第5章 中央处理部件CPU

5.1 CPU的组成

5.2 控制器的组成

5.3 组合逻辑（硬布线）控制器

5.4 微程序控制器

5.5 微程序设计技术

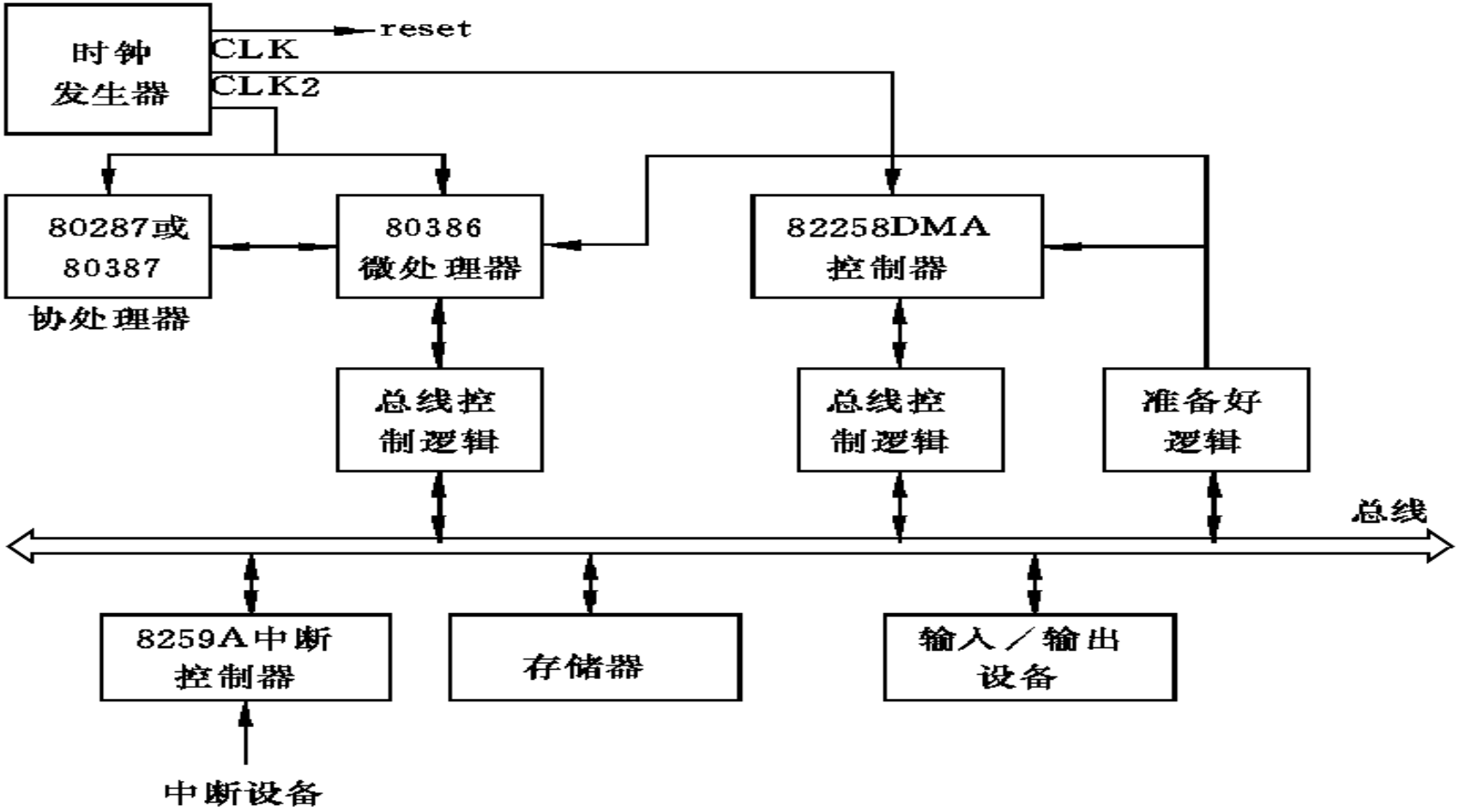
5.6 CPU内部数据通路结构及指令执行

5.7 流水线工作原理

主要知识点

- 从CPU的功能，了解控制器功能和结构
- 掌握控制器的主要组成部件
- 掌握组合逻辑（硬布线）控制的特点及工作原理
- 掌握微操作的概念
- 掌握微程序控制器的实现原理及工作原理
- 掌握控制器在不同的数据通路下，如何控制一条指令的执行
- 流水线工作原理

5.1 CPU的组成


时钟
发生器
reset
CLK
CLK2
80287或
80387
协处理器
80386
微处理器
82258DMA
控制器
总线控
制逻辑
总线控
制逻辑
准备好
逻辑
总线
8259A中断
控制器
存储器
输入／输出
设备
中断设备

5.1 CPU的组成

1.运算部件ALU

2.寄存器

（1）用于处理的寄存器

- 通用寄存组
- 暂存器

（2）用于控制的寄存器

5.1 CPU的组成

（3）用于主存和I/O接口的寄存器

- 地址寄存器AR
- 数据寄存器DR

3.总线

4.时序系统及微操作命令产生部件（控制器）

5.2 控制器的组成

5.2.1控制器的功能

计算机对信息进行处理(或计算)是通过程序的执行而实现的，程序是完成某个确定算法的指令序列，要预先存放在存储器中。所以，计算机的工作过程是程序的运行过程，也就是在控制器下逐条执行指令的过程。

控制器的基本功能：就是控制机器各部件执行指令

5.2 控制器的组成

1.取指令

当程序已在存储器中时，首先由控制器根据程序入口取出第一条指令，为此要发出指令地址及控制信号，把指令从内存取到CPU中。

2.分析指令

或叫解释指令、指令译码等。是对当前取得的指令进行分析，指出它要求作什么操作，并产生相应的操作控制命令，如果参与操作的数据在存储器中，还需要形成操作数地址。

5.2 控制器的组成

3.执行指令

根据分析指令时产生的“操作命令”和“操作数地址”形成相应的操作控制信号序列，通过CPU及输入输出设备的执行，实现每条指令的功能，其中还包括对运算结果的处理以及下条指令地址的形成。

5.2 控制器的组成

计算机不断重复顺序执行上述三种基本操作：取指、分析、执行；再取指、再分析、再执行……，如此循环，直到遇到停机指令或外来的干预为止。

此外，程序和数据要输入机器，运算结果要输出，机器运行过程中出现的某些异常情况或请求要进行处理，人与机器之间要进行对话，因此控制器还应该具有以下功能：

4. 控制程序和数据的输入与结果输出

根据程序的安排或人的干预，在适当的时候向输入输出设备发出一些相应的命令来完成I/O功能，这实际上也是通过执行程序来完成的。

5. 对异常情况和某些请求的处理

当机器出现某些异常情况，此时由这些部件或设备发出“中断请求”信号，当需要DMA传送时发DMA请求信号：

(1) **“中断请求”信号，**待CPU执行完当前指令后，响应该请求，中止当前执行的程序，转去执行中断程序。当处理完毕后，再返回原程序继续运行下去。

(2) **DMA请求信号，**等CPU完成当前机器周期操作后，暂停工作，让出总线给I/O设备，在完成I/O设备与存储器之间的传送数据操作后，CPU从暂时中止的机器周期开始继续执行指令。DMA操作不允许改变CPU中任一寄存器状态(除DMA专用部件外)，否则会影响CPU工作的正确性。

5.2.2 控制器的组成

1. 程序计数器(PC)

即指令地址寄存器。用来存放即将要执行的下一条指令地址。

有两种方法形成下一条指令地址：

（1）程序在顺序执行时，通过对PC的增值形成下一条指令地址。

（2）程序在非顺序执行时，一般由转移指令形成下一条指令地址对PC进行修改。

5.2 控制器的组成

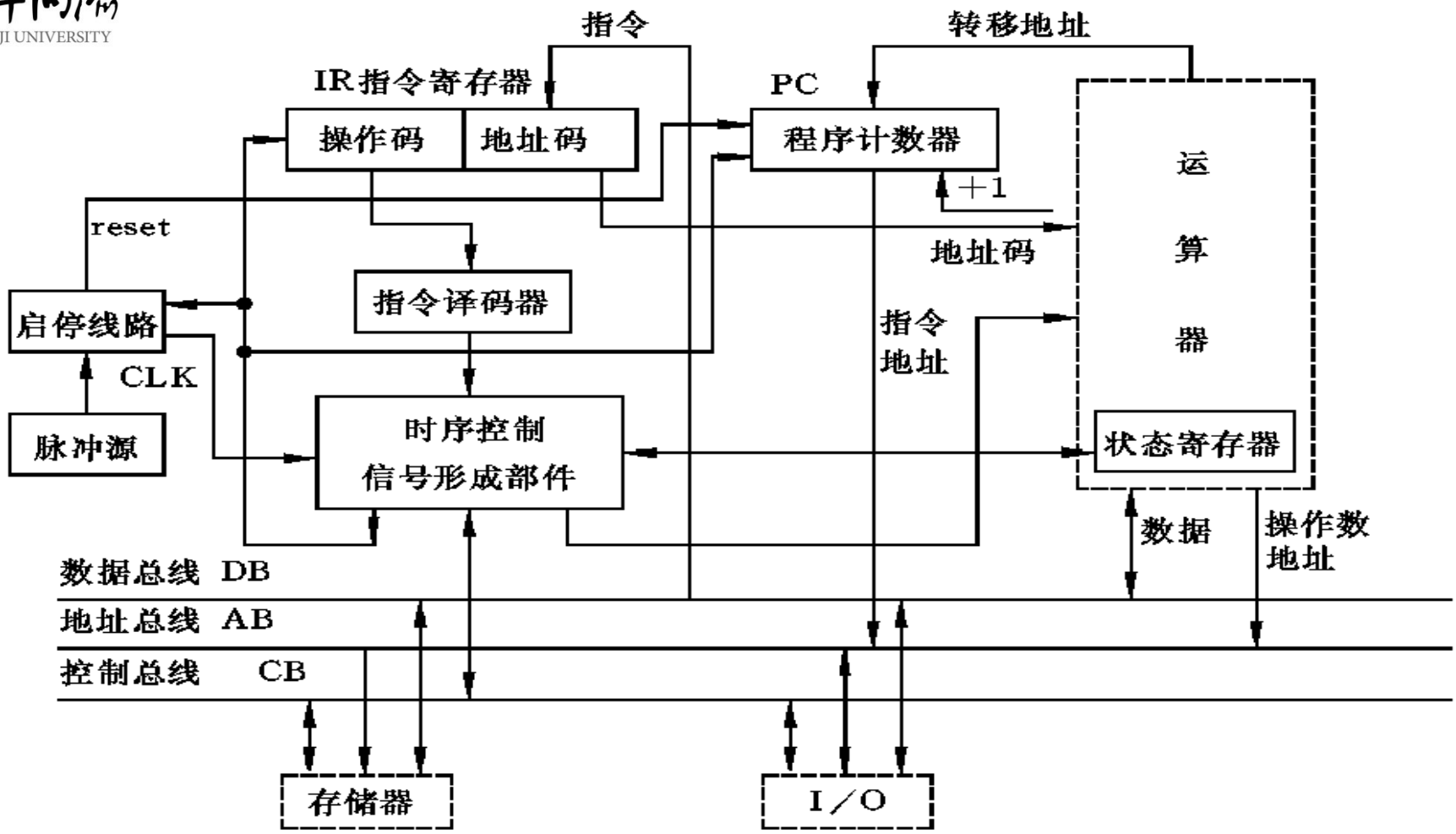
2. 指令寄存器(IR)

用以存放当前正在执行的指令，以便在指令执行过程中，控制完成一条指令的全部功能。

3. 指令译码器或操作码译码器

对指令寄存器中的操作码进行分析解释，产生相应的控制信号。

控制器的组成结构图


指令
转移地址
IR指令寄存器
PC
操作码
地址码
程序计数器
运
算
器
+1
reset
地址码
指令译码器
启停线路
指令
地址
CLK
脉冲源
时序控制
信号形成部件
状态寄存器
数据
操作数
地址
数据总线 DB
地址总线 AB
控制总线 CB
存储器
I/O

5.2 控制器的组成

在执行指令过程中，需要形成有一定时序关系的操作控制信号序列，为此还需要下述组成部分。

4. 脉冲源及启停线路

脉冲源产生一定频率的脉冲信号作为整个机器的时钟脉冲，是机器周期和工作脉冲的基准信号，在机器刚加电时，还应产生一个总清信号(reset)。启停线路保证可靠地送出或封锁时钟脉冲，控制时序信号的发生或停止，从而启动机器工作或使之停机。

5.2 控制器的组成

5. 时序控制信号形成部件

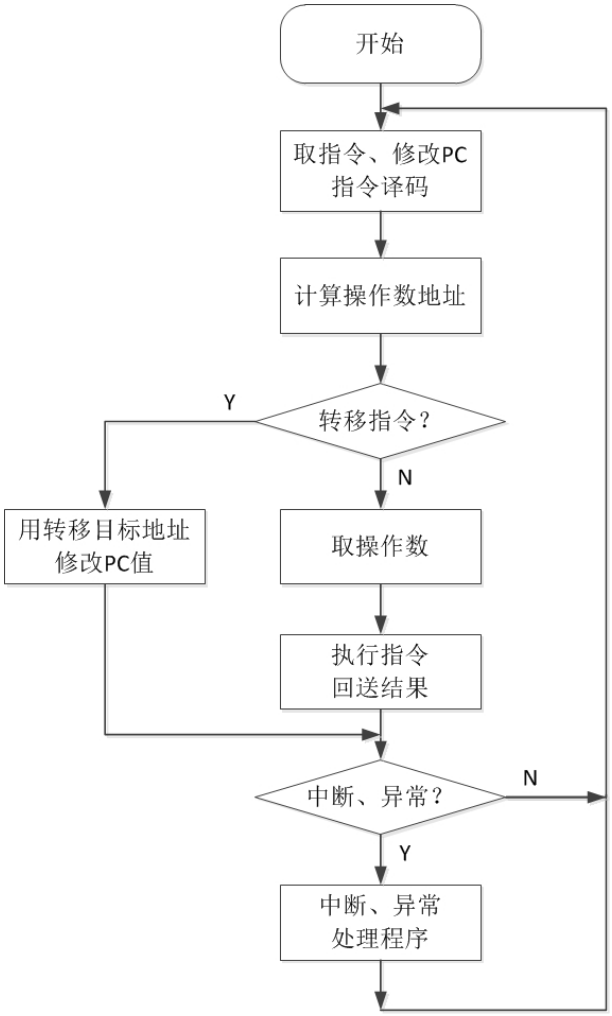
当机器启动后，在CLK时钟作用下，根据当前正在执行的指令的需要，产生相应的时序控制信号，并根据被控功能部件的反馈信号调整时序控制信号。

5.2 控制器的组成

5.2.3 机器指令执行过程

1.PC中的内容送入地址寄存器AR，并向内存发读命令，从内存将这条指令读入IR，修改PC。

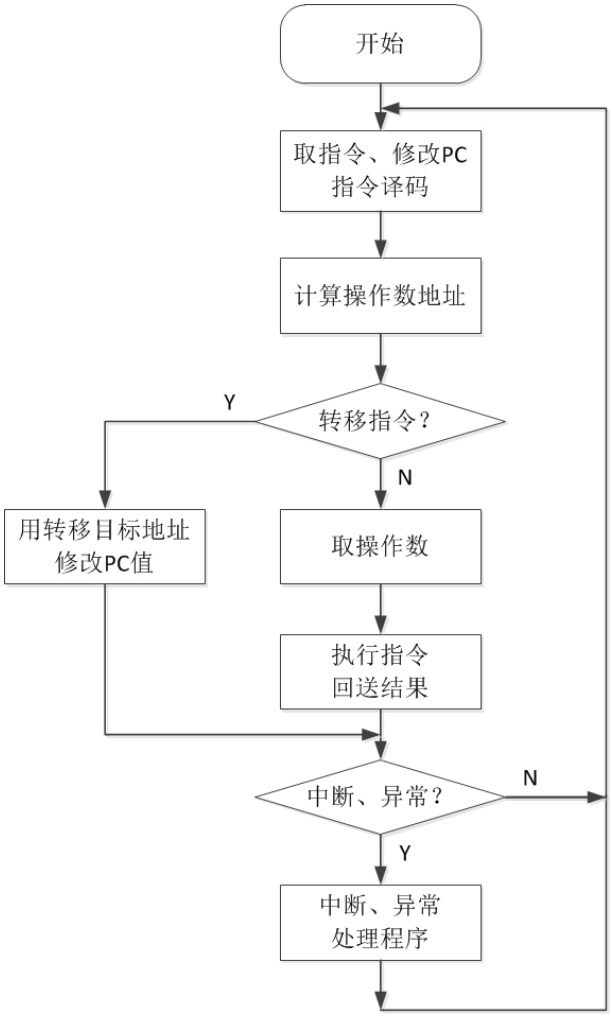
2.指令译码器分析指令的操作性质及操作数形成地址，以便向存储器、运算器等发出响应的微操作命令序列。



5.2 控制器的组成

3.如是操作类指令，需要内存提供数据的话，控制器就根据寻址方式形成有效地址送入AR，并向内存发读命令。

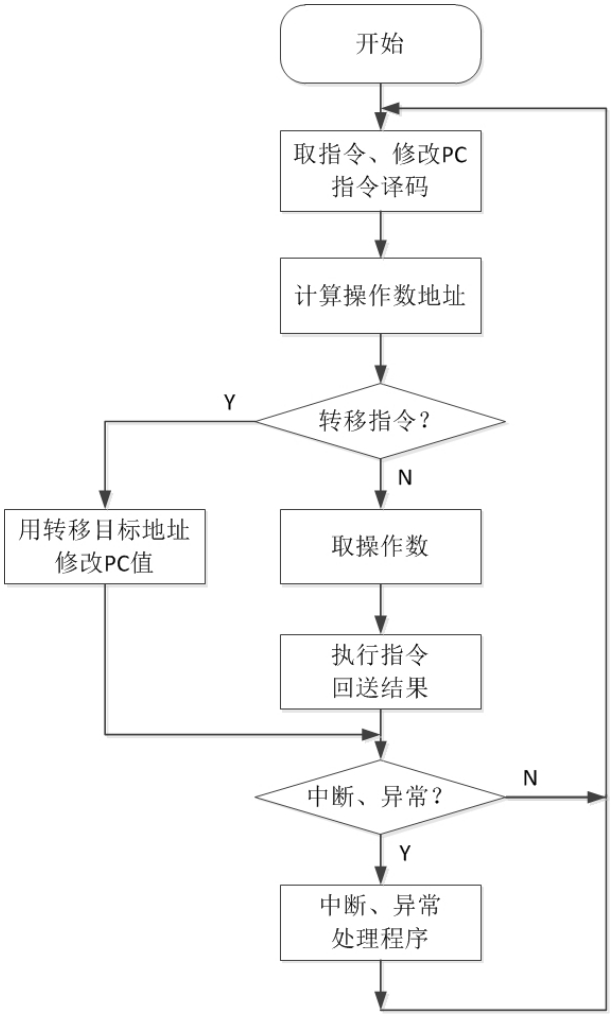
4.从内存取出的运算数据送入ALU并随即命令运算器对数据进行操作。



5.2 控制器的组成

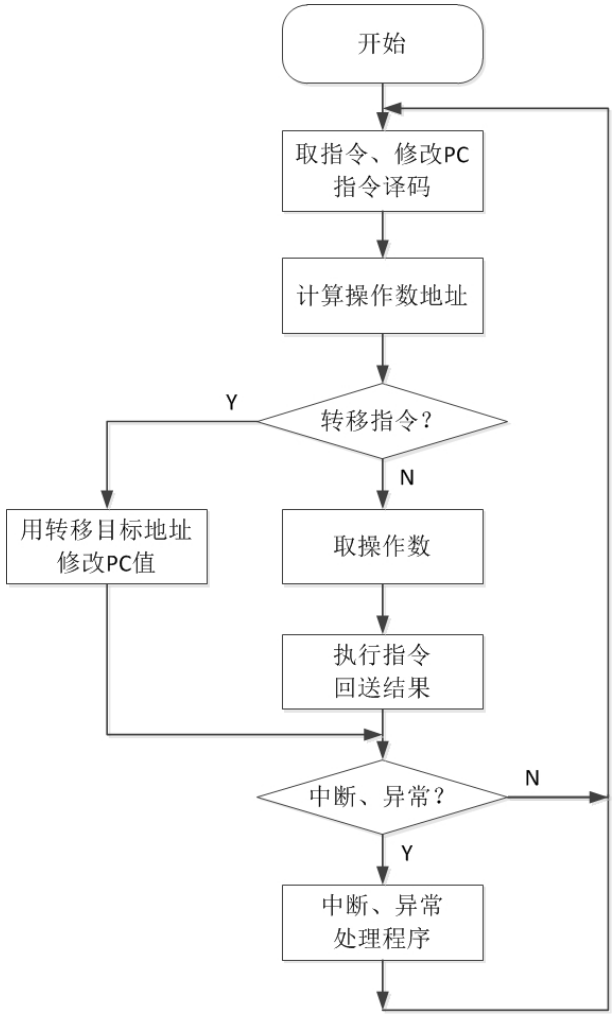
5.操作结果送入通用寄存器或内存。

6.一条指令执行完毕，控制器接着执行指令计数器PC指出的下一条指令，在顺序执行的情况下，是在上条指令被取出以后，PC内容加1。在非顺序执行的情况下，比如转移，就跳过4、5步，将转移地址写入PC。



5.2 控制器的组成

7.一条指令执行完检测是否有中断（异常），有转入处理程序，否则转回第一步，取下一条指令。



5.2 控制器的组成

5.2.4 指令周期

CPU 每取出并执行一条指令所需的全部时间称为指令周期。取指阶段完成取指令和分析指令的操作，也称取指周期。执行阶段完成执行指令的操作，也称执行周期。

因为各种指令操作功能不同，所以各种指令的指令周期可能是不同的。一个完整的指令周期可包括取指、间址、执行和中断4个周期。

5.2 控制器的组成

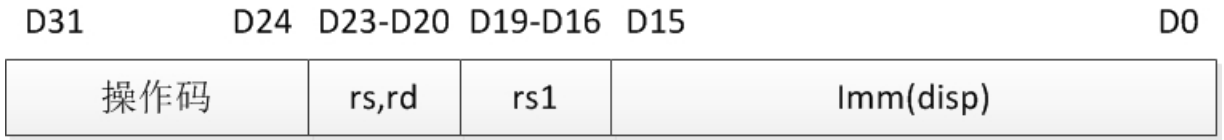
指令执行过程举例

运算器由8个通用寄存器GR及一个算逻运算部件ALU组成，并有4个记忆运算结果状态的标志触发器N，Z，V和C。

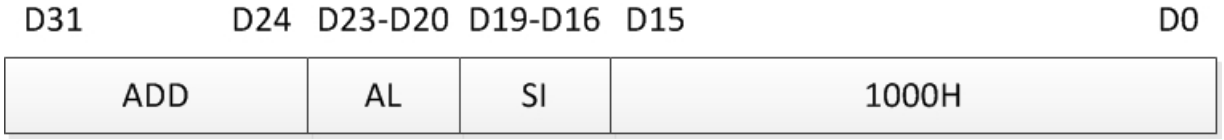
- **N(负数)：** 当运算结果为负数时，置“1”，否则为“0”。
- **Z(零)：** 当运算结果为零时，Z = 1，否则Z = 0。
- **V(溢出)：** 当运算结果溢出时，V = 1，否则V = 0。
- **C(进位)：** 当加法运算产生进位信号或减法运算产生借位信号时，C = 1，否则C = 0。

5.2 控制器的组成

假设指令格式：

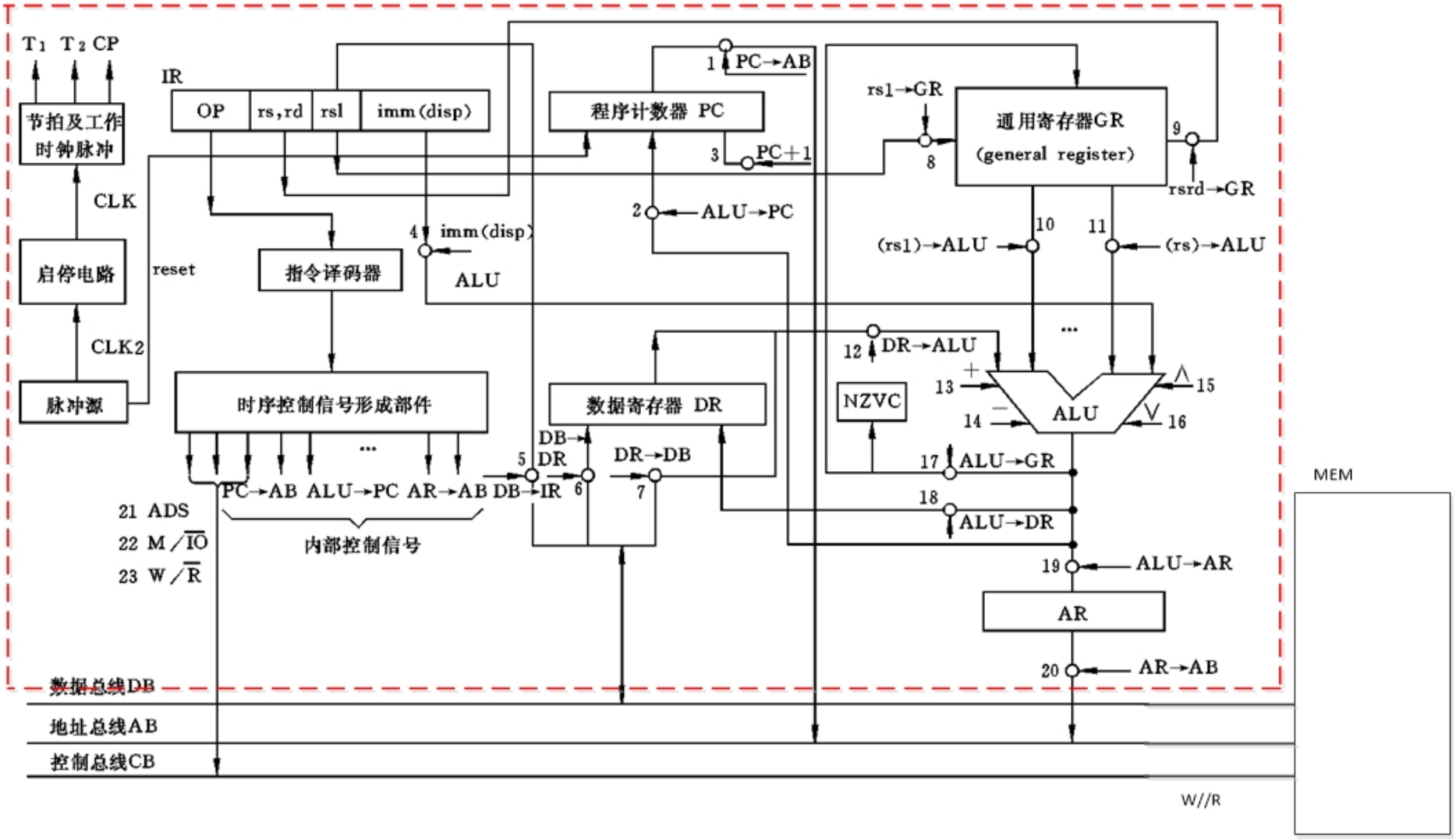


ADD AL，SI[1000H]；AL←AL+[SI+1000H]



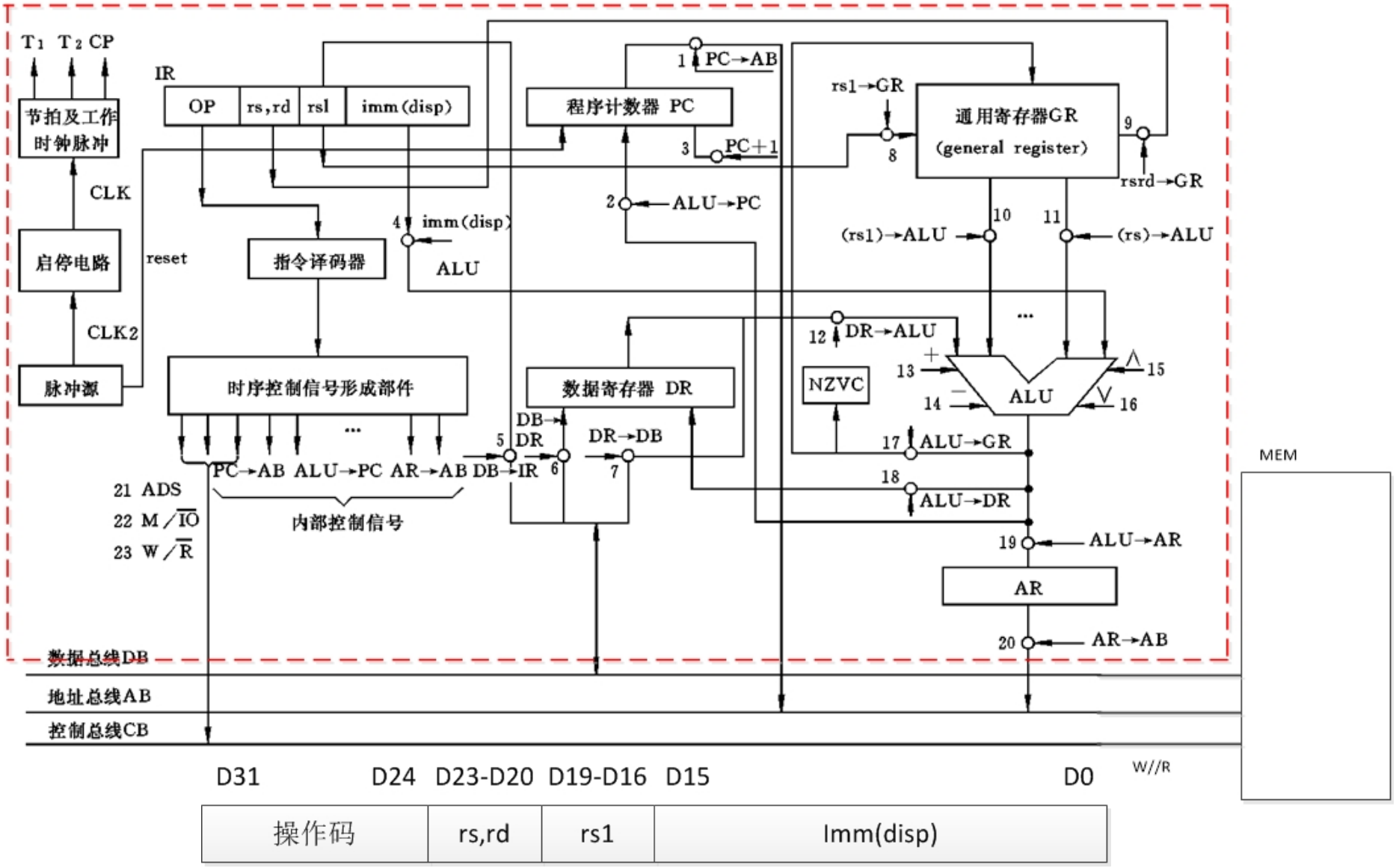
（1）一条加法指令的执行过程

加法指令完成以下操作



①从存储器取指令，送入指令寄存器，并进行操作码译码(分析指令)。

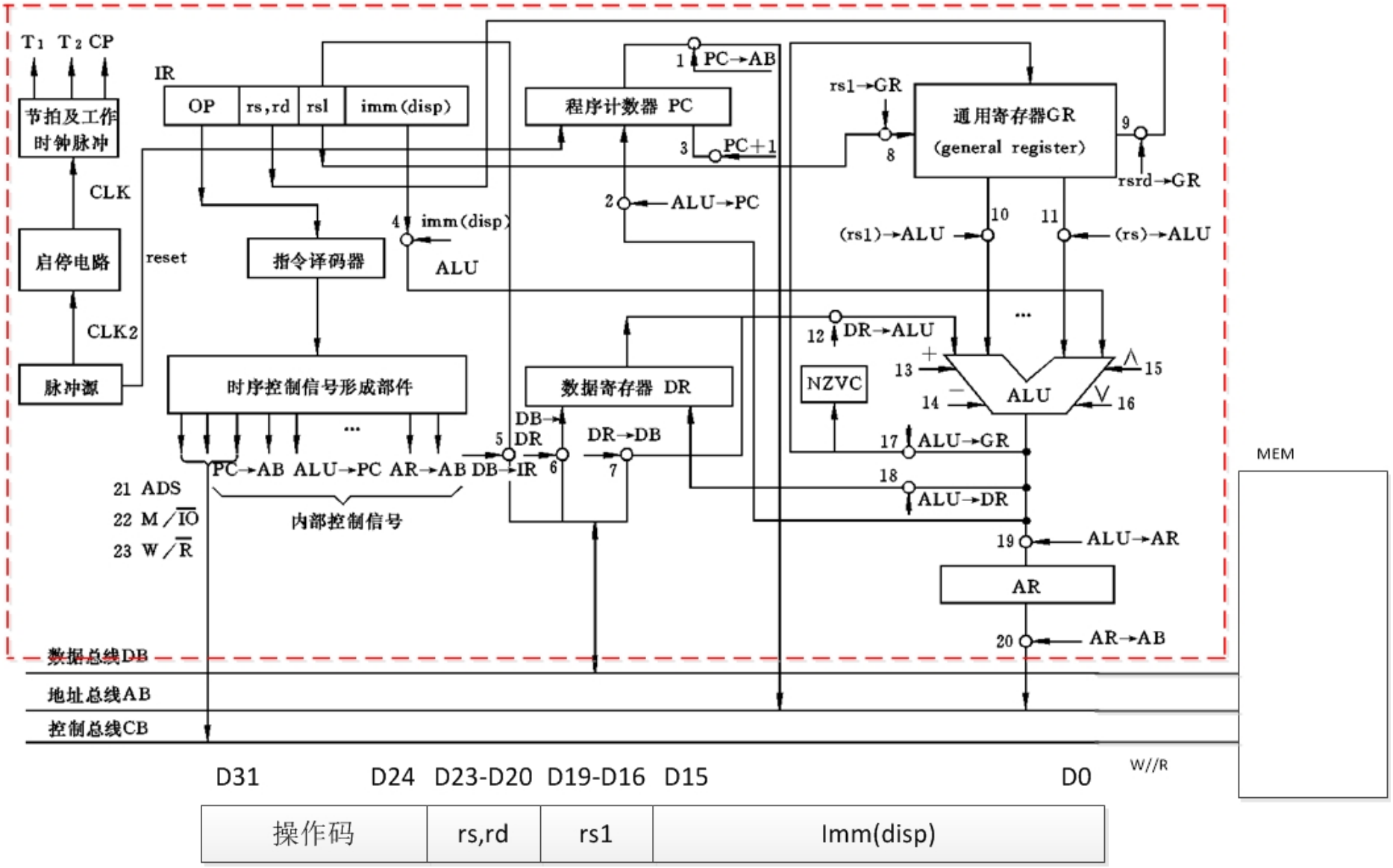
PC→AB，W//R=0,M//IO=1,DB→IR,PC+1



ADD AL，SI[1000H]；AL←AL+[SI+1000H]

②计算数据地址，将计算得到的有效地址送地址寄存器AR。

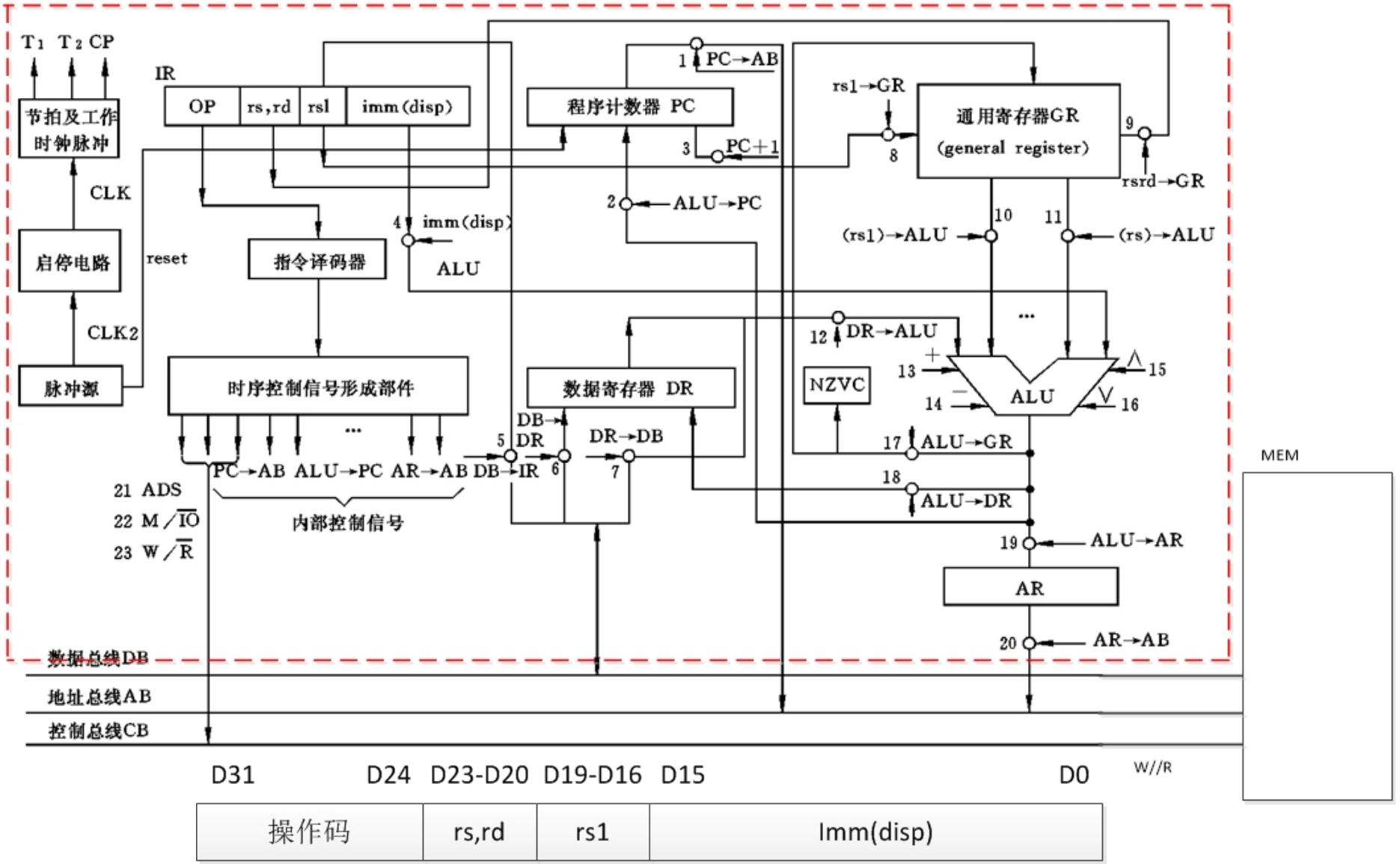
rs1→GR,(rs1)→ALU，disp→ALU，“+”,ALU→AR



ADD AL，SI[1000H]；AL←AL+[SI+1000H]

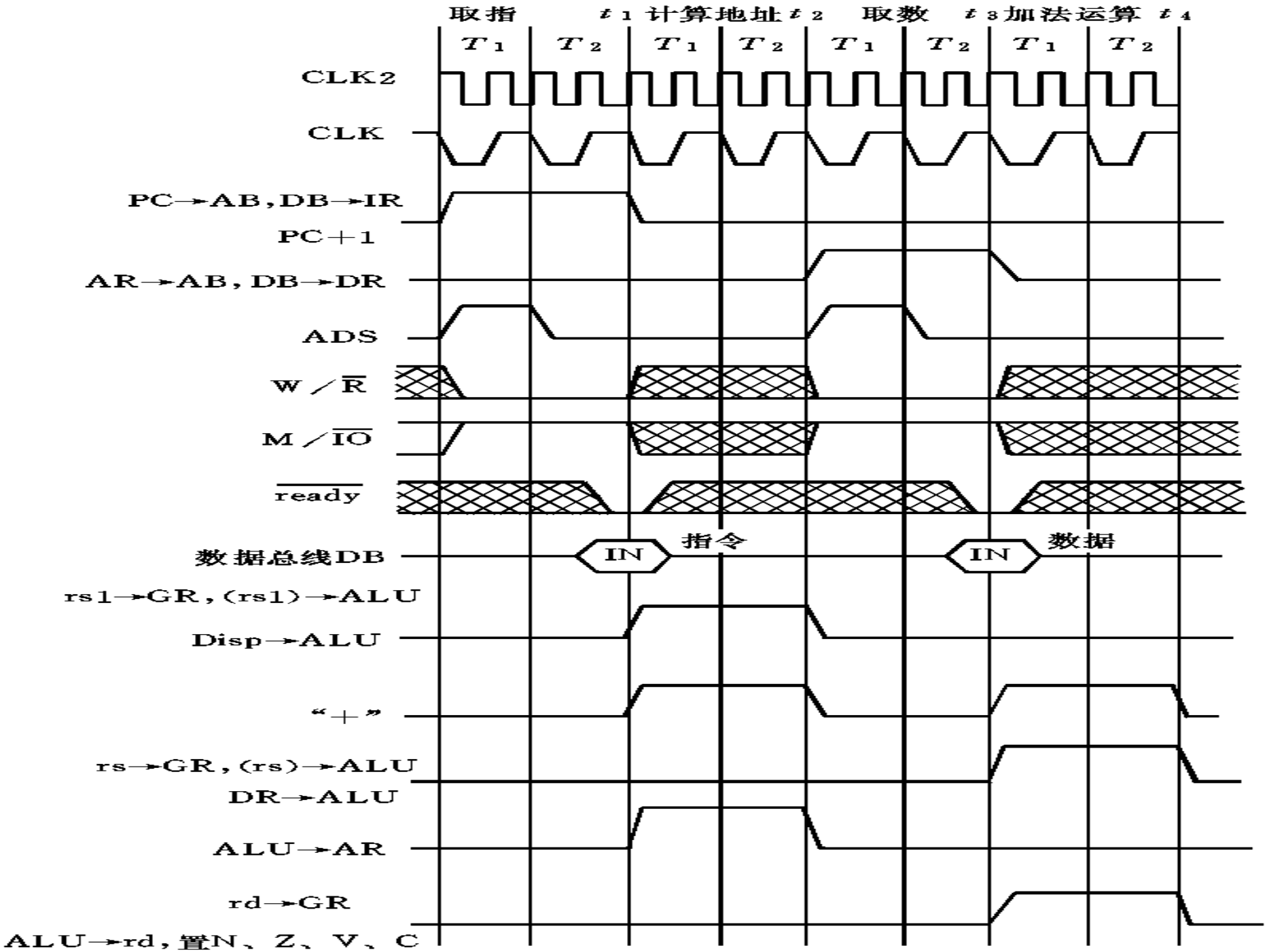
③到存储器取数。

AR→AB,W//R=0,M//IO=1,DB→DR



④进行加法运算，结果送寄存器，并根据运算结果置状态位N，Z，V，C。

rs→GR,(rs)→ALU,DR→ALU,“+”,rd→GR,ALU→GR(rd), **置**N,Z,V,C**状态位。**


取指
t_1
计算地址
t_2
取数
t_3
加法运算
t_4
T_1 T_2 T_1 T_2 T_1 T_2 T_1 T_2
CLK2
CLK
PC→AB,DB→IR
PC+1
AR→AB,DB→DR
ADS
W／$\overline{R}$
M／$\overline{IO}$
$\overline{ready}$
数据总线DB
IN
指令
IN
数据
rs1→GR,(rs1)→ALU
Disp→ALU
“+”
rs→GR,(rs)→ALU
DR→ALU
ALU→AR
rd→GR
ALU→rd,置N、Z、V、C

5.2 控制器的组成

(2) 条件转移指令的执行过程

指令功能：根据N，Z，V，C的状态，决定是否转换。如转移条件成立，则转移到本条指令所指定的地址，否则顺序执行下一条指令。

假设指令格式：

本条指令完成以下操作：

5.2 控制器的组成

指令格式：

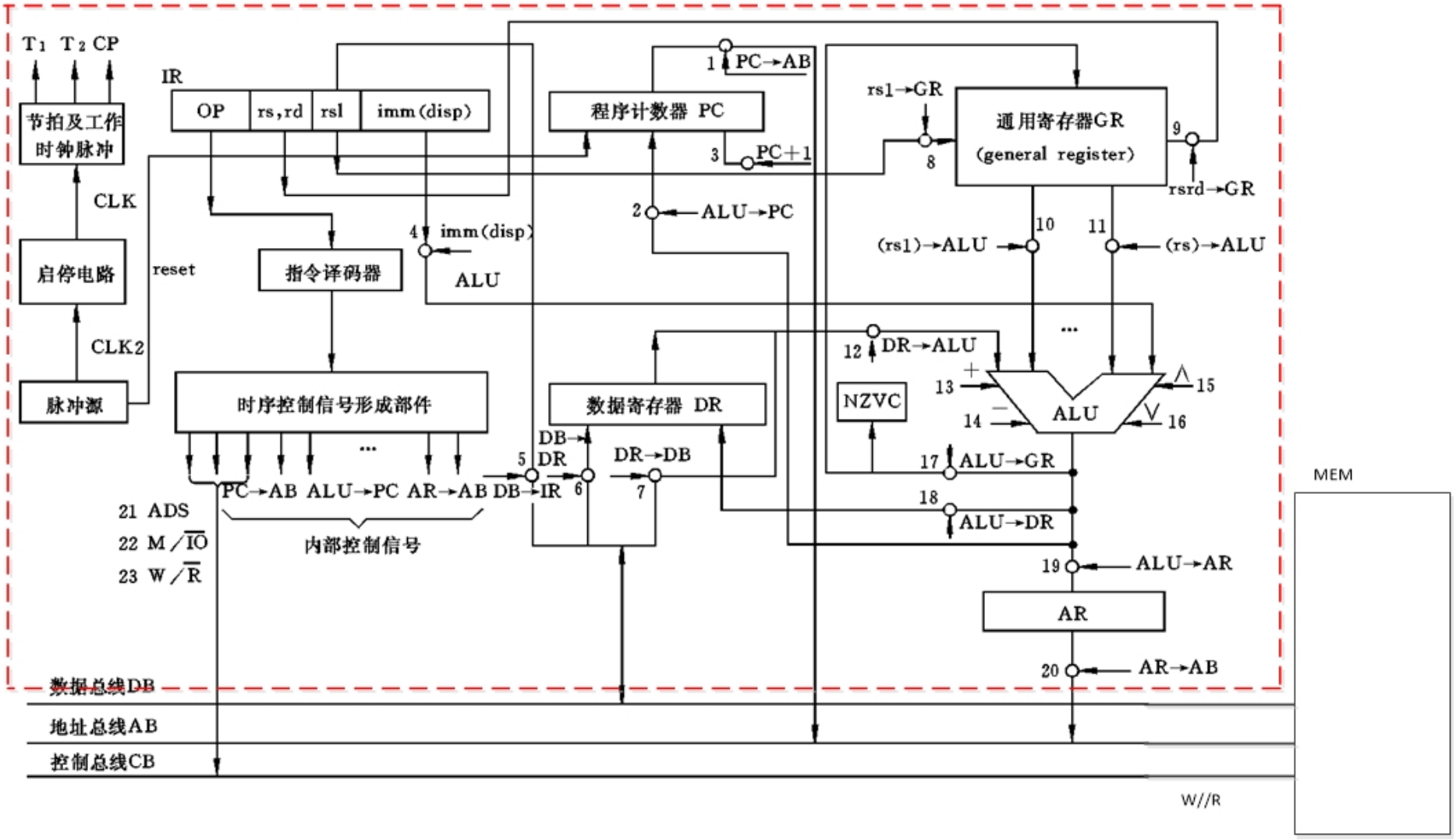
①从存储器取指令，送入指令寄存器，并进行操作码译码。

PC→AB， W//R=0, M//IO=1, DB→IR, PC+1

②如转移条件成立，根据指令规定的寻址方式计算有效地址，转移指令采用偏址寻址方式。

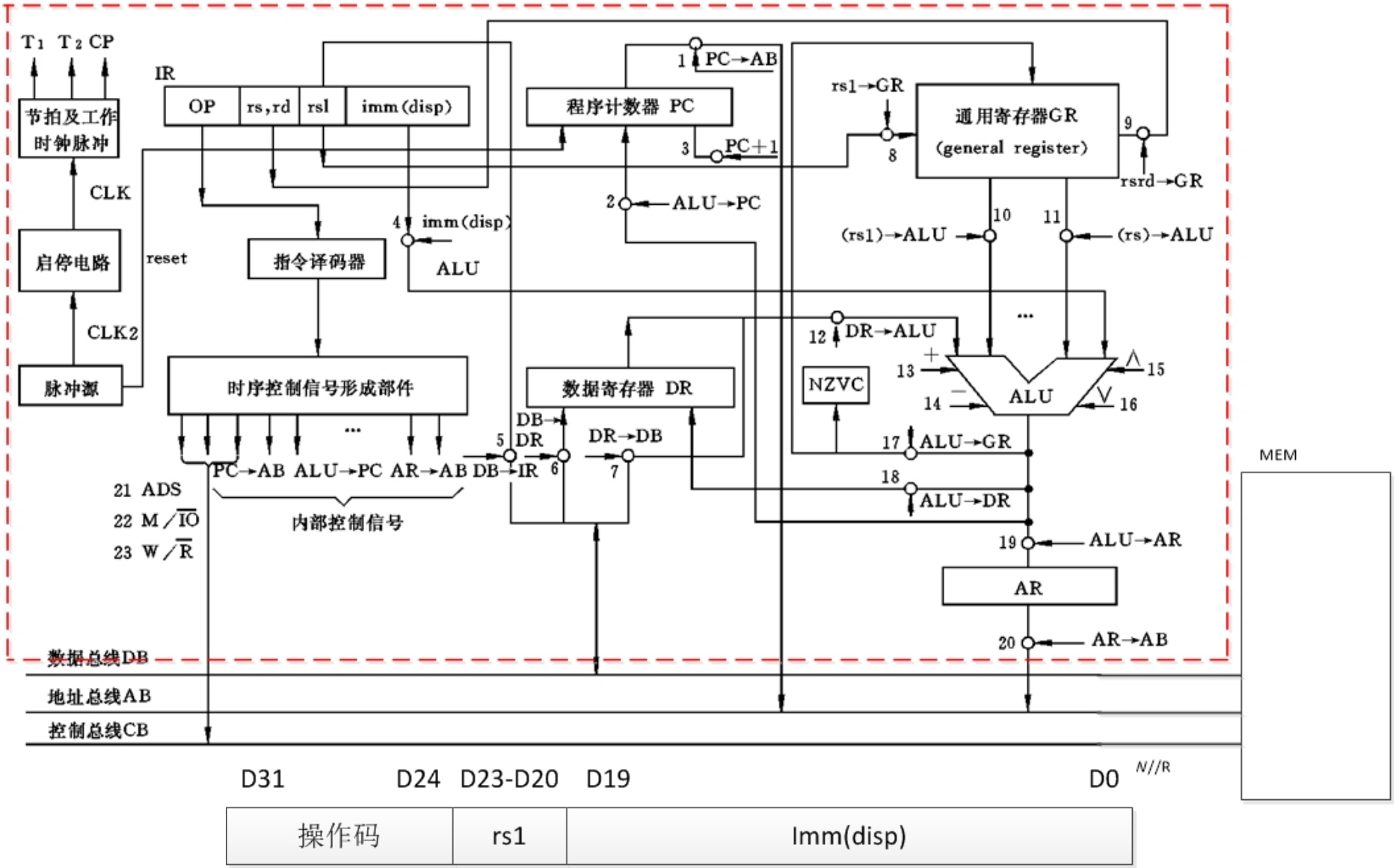
此时转移地址 = rs1 + disp

rs1→GR, (rs1)→ALU， disp→ALU， +， ALU→PC



①从存储器取指令，送入指令寄存器，并进行操作码译码(分析指令)。

PC→AB，W//R=0,M//IO=1,DB→IR,PC+1



②如转移条件成立，根据指令规定的寻址方式计算有效地址，转移指令采用偏址寻址方式。

此时转移地址 = rs1 + disp

rs1→GR, (rs1)→ALU, disp→ALU, +, ALU→PC

5.2 控制器的组成

5.2.4 控制器的控制方式

1.同步控制方式

计算机执行的每条指令操作都是在相同的时间里完成。

2.异步控制方式

计算机执行的每条指令操作不是在相同的时间里完成的，而是根据前一条指令操作是否已完成的“完成”信号来控制。

5.2 控制器的组成

3.联合控制方式

对大多数的指令采用同步控制方式，对个别少数无法采用同步控制方式的指令，采用区别对待的办法，在时间上给予延长。

5.2 控制器的组成

5.2.5 控制器的组成方式

1.组合逻辑控制器

组合逻辑控制器是以逻辑代数为工具进行设计的，是由各类门电路构成。

2.微程序控制器

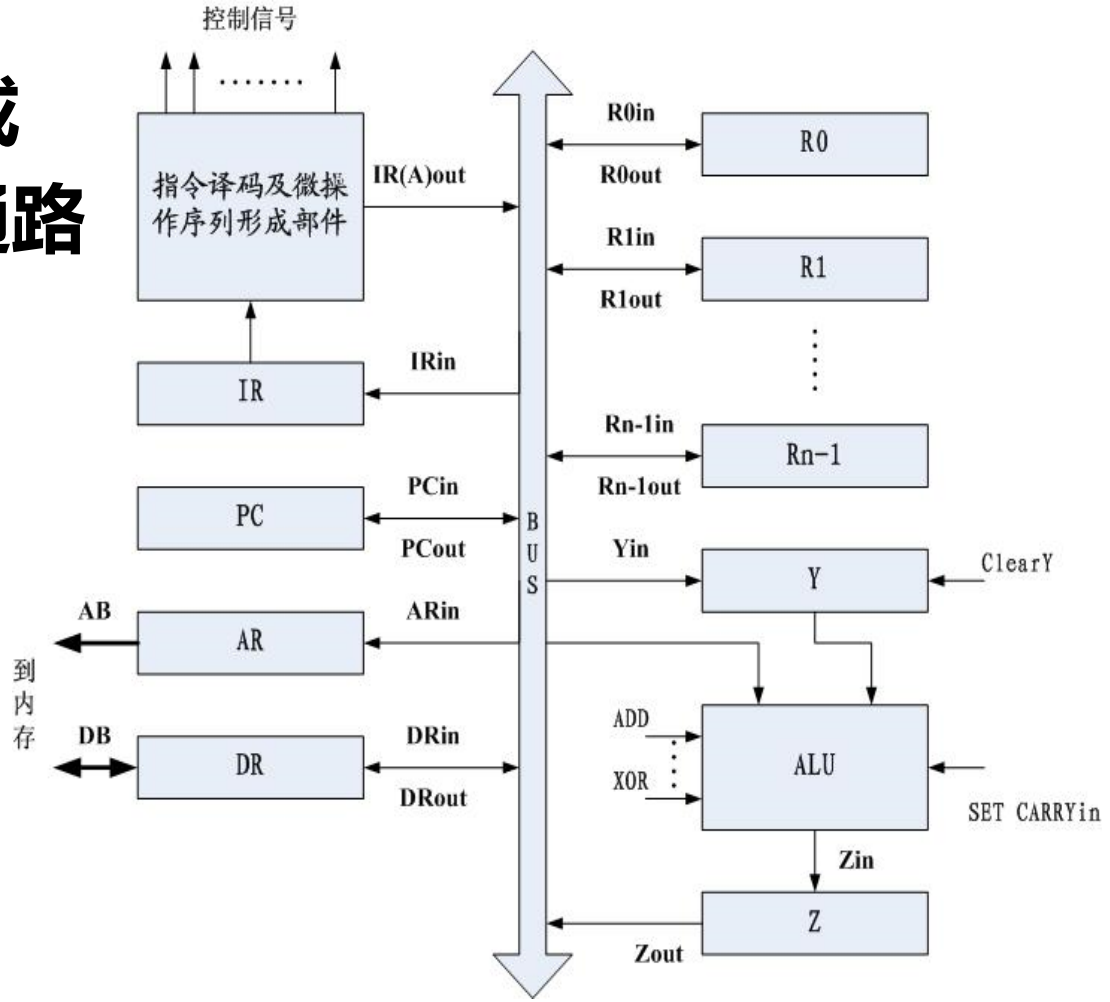
微程序控制器是用微程序设计的方法设计的，是由微程序存储器和门电路构成。

5.2 控制器的组成

5.2.6 数据通路的构成

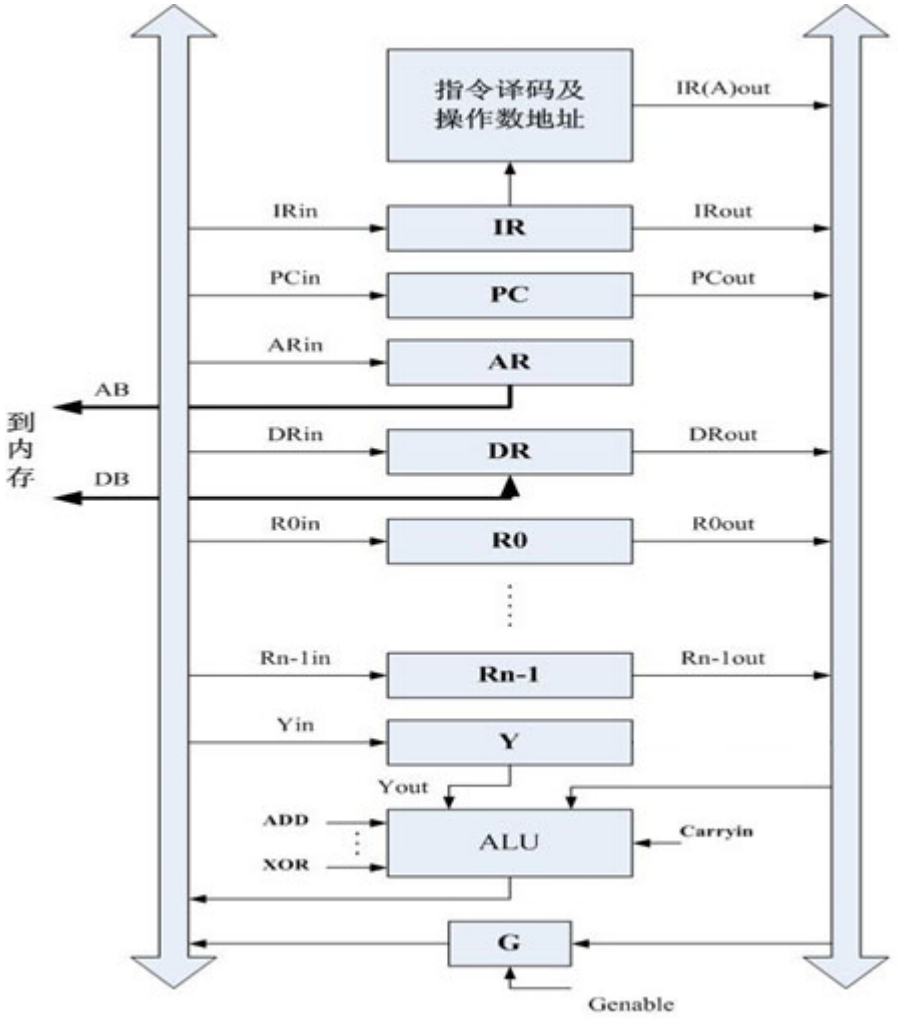
1.基于单总线的数据通路

在总线结构中，可同时进行的数据传输量取决于总线的数量。对于单总线结构而言，总线上可以有多个部件同时接收数据，但同一时刻只能一个部件向总线发送数据。因此，连接到总线上的部件需要进行输出控制，以防总线冲突。



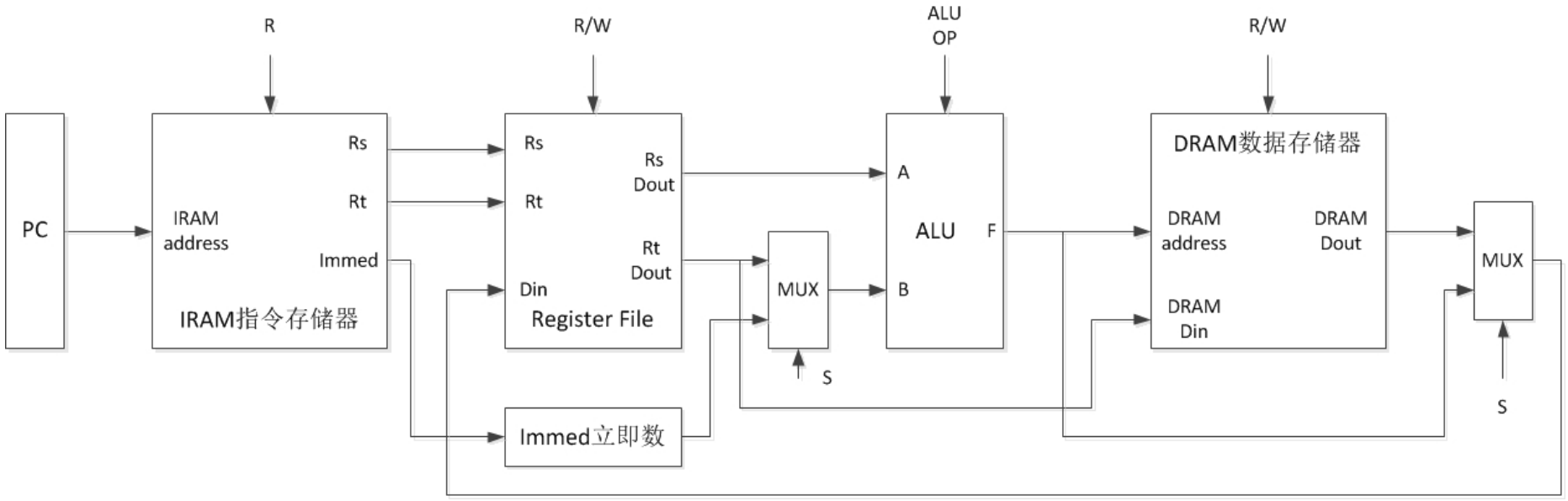
5.2 控制器的组成

2.基于双总线的数据通路



5.2 控制器的组成

3.基于专用通路结构的数据通路



5.2 控制器的组成

4.单周期和多周期处理器数据通路

单周期处理器是指所有的指令在一个时钟周期内完成的处理器，尽管不同指令执行时间不同，但对单周期处理器而言，时钟周期必须设计成对所有指令都等长。

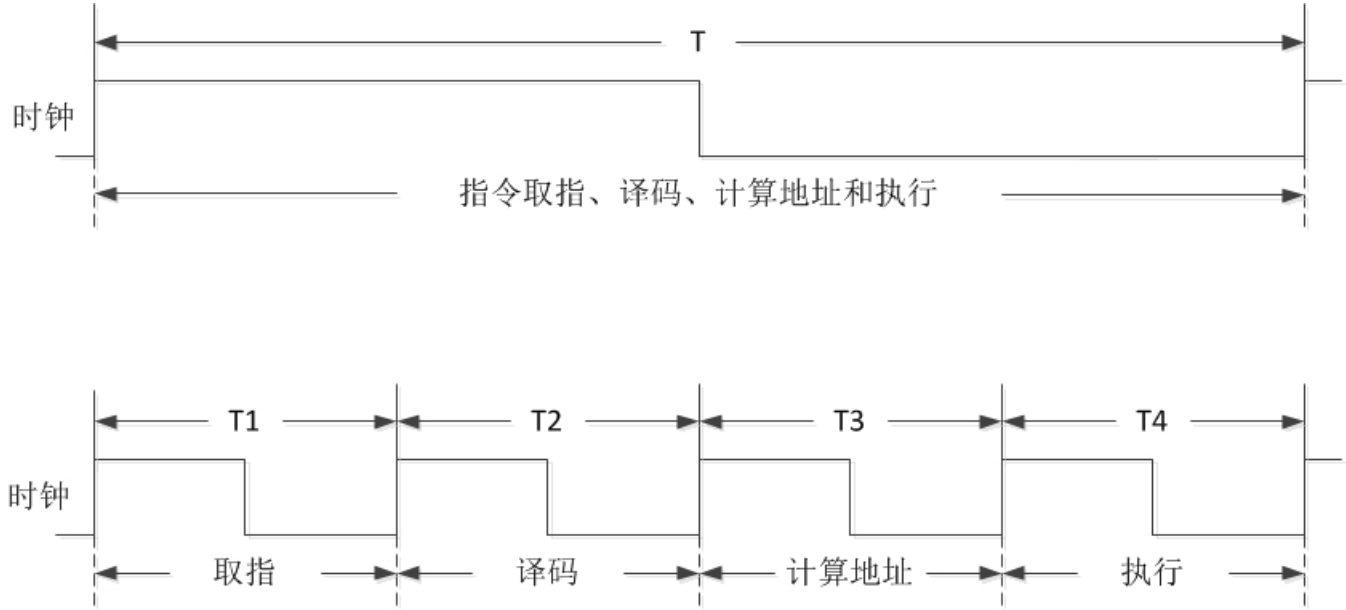
注意：在单周期处理器中，一条指令执行过程中数据通路的任何资源都不能被重复使用，因此，任何需要被多次使用的资源（如加法器）都需要设置多个，否则就会发生资源冲突，这是设计单周期处理器中数据通路时必须考虑的重要环节。

5.2 控制器的组成

多周期处理器是指指令可根据执行的时间长度，在一个时钟周期或多个时钟周期内完成的处理器，所以，对不同指令执行时间，对多周期处理器而言，指令所用的时钟周期数也不同。

注意：在多周期处理器中，一条指令执行过程中数据通路的任何资源都能被重复使用，因此，任何需要被多次使用的资源（如加法器）可设置一个，在不同的时钟周期中被使用。

5.2 控制器的组成

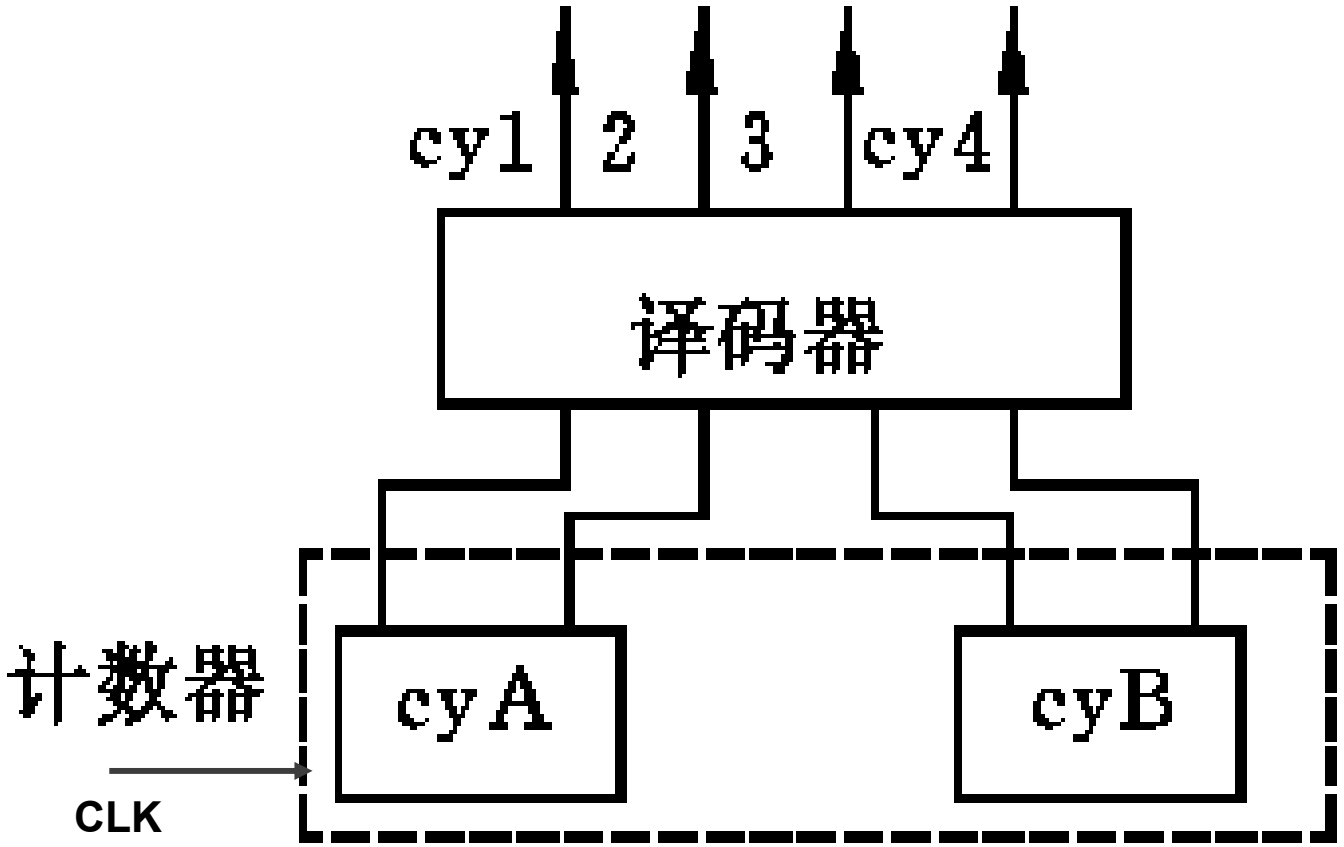


5.3 组合逻辑控制器

5.3.1 时序与节拍

一条指令的实现可分成取指、计算地址、取数及执行等几个步骤，对于多周期处理器在大部分情况下，每一步由一个机器周期实现。对于单周期处理器来说，所有操作就在一个机器周期中实现。

5.3 组合逻辑控制器



5.3 组合逻辑控制器

四个节拍cy1、cy2、cy3、cy4可形成四个周期

- **cy1----取指**
- **cy2----计算地址**
- **cy3----取数**
- **cy4----执行**

每次启动时总是cy1有效，就是取指，然后再是其它周期。由于每条指令的功能不同，所以每条指令所需的周期数可能就不同。对于多周期处理器：

指令周期是由若干个机器周期构成

机器周期是由若干个时钟周期构成

5.3.2 操作控制信号的产生

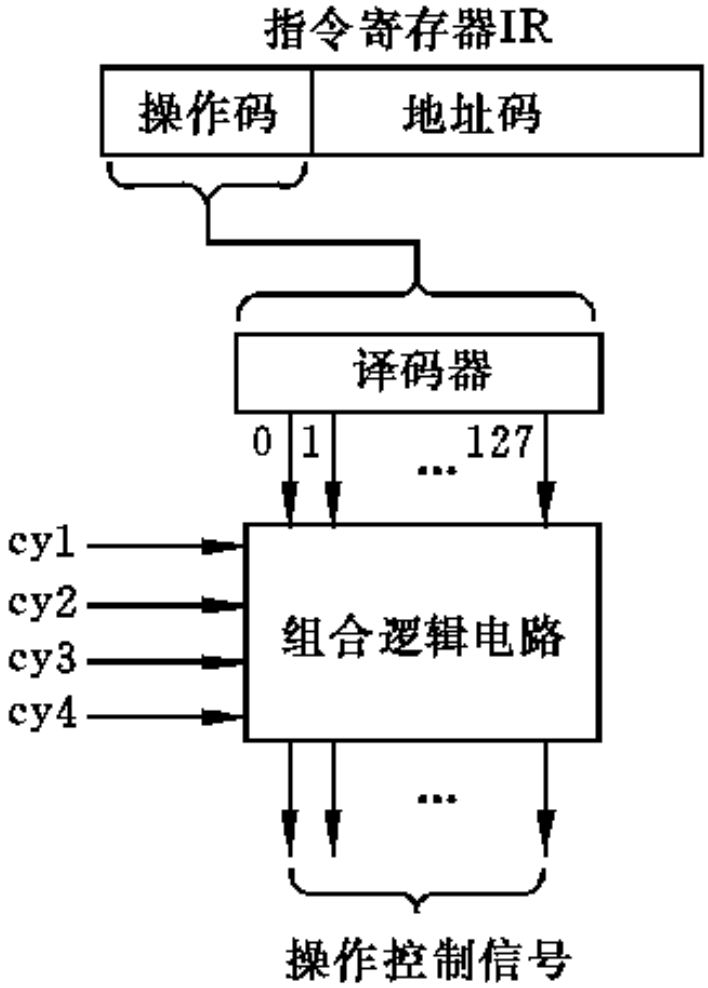
- **操作码译码器**

指令由操作码与地址码两部分组成，其中操作码表示当前正在执行的是什么指令。各条指令所需实现的操作随指令而异。假如操作码有7位，则最多可表示128条指令，一般在机器内设置一个指令译码器，其输入为操作码(7位)，输出有128根线，在任何时候，有且仅有一根线为高电位，其余均为低电位(或反之，即一根线为低电位，其余为高电位)，每根输出线表示一条指令，因此译码器的输出可以反映出当前正在执行的指令。

5.3 组合逻辑控制器

由译码器的输出和机器周期状态cy1～cy4作为输入，使用逻辑电路产生操作控制信号，其框图如图所示。实际上为了简化逻辑，译码器与组合逻辑是结合在一起设计的。

5.3 组合逻辑控制器



5.3 组合逻辑控制器

●组合逻辑控制器设计

1.根据指令功能设计数据通路

根据指令功能列出完成每条指令功能所需的部件，用部件关系表将每个部件的数据来源列出，根据关系表用连线将每个部件连接起来，设计出初步的数据通路。

2.绘制指令流程图

根据初步安排好的数据通路结构，把每一条机器指令都分解为一系列的微操作，并排列为有先后次序、互相衔接的指令流程图。

5.3 组合逻辑控制器

3.编排指令操作时间表

依据各条机器指令的操作流程图及控制方式，将每个微操作安排到周期节拍中。

4.进行微操作综合

按照所有机器指令的操作时间表，把相同的微操作综合起来，得到每个微操作的逻辑表达式。

5.根据各微操作的逻辑表达式以及所用的器件（门电路），构成微操作序列形成部件的组合逻辑网络。

5.3 组合逻辑控制器

5.3.3 单周期组合逻辑控制器设计举例

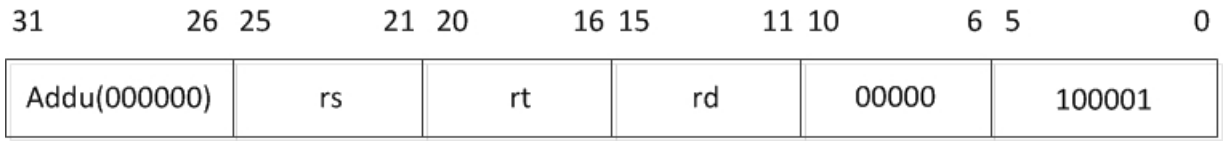
设计一台基于MIPS指令架构的单周期CPU，共有8条指令。

序号	指令	指令功能	备注
1	Addu rd,rs,rt	rd←rs+rt	寄存器加
2	Subu rd,rs,rt	rd←rs-rt	寄存器减
3	Ori rt,rs,immediate	rt←rs ∨ext_imm16	立即数或
4	Sll rd,rt,sa	rd←rt||0^s	左移
5	Beq rs,rt,offset	rs=rt， PC←NPC+Sign_ext(offset||0^2); rs≠rt， PC←NPC(PC+4)	条件转移
6	J target	PC←PC_{31-28}||instr_index||0^2	无条件转移
7	Lw rt,offset(base)	rt←[base+Sign_ext_offset]	取数
8	Sw rt,offset(base)	[base+Sign_ext_offset]←rt	存数

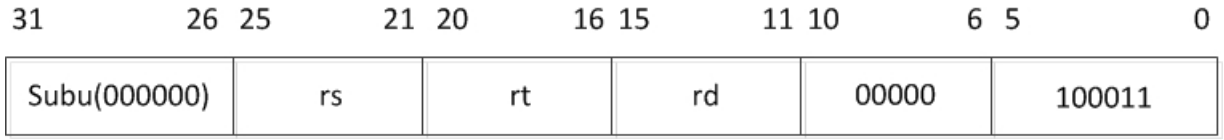
5.3 组合逻辑控制器

1.Addu rd,rs,rt ;

rd←rs+rt,PC←NPC(PC+4)

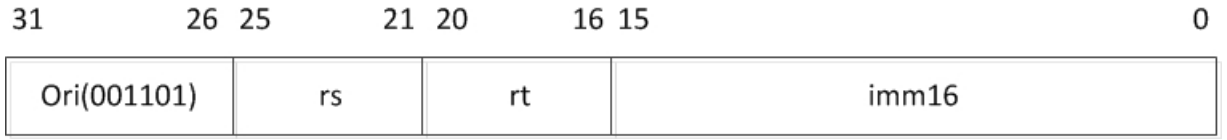


2.Subu rd,rs,rt ;



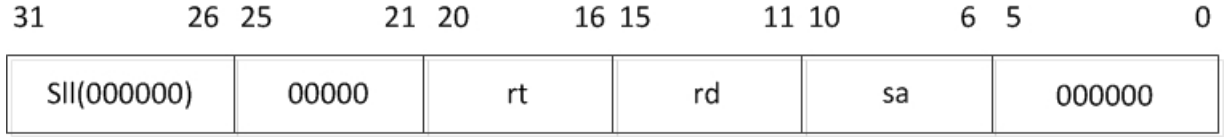
rd←rs-rt,PC←NPC(PC+4)

3.Ori rt,rs,immediate ;



rt←rs ∨ext_imm16, PC←NPC(PC+4)

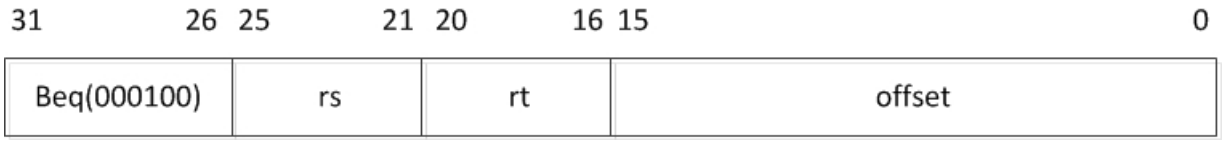
4.Sll rd,rt,sa ;



rd←rt||0^s,PC←NPC(PC+4)

5.3 组合逻辑控制器

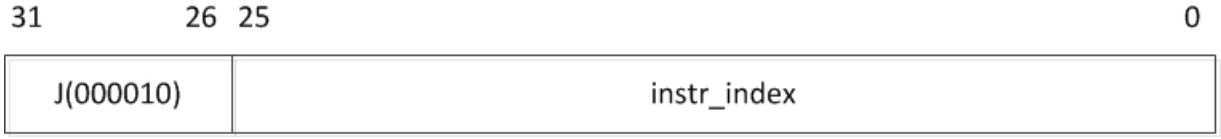
5.Beq rs,rt,offset ;



rs=rt,PC←NPC+Sign_ext(offset||0^2);

rs≠rt，PC←NPC(PC+4)

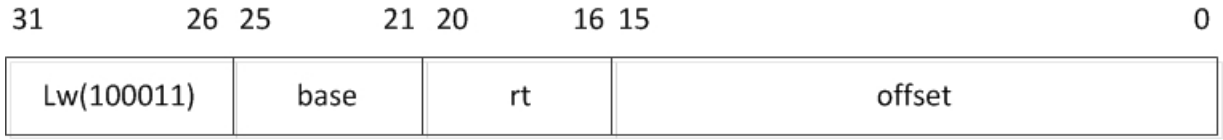
6.J target ;



PC←PC_{31-28}||instr_index||0^2 ，PC←NPC(PC+4)

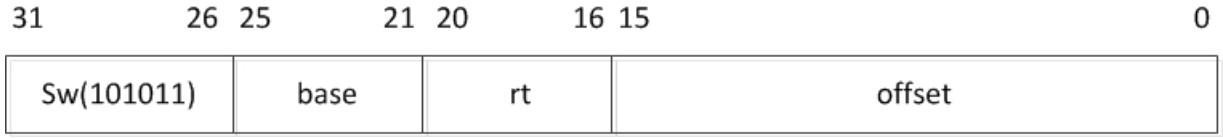
5.3 组合逻辑控制器

7.Lw rt,offset(base) ;



rt←[base+Sign_ext_offset],PC←NPC(PC+4)

8.Sw rt,offset(base) ;



[base+Sign_ext_offset]←rt,PC←NPC(PC+4)

5.3 组合逻辑控制器

1.根据指令功能设计数据通路

列出部件关系表

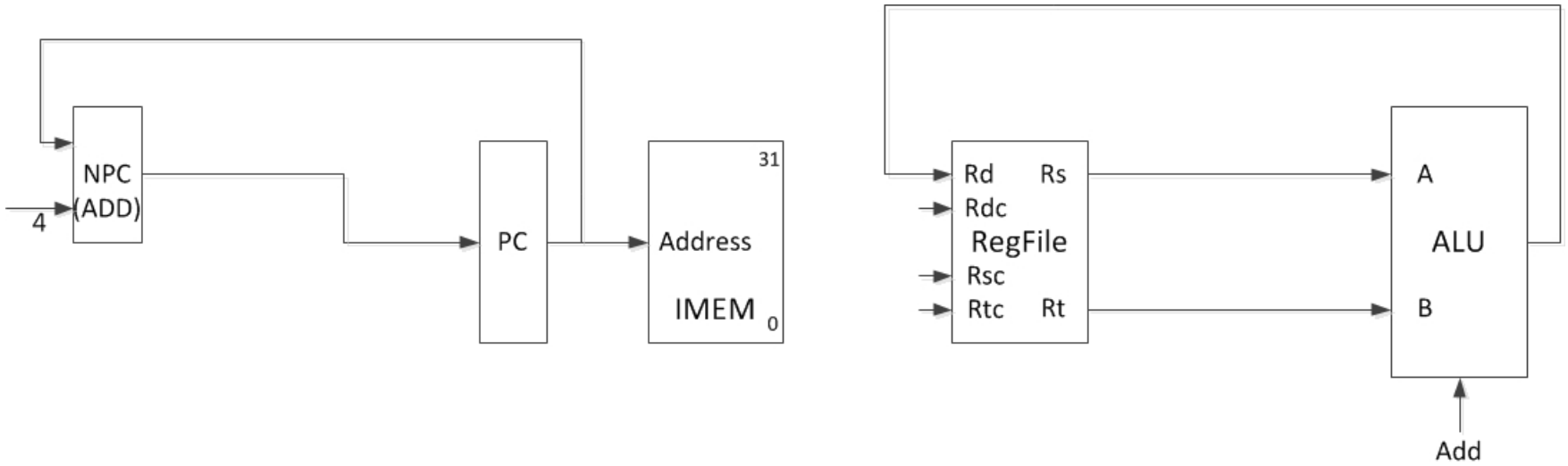
(1) Addu rd,rs,rt ;rd←rs+rt,PC←NPC(PC+4)

- 所需部件：PC、NPC（完成PC增值）、IMEM、Regfile、ALU
- 部件之间数据输入输出关系如下表：

	PC	NPC	IMEM	RegFile	ALU			
				Rd	A	B		
Addu	NPC	PC	PC	ALU	Rs	Rt		

5.3 组合逻辑控制器

	PC	NPC	IMEM	RegFile	ALU			
				Rd	A	B		
Addu	NPC	PC	PC	ALU	Rs	Rt		



5.3 组合逻辑控制器

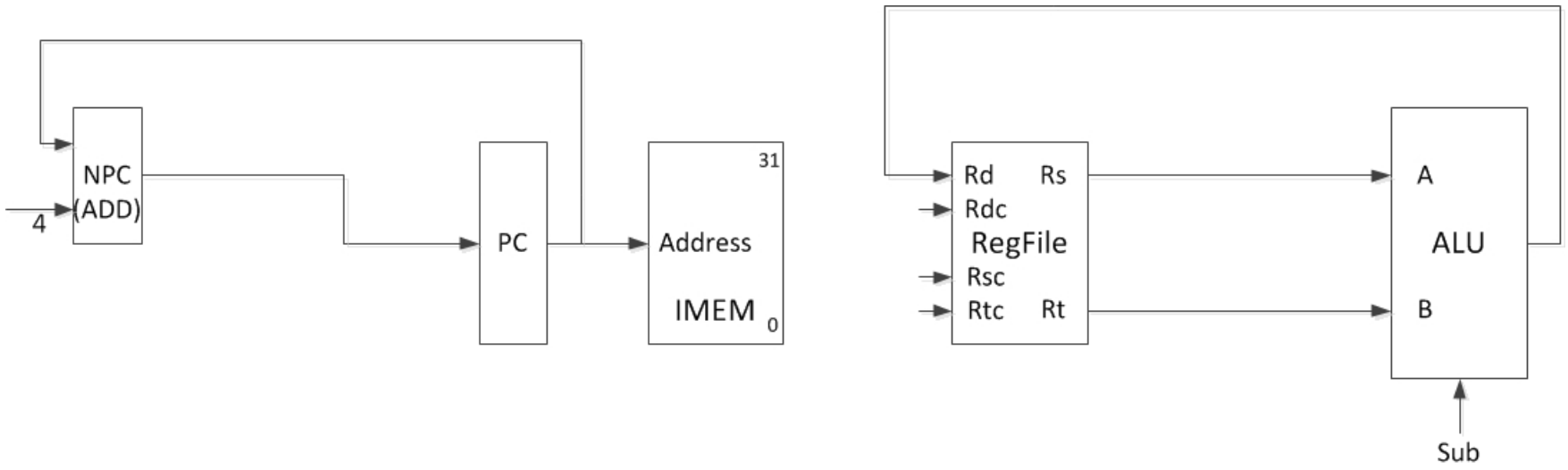
2.Subu rd,rs,rt ;rd←rs-rt,PC←NPC(PC+4)

- 所需部件：PC、NPC（完成PC增值）、IMEM、Regfile、ALU
- 部件之间数据输入输出关系如下表：

	PC	NPC	IMEM	RegFile	ALU			
				Rd	A	B		
Addu	NPC	PC	PC	ALU	Rs	Rt		
Subu	NPC	PC	PC	ALU	Rs	Rt		

5.3 组合逻辑控制器

	PC	NPC	IMEM	RegFile	ALU			
				Rd	A	B		
Addu	NPC	PC	PC	ALU	Rs	Rt		
Subu	NPC	PC	PC	ALU	Rs	Rt		



5.3 组合逻辑控制器

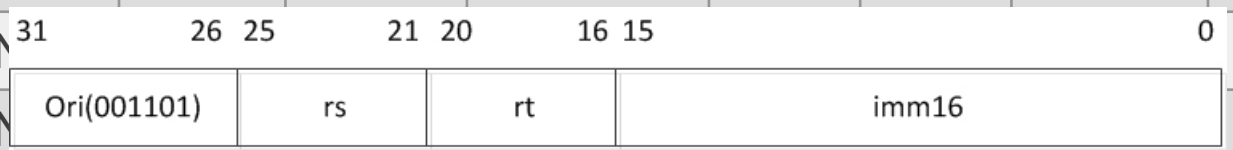
3.Ori rt,rs,imm16 ;rt←rs ∨ext.imm16, PC←NPC(PC+4)

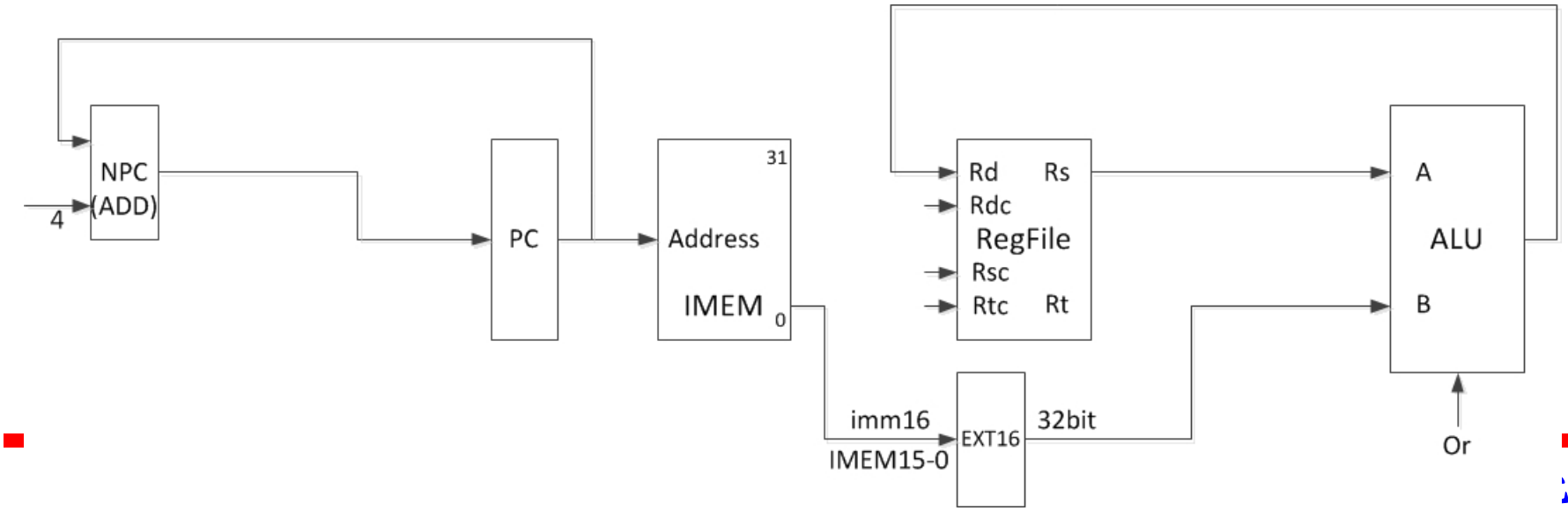
- 所需部件：PC、NPC（完成PC增值）、IMEM、Regfile、ALU、Ext16
- 部件之间数据输入输出关系如下表：

	PC	NPC	IMEM	RegFile	ALU		Ext16	
				Rd	A	B		
Addu	NPC	PC	PC	ALU	Rs	Rt		
Subu	NPC	PC	PC	ALU	Rs	Rt		
Ori	NPC	PC	PC	ALU	Rs	Ext16	imm16	

5.3 组合逻辑控制器

	PC	NPC	IMEM	RegFile	ALU		Ext16	
				Rd	A	B		
Addu	N							
Subu	N							
Ori	NPC	PC	PC	ALU	Rs	Ext16	imm16	





5.3 组合逻辑控制器

4.Sll rd,rt,sa ;rd←rt左移sa位,PC←NPC(PC+4)

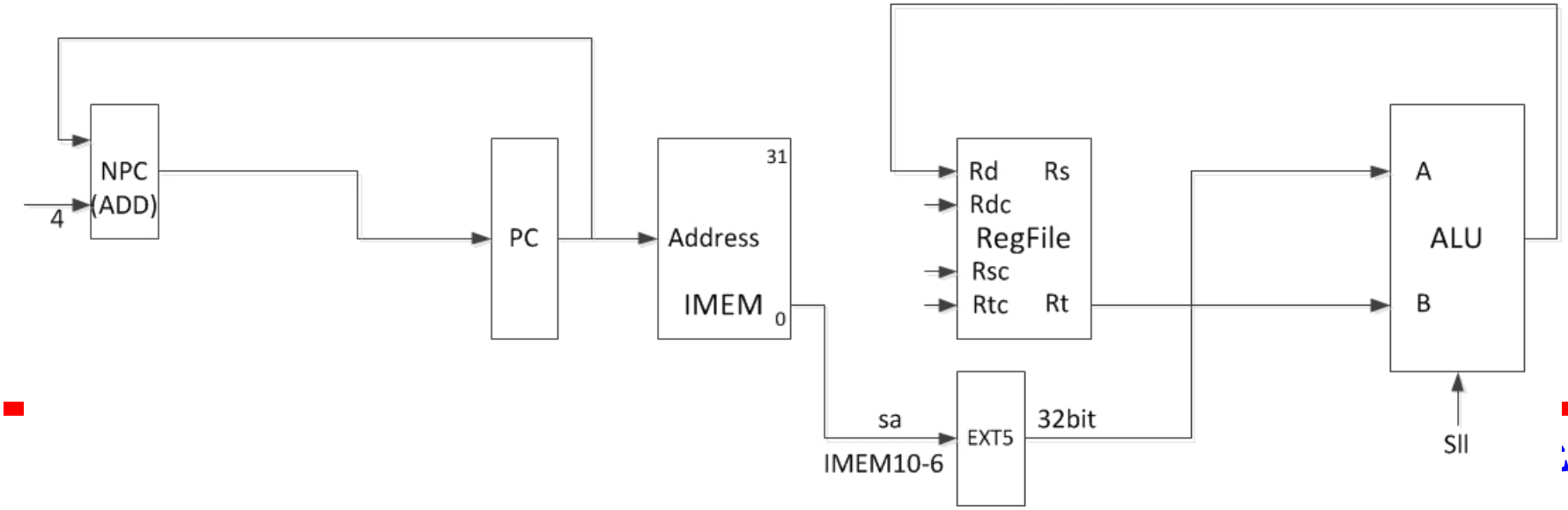
- 所需部件：PC、NPC（完成PC增值）、IMEM、Regfile、ALU、Ext5
- 部件之间数据输入输出关系如下表：

	PC	NPC	IMEM	RegFile	ALU		Ext16	Ext5
				Rd	A	B		
Addu	NPC	PC	PC	ALU	Rs	Rt		
Subu	NPC	PC	PC	ALU	Rs	Rt		
Ori	NPC	PC	PC	ALU	Rs	Ext16	imm16	
Sll	NPC	PC	PC	ALU	Ext5	Rt		sa

5.3 组合逻辑控制器

	PC	NPC	IMEM	RegFile	ALU		Ext16	Ext5
				Rd	A	B		
Addu	NPC	PC	PC	ALU	Rs	Rt		
Subu	N							
Ori	N							
Sll	NPC	PC	PC	ALU	Ext5	Rt		sa

31 26	25 21	20 16	15 11	10 6	5 0
Sll(000000)	00000	rt	rd	sa	000000



5.3 组合逻辑控制器

5.Beq rs,rt,offset ;

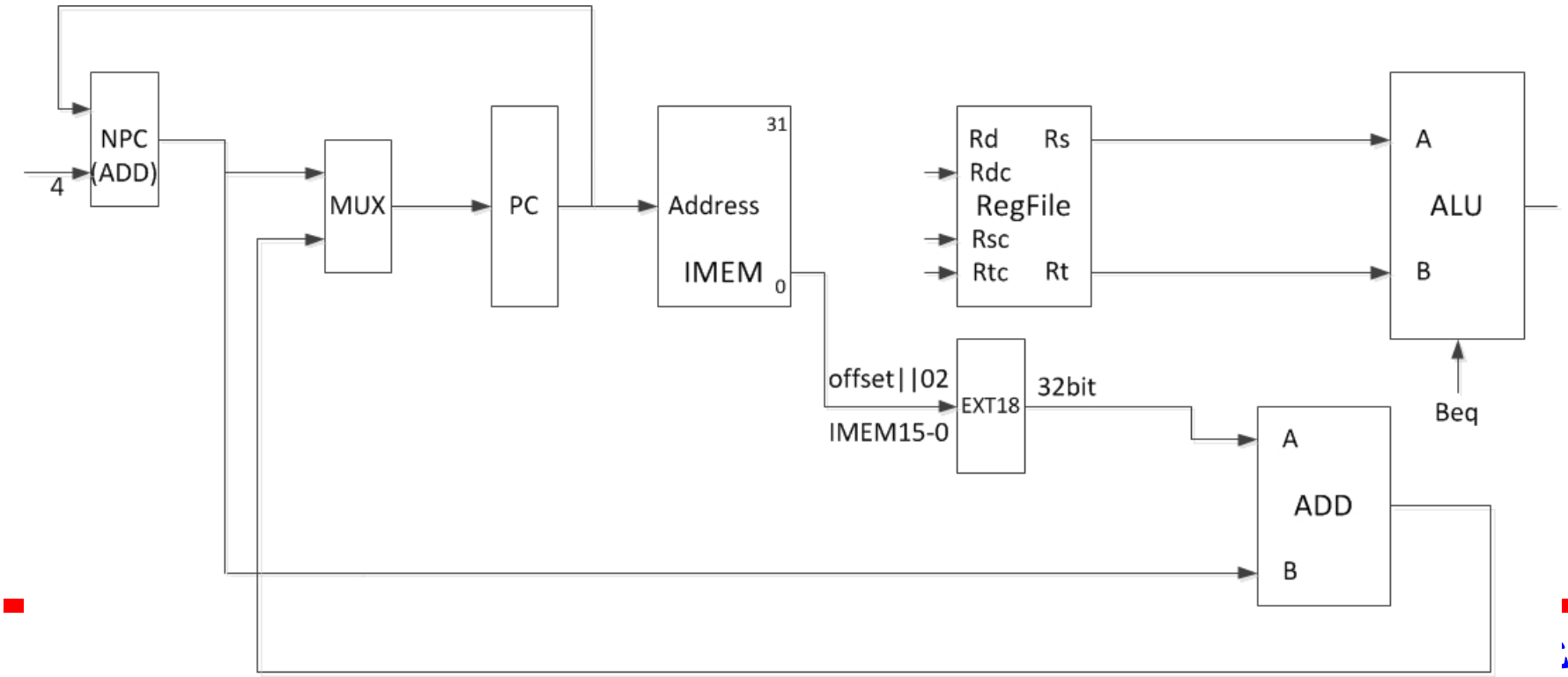
rs=rt,PC←NPC+Sign_ext(offset||0^2)否则PC←NPC(PC+4)

- 所需部件：PC、NPC（完成PC增值）、IMEM、Regfile、ALU、Ext18、ADD
- 部件之间数据输入输出关系如下表：

	PC	NPC	IMEM	RegFile	ALU		Ext16	Ext5	Ext18	ADD	
				Rd	A	B				A	B
Addu	NPC	PC	PC	ALU	Rs	Rt					
Subu	NPC	PC	PC	ALU	Rs	Rt					
Ori	NPC	PC	PC	ALU	Rs	Ext16	imm16				
Sll	NPC	PC	PC	ALU	Ext5	Rt		sa			
Beq	ADD	PC	PC		Rs	Rt			0ffset	NPC	Ext18

	PC	NPC	IMEM	RegFile	ALU		Ext16	Ext5	Ext18	ADD	
				Rd	A	B				A	B
Addu	NPC	PC	PC	ALU	Rs	Rt					
Subu	NPC										
Ori	NPC										
Sll	NPC	PC	PC	ALU	Ext5	Rt		sa			
Beq	ADD	PC	PC		Rs	Rt			0ffset	Ext18	NPC

31	26	25	21	20	16	15	0
Beq(000100)		rs		rt		offset	



5.3 组合逻辑控制器

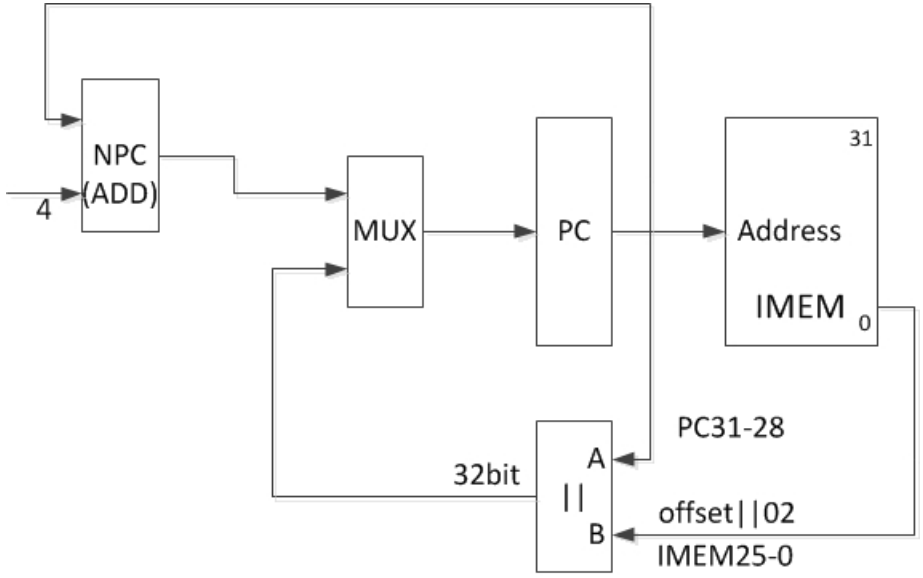
6.J target ;

$PC \leftarrow PC_{31-28} || instr\_index || 0^2$ ， $PC \leftarrow NPC(PC+4)$

- 所需部件：PC、NPC（完成PC增值）、IMEM、||
- 部件之间数据输入输出关系如下表：

	PC	NPC	IMEM	RegFile	ALU		Ext16	Ext5	Ext18	ADD		||	
				Rd	A	B				A	B	A	B
Addu	NPC	PC	PC	ALU	Rs	Rt							
Subu	NPC	PC	PC	ALU	Rs	Rt							
Ori	NPC	PC	PC	ALU	Rs	Ext16	imm16						
Sll	NPC	PC	PC	ALU	Ext5	Rt		sa					
Beq	ADD	PC	PC		Rs	Rt			0ffset	Ext18	NPC		
J	||	PC	PC									PC 31-28	IMEM 25-0

	PC	NPC	IMEM	RegFile	ALU		Ext16	Ext5	Ext18	ADD		||	
				Rd	A	B				A	B	A	B
Addu	NPC	PC	PC	ALU	Rs	Rt							
Subu	NPC	PC	PC	ALU	Rs	Rt							
Ori	NPC	PC	31	26	25						0		
Sll	NPC	PC	J(000010)		instr_index								
Beq	ADD	PC	PC		Rs	Rt			0ffset	Ext18	NPC		
J	||	PC	PC									PC 31-28	IMEM 25-0



5.3 组合逻辑控制器

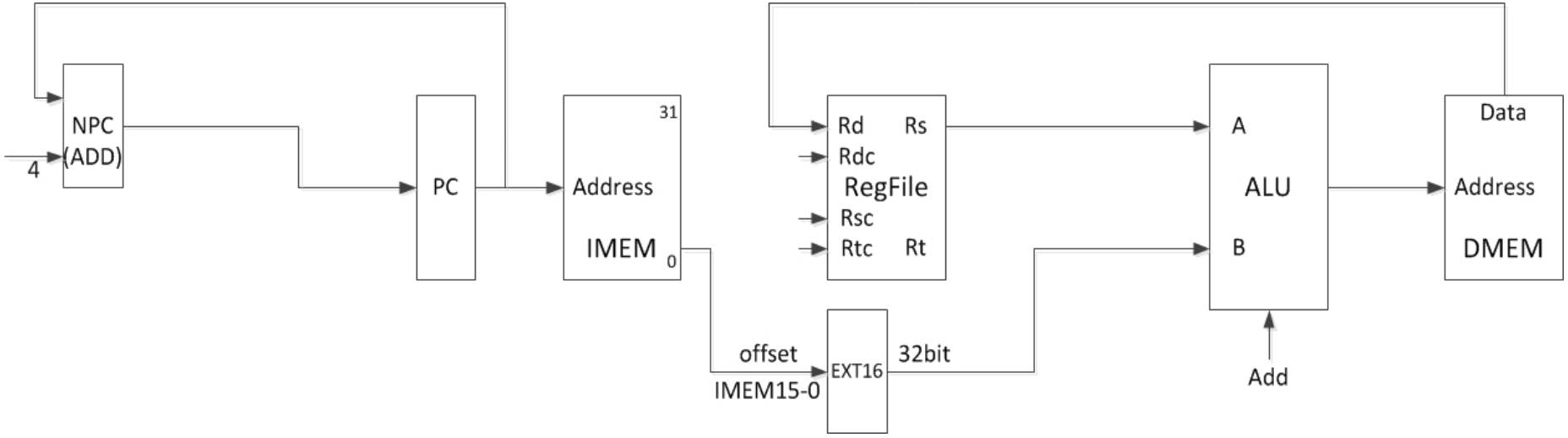
7.Lw rt,offset(base) ;rt←[rs+Sign_ext_offset]
PC←NPC(PC+4)

- 所需部件：PC、NPC、IMEM、 Regfile 、ALU、Ext16、DMEM
- 部件之间数据输入输出关系如下表：

	PC	NPC	IMEM	RegFile	ALU		Ext16	Ext5	Ext18	ADD		||		DMEM	
				Rd	A	B				A	B	A	B	Addr	Data
Addu	NPC	PC	PC	ALU	Rs	Rt									
Subu	NPC	PC	PC	ALU	Rs	Rt									
Ori	NPC	PC	PC	ALU	Rs	Ext16	imm16								
Sll	NPC	PC	PC	ALU	Ext5	Rt		sa							
Beq	ADD	PC	PC		Rs	Rt			0ffset	Ext18	NPC				
J	||	PC	PC									PC 31-28	IMEM 25-0		
Lw	NPC	PC	PC	DMEM （Data）	Rs	Ext16	offset							ALU	

	PC	NPC	IMEM	RegFile	ALU		Ext16	Ext5	Ext18	ADD		||		DMEM	
				Rd	A	B				A	B	A	B	Addr	Data
Addu	NPC	PC	PC	ALU	Rs	Rt									
Subu	NPC	PC	PC	ALU	Rs	Rt									
Ori	NPC	PC	PC	ALU	Rs	Ext16	imm16								
Sll	NPC	PC													
Beq	ADD	PC													
J	||	PC	PC									PC 31-28	IMEM 25-0		
Lw	NPC	PC	PC	DMEM （Data）	Rs	Ext16	offset							ALU	

31 26	25 21	20 16	15 0
Lw(100011)	base	rt	offset



5.3 组合逻辑控制器

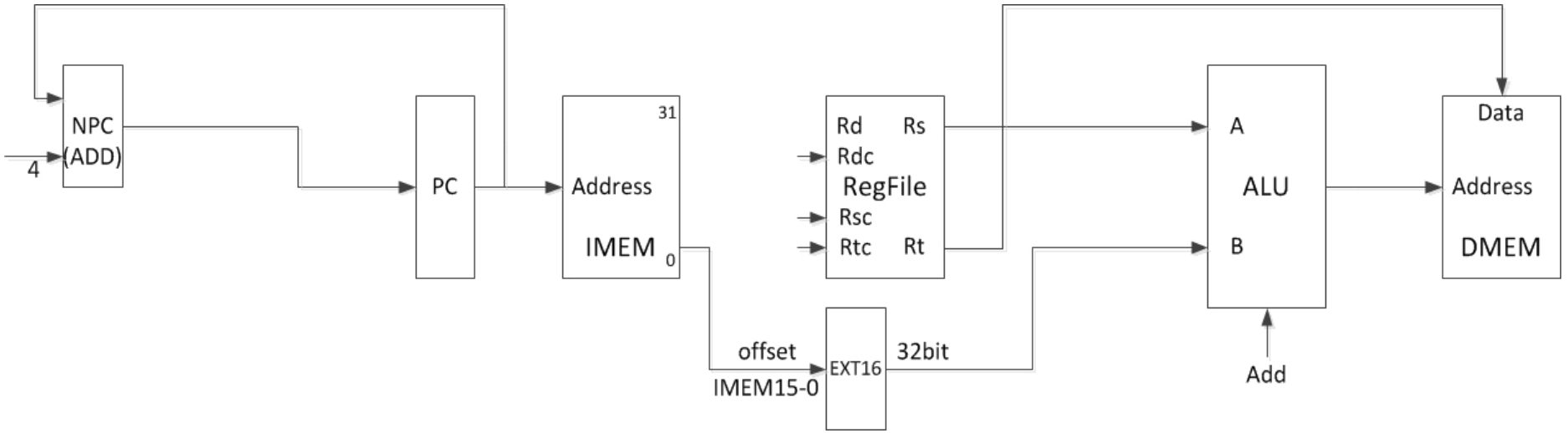
8.Sw rt,offset(base)；

[base+Sign_ext_offset]←rt,PC←NPC(PC+4)

- 所需部件：PC、NPC（完成PC增值）、IMEM、ALU、Ext16、DMEM
- 部件之间数据输入输出关系如下表：

	PC	NPC	IMEM	RegFile	ALU		Ext16	Ext5	Ext18	ADD		||		DMEM	
				Rd	A	B				A	B	A	B	Addr	Data
Addu	NPC	PC	PC	ALU	Rs	Rt									
Subu	NPC	PC	PC	ALU	Rs	Rt									
Ori	NPC	PC	PC	ALU	Rs	Ext16	imm16								
Sll	NPC	PC	PC	ALU	Ext5	Rt		sa							
Beq	ADD	PC	PC		Rs	Rt			0ffset	Ext18	NPC				
J	||	PC	PC									PC 31-28	IMEM 25-0		
Lw	NPC	PC	PC	DMEM（Data）	Rs	Ext16	offset							ALU	
Sw	NPC	PC	PC		Rs	Ext16	offset							ALU	Rt

	PC	NPC	IMEM	RegFile	ALU		Ext16	Ext5	Ext18	ADD		||		DMEM	
				Rd	A	B				A	B	A	B	Addr	Data
Addu	NPC	PC	PC	ALU	Rs	Rt									
Subu	NPC	PC	PC	ALU	Rs	Rt									
Ori	NPC	PC	PC	ALU	Rs	Ext16	imm16								
Sll	NPC	PC	PC	ALU	Ext5	Rt		sa							
Beq	ADD	PC	31 26 25 21 20 16 15 0												
J	||	PC	Sw(101011)	base	rt	offset							MEM 25-0		
Lw	NPC	PC	PC	DMEM （Data）	Rs	Ext16	offset							ALU	
Sw	NPC	PC	PC		Rs	Ext16	offset							ALU	Rt



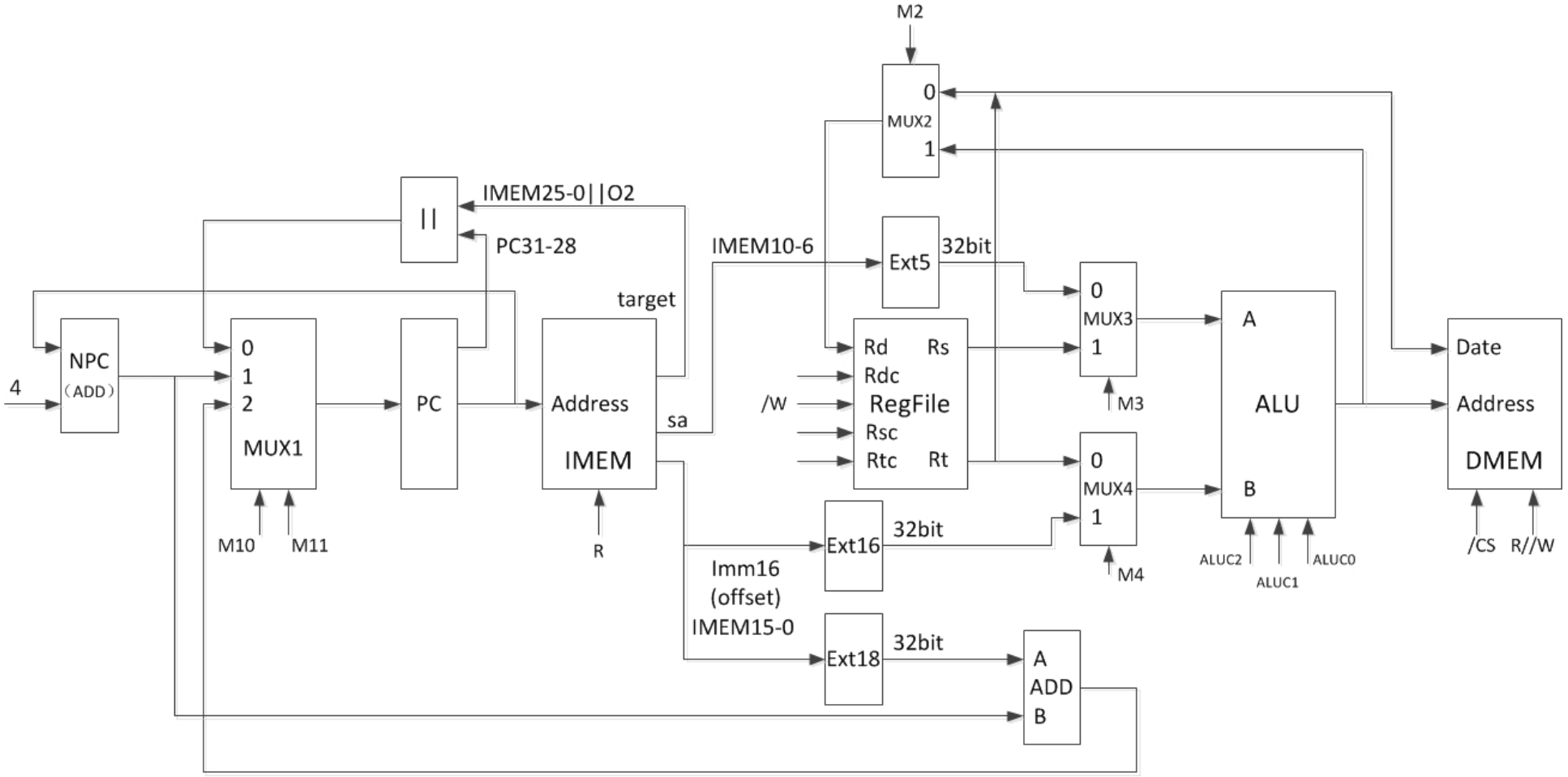
5.3 组合逻辑控制器

百年同济 TONGJI UNIVERSITY

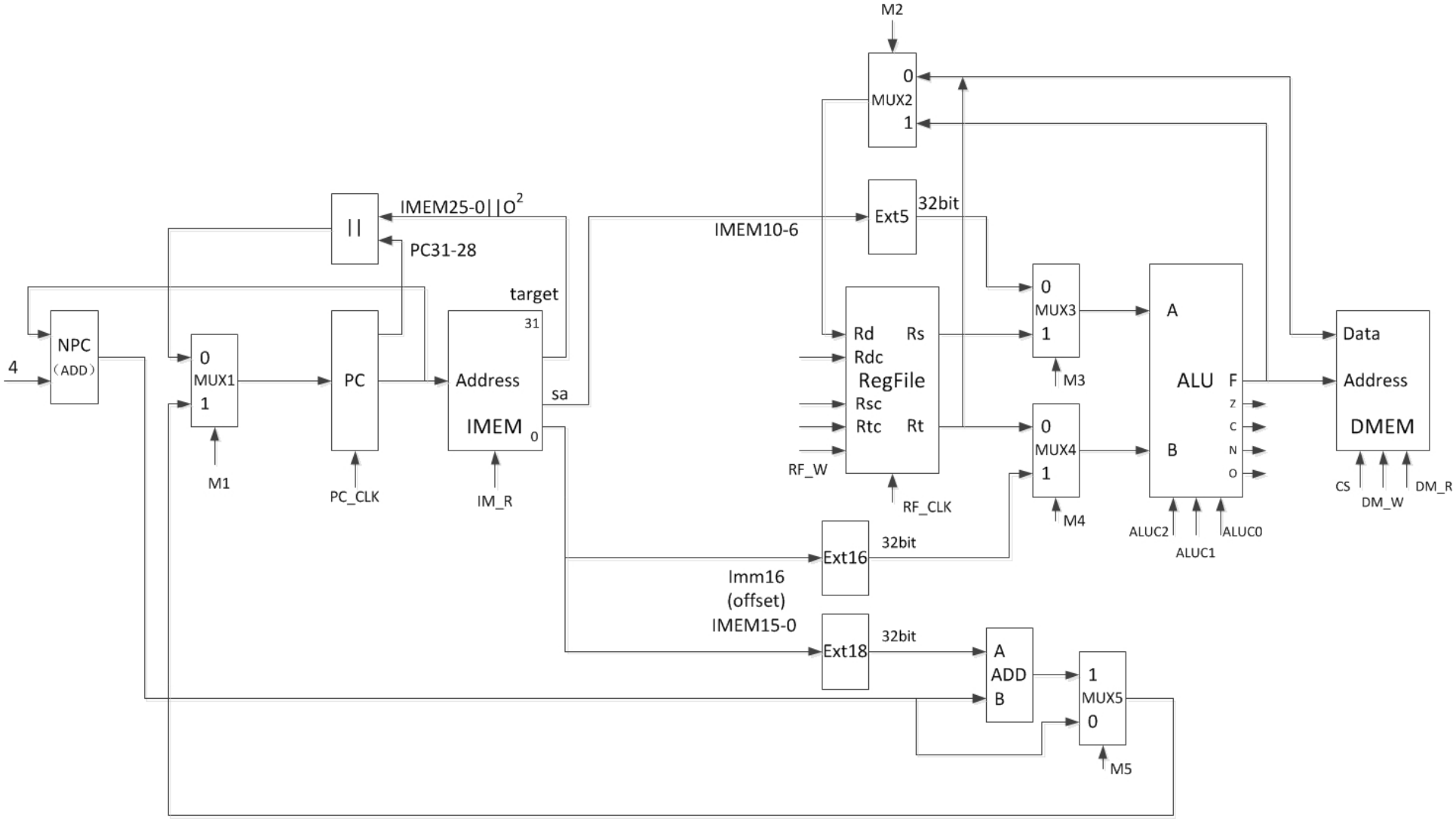
如何构成8条指令的数据通路?

	PC	NPC	IMEM	RegFile	ALU		Ext16	Ext5	Ext18	ADD		||		DMEM	
				Rd	A	B				A	B	A	B	Addr	Data
Addu	NPC	PC	PC	ALU	Rs	Rt									
Subu	NPC	PC	PC	ALU	Rs	Rt									
Ori	NPC	PC	PC	ALU	Rs	Ext16	Imm16								
Sll	NPC	PC	PC	ALU	Ext5	Rt		sa							
Beq	ADD	PC	PC		Rs	Rt			0ffset	Ext18	NPC				
J	||	PC	PC									PC 31-28	IMEM 25-0		
Lw	NPC	PC	PC	DMEM （Data）	Rs	Ext16	offset							ALU	
Sw	NPC	PC	PC		Rs	Ext16	offset							ALU	Rt

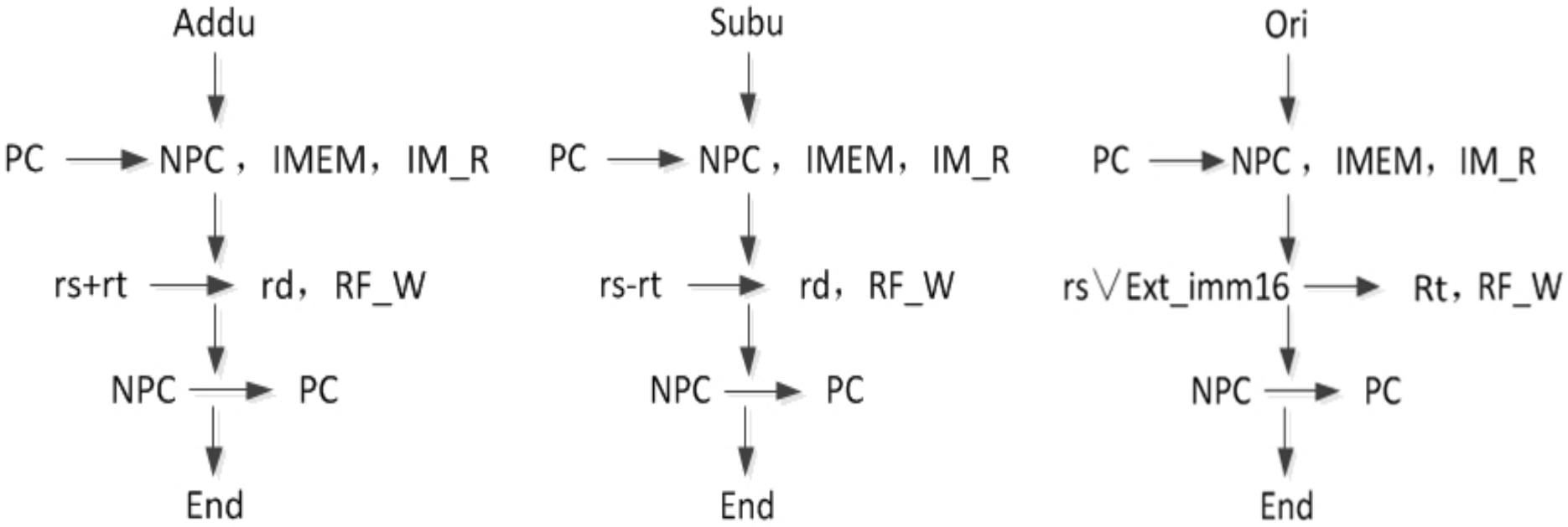
5.3 组合逻辑控制器


M2
0
MUX2
1
IMEM25-0||O2
||
PC31-28
IMEM10-6
Ext5
32bit
target
0
MUX3
1
M3
A
ALU
B
Date
Address
DMEM
/CS
R//W
ALUC2
ALUC1
ALUC0
NPC
(ADD)
4
0
1
2
MUX1
M10
M11
PC
Address
IMEM
R
sa
/W
Rd
Rs
Rdc
RegFile
Rsc
Rtc
Rt
0
MUX4
1
M4
Imm16
(offset)
IMEM15-0
Ext16
32bit
Ext18
32bit
A
ADD
B

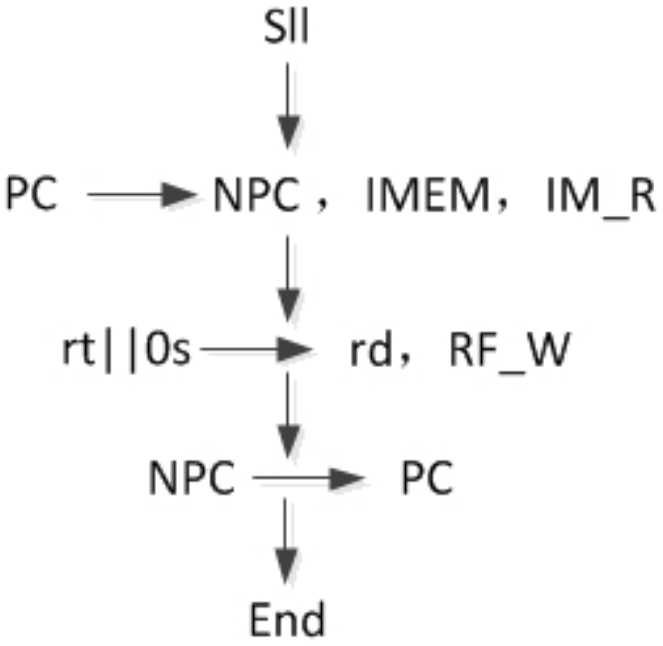
5.3 组合逻辑控制器

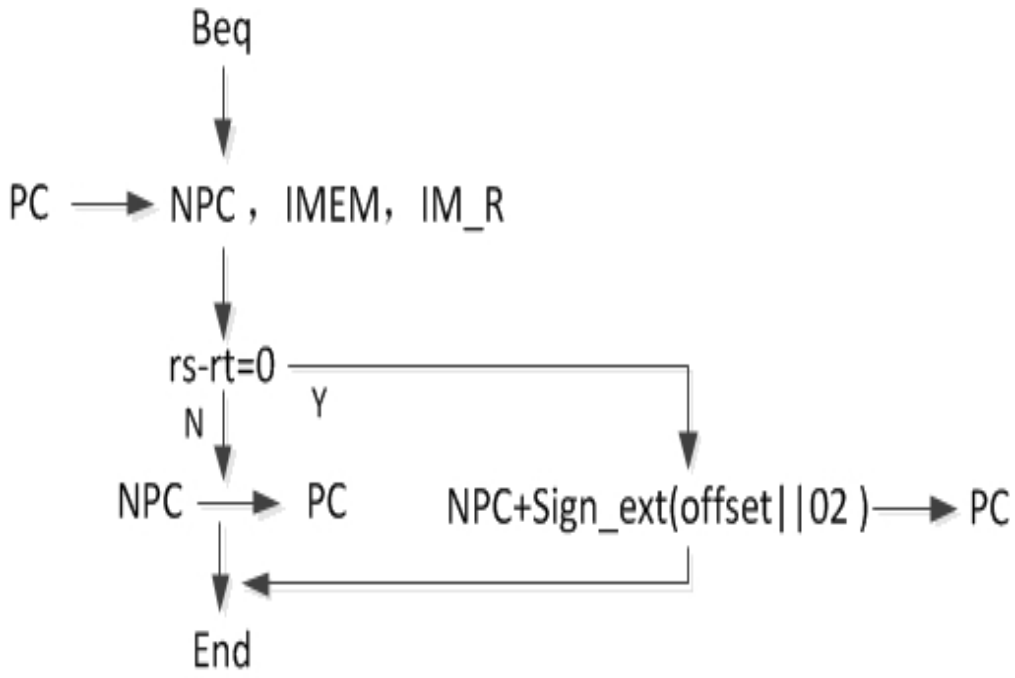

M2
0
MUX2
1
IMEM25-0||O2
||
PC31-28
target
IMEM10-6
Ext5
32bit
NPC
(ADD)
4
0
MUX1
1
M1
PC
PC_CLK
31
Address
IMEM
0
IM_R
sa
Rd
Rs
Rdc
RegFile
Rsc
Rtc
Rt
RF_W
RF_CLK
0
MUX3
1
M3
0
MUX4
1
M4
A
ALU
F
Z
C
N
O
B
ALUC2
ALUC1
ALUC0
Data
Address
DMEM
CS
DM_W
DM_R
Ext16
32bit
Imm16
(offset)
IMEM15-0
Ext18
32bit
A
ADD
B
1
MUX5
0
M5

2.绘制指令流程图

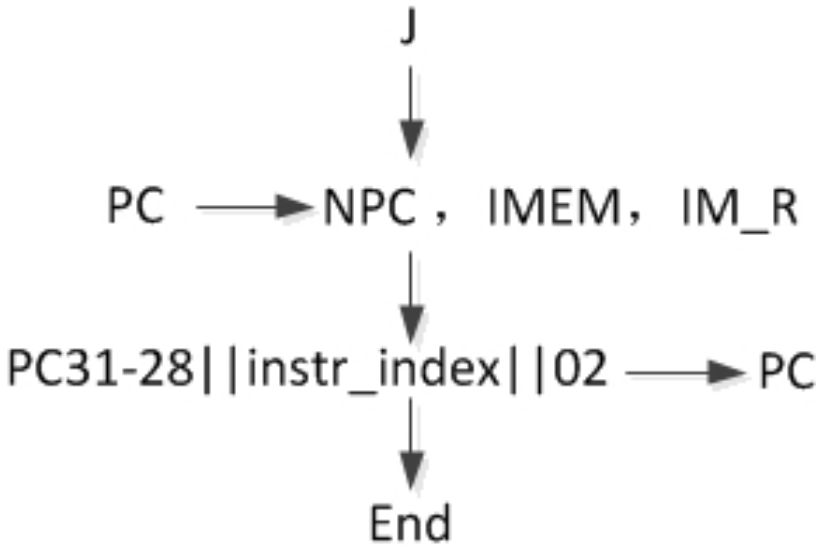

Addu
PC
NPC，IMEM，IM_R
rs+rt
rd，RF_W
NPC
PC
End
Subu
PC
NPC，IMEM，IM_R
rs-rt
rd，RF_W
NPC
PC
End
Ori
PC
NPC，IMEM，IM_R
rs∨Ext_imm16
Rt，RF_W
NPC
PC
End

5.3 组合逻辑控制器

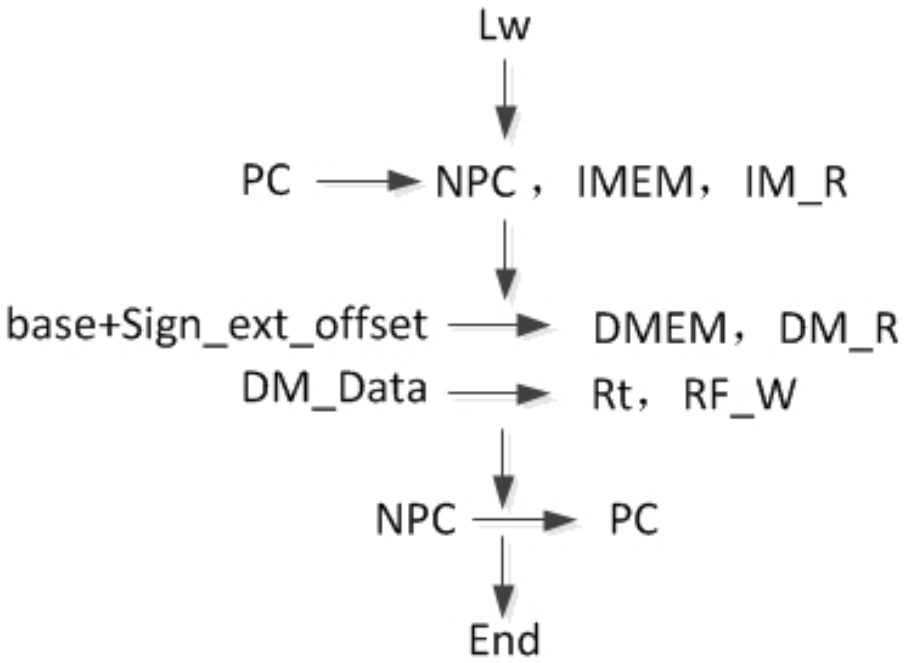

Sll
PC
NPC，IMEM，IM_R
rt||0s
rd，RF_W
NPC
PC
End


Beq
PC
NPC，IMEM，IM_R
rs-rt=0
Y
N
NPC
PC
NPC+Sign_ext(offset||02)
PC
End

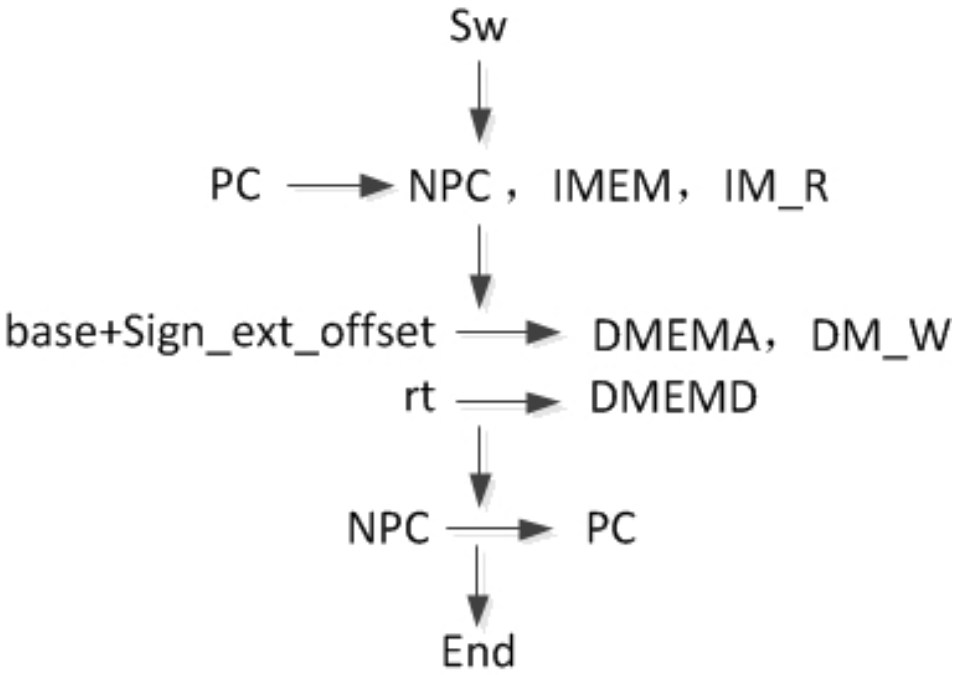
5.3 组合逻辑控制器



5.3 组合逻辑控制器


Lw
PC
NPC，IMEM，IM_R
base+Sign_ext_offset
DMEM，DM_R
DM_Data
Rt，RF_W
NPC
PC
End

5.3 组合逻辑控制器


Sw
PC
NPC，IMEM，IM_R
base+Sign_ext_offset
DMEMA，DM_W
rt
DMEMD
NPC
PC
End

5.3 组合逻辑控制器

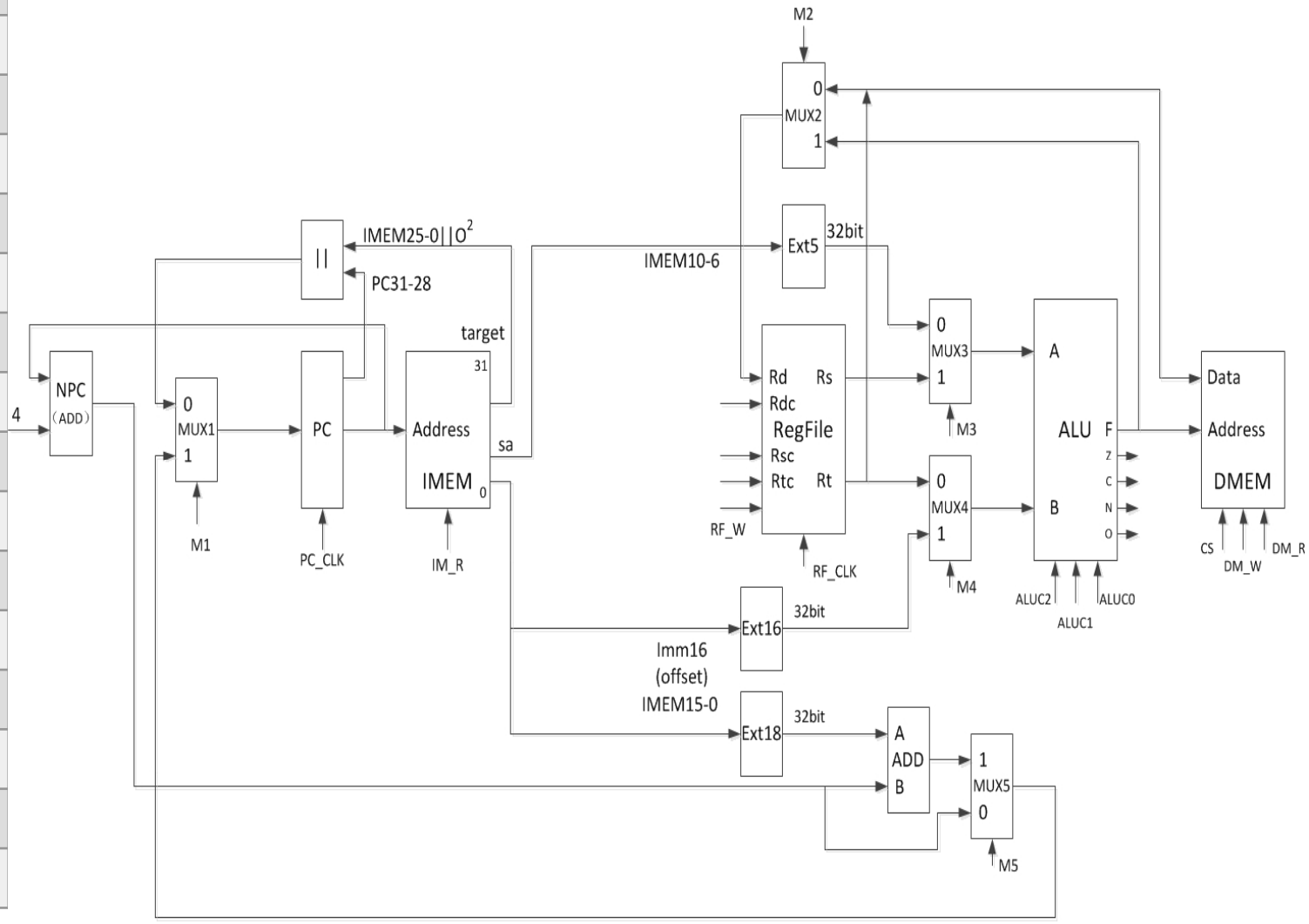
3.编排指令操作时间表

依据各条机器指令的操作流程图将每条指令执行所需的控制信号填入下表。

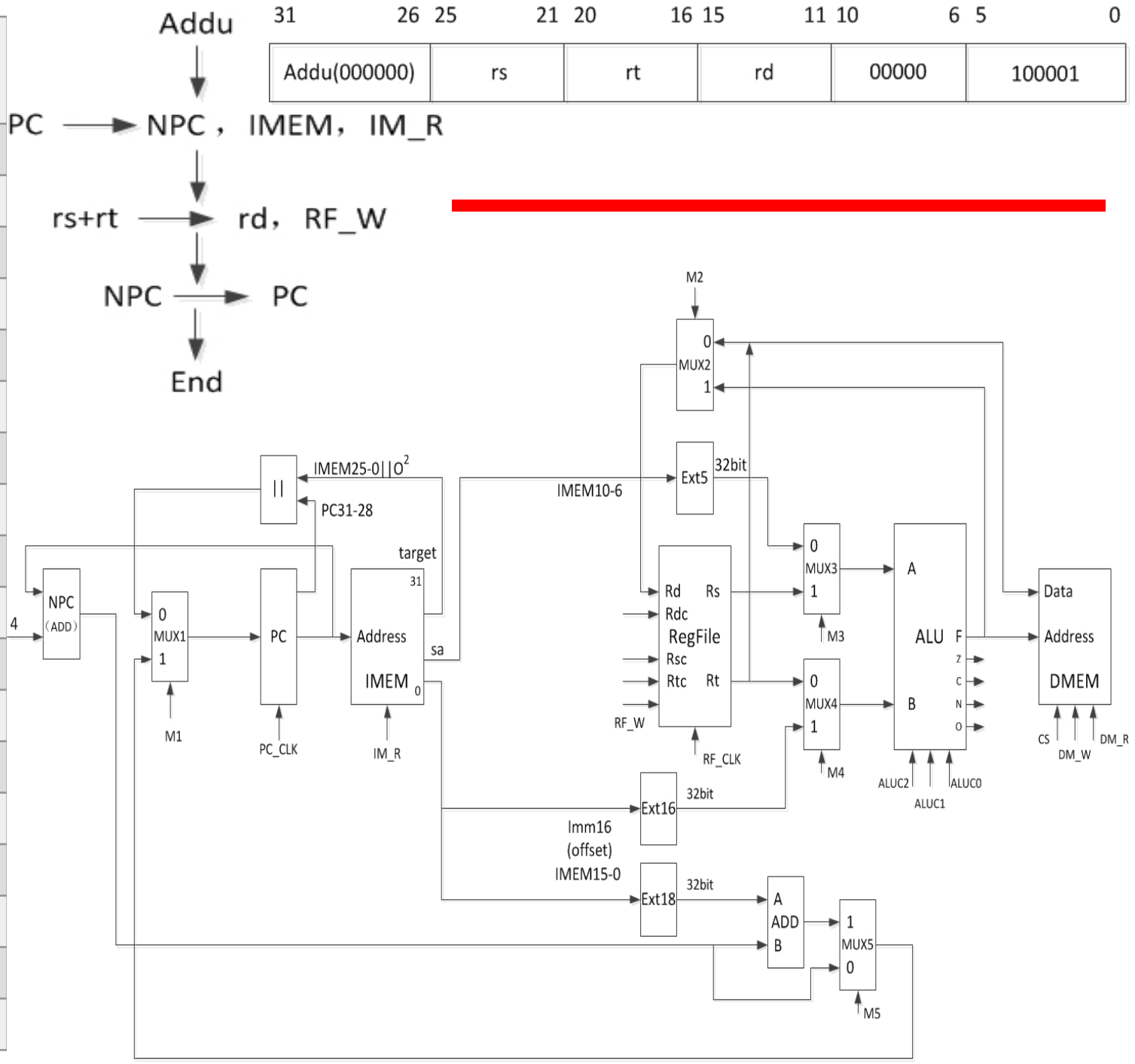
假设：ALU控制如下表。

	ALUC2	ALUC1	ALUC0
Add	0	0	0
Sub	0	0	1
Ori	0	1	0
Sll	0	1	1

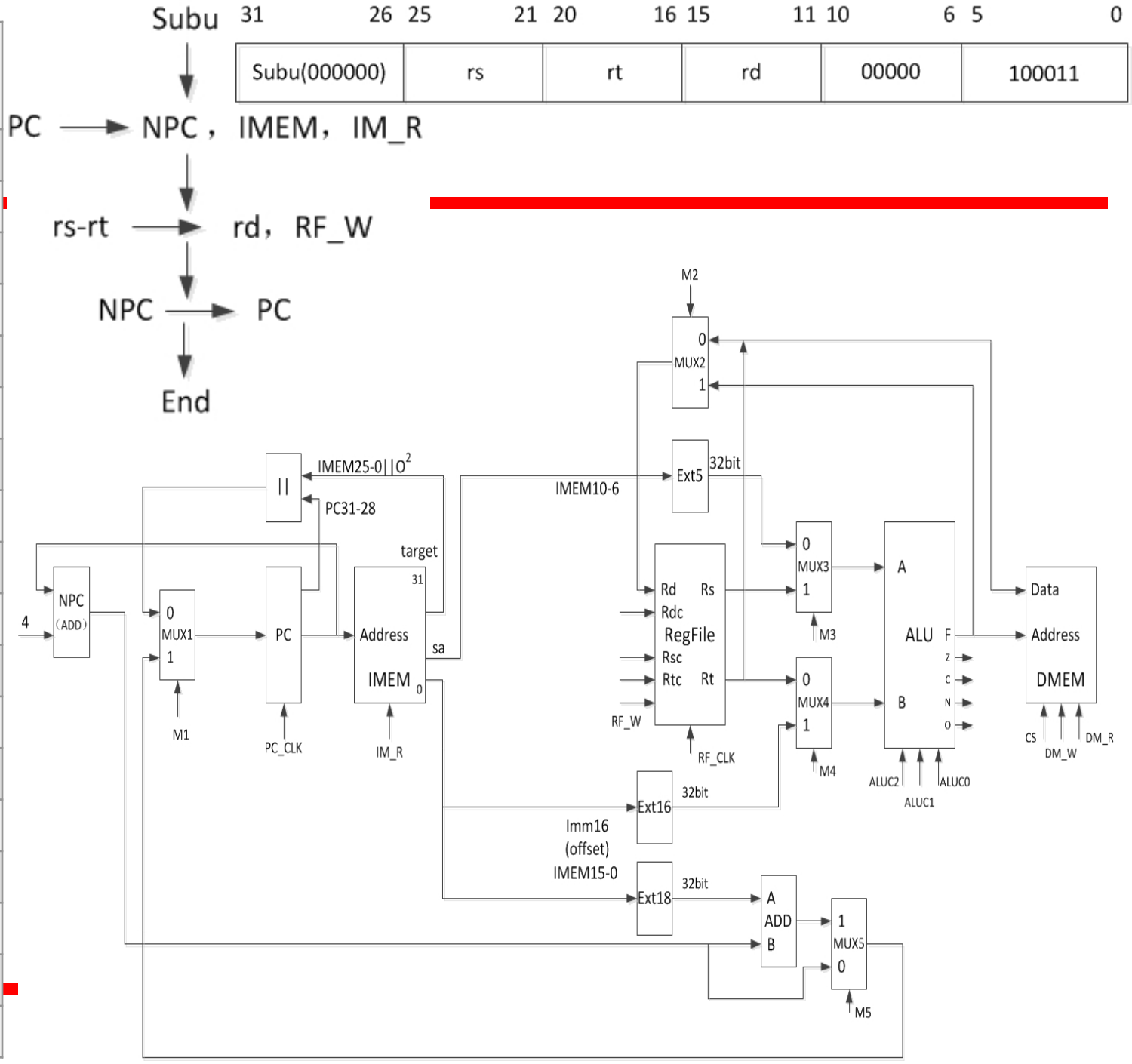
控制信号（微操作）	
PC_CLK	
IM_R	
Rsc4-0	
Rtc4-0	
M3	
M4	
ALUC2	
ALUC1	
ALUC0	
M2	
Rdc4-0	
RF_W	
RF_CLK	
M5	
M1	
DM_CS	
DM_R	
DM_W	



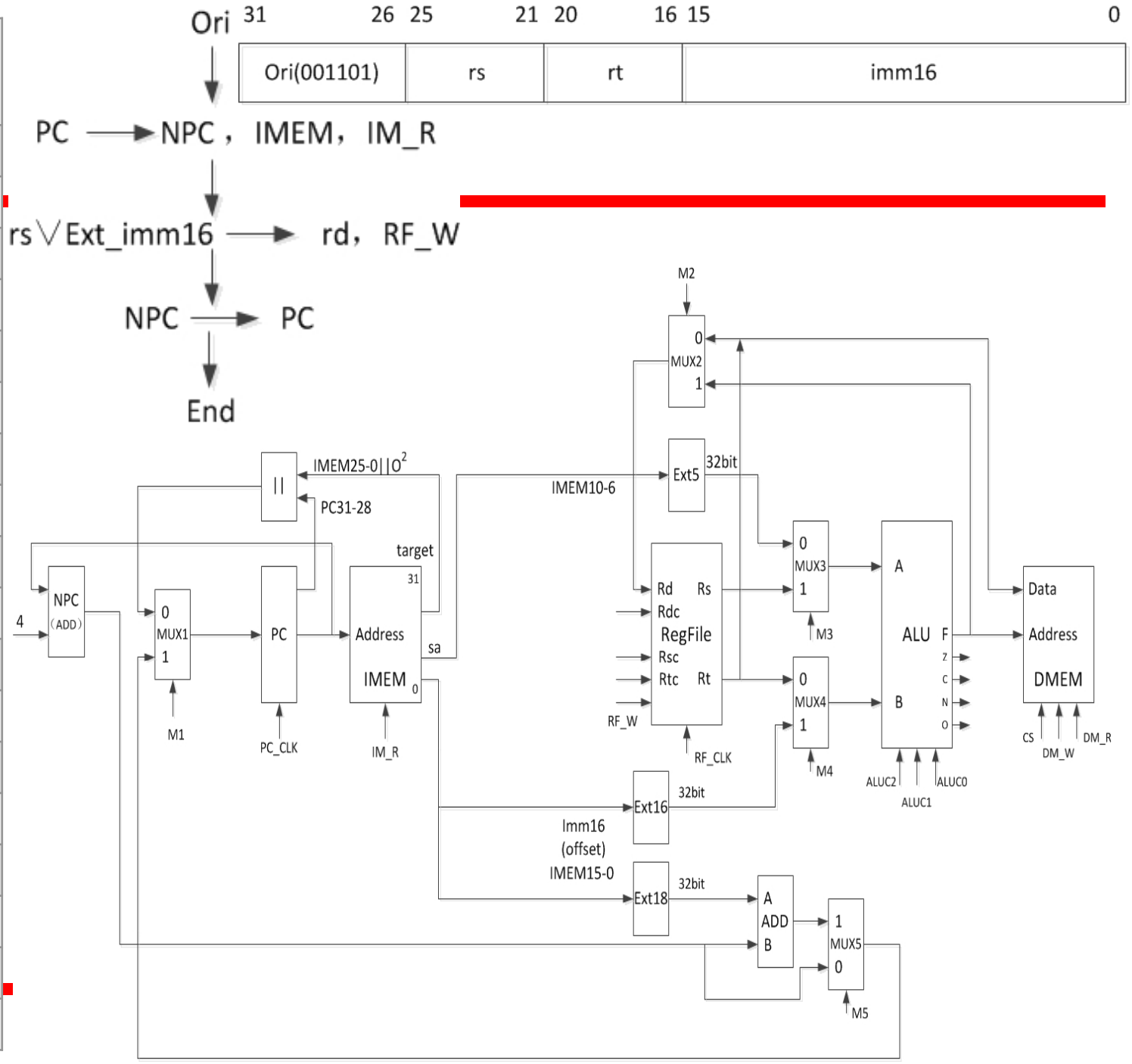
控制信号（微操作）	Addu
PC_CLK	1
IM_R	1
Rsc4-0	IM25-21
Rtc4-0	IM20-16
M3	1
M4	0
ALUC2	0
ALUC1	0
ALUC0	0
M2	1
Rdc4-0	IM15-11
RF_W	1
RF_CLK	1
M5	0
M1	1
DM_CS	0
DM_R	0
DM_W	0



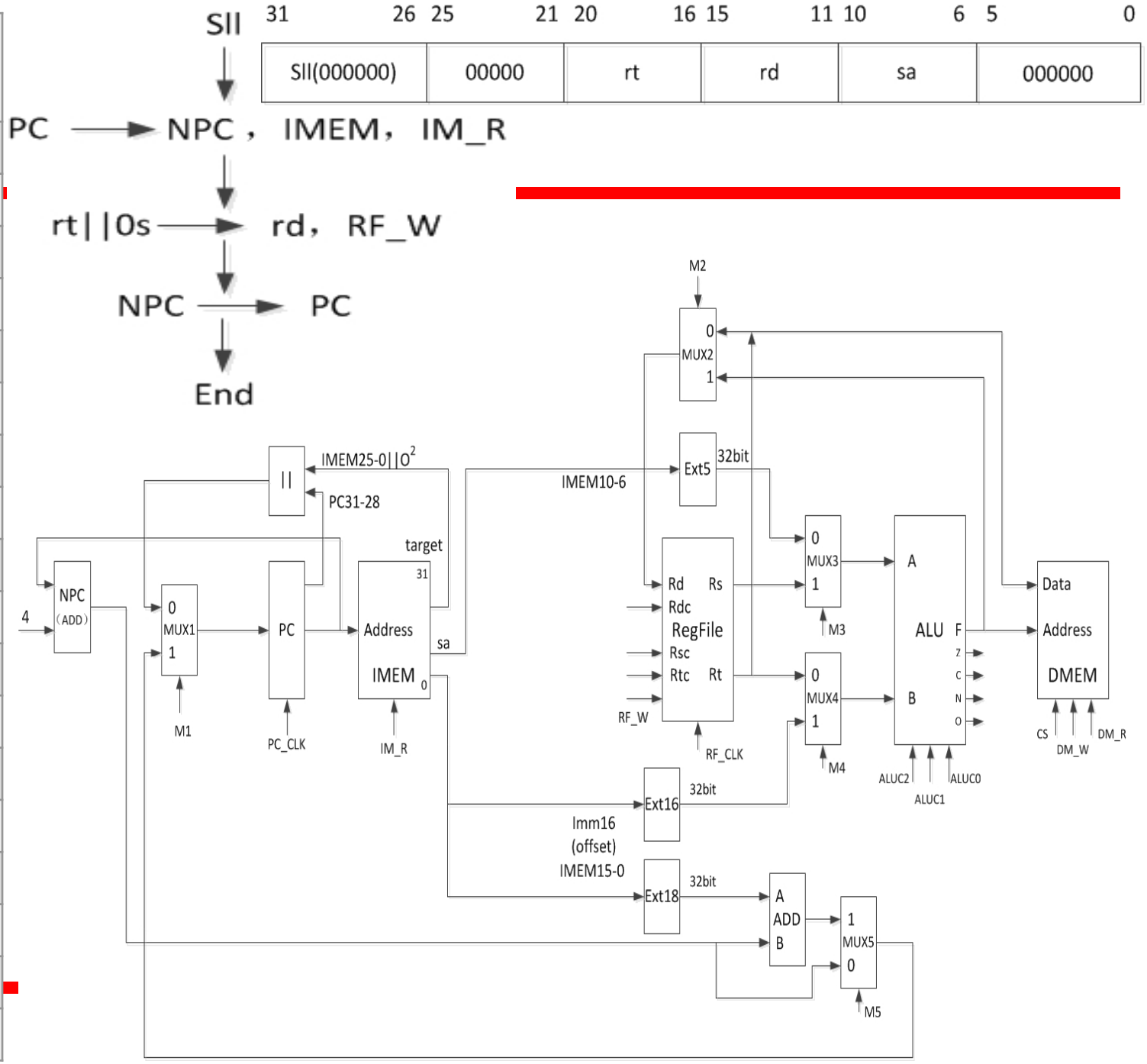
控制信号（微操作）	Subu
PC_CLK	1
IM_R	1
Rsc4-0	IM25-21
Rtc4-0	IM20-16
M3	1
M4	0
ALUC2	0
ALUC1	0
ALUC0	1
M2	1
Rdc4-0	IM15-11
RF_W	1
RF_CLK	1
M5	0
M1	1
DM_CS	0
DM_R	0
DM_W	0



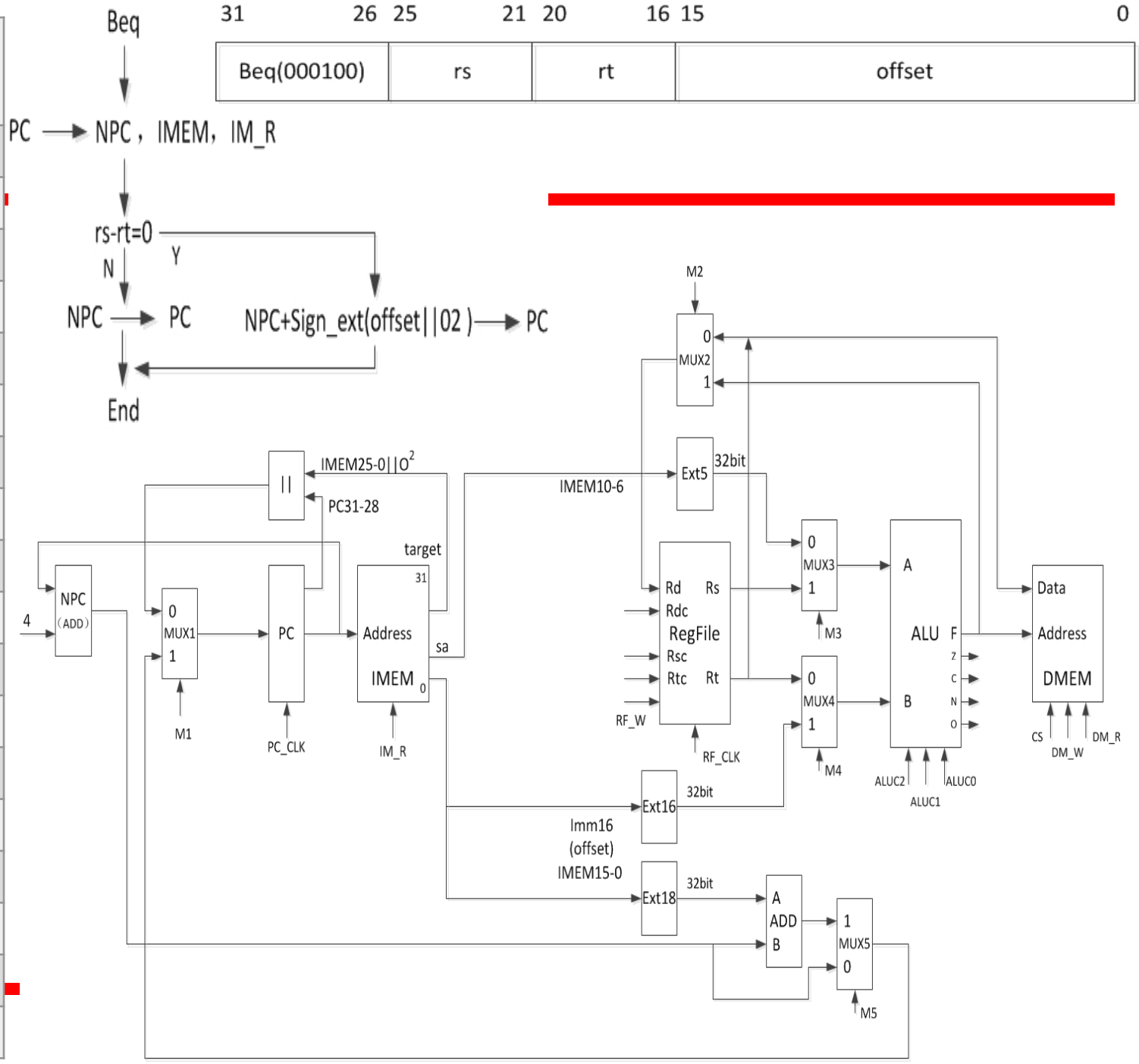
控制信号（微操作）	Ori
PC_CLK	1
IM_R	1
Rsc4-0	IM25-21
Rtc4-0	
M3	1
M4	1
ALUC2	0
ALUC1	1
ALUC0	0
M2	1
Rdc4-0	IM20-16
RF_W	1
RF_CLK	1
M5	0
M1	1
DM_CS	0
DM_R	0
DM_W	0



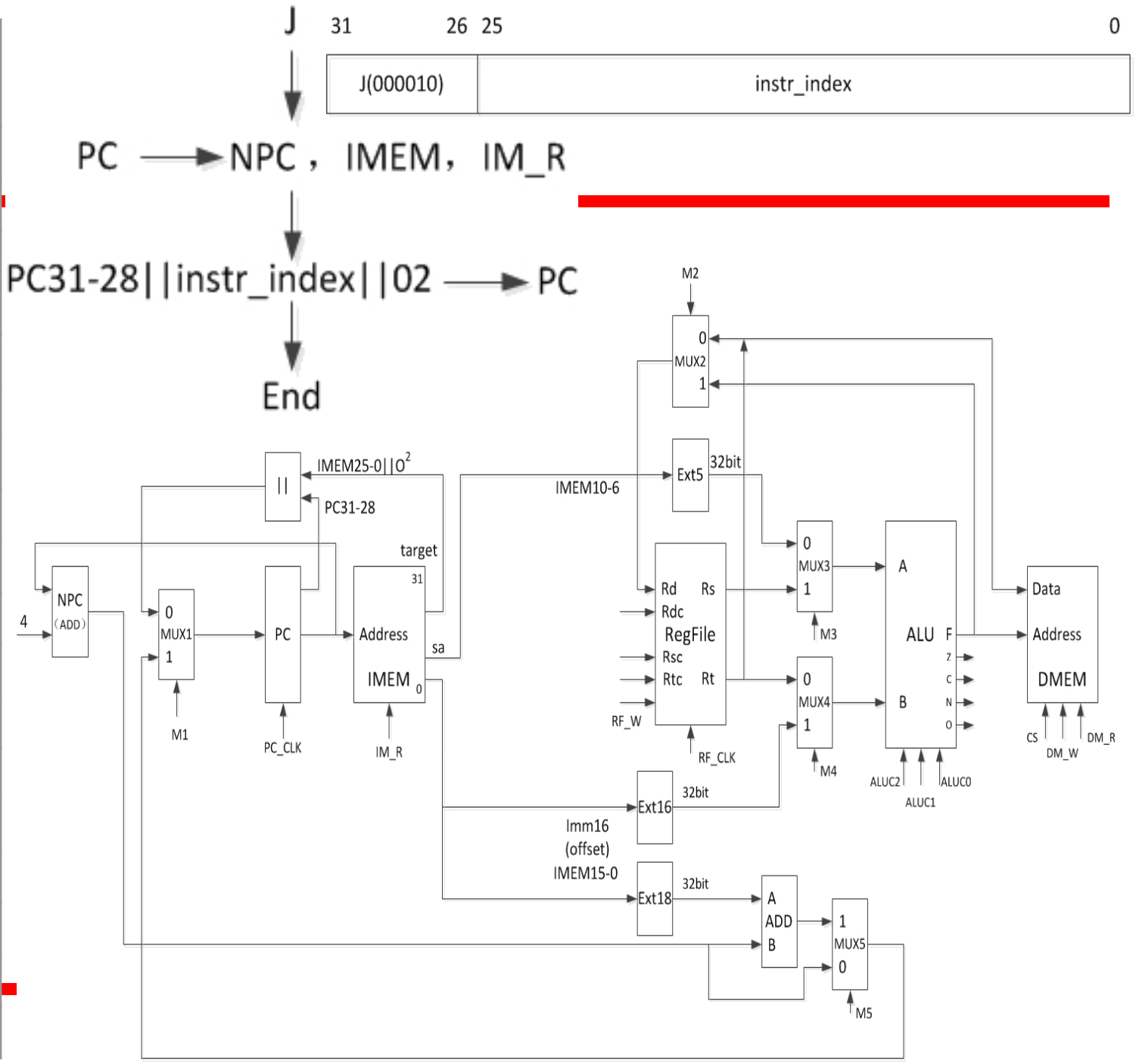
控制信号（微操作）	SII
PC_CLK	1
IM_R	1
Rsc4-0	
Rtc4-0	IM20-16
M3	0
M4	0
ALUC2	0
ALUC1	1
ALUC0	1
M2	1
Rdc4-0	IM15-11
RF_W	1
RF_CLK	1
M5	0
M1	1
DM_CS	0
DM_R	0
DM_W	0



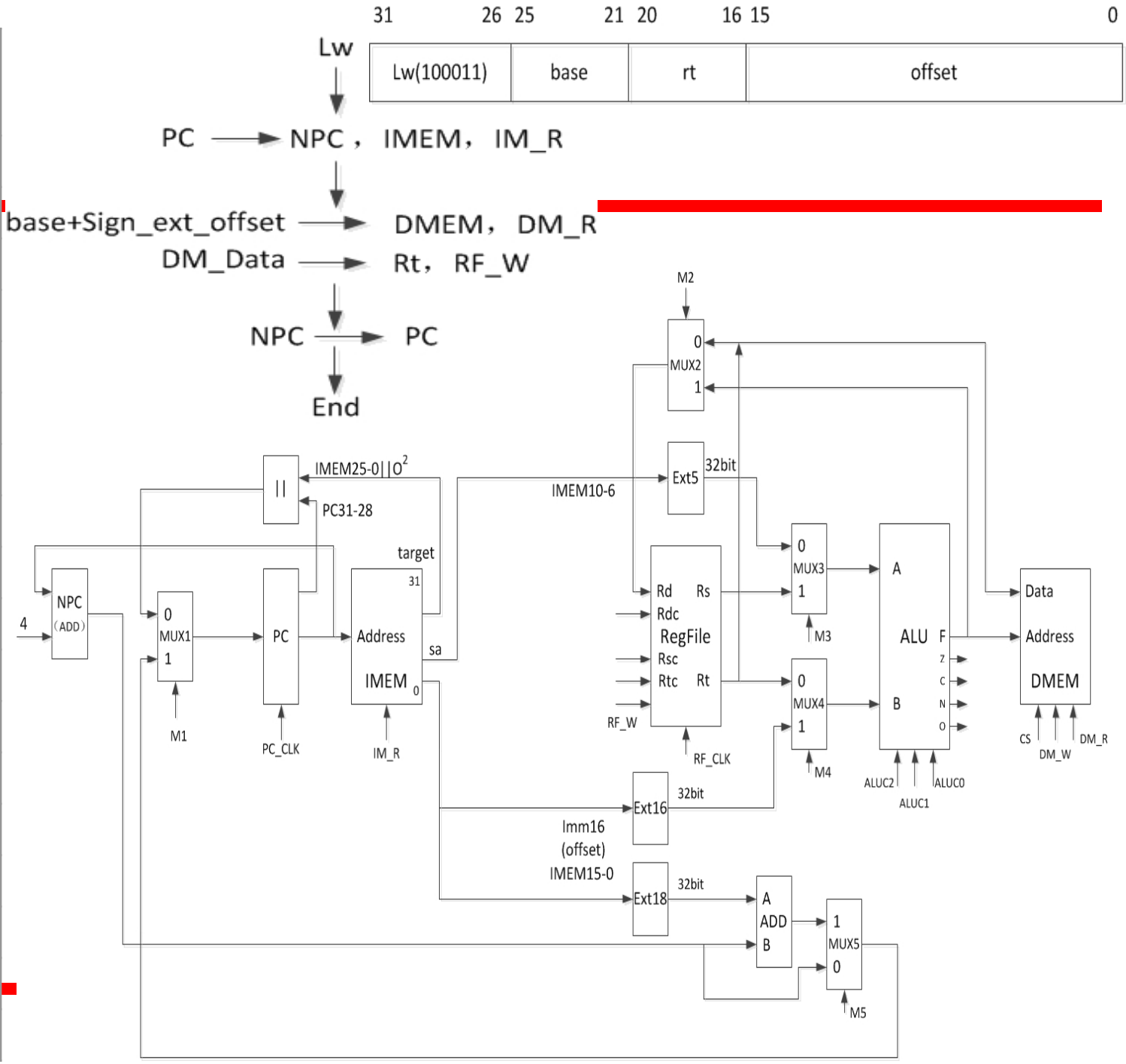
控制信号（微操作）	Beq（Z=1）
PC_CLK	1
IM_R	1
Rsc4-0	IM25-21
Rtc4-0	IM20-16
M3	1
M4	0
ALUC2	0
ALUC1	0
ALUC0	1
M2	1
Rdc4-0	
RF_W	0
RF_CLK	0
M5	1
M1	1
DM_CS	0
DM_R	0
DM_W	0



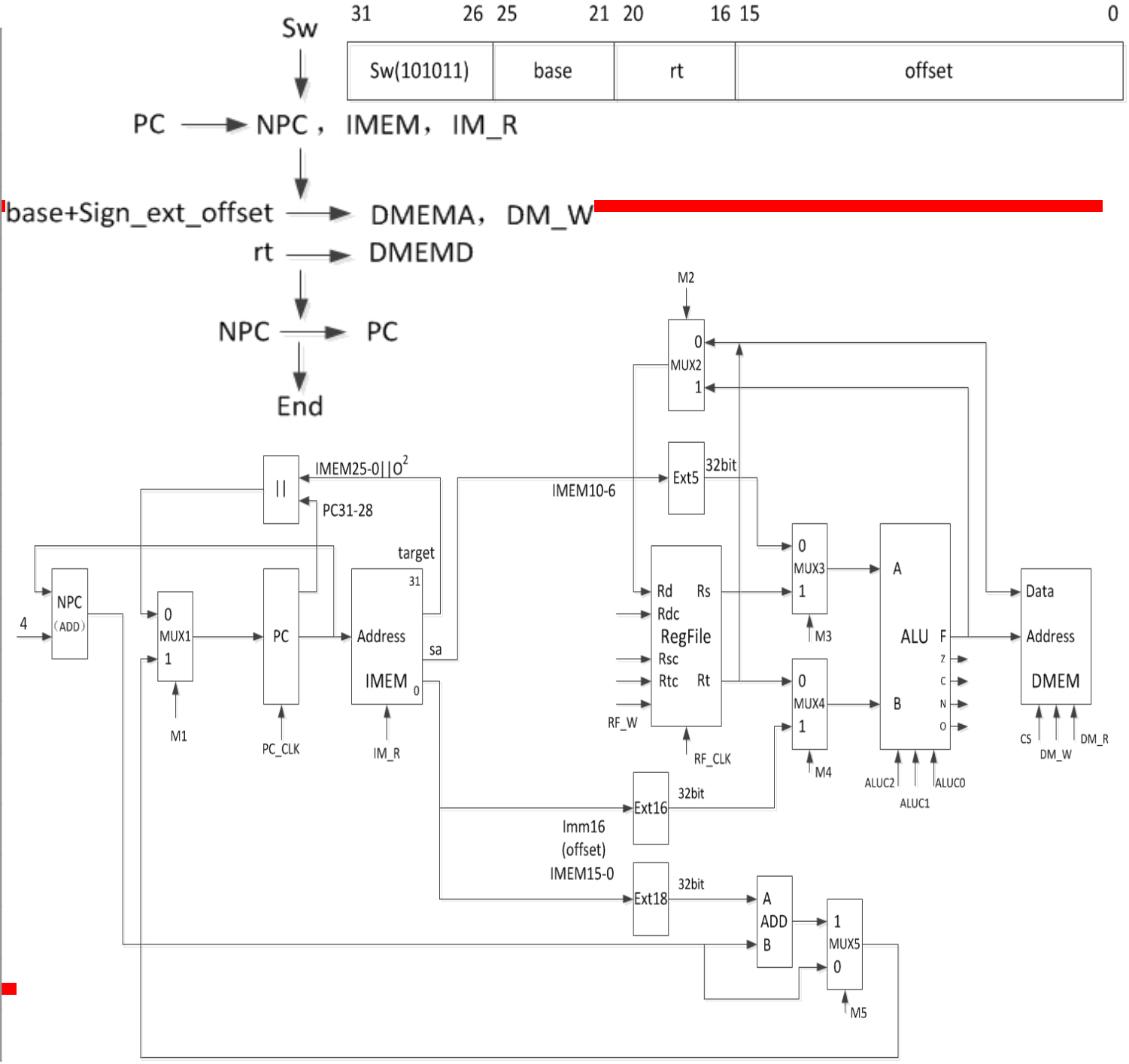
控制信号（微操作）	J
PC_CLK	1
IM_R	1
Rsc4-0	
Rtc4-0	
M3	1
M4	0
ALUC2	0
ALUC1	0
ALUC0	0
M2	1
Rdc4-0	
RF_W	0
RF_CLK	0
M5	0
M1	0
DM_CS	0
DM_R	0
DM_W	0



控制信号（微操作）	Lw
PC_CLK	1
IM_R	1
Rsc4-0	IM25-21
Rtc4-0	
M3	1
M4	1
ALUC2	0
ALUC1	0
ALUC0	0
M2	0
Rdc4-0	IM20-16
RF_W	1
RF_CLK	1
M5	0
M1	1
DM_CS	1
DM_R	1
DM_W	0



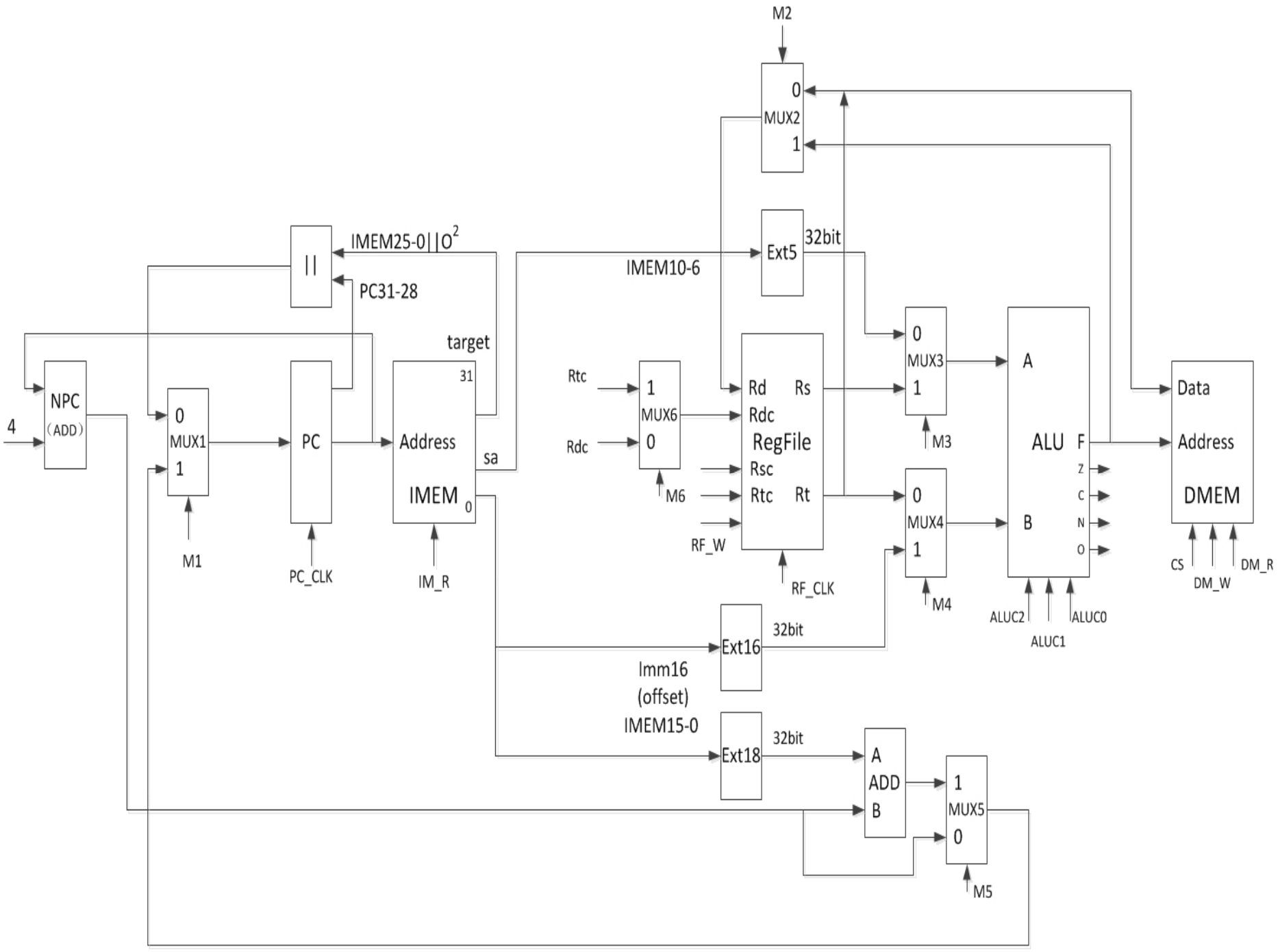
控制信号（微操作）	Sw
PC_CLK	1
IM_R	1
Rsc4-0	IM25-21
Rtc4-0	IM20-16
M3	1
M4	1
ALUC2	0
ALUC1	0
ALUC0	0
M2	1
Rdc4-0	
RF_W	0
RF_CLK	0
M5	0
M1	1
DM_CS	1
DM_R	0
DM_W	1



控制信号（微操作）	Addu	Subu	Ori	Sll	Beq（Z=1）	J	Lw	Sw
PC_CLK	1	1	1	1	1	1	1	1
IM_R	1	1	1	1	1	1	1	1
Rsc4-0	IM25-21	IM25-21	IM25-21		IM25-21		IM25-21	IM25-21
Rtc4-0	IM20-16	IM20-16		IM20-16	IM20-16			IM20-16
M3	1	1	1	0	1	1	1	1
M4	0	0	1	0	0	0	1	1
ALUC2	0	0	0	0	0	0	0	0
ALUC1	0	0	1	1	0	0	0	0
ALUC0	0	1	0	1	1	0	0	0
M2	1	1	1	1	1	1	0	1
Rdc4-0	IM15-11	IM15-11	IM20-16	IM15-11	IM15-11		IM20-16	
RF_W	1	1	1	1	0	0	1	0
RF_CLK	1	1	1	1	0	0	1	0
M5	0	0	0	0	1	0	0	0
M1	1	1	1	1	1	0	1	1
DM_CS	0	0	0	0	0	0	1	1
DM_R	0	0	0	0	0	0	1	0
DM_W	0	0	0	0	0	0	0	1

5.3 组合逻辑控制器

注意：从表中可以看出，Rdc的输入有两个来源，IM15-11和IM20-16，所以，在Rdc的输入端要加一个MUX6。


M2
0
MUX2
1
IMEM25-0||O2
||
PC31-28
IMEM10-6
Ext5
32bit
target
NPC
(ADD)
4
0
MUX1
1
M1
PC
PC_CLK
Address
IMEM
31
0
sa
IM_R
Rtc
1
MUX6
0
M6
Rdc
Rd
Rs
Rdc
RegFile
Rsc
Rtc
Rt
RF_W
RF_CLK
0
MUX3
1
M3
0
MUX4
1
M4
A
ALU
F
Z
C
N
O
B
ALUC2
ALUC1
ALUC0
Data
Address
DMEM
CS
DM_W
DM_R
Imm16
(offset)
IMEM15-0
Ext16
32bit
Ext18
32bit
A
ADD
B
1
MUX5
0
M5

控制信号（微操作）	Addu	Subu	Ori	Sll	Beq（Z=1）	J	Lw	Sw
PC_CLK	1	1	1	1	1	1	1	1
IM_R	1	1	1	1	1	1	1	1
Rsc4-0	IM25-21	IM25-21	IM25-21		IM25-21		IM25-21	IM25-21
Rtc4-0	IM20-16	IM20-16		IM20-16	IM20-16			IM20-16
M3	1	1	1	0	1	1	1	1
M4	0	0	1	0	0	0	1	1
ALUC2	0	0	0	0	0	0	0	0
ALUC1	0	0	1	1	0	0	0	0
ALUC0	0	1	0	1	1	0	0	0
M2	1	1	1	1	1	1	0	1
Rdc4-0	IM15-11	IM15-11	IM20-16	IM15-11	IM15-11		IM20-16	
RF_W	1	1	1	1	0	0	1	0
RF_CLK	1	1	1	1	0	0	1	0
M5	0	0	0	0	1	0	0	0
M1	1	1	1	1	1	0	1	1
DM_CS	0	0	0	0	0	0	1	1
DM_R	0	0	0	0	0	0	1	0
DM_W	0	0	0	0	0	0	0	1
M6	0	0	1	0	0	0	1	0

5.3 组合逻辑控制器

4.进行微操作综合

按照所有机器指令的操作时间表，把相同的微操作综合起来，得到每个微操作的逻辑表达式。本例共有19个控制信号。

PC_CLK=CLK（┌┘）

IM_R=1

Rsc4-0=IM25-21

Rtc4-0=IM20-16

5.3 组合逻辑控制器

M3=Addu+Subu+Ori+Beq+J+lw+Sw

M4=Ori+Lw+Sw

ALUC2=0

ALUC1=Ori+Sll

ALUC0=Subu+Sll+Beq

M2=Addu+Subu+Ori+Sll+Beq+J+Sw

Rdc4-0=IM15-11(Addu+Subu+Sll+Beq)+IM20-16(Ori+Lw)

RF_W=Addu+Subu+Ori+Sll+Lw

RF_CLK= (Addu+Subu+Ori+Sll+Lw) CLK ( )

5.3 组合逻辑控制器

M5=BeqZ

M1=Addu+Subu+Ori+Sll+Beq+Lw+Sw

DM_CS=Lw+Sw

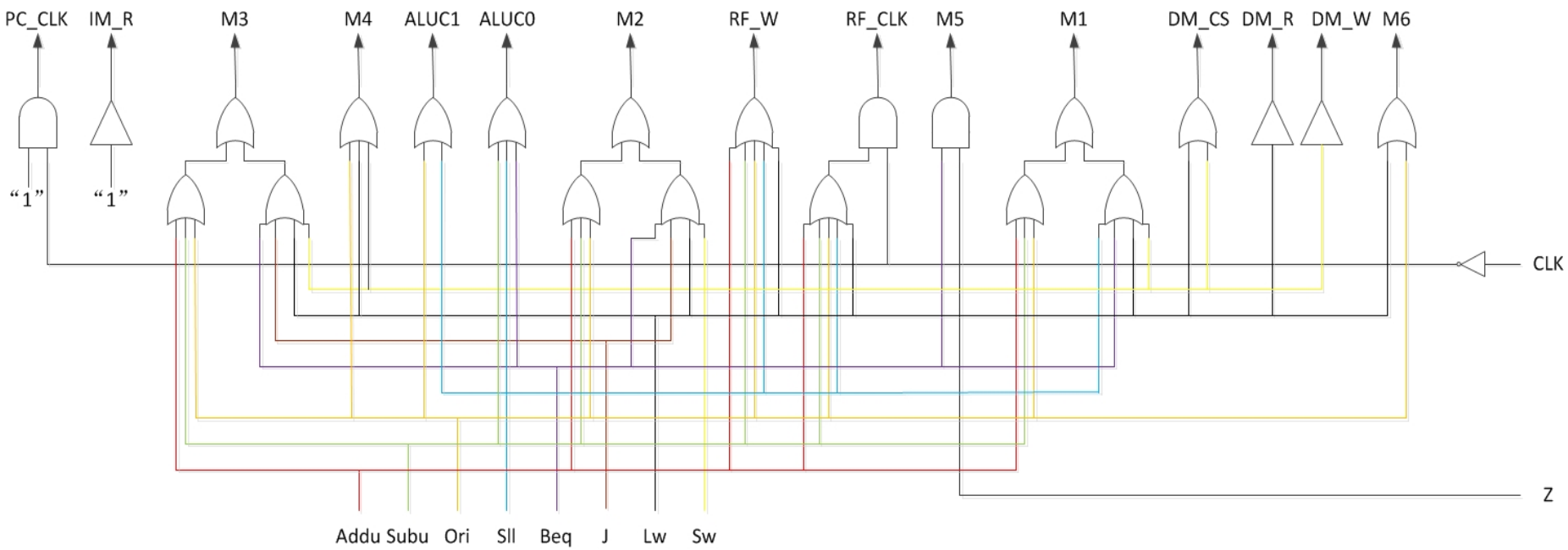
DM_R=Lw

DM_W=Sw

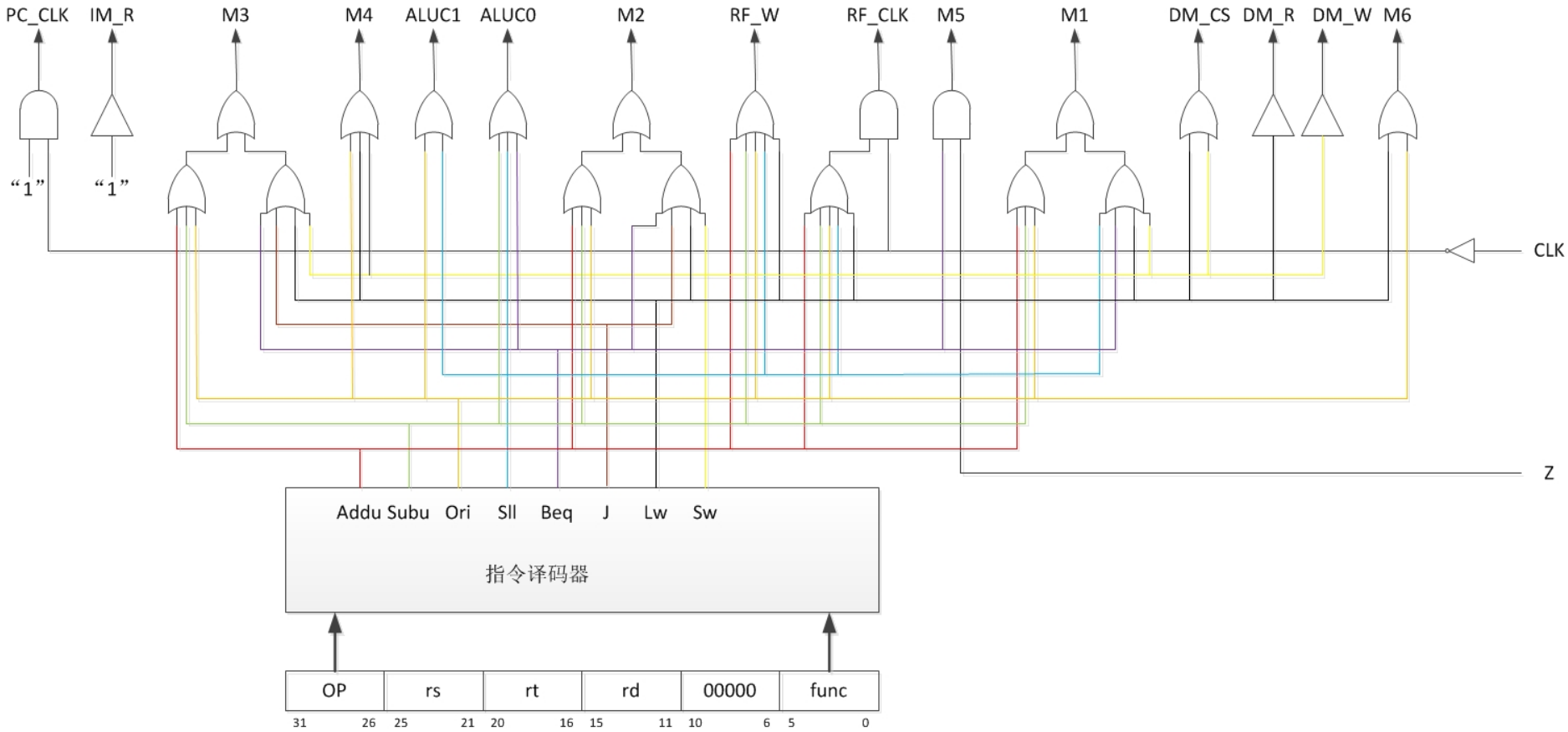
M6=Ori+Lw

5.3 组合逻辑控制器

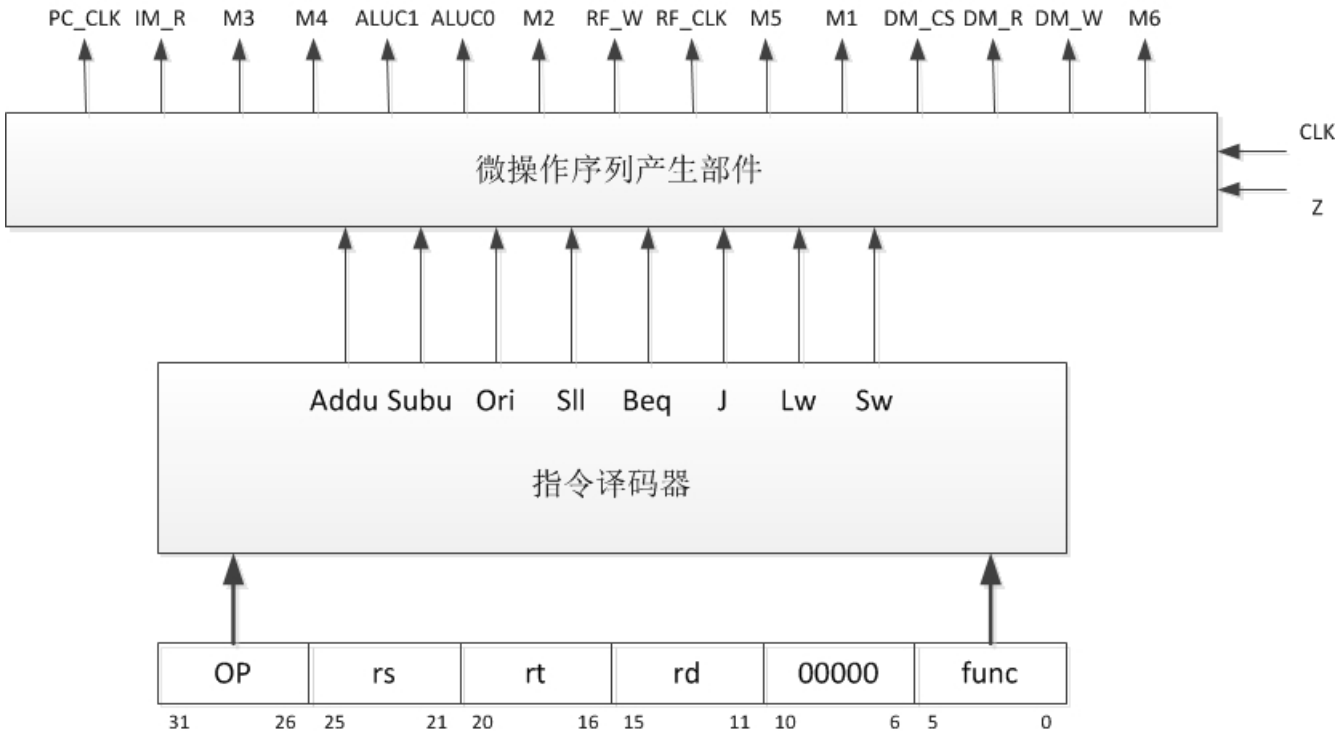
5.构成微操作序列形成部件的组合逻辑网络

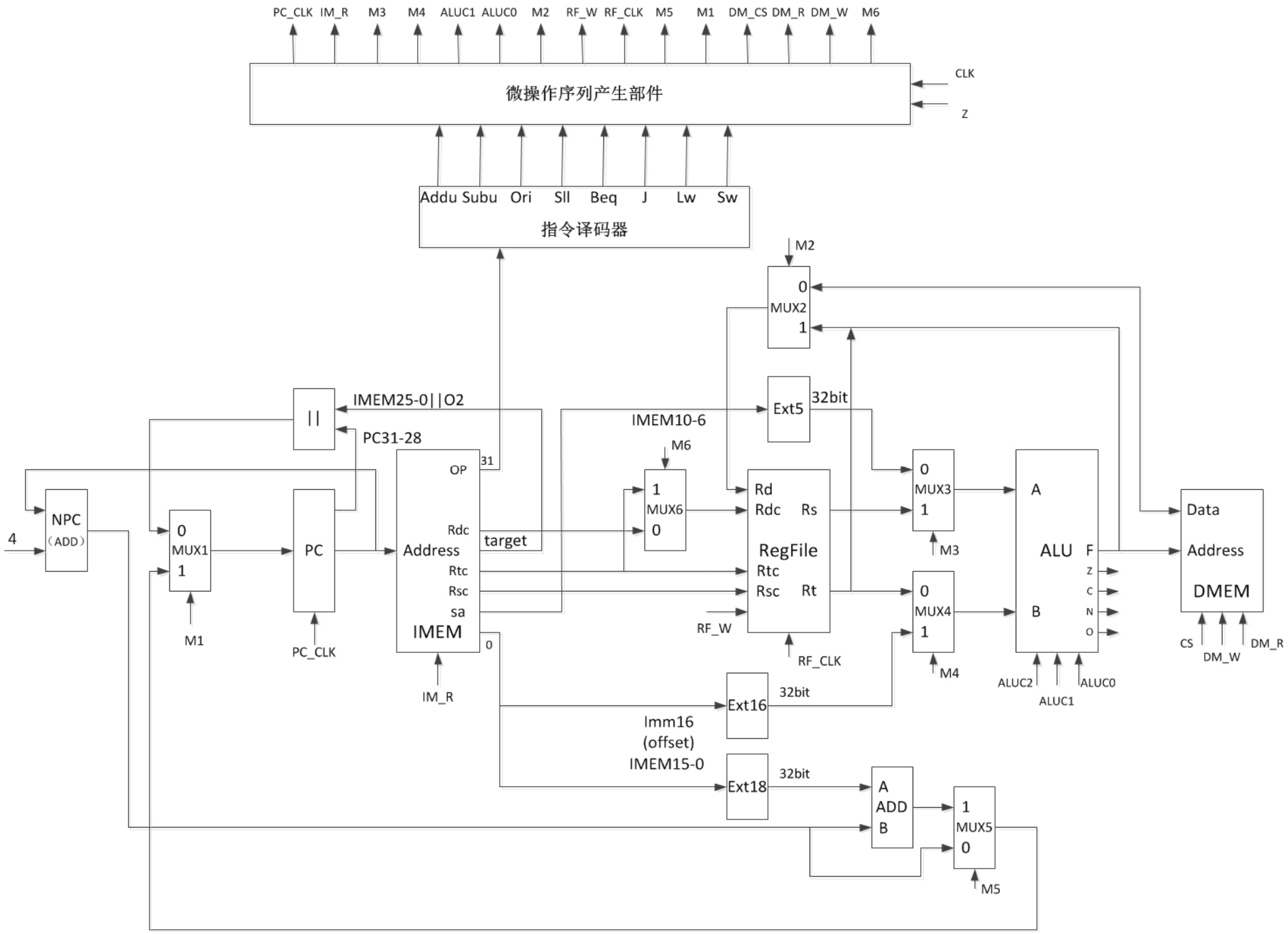


5.3 组合逻辑控制器


PC_CLK
IM_R
M3
M4
ALUC1
ALUC0
M2
RF_W
RF_CLK
M5
M1
DM_CS
DM_R
DM_W
M6
"1"
"1"
CLK
Z
Addu Subu Ori Sll Beq J Lw Sw
指令译码器
OP
rs
rt
rd
00000
func
31 26 25 21 20 16 15 11 10 6 5 0
31 26 25 21 20 16 15 0
Ori(001101)
rs
rt
imm16

5.3 组合逻辑控制器


PC_CLK IM_R M3 M4 ALUC1 ALUC0 M2 RF_W RF_CLK M5 M1 DM_CS DM_R DM_W M6
微操作序列产生部件
CLK
Z
Addu Subu Ori Sll Beq J Lw Sw
指令译码器
OP rs rt rd 00000 func
31 26 25 21 20 16 15 11 10 6 5 0


PC_CLK IM_R M3 M4 ALUC1 ALUC0 M2 RF_W RF_CLK M5 M1 DM_CS DM_R DM_W M6
微操作序列产生部件
CLK
Z
Addu Subu Ori Sll Beq J Lw Sw
指令译码器
M2
0
MUX2
1
IMEM25-0||O2
||
PC31-28
IMEM10-6
Ext5
32bit
M6
1
MUX6
0
OP
31
Rdc
Address
Rtc
Rsc
sa
IMEM
target
0
Rd
Rdc Rs
RegFile
Rtc
Rsc Rt
RF_W
RF_CLK
0
MUX3
1
M3
0
MUX4
1
M4
A
ALU F
Z
C
N
O
B
ALUC2
ALUC1
ALUC0
Data
Address
DMEM
CS
DM_W
DM_R
4
NPC
(ADD)
0
MUX1
1
M1
PC
PC_CLK
IM_R
Ext16
32bit
Imm16
(offset)
IMEM15-0
Ext18
32bit
A
ADD
B
1
MUX5
0
M5

习　题（1）

百年同濟
TONGJI UNIVERSITY

P161

习题：1，12，13

5.4 微程序控制器

微程序的基本思想是由威尔克斯在1950年首先提出的，当时主要解决两个问题：

（1）为微操作序列形成部件的设计提供统一的方法；

（2）使计算机的微操作序列形成部件线路整齐划一，便于工程人员维护和修理。

基本思想：

仿照通常的解题程序的方法，把操作控制信号编成所谓的“微指令”，存放在一个只读存储器中，机器执行指令时，就从只读存储器中一条一条地读出这些微指令，从而产生所需的各个控制信号。

5.4 微程序控制器

5.4.1微程序控制的基本概念

微操作----在计算机中，一条指令的功能是通过按一定次序执行一系列基本操作完成的，这些基本操作称为微操作。

微指令----在微程序控制的计算机中，将由同时发出的控制信号所执行的一组微操作称为微指令，所以微指令就是把同时发出的控制信号的有关信息汇集起来而形成的。将一条指令分成若干条微指令，按次序执行这些微指令，就可以实现指令的功能。组成微指令的微操作，又称微命令。

5.4 微程序控制器

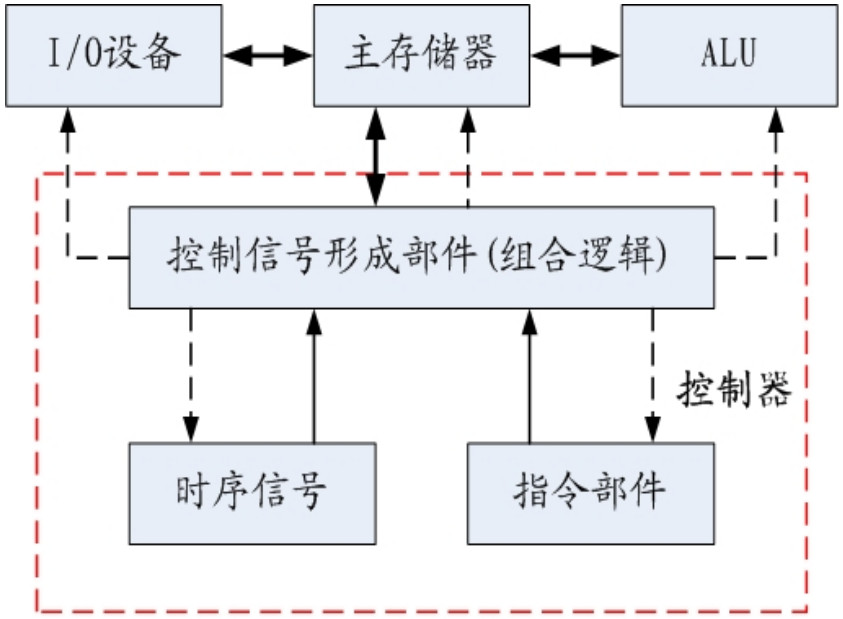
微程序----计算机的程序由指令序列构成，而计算机每条指令的功能均由微指令序列解释完成，这些微指令序列的集合就叫做微程序。

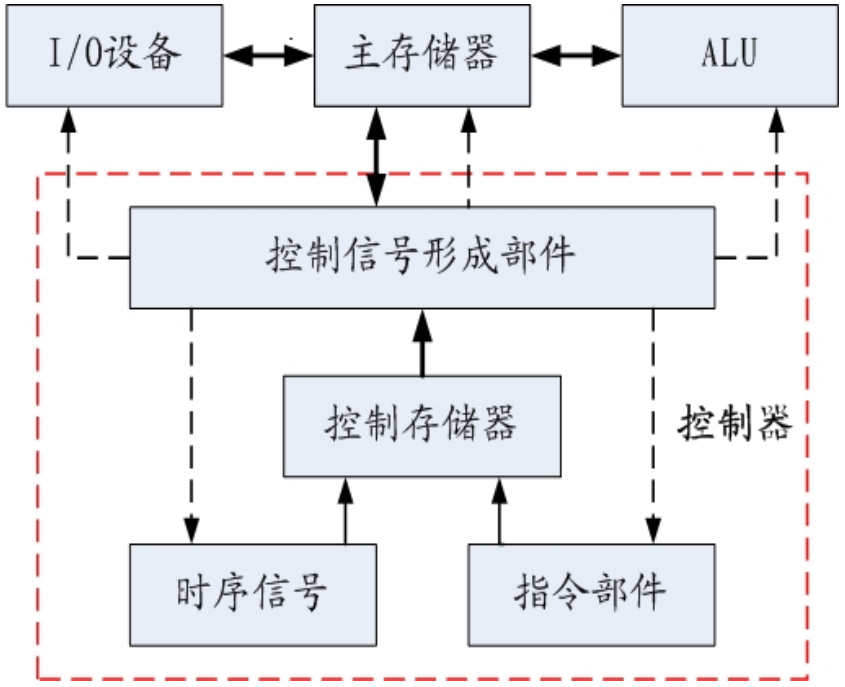
控制存储器----微程序是存放在存储器中的，由于该存储器主要存放控制命令(信号)与下一条执行的微指令地址(简称为下址)，所以被叫做控制存储器。一般计算机指令系统是固定的，所以实现指令系统的微程序也是固定的，于是控制存储器可以用只读存储器实现。

执行一条指令实际上就是执行一段存放在控制存储器中的微程序。

5.4 微程序控制器

微程序控制器和组合逻辑控制器区别





5.4 微程序控制器

- **要解决的问题：**

（1）如何把一个机器指令系统的操作时间表，有效地装进微指令存储器。

（2）如何根据不同的机器指令把相应的微程序（微指令序列）取出，产生对应的控制信号。

微指令通常包括控制字段和下址字段

5.4.2微程序设计技术

1.微指令字的编码设计

（1）直接控制法

在微指令的控制字段中，每一位代表一个微命令，在设计微指令时，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”，这样就可打开或关闭某个控制门。

但在某些复杂的计算机中，微命令甚至可多达三四百个，这使微指令字长达到难以接受的地步，并要求机器有大容量控制存储器。

5.4 微程序控制器

- **举例说明：**

用前面所介绍的组合逻辑Add指令，Add指令所需的控制信号：

Add

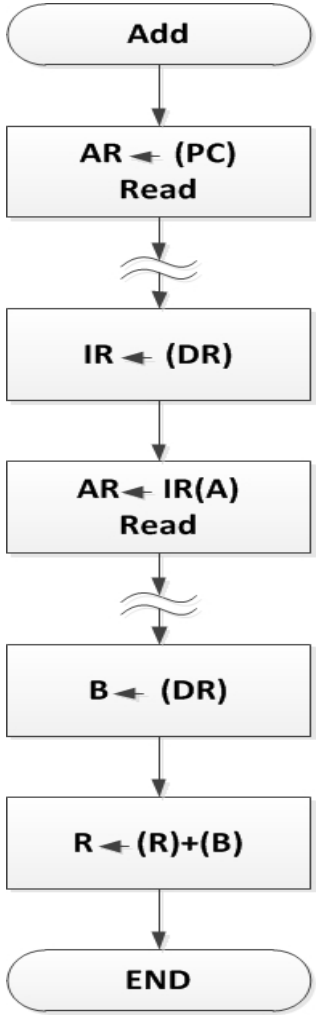
1.PCout,ARin,Read,Wait

2.DRout,IRin

3.IR(A)out,ARin,Read,Wait

4.DRout,Bin

5.Rout,Plus,Rin,End



5.4 微程序控制器

从取指令到执行结束需要以上5步，End信号是新增加的，表示该微程序执行完。

直接控制法的微指令是如何编码的。我们将所有的控制信号都列出：

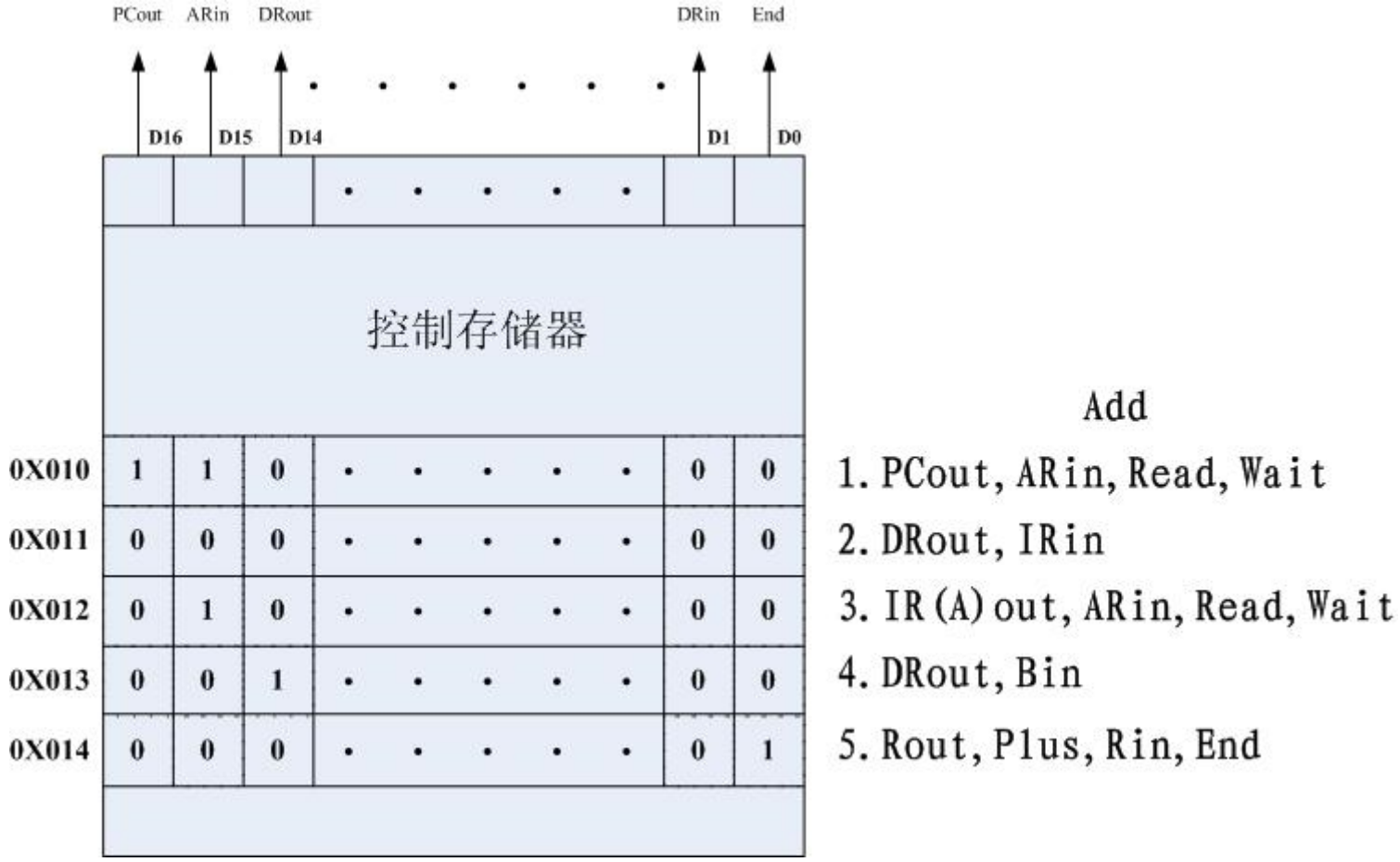
bit16	bit15	bit14	bit13	bit12	bit11	bit10	bit9	bit8	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
PCout	ARin	DRout	IRin	IR(A)out	Bin	Rout	Plus	Minus	Rin	Read	PCin	ClearB	Wait	Write	DRin	End

5.4 微程序控制器

- 在每一步中，凡是需要控制信号有效时填入“1”，需要控制信号有效时填入“0”。

	bit16	bit15	bit14	bit13	bit12	bit11	bit10	bit9	bit8	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
	PCout	ARin	DRout	IRin	IR(A)out	Bin	Rout	Plus	Minus	Rin	Read	PCin	ClearB	Wait	Write	DRin	End
1.	1	1	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0
2.	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
3.	0	1	0	0	1	0	0	0	0	0	1	0	0	1	0	0	0
4.	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0
5.	0	0	0	0	0	0	1	1	0	1	0	0	0	0	0	0	1

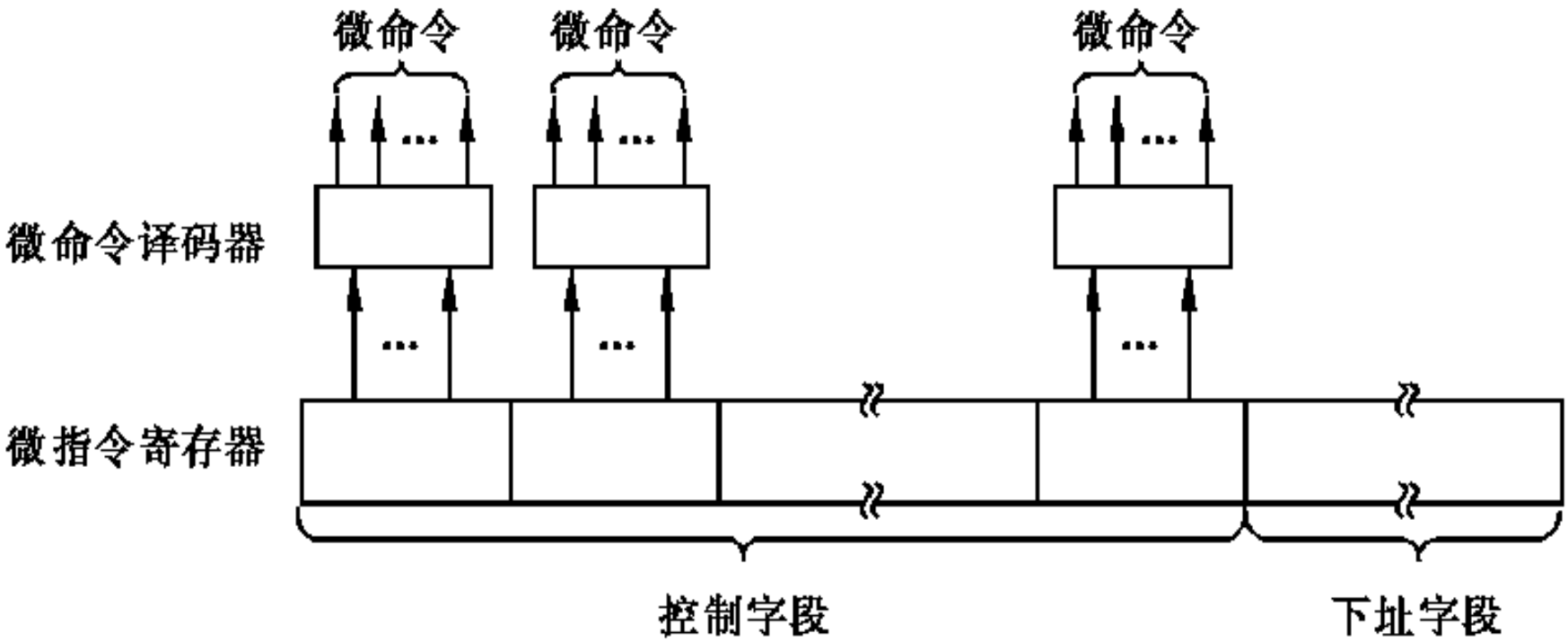
5.4 微程序控制器


PCout
ARin
DRout
DRin
End
D16
D15
D14
D1
D0
控制存储器
Add
0X010 1 1 0 0 0
0X011 0 0 0 0 0
0X012 0 1 0 0 0
0X013 0 0 1 0 0
0X014 0 0 0 0 1
1. PCout, ARin, Read, Wait
2. DRout, IRin
3. IR(A)out, ARin, Read, Wait
4. DRout, Bin
5. Rout, Plus, Rin, End

(2) 字段直接编译法

在计算机中的各个控制门，在任一微周期内，不可能同时被打开，而且大部分是关闭的(即相应的控制位为“0”)。所谓微周期，指的是一条微指令所需的执行时间。如果有若干个(一组)微命令，在每次选择使用它们的微周期内，只有一个微命令起作用，那么这若干个微命令是互斥的。选出互斥的微命令，并将这些微命令编成一组，成为微指令字的一个字段，用二进制编码来表示。而在微指令寄存器的输出端，为该字段增加一个译码器，该译码器的输出即为原来的微命令。

5.4 微程序控制器



5.4 微程序控制器

- **分段原则**

  分段要按相容性和相斥性原则进行。

- **相容性原则**

  就是把可以在同一时刻发出的微命令分在不同组内。

- **相斥性原则**

  就是把不能在同一时刻发出的微命令分在同一组内。

5.4 微程序控制器

- **举例说明**

以上例为例，先将17个信号分成5段。

3	3	2	2	1
无控制 000 PCout 001 DRout 010 Rout 011 IR(A)out 100	无控制 000 IRin 001 ARin 010 Bin 011 Rin 100 PCin 101 DRin 110	无控制 00 Plus 01 Minus 10 Wait 11	无控制 00 Read 01 Write 10	继续 0 End 1

5.4 微程序控制器

Add指令

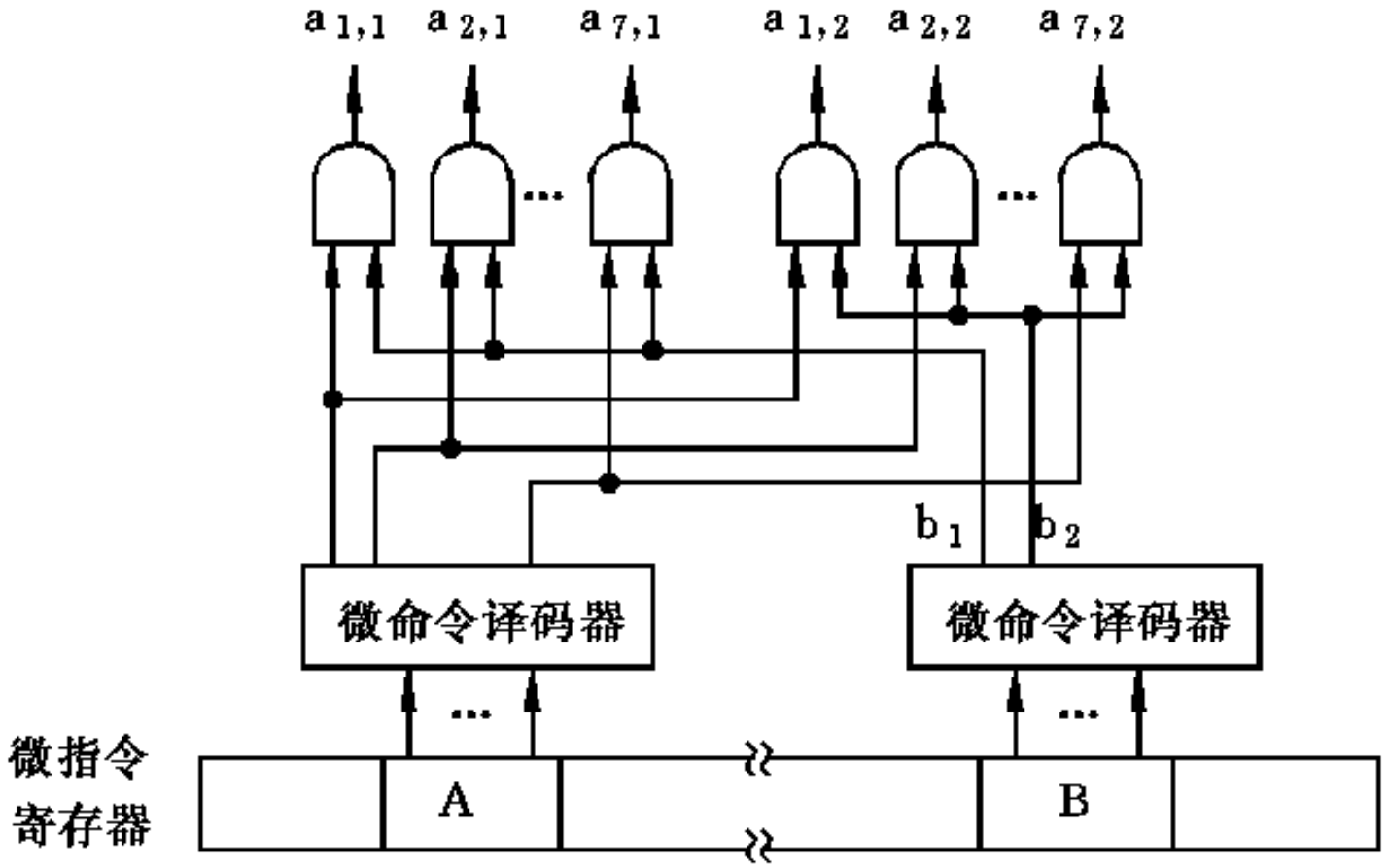
3	3	2	2	1
1.PCout,ARin,Read,Wait	001	010	11	01	0
2.DRout,IRin	010	001	00	00	0
3.IR(A)out,ARin,Read,Wait	100	010	11	01	0
4.DRout,Bin	010	011	00	00	0
5.Rout,Plus,Rin,End	011	100	01	00	1

(3) 字段间接编译法

字段间接编译法是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。如果在字段直接编译法中，还规定一个字段的某些微命令，要兼由另一字段中的某些微命令来解释，称为字段间接编译法。如图所示。

本方法进一步减少了指令长度，但很可能会削弱微指令的并行控制能力，因此通常只作为直接编译法的一种辅助手段。

5.4 微程序控制器



5.4 微程序控制器

(4) 常数源字段E

在微指令中，一般设有一个常数源字段E，就如指令中的直接操作数一样。E字段一般仅有几位，用来给某些部件发送常数，故有时称为发射字段。该常数有时作为操作数送入ALU运算；有时作为计数器初值，用来控制微程序的循环次数等。

5.4 微程序控制器

5.4.3 微程序流的控制

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制存储器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。所谓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。

5.4 微程序控制器

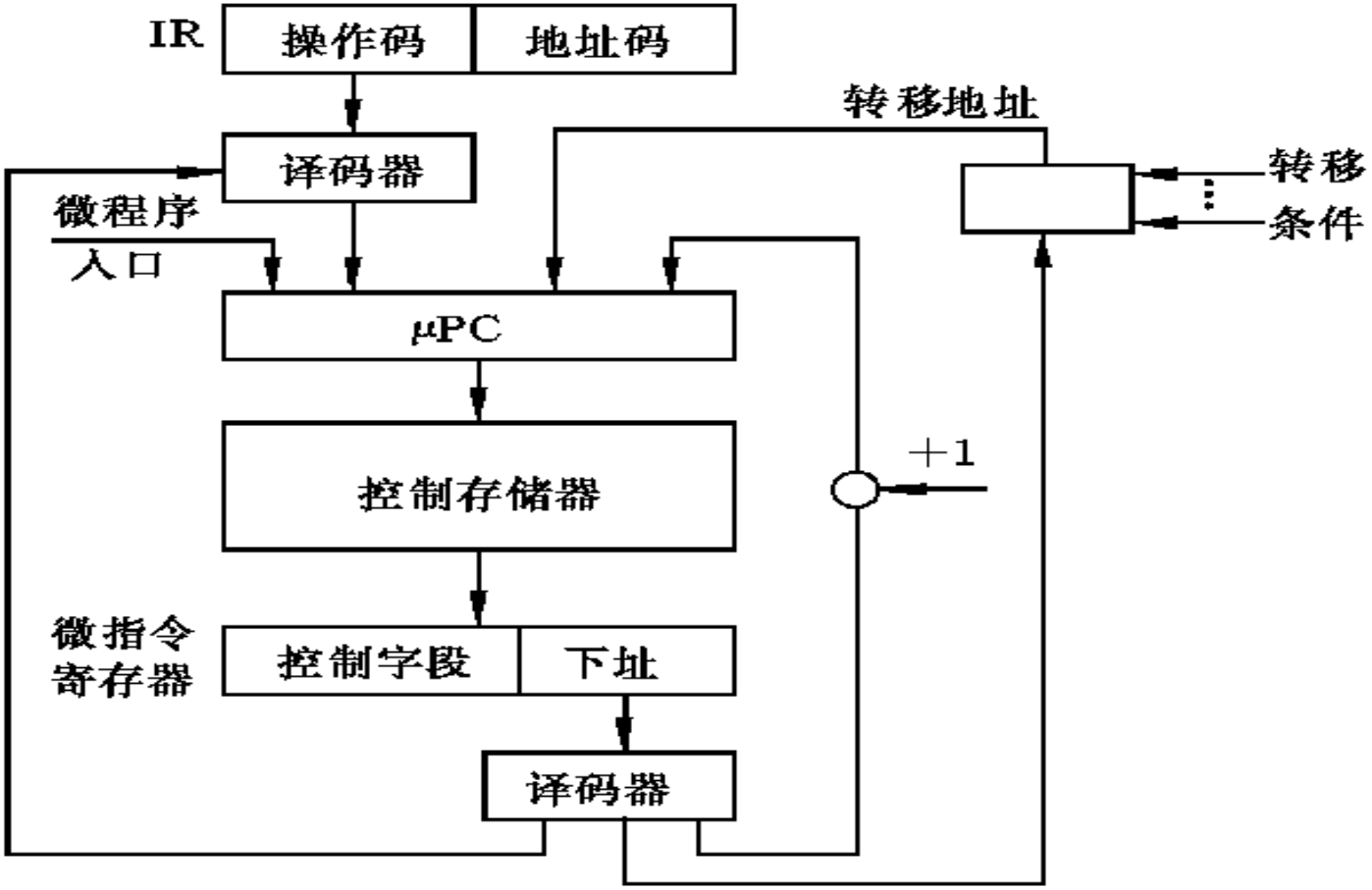
在实际应用中，存在多种变化情况。与程序设计相似，在微程序设计中除了顺序执行微程序外还存在转移功能和微循环程序与微子程序等，这将影响微指令下地址的形成。

1. 产生后继微指令地址的几种方法

(1) 以增量方式产生后继微地址。

在顺序执行微指令时，后继微地址由现行微地址加上一个增量(通常为1)形成的；而在非顺序执行时则要产生一个转移微地址，如图所示。

5.4 微程序控制器



5.4 微程序控制器

(2) 增量与下址字段结合产生后继微地址

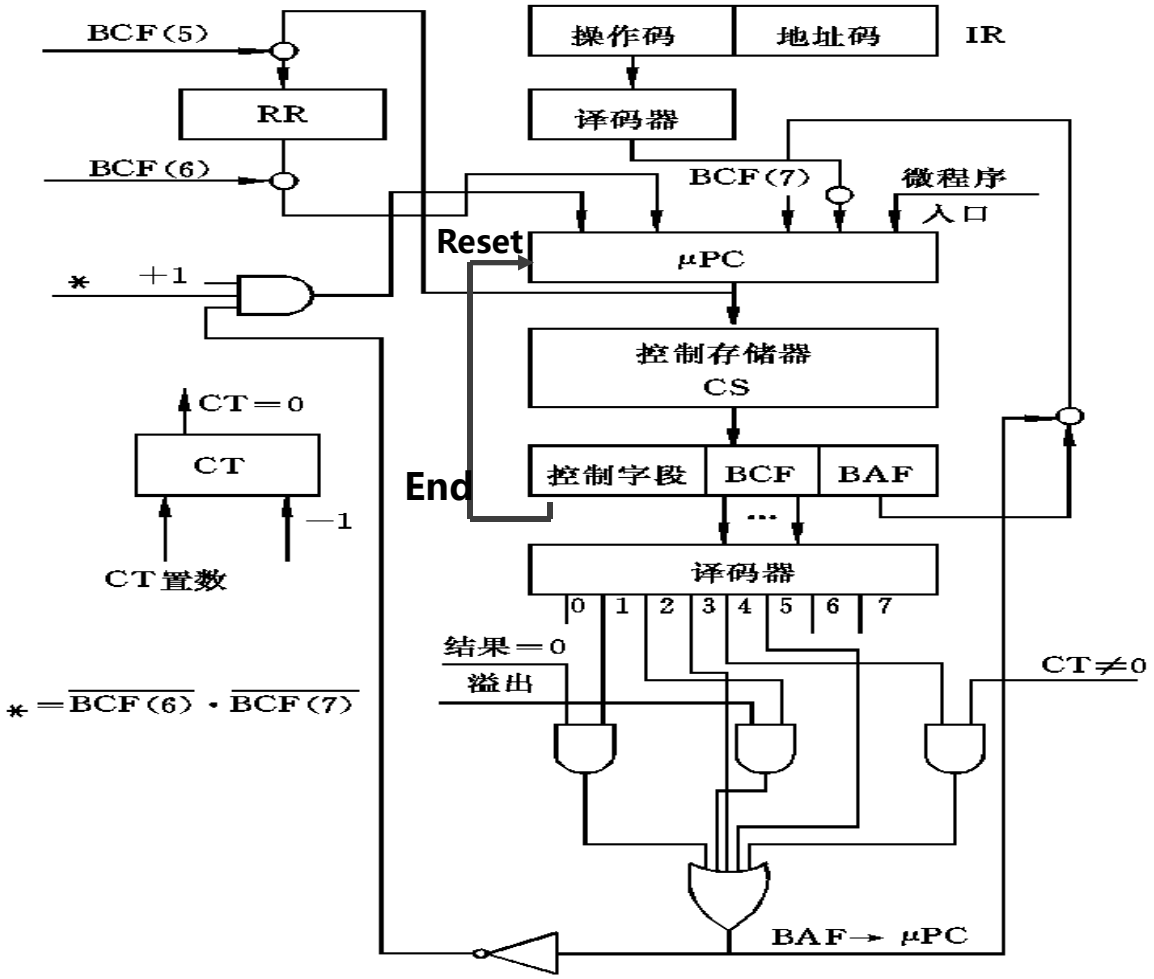
在图中将微指令的下址字段分成两部分：转移控制字段BCF和转移地址字段BAF，当微程序实现转移时，将BAF送μPC，否则顺序执行下一条微指令(μPC+1)。

由BCF定义的八个微命令见表。

5.4 微程序控制器

BCF字段		硬件条件	计数器CT		返回寄存器RR输入	后继微地址
编码	微命令名称		操作前	操作		
0	顺序执行	×	×	×	×	μPC+1
1	结果为0转移	结果为0	×	×	×	BAF
		结果不为0				μPC+1
2	结果溢出转移	溢出	×	×	×	BAF
		不溢出				μPC+1
3	无条件转移	×	×	×	×	BAF
4	测试循环	×	为0	CT-1	×	μPC+1
			不为0			BAF
5	转微子程序	×	×	×	μPC+1	BAF
6	返回	×	×	×	×	RR
7	操作码形成微址	×	×	×	×	由操作码形成

5.4 微程序控制器


BCF(5)
RR
BCF(6)
* +1
CT=0
CT
-1
CT置数
* = BCF(6) · BCF(7)
操作码
地址码
IR
译码器
BCF(7)
微程序
入口
Reset
μPC
控制存储器
CS
控制字段
BCF
BAF
End
...
译码器
0 1 2 3 4 5 6 7
结果=0
溢出
CT≠0
BAF→ μPC

5.4 微程序控制器

- 微程序的起始地址是由指令寄存器IR操作码字段经过译码产生的，并送入微程序计数器（μPC）；
- 当微指令顺序执行时，下一条微指令地址由微程序计数器（μPC）加1产生；
- 当微指令转移时，在微指令的转移地址和控制字段的控制下，与状态、条件码、标志组合产生新的微指令地址，送入微程序计数器（μPC）,实现各种转移。

在正常情况下，μPC总是逐次递增的，在以下三种情况下，μPC的值由其它部件送入。

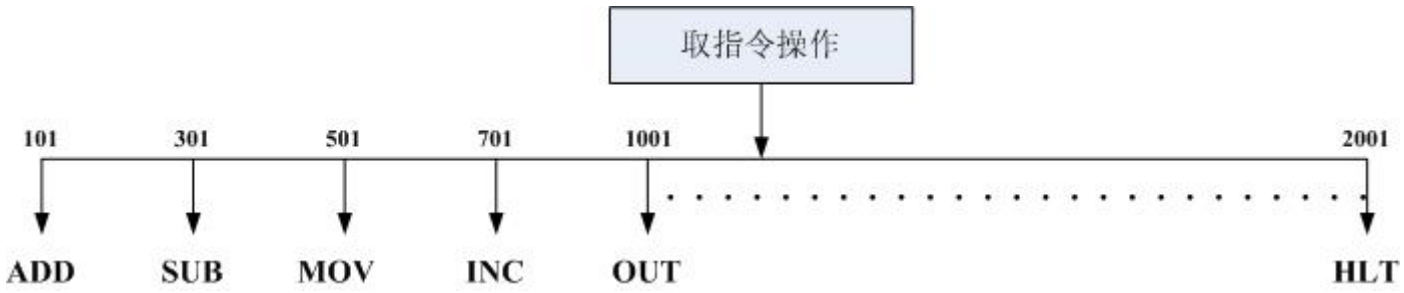
5.4 微程序控制器

- 当CPU刚上电运行或遇到微指令结束信号（End)时，μPC清零。
- 每当一条新指令送到IR时，由译码器等电路给出该指令的微程序首地址送入μPC 。
- 在微指令执行过程中，当遇到转移控制字段给出转移微命令时，BAF内容送入μPC。

5.4 微程序控制器

(3) 多路转移方式

一条微指令存在多个转移分支的情况称为多路转移。在执行某条微指令时，可能会遇到在若干个微地址中选择一个作为后继微地址的情况，最明显的例子是根据操作码产生不同的后继微地址。

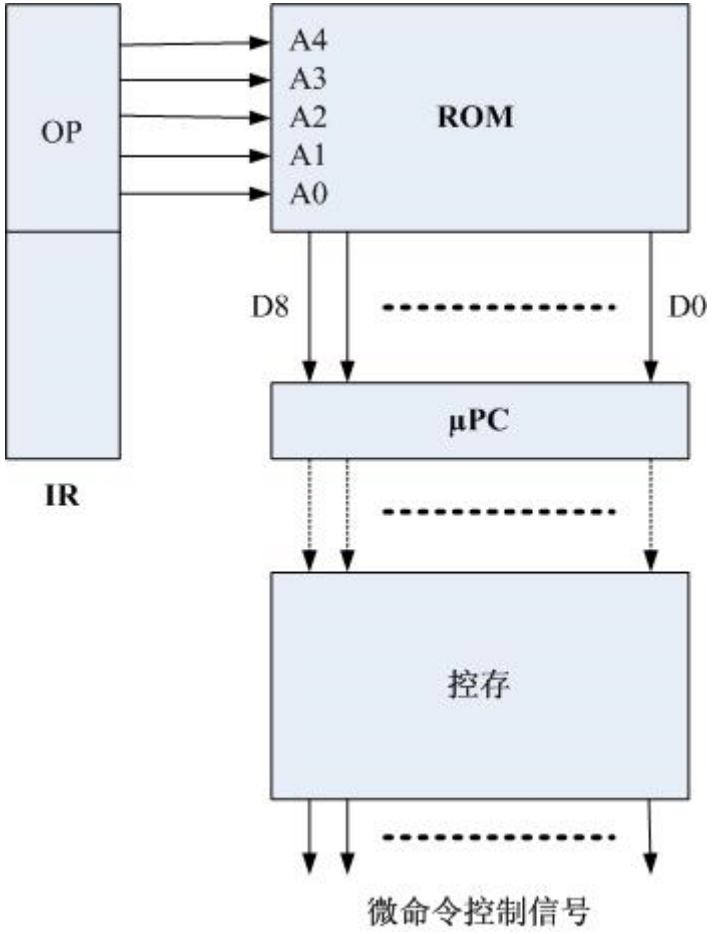


5.4 微程序控制器

实现此功能的电路通常有几种方法：

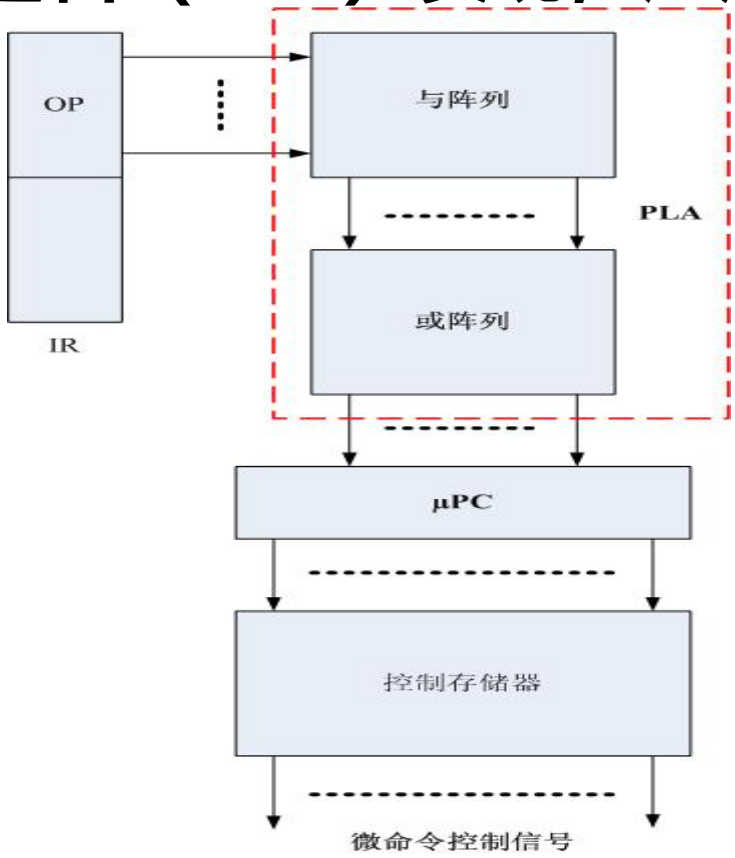
1）由PROM(可编程序只读存储器)组成，也有把它称为MAPROM(映像只读存储器)。该存储器的特点是以指令的操作码作为地址输入，而相应的存储单元内容即为该指令的第一条微指令的入口地址。该存储器的容量等于或略大于机器的指令数，所以容量小，速度快。如下图所示。

5.4 微程序控制器



5.4 微程序控制器

（2）用可编程逻辑（PLA）实现，应用的较多。

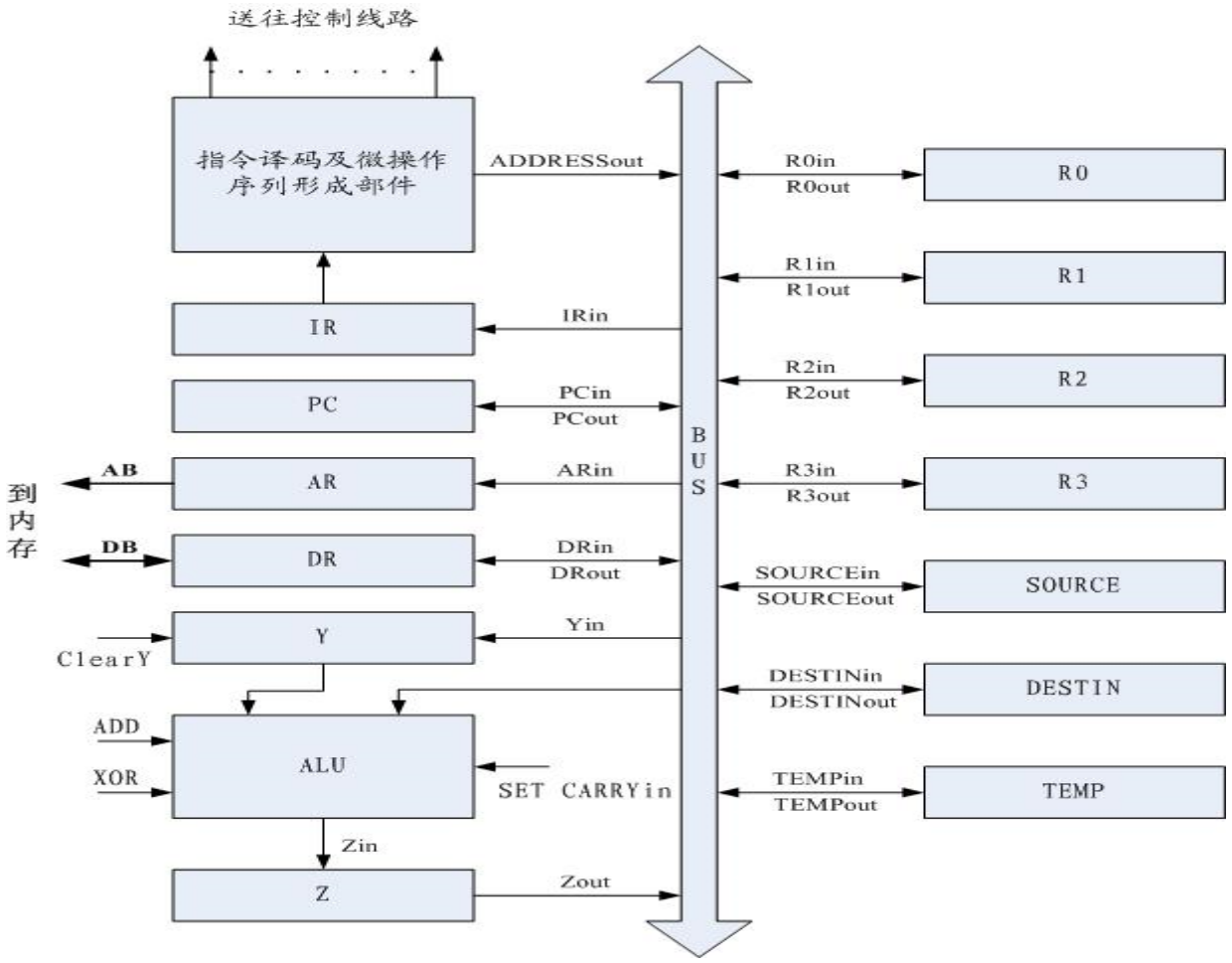


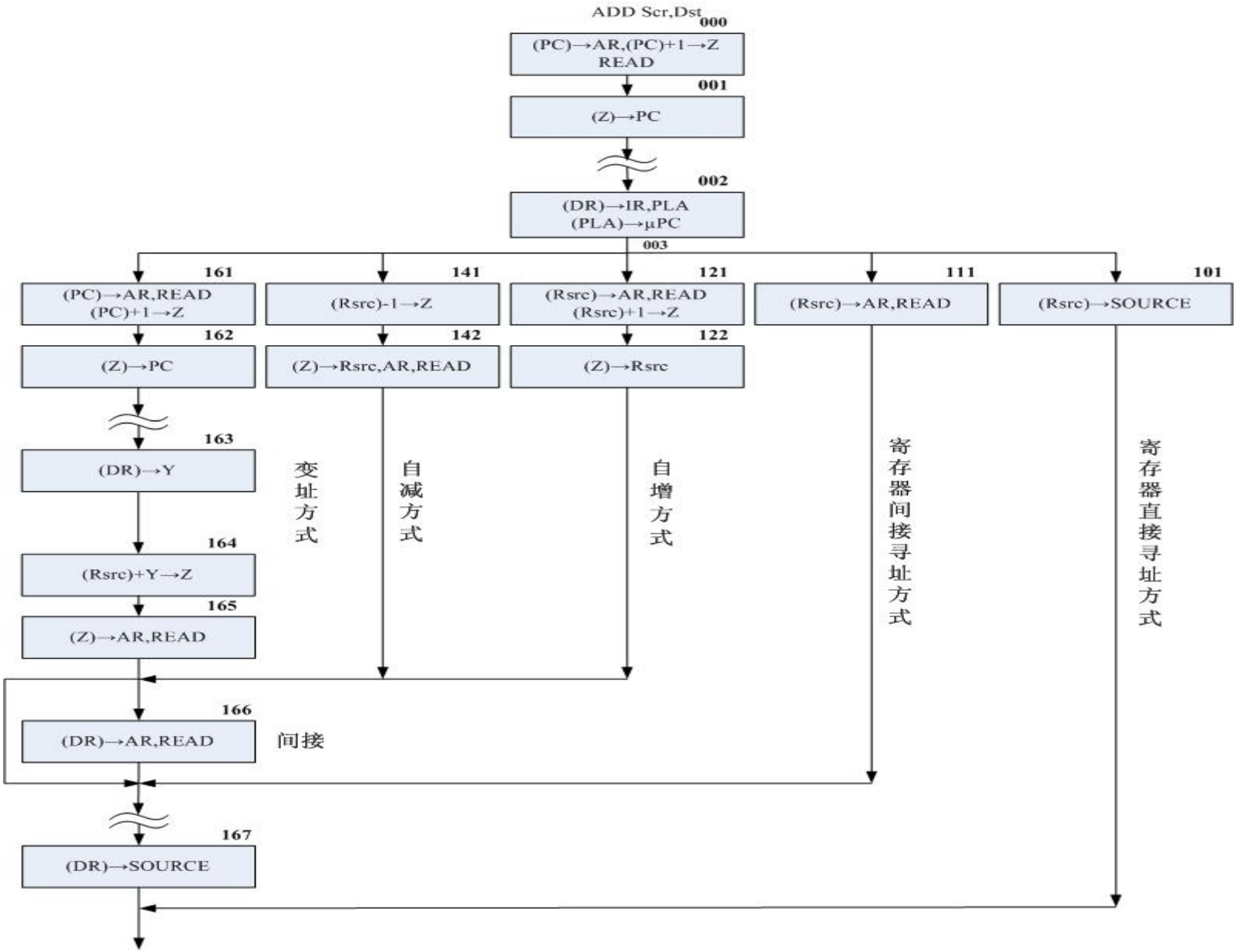
5.4 微程序控制器

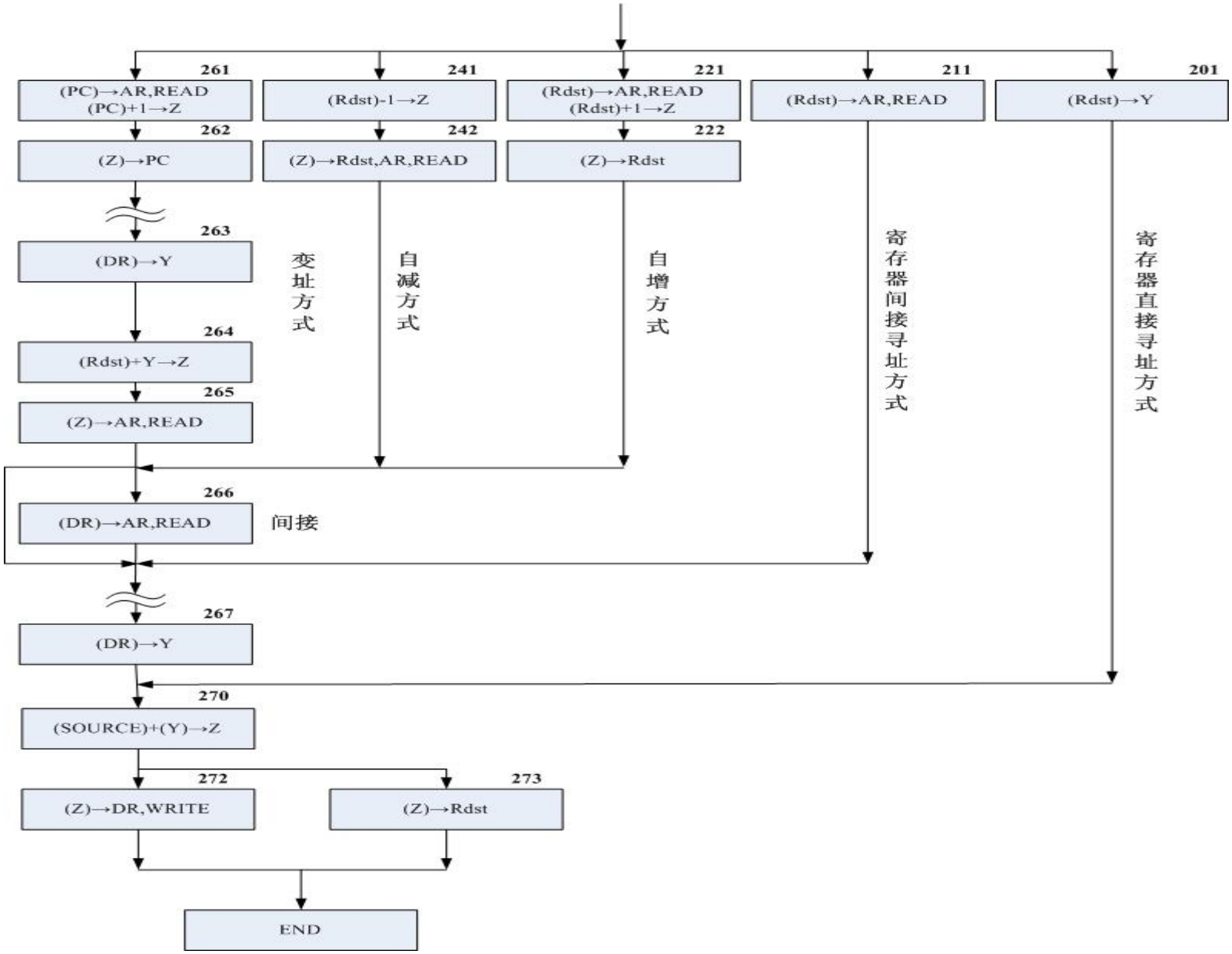
- **给出一个实际的指令微程序例子：**

下图是PDP11小型机的数据通路图及微指令流程图。

5.4 微程序控制器


送往控制线路
指令译码及微操作序列形成部件
ADDRESSout
IR
IRin
PC
PCin
PCout
AB
AR
ARin
到内存
DB
DR
DRin
DRout
Y
Yin
ClearY
ADD
XOR
ALU
SET CARRYin
Zin
Z
Zout
BUS
R0in
R0out
R0
R1in
R1out
R1
R2in
R2out
R2
R3in
R3out
R3
SOURCEin
SOURCEout
SOURCE
DESTINin
DESTINout
DESTIN
TEMPin
TEMPout
TEMP


ADD Scr,Dst
000
(PC)→AR,(PC)+1→Z
READ
001
(Z)→PC
002
(DR)→IR,PLA
(PLA)→μPC
003
161
(PC)→AR,READ
(PC)+1→Z
162
(Z)→PC
163
(DR)→Y
164
(Rsrc)+Y→Z
165
(Z)→AR,READ
166
(DR)→AR,READ
间接
167
(DR)→SOURCE
141
(Rsrc)-1→Z
142
(Z)→Rsrc,AR,READ
变址方式
121
(Rsrc)→AR,READ
(Rsrc)+1→Z
122
(Z)→Rsrc
自减方式
111
(Rsrc)→AR,READ
自增方式
寄存器间接寻址方式
101
(Rsrc)→SOURCE
寄存器直接寻址方式


261
(PC)→AR,READ
(PC)+1→Z
262
(Z)→PC
263
(DR)→Y
264
(Rdst)+Y→Z
265
(Z)→AR,READ
266
(DR)→AR,READ
267
(DR)→Y
270
(SOURCE)+(Y)→Z
272
(Z)→DR,WRITE
273
(Z)→Rdst
END
241
(Rdst)-1→Z
242
(Z)→Rdst,AR,READ
221
(Rdst)→AR,READ
(Rdst)+1→Z
222
(Z)→Rdst
211
(Rdst)→AR,READ
201
(Rdst)→Y
变址方式
自减方式
自增方式
寄存器间接寻址方式
寄存器直接寻址方式
间接

5.4 微程序控制器

- **ADD (Rs)，Rd ;**根据数据通路、ADD的微指令流程图，它的微程序如下：

微指令地址	微指令
000	PCout,ARin,READ,ClearY,Carry-in,Add,Zin
001	Zout,PCin,Wait MFC
002	DRout,IRin
111	Rsrcout,ARin,READ,Wait MFC
167	DRout,SOURCEin
201	Rdstout,Yin
270	SOURCEout,ADD,Zin
273	Zout,Rdstin,End

5.5 微指令格式

自从采用微程序设计以来，许多微程序设计都是针对具体机器要求、数据通路结构及控制存储器的速度制定各自的微指令格式，很难通过分析其类型来确定某种通用性的微指令格式。

六十年代Beekman等人曾将微程序设计分成两类：

1.水平微程序设计

2.垂直微程序设计

与其相对应的就有：水平型微指令与垂直型微指令

5.5 微指令格式

5.5.1 水平型微指令及水平微程序设计概念

其基本特点是在一条微指令中定义并执行多个并行操作微命令。在实际应用中，直接控制法、字段编译法(直接、间接编译法)经常应用在同一条水平型微指令中。从速度来看，直接控制法最快，字段编译法要经过译码，所以会增加一些延迟时间。

5.5 微指令格式

- **水平型微指令的基本特性是：**

采用长格式，一条微指令能控制多个部件并行操作，控制信息编码简单，尽可能使微命令与控制门之间具有直接的对应关系，微指令执行效率高，速度快。

采用水平型微指令编制微程序称为水平微程序设计。

这种设计由于横向信息位比较宽，编制微程序比较短，微程序执行速度也比较快。

5.5 微指令格式

- **水平型微指令主要缺点：**

微指令字长比较长，增加了ROM横向容量，编制微程序需要非常熟悉机器的数据通路，应用困难。

美国的DEC公司的VAX-11/780超级小型机，采用水平微指令格式，整个微指令字长96位，包含32个字段。

5.5 微指令格式

5.5.2 垂直型微指令及垂直微程序设计概念

在微指令中设置有微操作码字段，采用微操作码编译法，**由微操作码规定微指令的功能，称为垂直型微指令。**其特点是不强调实现微指令的并行控制功能，通常一条微指令只要求能控制实现一二种操作。这种微指令格式与指令相似：每条指令有一个操作码；每条微指令有一个微操作码。

有多种指令格式：寄存器传送、运算控制、移位控制、条件转移等。

5.5 微指令格式

- **垂直型微指令的基本特征是：**

采用短指令格式，一条微指令只能产生一、二个控制信号，并行控制能力低，全部微命令组成一个操作码字段，经过完全译码。微指令的各二进制位与数据通路的各控制点之间完全不存在直接的对应关系，因此，在微指令中需要给出目的部件或源部件的编址。

采用垂直型微指令编制微程序称为垂直微程序设计。

5.5 微指令格式

- **垂直型微指令主要缺点：**

横向信息位比较短，为解释一条机器指令或某种处理过程需要的微指令数多，因而编制出的微程序较长，要求ROM容量大，执行速度慢。

美国Burroughs公司的B1700系统就采用典型的垂直型微指令。它的微指令字长16位，共32条，每条微指令指定一种操作。

下面我们举一个经简化的例子，设微指令字长16位，微操作码3位，有八条微指令如下：

5.5 微指令格式

(1) 寄存器-寄存器传送型微指令

微指令格式

功能：把源寄存器数据送目标寄存器。

5.5 微指令格式

(2) 运算控制型微指令

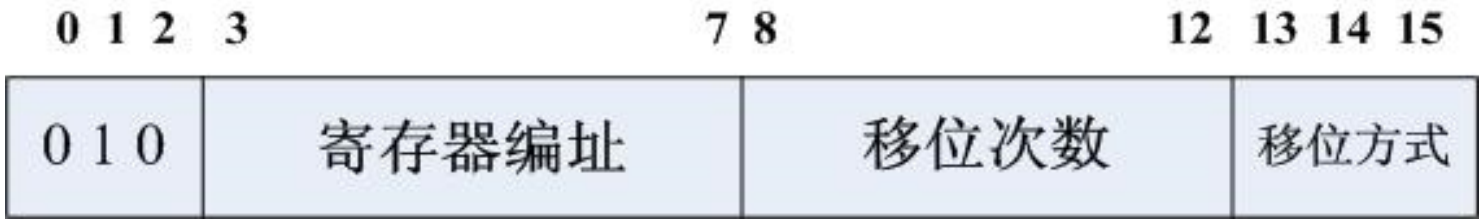
微指令格式

功能：选择运算器(ALU)的左、右两输入端的信息，按ALU字段所指定的运算功能进行处理，并将结果送入暂存器中。

5.5 微指令格式

(3) 移位控制型微指令

微指令格式



功能：将寄存器中的数据按指定的移位方式进行移位。

5.5 微指令格式

(4) 访问主存微指令

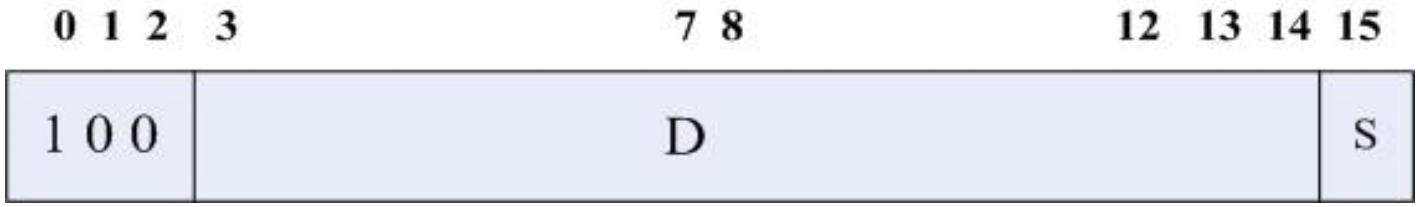
微指令格式

功能：将存储器中一个单元的信息送入寄存器或将寄存器中的数据送往存储器。

5.5 微指令格式

(5) 无条件转移微指令

微指令格式：

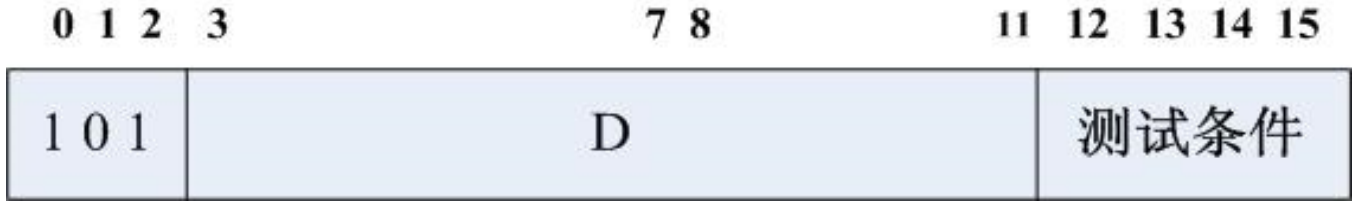


功能：实现无条件转移或转微子程序功能。

5.5 微指令格式

(6) 条件转移微指令

微指令格式：



功能：根据测试对象的状态(例如运算结果为0、结果溢出、计数器状态等)决定转移到D所指定的微地址单元，还是顺序执行下一条微指令。9位D字段不足以表示一个完整的微地址，但可以用来替代现行微地址μPC的低位。

5.5 微指令格式

(7) 其他

还有110与111两种操作码，可以用来定义输入/输出微操作或其他难以归类的杂操作，第3～15位可以根据需要定义各种相应的微命令字段。

- **水平型微指令与垂直型微指令的比较**

(1) 水平型微指令并行操作能力强，效率高，灵活性强，垂直型微指令则差。

(2) 水平型微指令执行一条指令的时间短，垂直型微指令执行时间长。

5.5 微指令格式

(3) 由水平型微指令解释指令的微程序，具有微指令字比较长，但微程序短的特点。垂直型微指令则相反，微指令字比较短而微程序长。

(4) 水平型微指令用户难以掌握，而垂直型微指令与指令比较相似，相对来说，比较容易掌握。

- **水平型微指令与垂直型微指令的本质区别：**

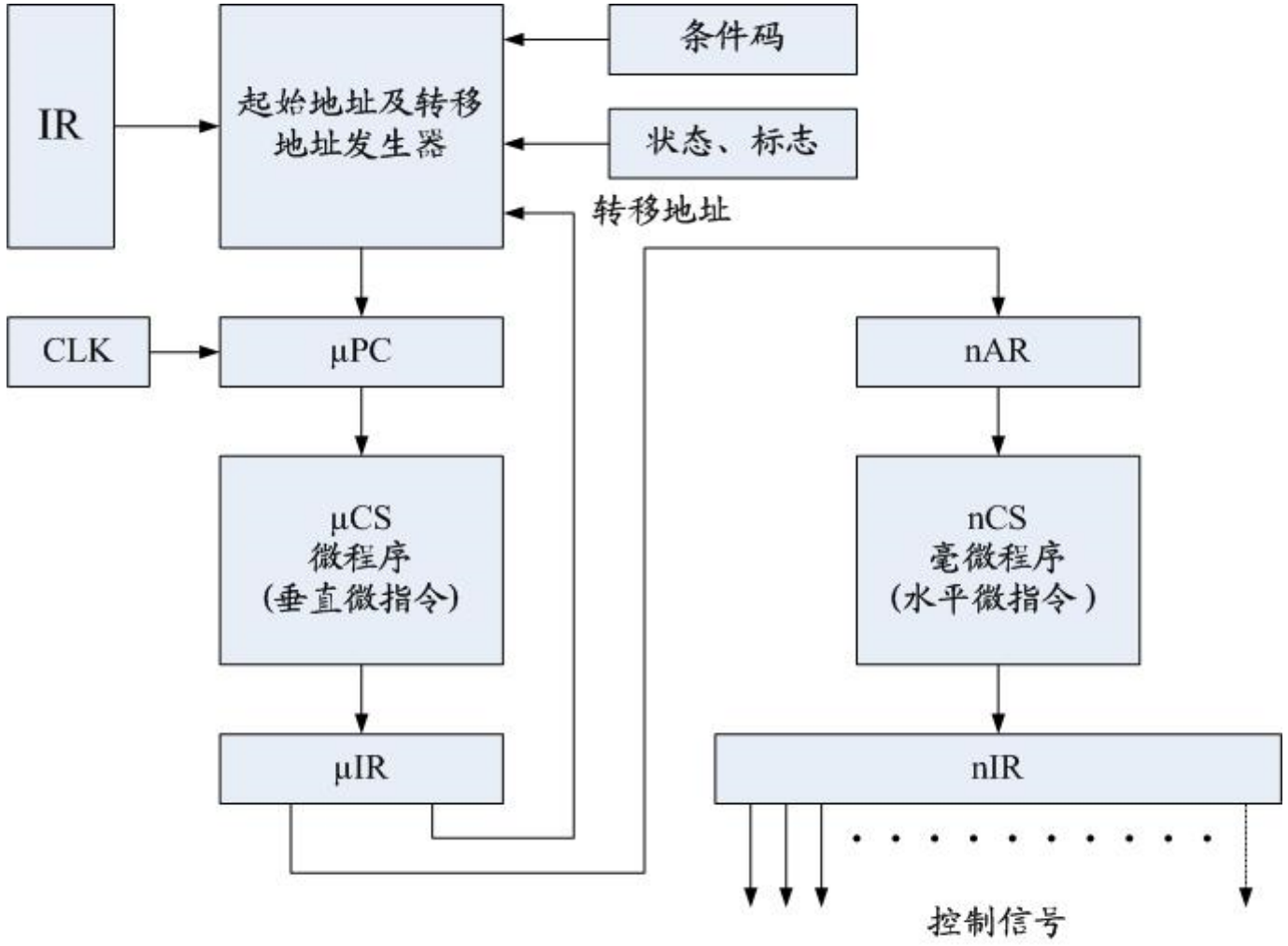
水平型微指令面向处理机内部控制逻辑的描述，垂直型微指令面向算法描述。

5.5.3 毫微程序设计的基本概念

毫微程序可以看作是用以解释微程序的一种微程序，因此组成毫微程序的毫微指令就可看作是解释微指令的微指令。

通常是将垂直微程序设计与水平微程序设计结合起来。第一级采用垂直微程序，第二级采用水平微程序。当执行一条指令时，首先进入第一级微程序，由于它是垂直型微指令，所以并行操作功能不强，可由它来调用第二级微程序(即水平微程序)，并行产生多个控制信号。结构如图所示。

5.5 微指令格式


IR
起始地址及转移
地址发生器
条件码
状态、标志
转移地址
CLK
μPC
μCS
微程序
(垂直微指令)
μIR
nAR
nCS
毫微程序
(水平微指令)
nIR
控制信号

5.5 微指令格式

- **毫微程序设计的基本思想：**

毫微指令存放在毫微程序存储器（nCS)中,由垂直型微指令指定毫微存储器的地址,取出毫微指令产生所需的控制信号。

毫微指令------解释微指令的微指令

毫微程序------由毫微指令组成的微程序

毫微程序设计------用毫微指令来编制毫微程序去解释微程序

5.5 微指令格式

- **毫微程序设计特点：**
  （1）毫微程序设计的两个控制存储器容量都不大，μCS横向字长较短，nCS纵向容量较小。
  （2）由于垂直微指令是面向算法描述的，用户编制微程序较容易。
  （3）水平的毫微指令能产生较多的并行操作，控制效率高。
- **毫微程序设计缺点：**
  由于要读两个存储器，速度较慢。

5.5 微指令格式

采用毫微程序设计的机器，如QM-1计算机，有128条毫微指令，字长360位，微指令字长18位。

5.5 微指令格式

5.5.4 微指令的执行方式

执行一条微指令的过程基本上分为两步，第一步将微指令从控制存储器中取出，称为取微指令，第二步执行微指令，产生所规定的各个控制信号。根据这两步是串行还是并行进行，对微程序控制器的结构和控制有重要的影响，具有两种执行方式：串行方式和并行方式。

5.5 微指令格式

1.串行执行方式

串行执行方式----取微指令和执行微指令按顺序进行，在一条微指令取出并执行后，才能取下一条微指令，即：访问控存的操作与数据通路的操作是顺序进行的，也称为顺序执行方式。

5.5 微指令格式

- **Tcs----控存的存取时间**
- **Tcpu----微指令的执行时间**
- **Tμ----执行一条微指令的时间（称微周期）**

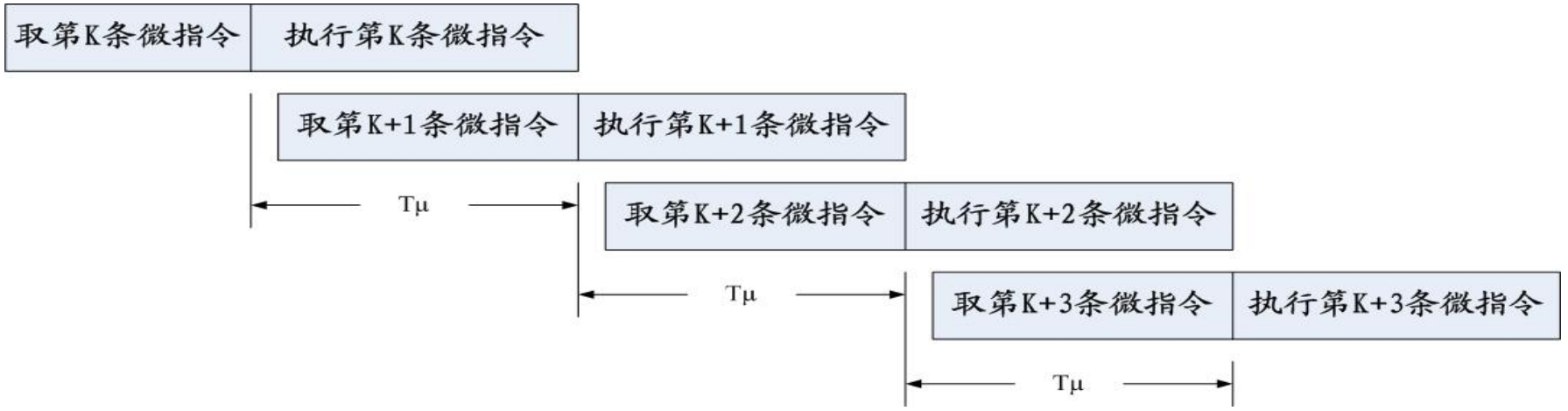
Tcpu相当于CPU的一个节拍时间，一般Tcpu约为Tcs的3倍，所以有1/3左右时间花费在等待从ROM中取出微指令的操作上，因此，Tcs越小越好。

串行工作方式的严重缺点是执行速度慢，因此，当速度的考虑是主要条件时，不宜采用这种方式，但微程序控制器结构较为简单。

5.5 微指令格式

2.并行执行方式

并行执行方式----取微指令和执行微指令是重叠进行的，在本条微指令执行结束前，下一条微指令提前从控存取出。即：访问控存的操作与数据通路的操作是重叠进行的，也称为流水执行方式。



5.5 微指令格式

这种方式缩短了微指令的执行时间。

并行执行还可分为微周期不重叠与微周期相重叠两种。

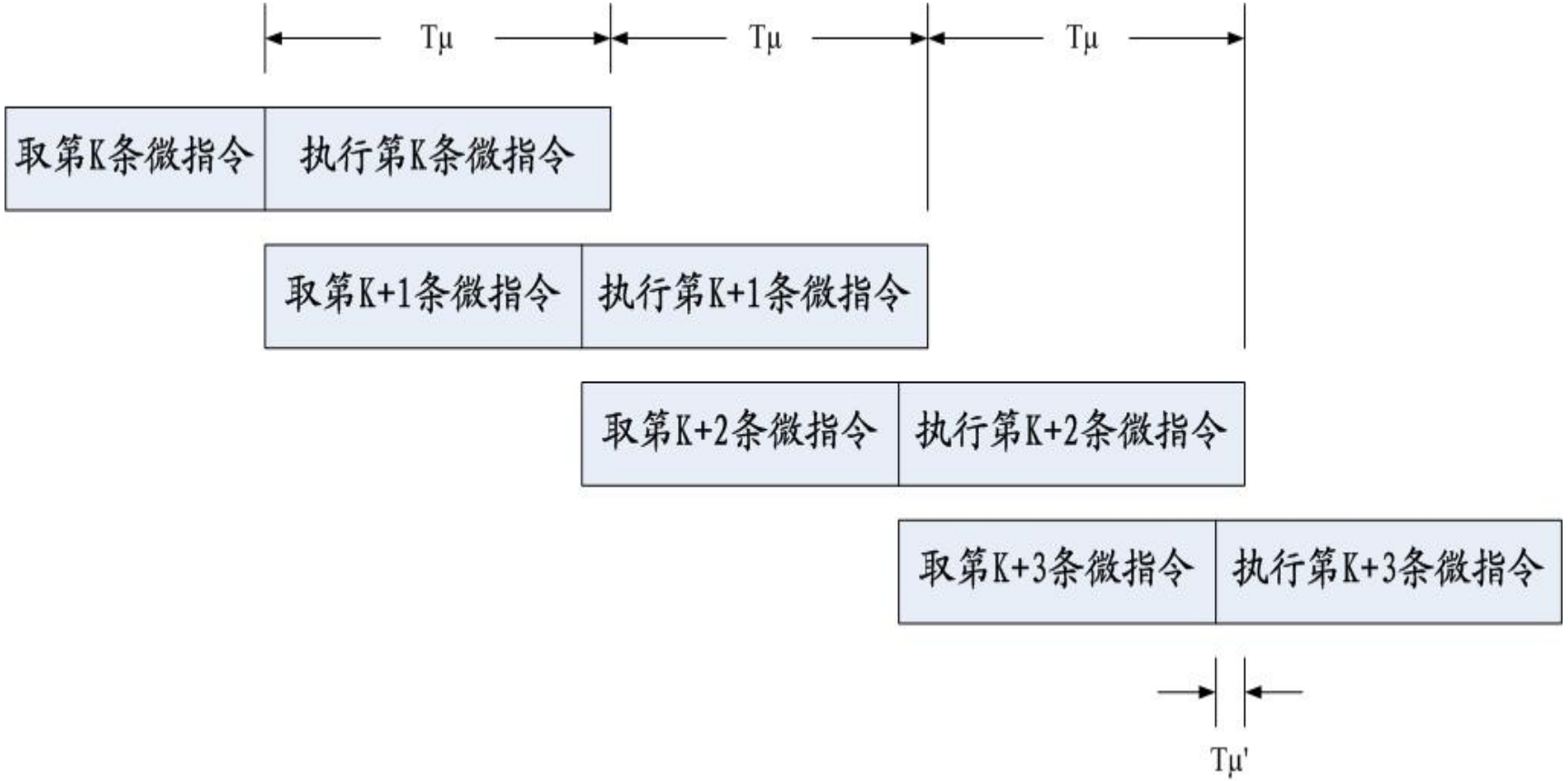
（1）微周期不重叠

如果每条微指令的执行和预取下条微指令的操作都能在一个微周期内完成。如前图。

（2）微周期相重叠

这种方式是微周期不重叠方式的一种改进，每条微指令的操作，在本微周期内不能完成，还借助于下一个微周期的时间里继续进行。如下图。

5.5 微指令格式


Tμ
Tμ
Tμ
取第K条微指令
执行第K条微指令
取第K+1条微指令
执行第K+1条微指令
取第K+2条微指令
执行第K+2条微指令
取第K+3条微指令
执行第K+3条微指令
Tμ'

3.并行执行方式的转移问题

如何解决按运算结果快速进行微程序转移是并行执行方式必需考虑的重要问题。因此，当执行需要根据现行微指令的操作结果，确定下一条微指令地址时，预取下一条微指令就会发生问题（串行执行就不存在这样的问题），这问题的解决难度较大，介绍几种常用方法：

5.5 微指令格式

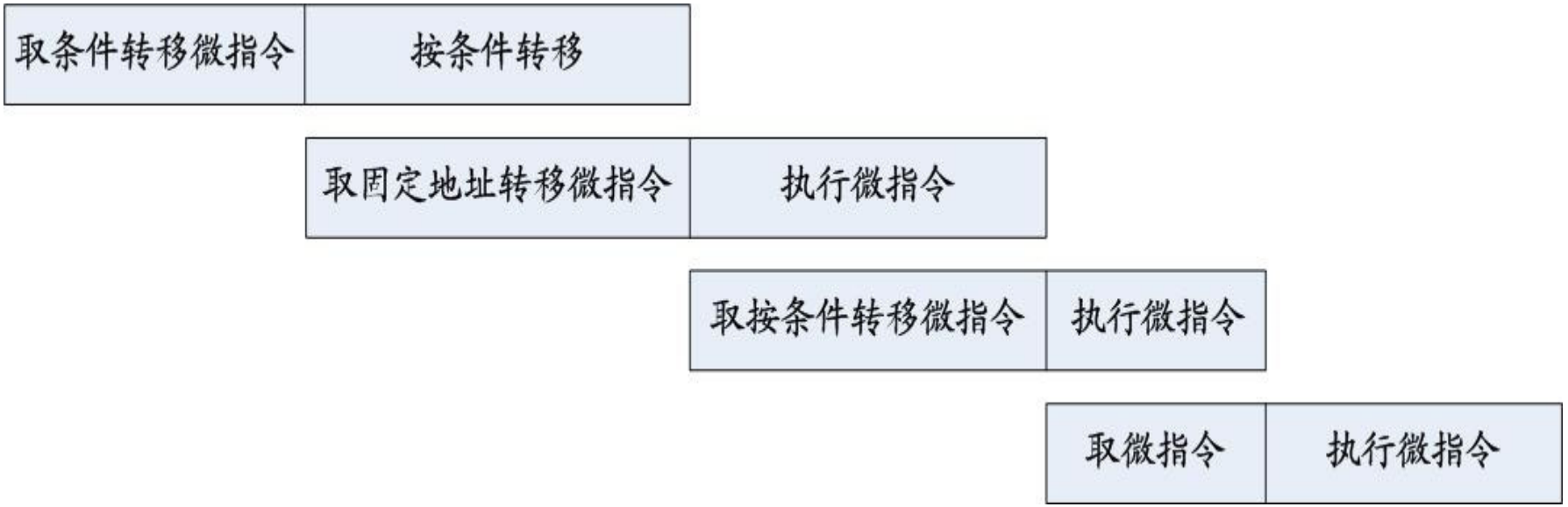
（1）退回到串行执行方式

主要思想：

当遇到按运算结果立即转移的微指令时，改为不立即转移，只将运算结果保存在状态器中，并安排一条专门实现转移的微指令作为下一条微指令，由于这条微指令地址是确定的，用它做缓冲，形成转移地址，就可以继续执行了。

这种处理实际上是退回到串行执行方式。

5.5 微指令格式


取条件转移微指令
按条件转移
取固定地址转移微指令
执行微指令
取按条件转移微指令
执行微指令
取微指令
执行微指令

5.5 微指令格式

由于插入一条微指令，因此多费了一个微周期，当转移微指令很多时，时间损失就大。

假定在并行执行方式下，整个微程序由N条微指令组成，其中有K条按运算结果立即转移的微指令，那么，为解决按运算结果立即转移而花费的总时间为KTμ,微指令由串行改为并行执行而节省的总时间为NTcs，

只有在：KTμ≪NTcs时，并行执行方式才比串行执行方式优越，此外，这种退回串行执行方式的办法还增加了控制存储器容量。

5.5 微指令格式

（2）猜测法

主要思想：

微程序设计者事先猜测一条被执行的可能性最大的微指令，作为下一条要执行的微指令，如果猜测命中，则按正常次序执行下去。如果猜测不命中，则废弃已读的微指令，重新取出按判断结果应执行的微指令，再执行下去。

如猜测成功的命中率高时，此方法损失的时间比退回到串行执行方式少。

5.5 微指令格式

- **假设：Pi表示微程序在第i转移点猜测失败的概率，即Pi=1猜测失败，Pi=0猜测成功。**

如果有K条按运算结果立即转移的微指令，由于猜测失败而额外花费总时间为$\sum_{i=1}^{k} PiT\mu$，而由并行执行改为串行执行而花费的总时间为KTμ。

只有当$\sum_{i=1}^{k} PiT\mu \ll KT\mu$时，猜测法才比退回到串行执行的方法优越。

同样，只有在$\sum_{i=1}^{k} PiT\mu \ll NTcs$时，采用猜测法的并行执行比串行执行方式优越。

5.5 微指令格式

(3)并行读出多条微指令

这种方法是将控存分为多个体或者采用单体多字结构，使一次访问能读出多条微指令，于是在转移点上同时读出多条后续微指令，一旦运算结果判别后，立即从中选取一条执行。

这种方法虽然可使因转移而损失的时间减少至最小，但是构成控存的硬件复杂性增加。

5.5 微指令格式

5.5.5 动态微程序设计

在一台微程序控制的计算机中，假如能根据用户的要求改变微程序，那么这台机器就具有动态微程序设计功能。

动态微程序设计的出发点是为了使计算机能更灵活、更有效地适应于各种不同的应用目标。

- **动态微程序设计定义：**

在采用可改写的高速存储器作为控存的计算机中，通过改写控存中的微程序来变更和使用微程序的一种设计技术。

5.5 微指令格式

- **动态微程序设计思想**

使计算机系统在可改写控存的支持下，能根据不同应用目标需要，随时改变或切换相应的微程序。

由于动态微程序设计要求用户对计算机的结构与组成非常熟悉，因此真正由用户自行编写微程序是很困难的，所以尽管设想很好，事实上难以推广。

5.5 微指令格式

- **微程序的变更方法有两种：**

（1）人工变更

又称低级动态微程序设计，它借助手工操作改变微程序。

缺点：费时间，效率低。

（2）自动变更

又称为高级动态微程序设计，它靠操作系统的管理和控制实现不同应用目标的微程序在主存与可写控存之间的调度，自动实现微程序的变更和切换。

5.6 CPU内部数据通路结构及指令的执行

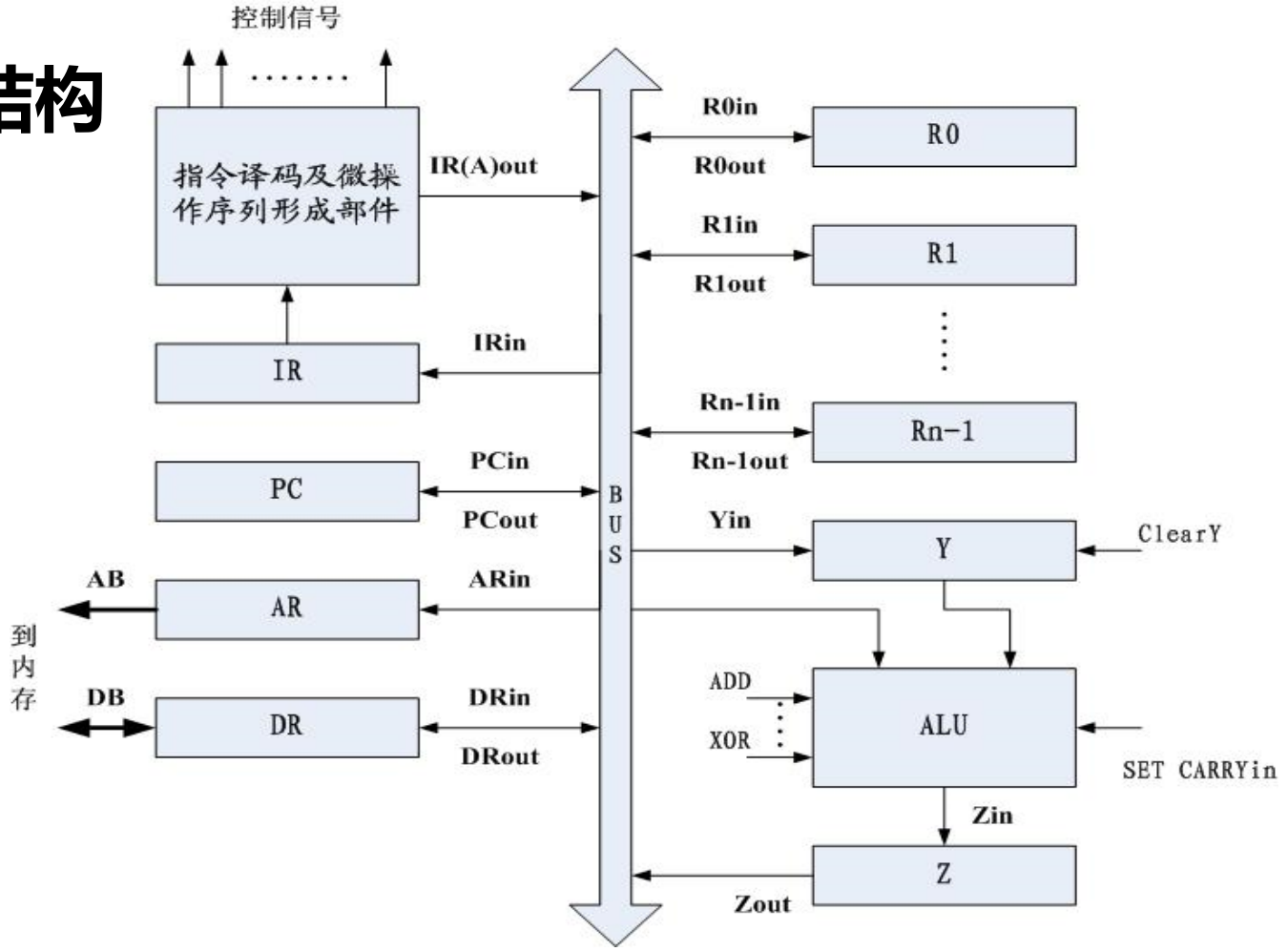
在前面的章节中，已经介绍了CPU中的ALU、寄存器、内部总线、时序电路和微操作信号产生部件。这一节讨论以内部总线为纽带建立各部件间的数据传送通路，即CPU内部数据通路结构，是CPU组成的核心问题。

由于设计目标和方法的不同，CPU内部数据通路结构差异很大。

介绍几种常用结构及该结构下的指令执行。

5.6 CPU内部数据通路结构及指令的执行

5.6.1 单总线结构



5.6 CPU内部数据通路结构及指令的执行

1.执行一条指令

CPU要完成一条指令的执行，需要完成下列三步：

（1）IR←((PC))

（2）PC←（PC）+1

（3）执行在IR中的指令所规定的动作

假设：每条指令占一个单元，（PC）+1。

（1）、（2）步的操作为取指过程，或称取指周期

（3）步为执行过程，或称执行周期

2.从存储器中“读”一个字

在随机存储器中，每个存储单元都有一个唯一的地址，因此，从存储单元中取一个信息字，CPU必需指定存储单元的地址，并发“读”操作。

从存储器中读出的可以是条指令，也可以是个操作数。

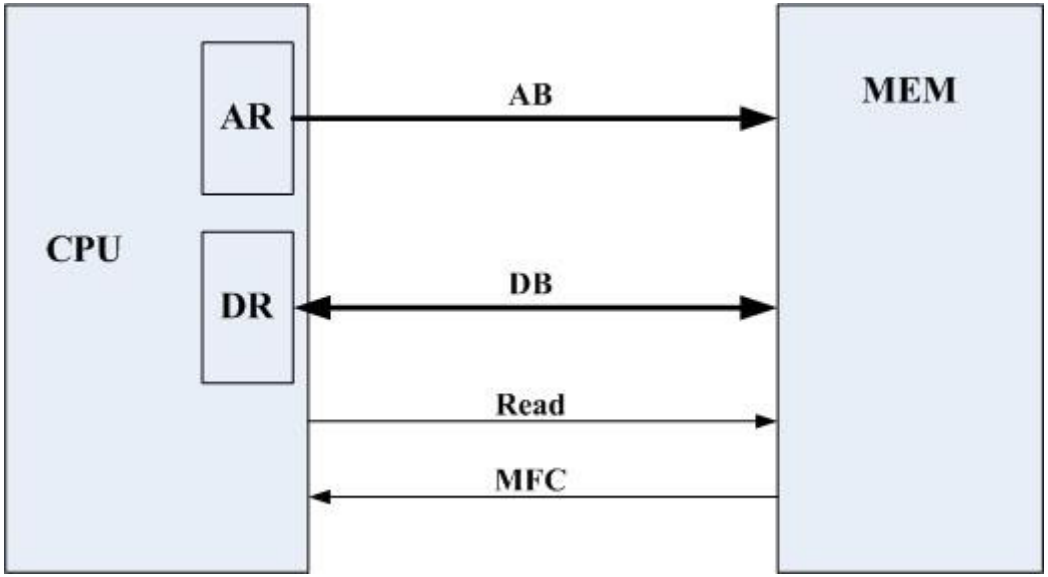
CPU完成读操作过程如下：

(1) 内存地址送AR

(2) 通过控制线上发“读”操作

(3) 等待MFC（Memory Function Completed）

5.6 CPU内部数据通路结构及指令的执行



例：假定要访问的存储单元地址在寄存器R1中，取出数据送到R2中。

（1）R1out,ARin,Read,Wait MFC

（2）DRout,R2in

例：从内存中读取一条指令

（1）PCout,ARin,Read,Wait MFC

（2）DRout,IRin

“等待”时间取于存储器的工作速度，主存工作速度快，“等待”时间短，反之，“等待”时间长。

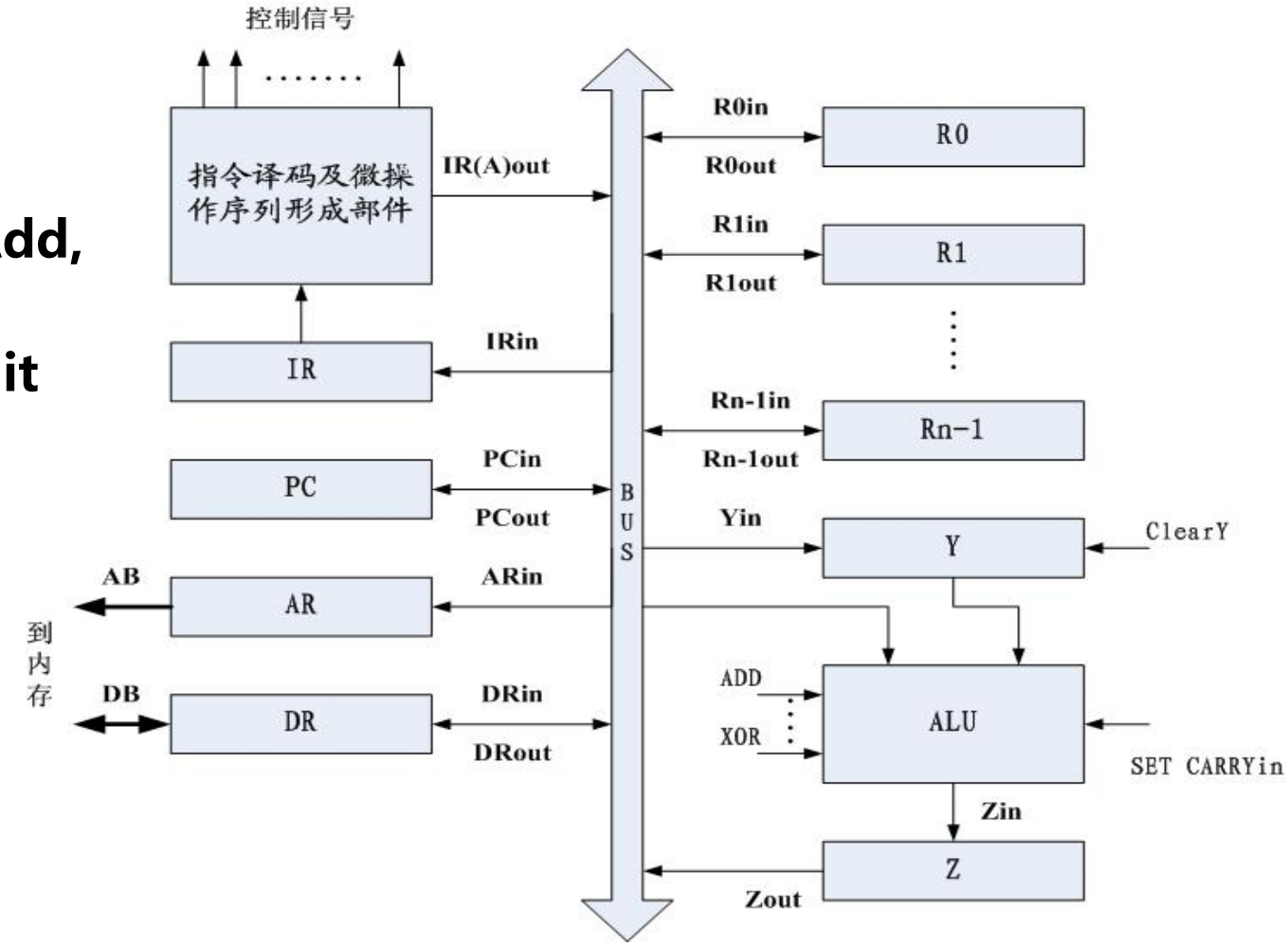
为了提高执行速度，提高CPU利用率，在CPU“等待”这段时间里，让CPU完成一些与DR、AR无关的操作，如PC增值。

5.6 CPU内部数据通路结构及指令的执行

(1) PCout,ARin, Read,ClearY, SetCarryin, Add, Zin

(2) Zout,PCin,Wait MFC

(3) DRout,IRin



3.“写”一个字到主存中

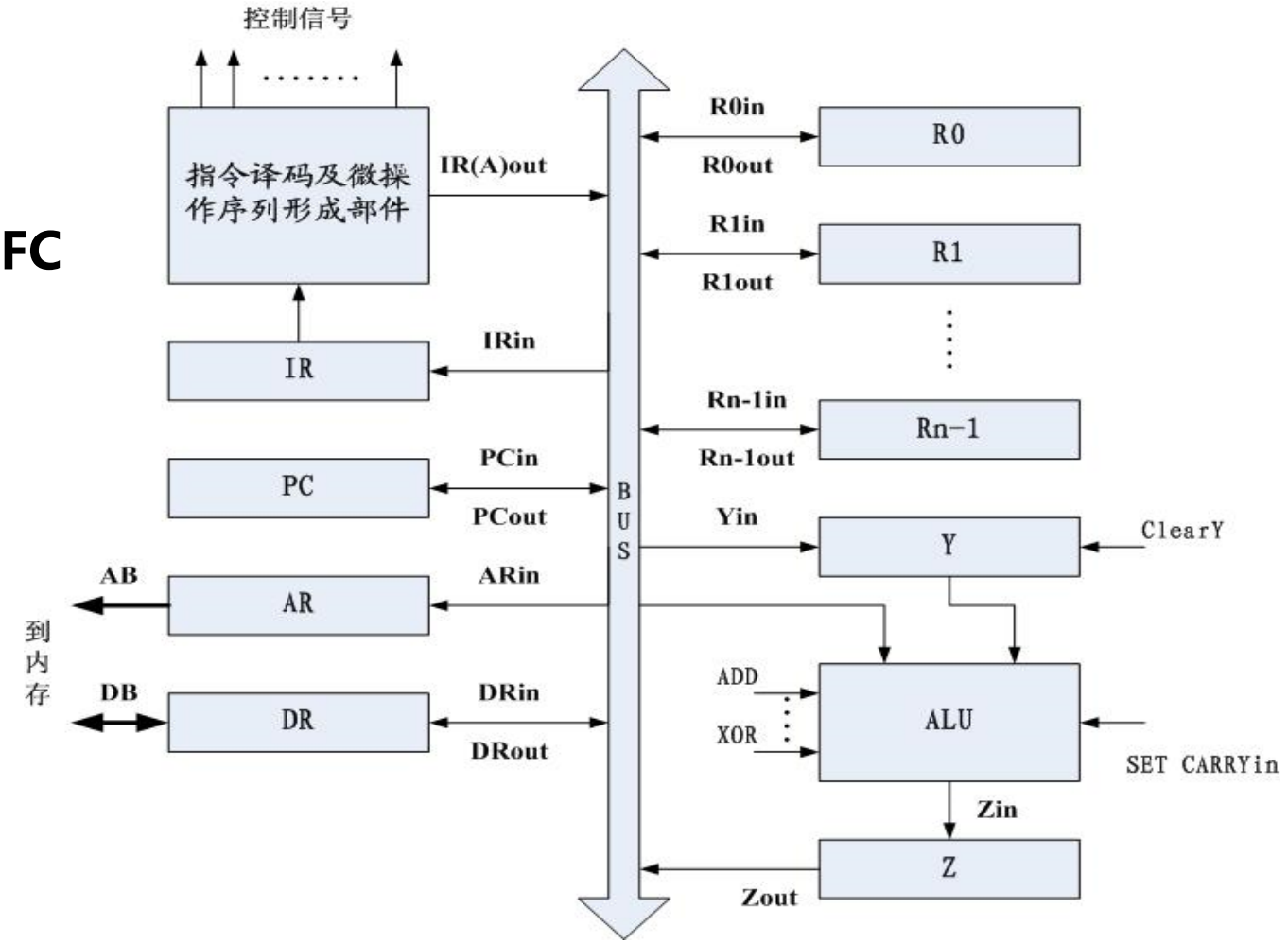
写入一个字到给定单元中的过程和从存储器中读一个字的过程相类似，其唯一差别是被写入的数据字应在发出“写”控制信号之前送入DR。

例：把R2的内容写入R1所指出的地址单元中。

5.6 CPU内部数据通路结构及指令的执行

(1) R1out,ARin

(2) R2out,DRin,
Write,Wait MFC



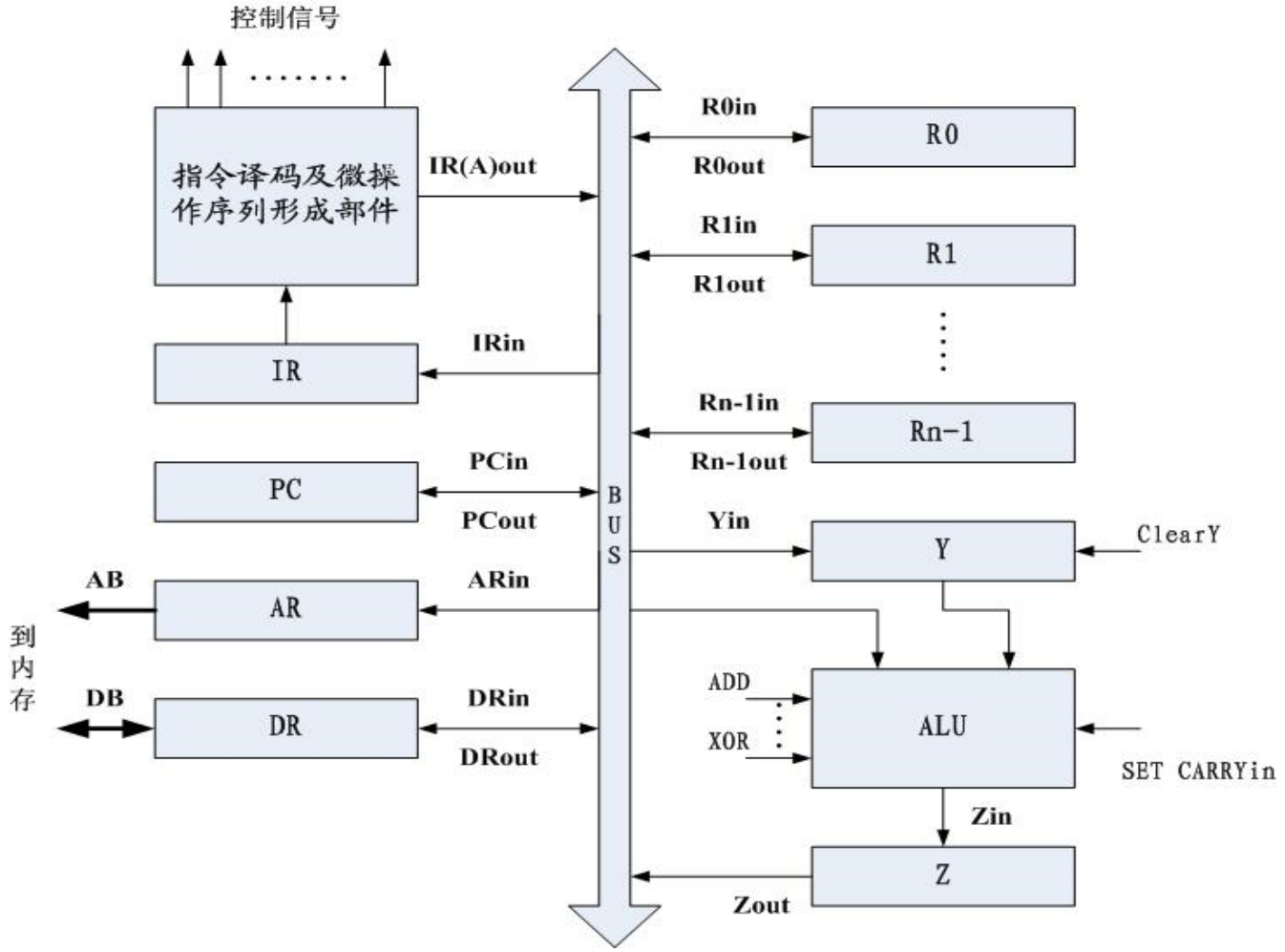
4.寄存器间传送

例：将R1内容送R3。

（1）R1out,R3in

5.6 CPU内部数据通路结构及指令的执行

(1) R1out,R3in



5.算术逻辑运算

执行算术逻辑运算时，ALU两输入端的数据必须同时有效。

例：将R1中内容加上R2中内容，结果送R2。

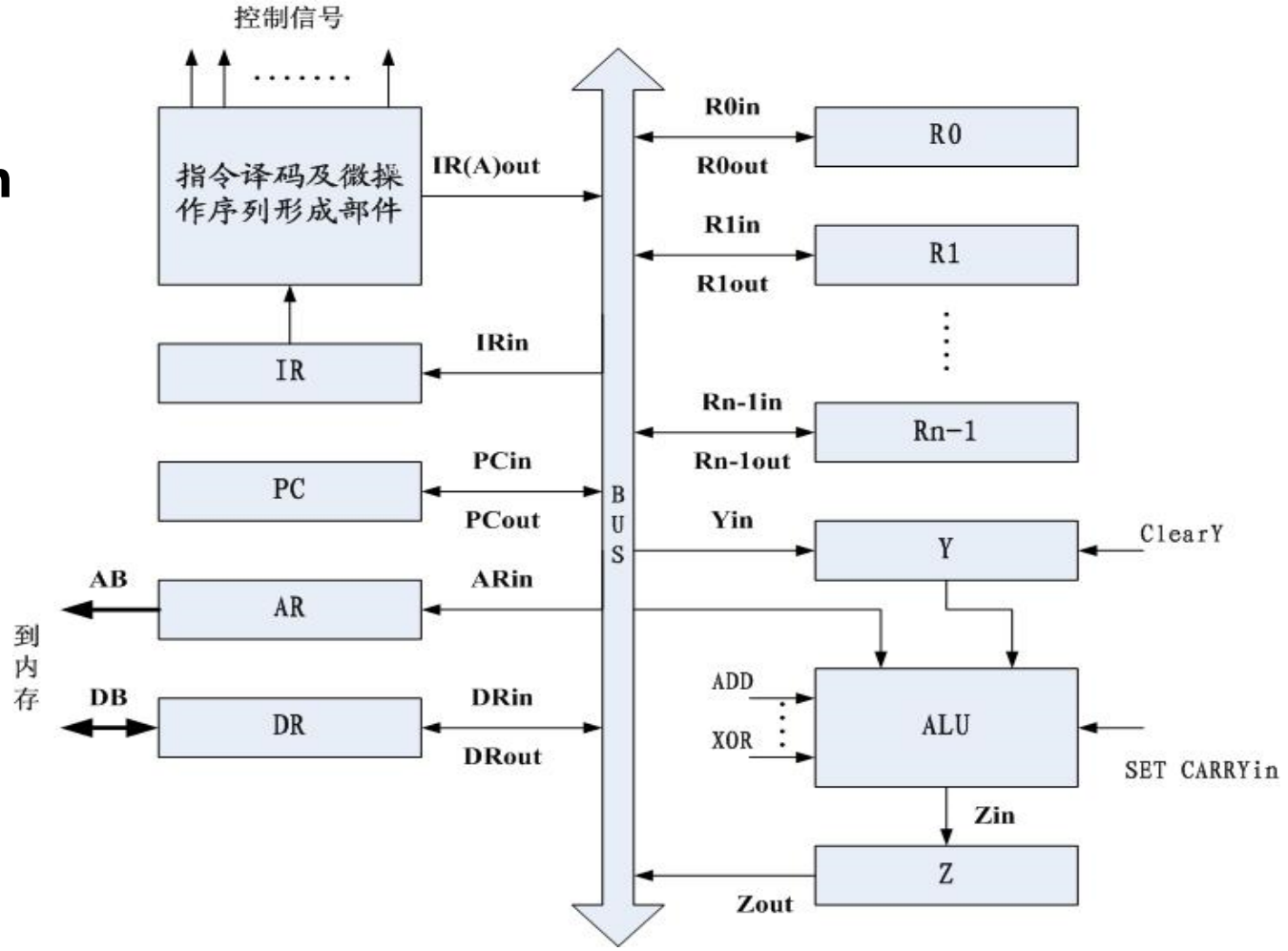
(1) R1out,Yin

(2) R2out,Add,Zin

(3) Zout,R2in

5.6 CPU内部数据通路结构及指令的执行

(1) R1out,Yin
(2) R2out,Add,Zin
(3) Zout,R2in



5.6.2 指令的执行

1.ADD指令

ADD Loca,R1; 将内存单元地址为Loca的内容加上R1内容，结果送R1。

指令格式:

ADD	Loca	R1

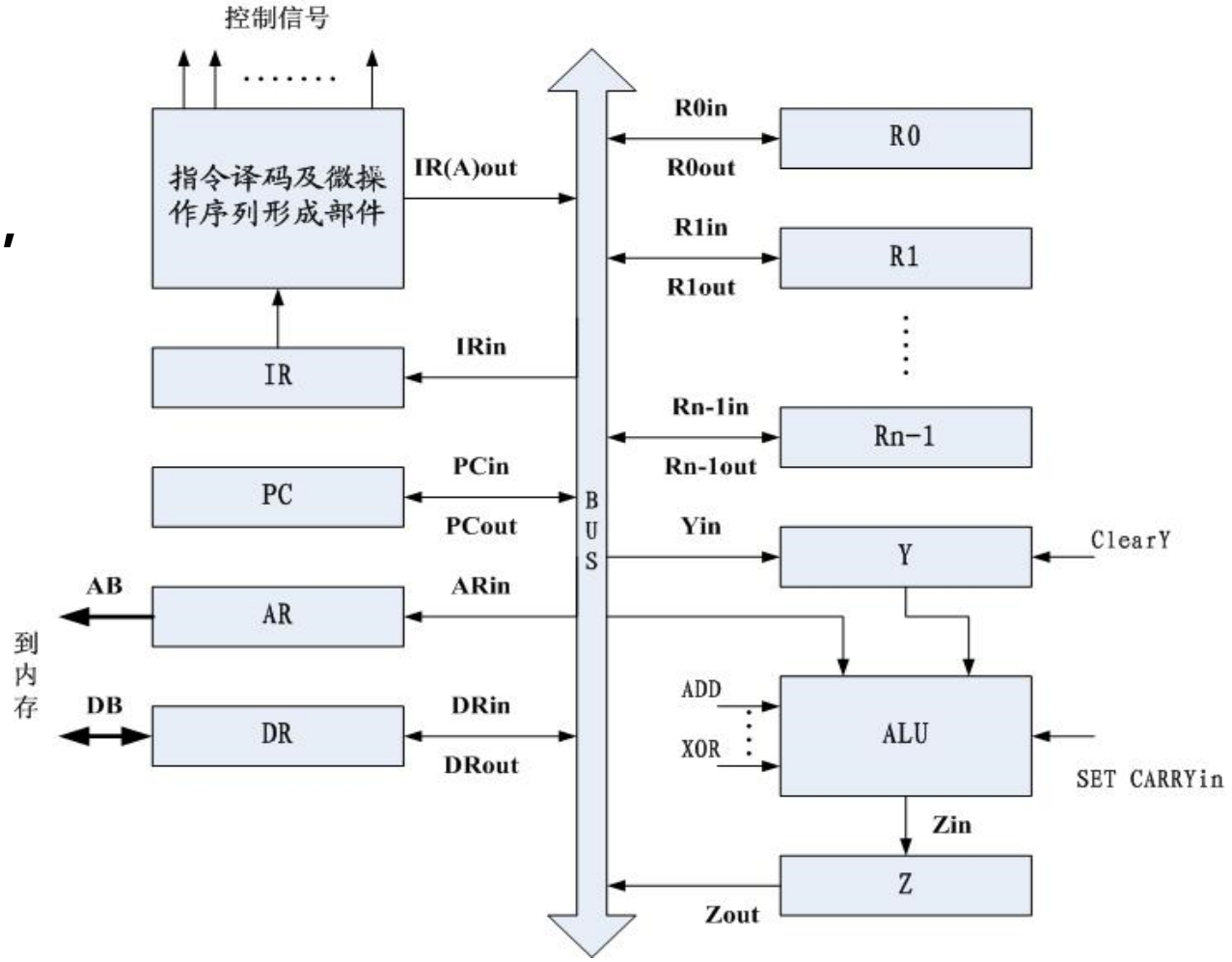
执行ADD指令需要下列操作:

(1) 取指令 (2) 取第一操作数 (3) 完成加法

(4) 结果送R1

5.6 CPU内部数据通路结构及指令的执行

- ADD微程序如下：
  - (1) PCout,ARin,Read, ClearY,Set Carryin, Add,Zin
  - (2) Zout,PCin,Wait MFC
  - (3) DRout,IRin
  - (4) IR（A）out,ARin, Read
  - (5) R1out,Yin, Wait MFC
  - (6) DRout,Add,Zin
  - (7) Zout,R1in
  - (8) End



2.转移指令

无条件相对转移指令

JMP D； （PC）+D→PC

指令格式：

JMP	D

5.6 CPU内部数据通路结构及指令的执行

JMP微程序如下：

(1) PCout,ARin,Read, ClearY,Set Carryin, Add,Zin

(2) Zout,PCin,Wait MFC

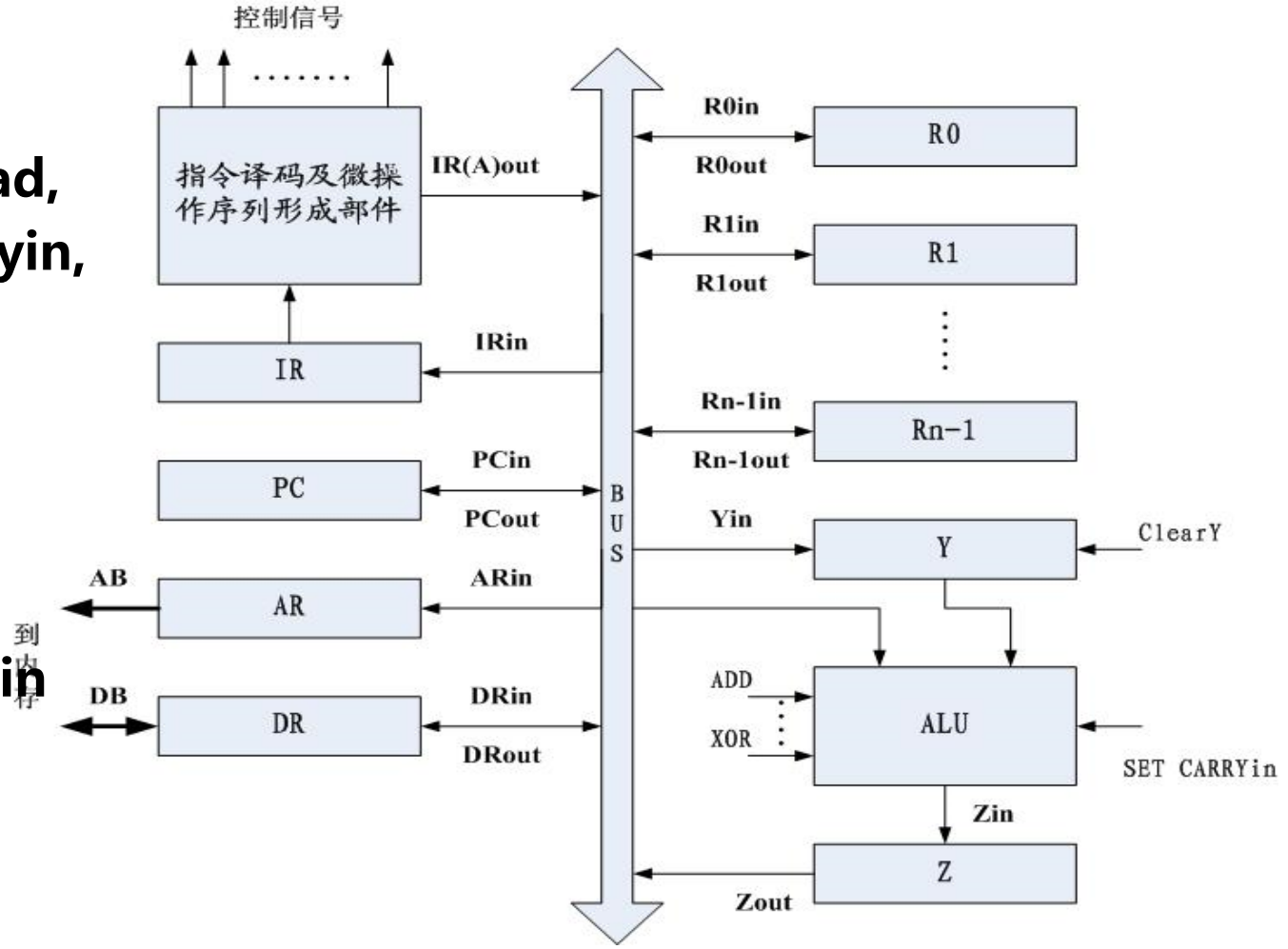
(3) DRout,IRin

(4) PCout,Yin

(5) IR(A)out,Add,Zin

(6) Zout,PCin

(7) End



条件转移指令

JNZ D ；如Z=0产生转移，（PC）+D→PC

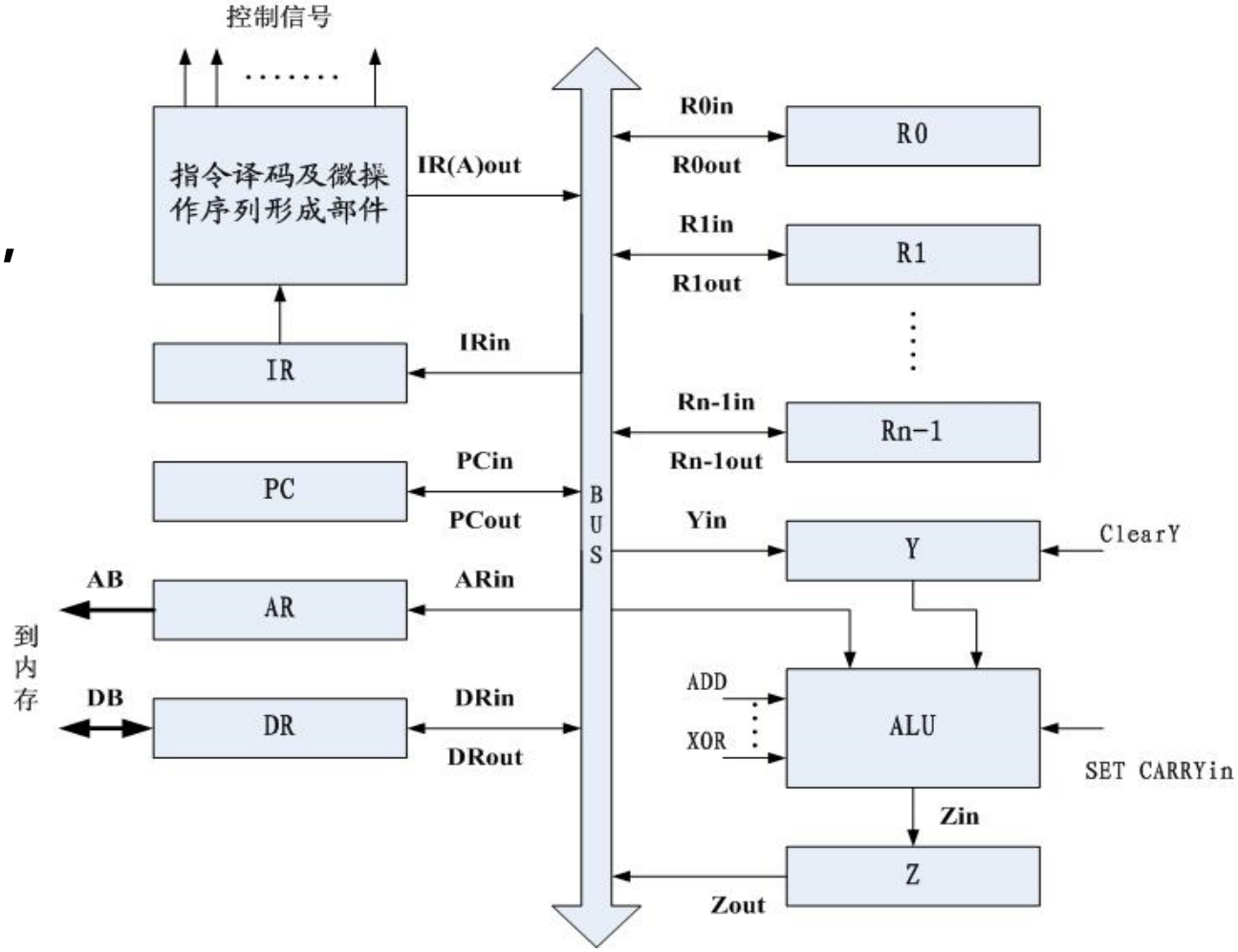
指令格式：

JNZ	D

5.6 CPU内部数据通路结构及指令的执行

JNZ微程序如下：

(1) PCout,ARin,Read, ClearY,Set Carryin, Add,Zin
(2) Zout,PCin,Wait MFC
(3) DRout,IRin
(4) IF Z=1 THEN End
(5) PCout,Yin
(6) IR(A)out,Add,Zin
(7) Zout,PCin
(8) End



3.数据传送指令

MOV R1，R2；　(R1) →R2

指令格式:

MOV	R1	R2

5.6 CPU内部数据通路结构及指令的执行

MOV微程序如下：

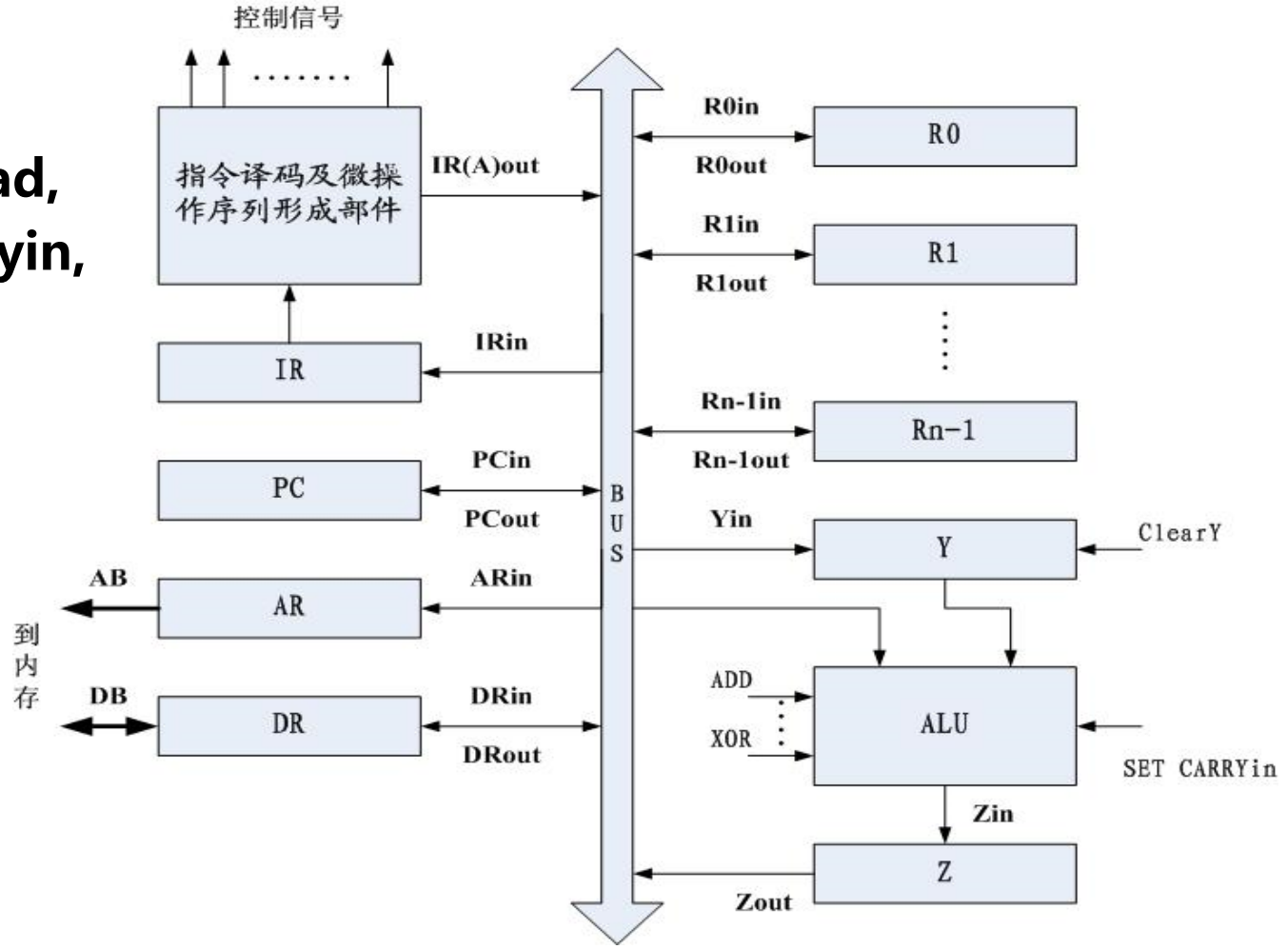
(1) PCout,ARin,Read, ClearY,Set Carryin, Add,Zin

(2) Zout,PCin,Wait MFC

(3) DRout,IRin

(4) R1out,R2in

(5) End



4.逻辑指令

XOR （R3），（R4）;((R3))XOR((R4))→(R4)

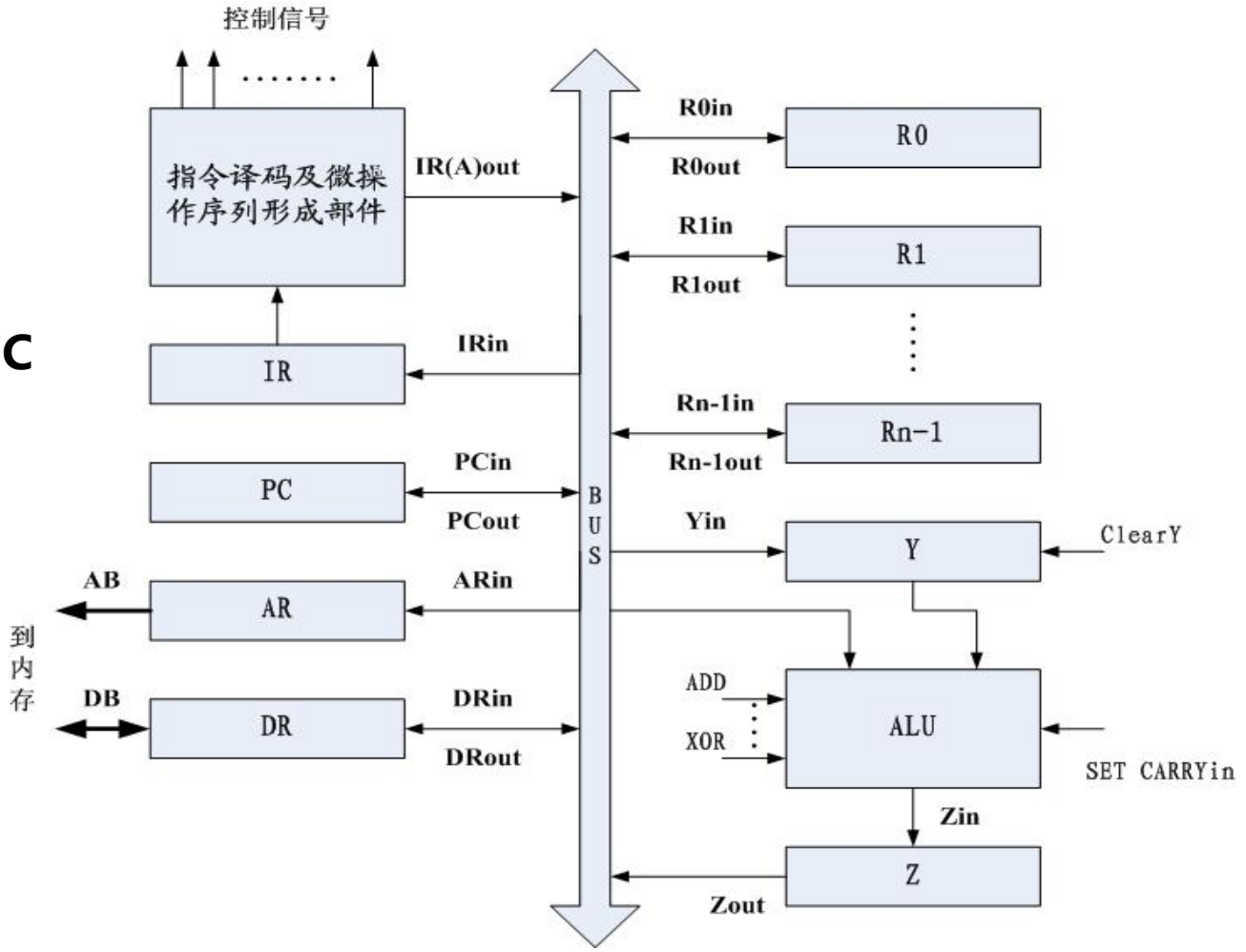
指令格式:

XOR	R3	R4

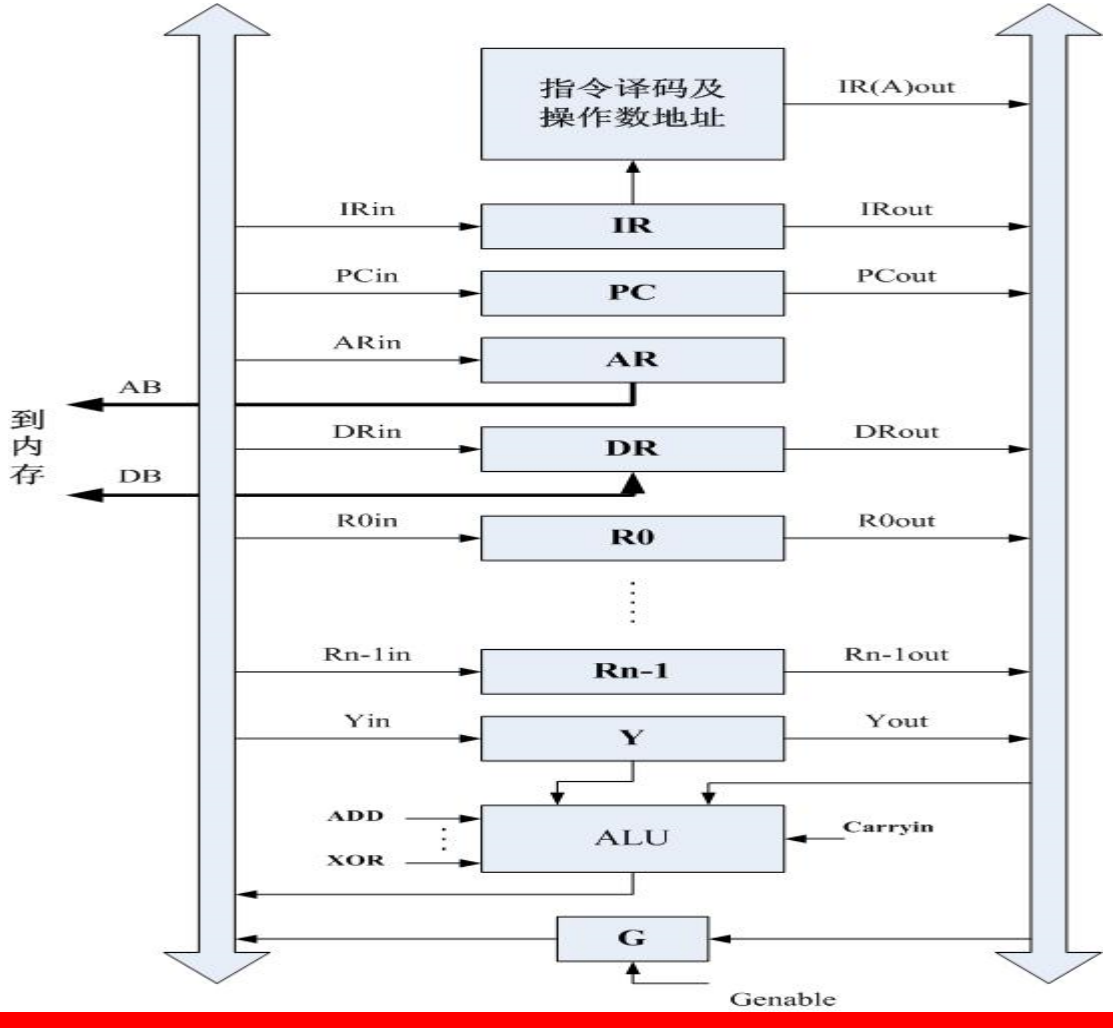
5.6 CPU内部数据通路结构及指令的执行

XOR微程序如下:

(1) PCout,ARin,Read, ClearY,Set Carryin, Add,Zin
(2) Zout,PCin,Wait MFC
(3) DRout,IRin
(4) R3out,ARin,Read, Wait MFC
(5) DRout,Yin
(6) R4out,ARin,Read, Wait MFC
(7) DRout,Xor,Zin
(8) Zout,DRin,Write, Wait MFC
(9) End



5.6.3 双总线结构


指令译码及
操作数地址
IR(A)out
IRin
IR
IRout
PCin
PC
PCout
ARin
AR
AB
到
内
存
DB
DRin
DR
DRout
R0in
R0
R0out
Rn-1in
Rn-1
Rn-1out
Yin
Y
Yout
ADD
XOR
ALU
Carryin
G
Genable

5.6.4 双总线CPU指令举例

1.ADD指令

指令格式和功能同前。

5.6 CPU内部数据通路结构及指令的执行

ADD微程序如下:

(1) PCout,Genable,ARin, Read,ClearY

(2) PCout,Set Carryin,Add, PCin,Wait MFC

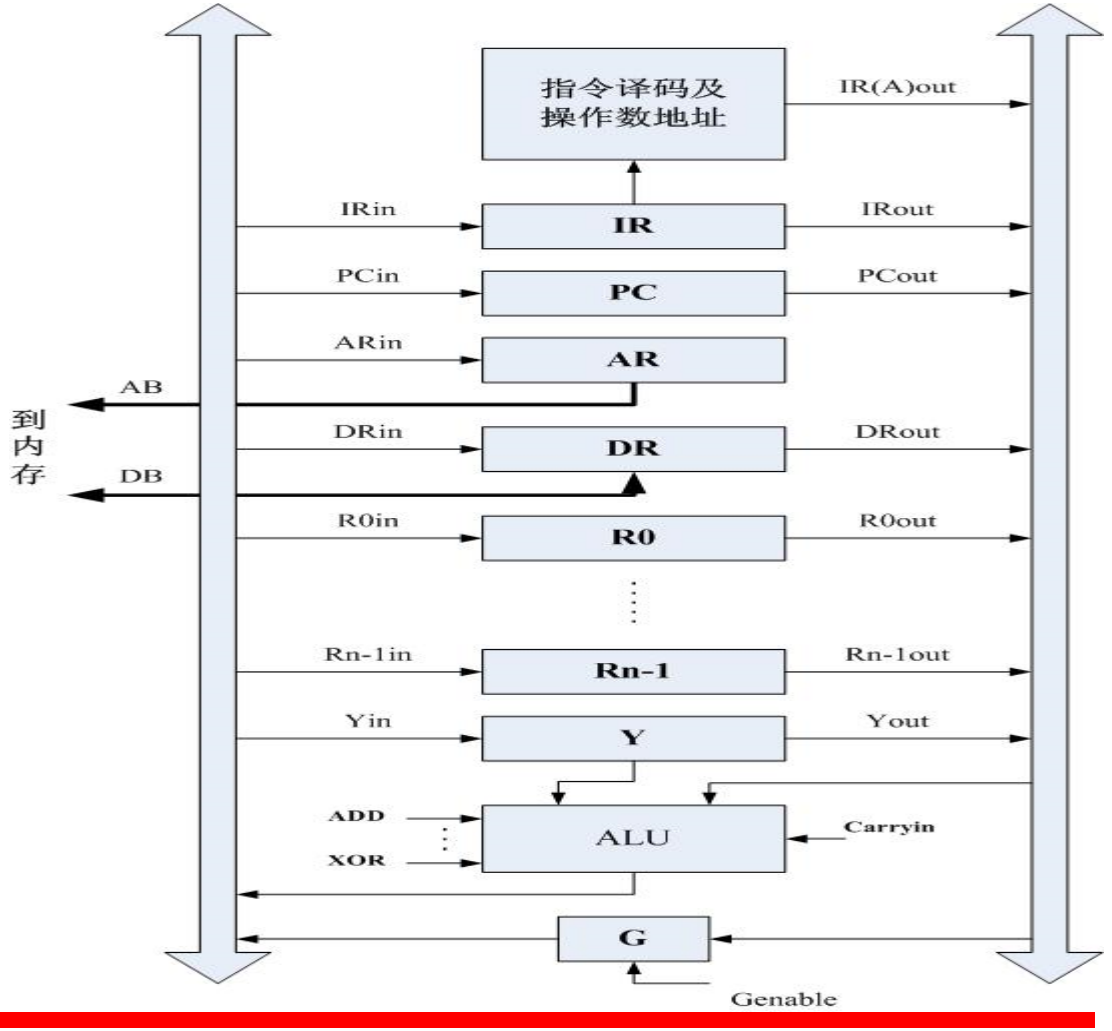
(3) DRout, Genable,IRin

(4) IR (A) out, Genable, ARin,Read

(5) R1out, Genable,Yin, Wait MFC

(6) DRout,Add,R1in

(7) End



2.条件转移指令

结果为零转移

JZ D；如Z=1成立产生转移，（PC）+D→PC

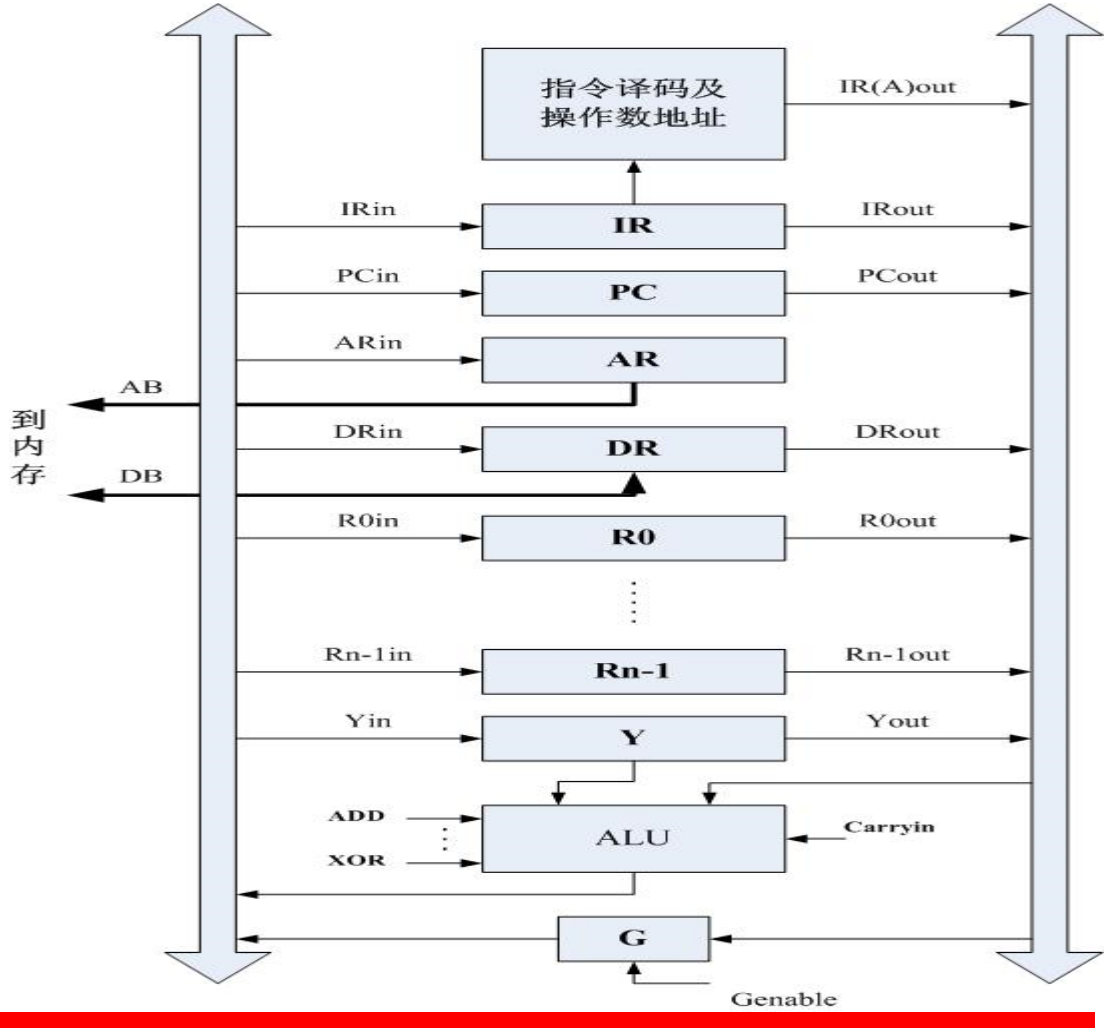
指令格式：

JZ	D

5.6 CPU内部数据通路结构及指令的执行

JZ微程序如下:

(1) PCout,Genable,ARin, Read,ClearY
(2) PCout,Set Carryin,Add, PCin,Wait MFC
(3) DRout, Genable,IRin
(4) IF Z=0 THEN End
(5) PCout,Genable,Yin
(6) IR(A)out,Add,PCin
(7) End



3.数据传送指令

MOV (R1), R2 ;((R1))→R2

指令格式:

MOV	R1	R2

5.6 CPU内部数据通路结构及指令的执行

MOV微程序如下:

(1) PCout,Genable,ARin, Read,ClearY

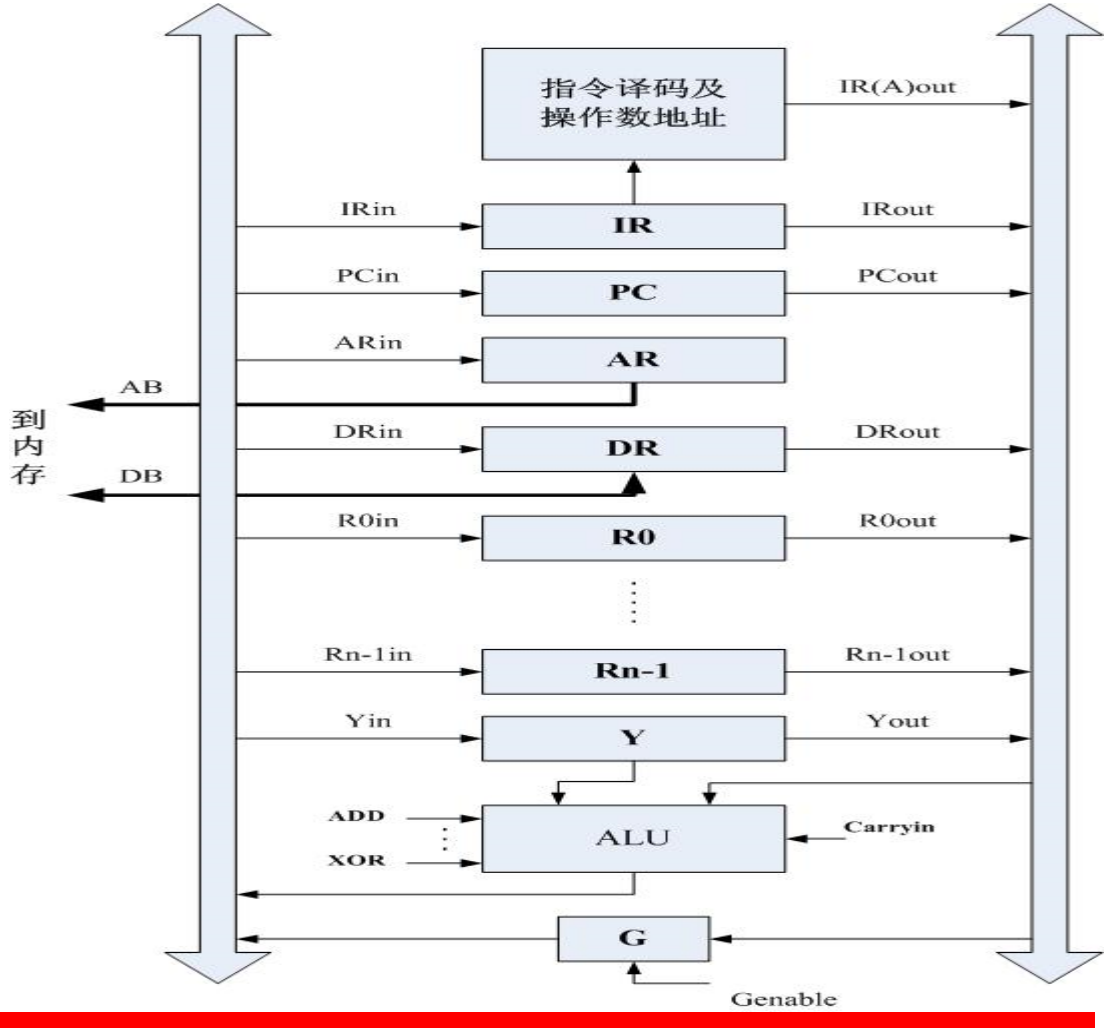
(2) PCout,Set Carryin,Add, PCin,Wait MFC

(3) DRout, Genable,IRin

(4) R1out, Genable,ARin, Read,Wait MFC

(5) DRout, Genable,R2in

(6) End



5.7 流水线工作原理

为了提高计算机的运行速度，加快指令的执行，可以选用高速器件或提高CPU的频率来实现。除此之外，还可以采用某些技术来提高CPU的运行速度，流水线技术就是其中之一。

5.7.1流水线工作原理

计算机执行程序是按顺序的方式进行的，即程序中各条机器指令是按顺序串行执行的。

取指1	执行1	取指2	执行2	取指3	执行3

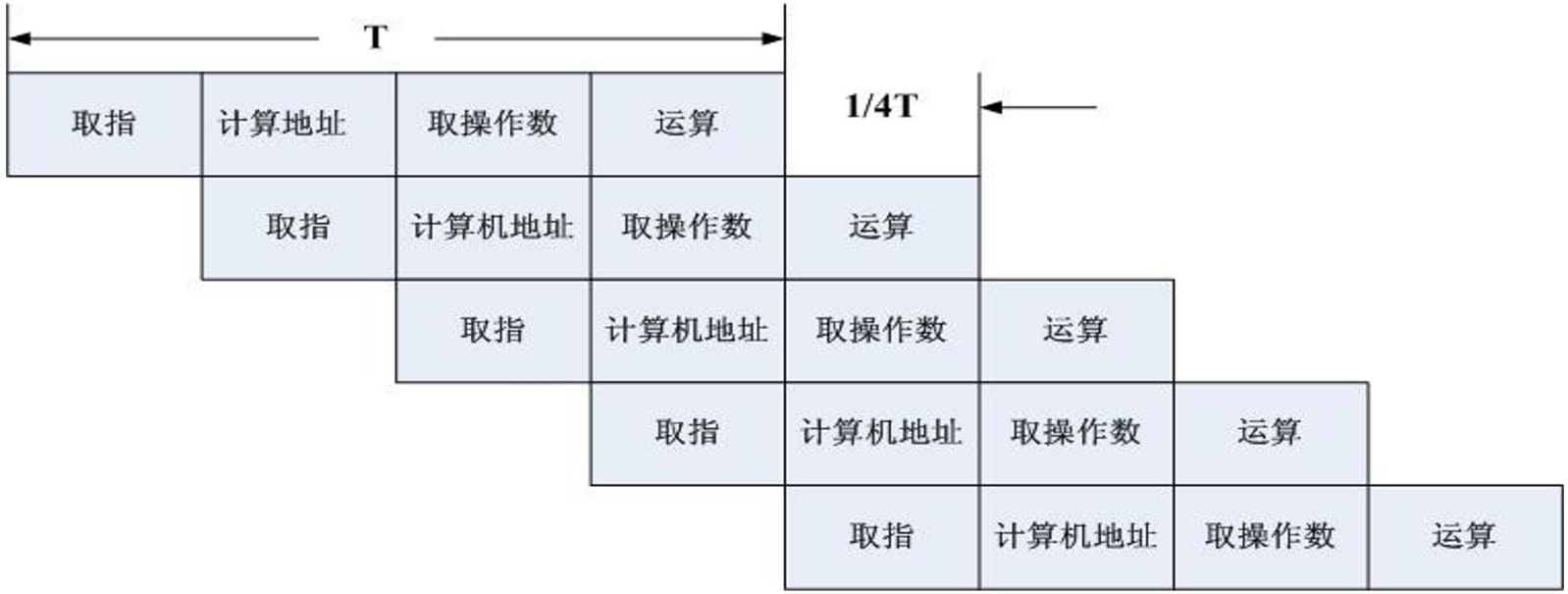
5.7 流水线工作原理

可以看出，程序是按指令的顺序执行完一条再执行下一条的。顺序执行的优点是控制简单，但是机器各部分的利用率不高。例如，指令部件工作时，执行部件基本空闲；而执行部件工作时，指令部件基本空闲。假如我们把上条指令的执行和取下条指令的操作，在时间上重叠起来进行，将大幅度提高程序的执行速度。

5.7 流水线工作原理

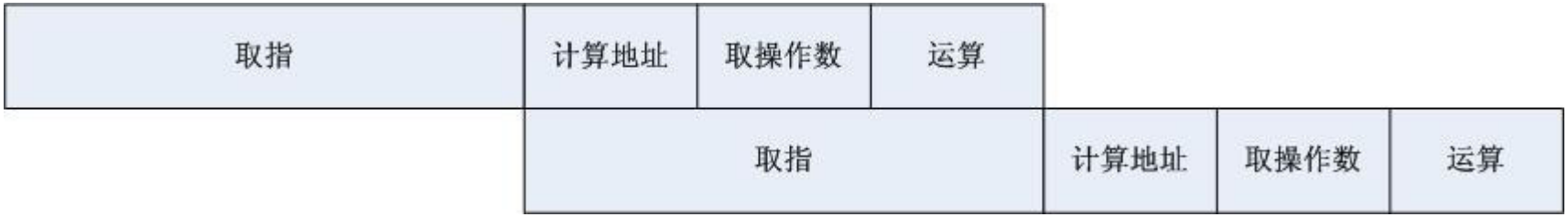
5.7 流水线工作原理

如将一条指令的取指到执行分成4段，除第一条指令外，以后每1/4T就完成一条指令的执行，效率提高4倍。



5.7 流水线工作原理

如果4段的时间长度是不一致的话，极端情况下，分成4段后，效率没有提高。



所以，在时间划分上，最好每段长度基本上一致。

对于流水线的流动顺序控制可有两种方法：

（1）顺序流动，即流入的顺序与流出的顺序相同。

（2）异步流动，即流入的顺序与流出的顺序不一致。

5.7 流水线工作原理

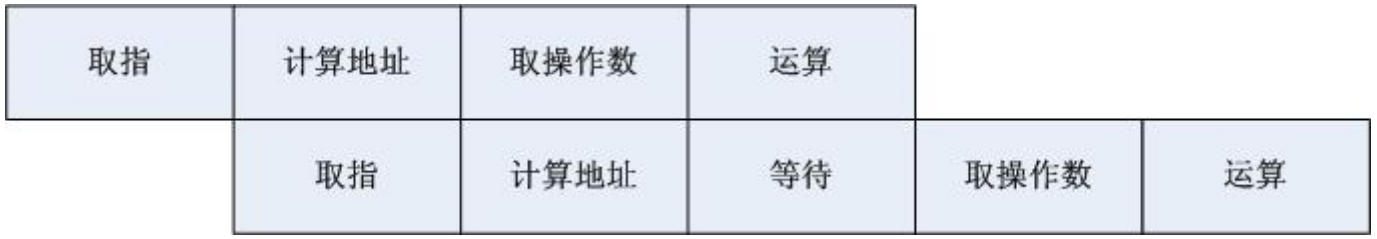
5.7.2流水线中的相关问题

流水线不能连续工作的原因，除了编译形成的程序不能发挥流水线的作用或存储器供应不上为连续流动所需的指令和数据以外，还因为出现了“相关”的问题。例如，在4级流水线中，假如第2条指令的执行要用到第一条指令的结果。

ADD R0，R1

ADD R2，R0

5.7 流水线工作原理



那么取操作数的动作需要等待t时间才能进行，否则取得的数据是错误的，这种情况称为数据相关，因数据可以是存放在存储器中或通用寄存器中，所以，分别称为存储器数据相关或寄存器数据相关。

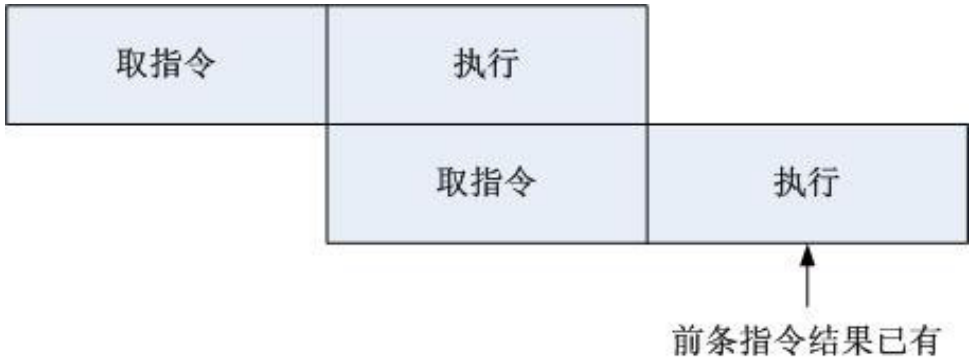
这种现象称为流水线阻塞。

5.7 流水线工作原理

- **解决方法：**

一般碰到数据相关时，就直接到数据处理部件去取数据，而不是等到数据存到内存或寄存器中再去取，这样可提高效率，但控制变为复杂。

一般情况，流水线级数越多，情况越复杂，但二级流水线则不存在数据相关现象。



5.7 流水线工作原理

5.7.3程序转移对流水线的影响

在大多数流水线机器中，当遇到条件转移指令时，确定转移与否的条件码往往由条件转移指令本身或由它前一条指令形成，只有当它流出流水线时，才能建立转移条件并决定下条指令地址。因此当条件转移指令进入流水线后直到确定下一地址之前，流水线不能继续处理后面的指令而处于等待状态，因而影响流水线效率。

解决的方法同微指令解决的方法相同。

5.7 流水线工作原理

5.7.4中断对流水线的影响

在计算机运行时，当I/O设备有中断请求或机器有故障中断请求时，要求中止当前程序的执行而转入中断处理程序。在流水线机器中，在流水线中已存在几条指令，因此就有一个如何“断流”的问题。当I/O系统提出中断时，可以考虑把流水线中的指令全部完成。但当出现诸如地址错、存储器错、运算错而中断时，假如这些错误是由第i条指令发生的，那么在其后的虽已进入流水线的第i+1条指令、第i+2条指令，…也是不应该再执行的。在第i条执行完后，立即响应中断。

5.7 流水线工作原理

流水线机器处理中断的方法有两种：不精确断点法和精确断点法。

不精确断点法---- 对中断时还未进入流水线的后续指令不允许其再进入，但已在流水线中的所有指令则仍执行完毕，然后转入中断处理程序。

精确断点法---- 在当前指令执行完后，进入中断处理程序，对在流水线中未执行完的指令，将其所有状态保存起来，等中断程序执行完后，恢复所有状态再执行下去。

习　题（2）

百年同濟
TONGJI UNIVERSITY

P161

习题：2，5，6，7，8，9，14，17，18