd18; addr_c = 16'd19; addr_d = 16'd20; #100;
 clock = 1'd0; addr_a   = 16'd17; addr_b = 16'd18; addr_c = 16'd19; addr_d = 16'd20; #100;
 clock = 1'd1; addr_a   = 16'd21; addr_b = 16'd22; addr_c = 16'd23; addr_d = 16'd24; #100;
 clock = 1'd0; addr_a   = 16'd21; addr_b = 16'd22; addr_c = 16'd23; addr_d = 16'd24; #100;
 clock = 1'd1; addr_a   = 16'd25; addr_b = 16'd26; addr_c = 16'd27; addr_d = 16'd28; #100;
 clock = 1'd0; addr_a   = 16'd25; addr_b = 16'd26; addr_c = 16'd27; addr_d = 16'd28; #100;
 clock = 1'd1; addr_a   = 16'd29; addr_b = 16'd30; addr_c = 16'd31; addr_d = 16'd32; #100;
 clock = 1'd0; addr_a   = 16'd29; addr_b = 16'd30; addr_c = 16'd31; addr_d = 16'd32; #100;

 clock = 1'd1; addr_a   = 16'd33; addr_b = 16'd34; addr_c = 16'd35; addr_d = 16'd36; #100;
 clock = 1'd0; addr_a   = 16'd33; addr_b = 16'd34; addr_c = 16'd35; addr_d = 16'd36; #100;

 clock = 1'd1; addr_a   = 16'd37; addr_b = 16'd38; addr_c = 16'd39; addr_d = 16'd40; #100;
 clock = 1'd0; addr_a   = 16'd37; addr_b = 16'd38; addr_c = 16'd39; addr_d = 16'd40; #100;


 clock = 1'd1; addr_a   = 16'd41; addr_b = 16'd42; addr_c = 16'd43; addr_d = 16'd44; #100;
 clock = 1'd0; addr_a   = 16'd41; addr_b = 16'd42; addr_c = 16'd43; addr_d = 16'd44; #100;


 clock = 1'd1; addr_a   = 16'd45; addr_b = 16'd46; addr_c = 16'd47; addr_d = 16'd48; #100;
 clock = 1'd0; addr_a   = 16'd45; addr_b = 16'd46; addr_c = 16'd47; addr_d = 16'd48; #100;

 clock = 1'd1; addr_a   = 16'd49; addr_b = 16'd50; addr_c = 16'd51; addr_d = 16'd52; #100;
 clock = 1'd0; addr_a   = 16'd49; addr_b = 16'd50; addr_c = 16'd51; addr_d = 16'd52; #100;

 clock = 1'd1; addr_a   = 16'd53; addr_b = 16'd54; addr_c = 16'd55; addr_d = 16'd56; #100;
 clock = 1'd0; addr_a   = 16'd53; addr_b = 16'd54; addr_c = 16'd55; addr_d = 16'd56; #100;

 clock = 1'd1; addr_a   = 16'd57; addr_b = 16'd58; addr_c = 16'd59; addr_d = 16'd60; #100;
 clock = 1'd0; addr_a   = 16'd57; addr_b = 16'd58; addr_c = 16'd59; addr_d = 16'd60; #100;

 clock = 1'd1; addr_a   = 16'd61; addr_b = 16'd62; addr_c = 16'd63; addr_d = 16'd64; #100;
 clock = 1'd0; addr_a   = 16'd61; addr_b = 16'd62; addr_c = 16'd63; addr_d = 16'd64; #100;

 clock = 1'd1; addr_a   = 16'd65; addr_b = 16'd66; addr_c = 16'd67; addr_d = 16'd68; #100;
 clock = 1'd0; addr_a   = 16'd65; addr_b = 16'd66; addr_c = 16'd67; addr_d = 16'd68; #100;

 clock = 1'd1; addr_a   = 16'd69; addr_b = 16'd70; addr_c = 16'd71; addr_d = 16'd72; #100;
 clock = 1'd0; addr_a   = 16'd69; addr_b = 16'd70; addr_c = 16'd71; addr_d = 16'd72; #100;

 clock = 1'd1; addr_a   = 16'd73; addr_b = 16'd74; addr_c = 16'd75; addr_d = 16'd76; #100;
 clock = 1'd0; addr_a   = 16'd73; addr_b = 16'd74; addr_c = 16'd75; addr_d = 16'd76; #100;
```

```
  clock = 1'd1; addr_a   = 16'd77; addr_b = 16'd78; addr_c = 16'd79; addr_d = 16'd80; #100;
  clock = 1'd0; addr_a   = 16'd77; addr_b = 16'd78; addr_c = 16'd79; addr_d = 16'd80; #100;

  clock = 1'd1; addr_a   = 16'd81; addr_b = 16'd82; addr_c = 16'd83; addr_d = 16'd84; #100;
  clock = 1'd0; addr_a   = 16'd81; addr_b = 16'd82; addr_c = 16'd83; addr_d = 16'd84; #100;

  clock = 1'd1; addr_a   = 16'd85; addr_b = 16'd86; addr_c = 16'd87; addr_d = 16'd88; #100;
  clock = 1'd0; addr_a   = 16'd85; addr_b = 16'd86; addr_c = 16'd87; addr_d = 16'd88; #100;

  clock = 1'd1; addr_a   = 16'd89; addr_b = 16'd90; addr_c = 16'd91; addr_d = 16'd92; #100;
  clock = 1'd0; addr_a   = 16'd89; addr_b = 16'd90; addr_c = 16'd91; addr_d = 16'd92; #100;


  end



endmodule
```

3. Data_memory_tb

```
module datamemory_tb();

        reg clock;
        reg data_write_en_a;
        reg [15:0] data_addr_a;
        reg [15:0] datain_a;

  reg data_write_en_b;
        reg [15:0] data_addr_b;
        reg [15:0] datain_b;

  reg data_write_en_c;
        reg [15:0] data_addr_c;
        reg [15:0] datain_c;

  reg data_write_en_d;
        reg [15:0] data_addr_d;
        reg [15:0] datain_d;

        wire [15:0] dataout_a;
        wire [15:0] dataout_b;
        wire [15:0] dataout_c;
        wire [15:0] dataout_d;

  datamemory                                                                    dut
(.clock(clock),.datain_a(datain_a),.data_write_en_a(data_write_en_a),.data_addr_a(data_addr_
a),.datain_b(datain_b),.data_write_en_b(data_write_en_b),.data_addr_b(data_addr_b),.datain_
c(datain_c),.data_write_en_c(data_write_en_c),.data_addr_c(data_addr_c),.datain_d(datain_d),
.data_write_en_d(data_write_en_d),.data_addr_d(data_addr_d),.dataout_a(dataout_a),.dataout
_b(dataout_b),.dataout_c(dataout_c),.dataout_d(dataout_d));

  initial begin
  #50;

  clock = 1'd0; datain_a=16'd255 ;data_write_en_a =1'd1 ;data_addr_a =16'd35
;datain_b=16'd0;data_write_en_b = 1'd0;data_addr_b =16'd0 ;datain_c= 16'd0;data_write_en_c
=1'd0 ;data_addr_c =16'd0 ;datain_d = 16'd0;data_write_en_d = 1'd0;data_addr_d =16'd0
;#100;
  clock = 1'd0; datain_a=16'd35 ;data_write_en_a =1'd0 ;data_addr_a =16'd10
;datain_b=16'd0;data_write_en_b = 1'd0;data_addr_b =16'd0 ;datain_c= 16'd0;data_write_en_c
=1'd0 ;data_addr_c =16'd0 ;datain_d = 16'd0;data_write_en_d = 1'd0;data_addr_d =16'd0
;#100;

  clock = 1'd0; datain_a=16'd25 ;data_write_en_a =1'd0 ;data_addr_a =16'd35
;datain_b=16'd17;data_write_en_b = 1'd1;data_addr_b =16'd28 ;datain_c=
16'd0;data_write_en_c =1'd0 ;data_addr_c =16'd0 ;datain_d = 16'd0;data_write_en_d =
1'd0;data_addr_d =16'd0 ;#100;
  clock = 1'd0; datain_a=16'd315 ;data_write_en_a =1'd0 ;data_addr_a =16'd10
;datain_b=16'd24;data_write_en_b = 1'd0;data_addr_b =16'd14 ;datain_c=
16'd0;data_write_en_c =1'd0 ;data_addr_c =16'd0 ;datain_d = 16'd0;data_write_en_d =
1'd0;data_addr_d =16'd0 ;#100;

  clock = 1'd0; datain_a=16'd0 ;data_write_en_a =1'd0 ;data_addr_a =16'd35
;datain_b=16'd0;data_write_en_b = 1'd0;data_addr_b =16'd0 ;datain_c=
16'd112;data_write_en_c =1'd1 ;data_addr_c =16'd56 ;datain_d = 16'd0;data_write_en_d =
1'd0;data_addr_d =16'd0 ;#100;
  clock = 1'd0; datain_a=16'd0 ;data_write_en_a =1'd0 ;data_addr_a =16'd10
;datain_b=16'd0;data_write_en_b = 1'd0;data_addr_b =16'd0 ;datain_c=
16'd45;data_write_en_c =1'd0 ;data_addr_c =16'd14 ;datain_d = 16'd0;data_write_en_d =
1'd0;data_addr_d =16'd0 ;#100;

 end


always begin
   #50;
         clock = ~clock;
  end
endmodule
```

4. Processor_tb

```
module processor_a_tb();
     reg clock;
                  reg [ 1: 0] status;
                  reg [ 15: 0] dm_out;
                  reg [ 15: 0] im_out;
```

```
            //output [ 15: 0] bus_out;
            wire dm_en;
            wire [ 15: 0] pc_out;
            wire [ 15: 0] ar_out;
            wire  end_process;
            wire [ 15: 0] r2_out;
            //wire [ 15: 0] ir_out;


            processor_a dut (.clock(clock), .status(status), .dm_out(dm_out),
.im_out(im_out),.dm_en(dm_en), .pc_out(pc_out), .ar_out(ar_out), .end_process(end_process),
.r2_out(r2_out));


 initial begin
 #500;
 clock = 1'd0; status =2'd0 ; dm_out = 16'd0; im_out = 16'd4132; #500;
 clock = 1'd1; status =2'd0 ; dm_out = 16'd0; im_out = 16'd4132; #500;
 clock = 1'd0; status =2'd1 ; dm_out = 16'd0; im_out = 16'd4132; #500;
 clock = 1'd1; status =2'd1 ; dm_out = 16'd0; im_out = 16'd4132; #500;
 clock = 1'd0; status =2'd1 ; dm_out = 16'd0; im_out = 16'd4132; #500;
 clock = 1'd1; status =2'd1 ; dm_out = 16'd0; im_out = 16'd4132; #500;
 clock = 1'd0; status =2'd1 ; dm_out = 16'd0; im_out = 16'd4132; #500;
 clock = 1'd1; status =2'd1 ; dm_out = 16'd0; im_out = 16'd4132; #500;
 clock = 1'd0; status =2'd1 ; dm_out = 16'd0; im_out = 16'd4132; #500;
 clock = 1'd1; status =2'd1 ; dm_out = 16'd0; im_out = 16'd4132; #500;
 clock = 1'd0; status =2'd1 ; dm_out = 16'd0; im_out = 16'd4132; #500;
 clock = 1'd1; status =2'd1 ; dm_out = 16'd0; im_out = 16'd4132; #500;
 clock = 1'd0; status =2'd1 ; dm_out = 16'd0; im_out = 16'd4132; #500;
 clock = 1'd1; status =2'd1 ; dm_out = 16'd0; im_out = 16'd4132; #500;

 clock = 1'd0; status =2'd1 ; dm_out = 16'd0; im_out = 16'd4103; #500;
 clock = 1'd1; status =2'd1 ; dm_out = 16'd0; im_out = 16'd4103; #500;
 clock = 1'd0; status =2'd1 ; dm_out = 16'd0; im_out = 16'd4103; #500;
 clock = 1'd1; status =2'd1 ; dm_out = 16'd0; im_out = 16'd4103; #500;
 clock = 1'd0; status =2'd1 ; dm_out = 16'd0; im_out = 16'd4103; #500;
 clock = 1'd1; status =2'd1 ; dm_out = 16'd0; im_out = 16'd4103; #500;
 clock = 1'd0; status =2'd1 ; dm_out = 16'd0; im_out = 16'd4103; #500;
 clock = 1'd1; status =2'd1 ; dm_out = 16'd0; im_out = 16'd4103; #500;
 clock = 1'd0; status =2'd1 ; dm_out = 16'd0; im_out = 16'd4103; #500;
 clock = 1'd1; status =2'd1 ; dm_out = 16'd0; im_out = 16'd4103; #500;
 clock = 1'd0; status =2'd1 ; dm_out = 16'd0; im_out = 16'd4103; #500;
 clock = 1'd1; status =2'd1 ; dm_out = 16'd0; im_out = 16'd4103; #500;

 clock = 1'd0; status =2'd1 ; dm_out = 16'd0; im_out = 16'd4101; #500;
 clock = 1'd1; status =2'd1 ; dm_out = 16'd0; im_out = 16'd4101; #500;
 clock = 1'd0; status =2'd1 ; dm_out = 16'd0; im_out = 16'd4101; #500;
 clock = 1'd1; status =2'd1 ; dm_out = 16'd0; im_out = 16'd4101; #500;
 clock = 1'd0; status =2'd1 ; dm_out = 16'd0; im_out = 16'd4101; #500;
 clock = 1'd1; status =2'd1 ; dm_out = 16'd0; im_out = 16'd4101; #500;
 clock = 1'd0; status =2'd1 ; dm_out = 16'd0; im_out = 16'd4101; #500;
```

```
  clock = 1'd1; status =2'd1 ; dm_out = 16'd0; im_out = 16'd4101; #500;
  clock = 1'd0; status =2'd1 ; dm_out = 16'd0; im_out = 16'd4101; #500;
  clock = 1'd1; status =2'd1 ; dm_out = 16'd0; im_out = 16'd4101; #500;
  clock = 1'd0; status =2'd1 ; dm_out = 16'd0; im_out = 16'd4101; #500;
  clock = 1'd1; status =2'd1 ; dm_out = 16'd0; im_out = 16'd4101; #500;
  clock = 1'd0; status =2'd1 ; dm_out = 16'd0; im_out = 16'd4101; #500;
  clock = 1'd1; status =2'd1 ; dm_out = 16'd0; im_out = 16'd4101; #500;

  end


//  always begin
//    #50;
//          clock = ~clock;
//  end

endmodule
```

5. control_tb

```
module control_tb();
          reg clock;
          reg [15:0] z;
          reg [ 1: 0] status;
          reg [ 15: 0] instruction;
          wire [ 2: 0] alu_op;
          wire [ 15: 0] write_en;
          wire [ 2: 0] rst;
          wire [ 15: 0] inc_en;
          wire [ 3: 0] read_en;
          wire mem_wr;
          wire end_process;

  control_a dut (.clock(clock),. z(z), .status(status), .instruction(instruction), .alu_op(alu_op),
.write_en(write_en),. rst(rst), .inc_en(inc_en), .read_en(read_en), .mem_wr(mem_wr),
.end_process(end_process));

  initial begin
  #50;
  clock = 1'd0; z=16'd0 ;status = 2'b01;instruction  = 16'd4101;#100;
  clock = 1'd1; z=16'd0 ;status = 2'b01;instruction  = 16'd4101;#100;
  clock = 1'd0; z=16'd0 ;status = 2'b01;instruction  = 16'd4101;#100;
  clock = 1'd1; z=16'd0 ;status = 2'b01;instruction  = 16'd4101;#100;
  clock = 1'd0; z=16'd0 ;status = 2'b01;instruction  = 16'd4101;#100;
  clock = 1'd1; z=16'd0 ;status = 2'b01;instruction  = 16'd4101;#100;
  clock = 1'd0; z=16'd0 ;status = 2'b01;instruction  = 16'd4101;#100;
  clock = 1'd1; z=16'd0 ;status = 2'b01;instruction  = 16'd4101;#100;
  clock = 1'd0; z=16'd0 ;status = 2'b01;instruction  = 16'd4101;#100;
  clock = 1'd1; z=16'd0 ;status = 2'b01;instruction  = 16'd4101;#100;
  clock = 1'd0; z=16'd0 ;status = 2'b01;instruction  = 16'd4101;#100;
  clock = 1'd1; z=16'd0 ;status = 2'b01;instruction  = 16'd4101;#100;
  clock = 1'd0; z=16'd0 ;status = 2'b01;instruction  = 16'd4101;#100;
  clock = 1'd1; z=16'd0 ;status = 2'b01;instruction  = 16'd4101;#100;


  clock = 1'd0; z=16'd0 ;status = 2'b01;instruction  = 16'd4124;#100;
  clock = 1'd1; z=16'd0 ;status = 2'b01;instruction  = 16'd4124;#100;
  clock = 1'd0; z=16'd0 ;status = 2'b01;instruction  = 16'd4124;#100;
  clock = 1'd1; z=16'd0 ;status = 2'b01;instruction  = 16'd4124;#100;
  clock = 1'd0; z=16'd0 ;status = 2'b01;instruction  = 16'd4124;#100;
  clock = 1'd1; z=16'd0 ;status = 2'b01;instruction  = 16'd4124;#100;
  clock = 1'd0; z=16'd0 ;status = 2'b01;instruction  = 16'd4124;#100;
  clock = 1'd1; z=16'd0 ;status = 2'b01;instruction  = 16'd4124;#100;
  clock = 1'd0; z=16'd0 ;status = 2'b01;instruction  = 16'd4124;#100;
  clock = 1'd1; z=16'd0 ;status = 2'b01;instruction  = 16'd4124;#100;

  clock = 1'd0; z=16'd0 ;status = 2'b01;instruction  = 16'd4141;#100;
  clock = 1'd1; z=16'd0 ;status = 2'b01;instruction  = 16'd4141;#100;
  clock = 1'd0; z=16'd0 ;status = 2'b01;instruction  = 16'd4141;#100;
  clock = 1'd1; z=16'd0 ;status = 2'b01;instruction  = 16'd4141;#100;
  clock = 1'd0; z=16'd0 ;status = 2'b01;instruction  = 16'd4141;#100;
  clock = 1'd1; z=16'd0 ;status = 2'b01;instruction  = 16'd4141;#100;
  clock = 1'd0; z=16'd0 ;status = 2'b01;instruction  = 16'd4141;#100;
  clock = 1'd1; z=16'd0 ;status = 2'b01;instruction  = 16'd4141;#100;
  clock = 1'd0; z=16'd0 ;status = 2'b01;instruction  = 16'd4141;#100;
  clock = 1'd1; z=16'd0 ;status = 2'b01;instruction  = 16'd4141;#100;


  clock = 1'd0; z=16'd0 ;status = 2'b01;instruction  = 16'd61476;#100;
  clock = 1'd1; z=16'd0 ;status = 2'b01;instruction  = 16'd61476;#100;
  clock = 1'd0; z=16'd0 ;status = 2'b01;instruction  = 16'd61476;#100;
  clock = 1'd1; z=16'd0 ;status = 2'b01;instruction  = 16'd61476;#100;
  clock = 1'd0; z=16'd0 ;status = 2'b01;instruction  = 16'd61476;#100;
  clock = 1'd1; z=16'd0 ;status = 2'b01;instruction  = 16'd61476;#100;
  clock = 1'd0; z=16'd0 ;status = 2'b01;instruction  = 16'd61476;#100;
  clock = 1'd1; z=16'd0 ;status = 2'b01;instruction  = 16'd61476;#100;
  clock = 1'd0; z=16'd0 ;status = 2'b01;instruction  = 16'd61476;#100;
  clock = 1'd1; z=16'd0 ;status = 2'b01;instruction  = 16'd61476;#100;
  //clock = 1'd0; status = 2'b01;instruction  = 16'd39;#100;
  //clock = 1'd1; status = 2'b01;instruction  = 16'd39;#100;
  //clock = 1'd0; status = 2'b01;instruction  = 16'd39;#100;
  //clock = 1'd1; status = 2'b01;instruction  = 16'd39;#100;



  end

  //always begin
    //#50;
            //clock = ~clock;
  //end
```

```
endmodule
```

6. Alu_tb

```
module alu_tb();
  reg clock;
  reg [2:0] alu_op;
  reg [15:0] in1;
  reg [15:0] in2;
  wire [15:0] alu_out;
  wire [15:0] z;

  alu dut (.clock(clock), .in1(in1), .in2(in2), .alu_op(alu_op), .alu_out(alu_out), .z(z));

  initial begin
  #50;
  clock = 1'd0; alu_op   = 3'd2; in1 = 16'd3; in2 = 16'd6; #100;
  clock = 1'd0; alu_op   = 3'd3; in1 = 16'd7; in2 = 16'd10; #100;
  clock = 1'd0; alu_op   = 3'd1; in1 = 16'd4; in2 = 16'd6; #100;
  clock = 1'd0; alu_op   = 3'd2; in1 = 16'd102; in2 = 16'd24; #100;
  clock = 1'd0; alu_op   = 3'd3; in1 = 16'd253; in2 = 16'd254; #100;
  clock = 1'd0; alu_op   = 3'd1; in1 = 16'd255; in2 = 16'd255; #100;

  end

  always begin
    #50;
          clock = ~clock;
  end

endmodule
```

7. Address_select_tb

```
module address_tb;

    // Inputs
          reg clock;
    reg [15:0] in1;
    reg [15:0] in2;
          reg [15:0] in3;
    // Outputs
    wire [15:0] out_address_aa;
          wire [15:0] out_address_ab;
          wire [15:0] out_address_ad;

    // Instantiate the Unit Under Test (UUT)
    address_select uut (
                .clock(clock),
        .in1(in1),
        .in2(in2),
                        .in3(in3),
        .out_address_aa(out_address_aa),
                        .out_address_ab(out_address_ab),
                        .out_address_ad(out_address_ad)
    );

    initial begin
        // Apply Inputs
                        #50
                        clock = 1'd0;
                        in1 = 16'd1;
                        in2 = 16'd1;
                        in3 = 16'd3;
                        #100;

                        clock = 1'd1;
                        in1 = 16'd1;
                        in2 = 16'd1;
                        in3 = 16'd3;
                        #100;

                        clock = 1'd0;
                        in1 = 16'd2;
                        in2 = 16'd2;
                        in3 = 16'd2;#100;

                        clock = 1'd1;
                        in1 = 16'd2;
                        in2 = 16'd2;
                        in3 = 16'd2;#100;

                        clock = 1'd0;
                        in1 = 16'd3;
                        in2 = 16'd2;
                        in3 = 16'd3;#100;

                        clock = 1'd1;
                        in1 = 16'd3;
                        in2 = 16'd2;
                        in3 = 16'd3;#100;


    end

endmodule
```

8. bus_tb

```
module bus_tb();

        reg clock;
        reg [ 3: 0] read_en;
        reg [ 15: 0] aa;
        reg [ 15: 0] r2;
        reg [ 15: 0] ab;
        reg [ 15: 0] ir;
        reg [ 15: 0] am;
        reg [ 15: 0] an;
        reg [ 15: 0] bn;
        reg [ 15: 0] arp;
        reg [ 15: 0] acp;
        reg [ 15: 0] bcp;
        reg [ 15: 0] ac;
        reg [ 15: 0] ad;
        reg [ 15: 0] dm;
        reg [ 15: 0] im;
        wire [ 15: 0] busout;

        bus dut( .clock(clock),.read_en(read_en) , .aa(aa), .r2(r2), .ab(ab), .ir(ir), .am(am),
.an(an), .bn(bn), .arp(arp), .acp(acp), .bcp(bcp),.ac(ac), .ad(ad), .dm(dm), .im(im),
.busout(busout) );

        initial
        begin


        clock = 0;read_en = 0; aa = 0; r2 = 0; ab = 0; ir = 0; am = 0; an = 0; bn = 0; arp= 0; acp =
0; bcp = 0; ac = 0; ad = 0; dm = 0; im = 0; #100
        clock = 1;read_en = 0; aa = 33; r2 = 0; ab = 0; ir = 0; am = 0; an = 0; bn = 0; arp= 0; acp
= 0; bcp = 0; ac = 0; ad = 0; dm = 0; im = 0; #100
        clock = 0;read_en = 0; aa = 33; r2 = 0; ab = 0; ir = 0; am = 0; an = 0; bn = 0; arp= 0; acp
= 0; bcp = 0; ac = 0; ad = 0; dm = 0; im = 0; #100
        clock = 1;read_en = 1; aa = 33; r2 = 0; ab = 0; ir = 0; am = 0; an = 0; bn = 0; arp= 0; acp
= 0; bcp = 0; ac = 0; ad = 0; dm = 0; im = 0; #100
        clock = 0;read_en = 1; aa = 33; r2 = 0; ab = 0; ir = 0; am = 0; an = 0; bn = 0; arp= 0; acp
= 0; bcp = 0; ac = 0; ad = 0; dm = 0; im = 0; #100
        clock = 1;read_en = 0; aa = 33; r2 = 0; ab = 0; ir = 0; am = 0; an = 0; bn = 0; arp= 0; acp
= 0; bcp = 0; ac = 0; ad = 0; dm = 0; im = 0; #100
        clock = 0;read_en = 0; aa = 33; r2 = 0; ab = 0; ir = 0; am = 0; an = 0; bn = 0; arp= 0; acp
= 0; bcp = 0; ac = 0; ad = 0; dm = 0; im = 0; #100
        clock = 1;read_en = 0; aa = 0; r2 = 0; ab = 0; ir = 0; am = 0; an = 0; bn = 0; arp= 0; acp =
0; bcp = 0; ac = 0; ad = 0; dm = 0; im = 0; #100

        clock = 0;read_en = 0; aa = 0; r2 = 0; ab = 0; ir = 0; am = 0; an = 0; bn = 0; arp= 0; acp =
0; bcp = 0; ac = 0; ad = 0; dm = 0; im = 0; #100
        clock = 1;read_en = 0; aa = 0; r2 = 44; ab = 0; ir = 0; am = 0; an = 0; bn = 0; arp= 0; acp
= 0; bcp = 0; ac = 0; ad = 0; dm = 0; im = 0; #100
```

```
        clock = 0;read_en = 0; aa = 0; r2 = 44; ab = 0; ir = 0; am = 0; an = 0; bn = 0; arp= 0; acp
= 0; bcp = 0; ac = 0; ad = 0; dm = 0; im = 0; #100
        clock = 1;read_en = 2; aa = 0; r2 = 44; ab = 0; ir = 0; am = 0; an = 0; bn = 0; arp= 0; acp
= 0; bcp = 0; ac = 0; ad = 0; dm = 0; im = 0; #100
        clock = 0;read_en = 2; aa = 0; r2 = 44; ab = 0; ir = 0; am = 0; an = 0; bn = 0; arp= 0; acp
= 0; bcp = 0; ac = 0; ad = 0; dm = 0; im = 0; #100
        clock = 1;read_en = 0; aa = 0; r2 = 44; ab = 0; ir = 0; am = 0; an = 0; bn = 0; arp= 0; acp
= 0; bcp = 0; ac = 0; ad = 0; dm = 0; im = 0; #100
        clock = 0;read_en = 0; aa = 0; r2 = 44; ab = 0; ir = 0; am = 0; an = 0; bn = 0; arp= 0; acp
= 0; bcp = 0; ac = 0; ad = 0; dm = 0; im = 0; #100
        clock = 1;read_en = 0; aa = 0; r2 = 0; ab = 0; ir = 0; am = 0; an = 0; bn = 0; arp= 0; acp =
0; bcp = 0; ac = 0; ad = 0; dm = 0; im = 0; #100

        clock = 0;read_en = 0; aa = 0; r2 = 0; ab = 0; ir = 0; am = 0; an = 0; bn = 0; arp= 0; acp =
0; bcp = 0; ac = 0; ad = 0; dm = 0; im = 0; #100
        clock = 1;read_en = 0; aa = 0; r2 = 0; ab = 55; ir = 0; am = 0; an = 0; bn = 0; arp= 0; acp
= 0; bcp = 0; ac = 0; ad = 0; dm = 0; im = 0; #100
        clock = 0;read_en = 0; aa = 0; r2 = 0; ab = 55; ir = 0; am = 0; an = 0; bn = 0; arp= 0; acp
= 0; bcp = 0; ac = 0; ad = 0; dm = 0; im = 0; #100
        clock = 1;read_en = 3; aa = 0; r2 = 0; ab = 55; ir = 0; am = 0; an = 0; bn = 0; arp= 0; acp
= 0; bcp = 0; ac = 0; ad = 0; dm = 0; im = 0; #100
        clock = 0;read_en = 3; aa = 0; r2 = 0; ab = 55; ir = 0; am = 0; an = 0; bn = 0; arp= 0; acp
= 0; bcp = 0; ac = 0; ad = 0; dm = 0; im = 0; #100
        clock = 1;read_en = 0; aa = 0; r2 = 0; ab = 55; ir = 0; am = 0; an = 0; bn = 0; arp= 0; acp
= 0; bcp = 0; ac = 0; ad = 0; dm = 0; im = 0; #100
        clock = 0;read_en = 0; aa = 0; r2 = 0; ab = 55; ir = 0; am = 0; an = 0; bn = 0; arp= 0; acp
= 0; bcp = 0; ac = 0; ad = 0; dm = 0; im = 0; #100
        clock = 1;read_en = 0; aa = 0; r2 = 0; ab = 0; ir = 0; am = 0; an = 0; bn = 0; arp= 0; acp =
0; bcp = 0; ac = 0; ad = 0; dm = 0; im = 0;
        end

endmodule
```

9. Regr_tb

```
module regr_tb();
	reg clock;
	reg write_en;
	reg [15:0] datain;
	wire[15:0] dataout;

	regr dut(
				.clock(clock),
				.write_en(write_en),
				.datain(datain),
				.dataout(dataout)
				);

	initial begin
		#50
		clock = 0; write_en = 0; datain = 16'd0;#100;
		clock = 1; write_en = 0; datain = 16'd0;#100;
		clock = 0; write_en = 0; datain = 16'd25;#100;
		clock = 1; write_en = 1; datain = 16'd25;#100;
		clock = 0; write_en = 1; datain = 16'd47;#100;
		clock = 1; write_en = 1; datain = 16'd47;#100;
		clock = 0; write_en = 1; datain = 16'd48;#100;
		clock = 1; write_en = 1; datain = 16'd49;#100;


	end


endmodule
```

10. Regrinc_tb

```
module regrinc_tb();
	reg clock, write_en, inc_en;
	reg [15:0] datain;
	wire[15:0] dataout;

	regrinc dut(
				.clock(clock),
				.write_en(write_en),
				.inc_en(inc_en),
				.datain(datain),
				.dataout(dataout)
				);
	initial begin
	#50
	clock = 0; write_en = 0; inc_en = 0; datain = 16'd60; #100;
	clock = 1; write_en = 0; inc_en = 0; datain = 16'd60; #100;
	clock = 0; write_en = 0; inc_en = 0; datain = 16'd60; #100;
	clock = 1; write_en = 1; inc_en = 0; datain = 16'd60; #100;
	clock = 0; write_en = 1; inc_en = 0; datain = 16'd8; #100;
	clock = 1; write_en = 1; inc_en = 0; datain = 16'd8; #100;
	clock = 0; write_en = 0; inc_en = 1; datain = 16'd8;#100 ;
	clock = 1; write_en = 0; inc_en = 1; datain = 16'd8;#100 ;

	end


endmodule
```

11. Regr2_tb

```
module regr2_tb();
	reg clock;
	reg write_en;
	reg dm_mem_wr_en;
	reg datain;
	wire dataout;
	wire dm_mem_out;


	regr2 dut(
					.clock(clock),
					.write_en(write_en),
					.dm_mem_wr_en(dm_mem_wr_en),
					.datain(datain),
					.dataout(dataout),
					.dm_mem_out(dm_mem_out)
					);

	initial begin
		clock = 0; write_en = 0; dm_mem_wr_en = 0 ; datain = 0 ;#100
		clock = 1; write_en = 0; dm_mem_wr_en = 0 ; datain = 0 ;#100
		clock = 0; write_en = 0; dm_mem_wr_en = 0 ; datain = 0 ;#100
		clock = 1; write_en = 1; dm_mem_wr_en = 0 ; datain = 0 ;#100
		clock = 0; write_en = 1; dm_mem_wr_en = 0 ; datain = 0 ;#100
		clock = 1; write_en = 1; dm_mem_wr_en = 1 ; datain = 0 ;#100
		clock = 0; write_en = 1; dm_mem_wr_en = 1 ; datain = 0 ;#100
		clock = 1; write_en = 1; dm_mem_wr_en = 1 ; datain = 300 ;#100
		clock = 0; write_en = 1; dm_mem_wr_en = 1 ; datain = 300 ;

	end

endmodule
```

12. Regran_tb

```
module regran_tb();
        reg clock, write_en, inc_en,rst;
        reg [15:0] datain;
        wire[15:0] dataout;

        regran dut(
                        .clock(clock),
                        .write_en(write_en),
                        .rst(rst),
                        .inc_en(inc_en),
                        .datain(datain),
                        .dataout(dataout)
                        );

        initial begin
        #50
        clock = 0; write_en = 0; inc_en = 0;rst = 0;datain = 16'd60; #100;
        clock = 1; write_en = 0; inc_en = 0;rst = 0; datain = 16'd60; #100;
        clock = 0; write_en = 0; inc_en = 0;rst = 0; datain = 16'd60; #100;
        clock = 1; write_en = 1; inc_en = 0;rst = 0; datain = 16'd60; #100;
        clock = 0; write_en = 1; inc_en = 0;rst = 0; datain = 16'd8; #100;
        clock = 1; write_en = 1; inc_en = 0;rst = 0; datain = 16'd8; #100;
        clock = 0; write_en = 0; inc_en = 1;rst = 0; datain = 16'd8;#100 ;
        clock = 1; write_en = 0; inc_en = 1;rst = 0; datain = 16'd8;#100 ;
        clock = 0; write_en = 0; inc_en = 0;rst = 1; datain = 16'd8;#100 ;
        clock = 1; write_en = 0; inc_en = 0;rst = 1; datain = 16'd8;#100 ;


        end



endmodule
```