



简单cpu的设计

智能1601

樊龙

201608010325



目录

CONTENTS

ENTER YOUR COMPANY NAME



cpu设计步骤



cpu规格



指令集



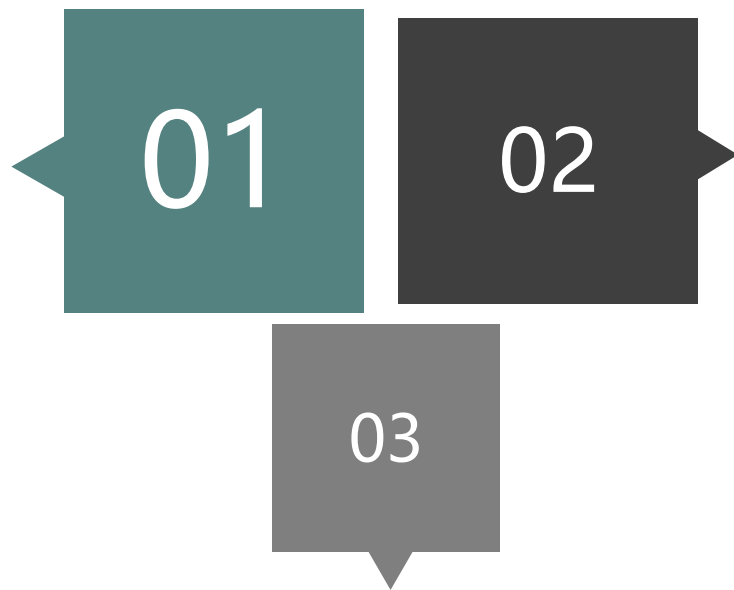
寄存器



cpu设计实现

01：确定它的用途

关键：使CPU的处理能力和它所执行的任务匹配。



02：设计指令集结构

03：设计状态图

- 列出在每个状态中执行的微操作
- 从一个状态转移到另外一个状态的条件

CPU执行如下的操作序列

取指令周期：从存储器中
取出一条指令，然后转到
译码周期。

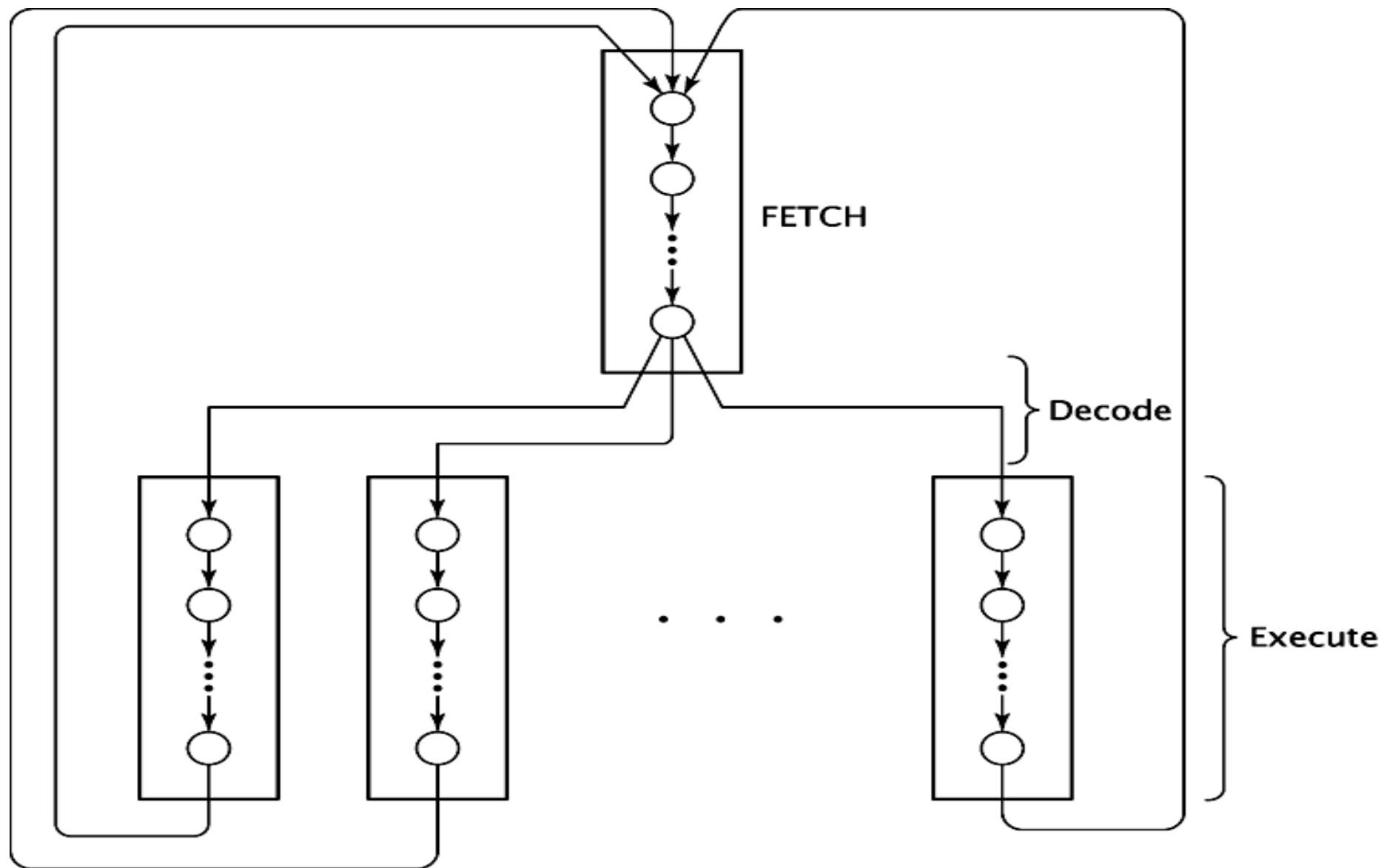
执行周期：执行该指令，
然后转移到取指令周期去
取下一条指令



译码周期：对该指
令进行译码，即确
定取到的是哪一种
指令，然后转移到这
种指令对应的执行周
期。

状态图

一般cpu状态图:



cpu的规格

1. 64K字节的存储器，每个存储单元8位宽。

地址引脚A[15..0]

数据引脚D[7..0]

2. CPU的三个内部寄存器

- ◆ 8位累加器AC：接受任何算术或者逻辑运算的结果，并为使用两个操作数的算术或者逻辑操作指令提供一个操作数。
- ◆ 寄存器R：一个8位通用寄存器。它为所有的双操作数算术和逻辑运算指令提供第二个操作数。它也可以用来暂时存放累加器马上要用到的数据。（减少存储器访问次数提高CPU的性能）
- ◆ 零标志位Z：每次执行算术运算或者逻辑运算的时候，它都将被置位。

指令集

指令	指令码	操作
NOP	0000 0000	无
LDAC	0000 0001 Γ	$AC \leftarrow M[\Gamma]$
STAC	0000 0010 Γ	$M[\Gamma] \leftarrow AC$
MVAC	0000 0011	$R \leftarrow AC$
MOV R	0000 0100	$AC \leftarrow R$
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

指令集

ADD	0000 1000	$AC \leftarrow AC + R$, IF $(AC + R = 0)$ THEN $Z \leftarrow 1$ ELSE $Z \leftarrow 0$
SUB	0000 1001	$AC \leftarrow AC - R$, IF $(AC - R = 0)$ THEN $Z \leftarrow 1$ ELSE $Z \leftarrow 0$
INAC	0000 1010	$AC \leftarrow AC + 1$, IF $(AC + 1 = 0)$ THEN $Z \leftarrow 1$ ELSE $Z \leftarrow 0$
CLAC	0000 1011	$AC \leftarrow 0$, $Z \leftarrow 1$
AND	0000 1100	$AC \leftarrow AC \wedge R$, IF $(AC \wedge R = 0)$ THEN $Z \leftarrow 1$ ELSE $Z \leftarrow 0$
OR	0000 1101	$AC \leftarrow AC \vee R$, IF $(AC \vee R = 0)$ THEN $Z \leftarrow 1$ ELSE $Z \leftarrow 0$
XOR	0000 1110	$AC \leftarrow AC \oplus R$, IF $(AC \oplus R = 0)$ THEN $Z \leftarrow 1$ ELSE $Z \leftarrow 0$
NOT	0000 1111	$AC \leftarrow AC'$, IF $(AC' = 0)$ THEN $Z \leftarrow 1$ ELSE $Z \leftarrow 0$

有那些寄存器？寄存器的功能是什么？

- ◆ 16位的地址寄存器AR：通过引脚A[15..0]向存储器提供地址。
- ◆ 16位的程序计数器PC：存放的是将要执行的下一条指令的地址，或者指令需要的下一个操作数的地址。

AR和PC必须能够执行并行的装载和递增的操作。两个寄存器都从内部总线上接受数据。

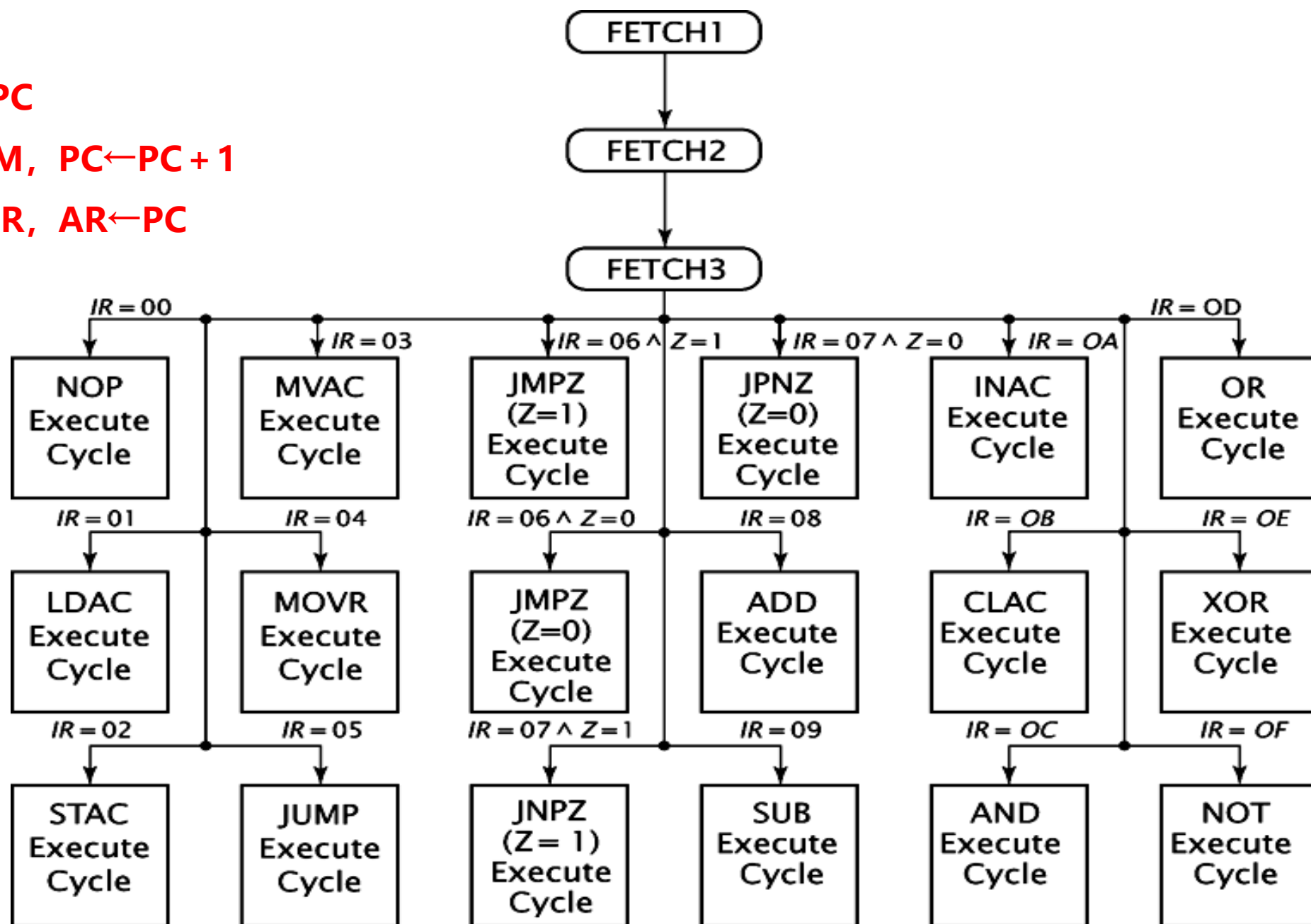
- ◆ 8位的数据寄存器DR：通过D[7..0]从存储器中接收指令和数据并且向存储器传送数据。
- ◆ 8位的指令寄存器IR：存放的是从存储器中取出来的操作码。
- ◆ 8位的临时寄存器TR：在指令执行过程中，临时存储数据。（程序员不能访问）
- ◆ DR, IR, R, TR必须能够并行装载数据。
CPU用一个ALU来完成所有这些功能。
ALU能够接受AC的数据作为一个输入，接受内部总线上的数据作为另外一个输入。AC总是从ALU得到它的输入。CPU同时也根据ALU的输出来决定结果是否0，从而设置Z。

指令执行过程

FETCH1: $AR \leftarrow PC$

FETCH2: $DR \leftarrow M, PC \leftarrow PC + 1$

FETCH3: $IR \leftarrow DR, AR \leftarrow PC$



数据通路

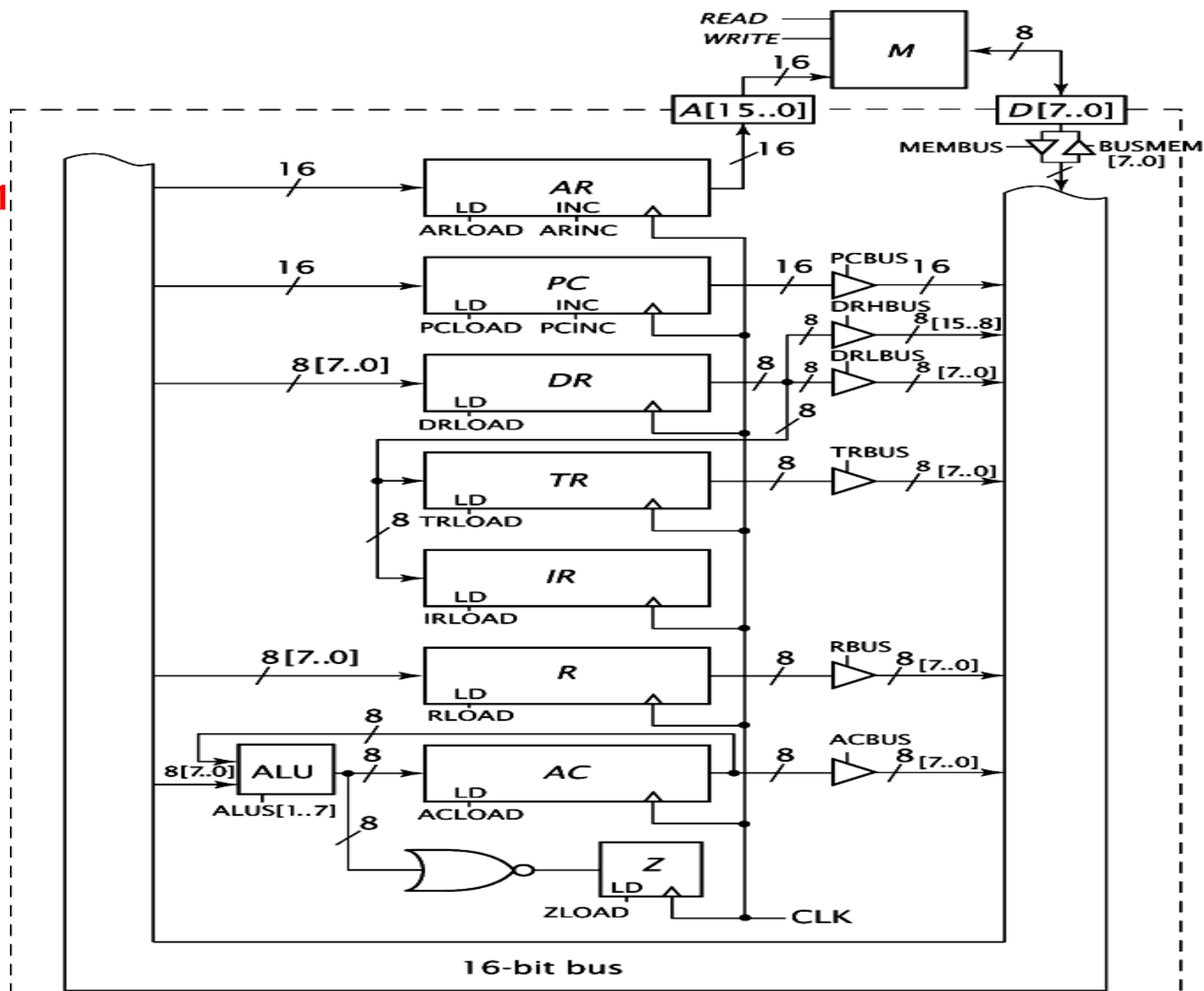
STAC1: $DR \leftarrow M$, $PC \leftarrow PC + 1$, $AR \leftarrow AR + 1$

STAC2: $TR \leftarrow DR$, $DR \leftarrow M$, $PC \leftarrow PC + 1$

STAC3: $AR \leftarrow DR$, TR

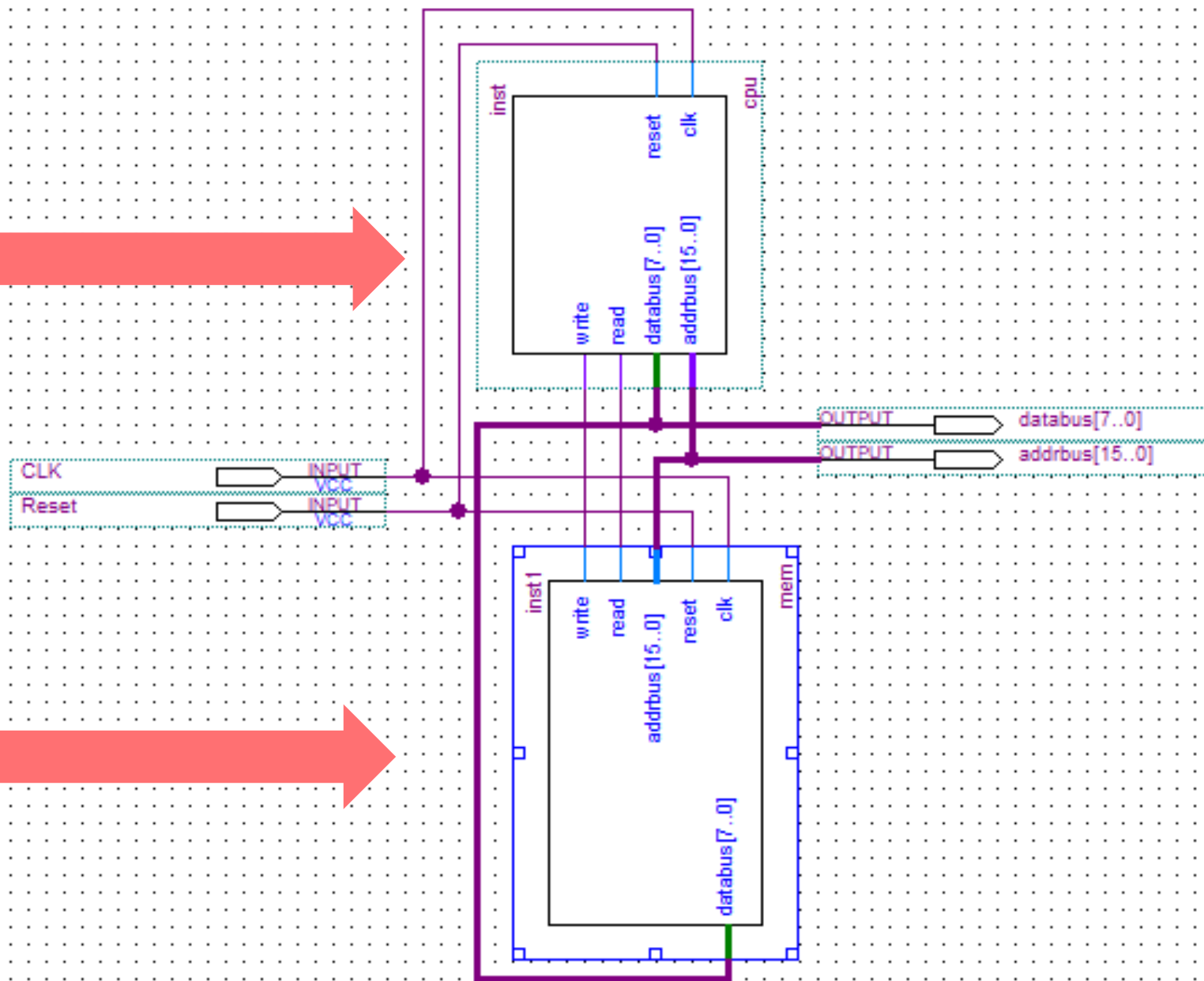
STAC4: $DR \leftarrow AC$

STAC5: $M \leftarrow DR$



CPU

内存



用相对简单CPU编程计算 $1 + 2 + \dots + n$ 。

实现这一算法的相对简单CPU的代码如下：

算法步骤：

1: $total = 0, i = 0$

2: $i = i + 1$

3: $total = total + i$

4: IF $i \neq n$ THEN GOTO 2

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

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

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

LDAC	n	}	$IF i \neq n THEN GOTO Loop$
SUB			
JPNZ	Loop		

total:
i:

内存实现

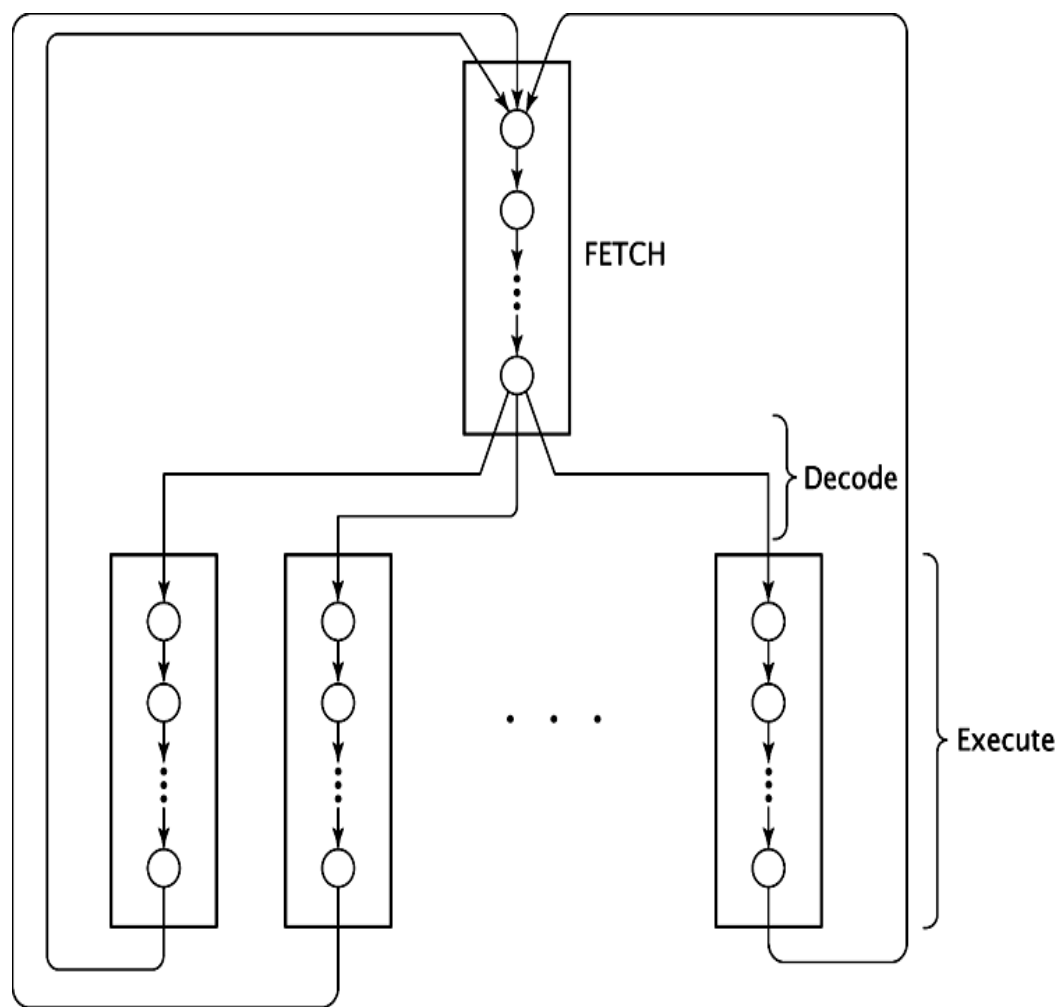
指令	1st Loop	2nd Loop	3rd Loop	4th Loop	5th Loop
CLAC	AC=0				
STAC total	total=0				
STAC i	i=0				
LDAC i	AC=0	AC=1	AC=2	AC=3	AC=4
INAC	AC=1	AC=2	AC=3	AC=4	AC=5
STAC i	i=1	i=2	i=3	i=4	i=5
MVAC	R=1	R=2	R=3	R=4	R=5
LDAC total	AC=0	AC=1	AC=3	AC=6	AC=10
ADD	AC=1	AC=3	AC=6	AC=10	AC=15
STAC total	total=1	total=3	total=6	total=10	total=15
LDAC n	AC=5	AC=5	AC=5	AC=5	AC=5
SUB	AC=4, Z=0	AC=3, Z=0	AC=2, Z=0	AC=1, Z=0	AC=0, Z=0
JPNZ Loop	JUMP	JUMP	JUMP	JUMP	NO JUMP

内存实现

```
signal memdata: memtype(4095 downto 0) := (  
0 => RSCLAC,  
1 => RSSTAC, --m[total_total]<=ac    total=0  
2 => std_logic_vector(to_unsigned(total_addr, 8)),  
3 => X"00",  
4 => RSSTAC, --m[i_addr]<=ac    i=0  
5 => std_logic_vector(to_unsigned(i_addr, 8)),  
6 => X"00",  
7 => RSLDAC,  -- loop    --ac<=m[i_addr]    ac=i  
8 => std_logic_vector(to_unsigned(i_addr, 8)),  
9 => X"00",  
10 => RSINAC, --ac++  
11 => RSSTAC, --i=ac  
12 => std_logic_vector(to_unsigned(i_addr, 8)),  
13 => X"00",  
14 => RSMVAC, --r=ac
```

```
15 => RSLDAC, --ac=total  
16 => std_logic_vector(to_unsigned(total_addr, 8)),  
17 => X"00",  
18 => RSADD,  --ac=ac+r  
19 => RSSTAC, --total=ac  
20 => std_logic_vector(to_unsigned(total_addr, 8)),  
21 => X"00",  
22 => RSLDAC, --ac=n  
23 => std_logic_vector(to_unsigned(n_addr, 8)),  
24 => X"00",  
25 => RSSUB,  --ac=ac-r  
26 => RSJPNZ, --  
27 => std_logic_vector(to_unsigned(loop_addr, 8)),  
28 => X"00",  
29 => X"00",  -- total  
30 => X"00",  -- i  
31 => "00000100", -- n  
others => RSNOP
```


CPU实现



```
if (state=fetch1) then nextstate<=fetch2;
elseif(state=fetch2) then nextstate<=fetch3;
elseif(state=fetch3) then nextstate<=fetch4;
elseif(state=fetch4) then
    if(ir=RSCLAC) then nextstate<=clac1;
    elseif(ir=RSINAC) then nextstate<=incac1;
    elseif(ir=RSADD) then nextstate<=add1;
    elseif(ir=RSSUB) then nextstate<=sub1;
    elseif(ir=RSAND) then nextstate<=and1;
    elseif(ir=RSOR) then nextstate<=or1;
    elseif(ir=RSXOR) then nextstate<=xor1;
    elseif(ir=RSNOT) then nextstate<=not1;
    elseif(ir=RSMVAC) then nextstate<=mvac1;
    elseif(ir=RSMOVR) then nextstate<=movr1;
    elseif(ir=RSLDAC) then nextstate<=ldac1;
    elseif(ir=RSSTAC) then nextstate<=stac1;
    elseif(ir=RSJUMP) then nextstate<=jump1;
```

STAC指令

STAC1: $DR \leftarrow M$, $PC \leftarrow PC + 1$, $AR \leftarrow AR + 1$

STAC2: $TR \leftarrow DR$, $DR \leftarrow M$, $PC \leftarrow PC + 1$

STAC3: $AR \leftarrow DR$, TR

STAC4: $DR \leftarrow AC$

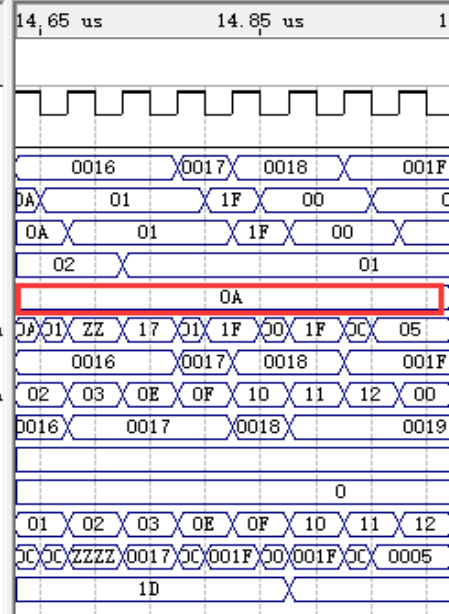
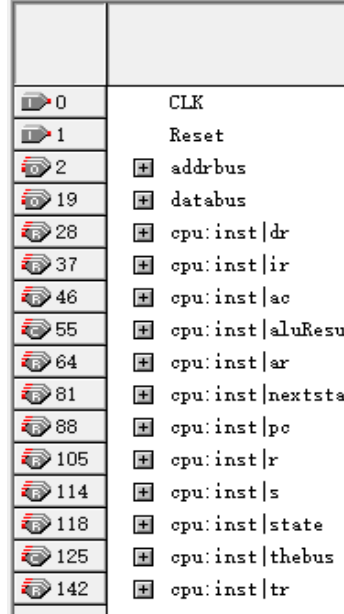
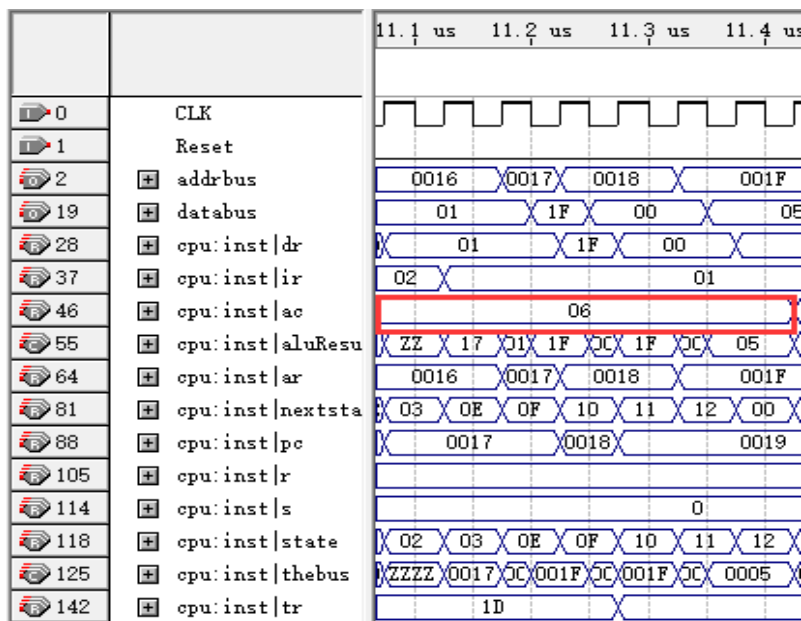
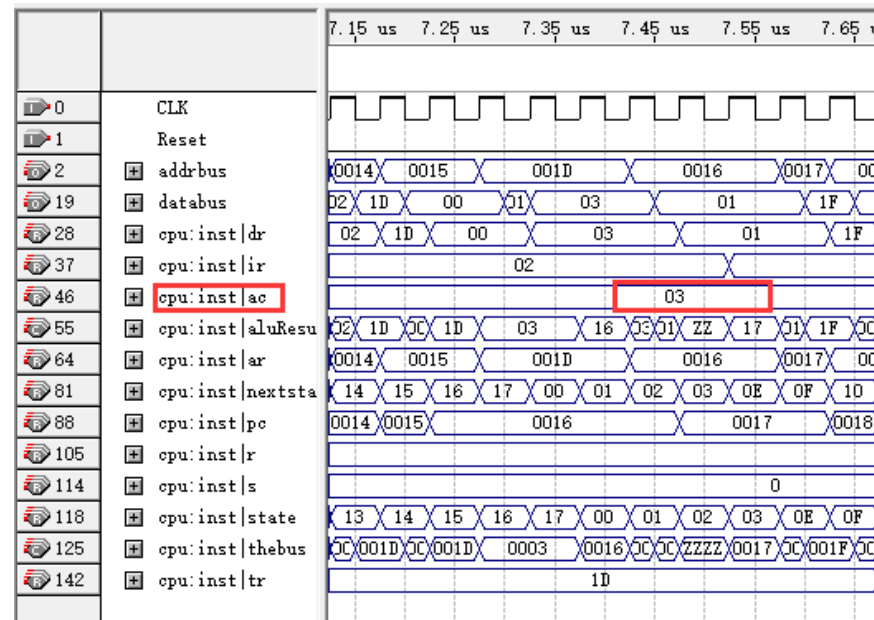
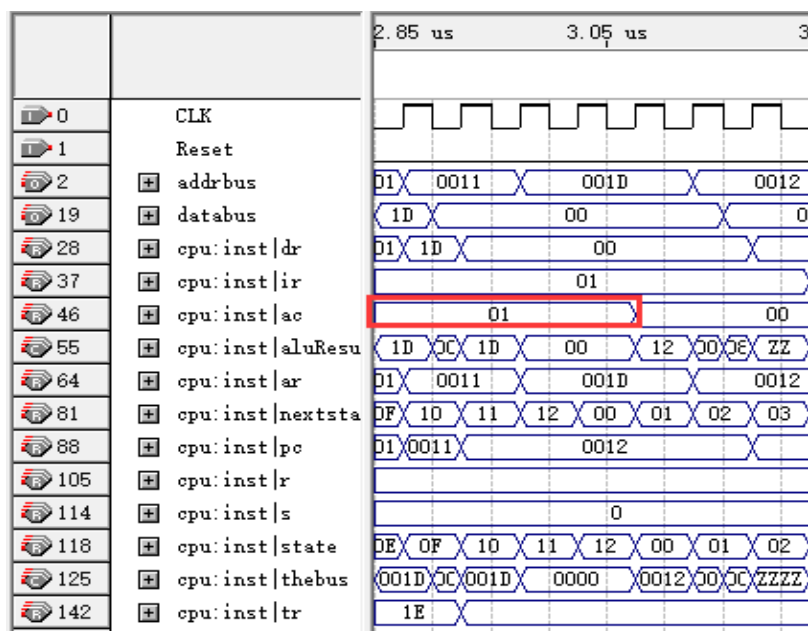
STAC5: $M \leftarrow DR$

```
elseif(state=stac1) then nextstate<=stac2;
elseif(state=stac2) then nextstate<=stac3;
elseif(state=stac3) then nextstate<=stac4;
elseif(state=stac4) then nextstate<=stac5;
elseif(state=stac5) then nextstate<=fetch1;
```

--store ac

```
elseif(state=stac1) then--dr<=m pc=pc+1 ar=ar+1
arload<='0';pcbus<='0';pcinc<='1';drload<='1';membus<='1';irload<='0';acreset<='0';acinc<='0';
rbus<='0';s<="0000";acload<='0';rload<='0';acbus<='0';arinc<='1';trbus<='0';drbus<='0';trload<='0';
read<='1';writel<='0';write<='0';pcload<='0';
elseif(state=stac2) then--tr<=dr dr<=m pc=pc+1
arload<='0';pcbus<='0';pcinc<='1';drload<='1';membus<='1';irload<='0';acreset<='0';acinc<='0';
rbus<='0';s<="0000";acload<='0';rload<='0';acbus<='0';arinc<='0';trbus<='0';drbus<='0';trload<='1';
read<='1';writel<='0';write<='0';pcload<='0';
elseif(state=stac3) then--ar<=dr, tr
arload<='1';pcbus<='0';pcinc<='0';drload<='0';membus<='0';irload<='0';acreset<='0';acinc<='0';
rbus<='0';s<="0000";acload<='0';rload<='0';acbus<='0';arinc<='0';trbus<='1';drbus<='1';trload<='0';
read<='0';writel<='0';write<='0';pcload<='0';
elseif(state=stac4) then--dr<=ac
arload<='0';pcbus<='0';pcinc<='0';drload<='1';membus<='0';irload<='0';acreset<='0';acinc<='0';
rbus<='0';s<="0000";acload<='0';rload<='0';acbus<='1';arinc<='0';trbus<='0';drbus<='0';trload<='0';
read<='0';writel<='0';write<='0';pcload<='0';
elseif(state=stac5) then--m<=dr
arload<='0';pcbus<='0';pcinc<='0';drload<='0';membus<='0';irload<='0';acreset<='0';acinc<='0';
rbus<='0';s<="0000";acload<='0';rload<='0';acbus<='0';arinc<='0';trbus<='0';drbus<='1';trload<='0';
read<='0';writel<='1';write<='1';pcload<='0';
```

仿真结果





THANKS !

智能1601
樊龙

201608010325