

|  |  |
| --- | --- |
| **Name, Surname** | Doğukan Fikri Arat |
| **Student ID** | 2079648 |
| **Name, Surname** | Saad Yousaf |
| **Student ID** | 2246536 |

**MIDDLE EAST TECHNICAL UNIVERSITY NORTHERN CYPRUS CAMPUS**

**ELECTRICAL AND ELECTRONICS ENGINEERING PROGRAM**

**EEE 446 Computer Architecture II**

**8-Bit Multi-Cycle Processor Design and Implementation**

Table of Contents

[Objectives 3](#_Toc42430418)

[Multi-8 ISA Specifications and Design 4](#_Toc42430419)

[Instruction Types 4](#_Toc42430420)

[List of instructions 5](file:///C:\Users\Saad%20Yousaf\Documents\GitHub\documents\module-6\Report.docx#_Toc42430421)

[Funtira CPU Design and Validation 0](#_Toc42430422)

[ISA Validation and Performance 0](#_Toc42430423)

[J-type Instructions 0](#_Toc42430424)

[I-type Instructions 1](#_Toc42430425)

[R-type Instructions 3](#_Toc42430426)

[Assembler Design 0](#_Toc42430427)

[CPU Benchmark Definitions and Testing 1](#_Toc42430428)

[Power Calculation 2](#_Toc42430429)

[Conclusion 0](#_Toc42430430)

[Appendix A: Assembler Source Code in C# 1](#_Toc42430431)

[Appendix B: Source Code of Control Unit 17](#_Toc42430432)

# Objectives

In this project, we were responsible for designing and implementing an 8-bit CPU, codename Funtira. We developed and tailored the Multi-8 ISA for Funtira by considering our performance and benchmarking requirements. To convert the assembly code written in Multi-8 assembly language to machine code, we developed the Codalex assembler. In this design report we will give a complete description of our ISA. The detailed high-level and low-level schematics of Funtira will be included in the report to show the CPU design implementation. Test benchmarks compiled using the Codalex assembler will be provided in this report along with their results to demonstrate the functionality and performance of Funtira.

# Multi-8 ISA Specifications and Design

## Instruction Types

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| R-TYPE  15  9  11 | | | | |
| 4 | 3 | 3 | 3  6  5 | 3  0 |
| OPCODE (Op)  12 | DESTINATION (Rd) | SOURCE\_1 (Rs)  8 | SOURCE\_2 (Rt) | OAP  2  3 |
|  |  |  |  |  |
| I-TYPE  8  12  0  15  9  11 | | | | |
| 3 | 3 | 3 | 6 | |
| OPCODE (Op) | DESTINATION (Rd) | SOURCE\_1 (Rs)  6d | CONSTANT (C)  5 | |
|  |  |  |  |  |
| J-TYPE | | | | |
| 4  15 | 12  12  11  0 | | | |
| OPCODE (Op) | CONSTANT (C) | | | |

Table 1. Multi-8 ISA instruction types

* R-type

All R-type instructions have a 4-bit opcode, which is 0000 for all R-type instructions. The instruction itself is determined by the 3 OAP bits, which means we have 8 R-type instructions. There are two source registers Rs and Rt, 3-bit address each, and one 3-bit destination register Rd. R-type instructions are arithmetic-logic operations and some of them have immediate versions as well which are defined under I-type instructions.

* I-type

All I-type instructions have a 4-bit OPCODE which is used to identify the instruction as well in the decode stage. The is one destination (Rd) and one source (Rs) register, both having 3-bit addresses. The immediate is a 6-bit constant (C) which allows values -32<C<31 to be used as an immediate. This constant is sign-extended to 8-bit since all operations are using 8-bits.

* J-type

Multi-8 ISA supports one J-type instruction, which is Jump instruction. It has a 4-bit OPCODE and a 12-bit constant which is the destination address for the jump target. The jump instruction supports direct memory addressing since the 12-bit constant allows the jump instruction to span the whole 2kB instruction memory.

The special DMADDR instruction allows to access Data memory in quarters of 256 memory locations due to the limited 8-bit size of the user registers which cannot address the full data memory.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Instruction | Type | Operands | Description | Operation | Opcode | OAP |
| ADD | R | Rd, Rs, Rt, Cin | Add two registers and a carry (Cin) | Rd <= Rs + Rt + Cin  PC <= PC + 1 | 0 | 0 |
| ADDI | I | Rd, Rs, C | Add a register and a constant ( C ) | Rd <= Rs + C,  PC <= PC + 1 | 1 | X |
| SUB | R | Rd, Rs, Rt, Cin | Subtract two registers and a carry (Cin) | Rd <= Rs - Rt – Cin  PC <= PC + 1 | 0 | 1 |
| AND | R | Rd, Rs, Rt | Logical AND two registers | Rd <= Rs ^ Rt  PC <= PC + 1 | 0 | 4 |
| ANDI | I | Rd, Rs, C | Logical AND two registers and a constant ( C ) | Rd <= Rs ^ C  PC <= PC + 1 | 2 | X |
| OR | R | Rd, Rs, Rt | Logical OR two registers | Rd <= Rs v Rt  PC <= PC + 1 | 0 | 3 |
| DMADDR | J | C | Set most significant 2-bits of Data Memory Address to least significant 2-bits of constant ( C ) | DM\_address[8:9] <= C [1:0]  PC <= PC + 1 | 3 | X |
| XOR | R | Rd, Rs, Rt | Logical XOR two registers | Rd <= Rs ⊕ Rt  PC <= PC + 1 | 0 | 6 |
| SLT | R | Rd, Rs, Rt | If Rs<Rt, Rd=1, else Rd=0. | Rs-Rt, Rd <= N (zero flag)  PC <= PC + 1 | 0 | 5 |
| MUL | R | Rd, Rs, Rt | Multiply two registers | [Rd,Rd+1] <= Rs \* Rt  PC <= PC + 1 | 0 | 2 |
| DIV | R | Rd, Rs, Rt | Divide two registers | Rs / Rt, [Rd,Rd+1] <= [Q, A]  PC <= PC + 1 | 0 | 7 |
| SLL | I | Rd, Rs, C | Logical shift left Rs and save into Rd with constant amount ( C ) | Rd <= Rs << C  PC <= PC + 1 | 4 | X |
| SRL | I | Rd, Rs, C | Logical shift right Rs and save into Rd with constant amount ( C ) | Rd <= Rs >> C  PC <= PC + 1 | 5 | X |
| SRA | I | Rd, Rs, C | Arithmetic shift right Rs and save into Rd with constant amount ( C ) | Rd <= Rs >> C  PC <= PC + 1 | 6 | X |
| LW | I | Rd, Rs, C | Load 16-bit word from data memory | [Rd,Rd+1] <= MEM [Rs + C]  PC <= PC + 1 | 7 | X |
| LWU | I | Rd, Rs, C | Load upper byte from data memory location | [Rd, -] <= MEM [Rs + C]  PC <= PC + 1 | 8 | X |

## List of instructions

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Instruction | Type | Operands | Description | Operation | Opcode | OAP |
| LWL | I | Rd, Rs, C | Load lower byte from data memory location | [-, Rd] <= MEM [Rs + C]  PC <= PC + 1 | 9 | X |
| SW | I | Rd, Rs, C | Store 16-bit word to data memory | MEM [Rs + C] <= [Rd,Rd+1]  PC <= PC + 1 | 10 | X |
| SWU | I | Rd, Rs, C | Store 8-bit data to upper byte of data memory location | MEM [Rs + C] <= [Rd, -]  PC <= PC + 1 | 11 | X |
| SWL | I | Rd, Rs, C | Store 8-bit data to lower byte of data memory location | MEM [Rs + C] <= [-,Rd]  PC <= PC + 1 | 12 | X |
| BREQ | I | Rd, Rs, D | Branch if Rd equals Rs | if Rs-Rd=0, i.e. Z=1  PC <= PC + D | 13 | X |
| BRNE | I | Rd, Rs, D | Branch if Rd not equal to Rs | if Rs-Rd!=0, i.e. Z=0  PC <= PC + D | 14 | X |
| JUMP | J | D | Jump to address location (C) | PC <= PC+C | 15 | X |
| \* D is the branch Destination Address, calculated by current memory address minus destination address minus 1 | | | | | | |

Table 2. Multi-8 ISA Instructions list

# Funtira CPU Design and Validation

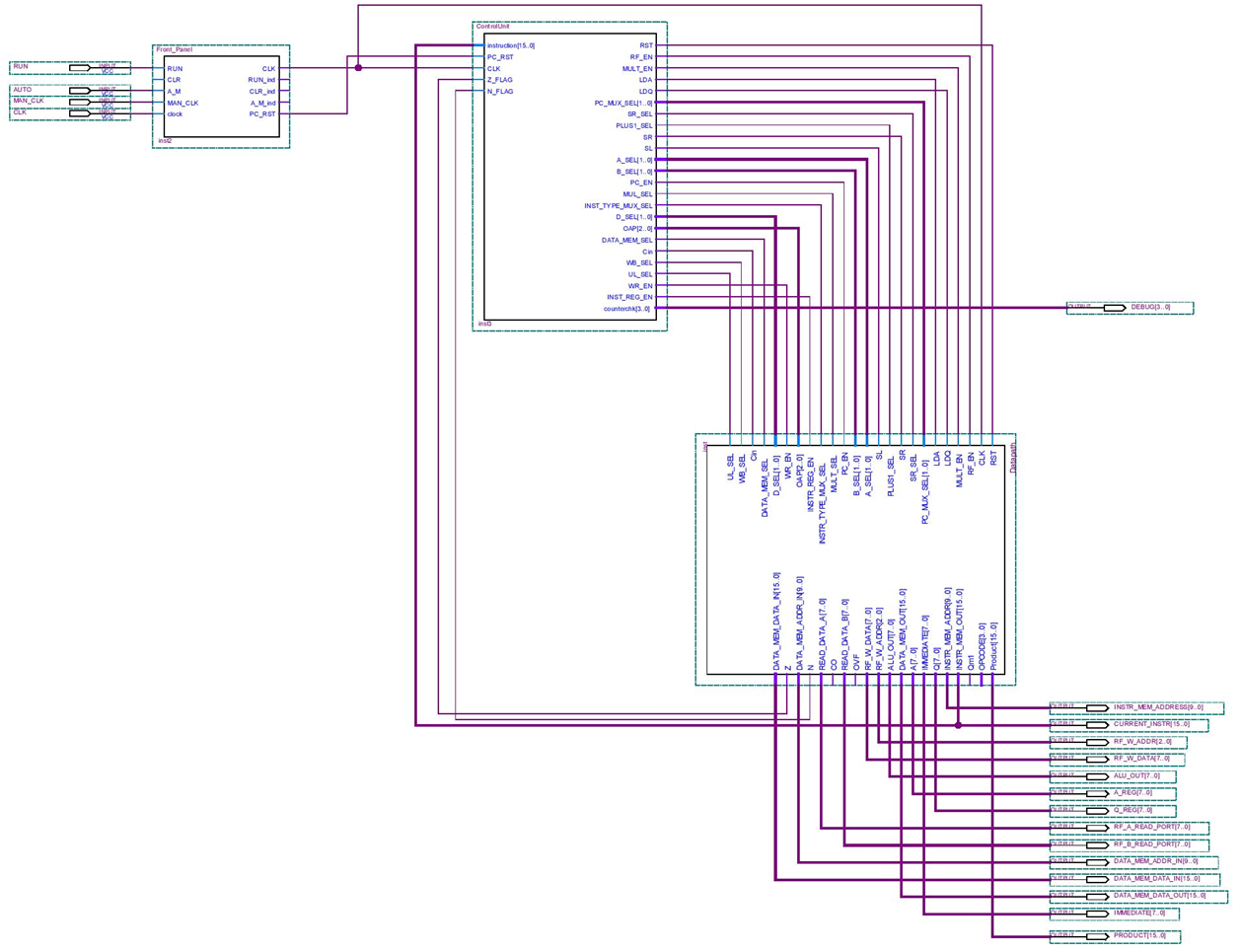
The Funtira CPU has been designed to run on the Multi-8 ISA. We have implemented a split data and instruction memory, both having 1024x16-bit words which totals to a size of 2kB. In our ISA we have added special instructions such as Load Word Lower (LWL), Load Word Upper (LWU), Store Word Lower (LWL) and Store Word Upper (LWU) to access the upper and lower byte of the Data memory locations. The DMADDR instruction has been added to access the entire range of the Data memory. Eight 8-bit registers R0 to R7 are available to the user to implement their code on the Funtira CPU. The CPU has a front panel which allows the user to start/stop the execution of instructions, select auto/manual mode clock and perform reset and clear operations. The front panel also has status indicators for these operations. In this section of the report, the high-level and low-level schematics of Funtira will be shown along with the functionality of the CPU using the front panel. The Funtira CPU can run at a maximum frequency of 50 MHz.

Figure 1. Complete CPU high-level design with front panel

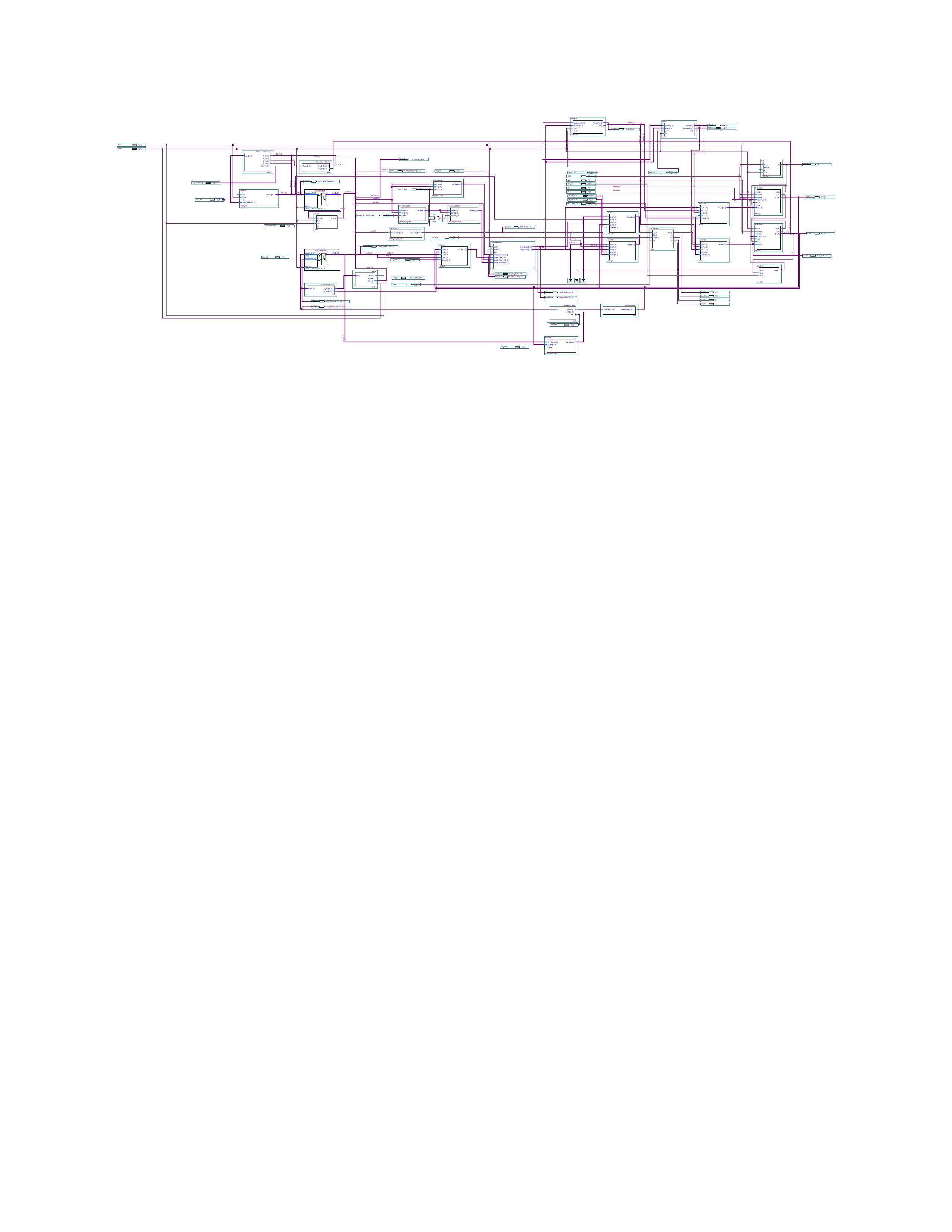


Figure 2. Datapath schematic

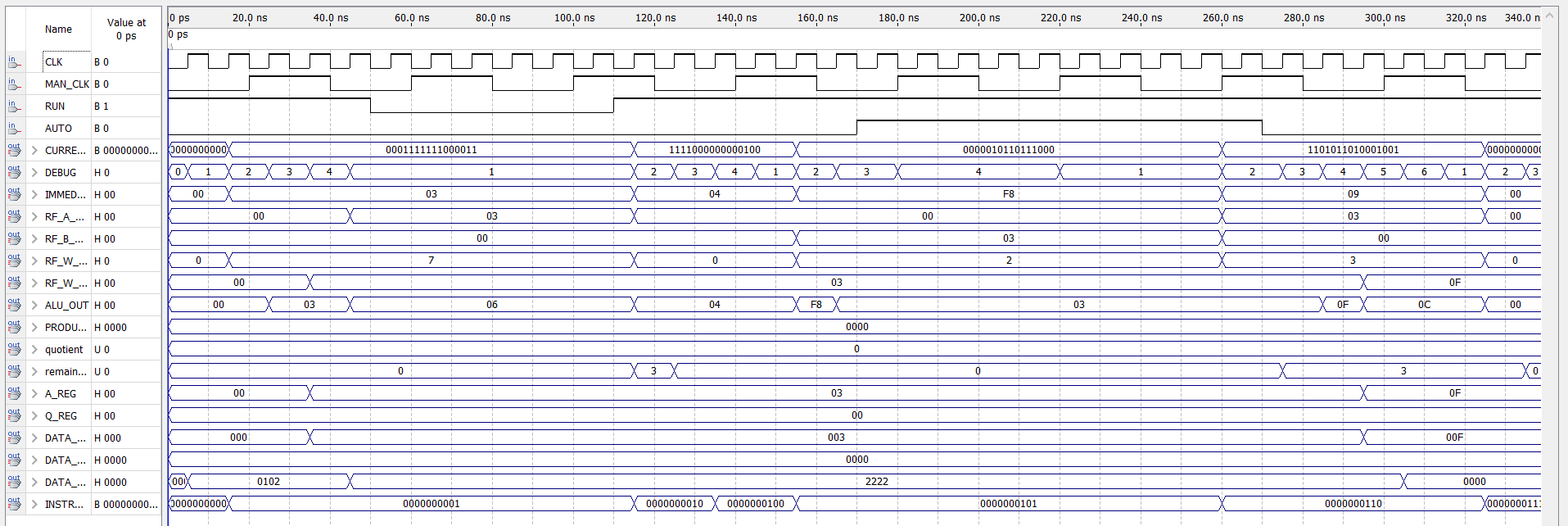


Figure 3. Front panel validation

When AUTO signal is asserted the clock shifts from CLK (auto clock) to MAN\_CLK (manual clock input)

RUN deserted, upon reassertion it resumes operation from last state

# ISA Validation and Performance

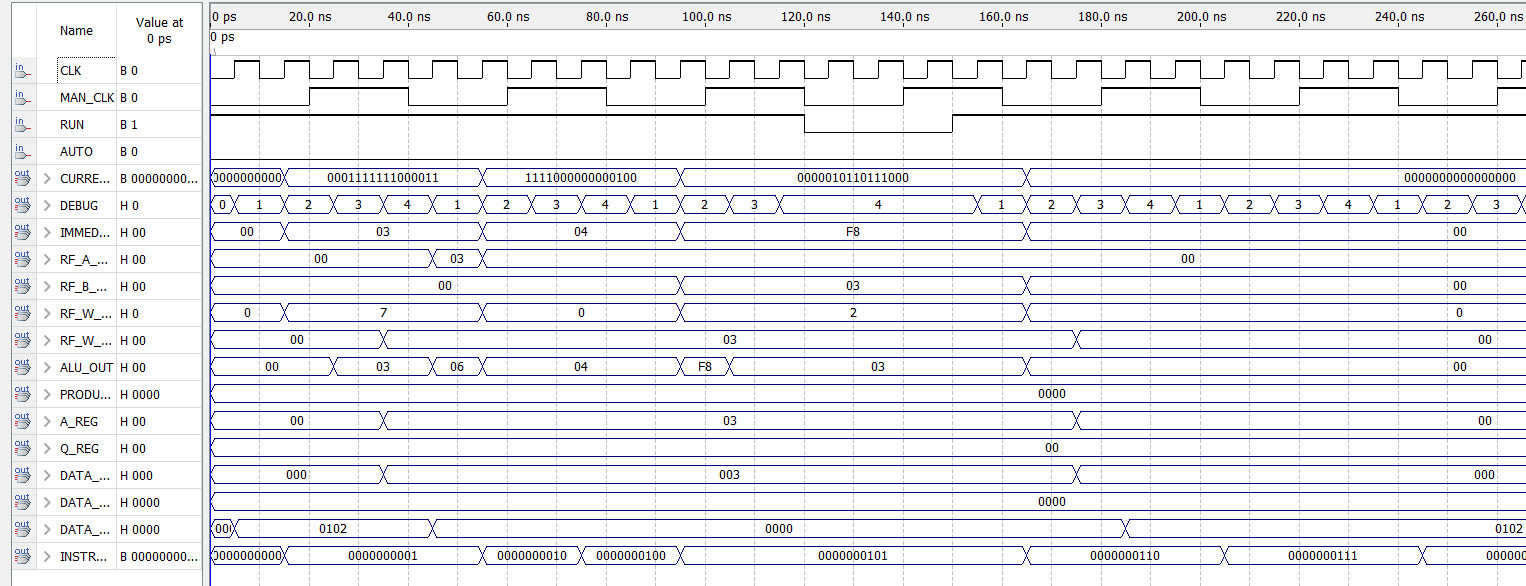
For validation of the Multi-8 ISA, sample instructions from each instruction type are simulated below and the fetch-decode-execute cycles are annotated on the waveforms below. The multi cycle Funtira CPU has been designed considering CPI optimizations. The datapath has been optimized to provide the least CPI and average CPI to gain maximum performance.

CPI and AV CPI Fmax

## J-type Instructions

RUN deserted to show functionality of front panel. Upon reassertion it resumes from last state

Example: Jump instruction: **J 100 (11110000000100)** (i.e. jump to instruction memory address 100)



The next instruction at the jump target address 0000000100 fetched

Fetches the jump instruction at address 0000000001 and increment PC to point to next instruction at address 0000000010 in the decode cycle

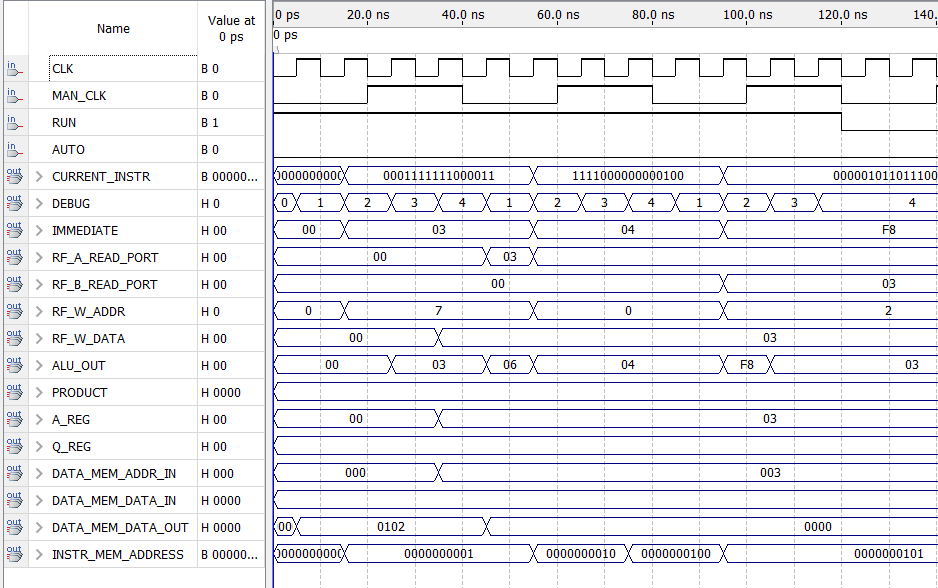
The PC is now fed the jump target address and once execution is done the next instruction at the jump target is fetched

Figure 4. J type instruction state transitions

## I-type Instructions

Example: Add immediate instruction: **ADDI R7,R7,2 (0001111111000010) (R7=0 initially)** (Add 2 to register 7 and store to register 7)

Reads from source register and adds immediate to it in execute stage



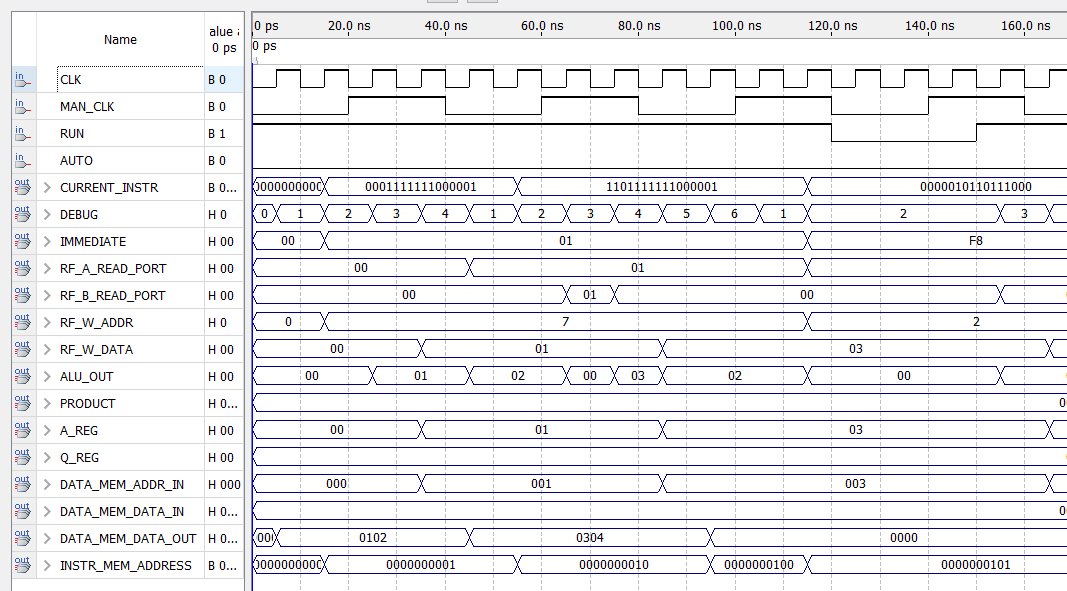
The result is then written back to destination register

Fetches the ADDI instruction at address 0000000001 and increment PC to point to next instruction at address 0000000010 in the decode cycle

Figure 5. I type instruction state transitions (Immediate)

Example: Conditional Branch: **BREQ R7,R7,1 (1101111111000001) (R7=1 initially)** (compare two registers, if equal, load PC with PC plus immediate)

Data is read from the two registers being compared and subtracted. Zero flag is monitored



Fetches the BREQ instruction at address 0000000001 and increment PC to point to next instruction at address 0000000010 in the decode cycle

PC gets updated and branch is executed

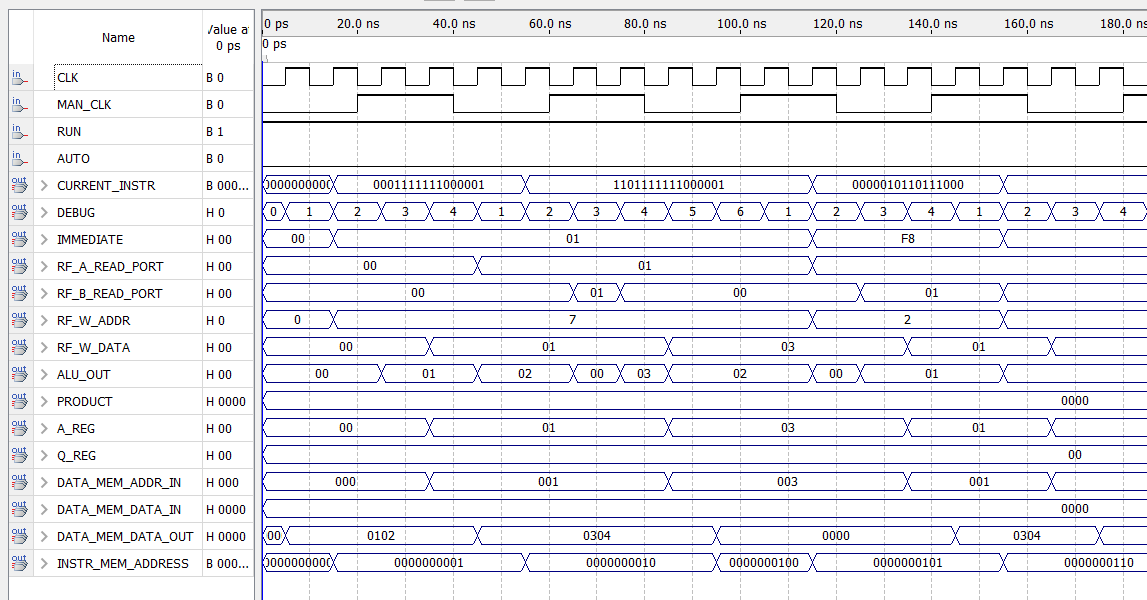
Once Zero flag is detected, destination address is calculated. i.e. current PC + immediate

Figure 6. I type instruction state transitions (Conditional Branch)

## R-type Instructions

Example: Add instruction: **ADD R2,R6,R7 (0000010110111000) (R7=1 and R6=R2=0 initially)** (Add R6 to R7 and store to R2)

Data is read from the two source registers and result is computed



00

Fetches the ADD instruction at address 0000000100 and increment PC to point to next instruction at address 0000000101 in the decode cycle

The result is stored into the destination register

Figure 7. R type instruction state transitions

# Assembler Design

In the assembler design of the project, one of the strong high-level language, which is C# is chosen. C# is one of the Microsoft products. This OOP language is very fast and stable. The source code of the assembler is given in the appendix. To debug the written code, this assembler helps the user to see some inside information for the assembled code piece.

For the sample code below, the assembler shows the information in Figure 8. Assembler screenshot.

LW $R0, $R0, 0

DIV $R2, $R0, $R1

SW $R2, $R4, 1

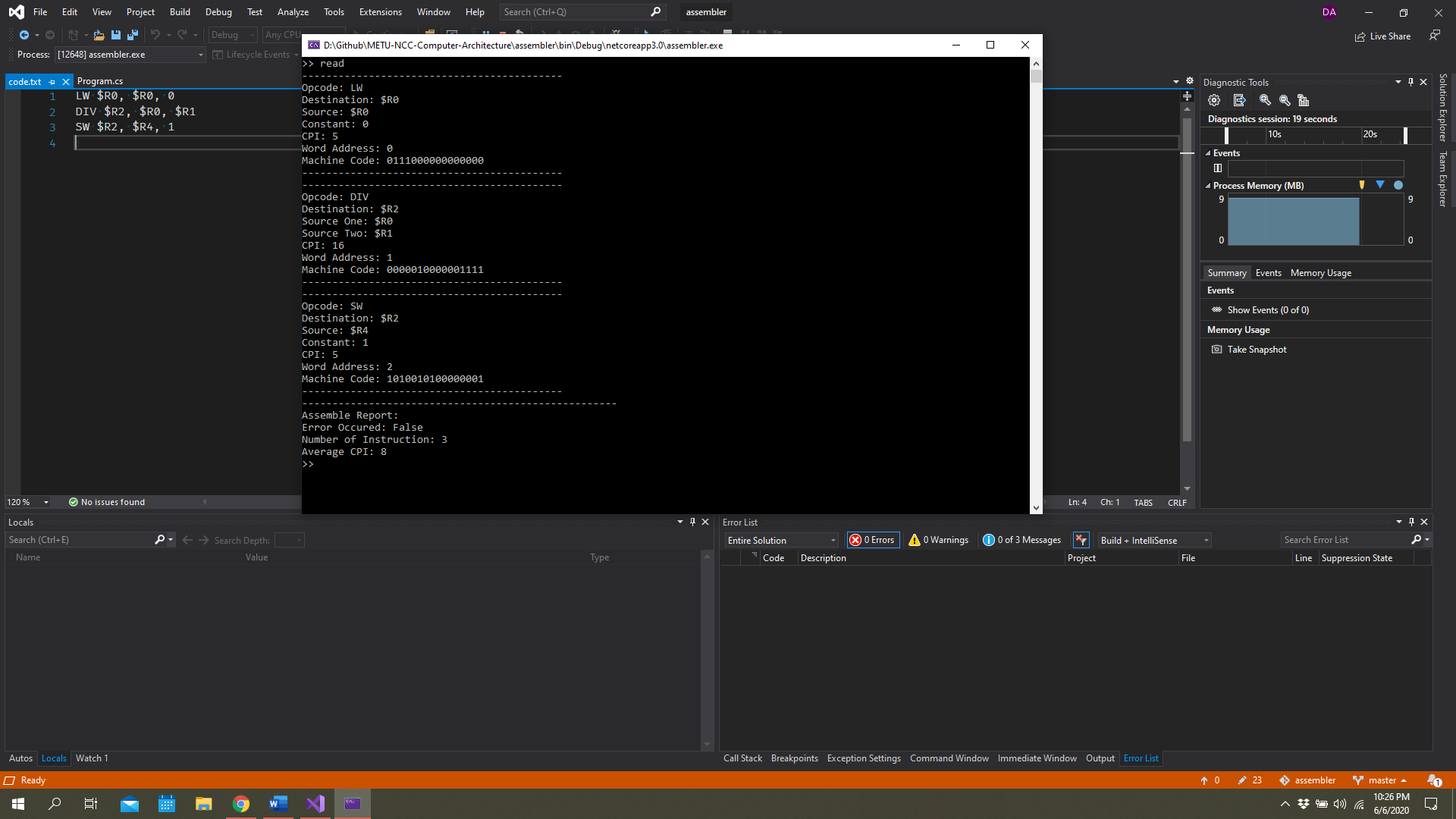


Figure . Assembler screenshot

# CPU Benchmark Definitions and Testing

Lorem ipsum

# Power Calculation

# 

# Conclusion

Lorem ipsum

# Appendix A: Assembler Source Code in C#

using System;

using System.IO;

using System.Collections.Generic;

namespace assembler

{

public class Content

{

public static List<Instruction> Instructions = new List<Instruction>();

public static List<Label> Labels = new List<Label>();

public static List<string> Instruction\_Content = new List<string>();

public static bool Assemble\_Error;

public static int Average\_CPI;

public static int Num\_Of\_Instructions;

}

public class Instruction

{

public string Opcode { get; set; }

public string Destination { get; set; }

public string Source\_One { get; set; }

public string Source\_Two { get; set; }

public int Position { get; set; }

public int CPI { get; set; }

public string Binary\_Opcode { get; set; }

public string Binary\_Destination { get; set; }

public string Binary\_Source\_One { get; set; }

public string Binary\_Source\_Two { get; set; }

public string Binary\_Function { get; set; }

public string Binary\_Constant { get; set; }

public string Binary\_MachineCode { get; set; }

public string Type { get; set; }

public void Print()

{

Console.WriteLine("-------------------------------------------");

Console.WriteLine($"Opcode: {Opcode}");

if(this.Type == "R")

{

Console.WriteLine($"Destination: {Destination}");

Console.WriteLine($"Source One: {Source\_One}");

Console.WriteLine($"Source Two: {Source\_Two}");

}

if(this.Type == "I")

{

Console.WriteLine($"Destination: {Destination}");

Console.WriteLine($"Source: {Source\_One}");

Console.WriteLine($"Constant: {Source\_Two}");

}

if(this.Type == "J")

{

if(this.Opcode == "JUMP")

{

Console.WriteLine($"Label: {Source\_Two}");

} else

{

Console.WriteLine($"Constant: {Source\_Two}");

}

}

Console.WriteLine($"CPI: {CPI}");

Console.WriteLine($"Word Address: {Position}");

Console.WriteLine($"Machine Code: {Binary\_MachineCode}");

Console.WriteLine("-------------------------------------------");

}

public void Encode()

{

try

{

var Types = new Dictionary<string, string> {

{ "ADD", "R"},

{ "ADDI", "I"},

{ "SUB", "R" },

{ "AND", "R" },

{ "ANDI", "I" },

{ "OR", "R" },

{ "DMADDR", "J" },

{ "XOR", "R" },

{ "SLT", "R" },

{ "MUL", "R" },

{ "DIV", "R" },

{ "SLL", "I" },

{ "SRL", "I" },

{ "SRA", "I" },

{ "LW", "I" },

{ "LWU", "I" },

{ "LWL", "I" },

{ "SW", "I" },

{ "SWU", "I" },

{ "SWL", "I" },

{ "BREQ", "I" },

{ "BRNE", "I" },

{ "JUMP", "J" }

};

var CPI = new Dictionary<string, int> {

{ "ADD", 4},

{ "ADDI", 4},

{ "SUB", 4},

{ "AND", 4},

{ "ANDI", 4},

{ "OR", 4},

{ "DMADDR", 4},

{ "XOR", 4},

{ "SLT", 4},

{ "MUL", 14},

{ "DIV", 16},

{ "SLL", 3},

{ "SRL", 3},

{ "SRA", 3},

{ "LW", 5},

{ "LWU", 4},

{ "LWL", 4},

{ "SW", 5},

{ "SWU", 4},

{ "SWL", 4},

{ "BREQ", 4},

{ "BRNE", 4},

{ "JUMP", 4}

};

var BinaryOpcodes = new Dictionary<string, string> {

{ "ADD", "0000"},

{ "ADDI", "0001"},

{ "SUB", "0000" },

{ "AND", "0000" },

{ "ANDI", "0010" },

{ "OR", "0000" },

{ "DMADDR", "0011" },

{ "XOR", "0000" },

{ "SLT", "0000"},

{ "MUL", "0000" },

{ "DIV", "0000" },

{ "SLL", "0100" },

{ "SRL", "0101" },

{ "SRA", "0110" },

{ "LW", "0111" },

{ "LWU", "1000" },

{ "LWL", "1001" },

{ "SW", "1010" },

{ "SWU", "1011" },

{ "SWL", "1100" },

{ "BREQ", "1101" },

{ "BRNE", "1110" },

{ "JUMP", "1111" }

};

var BinaryFunctions = new Dictionary<string, string> {

{ "ADD", "000"},

{ "SUB", "001" },

{ "AND", "100" },

{ "OR", "011" },

{ "XOR", "110" },

{ "SLT", "101"},

{ "MUL", "010" },

{ "DIV", "111" },

};

var BinaryRegisters = new Dictionary<string, string>

{

{ "$R0", "000"},

{ "$R1", "001"},

{ "$R2", "010"},

{ "$R3", "011"},

{ "$R4", "100"},

{ "$R5", "101"},

{ "$R6", "110"},

{ "$R7", "111"}

};

this.Type = Types[Opcode];

this.Binary\_Opcode = BinaryOpcodes[Opcode];

this.CPI = CPI[Opcode];

if (this.Type == "R")

{

this.Binary\_Function = BinaryFunctions[Opcode];

this.Binary\_Destination = BinaryRegisters[this.Destination];

this.Binary\_Source\_One = BinaryRegisters[this.Source\_One];

this.Binary\_Source\_Two = BinaryRegisters[this.Source\_Two];

this.Binary\_MachineCode = this.Binary\_Opcode + this.Binary\_Destination + this.Binary\_Source\_One + this.Binary\_Source\_Two + this.Binary\_Function;

}

if (this.Type == "I")

{

if (this.Opcode == "BREQ" || this.Opcode == "BRNE")

{

this.Binary\_Destination = BinaryRegisters[this.Destination];

this.Binary\_Source\_One = BinaryRegisters[this.Source\_One];

foreach (Label label in Content.Labels)

{

if (label.Name == this.Source\_Two)

{

int Offset = label.Instruction\_Position - this.Position - 2;

this.Binary\_Constant = Convert.ToString(Offset, 2).PadLeft(6, '0');

}

}

}

else

{

this.Binary\_Destination = BinaryRegisters[this.Destination];

this.Binary\_Source\_One = BinaryRegisters[this.Source\_One];

this.Binary\_Constant = Convert.ToString(Convert.ToInt32(this.Source\_Two), 2).PadLeft(6, '0');

}

if (this.Binary\_Constant.Length > 6)

{

this.Binary\_Constant = this.Binary\_Constant.Substring(this.Binary\_Constant.Length - 6);

}

this.Binary\_MachineCode = this.Binary\_Opcode + this.Binary\_Destination + this.Binary\_Source\_One + this.Binary\_Constant;

}

if (this.Type == "J")

{

if (this.Opcode == "JUMP")

{

foreach (Label label in Content.Labels)

{

if (label.Name == this.Source\_Two)

{

this.Binary\_Constant = Convert.ToString(label.Instruction\_Position, 2).PadLeft(12, '0');

}

}

this.Binary\_MachineCode = this.Binary\_Opcode + this.Binary\_Constant;

}

else

{

this.Binary\_Constant = Convert.ToString(Convert.ToInt32(this.Source\_Two), 2).PadLeft(12, '0');

if (this.Binary\_Constant.Length > 12)

{

this.Binary\_Constant = this.Binary\_Constant.Substring(this.Binary\_Constant.Length - 12);

}

this.Binary\_MachineCode = this.Binary\_Opcode + this.Binary\_Constant;

}

}

}

catch (Exception e)

{

Console.WriteLine("Error occured. {0} Exception caught.", e);

Content.Assemble\_Error = true;

}

}

public string Get\_MachineCode()

{

return this.Binary\_MachineCode;

}

}

public class Label

{

public string Name { get; set; }

public int Instruction\_Position { get; set; }

public void Print()

{

Console.WriteLine($"{Name} @ {Instruction\_Position}");

}

}

public class Datapath

{

Dictionary<string, string>BinaryRegisters = new Dictionary<string, string>

{

{ "$R0", "000"},

{ "$R1", "001"},

{ "$R2", "010"},

{ "$R3", "011"},

{ "$R4", "100"},

{ "$R5", "101"},

{ "$R6", "110"},

{ "$R7", "111"}

};

public int Program\_Counter { get; set; }

public void Next()

{

}

}

class Program

{

static void Main(string[] args)

{

string command = "None";

while (command != "exit")

{

Console.Write(">> ");

command = Console.ReadLine();

if(command == "read")

{

string[] lines = File.ReadAllLines(Path.Combine(Environment.CurrentDirectory, "..\\..\\..\\code.txt"));

int instruction\_position = 0;

foreach (string line in lines)

{

string label = "";

string opcode = "";

string destination = "";

string source\_one = "";

string source\_two = "";

string temp = "";

for (int counter = 0; counter<line.Length; counter++)

{

if(line[counter].Equals(':'))

{

label = temp;

temp = "";

}

if(Char.IsWhiteSpace(line[counter]) & !(opcode.Length>0))

{

opcode = temp;

temp = "";

}

if(line[counter].Equals(',') & (opcode.Length > 0) & !(destination.Length > 0))

{

destination = temp;

temp = "";

}

if(line[counter].Equals(',') & (destination.Length > 0) & !(source\_one.Length > 0))

{

source\_one = temp;

temp = "";

}

temp = $"{temp}{line[counter]}";

if(counter == line.Length - 1 || line[counter+1].Equals('#'))

{

source\_two = temp;

temp = "";

break;

}

}

label = label.Trim(' ', ',', '#', ':').ToUpper();

opcode = opcode.Trim(' ', ',', '#', ':').ToUpper();

destination = destination.Trim(' ', ',', '#', ':').ToUpper();

source\_one = source\_one.Trim(' ', ',', '#', ':').ToUpper();

source\_two = source\_two.Trim(' ', ',', '#', ':').ToUpper();

if (label.Length > 0)

Content.Labels.Add(new Label {

Name = label,

Instruction\_Position = instruction\_position

});

if( opcode.Length > 0 )

{

Content.Instructions.Add(new Instruction {

Opcode = opcode,

Destination = destination,

Source\_One = source\_one,

Source\_Two = source\_two,

Position = instruction\_position

});

instruction\_position++;

}

// End of the line

}

foreach (Instruction instruction in Content.Instructions)

{

instruction.Encode();

instruction.Print();

Content.Average\_CPI += instruction.CPI;

Content.Instruction\_Content.Add(instruction.Get\_MachineCode());

}

Content.Num\_Of\_Instructions = Content.Instructions.Count;

Content.Average\_CPI /= Content.Num\_Of\_Instructions;

Console.WriteLine("----------------------------------------------------");

Console.WriteLine("Assemble Report:");

Console.WriteLine("Error Occured: {0}", Content.Assemble\_Error);

Console.WriteLine("Number of Instruction: {0}", Content.Num\_Of\_Instructions);

Console.WriteLine("Average CPI: {0}", Content.Average\_CPI);

}

if(command == "convert")

{

String[] MachineCodeLines = Content.Instruction\_Content.ToArray();

System.IO.File.WriteAllLines(Path.Combine(Environment.CurrentDirectory, "..\\..\\..\\machinecode.txt"), MachineCodeLines);

}

}

}

}

}

# Appendix B: Source Code of Control Unit

module ControlUnit(instruction,

PC\_RST,

RST,

CLK,

RF\_EN,

MULT\_EN,

LDA, LDQ,

PC\_MUX\_SEL,

SR\_SEL,

PLUS1\_SEL,

SR,

SL,

A\_SEL,

B\_SEL,

PC\_EN,

MUL\_SEL,

INST\_TYPE\_MUX\_SEL,

D\_SEL,

OAP,

DATA\_MEM\_SEL,

Cin,

WB\_SEL,

UL\_SEL,

WR\_EN,

INST\_REG\_EN,

Z\_FLAG,

N\_FLAG,

counterchk,

DIV\_EN);

output reg RST, RF\_EN, MULT\_EN, LDA, LDQ, SR\_SEL, PLUS1\_SEL, SR, SL, PC\_EN, DIV\_EN, INST\_TYPE\_MUX\_SEL, DATA\_MEM\_SEL, Cin, WB\_SEL, UL\_SEL, WR\_EN, INST\_REG\_EN;

output reg [1:0] PC\_MUX\_SEL, A\_SEL, B\_SEL, D\_SEL, MUL\_SEL;

output reg [2:0] OAP;

output reg [3:0] counterchk;

input CLK, PC\_RST, Z\_FLAG, N\_FLAG;

input [15:0] instruction;

integer Count, ShiftCount, MultCount ,DivCount;

integer available, fetch, shift\_available, mult\_available, div\_available, temp\_Z;

initial

begin

RST = 1'b1;

Count = 0;

ShiftCount = 0;

MultCount = 0;

DivCount = 0;

MUL\_SEL = 2'b00;

MULT\_EN = 0; DIV\_EN = 0;

DIV\_EN = 0;

available = 0;

PC\_MUX\_SEL = 0;

INST\_REG\_EN = 0;

PC\_EN = 0;

fetch = 1;

available = 0;

shift\_available = 0;

mult\_available = 0;

div\_available = 0;

end

always @(posedge CLK)

begin

RST = 1'b0;

if (available == 0) begin

Count = 0;

ShiftCount = 0;

MultCount = 0;

DivCount = 0;

end

if (fetch == 0) begin

INST\_REG\_EN = 0;

PC\_EN = 0;

end

if (ShiftCount !== 0) begin

ShiftCount = ShiftCount - 1;

Count = Count - 1;

end

if (MultCount !== 0) begin

MultCount = MultCount - 1;

Count = Count - 1;

end

if (DivCount !== 0) begin

DivCount = DivCount - 1;

Count = Count - 1;

end

if (fetch) begin

INST\_REG\_EN = 1;

PC\_EN = 1;

fetch = 0;

available = 1;

end

if (available) begin

Count = Count + 1;

end

counterchk = Count;

case (instruction[15:12])

4'b0000: // R-type

begin

if(instruction[2:0] == 3'b111) begin // DIV

case(Count)

4'b0010:

begin

A\_SEL = 2'b00;

B\_SEL = 2'b00;

D\_SEL = 2'b01;

INST\_TYPE\_MUX\_SEL = 0;

LDA = 0;

LDQ = 0;

MULT\_EN = 0;

DIV\_EN = 0;

MUL\_SEL = 2'b00;

PC\_MUX\_SEL = 0;

PLUS1\_SEL = 0;

RF\_EN = 0;

OAP = 3'b0;

fetch = 0;

end

4'b0011:

begin

A\_SEL = 2'b00;

B\_SEL = 2'b00;

D\_SEL = 2'b01;

INST\_TYPE\_MUX\_SEL = 0;

LDA = 0;

LDQ = 0;

MULT\_EN = 0;

DIV\_EN = 1;

MUL\_SEL = 2'b10;

PC\_MUX\_SEL = 0;

PLUS1\_SEL = 0;

RF\_EN = 0;

MUL\_SEL = 2'b10;

OAP = 3'b0;

fetch = 0;

end

4'b0100:

begin

if(DivCount == 0 && div\_available == 0) begin

DivCount = 6;

div\_available = 1;

end

if( div\_available == 1 ) begin

if(DivCount == 0) begin

div\_available= 0;

end

else begin

end

end

DIV\_EN = 1;

D\_SEL = 2'b01;

end

4'b0101:

begin

A\_SEL = 2'b00;

B\_SEL = 2'b00;

D\_SEL = 2'b01;

INST\_TYPE\_MUX\_SEL = 0;

LDA = 1;

LDQ = 1;

MULT\_EN = 0; DIV\_EN = 0;

DIV\_EN = 0;

MUL\_SEL = 2'b10;

PC\_MUX\_SEL = 0;

PLUS1\_SEL = 0;

RF\_EN = 0;

MUL\_SEL = 2'b10;

OAP = 3'b0;

fetch = 0;

end

4'b0110:

begin

A\_SEL = 2'b00;

B\_SEL = 2'b00;

D\_SEL = 2'b01;

INST\_TYPE\_MUX\_SEL = 0;

LDA = 0;

LDQ = 0;

MULT\_EN = 0; DIV\_EN = 0;

DIV\_EN = 0;

MUL\_SEL = 2'b10;

PC\_MUX\_SEL = 0;

PLUS1\_SEL = 0;

RF\_EN = 1;

MUL\_SEL = 2'b10;

OAP = 3'b0;

fetch = 0;

end

4'b0111:

begin

A\_SEL = 2'b00;

B\_SEL = 2'b00;

D\_SEL = 2'b00;

INST\_TYPE\_MUX\_SEL = 0;

LDA = 0;

LDQ = 0;

MULT\_EN = 0; DIV\_EN = 0;

DIV\_EN = 0;

MUL\_SEL = 2'b10;

PC\_MUX\_SEL = 0;

PLUS1\_SEL = 1;

RF\_EN = 1;

MUL\_SEL = 2'b10;

OAP = 3'b0;

available = 0;

fetch = 1;

end

endcase

end

else if(instruction[2:0] == 3'b010) begin // MUL

case(Count)

4'b0010:

begin

A\_SEL = 2'b00;

B\_SEL = 2'b00;

D\_SEL = 2'b01;

INST\_TYPE\_MUX\_SEL = 0;

LDA = 0;

LDQ = 0;

MULT\_EN = 0; DIV\_EN = 0;

MUL\_SEL = 2'b00;

PC\_MUX\_SEL = 0;

PLUS1\_SEL = 0;

RF\_EN = 0;

OAP = 3'b0;

fetch = 0;

end

4'b0011:

begin

A\_SEL = 2'b00;

B\_SEL = 2'b00;

D\_SEL = 2'b01;

INST\_TYPE\_MUX\_SEL = 0;

LDA = 0;

LDQ = 0;

MULT\_EN = 1;

MUL\_SEL = 2'b01;

PC\_MUX\_SEL = 0;

PLUS1\_SEL = 0;

RF\_EN = 0;

MUL\_SEL = 2'b01;

OAP = 3'b0;

fetch = 0;

end

4'b0100:

begin

if(MultCount == 0 && mult\_available == 0) begin

MultCount = 7;

mult\_available = 1;

end

if( mult\_available == 1 ) begin

if(MultCount == 0) begin

mult\_available= 0;

end

else begin

end

end

MULT\_EN = 0; DIV\_EN = 0;

D\_SEL = 2'b01;

end

4'b0101:

begin

A\_SEL = 2'b00;

B\_SEL = 2'b00;

D\_SEL = 2'b01;

INST\_TYPE\_MUX\_SEL = 0;

LDA = 1;

LDQ = 1;

MULT\_EN = 0; DIV\_EN = 0;

MUL\_SEL = 2'b01;

PC\_MUX\_SEL = 0;

PLUS1\_SEL = 0;

RF\_EN = 0;

MUL\_SEL = 2'b01;

OAP = 3'b0;

fetch = 0;

end

4'b0110:

begin

A\_SEL = 2'b00;

B\_SEL = 2'b00;

D\_SEL = 2'b01;

INST\_TYPE\_MUX\_SEL = 0;

LDA = 0;

LDQ = 0;

MULT\_EN = 0; DIV\_EN = 0;

MUL\_SEL = 2'b01;

PC\_MUX\_SEL = 0;

PLUS1\_SEL = 0;

RF\_EN = 1;

MUL\_SEL = 2'b01;

OAP = 3'b0;

fetch = 0;

end

4'b0111:

begin

A\_SEL = 2'b00;

B\_SEL = 2'b00;

D\_SEL = 2'b00;

INST\_TYPE\_MUX\_SEL = 0;

LDA = 0;

LDQ = 0;

MULT\_EN = 0; DIV\_EN = 0;

MUL\_SEL = 2'b01;

PC\_MUX\_SEL = 0;

PLUS1\_SEL = 1;

RF\_EN = 1;

MUL\_SEL = 2'b01;

OAP = 3'b0;

available = 0;

fetch = 1;

end

endcase

end

else if(instruction[2:0] == 3'b101) begin // SLT

case(Count)

4'b0010:

begin

A\_SEL = 2'b10;

B\_SEL = 2'b10;

D\_SEL = 2'b01;

INST\_TYPE\_MUX\_SEL = 0;

LDA = 0;

LDQ = 0;

MULT\_EN = 0; DIV\_EN = 0;

MUL\_SEL = 2'b00;

PC\_MUX\_SEL = 0;

PLUS1\_SEL = 0;

RF\_EN = 0;

OAP = 3'b001;

fetch = 0;

end

4'b0011:

begin

if(N\_FLAG == 1 ) begin

A\_SEL = 2'b00;

B\_SEL = 2'b01;

D\_SEL = 2'b01;

INST\_TYPE\_MUX\_SEL = 0;

LDA = 1;

LDQ = 0;

MULT\_EN = 0; DIV\_EN = 0;

MUL\_SEL = 2'b00;

PC\_MUX\_SEL = 0;

PLUS1\_SEL = 0;

RF\_EN = 0;

OAP = instruction[2:0];

fetch = 0;

end

if(N\_FLAG == 0 ) begin

A\_SEL = 2'b00;

B\_SEL = 2'b00;

D\_SEL = 2'b01;

INST\_TYPE\_MUX\_SEL = 0;

LDA = 1;

LDQ = 0;

MULT\_EN = 0; DIV\_EN = 0;

MUL\_SEL = 2'b00;

PC\_MUX\_SEL = 0;

PLUS1\_SEL = 0;

RF\_EN = 0;

OAP = 3'b001;

fetch = 0;

end

end

4'b0100:

begin

A\_SEL = 2'b10;

B\_SEL = 2'b10;

D\_SEL = 2'b01;

INST\_TYPE\_MUX\_SEL = 0;

LDA = 0;

LDQ = 0;

MULT\_EN = 0; DIV\_EN = 0;

MUL\_SEL = 2'b00;

PC\_MUX\_SEL = 0;

PLUS1\_SEL = 0;

RF\_EN = 1;

OAP = 3'b001;

available = 0;

fetch = 1;

end

default:

begin

end

endcase

end

else begin //ADD,SUB,OR,AND,XOR

case(Count)

4'b0010:

begin

A\_SEL = 2'b10;

B\_SEL = 2'b10;

D\_SEL = 2'b01;

INST\_TYPE\_MUX\_SEL = 0;

LDA = 1;

LDQ = 0;

MULT\_EN = 0; DIV\_EN = 0;

MUL\_SEL = 2'b00;

PC\_MUX\_SEL = 0;

PLUS1\_SEL = 0;

RF\_EN = 0;

OAP = instruction[2:0];

fetch = 0;

end

4'b0011:

begin

A\_SEL = 2'b10;

B\_SEL = 2'b10;

D\_SEL = 2'b01;

INST\_TYPE\_MUX\_SEL = 0;

LDA = 1;

LDQ = 0;

MULT\_EN = 0; DIV\_EN = 0;

MUL\_SEL = 2'b00;

PC\_MUX\_SEL = 0;

PLUS1\_SEL = 0;

RF\_EN = 0;

OAP = instruction[2:0];

fetch = 0;

end

4'b0100:

begin

A\_SEL = 2'b10;

B\_SEL = 2'b10;

D\_SEL = 2'b01;

INST\_TYPE\_MUX\_SEL = 0;

LDA = 0;

LDQ = 0;

MULT\_EN = 0; DIV\_EN = 0;

MUL\_SEL = 2'b00;

PC\_MUX\_SEL = 0;

PLUS1\_SEL = 0;

RF\_EN = 1;

OAP = instruction[2:0];

available = 0;

fetch = 1;

end

default:

begin

end

endcase

end

end

4'b1101: // BREQ

begin

case(Count)

4'b0010:

begin

INST\_TYPE\_MUX\_SEL = 1;

A\_SEL = 2'b10;

B\_SEL = 2'b10;

D\_SEL = 2'b01;

LDA = 0;

LDQ = 0;

MULT\_EN = 0; DIV\_EN = 0;

MUL\_SEL = 2'b00;

PC\_MUX\_SEL = 2'b0;

PLUS1\_SEL = 0;

RF\_EN = 0;

OAP = 3'b001;

fetch = 0;

end

4'b0011:

begin

A\_SEL = 2'b10;

B\_SEL = 2'b10;

D\_SEL = 2'b01;

INST\_TYPE\_MUX\_SEL = 1;

LDA = 0;

LDQ = 0;

MULT\_EN = 0; DIV\_EN = 0;

MUL\_SEL = 2'b00;

PC\_MUX\_SEL = 2'b0;

PC\_EN = 0;

PLUS1\_SEL = 0;

RF\_EN = 0;

OAP = 3'b001;

fetch = 0;

end

4'b0100:

begin

if(Z\_FLAG == 1)

begin

temp\_Z = Z\_FLAG;

A\_SEL = 2'b01;

B\_SEL = 2'b11;

D\_SEL = 2'b01;

INST\_TYPE\_MUX\_SEL = 0;

LDA = 1;

LDQ = 0;

MULT\_EN = 0; DIV\_EN = 0;

MUL\_SEL = 2'b00;

PC\_MUX\_SEL = 2'b10;

PC\_EN = 0 ;

PLUS1\_SEL = 0;

RF\_EN = 0;

OAP = 3'b000;

fetch = 0;

end

if(Z\_FLAG == 0)

begin

temp\_Z = Z\_FLAG;

A\_SEL = 2'b01;

B\_SEL = 2'b11;

D\_SEL = 2'b01;

INST\_TYPE\_MUX\_SEL = 0;

LDA = 1;

LDQ = 0;

MULT\_EN = 0; DIV\_EN = 0;

MUL\_SEL = 2'b00;

PC\_MUX\_SEL = 2'b00;

PLUS1\_SEL = 0;

RF\_EN = 0;

OAP = 3'b000;

fetch = 0;

end

end

4'b0101:

begin

if(temp\_Z == 1)

begin

A\_SEL = 2'b10;

B\_SEL = 2'b11;

D\_SEL = 2'b01;

INST\_TYPE\_MUX\_SEL = 0;

LDA = 0;

LDQ = 0;

MULT\_EN = 0; DIV\_EN = 0;

MUL\_SEL = 2'b00;

PC\_MUX\_SEL = 2'b10;

PC\_EN = 1;

PLUS1\_SEL = 0;

RF\_EN = 0;

OAP = 3'b000;

fetch = 0;

end

if(temp\_Z == 0)

begin

A\_SEL = 2'b10;

B\_SEL = 2'b11;

D\_SEL = 2'b01;

INST\_TYPE\_MUX\_SEL = 0;

LDA = 0;

LDQ = 0;

MULT\_EN = 0; DIV\_EN = 0;

MUL\_SEL = 2'b00;

PC\_MUX\_SEL = 2'b00;

PC\_EN = 0;

PLUS1\_SEL = 0;

RF\_EN = 0;

OAP = 3'b000;

fetch = 0;

end

end

4'b0110:

begin

A\_SEL = 2'b10;

B\_SEL = 2'b11;

D\_SEL = 2'b01;

INST\_TYPE\_MUX\_SEL = 0;

LDA = 0;

LDQ = 0;

MULT\_EN = 0; DIV\_EN = 0;

MUL\_SEL = 2'b00;

PC\_MUX\_SEL = 2'b00;

PC\_EN = 0;

PLUS1\_SEL = 0;

RF\_EN = 0;

OAP = 3'b000;

fetch = 1;

available = 0;

end

default:

begin

end

endcase

end

4'b1110: // BRNE

begin

case(Count)

4'b0010:

begin

INST\_TYPE\_MUX\_SEL = 1;

A\_SEL = 2'b10;

B\_SEL = 2'b10;

D\_SEL = 2'b01;

LDA = 0;

LDQ = 0;

MULT\_EN = 0; DIV\_EN = 0;

MUL\_SEL = 2'b00;

PC\_MUX\_SEL = 2'b0;

PLUS1\_SEL = 0;

RF\_EN = 0;

OAP = 3'b001;

fetch = 0;

end

4'b0011:

begin

A\_SEL = 2'b10;

B\_SEL = 2'b10;

D\_SEL = 2'b01;

INST\_TYPE\_MUX\_SEL = 1;

LDA = 0;

LDQ = 0;

MULT\_EN = 0; DIV\_EN = 0;

MUL\_SEL = 2'b00;

PC\_MUX\_SEL = 2'b0;

PC\_EN = 0;

PLUS1\_SEL = 0;

RF\_EN = 0;

OAP = 3'b001;

fetch = 0;

end

4'b0100:

begin

if(Z\_FLAG == 0)

begin

temp\_Z = Z\_FLAG;

A\_SEL = 2'b01;

B\_SEL = 2'b11;

D\_SEL = 2'b01;

INST\_TYPE\_MUX\_SEL = 0;

LDA = 1;

LDQ = 0;

MULT\_EN = 0; DIV\_EN = 0;

MUL\_SEL = 2'b00;

PC\_MUX\_SEL = 2'b10;

PC\_EN = 0 ;

PLUS1\_SEL = 0;

RF\_EN = 0;

OAP = 3'b000;

fetch = 0;

end

if(Z\_FLAG == 1)

begin

temp\_Z = Z\_FLAG;

A\_SEL = 2'b01;

B\_SEL = 2'b11;

D\_SEL = 2'b01;

INST\_TYPE\_MUX\_SEL = 0;

LDA = 1;

LDQ = 0;

MULT\_EN = 0; DIV\_EN = 0;

MUL\_SEL = 2'b00;

PC\_MUX\_SEL = 2'b00;

PLUS1\_SEL = 0;

RF\_EN = 0;

OAP = 3'b000;

fetch = 0;

end

end

4'b0101:

begin

if(temp\_Z == 0)

begin

A\_SEL = 2'b10;

B\_SEL = 2'b11;

D\_SEL = 2'b01;

INST\_TYPE\_MUX\_SEL = 0;

LDA = 0;

LDQ = 0;

MULT\_EN = 0; DIV\_EN = 0;

MUL\_SEL = 2'b00;

PC\_MUX\_SEL = 2'b10;

PC\_EN = 1;

PLUS1\_SEL = 0;

RF\_EN = 0;

OAP = 3'b000;

fetch = 0;

end

if(temp\_Z == 1)

begin

A\_SEL = 2'b10;

B\_SEL = 2'b11;

D\_SEL = 2'b01;

INST\_TYPE\_MUX\_SEL = 0;

LDA = 0;

LDQ = 0;

MULT\_EN = 0; DIV\_EN = 0;

MUL\_SEL = 2'b00;

PC\_MUX\_SEL = 2'b00;

PC\_EN = 0;

PLUS1\_SEL = 0;

RF\_EN = 0;

OAP = 3'b000;

fetch = 0;

end

end

4'b0110:

begin

A\_SEL = 2'b10;

B\_SEL = 2'b11;

D\_SEL = 2'b01;

INST\_TYPE\_MUX\_SEL = 0;

LDA = 0;

LDQ = 0;

MULT\_EN = 0; DIV\_EN = 0;

MUL\_SEL = 2'b00;

PC\_MUX\_SEL = 2'b00;

PC\_EN = 0;

PLUS1\_SEL = 0;

RF\_EN = 0;

OAP = 3'b000;

fetch = 1;

available = 0;

end

default:

begin

end

endcase

end

4'b1111: // JUMP

begin

case(Count)

4'b0010:

begin

A\_SEL = 2'b10;

B\_SEL = 2'b11;

D\_SEL = 2'b01;

INST\_TYPE\_MUX\_SEL = 0;

LDA = 0;

LDQ = 0;

MULT\_EN = 0; DIV\_EN = 0;

MUL\_SEL = 2'b00;

PC\_MUX\_SEL = 2'b01;

PC\_EN = 0;

PLUS1\_SEL = 0;

RF\_EN = 0;

OAP = 3'b0;

fetch = 0;

end

4'b0011:

begin

A\_SEL = 2'b10;

B\_SEL = 2'b11;

D\_SEL = 2'b01;

INST\_TYPE\_MUX\_SEL = 0;

LDA = 0;

LDQ = 0;

MULT\_EN = 0; DIV\_EN = 0;

MUL\_SEL = 2'b00;

PC\_MUX\_SEL = 2'b01;

PC\_EN = 1;

PLUS1\_SEL = 0;

RF\_EN = 0;

OAP = 3'b0;

fetch = 0;

end

4'b0100:

begin

A\_SEL = 2'b10;

B\_SEL = 2'b11;

D\_SEL = 2'b01;

INST\_TYPE\_MUX\_SEL = 0;

LDA = 0;

LDQ = 0;

MULT\_EN = 0; DIV\_EN = 0;

MUL\_SEL = 2'b00;

PC\_MUX\_SEL = 2'b00;

PLUS1\_SEL = 0;

RF\_EN = 0;

OAP = 3'b0;

fetch = 1;

available = 0;

end

default:

begin

end

endcase

end

4'b0001: // ADDI

begin

case(Count)

4'b0010:

begin

A\_SEL = 2'b10;

B\_SEL = 2'b11;

D\_SEL = 2'b01;

INST\_TYPE\_MUX\_SEL = 0;

LDA = 0;

LDQ = 0;

MULT\_EN = 0; DIV\_EN = 0;

MUL\_SEL = 2'b00;

PC\_MUX\_SEL = 0;

PLUS1\_SEL = 0;

RF\_EN = 0;

SR = 0;

SL = 0;

SR\_SEL = 0;

OAP = 3'b0;

fetch = 0;

end

4'b0011:

begin

A\_SEL = 2'b10;

B\_SEL = 2'b11;

D\_SEL = 2'b01;

INST\_TYPE\_MUX\_SEL = 0;

LDA = 1;

LDQ = 0;

MULT\_EN = 0; DIV\_EN = 0;

MUL\_SEL = 2'b00;

PC\_MUX\_SEL = 0;

PLUS1\_SEL = 0;

RF\_EN = 0;

SR = 0;

SL = 0;

SR\_SEL = 0;

OAP = 3'b0;

fetch = 0;

end

4'b0100:

begin

A\_SEL = 2'b10;

B\_SEL = 2'b11;

D\_SEL = 2'b01;

INST\_TYPE\_MUX\_SEL = 0;

LDA = 0;

LDQ = 0;

MULT\_EN = 0; DIV\_EN = 0;

MUL\_SEL = 2'b00;

PC\_MUX\_SEL = 0;

PLUS1\_SEL = 0;

RF\_EN = 1;

SR = 0;

SL = 0;

SR\_SEL = 0;

OAP = 3'b0;

fetch = 1;

available = 0;

end

default:

begin

end

endcase

end

4'b0010: // ANDI

begin

case(Count)

4'b0010:

begin

A\_SEL = 2'b10;

B\_SEL = 2'b11;

D\_SEL = 2'b01;

INST\_TYPE\_MUX\_SEL = 0;

LDA = 1;

LDQ = 0;

MULT\_EN = 0; DIV\_EN = 0;

MUL\_SEL = 2'b00;

PC\_MUX\_SEL = 0;

PLUS1\_SEL = 0;

RF\_EN = 0;

SR = 0;

SL = 0;

SR\_SEL = 0;

OAP = 3'b100;

fetch = 0;

end

4'b0011:

begin

A\_SEL = 2'b10;

B\_SEL = 2'b11;

D\_SEL = 2'b01;

INST\_TYPE\_MUX\_SEL = 0;

LDA = 1;

LDQ = 0;

MULT\_EN = 0; DIV\_EN = 0;

MUL\_SEL = 2'b00;

PC\_MUX\_SEL = 0;

PLUS1\_SEL = 0;

RF\_EN = 0;

SR = 0;

SL = 0;

SR\_SEL = 0;

OAP = 3'b100;

fetch = 0;

end

4'b0100:

begin

A\_SEL = 2'b10;

B\_SEL = 2'b11;

D\_SEL = 2'b01;

INST\_TYPE\_MUX\_SEL = 0;

LDA = 0;

LDQ = 0;

MULT\_EN = 0; DIV\_EN = 0;

MUL\_SEL = 2'b00;

PC\_MUX\_SEL = 0;

PLUS1\_SEL = 0;

RF\_EN = 1;

SR = 0;

SL = 0;

SR\_SEL = 0;

OAP = 3'b100;

available = 0;

fetch = 1;

end

default:

begin

end

endcase

end

4'b0100: // SLL

begin

case(Count)

4'b0010:

begin

A\_SEL = 2'b10;

B\_SEL = 2'b00;

D\_SEL = 2'b01;

INST\_TYPE\_MUX\_SEL = 0;

LDA = 1;

LDQ = 0;

MULT\_EN = 0; DIV\_EN = 0;

MUL\_SEL = 2'b00;

PC\_MUX\_SEL = 0;

PLUS1\_SEL = 0;

RF\_EN = 0;

SR = 0;

SL = 0;

SR\_SEL = 0;

OAP = 3'b0;

fetch = 0;

end

4'b0011:

begin

if(ShiftCount == 0 && shift\_available == 0) begin

ShiftCount = instruction[5:0];

shift\_available = 1;

SL = 1;

end

if( shift\_available == 1 ) begin

if(ShiftCount == 0) begin

shift\_available= 0;

SL = 0;

RF\_EN = 1;

available = 0;

fetch = 1;

end

else begin

SL = 1;

end

end

A\_SEL = 2'b10;

B\_SEL = 2'b00;

D\_SEL = 2'b01;

INST\_TYPE\_MUX\_SEL = 0;

LDA = 0;

LDQ = 0;

MULT\_EN = 0; DIV\_EN = 0;

MUL\_SEL = 2'b00;

PC\_MUX\_SEL = 0;

PLUS1\_SEL = 0;

SR = 0;

SR\_SEL = 0;

OAP = 3'b0;

end

default:

begin

end

endcase

end

4'b0101: // SRL

begin

case(Count)

4'b0010:

begin

A\_SEL = 2'b10;

B\_SEL = 2'b00;

D\_SEL = 2'b01;

INST\_TYPE\_MUX\_SEL = 0;

LDA = 1;

LDQ = 0;

MULT\_EN = 0; DIV\_EN = 0;

MUL\_SEL = 2'b00;

PC\_MUX\_SEL = 0;

PLUS1\_SEL = 0;

RF\_EN = 0;

SR = 0;

SL = 0;

SR\_SEL = 0;

OAP = 3'b0;

fetch = 0;

end

4'b0011:

begin

if(ShiftCount == 0 && shift\_available == 0) begin

ShiftCount = instruction[5:0];

shift\_available = 1;

SR = 1;

end

if( shift\_available == 1 ) begin

if(ShiftCount == 0) begin

shift\_available= 0;

SR = 0;

RF\_EN = 1;

available = 0;

fetch = 1;

end

else begin

SR = 1;

end

end

A\_SEL = 2'b10;

B\_SEL = 2'b00;

D\_SEL = 2'b01;

INST\_TYPE\_MUX\_SEL = 0;

LDA = 0;

LDQ = 0;

MULT\_EN = 0; DIV\_EN = 0;

MUL\_SEL = 2'b00;

PC\_MUX\_SEL = 0;

PLUS1\_SEL = 0;

SL = 0;

SR\_SEL = 0;

OAP = 3'b000;

end

default:

begin

end

endcase

end

4'b0110: // SRA

begin

case(Count)

4'b0010:

begin

A\_SEL = 2'b10;

B\_SEL = 2'b00;

D\_SEL = 2'b01;

INST\_TYPE\_MUX\_SEL = 0;

LDA = 1;

LDQ = 0;

MULT\_EN = 0; DIV\_EN = 0;

MUL\_SEL = 2'b00;

PC\_MUX\_SEL = 0;

PLUS1\_SEL = 0;

RF\_EN = 0;

SR = 0;

SL = 0;

SR\_SEL = 0;

OAP = 3'b0;

fetch = 0;

end

4'b0011:

begin

if(ShiftCount == 0 && shift\_available == 0) begin

ShiftCount = instruction[5:0];

shift\_available = 1;

SR = 1;

SR\_SEL = 1;

end

if( shift\_available == 1 ) begin

if(ShiftCount == 0) begin

shift\_available= 0;

SR = 0;

SR\_SEL = 0;

RF\_EN = 1;

available = 0;

fetch = 1;

end

else begin

SR = 1;

SR\_SEL = 1;

end

end

A\_SEL = 2'b10;

B\_SEL = 2'b00;

D\_SEL = 2'b01;

INST\_TYPE\_MUX\_SEL = 0;

LDA = 0;

LDQ = 0;

MULT\_EN = 0; DIV\_EN = 0;

MUL\_SEL = 2'b00;

PC\_MUX\_SEL = 0;

PLUS1\_SEL = 0;

SL = 0;

OAP = 3'b0;

end

default:

begin

end

endcase

end

4'b0111: // LW

begin

case(Count)

4'b0010:

begin

A\_SEL = 2'b10;

B\_SEL = 2'b11;

D\_SEL = 2'b11;

INST\_TYPE\_MUX\_SEL = 0;

LDA = 1;

LDQ = 0;

MULT\_EN = 0; DIV\_EN = 0;

MUL\_SEL = 2'b00;

PC\_MUX\_SEL = 0;

PLUS1\_SEL = 0;

RF\_EN = 0;

SR = 0;

SL = 0;

SR\_SEL = 0;

OAP = 3'b0;

fetch = 0;

end

4'b0011:

begin

A\_SEL = 2'b10;

B\_SEL = 2'b11;

D\_SEL = 2'b11;

INST\_TYPE\_MUX\_SEL = 0;

LDA = 0;

LDQ = 0;

MULT\_EN = 0; DIV\_EN = 0;

MUL\_SEL = 2'b00;

PC\_MUX\_SEL = 0;

PLUS1\_SEL = 0;

RF\_EN = 0;

SR = 0;

SL = 0;

SR\_SEL = 0;

OAP = 3'b0;

fetch = 0;

end

4'b0100:

begin

A\_SEL = 2'b10;

B\_SEL = 2'b11;

D\_SEL = 2'b11;

INST\_TYPE\_MUX\_SEL = 0;

LDA = 0;

LDQ = 0;

MULT\_EN = 0; DIV\_EN = 0;

MUL\_SEL = 2'b00;

PC\_MUX\_SEL = 0;

PLUS1\_SEL = 0;

RF\_EN = 1;

SR = 0;

SL = 0;

SR\_SEL = 0;

OAP = 3'b0;

fetch = 0;

end

4'b0101:

begin

A\_SEL = 2'b10;

B\_SEL = 2'b11;

D\_SEL = 2'b10;

INST\_TYPE\_MUX\_SEL = 0;

LDA = 0;

LDQ = 0;

MULT\_EN = 0; DIV\_EN = 0;

MUL\_SEL = 2'b00;

PC\_MUX\_SEL = 0;

PLUS1\_SEL = 1;

RF\_EN = 1;

SR = 0;

SL = 0;

SR\_SEL = 0;

OAP = 3'b0;

available = 0;

fetch = 1;

end

default:

begin

end

endcase

end

4'b1000: // LWU

begin

case(Count)

4'b0010:

begin

A\_SEL = 2'b10;

B\_SEL = 2'b11;

D\_SEL = 2'b11;

INST\_TYPE\_MUX\_SEL = 0;

LDA = 1;

LDQ = 0;

MULT\_EN = 0; DIV\_EN = 0;

MUL\_SEL = 2'b00;

PC\_MUX\_SEL = 0;

PLUS1\_SEL = 0;

RF\_EN = 0;

SR = 0;

SL = 0;

SR\_SEL = 0;

OAP = 3'b0;

fetch = 0;

end

4'b0011:

begin

A\_SEL = 2'b10;

B\_SEL = 2'b11;

D\_SEL = 2'b11;

INST\_TYPE\_MUX\_SEL = 0;

LDA = 0;

LDQ = 0;

MULT\_EN = 0; DIV\_EN = 0;

MUL\_SEL = 2'b00;

PC\_MUX\_SEL = 0;

PLUS1\_SEL = 0;

RF\_EN = 0;

SR = 0;

SL = 0;

SR\_SEL = 0;

OAP = 3'b0;

fetch = 0;

end

4'b0100:

begin

A\_SEL = 2'b10;

B\_SEL = 2'b11;

D\_SEL = 2'b10;

INST\_TYPE\_MUX\_SEL = 0;

LDA = 0;

LDQ = 0;

MULT\_EN = 0; DIV\_EN = 0;

MUL\_SEL = 2'b00;

PC\_MUX\_SEL = 0;

PLUS1\_SEL = 1;

RF\_EN = 1;

SR = 0;

SL = 0;

SR\_SEL = 0;

OAP = 3'b0;

available = 0;

fetch = 1;

end

default:

begin

end

endcase

end

4'b1001: // LWL

begin

case(Count)

4'b0010:

begin

A\_SEL = 2'b10;

B\_SEL = 2'b11;

D\_SEL = 2'b11;

INST\_TYPE\_MUX\_SEL = 0;

LDA = 0;

LDQ = 0;

MULT\_EN = 0; DIV\_EN = 0;

MUL\_SEL = 2'b00;

PC\_MUX\_SEL = 0;

PLUS1\_SEL = 0;

RF\_EN = 0;

SR = 0;

SL = 0;

SR\_SEL = 0;

OAP = 3'b0;

fetch = 0;

end

4'b0011:

begin

A\_SEL = 2'b10;

B\_SEL = 2'b11;

D\_SEL = 2'b11;

INST\_TYPE\_MUX\_SEL = 0;

LDA = 1;

LDQ = 0;

MULT\_EN = 0; DIV\_EN = 0;

MUL\_SEL = 2'b00;

PC\_MUX\_SEL = 0;

PLUS1\_SEL = 0;

RF\_EN = 0;

SR = 0;

SL = 0;

SR\_SEL = 0;

OAP = 3'b0;

fetch = 0;

end

4'b0100:

begin

A\_SEL = 2'b10;

B\_SEL = 2'b11;

D\_SEL = 2'b11;

INST\_TYPE\_MUX\_SEL = 0;

LDA = 0;

LDQ = 0;

MULT\_EN = 0; DIV\_EN = 0;

MUL\_SEL = 2'b00;

PC\_MUX\_SEL = 0;

PLUS1\_SEL = 0;

RF\_EN = 0;

SR = 0;

SL = 0;

SR\_SEL = 0;

OAP = 3'b0;

fetch = 0;

end

4'b0101:

begin

A\_SEL = 2'b10;

B\_SEL = 2'b11;

D\_SEL = 2'b11;

INST\_TYPE\_MUX\_SEL = 0;

LDA = 0;

LDQ = 0;

MULT\_EN = 0; DIV\_EN = 0;

MUL\_SEL = 2'b00;

PC\_MUX\_SEL = 0;

PLUS1\_SEL = 0;

RF\_EN = 1;

SR = 0;

SL = 0;

SR\_SEL = 0;

OAP = 3'b0;

available = 0;

fetch = 1;

end

default:

begin

end

endcase

end

4'b1010: // SW

begin

case(Count)

4'b0010:

begin

A\_SEL = 2'b10;

B\_SEL = 2'b11;

D\_SEL = 2'b01;

INST\_TYPE\_MUX\_SEL = 1;

LDA = 0;

LDQ = 0;

MULT\_EN = 0; DIV\_EN = 0;

MUL\_SEL = 2'b00;

PC\_MUX\_SEL = 0;

PLUS1\_SEL = 0;

RF\_EN = 0;

SR = 0;

SL = 0;

SR\_SEL = 0;

WR\_EN = 0;

OAP = 3'b0;

fetch = 0;

end

4'b0011:

begin

A\_SEL = 2'b10;

B\_SEL = 2'b11;

D\_SEL = 2'b01;

INST\_TYPE\_MUX\_SEL = 1;

LDA = 1;

LDQ = 1;

MULT\_EN = 0; DIV\_EN = 0;

MUL\_SEL = 2'b00;

PC\_MUX\_SEL = 0;

PLUS1\_SEL = 0;

RF\_EN = 0;

SR = 0;

SL = 0;

SR\_SEL = 0;

WR\_EN = 0;

OAP = 3'b0;

fetch = 0;

end

4'b0100:

begin

A\_SEL = 2'b10;

B\_SEL = 2'b11;

D\_SEL = 2'b10;

INST\_TYPE\_MUX\_SEL = 1;

LDA = 0;

LDQ = 1;

MULT\_EN = 0; DIV\_EN = 0;

MUL\_SEL = 2'b00;

PC\_MUX\_SEL = 0;

PLUS1\_SEL = 1;

RF\_EN = 0;

SR = 0;

SL = 0;

SR\_SEL = 0;

WR\_EN = 1;

WB\_SEL = 1;

UL\_SEL = 0;

OAP = 3'b0;

fetch = 0;

end

4'b0101:

begin

A\_SEL = 2'b10;

B\_SEL = 2'b11;

D\_SEL = 2'b10;

INST\_TYPE\_MUX\_SEL = 1;

LDA = 0;

LDQ = 0;

MULT\_EN = 0; DIV\_EN = 0;

MUL\_SEL = 2'b00;

PC\_MUX\_SEL = 0;

PLUS1\_SEL = 1;

RF\_EN = 0;

SR = 0;

SL = 0;

SR\_SEL = 0;

WR\_EN = 1;

WB\_SEL = 1;

UL\_SEL = 1;

OAP = 3'b0;

available = 0;

fetch = 1;

end

default:

begin

end

endcase

end

4'b1011: // SWU

begin

case(Count)

4'b0010:

begin

A\_SEL = 2'b10;

B\_SEL = 2'b11;

D\_SEL = 2'b01;

INST\_TYPE\_MUX\_SEL = 1;

LDA = 0;

LDQ = 0;

MULT\_EN = 0; DIV\_EN = 0;

MUL\_SEL = 2'b00;

PC\_MUX\_SEL = 0;

PLUS1\_SEL = 0;

RF\_EN = 0;

SR = 0;

SL = 0;

SR\_SEL = 0;

WR\_EN = 0;

OAP = 3'b0;

fetch = 0;

end

4'b0011:

begin

A\_SEL = 2'b10;

B\_SEL = 2'b11;

D\_SEL = 2'b01;

INST\_TYPE\_MUX\_SEL = 1;

LDA = 1;

LDQ = 1;

MULT\_EN = 0; DIV\_EN = 0;

MUL\_SEL = 2'b00;

PC\_MUX\_SEL = 0;

PLUS1\_SEL = 0;

RF\_EN = 0;

SR = 0;

SL = 0;

SR\_SEL = 0;

WR\_EN = 0;

OAP = 3'b0;

fetch = 0;

end

4'b0100:

begin

A\_SEL = 2'b10;

B\_SEL = 2'b11;

D\_SEL = 2'b10;

INST\_TYPE\_MUX\_SEL = 1;

LDA = 0;

LDQ = 0;

MULT\_EN = 0; DIV\_EN = 0;

MUL\_SEL = 2'b00;

PC\_MUX\_SEL = 0;

PLUS1\_SEL = 1;

RF\_EN = 0;

SR = 0;

SL = 0;

SR\_SEL = 0;

WR\_EN = 1;

WB\_SEL = 1;

UL\_SEL = 1;

OAP = 3'b0;

available = 0;

fetch = 1;

end

default:

begin

end

endcase

end

4'b1100: // SWL

begin

case(Count)

4'b0010:

begin

A\_SEL = 2'b10;

B\_SEL = 2'b11;

D\_SEL = 2'b01;

INST\_TYPE\_MUX\_SEL = 1;

LDA = 0;

LDQ = 0;

MULT\_EN = 0; DIV\_EN = 0;

MUL\_SEL = 2'b00;

PC\_MUX\_SEL = 0;

PLUS1\_SEL = 0;

RF\_EN = 0;

SR = 0;

SL = 0;

SR\_SEL = 0;

WR\_EN = 0;

OAP = 3'b0;

fetch = 0;

end

4'b0011:

begin

A\_SEL = 2'b10;

B\_SEL = 2'b11;

D\_SEL = 2'b01;

INST\_TYPE\_MUX\_SEL = 1;

LDA = 1;

LDQ = 1;

MULT\_EN = 0; DIV\_EN = 0;

MUL\_SEL = 2'b00;

PC\_MUX\_SEL = 0;

PLUS1\_SEL = 0;

RF\_EN = 0;

SR = 0;

SL = 0;

SR\_SEL = 0;

WR\_EN = 0;

OAP = 3'b0;

fetch = 0;

end

4'b0100:

begin

A\_SEL = 2'b10;

B\_SEL = 2'b11;

D\_SEL = 2'b10;

INST\_TYPE\_MUX\_SEL = 1;

LDA = 0;

LDQ = 0;

MULT\_EN = 0; DIV\_EN = 0;

MUL\_SEL = 2'b00;

PC\_MUX\_SEL = 0;

PLUS1\_SEL = 1;

RF\_EN = 0;

SR = 0;

SL = 0;

SR\_SEL = 0;

WR\_EN = 1;

WB\_SEL = 1;

UL\_SEL = 0;

OAP = 3'b0;

available = 0;

fetch = 1;

end

default:

begin

end

endcase

end

4'b0011: // DMADDR

begin

case(Count)

4'b0010:

begin

A\_SEL = 2'b10;

B\_SEL = 2'b11;

D\_SEL = 2'b01;

INST\_TYPE\_MUX\_SEL = 0;

LDA = 0;

LDQ = 0;

MULT\_EN = 0; DIV\_EN = 0;

MUL\_SEL = 2'b00;

PC\_MUX\_SEL = 2'b00;

PC\_EN = 0;

PLUS1\_SEL = 0;

RF\_EN = 0;

OAP = 3'b0;

fetch = 0;

DATA\_MEM\_SEL = 0;

end

4'b0011:

begin

A\_SEL = 2'b10;

B\_SEL = 2'b11;

D\_SEL = 2'b01;

INST\_TYPE\_MUX\_SEL = 0;

LDA = 0;

LDQ = 0;

MULT\_EN = 0; DIV\_EN = 0;

MUL\_SEL = 2'b00;

PC\_MUX\_SEL = 2'b00;

PC\_EN = 0;

PLUS1\_SEL = 0;

RF\_EN = 0;

OAP = 3'b0;

fetch = 0;

DATA\_MEM\_SEL = 1;

end

4'b0100:

begin

A\_SEL = 2'b10;

B\_SEL = 2'b11;

D\_SEL = 2'b01;

INST\_TYPE\_MUX\_SEL = 0;

LDA = 0;

LDQ = 0;

MULT\_EN = 0; DIV\_EN = 0;

MUL\_SEL = 2'b00;

PC\_MUX\_SEL = 2'b00;

PLUS1\_SEL = 0;

RF\_EN = 0;

OAP = 3'b0;

DATA\_MEM\_SEL = 0;

fetch = 1;

available = 0;

end

default:

begin

end

endcase

end

default:

begin

//fetch = 0;

end

endcase

end

endmodule