Computer Organization and Architecture

计算机组织与架构

Course Design

课程设计

The Experiment Report
实验报告

Of

他说

CPU
CPU

Ⅰ. Purpose

I. 目的

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.***

这个项目的目的是设计一个简单的 CPU (中央处理器)。这个 CPU 有基本的指令集，我们将利用它的指令集生成一个非常简单的程序来验证它的性能。为了简单起见，我们将只考虑 CPU、寄存器、内存和指令集之间的关系。也就是说，我们只需要考虑以下几项: 读/写寄存器，读/写内存和执行指令。

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.

至少有四个部分构成了一个简单的 CPU: 控制单元，内部寄存器，ALU 和指令集，这是我们项目设计的主要方面，并将进行研究。

Ⅱ. Instruction Set

指令集

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*.

在我们简单的 CPU 设计中使用了单地址指令格式。指令字包含两个部分: 操作码(操作码) ，它定义了指令的功能(加法、减法、逻辑操作等)；地址部分，在大多数指令中，地址部分包含要操作的数据的存储位置，我们称之为直接寻址。在一些指令中，地址部分是操作数，称为即时寻址。

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:
为了简单起见，内存的大小是计算机的 256 倍。指令字有 16 位。操作码部分有 8 位，地址部分有 8 位。指令字格式如图 1 所示:

OPCODE 操作密码 [15..0] [15.0]	ADDRESS 地址 [7..0] [7.0]

Figure 1 the instruction format
图 1 指令格式

The opcode of the relevant instructions are listed in Table 1.
表 1 列出了相关指令的操作码。
In Table 1, the notation [x] represents the contents of the location x in the memory. For example, the instruction word *0000001110111001*$_2$ (*03B9*$_{16}$) means that the CPU adds word at location B9$_{16}$ in memory into the accumulator (ACC); the instruction word *0000010100000111*$_2$ (*0507*$_{16}$) means if the sign bit of the ACC (ACC [15]) is 0, the CPU will use the address part of the instruction as the 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.
在表 1 中，符号[x]表示内存中位置 x 的内容。例如，指令字 000001101110012(03B916)意味着 CPU 将位于 b 的单词添加到累加器(ACC)中; 指令字 000010100001112(050716)意味着如果 ACC (ACC [15])的符号位为 0，CPU 将使用指令的地址部分作为下一条指令的地址，如果符号位为 1，CPU 将增加程序计数器(PC)并使用其内容作为下一条指令的地址。

Table 1 List of opcode of the relevant instructions
表 1 相关指令的操作码列表

INSTRUCTION 说明	OPCODE 操作密码	COMMENTS 评论
STORE X 商店 x	01H 01H	ACC→[X] ACC →[x]
LOAD X 负载 x	02H 02H	[X]→ACC 【 x 】→ ACC
ADD X ADD x	03H 03H	ACC+[X]→ACC ACC + [x]→ ACC
SUB X SUB x	04H 04 小时	ACC-[X]→ACC ACC-[x]→ ACC
JMPGZ X JMPGZ x	05H 05H	IF ACC>0 THEN X→PC ELSE PC+1→PC 如果 ACC > 0，则 x → PC ELSE PC + 1→ PC
AND X 和 x	06H 06H	ACC and [X]→ACC ACC 和[x]→ ACC
OR X 或者 x	07H 07 小时	ACC or [X]→ACC ACC 或[x]→ ACC
NOT X 不是 x	08H 08H	Not [X]→ACC 不是[x]→ ACC
SHIFTR X SHIFTR x	09H 09H	SHIFL ACC to RIGHT 1 bit, Logic Shift 逻辑移位到右 1 位
SHIFTL X	0AH	SHIFT ACC to LEFT 1 bit, Logic Shift

Shift x	0AH	SHIFT ACC to LEFT 1 bit，Logic SHIFT ACC 到左 1 位，逻辑移位
MPY X MPY x	0BH 0BH	ACC×[X]→ACC ACC × [x]→ ACC
HALT 立定	0CH 哦，天啊	HALT A PROGRAM 暂停一个程序

A program is designed to test these instructions:
一个程序被设计用来测试这些指令:
Calculate the sum of all integers from 1 to 100.
计算从 1 到 100 的所有整数之和。
(1), programming with C language:
(1)用 c 语言编程:

```
sum=0;
总和 = 0;
temp=100;
温度 = 100;
loop :sum=sum+temp;
循环: sum = sum + temp;
      temp=temp-1;
      Temp = temp-1;
      if temp>=0 goto loop;
      如果 temp > = 0 goto 循环;
end
结束
```

(2), Assume in the RAM_DQ:
(2) ，在 RAM _ dq 中假设:

sum is stored at location **A4**,
和存储在位置 A4,
temp is stored at location **A3**,
温度存储在位置 A3,
the contents of location **A0** is **0**,
位置 a0 的内容是 0,
the contents of location **A1** is **1,**
位置 a1 的内容是 1,
the contents of location **A2** is **100_{10}=64_{16}.**
位置 a2 的内容是 10010 = 6416。

We can translate the above C language program with the instructions listed in Table 1 into the instruction program as shown in Table 2.
我们可以用表 1 中列出的指令将上面的 c 语言程序翻译成表 2 中列出的指令程序。

Table 2 Example of a program to sum from 1 to 100
表 2 从 1 到 100 求和的程序示例

Program with C 用 c 语言编写程序	Program with 程序	Contents of RAM_DQ in HEX 十六进制中 RAM _ dq 的内容

	instructions 说明书	Address 地址	Contents 目录

sum=0;
总和 = 0;
	LOAD A0
	装载 A0
		00	02A0

	STORE A4 A4 商店	01	01A4

temp=100
Temp = 100
	LOAD A2
	装载 A2
		02	02A2

	STORE A3 A3 商店	03	01A3

loop:sum=sum+temp;
循环: sum = sum + temp;
	:LOAD A4
	翻译: LOAD A4
		04	02A4

	ADD A3 添加 A3	05	03A3
	STORE A4 A4 商店	06	01A4

temp=temp-1;
Temp = temp-1;
	LOAD A3
	装载 A3
		07	02A3

	SUB A1 SUB A1	08	04A1
	STORE A3 A3 商店	09	01A3

if temp>0 goto loop;
如果温度 >0，则循环;
	JMPGZ
	JMPGZ
		0A	0504

end;
完;
	HALT
	立定
		0B
		0B
			0C00

		0C	
		…	…
		A0 A0	0000
		A1 答 1	0001
		A2 A2	0064
		A3 A3	
		A4 A4	
		…	…

III. Internal Registers and Memory

III. 内部寄存器和存储器

***MAR** (Memory Address Register)*
内存地址寄存器
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.
MAR 包含要从内存中读取或写入内存的单词的内存位置。在这里，READ 操作被表示为 CPU 从内存读取，WRITE 操作被表示为 CPU 写入内存。在我们的设计中，MAR 有 8 位来访问 256 个内存地址中的一个。

***MBR** (Memory Buffer Register)*
记忆体缓冲寄存器
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.
MBR 包含要存储在内存中的值或从内存中读取的最后一个值。MBR 连接到系统总线的地址行。在我们的设计中，MBR 有 16 个 bit。

***PC** (Program Counter)*
计算机(程序计数器)
PC keeps track of the instructions to be used in the program. In our design, PC has 8 bits.
PC 跟踪程序中使用的指令。在我们的设计中，PC 有 8 位。

***IR** (Instruction Register)*
指令寄存器
IR contains the opcode part of an instruction. In our design, IR has 8 bits.
IR 包含指令的操作码部分。在我们的设计中，IR 有 8 位。

***BR** (Buffer Register)*
缓冲寄存器
BR is used as an input of ALU, it holds other operand for ALU. In our design, BR has
BR 被用作 ALU 的输入，它包含 ALU 的其他操作数。在我们的设计中，BR 包含
16 bits.
16 位。

LPM_RAM_DQ
LPM _ ram _ dq
LPM_RAM_DQ is a RAM with separate input and output ports. It works as a memory, 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.
LPM _ RAM _ dq 是一个具有独立输入和输出端口的 RAM。它作为一个内存工作，其大小为 256 × 16。虽然它不是 CPU 的内部寄存器，但是我们需要它来模拟和测试 CPU 的性能。

LPM_ROM
LPM _ rom
LPM_ROM is a ROM with one address input port and one data output port, and its size of data is 32bits which contains control signals to execute micro-operations.
ROM 是一个只有一个地址输入端口和一个数据输出端口的 ROM，其数据大小为 32 位，包含执行微操作的控制信号。

IV.ALU

逻辑运算器(IV.ALU)

ALU (Arithmetic Logic Unit) is a calculation unit which accomplishes basic arithmetic and logic operations. In our design, some operations must be supported which are listed as follows:
算术逻辑单元(ALU)是完成基本算术和逻辑运算的计算单元。在我们的设计中，必须支持以下一些操作:

Table 3 ALU Operations
表 3 ALU 操作

ALU control signal ALU 控制信号	Operations 操作	Explanations 解释

3H
3小时
	ADD
	注意缺陷障碍
		ACC←ACC+BR
		ACC ← ACC + br

4H 4小时	SUB SUB 小组	ACC←ACC- BR ← ACC-BR

6H
6小时
	AND
	还有
		ACC←ACC and BR
		ACC ← ACC and BR

7H
七小时
	OR
	或者
		ACC←ACC or BR
		ACC ← ACC 或 BR

8H 8小时	NOT 不是	ACC←not ACC ←不是 ACC
9H 9小时	SHIFTR SHIFTR	ACC←Shift ACC to Right 1 bit ←将 ACC 向右移动1位
0AH 0AH	SHIFTL SHIFTL	ACC←Shift ACC to Left 1 bit ← Shift ACC to Left 1 bit

V. Micro-programmed Control Unit

微程序控制单元

In the Microprogrammed control, the microprogram consists of some microinstruction 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.

在微程序控制中，微程序由一些微指令组成，微程序存储在控制存储器中，产生正确执行指令集所需的所有控制信号。微指令包含一些同时执行的微操作。

Figure 2 shows the key elements of such an implementation.

图 2 显示了这种实现的关键元素。

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.

微指令集存储在控制存储器中。控制地址寄存器包含下一个要读取的微指令的地址。当一条微指令从控制存储器读取时，它被传输到一个控制缓冲寄存器。寄存器连接到从控制单元发出的控制线。因此，从控制存储器读取微指令与执行微指令是一样的。图中显示的第三个元素是一个序列单元，它加载控制地址寄存器并发出读取命令。

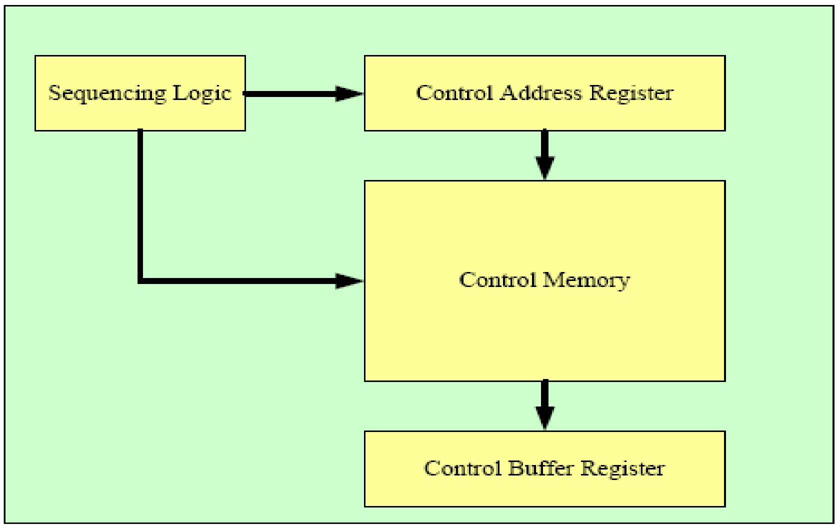


Figure 2 Control Unit Micro-architecture

图 2 控制单元微结构

(I)Total control signals for instructions are listed as follows:

(i)用于指令的总控制信号如下:

Table 4 Control signals for the micro-operations

表 4 微操作的控制信号

Bits in Control Memory Bit in Control Memory 控制内存中的位	Micro-operation 微操作	Meaning 意义
C0~C7 C0 ~ C7	/	Branch Addresses 分支地址

C8
C8
	PC←0
	PC ←0
		Clear PC
		清除电脑

C9
C9
	PC←PC+1
	PC ← PC + 1
		Increment PC
		Increment PC 增量 PC

C10
C10
	PC←MBR[7..0]
	PC ← MBR [7.0]
		MBR[7..0] to PC
		MBR [7.0]到 PC

C11	ACC←0	Clear ACC

C11	ACC ←0	清晰的 ACC
C12--C15 C12 -- C15	ALU CONTROL ALU CONTROL ALU CONTROL	Control operations of ALU ALU 的控制操作
C16 C16	R	Read data from Memory to MBR 将数据从内存读取到 MBR
C17 C17	W	Write data to Memory 将数据写入内存

C18
C18
	MAR←MBR[7..0]
	MAR ← MBR [7.0]
		MBR[7..0] to MAR as address
		MBR [7.0]到 MAR 作为地址

C19
C19
	MAR←PC
	MAR ← PC
		PC value to MAR
		个人计算机对 MAR 的价值

C20
C20
	MBR←ACC
	MBR ← ACC
		ACC value to MBR
		对 MBR 的 ACC 值

C21
C21
	IR←MBR[15..8]
	IR ← MBR [15.8]
		MBR[15..8] to IR as opcode
		MBR [15.8]到 IR 作为操作码

C22 C22	BR←MBR ← MBR	Copy MBR to BR 将 MBR 复制到 BR

C23
C23
	CAR←CAR+1
	CAR ← CAR + 1
		Increment CAR
		增量汽车

C24
C24
	CAR←C0~C7
	CAR ← C0 ~ C7
		C7~C0 to CAR
		C7 ~ c0 到 CAR

C25
C25
	CAR←OPCODE+CAR
	CAR ← opcode + CAR
		Add OP to CAR
		向 CAR 添加 OP

C26
C26
	CAR←0
	CAR ←0
		Reset CAR
		重置汽车

C27--C31 C27 -- C31	Not use 不使用	----------- -----------

(II)The contents in rom.mif and the corresponding microprograms are listed as follows:
(II) rom.mif 中的内容和相应的微程序如下:

0 : 00810000; R←1, CAR←CAR+1 1 : 00000; OP←MBR[15..8],CAR←CAR+1

0:00810000; r ←1，CAR ← CAR + 1:00000; OP ← MBR [15。. 8] ，CAR ← CAR + 1

2 : 02000000; CAR←CAR+OP

2:02000000; CAR ← CAR + op

3 : 01000014; CAR←14H

3:01000014; CAR ←14H

4 : 01000019; CAR←19H

4:010000019; CAR ←19H

5 : 0100001E; CAR←1EH

5:0100001E; CAR ←1EH

6 : 01000023; CAR←23H

6:010000023; CAR ←23H

7 : 01000041; CAR←41H

7:010000041; CAR ←41H

8 : 01000028; CAR←28H

8:010000028; CAR ←28H

9 : 0100002D; CAR←2DH

9:0100002D; CAR ←2DH

a : 01000032; CAR←32H

A: 01000032; CAR ←32H

b : 01000037; CAR←37H

B: 010000037; CAR ←37H

c : 0100003C; CAR←3CH

C: ; CAR ←3CH

d : 01000046; CAR←46H

D: 01000046; CAR ←46H

e : 0100004B; CAR←4H

E: 0100004B; CAR ←4H

f : 00000000;

F: 00000000;

… ……

... .

14 : 00840000; MAR←MBR[7..0], CAR←CAR+1 ------STORE 15 : 00920200; MBR←ACC, PC←PC+1,W←1,CAR←CAR+1

14:00840000; MAR ← MBR [7.0] ，CAR ← CAR + 1—— STORE 15:00920200; MBR ← ACC，PC ← PC + 1，w ←1，CAR ← CAR + 1

16 : 04080000; CAR←0

16:04080000; CAR ←0

17 : 00000000;

17:00000000;

18 : 00000000;

18:0000000;

19 : 00840000; MAR←MBR[7..0], CAR←CAR+1 ------LOAD : 00; PC ←PC+1,R←1,ACC←0,CAR←CAR+1 1b : 03000; BR←MBR,ACC←ACC+BR, CAR←CAR+1

19:00840000; MAR ← MBR [7。. 0] ，CAR ← CAR + 1—— LOAD: 00; PC ← PC + 1，r ←1，ACC ←0，CAR ← CAR + 11b: 03000; BR ← MBR，ACC ← ACC + BR，CAR ← CAR + 1

1c : 04080000; CAR←0

: 04080000;

1d : 00000000;

1 d: 00000000;

1e : 00840000; MAR←MBR[7..0], CAR←CAR+1 ----------ADD : 00810200; PC←PC+1,R←1,CAR←CAR+1 20 : 03000; BR←MBR,ACC←ACC+BR, CAR←CAR+1

1e: 00840000; MAR ← MBR [7.0] ，CAR ← CAR + 1———— ADD: 00810200; PC ← PC + 1，r ←1，CAR ← CAR + 120:03000; BR ← MBR，ACC ← ACC + BR，CAR ← CAR ← CAR + 1

21 : 04080000; CAR←0

21:04080000; CAR ←0

22 : 00000000;
22:00000000;
23 : 00840000; MAR←MBR[7..0], CAR←CAR+1 ----------SUB 24 : 00810200; PC←PC+1,R←1,CAR←CAR+1 25 : 04000; BR←MBR,ACC←ACC-BR, CAR←CAR+1
23:00840000; MAR ← MBR [7.0] ，CAR ← CAR + 1———— SUB 24:00810200; PC ← PC + 1，r ←1，CAR ← CAR + 125:04000; BR ← MBR，ACC ← ACC-BR，CAR ← CAR + 1
26 : 04080000; CAR←0
26:04080000; CAR ←0
27 : 00000000;
27:00000000;
28 : 00840000; MAR←MBR[7..0], CAR←CAR+1 ---------AND 29 : 00810200; PC←PC+1,R←1,CAR←CAR+1 : 06000; BR←MBR,ACC←ACC AND BR,CAR←CAR+1
28:00840000; MAR ← MBR [7。. 0] ，CAR ← CAR + 1———— AND 29:00810200; PC ← PC + 1，r ←1，CAR ← CAR + 1:06000; BR ← MBR，ACC ← ACC AND BR，CAR ← CAR + 1
2b : 04080000; CAR←0
2b: 04080000; CAR ←0
2c : 00000000;
: 00000000;
2d : 00840000; MAR←MBR[7..0], CAR←CAR+1 ---------OR 2e : 00810200; PC←PC+1,R←1,CAR←CAR+1 : 07000; BR←MBR,ACC←ACC OR BR, CAR←CAR+1
2d: 00840000; MAR ← MBR [7。. 0] ，CAR ← CAR + 1—— -- OR 2e: 00810200; PC ← PC + 1，r ←1，CAR ← CAR + 1:07000; BR ← MBR，ACC ← ACC OR BR，CAR ← CAR + 1
30 : 04080000; CAR←0
30:04080000; CAR ←0
31 : 00000000;
31:00000000;
32 : 00840000; MAR←MBR[7..0], CAR←CAR+1 ----------NOT 33 : 00808200; PC←PC+1, ACC←NOT ACC,CAR←CAR+1
32:00840000; MAR ← MBR [7.0] ，CAR ← CAR + 1—— -- NOT 33:00808200; PC ← PC + 1，ACC ← NOT ACC，CAR ← CAR + 1
34 : 04080000; CAR←0
34:04080000; CAR ←0
35 : 00000000;
35:00000000;
36 : 00000000;
36:00000000;
37 : 00840000; MAR←MBR[7..0], CAR←CAR+1 ----------SHIFTR 38 : 08092000; PC←PC+1, ACC←SHIFT ACC to Right 1 bit,CAR←CAR+1
37:00840000; MAR ← MBR [7。. 0] ，CAR ← CAR + 1———— SHIFTR 38:08092000; PC ← PC + 1，ACC ← SHIFT ACC 到右 1 位，CAR ← CAR + 1
39 : 04080000; CAR←0
39:04080000; CAR ←0

3a : 00000000;

: 00000000;

3b : 00000000;

3b: 00000000;

3c : 00840000; MAR←MBR[7..0], CAR←CAR+1 -----------SHIFTL 3d : 200; PC←PC+1, ACC←SHIFT ACC to Left 1 bit,CAR←CAR+1

: 00840000; MAR ← MBR [7.0] ，CAR ← CAR + 1———— SHIFTL 3d: 200; PC ← PC + 1，ACC ← SHIFT ACC to Left 1 bit，CAR ← CAR + 1

3e : 04080000; CAR←0

3e: 04080000; CAR ←0

3f : 00000000;

: 00000000;

40 : 00000000;

40:00000000;

41 : 00840000; MAR←MBR[7..0], CAR←CAR+1 -----------JMPGEZ

41:00840000; MAR ← MBR [7.0] ，CAR ← CAR + 1—————— JMPGEZ

42 : 00805000; CAR←CAR+1,

42:00805000 CAR ← CAR + 1,

43 : 04080000; CAR←0

43:04080000; CAR ←0

44 : 00000000;

44:00000000;

45 : 00000000;

45:00000000;

46 : 00840000; MAR←MBR[7..0], CAR←CAR+1 ------------MPY 47 : 00810200; PC←PC+1,R←1,CAR←CAR+1 48 : 0B000; BR←MBR,ACC←ACC*BR, CAR←CAR+1

46:00840000; MAR ← MBR [7.0] ，CAR ← CAR + 1————— MPY 47:00810200; PC ← PC + 1，r ←1，CAR ← CAR + 148:0B000; BR ← MBR，ACC ← ACC * BR，CAR ← CAR + 1

49 : 04080000; CAR←0

49:04080000; CAR ←0

4a : 00000000;

: 00000000;

4b : 0100004B; CAR←4BH -----------------------------HALT

4b: 0100004B; CAR ←4BH ————————————————————停

4c : 00000000;

: 00000000;

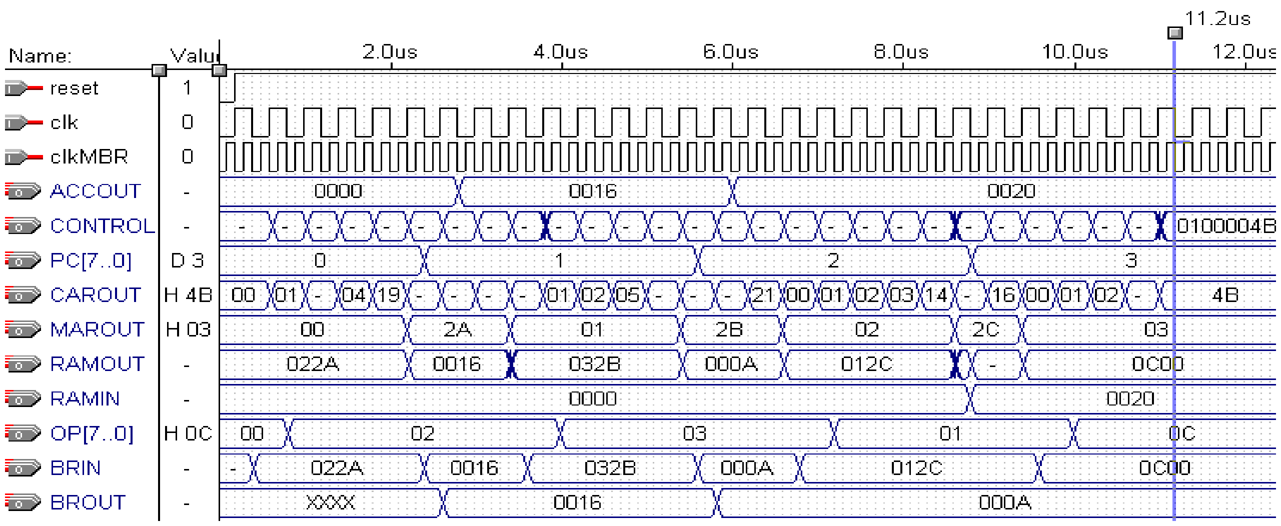
(IIa) CPU电路图

(IIa) cpu 电子路径图

(III)The simulation waveforms of some operates
(III)某些操作的模拟波形
1, load, add, store, halt (22+10)
1，加载，添加，存储，暂停(22 + 10)
The contents in RAM:
内存中的内容:

0 : 022A; Load 2A
加载
1 : 032B; ADD 2B
1:032B; ADD 2B
2 : 012C; Store 2C
2: ; 商店
3 : 0C00; Halt
3:00; Halt
2a : 0016;
: 0016;
2b : 000A;
二乙、

The content in RAM addressed of 2b is **0020(H)**.
RAM 中地址为 2b 的内容是 0020(h)。
The waveform of the operate:
操作的波形:



11.2us
Name:
Value
2.0us
4.0us
6.0us
8.0us
10.0us
12.0us
reset 1
clk 0
clkMBR 0
ACCOUT - 0000 0016 0020
CONTROL - 0100004B
PC[7..0] D 3 0 1 2 3
CAROUT H 4B 00 01 04 19 01 02 05 21 00 01 02 03 14 16 00 01 02 4B
MAROUT H 03 00 2A 01 2B 02 2C 03
RAMOUT - 022A 0016 032B 000A 012C 0C00
RAMIN - 0000 0020
OP[7..0] H 0C 00 02 03 01 0C
BRIN - 022A 0016 032B 000A 012C 0C00
BROUT - XXXX 0016 000A

程序1波形图及逐条指令、逐个阶段、逐个操作、逐个部件逐个数据变化的对应关系分析，数据运算过程分析。
程序 1 波形图及条件逐项逐项逐项逐项变化的对应关系分析、数据运算过程分析。
详细分析可任选一条指令为例，必须结合测试波形、电路图、代码、指令系统设计具体说明，标注控制信号，地址，数据等信息的每一次流动、变化。

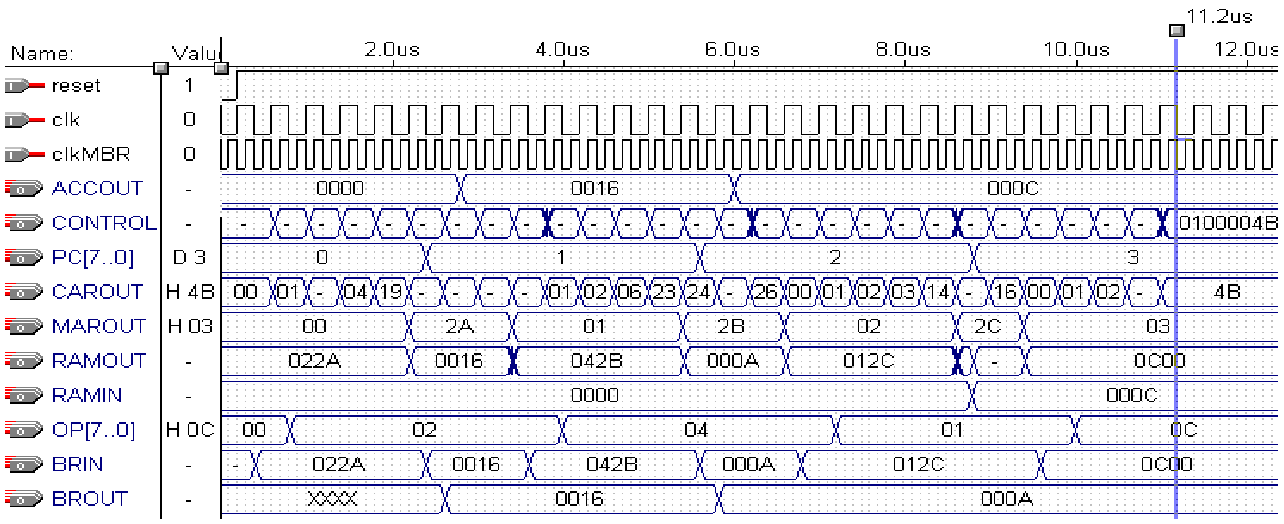
详细分析可任命选举细则试验必须结合波形图、电子路径图、测量系统设计指标具体说明、标准控制号、地址数字等息的每一次流动信息化。
（必做）
(必要做法)

2, load, SUB, store, halt (22-10)
2，装载，SUB，存储，停止(22-10)

The contents in RAM:
内存中的内容:

    0 : 022A; Load 2A
加载
    1 : 042B; SUB 2B
1:042B; SUB 2B
    2 : 012C; Store 2C
2: ; 商店
    3 : 0C00; Halt
3:00; Halt
    2a : 0016;
    : 0016;
    2b : 000A;
    二乙、

The content in RAM addressed of 2c is **000C(H)**.
RAM 中寻址的内容是(h)。
The waveform of the operate:
操作的波形:

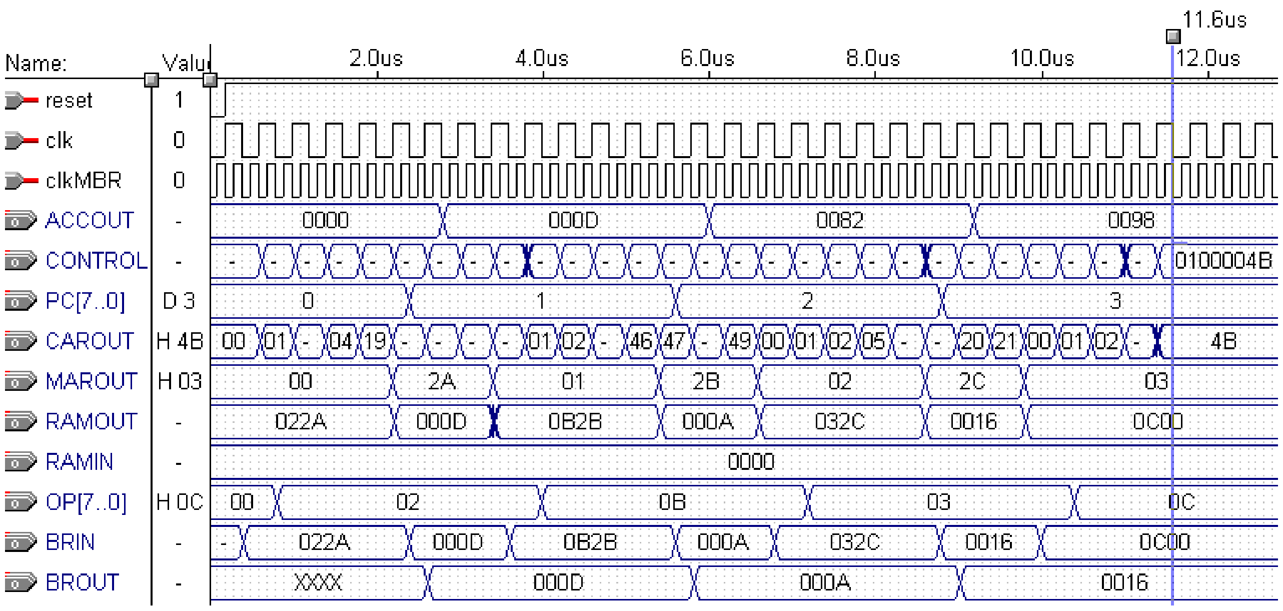


11.2us
Name:
Valu
2.0us
4.0us
6.0us
8.0us
10.0us
12.0us
reset 1
clk 0
clkMBR 0
ACCOUT - 0000 0016 000C
CONTROL - 0100004B
PC[7..0] D 3 0 1 2 3
CAROUT H 4B 00 01 04 19 01 02 06 23 24 26 00 01 02 03 14 16 00 01 02 4B
MAROUT H 03 00 2A 01 2B 02 2C 03
RAMOUT - 022A 0016 042B 000A 012C 0C00
RAMIN - 0000 000C
OP[7..0] H 0C 00 02 04 01 0C
BRIN - 022A 0016 042B 000A 012C 0C00
BROUT - XXXX 0016 000A

程序2波形图及逐条指令、逐个阶段、逐个操作、逐个部件逐个数据变化的对应关系分析，数据运算过程分析。(可选做)
程序 2 波形图及段、个部分逐步逐步系统变化的对应关系分析、数据运算过程分析(可选)

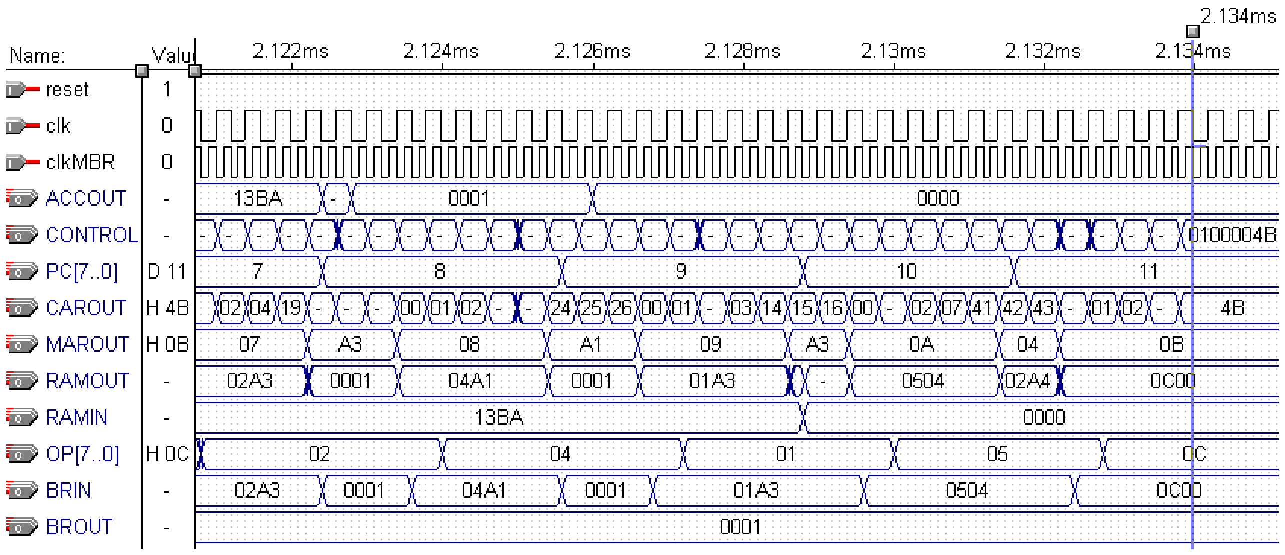
3, load, mpy, add, store, halt (13*10+22)

3，装载，卸载，添加，存储，停止(13 * 10 + 22)
The contents in RAM:
内存中的内容:
0 : 022A; Load 2A
加载
1 : 0B2B; MPY 2B
1:0B2B; MPY 2B
2 : 032C; ADD 2C
2: ; ADD
3 : 012D; Store 2D
3:012D 商店 2d
4 : 0C00; Halt
4:00; 立定
2a : 000D;
: 000D;
2b : 000A;
二乙、
2c : 0016;
: 0016;
The content in RAM addressed of 2d is **0098(H)**.
RAM 中 2d 的内容是 0098(h)。
The waveform of the operate:
操作的波形:


11.6us
Name:
Value
2.0us
4.0us
6.0us
8.0us
10.0us
12.0us
reset 1
clk 0
clkMBR 0
ACCOUT - 0000 000D 0082 0098
CONTROL - 0100004B
PC[7..0] D 3 0 1 2 3
CAROUT H 4B 00 01 04 19 01 02 46 47 49 00 01 02 05 20 21 00 01 02 4B
MAROUT H 03 00 2A 01 2B 02 2C 03
RAMOUT - 022A 000D 0B2B 000A 032C 0016 0C00
RAMIN - 0000
OP[7..0] H 0C 00 02 0B 03 0C
BRIN - 022A 000D 0B2B 000A 032C 0016 0C00
BROUT - XXXX 000D 000A 0016
程序3波形图及逐条指令、逐个阶段、逐个操作、逐个部件逐个数据变化的对应关系分析，数据运算过程分析。(可选做)
程序 3 波形图及其段落逐个作业个部分逐个系统逐步变化的对应关系分析、数据运算过程分析(可选)

4, Sum from 1 to 100
4，Sum 从 1 到 100

The contents in RAM are shown in table2.
RAM 中的内容如表 2 所示。
The content in RAM addressed of A4 is **13BA(H)**.
A4 所处理的 RAM 中的内容是 13ba (h)。
The waveform of the operate:
操作的波形:



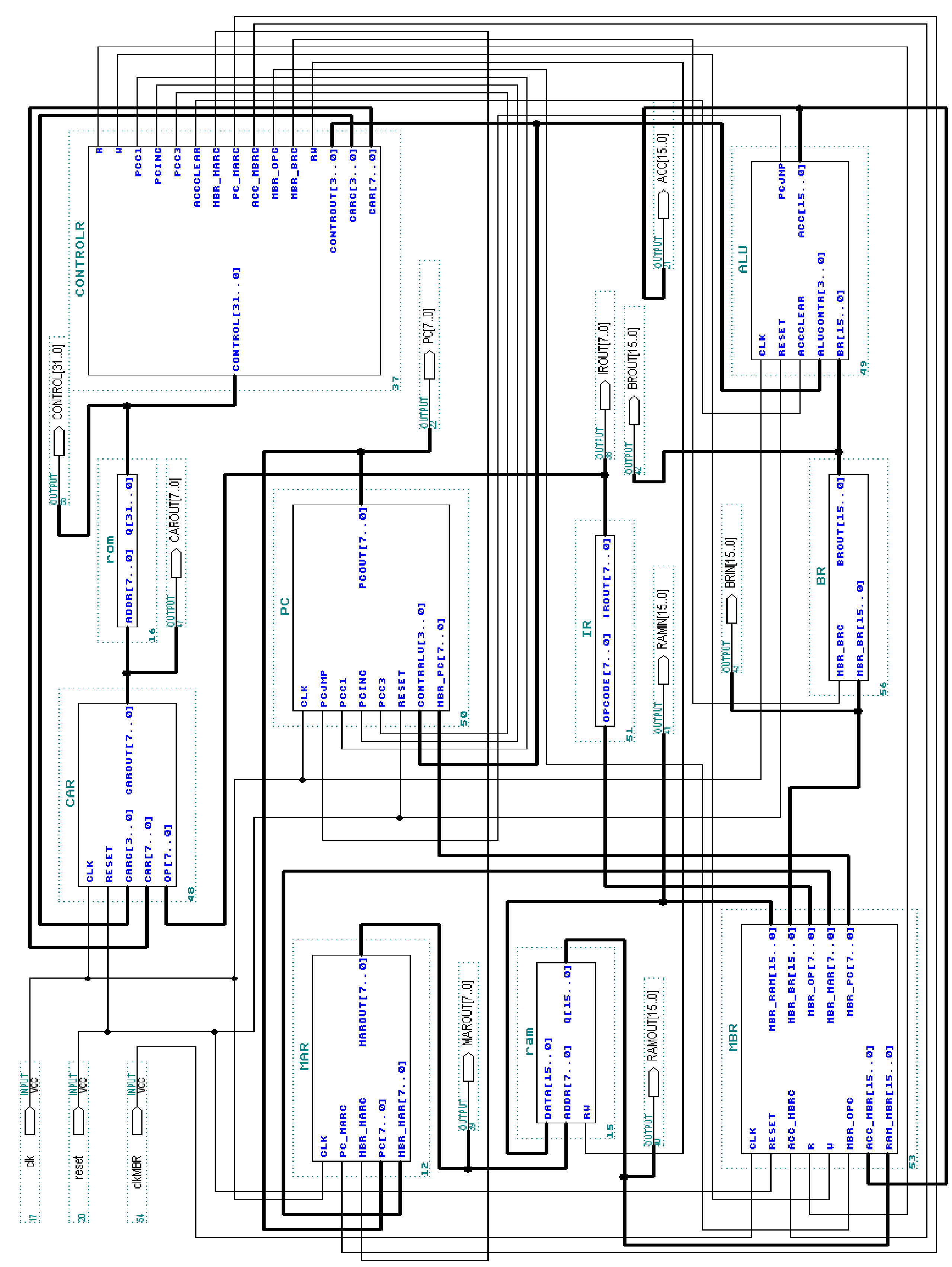
The clock cycle of CAR is 400 ns.
CAR 的时钟周期是 400 ns。
From the waveform, it takes 2.314ms to execute the operate. So the number of the executing cycles is 2.134/0.0004=**5335.**
从波形来看，它需要 2.314 毫秒来执行这个操作。所以执行周期的数目是 2.134/0.0004 = 5335。

程序4波形图及逐条指令、逐个阶段、逐个操作、逐个部件逐个数据变化的对应关系分析，数据运算过程分析。（必做）
程序 4 波形图及其段落逐个作业个部分逐个系统逐个变化的对应关系分析数据运算过程分析

VI. Appendix:

附录:

(I)The GDF of CPU:
(i)中央处理器的 GDF:



(II) The code of the CPU program:

(II)中央处理器程式码:

1, MBR (Memory Buffer Register)

记忆体缓冲寄存器

```
library ieee;
库  ieee;
use ieee.std_logic_1164.all;
使用  ieee.std _ logic _ 1164. all;
use ieee.std_logic_unsigned.all;
使用  ieee.std _ logic _ unsigned. all;
entity MBR is
实体  MBR  是
port( clk, reset, MBR_OPc, ACC_MBRc,R,W:in std_logic;
端口(clk，reset，MBR _ opc，ACC _ mbrc，r，w:  在标准逻辑中;
        ACC_MBR :in std_logic_vector(15 downto 0);
ACC _ mbr:  在标准逻辑向量(15 到 0)中;
        RAM_MBR :in std_logic_vector(15 downto 0);
RAM _ mbr:  在  std _ logic _ vector (15 down to 0)中;
        MBR_RAM :out std_logic_vector(15 downto 0);
MBR _ ram: out std _ logic _ vector (15 down to 0) ;
        MBR_BR    :out std_logic_vector(15 downto 0);
MBR _ br: out std _ logic _ vector (15 down to 0) ;
        MBR_OP    :out std_logic_vector(7 downto 0);
MBR _ op: out std _ logic _ vector (7 down to 0) ;
        MBR_MAR :out std_logic_vector(7 downto 0);
MBR _ mar: out std _ logic _ vector (7 down to 0) ;
        MBR_PC    :out std_logic_vector(7 downto 0));
MBR _ pc:  输出标准逻辑向量(7 到 0) ;
end MBR;
结束  MBR;
architecture behave of MBR is
膜生物反应器的结构特性是
begin
开始
   process(clk)
程序(clk)
   variable temp:std_logic_vector(15 downto 0);
变量  temp: std _ logic _ vector (15 down to 0) ;
   begin
开始
      if(clk'event and clk='0')then
如果(clk‘ event and clk = ‘0’)
        if reset='1' then
如果重置  =’1’，则
           if ACC_MBRc='1' then    temp:=ACC_MBR;      end if;
如果  ACC _ mbrc =’1’，则  temp: = ACC _ mbr;
           if R='1' then        MBR_BR<=RAM_MBR;         end if;
如果  r =’1’，则  MBR _ br < = RAM _ MBR;
```

```
            if W='1' then           MBR_RAM<=temp;           end if;
如果 w ='1'，则 MBR _ ram < = temp;
            MBR_MAR<=RAM_MBR(7 downto 0);
MBR _ mar < = RAM _ MBR (7→0) ;
            MBR_PC<=RAM_MBR(7 downto 0);
MBR _ pc < = RAM _ MBR (7 至 0) ;
            if MBR_OPc='1' then      MBR_OP<=RAM_MBR(15 downto 8);       end if;
如果 MBR _ opc ='1'，则 MBR _ op < = RAM _ MBR (15 到 8) ;
           else MBR_BR<=x"0000";
否则 MBR _ br < = x"0000";
                  MBR_MAR<="00000000";
MBR _ mar < = "00000000";
                  MBR_OP<="00000000";
MBR _ op < = "000000000";
                  MBR_PC<="00000000";
MBR _ pc < = "00000000";
           end if;
结束如果;
        end if;
结束如果;
    end process;
结束过程;
end behave;
结束行为;
```

2, BR (Buffer Register)

2，BR (缓冲寄存器)

```
library ieee;
库 ieee;
use ieee.std_logic_1164.all;
使用 ieee.std _ logic _ 1164. all;

entity BR is
实体 BR 是
port( MBR_BRc:in std_logic;
端口(MBR _ brc: 在标准逻辑中;
        MBR_BR:in std_logic_vector(15 downto 0);
MBR: 在标准逻辑向量(15 到 0)中;
        BRout:out std_logic_vector(15 downto 0));
BRout: out std _ logic _ vector (15 down to 0) ;
end BR;
结束 BR;
architecture behave of BR is
BR 的体系结构行为是
begin
```

开始
   process
过程
   begin
开始
           if MBR_BRc='1' then   BRout<=MBR_BR;   end if;
如果 MBR _ brc ='1'，则 BRout < = MBR _ br;
   end process;
结束过程;
end behave;
结束行为;

3,   MAR   (Memory Address Register)
3，MAR (内存地址寄存器)
library ieee;
库 ieee;
use ieee.std_logic_1164.all;
使用 ieee.std _ logic _ 1164. all;
use ieee.std_logic_unsigned.all;
使用 ieee.std _ logic _ unsigned. all;
entity MAR is
实体 MAR 是
port(clk,PC_MARc,MBR_MARc:in std_logic;
端口(clk，pc _ marc，MBR _ marc: 在标准逻辑;
        PC,MBR_MAR:in std_logic_vector(7 downto 0);
Mbr mar: 在标准逻辑向量(7 到 0)中;
        MARout:out std_logic_vector(7 downto 0));
MARout: out std _ logic _ vector (7 down to 0) ;
end MAR;
结束 MAR;
architecture behave of MAR is
MAR 的体系结构行为是
begin
开始
   process(clk)
程序(clk)
   begin
开始
      if(clk'event and clk='1')then
如果(clk‘ event and clk ='1’) ，则
           if PC_MARc='1' then   MARout<=PC;   end if;
如果 PC _ marc ='1'，则 MARout < = PC;
           if MBR_MARc='1' then   MARout<=MBR_MAR;   end if;
如果 MBR _ marc ='1'，则 MARout < = MBR _ mar;
      end if;

```
结束如果;
   end process;
结束过程;
end behave;
结束行为;
```

4, PC   (Program Counter)

4，PC (程序计数器)

```
library ieee;
库  ieee;
use ieee.std_logic_1164.all;
使用  ieee.std _ logic _ 1164. all;
use ieee.std_logic_unsigned.all;
使用  ieee.std _ logic _ unsigned. all;
entity PC is
实体  PC  是
port( clk,PCjmp,PCc1,PCinc,PCc3,reset:in std_logic;
端口(clk，PCjmp，PCc1，PCinc，PCc3，reset: in std _ logic;
        CONTRalu    :in std_logic_vector(3 downto 0);
CONTRalu:  在  std _ logic _ vector (3 到 0)中;
        MBR_PC      :in std_logic_vector(7 downto 0);
MBR _ pc:  在标准逻辑向量(7 到 0)中;
        PCout           :buffer std_logic_vector(7 downto 0));
PCout: buffer std _ logic _ vector (7 down to 0) ;
end PC;
终端  PC;
architecture behave of PC is
PC  的体系结构行为是
begin
开始
   process(clk)
程序(clk)
   begin
开始
      if(clk'event and clk='0')then
如果(clk‘ event and clk = ‘0’)
         if reset='1' then
如果重置  =’1’，则
              if CONTRalu="0101" then
如果  CONTRalu = “0101”
                 if PCjmp='1' then                    PCout<=MBR_PC;
如果  PCjmp =’1’，则  PCout < = MBR _ pc;
                   elsif PCjmp='0' then PCout<=PCout+1;
Elsif PCjmp =’0’然后  PCout < = PCout + 1;
                 end if;
```

```
结束如果;
              end if;
如果结束;
              if PCc1='1' then               PCout<="00000000";            end if;
如果  PCc1 =’1’，则  PCout < = “00000000”;
              if PCinc='1' then               PCout<=PCout+1;               end if;
如果  PCinc =’1’，则  PCout < = PCout + 1;  如果;
              if PCc3='1' then               PCout<=MBR_PC;                end if;
如果  PCc3 =’1’，则  PCout < = MBR _ pc;
         else PCout<="00000000";
其他  PCout < = “000000000”;
         end if;
结束如果;
      end if;
结束如果;
   end process;
结束过程;
end behave;
结束行为;
```

5, IR   (Instruction Register)
5，IR (指令寄存器)

```
library ieee;
库  ieee;
use ieee.std_logic_1164.all;
使用  ieee.std _ logic _ 1164. all;
use ieee.std_logic_unsigned.all;
使用  ieee.std _ logic _ unsigned. all;
entity IR is
实体  IR  是
port( opcode    :in std_logic_vector(7 downto 0);
端口(操作码:  在标准逻辑向量(7 至 0) ;
        IRout      :out std_logic_vector(7 downto 0));
输出:  输出标准逻辑向量(7 到 0) ;
end IR;
结束  IR;
architecture behave of IR is
红外光谱的结构特征是
begin
开始
      IRout<=opcode;
输出操作码;
end behave;
结束行为;
```

6, CAR (Control Address Register)

6，CAR (控制地址寄存器)

```
library ieee;
库  ieee;
use ieee.std_logic_1164.all;
使用  ieee.std _ logic _ 1164. all;
use ieee.std_logic_unsigned.all;
使用  ieee.std _ logic _ unsigned. all;
entity CAR is
实体  CAR  是
port( clk,reset :in std_logic;
端口(clk，reset: in std _ logic;
        CARc    :in std_logic_vector(3 downto 0);
在标准逻辑向量(3 到 0)中;
        CAR,OP      :in std_logic_vector(7 downto 0);
CAR，OP:  在  std _ logic _ vector (7 down 0)中;
        CARout:buffer std_logic_vector(7 downto 0));
CARout: buffer std _ logic _ vector (7 down to 0) ;
end CAR;
结束  CAR;
architecture behave of CAR is
CAR  的体系结构特性是
begin
开始
   process(clk)
程序(clk)
   begin
开始
      if(clk'event and clk='1')then
如果(clk‘ event and clk =’1’) ，则
        if reset='1' then
如果重置  =’1’，则
          if CARc="1000" then               CARout<="00000000";            end if;
如果  CARc = “1000”，则  CARout < = “00000000”;
          if CARc="0100" then               CARout<=OP+CARout;          end if;
如果  CARc = “0100”，则  CARout < = op + car out;
          if CARc="0010" then            CARout<=CAR;                      end if;
如果  CARc = “0010”，则  CARout < = CAR;
          if CARc="0001" then            CARout<=CARout+1;               end if;
如果  CARc = “0001”，则  CARout < = CARout + 1;
        else CARout<="00000000";
“000000000”;
        end if;
结束如果;
      end if;
```

```
结束如果;
  end process;
结束过程;
end behave;
结束行为;
```

7, CONTRALR (Control Buffer Register)
7，CONTRALR (控制缓冲寄存器)

```
library ieee;
库  ieee;
use ieee.std_logic_1164.all;
使用  ieee.std _ logic _ 1164. all;
use ieee.std_logic_unsigned.all;
使用  ieee.std _ logic _ unsigned. all;
entity CONTROLR is
实体  CONTROLR  是
port(
港口(
        control :in std_logic_vector(31 downto 0);
控制:  在标准逻辑向量(31 到 0) ;
        R,W, RW, PCc1,PCinc,PCc3:out std_logic;
R，w，RW，PCc1，PCinc，PCc3:  出标准逻辑;
        ACCclear,MBR_MARc,PC_MARc:out std_logic;
Accclear，mbr _ marc，PC _ marc: out std _ logic;
        ACC_MBRc,MBR_OPc,MBR_BRc:out std_logic;
ACC _ mbrc，MBR _ opc，MBR _ brc: out  标准逻辑;
        CONTRout:out std_logic_vector(3 downto 0);
Conout: out std _ logic _ vector (3 down to 0) ;
        CARc    :out std_logic_vector(3 downto 0);
输出标准逻辑向量(3 到 0) ;
        CAR      :out std_logic_vector(7 downto 0));
CAR: out std _ logic _ vector (7 down to 0) ;
end CONTROLR;
端部控制器;
architecture behave of CONTROLR is
CONTROLR  的体系结构行为是
begin
开始
   process
过程
   begin
开始
           CAR<=control(7 downto 0);
CAR < =  控制(7 至 0) ;
           PCc1<=control(8);
```

```
PCc1 < =  对照组(8) ;
           PCinc<=control(9);
PCinc < =  对照(9) ;
           PCc3<=control(10);
PCc3 < =  对照组(10) ;
           ACCclear<=control(11);
ACCclear < =  控制(11) ;
           CONTRout<=control(15 downto 12);
Control < = control (15 降至 12) ;
           R<=control(16);
R < =  对照(16) ;
           W<=control(17);
W < =  对照(17) ;
           MBR_MARc<=control(18);
MBR _ marc < =  对照(18) ;
           PC_MARc<=control(19);
PC _ marc < =  控制(19) ;
           ACC_MBRc<=control(20);
ACC _ mbrc < =  对照(20) ;
           MBR_OPc<=control(21);
MBR _ opc < =  对照(21) ;
           MBR_BRc<=control(22);
MBR _ brc < =  对照(22) ;
           CARc<=control(26 downto 23);
CARc < =  对照(从 26 到 23) ;
           RW<=control(17);
RW < =  对照(17) ;
end process;
最终过程;
end behave;
结束行为;
```

8, ALU (Arithmetic Logic Unit)
算术逻辑单元

```
library ieee;
库  ieee;
use ieee.std_logic_1164.all;
使用  ieee.std _ logic _ 1164. all;
use ieee.std_logic_unsigned.all;
使用  ieee.std _ logic _ unsigned. all;
entity ALU is
实体  ALU  是
port( clk,reset,ACCclear:in std_logic;
端口(clk，reset，ACCclear: in std _ logic;
          aluCONTR :in std_logic_vector(3 downto 0);
      aluCONTR:  在  std _ logic _ vector (3 down to 0)中;
          BR            :in std_logic_vector(15 downto 0);
      BR:  在  std _ logic _ vector (15 down to 0)中;
          PCjmp       :out std_logic;
      输出标准逻辑;
          ACC           :buffer std_logic_vector(15 downto 0));
      ACC: buffer std _ logic _ vector (15 down to 0) ;
      end ALU;
      结束  ALU;
      architecture behave of ALU is
      ALU  的架构行为是
      begin
      开始
        process(clk)
      程序(clk)
        begin
      开始
          if(clk'event and clk='0')then
      如果(clk‘ event and clk = ‘0’)
            if reset='0' then ACC<=x"0000";
      如果  reset =’0’，则  ACC < = x“0000”;
             else
      其他
              if ACCclear='1' then              ACC<=x"0000";            end if;              if
aluCONTR="0011" then     ACC<=BR+ACC;          end if;          --ADD             if
aluCONTR="0100" then     ACC<=ACC-BR;           end if;          --SUB             if
aluCONTR="0110" then     ACC<=ACC and BR;     end if;          --AND             if
aluCONTR="0111" then     ACC<=ACC or BR;       end if;           --OR            if
aluCONTR="1000" then     ACC<=not ACC;         end if;           --NOT
      如果  ACCclear = ‘1’，则  ACC < = x“0000”;  结束  if;  如果  aluCONTR = “0011”，则
ACC < = BR + ACC;  结束  if;-ADD  如果  aluCONTR = “0100”，则  ACC < = ACC-BR;  结束
if;-SUB  如果  aluCONTR = “0110”，则  ACC < = ACC  和  BR;  结束  if;-AND  如果
aluCONTR = “0111”，则  ACC < = ACC  或  BR;  结束  if;-OR  如果  aluCONTR = “1000”，则
ACC < = NOT ACC;  结束  if;-NOT
              if aluCONTR="1001" then                                                --SRR
      如果  aluCONTR = “1001”，则  -- SRR
                  ACC(14 downto 0)<=ACC(15 downto 1);       ACC(15)<='0';
      ACC (14 降至 0) < = ACC (15 降至 1) ACC (15) < =’0’;
              end if;
      结束如果;
              if aluCONTR="1010" then                                                --SRL
      如果  aluCONTR = “1010”那么  -- SRL
                  ACC(15 downto 1)<=ACC(14 downto 0);       ACC(0)<='0';
      ACC (15 降至 1) < = ACC (14 降至 0) ; ACC (0) < =’0’;
```

```
              end if;
结束如果;
              if aluCONTR="1011" then       ACC<=ACC*BR;              end if;           --MPY
如果  aluCONTR = “1011”，则  ACC < = ACC * BR;  结束  if; -- MPY
            end if;
结束如果;
          end if;
结束如果;
          if ACC>0 then                   PCjmp<='1';
如果  ACC > 0，则  PCjmp < =’1’;
                  else PCjmp<='0';
否则  PCjmp < =’0’;
          end if;
结束如果;
    end process;
结束过程;
end behave;
结束行为;
```