CPU设计汇报

湛林莉　智能1601 2016080103 24

相对简单CPU的设计需求：

1、地址总线16位，数据总线8位

2、有一个8位累加寄存器AC，一个8位通用寄存器R，一个1位的零标志

3、有一个16位AR寄存器，一个16位程序计数器PC，一个8位数据寄存器DR，一个8位指令寄存器IR，一个8位临时寄存器TR

4、有16条指令，每条指令1个或3个字节，其中操作码8位。3字节的指令有16位的地址

相对简单CPU的设计思路：

1. 指令执行过程分为取指、译码、执行三个阶段

2. 取指包括三个状态，FETCH1，FETCH2,FETCH3

3. 译码体现为从FETCH3状态到各指令执行状态序列的第一个状态

4. 执行根据指令的具体操作分为若干状态

5. 执行的最后一个状态转移到FETCH1状态

6. 控制器根据每个状态需要完成的操作产生相应的控制信号

指令	指令码	操作
NOP	0000 0000	无
LDAC	0000 0001 Γ	AC←M[Γ]
STAC	0000 0010 Γ	M[Γ]←AC
MVAC	0000 0011	R←AC
MOVR	0000 0100	AC←R
JUMP	0000 0101 Γ	GOTO Γ
JMPZ	0000 0110 Γ	IF（Z=1） THEN GOTO Γ
JPNZ	0000 0111 Γ	IF（Z=0） THEN GOTO Γ

指令码：

ADD	0000 1000	AC←AC＋R，IF（AC＋R＝0） THEN Z←1 ELSE Z←0
SUB	0000 1001	AC←AC－R，IF（AC－R＝0） THEN Z←1 ELSE Z←0
INAC	0000 1010	AC←AC＋1，IF（AC＋1＝0） THEN Z←1 ELSE Z←0
CLAC	0000 1011	AC←0，Z←1
AND	0000 1100	AC←AC∧R，IF（AC∧R＝0） THEN Z←1 ELSE Z←0
OR	0000 1101	AC←AC∨R，IF（AC∨R＝0） THEN Z←1 ELSE Z←0
XOR	0000 1110	AC←AC⊕R，IF（AC⊕R＝0） THEN Z←1 ELSE Z←0
NOT	0000 1111	AC←AC’，IF（AC’＝0） THEN Z←1 ELSE Z←0

各指令对应的具体状态（每个状态对应一个时钟周期）：

取指令和译码

FETCH1：AR←PC

FETCH2：DR←M，PC←PC＋1

FETCH3：IR←DR，AR←PC

STAC指令执行的是与LDAC完全相反的操作。

STAC1：DR←M，PC←PC＋1，AR←AR＋1

STAC2：TR←DR，DR←M，PC←PC＋1

STAC3：AR←DR，TR

STAC4：DR←AC

STAC5：M←DR

LDAC指令

第一个状态：

LDAC1：DR←M，PC←PC＋1，AR←AR＋1

第二个状态：

LDAC2：TR←DR，DR←M，PC←PC＋1

LDAC3：AR←DR，TR

LDAC4：DR←M

LDAC5：AC←DR

MVAC和MOVR指令

MVAC1：R←AC

MOVR1：AC←R

JUMP指令

JUMP1：DR←M，AR←AR+1

JUMP2：TR←DR，DR←M

JUMP3：PC←DR，TR

JMPZ和JPNZ指令

JMPZ指令的状态：

JMPZY1: DR←M，AR←AR＋1

JMPZY2: TR←DR，DR←M

JMPZY3: PC←DR，TR

JMPZN1: PC←PC＋1

JMPZN2: PC←PC＋1

PNZ指令的状态：

JPNZY1: DR←M，AR←AR＋1

JPNZY2: TR←DR，DR←M

JPNZY3: PC←DR，TR

PNZN1: PC←PC＋1

JPNZN2: PC←PC＋1

其余的指令都是在一个状态内完成的。

ADD1: AC←AC＋R，IF（AC＋R＝0）THEN Z←1 ELSE Z←0

SUB1: AC←AC－R，IF（AC－R＝0）THEN Z←1 ELSE Z←0

INAC1: AC←AC＋1，IF（AC＋1＝0）THEN Z←1 ELSE Z←0

CLAC1: AC←0，Z←1

AND1: AC←AC∧R，IF（AC∧R＝0）THEN Z←1 ELSE Z←0

OR1: AC←AC∨R，IF（AC∨R＝0）THEN Z←1 ELSE Z←0

XOR1: AC←AC⊕ R，IF（AC⊕ R＝0）THEN Z←1 ELSE Z←0

NOT1: AC←AC'，IF（AC'＝0）THEN Z←1 ELSE Z←0

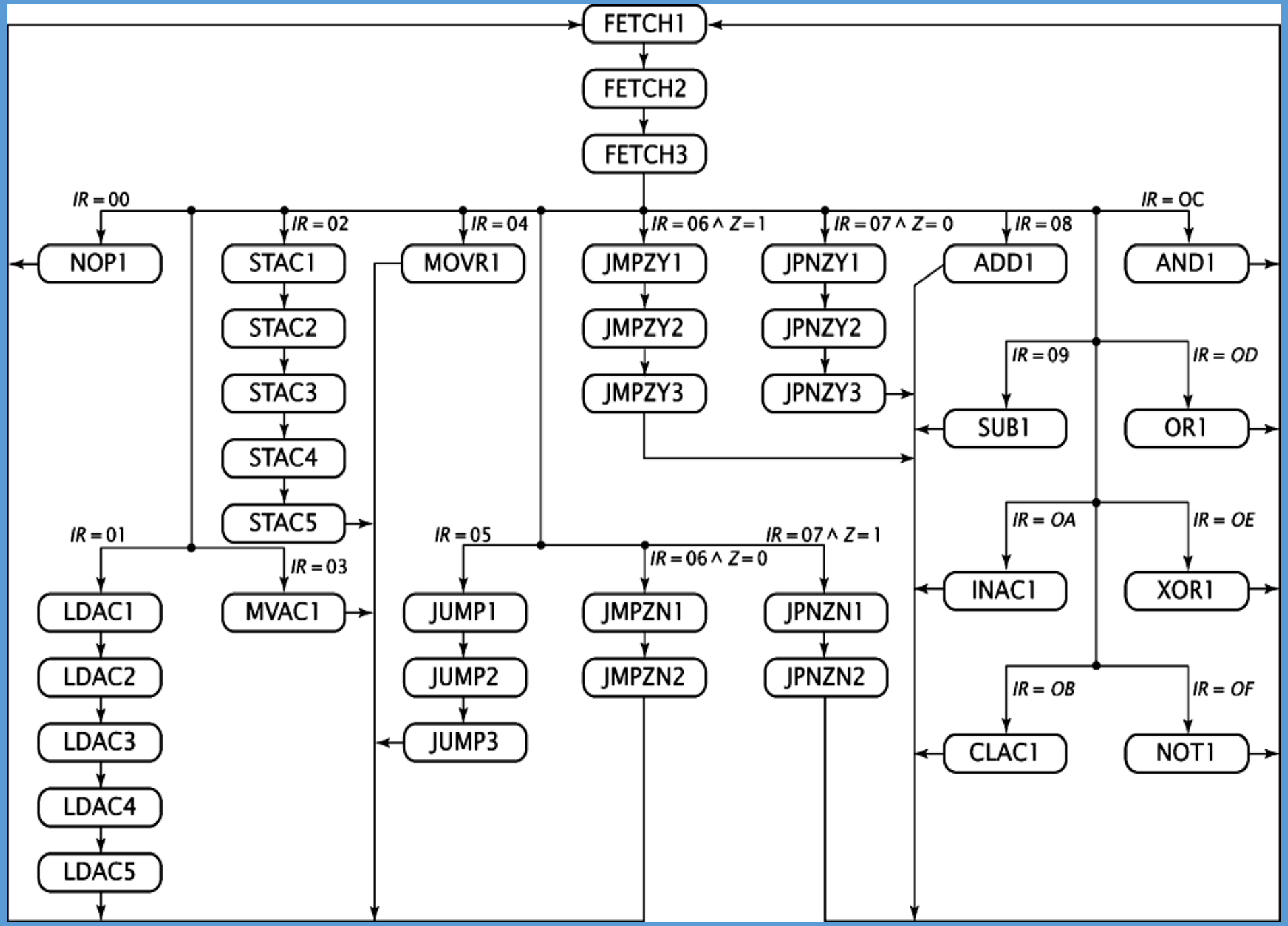
为每个具体的指令状态设计指令译码：

存在的问题及解决方法：
对于FETCH3状态对应的操作为“ir<--dr；ar<--pc”，也就是说在这个过程中需要打开dr_bus和pc_bus；若在同一个时钟周期执行该过程，则有两个数据（寄存器PC和DR的值）进入总线，会引起数据的混乱，所以我就把原来的FETCH3分成了FETCH3和FETCH4两个状态

```
constant fetch1:    std_logic_vector(5 downto 0) := "000000";
constant fetch2:    std_logic_vector(5 downto 0) := "000001";
constant fetch3:    std_logic_vector(5 downto 0) := "000010";
constant fetch4:    std_logic_vector(5 downto 0) := "000011";
constant clac1:     std_logic_vector(5 downto 0) := "000100";
constant inac1:     std_logic_vector(5 downto 0) := "000101";
constant add1:      std_logic_vector(5 downto 0) := "000110";
constant sub1:      std_logic_vector(5 downto 0) := "000111";
constant and1:      std_logic_vector(5 downto 0) := "001000";
constant or1:       std_logic_vector(5 downto 0) := "001001";
constant xor1:      std_logic_vector(5 downto 0) := "001010";
constant not1:      std_logic_vector(5 downto 0) := "001011";
constant mvac1:     std_logic_vector(5 downto 0) := "001100";
constant movr1:     std_logic_vector(5 downto 0) := "001101";
constant ldac1:     std_logic_vector(5 downto 0) := "001110";
constant ldac2:     std_logic_vector(5 downto 0) := "001111";
constant ldac3:     std_logic_vector(5 downto 0) := "010000";
constant ldac4:     std_logic_vector(5 downto 0) := "010001";
constant ldac5:     std_logic_vector(5 downto 0) := "010010";
constant stac1:     std_logic_vector(5 downto 0) := "010011";
constant stac2:     std_logic_vector(5 downto 0) := "010100";
constant stac3:     std_logic_vector(5 downto 0) := "010101";
constant stac4:     std_logic_vector(5 downto 0) := "010110";
constant stac5:     std_logic_vector(5 downto 0) := "010111";
```

```
constant jump1:     std_logic_vector(5 downto 0) := "011000";
constant jump2:     std_logic_vector(5 downto 0) := "011001";
constant jump3:     std_logic_vector(5 downto 0) := "011010";

constant jmpzy1:    std_logic_vector(5 downto 0) := "011011";
constant jmpzy2:    std_logic_vector(5 downto 0) := "011100";
constant jmpzy3:    std_logic_vector(5 downto 0) := "011101";
constant jmpzn1:    std_logic_vector(5 downto 0) := "011110";
constant jmpzn2:    std_logic_vector(5 downto 0) := "011111";

constant jpnzy1:    std_logic_vector(5 downto 0) := "100000";
constant jpnzy2:    std_logic_vector(5 downto 0) := "100001";
constant jpnzy3:    std_logic_vector(5 downto 0) := "100010";
constant jpnzn1:    std_logic_vector(5 downto 0) := "100011";
constant jpnzn2:    std_logic_vector(5 downto 0) := "100100";

constant nop1: std_logic_vector(5 downto 0) :="111111";--no op
```

状态图：

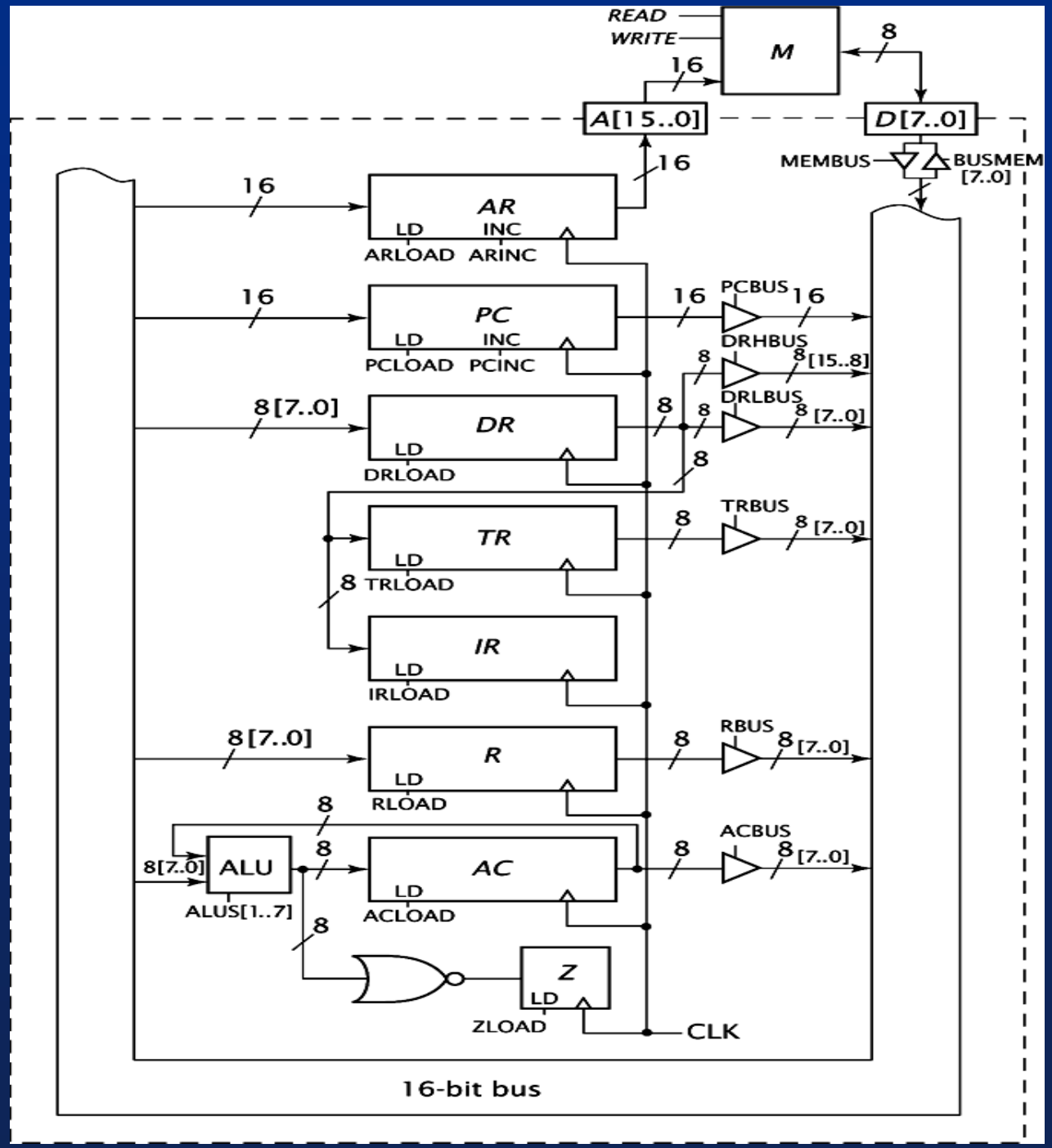

FETCH1
FETCH2
FETCH3
IR = 00
NOP1
IR = 02
STAC1
STAC2
STAC3
STAC4
STAC5
IR = 04
MOVR1
IR = 06 ∧ Z = 1
JMPZY1
JMPZY2
JMPZY3
IR = 07 ∧ Z = 0
JPNZY1
JPNZY2
JPNZY3
IR = 08
ADD1
IR = 0C
AND1
IR = 09
SUB1
IR = 0D
OR1
IR = 0A
INAC1
IR = 0E
XOR1
IR = 0B
CLAC1
IR = 0F
NOT1
IR = 01
LDAC1
LDAC2
LDAC3
LDAC4
LDAC5
IR = 03
MVAC1
IR = 05
JUMP1
JUMP2
JUMP3
IR = 06 ∧ Z = 0
JMPZN1
JMPZN2
IR = 07 ∧ Z = 1
JPNZN1
JPNZN2

根据状态图设计从当前状态变化到下一个状态的选择过程：

```
-- generate control signals for each st
for_nextstate: process(state, ir, z)
begin

    if(state=fetch1)     then    nextsta
    elsif(state=fetch2)  then    nextsta
    elsif(state=fetch3)  then    nextsta
    elsif(state=fetch4)  then
        if(ir=RSCLAC)        then    nex
        elsif(ir=RSINAC)     then    nex
        elsif(ir=RSADD)      then    nex
        elsif(ir=RSSUB)      then    nex
        elsif(ir=RSAND)      then    nex
        elsif(ir=RSOR)       then    nex
        elsif(ir=RSXOR)      then    nex
        elsif(ir=RSNOT)      then    nex
        elsif(ir=RSMVAC)     then    nex
        elsif(ir=RSMOVR)     then    nex
        elsif(ir=RSLDAC)     then    nex
        elsif(ir=RSSTAC)     then    nex
        elsif(ir=RSJUMP)     then    nex

        elsif(ir=RSJMPZ)     then
            if(z='1')   then
                nextstate<=jmpzy1;
            else
                nextstate<=jmpzn1;
            end if;
        elsif(state=ldac4)   then    nextstate<=ldac5;
        elsif(state=ldac5)   then    nextstate<=fetch1;
        --store
        elsif(state=stac1)   then    nextstate<=stac2;
        elsif(state=stac2)   then    nextstate<=stac3;
        elsif(state=stac3)   then    nextstate<=stac4;
        elsif(state=stac4)   then    nextstate<=stac5;
        elsif(state=stac5)   then    nextstate<=fetch1;
        --jmp
        elsif(state=jump1)   then    nextstate<=jump2;
        elsif(state=jump2)   then    nextstate<=jump3;
        elsif(state=jump3)   then    nextstate<=fetch1;
        --jmpz
        elsif(state=jmpzy1)  then    nextstate<=jmpzy2;
        elsif(state=jmpzy2)  then    nextstate<=jmpzy3;
        elsif(state=jmpzy3)  then    nextstate<=fetch1;
        elsif(state=jmpzn1)  then    nextstate<=jmpzn2;
        elsif(state=jmpzn2)  then    nextstate<=fetch1;
        --jpnz
        elsif(state=jpnzy1)  then    nextstate<=jpnzy2;
        elsif(state=jpnzy2)  then    nextstate<=jpnzy3;
        elsif(state=jpnzy3)  then    nextstate<=fetch1;
        elsif(state=jpnzn1)  then    nextstate<=jpnzn2;
        elsif(state=jpnzn2)  then    nextstate<=fetch1;

        end if;

end process for_nextstate;
```

```
:RSJPNZ)     then
:'0')    then
extstate<=jpnzy1;

extstate<=jpnzn1;
.f;

extstate<=fetch1;


:1)      then    nextstate<=fetch1;
.nac1)   then    nextstate<=fetch1;

idd1)    then    nextstate<=fetch1;
ub1)     then    nextstate<=fetch1;
ind1)    then    nextstate<=fetch1;
)r1)     then    nextstate<=fetch1;
:or1)    then    nextstate<=fetch1;
iot1)    then    nextstate<=fetch1;

ivac1)   then    nextstate<=fetch1;
iovr1)   then    nextstate<=fetch1;

elsif(state=ldac1)   then    nextstate<=ldac2;
elsif(state=ldac2)   then    nextstate<=ldac3;
elsif(state=ldac3)   then    nextstate<=ldac4;
```

相对简单CPU设计图：


READ
WRITE
M
16
8
A[15..0]
D[7..0]
MEMBUS
BUSMEM
[7..0]
16
AR
LD
INC
ARLOAD
ARINC
PC
PCLOAD
PCINC
PCBUS
DRHBUS
[15..8]
8[7..0]
DR
DRLOAD
DRLBUS
[7..0]
TR
TRLOAD
TRBUS
IR
IRLOAD
R
RLOAD
RBUS
ALU
ALUS[1..7]
AC
ACLOAD
ACBUS
Z
ZLOAD
CLK
16-bit bus

根据CPU设计图中的控制信号的值更新各个寄存器的值：

```
addrbus <= ar;
databus <= thebus(7 downto 0) when writel='1' else "ZZZZZZZZ";

--the bus
thebus<=pc                  when pcbus='1'      else    "ZZZZZZZZZZZZZZZZ";
thebus<="00000000"&databus  when membus='1'     else    "ZZZZZZZZZZZZZZZZ";
thebus<="00000000"&r        when rbus='1'       else    "ZZZZZZZZZZZZZZZZ";
thebus<="00000000"&ac       when acbus='1'      else    "ZZZZZZZZZZZZZZZZ";
thebus<=dr&tr               when(trbus='1' and drbus='1')   else    "ZZZZZZZZZZZZZZZZ";
thebus<="00000000"&dr       when (drbus='1' and trbus/='1')     else    "ZZZZZZZZZZZZZZZZ";
```

```
-- update pc, state and other registers
update_regs: process(clk)
begin
    if(rising_edge(clk)) then
        --update registers
        if(pcinc='1')   then pc<=std_logic_vector(unsigned(pc) + 1); end if;
        if(arinc='1')   then ar<=std_logic_vector(unsigned(ar) + 1); end if;
        if(arload='1')  then    ar<=thebus; end if;
        if(drload='1')  then    dr<=thebus(7 downto 0); end if;
        if(irload='1')  then    ir<=dr; end if;
        if(acload='1')  then
            ac<=alu;
            if(s="0001" or s="0010" or s="0011" or s="0100" or s="0101" or s="0110" or s
                if(alu="00000000")  then z<='1';
                else    z<='0';
                end if;
            end if;
        end if;
        if(rload='1')   then    r<=thebus(7 downto 0); end if;
        if(trload='1')  then    tr<=dr; end if;
        if(pcload='1')  then    pc<=thebus; end if;
        if(reset='1') then
            pc <= "0000000000000000";
            state <= fetch1;
        else
            state <= nextstate;
        end if;
    end if;
```

```
                                    when s="0000"   else
                            downto 0))) when s="0001"   else--ac+R
                            downto 0))) when s="0010"   else--ac-R
                                    when s="0011"   else--ac and R
                                    when s="0100"   else--ac or R
                                    when s="0101"   else--ac xor R
                                    when s="0110"   else--not ac
                                    when s="0111"   else--ac+1
                                    when s="1000"   else--ac=0
```

S[3..0]	指令	运算	Z
0000	直接传送	ALU=bus[7..0]	ALU=0 则 Z=1
0001	ADD	AC+R	
0010	SUB	AC-R	
0011	AND	AC ^ R	
0100	OR	AC I R	
0101	XOR	AC⊕R	
0110	NOT	~AC	
0111	INAC	AC++	
1000	CLAC	AC=0	

根据不同状态值对应的操作打开或关闭相应的控制信号（部分截图）：

```
-- generate control signals for each state
gen_controls: process(state)
begin
    if(state=fetch1)    then --AR<-- PC
        arload<='1';pcbus<='1';pcinc<='0';drload<='0';membus<='0';irload<='0';--ar<=pc
        rbus<='0';s<="0000";acload<='0';rload<='0';acbus<='0';arinc<='0';trbus<='0';drbus<='0';trload<='0';
        read<='0';writel<='0';write<='0';pcload<='0';
    elsif(state=fetch2) then --DR<--M PC<--PC+1
        arload<='0';pcbus<='0';pcinc<='1';drload<='1';membus<='1';irload<='0';-- dr<=m  pc<=pc+1
        rbus<='0';s<="0000";acload<='0';rload<='0';acbus<='0';arinc<='0';trbus<='0';drbus<='0';trload<='0';
        read<='1';writel<='0';write<='0';pcload<='0';
    elsif(state=fetch3) then    --IR<--DR
        arload<='0';pcbus<='0';pcinc<='0';drload<='0';membus<='0';irload<='1';--ir<=dr
        rbus<='0';s<="0000";acload<='0';rload<='0';acbus<='0';arinc<='0';trbus<='0';drbus<='0';trload<='0';
        read<='0';writel<='0';write<='0';pcload<='0';
    elsif(state=fetch4) then    -- AR<--PC
        arload<='1';pcbus<='1';pcinc<='0';drload<='0';membus<='0';irload<='0';--ar<=pc
        rbus<='0';s<="0000";acload<='0';rload<='0';acbus<='0';arinc<='0';trbus<='0';drbus<='0';trload<='0';
        read<='0';writel<='0';write<='0';pcload<='0';
    elsif(state=clac1)  then    --AC<--0 Z<--1
        arload<='0';pcbus<='0';pcinc<='0';drload<='0';membus<='0';irload<='0';--ac<=0   z<=1
        rbus<='0';s<="1000";acload<='1';rload<='0';acbus<='0';arinc<='0';trbus<='0';drbus<='0';trload<='0';
        read<='0';writel<='0';write<='0';pcload<='0';
    elsif(state=inac1)  then --AC<--AC+1
        arload<='0';pcbus<='0';pcinc<='0';drload<='0';membus<='0';irload<='0';--ac++ z change
        rbus<='0';s<="0111";acload<='1';rload<='0';acbus<='0';arinc<='0';trbus<='0';drbus<='0';trload<='0';
        read<='0';writel<='0';write<='0';pcload<='0';
```

设计存储器中指令序列:采用从1累加到n的程序作为测试输入，根据mem.vhd设计.mif文件

```
        CLAC
        STAC  total      } total = 0 ,  i = 0
        STAC  i

Loop:   LDAC  i
        INAC             } i = i + 1
        STAC  i

        MVAC
        LDAC  total
        ADD              } total = total + i
        STAC  total

        LDAC  n
        SUB
        JPNZ  Loop       } IF i≠n THEN GOTO Loop
total:
    i:
```

地址00011101（29）存放的是每次求和的结果即被加数；地址00011110（30）存放的另一个加数,地址00011111（31）存放的是n的取值

CLAC:00001011	AC=0
STAC:**00000010**	Mem[29]=AC　total
STAC	Mem[30]=AC　i
LDAC:00000001	AC=Mem[30]
INAC:00001010	AC++
STAC	Mem[30]=AC
MVAC：00000011	R=AC（即R=i）
LDAC	AC=Mem[29]
ADD:00001000	AC=AC+R
STAC	Mem[29]=AC
LDAC	AC=Mem[31]=5
SUB:00001001	AC=AC-R（即n-i，结果为0时Z=1）
JPNZ:00000111	Z=0时即 i！=n(n=5)接着传入跳转地址00000111即第一个LDAC

mem.vhd:

```
signal memdata: memtype(4095 downto 0) := (
0 => RSCLAC,
1 => RSSTAC,
2 => std_logic_vector(to_unsigned(total_addr, 8)),
3 => X"00",
4 => RSSTAC,
5 => std_logic_vector(to_unsigned(i_addr, 8)),
6 => X"00",
7 => RSLDAC,  -- loop
8 => std_logic_vector(to_unsigned(i_addr, 8)),
9 => X"00",
10 => RSINAC,
11 => RSSTAC,
12 => std_logic_vector(to_unsigned(i_addr, 8)),
13 => X"00",
14 => RSMVAC,
15 => RSLDAC,
16 => std_logic_vector(to_unsigned(total_addr, 8)),
17 => X"00",
18 => RSADD,
19 => RSSTAC,
```

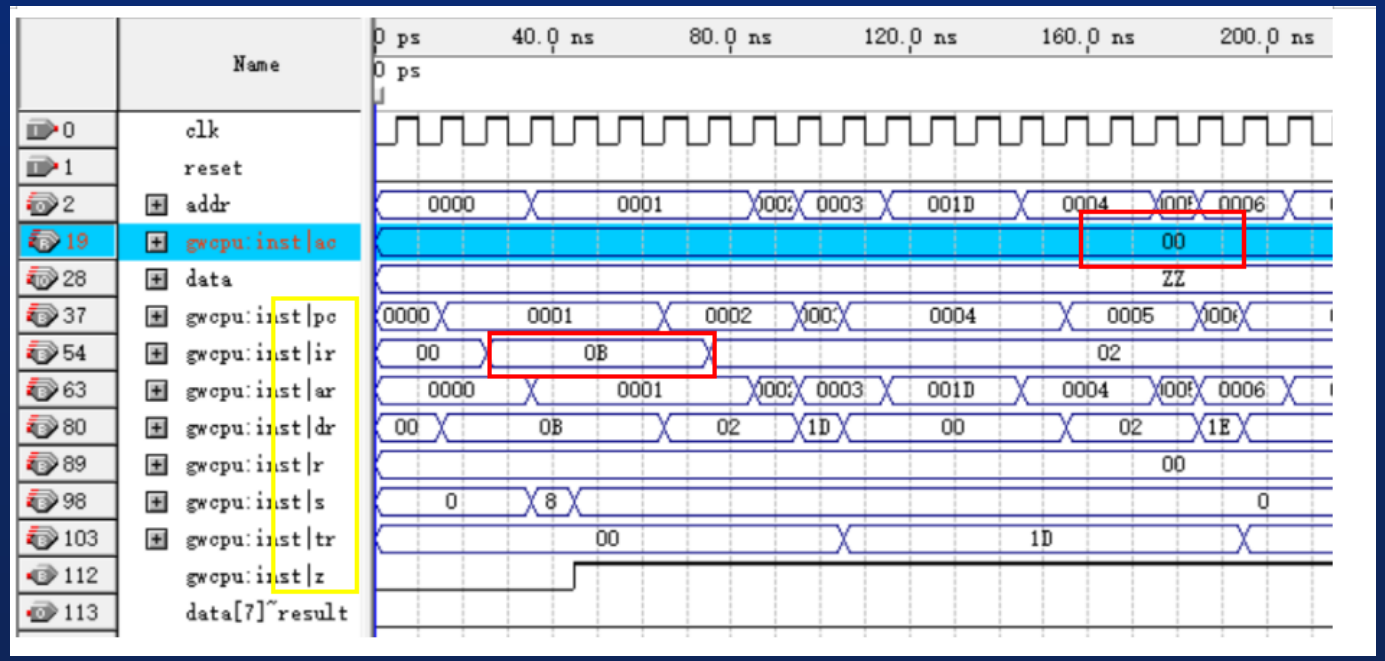
```
20 => std_logic_vector(to_unsigned(total_addr, 8)),
21 => X"00",
22 => RSLDAC,
23 => std_logic_vector(to_unsigned(n_addr, 8)),
24 => X"00",
25 => RSSUB,
26 => RSJPNZ,
27 => std_logic_vector(to_unsigned(loop_addr, 8)),
28 => X"00",
29 => X"00",  -- total
30 => X"00",  -- i
31 => "00000101",  -- n
others => RSNOP
```

累加到5，n=5

Addr	+0	+1	+2	+3	+4	+5	+6	+7
00	00001011	00000010	00011101	00000000	00000010	00011110	00000000	00000001
08	00011110	00000000	00001010	00000010	00011110	00000000	00000011	00000001
10	00011101	00000000	00001000	00000010	00011101	00000000	00000001	00011111
18	00000000	00001001	00000111	00000111	00000000	00000000	00000000	00000101
20	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
28	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
30	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
38	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000

仿真结果：

第一条指令运行后各个信号的输出：即CLAC（00001011即0B）指令，此时IR=0B，AC=0，PC在FETCH2加1变成0001，AR对应FETCH1和FETCH4时PC的值即先后为0000、0001



仿真结果：

i=5即i=n时，不满足JPNZ跳转条件，执行JPNZ的下一条指令即NOP指令：此时累加和的结果为）0F（=0+1+2+3+4+5），由此可知结果正确。

第一次执行JPNZ，即i=1时