**Computer Organization and Architecture**

**COURSE DESIGN**

****

**NAME**

**STUDENT NUMBER**

**COURSE**

Computer Organization and Architecture Course Design

**PROJECT**

A Central Processing Unit (CPU) Basing On Verilog HDL

**SOFTWARE PLATFORM**

ISE 14.1

**DATE**

APR.15.2014

**1. Purpose**

(1) The purpose of this project is to design a simple CPU (Central Processing Unit). This CPU has basic instruction set, and we will utilize its instruction set to generate a very simple program to verify its performance. For simplicity, we will only consider the relationship among the CPU, registers, memory and instruction set. That is to say we only need consider the following items: Read/Write Registers, Read/Write Memory and Execute the instructions.

At least four parts constitute a simple CPU: the control unit, the internal registers, the ALU and instruction set, which are the main aspects of our project design and will be studied.

(2) The use of ISE software for design and simulation.

**2. Tasks**

Single-address instruction format is used in our simple CPU design. The instruction word contains two sections: the operation code (opcode), which defines the function of instructions (addition, subtraction, logic operations, etc.); the address part, in most instructions, the address part contains the memory location of the datum to be operated, we called it direct addressing. In some instructions, the address part is the operand, which is called immediate addressing.

For simplicity, the size of memory is 256× 16 in the computer. The instruction word has 16 bits. The opcode part has 8 bits and address part has 8 bits. The instruction word format can be expressed in Figure 1.

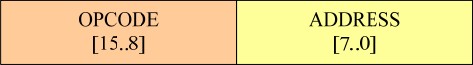


Figure1. The instruction format

The opcode of the relevant instructions are listed in Table 1.

|  |  |  |
| --- | --- | --- |
| Instruction | OPCODE | Comments |
| **ADD X** | 00000001 | ACC+[X]->ACC |
| **SUB X** | 00000010 | ACC-[X]->ACC |
| **AND X** | 00000011 | ACC and [X]->ACC |
| **OR X** | 00000100 | ACC or [X]->ACC |
| **NOT X** | 00000101 | NOT [X]->ACC |
| **SHIFTR** | 00000110 | SHIFT ACC to Right 1bit,Logic Shift |
| **SHIFTL** | 00000111 | SHIFT ACC to Left 1bit,Logic Shift |
| **MPY X** | 00001000 | ACC×[X]->ACC |
| **DIV X** | 00001001 | ACC÷[X]->ACC |
| JMPGEZ X | 00001010 | If ACC≥0 then X->PC else PC+1->PC |
| JMP X | 00001011 | X->PC |
| HALT | 00001100 | Halt a program |
| STORE X | 00001101 | ACC->[X] |
| LOAD X | 00001110 | [X]->ACC |

Table1. List of instructions and relevant opcodes

**3. The overall connection expressed in the top module form**

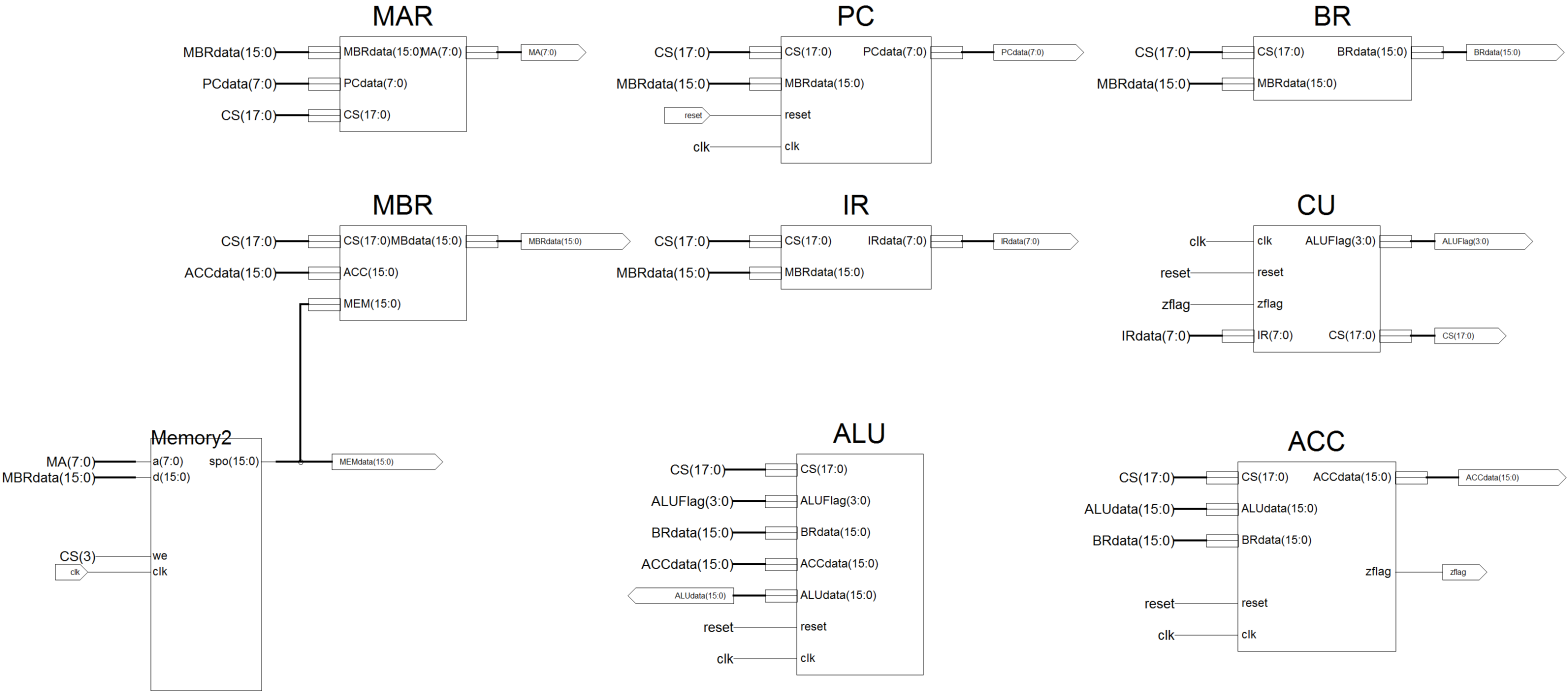


Figure2. The overall connection expressed in the top module form

**4. Internal Registers and Memory**

4.1 Memory (Dist\_mem\_gen\_v7\_1)

The Xilinx LogiCORE™ IP Distributed Memory Generator core uses Xilinx Synthesis Technology (XST) to create a variety of distributed memories. It generates read-only memories (ROMs), single, simple dual, and dual-port random access memories (RAMs), and SRL16-based memories; Supports data depths ranging from 16–65,536 words; Supports data widths ranging from 1–1024 bits.

This memory is a single port RAM generated by Dist\_mem\_gen\_v7\_1.

4.2 MAR (Memory Address Register)

MAR contains the memory location of the word to be read from the memory or written into the memory. Here, READ operation is denoted as the CPU reads from memory, and WRITE operation is denoted as the CPU writes to memory. In our design, MAR has 8 bits to access one of 256 addresses of the memory.

4.3 MBR (Memory Buffer Register)

MBR contains the value to be stored in memory or the last value read from memory. MBR is connected to the address lines of the system bus. In our design, MBR has 16 bits. The module has been horizontally inverted.

4.4 PC (Program Counter)

PC keeps track of the instructions to be used in the program. In our design, PC has 8 bits.

4.5 IR (Instruction Register)

IR contains the opcode part of an instruction. In our design, IR has 8 bits.

4.6 ACC (Accumulator)

ACC holds one operand for ALU, and generally ACC holds the calculation result of ALU. In our design, ACC has 16 bits.BR is used as an input of ALU, it holds other operand for ALU. In our design, BR has 16 bits. The module has been horizontally inverted.

4.7 ALU (Arithmetic Logic Unit)

ALU is a calculation unit which accomplishes basic arithmetic and logic operations. In our design, some operations must be supported which are listed as follows:

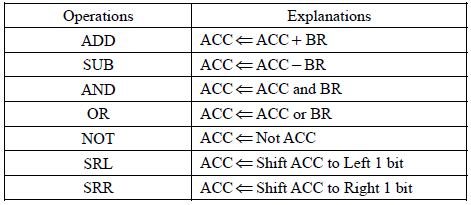


Table2. ALU Operations

4.8 CU (Microprogrammed Control Unit)

We have learnt the knowledge of Microprogrammed control unit. Here, we only review some terms and basic structures.

In the Microprogrammed control, the microprogram consists of some microinstructions and the microprogram is stored in control memory that generates all the control signals required to execute the instruction set correctly. The microinstruction contains some micro-operations which are executed at the same time.

Figure 3 shows the key elements of such an implementation. The set of microinstructions is stored in the control memory. The control address register contains the address of the next microinstructions to be read. When a microinstruction is read from the control memory, it is transferred to a control buffer register. The register connects to the control lines emanating from the control unit. Thus, reading a microinstruction from the control memory is the same as executing that microinstruction. The third element shown in the figure is a sequencing unit that loads the control address register and issues a read command.

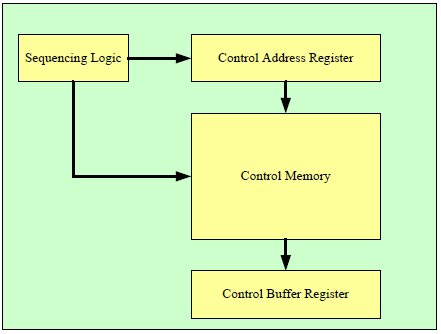


Figure3. Control Unit Micro-architecture

In my design, I simplify the structure of MU. Figure 4 shows the Control Unit Micro-architecture in my design. CM is a Distributed ROM generated by Dist\_mem\_gen\_v7\_1. Its function is the same with the control memory plus the control buffer register in figure 3. Similarly, the SequencingLogic’s function is the same with sequencing logic plus control address register in figure 3.

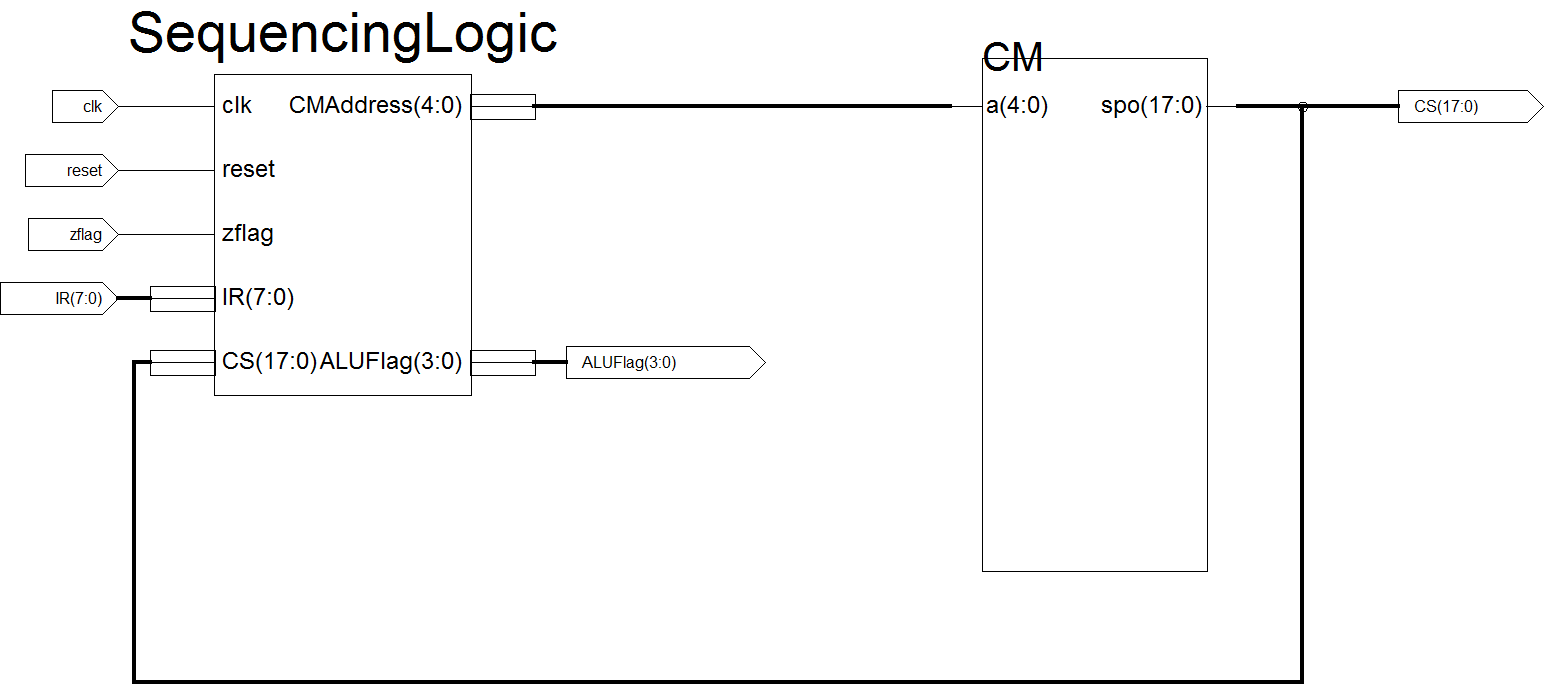


Figure4. Control Unit Micro-architecture in my design

|  |  |  |
| --- | --- | --- |
|  | Control Signals | |
| CM地址控制 | C0 | CAR<=CAR+1 |
| C1 | CAR<=\*\*\* |
| C2 | CAR<=0 |
| 系统总线 | C3 | RWmemory |
| C5 | Wmemory |
| 数据通路 | C4 | memory<=MBR |
| C6 | MBR<=memory |
| C7 | IR<=MBR[15..8] |
| C8 | MAR<=MBR[7..0] |
| C9 | BR<=MBR |
| C10 | PC<=MBR |
| C11 | MAR<=PC |
| C12 | PC<=PC+1 |
| C13 | ALU<=BR |
| C14 | ALU<=ACC |
| C15 | ACC<=ALU |
| C16 | ACC<=BR |
| C17 | MBR<=ACC |
| ALU控制 |  | ALU0 |
|  | ALU1 |
|  | ALU2 |
|  | ALU3 |

Table3. Meanings of each bit of control signal

|  |  |
| --- | --- |
| **运算** | **ALU[3：0]** |
| ADD | 0001 |
| SUB X | 0010 |
| AND X | 0011 |
| OR X | 0100 |
| NOT X | 0101 |
| SHIFTR | 0110 |
| SHIFTL | 0111 |
| MPY X | 1000 |
| DIV X | 1001 |

Table4. Meanings of ALU[3：0] signal

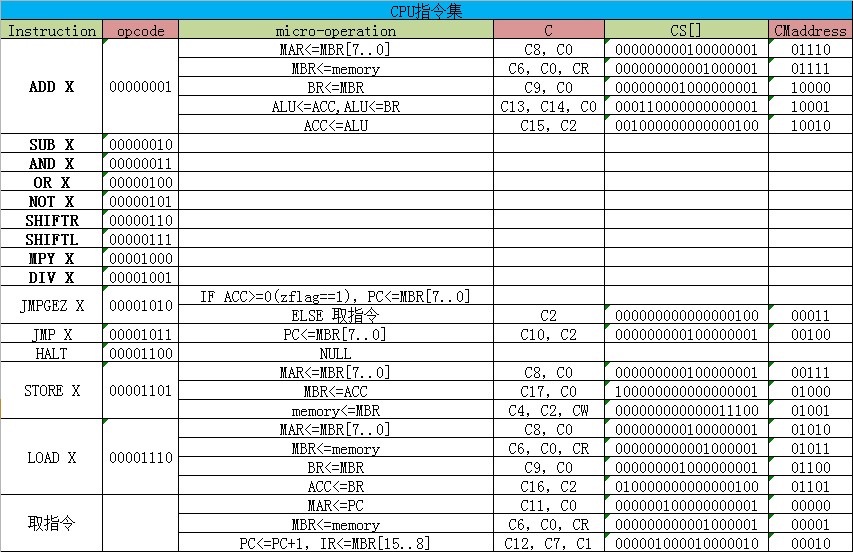


Table5. CPU instruction set and relevant micro-operations and control signals

As figure 4 and table 3 shows, there are 2 output bus from Control Unit. One is the control signal bus CS[17：0], the other one is the ALUFlag signal bus ALUFlag[3：0], which indicates that which calculation is going to be executed in ALU. More specifically, table 4 shows meanings of ALU[3：0] signal.

Table 5 shows the CPU instruction set and accordingly micro-operations and control signals. This is the foundation of the whole design.

**5. Design description of the simulation input waveforms**

Two simulation input signals are needed in my design.

(1) clk: clock signal of the whole system. The clock period is 10 us. All the registers are positive-edge-triggered.

(2) reset: reset when reset = 1. All the reset signals for the registers are synchronized to the clock signal.

**6. Simulation results**

**(1) 1+2+3+……+100=5050**

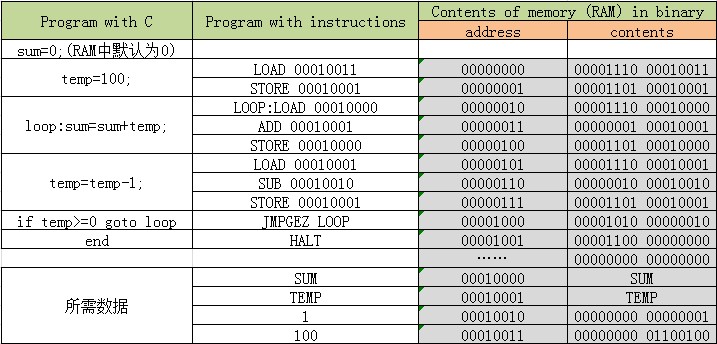
****

Table6. Program of 1+2+3+……+100

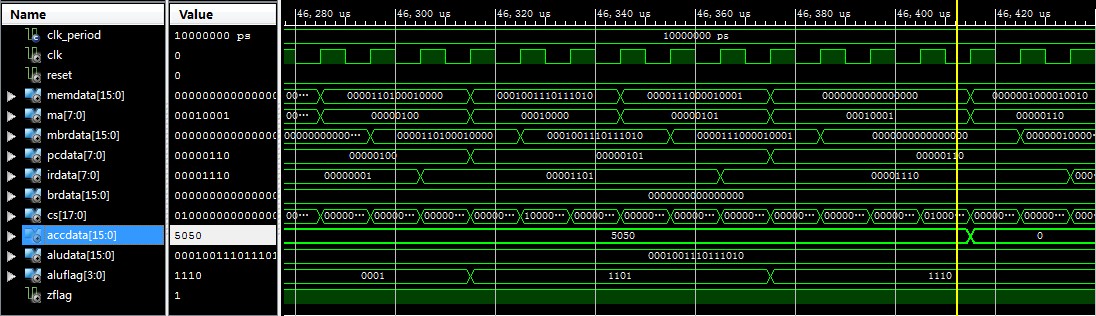


Figure5. Waveform of 1+2+3+……+100

**(2)12×3=36**

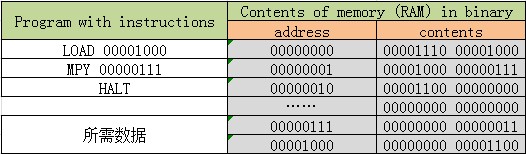


Table7. Program of 12×3

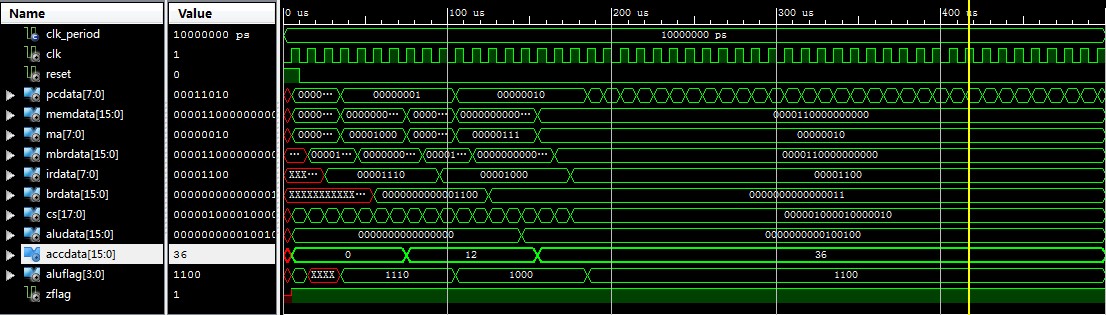
****

Figure6. Waveform of 12×3

**(3)12÷3=4**

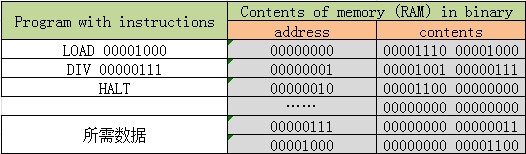


Table8. Program of 12÷3

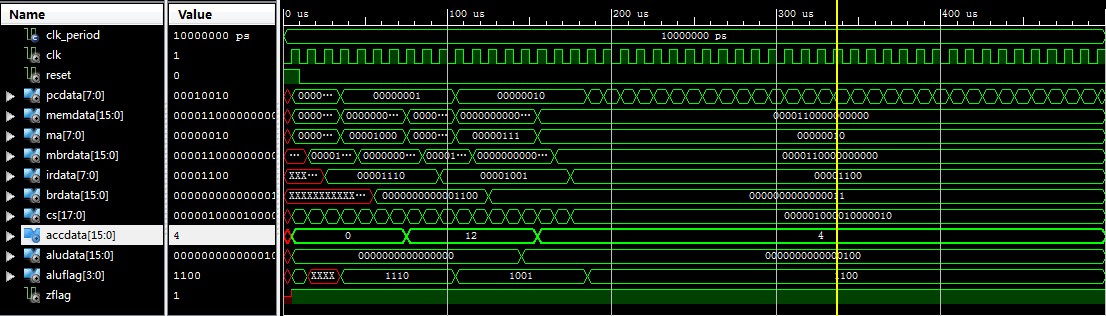


Figure7. Waveform of 12÷3

**7. Conclusions and Discussions**

(1) It takes 46ms to finish the calculation 1+2+3+……+100=5050, in the condition of the clock period 10us.

(2) The result of the multiplication calculation has been limited to 8bit since the MR register is not used.

(3) The division calculation can only been called exact division, since the operator which is used in the code is “/ ”.

(4)感想：CPU的设计相对比较复杂，在真正用ISE进行编程仿真之前需要把课本上CPU部分尤其是CU的工作原理弄清楚，并用EXCEL工具设计自己的指令集和其对应的微操作。这中间很多地方容易出错，也容易使人迷糊，这时就要勤于动手在纸上规划自己的设计。

当草稿纸上和EXCEL中的设计完成后，再用ISE进行编程仿真就会很快了，良好的设计会对后面的工作起到很大的促进作用。在这中间会对课本知识有更加透彻深入的理解。这个实验完全由我独立完成，其中前期设计所使用的时间占到了总花费时间的一半，后期编程调试仿真花费了总时间的另一半。

**Appendix**

**----------MAR----------**

module MAR(

input [15:0] MBRdata,

input [7:0] PCdata,

input [17:0] CS,

output [7:0] MA

);

reg [7:0] MA;

always@(posedge CS[8] or posedge CS[11])

begin

if(CS[8]==1)begin MA<=MBRdata[7:0];end

if(CS[11]==1)begin MA<=PCdata;end

end

endmodule

**----------MBR----------**

module MBR(

output [15:0] MBdata,

input [17:0] CS,

input [15:0] ACC,

input [15:0] MEM

);

reg [15:0] MBdata;

always @ (posedge CS[6] or posedge CS[17])

begin

if(CS[6]==1)begin MBdata<=MEM;end

if(CS[17]==1)begin MBdata<=ACC;end

end

endmodule

**-----------PC----------**

module PC(

input reset,

input [17:0] CS,

input [15:0] MBRdata,

input clk,

output [7:0] PCdata

);

reg [7:0] PCdata;

always@(posedge clk)

begin

if(reset==1) begin PCdata<=8'b00000000; end

if(CS[10]==1)begin PCdata<=MBRdata[7:0]; end

if(CS[12]==1)begin PCdata<=PCdata+1; end

end

endmodule

**----------IR----------**

module IR(

input [17:0] CS,

input [15:0] MBRdata,

output [7:0] IRdata

);

reg [7:0] IRdata;

always@(posedge CS[7])

begin

if(CS[7]==1)begin IRdata<=MBRdata[15:8]; end

end

endmodule

**----------ACC----------**

module ACC(

input [17:0] CS,

input [15:0] ALUdata,

input [15:0] BRdata,

input reset,

input clk,

output [15:0] ACCdata,

output zflag

);

reg [15:0] ACCdata;

reg zflag;

always@(posedge clk)

begin

if(reset==1)

begin

ACCdata<=16'b0000000000000000;

zflag<=1;

end

else if(ACCdata[15]==0)

begin

if(CS[15]==1)

begin

ACCdata<=ALUdata;

zflag<=1;

end

else if(CS[16]==1)

begin

ACCdata<=BRdata;

zflag<=1;

end

else begin zflag<=1;end

end

else

begin

if(CS[15]==1)

begin

ACCdata<=ALUdata;

zflag<=0;

end

else if(CS[16]==1)

begin

ACCdata<=BRdata;

zflag<=0;

end

else begin zflag<=0;end

end

end

endmodule

**----------ALU----------**

module ALU(

input [17:0] CS,

input [3:0] ALUFlag,

input [15:0] BRdata,

input [15:0] ACCdata,

input reset,

input clk,

output [15:0] ALUdata

);

reg [15:0] ALUdata;

always@(posedge clk)

begin

if(reset==1)

begin

ALUdata<=16'b0000000000000000;

end

else if(CS[13]==1&&CS[14]==1)

begin

case(ALUFlag)

1:ALUdata<=ACCdata+BRdata;

2:ALUdata<=ACCdata-BRdata;

3:ALUdata<=ACCdata&BRdata;

4:ALUdata<=ACCdata|BRdata;

5:ALUdata<=~BRdata;

6:ALUdata<=ACCdata>>1;

7:ALUdata<=ACCdata<<1;

8:ALUdata<=ACCdata\*BRdata;

9:ALUdata<=ACCdata/BRdata;

default:;

endcase

end

end

endmodule

**---------- SequencingLogic----------**

module SequencingLogic(

input clk,

input reset,

input zflag,

input [7:0] IR,

input [17:0] CS,

output [4:0] CMAddress,

output [3:0] ALUFlag

);

reg [4:0] CMAddress;

reg [3:0] ALUFlag;

always @ (posedge clk)

begin

if(reset==1)begin CMAddress<=5'b00000;ALUFlag<=4'b0000; end

else

begin

if(CS[2]==0&&CS[1]==0&&CS[0]==1)begin CMAddress<=CMAddress+1;ALUFlag<=IR[3:0]; end

if(CS[2]==1&&CS[1]==0&&CS[0]==0)begin CMAddress<=5'b00000;ALUFlag<=IR[3:0]; end

if(CS[2]==0&&CS[1]==1&&CS[0]==0)

begin

case(IR)

1,2,3,4,5,6,7,8,9:begin CMAddress<=5'b01110;ALUFlag<=IR[3:0];end

10:begin

if(zflag==1)begin CMAddress<=5'b00100;ALUFlag<=IR[3:0];end

else begin CMAddress<=5'b00000;ALUFlag<=IR[3:0];end

end

11:begin CMAddress<=5'b00100;ALUFlag<=IR[3:0];end

12:ALUFlag<=IR[3:0];

13:begin CMAddress<=5'b00111;ALUFlag<=IR[3:0];end

14:begin CMAddress<=5'b01010;ALUFlag<=IR[3:0];end

default:ALUFlag<=4'b0000;

endcase

end

end

end

endmodule