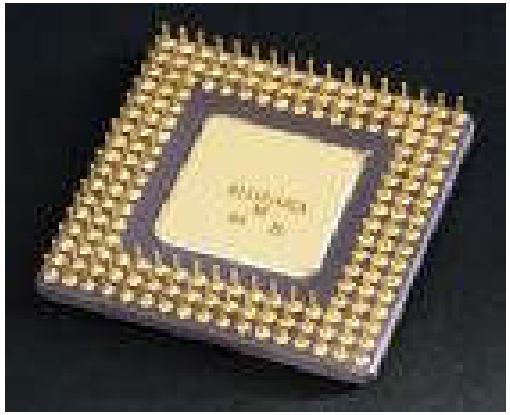
第六讲 8位CPU分析与设计

- 6.1 CPU组成结构
- 6.2 Richard CPU分析
- 6.3 CPU设计方法
- 6.4 指令系统设计
- 6.5 CPU设计思路
- 6.6 CPU功能模块设计

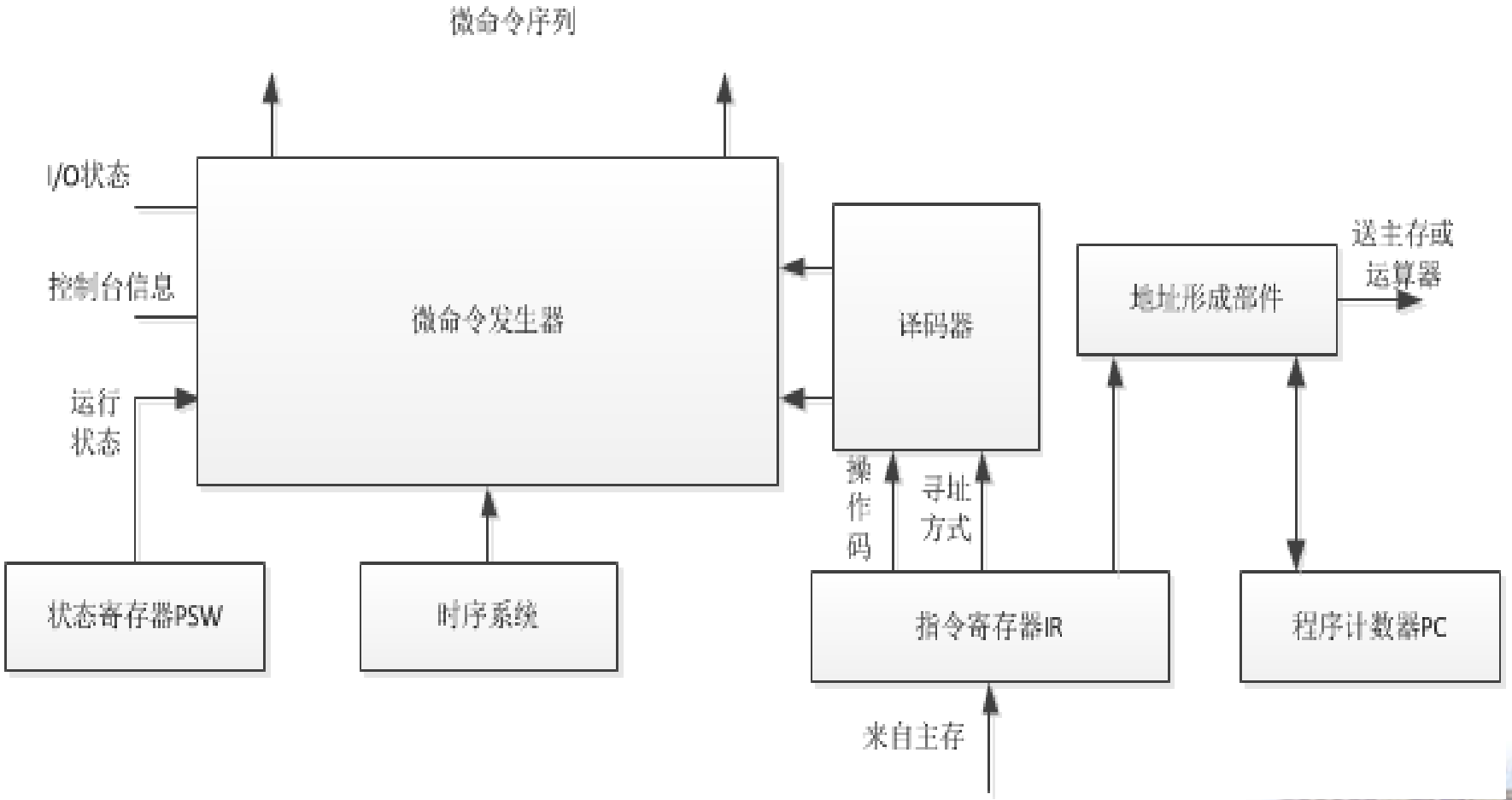


6.1 CPU组成结构

- 按照冯•诺依曼计算机的划分方式，运算部件和与运算有关的寄存器属于运算器，而与程序执行有关的寄存器、微命令产生部件及时序系统等则属于控制器部分。
- CPU主要由控制部件和运算部件两部分构成。
- CPU的主要功能单元通过内部总线建立CPU内部的信号传送通路，实现信息交换。

6.1.1 控制部件

- 控制器主要由以下几部分组成：



6.1.1 控制部件

- **控制器分为硬布线控制器和微程序控制器两种基本类型。**
- **硬布线控制器**
  - 将控制部件看做产生专门固定时序控制信号的逻辑电路，以使用最少的元件和取得最高操作速度作为设计目标。
  - 缺点：设计不规整，并且不易修改或扩展。
- **微程序控制器**
  - 将机器指令的操作（从取指令到执行）分解为若干更基本的微操作序列，并将有关的控制信息（微命令）以微码形式编成微指令输入控制存储器中。
  - 优点：设计规则，方便修改及功能扩展。

6.1.2 运算部件

- 运算部件是计算机中对数据进行加工处理的主要场所，其最重要的功能是**执行算术和逻辑运算**。

- 运算器的性能直接决定计算机的处理能力，而运算器的设计与数据在计算机内的表示、存储方式、完成运算所用的算法及实现算法所用的逻辑电路都有密切关联。

6.1.2 运算部件

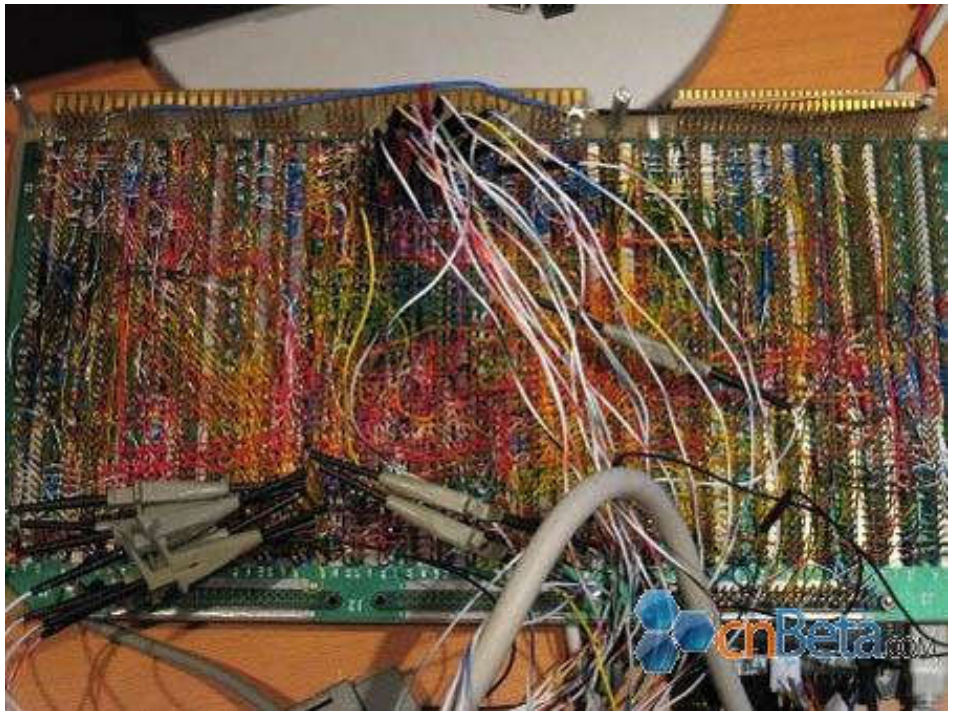
- 运算部件主要由**输入逻辑**、**算术/逻辑运算单元**及**输出逻辑等**三部分组成。
- **输入逻辑**
  - 对输入到算术/逻辑运算单元的操作数进行选择
- **算术/逻辑运算单元**
  - 运算部件的核心，完成具体的算术、逻辑运算操作。
- **输出逻辑**
  - 将算术/逻辑运算单元的运算结果经直传、左移、右移或者字节交换后送入相应的寄存器。

6.1.3 寄存器组

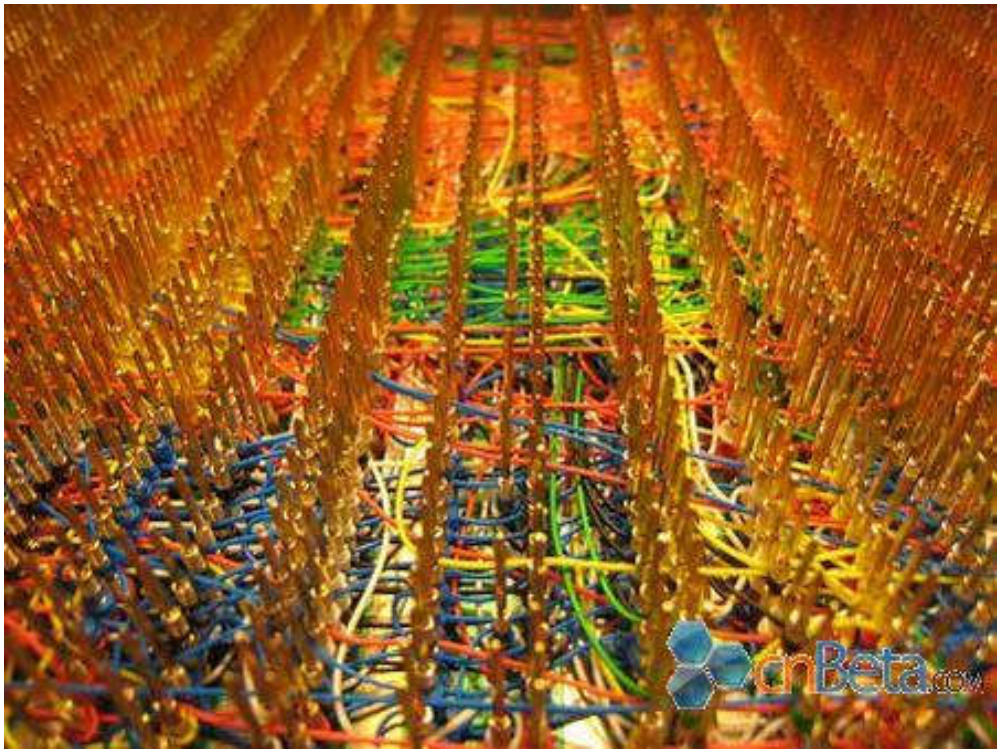
在复杂指令集计算机系统中一般有5种类型的寄存器：

- 指令寄存器
- 程序计数器
- 存储器数据缓冲寄存器（Memory Buffer Register,MBR）
- 存储器地址寄存器（Memory Address Register,MAR）
- 程序状态字寄存器（Program Status Word,PSW）

CPU设计实例



CPU设计实例

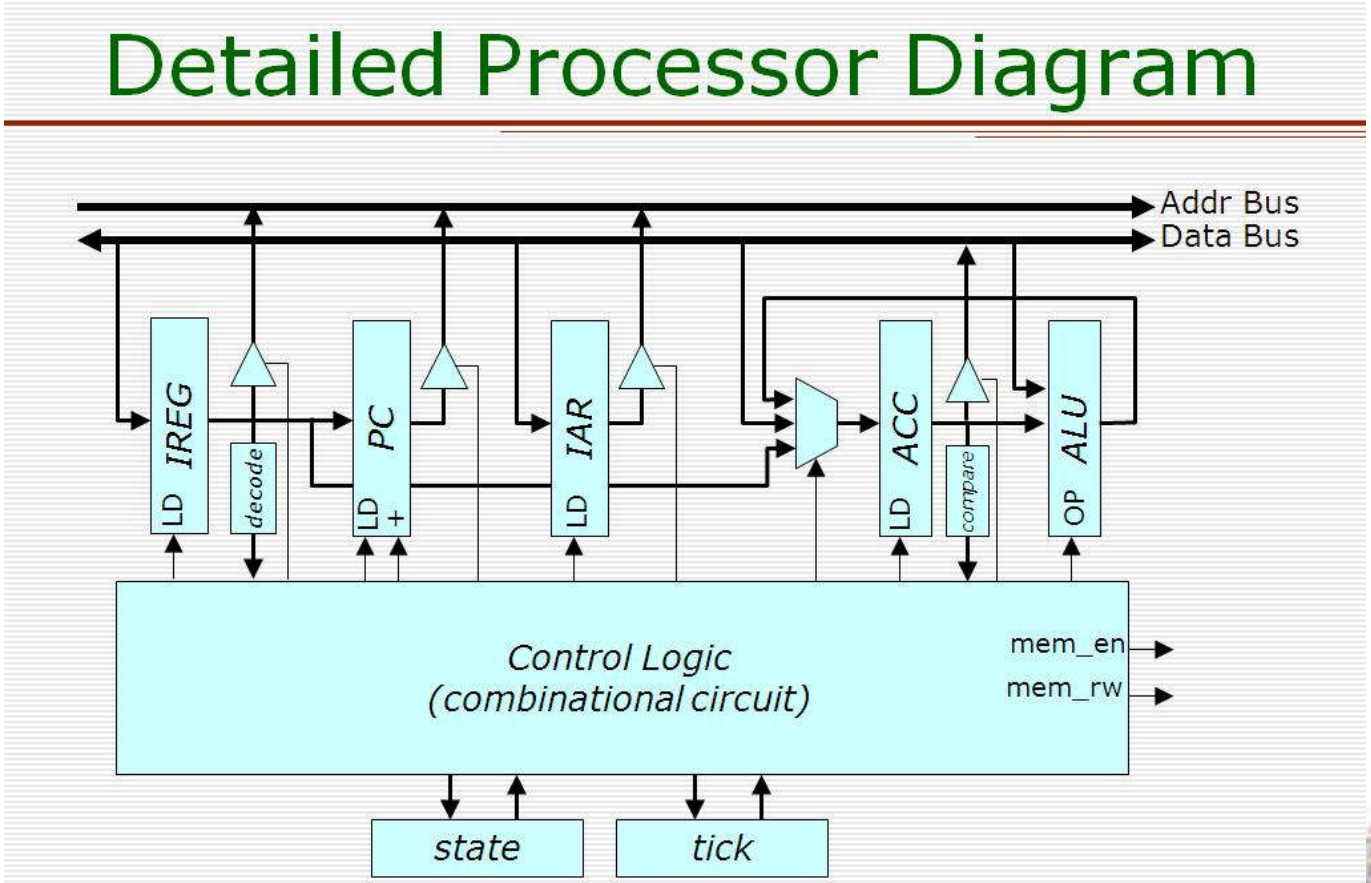


CPU设计实例

- BMOW（Big Mess of Wires）

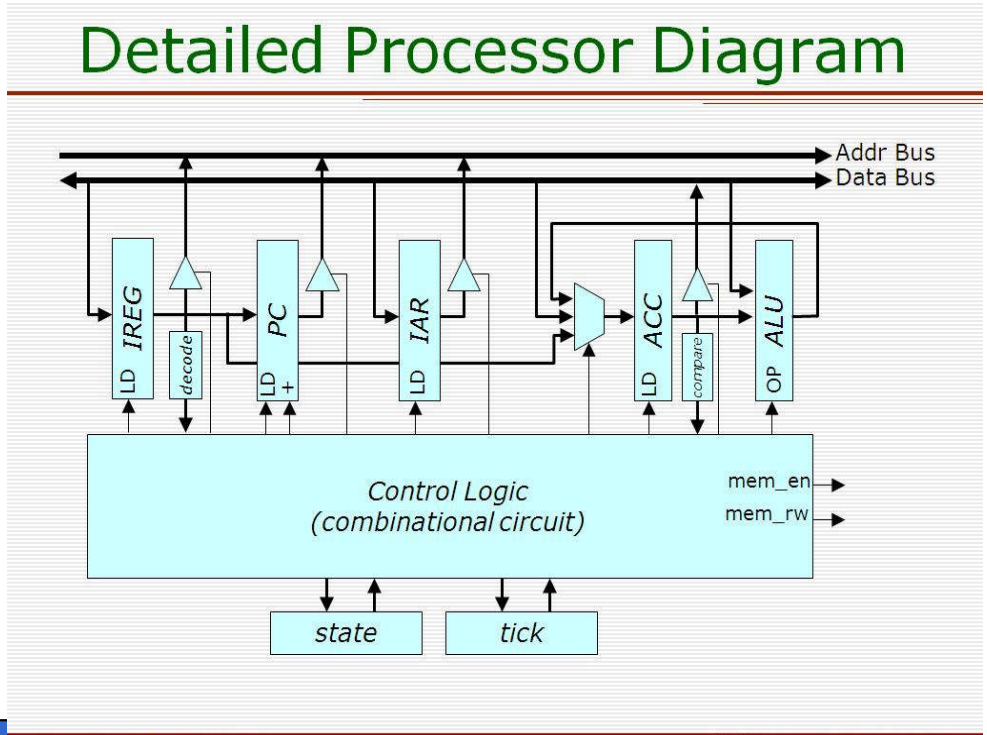
6.2 Richard CPU分析

- 美国华盛顿大学William D. Richard 采用VHDL语言设计的16位CPU，**仅有200行代码。**



6.2 Richard CPU分析

- IREG:指令寄存器(Instruction Register)
- PC:程序计数器(Program Counter)
- IAR:间接地址寄存器(Indirect Address Register)
- ALU:算术逻辑单元 (Arithmetic Logic Unit)
- ACC:累加器(Accumulator)



6.2.1 指令集设计分析

代码	指令	說明	运算方法
0000	halt	暂停 (halt execution)	
0001	negate	反相 (negation)	ACC := -ACC
1xxx	load	立即载入 (immediate load)	if sign bit of xxx is 0 then ACC := 0xxx else ACC := fxxx
2xxx	dload	直接载入(direct load)	ACC := M[0xxx]
3xxx	iload	间接载入(indirect load)	ACC := M[M[0xxx]]
4xxx	dstore	直接存储 (direct store)	M[0xxx] := ACC
5xxx	istore	间接存储 (indirect store)	M[M[0xxx]] := ACC
6xxx	br	分枝 (branch)	PC := 0xxx
7xxx	brZero	零分枝 (branch if zero)	if ACC = 0 then PC := 0xxx
8xxx	brPos	正分枝 (branch if positive)	if ACC > 0 then PC := 0xxx
9xxx	brNeg	负分枝 (branch if negative)	if ACC < 0 then PC := 0xxx
axxx	add	加法	ACC := ACC + M[0xxx]

6.2.2 指令编码分析

- halt与negate没有运算参数，编码为0000H和0001H。
- 指令编码

  IR(15..12) : 运算码

  IR(11..0) ：运算元
- 四种指令类型
  - 载入指令(load, dload, iload)
  - 存储指令(dstore, istore)
  - 分支指令(br, brZero, brZero, brPos, brNeg)
  - 运算指令，加法运算(add)

6.2.3 设计思路

- 采用Mealy状态机设计方式
  输出由当前状态state与输入信号t0-t7决定。
- CPU基本状态

```
type state_type is (
        reset_state, fetch, halt, negate, mload, dload, iload,
        dstore, istore, branch, brZero, brPos, brNeg, add
);

signal state: state_type;

type tick_type is (t0, t1, t2, t3, t4, t5, t6, t7);
signal tick: tick_type;
```

6.2.4 指令周期分析

- istore -- M[M[0xxx]]:= ACC

①IR(11..0)（[0xxx]） →地址总线
②数据总线(M[0xxx]) →IAR
③清除地址线
④IAR→地址总线
⑤ACC→数据总线
⑥使能存储器写操作
⑦等待完成写操作
⑧清除地址总线、数据总线三态

6.2.5 时钟节拍产生

- 指令执行最多包含8个CPU周期，每一个CPU周期为一个节拍：t0 t1 … t7。

```
function nextTick(tick: tick_type) return tick_type is
begin
      -- return next logical value for tick
   case tick is
      when t0 => return t1; when t1 => return t2; when t2 =>
      return t3;
      when t3 => return t4; when t4 => return t5; when t5 =>
      return t6;
      when t6 => return t7; when others => return t0;
   end case;
end function nextTick;
```

6.2.6 指令译码器设计

```
procedure decode is begin
        -- Instruction decoding.
        case iReg(15 downto 12) is
        when x"0" =>
                if iReg(11 downto 0) = x"000" then
                        state <= halt;
                elsif iReg(11 downto 0) = x"001" then
                        state <= negate;
                end if;
        when x"1" =>            state <= mload;
        when x"2" =>            state <= dload;
        when x"3" =>            state <= iload;
        when x"4" =>            state <= dstore;
        when x"5" =>            state <= istore;
        when x"6" =>            state <= branch;
        when x"7" =>            state <= brZero;
        when x"8" =>            state <= brPos;
        when x"9" =>            state <= brNeg;
        when x"a" =>            state <= add;
        when others => state <= halt;
        end case;
end procedure decode;
```

6.2.7 微控制器设计

- 微控制器将每条指令分解为若干条微操作。
- 为了提高执行效率，在时钟上升和下降边沿都有执行动作。

```
tick <=
case s
when r
```

```
alu <=    (not acc) + x"0001" when state = negate else
                acc + dbus when state = add else
                (alu'range => '0');
 pcX <= pc;
iregX <= ireg;
iarX <= iar;
accX <= acc;
aluX <= alu;
```

（1）Fetch指令分析

```
fetch
  -- rising edge
  if tick = t1 then
   iReg <= dBus;
 end if;     -- 2. get Intstruction from data bus to iReg
  if tick = t2 then    -- 3. decode and PC++, go to next state
    decode;
     pc <= pc + '1';
     tick <= t0;
  end if;
  -- falling edge
  if tick = t0 then
    m_en <= '1';
    aBus <= pc;
 end if;         -- 1. put PC on to address bus, enable read
instruction
 if tick = t2 then
   m_en <= '0';
  aBus <= (aBus'range => '0');
end if;              -- 4. clear address bus
```

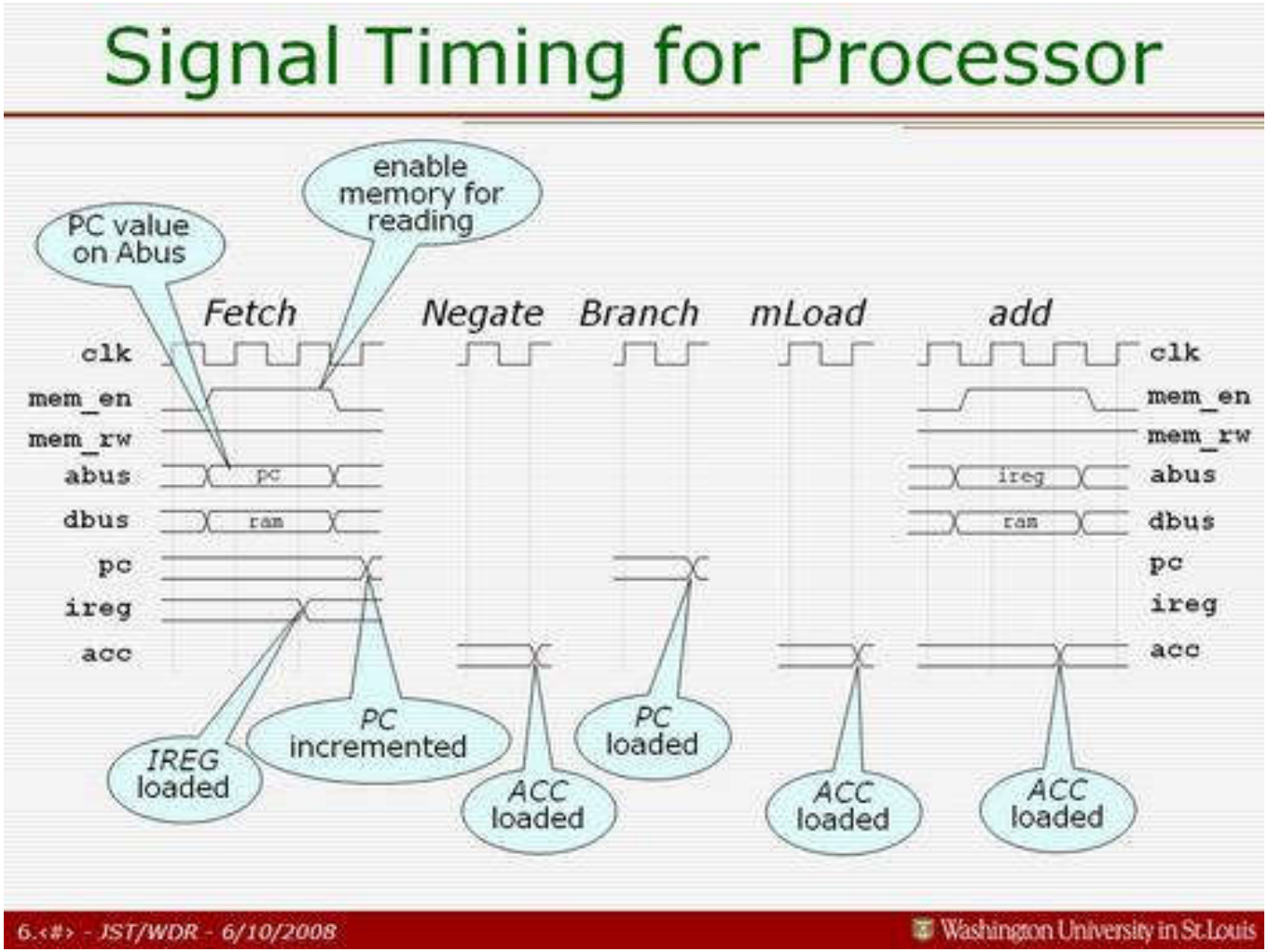
（2）Dload指令分析

```
dload m -- ACC := M[0xxx]
  -- rising edge
  if tick = t1 then
 acc <= dBus;
 end if;
  if tick = t2 then
   wrapup;
  end if;

  -- falling edge
  if tick = t0 then     -- 1. Put IR(11..0) onto address bus.
     m_en <= '1';
    aBus <= x"0" & iReg(11 downto 0);
  end if;
  if tick = t2 then     -- 3. clear address bus
     m_en <= '0';
    aBus <= (aBus'range => '0');
  end if;
```

```
procedure wrapup is begin
        -- Do this at end of every instruction
        state <= fetch; tick <= t0;
end procedure wrapup;
```

（3）仿真波形


Signal Timing for Processor
enable memory for reading
PC value on Abus
Fetch
Negate
Branch
mLoad
add
clk
mem_en
mem_rw
abus
dbus
pc
ireg
acc
pc
ram
ireg
ram
IREG loaded
PC incremented
ACC loaded
PC loaded
ACC loaded
ACC loaded
6.<#> - JST/WDR - 6/10/2008
Washington University in St.Louis

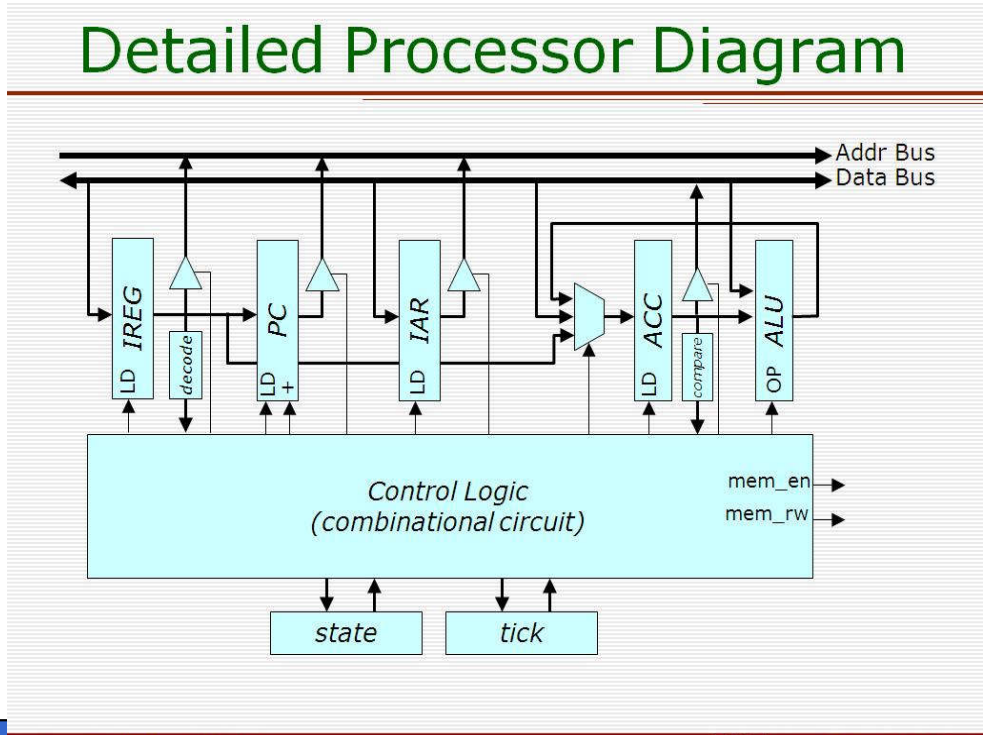
6.2.8 RAM设计

- 为了简化设计，在RAM中存放测试程序代码。

```
  -- basic instruction check
ram(0)  <= x"1a0f"; -- immediate load
ram(1)  <= x"2010"; -- direct load
ram(2)  <= x"3030"; -- indirect load
ram(3)  <= x"4034"; -- direct store
ram(4)  <= x"0001"; -- negate
ram(5)  <= x"2034"; -- direct load
ram(6)  <= x"0001"; -- negate
ram(7)  <= x"5032"; -- indirect store
ram(8)  <= x"0001"; -- negate
ram(9)  <= x"1fff"; -- immediate load
ram(10) <= x"a008"; -- add
ram(11) <= x"700d"; -- brZero
```

6.3 CPU设计方法

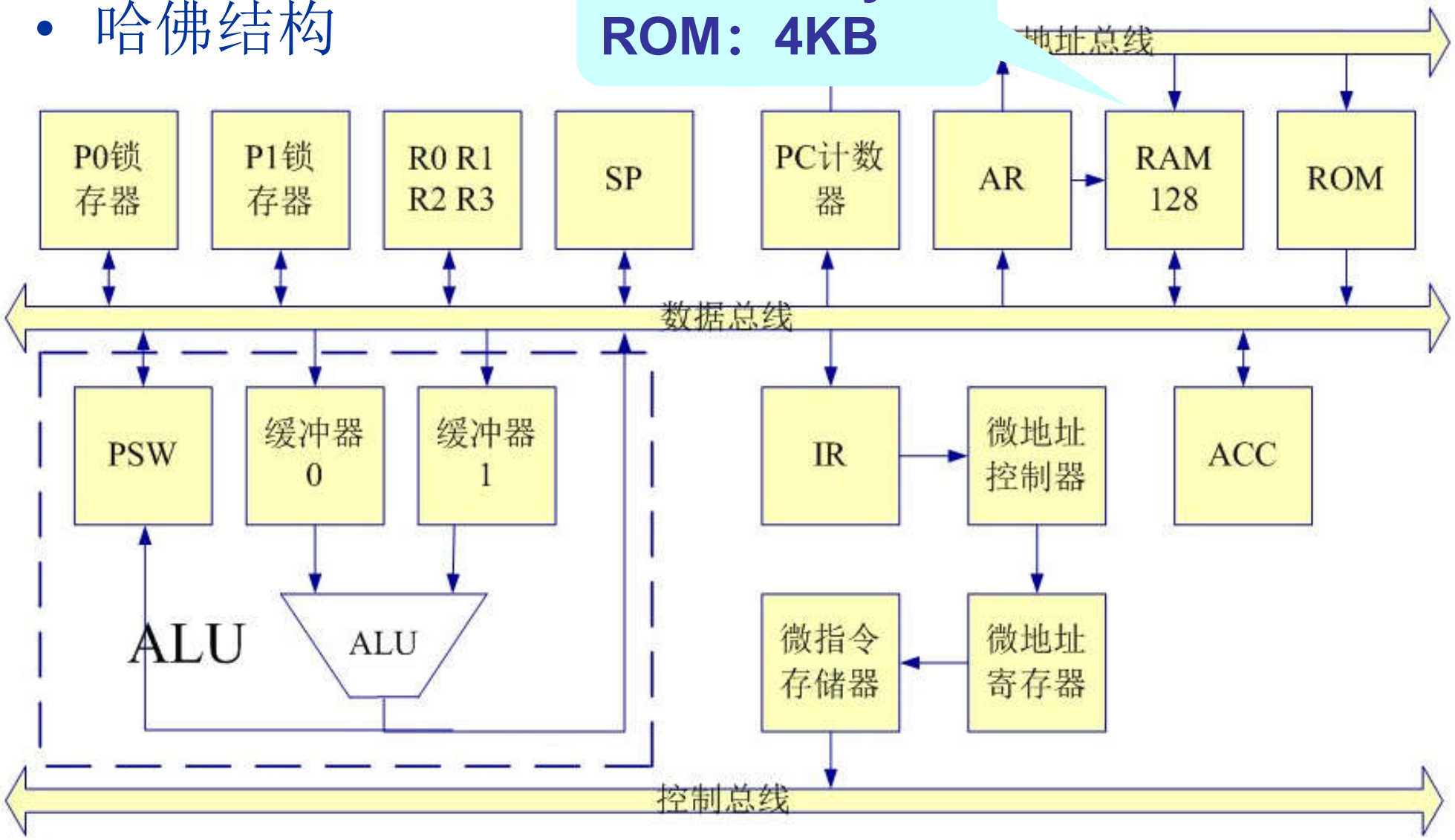
- ①CPU基本结构
- ②指令集设计（功能设计）
- ③指令编码设计
- ④整体设计思路
- ⑤确定指令周期
- ⑥CPU功能模块设计
- ⑦CPU模块联合调试测试



8位CPU结构

- 哈佛结构

RAM:128Byte
ROM：4KB



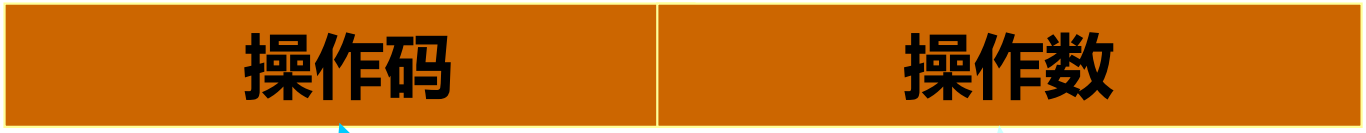
6.4 指令集设计

- **四种类型指令**
  - 传送类型指令
  - 逻辑运算指令
  - 算术运算指令
  - 调用及转移指令

6.4 指令集设计

- **四种类型指令，27条指令**
  - 传送类型指令（7条指令）
  - 逻辑运算指令（5条指令）
  - 算术运算指令（7条指令）
  - 调用及跳转指令（8条指令）
- **指令寻址方式**
  - 立即寻址
  - 直接寻址
  - 寄存器直接寻址

6.4.1 指令组成



1）指令的操作种类
2）所用操作数数据类型

1）操作数地址
2）地址附加信息
3）寻址方式

6.4.2 指令编码

- 操作码优化编码的方法有三种：**定长编码、哈夫曼编码和扩展编码。**

➢ **定长编码：**是指所有指令的操作码长度都是相等的。如果有n个需要编码的操作码，定长操作码的位数最少需要$\log_2^n$位。

➢ **哈夫曼编码：**哈夫曼方法构造哈夫曼树进行编码。

➢ **扩展编码：**继承了哈夫曼思想，限制了操作码长度为有限个数。

6.4.3 指令集设计

传送类 001

```
1 MOV     Ri , #data          ; Ri ← data
2 MOV     Ri , Rj             ; Ri ← (Rj)
3 MOV     Ri , direct         ; Ri ← direct
4 MOV     direct , Ri         ; direct ← (Ri)
5 MOV     P0 , Ri             ; P0 ← (Ri)
6 MOV     Ri , PI             ; Ri ← (PI)
7 MOV     SP , #data          ; SP ← data
```

端口操作

6.4.3 指令集设计

逻辑运算类 010

```
1 ANL     Ri , Rj          ; Ri ← (Ri) and (Rj)
2 ORL     Ri , Rj          ; Ri ← (Ri) or (Rj)
3 XRL     Ri , Rj          ; Ri ← (Ri) xor (Rj)
4 CLR     Ri               ; 把 Ri 中的数据清零
5 CPL     Ri               ; 把 Ri 中的数据取反
```

6.4.3 指令集设计

算术运算类 011

```
1 ADD     Ri , Rj          ; Ri ← (Ri) + (Rj)
2 ADDC    Ri , Rj          ; Ri ← (Ri) + (Rj)
3 INC     Ri               ; Ri ← (Ri) + 1
4 DEC     Ri               ; Ri ← (Ri) − 1
5 SUBB    Ri , Rj          ; R0 ← (R0) − (R1) − (Cy)
6 RL      Ri               ; 不带进位位的循环左移
7 RR      Ri               ; 不带进位位的循环右移
```

6.4.3 指令集设计

调用及跳转类 100

1 PUSH Ri ; (SP) ← (direct) , SP ← (SP) + 1

2 POP Ri ; SP ← (SP) − 1 , direct ← ((SP))

3 JMP addr[11..0] ; 这是一条二字节指令，指令格式为：

X	X	X	X	a11	a10	a9	a8
a7	a6	a5	a4	a3	a2	a1	a0

; PC ← addr[11..0]

4 JZ addr[11..0] ; 若(A)＝0，则 PC ← addr[11..0]

; 若(A)≠0，则 PC ← (PC)＋3

5 JC addr[11..0] ; (CY)＝1 转移指令，其转移控制为：

; 若(CY)＝1，则 PC ← addr[11..0]

; 若(CY)≠1，则 PC ← (PC)＋3

6 NOP ; PC ← (PC) + 1

7 CALL addr[11..0] ; 这是一条三字节指令，指令格式为：

X	X	X	X	a11	a10	a9	a8
a7	a6	a5	a4	a3	a2	a1	a0

; 本指令的主要功能是断点保护和构造目的地址。其操作内容可表示为：

; SP←(SP)＋1，(SP)←(SP)7～PC0

; SP←(SP)＋1，(SP)←(SP)15～PC8

; PC11～PC0 ← addr11～addr0

8 RET 返回

6.4.5 指令编码设计

- 传送类指令001

指令	字节	编码	说明
MOV Ri , #data	2	001 001	001001Rix xxxx xxxx
MOV Ri , Rj	1	001 010	001010RiRj
MOV Ri , direct	2	001 011	001011Rix xxxx xxxx
MOV direct , Ri	2	001 100	001011xRi xxxx xxxx
MOV PO , Ri	1	001 101	001101xRj
MOV Ri , PI	1	001 110	001110Rix
MOV SP , #data	2	001 111	001001xx xxxx xxxx

6.4.5 指令编码设计

- 逻辑运算指令010

ANL	Ri , Rj	1	010 001	010001RiRj
ORL	Ri , Rj	1	010 010	010010RiRj
XRL	Ri , Rj	1	010 011	010011RiRj
CLR	Ri	1	010 100	010100Rix
CPL	Ri	1	010 101	010101Rix

6.4.5 指令编码设计

- 算术运算指令011

ADD Ri , Rj	1	011 001	011001R0R1
ADDC Ri , Rj	1	011 010	011010R0R1
INC Ri	1	011 011	011011Rix
DEC Ri	1	011 100	011100Rix
SUB Ri , Rj	1	011 101	011101R1R0
RL Ri	1	011 110	011110Rix
RR Ri	1	011 111	011111Rix

6.4.5 指令编码设计

- 跳转及调用指令100

PUSH Ri	1	100 000	100000Rix
POP Ri	1	100 001	100001Rix
JMP addr12	3	100 010	100010xx xxxxAAAA AAAA AAAA
JZ addr12	3	100 011	100011xx xxxxAAAA AAAA AAAA
JC addr12	3	100 100	100100xx xxxxAAAA AAAA AAAA
NOP	1	100 101	100101xx
CALL addr12	3	100 110	100110xx xxxxAAAA AAAA AAAA
RET 返回	1	100 111	100111xx

6.5 CPU设计思路

- 模块化设计方法
  (分析模块功能、模块间信号传递)
- 时钟控制信号
- 微程序控制方式

6.5.1 指令周期确定

CALL addr[11..0] ；这是一条三字节指令，指令格式为：

X	X	X	X	a11	a10	a9	a8
a7	a6	a5	a4	a3	a2	a1	a0

本指令的主要功能是断点保护和构造目的地址。其操作内容可表示为：

；SP←(SP)＋1，(SP)←(SP)7～PC0

；SP←(SP)＋1，(SP)←(SP)15～PC8

；PC11～PC0 ← addr11～addr0

PUSH Ri	1	100 000	100000Rix
POP Ri	1	100 001	100001Rix
JMP addr12	3	100 010	100010xx xxxxAAAA AAAA AAAA
JZ addr12	3	100 011	100011xx xxxxAAAA AAAA AAAA
JC addr12	3	100 100	100100xx xxxxAAAA AAAA AAAA
NOP	1	100 101	100101xx
CALL addr12	3	100 110	100110xx xxxxAAAA AAAA AAAA
RET 返回	1	100 111	100111xx

调用指令实现过程

- CALL Addr[11..0];
  指令编码为100 11000，即98H
- **取指操作**

微操作	控制信号	功能说明
PC-->ADDR[11..0] 1	M_ROM; /ROM_EN	ROM片选信号有效，ROM读使能，PC指向程序入口地址
BUS-->IR; PC =PC + 1 ; 2	LDIR1; M_PC	IR使能，指令通过总线传送到IR，PC+1。
IR -->Microcontrol; 2 addr[7:0]->CM[47:0]	M_uROM	微控制器使能，IR送入指令，生成下一条微程序地址。

调用指令实现过程

- CALL Addr12;
  指令编码为100 11000，即98H
- **取操作数操作**

微操作	控制信号	功能说明
PC-->ADDR[11..0] 3	M_ROM; /ROM_EN	ROM片选信号有效，ROM读使能，PC指向程序入口地址
BUS-->IR; PC =PC + 1 ; 4	LDIR2; M_PC	IR使能，指令通过总线传送到IR，PC+1。
IR -->Microcontrol; addr[7:0]->CM[47:0]	M_uROM 4	微控制器使能，IR送入指令，生成下一条微程序地址。

调用指令实现过程

- CALL Addr12;
  指令编码为100 11000，即98H
- **取操作数操作**

微操作	控制信号	功能说明
PC-->ADDR[11..0] 5	M_ROM; /ROM_EN	ROM片选信号有效，ROM读使能，PC指向程序入口地址
BUS-->IR; PC =PC + 1 ; 6	LDIR3; M_PC	IR使能，指令通过总线传送到IR，PC+1。
IR -->Microcontrol; addr[7:0]->CM[47:0]	M_uROM 6	微控制器使能，IR送入指令，生成下一条微程序地址。

调用指令实现过程

- 指令执行

微操作	控制信号	功能说明
SP-->AR 7	/SP_EN /RAM_EN	SP使能，将SP指针地址送到地址寄存器，使能RAM
PC[11..8] -->BUS 7 SP+1-->SP	/PCH;M_SP_UP	PC高8位送到SP SP指针加1
SP-->AR 8	/SP_EN /RAM_EN	SP使能，将SP指针地址送到地址寄存器，使能RAM
PC[7..0] -->BUS 8 SP+1-->SP PC-->addr12	/PCL; M_SP_UP ; /LD_PC	PC高8位送到SP SP指针加1 PC指向新的地址

6.5.2 指令周期确定

CALL addr[11..0]　　；这是一条三字节指令，指令格式为：

×	×	×	×	a11	a10	a9	a8

a7	a6	a5	a4	a3	a2	a1	a0

本指令的主要功能是断点保护和构造目的地址。其操作内容可表示为：

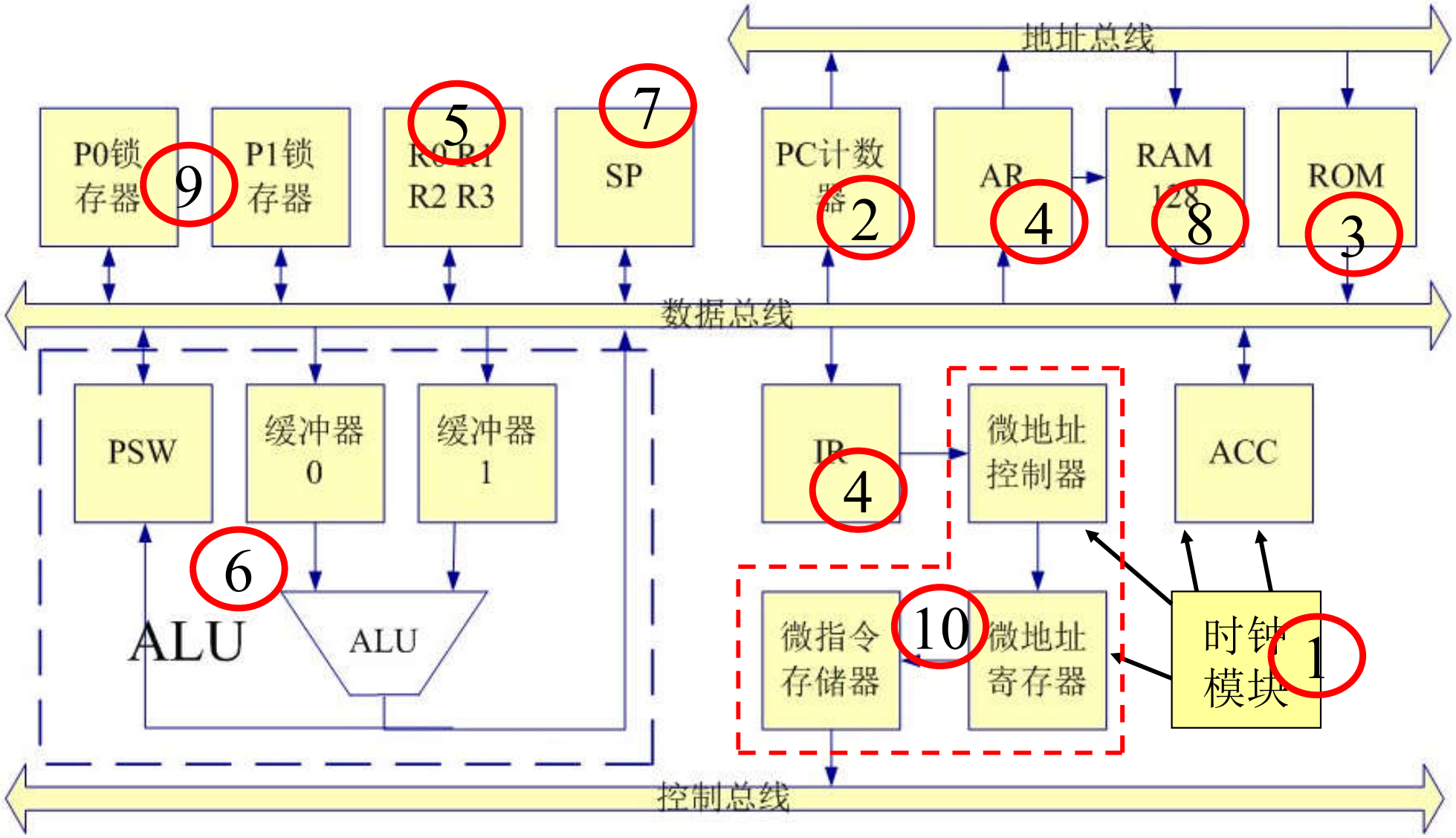
；SP←(SP)＋1，(SP)←(SP)7～PC0

；SP←(SP)＋1，(SP)←(SP)15～PC8

；PC11～PC0 ← addr11～addr0

- 定长指令周期（8个CPU周期）

6.6 CPU功能模块设计

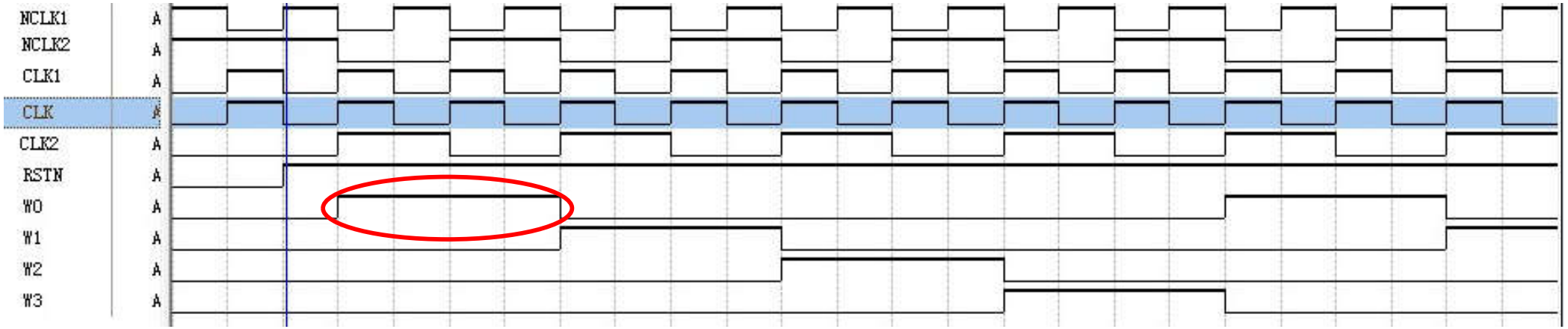


6.6.1 时钟节拍设计

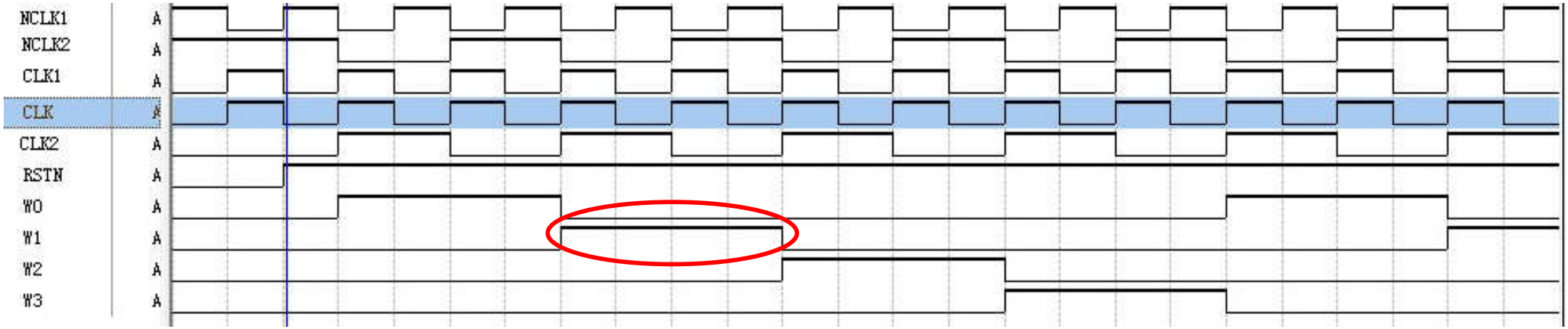
- 8个时钟节拍?
- 4个时钟节拍
- **一个节拍包含2个时钟周期**
- **时钟上升沿或下降沿触发**

```
entity clk_gen is
  port(
       clk,reset:in std_logic;          --时钟信号和复位信号
       clk1,nclk1:out std_logic;        --输出时钟信号及反向时钟信号
       clk2,nclk2:out std_logic;        --输出时钟信号的2分频及其反向时钟信号
       w0,w1,w2,w3:out std_logic        --节拍信号
      );
end clk_gen;
```

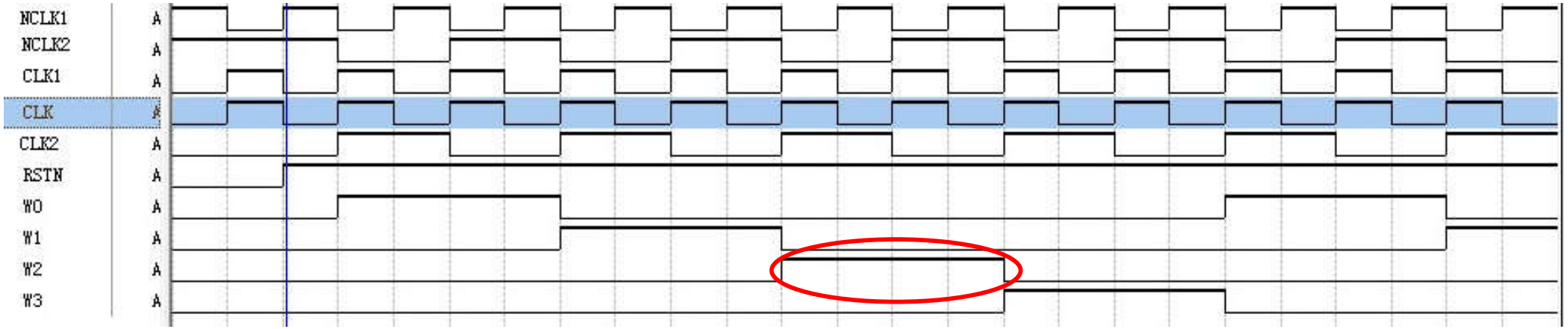
时钟模块仿真结果



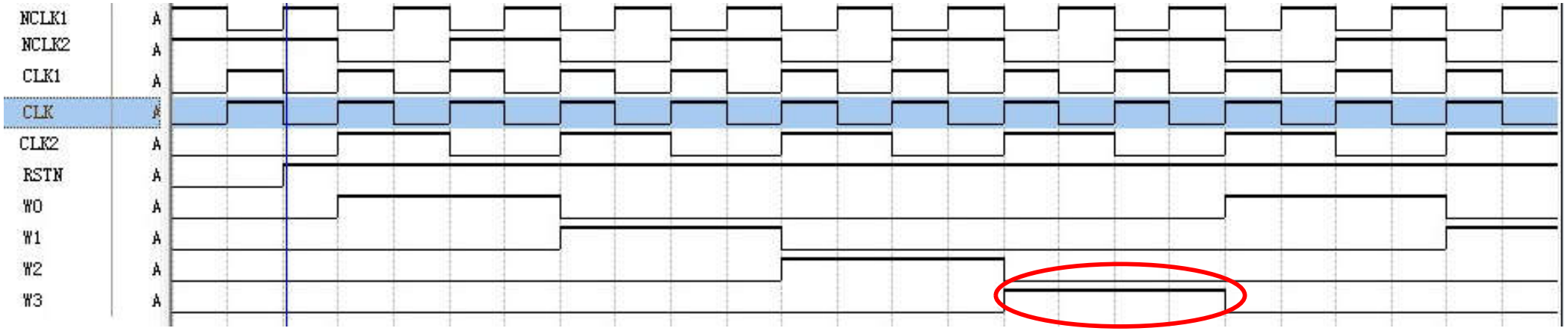
时钟模块仿真结果



时钟模块仿真结果



时钟模块仿真结果



指令中的数据通路

微操作	控制信号	功能说明
PC-->ADDR[11..0]	M_ROM; /ROM_EN	ROM片选信号有效，ROM读使能，PC指向程序入口地址

- PC程序计数器
- 只读存储器ROM
- 指令寄存器

6.6.2 PC程序计数器设计

- PC功能分析
  - ➢ 加1功能
  - ➢ 更新地址功能
  - ➢ PC数值送到数据总线

跳转指令：
JMP,JZ

调用函数指令：
CALL

计数器、锁存器

（1）PC端口信号分析

```
entity module_PC is
        port(
                        clk_PC:in std_logic;                        --PC时钟信号
                        nreset :in std_logic;                        --全局复位信号
                        nLD_PC:in std_logic;                         --装载新地址
                        M_PC:in std_logic;                           --PC加1控制信号
                        nPCH,nPCL::in std_logic;                     --PC输出总线控制信号
                        PC:in std_logic_vector(11 downto 0);         --PC指针
                        ADDR:out std_logic_vector(11 downto 0); --ROM读地址输出
                        d:inout std_logic_vector(7 downto 0)     --PC数值输出到数据总线
                );
end module_PC;
```

（2）PC功能实现分析

- 全局异步复位功能

  ADDR<=“000000000000”;

  数据总线高阻态;

- 加1功能

  clk_PC上升沿有效;

  M_PC高电平有效，PC+1→ADDR;

clk_PC=nclk2；

（2）PC功能实现分析

- 地址更新功能
- clk_PC上升沿有效，nLD_PC低电平有效
  新的PC→ADDR

- PC数值送到数据总线
  nPCH和nPCL低电平有效，注意分两次输出到总线上，先高8位后低8位。

6.6.3 程序存储器ROM设计

```
entity module_rom is
  port(
       clk_ROM :in std_logic;                                  --ROM时钟信号
       M_ROM   :in std_logic;                                  --ROM片选信号
       ROM_EN  :in std_logic;                                  --ROM使能信号
       addr    :in std_logic_vector(11 downto 0);--ROM地址信号
       data    :inout std_logic_vector(7 downto 0) --数据总线
      );
end module_rom;
```

clk_ROM=clk2 & nclk1

6.6.4 指令存储器IR设计

- IR功能分析
  - 传送指令编码到微控制器
  - 生成PC的新地址
  - 生成RAM的读写地址

IR不作译码操作，仅暂存数据。

（1）IR端口定义

```
entity module_IR is
  port(
      clk_IR              :in std_logic;                          --IR时钟信号
      nreset              :in std_logic;                          --复位信号
      LD_IR1,LD_IR2,LD_IR3 :in std_logic;                         --IR指令存储控制信号
      nARen               :in std_logic;                          --IR中RAM地址控制信号
      data                :inout std_logic_vector(7 downto 0);--数据总线
      IR                  :out std_logic_vector(7 downto 2);   --IR指令编码
      PC                  :out std_logic_vector(11 downto 0)   --PC新地址
      AR                  :out std_logic_vector(6 downto 0);   --RAM读写地址
      RS                  :out std_logic;                         --源寄存器
      RD                  :out std_logic;                         --目的寄存器
     );
end module_IR;
```

clk_IR=nclk2

（2）IR功能实现分析

- **传送指令编码到微控制器**

  clk_IR 上升沿有效，LD_IR1高电平有效

  data→IR。

- **寄存器地址操作**

Data[0]→ RS

Data[1]→ RD

（2）IR功能实现分析

- **生成PC的新地址**

clk_IR 上升沿有效，LD_IR2高电平有效，
data[3..0]→PC[11..8];

clk_IR 上升沿有效，LD_IR3高电平有效，
data[7..0]→PC[7..0]。

- **生成RAM的读写地址**

clk_IR 上升沿有效，LD_IR3高电平有效
data[7..0]→PC[7..0];

nARen低电平有效，PC[6..0]→AR[6..0]。

功能模块设计顺序

① 寄存器
② ALU
③ RAM
④ SP
⑤ IO端口
⑥ 微控制器

指令	字节	编码	说明
MOV Ri , #data	2	001 001	001001Rix xxxx xxxx
MOV Ri , Rj	1	001 010	001010RiRj
MOV Ri , direct	2	001 011	001011Rix xxxx xxxx
MOV direct , Ri	2	001 100	001011xRi xxxx xxxx
MOV PO , Ri	1	001 101	001101xRj
MOV Ri , PI	1	001 110	001110Rix
MOV SP , #data	2	001 111	001001xx xxxx xxxx
ANL Ri , Rj	1	010 001	010001RiRj
ORL Ri , Rj	1	010 010	010010RiRj
XRL Ri , Rj	1	010 011	010011RiRj
CLR Ri	1	010 100	010100Rix
CPL Ri	1	010 101	010101Rix
ADD Ri , Rj	1	011 001	011001R0R1
ADDC Ri , Rj	1	011 010	011010R0R1
INC Ri	1	011 011	011011Rix
DEC Ri	1	011 100	011100Rix
SUB Ri , Rj	1	011 101	011101R1R0
RL Ri	1	011 110	011110Rix
RR Ri	1	011 111	011111Rix

6.6.5 寄存器RN设计

- RN功能分析
- 数据锁存功能
- 读写功能

D触发器

```
entity module_Rn is
  port(
       clk_RN              :in std_logic;          --RN时钟信号
       nreset              :in std_logic;          --复位信号
       Ri_EN               :in std_logic;          --RN寄存器使能
       RDRi,WRRi           :in std_logic;          --RN读写信号
       RS                                          --源寄存器地址
       RD                                          --目的寄存器地址
       data                                        downto 0)    --数据总线
       );
end module_Rn;
```

clk_RN=nclk2

RN功能实现分析

- 读寄存器操作

  clk_RN 上升沿有效，Ri_EN低电平有效，读信号RDRi高电平有效，选择RS寄存器，输出data[7..0]。

- 写寄存器操作

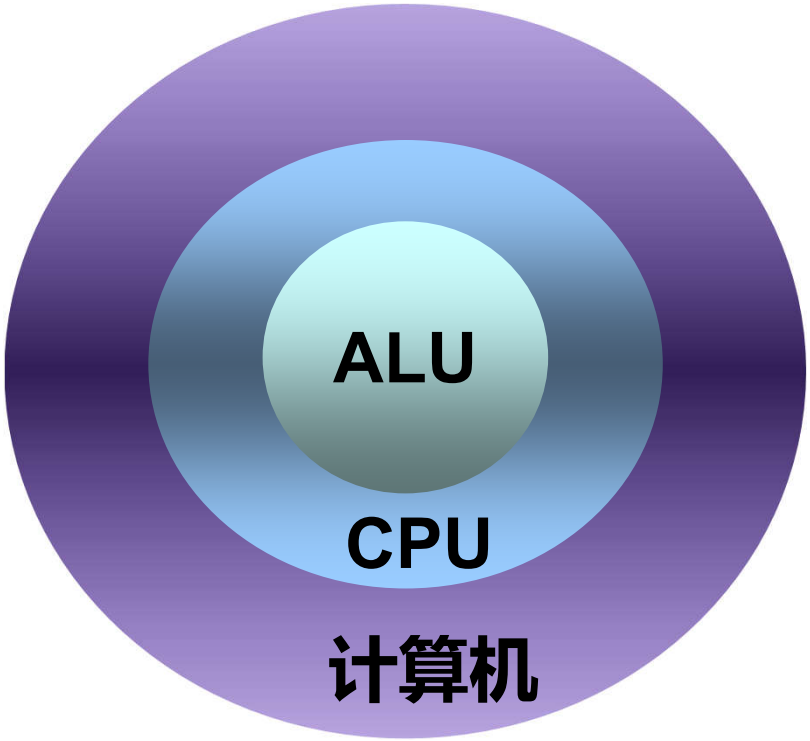
  clk_RN 上升沿有效，Ri_EN低电平有效，写信号WRRi高电平有效，选择RD寄存器，data[7..0]→ RD 。

6.6.6 ALU模块设计

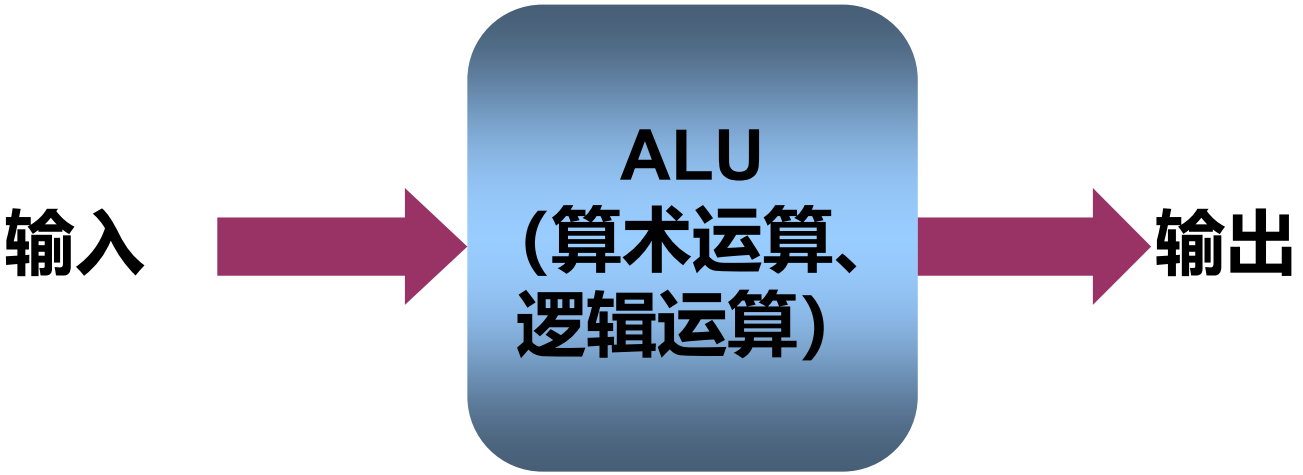
- ALU功能与结构
- ALU设计方法
- 8位ALU设计

6.6.6.1 ALU功能与结构

- 算术逻辑单元（ALU）
  执行各种算术和逻辑运算
- 算术运算操作
  加、减、乘、除
- 逻辑运算操作
  与、或、非、异或



6.6.6.1 ALU功能与结构

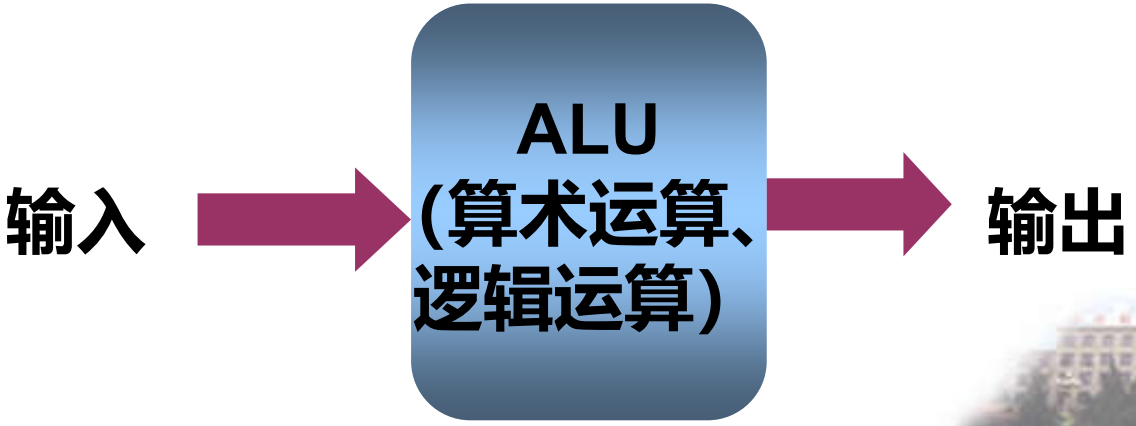


6.6.6.1 ALU功能与结构

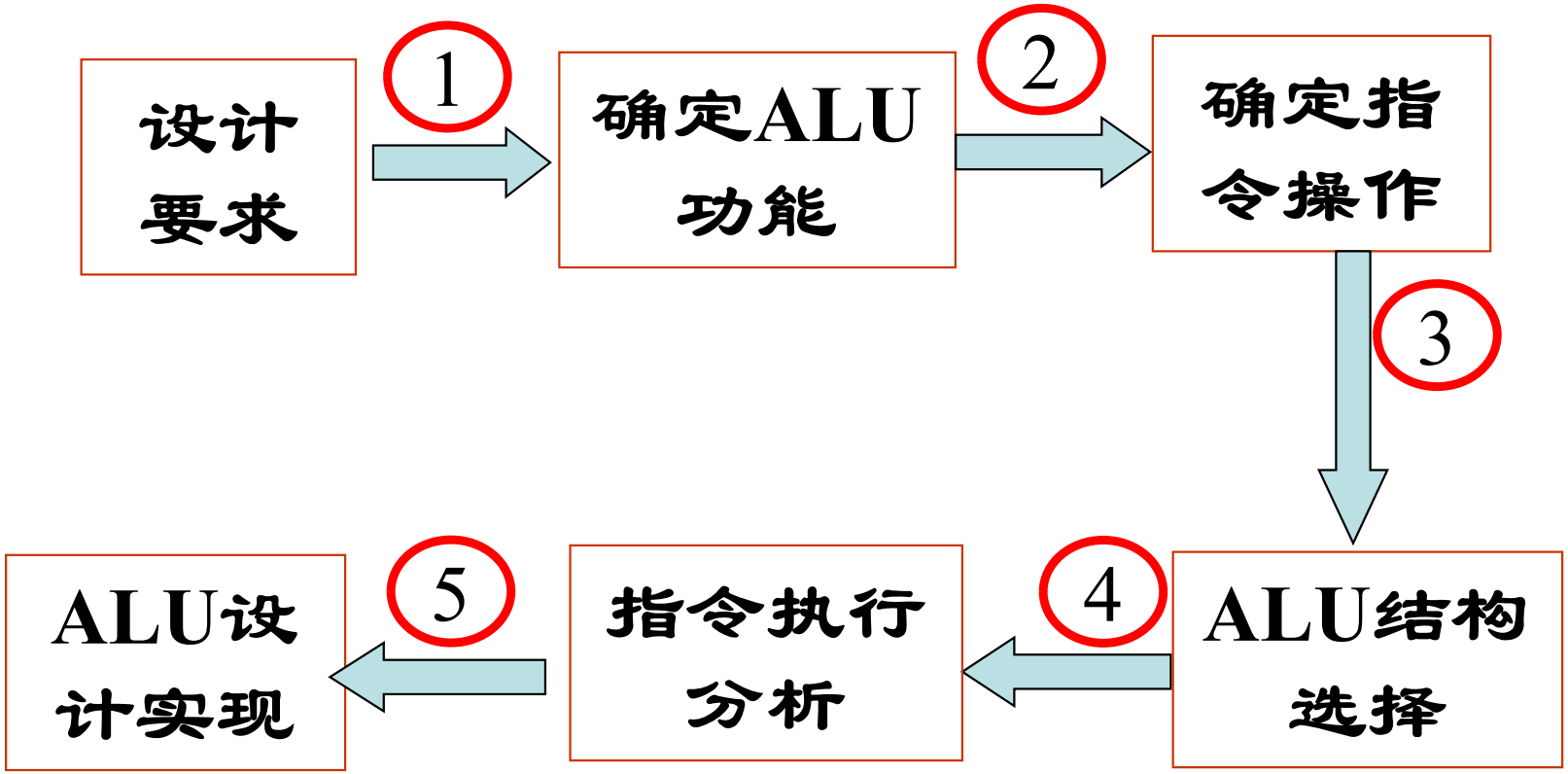
ADD R0,R1;

- **ALU输入**
  操作数以及来自控制单元的控制命令

- **ALU输出**
  运算结果，以及状态信息



6.6.6.2 ALU设计方法



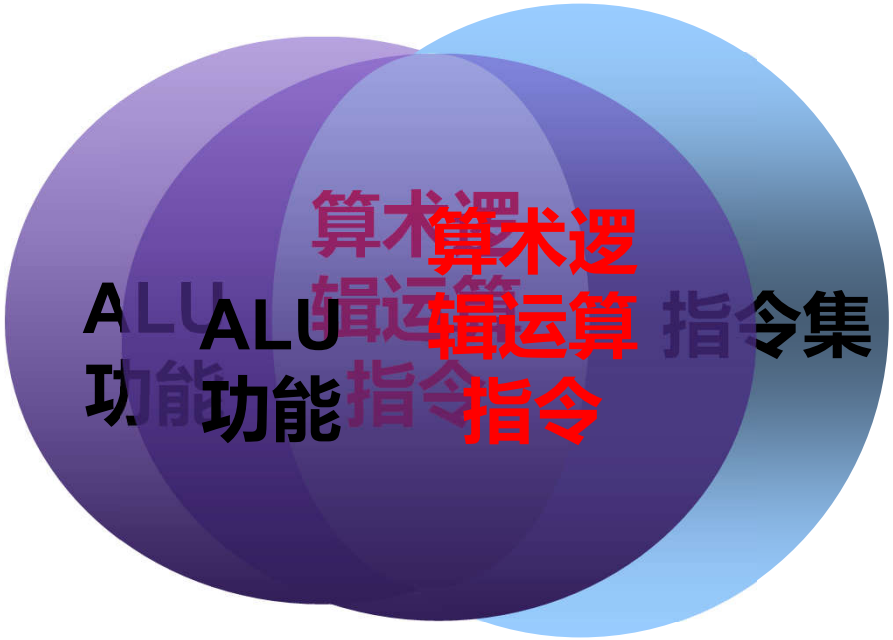
（1）确定ALU功能

- **算术运算：加减运算**
  - 不带进位加法运算
  - 不带进位减法运算
  - 带进位加法运算
  - 带进位减法运算
- **逻辑运算**
  - 基本逻辑运算
  - 混合逻辑运算

（2）确定指令操作

- ALU功能必须支持指令集中所有的算术运算和逻辑运算类型指令。
- 系统可扩展性。

ADDC R0,R1;



（2）确定指令操作

- **算术运算指令**

指令	指令编码	指令说明	指令功能
ADD Ri , Rj	011 001	011001RiRj	Ri ← (Ri) + (Rj)
ADDC Ri , Rj	011 010	011010RiRj	Ri ← (Ri) + (Rj) + (Cy)
INC Ri	011 011	011011RiX	Ri ← (Ri) + 1
DEC Ri	011 100	011100RiX	Ri ← (Ri) − 1
SUBB Ri , Rj	011 101	011101RiRj	R0 ← (R0)−(R1)− (Cy)

（2）确定指令操作

- **逻辑运算指令**

指令	编码	说明	解释
ANL Ri , Rj	010 001	010001RiRj	Ri ← (Ri) ∧ (Rj)
ORL Ri , Rj	010 010	010010RiRj	Ri ← (Ri) ∨ (Rj)
XRL Ri , Rj	010 011	010011RiRj	Ri ← (Ri) ⊕ (Rj)
CLR Ri	010 100	010100Rix	把 Ri 中的数据清零
CPL Ri	010 101	010101Rix	把 Ri 中的数据取反

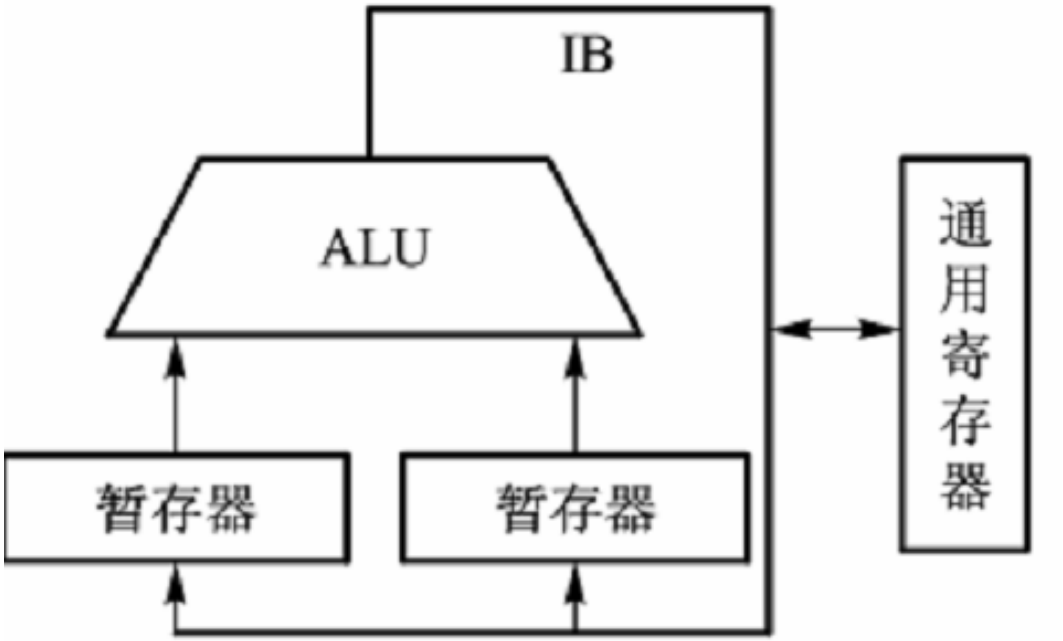
（3）ALU结构选择

根据运算器内部总线与构成运算器的基本部件的连接情况运算器分为 3 种基本结构：

- **单总线结构**
- **双总线结构**
- **三总线结构**

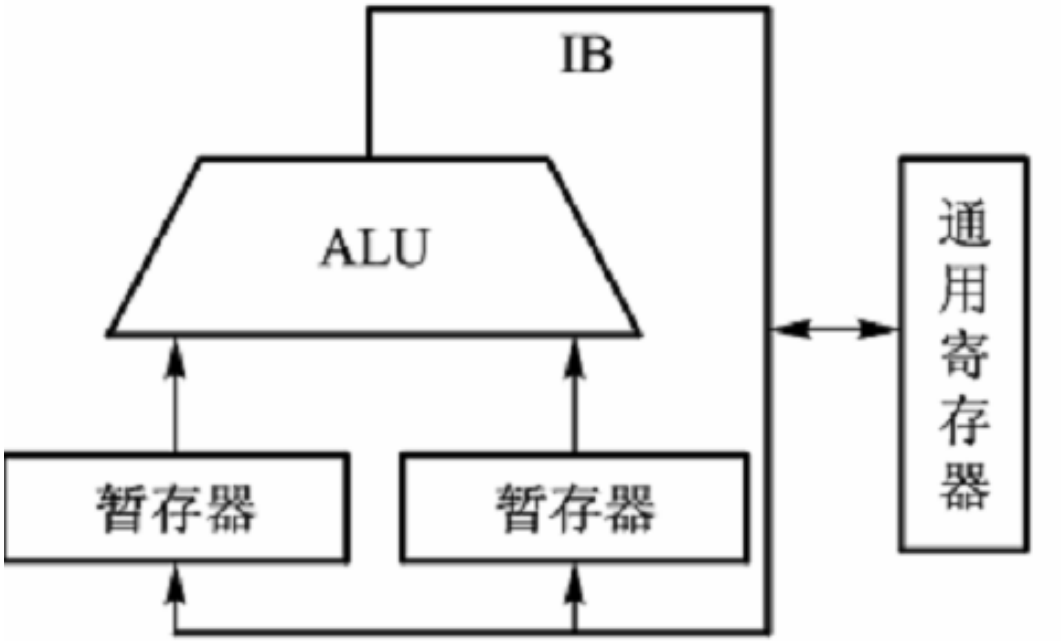
单总线结构

- 所有部件都接到同一总线上，数据可以在任何两个寄存器之间,或者在任一个寄存器和ALU之间传送。
- 在同一时间内,只能有一个操作数放在总线上进行传输。



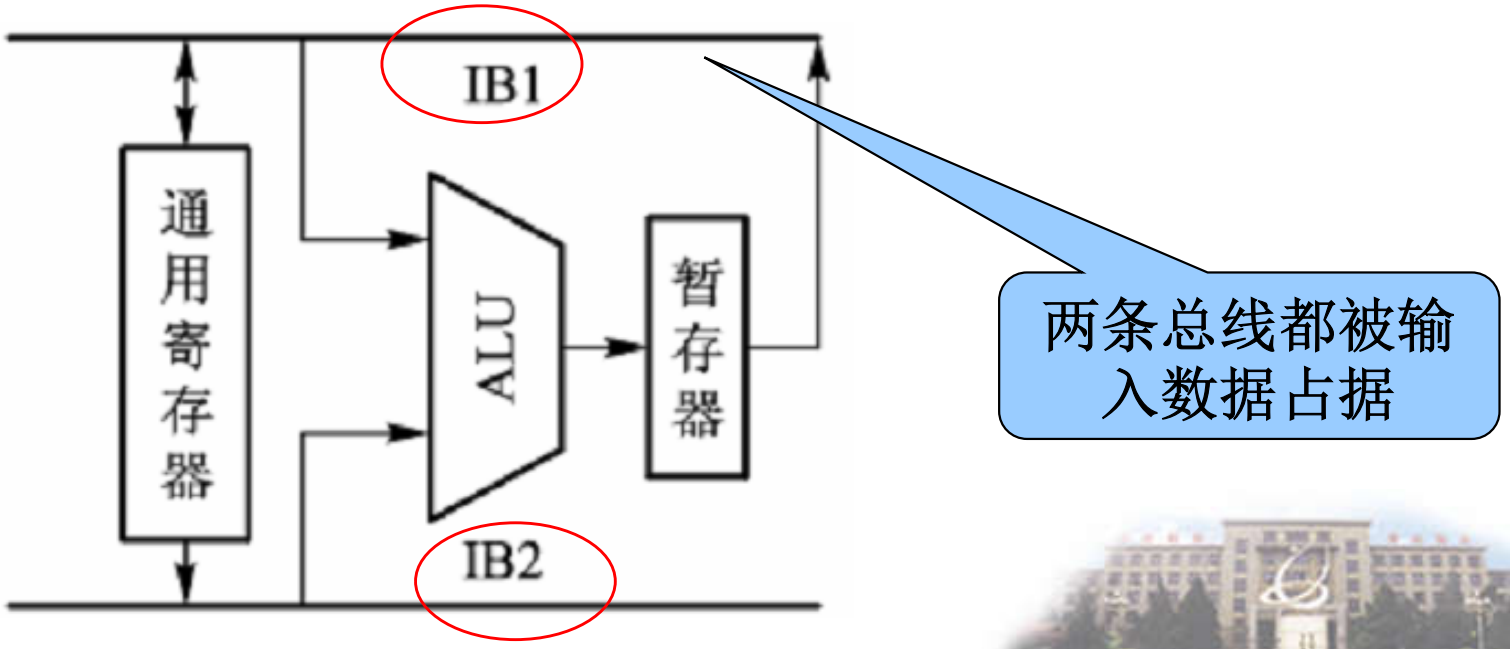
单总线结构

- 需要分两次才能将两个操作数输入到ALU，并且需要A、B两个缓冲寄存器。
- 优点：控制电路比较简单。
- 缺点：操作速度较慢。



双总线结构

- 两个操作数同时加到ALU进行运算,只需一次操作控制, 可得到运算结果。
- ALU的输出不能直接加到总线上去。
- 必须在ALU输出端设置缓冲寄存器。

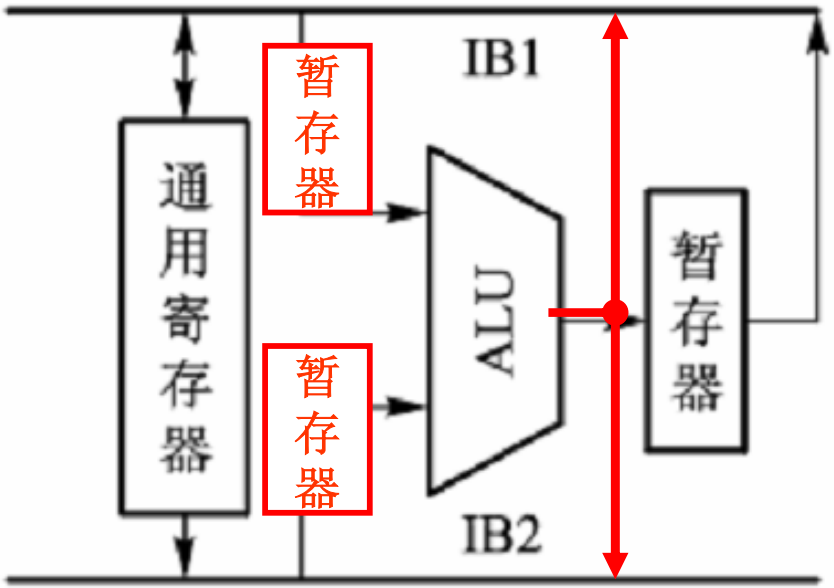


双总线结构

- **操作的控制要分两步完成:**

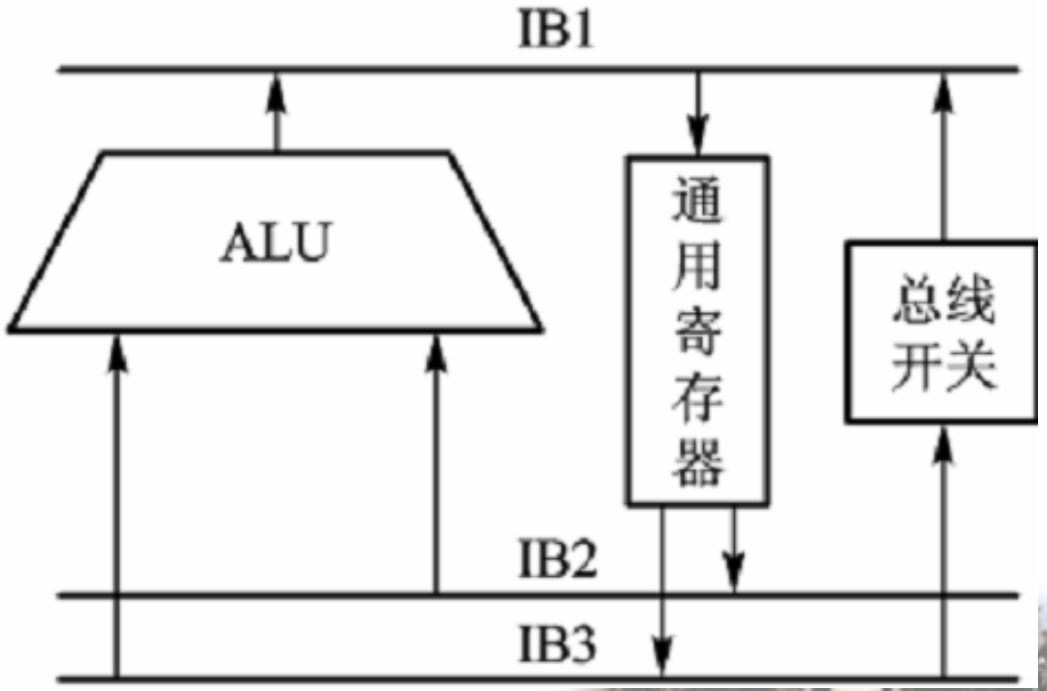
(1) 在ALU的两个输入端输入操作数,形成结果并送　入缓冲寄存器;

(2) 暂存器将结果送入目的寄存器。



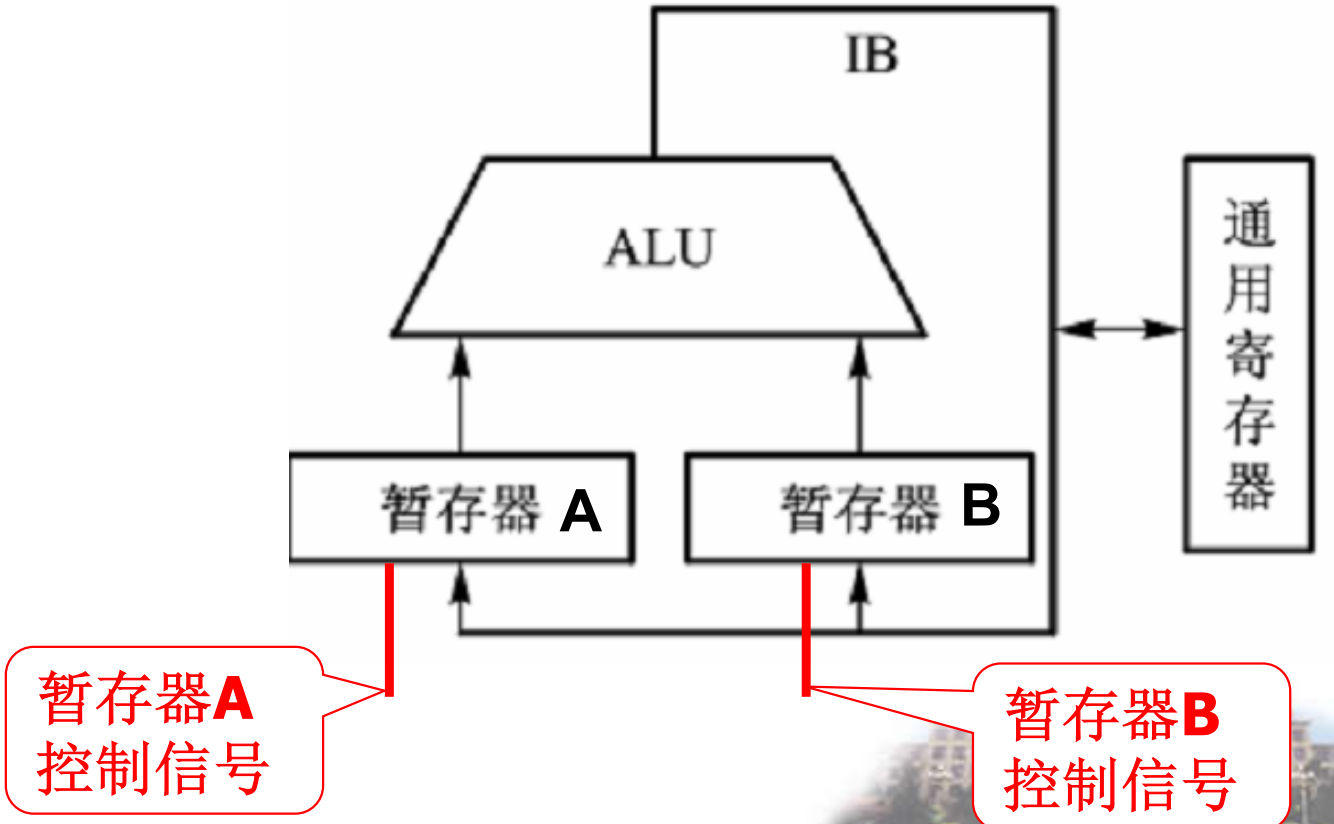
三总线结构

- ALU两个输入端分别连接两条总线，ALU的输出与第三条总线相连。
- 附加直接传送功能，当一个操作数不需要修改,可通过总线开关将数据从输入总线直接传送到输出总线。
- 特点是操作时间快。
- 缺点是结构复杂。



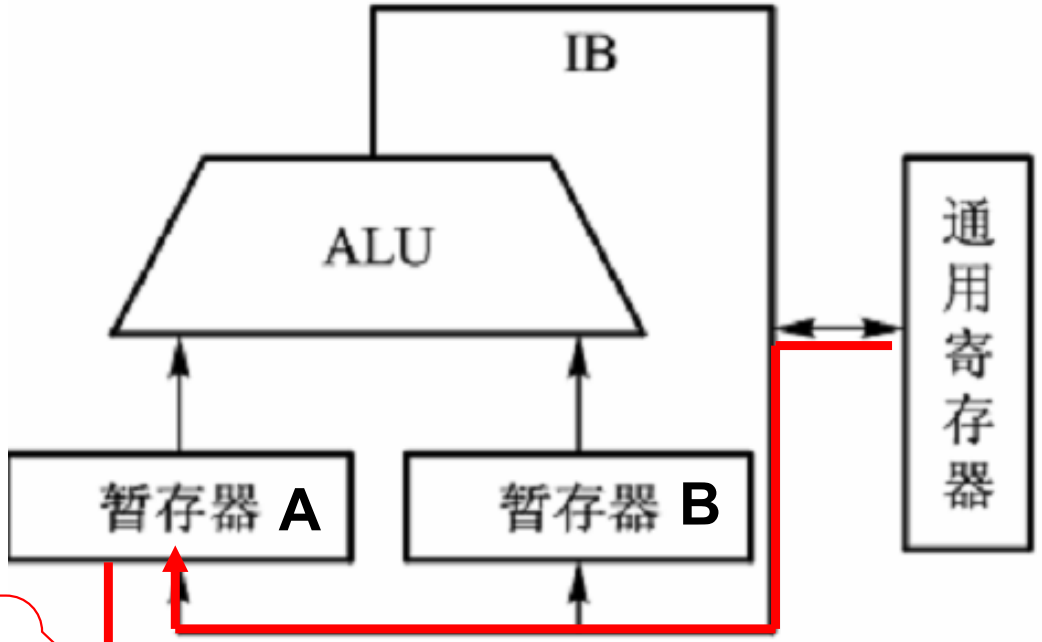
（4）指令执行分析

- **ADD R0，R1；**
- **两个控制信号不能同时有效。**



（3）指令执行分析

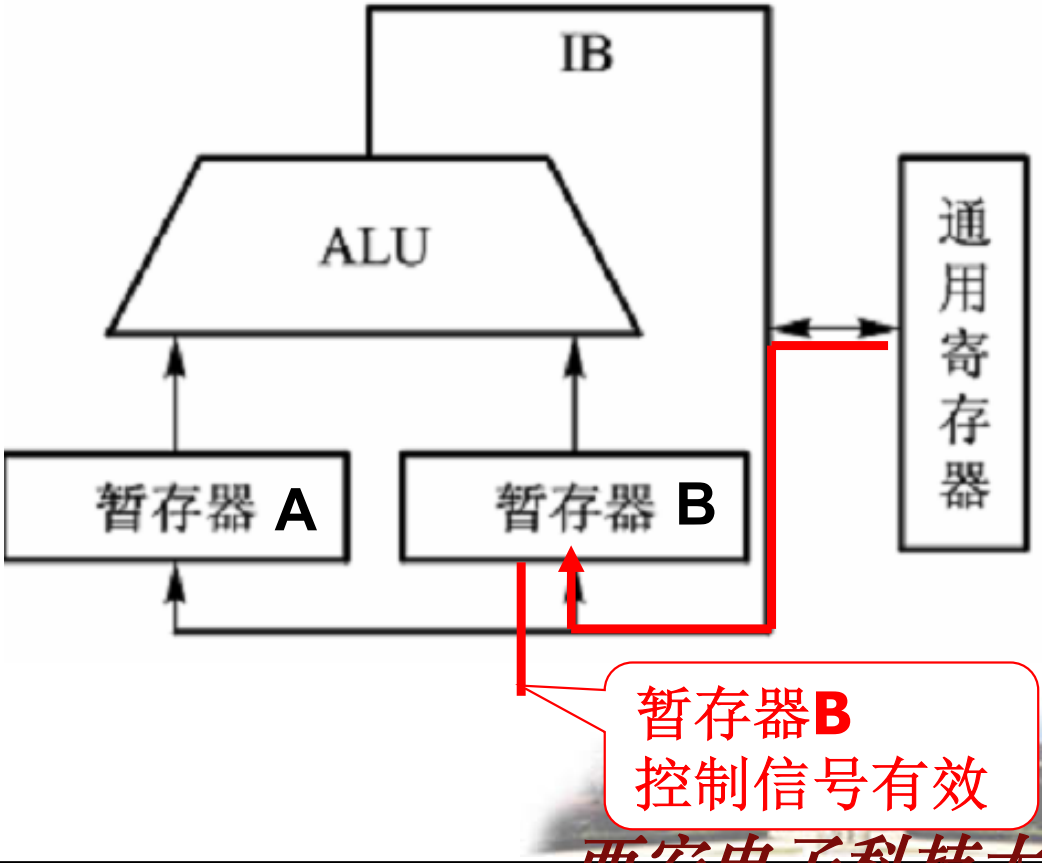
- ADD R0，R1；
- 在**时钟上升沿**有效，暂存器A控制信号有效，R0→总线→暂存器A 。



（4）指令执行分析

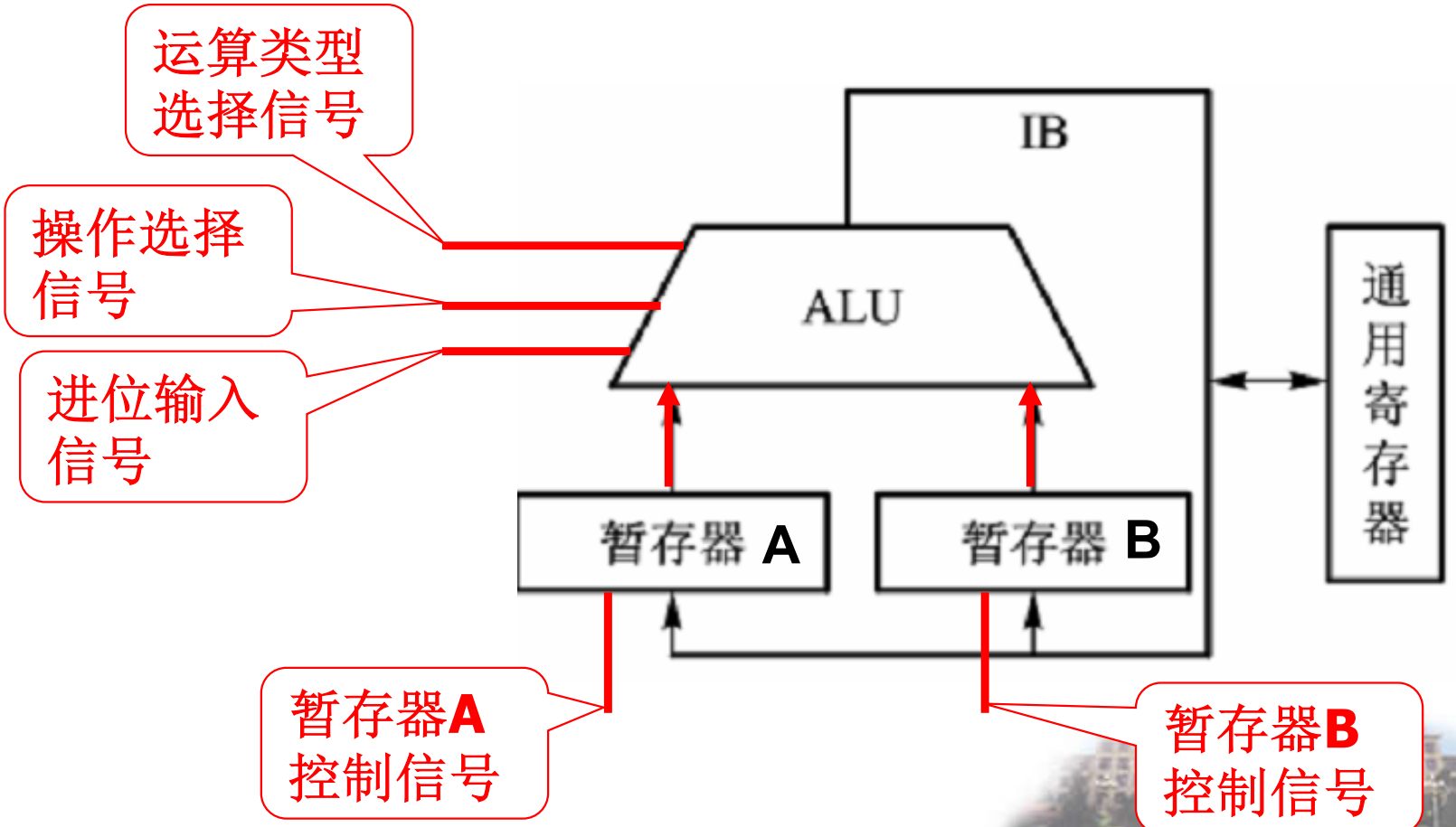
两个控制信号在不同的时钟节拍内有效，实现暂存器的控制选择。

- ADD R0，R1
- 在下一个时钟上升沿有效的时候，暂存器B控制信号有效，R1→总线→暂存器B。



（4）指令执行分析

- **ADD R0，R1；**



（5）ALU设计实现

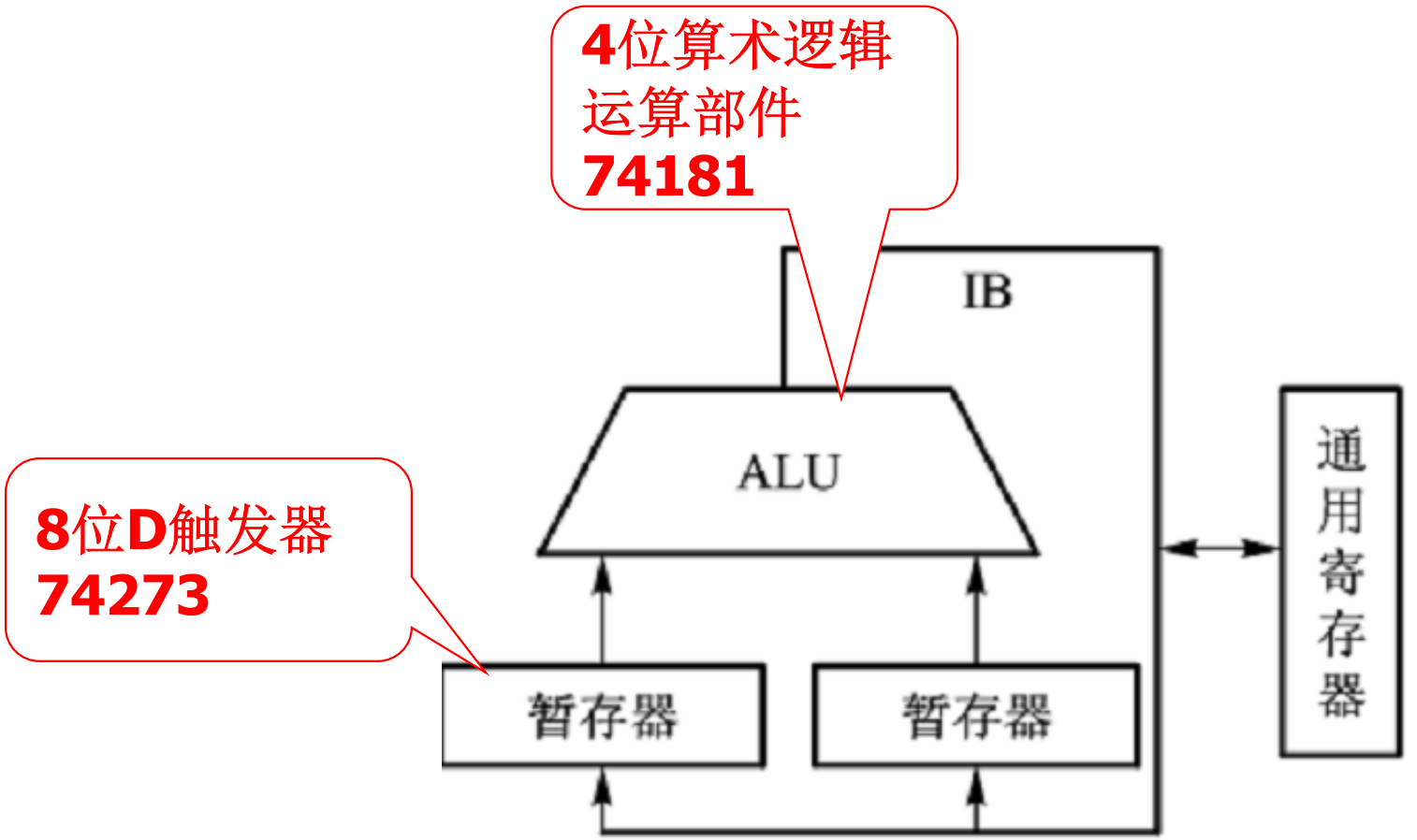
- 基于器件电路的ALU设计
- 基于硬件描述语言的ALU设计

} FPGA验证

6.6.6.3 8位ALU设计

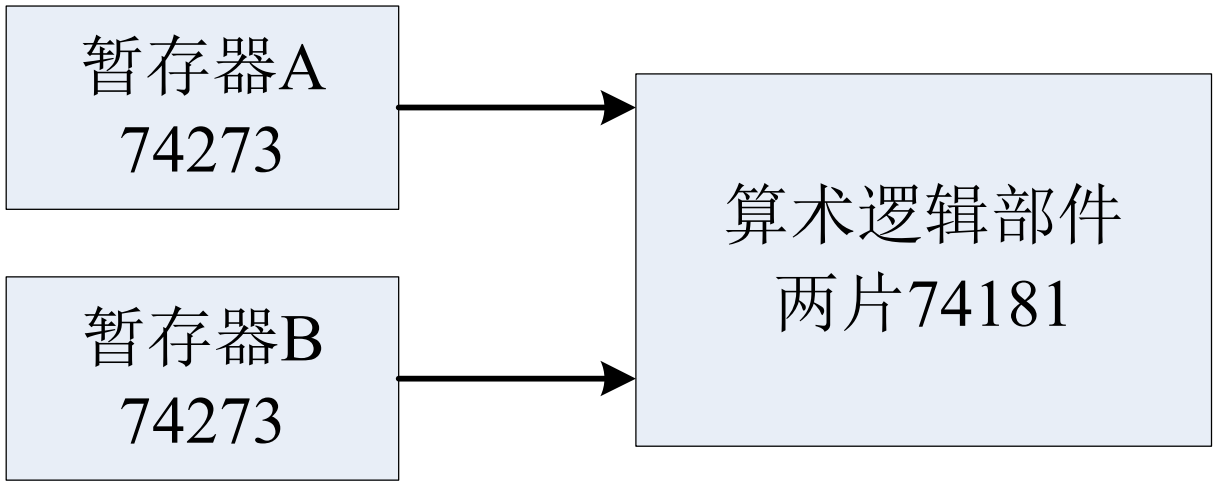
- 基于器件的8位ALU设计
- 基于VHDL的8位ALU设计

（1）基于器件的8位ALU设计

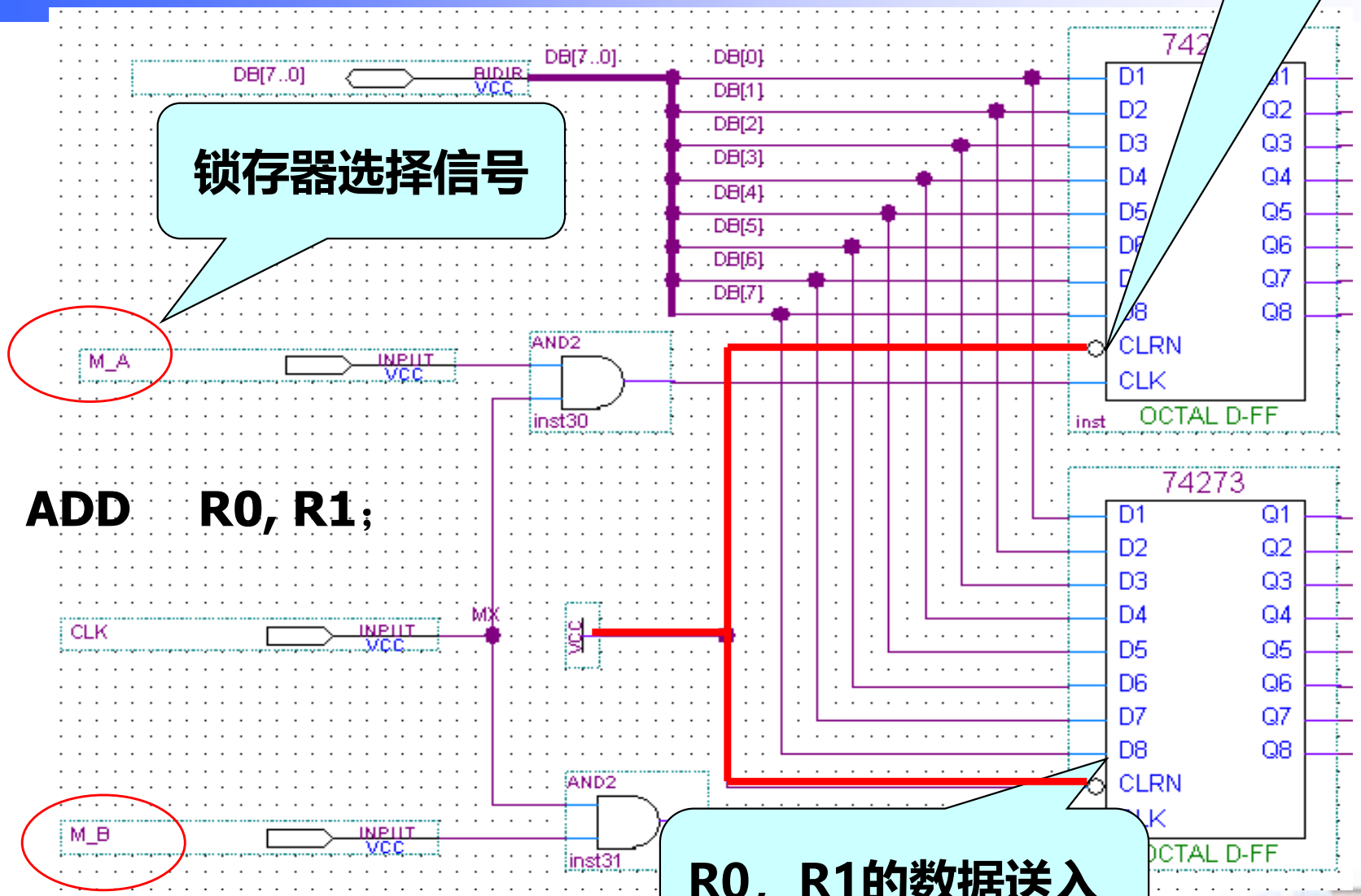


（1）基于器件的8位ALU设计

- **暂存器设计**
- **算术逻辑运算部件设计**



① 暂存器电路实

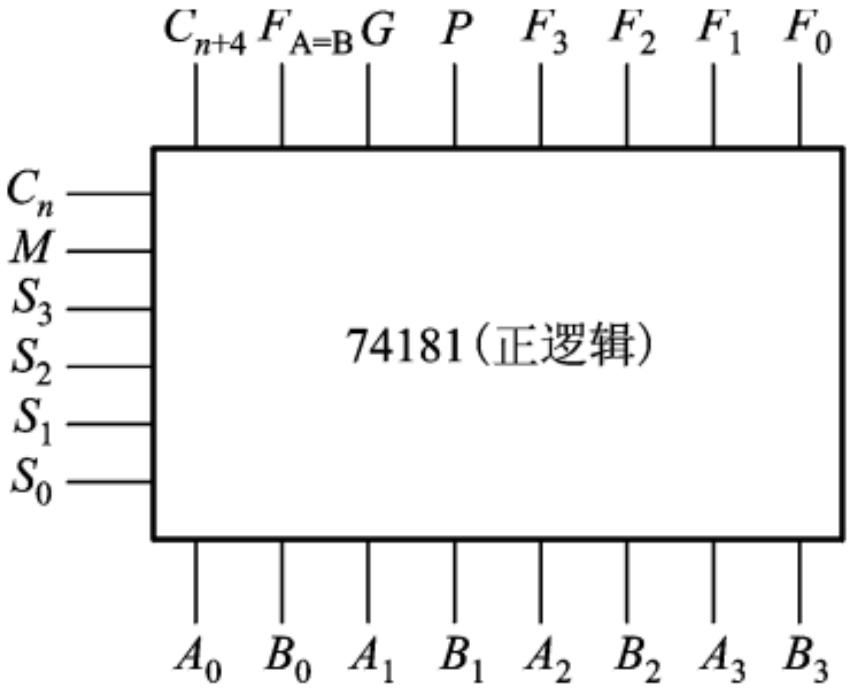

清零信号CLRN
接高电平
锁存器选择信号
ADD R0, R1;
R0，R1的数据送入
到暂存器A和B。
DB[7..0]
BIDIR
VCC
DB[0]
DB[1]
DB[2]
DB[3]
DB[4]
DB[5]
DB[6]
DB[7]
M_A
INPUT
AND2
inst30
CLRN
CLK
OCTAL D-FF
inst
74273
D1
D2
D3
D4
D5
D6
D7
D8
Q1
Q2
Q3
Q4
Q5
Q6
Q7
Q8
CLK
MX
M_B
inst31

② 算术逻辑运算部件设计

- 根据设计的指令集，包括加法、减法、与或非等算术逻辑运算。
- 如何利用现有的多位的算术逻辑器件构建ALU中的算术逻辑运算部件。

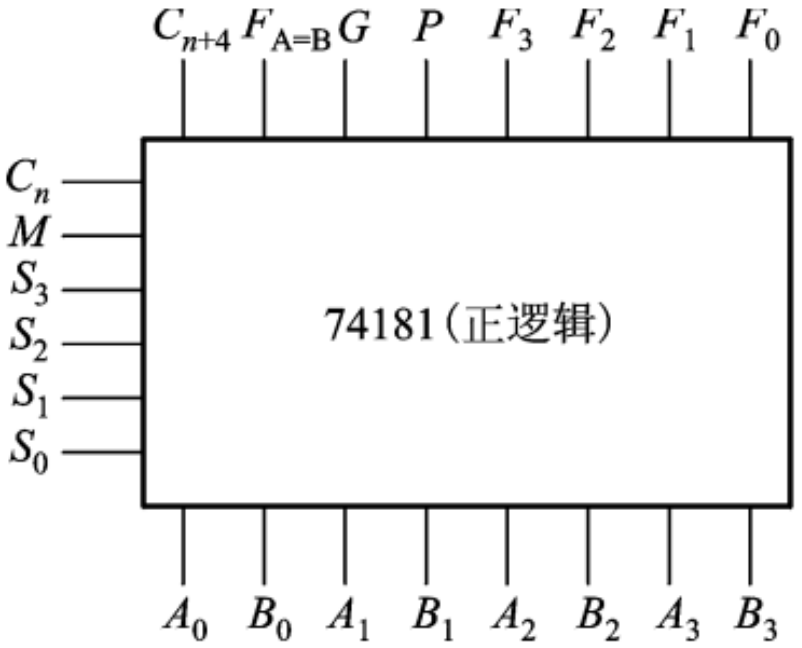
4位算术逻辑运算器件74181

- M 运算类型选择输入
- S3—S0 操作选择输入
- Cn 进位输入
- A3—A0 数据A输入
- B3—B0 数据B输入



4位算术逻辑运算器件74181

- Cn+4 进位输出
- $F_{A=B}$ A=B比较输出
- G 进位产生输出
- P 进位传送输出
- F3—F0 运算结果输出

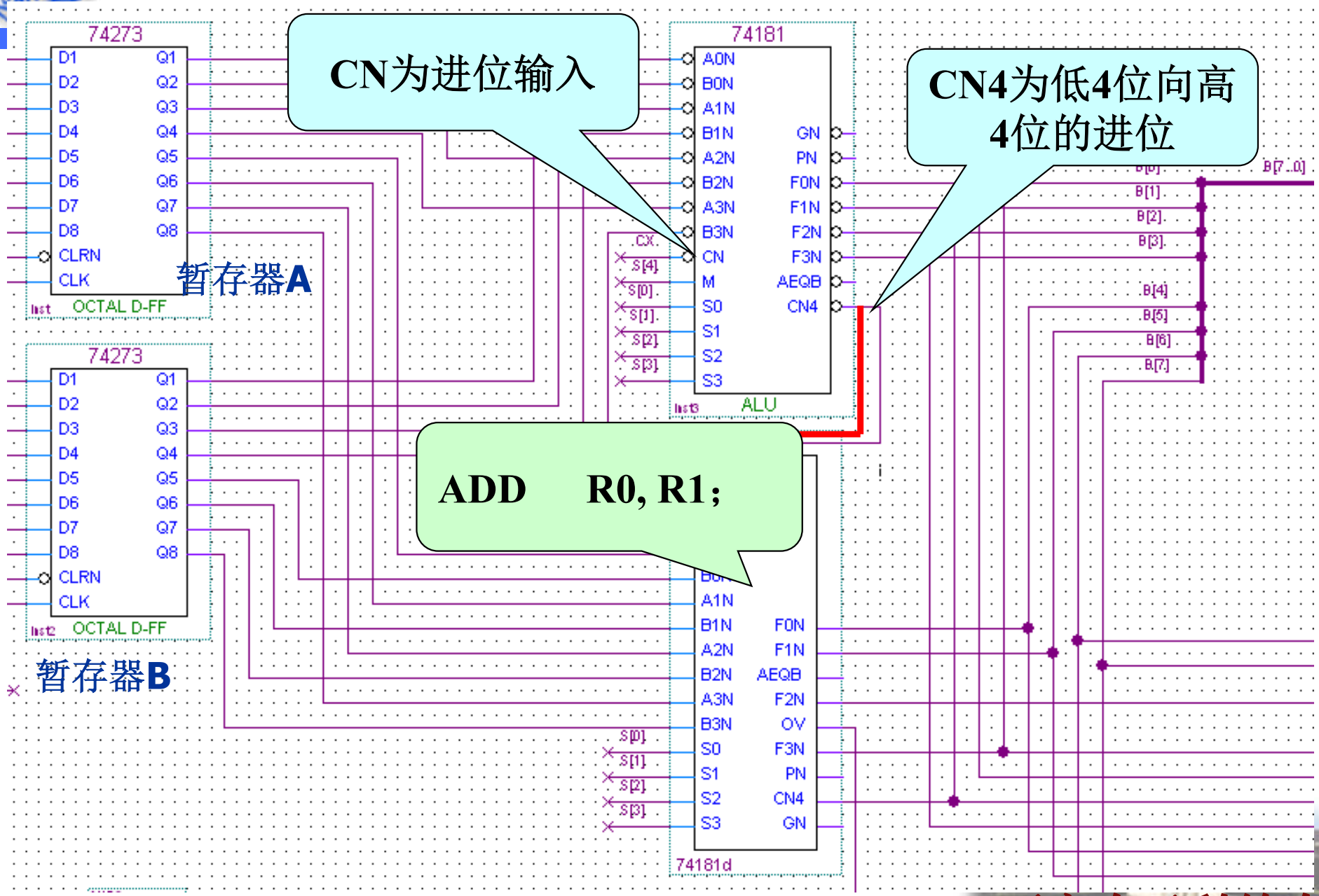


74181运算功能

操作选择				运算	
S_3 S_2 S_1 S_0	$M=1$ 逻辑运算	Cn=1（无进位）	
0 0 0 0	$F=\overline{A}$	$F=A$	$F=A$加1
0 0 0 1	$F=\overline{A+B}$	$F=A+B$	$F=(A+B)$加1
0 0 1 0	$F=\overline{A}B$	$F=A+\overline{B}$	$F=(A+\overline{B})$加1
0 0 1 1	$F=0$	$F=$减1	$F=0$
0 1 0 0	$F=\overline{AB}$	$F=A$加$A\overline{B}$	$F=A$加$A\overline{B}$加1
0 1 0 1	$F=\overline{B}$	$F=(A+B$加$)A\overline{B}$	$F=(A+B)$加$A\overline{B}$加1
0 1 1 0	$F=A\oplus B$	$F=A$减B减1	$F=A$减B
0 1 1 1	$F=A\overline{B}$	$F=A\overline{B}$减1	$F=A\overline{B}$
1 0 0 0	$F=A+B$	$F=A$加AB	$F=A$加AB加1
1 0 0 1	$F=\overline{A\oplus B}$	$F=A$加B	$F=A$加B加1
1 0 1 0	$F=B$	$F=(A+\overline{B})$加AB	$F=(A+\overline{B})$加AB加1
1 0 1 1	$F=AB$	$F=AB$减1	$F=AB$
1 1 0 0	$F=1$	$F=A$加A(相当A乘以2)	$F=A$加A加1
1 1 0 1	$F=A+\overline{B}$	$F=(A+B)$加A	$F=(A+\overline{B})$加A加1
1 1 1 0	$F=A+B$	$F=(A+\overline{B})$加A	$F=(A+B)$加A加1
1 1 1 1	$F=A$	$F=A$减1	$F=A$

“加”指算术加；
符号“＋”指“逻辑加”；
减法采用补码进行。

算术逻辑部件电路实现


74273
D1
D2
D3
D4
D5
D6
D7
D8
CLRN
CLK
Q1
Q2
Q3
Q4
Q5
Q6
Q7
Q8
OCTAL D-FF
暂存器A
74273
OCTAL D-FF
暂存器B
CN为进位输入
74181
A0N
B0N
A1N
B1N
A2N
B2N
A3N
B3N
CN
M
S0
S1
S2
S3
GN
PN
F0N
F1N
F2N
F3N
AEQB
CN4
ALU
CN4为低4位向高4位的进位
ADD R0, R1;
OV
74181d
B[7..0]

加法运算实现过程

- ADD R0, R1;
  指令编码为011 00100，即64H
- **取指操作**

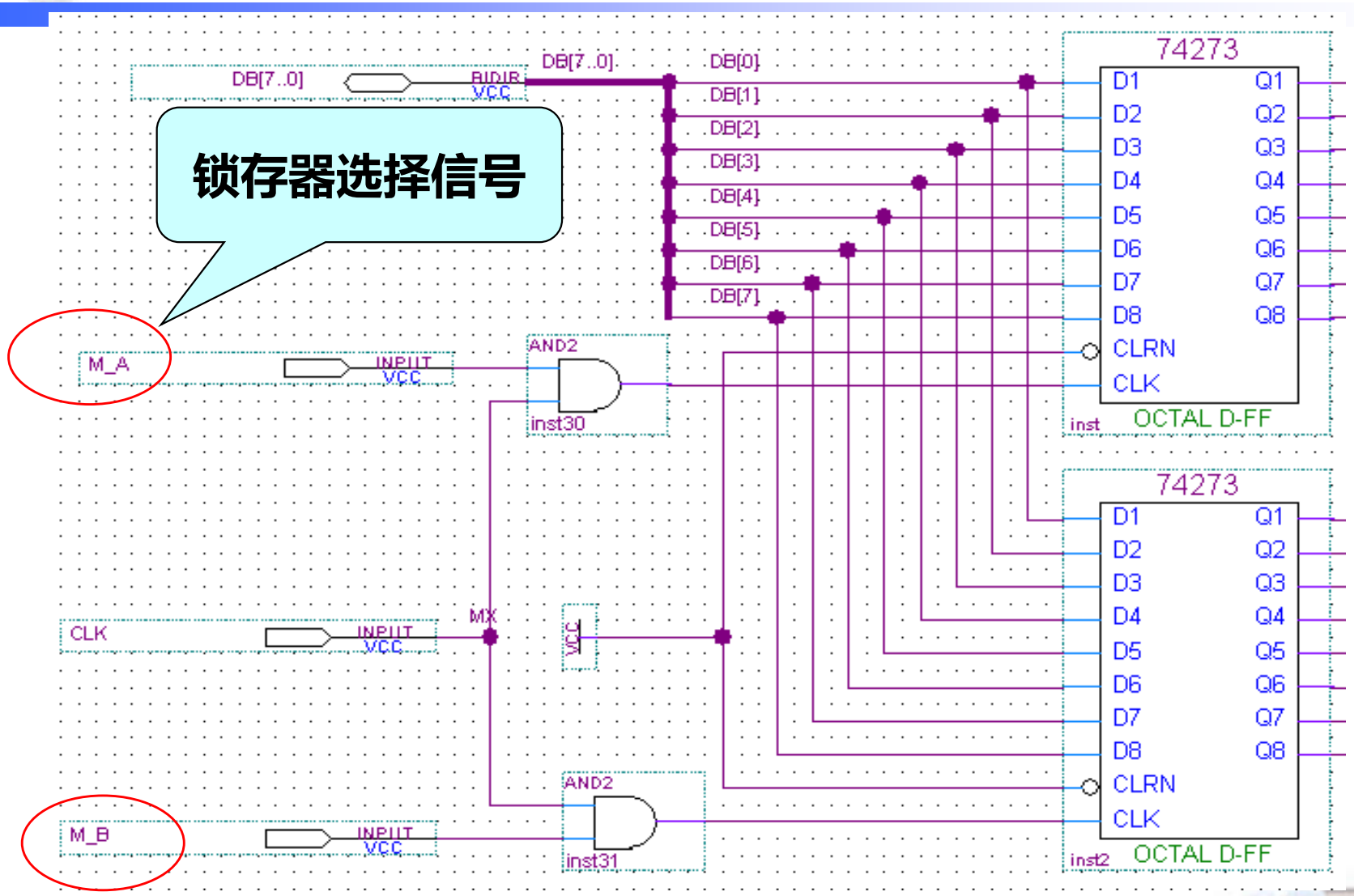
微操作	控制信号	功能说明
PC-->ADDR[11..0]	M_ROM; /ROM_EN	ROM片选信号有效，ROM读使能，PC指向程序入口地址
BUS-->IR; PC =PC + 1 ;	LDIR1; M_PC	IR使能，指令通过总线传送到IR，PC+1。
IR -->Microcontrol; addr[7:0]->CM[47:0]	M_uROM	微控制器使能，IR送入指令，生成控制信号。

加法运算实现过程

- 取操作数

微操作	控制信号	功能说明
R0-->BUS	RDRi， /Ri_EN	寄存器读使能，读信号有效，R0数据送到数据总线
BUS-->A	M_A	暂存器A使能，数据从总线输入到暂存器A
R1-->BUS	RDRi， /Ri_EN	寄存器读使能，读信号有效，R1数据送到数据总线
BUS-->B	M_B	暂存器B使能，数据从总线输入到暂存器B

暂存器电路

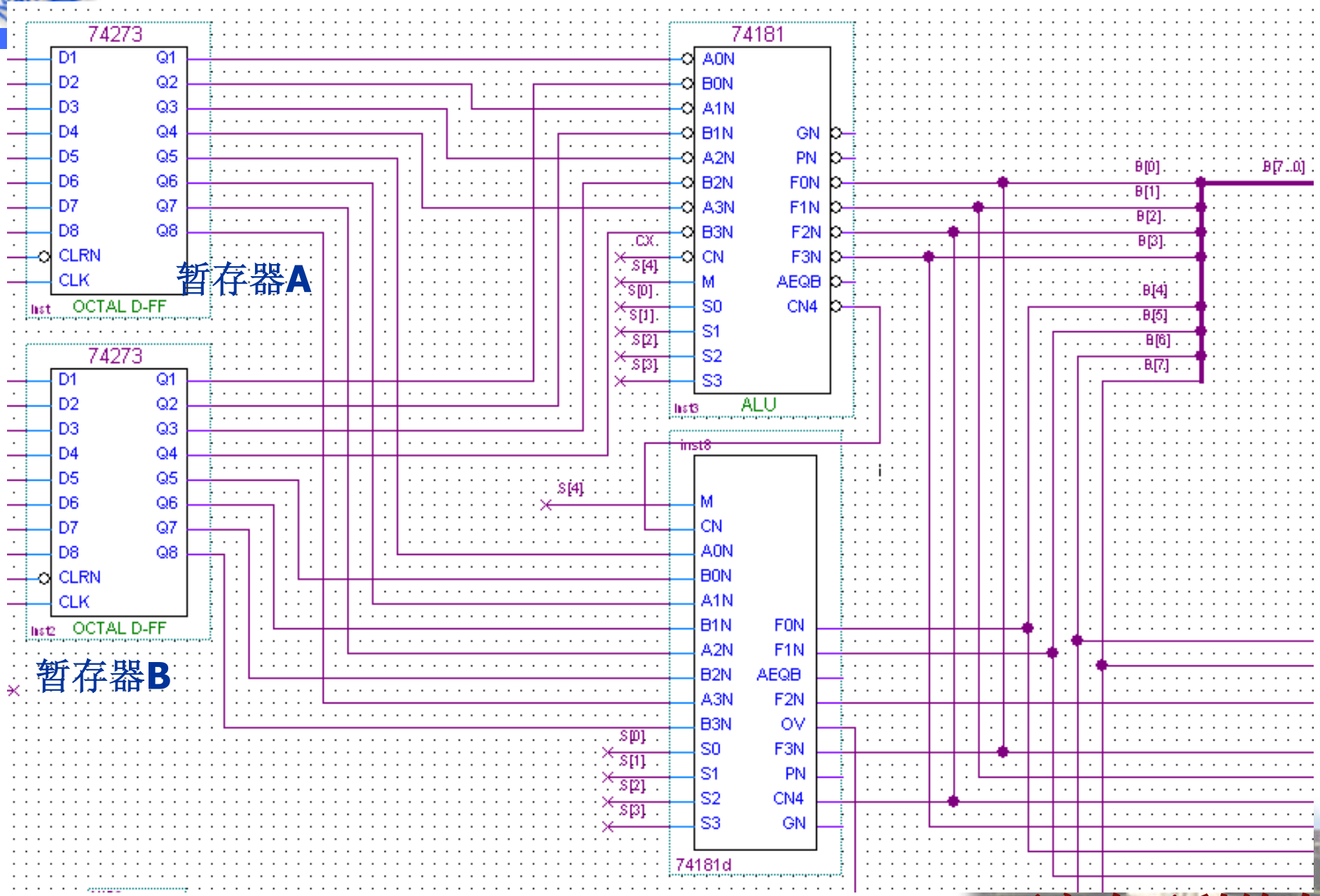

锁存器选择信号
DB[7..0]
BIDIR
VCC
DB[7..0]
DB[0]
DB[1]
DB[2]
DB[3]
DB[4]
DB[5]
DB[6]
DB[7]
74273
D1
D2
D3
D4
D5
D6
D7
D8
CLRN
CLK
Q1
Q2
Q3
Q4
Q5
Q6
Q7
Q8
inst
OCTAL D-FF
M_A
INPUT
VCC
AND2
inst30
CLK
MX
VCC
M_B
inst31
inst2

加法运算实现过程

- **执行指令**

微操作	控制信号	功能说明
A-->ALU， B-->ALU	M=0； Cn=1； S3…S0=1001	暂存器数据送到ALU，选择不带进位算术加法运算
ALU-->BUS	/ALU_EN=0	ALU输出使能，ALU运算结果输出到数据总线
BUS-->R0	M_Rn; WRRi	寄存器使能，写信号有效，数据通过总线写入R0寄存器

算术逻辑部件电路实现


74273
D1
D2
D3
D4
D5
D6
D7
D8
CLRN
CLK
Q1
Q2
Q3
Q4
Q5
Q6
Q7
Q8
OCTAL D-FF
暂存器A
74273
OCTAL D-FF
暂存器B
74181
A0N
B0N
A1N
B1N
A2N
B2N
A3N
B3N
CN
M
S0
S1
S2
S3
GN
PN
F0N
F1N
F2N
F3N
AEQB
CN4
ALU
inst8
OV
74181d
B[0]
B[1]
B[2]
B[3]
B[4]
B[5]
B[6]
B[7]
B[7..0]

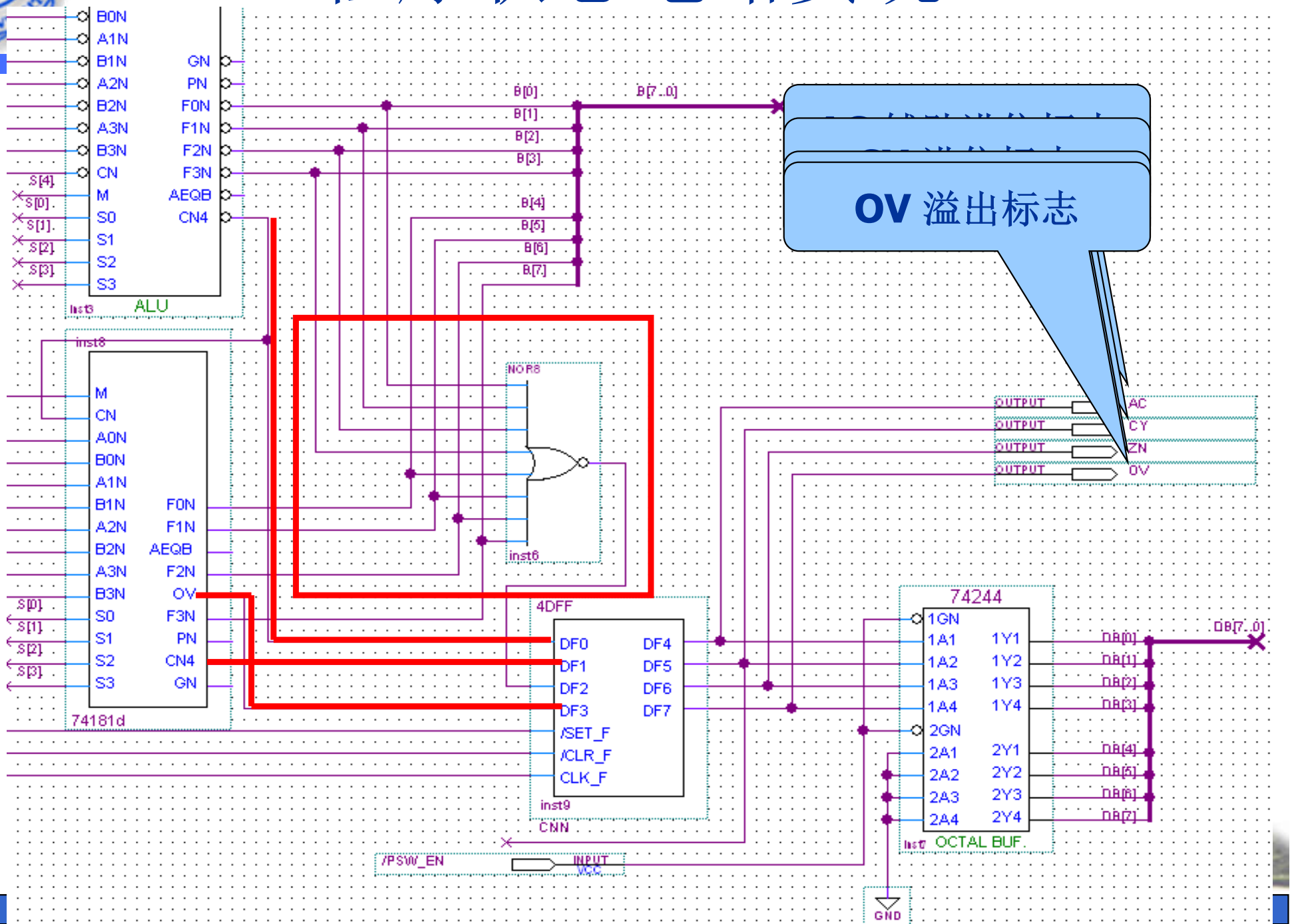
③ 程序状态标志设计

- 算术逻辑运算影响程序状态
  - AC (PSW.0) 辅助进位标志位，用于BCD码的十进制调整运算。
  - CY(PSW.1) 进位标志位 在执行算术指令时，指示运算是否产生进位。
  - ZN(PSW.2) 零标志位 用来判断最近一次的运算结果是否为零。
  - OV(PSW.3) 溢出标志位 在执行算术指令时，指示运算是否产生溢出。

③ 程序状态标志设计

- 用户可以访问和控制程序状态
- 具有置位、清零、数据存储功能
- 集成置位、清零功能的D触发器

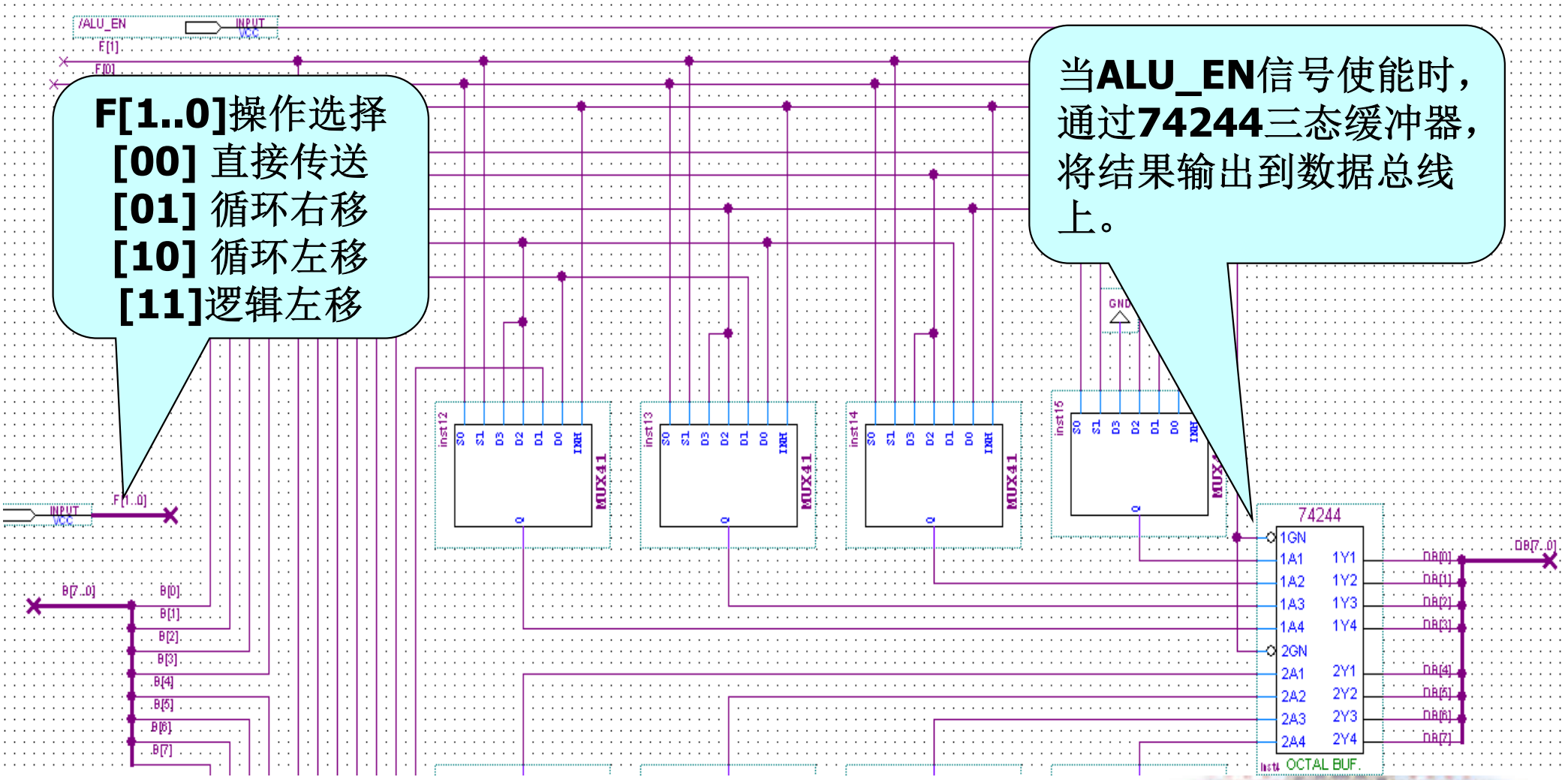
程序状态电路实现


B0N
A1N
B1N
A2N
B2N
A3N
B3N
CN
M
S0
S1
S2
S3
GN
PN
F0N
F1N
F2N
F3N
AEQB
CN4
ALU
B[0]
B[1]
B[2]
B[3]
B[4]
B[5]
B[6]
B[7]
B[7..0]
OV 溢出标志
inst8
74181d
NOR8
inst6
4DFF
DF0
DF1
DF2
DF3
/SET_F
/CLR_F
CLK_F
DF4
DF5
DF6
DF7
inst9
CMN
/PSW_EN
INPUT
VCC
GND
74244
1GN
1A1
1A2
1A3
1A4
2GN
2A1
2A2
2A3
2A4
1Y1
1Y2
1Y3
1Y4
2Y1
2Y2
2Y3
2Y4
OCTAL BUF
DB[7..0]
OUTPUT
AC
CY
ZN
OV

④ 移位寄存器设计

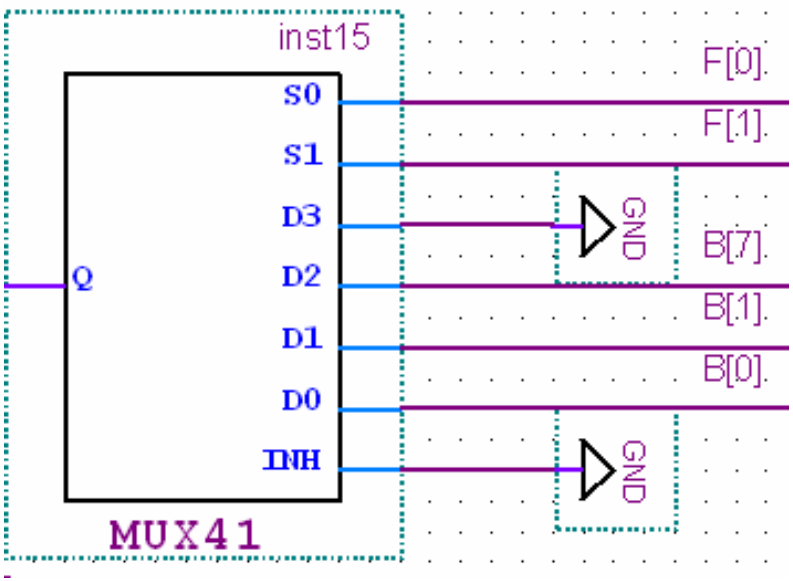
- 逻辑左移
- 逻辑右移
- 循环逻辑左移
- 循环逻辑右移

移位寄存器电路实现


F[1..0]操作选择
[00] 直接传送
[01] 循环右移
[10] 循环左移
[11]逻辑左移
当ALU_EN信号使能时，通过74244三态缓冲器，将结果输出到数据总线上。
/ALU_EN
INPUT
F[1..0]
B[7..0]
MUX41
74244
OCTAL BUF.
DB[7..0]

四选一选择电路

RL　R0；循环左移



循环左移运算实现过程

- RL R0;
  指令编码为011 11000，即78H
- 取指操作

微操作	控制信号	功能说明
PC-->ADDR[11..0]	M_ROM; /ROM_EN	ROM片选信号有效， PC指向程序入口地址
BUS-->IR; PC + 1 -->PC	LDIR1; M_PC	IR使能，微指令通过总线传送到IR，PC+1。

循环左移运算实现过程

- 取操作数

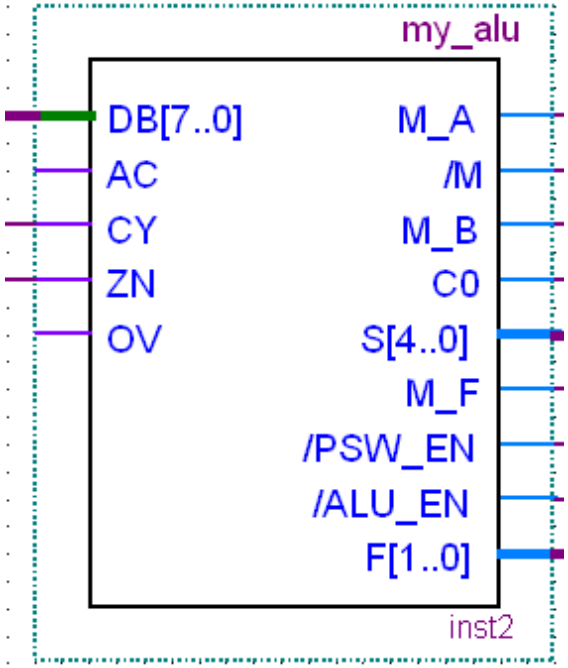
微操作	控制信号	功能说明
R0-->BUS	RDRi， /Ri_EN	寄存器读使能，读信号有效，R0数据送到数据总线
BUS-->A	M_A	暂存器A使能，数据从总线输入到暂存器A

循环左移运算实现过程

- 执行指令

微操作	控制信号	功能说明
A-->ALU，	M=0；Cn=1； S3…S0=0000	暂存器数据送到ALU，直接输出到移位寄存器
ALU-->BUS	F1F0=10； /ALU_EN=0	循环左移运算，ALU输出使能，ALU运算结果输出到总线
BUS-->R0	M_Rn; WRRi	寄存器使能，写信号有效，数据通过总线写入寄存器R0

ALU模块



模块	信号	说明
ALU	/M	时钟控制信号
	M_A	暂存器 A 的控制信号。
	M_B	暂存器 B 的控制信号。
	M_F	程序状态字的控制信号。
	S[4..0]	ALU 的运算方式选择信号，电平有效。
	F[1..0]	ALU 的运算结果输出方式选择信号，电平有效。 [00]为直传；[01]为右移；[10]为左移；[11]为取反。
	/ALU_EN	ALU运算结果输出到总线的使能信号,低电平有效。
	/PSW_EN	程序状态字输出到总线的使能信号，低电平有效。
	AC	ALU 输出的半进位标志位，高电平表示有进位。
	CY	ALU 输出的进位标志位，高电平表示有进位。
	ZN	ALU 运算结果 0 标志位，高电平表示结果为零。
	OV	ALU 运算结果溢出标志位，高电平表示结果溢出。

（2）基于VHDL语言的ALU设计

```
entity module_ALU is
        port(
                        clk_ALU         :in std_logic;      --ALU时钟信号
                        nreset          :in std_logic;      --全局复位信号
                        M_A,M_B         :in std_logic;      --暂存器控制信号
                        M_F             :in std_logic;      --程序状态字控制信号
                        nALU_EN         :in std_logic;      --ALU运算结果输出使能
                        nPSW_EN         :in std_logic;      --PSW输出使能
                        C0              :in std_logic;      --进位输入
                        S:in std_logic_vector(4 downto 0);  --运算类型和操作选择
                        F_in:in std_logic_vector(1 downto 0); --移位功能选择
                        data:inout std_logic_vector(7 downto 0)    --数据总线
                        AC              :out std_logic;     --半进位标志
                        CY              :out std_logic;     --进位标志
                        ZN              :out std_logic;     --零标志
                        OV              :out std_logic      --溢出标志
                );
end module_ALU;
```

clk_ALU=nclk2

6.6.7 数据存储器RAM设计

- MOV Ri，direct；
- MOV direct，Ri；
- RAM功能分析
  - ➢ 数据存储功能
  - ➢ 数据读写操作

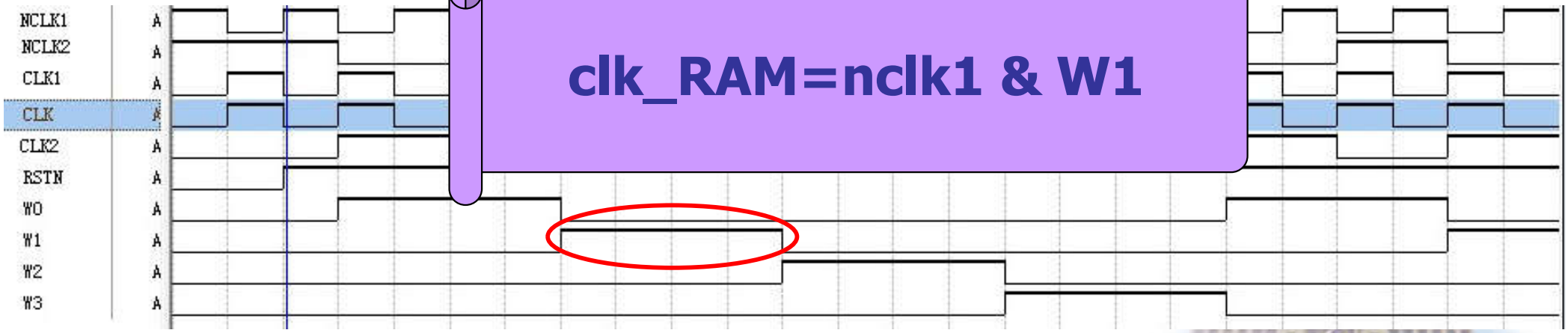
（1）RAM端口定义

```
entity module_ram is
  port(
        clk_RAM          :in std_logic;                --
        n_reset          :in std_logic;                --R
        RAM_CS           :in std_logic;                --RAM
        nRAM_EN           :in std_logic;               --RA
        wr_nRD           :in std_logic;                --
        AR               :in std_logic_vector(6 downto 0);
        data:inout std_logic_vector(7 downto 0)    --数据总线

     );
end module_ram;
```

高电平写操作有效，低电平读有效

MOV R0,[16H]



clk_RAM=nclk1 & W1

（2）RAM功能实现

- **读数据操作**

  clk_RAM上升沿有效，RAM_CS高电平，wr_nRD低电平，nRAM_EN低电平，[AR] → data 。

- **写数据操作**

  clk_RAM上升沿有效，RAM_CS高电平，wr_nRD高电平有效，data→[AR]。

6.6.8 堆栈指针SP设计

- MOV SP，#data；
- PUSH Ri;
- POP Ri;
- SP功能分析
  - 数据存储功能
  - 加1功能（**出栈**）
  - 减1功能（**压栈**）

（1）SP端口定义

```
entity module_sp is
  port(
       clk_SP       :in std_logic;--SP时钟信号
       nreset       :in std_logic;--复位信号
       SP_CS        :in std_logic;--SP选择信号
       SP_UP        :in std_logic;--SP+1控制
       SP_DN        :in std_logic;--SP-1控制信号
       nSP_EN        :in std_logic;--SP输出使能
       AR           :in std_logic_vector(6 downto 0);--SP指向的RAM地址
       data         :inout std_logic_vector(7 downto 0)--数据总线
     );
end module_sp;
```

clk_SP=nclk2

（2）SP功能实现分析

- **数据存储功能**

  clk_SP上升沿有效，SP_CS高电平，data→SP。

- **加1功能（出栈）**

  clk_SP上升沿有效，SP_CS高电平，SP_UP高电平，nSP_EN低电平有效，SP+1→SP,SP→AR。

- **减1功能（压栈）**

  clk_SP上升沿有效，SP_CS高电平，SP_DN高电平，nSP_EN低电平有效，SP-1→SP,SP→AR。

6.6.9 IO端口设计

- MOV P0，Ri；
- MOV Ri，P0；
- IO端口功能分析
- 输入锁存
- 输出锁存

（1）IO端口定义

```
entity module_P0 is
  port(
       clk_P0          :in std_logic;      --P0时钟信号
       nreset          :in std_logic;      --复位信号
       P0_CS           :in std_logic;      --P0选择信号
       nP0_IEN         :in std_logic;      --P0输入使能信号
       nP0_OEN         :in std_logic;      --P0输出使能信号
       P0_IN           :in std_logic_vector(7 downto 0);     --P0输入信号
       P0_OUT          :out std_logic_vector(7 downto 0);    --P0输出信号
       data            :inout std_logic_vector(7 downto 0)   --数据总线
      );
end module_P0;
```

clk_P0=nclk2

（2）IO功能实现分析

- **输入锁存**

clk_PO上升沿有效，P0_CS高电平，
nP0_IEN低电平，P0_IN→data。

- **输出锁存**

clk_PO上升沿有效，P0_CS高电平，
nP0_OEN低电平， data→P0_OUT。

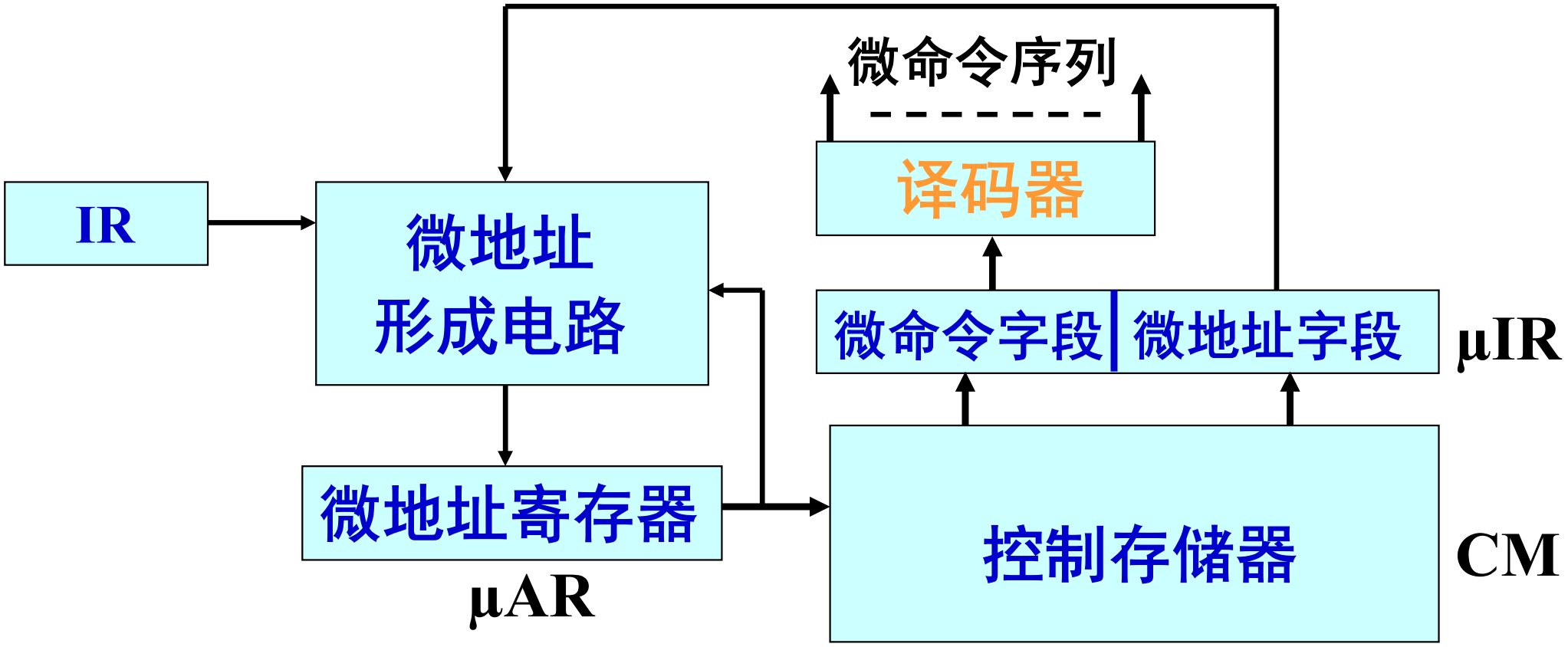
6.6.10 微程序控制器设计

- 微程序控制器基本原理
- 微程序控制器基本结构
- 微程序控制器设计方法
- 基于VHDL的微程序控制器设计

6.6.10.1 微程序控制器基本原理

- 将指令分解为基本的微命令序列，把操作控制信号编制成微指令，存放到控制存储器(CM)。
- 运行时，从控存中取出微指令，产生指令运行所需的操作控制信号。

6.6.10.2 微程序控制器基本结构



6.6.10.2 微程序控制器基本结构

- 控制存储器CM --存放微程序
- 微指令寄存器μIR --存放现行微指令
- 微地址形成电路--提供下一条微地址
- 微地址寄存器μAR--存放现在微地址

加法指令的微程序分析

- ADD R0, R1;
- 取指令

微操作	控制信号	功能说明
PC-->ADDR[11..0]	M_ROM; /ROM_EN	ROM片选信号有效，ROM读使能，PC指向程序入口地址
BUS-->IR; PC =PC + 1 ;	LDIR1; M_PC	IR使能，指令通过总线传送到IR，PC+1。
IR -->Microcontrol; addr[7:0]->CM[47:0]	M_uROM	微控制器使能，IR送入指令，生成控制信号。

加法指令的微程序分析

- 取操作数

微操作	控制信号	功能说明
R0-->BUS	RDRi， /Ri_EN	寄存器读使能，读信号有效，R0数据送到数据总线
BUS-->A	M_A	暂存器A使能，数据从总线输入到暂存器A
R1-->BUS	RDRi， /Ri_EN	寄存器读使能，读信号有效，R1数据送到数据总线
BUS-->B	M_B	暂存器B使能，数据从总线输入到暂存器B

加法指令的微程序分析

- **执行指令**

微操作	控制信号	功能说明
A-->ALU， B-->ALU	M=0； Cn=1； S3…S0=1001	暂存器数据送到ALU，选择不带进位算术加法运算
ALU-->BUS	/ALU_EN=0	ALU输出使能，ALU运算结果输出到数据总线
BUS-->R0	M_Rn; WRRi	寄存器使能，写信号有效，数据通过总线写入R0寄存器

（1）控制存储器CM

- 控制存储器与主存储器的区别

	控制存储器	主存储器
位置	CPU内	CPU外
器件	ROM	RAM和ROM
内容	微程序、微指令	程序、指令和数据

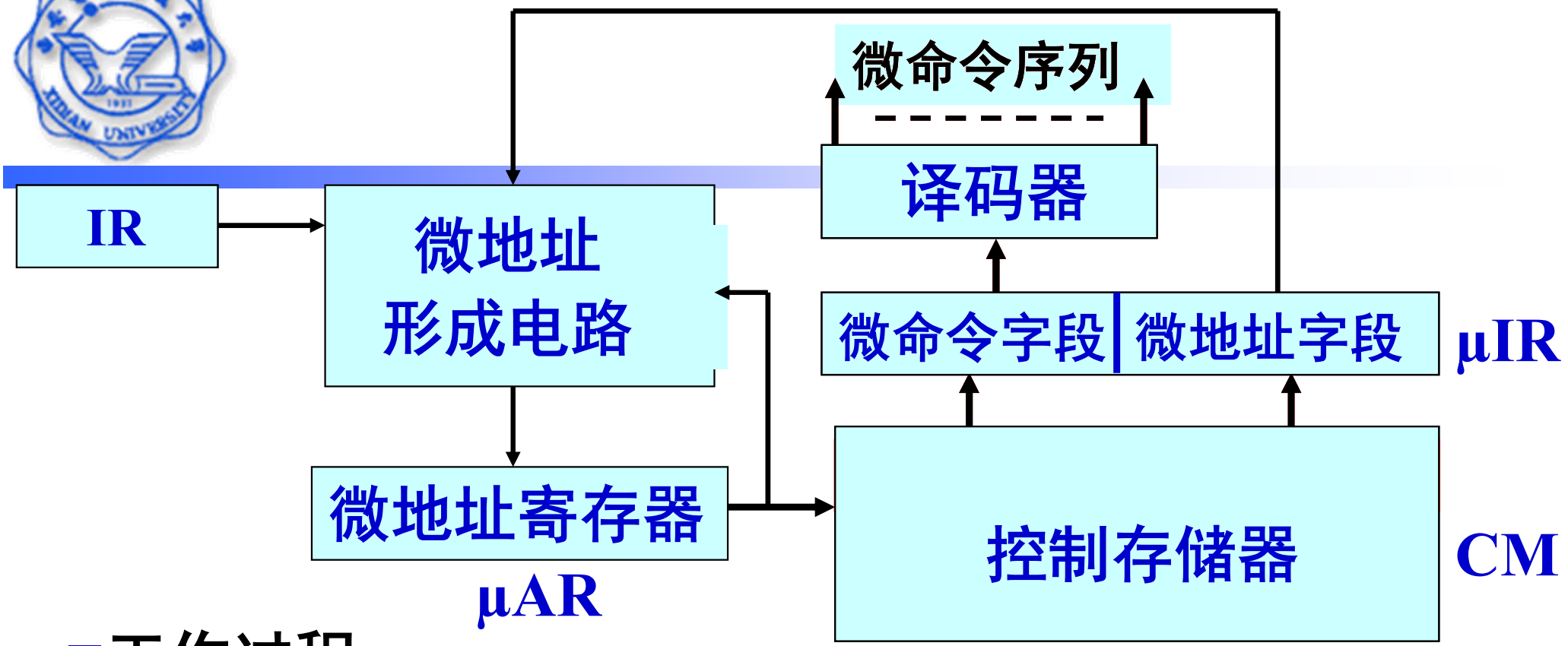
（2）微指令寄存器μIR

微命令字段 微地址字段

- **微命令（微操作）控制字段：提供当前操作所需的微命令。**
- **微地址（顺序控制）字段：**
  - **指明后续微地址的形成方式。**
  - **提供微地址的给定部分。**

（3）微地址形成电路

- **微程序入口地址：** **由机器指令操作码形成。**
- **后续微地址：** **由微地址字段、现行微地址等形成。**

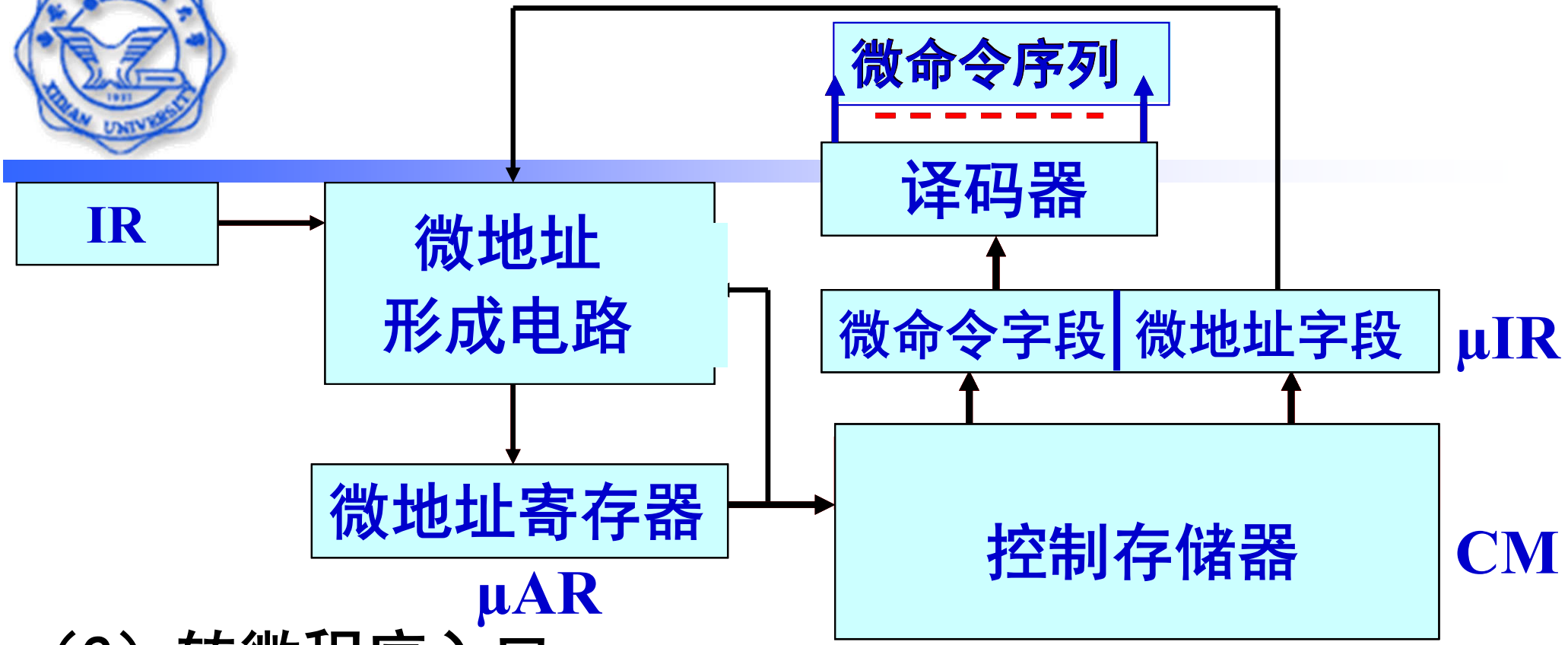


■工作过程

（1）取机器指令

CM —取指微指令→ μIR —微命令字段→ 译码器 —微命令→ 主存

IR ←机器指令—

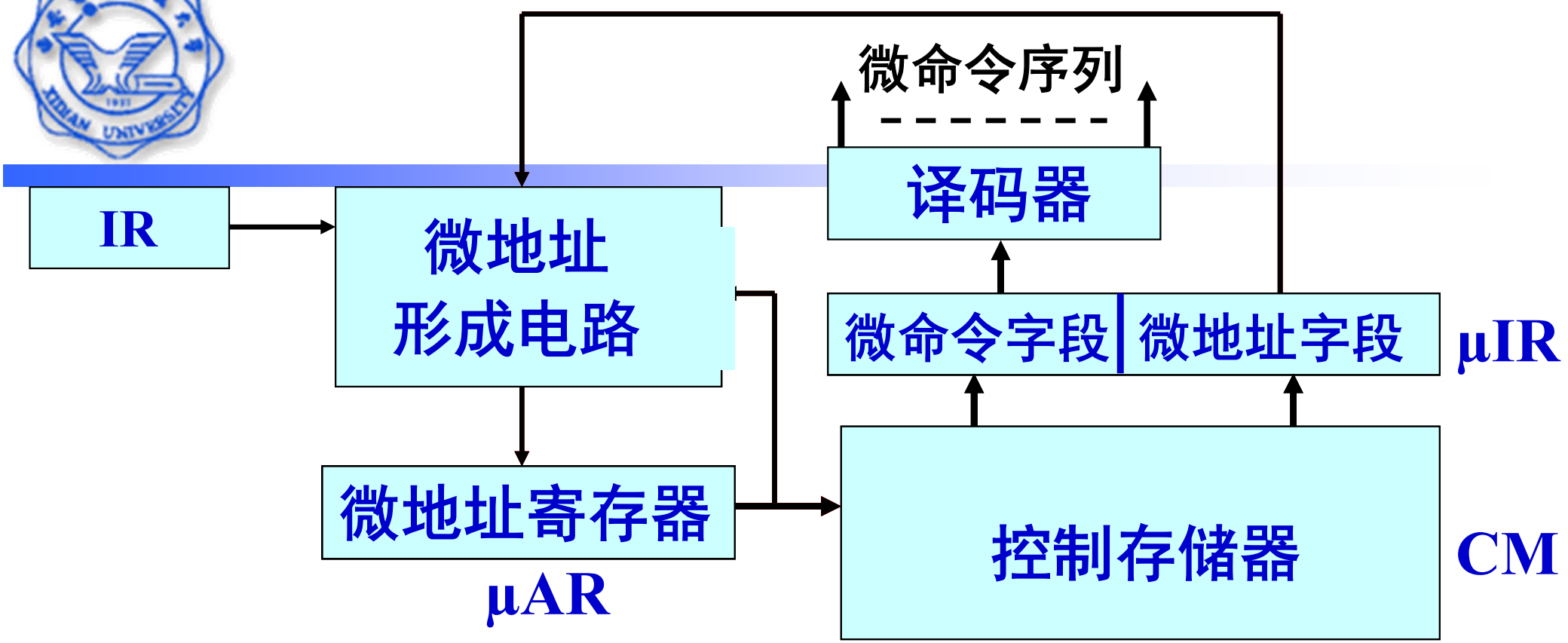


（2）转微程序入口

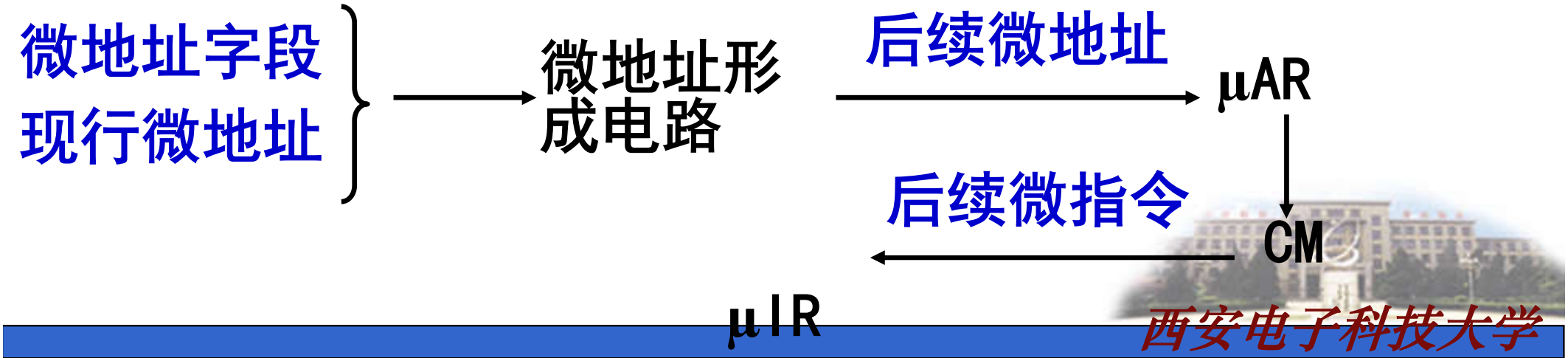
IR —操作码→ 微地址形成电路 —入口→ μAR → CM —首条微指令→ μIR

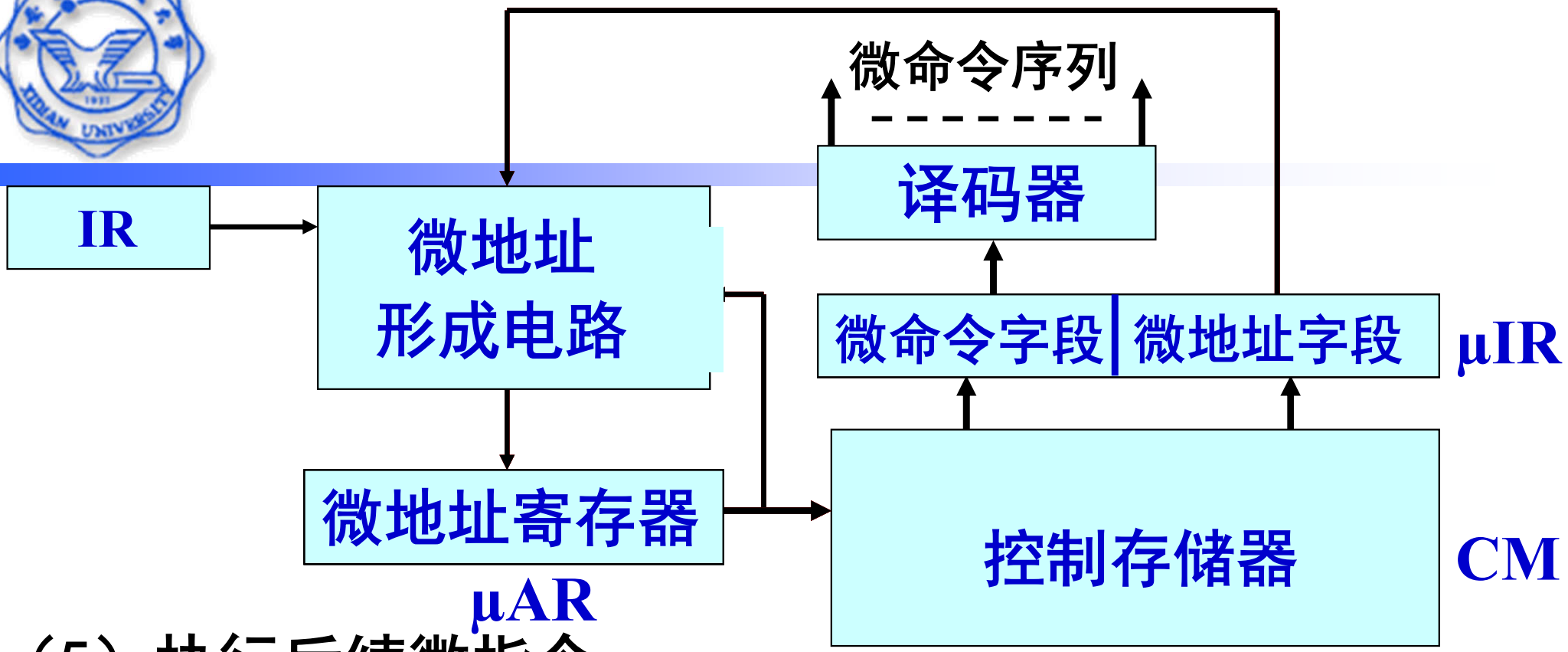
（3）执行首条微指令

μIR —微命令字段→ 译码器 —微命令→ 操作部件


微命令序列
译码器
IR
微地址
形成电路
微命令字段
微地址字段
μIR
微地址寄存器
μAR
控制存储器
CM

（4）取后续微指令


微地址字段
现行微地址
微地址形
成电路
后续微地址
μAR
后续微指令
CM
μIR



（5）执行后续微指令

同（3）

（6）返回

微程序执行完，返回CM（存放取指微指令的固定单元）。

6.6.10.3 微程序控制器设计方法

- 根据指令系统，列出微操作序列
- 微指令编码
- 控制微程序流
- 确定指令格式
- 微程序写入控制存储器
- 功能模块设计

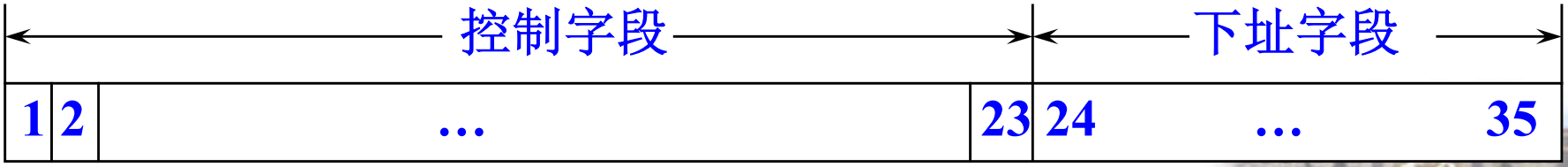
（1）指令的微操作分析

微操作	控制信号	功能说明
PC-->ADDR[11..0]	M_ROM; /ROM_EN	ROM片选信号有效，ROM读使能，PC指向程序入口地址
BUS-->IR; PC =PC + 1 ;	LDIR1; M_PC	IR使能，指令通过总线传送到IR，PC+1。
IR -->Microcontrol; addr[7:0]->CM[47:0]	M_uROM	微控制器使能，IR送入指令，生成控制信号。

（2）微指令编码

- 直接控制法
- 字段直接编译法
- 字段间接编译法

- 假设控存容量为4K,则需12位来表示下一个微指令地址。
- 控制存储器的容量由实现指令系统所需要的微程序长度决定。



① 直接控制编码

- 不译码法：微指令的控制字段中，每一位代表一个微命令。是否发出某个微命令，只要将控制字段中相应位置成“1”或“0”，就可以打开或关闭某个控制门。
- 优点：控制简单、 速度快、并行好 。
- 缺点：微指令字长度长，需要大容量存储空间。

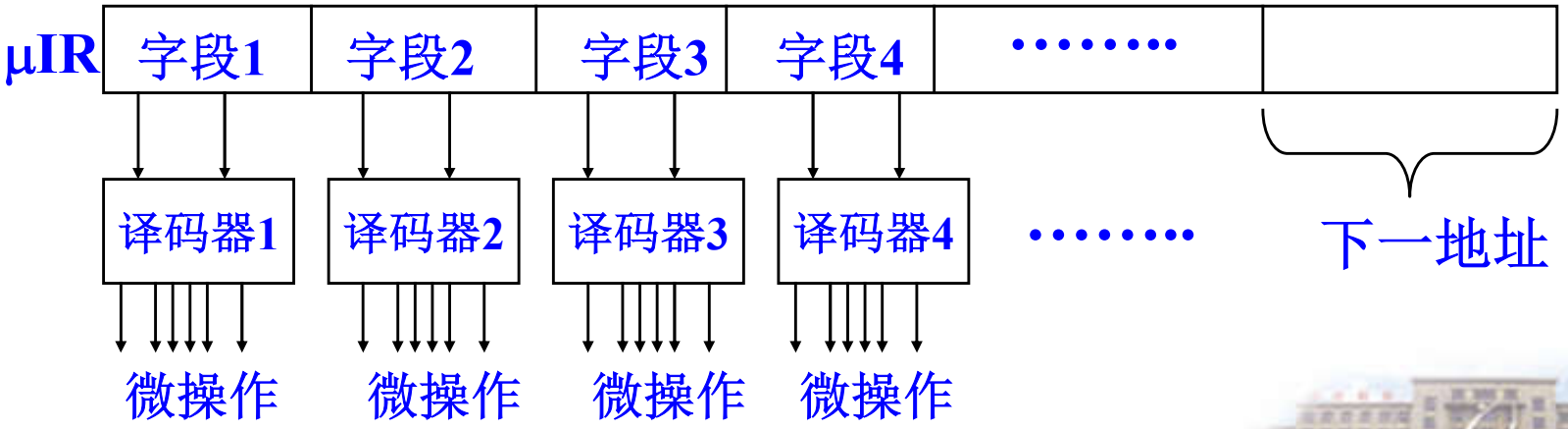
互斥与相容

- 互斥的微操作：是指不能同时或不能在同一个节拍内并行执行的微操作。
- 相容的微操作：是指能够同时或在同一个节拍内并行执行的微操作。

- 把互斥的微操作组合在同一字段中，采用编码方式存取。
- 把相容的微操作组合在不同字段中，各段单独译码。

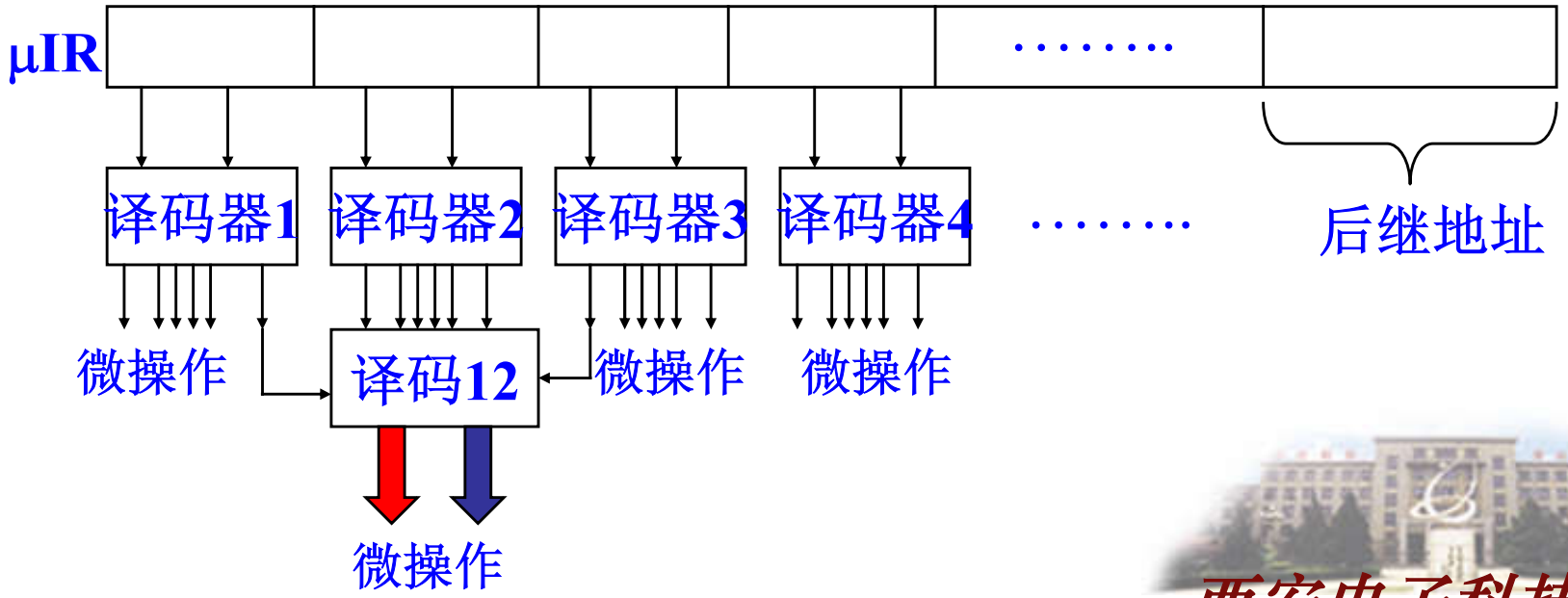
② 字段直接编译法

- 把互斥的微命令编成一组，用二进制编码表示，成为微指令字的一个字段。
- 在微指令寄存器的输出端，为该字段增加一个译码器。
- 优点：缩短了微指令长度。



③ 分段间接编译法

- 在字段直接编译法的基础上，进一步缩短微指令字长的一种编译法。
- 一个字段的某些微命令，要兼由另一些字段中的某些微命令来解释。
- 缺点：可能会削弱微指令的并行控制能力。



（3）控制微程序流

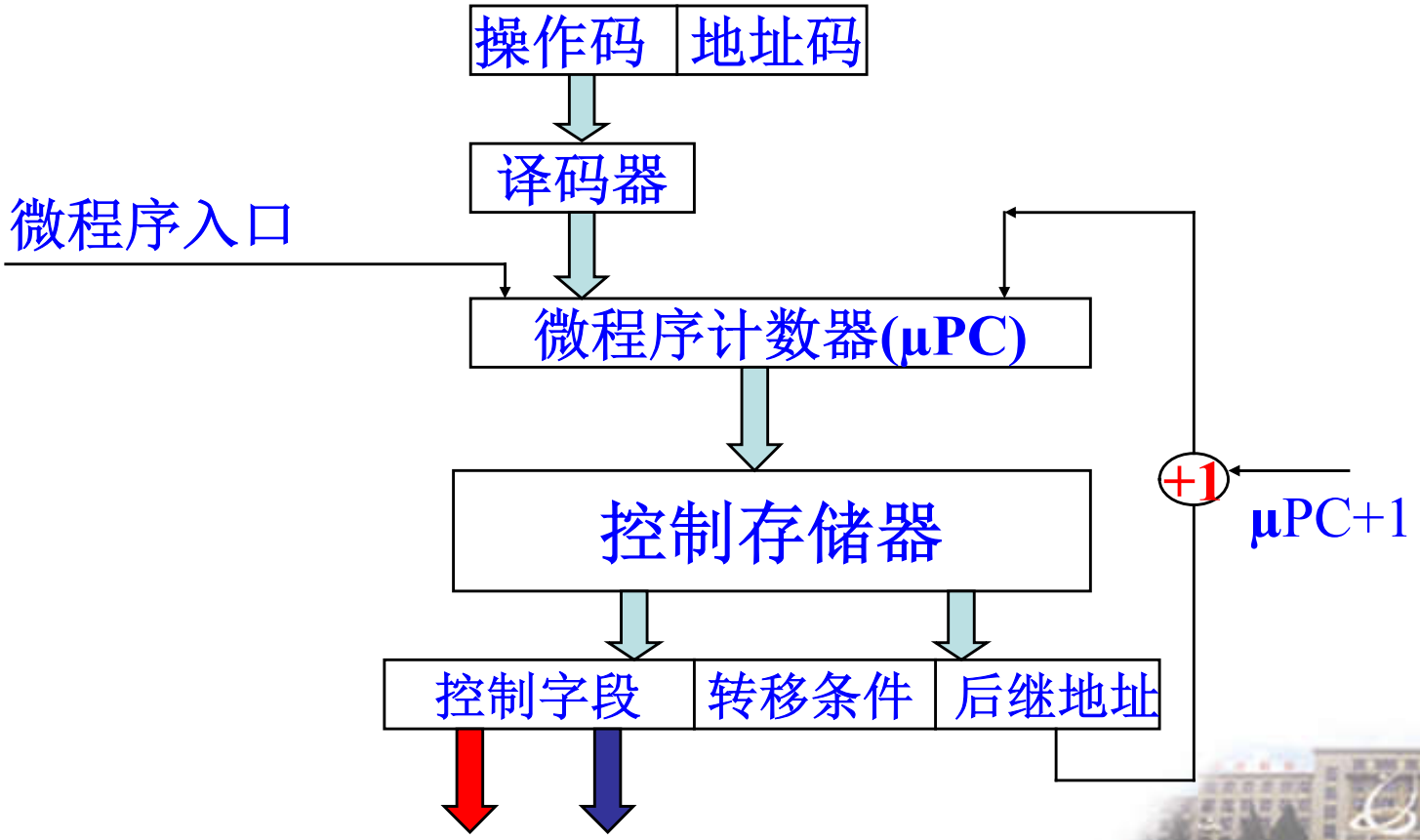
- 当前正在执行的微指令，称为现行微指令，现行微指令所在的控存单元的地址称为现行微地址。
- 现行微指令执行完毕后，下一条要执行的微指令称为后继微指令，后继微指令所在的控存单元地址称为后继微地址。

（3）控制微程序流

- 微程序流的控制：是指当前微指令执行完毕后,怎样控制产生后继微指令的微地址。
  - 由指令操作码译码器产生
  - 由微指令的下址字段指出

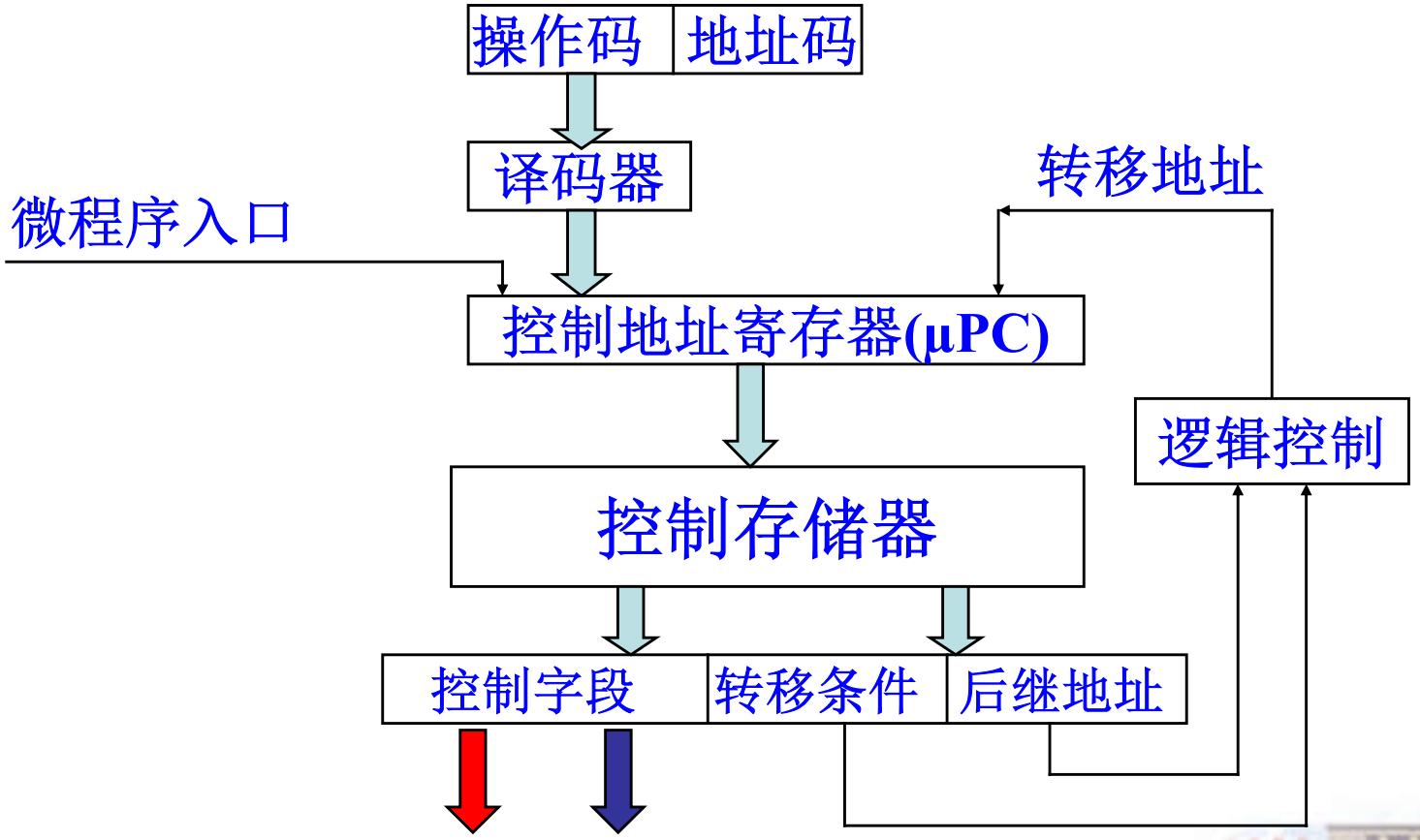
以增量方式产生后继地址

- 顺序执行微程序。
  后继微地址由现行微地址加上一个增量(通常为1)

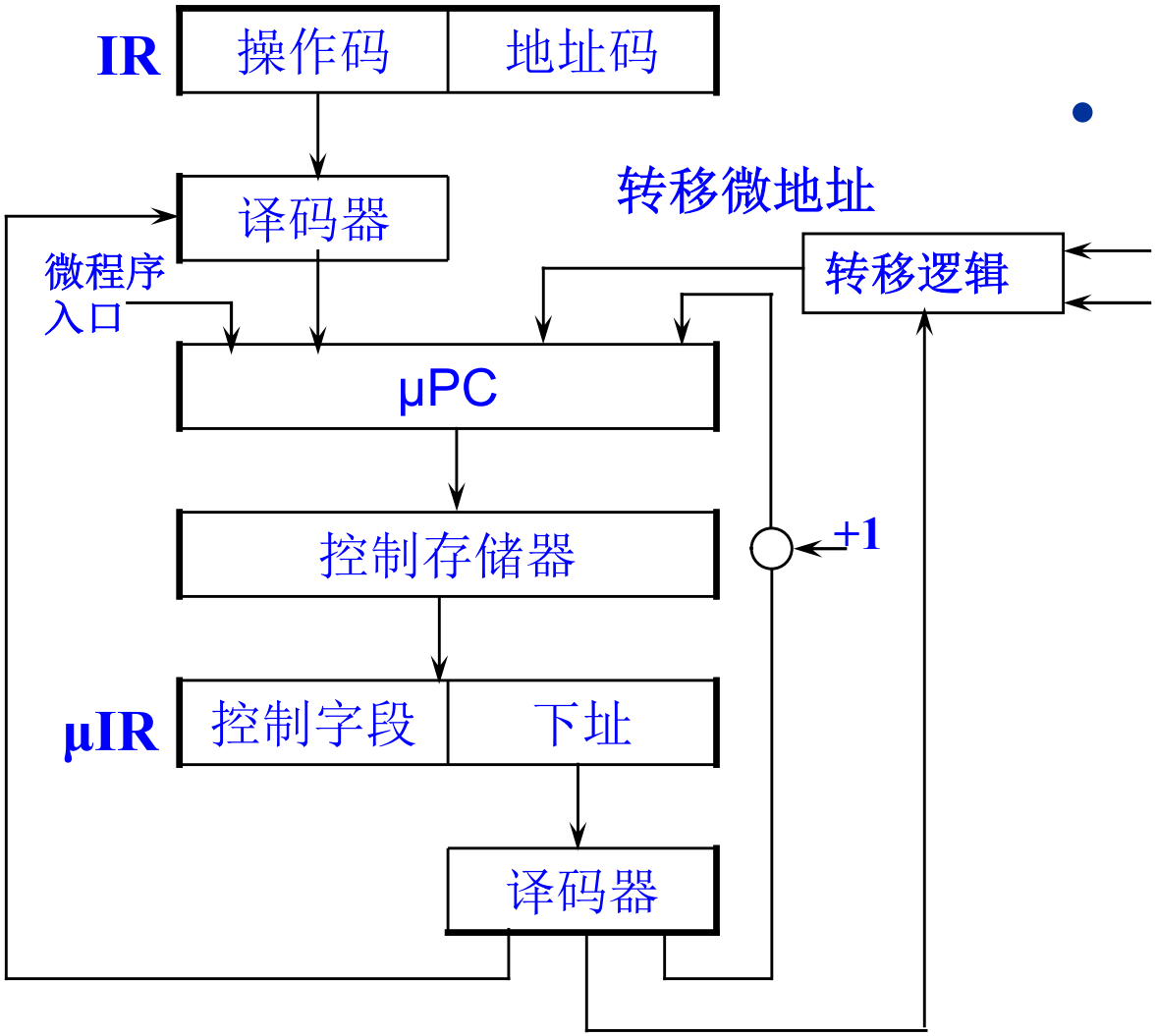


以增量方式产生后继地址

- 转移的控制



以增量方式产生后继地址



- “计数器”方式
  - 下址部分很短，只有两位，它选择三个输入源中的一个作为μPC的输入
  - ①(μPC)+1→μPC
  - ②转移微地址
  - ③操作码译码器的输出。

非顺序执行的下址

- 初始地址：控制存储器的0号或1号单元
  - 开机时，微地址形成部件复位；
  - 开机后，执行后续指令时，由现行微程序的最后一条微指令给出。
- 转移地址：由微指令给出。
- 微中断地址：入口地址是固定的，由硬件直接赋值给微地址形成部件（微中断信号由程序的中断引起）。

（4）微指令格式

- 垂直型微指令
  - 一条微指令定义并执行一种基本操作
  - 优点：微指令短、简单、规整、便于编写微程序
  - 缺点：微程序长，执行速度慢，工作效率低
- 水平型微指令
  - 一条微指令定义并执行几种并行的基本操作
  - 优点：微程序短、执行速度快
  - 缺点：微指令长，编写微程序较麻烦

水平微指令与垂直微指令比较

	水平微指令	垂直微指令
能力 效率 灵活性	高 强	低 弱
速度 执行时间	快 短	慢 长
字长	长	短
掌握难度	难	容易

（5）微指令设计思路

- 采用直接控制编码，增量方式产生后继地址，水平型微指令。
- 所有模块控制信号

控制信号汇总

PC模块（4条）

- LD_PC:in std_logic; --装载新地址
- M_PC:in std_logic; --PC加1控制信号
- nPCH,nPCL::in std_logic; --PC输出总线控制信号

ROM模块（2条）

- M_ROM :in std_logic; --ROM片选信号
- ROM_EN :in std_logic; --ROM使能信号

控制信号汇总

IR模块（4条）

- LD_IR1,LD_IR2,LD_IR3 :in std_logic;--IR指令存储控制信号
- nARen :in std_logic; --IR中RAM地址控制信号

RN模块（4条）

- Ri_CS :in std_logic; --RN选择信号
- Ri_EN :in std_logic; --RN寄存器使能
- RDRi,WRRi :in std_logic; --RN读写信号

控制信号汇总

ALU模块（13条）

- M_A,M_B :in std_logic; --暂存器控制信号
- M_F :in std_logic; --程序状态字控制信号
- nALU_EN :in std_logic; --ALU运算结果输出使能
- nPSW_EN :in std_logic; --PSW输出使能
- C0 :in std_logic; --进位输入
- S:in std_logic_vector(4 downto 0); --运算类型和操作选择
- F_in:in std_logic_vector(1 downto 0); --移位功能选择

RAM模块（3条）

- RAM_CS :in std_logic; --RAM片选信号
- nRAM_EN :in std_logic; --RAM输出使能信号
- wr_nRD :in std_logic; -- 读写信号

控制信号汇总

SP模块（4条）

- SP_CS :in std_logic;--SP选择信号
- SP_UP :in std_logic;--SP+1控制
- SP_DN :in std_logic;--SP-1控制信号
- nSP_EN :in std_logic;--SP输出使能

P0模块（3条）

- P0_CS :in std_logic; --P0选择信号
- nP0_IEN :in std_logic; --P0输入使能信号
- nP0_OEN :in std_logic; --P0输出使能信号

CM模块（2条）

- M_uA :IN std_logic; --微地址控制信号
- CMROM_CS :IN std_logic; --控制存储器选通信号

（6）控制信号设计

- 39条控制信号（39位编码）
- 27条指令（5位编码）→8位微地址

指令助记符	位	入口地址	47 46 45 44	43 42 41 40	39 38 37 36	35 34 33 32	31 30 29 28	27 26 25 24	23 22 21 20	19 18 17 16	15 14 13 12	11 10 9 8	7 6 5 4	3 2 1 0	微指令码 (16 进制)
	信号		M_A M_B M_F S4	S3S2S1S0	F1 F0 nALU_EN nPSW_EN	C0 RAM_CS Wr_nRD nRAM_EN	Rn_CS RDRi WRRi nRi_EN	LDIR1 LDIR2 LDIR3 nAREN	M_PC nLD_PC nPCH nPCL	SP_UP SP_DN SP_CS nSP_EN	P_CS nP_IEN nP_OEN X	M_ROM nROM_EN M_uA CMROM_CS	u7 u6 u5 u4	u3 u2 u1 u0	

① 取指公操作

- 指令集中每条指令的执行都先进行取指操作，独立出来作为公共微操作。
- 取指公操作
- 微程序入口地址：00H

指令助记符	位	入口地址	47 46 45 44	43 42 41 40	39 38 37 36	35 34 33 32	31 30 29 28	27 26 25 24	23 22 21 20
	信号		M_A M_B M_F S4	S3S2S1S0	F1 F0 nALU_EN nPSW_EN	C0 RAM_CS Wr_nRD nRAM_EN	Rn_CS RDRi WRRi nRi_EN	LDIR1 LDIR2 LDIR3 nAREN	M_PC nLD_PC nPCH nPCL
取指公操作		00H	0000	0000	0011	0001	0001	1001	1111

19 18 17 16	15 14 13 12	11 10 9 8	7 6 5 4	3 2 1 0	微指令码 (16 进制)
SP_UP SP_DN SP_CS nSP_EN	P_CS nP_IEN nP_OEN X	M_ROM nROM_EN M_uA CMROM_CS	u7 u6 u5 u4	u3 u2 u1 u0	
0011	0111	1000	0	0	003119F37800

② 微指令码分析

指令助记符	位	入口地址	47 46 45 44	43 42 41 40	39 38 37 36	35 34 33 32	31 30 29 28	27 26 25 24	23 22 21 20
	信号		M_A M_B M_F S4	S3S2S1S0	F1 F0 nALU_EN nPSW_EN	C0 RAM_CS Wr_nRD nRAM_EN	Rn_CS RDRi WRRi nRi_EN	LDIR1 LDIR2 LDIR3 nAREN	M_PC nLD_PC nPCH nPCL
取指公操作		00H	0000	0000	0011	0001	0001	1001	1111

19 18 17 16	15 14 13 12	11 10 9 8	7 6 5 4	3 2 1 0	微指令码 (16 进制)
SP_UP SP_DN SP_CS nSP_EN	P_CS nP_IEN nP_OEN X	M_ROM nROM_EN M_uA CMROM_CS	u7 u6 u5 u4	u3 u2 u1 u0	
0011	0111	1000	0	0	003119F37800

微操作	控制信号	功能说明
PC-->ADDR[11..0]	M_ROM; /ROM_EN	ROM片选信号有效，ROM读使能，PC指向程序入口地址

② 微指令码分析

指令助记符	位	入口地址	47 46 45 44	43 42 41 40	39 38 37 36	35 34 33 32	31 30 29 28	27 26 25 24	23 22 21 20
	信号		M_A M_B M_F S4	S3S2S1S0	F1 F0 nALU_EN nPSW_EN	C0 RAM_CS Wr_nRD nRAM_EN	Rn_CS RDRi WRRi nRi_EN	LDIR1 LDIR2 LDIR3 nAREN	M_PC nLD_PC nPCH nPCL
取指公操作		00H	0000	0000	0011	0001	0001	1001	1111

19 18 17 16	15 14 13 12	11 10 9 8	7 6 5 4	3 2 1 0	微指令码 (16 进制)
SP_UP SP_DN SP_CS nSP_EN	P_CS nP_IEN nP_OEN X	M_ROM nROM_EN M_uA CMROM_CS	u7 u6 u5 u4	u3 u2 u1 u0	
0011	0111	1000	0	0	003119F37800

BUS-->IR;PC =PC + 1 ; IR -->Microcontrol;	LDIR1; M_PC	IR使能，指令通过总线传送到IR，PC+1。

③ 微程序设计

- MOV Ri，#data
- 微程序入口地址：24H

指令助记符	位	入口地址	47 46 45 44	43 42 41 40	39 38 37 36	35 34 33 32	31 30 29 28	27 26 25 24	23 22 21 20
	信号		M_A M_B M_F S4	S3S2S1S0	F1 F0 nALU_EN nPSW_EN	C0 RAM_CS Wr_nRD nRAM_EN	Rn_CS RDRi WRRi nRi_EN	LDIR1 LDIR2 LDIR3 nAREN	M_PC nLD_PC nPCH nPCL
MOV Ri，#data		24H	0000	0000	0011	0001	0011	0001	1111
		25H	0000	0000	0011	0001	0001	0001	0111
		26H	0000	0000	0011	0001	0001	0001	0111
		27H							

MOV SP，#	19 18 17 16	15 14 13 12	11 10 9 8	7 6 5 4	3 2 1 0	
	SP_UP SP_DN SP_CS nSP_EN	P_CS nP_IEN nP_OEN X	M_ROM nROM_EN M_uA CMROM_CS	u7 u6 u5 u4	u3 u2 u1 u0	微指令码 (16 进制)
	0011	0111	1011	2	5	003131F37B25
	0011	0111	0111	2	6	003111737726
	0011	0111	0111	0	0	003111737700

③ 微程序设计

- SUBC Ri，Rj；
- 微程序入口地址: 74H

指令助记符	位	入口地址	47 46 45 44	43 42 41 40	39 38 37 36	35 34 33 32	31 30 29 28	27 26 25 24	23 22 21 20
	信号		M_A M_B M_F S4	S3S2S1S0	F1 F0 nALU_EN nPSW_EN	C0 RAM_CS Wr_nRD nRAM_EN	Rn_CS RDRi WRRi nRi_EN	LDIR1 LDIR2 LDIR3 nAREN	M_PC nLD_PC nPCH nPCL
SUB Ri，Rj		74H	0100	0000	0011	0001	1100	0001	0111
		75H	1000	0000	0011	0001	1100	0001	0111
		76H	0000	0110	0001	1001	0011	0001	1111
		77H							

19 18 17 16	15 14 13 12	11 10 9 8	7 6 5 4	3 2 1 0	微指令码 (16 进制)
SP_UP SP_DN SP_CS nSP_EN	P_CS nP_IEN nP_OEN X	M_ROM nROM_EN M_uA CMROM_CS	u7 u6 u5 u4	u3 u2 u1 u0	
0011	0111	0111	7	5	4031C1737775
0011	0111	0111	7	6	8031C1737776
0011	0111	0111	0	0	031931F37700

③ 微程序设计

- SUB Ri，Rj；
- 微程序入口地址: 74H

指令助记符	位 / 信号	入口地址	47 46 45 44 M_A M_B M_F S4	43 42 41 40 S3S2S1S0	39 38 37 36 F1 F0 nALU_EN nPSW_EN	35 34 33 32 C0 RAM_CS Wr_nRD nRAM_EN	31 30 29 28 Rn_CS RDRi WRRi nRi_EN	27 26 25 24 LDIR1 LDIR2 LDIR3 nAREN	23 22 21 20 M_PC nLD_PC nPCH nPCL
SUB Ri，Rj		74H	0100	0000	0011	0001	1100	0001	0111
		75H	1000	0000	0011	0001	1100	0001	0111
		76H	0000	0110	0001	1001	0011	0001	1111
		77H							

19 18 17 16 SP_UP SP_DN SP_CS nSP_EN	15 14 13 12 P_CS nP_IEN nP_OEN X	11 10 9 8 M_ROM nROM_EN M_uA CMROM_CS	7 6 5 4 u7 u6 u5 u4	3 2 1 0 u3 u2 u1 u0	微指令码 (16 进制)
0011	0111	0111	7	5	4031C1737775
0011	0111	0111	7	6	8031C1737776
0011	0111	0111	0	0	031931F37700

③ 微程序设计

- SUB Ri，Rj；
- 微程序入口地址: 74H

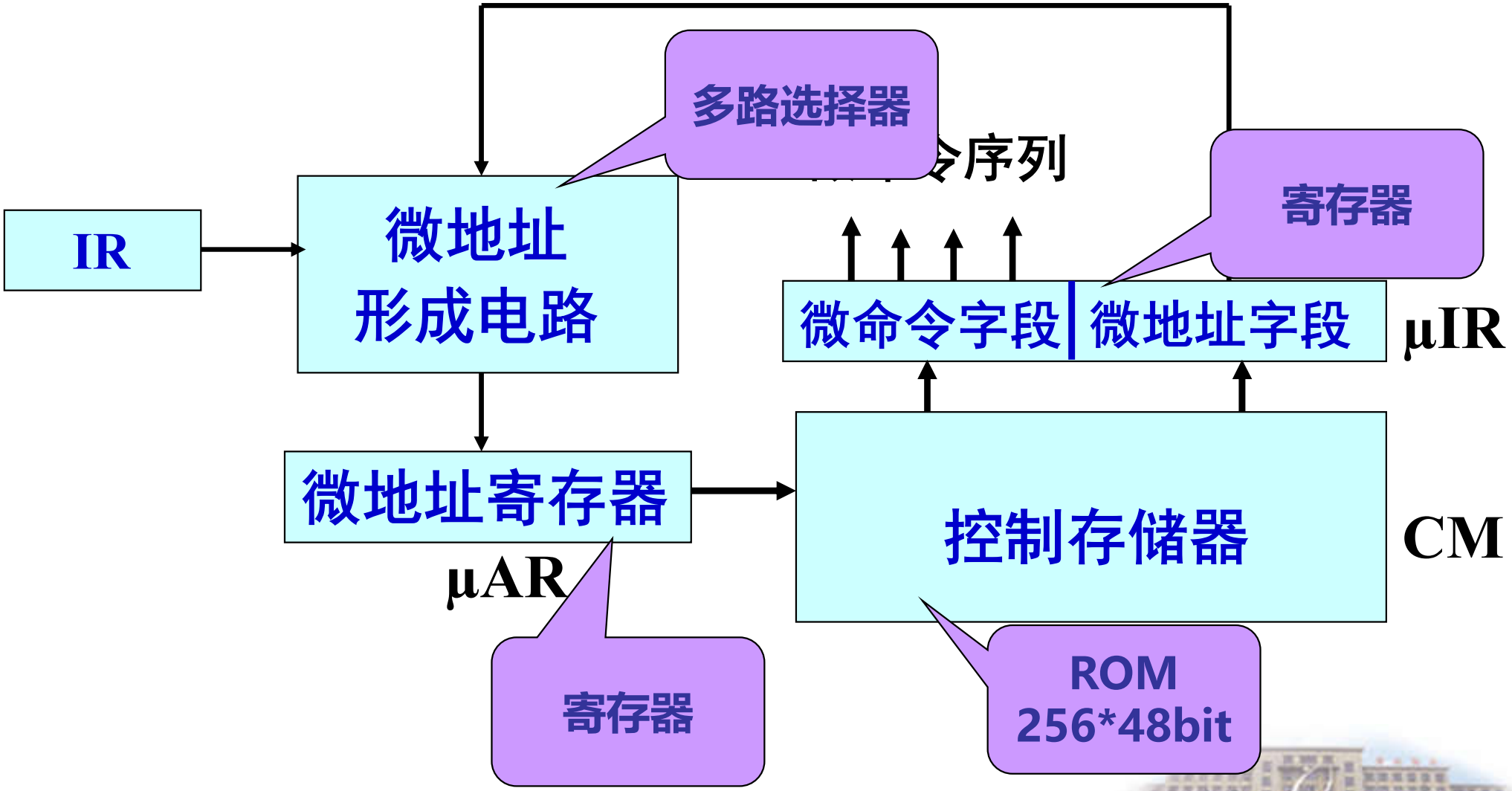
指令助记符	位 / 信号	入口地址	47 46 45 44 M_A M_B M_F S4	43 42 41 40 S3S2S1S0	39 38 37 36 F1 F0 nALU_EN nPSW_EN	35 34 33 32 C0 RAM_CS Wr_nRD nRAM_EN	31 30 29 28 Rn_CS RDRi WRRi nRi_EN	27 26 25 24 LDIR1 LDIR2 LDIR3 nAREN	23 22 21 20 M_PC nLD_PC nPCH nPCL
SUB Ri , Rj		74H	0100	0000	0011	0001	1100	0001	0111
		75H	1000	0000	0011	0001	1100	0001	0111
		76H	0000	0110	0001	1001	1011	0001	1111
		77H							

19 18 17 16 SP_UP SP_DN SP_CS nSP_EN	15 14 13 12 P_CS nP_IEN nP_OEN X	11 10 9 8 M_ROM nROM_EN M_uA CMROM_CS	7 6 5 4 u7 u6 u5 u4	3 2 1 0 u3 u2 u1 u0	微指令码 (16 进制)
0011	0111	0111	7	5	4031C1737775
0011	0111	0111	7	6	8031C1737776
0011	0111	0111	0	0	031931F37700

微程序控制器设计

- 控制存储器CM --存放微程序
- 微指令寄存器μIR --存放现行微指令
- 微地址形成电路--提供微地址
- 微地址寄存器μAR--存放现在微地址

微程序控制器设计


多路选择器
令序列
寄存器
IR
微地址
形成电路
微命令字段
微地址字段
μIR
微地址寄存器
μAR
控制存储器
CM
寄存器
ROM
256*48bit

6.6.10.4微程序控制器VHDL描述

```
entity micro_controller is
        port(
                clk_MC              :IN std_logic;              --微程序控制器时钟信号
                nreset              :in std_logic;              --复位信号
                IR                  :IN std_logic_vector(7 downto 2);--IR操作码信息
                M_uA                :IN std_logic;              --微地址控制信号
                CMROM_CS            :IN std_logic;              --控制存储器选通信号
                CM:out std_logic_vector(47 downto 0)            --控制信号输出
                 );
end micro_controller;
```