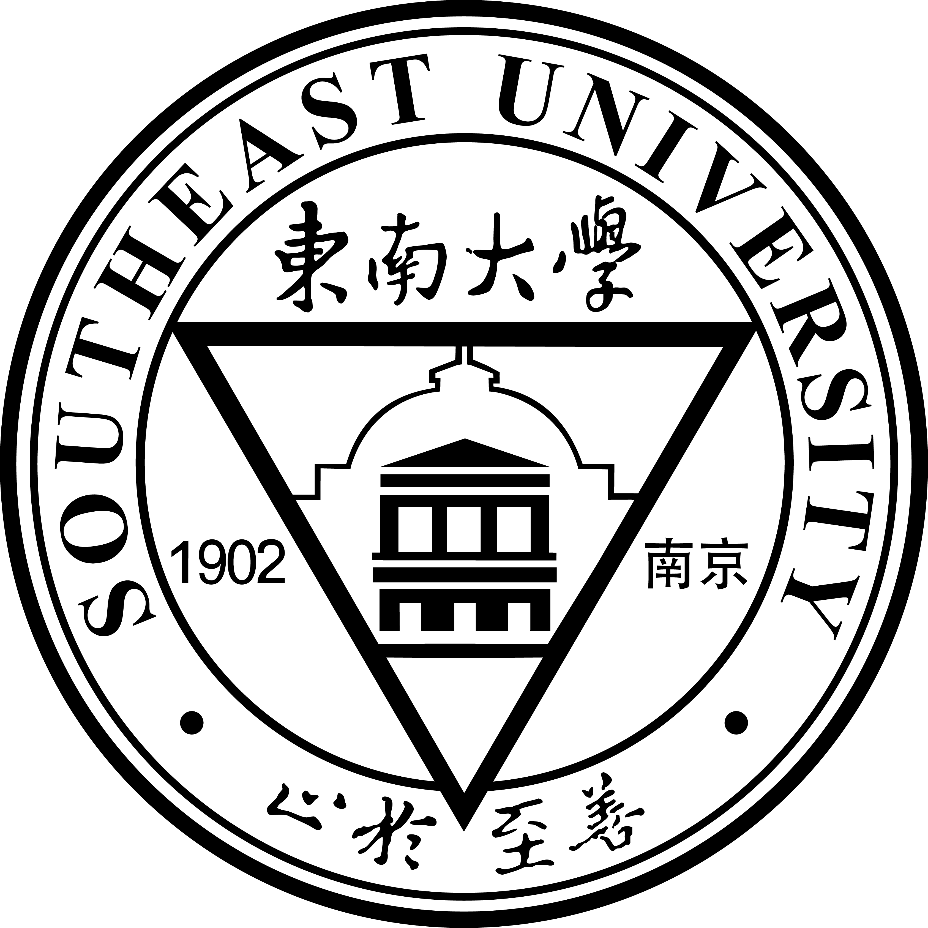
**Computer Organization and Architecture**



**School of Information Science and Engineering**

**Southeast University**

**April 2019**

Microprogrammed CPU Design

Group members :

04016613 Wencheng Qiu

04016640 Jialong Xue

# 1.Purpose

The purpose of this project is to design and verify 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.Tasks

## 2.1 Main 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**

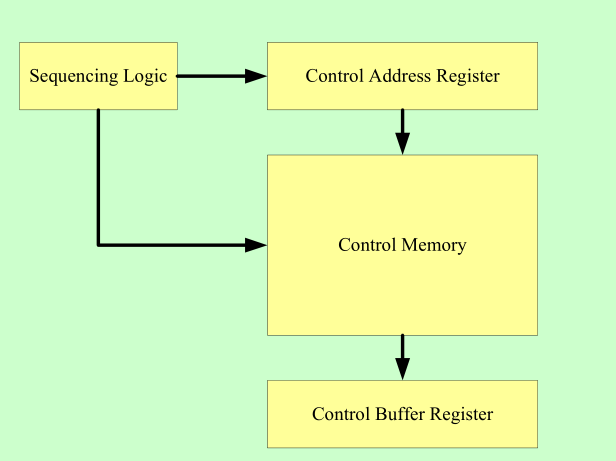
|  |  |
| --- | --- |
| OPCODE  [15…8] | ADDRESS  [7…0] |

**Figure 1** The instruction format

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

**Table 1** List of instructions and relevant opcodes

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



**Figure 2** Control Unit Micro-architecture

**Figure 2** 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.

M

B R

ACC

PC

IR

B

R

ALU

Control signals

M

A R

Control

Unit

Flags

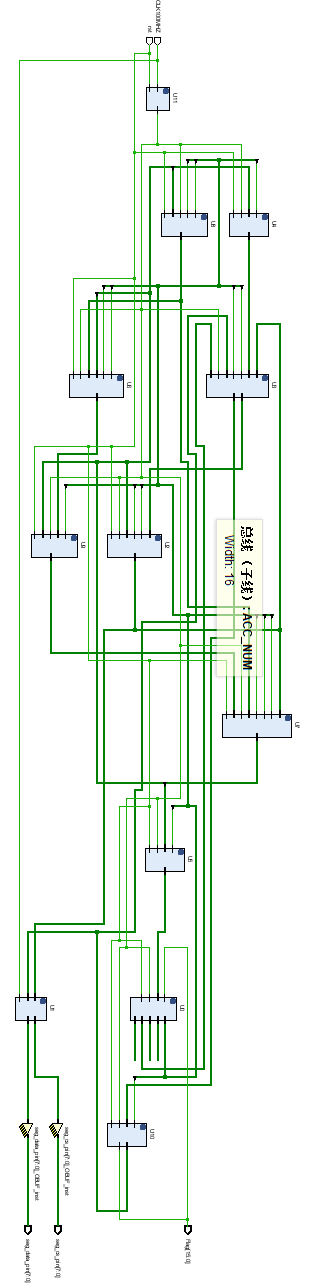
Control

Signals

**Figure 3** CPU data path and control signals

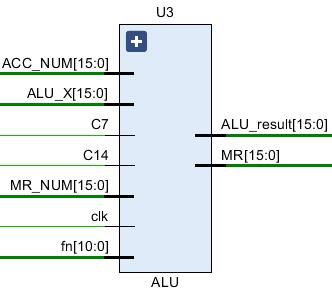
**Figure 3** indicates a simple CPU architecture and its use of a variety of internal data paths and control signals. Our CPU design should be based on this architecture.

# 3.Top module form



# 4. Design for important modules

## 4.1 ALU



**Figure 4-1** The Schematic Symbol of ALU

ALU (Arithmetic Logic Unit) is a calculation unit which accomplishes basic arithmetic and logic operations.

We did not use clock signal in ALU, but adopted combinational logic. The output result adopted a form similar to a MUX, in which fn[], as the control signal transmitted from Control Unit, controls which operation to open and output its result.

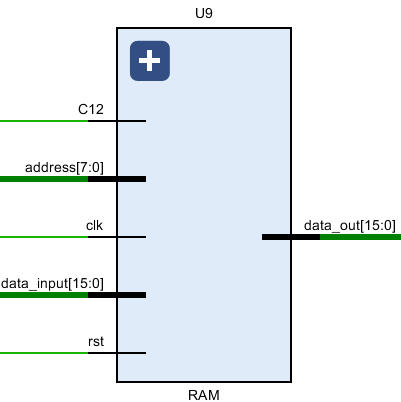
Module Input

|  |  |
| --- | --- |
| Pin | Function |
| C7 | Control signal from AC to ALU |
| ACC\_NUM(15:0) | Data from ACC to ALU |
| C14 | Control signal from BR to ALU |
| ALU\_X | Data from BR to ALU |
| MR\_NUM | Data from MR to ALU |
| fn | Control signals for ALU from Control Unit |

Module Output

|  |  |
| --- | --- |
| Pin | Function |
| ALU\_result | The output result of ALU |
| MR | The high-order result of MPY after overflow |

## 4.2 RAM



**Figure 4-2** The Schematic Symbol of RAM

RAM with separate input and output ports, it works as memory which stores the instructions and data, and its size is 256×16. Although it’s not an internal register of CPU, we need it to simulate and test the performance of CPU.

We use a two-dimensional array that the size is 256×16 to represent the RAM, we write the corresponding assembly statement according to the different program and initialize the value of each unit in RAM for calculation.

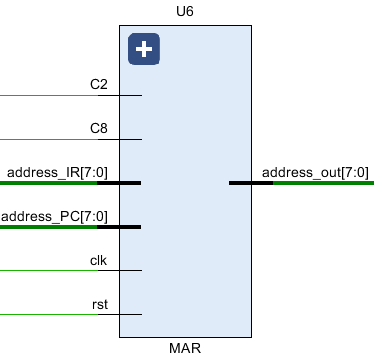
Module Input

|  |  |
| --- | --- |
| Pin | Function |
| C12 | Control signal from MBR to RAM |
| address(7:0) | Address to know the location in RAM |
| clk | Clock signal |
| data\_input(15:0) | Data from MAR/MBR to RAM |
| rst | Reset signal |

Module Output

|  |  |
| --- | --- |
| Pin | Function |
| data\_out(15:0) | Data from RAM to MBR |

## 4.3 MAR (Memory Address Register)



**Figure 4-3** 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.

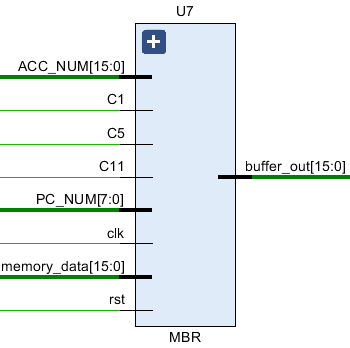
Module Input

|  |  |
| --- | --- |
| Pin | Function |
| C2 | Control signal from PC to MAR |
| C8 | Data from IR to MAR |
| address\_IR(7:0) | The input IR(Addrsss) |
| address\_PC(7:0) | The input program counter |
| clk | Clock signal |
| rst | Reset signal |

Module Output

|  |  |
| --- | --- |
| Pin | Function |
| address\_out(7:0) | The output of PC to MAR |

## 4.4 MBR (Memory Buffer Register)



**Figure 4-4** 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.

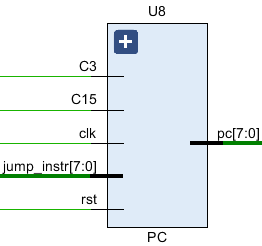
Module Input

|  |  |
| --- | --- |
| Pin | Function |
| ACC\_NUM(15:0) | Data from RAM to MBR |
| C1 | Control signal from PC to MBR |
| C5 | Control signal from RAM to ALU |
| C11 | Control signal from ACC to MBR |
| PC\_NUM(7:0) | Dara from PC to MBR |
| clk | Clock signal |
| memory\_data(15:0) | Data from RAM to MBR |
| rst | Reset signal |

Module Output

|  |  |
| --- | --- |
| Pin | Function |
| Buffer\_out(15:0) | The output of MBR to RAM/ACC/ALU |

## 4.5 PC (Program Counter)



**Figure 4-5** The Schematic Symbol of PC

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

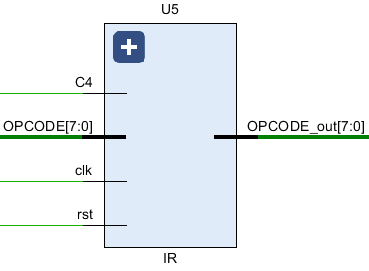
Module Input

|  |  |
| --- | --- |
| Pin | Function |
| C3 | Control signal for conditional jump |
| C15 | Control for PC’s increment |
| clk | Clock signal |
| jump\_instr(7:0) | The location of instruction to jump |
| rst | Reset signal |

Module Output

|  |  |
| --- | --- |
| Pin | Function |
| pc(7:0) | The output of PC to MAR |

## 4.6 IR (Instruction Register)



**Figure 4-6** The Schematic Symbol of IR

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

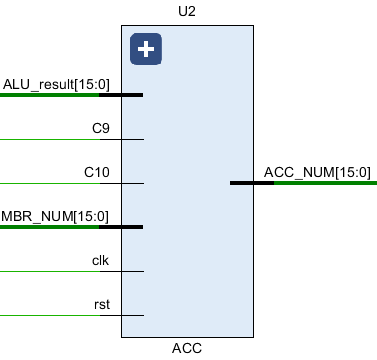
Module Input

|  |  |
| --- | --- |
| Pin | Function |
| C4 | Control signal from MBR to IR |
| OPCODE(7:0) | The input from MBR to IR,including opcode and address |
| clk | Clock signal |
| rst | Reset signal |

Module Output

|  |  |
| --- | --- |
| Pin | Function |
| OPCODE\_out(7:0) | The output of opcode to Control Unit |

## 4.7 ACC (Accumulator)



**Figure 4-7** 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.

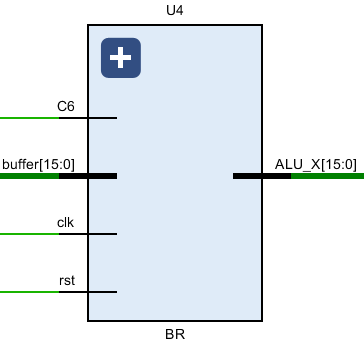
Module Input

|  |  |
| --- | --- |
| Pin | Function |
| ALU\_result(15:0) | The calculating result of ALU to ACC |
| C9 | Control signal from ALU to ACC |
| C10 | Control signal from MBR to ACC |
| MBR\_NUM(15:0) | Data from MBR to ACC |
| clk | Clock signal |
| rst | Reset signal |

Module Output

|  |  |
| --- | --- |
| Pin | Function |
| ACC\_NUM(15:0) | The output of ACC to MBR/ALU |

## 4.8 BR(Buffer Register)



**Figure 4-8** 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.

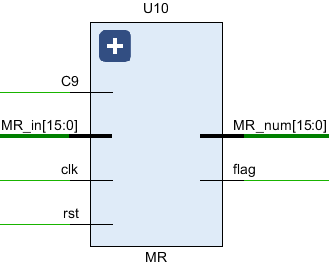
Module Input

|  |  |
| --- | --- |
| Pin | Function |
| C6 | Control signal from MBR to BR |
| buffer(15:0) | Data from MBR to BR |
| clk | Clock signal |
| rst | Reset signal |

Module Output

|  |  |
| --- | --- |
| Pin | Function |
| ALU\_X (15:0) | The output of BR to ALU |

## 4.9 MR(Multiplier Register)



**Figure 4-9** The Schematic Symbol of MR

MR is used for implementing the MPY instruction, holding the multiplier at the beginning of the instruction. When the instruction is executed, it holds part of the product.

Module Input

|  |  |
| --- | --- |
| Pin | Function |
| C9 | Control signal from ALU to ACC |
| MR\_in(15:0) | Data from ALU to MR after the overflow of MPY |
| clk | Clock signal |
| rst | Reset signal |

Module Output

|  |  |
| --- | --- |
| Pin | Function |
| MR\_num (15:0) | The output of MR for further calculation |
| flag | Indicates whether the result is greater than 0 |

# 5.Calculation Process

## 5.1 Problems



## 5.2 programming with C language:

|  |
| --- |
| sum=0;  sum\_1=0;  temp=19;  temp\_1=20;  mul=-13;  loop :sum=sum+temp;  temp=temp-2;  if temp>=0 goto loop;  sum=sum\*mul;  sum=sum>>2;  loop\_1 :sum\_1=sum\_1+temp\_1;  temp\_1=temp\_1-1;  if temp\_1>=0 goto loop\_1;  sum=sum AND sum\_1;  end |

**The result is 146.**

## 5.3 Calculate in assembly language

Assume in the memory:

**sum** is stored at location A5,

**temp** is stored at location A6,

**sum\_1** is stored at location A7,

**temp\_1** is stored at location A8,

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

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

the contents of location A2 is **1910=001316,**

the contents of location A3 is **2010=001416,**

the contents of location A4 **is -1310=FFF316.**

We can translate the above C language program with the instructions listed in **Table 1** into the instruction program as shown in **Table 2**.

**Table 2** The calculate process in assembly language

|  |  |  |  |
| --- | --- | --- | --- |
| Program with C | Program with instructions | Contents of Memory (RAM) in HEX | |
| Address | Contents |
| sum=0; | LOAD A0 | 00 | 02A0 |
| STORE A5 | 01 | 01A5 |
| temp=19; | LOAD A2 | 02 | 02A2 |
| STORE A6 | 03 | 01A6 |
| sum\_1=0; | LOAD A0 | 04 | 02A0 |
| STORE A7 | 05 | 01A7 |
| temp\_1=20; | LOAD A3 | 06 | 02A3 |
| STORE A8 | 07 | 01A8 |
| loop :sum=sum+temp; | LOOP:LOAD A5 | 08 (so LOOP=08) | 02A5 |
| ADD A6 | 09 | 03A6 |
| STORE A5 | 0A | 01A5 |
| temp=temp-2; | LOAD A6 | 0B | 02A6 |
| SUB A1 | 0C | 04A1 |
| SUB A1 | 0D | 04A1 |
| STORE A6 | 0E | 01A6 |
| if temp>=0 goto loop; | JMPGEZ LOOP | 0F | 0508 |
| sum=sum\*mul  sum SHR 2bit | LOAD A5 | 10 | 02A5 |
| MPY A4 | 11 | 08A4 |
| SHR | 12 | 0D00 |
| SHR | 13 | 0D00 |
| STORE A5 | 14 | 01A5 |
| loop :sum\_1=sum\_1+temp\_1; | LOOP:LOAD A7 | 15(so LOOP\_1=15) | 02A7 |
| ADD A8 | 16 | 03A8 |
| STORE A7 | 17 | 01A7 |
| temp\_1=temp\_1-1 | LOAD A8 | 18 | 02A8 |
| SUB A1 | 19 | 04A1 |
| STORE A8 | 1A | 01A8 |
| if temp\_1>=0 goto loop; | JMPGEZ LOOP | 1B | 0515 |
| sum=sum AND sum\_1; | LOAD A5 | 1C | 02A5 |
| AND A7 | 1D | 0AA7 |
| STORE A5 | 1E | 01A5 |
| end | HALT | 1F | 0700 |
|  |  | 20 |  |
|  |  | ….. | ….. |
|  |  | A0 | 0000 |
|  |  | A1 | 0001 |
|  |  | A2 | 0013 |
|  |  | A3 | 0014 |
|  |  | A4 | FFF3 |
|  |  | A5 |  |
|  |  | A6 |  |
|  |  | A7 |  |
|  |  | A8 |  |

# 6.Simulation Results

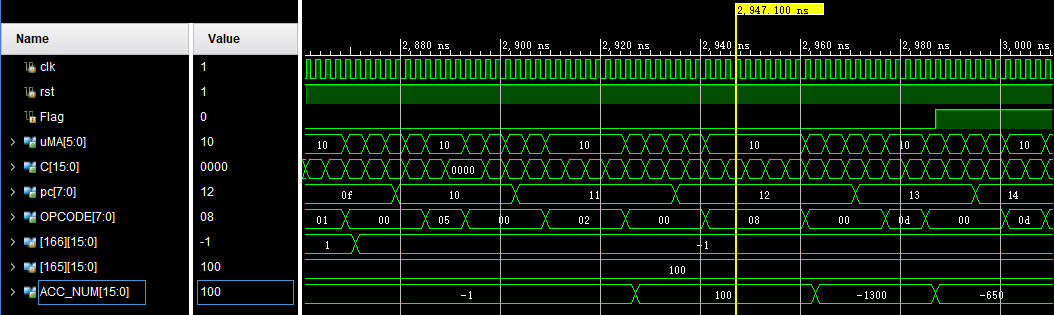
## 6.1 Introduction

The 100MHz internal is divided into 50MHz for the registers, and is divided into 5000Hz for the LED digital tube.

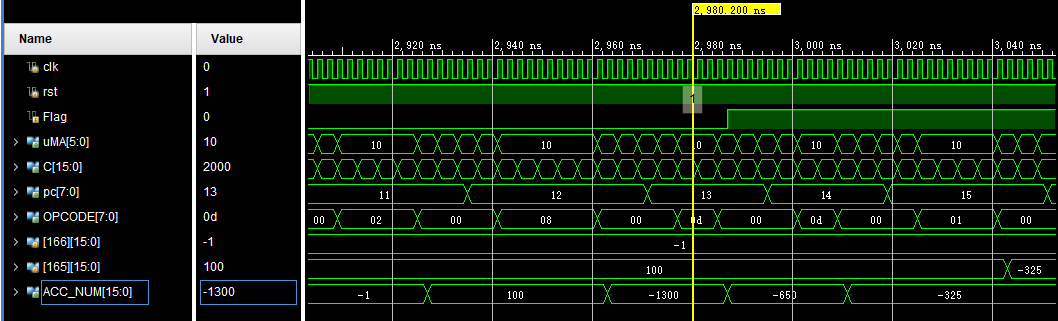
The content of the RAM.

|  |
| --- |
| ram**[**8'h00**]** **=** 16'h02A0**;**  ram**[**8'h01**]** **=** 16'h01A5**;**  ram**[**8'h02**]** **=** 16'h02A2**;**  ram**[**8'h03**]** **=** 16'h01A6**;**  ram**[**8'h04**]** **=** 16'h02A0**;**  ram**[**8'h05**]** **=** 16'h01A7**;**  ram**[**8'h06**]** **=** 16'h02A3**;**  ram**[**8'h07**]** **=** 16'h01A8**;**  ram**[**8'h08**]** **=** 16'h02A5**;**  ram**[**8'h09**]** **=** 16'h03A6**;**  ram**[**8'h0A**]** **=** 16'h01A5**;**  ram**[**8'h0B**]** **=** 16'h02A6**;**  ram**[**8'h0C**]** **=** 16'h04A1**;**  ram**[**8'h0D**]** **=** 16'h04A1**;**  ram**[**8'h0E**]** **=** 16'h01A6**;**  ram**[**8'h0F**]** **=** 16'h0508**;**  ram**[**8'h10**]** **=** 16'h02A5**;**  ram**[**8'h11**]** **=** 16'h08A4**;**  ram**[**8'h12**]** **=** 16'h0D00**;**  ram**[**8'h13**]** **=** 16'h0D00**;**  ram**[**8'h14**]** **=** 16'h01A5**;**  ram**[**8'h15**]** **=** 16'h02A7**;**  ram**[**8'h16**]** **=** 16'h03A8**;**  ram**[**8'h17**]** **=** 16'h01A7**;**  ram**[**8'h18**]** **=** 16'h02A8**;**  ram**[**8'h19**]** **=** 16'h04A1**;**  ram**[**8'h1A**]** **=** 16'h01A8**;**  ram**[**8'h1B**]** **=** 16'h0515**;**  ram**[**8'h1C**]** **=** 16'h02A5**;**  ram**[**8'h1D**]** **=** 16'h0AA7**;**  ram**[**8'h1E**]** **=** 16'h01A5**;**  ram**[**8'h1F**]** **=** 16'h0700**;**  ram**[**8'hA0**]** **=** 16'h0000**;**  ram**[**8'hA1**]** **=** 16'h0001**;**  ram**[**8'hA2**]** **=** 16'h0013**;**  ram**[**8'hA3**]** **=** 16'h0014**;**  ram**[**8'hA4**]** **=** 16'hFFF3**;** |

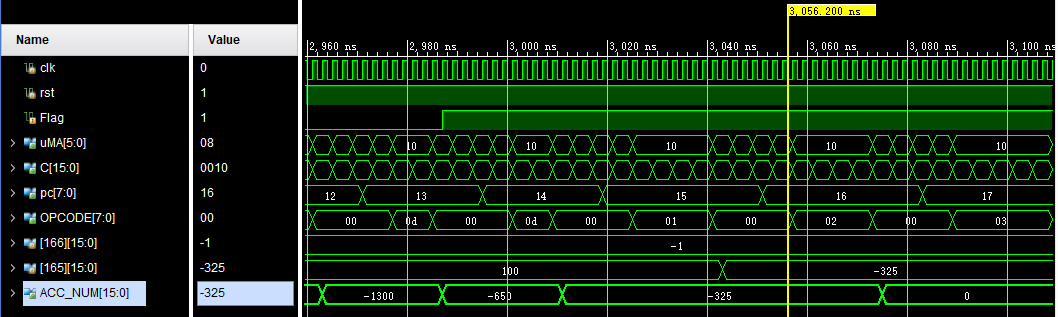
## 6.2 Simulation Result



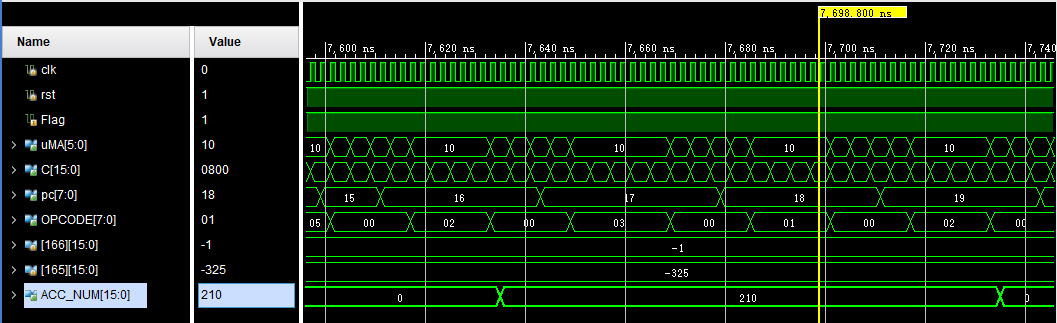
**Figure 6-1** : The result of 



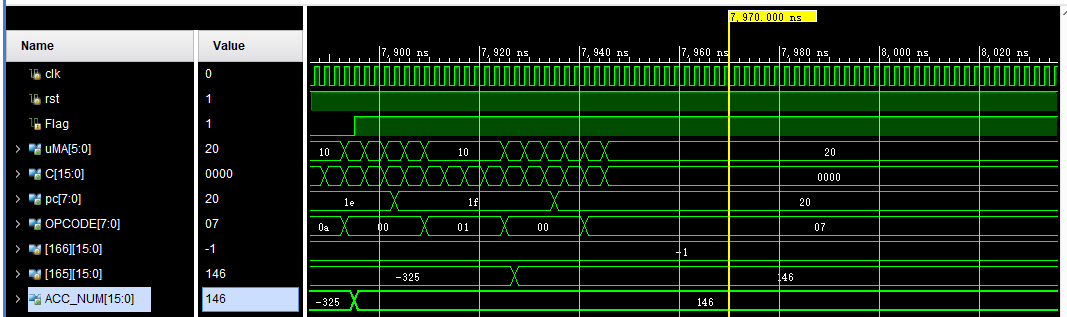
**Figure 6-2** : The result of 



**Figure 6-3**: The result of 

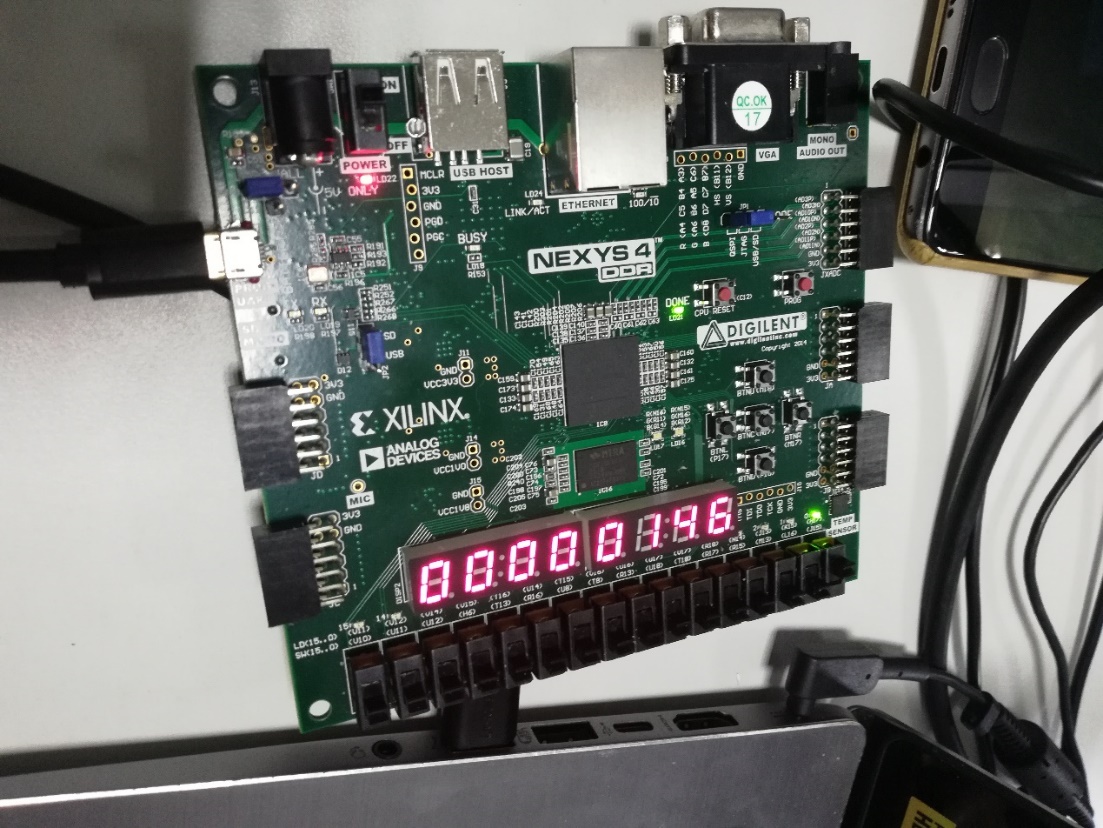


**Figure 6-4**: The result of 



**Figure 6-5**: The result of:

# 7. The result on FPGA



**Figure 7** The result on FPGA

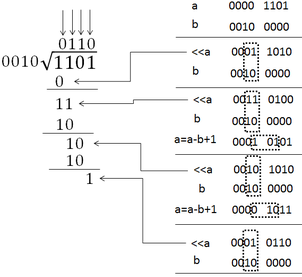
# 8.Conclusions and Discussions

Through this experiment of realizing a CPU on FPGA, we have reviewed the knowledge of microprogram style of computer architecture in COA last semester, using HDL language to generate various registers, the control unit and system bus. We closely combine the theory with practice, and have not only deepen the understanding of knowledge, but also strengthened our ability of writing and debugging codes.

Of course, the experiment process is not well-off. After the building of modules according to the basic concept, we simulated successfully, but still couldn’t produce correct results on FPGA. We looked up much Internet related data, and then found it was due to the use of overdue vivado software, which may automatically optimize various signals and led to the loss of function. According to the proposal on the CSDN, we plus the statement of **(\* dont\_touch = "true"\*)** before each input/output port to prevent this circumstance.

After correcting this error, some calculation results were still incorrect. It was found that the clock frequency was too fast, so the clock of 100MHz was divided once and the correct result was obtained.

In order to improve the design, we have also added division module. The principle is shown in the figure below :(taking 1101/0010 as an example)



The purpose of moving the first four digits left of a is to start from the MSB of itself. If the divisor is larger, then it is subtracted, and it is the remainder left at this moment and the quotient is going to be added to the end of this number, because as long as it's bigger than the divisor, the quotient is going to be 1.

This is a little bit tricky because the quotient can keep going to the left with a, and then the new quotient will keep going to the end. After comparison, we will find that the left and right sides are the remainder and quotient respectively after the 4 times’ shifts.

The relevant division code are as follows:

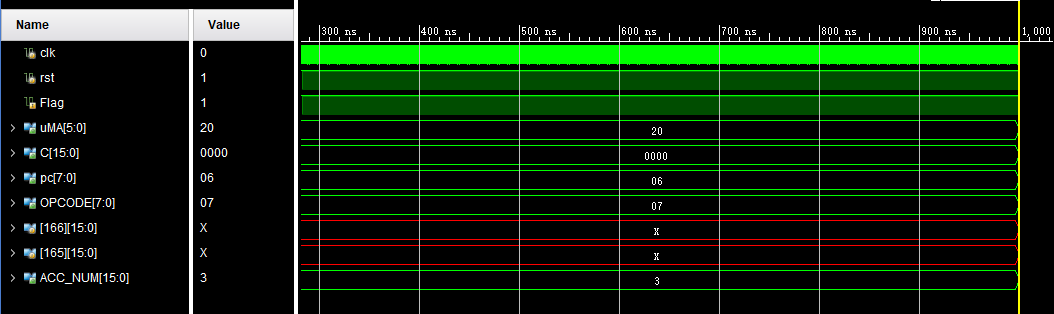
|  |
| --- |
| **always** **@(negedge** clk**)**  **begin**  **if(**fn**[**4**]==**1**)**  **begin**  temp\_a **=** **{**16'b0000000000000000**,**x0**};**  temp\_b **=** **{**y0**,**16'b0000000000000000**};**  **for(**i **=** 0**;**i **<** 16**;**i **=** i **+** 1**)**  **begin**  temp\_a **=** **{**temp\_a**[**30**:**0**],**1'b0**};**  **if(**temp\_a**[**31**:**16**]** **>=** y0**)**  temp\_a **=** temp\_a **-** temp\_b **+** 1'b1**;**  **else**  temp\_a **=** temp\_a**;**  **end**  shang **<=** temp\_a**[**15**:**0**];**  yushu **<=** temp\_a**[**31**:**16**];**  **end**  **end** |

The verification results are as follows :(9/3=3)

RAM contents:

|  |
| --- |
| ram**[**8'h00**]** **=** 16'h02A0**;**  ram**[**8'h01**]** **=** 16'h01A2**;**  ram**[**8'h02**]** **=** 16'h02A2**;**  ram**[**8'h03**]** **=** 16'h09A1**;**  ram**[**8'h04**]** **=** 16'h01A2**;**  ram**[**8'h05**]** **=** 16'h0700**;**  ram**[**8'hA0**]** **=** 16'h0009**;**  ram**[**8'hA1**]** **=** 16'h0003**;** |

The simulation result:



**Figure 8** The simulation result of 9/3=3

All in all, this experiment has benefited us a lot, thanks to the selfless help of teacher and classmates. At the same time, we have seen many shortcomings of our own, in the future we still need to keep learning, expand the scope of knowledge, and make continuous efforts to become a qualified EE engineer.

# 9.Appendix

Because the codes are too long, we give the source codes in the attachment.