

合肥工业大学

计算机与信息学院

计算机组成原理实验报告

专 业 班 级

物联网 18-1 班

学生姓名及学号

郭子煜 2018217845

实 验 地 点

综合实验楼 312

2020~2021 学年第 1 学期

目录

一、实验目的	3
二、实验内容	3
非访存指令	3
访存指令	3
转移类指令	3
三、实验原理	3
四、实验步骤	4
1. CPU 各模块 Verilog 实现	5
2. CPU 顶层文件封装实现	6
3. CPU 模拟仿真	6
1) TestBench 测试文件	6
2) ModelSim 仿真及分析	6
指令 1 LDA 取数指令	6
指令 2 COM 取反指令	7
指令 3 SHR 算数右移指令	7
指令 4 CSL 循环左移指令	8
指令 5 CLA 清除指令	8
指令 6 ADD 加法指令	9
指令 7 STA 存数指令	9
指令 8 JMP 无条件转移指令	10
指令 9 BAN 条件转移指令	10
指令 10 STP 停机指令	11
五. 源代码	11
六. 总结及感想	18

实验五、单周期 CPU 设计与实现——十条指令 CPU

一、实验目的

通过设计并实现支持 10 条指令的 CPU，进一步理解和掌握 CPU 设计的基本原理和过程。

二、实验内容

设计和实现一个支持如下十条指令的单周期 CPU。

非访存指令：

清除累加器指令 CLA

累加器取反指令 COM

算术右移一位指令 SHR：将累加器 ACC 中的数右移一位，结果放回 ACC

循环左移一位指令 CSL：对累加器中的数据进行操作

停机指令 STP

访存指令：

加法指令 ADD X: $[X] + [ACC] \rightarrow ACC$, X 为存储器地址，直接寻址

存数指令 STA X, 采用直接寻址方式

取数指令 LDA X, 采用直接寻址

转移类指令：

无条件转移指令 JMP imm: $\text{signExt}(\text{imm}) \rightarrow PC$

有条件转移（负则转）指令 BAN X: ACC 最高位为 1 则 $(PC) + X \rightarrow PC$,

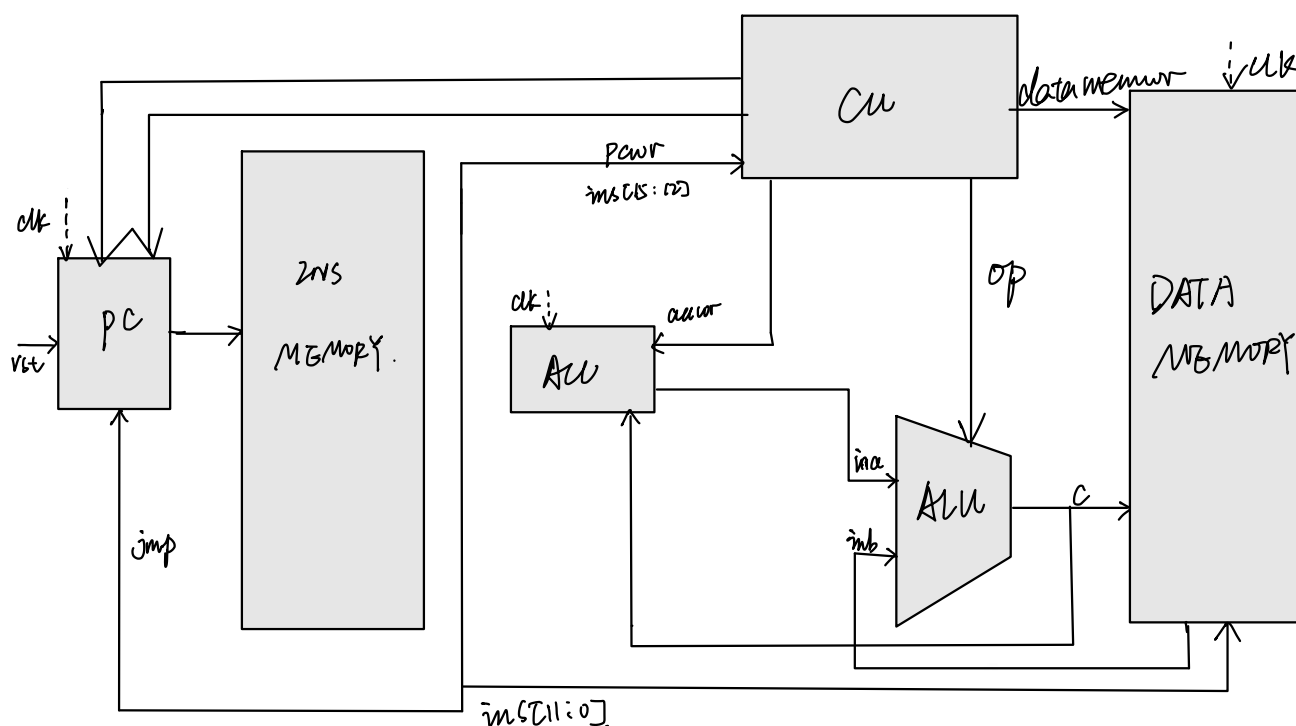
否则 PC 不变

三、实验原理

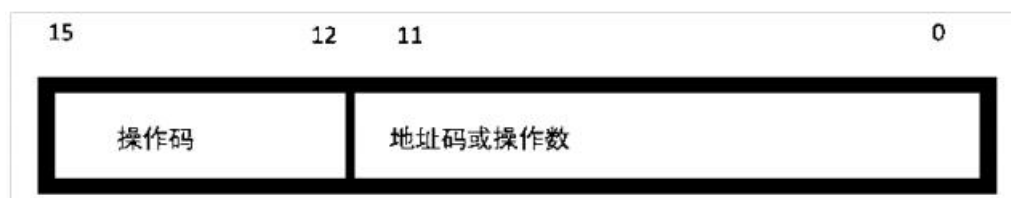
1. 单周期 CPU 是指所有指令均在一个时钟周期内完成的 CPU。CPU 由数据通路及其控制部件两部分构成，因而要完成一个支持十条指令 CPU 的设计，需要依次完成以下两件事：

- 1) 根据指令功能和格式设计 CPU 的数据通路；
- 2) 根据指令功能和数据通路设计控制部件。

CPU 的原理图：



2. 约定机器字长、指令字长和存储字长均为 16 位。
3. 约定指令格式高 4 位为指令的操作码字段，低 12 位为指令的地址码或者立即寻址的操作数，如下图所示。



4. 指令和相应的操作码对照表

CLA	0000	ADD	0100
COM	0001	STA	0101
SHR	0010	LDA	0110
CSL	0011	JMP	0111
STP	1111	BAN	1000

四、实验步骤

在第三部分通过该 CPU 实现细节的分析、设计，并得到该 CPU 的原理图后，就可以依次实现各个模块，并进行仿真验证了。

1. CPU 各模块 Verilog 实现

在前面实验中，已经分别设计和实现了 PC、指令存储器、寄存器组和 ALU，这里只给出各个模块的功能描述及其接口定义，具体实现可以直接使用或者调整前面试验的实现代码。

(1) PC 模块

输入：时钟信号 `clk`、重置信号 `rst`、停机信号 `stop`、无条件转移信号 `wr`、条件转移信号 `judge`、pc 修改量 `jmp`(12 位)

输出：指令地址 `addr`(12 位)

功能：每个时钟上升沿 `addr` 的值自动加 1，并输出；

如果 `wr` 为 1，修改 `addr` 为 `jmp - 1`；

如果 `judge` 为 1，修改 `addr` 为 `addr + jmp`；

如果 `rst` 为 1，修改 `addr` 为 0；

(2) 指令存储器模块

输入：12 位指令地址 `addr`

输出：16 位指令 `ins`

功能：存放待执行的指令(初始化)，并根据地址输出指令。

(3) 累加器

输入：时钟信号 `clk`、读写控制线 `wr`、输入数据 `indata`

输出：输出数据 `outdata`

功能：ACC 数据从 `outdata` 输出，如果 `wr` 有效，在 `clk` 下降沿将会把 `indata` 写入 ACC 中。

(4) ALU

输入：操作数 `ina` 和 `inb`、操作选择信号 `alu_op`

输出：ALU 运算结果 `z`

功能：根据操作选择信号计算 `ina` 和 `inb` 的运算结果 `z`

(5) 控制单元

输入：指令（操作码）

输出：停机信号 `stop`、PC 读写控制线 `pcwr`、ACC 读写控制线 `accwr`、数 5

据存储器读写控制线 `datamemorywr`、ALU 的操作选择信号 `op`

功能：根据当前指令功能对 stop、pcwr、accwr、datamemorywr 和 op 赋值

(6) 数据存储模块

输入： 时钟信号 clk、读写控制线 wr、12 位指令地址 addr、输入数据 indata

输出： 16 位输出数据 outdata

功能： 存放数据，可以读出，或在 clk 下降沿写入。

2. CPU 顶层文件封装实现

在这个文件中，声明了 pc, insmemory, datamemory, acc, cu, alu 等模块。并且声明了几个 wire 类型的变量用于连接各个部件。

3. CPU 模拟仿真

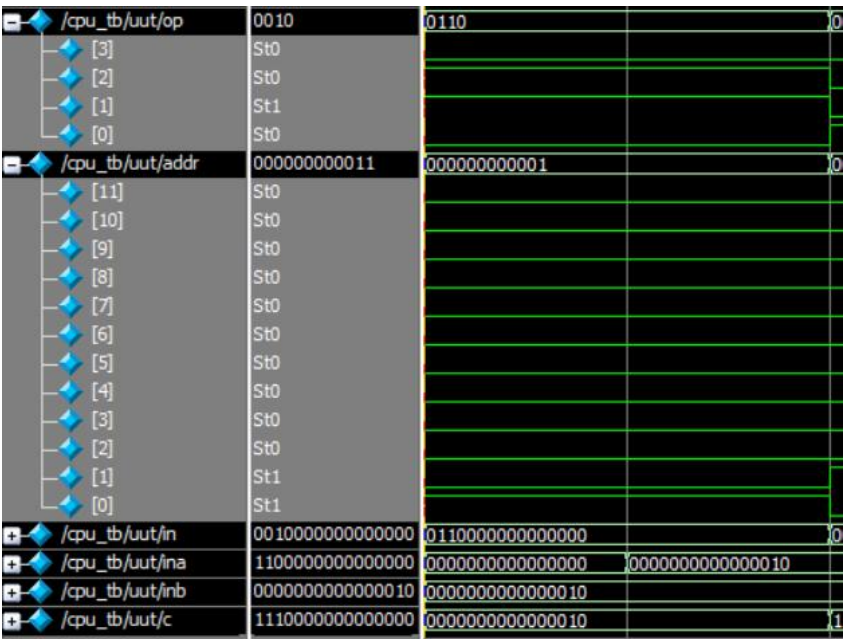
1) TestBench 测试文件

CPU 的测试文件的主要功能是产生了时钟信号，并且使用 rst 信号初始化 pc。clk 信号每 1ns 反转一次，一个时钟周期 2ns。20ns 后结束测试，刚好测试完 10 条指令

2) ModelSim 仿真及分析

指令 1 LDA 取数指令

这条指令在指令存储器的地址 000000000000 处。传送给 ALU 的命令是 0110。将数据地址为 000000000000 处的数据 0000000000000010 存储进 ACC 中。CU 会向 ACC 发出写信号，在时钟的下降沿写入数据。



指令 2 COM 取反指令

这条指令在指令存储器的地址 000000000001 处。传送给 ALU 的命令是 0001。将 ACC 中的数据 0000000000000010 取反后得到 1111111111111101。CU 会向 ACC 发出写信号，在时钟的下降沿写入数据。

/cpu_tb/uut/op		0010	0001	
[3]	St0			
[2]	St0			
[1]	St1			
[0]	St0			
/cpu_tb/uut/addr		000000000011	000000000010	
[11]	St0			
[10]	St0			
[9]	St0			
[8]	St0			
[7]	St0			
[6]	St0			
[5]	St0			
[4]	St0			
[3]	St0			
[2]	St0			
[1]	St1			
[0]	St1			
/cpu_tb/uut/in		0010000000000000	0001000000000000	
/cpu_tb/uut/ina		1100000000000000	0000000000000010	1111111111111101
/cpu_tb/uut/inb		0000000000000010	0000000000000010	
/cpu_tb/uut/c		1110000000000000	1111111111111101	0000000000000010

指令 3 SHR 算数右移指令

这条指令在指令存储器的地址 000000000010 处。传送给 ALU 的命令是 0010。将 ACC 中的数据 1111111111111101 算数右移后 1111111111111110。CU 会向 ACC 发出写信号，在时钟的下降沿写入数据。

/cpu_tb/uut/op		0010	0010	
[3]	St0			
[2]	St0			
[1]	St1			
[0]	St0			
/cpu_tb/uut/addr		000000000011	000000000011	
[11]	St0			
[10]	St0			
[9]	St0			
[8]	St0			
[7]	St0			
[6]	St0			
[5]	St0			
[4]	St0			
[3]	St0			
[2]	St0			
[1]	St1			
[0]	St1			
/cpu_tb/uut/in		0010000000000000	0010000000000000	
/cpu_tb/uut/ina		1100000000000000	1111111111111101	1111111111111110
/cpu_tb/uut/inb		0000000000000010	0000000000000010	
/cpu_tb/uut/c		1110000000000000	1111111111111110	1111111111111111

指令 4 CSL 循环左移指令

这条指令在指令存储器的地址 000000000011 处。传送给 ALU 的命令是 0011。将 ACC 中的数据 111111111111110 循环左移后 111111111111101。CU 会向 ACC 发出写信号，在时钟的下降沿写入数据。

/cpu_tb/uut/op		0010	0011	
[3]	St0			
[2]	St0			
[1]	St1			
[0]	St0			
/cpu_tb/uut/addr		000000000011	000000000100	
[11]	St0			
[10]	St0			
[9]	St0			
[8]	St0			
[7]	St0			
[6]	St0			
[5]	St0			
[4]	St0			
[3]	St0			
[2]	St0			
[1]	St1			
[0]	St1			
/cpu_tb/uut/in		0010000000000000	0010000000000000	
/cpu_tb/uut/ina		1100000000000000	1111111111111110	111111111111101
/cpu_tb/uut/inb		0000000000000010	0000000000000010	
/cpu_tb/uut/c		1110000000000000	111111111111101	111111111111011













指令 5 CLA 清除指令

这条指令在指令存储器的地址 000000000100 处。传送给 ALU 的命令 0000。将 ACC 中的数据 111111111111101 清零后得到 0000000000000000。CU 会向 ACC 发出写信号，在时钟的下降沿写入数据。

/cpu_tb/uut/op		0010	0000	
[3]	St0			
[2]	St0			
[1]	St1			
[0]	St0			
/cpu_tb/uut/addr		000000000011	000000000101	
[11]	St0			
[10]	St0			
[9]	St0			
[8]	St0			
[7]	St0			
[6]	St0			
[5]	St0			
[4]	St0			
[3]	St0			
[2]	St0			
[1]	St1			
[0]	St1			
/cpu_tb/uut/in		0010000000000000	0000000000000000	
/cpu_tb/uut/ina		1100000000000000	111111111111101	0000000000000000
/cpu_tb/uut/inb		0000000000000010	0000000000000010	
/cpu_tb/uut/c		1110000000000000	0000000000000000	













指令 6 ADD 加法指令

这条指令在指令存储器的地址 000000000101 处。传送给 ALU 的命令是 0100。将 ACC 中的数据 0000000000000000 和数据存储器的地址为 000000000001 处的数据相加得到 1000000000000000 再保存到 ACC 中。CU 会向 ACC 发出写信号，在时钟的下降沿写入数据。

		/cpu_tb/uut/op	0010	0100	
		[3]	St0		
		[2]	St0		
		[1]	St1		
		[0]	St0		
		/cpu_tb/uut/addr	00000000011	000000000110	
		[11]	St0		
		[10]	St0		
		[9]	St0		
		[8]	St0		
		[7]	St0		
		[6]	St0		
		[5]	St0		
		[4]	St0		
		[3]	St0		
		[2]	St0		
		[1]	St1		
		[0]	St1		
		/cpu_tb/uut/in	0010000000000000	0100000000000001	
		/cpu_tb/uut/ina	1100000000000000	0000000000000000	1000000000000000
		/cpu_tb/uut/inb	0000000000000010	1000000000000000	
		/cpu_tb/uut/c	1110000000000000	1000000000000000	0000000000000000

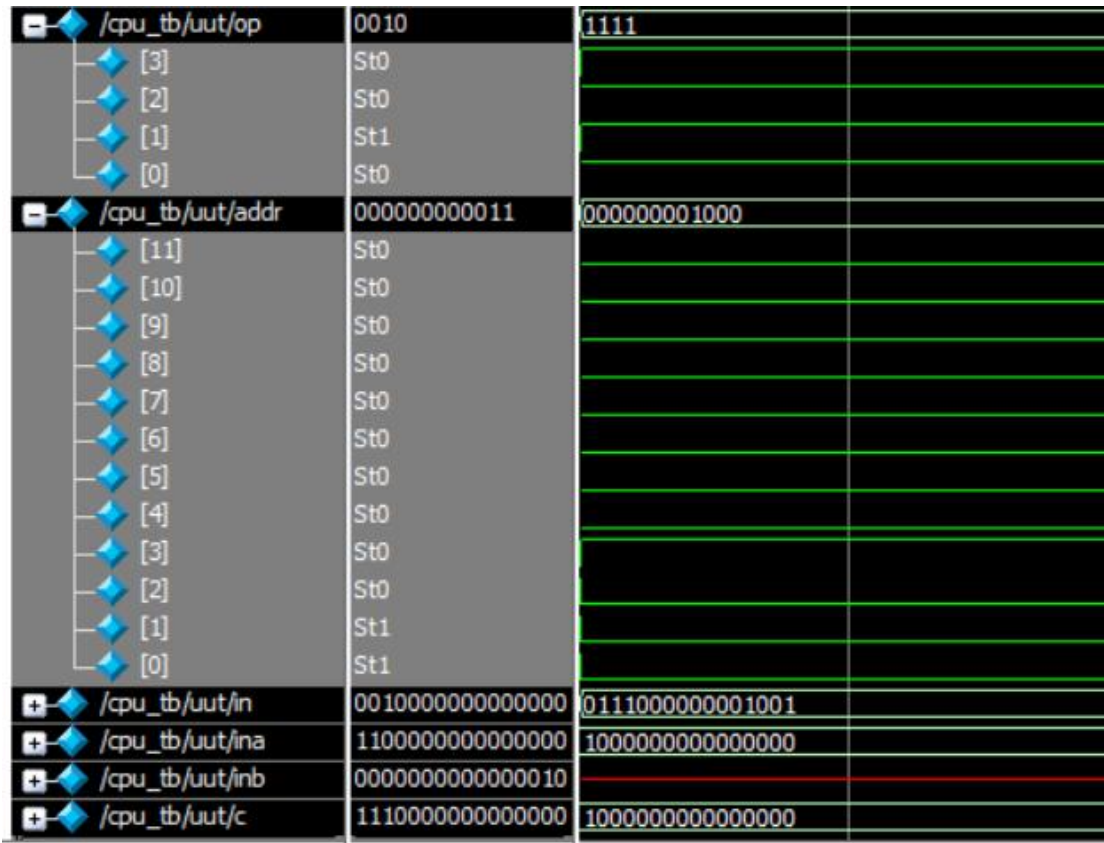
指令 7 STA 存数指令

这条指令在指令存储器的地址 000000000110 处。传送给 ALU 的命令是 0101。ACC 中的数据存储在数据存储器地址为 000000000010 的单元。可以从图中看到 000000000010 单元的值由不确定变为 1000000000000000。CU 会向数据存储器发出写信号，在时钟的下降沿写入数据。

		/cpu_tb/uut/op	0010	0101	
		[3]	St0		
		[2]	St0		
		[1]	St1		
		[0]	St0		
		/cpu_tb/uut/addr	00000000011	000000000111	
		[11]	St0		
		[10]	St0		
		[9]	St0		
		[8]	St0		
		[7]	St0		
		[6]	St0		
		[5]	St0		
		[4]	St0		
		[3]	St0		
		[2]	St0		
		[1]	St1		
		[0]	St1		
		/cpu_tb/uut/in	0010000000000000	0101000000000010	
		/cpu_tb/uut/ina	1100000000000000	1000000000000000	
		/cpu_tb/uut/inb	0000000000000010		1000000000000000
		/cpu_tb/uut/c	1110000000000000	1000000000000000	

指令 8 JMP 无条件转移指令

这条指令在指令存储器的地址 000000000111 处。传送给 ALU 的命令是 1111。这条指令使用地址码采用立即寻址方式向 PC 传送跳转的目标地址。这里我们跳转到指令存储器地址为 000000001001 的单元(该单元保存了条件转移指令)。可以看到下降沿的时候 PC 的值被修改为 000000001000 而不是 000000001001，因为下一个时钟周期上升沿 PC 自动加 1 会进行修正。CU 会向 PC 发出写信号，在时钟的下降沿写入数据。



指令 9 BAN 条件转移指令

这条指令在指令存储器的地址 000000001001 处。传送给 ALU 的命令是 0111。这里我这条指令设置为会判断 ACC 最高位是否位 1，如果为 1，会把 PC 增加 2。注意这里不需要进行 PC 的修正，直接在下降沿将 PC 加 2 修改 000000001011。因为在相对寻址中，位移量是以条件转移指令的下一条指令的地址为基准的做偏移的。而指令字长刚好是一个存储字，所以最终会跳转到地址为 000000001100 的指令存储器单元。该单元内的指令为停机指令。ALU 会根据判断的结果向 PC 发出写信号 judge，如果为 1，在时钟的下降沿修改 PC。

/cpu_tb/uut/op		0010	1111	0111	0001
[3]	St0				
[2]	St0				
[1]	St1				
[0]	St0				
/cpu_tb/uut/addr		00000000011	000000001000	000000001001	00000000010
[11]	St0				
[10]	St0				
[9]	St0				
[8]	St0				
[7]	St0				
[6]	St0				
[5]	St0				
[4]	St0				
[3]	St0				
[2]	St0				
[1]	St1				
[0]	St1				
/cpu_tb/uut/in		0010000000000000	0111000000001001	1000000000000010	0001000000000000
/cpu_tb/uut/ina		1100000000000000	1000000000000000		
/cpu_tb/uut/inb		0000000000000010		1000000000000000	0000000000000010
/cpu_tb/uut/c		1110000000000000	1000000000000000		0111111111111111

指令 10 STP 停机指令

这条指令在指令存储器的地址 000000001001 处。传送给 ALU 的命令是 1111。这条指令会将 PC 内的标志位 state 设置为 0。如果 state 为 0 的话，PC 在每个时钟上升沿将不会自动加 1。在图中可以看到下一个时钟上升沿到来的时候，地址保持 000000001001 无变化。CU 会向 PC 发出停机信号 STOP，使 PC 不再自动增加

/cpu_tb/uut/op		0010	1111	0111	0001	0010	0001	0010
[3]	St0							
[2]	St0							
[1]	St1							
[0]	St0							
/cpu_tb/uut/addr		00000000011	000000001000	000000001001	00000000010	00000000011	00000000010	00000000011
[11]	St0							
[10]	St0							
[9]	St0							
[8]	St0							
[7]	St0							
[6]	St0							
[5]	St0							
[4]	St0							
[3]	St0							
[2]	St0							
[1]	St1							
[0]	St1							
/cpu_tb/uut/in		0010000000000000	0111000000001001	1000000000000010	0001000000000000	0001000000000000	0001000000000000	0001000000000000
/cpu_tb/uut/ina		1100000000000000	1000000000000000		1100000000000000			
/cpu_tb/uut/inb		0000000000000010		1000000000000000	0000000000000010			
/cpu_tb/uut/c		1110000000000000	1000000000000000	0111111111111111	1100000000000000	0011111111111111	0011111111111111	1110000000000000

五. 源代码

pc. v

```
module pc(
    input wire clk,rst,stop,wr,judge,
    input wire[11:0] jmp,
```

```

output reg[11:0]addr
);
reg state;
initial begin
state=1;
addr=0;
end
always@*begin
if(stop==1)
state=0;
if(rst==1)
addr=0;
end
always@(posedge clk)begin
if(state==1)
addr=addr+1;
end
always@(negedge clk)begin
if(wr==1)
addr=jump-1;
if(judge==1)
addr=addr+jump-1;
end
endmodule

```

insmemory.v

```

module insmemory(
input wire[11:0]addr,
output reg[15:0]ins
);

```

```

reg[15:0]unit[12'b1111111111:0];

initial begin
unit[1]=16'b0110000000000000;//LDA 取数
unit[2]=16'b0001000000000000;//COM 累加器取反
unit[3]=16'b0010000000000000;//SHR 算术右移
unit[4]=16'b0011000000000000;//CSL 循环左移
unit[5]=16'b0000000000000000;//CLA 清除累加器
unit[6]=16'b0100000000000001;//ADD 加法
unit[7]=16'b0101000000000010;//STA 存数
unit[8]=16'b0111000000001001;//JMP 无条件转移
unit[9]=16'b1000000000000010;//BAN 条件转移 符合地址加 2
unit[12]=16'b1111000000000000;//STP 停机
end

always@*begin
ins=unit[addr];
end

endmodule

```

acc. v

```

module acc(
input wire clk,wr,
input wire[15:0]indata,
output wire[15:0]outdata
);
reg[15:0]acc;
assign outdata=acc;
initial begin acc=0;
end

always@(negedge clk)begin
if(wr==1)

```

```

acc=indata;

end

endmodule

```

alu.v

```

module alu(

    input wire[3:0]op,

    input wire[15:0]ina,inb,

    output reg judge,

    output reg[15:0]z

);

    initial begin

        judge=0;

        z=0;

    end

    always@*begin

        case(op)

            4'b0000:z=0;//CLA

            4'b0001:z=~ina;//COM

            4'b0010:z=ina[15]==1?{1'b1,ina[15:1]}:{1'b0,ina[15:1]};//SHR

            4'b0011:z={ina[14:0],ina[15]};//CSL

            4'b0100:z=ina+inb;//ADD

            4'b0101:z=ina;//STA

            4'b0110:z=inb;//LDA

            4'b0111: judge=ina[15]==1?1:0;//BAN

            4'b1111:;//NOP

        endcase

    end

endmodule

```

cu. v

```
module cu(  
  
    input wire[3:0]ins,  
  
    output reg stop,pcwr,accwr,datamemwr,  
  
    output reg[3:0]op  
  
);  
  
    initial begin stop=0;  
  
    pcwr=0;  
  
    accwr=0;  
  
    datamemwr=0;  
  
    op=4'b0000;//addr 从 0 开始执行, 执行下一条指令时加一  
  
end  
  
    always@(ins)begin  
  
        case(ins)  
  
            4'b0000:{stop,pcwr,accwr,datamemwr,op}=8'b00100000;//CLA  
            4'b0001:{stop,pcwr,accwr,datamemwr,op}=8'b00100001;//COM  
            4'b0010:{stop,pcwr,accwr,datamemwr,op}=8'b00100010;//SHR  
            4'b0011:{stop,pcwr,accwr,datamemwr,op}=8'b00100011;//CSL  
            4'b1111:{stop,pcwr,accwr,datamemwr,op}=8'b10001111;//STP  
            4'b0100:{stop,pcwr,accwr,datamemwr,op}=8'b00100100;//A00  
            4'b0101:{stop,pcwr,accwr,datamemwr,op}=8'b00010101;//STA  
            4'b0110:{stop,pcwr,accwr,datamemwr,op}=8'b00100110;//LDA  
            4'b0111:{stop,pcwr,accwr,datamemwr,op}=8'b01001111;//JMP  
            4'b1000:{stop,pcwr,accwr,datamemwr,op}=8'b01000111;//BAN  
  
        endcase  
  
    end  
  
endmodule
```

datamemory. v

```
module datamemory(  

```

```

input wire wr,clk,

input wire[11:0]addr,

input wire[15:0]indata,

output wire[15:0]outdata

);

reg[15:0] unit[12'b111111111111:0];

assign outdata=unit[addr];

initial begin

unit[0]=16'b0000000000000010;//初始化

unit[1]=16'b1000000000000000;//最高位初始化为 1

end

always@(negedge clk)begin

if(wr==1)

unit[addr]=indata;

end

endmodule

```

cpu. v

```

module cpu(

input wire clk,rst

);

wire stop,pcwr,accwr,datamemwr,judge;

wire[3:0]op;

wire[11:0]addr;

wire[15:0]in,ina,inb,c;

pc pc(

.clk(clk),.rst(rst),.stop(stop),.wr(pcwr), .judge(judge),

.jump(in[11:0]),.addr(addr)

);

insmemory insmemory(

```



```

    .addr(addr), .ins(in)
  );
acc acc(
  .clk(clk), .wr(accwr),
  .indata(c), .outdata(ina)
);
datamemory datamemory(
  .wr(datamemwr), .clk(clk),
  .addr(in[11:0]),
  .indata(c),
  .outdata(inb)
);
cu cu(
  .ins(in[15:12]
),
  .stop(stop), .pcwr(pcwr), .accwr(accwr), .datamemwr(datamemwr),
  .op(op)
);
alu alu(
  .op(op),
  .ina(ina), .inb(inb),
  .judge(judge),
  .z(c)
);
endmodule

```

cpu_tb.v

```

module cpu_tb;

reg clk, rst;

initial begin

```

```

clk=1;

rst=1;

#1rst=0;

#20$stop;

end

always #1 clk=~clk;

cpu uut(
    .clk(clk),.rst(rst)
);

endmodule

```

六. 总结及感想

通过实验，我掌握了一些 Verilog 语言的基本规则，尝试写了 4 位全加器、8 位比较器、74138 译码器、ALU、RAM、PC、指令存储器、寄存器堆等模块的代码，并进行了仿真，学会了如何看仿真图像。这个实验设计了一个单周期 CPU 其实是之前实验的综合。感谢三位老师的帮助让我完成了这个实验，希望我再往后的学习中还可以通过自己的努力进行改进提高。