计算机设计与实践

CPU设计报告

班号：110103

学号：1110310305

姓名：王天博

目录

[一． 指令格式设计 1](#_Toc374801680)

[二． 微操作的定义 2](#_Toc374801681)

[三． 节拍的划分 3](#_Toc374801682)

[3.1. 非访存类指令 3](#_Toc374801683)

[3.2. 访存类指令 4](#_Toc374801684)

[3.3. 转跳指令 5](#_Toc374801685)

[四． 处理器结构框图和功能描述 6](#_Toc374801686)

[4.1. 整体设计框图 6](#_Toc374801687)

[4.2. 处理器功能描述 6](#_Toc374801688)

[五． 各功能模块结构框图和功能描述 7](#_Toc374801689)

[5.1. 时钟模块 7](#_Toc374801690)

[5.2. 取指模块 8](#_Toc374801691)

[5.3. 运算模块 9](#_Toc374801692)

[5.4. 存储模块 10](#_Toc374801693)

[5.5. 回写模块 11](#_Toc374801694)

[5.6. 访存控制模块 12](#_Toc374801695)

[六． 实验代码和测试序列 13](#_Toc374801696)

[6.1. VHDL代码 13](#_Toc374801697)

[6.2. UCF文件 27](#_Toc374801698)

[6.3. 测试指令序列 30](#_Toc374801699)

[七． 总结与问题 32](#_Toc374801700)

[7.1. 实验总结 32](#_Toc374801701)

[7.2. 遇到的问题 32](#_Toc374801702)

## 指令格式设计

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ***指令伪码*** | ***指 令*** | | | | ***寻址方式*** |
| mov ri, n | 00000 | Ri | N | | 立即寻址 |
| mov [addr], ri | 00001 | Ri | ADDR | | 直接寻址（寄存器->主存） |
| mov ri, [addr] | 00010 | Ri | ADDR | | 直接寻址（主存->寄存器） |
| mov ri, rj | 01000 | Ri |  | Rj | 寄存器寻址 |
| mov ri, [addr] | 00011 | Ri | ADDR | | 变址寻址 |
| mov ri, @rj | 11000 | Ri |  | Rj | 寄存器间址 |
| adc ri, rj | 01001 | Ri |  | Rj | 寄存器寻址 |
| adc ri, n | 00100 | Ri | N | | 立即寻址 |
| sbb ri, rj | 01010 | Ri |  | Rj | 寄存器寻址 |
| sbb ri, n | 00101 | Ri | N | | 立即寻址 |
| and ri, rj | 01011 | Ri |  | Rj | 寄存器寻址 |
| and ri, n | 00110 | Ri | N | | 立即寻址 |
| or ri, rj | 01100 | Ri |  | Rj | 寄存器寻址 |
| or ri, n | 00111 | Ri | N | | 立即寻址 |
| stc | 10111 |  | | | 无 |
| clc | 11111 |  | | | 无 |
| jmp [addr] | 10000 |  | ADDR | | 直接寻址 |
| jz sign | 10001 |  | SIGN | | 相对寻址 |
| jc sign | 10010 |  | SIGN | | 相对寻址 |

说明：

1. N为8位立即数
2. ADDR为地址低八位
3. Ri，Rj为寄存器编号，3位
4. SIGN为8位有符号数

## 微操作的定义

|  |  |
| --- | --- |
| ***微操作*** | ***含义*** |
| M(AD)->IR | 将存储器AD地址中的数据读到IR中 |
| TD->M(TA) | 将TD中的数据写入存储器的TA地址中 |
| 1->RD | 将RD读信号置位 |
| 1->WR | 将WR写信号置位 |
| 1->RUD | 将RUD寄存器更新信号置位 |
| 1->PUD | 将PUD PC更新信号置位 |
| 0->Cy | 将Cy标志位清零 |
| 1->Cy | 将Cy标志位置位 |
| OP(IR) | 取IR的15-11位的操作码 |
| NM(IR) | 取IR的7-0位的操作数 |
| Ad1(IR) | 取IR的10-8位的第一个寄存器编号 |
| Ad2(IR) | 取IR的2-0位的第二个寄存器编号 |
| Ad(IR) | 取IR的7-0位作为地址低八位 |
| Reg(RN) | 编号为RN的寄存器 |
| A+B+Cy | 寄存器A中的值与寄存器B中的值进行带进位加运算 |
| A-B-Cy | 寄存器A中的值与寄存器B中的值进行带借位减运算 |
| A and B | 寄存器A中的值与寄存器B中的值进行按位与运算 |
| A or B | 寄存器A中的值与寄存器B中的值进行按位或运算 |
| R7//Ad(IR) | R7作为地址的高八位，AD(IR)作为地址的低八位，拼接成16位物理地址 |
| Z·Aluout->PDATA | 如果Z标志位为0，则将aluout的值送至PDATA |
| Cy·Aluout->PDATA | 如果Cy标志位为0，则将aluout的值送至PDATA |

## 节拍的划分

### 3.1. 非访存类指令

**mov Ri, x**

T0 M(PC)->IR, 1->RD, PC->PC+1

T1 OP(IR)->C, NM(IR)->RDATA

T2

T3 Ad1(IR)->RN, 1->RUD, RDATA->Reg(RN)

**mov Rj, Ri**

T0 M(PC)->IR, 1->RD, PC->PC+1

T1 OP(IR)->C, Reg(Ad2(IR))->RDATA

T2

T3 Ad1(IR)->RN, 1->RUD, RDATA->Reg(RN)

**adc Ri, x**

T0 M(PC)->IR, 1->RD, PC->PC+1

T1 OP(IR)->C, Reg(Ad1(IR))->A, NM(IR)->B, A+B+Cy->Aluout

T2

T3 Ad1(IR)->RN, Aluout->RDATA, 1->RUD, RDATA->Reg(RN)

**adc Rj, Ri**

T0 M(PC)->IR, 1->RD, PC->PC+1

T1 OP(IR)->C, Reg(Ad1(IR))->A, Reg(Ad2(IR))->B, A+B+Cy->Aluout

T2

T3 Ad1(IR)->RN, Aluout->RDATA, 1->RUD, RDATA->Reg(RN)

**sbb Ri, x**

T0 M(PC)->IR, 1->RD, PC->PC+1

T1 OP(IR)->C, Reg(Ad1(IR))->A, NM(IR)->B, A-B-Cy->Aluout

T2

T3 Ad1(IR)->RN, Aluout->RDATA, 1->RUD, RDATA->Reg(RN)

**sbb Rj, Ri**

T0 M(PC)->IR, 1->RD, PC->PC+1

T1 OP(IR)->C, Reg(Ad1(IR))->A, Reg(Ad2(IR))->B, A-B-Cy->Aluout

T2

T3 Ad1(IR)->RN, Aluout->RDATA, 1->RUD, RDATA->Reg(RN)

**and Ri, x**

T0 M(PC)->IR, 1->RD, PC->PC+1

T1 OP(IR)->C, Reg(Ad1(IR))->A, NM(IR)->B, A and B->Aluout

T2

T3 Ad1(IR)->RN, Aluout->RDATA, 1->RUD, RDATA->Reg(RN)

**and Rj, Ri**

T0 M(PC)->IR, 1->RD, PC->PC+1

T1 OP(IR)->C, Reg(Ad1(IR))->A, Reg(Ad2(IR))->B, A and B->Aluout

T2

T3 Ad1(IR)->RN, Aluout->RDATA, 1->RUD, RDATA->Reg(RN)

**or Ri, x**

T0 M(PC)->IR, 1->RD, PC->PC+1

T1 OP(IR)->C, Reg(Ad1(IR))->A, NM(IR)->B, A or B->Aluout

T2

T3 Ad1(IR)->RN, Aluout->RDATA, 1->RUD, RDATA->Reg(RN)

**or Rj, Ri**

T0 M(PC)->IR, 1->RD, PC->PC+1

T1 OP(IR)->C, Reg(Ad1(IR))->A, Reg(Ad2(IR))->B, A or B->Aluout

T2

T3 Ad1(IR)->RN, Aluout->RDATA, 1->RUD, RDATA->Reg(RN)

**clc**

T0 M(PC)->IR, 1->RD, PC->PC+1

T1 OP(IR)->C, 0->Cy

T2

T3

**stc**

T0 M(PC)->IR, 1->RD, PC->PC+1

T1 OP(IR)->C, 1->Cy

T2

T3

### 3.2. 访存类指令

**mov addr, Ri**

T0 M(PC)->IR, 1->RD, PC->PC+1

T1 OP(IR)->C, R7//Ad(IR)->TA, Reg(Ri)->TD

T2 TD->M(TA), 1->WR

T3

**mov Ri, addr**

T0 M(PC)->IR, 1->RD, PC->PC+1

T1 OP(IR)->C, R7//Ad(IR)->TA

T2 Aluout->TA, M(TA)->TD, 1->RD

T3 TD->RDATA, Ad1(IR)->RN, 1->RUD, RDATA->Reg(RN)

**mov Ri, x (变址寻址，IX=R7）**

T0 M(PC)->IR, 1->RD, PC->PC+1

T1 OP(IR)->C, R7+x->TA

T2 M(TA)->TD, 1->RD

T3 TD->RDATA, Ad1(IR)->RN, 1->RUD, RDATA->Reg(RN)

**mov Ri, Rj (寄存器间址)**

T0 M(PC)->IR, 1->RD, PC->PC+1

T1 OP(IR)->C, R7//Rj->TA

T2 M(TA)->TD, 1->RD

T3 TD->RDATA, Ad1(IR)->RN, 1->RUD, RDATA->Reg(RN)

### 3.3. 转跳指令

**jmp addr**

T0 M(PC)->IR, 1->RD, PC->PC+1

T1 OP(IR)->C, R7//Ad(IR)->PDATA

T2

T3 PDATA->PC, 1->PUD

**jz sign**

T0 M(PC)->IR, 1->RD, PC->PC+1

T1 OP(IR)->C, SN(IR)->A, A+PC->Aluout, Z·Aluout->PDATA

T2

T3 PDATA->PC, 1->PUD

**jc sign**

T0 M(PC)->IR, 1->RD, PC->PC+1

T1 OP(IR)->C, SN(IR)->A, A+PC->Aluout, Cy·Aluout->PDATA

T2

T3 PDATA->PC, 1->PUD

## 处理器结构框图和功能描述

### 4.1. 整体设计框图



注：1. 图中连线名称为例化时相应的信号名

2. 由于图中空间有限，因而2根以上线的根数没有在图中标出，现单独列出如下：

|  |  |
| --- | --- |
| ***连线名*** | ***连线根数*** |
| pc\_o | 16 |
| pc\_i | 16 |
| ir\_i | 16 |
| ir\_o | 16 |
| ad\_o | 16 |
| ad\_i | 16 |
| aluout | 8 |
| r\_n | 3 |
| r\_i | 8 |
| r\_o | 8 |
| dt\_i | 8 |
| dt\_o | 8 |
| a\_bus | 16 |
| DBUS | 16 |

### 4.2. 处理器功能描述

处理器共分为六个模块。由机器周期的定义，将系统划分为四个主要模块：取指模块掌管取指周期，运算模块掌管运算周期，存储模块掌管访存周期，回写模块掌管回写周期。四个模块的工作流程由时钟模块统一调度。同时，在主存储器和各管理模块之间增加访存控制模块，来避免发生访问冲突以及保证空闲时总线不被占用。

在处理器开始工作时：

第一个节拍，取指模块将当前pc值发送给访存控制模块，由访存控制模块根据pc值读出存储器中的数据回传给取指模块，作为当前的ir指令，取指模块将ir指令发送给各模块；之后在第二个节拍的下降沿进行pc+1

第二个节拍，运算模块将传来的ir指令进行译码，根据操作码进行不同的运算。非访存类指令，运算的结果通过aluout输出给回写模块进行回写；访存类指令，将算出的地址通过ad\_o输出给存储模块，若有要写入的数据则通过aluout输出到存储模块；转跳类指令，计算出新的pc值，然后将新的pc值通过ad\_o输出给回写模块；

第三个节拍，如果有访存活动，则存储模块根据操作码的不同，或读或写，向访存控制模块发送读写信号，地址和数据，并将读出的数据通过Reg\_o送到回写模块；

第四个节拍，如果有转跳指令，判断转跳的条件，若成立则发送pc更新信号和新的pc值给取指模块；若有回写寄存器的要求，则将回写的数值和回写的寄存器标号送到运算模块并发送回写信号。

## 各功能模块结构框图和功能描述

### 5.1. 时钟模块

#### 5.1.1. 结构框图

clk

reset

c1

c2

c3

c4

时钟模块

#### 5.1.2. 信号定义

|  |  |  |  |
| --- | --- | --- | --- |
| 信号名 | 位数 | 方向 | 含义 |
| clk | 1 | I | 时钟信号 |
| reset | 1 | I | 复位信号 |
| c1 | 1 | O | 节拍1 |
| c2 | 1 | O | 节拍2 |
| c3 | 1 | O | 节拍3 |
| c4 | 1 | O | 节拍4 |

#### 5.1.3. 功能描述

向其他模块发送节拍信号，控制整个运行流程。

### 5.2. 取指模块

#### 5.2.1. 结构框图

c1

c2

reset

PC\_O

IR\_O

IR\_R

PC\_C

PC\_I

取指模块

#### 5.2.2. 信号定义

|  |  |  |  |
| --- | --- | --- | --- |
| 信号名 | 位数 | 方向 | 含义 |
| c1 | 1 | I | 节拍1，控制取值 |
| c2 | 1 | I | 节拍2，控制PC+1 |
| reset | 1 | I | 复位信号 |
| IR\_I | 16 | I | 访存控制模块传入的IR |
| PC\_C | 1 | I | 回写模块传来的PC更新信号 |
| PC\_I | 16 | I | 回写模块传来的PC的更新值 |
| PC\_O | 16 | O | 向其他模块输出的PC值 |
| IR\_O | 16 | O | 向其他模块输出的IR值 |
| IR\_R | 1 | O | 向访存控制模块发送的读IR信号 |

#### 5.2.3. 功能描述

1. 向访存控制模块发送PC的内容，并接受访存控制模块传来的IR值；

2. 向各个模块发送IR的值

3. 若回写模块没有更新信号则将PC+1，若有则接收回写模块的信号更新PC

### 5.3. 运算模块

#### 5.3.1. 结构框图

c2

reset

IR\_O

PC\_O

Reg\_C

Reg\_I

Reg\_N

AD\_O

aluout

Z

Cy

运算模块

#### 5.3.2. 信号定义

|  |  |  |  |
| --- | --- | --- | --- |
| 信号名 | 位数 | 方向 | 含义 |
| c2 | 1 | I | 节拍2，控制运算模块的执行 |
| reset | 1 | I | 复位信号 |
| IR\_O | 16 | I | 取指模块传来的IR |
| PC\_O | 16 | I | 取指模块传来的PC |
| Reg\_C | 1 | I | 回写模块传来的寄存器更新信号 |
| Reg\_N | 3 | I | 回写模块传来的寄存器更新编号 |
| Reg\_I | 8 | I | 回写模块传来的寄存器更新数值 |
| AD\_O | 16 | O | 输出算出的地址 |
| aluout | 8 | O | 输出算出的数据 |
| Z | 1 | O | 零标志新号 |
| Cy | 1 | O | 进位标志信号 |

#### 5.3.3. 功能描述

1. 根据IR操作码部分执行不同的运算

2. 接收回写模块传入的寄存器数据进行寄存器回写

### 5.4. 存储模块

#### 5.4.1. 结构框图

c3

reset

aluout

AD\_O

IR\_O

DT\_O

Reg\_O

AD\_I

DT\_I

WR\_O

RD\_O

存储模块

#### 5.4.2. 信号定义

|  |  |  |  |
| --- | --- | --- | --- |
| 信号名 | 位数 | 方向 | 含义 |
| c3 | 1 | I | 节拍3，控制存储模块的执行 |
| reset | 1 | I | 复位信号 |
| IR\_O | 16 | I | 取指模块传来的IR |
| AD\_O | 16 | I | 运算模块传来的地址 |
| aluout | 8 | I | 运算模块传来的数据 |
| DT\_I | 8 | I | 访存控制模块传来的数据 |
| DT\_O | 8 | O | 给访存控制模块送要写入的数据 |
| RD\_O | 1 | O | 给访存控制模块的读信号 |
| WR\_O | 1 | O | 给访存控制模块的写信号 |
| Reg\_O | 8 | O | 给回写模块送的数据 |
| AD\_I | 16 | O | 给访存控制模块送的地址 |

#### 5.4.3. 功能描述

1. 将要写入存储器的数据发送到访存控制模块，并向访存控制模块发送写入信号

2. 接收访存控制模块传来的数据送到回写模块

### 5.5. 回写模块

#### 5.5.1. 结构框图

reset

c4

aluout

AD\_O

IR\_O

Reg\_O

Z

Cy

Reg\_I

Reg\_N

PC\_I

Reg\_C

PC\_C

回写模块

#### 5.5.2. 信号定义

|  |  |  |  |
| --- | --- | --- | --- |
| 信号名 | 位数 | 方向 | 含义 |
| c4 | 1 | I | 节拍4，控制回写模块的执行 |
| reset | 1 | I | 复位信号 |
| IR\_O | 16 | I | 取指模块传来的IR |
| AD\_O | 16 | I | 运算模块传来的地址 |
| aluout | 8 | I | 运算模块传来的数据 |
| Reg\_O | 8 | I | 存储模块传来的数据 |
| Z | 1 | I | 零标志信号 |
| Cy | 1 | I | 进位标志信号 |
| Reg\_N | 3 | O | 寄存器更新编号 |
| Reg\_C | 1 | O | 寄存器更新信号 |
| Reg\_I | 8 | O | 寄存器更新数值 |
| PC\_C | 1 | O | PC更新信号 |
| PC\_I | 16 | O | PC更新数值 |

#### 5.5.3. 功能描述

1. 若转跳指令生效，则将转跳的地址写入PC；

2. 若有写入寄存器的操作，则发送寄存器回写信号，回写寄存器的值还有回写的寄存器编号

### 5.6. 访存控制模块

#### 5.5.1. 结构框图

RD\_O

WR\_O

PC\_O

IR\_R

AD\_O

DT\_O

DT\_I

IR\_I

nRD

nWR

ABUS

DBUS

nMREQ

nBLE

nBHE

访存控制模块

#### 5.5.2. 信号定义

|  |  |  |  |
| --- | --- | --- | --- |
| 信号名 | 位数 | 方向 | 含义 |
| RD\_O | 1 | I | 存储模块发来的读信号 |
| WR\_O | 1 | I | 存储模块发来的写信号 |
| PC\_O | 16 | I | 取指模块发来的PC值 |
| IR\_R | 1 | I | 取指模块发来的IR读信号 |
| AD\_O | 16 | I | 存储模块发来的地址 |
| DT\_O | 8 | I | 存储模块发来的写数据 |
| DT\_I | 8 | O | 向存储模块送出的读出数据 |
| nRD | 1 | O | 存储器读控制 |
| nWR | 1 | O | 存储器写控制 |
| ABUS | 16 | O | 地址总线 |
| DBUS | 16 | I/O | 数据总线 |
| IR\_I | 16 | O | 向取指模块发送的IR值 |
| nBHE | 1 | O | 存储器高位字节允许访问 |
| nBLE | 1 | O | 存储器低位字节允许访问 |
| nMREQ | 1 | O | 存储器片选 |

#### 5.5.3. 功能描述

1. 向存储器中取指令或数据

2. 向存储器中写入数据

## 实验代码和测试序列

### 6.1. VHDL代码

#### 时钟模块-clkblock.vhd

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_ARITH.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

entity clkblock is

port(clk : in std\_logic;

reset : in std\_logic;

c1 : out std\_logic;

c2 : out std\_logic;

c3 : out std\_logic;

c4 : out std\_logic);

end clkblock;

architecture Behavioral of clkblock is

signal t : std\_logic\_vector(3 downto 0);

begin

process (reset, clk)

begin

if reset = '1' then

t <= "0000";

else

if clk'event and clk='1' then

case t is

when "0000" => t <= "0001";

when "0001" => t <= "0010";

when "0010" => t <= "0100";

when "0100" => t <= "1000";

when "1000" => t <= "0001";

when others => t <= "0000";

end case;

end if;

end if;

end process;

c1 <= t(0);

c2 <= t(1);

c3 <= t(2);

c4 <= t(3);

end Behavioral;

#### 取指模块-ifblock.vhd

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_ARITH.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

entity ifblock is

Port ( c1 : in STD\_LOGIC;

c2 : in STD\_LOGIC;

PC\_O : out STD\_LOGIC\_VECTOR (15 downto 0);

IR\_I : in STD\_LOGIC\_VECTOR (15 downto 0);

IR\_O : out STD\_LOGIC\_VECTOR (15 downto 0);

PC\_C : in STD\_LOGIC;

PC\_I : in STD\_LOGIC\_VECTOR (15 downto 0);

IR\_R : out STD\_LOGIC;

reset : in STD\_LOGIC);

end ifblock;

architecture Behavioral of ifblock is

signal PC\_T : std\_logic\_vector (15 downto 0);

begin

process (reset, c1, c2, PC\_C, IR\_I)

begin

if reset='1' then

PC\_T <= "0000000000000000";

IR\_O <= "0000000000000000";

IR\_R <= '0';

elsif c1='1' then

IR\_R <= '1';

IR\_O <= IR\_I;

else

IR\_R <= '0';

if c2='0' and c2'event then

PC\_T <= PC\_T+1;

end if;

if PC\_C = '1' then

PC\_T <= PC\_I;

end if;

end if;

end process;

PC\_O <= PC\_T;

end Behavioral;

#### 运算模块-alublock.vhd

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_ARITH.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

entity alublock is

Port ( c2 : in STD\_LOGIC;

reset : in STD\_LOGIC;

IR\_O : in STD\_LOGIC\_VECTOR (15 downto 0);

AD\_O : out std\_logic\_vector (15 downto 0);

PC\_O : in std\_logic\_vector (15 downto 0);

aluout : out STD\_LOGIC\_VECTOR (7 downto 0);

Reg\_C : in STD\_LOGIC;

Reg\_N : in STD\_LOGIC\_VECTOR (2 downto 0);

Reg\_I : in STD\_LOGIC\_VECTOR (7 downto 0);

Z : out std\_logic;

Cy : out STD\_LOGIC);

end alublock;

architecture Behavioral of alublock is

type reg IS array(0 to 7) of std\_logic\_vector(7 downto 0);

signal R : reg;

signal tc, tz : std\_logic;

shared variable aluv : std\_logic\_vector (8 downto 0);

begin

process (ir\_o, reset, c2, Reg\_C)

variable ctr : std\_logic\_vector (4 downto 0);

variable r1, r2, x : std\_logic\_vector (8 downto 0);

begin

ctr := IR\_O(15 downto 11);

r1 := '0'&R(conv\_integer(IR\_O(10 downto 8)));

r2 := '0'&R(conv\_integer(IR\_O(2 downto 0)));

x := '0'&IR\_O(7 downto 0);

if reset='1' then

tz <= '0';

tc <= '0';

aluv := "000000000";

R(0) <= "00000000";

R(1) <= "00000000";

R(2) <= "00000000";

R(3) <= "00000000";

R(4) <= "00000000";

R(5) <= "00000000";

R(6) <= "00000000";

R(7) <= "00000000";

AD\_O <= "0000000000000000";

elsif Reg\_C = '1' then

R(conv\_integer(IR\_O(10 downto 8))) <= Reg\_I;

elsif c2 = '1' and c2'event then

AD\_O(15 downto 8) <= R(7);

case ctr is

when "11111" =>

tc <= '0';

when "10111" =>

tc <= '1';

when "00000" =>

aluv := x;

when "01000" =>

aluv := r2;

when "00100" =>

tz <= '0';

aluv := r1 + x + tc;

tc <= aluv(8);

if aluv(7 downto 0)="00000000" then

tz <= '1';

end if;

when "01001" =>

tz <= '0';

aluv := r1 + r2 + tc;

tc <= aluv(8);

if aluv(7 downto 0)="00000000" then

tz <= '1';

end if;

when "00101" =>

tz <= '0';

aluv := r1 - x - tc;

tc <= aluv(8);

if aluv(7 downto 0)="00000000" then

tz <= '1';

end if;

when "01010" =>

tz <= '0';

aluv := r1 - r2 - tc;

tc <= aluv(8);

if aluv(7 downto 0)="00000000" then

tz <= '1';

end if;

when "00110" =>

tz <= '0';

aluv := r1 and x;

if aluv(7 downto 0)="00000000" then

tz <= '1';

end if;

when "01011" =>

tz <= '0';

aluv := r1 and r2;

if aluv(7 downto 0)="00000000" then

tz <= '1';

end if;

when "00111" =>

tz <= '0';

aluv := r1 or x;

if aluv(7 downto 0)="00000000" then

tz <= '1';

end if;

when "01100" =>

tz <= '0';

aluv := r1 or r2;

if aluv(7 downto 0)="00000000" then

tz <= '1';

end if;

when "00001" =>

AD\_O(15 downto 8) <= R(7);

AD\_O(7 downto 0) <=x(7 downto 0);

aluv := r1;

when "00010" =>

AD\_O(7 downto 0) <= x(7 downto 0);

when "00011" =>

AD\_O(7 downto 0) <= (R(6)+x(7 downto 0));

when "10000" =>

AD\_O(7 downto 0) <= x(7 downto 0);

when "11000" =>

AD\_O(7 downto 0) <=r2(7 downto 0);

when "10001" =>

if tz = '1' then

if (IR\_O(7) = '0') then

AD\_O <= PC\_O + ("00000000"&x(7 downto 0))+1;

else

AD\_O <= PC\_O - ("00000000"&x(7 downto 0))+1;

end if;

end if;

when "10010" =>

if tc = '1' then

if (IR\_O(7) = '0') then

AD\_O <= PC\_O + ("00000000"&x(7 downto 0))+1;

else

AD\_O <= PC\_O - ("00000000"&x(7 downto 0))+1;

end if;

end if;

when others => NULL;

end case;

end if;

aluout <= aluv(7 downto 0);

end process;

Z <= tz;

Cy <= tc;

end Behavioral;

#### 存储模块-stblock.vhd

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_ARITH.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

entity stblock is

Port ( c3 : in STD\_LOGIC;

reset : in STD\_LOGIC;

IR\_O : in STD\_LOGIC\_VECTOR (15 downto 0);

AD\_O : in STD\_LOGIC\_VECTOR (15 downto 0);

aluout : in STD\_LOGIC\_VECTOR (7 downto 0);

DT\_I : in STD\_LOGIC\_VECTOR (7 downto 0);

DT\_O : out std\_logic\_vector (7 downto 0);

RD\_O : out STD\_LOGIC;

WR\_O : out STD\_LOGIC;

Reg\_O : out STD\_LOGIC\_VECTOR (7 downto 0);

AD\_I : out std\_logic\_vector (15 downto 0));

end stblock;

architecture Behavioral of stblock is

begin

process (ir\_o, c3, reset, DT\_I)

variable ctr : std\_logic\_vector (4 downto 0);

begin

ctr := IR\_O(15 downto 11);

if reset = '1' then

DT\_O <= "00000000";

RD\_O <= '0';

WR\_O <= '0';

Reg\_O <= "00000000";

AD\_I <= "0000000000000000";

elsif c3 = '1' then

case ctr is

when "00001" =>

DT\_O <= aluout;

AD\_I <= AD\_O;

WR\_O <= '1';

RD\_O <= '0';

when "00010"|"00011"|"11000" =>

AD\_I <= AD\_O;

RD\_O <= '1';

WR\_O <= '0';

Reg\_O <= DT\_I;

when others => NULL;

end case;

else

RD\_O <= '0';

WR\_O <= '0';

end if;

end process;

end Behavioral;

#### 回写模块-rewblock.vhd

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_ARITH.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

entity rewblock is

Port ( c4 : in STD\_LOGIC;

reset : in STD\_LOGIC;

IR\_O : in STD\_LOGIC\_VECTOR (15 downto 0);

aluout : in STD\_LOGIC\_VECTOR (7 downto 0);

Reg\_O : in STD\_LOGIC\_VECTOR (7 downto 0);

AD\_O : in STD\_LOGIC\_VECTOR (15 downto 0);

Z : in std\_logic;

Cy : in std\_logic;

Reg\_N : out STD\_LOGIC\_VECTOR (2 downto 0);

Reg\_C : out STD\_LOGIC;

Reg\_I : out STD\_LOGIC\_VECTOR (7 downto 0);

PC\_C : out STD\_LOGIC;

PC\_I : out STD\_LOGIC\_VECTOR (15 downto 0));

end rewblock;

architecture Behavioral of rewblock is

begin

process(ir\_o, c4, reset)

variable ctr : std\_logic\_vector (4 downto 0);

begin

if reset = '1' then

Reg\_N <= "000";

Reg\_C <= '0';

Reg\_I <= "00000000";

PC\_C <= '0';

PC\_I <= "0000000000000000";

elsif c4 = '1' then

ctr := IR\_O(15 downto 11);

case ctr is

when "00000"|"01000"|"00100"|"01001"|"01010"|"00110"|"01011"|"00111"|"01100"|"00101" =>

Reg\_N <= IR\_O(10 downto 8);

Reg\_C <= '1';

Reg\_I <= aluout;

when "00010"|"00011"|"11000" =>

Reg\_N <= IR\_O(10 downto 8);

Reg\_C <= '1';

Reg\_I <= Reg\_O;

when "10000" =>

PC\_C <= '1';

PC\_I <= AD\_O;

when "10001" =>

if Z = '1' then

PC\_C <= '1';

PC\_I <= AD\_O;

end if;

when "10010" =>

if Cy = '1' then

PC\_C <= '1';

PC\_I <= AD\_O;

end if;

when others => NULL;

end case;

else

PC\_C <= '0';

Reg\_C <= '0';

end if;

end process;

end Behavioral;

#### 访存控制模块-stctrblock.vhd

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_ARITH.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

entity stctrblock is

Port ( RD\_O : in STD\_LOGIC;

WR\_O : in STD\_LOGIC;

PC\_O : in STD\_LOGIC\_VECTOR (15 downto 0);

IR\_R : in std\_logic;

AD\_O : in STD\_LOGIC\_VECTOR (15 downto 0);

DT\_O : in STD\_LOGIC\_VECTOR (7 downto 0);

DT\_I : out STD\_LOGIC\_VECTOR (7 downto 0);

nRD : out STD\_LOGIC;

nWR : out STD\_LOGIC;

ABUS : out STD\_LOGIC\_VECTOR (15 downto 0);

DBUS : inout STD\_LOGIC\_VECTOR (15 downto 0);

IR\_I : out STD\_LOGIC\_VECTOR (15 downto 0);

nBHE : out STD\_LOGIC;

nBLE : out STD\_LOGIC;

nMERQ : out STD\_LOGIC);

end stctrblock;

architecture Behavioral of stctrblock is

begin

process (RD\_O, WR\_O, IR\_R)

begin

if (IR\_R = '1') then

nRD <= '0';

nWR <= '1';

nBHE <= '0';

nBLE <= '0';

nMERQ <= '0';

ABUS <= PC\_O;

IR\_I <= DBUS;

DBUS <= "ZZZZZZZZZZZZZZZZ";

elsif (RD\_O = '1') then

nRD <= '0';

nWR <= '1';

nBHE <= '0';

nBLE <= '0';

nMERQ <= '0';

ABUS <= AD\_O;

DT\_I <= DBUS(7 downto 0);

DBUS <= "ZZZZZZZZZZZZZZZZ";

elsif (WR\_O = '1') then

nRD <= '1';

nWR <= '0';

nBHE <= '0';

nBLE <= '0';

nMERQ <= '0';

ABUS <= AD\_O;

DBUS <= "00000000"&DT\_O;

else

nRD <= '1';

nWR <= '1';

nBHE <= '1';

nBLE <= '1';

nMERQ <= '1';

ABUS <= "0000000000000000";

IR\_I <= "0000000000000000";

DBUS <= "ZZZZZZZZZZZZZZZZ";

DT\_I <= "00000000";

end if;

end process;

end Behavioral;

#### 例化文件-cpuall.vhd

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_ARITH.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

entity cpuall is

port(clk: in std\_logic;

reset: in std\_logic;

Dbus: inout std\_logic\_vector(15 downto 0);

Abus: out std\_logic\_vector(15 downto 0);

nMREQ: out std\_logic;

nWR: out std\_logic;

nRD: out std\_logic;

nBLE,nBHE: out std\_logic;

Sd : out std\_logic\_vector(15 downto 0);

Sa : out std\_logic\_vector(15 downto 0);

Sp : out std\_logic\_vector(15 downto 0);

Ct : out std\_logic\_vector(7 downto 0);

CL : out std\_logic\_vector(3 downto 0));

end cpuall;

architecture Behavioral of cpuall is

component clkblock is

port(clk : in std\_logic;

reset : in std\_logic;

c1 : out std\_logic;

c2 : out std\_logic;

c3 : out std\_logic;

c4 : out std\_logic);

end component;

component ifblock is

Port ( c1 : in STD\_LOGIC;

c2 : in STD\_LOGIC;

PC\_O : out STD\_LOGIC\_VECTOR (15 downto 0);

IR\_I : in STD\_LOGIC\_VECTOR (15 downto 0);

IR\_O : out STD\_LOGIC\_VECTOR (15 downto 0);

PC\_C : in STD\_LOGIC;

PC\_I : in STD\_LOGIC\_VECTOR (15 downto 0);

IR\_R : out STD\_LOGIC;

reset : in STD\_LOGIC);

end component;

component alublock is

Port ( c2 : in STD\_LOGIC;

reset : in STD\_LOGIC;

IR\_O : in STD\_LOGIC\_VECTOR (15 downto 0);

AD\_O : out std\_logic\_vector (15 downto 0);

PC\_O : in std\_logic\_vector (15 downto 0);

aluout : out STD\_LOGIC\_VECTOR (7 downto 0);

Reg\_C : in STD\_LOGIC;

Reg\_N : in STD\_LOGIC\_VECTOR (2 downto 0);

Reg\_I : in STD\_LOGIC\_VECTOR (7 downto 0);

Z : out std\_logic;

Cy : out STD\_LOGIC);

end component;

component stblock is

Port ( c3 : in STD\_LOGIC;

reset : in STD\_LOGIC;

IR\_O : in STD\_LOGIC\_VECTOR (15 downto 0);

AD\_O : in STD\_LOGIC\_VECTOR (15 downto 0);

aluout : in STD\_LOGIC\_VECTOR (7 downto 0);

DT\_I : in STD\_LOGIC\_VECTOR (7 downto 0);

DT\_O : out std\_logic\_vector (7 downto 0);

RD\_O : out STD\_LOGIC;

WR\_O : out STD\_LOGIC;

Reg\_O : out STD\_LOGIC\_VECTOR (7 downto 0);

AD\_I : out std\_logic\_vector (15 downto 0));

end component;

component rewblock is

Port ( c4 : in STD\_LOGIC;

reset : in STD\_LOGIC;

IR\_O : in STD\_LOGIC\_VECTOR (15 downto 0);

aluout : in STD\_LOGIC\_VECTOR (7 downto 0);

Reg\_O : in STD\_LOGIC\_VECTOR (7 downto 0);

AD\_O : in STD\_LOGIC\_VECTOR (15 downto 0);

Z : in std\_logic;

Cy : in std\_logic;

Reg\_N : out STD\_LOGIC\_VECTOR (2 downto 0);

Reg\_C : out STD\_LOGIC;

Reg\_I : out STD\_LOGIC\_VECTOR (7 downto 0);

PC\_C : out STD\_LOGIC;

PC\_I : out STD\_LOGIC\_VECTOR (15 downto 0));

end component;

component stctrblock is

Port ( RD\_O : in STD\_LOGIC;

WR\_O : in STD\_LOGIC;

PC\_O : in STD\_LOGIC\_VECTOR (15 downto 0);

IR\_R : in std\_logic;

AD\_O : in STD\_LOGIC\_VECTOR (15 downto 0);

DT\_O : in STD\_LOGIC\_VECTOR (7 downto 0);

DT\_I : out STD\_LOGIC\_VECTOR (7 downto 0);

nRD : out STD\_LOGIC;

nWR : out STD\_LOGIC;

ABUS : out STD\_LOGIC\_VECTOR (15 downto 0);

DBUS : inout STD\_LOGIC\_VECTOR (15 downto 0);

IR\_I : out STD\_LOGIC\_VECTOR (15 downto 0);

nBHE : out STD\_LOGIC;

nBLE : out STD\_LOGIC;

nMERQ : out STD\_LOGIC);

end component;

signal c1, c2, c3, c4 : std\_logic;

signal n\_BLE, n\_BHE, n\_MREQ, n\_RD, n\_WR : std\_logic;

signal pc\_i, pc\_o : std\_logic\_vector(15 downto 0);

signal ir\_i, ir\_o : std\_logic\_vector(15 downto 0);

signal pc\_c, r\_c : std\_logic;

signal ir\_r : std\_logic;

signal z, c : std\_logic;

signal ad\_o, ad\_i : std\_logic\_vector(15 downto 0);

signal aluout : std\_logic\_vector(7 downto 0);

signal r\_n : std\_logic\_vector(2 downto 0);

signal r\_i, r\_o : std\_logic\_vector(7 downto 0);

signal dt\_i, dt\_o : std\_logic\_vector(7 downto 0);

signal rd\_o, wr\_o : std\_logic;

signal a\_bus : std\_logic\_vector(15 downto 0);

begin

U0 : clkblock port map (clk, reset, c1, c2, c3, c4);

U1 : ifblock port map (c1, c2, pc\_o, ir\_i, ir\_o, pc\_c, pc\_i, ir\_r, reset);

U2 : alublock port map (c2, reset, ir\_o, ad\_o, pc\_o, aluout, r\_c, r\_n, r\_i, z, c);

U3 : stblock port map (c3, reset, ir\_o, ad\_o, aluout, dt\_i, dt\_o, rd\_o, wr\_o, r\_o, ad\_i);

U4 : rewblock port map (c4, reset, ir\_o, aluout, r\_o, ad\_o, z, c, r\_n, r\_c, r\_i, pc\_c, pc\_i);

U5 : stctrblock port map (rd\_o, wr\_o, pc\_o, ir\_r, ad\_o, dt\_o, dt\_i, n\_RD, n\_WR, a\_bus, Dbus, ir\_i, n\_BHE, n\_BLE, n\_MREQ);

Abus <= a\_bus;

nBHE <= n\_BHE;

nBLE <= n\_BLE;

nMREQ <= n\_MREQ;

nRD <= n\_RD;

nWR <= n\_WR;

Sd <= Dbus;

Sa <= a\_bus;

Sp <= ir\_o;

Ct(7) <= not n\_BHE;

Ct(6) <= not n\_BLE;

Ct(5) <= not n\_MREQ;

Ct(4) <= rd\_o;

Ct(3) <= wr\_o;

Ct(2) <= r\_c;

Ct(1) <= z;

Ct(0) <= c;

CL(3) <= c4;

CL(2) <= c3;

CL(1) <= c2;

CL(0) <= c1;

end Behavioral;

### 6.2. UCF文件

#PACE: Start of Constraints generated by PACE

#PACE: Start of PACE I/O Pin Assignments

NET "Abus<0>" LOC = "P179" ;

NET "Abus<10>" LOC = "P115" ;

NET "Abus<11>" LOC = "P116" ;

NET "Abus<12>" LOC = "P119" ;

NET "Abus<13>" LOC = "P140" ;

NET "Abus<14>" LOC = "P144" ;

NET "Abus<15>" LOC = "P145" ;

NET "Abus<1>" LOC = "P178" ;

NET "Abus<2>" LOC = "P177" ;

NET "Abus<3>" LOC = "P172" ;

NET "Abus<4>" LOC = "P171" ;

NET "Abus<5>" LOC = "P151" ;

NET "Abus<6>" LOC = "P150" ;

NET "Abus<7>" LOC = "P147" ;

NET "Abus<8>" LOC = "P146" ;

NET "Abus<9>" LOC = "P113" ;

NET "CL<0>" LOC = "P102" ;

NET "CL<1>" LOC = "P100" ;

NET "CL<2>" LOC = "P99" ;

NET "CL<3>" LOC = "P98" ;

NET "clk" LOC = "P75" ;

NET "Ct<0>" LOC = "P196" ;

NET "Ct<1>" LOC = "P193" ;

NET "Ct<2>" LOC = "P192" ;

NET "Ct<3>" LOC = "P190" ;

NET "Ct<4>" LOC = "P3" ;

NET "Ct<5>" LOC = "P200" ;

NET "Ct<6>" LOC = "P199" ;

NET "Ct<7>" LOC = "P197" ;

NET "Dbus<0>" LOC = "P167" ;

NET "Dbus<10>" LOC = "P123" ;

NET "Dbus<11>" LOC = "P128" ;

NET "Dbus<12>" LOC = "P132" ;

NET "Dbus<13>" LOC = "P133" ;

NET "Dbus<14>" LOC = "P134" ;

NET "Dbus<15>" LOC = "P135" ;

NET "Dbus<1>" LOC = "P165" ;

NET "Dbus<2>" LOC = "P164" ;

NET "Dbus<3>" LOC = "P163" ;

NET "Dbus<4>" LOC = "P162" ;

NET "Dbus<5>" LOC = "P161" ;

NET "Dbus<6>" LOC = "P160" ;

NET "Dbus<7>" LOC = "P153" ;

NET "Dbus<8>" LOC = "P120" ;

NET "Dbus<9>" LOC = "P122" ;

NET "nBHE" LOC = "P138" ;

NET "nBLE" LOC = "P137" ;

NET "nMREQ" LOC = "P168" ;

NET "nRD" LOC = "P139" ;

NET "nWR" LOC = "P152" ;

NET "reset" LOC = "P154" ;

NET "Sa<0>" LOC = "P31" ;

NET "Sa<10>" LOC = "P47" ;

NET "Sa<11>" LOC = "P48" ;

NET "Sa<12>" LOC = "P49" ;

NET "Sa<13>" LOC = "P50" ;

NET "Sa<14>" LOC = "P55" ;

NET "Sa<15>" LOC = "P56" ;

NET "Sa<1>" LOC = "P33" ;

NET "Sa<2>" LOC = "P34" ;

NET "Sa<3>" LOC = "P35" ;

NET "Sa<4>" LOC = "P36" ;

NET "Sa<5>" LOC = "P39" ;

NET "Sa<6>" LOC = "P40" ;

NET "Sa<7>" LOC = "P41" ;

NET "Sa<8>" LOC = "P42" ;

NET "Sa<9>" LOC = "P45" ;

NET "Sd<0>" LOC = "P4" ;

NET "Sd<10>" LOC = "P22" ;

NET "Sd<11>" LOC = "P23" ;

NET "Sd<12>" LOC = "P24" ;

NET "Sd<13>" LOC = "P25" ;

NET "Sd<14>" LOC = "P28" ;

NET "Sd<15>" LOC = "P29" ;

NET "Sd<1>" LOC = "P5" ;

NET "Sd<2>" LOC = "P8" ;

NET "Sd<3>" LOC = "P9" ;

NET "Sd<4>" LOC = "P11" ;

NET "Sd<5>" LOC = "P12" ;

NET "Sd<6>" LOC = "P15" ;

NET "Sd<7>" LOC = "P16" ;

NET "Sd<8>" LOC = "P18" ;

NET "Sd<9>" LOC = "P19" ;

NET "Sp<0>" LOC = "P60" ;

NET "Sp<10>" LOC = "P129" ;

NET "Sp<11>" LOC = "P202" ;

NET "Sp<12>" LOC = "P203" ;

NET "Sp<13>" LOC = "P205" ;

NET "Sp<14>" LOC = "P206" ;

NET "Sp<15>" LOC = "P103" ;

NET "Sp<1>" LOC = "P61" ;

NET "Sp<2>" LOC = "P62" ;

NET "Sp<3>" LOC = "P63" ;

NET "Sp<4>" LOC = "P2" ;

NET "Sp<5>" LOC = "P108" ;

NET "Sp<6>" LOC = "P109" ;

NET "Sp<7>" LOC = "P112" ;

NET "Sp<8>" LOC = "P126" ;

NET "Sp<9>" LOC = "P127" ;

#PACE: Start of PACE Area Constraints

#PACE: Start of PACE Prohibit Constraints

#PACE: End of Constraints generated by PACE

### 6.3. 测试指令序列

表1：测试指令含义

|  |  |  |
| --- | --- | --- |
| 指令 | 含义 | 寻址方式 |
| mov r0, 01 | 将01赋给寄存器R0 | 立即寻址 |
| mov r1, 02 | 将02赋给寄存器R1 | 立即寻址 |
| mov r7, 03 | 将03赋给寄存器R7 | 立即寻址 |
| mov [02], r0 | 将R0的值写入主存地址[R7//02] | 直接寻址 |
| mov r2, [02] | 将主存地址[R7//02]的值写入R2 | 直接寻址 |
| mov R6, R1 | 将R1的值写入R6 | 寄存器寻址 |
| mov R4, 05 | 将主存地址[R7//(R6+05)]的值写入R4 | 变址寻址 |
| mov R5，R2 | 将主存地址[R7//R2]的值写入R5 | 寄存器间址 |
| adc R0, R1 | 将R0和R1中的值相加结果回写回R0 | 寄存器寻址 |
| adc R0, FD | 将R0中的值和FD相加结果回写回R0 | 立即寻址 |
| sbb R1, 01 | 将R1中的值减去01，结果回写回R1 | 立即寻址 |
| jz 05 | 若Z标志位为1，则转跳到PC+5+1 | 相对寻址 |
| sbb R4, R2 | 将R4中的值减去R2，结果回写回R4 | 寄存器寻址 |
| STC | 将Cy标志位置为1 |  |
| jc 08 | 若Cy标志位为1，则转跳到pc+8+1 | 相对寻址 |
| CLC | 将Cy标志位置为0 |  |
| and R4, 07 | 将R4中的值和07进行and运算，结果回写回R4 | 立即寻址 |
| and R4, R2 | 将R4和R2进行and运算，结果回写回R4 | 寄存器寻址 |
| or R4, 02 | 将R4中的值和02进行or运算，结果回写回R4 | 立即寻址 |
| or R4, r2 | 将R4和R2进行or运算，结果回写回R4 | 寄存器寻址 |
| mov [10],r4 | 将R4中的值写入主存地址[R7//10] | 直接寻址 |
| jmp [09] | 转跳到地址[R7//09] | 直接寻址 |
| jz 05 | 若Z标志位为1，则转跳到PC+5+1 | 相对寻址 |
| jc 08 | 若Cy标志位为1，则转跳到pc+8+1 | 相对寻址 |
| mov r7, 00 | 将00赋给寄存器R7 | 立即寻址 |
| jmp [00] | 转跳到地址[R7//00]，即地址0000h，为程序的开始地址 | 直接寻址 |

表2：测试指令的运行结果

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 地址 | 指令 | 二进制 | 16进制 | 结果 |
| 0000 | mov r0, 01 | 00000 000 00000001 | 0001 | R0=01 |
| 0001 | mov r1, 02 | 00000 001 00000010 | 0102 | R1=02 |
| 0002 | mov r7, 03 | 00000 111 00000011 | 0703 | R7=03 |
| 0003 | mov [02], r0 | 00001 000 00000010 | 0802 | [02]=01 |
| 0004 | mov r2, [02] | 00010 010 00000010 | 1202 | R2=[02]=01 |
| 0005 | mov R6, R1 | 01000 110 00000001 | 4601 | R6=02 |
| 0006 | mov R4, 05 | 00011 100 00000101 | 1C05 | R4=[R7//(R6+05)] |
| 0007 | mov R5, R2 | 11000 101 00000010 | C502 | R5=[R7//R2] |
| 0008 | adc R0, R1 | 01001 000 00000001 | 4801 | R0=03 |
| 0009 | adc R0, FD | 00100 000 11111101 | 20FD | R0=00, CY=1，Z=1 |
| 000a | sbb R1, 01 | 00101 001 00000001 | 2901 | R1=00, Z=1 |
| 000b | jz 05 | 10001 000 00000101 | 8805 | PC=PC+5+1 |
| 0011 | sbb R4, R2 | 01010 100 00000010 | 5402 | R4=05 |
| 0012 | STC | 10111 000 00000000 | B800 | CY=1 |
| 0013 | jc 08 | 10010 000 00001000 | 9008 | PC=PC+8+1 |
| 001c | CLC | 11111 000 00000000 | F800 | CY=0 |
| 001d | and R4, 07 | 00110 100 00000111 | 3407 | R4=05 |
| 001e | and R4, R2 | 01011 100 00000010 | 5C02 | R4=01 |
| 001f | or R4, 02 | 00111 100 00000010 | 3c02 | R4=03 |
| 0020 | or R4, r2 | 01100 100 00000010 | 6402 | R4=03 |
| 0021 | mov [10],r4 | 00001 100 00010000 | 0c10 | [10]=03 |
| 0022 | jmp [09] | 10000 000 00001001 | 8009 | PC=R7//09 |
| 0309 | jz 05 | 10001 000 00000101 | 8805 |  |
| 030a | jc 08 | 10010 000 00001000 | 9008 |  |
| 030b | mov r7, 00 | 00000 111 00000000 | 0700 | R7=00 |
| 030c | jmp [00] | 10000 000 00000000 | 8000 | pc=0000 |

## 总结与问题

### 7.1. 实验总结

这次实验一共用了四周时间。在这四周时间里，我学到的东西，可能比过去一个学期里面学习到的东西都多。第一次从头到尾设计一个大的vhdl工程，第一次编写这么多vhdl代码，第一次认识到硬件程序的设计是多么的复杂。从开始的整体结构的设计，节拍的划分，到各个模块的细节设计，各个模块的仿真，到最后整体的仿真和下载，我熟悉了一个工程的建立过程，熟悉了vhdl的各种知识，对硬件也有了更深刻的了解。我意识到，我原来vhdl的知识还很薄弱，比如最简单但是最重要的时序控制，对变量和信号的使用，许多原来没用过的函数，也遇到了许多原来没遇到的问题。经过了这次实验，现在我对vhdl语言的编程以及控制更加的熟悉了，也对整个设计流程有了深入的了解。

### 7.2. 遇到的问题

1. 设计过程

(1). 开始的时候，由于对老师的要求理解有错误，指令格式设计的有问题，并且不得不采用单双混长的格式设计指令；后来经过与老师的交流明白了实验的要求，采用了简单易行的单字节指令。

(2). Pc+1的时间问题。由于直接在第一个节拍进行pc+1会造成错误，因此在请教了老师之后，将pc+1放在第二个节拍的下降沿进行，保证取值时候pc的正确。代码如下：

if c2='0' and c2'event then

PC\_T <= PC\_T+1;

end if;

(3). 寄存器类型定义的问题。开始的时候不知道寄存器该怎么定义，通过查资料知道了可以采用如下方式进行定义：

type reg IS array(0 to 7) of std\_logic\_vector(7 downto 0);

signal R : reg;

首先定义一个reg类型，该类型为一个8\*8的二维数组，然后利用这个数组声明一个寄存器组R，实现了通用寄存器的定义。

(4). 寄存器访问的问题。定义了寄存器之后，不知道该怎么对指定的寄存器进行操作，后来通过请教同学知道了可以采用如下方法：

R(conv\_integer(IR\_O(10 downto 8))

利用conv\_integer()函数将二进制数转化为整形数。

2. 仿真过程

(1). 仿真过程出现的最大的问题就是运算模块的时序问题。由于开始设计的时候对vhdl语言并行语句和顺序语句的性质不是很熟悉，而且对信号的赋值时机的知识没有掌握好，导致运算模块出现了各种各样的问题，如输出结果延后一周期，Cy不能正确赋值等。后来通过对这些知识的熟悉，调整了语句的顺序，并且对赋值后立即就要用的量，比如操作码，采取变量方式进行存储，解决了运算模块出现的各种各样的问题，通过了仿真；

(2). 仿真过程中的第二个问题出现在访存控制模块。由于对inout类型的信号怎么使用不是很熟悉，导致在向inout信号写入的时候仿真出现错误。后来在老师的指导下，通过设置高阻的方式，成功通过了仿真。

3. 下载过程

(1). 下载过程遇到的第一个问题，是写入寄存器的时候，每次总是写入固定的一个寄存器，经过分析，觉得是实验台的问题。更换了实验台，问题得到了解决。

(2). 下载过程中遇到的最大的问题，是Cy和Z的两个标志位的问题。由于硬件实验台和软件仿真的区别，很多在仿真的时候没有暴漏出来的问题在下载的时候暴露了出来。首先是stc和clc这两条指令，在仿真的时候，没有看出来当时赋值的是alu（8）而不是tc，导致stc和clc这两条指令无效，后来经过检查代码改了过来；后来做带进位加法的时候发现本周期产生的进位会加到本周期的计算结果中，于是继续对时序进行更改，解决了这个问题；后来又发现，Z在置位之后无法变为0，检查代码发现，每次运算的时候只是对Z需要置位的情况进行了处理而没有对Z置为零的情况进行处理，于是加上了Z要置为0的情况。