**Computer Organization and Architecture**

**Course Design**

**Microprogrammed CPU Design**

****

**School of Information Science and Engineering**

**Southeast University**

**April 2017**

Name Linze Li Chengqi Lv

Student Number 04014008 04014013

**1 Purpose**

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.

1. **Introduction Set**

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**

|  |  |
| --- | --- |
| **OPCODE**  **[15:8]** | **ADDRESS**  **[7:0]** |

*Figure 1 the instruction format*

The opcodes of the relevant instructions are listed in **Table 1**. In Table 1, the notation [x] represents the contents of the location x in the memory. For example, the instruction word 0000\_0001\_1011\_10012  ( 01B916) means that the CPU adds word at location B916 in memory to the accumulator (ACC); the instruction word 0000\_0101\_0000\_0111 2( 0A0716) means if he sign bit of the ACC (ACC [15]) is 0, the CPU will use the address part of the instruction’s address of next instruction, if the sign bit is 1, the CPU will increase the program counter (PC) and use its content as the address of the next instruction.

**Table 1 List of instruct ions and relevant opcodes**

|  |  |  |
| --- | --- | --- |
| **INSTRUCTION** | **OPCODE** | |
| STORE | 0000\_0001 | 01 |
| LOAD | 0000\_0010 | 02 |
| ADD | 0000\_0011 | 03 |
| SUB | 0000\_0100 | 04 |
| JMPGEZ | 0000\_0101 | 05 |
| JMP | 0000\_0110 | 06 |
| HALT | 0000\_0111 | 08 |
| MPY | 0000\_1000 | 09 |
| DIV | 0000\_1001 | 0A |
| AND | 0000\_1010 | 0B |
| OR | 0000\_1011 | 0C |
| NOT | 0000\_1100 | 0D |
| SHR | 0000\_1101 | 0E |
| SHL | 0000\_1110 | 0F |
| INC | 0000\_1111 | 10 |
| DEC | 0001\_0000 | 11 |

**3 Design and Practice**

**3.1 Top Design**



ACC

PC

Control signals

Flags

Control

Signals

IR

B R

ALU

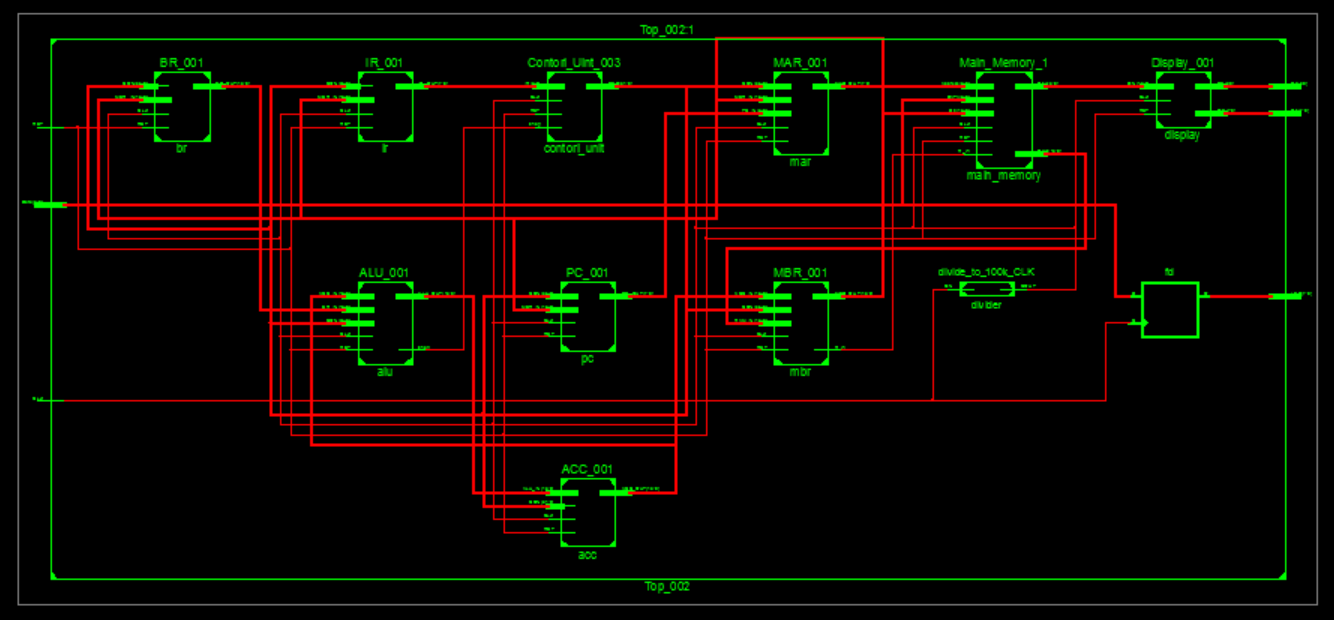
M A R

Control Unit

M B R

*Figure 2 CPU data path and control signals*

**Figure 6** indicates a simple CPU architecture and its use of a variety of internal data paths and control signals. Our CPU logic design is based on this architecture.

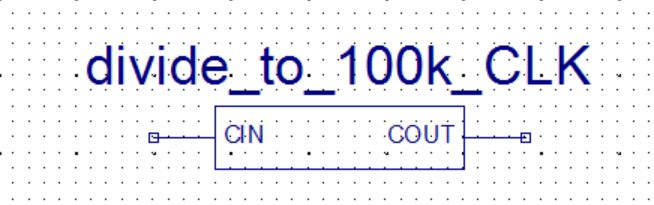


*Figure 3 The top RTL schematic of the CPU*

**Figure 3** is the top RTL schematic of the CPU, which is the visual form created by Verilog. There are eleven modules in figure 6 used to accomplish the basic operation of CPU.

**3.2 Internal Registers and Memory**

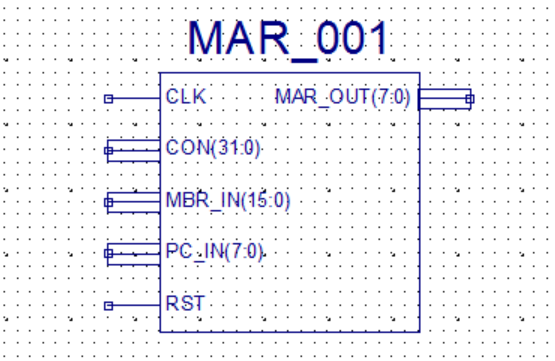
3.2.1 Frequency Divider



*Figure 4 The schematic symbol of Frequency Divider*

The first module is Frequency Divider, which is used to decrease the system clock frequency to 100kHz. Because of it, this all design could work on the Nexys 3 as we expected.

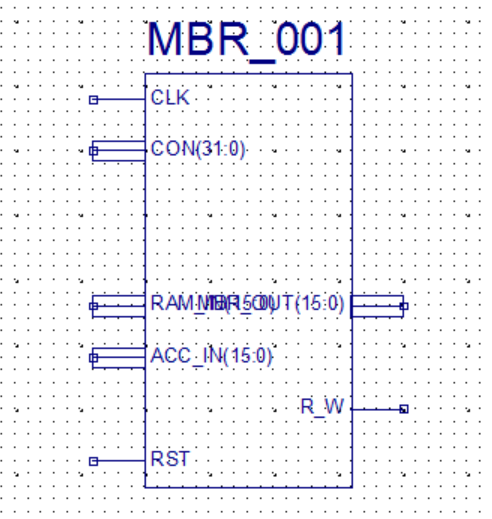
3.2.2 MAR (Memory Address Register)



*Figure 5 The schematic symbol of MAR*

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.

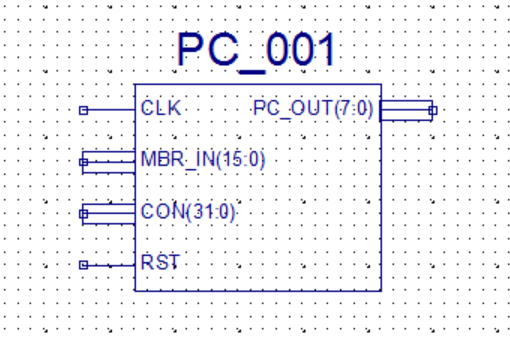
3.2.3 MBR (Memory Buffer Register)



*Figure 6 The schematic symbol of MBR*

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.

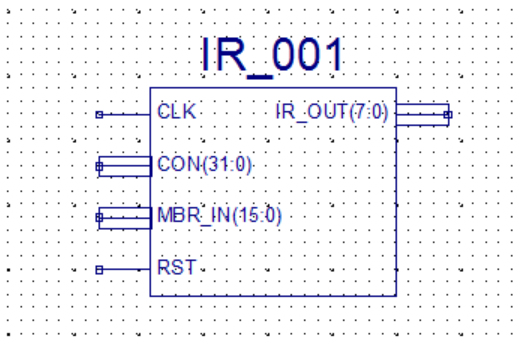
3.2.4 PC (Program Counter)



*Figure 7 The schematic symbol of PC*

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

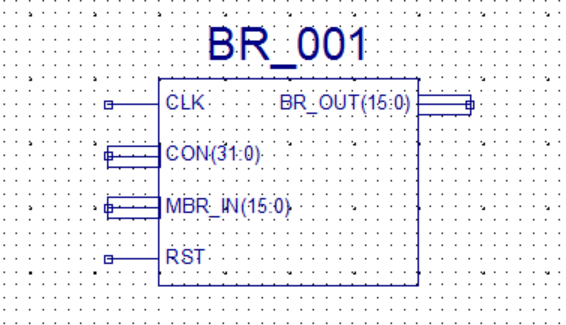
3.2.5 IR (Instruction Register)



*Figure 8 The schematic symbol of PC*

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

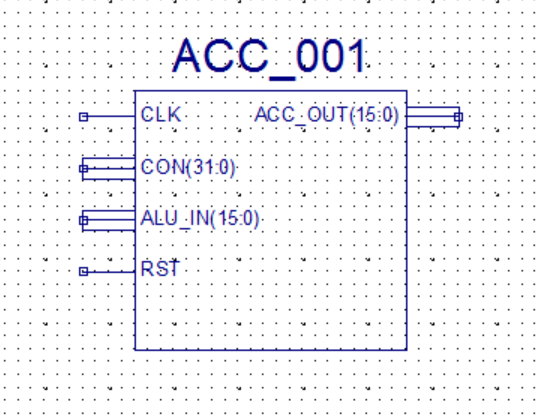
3.2.6 BR (Buffer Register)



*Figure 9 The schematic symbol of BR*

BR is used as an input of ALU, it holds other operand for ALU. In our design, BR has 16 bits.

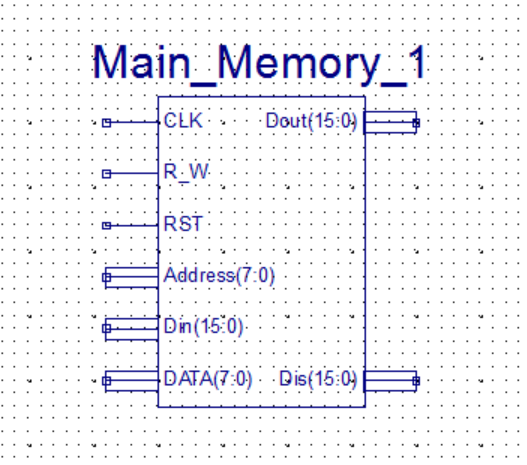
3.2.7 ACC (Accumulator)



*Figure 10 The schematic symbol of ACC*

ACC holds one operand for ALU, and generally ACC holds the calculation result of ALU. In our design, ACC has 16 bits.

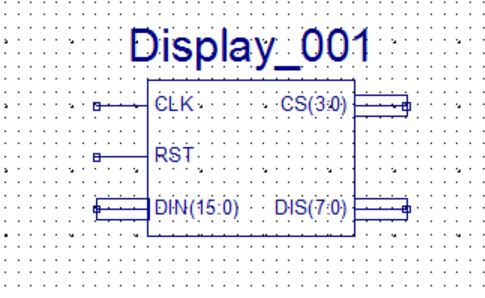
3.2.8 Main Memory



*Figure 8 The schematic symbol of Main Memory*

Main Memory is a RAM with separate input and output ports, it works as memory. Although it’s not an internal register of CPU, we need it to simulate and test the performance of CPU.

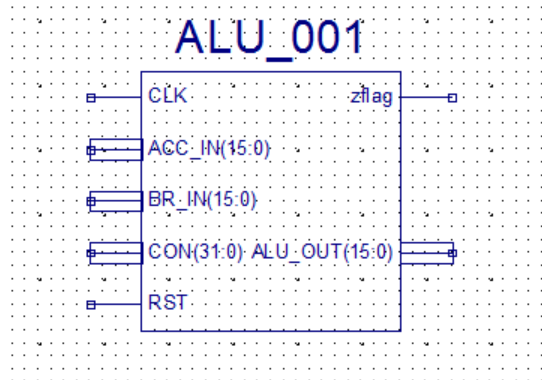
3.2.9 Display



*Figure 8 The schematic symbol of Display*

Display module converts the 16-bit binary number to seven-segment codes and scans the LED segment displays to display the results.

**3.3 ALU**



*Figure 9 The schematic symbol of ALU*

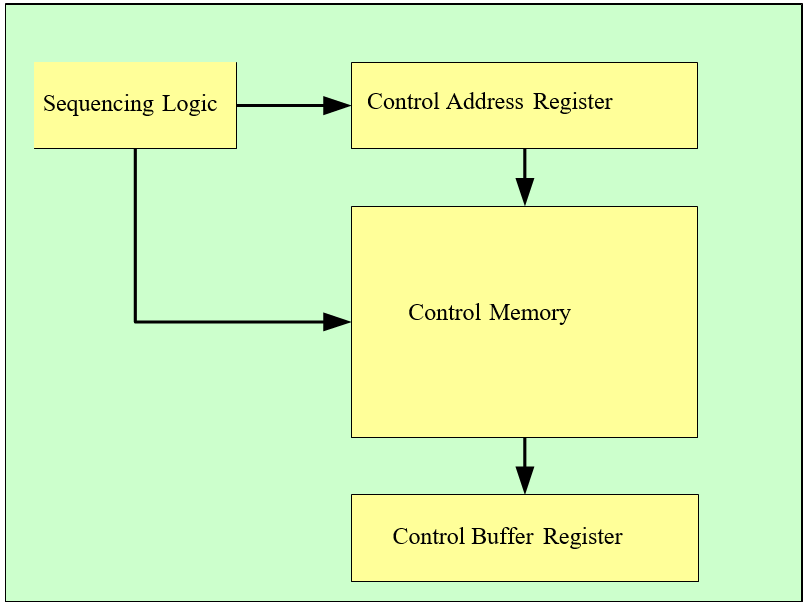
ALU (Arithmetic Logic Unit) is a calculation unit which accomplishes basic arithmetic and logic operations. In our design, some operations will be supported which are listed as follows

**Table 2 ALU Operations**

|  |  |
| --- | --- |
| INSTRUCTION | OPCODE |
| ADD | ACC  ACC + BR |
| SUB | ACC  ACC - BR |
| MPY | ACC ACC \* BR |
| DIV | ACC  ACC / BR |
| AND | ACC  ACC and BR |
| OR | ACC  ACC or BR |
| NOT | ACC  Not ACC |
| SHR | ACC  Shift ACC to Left 1 bit |
| SHL | ACC  Shift ACC to Right 1 bit |

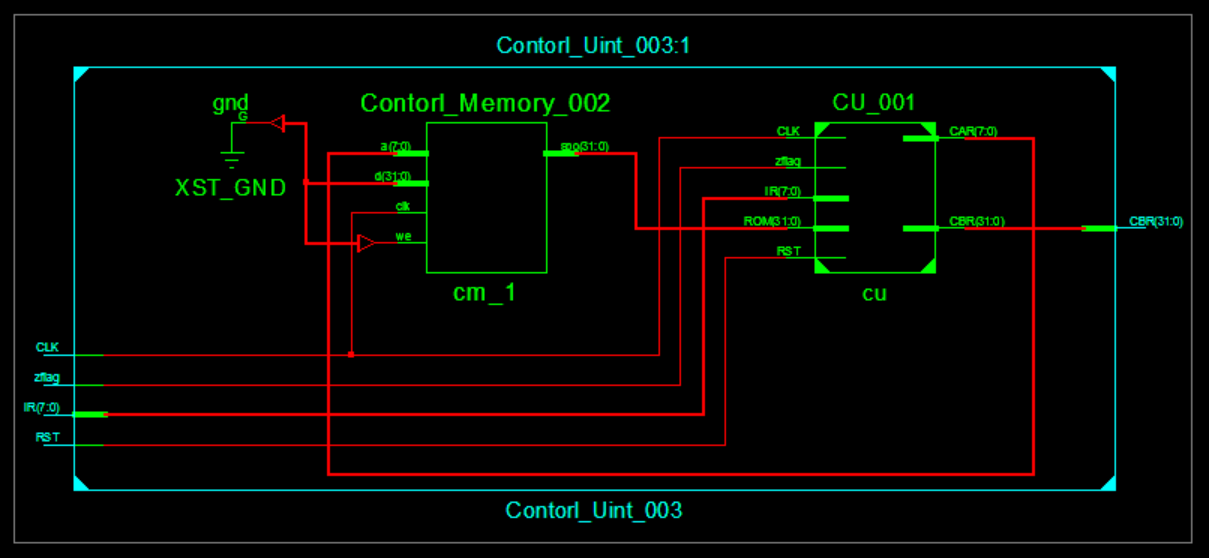
**3.4 Microprogrammed Control Unit**

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 10 Control Unit data path and control signals*

**Figure 5** shows the key elements of such animplementation. 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.



*Figure 11 The top RTL schematic of the Control Unit*

**Figure 11** is the top RTL schematic of the Control Unit, which is the visual form created by Verilog. There are two modules in figure 11, which are Control Memory and control logic model (sequencing logic/control address register/control buffer register).

**Table 3 Control signals**

|  |  |
| --- | --- |
| Bit in Control Memory | Micro-operation |
| C0 | RAM 🡨 MAR |
| C1 | MAR 🡨 PC |
| C2 | PC 🡨 MBR |
| C3 | IR 🡨 MBR |
| C4 | MBR 🡨 RAM |
| C5 | BR 🡨 MBR |
| C6 | ALU 🡨 BR |
| C7 | ALU 🡨 ACC |
| C8 | MAR 🡨 MBR |
| C9 | ACC 🡨 ALU |
| C10 | ACC 🡨 MBR |
| C11 | MBR 🡨 ACC |
| C12 | RAM 🡨 MBR |
| C13 | CU 🡨 IR |
| C14 | PC 🡨 PC+1 |
| C15 | PC=0, ACC=0 |
| C16 | ALU Function Bits (4 Bits) |
| C17 |
| C18 |
| C19 |
| C20 | CAR Control Bits (3 Bits) |
| C21 |
| C22 |
| C23 |  |
| C24 | Control Memory Address (8 Bits) |
| C25 |
| C26 |
| C27 |
| C28 |
| C29 |
| C30 |
| C31 |

**Table 4 Contents of Control Memory**

|  |  |  |
| --- | --- | --- |
| Address(Hex) | Micro-instructions | Control Signals |
| 00 (FETCH) | MBR🡨RAM, CAR🡨CAR+1 | 00000010 |
| 01 | IR🡨MBR, CAR🡨CAR+1 | 00000008 |
| 02 | CAR🡨IR | 00100000 |
| 10 (LOAD) | MAR🡨MBR, PC🡨PC+1, CAR🡨CAR+1 | 00004100 |
| 11 | MBR🡨RAM, CAR🡨CAR+1 | 00000010 |
| 12 | BR🡨MBR, CAR🡨CAR+1 | 00000020 |
| 13 | ALU🡨BR, CAR🡨CAR+1 | 000700C0 |
| 14 | ACC🡨ALU, CAR🡨CAR+1 | 00000200 |
| 15 | MAR🡨PC, CAR🡨0 | 00200002 |
| 20 (STORE) | MAR🡨MBR, PC🡨PC+1, CAR🡨CAR+1 | 00004100 |
| 21 | MBR🡨ACC, CAR🡨CAR+1 | 00000800 |
| 22 | RAM🡨MBR, CAR🡨CAR+1 | 00001000 |
| 23 | MAR🡨PC, CAR🡨0 | 00200002 |
| 30 (ADD) | MAR🡨MBR, PC🡨PC+1, CAR🡨CAR+1 | 00004100 |
| 31 | MBR🡨RAM, CAR🡨CAR+1 | 00000010 |
| 32 | BR🡨MBR, CAR🡨CAR+1 | 00000020 |
| 33 | ALU🡨ACC+BR, CAR🡨CAR+1 | 000000C0 |
| 34 | ACC🡨ALU, CAR🡨CAR+1 | 00000200 |
| 35 | MAR🡨PC, CAR🡨0 | 00200002 |
| 40 (SUB) | MAR🡨MBR, PC🡨PC+1, CAR🡨CAR+1 | 00004100 |
| 41 | MBR🡨RAM, CAR🡨CAR+1 | 00000010 |
| 42 | BR🡨MBR, CAR🡨CAR+1 | 00000020 |
| 43 | ALU🡨ACC-BR, CAR🡨CAR+1 | 000100C0 |
| 44 | ACC🡨ALU, CAR🡨CAR+1 | 00000200 |
| 45 | MAR🡨PC, CAR🡨0 | 00200002 |
| 50 (HALT) | ACC🡨0, CAR🡨0, PC🡨0 | 00208000 |
| 60 (AND) | MAR🡨MBR, PC🡨PC+1, CAR🡨CAR+1 | 00004100 |
| 61 | MBR🡨RAM, CAR🡨CAR+1 | 00000010 |
| 62 | BR🡨MBR, CAR🡨CAR+1 | 00000020 |
| 63 | ALU🡨ACC and BR, CAR🡨CAR+1 | 000400C0 |
| 64 | ACC🡨ALU, CAR🡨CAR+1 | 00000200 |
| 65 | MAR🡨PC, CAR🡨0 | 00200002 |
| 70 (OR) | MAR🡨MBR, PC🡨PC+1, CAR🡨CAR+1 | 00004100 |
| 71 | MBR🡨RAM, CAR🡨CAR+1 | 00000010 |
| 72 | BR🡨MBR, CAR🡨CAR+1 | 00000020 |
| 73 | ALU🡨ACC or BR, CAR🡨CAR+1 | 000500C0 |
| 74 | ACC🡨ALU, CAR🡨CAR+1 | 00000200 |
| 75 | MAR🡨PC, CAR🡨0 | 00200002 |
| 80 (NOT) | ALU🡨~ACC, CAR🡨CAR+1, PC🡨PC+1 | 000640C0 |
| 81 | ACC🡨ALU, CAR🡨CAR+1 | 00000200 |
| 82 | MAR🡨PC, CAR🡨0 | 00200002 |
| 90 (SHR) | ALU🡨ACC>>1, CAR🡨CAR+1, PC🡨PC+1 | 000940C0 |
| 91 | ACC🡨ALU, CAR🡨CAR+1 | 00000200 |
| 92 | MAR🡨PC, CAR🡨0 | 00200002 |
| A0 (SHL) | ALU🡨ACC<<1, CAR🡨CAR+1, PC🡨PC+1 | 000840C0 |
| A1 | ACC🡨ALU, CAR🡨CAR+1 | 00000200 |
| A2 | MAR🡨PC, CAR🡨0 | 00200002 |
| B0 (MPY) | MAR🡨MBR, PC🡨PC+1, CAR🡨CAR+1 | 00004100 |
| B1 | MBR🡨RAM, CAR🡨CAR+1 | 00000010 |
| B2 | BR🡨MBR, CAR🡨CAR+1 | 00000020 |
| B3 | ALU🡨ACC \* BR, CAR🡨CAR+1 | 000200C0 |
| B4 | ACC🡨ALU, CAR🡨CAR+1 | 00000200 |
| B5 | MAR🡨PC, CAR🡨0 | 00200002 |
| C0 (DIV) | MAR🡨MBR, PC🡨PC+1, CAR🡨CAR+1 | 00004100 |
| C1 | MBR🡨RAM, CAR🡨CAR+1 | 00000010 |
| C2 | BR🡨MBR, CAR🡨CAR+1 | 00000020 |
| C3 | ALU🡨ACC / BR, CAR🡨CAR+1 | 000300C0 |
| C4 | ACC🡨ALU, CAR🡨CAR+1 | 00000200 |
| C5 | MAR🡨PC, CAR🡨0 | 00200002 |
| F0 (JUMPZ) | PC🡨PC+1, CAR🡨CAR+1 | 00004000 |
| F1 | CAR🡨0, MAR🡨PC | 00200002 |
| F2 | PC🡨MBR, CAR🡨CAR+1 | 00000004 |
| F3 | CAR🡨0, MAR🡨PC | 00200002 |

**4 Simulation Results**

**4.1 test program**

We designed a program to test these instructions.

The program is: ***Calculate the sum of all integers from 1 to 100, then multiply -5, and then shift left three times.***

4.1.1 Programming with C language:

sum = 0;

temp = 100;

loop : sum = sum + temp;

temp = temp - 1;

if temp >= 0 goto loop;

sum = sum \* (-5);

sum = sum << 3;

end

4.1.2 Assume in the memory:

**sum** is stored at location **A4**,

**temp** is stored at location **A3**,

the contents of location **A0** is 0,

the contents of location **A1** is 1,

the contents of location **A2** is 10010=6416.

the contents of location **A5** is 16'b1000\_0000\_0000\_0101

(-5 represented in its true form)

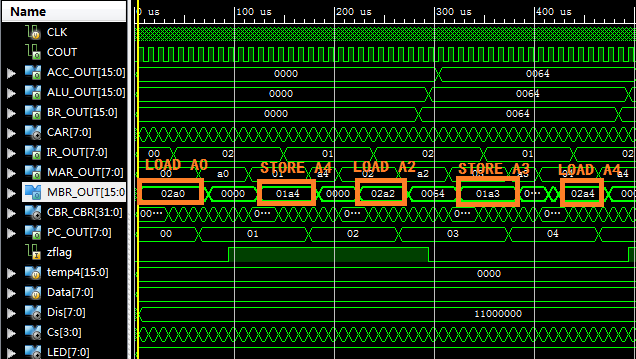
We can translate the above C language program with the instructions listed in Table 1 into the instruction program as shown in Table 4.

**Table 4 The opcode in of the program**

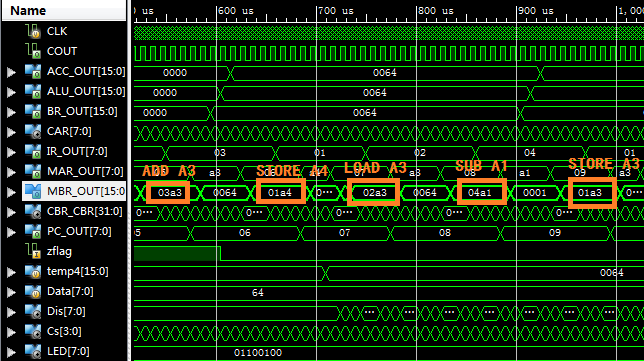
|  |  |  |  |
| --- | --- | --- | --- |
| **Program with C** | **Program with**  **instruction** | **Contents of Memory (RAM) in HEX** | |
| **ADDRESS** | **Contents** |
| sum = 0; | LOAD A0 | 00 | 02A0 |
| STORE A4 | 01 | 01A4 |
| temp = 100; | LOAD A2 | 02 | 02A2 |
| STORE A3 | 03 | 01A3 |
| loop : sum = sum + temp; | LOAD A4 | 04 | 02A4 |
| ADD A3 | 05 | 03A3 |
| STORE A4 | 06 | 01A4 |
| temp = temp – 1; | LOAD A3 | 07 | 02A3 |
| SUB A1 | 08 | 04A1 |
| STORE A3 | 09 | 01A3 |
| if temp >= 0 goto loop; | JMPEZ 04 | 0A | 0504 |
| sum = sum \* ( -5 ); | LOAD A4 | 0B | 02A4 |
| MUL A5 | 0C | 08A5 |
| STORE A4 | 0D | 01A4 |
| sum=sum << 3; | SHL | 0E | 0E00 |
| SHL | 0F | 0E00 |
| SHL | 10 | 0E00 |
| STORE A4 | 11 | 01A4 |
| end | HALT | 12 | 0700 |

***Note:*** *All data are represented in* ***2s true form*** *in memory.*

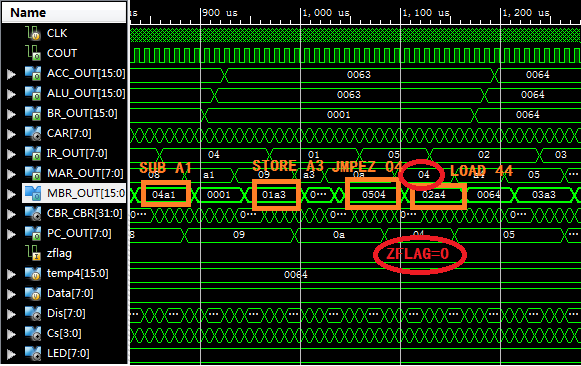
**4.2 Check and Analysis**



*Figure 12*

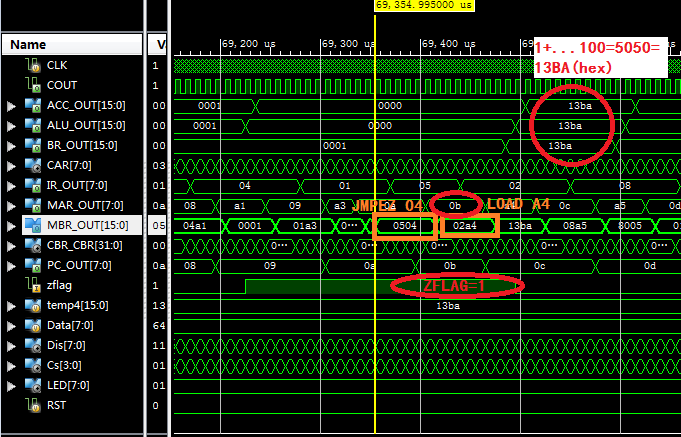


*Figure 13*



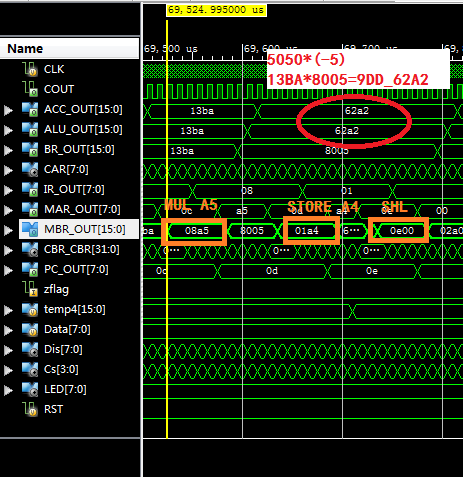
*Figure 14*

Zflag = 0, jump to address 04.



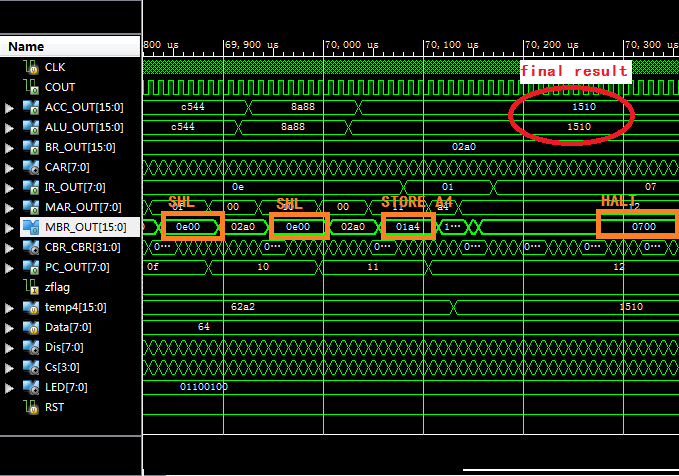
*Figure 15*

When A1=0, it means that the accumulation has finished. The result of “1+...+100” is 5050, in hex is 13BA.



*Figure 16*

Then calculate 5050\*(-5). -5 is represented in 2s true form, the first bit is the sign bit. So that -5 is 8005(HEX).And the result is 9DD\_62A2. Because of the register is only 16-bits, the result is 62A2.



*Figure 17*

After three times left shifting, the final result is 1510, equal to the theoretical result.

**5 Test on Board**



1. SW7~SW0 represent the number to accumulate. (Here is 0110\_0100, 100 in decimal)
2. The middle button of BTS is reset button.
3. LED7~LED0represent the state of the switches.
4. 4-bit digital shows the results of the calculation.

**5 Thinking and Discussion**

The whole process of the project is relatively tortuous, while fortunately, we finally we implemented the requested function. What makes us really pleased is that the program downloaded to FPGA development board operated very well.

Of course, we are benefited from the various problems we encountered in this experiment. The following part introduces what difficulties we encountered and how we solved them.

**5.1 Problems and Solving Methods**

At the beginning of the experiment of CPU, we supposed that the experiment would be relatively easy. But with the progress of the work, we found that the problem was more complex than imagined when all the modules began to debug.

First, in the previously POC experiment, there are only three modules in total. The coordination between them is easier. While the CPU has a total of 11 modules need to coordinate with each other to complete the corresponding work, so their control and information exchange are more complex. But thanks to the current theoretical course, we have a general understanding of CPU activities, so the problem is resolved after our effort.

Then we encountered the most difficult problem -- timing issues. Since IR, Main Memory, MBR, and MAR caused an unexpected error at the same clock rising edge. In order to solve this problem, we spent a lot of time trying to discuss after class. Finally, we solved this problem by adding different time delay in different modules.

**5.2 Conclusion**

In my opinion, the experiment of CPU design is great helpful to us. And the process of finding and solving problems enhance my ability of researching. All the theories are clear after the classes; the understanding of the design is further completed by referring to books rented from our library. This helped me to concentrate myself on the procedure of the whole experiment.

Although it took a lot of time, we were still able to carefully debug of each program. The happiest moment is to see the program running perfect as we expected.

**6 Appendix**

**6.1 Program of CPU Top Design**

module Top\_002(CLK,RST,Data,Dis,Cs,LED);

input CLK,RST;

input [7:0]Data;

output [7:0]Dis;

output [3:0]Cs;

output reg [7:0]LED;

wire [15:0]MBR\_OUT,RAM\_DATA,ACC\_OUT,ALU\_OUT,BR\_OUT;

wire R\_W,flag,CLK\_10K;

wire [31:0]CON;

wire [7:0]MAR\_OUT,PC\_OUT,IR\_OUT;

wire [15:0]Dout;

always@ (posedge CLK) //提示当前拨码开关状态

begin

LED <= Data;

end

divide\_to\_100k\_CLK divider(.CIN(CLK),

.COUT(CLK\_10K));

MBR\_001 mbr(.CON(CON),

.CLK(CLK\_10K),

.RAM\_IN(RAM\_DATA),

.MBR\_OUT(MBR\_OUT),

.ACC\_IN(ACC\_OUT),

.R\_W(R\_W),

.RST(RST));

MAR\_001 mar(.CON(CON),

.CLK(CLK\_10K),

.MBR\_IN(MBR\_OUT),

.MAR\_OUT(MAR\_OUT),

.PC\_IN(PC\_OUT),

.RST(RST));

Main\_Memory\_1 main\_memory(.Address(MAR\_OUT),

.Dout(RAM\_DATA),

.CLK(CLK\_10K),

.R\_W(R\_W),

.Din(MBR\_OUT),

.Dis(Dout),

.DATA(Data),

.RST(RST));

Display\_001 display(.CLK(CLK\_10K),

.DIN(Dout),

.CS(Cs),

.DIS(Dis),

.RST(RST));

PC\_001 pc(.CLK(CLK\_10K),

.MBR\_IN(MBR\_OUT),

.PC\_OUT(PC\_OUT),

.CON(CON),

.RST(RST));

ACC\_001 acc(.CON(CON),

.CLK(CLK\_10K),

.ALU\_IN(ALU\_OUT),

.ACC\_OUT(ACC\_OUT),

.RST(RST));

ALU\_001 alu(.ACC\_IN(ACC\_OUT),

.BR\_IN(BR\_OUT),

.ALU\_OUT(ALU\_OUT),

.CON(CON),

.CLK(CLK\_10K),

.zflag(flag),

.RST(RST));

IR\_001 ir(.CLK(CLK\_10K),

.CON(CON),

.MBR\_IN(MBR\_OUT),

.IR\_OUT(IR\_OUT),

.RST(RST));

BR\_001 br(.CON(CON),

.CLK(CLK\_10K),

.MBR\_IN(MBR\_OUT),

.BR\_OUT(BR\_OUT),

.RST(RST));

Contorl\_Uint\_003 contorl\_unit(.CLK(CLK\_10K),

.zflag(flag),

.IR(IR\_OUT),

.CBR(CON),

.RST(RST));

endmodule

**6.2 Program of Frequency Divide part**

module divide\_to\_100k\_CLK(

CIN,COUT);

input CIN;

output COUT;

reg [8:0] count;

reg [8:0] count\_1;

reg COUT;

initial

begin

count<=0;

count\_1<=0;

COUT<=0;

end

always @(posedge CIN)

begin

count=count+1;

if(count==500)

begin

count\_1=count\_1 + 1;

count=0;

end

else;

if(count\_1==10)

begin

COUT=!COUT;

count\_1=0;

end

else;

end

endmodule

**6.3 Program of MBR part**

module MBR\_001(CON,CLK,RAM\_IN,MBR\_OUT,ACC\_IN,R\_W,RST);

input CLK,RST;

input [15:0] RAM\_IN,ACC\_IN;

input [31:0] CON;

output reg[15:0] MBR\_OUT;

output reg R\_W;

initial

begin

R\_W <= 0;

MBR\_OUT <= 0;

end

always@(posedge CLK or posedge RST)

begin

if(RST)

begin

R\_W <= 0;

MBR\_OUT <= 0;

end

else

begin

if(CON[4])

begin

MBR\_OUT <= RAM\_IN;

R\_W <= 0;

end

else;

if(CON[11])

begin

MBR\_OUT <= ACC\_IN;

R\_W <= 0;

end

else;

if(CON[12])

begin

//MBR\_OUT <= ACC\_IN;

R\_W <= 1;

end

else;

end

end

endmodule

**6.4 Program of MAR part**

module MAR\_001(CON,CLK,MBR\_IN,MAR\_OUT,PC\_IN,RST);

input CLK,RST;

input [7:0] PC\_IN;

input [15:0] MBR\_IN;

input [31:0] CON;

output reg [7:0] MAR\_OUT;

initial

begin

MAR\_OUT = 0;

end

always@(posedge CLK or posedge RST)

begin

if(RST)

begin

MAR\_OUT <= 0;

end

else

begin

if(CON[8])

MAR\_OUT <= MBR\_IN[7:0];

else;

if(CON[1])

MAR\_OUT <= PC\_IN;

else;

end

end

endmodule

**6.5 Program of Main Memory part**

module Main\_Memory\_1(

Address,

Dout,

CLK,

R\_W,

Din,

Dis,

DATA,

RST

);

input [7:0]Address,DATA;

input CLK,R\_W,RST;

input [15:0]Din;

output reg [15:0]Dout;

output reg [15:0]Dis;

reg [15:0]temp0,temp1,temp2,temp3,temp4;

initial

begin

Dout <= 0;

temp3 <= 0;

temp4 <= 0;

end

always@(negedge CLK or posedge RST)

begin

if(RST)

begin

Dout = 0;

temp3 = 0;

temp4 = 0;

end

else

begin

if(!R\_W)

case(Address)

//完成从1到A2的求和

8'h00: Dout=16'h02A0; //LOAD A0

8'h01: Dout=16'h01A4; //Store A4 = A0

8'h02: Dout=16'h02A2; //LOAD A2

8'h03: Dout=16'h01A3; //Store A3 = A2

8'h04: Dout=16'h02A4; //LOAD A4

8'h05: Dout=16'h03A3; //ADD A3 A4 + A3

8'h06: Dout=16'h01A4; //Store A4 = A4 + A3

8'h07: Dout=16'h02A3; //LOAD A3

8'h08: Dout=16'h04A1; //SUB A1

8'h09: Dout=16'h01A3; //STORE A3 = A3 - A1

8'h0A: Dout=16'h0504; //JMPEZ 04

//完成A4\*（-5）

8'h0B: Dout=16'h02A4; //LOAD A4

8'h0C: Dout=16'h08A5; //MUL A5

8'h0D: Dout=16'h01A4; //Store A4 =A4\*A5

//完成A4左移3位

8'h0E: Dout=16'h0E00; //SHL

8'h0F: Dout=16'h0E00; //SHL

8'h10: Dout=16'h0E00; //SHL

8'h11: Dout=16'h01A4; //Store A4

//结束程序

8'h12: Dout=16'h0700; //HALT

//数据存储区

8'hA0: Dout=16'h0000;

8'hA1: Dout=16'h0001;

8'hA2: begin Dout[7:0]=DATA; Dout[15:8]=0;end

8'hA3: Dout = temp3;

8'hA4: Dout = temp4;

8'hA5: Dout = 16'b1000\_0000\_0000\_0101;

default:;

endcase

else

case(Address)

8'hA3: temp3 = Din;

8'hA4: temp4 = Din;

default:;

endcase

Dis = temp4;

end

end

endmodule

**6.6 Program of Display part**

module Display\_001(CLK,DIN,CS,DIS,RST);

input CLK,RST;

input [15:0]DIN;

output reg [7:0]DIS;

output reg [3:0]CS;

reg [1:0] STAT;

reg [3:0] temp;

initial

begin

DIS <= 0;

STAT <= CS\_1;

end

parameter

CS\_1 = 2'b00,

CS\_2 = 2'b01,

CS\_3 = 2'b10,

CS\_4 = 2'b11;

always @ (posedge CLK or posedge RST)

begin

if(RST)

begin

DIS = 0;

STAT = CS\_1;

CS = 4'b1111;

end

else

begin

case(STAT)

CS\_1: begin CS = 4'b1110; temp = DIN[3:0];end

CS\_2: begin CS = 4'b1101; temp = DIN[7:4];end

CS\_3: begin CS = 4'b1011; temp = DIN[11:8];end

CS\_4: begin CS = 4'b0111; temp = DIN[15:12];end

default:;

endcase

case(temp)

4'b0000 : DIS =8'b11000000;

4'b0001 : DIS =8'b11111001;

4'b0010 : DIS =8'b10100100;

4'b0011 : DIS =8'b10110000;

4'b0100 : DIS =8'b10011001;

4'b0101 : DIS =8'b10010010;

4'b0110 : DIS =8'b10000010;

4'b0111 : DIS =8'b11111000;

4'b1000 : DIS =8'b10000000;

4'b1001 : DIS =8'b10010000;

4'b1010 : DIS =8'b10001000;

4'b1011 : DIS =8'b10000011;

4'b1100 : DIS =8'b11000110;

4'b1101 : DIS =8'b10100001;

4'b1110 : DIS =8'b10000110;

4'b1111 : DIS =8'b10001110;

default : DIS =0;

endcase

STAT = STAT + 1;

end

end

endmodule

**6.7 Program of PC part**

module PC\_001(CLK,MBR\_IN,PC\_OUT,CON,RST);

input CLK,RST;

input[15:0] MBR\_IN;

input[31:0] CON;

output reg [7:0] PC\_OUT;

initial

begin

PC\_OUT <= 0;

end

always@(posedge CLK or posedge RST)

begin

if(RST)

begin

PC\_OUT <= 0;

end

else

begin

if(CON[14])

PC\_OUT <= PC\_OUT + 1;

else;

if(CON[2])

PC\_OUT <= MBR\_IN[7:0];

else;

end

end

endmodule

**6.8 Program of ACC part**

module ACC\_001(CON,CLK,ALU\_IN,ACC\_OUT,RST);

input CLK,RST;

input[15:0] ALU\_IN;

input[31:0] CON;

output reg[15:0] ACC\_OUT;

initial

begin

ACC\_OUT <= 0;

end

always@(posedge CLK or posedge RST)

begin

if(RST)

begin

ACC\_OUT <= 0;

end

else

begin

if(CON[9])

ACC\_OUT <= ALU\_IN;

else;

end

end

endmodule

**6.9 Program of ALU part**

module ALU\_001(ACC\_IN,BR\_IN,ALU\_OUT,CON,CLK,zflag,RST);

input CLK,RST;

input signed[15:0] BR\_IN,ACC\_IN;

input [31:0] CON;

output reg zflag;

output reg [15:0] ALU\_OUT;

initial

begin

ALU\_OUT <= 0;

zflag <= 0;

end

always@(posedge CLK or posedge RST)

begin

if(RST)

begin

ALU\_OUT = 0;

zflag = 0;

end

else

begin

if(CON[6]&&CON[7])

begin

case(CON[19:16])

4'b0000: ALU\_OUT = ACC\_IN + BR\_IN;

4'b0001: ALU\_OUT = ACC\_IN - BR\_IN;

4'b0010: ALU\_OUT = ACC\_IN \* BR\_IN;

4'b0011: ALU\_OUT = ACC\_IN / BR\_IN;

4'b0100: ALU\_OUT = ACC\_IN & BR\_IN;

4'b0101: ALU\_OUT = ACC\_IN | BR\_IN;

4'b0110: ALU\_OUT = ~ACC\_IN ;

//4'b0111: ALU\_OUT = BR\_IN;

4'b1000: ALU\_OUT = ACC\_IN << 1;

4'b1001: ALU\_OUT = ACC\_IN >> 1;

//4'b1010: ALU\_OUT = BR\_IN + 1;

//4'b1011: ALU\_OUT = BR\_IN - 1;

default:;

endcase

if(ALU\_OUT==0)

zflag = 1;

else

zflag = 0;

end

else;

end

end

endmodule

**6.10 Program of IR part**

module IR\_001(CLK,CON,MBR\_IN,IR\_OUT,RST);

input CLK,RST;

input[15:0] MBR\_IN;

input[31:0] CON;

output reg [7:0] IR\_OUT;

initial

begin

IR\_OUT <= 0;

end

always@(negedge CLK or posedge RST)

begin

if(RST)

begin

IR\_OUT <= 0;

end

else

begin

if(CON[3])

IR\_OUT <= MBR\_IN[15:8];

else;

end

end

endmodule

**6.11 Program of BR part**

module BR\_001(CON,CLK,MBR\_IN,BR\_OUT,RST);

input CLK,RST;

input[15:0] MBR\_IN;

input[31:0] CON;

output reg[15:0] BR\_OUT;

initial

begin

BR\_OUT <= 0;

end

always@(posedge CLK or posedge RST)

begin

if(RST)

begin

BR\_OUT <= 0;

end

else

begin

if(CON[5])

BR\_OUT<=MBR\_IN;

else;

end

end

endmodule

**6.12 Program of Control Unit part**

6.12.1 Top Design

module Contorl\_Uint\_003(CLK,zflag,IR,CBR,RST);

input CLK,zflag,RST;

input [7:0]IR;

output [31:0]CBR;

wire [7:0]CAR;

wire [31:0]ROM;

wire [31:0]CBR\_CBR;

reg [31:0]DATA;

reg R\_W;

initial

begin

DATA <= 0;

R\_W <= 0;

end

CU\_001 cu(.CLK(CLK),

.IR(IR),

.CAR(CAR),

.CBR(CBR\_CBR),

.ROM(ROM),

.zflag(zflag),

.RST(RST));

Contorl\_Memory\_002 cm\_1(.a(CAR),

.d(DATA),

.clk(CLK),

.we(R\_W),

.spo(ROM));

assign CBR = CBR\_CBR;

endmodule

6.12.2 Program of Control Logic part

module CU\_001(CLK,IR,CAR,CBR,ROM,zflag,RST);

input CLK,zflag,RST;

input [7:0] IR;

input [31:0] ROM;

output reg [7:0] CAR;

output reg [31:0] CBR; //CBR[31:24]---CAR

//CBR[22:20]---mux

//CBR[19:16]---ALU

//CBR[14:0] ---micro-operation

initial

begin

CAR <= 8'h00;

CBR <= 32'h0000\_0010;

end

parameter STORE = 8'b0000\_0001,

LOAD = 8'b0000\_0010,

ADD = 8'b0000\_0011,

SUB = 8'b0000\_0100,

JMPGEZ = 8'b0000\_0101,

JMP = 8'b0000\_0110,

HALT = 8'b0000\_0111,

MPY = 8'b0000\_1000,

DIV = 8'b0000\_1001,

AND = 8'b0000\_1010, //AND--0A

OR = 8'b0000\_1011, //OR --0B

NOT = 8'b0000\_1100, //NOT--0C

SHR = 8'b0000\_1101, //SHR--OD

SHL = 8'b0000\_1110, //SHL--0E

INC = 8'b0000\_1111, //INC--OF

DEC = 8'b0001\_0000;

always@(posedge CLK or posedge RST)

begin

if(RST)

begin

CAR = 8'h00;

CBR = 32'h0000\_0010;

end

else

begin

CBR = ROM; //将控制存储器中的数据传入CBR

case(CBR[22:20])

3'b010: CAR = 0;

3'b100: CAR = 8'h50;

3'b001:begin

case(IR)

STORE: CAR=8'h20;

LOAD: CAR=8'h10;

ADD: CAR=8'h30;

SUB: CAR=8'h40;

DIV: CAR=8'h00;

AND: CAR=8'h60;

OR: CAR=8'h70;

NOT: CAR=8'h80;

SHR: CAR=8'h90;

SHL: CAR=8'hA0;

MPY: CAR=8'hB0;

DIV: CAR=8'hC0;

HALT: CAR=8'h50;

JMP: CAR=8'hF0;

JMPGEZ: if(!zflag)

CAR=8'hF2;

else

CAR=8'hF0;

default: CAR=8'h00;

endcase

end

3'b000: CAR = CAR+1;

default:;

endcase

end

end

endmodule

6.12.3 Program of Control Memory part

module Contorl\_Memory\_002(

a,

d,

clk,

we,

spo

);

input [7 : 0] a;

input [31 : 0] d;

input clk;

input we;

output [31 : 0] spo;

// synthesis translate\_off

DIST\_MEM\_GEN\_V7\_2 #(

.C\_ADDR\_WIDTH(8),

.C\_DEFAULT\_DATA("0"),

.C\_DEPTH(256),

.C\_FAMILY("spartan6"),

.C\_HAS\_CLK(1),

.C\_HAS\_D(1),

.C\_HAS\_DPO(0),

.C\_HAS\_DPRA(0),

.C\_HAS\_I\_CE(0),

.C\_HAS\_QDPO(0),

.C\_HAS\_QDPO\_CE(0),

.C\_HAS\_QDPO\_CLK(0),

.C\_HAS\_QDPO\_RST(0),

.C\_HAS\_QDPO\_SRST(0),

.C\_HAS\_QSPO(0),

.C\_HAS\_QSPO\_CE(0),

.C\_HAS\_QSPO\_RST(0),

.C\_HAS\_QSPO\_SRST(0),

.C\_HAS\_SPO(1),

.C\_HAS\_SPRA(0),

.C\_HAS\_WE(1),

.C\_MEM\_INIT\_FILE("Contorl\_Memory\_002.mif"),

.C\_MEM\_TYPE(1),

.C\_PARSER\_TYPE(1),

.C\_PIPELINE\_STAGES(0),

.C\_QCE\_JOINED(0),

.C\_QUALIFY\_WE(0),

.C\_READ\_MIF(1),

.C\_REG\_A\_D\_INPUTS(0),

.C\_REG\_DPRA\_INPUT(0),

.C\_SYNC\_ENABLE(1),

.C\_WIDTH(32)

)

inst (

.A(a),

.D(d),

.CLK(clk),

.WE(we),

.SPO(spo),

.DPRA(),

.SPRA(),

.I\_CE(),

.QSPO\_CE(),

.QDPO\_CE(),

.QDPO\_CLK(),

.QSPO\_RST(),

.QDPO\_RST(),

.QSPO\_SRST(),

.QDPO\_SRST(),

.DPO(),

.QSPO(),

.QDPO()

);

// synthesis translate\_on

endmodule