

|  |  |
| --- | --- |
| **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 2](#_Toc42349700)

[ISA Specifications and Design 3](#_Toc42349701)

[CPU Design and Validation 0](#_Toc42349702)

[ISA Validation and Performance 1](#_Toc42349703)

[Assembler Design 2](#_Toc42349704)

[CPU Benchmark Definitions and Testing 3](#_Toc42349705)

[Power Calculation 4](#_Toc42349706)

[Conclusion 5](#_Toc42349707)

[Appendix A: Assembler Source Code in C# 6](#_Toc42349708)

[Appendix B: Source Code of Control Unit 6](#_Toc42349709)

# Objectives

Lorem ipsum

# ISA Specifications and Design

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **R-TYPE** | | | | |
| 4 | 3 | 3 | 3 | 3 |
| OPCODE (Op) | DESTINATION (Rd) | SOURCE\_1 (Rs) | SOURCE\_2 (Rt) | OAP |
|  |  |  |  |  |
| **I-TYPE** | | | | |
| 4 | 3 | 3 | 6 | |
| OPCODE (Op) | DESTINATION (Rd) | SOURCE\_1 (Rs) | CONSTANT (C) | |
|  |  |  |  |  |
| **J-TYPE** | | | | |
| 4 | 12 | | | |
| OPCODE (Op) | CONSTANT (C) | | | |

* 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. This can be used with I-type instructions after sign-extension to 8-bit, since all operations are using 8-bits.

* J-type

We currently have 1 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.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Instruction** | **Type** | **CPI** | **Operands** | **Description** | **Operation** | **Opcode** | **OAP** |
| ADD | R | 4 | Rd, Rs, Rt | Add two registers and a carry (Cin) | Rd <= Rs + Rt + Cin | 0 | 0 |
| ADDI | I | 4 | Rd, Rs, C | Add a register and a constant (C) | Rd <= Rs + C | 1 | X |
| SUB | R | 4 | Rd, Rs, Rt | Subtract two registers and a carry (Cin) | Rd <= Rs - Rt - Cin | 0 | 1 |
| AND | R | 4 | Rd, Rs, Rt | Logical AND two registers | Rd <= Rs ^ Rt | 0 | 4 |
| ANDI | I | 4 | Rd, Rs, C | Logical AND two registers and a constant (C) | Rd <= Rs ^ C | 2 | X |
| OR | R | 4 | Rd, Rs, Rt | Logical OR two registers | Rd <= Rs v Rt | 0 | 3 |
| DMADDR | J | 4 | 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] | 3 | X |
| XOR | R | 4 | Rd, Rs, Rt | Logical XOR two registers | Rd <= Rs ⊕ Rt | 0 | 6 |
| SLT | R | 4 | Rd, Rs, Rt | If Rs<Rt, Rd=1, else Rd=0. | Rs-Rt, Rd <= N (zero flag) | 0 | 5 |
| MUL | R | 14 | Rd, Rs, Rt | Multiply two registers | [Rd,Rd+1] <= Rs \* Rt | 0 | 2 |
| DIV | R | 16 | Rd, Rs, Rt | Divide two registers | Rd <= Rs / Rt | 0 | 7 |
| SLL | I | 3+C | Rd, Rs, C | Logical shift left Rs and save into Rd with constant amount (C) | Rd <= Rs << C | 4 | X |
| SRL | I | 3+C | Rd, Rs, C | Logical shift right Rs and save into Rd with constant amount (C) | Rd <= Rs >> C | 5 | X |
| SRA | I | 3+C | Rd, Rs, C | Arithmetic shift right Rs and save into Rd with constant amount (C) | Rd <= Rs >> C | 6 | X |
| LW | I | 5 | Rd, Rs, C | Load 16-bit word from data memory | [Rd, Rd+1] <= MEM [Rs + C] | 7 | X |
| LWU | I | 4 | Rd, Rs, C | Load upper byte from data memory location | [Rd, -] <= MEM [Rs + C] | 8 | X |
| LWL | I | 4 | Rd, Rs, C | Load lower byte from data memory location | [-, Rd] <= MEM [Rs + C] | 9 | X |
| SW | I | 5 | Rd, Rs, C | Store 16-bit word to data memory | MEM [Rs + C] <= [Rd,Rd+1] | 10 | X |
| SWU | I | 4 | Rd, Rs, C | Store 8-bit data to upper byte of data memory location | MEM [Rs + C] <= [Rd, -] | 11 | X |
| SWL | I | 4 | Rd, Rs, C | Store 8-bit data to lower byte of data memory location | MEM [Rs + C] <= [-, Rd] | 12 | X |
| BREQ | I | 4 | Rd, Rs, D | Branch if Rd equals Rs | if Rs-Rd=0, i.e. Z=1, PC <= PC + D | 13 | X |
| BRNE | I | 4 | 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 | 4 | D | Jump to address location (D) | PC <= PC+D | 15 | X |
|  |  |  |  | \*\* D is the jump or branch Destination Address |  |  |  |

# CPU Design and Validation

Lorem ipsum

# ISA Validation and Performance

Lorem ipsum

# 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 Microsoft product. 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 1. Assembler screenshot.

LW $R0, $R0, 0

DIV $R2, $R0, $R1

SW $R2, $R4, 1

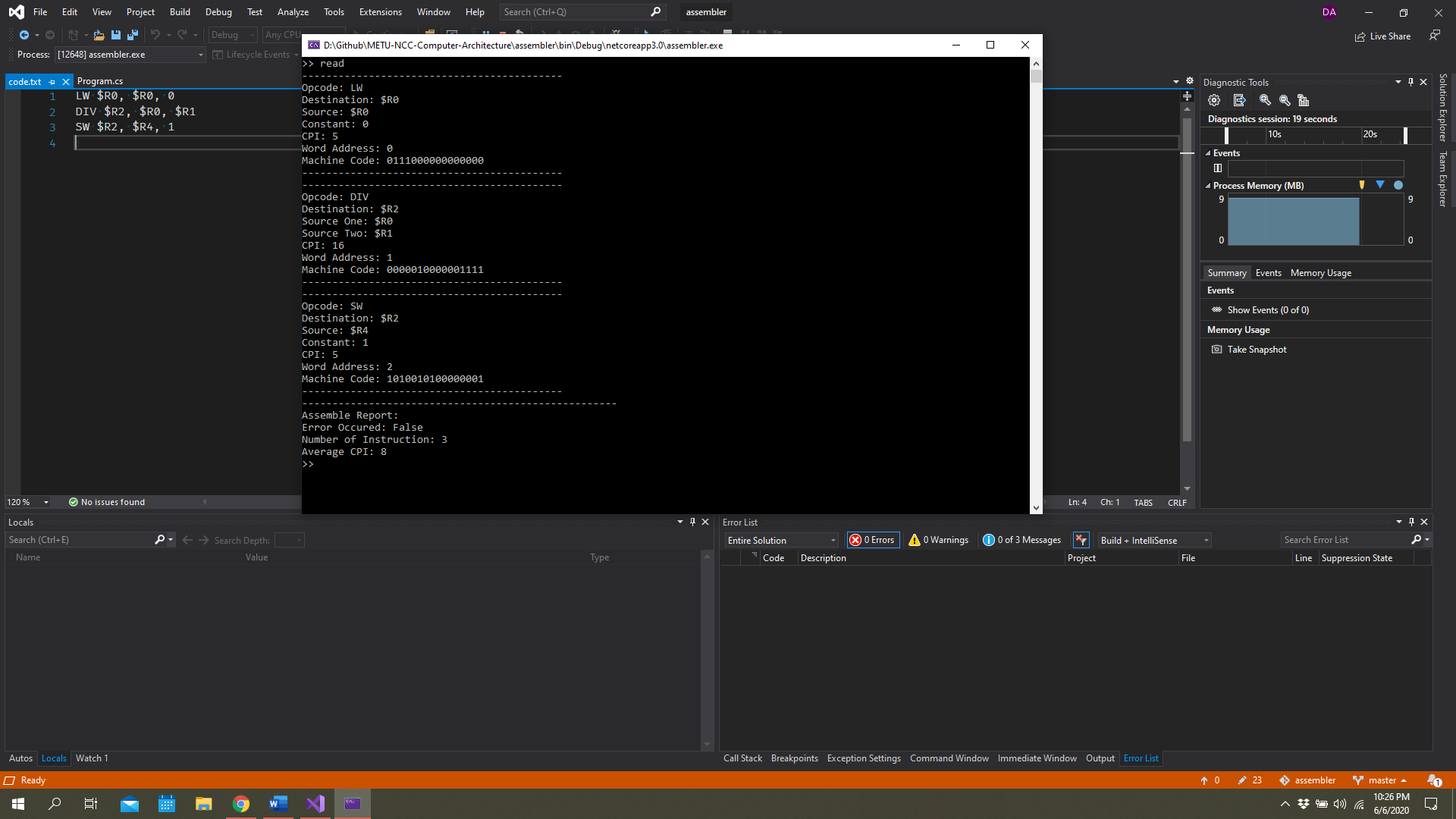


Figure . Assembler screenshot

# 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