东南大学软件学院

计算机系统组成

主讲教师： 徐造林

第六章　中央处理器

※主要内容

(1)CPU的组成与工作流程：功能(需求)，基本组成，工作流程，
指令的执行过程(需求)

(2)数据通路的组织：DP基本组成(部件/互连/μOP控制)，
指令执行过程的组织(基于μOP)，
DP设计方法，单周期DP、多周期DP设计

(3)控制器的组成：基本结构，时序信号形成(信号序列/定时)，
μOP控制信号形成

(4)硬布线CU设计：设计步骤，单周期CU设计，多周期CU设计

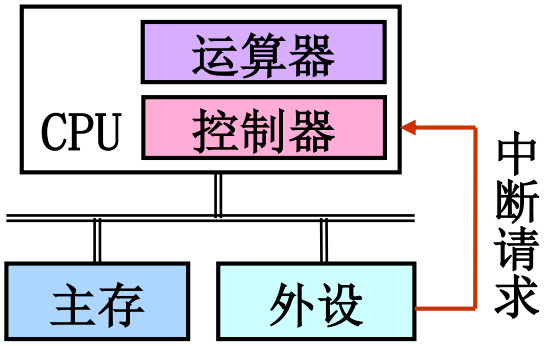
(5)微程序CU设计：微程序控制思想，组成及工作原理，
微指令格式，CU设计

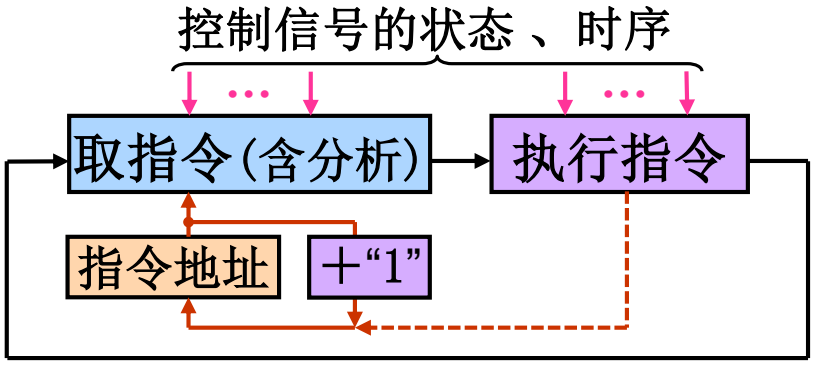
(6)异常及中断的处理：基本概念，处理过程，CPU的相应设计

(7)指令流水线技术：组成及性能，冒险处理，并行技术

§6.1 CPU的组成与工作流程

一、CPU的功能





(1)指令控制：控制指令的执行顺序(执行过程、下条指令地址)

(2)操作控制：产生指令执行所需的操作控制信号

(3)时间控制：控制操作控制信号的时序(时长及次序)

(4)数据加工：实现指令约定的数据运算(即指令系统的运算功能)

(5)外部访问：实现对存储器、外设的访问

(6)异常及中断处理：实现异常及中断的检测及处理

二、CPU的组成

1、CPU的基本组成

*基本部件：指令控制— PC、IR、ID

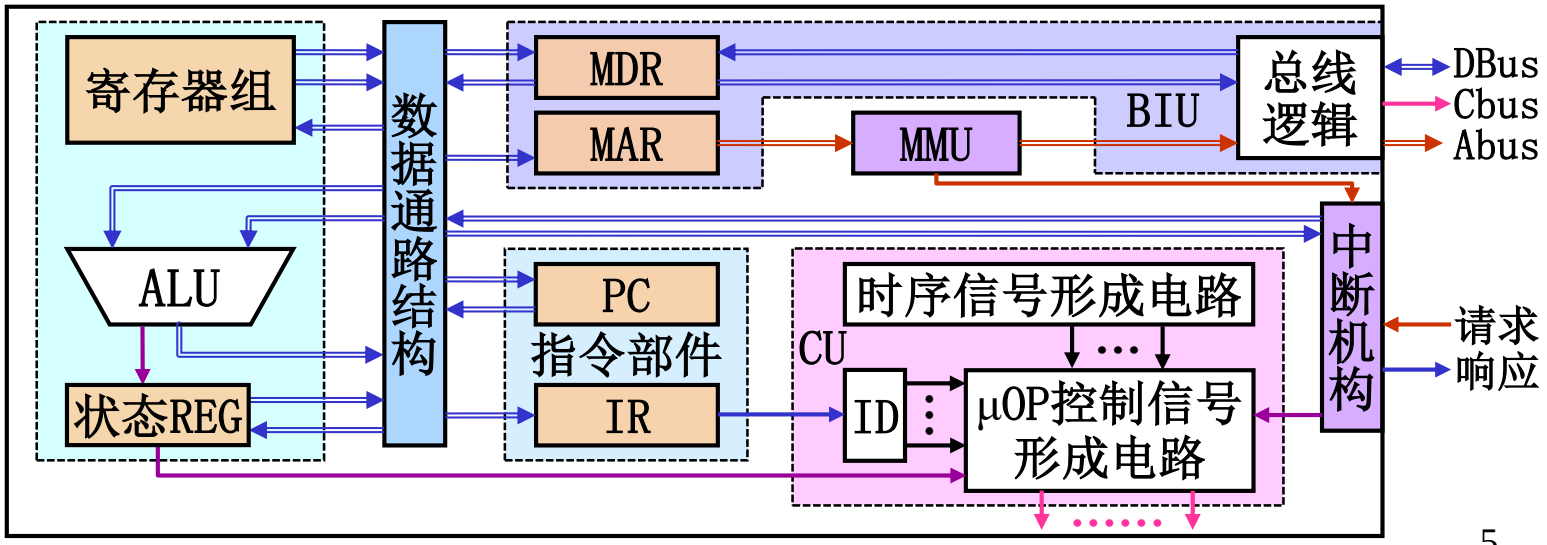
操作及时间控制— 时序信号、操作控制信号形成电路

数据加工— ALU、FPU、状态REG、REG组等

外部访问— 总线逻辑电路、缓冲寄存器，MMU

中断处理— 中断机构

*基本结构：运算器、BIU、MMU、指令部件、CU、中断机构



2、CPU的寄存器组织

***用户可见寄存器：** —程序中存放地址及数据

数据REG—存放操作数(OPD)，长度＝机器字长*n*

累加REG(AC)：指令中同时用作源及目的OPD的REG

地址REG—存放OPD地址或指令地址，

长度＝逻辑地址位数*m*(或段号、段内地址的位数)

通用REG(GPR)—可用作数据REG和地址REG

→长度相同(*m*≤*n*)

状态REG(PSR)—存放程序执行的状态，又称标志REG

结果状态标志：ZF/CF/SF/OF，常用作条件码(可修改)

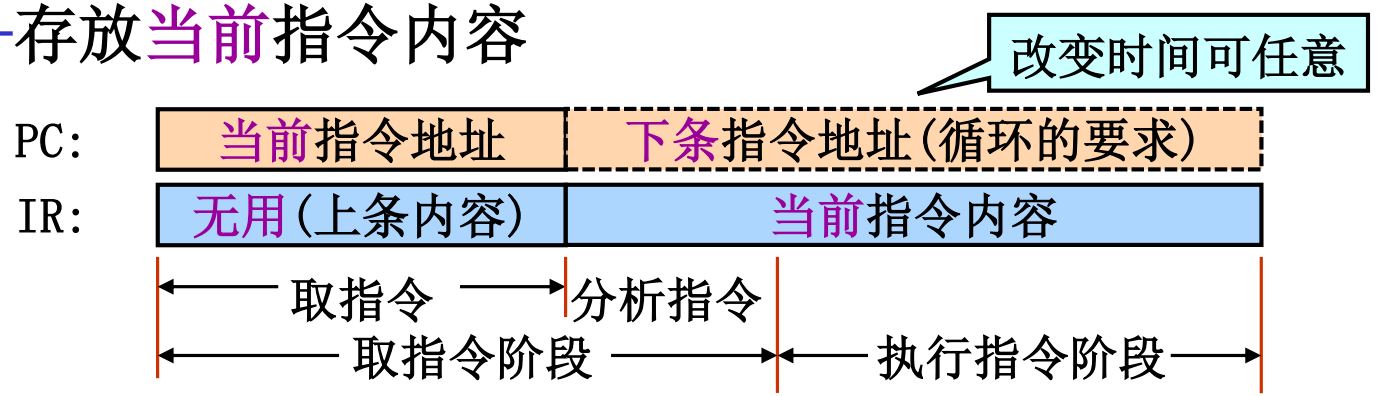
执行方式标志：跟踪标志TF、中断允许标志IF

(如TF＝1时单步执行) (如IF＝0时禁止中断)

*专用寄存器：　—控制CPU的操作和运算

PC—存放指令地址，用作循环变量

IR—存放当前指令内容



MAR—存放CPU外部访问的部件地址(MEM或I/O设备)

MDR—存放CPU已读出或欲写入的数据

MAR/MDR的作用：可使外部操作与内部操作并行(性能)

控制REG—系统模式REG、段REG等

三、CPU的工作流程

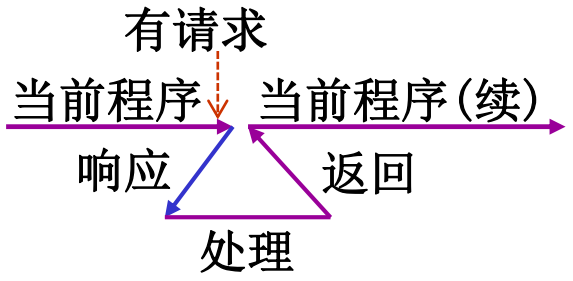
*指令周期：CPU取出并执行一条指令所需的时间

即指令周期＝取指周期(含分析指令)＋执行周期

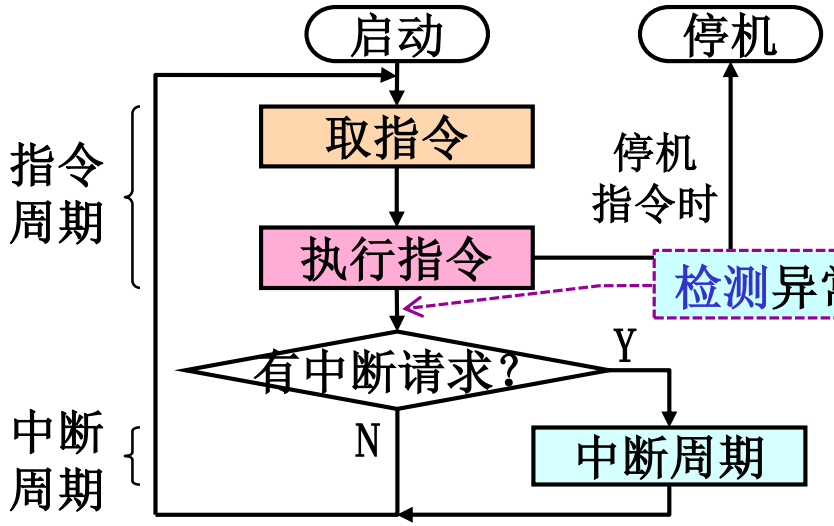
*异常及中断的检测与处理：

检测—硬件实现

处理—响应(硬件)、处理、返回



*CPU的工作流程：循环的指令周期、中断周期(可缺省)



检测异常及中断

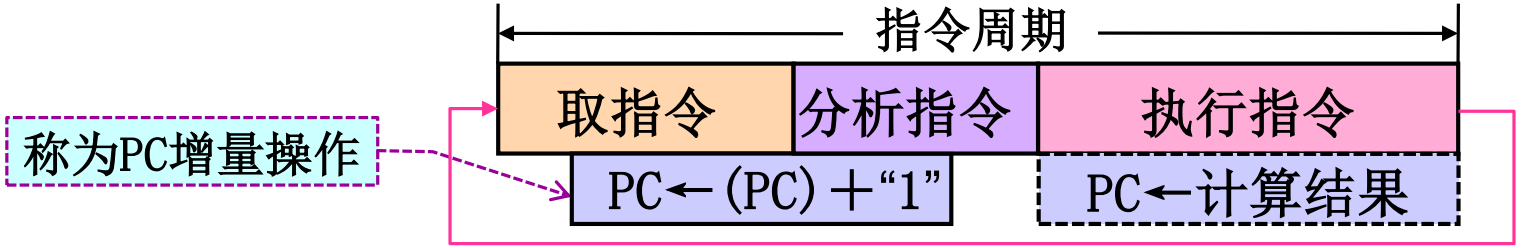
时间控制—主时钟脉冲

分类—单周期CPU、多周期CPU

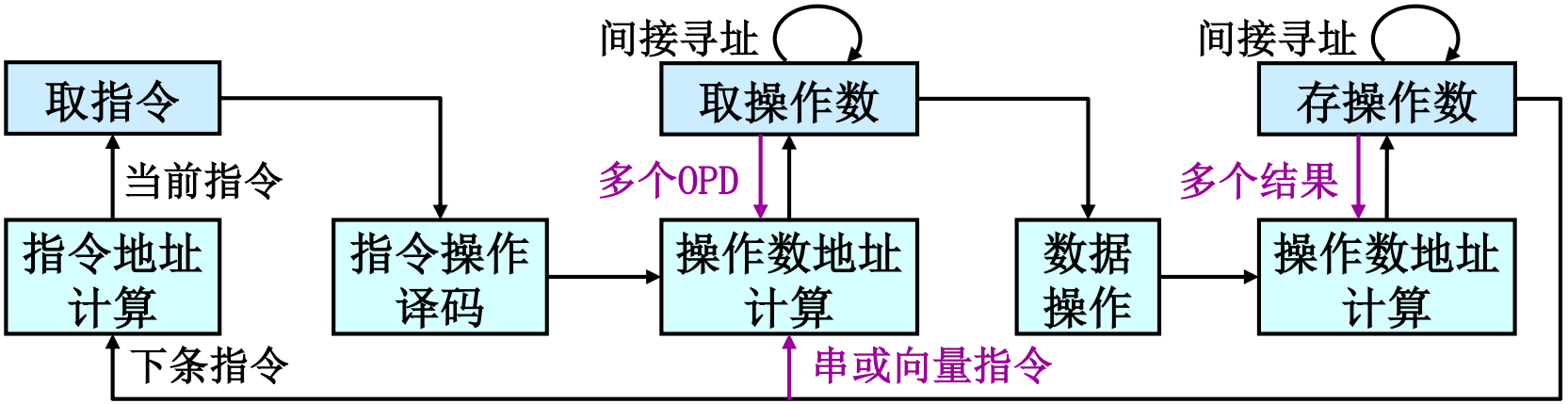
四、指令的执行过程

1、指令执行过程步骤

*程序的执行过程：循环的指令执行过程



*指令的执行过程：由若干操作构成，指令地址计算操作常提前

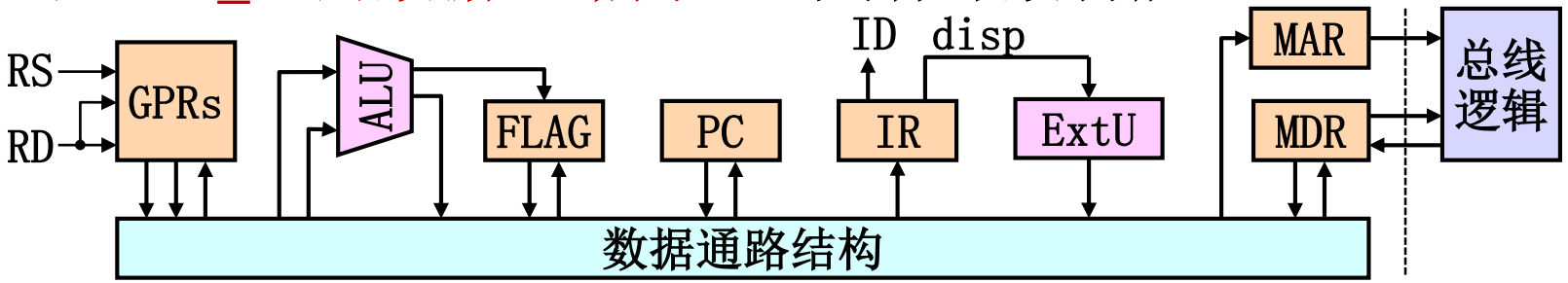


特征—所有指令取指、译码步骤的操作相同，

不同指令执行步骤的操作有所不同

2、指令执行过程示例

*基于Demo_IS的数据通路例：PC具有计数功能



指令功能	指令格式	数据寻址方式	指令寻址方式
赋值(MOV)：RD←Imme	0000 RD 空 Imme	寄存器、立即	隐含EA=(PC)＋2
取数(LD) ：RD←M[(RS)]	0010 RD RS	寄存器、寄存器间接	隐含EA=(PC)＋1
存数(ST) ：M[(RS)]←(RD)	0011 RD RS	寄存器间接、寄存器	隐含EA=(PC)＋1
加法(ADD)：RD←(RD)＋(RS)	0100 RD RS	寄存器、寄存器	隐含EA=(PC)＋1
RD←(RD)＋M[(RS)]	0101 RD RS	寄存器、寄存器间接	隐含EA=(PC)＋1
减法(SUB)：RD←(RD)－(RS)	0110 RD RS	寄存器、寄存器	隐含EA=(PC)＋1
自增(INC)：RD←(RD)＋1	1000 RD 00	寄存器、隐含	隐含EA=(PC)＋1
自减(DEC)：RD←(RD)－1	1001 RD 00	寄存器、隐含	隐含EA=(PC)＋1
分支(JNZ)：	1100 0000 Addr	无	直接EA=ADDR
ZF＝0时PC←Addr			或隐含EA=(PC)＋2
或PC←(PC)＋Disp	1101 Disp	无	相对EA=(PC)＋Disp
	←4位→ 2位 2位		或隐含EA=(PC)＋1

*指令执行过程的表示：实现相应功能的操作序列

*操作的特性：①源/目的数据放在时序逻辑部件(SLM)中

②功能应不可再细分(SLM→CLM→SLM或SLM→SLM)

*程序执行环境例：(PC)＝10H，(R0)＝20H，

(R2)＝30H，主存内容→

10H	00100100
11H	00111000
12H	01100110
13H	11011110
	…
20H	01001000

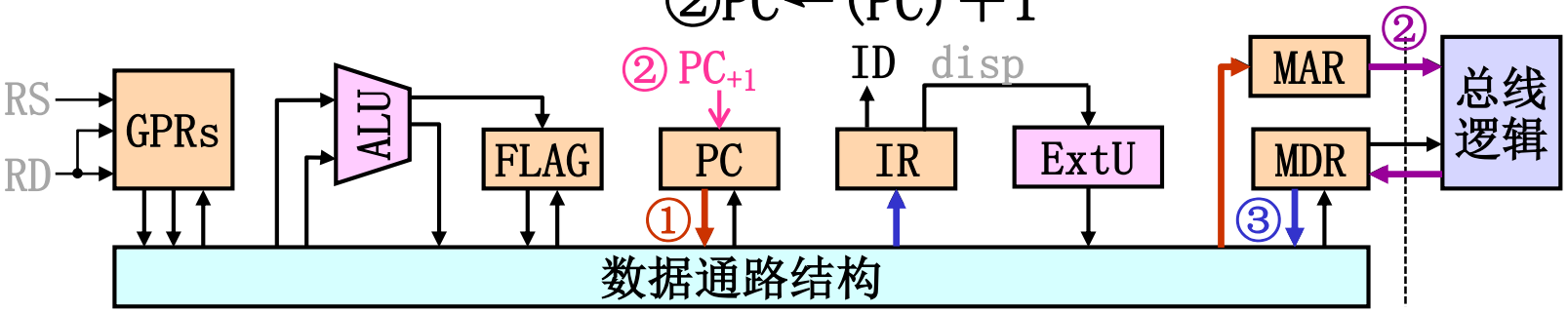
主存

(1)取数指令(LD)的执行过程分析

*取指令阶段：IR←M[(PC)]、PC←(PC)＋“1”

操作序列— ①MAR←(PC)，②MDR←M[(MAR)]，③IR←(MDR)，

②PC←(PC)＋1



操作结果— (PC)＝11H，(IR)＝24H，其余不变

*分析指令阶段：识别当前指令的内容

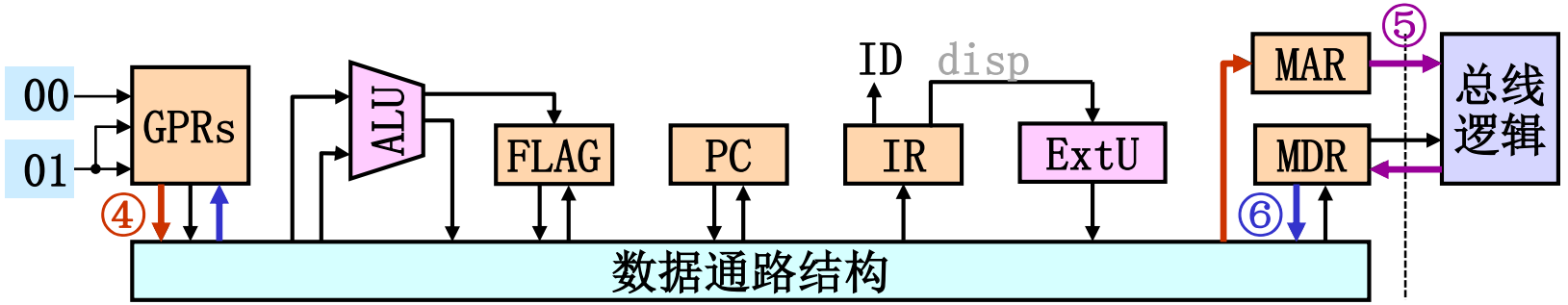
操作序列— 无

(R0)＝20H

分析结果— 指令功能为RD←M[(RS)]，RS＝00、RD＝01

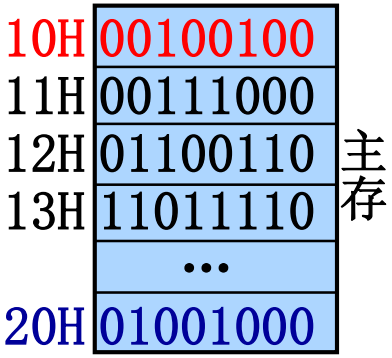
*执行指令阶段：实现当前指令的约定功能

操作序列— ④MAR←(R0)，⑤MDR←M[(MAR)]，⑥R1←(MDR)



操作结果— (R1)＝48H，其余不变

指令地址计算— 无操作(LD为顺序型指令)



(2) 存数指令(ST)的执行过程分析

10H	00100100
11H	00111000
12H	01100110
13H	11011110
	…
20H	01001000

主存

*取指令阶段：IR←M[(PC)]、PC←(PC)＋“1”

操作序列— 同取数指令LD(①~③)

操作结果— (PC)＝12H，(IR)＝38H，其余不变

*分析指令阶段：识别当前指令内容

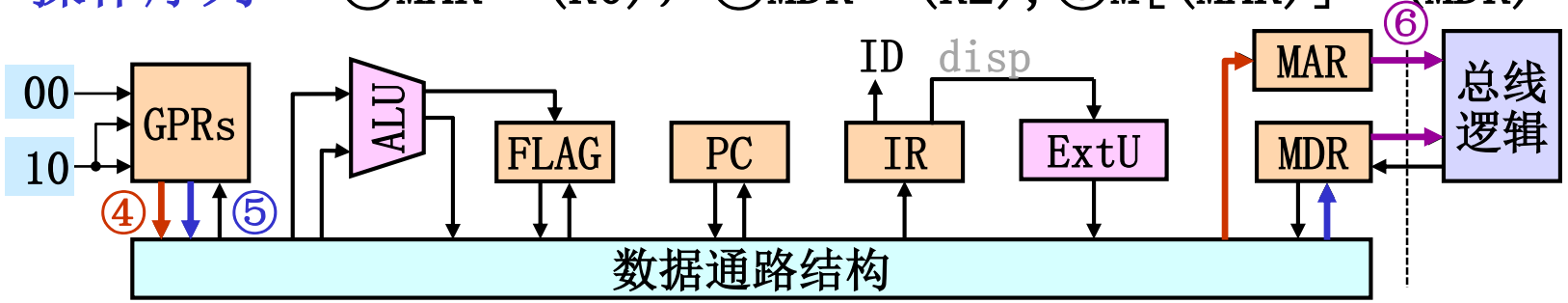
操作序列— 无

(R0)＝20H，(R2)＝30H

分析结果— 指令功能为M[(RS)]←(RD)，RS＝00、RD＝10

*执行指令阶段：实现当前指令的约定功能

操作序列— ④MAR←(R0)，⑤MDR←(R2)，⑥M[(MAR)]←(MDR)



操作结果— M[20H]＝30H，其余不变

指令地址计算— 无操作(ST为顺序型指令)

10 9

(3)减法指令(SUB)的执行过程分析

10H	00100100	
11H	00111000	
12H	01100110	主存
13H	11011110	
	…	
20H	01001000	

*取指令阶段：IR←M[(PC)]、PC←(PC)＋“1”

操作序列— 同取数指令LD(①~③)

操作结果— (PC)＝13H，(IR)＝66H，其余不变

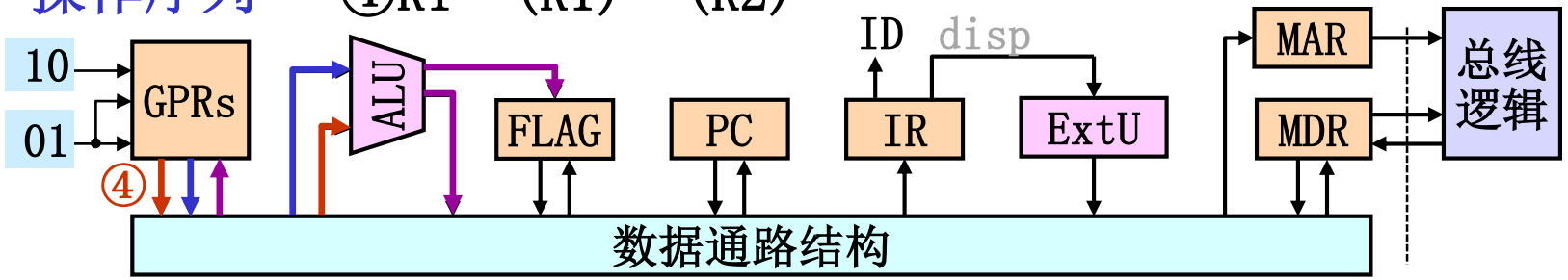
*分析指令阶段：识别当前指令内容

操作序列— 无

分析结果— 指令功能为RD←(RD)－(RS)，RS＝10、RD＝01

*执行指令阶段：实现当前指令的约定功能 (R1)＝48H，(R2)＝30H

操作序列— ④R1←(R1)－(R2)



操作结果— (R1)＝18H，ZF/CF/SF/OF＝0/0/0/0、其余不变

指令地址计算— 无操作(SUB为顺序型指令)

(4) 分支指令(JNZ)的执行过程分析

10H	00100100
11H	00111000
12H	01100110
13H	11011110
	…
20H	01001000

主存

*取指令阶段：IR←M[(PC)]、PC←(PC)＋“1”

操作序列— 同取数指令LD(①~③)

操作结果— (PC)＝14H，(IR)＝DEH，其余不变

思考—若上条指令结果的ZF=1，操作序列及结果?

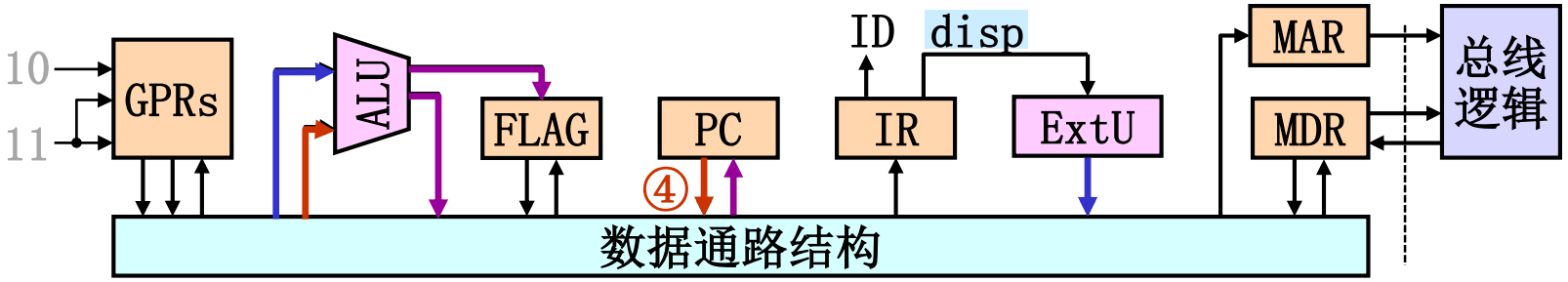
*分析指令阶段：识别当前指令内容

操作序列— 无

分析结果— 指令功能为ZF＝0时PC←(PC)＋disp，disp＝1110 补码表示

*执行指令阶段：实现当前指令的约定功能

操作序列— 无　　操作结果— 无



指令地址计算— 操作序列为 ④PC←(PC)＋(ExtU)，

操作结果为 (PC)＝12H

※指令执行过程的特征：

(1)由取指、分析、执行阶段的操作组成

(2)取指令阶段的操作对所有指令通用

①MAR←(PC)，②MDR←M[(MAR)]、PC←(PC)＋1，③IR←(MDR)

多字长指令处理—通常仅取首字内容，其余作为OPD参数

(须含操作码、寻址方式)（在执行阶段再取）

(3)执行阶段的操作受OP类型、寻址方式、指令字长影响

(4)所有的操作是一个基本操作序列

REG间传送— $R_D \leftarrow R_S$

存储器读— MDR←M[(MAR)]

存储器写— M[(MAR)]←(MDR)

算逻运算— $R_D \leftarrow (R_{S1})$ op (R_{S2})

MOV指令的操作序列：

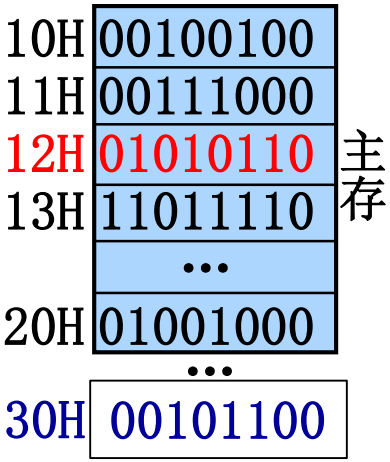
④ MAR←(PC)，

⑤ MDR←M[(MAR)]、
PC←(PC)＋1，

⑥ R3←(MDR)

练习一若M[12H]=56H，操作序列及结果？ (R1)＝48H，(R2)＝30H

① MAR←(PC)，②MDR←M[(MAR)]，③IR←(MDR)，
②PC←(PC)＋1
④ MAR←(R2)，⑤MDR←M[(MAR)]，⑥R1←(R1)+(MDR)



指令功能	指令格式	数据寻址方式	指令寻址方式
赋值(MOV)：RD←Imme	0000 RD 空 Imme	寄存器、立即	隐含EA=(PC)＋2
取数(LD)：RD←M[(RS)]	0010 RD RS	寄存器、寄存器间接	隐含EA=(PC)＋1
存数(ST)：M[(RS)]←(RD)	0011 RD RS	寄存器间接、寄存器	隐含EA=(PC)＋1
加法(ADD)：RD←(RD)＋(RS)	0100 RD RS	寄存器、寄存器	隐含EA=(PC)＋1
RD←(RD)＋M[(RS)]	0101 RD RS	寄存器、寄存器间接	隐含EA=(PC)＋1
减法(SUB)：RD←(RD)－(RS)	0110 RD RS	寄存器、寄存器	隐含EA=(PC)＋1
自增(INC)：RD←(RD)＋1	1000 RD 00	寄存器、隐含	隐含EA=(PC)＋1
自减(DEC)：RD←(RD)－1	1001 RD 00	寄存器、隐含	隐含EA=(PC)＋1
分支(JNZ)：	1100 0000 Addr	无	直接EA=ADDR
ZF＝0时PC←Addr			或隐含EA=(PC)＋2
或PC←(PC)＋Disp	1101 Disp	无	相对EA=(PC)＋Disp
	←4位→ 2位 2位		或隐含EA=(PC)＋1

§6.2 数据通路的组织

一、数据通路(DataPath)的组成

*数据通路：指令执行过程中数据所经过的路径及部件

*数据通路的组成：功能部件、互连结构(数据通路结构)

1、数据通路部件

*部件类型：操作部件(组合逻辑)、状态部件(时序逻辑)

└←实现操作　　└←保存数据

*常见部件设置：受多个因素影响(如指令的执行过程、指令功能)

取指阶段—PC、IR、IMEM(指令存储器)、Adder(加法器)等

分析阶段—ID，但不属于数据通路

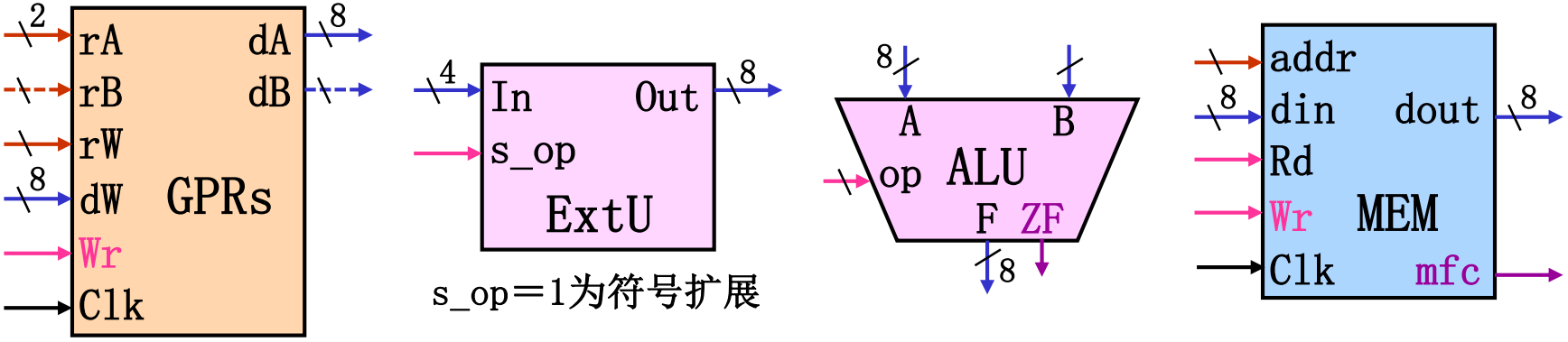
执行阶段—GPRs、ALU、FLAG、DMEM(数据存储器)等

注意：IMEM、DMEM可以为哈佛结构，或冯·诺依曼结构

*Demo_IS的数据通路部件：

部件设置—PC、IR，MAR、MDR、MEM，GPRs、ExtU、ALU、FLAG

接口组织—满足指令系统的需要

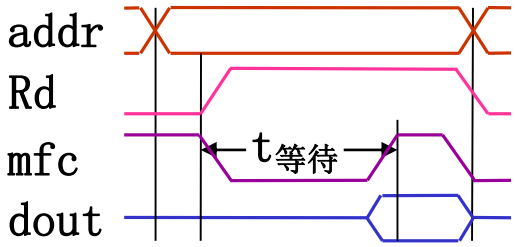


注意：(1)GPRs的写为时序逻辑操作(需Clk控制)，读端口可能只需1个

(2)ALU只需产生ZF标志

(3)同步MEM的读/写由Rd/Wr及Clk控制(异步MEM无需Clk)，

MEM的操作完成状态(操作时延)由mfc给出(开始时=0、完成时=1)



2、数据通路结构

*结构类型：总线结构、点点结构

(1) 总线结构数据通路

*连接方式：多个部件输出端通过同一信号线连接其他部件输入端

*结构分类：单总线结构、双总线结构等

*部件互连方法：

部件输出端—共用信号线时，通过三态门连接到总线

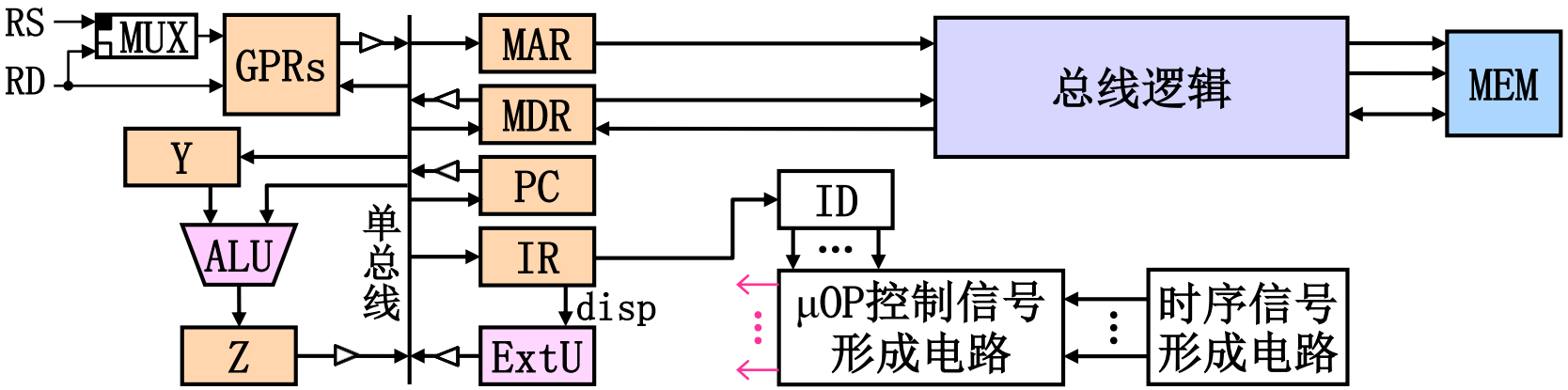
(发送可控制)

部件输入端—有多个时，增设锁存器，或增加总线数量

(分时接收)　　(无冲突接收)

*数据传送特性：只能同时实现1个(或几个)数据传送

*Demo_IS的单总线结构数据通路示例：



REG的连接—输出端增设三态门，输入端(仅1个)直连

思考1：GPRs为什么只有一个读端口？

ALU的连接—设置2个锁存器(所有入端/出端只能有1个直连)

思考2：为什么ALU要设置锁存器(或寄存器)Z？

ALU入端、出端的传送操作存在冲突。

思考3：为什么Y输出端、IR输出端不设置三态门？

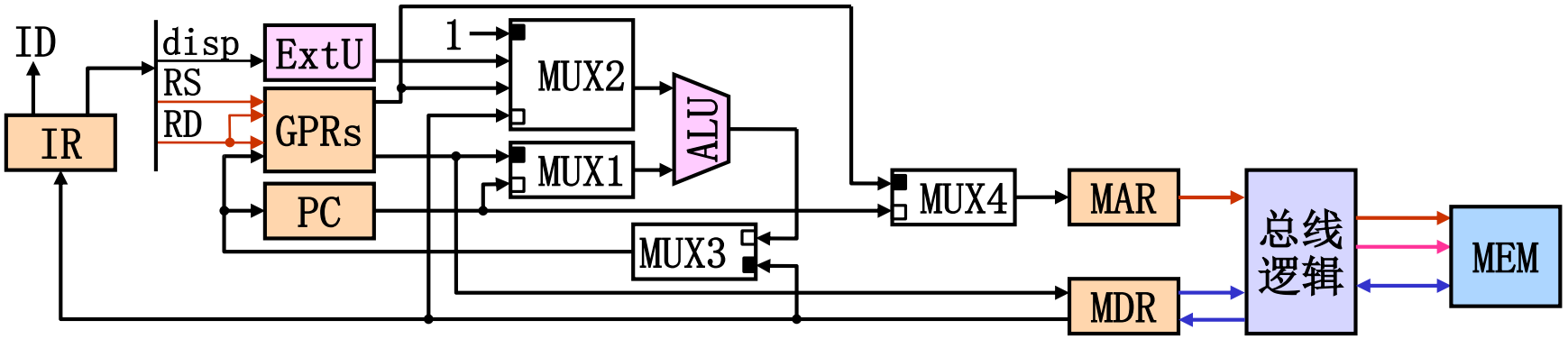
(2) 点点结构数据通路

*连接方式：部件输入端通过不同信号线连接其他部件输出端

*部件互连方法：连接多个输出端时，输入端增设多路选择器

*数据传送特性：可同时实现所有(每个输入端)的数据传送

*Demo_IS的点点结构数据通路示例：



思考1：若GPRs只有一个读端口，通路应如何改变？ ALU增设锁存器

*2种结构比较：总线结构互连简单、分时传送、多周期CPU；

点点结构互连复杂、同时传送、可单周期CPU

3、数据通路的微操作及其控制

*术语：微操作(μOP)—CPU内部的原子(基本部件)操作

微操作命令(μOPCmd)—实现μOP的部件控制信号

节拍—μOP序列的时序信号(1个节拍/μOP)

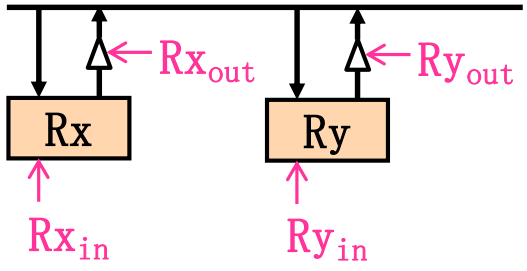
*μOP的特性：源/目的数据在时序逻辑部件中，原子操作

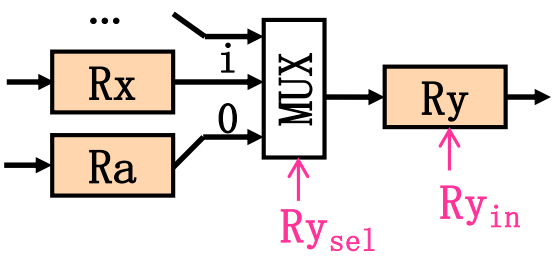
*μOP的类型：基本操作(4种)、特殊操作(PC增量、置位/复位等)

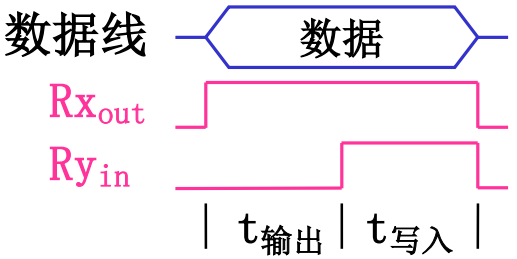
(1)寄存器间传送μOP

*操作功能：Ry←Rx

*部件连接：总线结构、点点结构有所不同





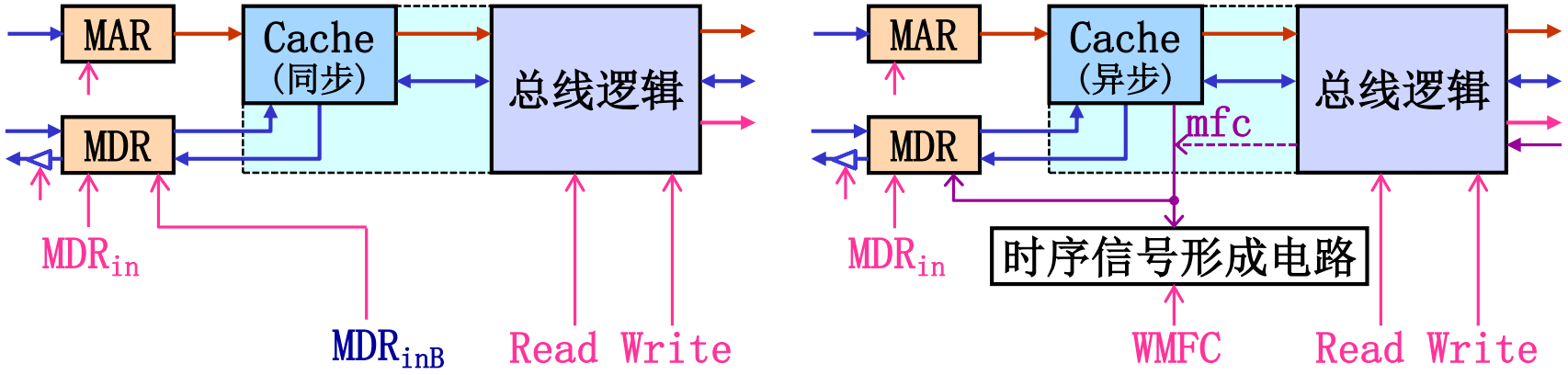


*μOPCmd：Rx_{out}、Ry_{in}；　或Ry_{sel}=i、Ry_{in}；　电位-脉冲制

(2)存储器读、存储器写μOP

*功能：MDR←M[(MAR)]、M[(MAR)]←(MDR)

*部件连接：同步/异步控制方式有所不同



同步控制方式—m个时钟后写MDR(用MDR_{inB}控制），CPU无需等待

异步控制方式—等着写 常数(最坏情况的时延)，性能差 WMFC控制)

指令执行过程的其他μOP无需等待

*μOPCmd：

同步控制方式—读为Read及MDR_{inB}(第m个时钟时)，写为Write

异步控制方式—读为Read、WMFC，写为Write、WMFC

等到何时？见5.3.2 μOP的定时方式

*存储器读μOP的应用示例：总线结构的Demo_IS数据通路

同步控制方式—设GPRs的读/写信号为GR_{out} 、GR_{in}，

通常MDR_{inB}前的μOP为空闲(需与MDR无关)

μOP例	对应的μOPCmd	所需时间
MAR←(R1)	GR_{out}(rA=01)、MAR_{in}	1个Clk
启动M[(MAR)]	Read	1个Clk
…; R3←(R2)	…; GR_{out}(rA=10)、GR_{in}(rW=11)	m-1个Clk
MDR←M[(MAR)]结果	MDR_{inB}	1个Clk

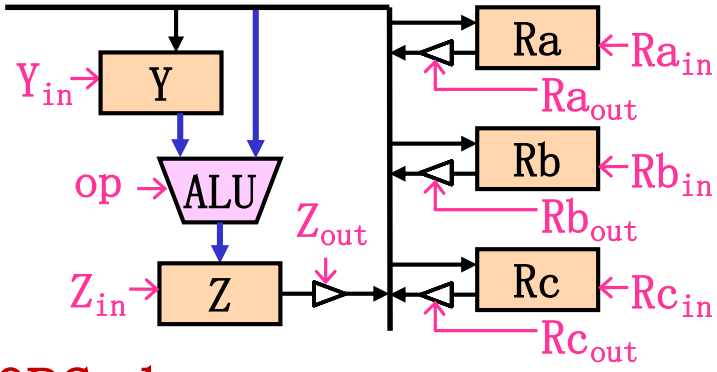
异步控制方式—通常p＜m(如Cache命中时)

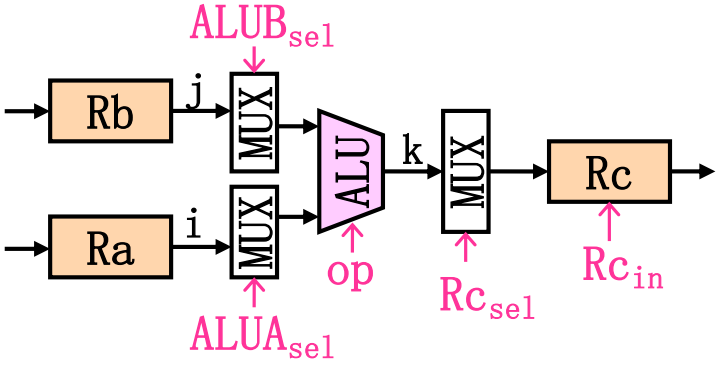
μOP例	对应的μOPCmd	所需时间
MAR←(R1)	GR_{out}(rA=01)、MAR_{in}	1个Clk
MDR←M[(MAR)]	Read、WMFC	p个Clk
…; R3←(MDR)	…; MDR_{out}、GR_{in}(rW=11)	

(3)算术逻辑运算μOP

*功能：$R_D \leftarrow (R_{S1})\ op_n\ (R_{S2})$

*部件连接：总线结构、点点结构有所不同





*μOPCmd：

P180勘误：控制线与操作功能同名了

总线结构— Rb_{out}、$op=op_n$、Z_{in} [即$Z \leftarrow (Y)\ op_n\ (Rb)$]

思考1：$Rc \leftarrow (Ra)\ op_n\ (Rb)$的μOPCmd序列是什么？

答：①Ra_{out}、Y_{in} ②Rb_{out}、$op=op_n$、Z_{in} ③Z_{out}、Rc_{in}

点点结构— $ALUA_{sel}=i$、$ALUB_{sel}=j$、$op=op_n$、$Rc_{sel}=k$、Rc_{in}

思考2：若Ra、Rb、Rc封装在GPRs中，μOPCmd是什么？

答：使$rA=a$、$rB=b$、$rW=c$，Rc_{in}改为GP_{in}

4、指令执行过程的组织

*指令执行过程的表示：μOP序列或μOPCmd序列

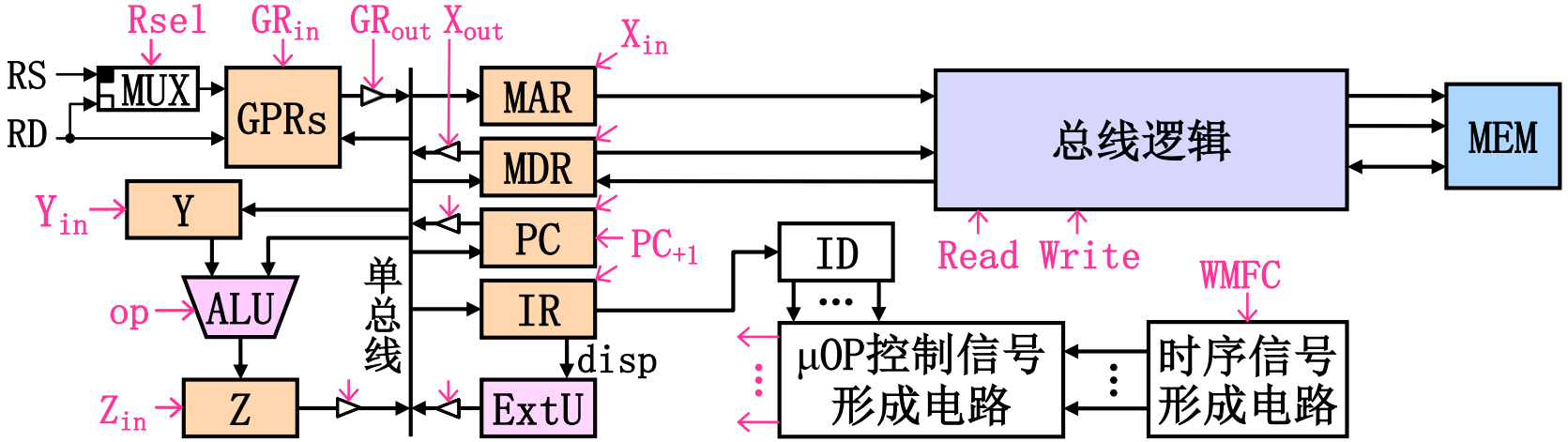
*指令执行过程组织时的要求：

保证正确性—按指令的执行过程安排μOP的顺序

缩短执行时间—可同时执行的μOP安排在同一个步骤中

*单总线结构DP的指令执行过程组织：以Demo_IS为例

假设：op＝00~11表示＋、－、+1、-1，disp采用零扩展



※约定：MUX中的□、■表示控制信号为0、1(最大值)时的入端

取指令的μOP序列及μOPCmd序列—IR←M[(PC)]、PC←(PC)＋1

t1：MAR←(PC)　　t1：PC_{out}、MAR_{in}

t2：MDR←M[(MAR)]，PC←(PC)＋1　　t2：Read、WMFC，PC_{+1}

t3：IR←(MDR)　　t3：MDR_{out}、IR_{in}

指令译码的μOPCmd序列—无，译码常放在t3步或独立的t4步实现

执行指令R2←M[(R1)]的μOP序列及μOPCmd序列— （REG间接寻址）

t4：MAR←(R1)　　t4：GR_{out}、Rsel、MAR_{in}

t5：MDR←M[(MAR)]　　t5：Read、WMFC

t6：R2←(MDR)，End←1　　t6：MDR_{out}、GR_{in}，End

注—End信号表示指令周期是否结束，用于触发中断请求的检测

执行指令M[(R1)]←(R2)的μOP序列及μOPCmd序列—

t4：MAR←(R1)　　t4：GR_{out}、Rsel、MAR_{in}

t5：MDR←(R2)　　t5：GR_{out}、MDR_{in}　;Rsel=0

t6：M[(MAR)]←(MDR)，End←1　　t6：Write、WMFC，End





执行指令R1←(R1)－(R2)的μOP序列及μOPCmd序列—

t4：Y←(R1)　　t4：GR_{out}、Y_{in} ;Rsel=0

t5：Z←(Y)－(R2)　　t5：GR_{out}、Rsel、op＝01、Z_{in}

t6：(R1)←(Z)，End←1　　t6：Z_{out}、GR_{in}，End

注—被加/减数应送到ALU的A端，Y的输出无需控制

执行指令JNZ disp的μOP序列及μOPCmd序列—　(相对寻址)

ZF＝0时：

t4：Y←(PC)　　t4：PC_{out}、Y_{in}

t5：Z←(Y)＋(ExtU)　　t5：$ExtU_{out}$、op＝00、Z_{in}

t6：(PC)←(Z)，End←1　　t6：Z_{out}、PC_{in}，End

ZF＝1时：

t4：End←1　　t4：End

执行指令R1←23H的μOP序列及μOPCmd序列—　（双字长指令）

t4：MAR←(PC)　　t4：PC_{out}、MAR_{in}

t5：MDR←M[(MAR)]，PC←(PC)＋1　　t5：Read、WMFC，PC_{+1}

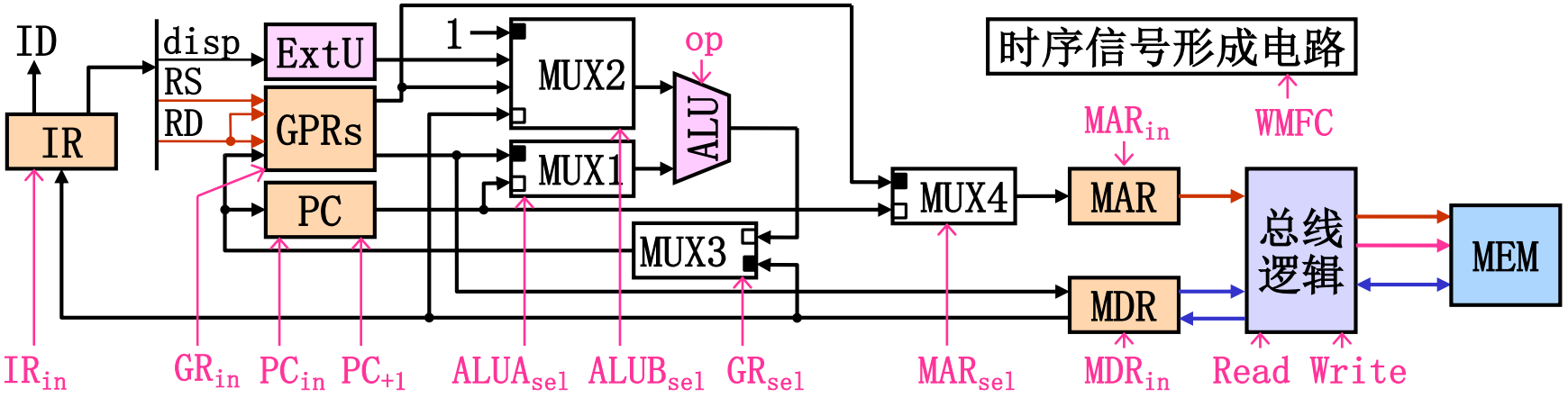
t6：R1←(MDR)，End←1　　t6：MDR_{out}、$R1_{in}$，End

注—取指令阶段通常只取第一个指令字，其余作为OPD参数；

每取指令字的一个字，PC应同步进行增量操作

*点点结构DP的指令执行过程组织：以Demo_IS为例

假设：op＝00~11表示＋、－、+1、-1，disp采用零扩展



取指令的μOP序列及μOPCmd序列—IR←M[(PC)]、PC←(PC)+1

t1：MAR←(PC)　　t1：MAR_{sel}=0、MAR_{in}

t2：MDR←M[(MAR)]，PC←(PC)+1　　t2：Read、WMFC，PC_{+1}

t3：IR←(MDR)　　t3：IR_{in}

执行指令M[(R1)]←(R2)的μOP序列及μOPCmd序列—

t4：MAR←(R1)，MDR←(R2)　　t4：MAR_{sel}=1、MAR_{in}，MDR_{in}

t5：M[(MAR)]←(MDR)，End←1　　t5：Write、WMFC，End

执行指令R1←(R1)－(R2)的μOP序列及μOPCmd序列—

t4：R1←(R1)－(R2)，End←1

t4：$ALUA_{sel}$=1、$ALUB_{sel}$=01、op=01、GR_{sel}=0、GR_{in}，End

*总线结构、点点结构数据通路的性能：后者更优

*影响指令执行过程uOPCmd序列组织的因素：

数据通路结构、指令类型及寻址方式、上条指令状态

作业5-1：P236— 3、5、6(1)和(4)

二、数据通路的设计方法

1、指令周期与数据通路结构

*单周期CPU：CPI＝1，$T_C=\max\{T_{指令i}\}$

数据通路结构— 点点结构

特征— 部件不能复用(需要时重复配置)

└←部件的μOPCmd无法改变(仅1个Clk)

*多周期CPU：$T_C=\max\{T_{\mu OPi}\}$，CPI＝n(随指令而不同)

数据通路结构— 点点结构或总线结构

特征— 部件可以复用

└←部件的μOPCmd可以改变(随Clk不同)

*单周期/多周期CPU的比较：

多周期CPU性能好(实际应用)，单周期CPU简单(用于教学)

2、数据通路的设计方法

⑴指令系统分析

内容—操作类型(含OPD类型)、OPD寻址方式、指令寻址方式

结果—支持的操作类型(含OPD类型)，地址计算方法及参数，
寄存器的位数及个数，存储器的编址单位、地址空间等

⑵功能部件设计

数据操作单元—基于操作类型(含OPD类型)、数据寻址方式

地址计算单元—基于指令寻址方式，与指令周期类型相关(复用)

寄存器组—基于相关参数，读端口数与DP结构相关(单总线1个)

存储器—基于相关参数，通常数据线数＝机器字长(多体交叉MEM)

特殊寄存器—基于实现方法，与指令周期类型等相关(缺省)

⑶部件互连设计

建立各指令的数据路径，与DP结构、指令功能、复用方案有关

总线结构—出端设置三态门，入/出端口设置锁存器(少设1个)

点点结构—出端直连入端，入端设置选择器MUX(＞1个时)

三、单周期数据通路的设计

1、指令系统分析　　--以MIPS为例

指令名	指令功能	功能说明
有符号加add	rd←(rs)＋(rt)	OPD为有符号整数(补码表示)
有符号减sub	rd←(rs)－(rt)	
按位或ori	rt←(rs) | ZExt(imme)	ZExt表示零扩展，OPD为逻辑数
取数lw	rt←M[(rs)＋SExt(imme)]	Imme为16位立即数或偏移量 SExt表示符号扩展
存数sw	M[(rs)＋SExt(imme)]←(rt)	
相等转移beq	if ((rs)=(rt)) PC←(PC)＋SExt(imme)<<2	可为无符号减法， 取指时已实现PC←(PC)＋4
跳转j	PC←$(PC)_{高4位}$ ‖ addr<<2	‖表示拼接，<<2扩大寻址范围

结果：操作类型—32位的有符号加、有符号减、按位或、无符号减

数据寻址计算方法—有符号加、位扩展(零/符号)

指令寻址计算方法—有符号加、符号扩展、<<2、拼接

寄存器—32位×32个，每条指令最多2次读、1 次写

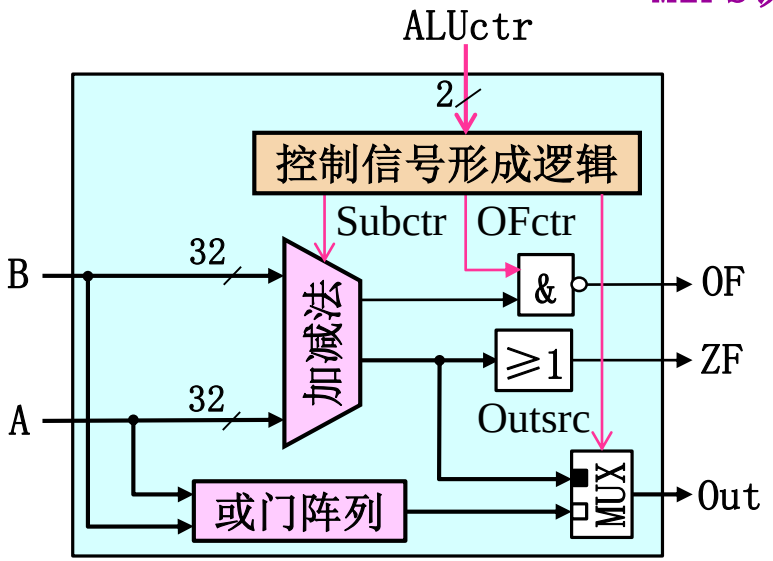
存储器—按字节编址，32位地址空间

2、功能部件设计

***数据操作单元：** 包括ALU(可用于数据寻址)、ExtU

ALU设计—支持4种运算(32位)，产生ZF、OF(有符号加/减时)

MIPS认为无符号加/减主要用于地址计算→┘



假设：ALUctr＝00/01/10/11分别表示有符号加/有符号减/按位或/无符号减

通常，加减法器的op＝0/1表示加/减

(相反时增加“非门”即可)

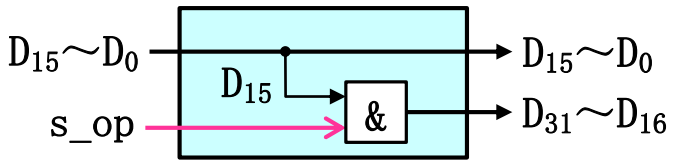
则有，Subctr＝

OFctr＝

Outsrc＝

ExtU设计—支持零扩展、符号扩展，16位→32位

D_{15} \ s_op	0(零扩展)	1(符号扩展)
0	$Q=0\cdots0D_{15}\sim D_0$	$Q=0\cdots0D_{15}\sim D_0$
1	$Q=0\cdots0D_{15}\sim D_0$	$Q=1\cdots1D_{15}\sim D_0$



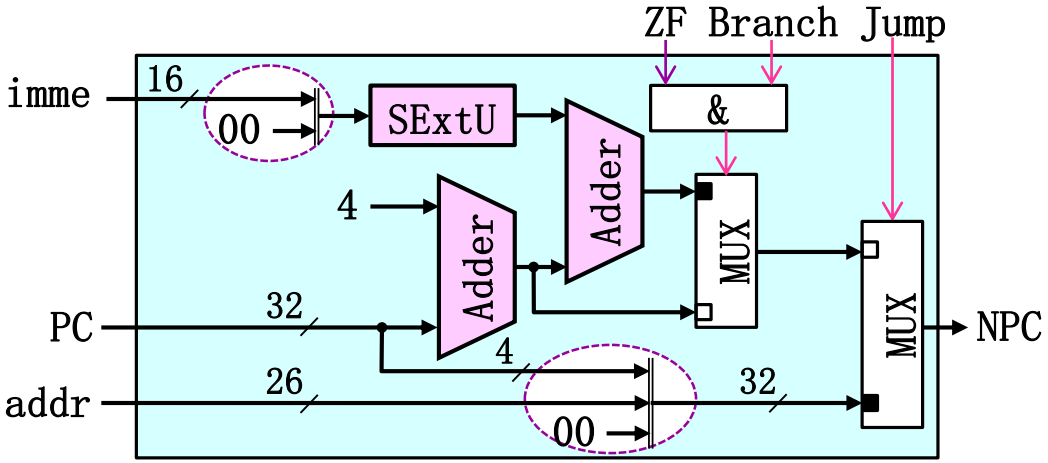
*地址计算单元(ACU)：ACU与ALU间、ACU内部不能复用部件

指令寻址要求—设j、beq指令的操作码译码信号为Jump、Branch

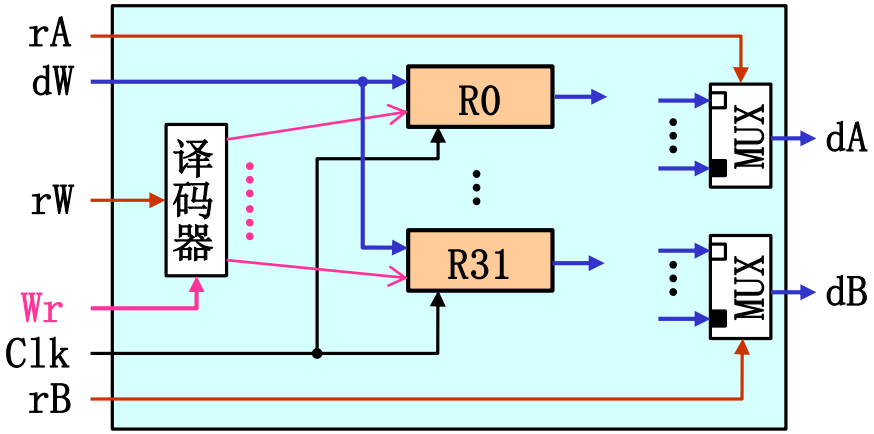
$$\begin{cases} \text{Jump}=1\text{时，NPC}=(\text{PC})_{\text{高4位}} \,\|\, \text{addr}<<2, \\ \text{Branch}\cdot\text{ZF}=1\text{时，NPC}=(\text{PC})+4+\text{SExt(imme)}<<2, \\ \text{否则，NPC}=(\text{PC})+4 \end{cases}$$

ACU设计—支持3种功能(部件不能复用)，

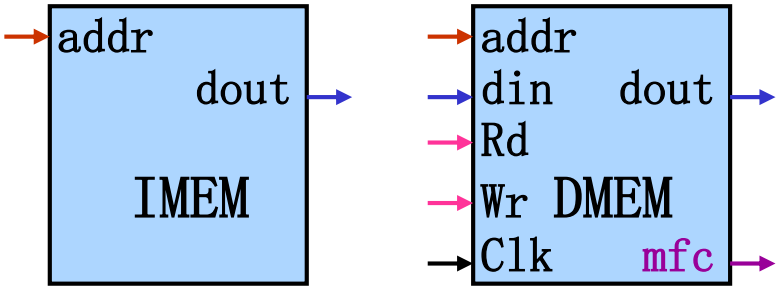
<<2可用拼接实现，Adder、SExtU无控制信号



*寄存器组(GPRs)设计：支持2个读端口、1个写端口



*存储器(MEM)组织：采用哈佛结构，数据宽度为32位



假设：IMEM为异步RAM，
DMEM为同步RAM

要求：多体交叉MEM
(并行访问方式)

*特殊功能寄存器：

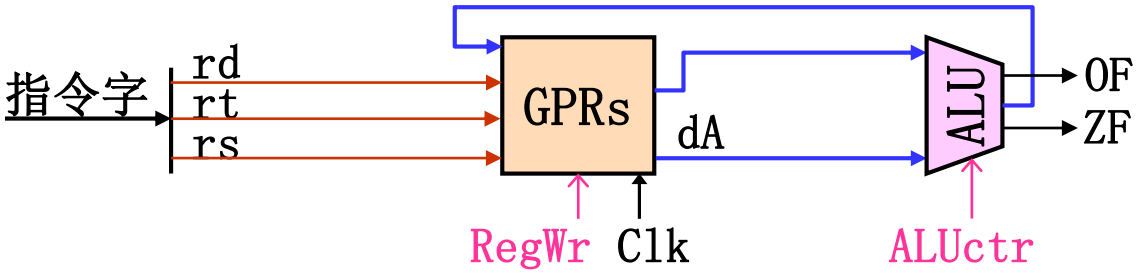
设计—仅设置PC(IR/MAR/MDR缺省，减少时序操作)

3、部件互连设计

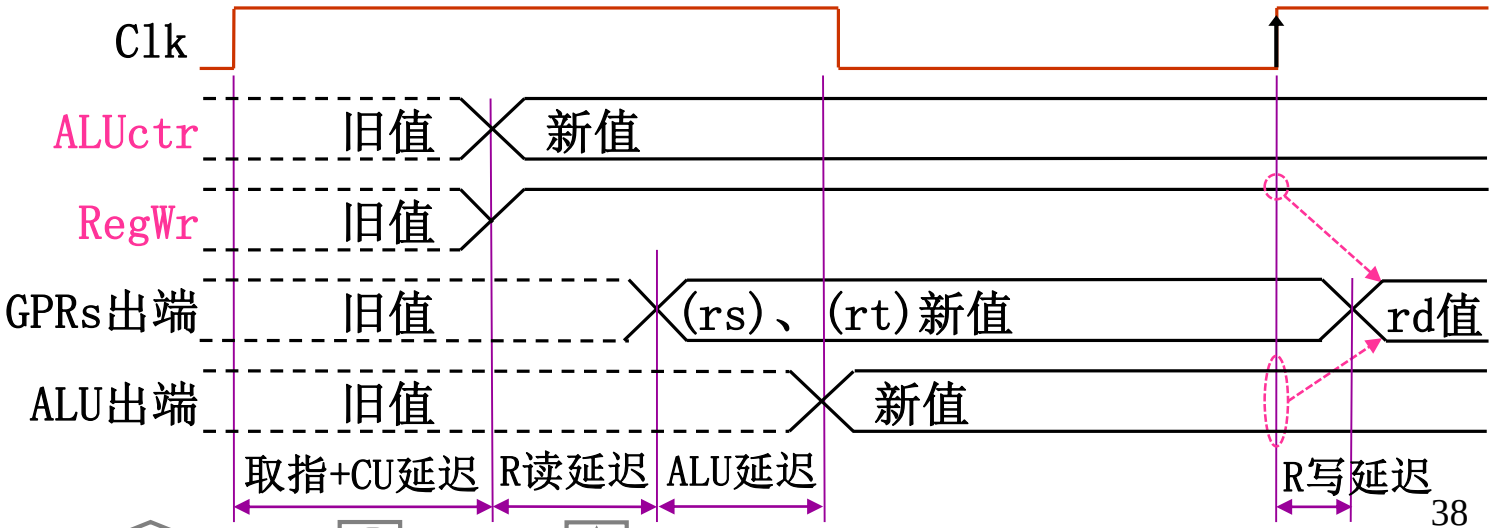
*设计方法：设计每条指令的数据路径，边设计边汇总(或后汇总)

*add/sub指令：rd←(rs)＋(rt)及rd←(rs)－(rt)

数据路径的设计—ALU的A端连接GPRs的dA

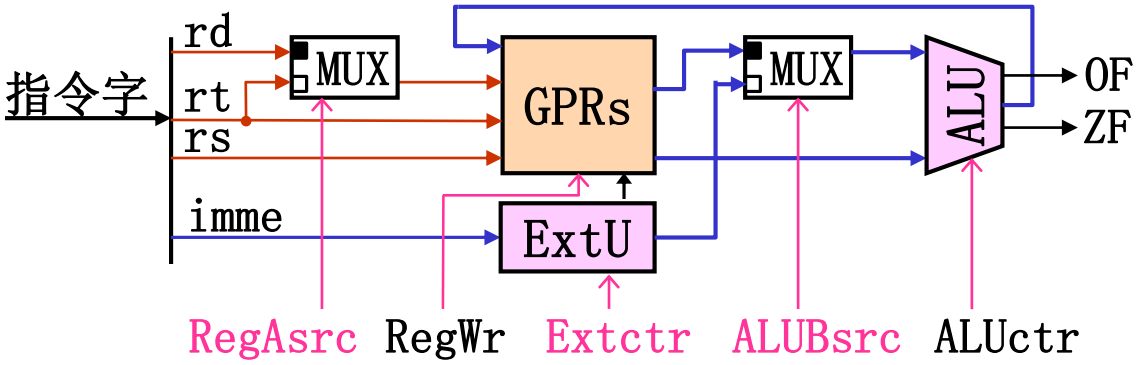


μOP时序的组织—Clk结束时写GPRs(性能好)



*ori指令：rt←(rs)| ZExt(imme)

数据路径的设计—不改变已有路径

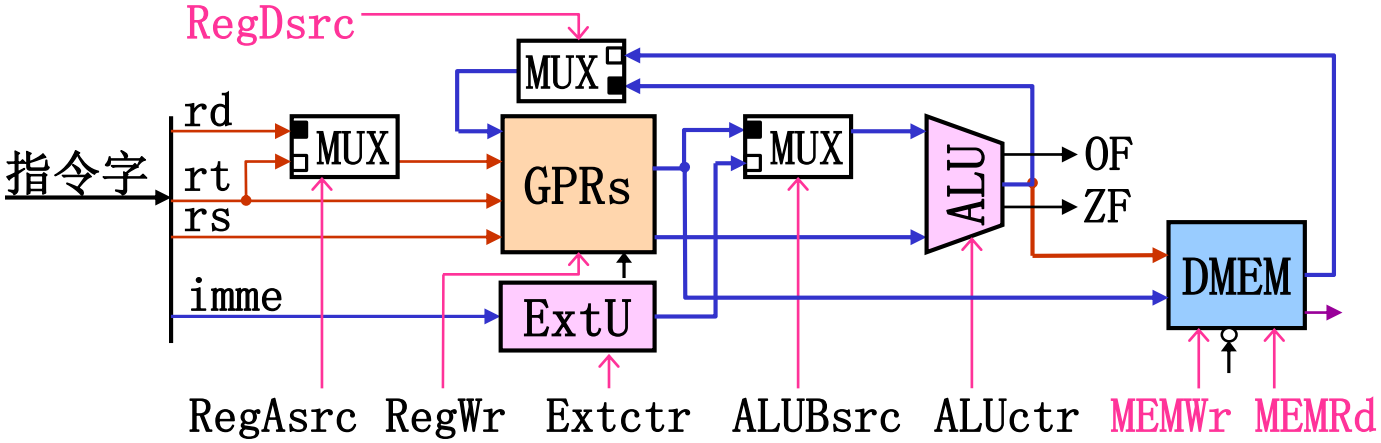


μOP时序的组织—同add/sub指令

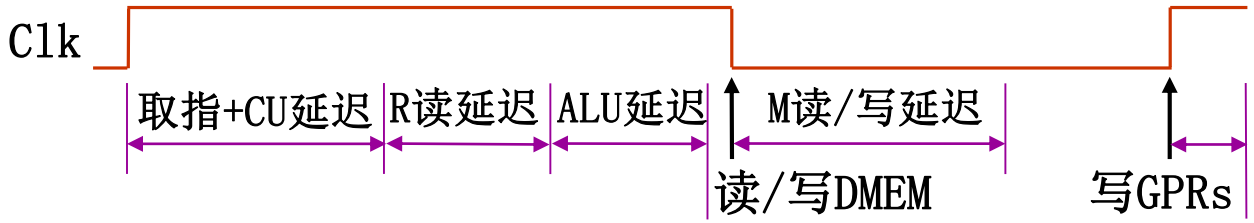
*lw/sw指令：rt←M[(rs)＋SExt(imme)]及

M[(rs)＋SExt(imme)]←(rt)

数据路径的设计—

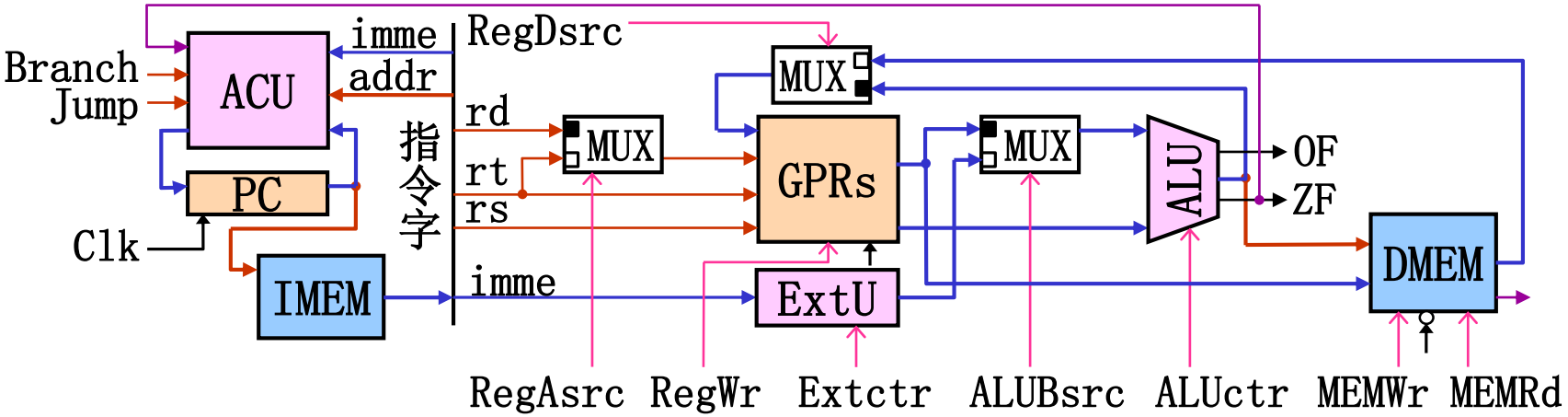


μOP时序的组织—Clk的中间读/写DMEM(还需写GPRs)



*beq指令：PC＝((rs)=(rt))？(PC)＋4＋SExt(imme)<<2 ：(PC)＋4

数据路径的设计—比较用ALU实现、输出ZF



μOP时序的组织—Clk结束时写PC(否则会改变μOPCmd)

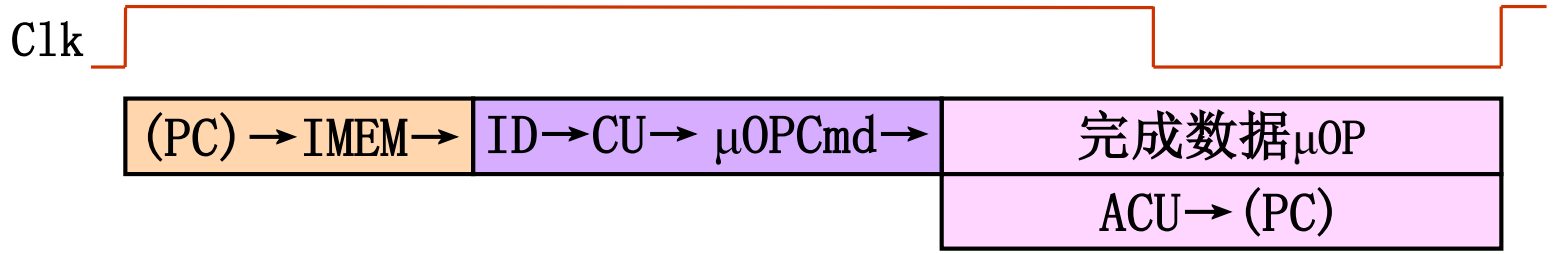
*j指令：PC＝$PC_{高4位}$ ‖ addr<<2

数据路径的设计—同beq指令(无数据操作)

μOP时序的组织—同beq指令

4、指令执行过程的组织

***指令执行过程：**



***指令执行过程的状态转换图：** μOPCmd只有一种状态

	Branch	Jump	Extctr	ALUBsrc	ALUctr	RegAsrc	RegDsrc	RegWr	MemRd	MemWr
add	0	0	×	1	00	1	1	1	0	0
sub	0	0	×	1	01	1	1	1	0	0
ori	0	0	0	0	10	0	1	1	0	0
lw	0	0	1	0	00	0	0	1	1	0
sw	0	0	1	0	00	×	×	0	0	1
beq	1	0	×	1	11	×	×	0	0	0
j	0	1	×	×	×	×	×	0	0	0

注意：路径无关信号中，时序逻辑部件的μOPCmd须为无效

四、多周期数据通路的设计

*多周期CPU的思想：

指令周期＝*n*个阶段，1个阶段/μOP，1个T_C/阶段

时钟周期T_C＝max{$T_{\mu OPi}$}，指所有μOP(如GPRs、ALU、MEM等)

*多周期CPU的实现：　　←优化阶段的性能

指令周期＝ *n*个节拍(阶段)，1个节拍/μOP，1~*p*个T_C /节拍

时钟周期T_C＝max{$T_{基本\mu OPi}$}，指小时延μOP(如GPRs、ALU等)

1、功能部件设计　　--以MIPS为例

*部件复用方案：有多种，如ACU功能复用ALU、ExtU实现

*部件类型：ALU、ExtU、SL2、Splice、GPRs、IMEM、DMEM

(<<2)　(拼接)　　(或冯诺依曼结构)

假设—IMEM为同步RAM，IMEM及DMEM的时延与ALU相当

(同步MEM为主流)　　(暂不考虑MEM读写μOP的定时)

2、部件互连设计

*设计方法：基于单周期数据通路，进行修改

*节拍的划分：（考虑大多数操作的时延）

GPRs、ALU、IMEM、DMEM的操作各占一个节拍

*μOP的组织：（源/结果数据需放在寄存器/存储器中）

IMEM— IR←IM[(PC)]， 需增设IR

读GPRs— A←(rs)、B←(rt)， 需增设A、B

ALU操作— ALUOut←(A)op(B)或imme， 需增设ALUOut

或PC←(PC)＋4或imme<<2

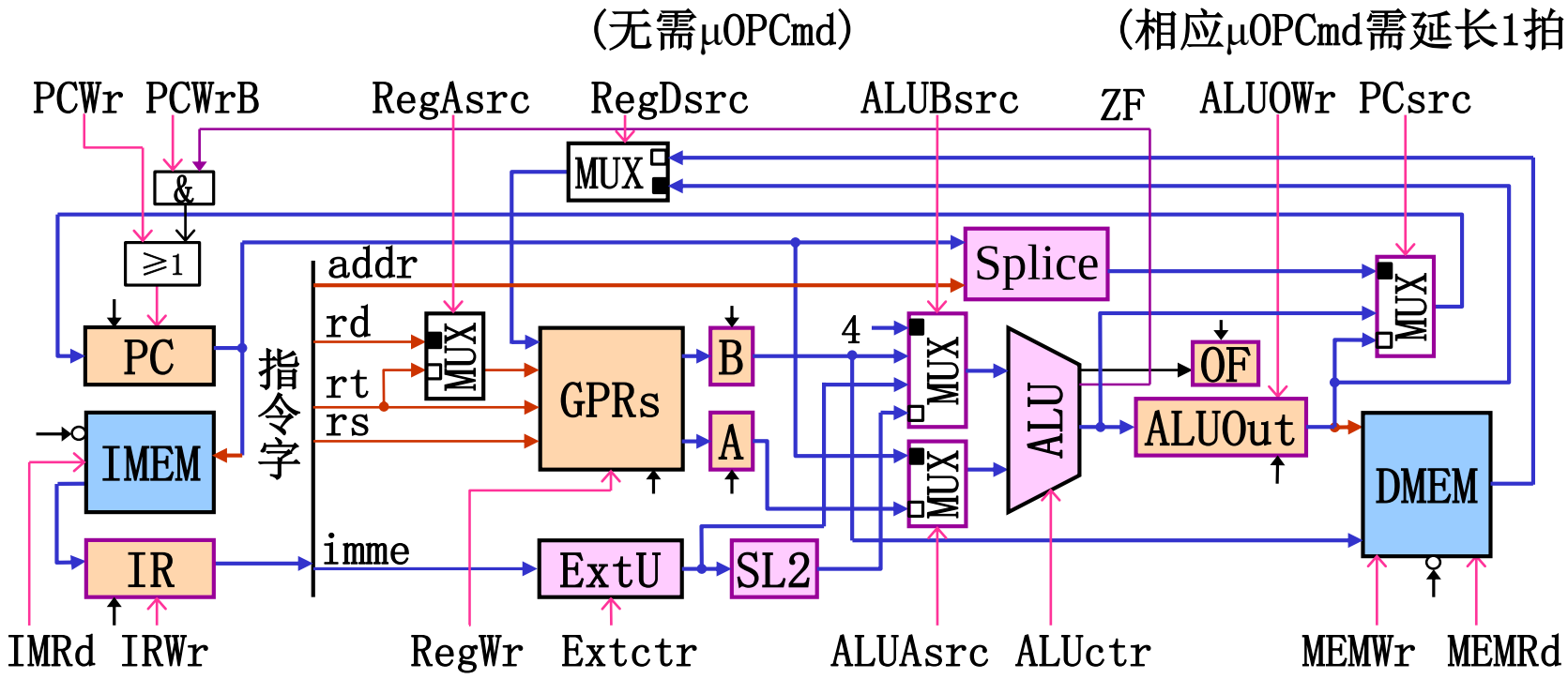
DMEM操作— MDR←DM[(ALUOut)]， 需增设MDR

或DM[(ALUOut)]←(B)

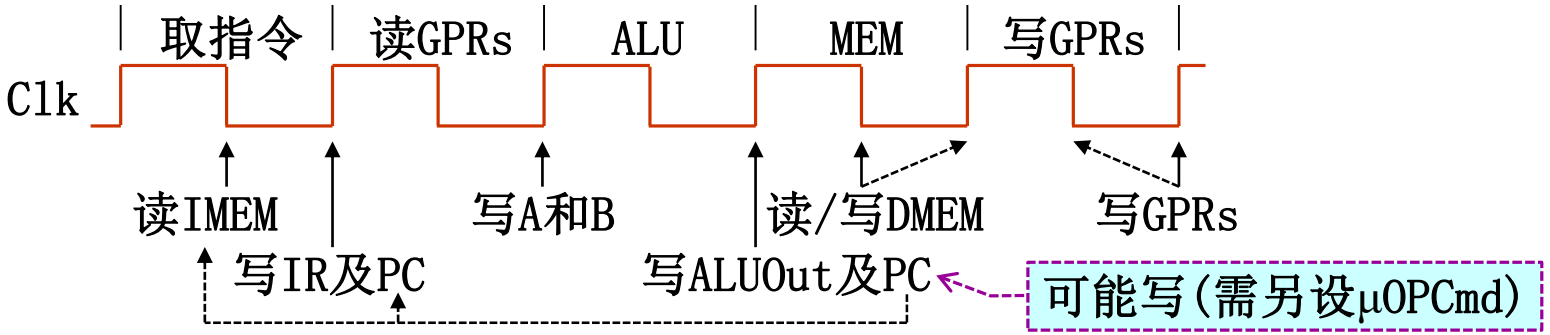
写GPRs— rt/rd←(ALUOut)或(MDR)

其他—PC←(Splice)

*部件互连设计：增设IR、A和B、ALUOut及OF，未设MDR[反例]



*μOP时序的组织：时延最小化(如A及B)、简化实现(如PC)



3、指令执行过程的组织

*指令执行过程：取指令、分析指令、执行指令

*取指令阶段的μOPCmd：

t1：IMRd、IWMFC、IRWr，　　；本例IWMFC可缺省

ALUAsrc＝1、ALUBsrc＝3、ALUctr＝0、PCsrc＝1、PCWr

*执行指令阶段的μOPCmd序列：

add/sub指令—

t2：无　；μOP为A←(rs)、B←(rt)

t3：ALUAsrc＝0、ALUBsrc＝2、ALUctr＝0/1、ALUOWr

t4：RegAsrc＝1、RegDsrc＝1、RegWr，End

ori指令—

t2：无　；μOP为为A←(rs)

t3：ALUAsrc＝0、ALUBsrc＝1、ALUctr＝2、ALUOWr

t4：RegAsrc＝0、RegDsrc＝1、RegWr，End

lw指令—　　（由于未设置MDR，DMEM都操作需保持）

t2：无　；μOP为为A←(rs)

t3：Extctr、ALUAsrc＝0、ALUBsrc＝1、ALUctr＝0、ALUOWr

t4：MemRd、WMFC　　；本例WMFC可缺省

t5：MemRd、RegAsrc＝0、RegDsrc＝0、RegWr，End

sw指令—

t2：无　；μOP为为A←(rs)、B←(rt)

t3：Extctr、ALUAsrc＝0、ALUBsrc＝1、ALUctr＝0、ALUOWr

t4：MemWr、WMFC，End　　；本例WMFC可缺省

beq指令—

t2：Extctr、ALUAsrc＝1、ALUBsrc＝0、ALUctr＝0、ALUOWr

；μOP还包括A←(rs)、B←(rt)

t3：ALUAsrc＝0、ALUBsrc＝2、ALUctr＝3、PCsrc＝0、PCWrB，End

j指令—

t2：无　；等待指令译码结果

t3：PCsrc＝2、PCWr，End

*分析指令阶段的操作安排：放在t2步

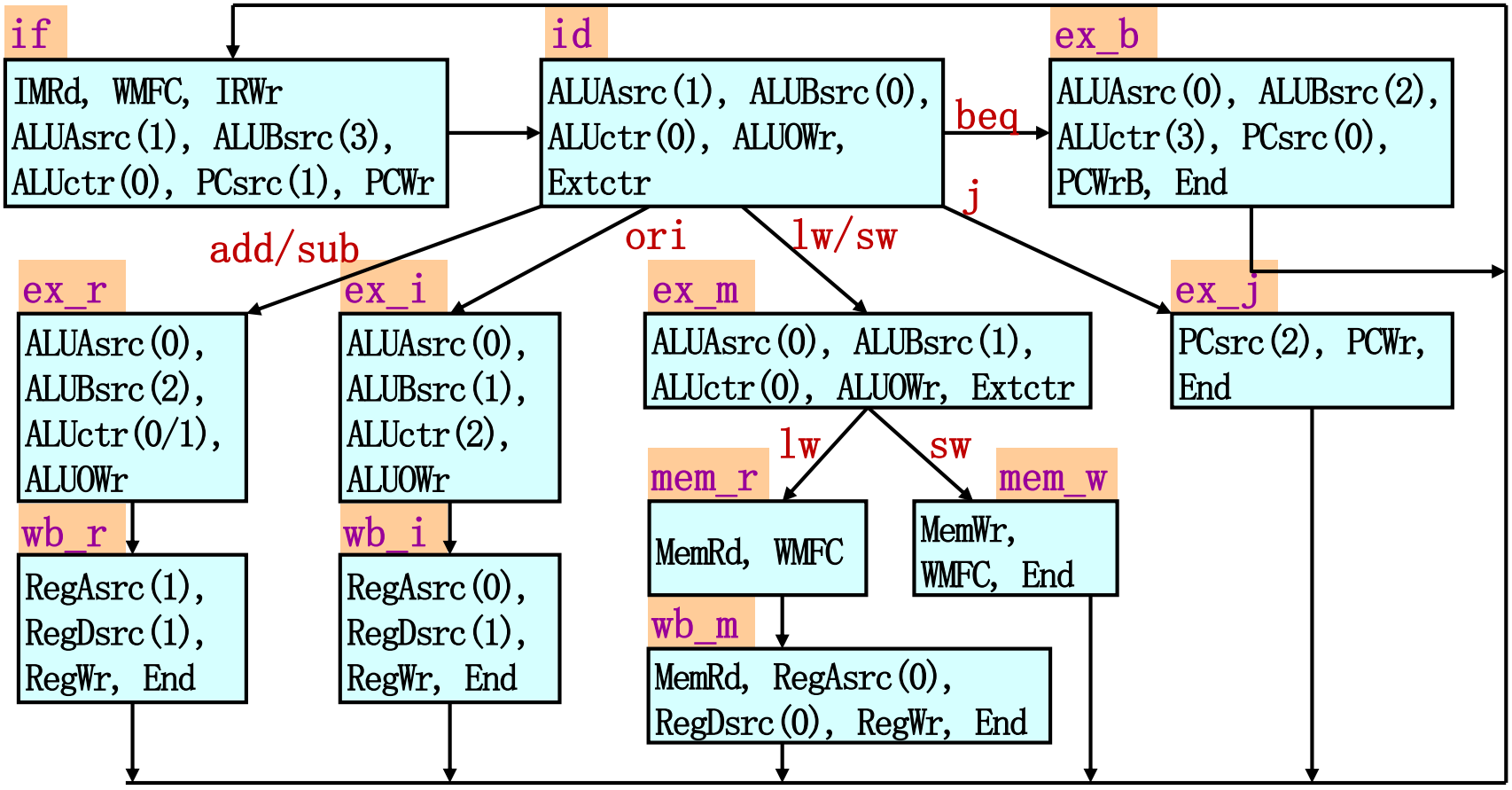
t2步的功能为— 对所有指令通用

指令译码、A←(rs)、B←(rt)、 ALUOut←(PC)＋imme<<2

t2步μOPCmd为—

Extctr、ALUAsrc＝1、ALUBsrc＝0、ALUctr＝0、ALUOWr

*指令执行过程的状态转换图：不同指令的状态数不同



※状态转换图是CPU的功能需求，是控制器的设计目标！

作业5-2：P237— 10、11、12

实验四　CPU数据通路实验

◆ 实验目的

（1）掌握CPU数据通路的逻辑组成。

（2）了解指令功能的实现过程及其控制方法。

◆ 实验内容

（1）设计一个单总线结构（Demo_IS）的CPU数据通路，部件包括4种功能的8位ALU、4×8位的寄存器文件、256×8位的RAM、8位计数器各一个。

（2）给出相关部件控制信号，分别实现取数、加法、条件转移指令的功能。

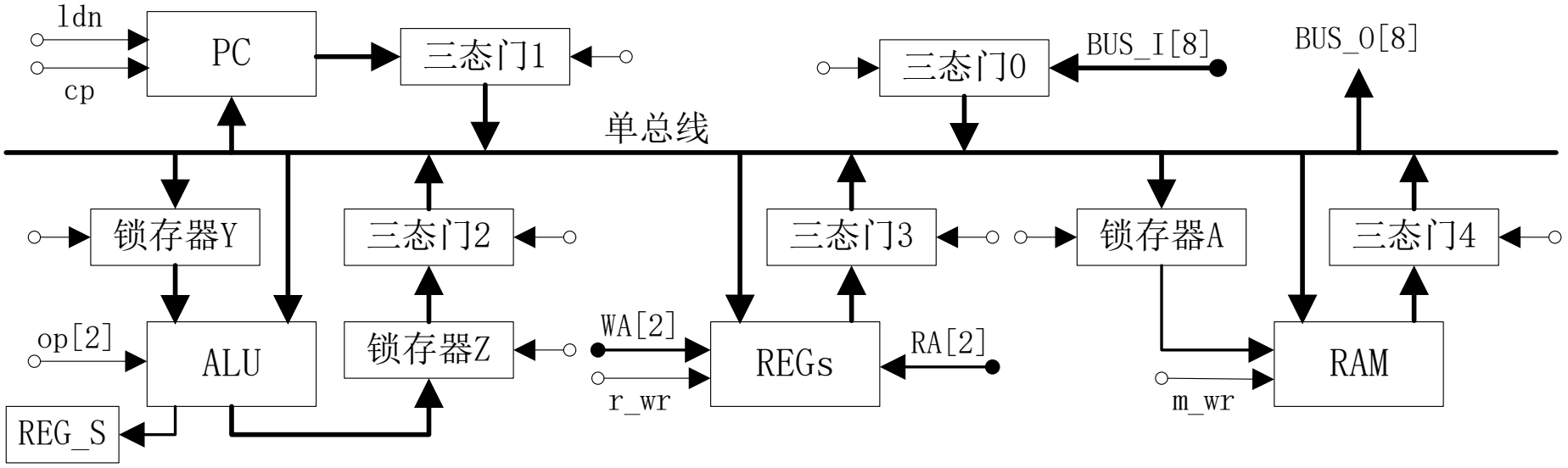
◆ 实验原理及方案

1、CPU数据通路的设计

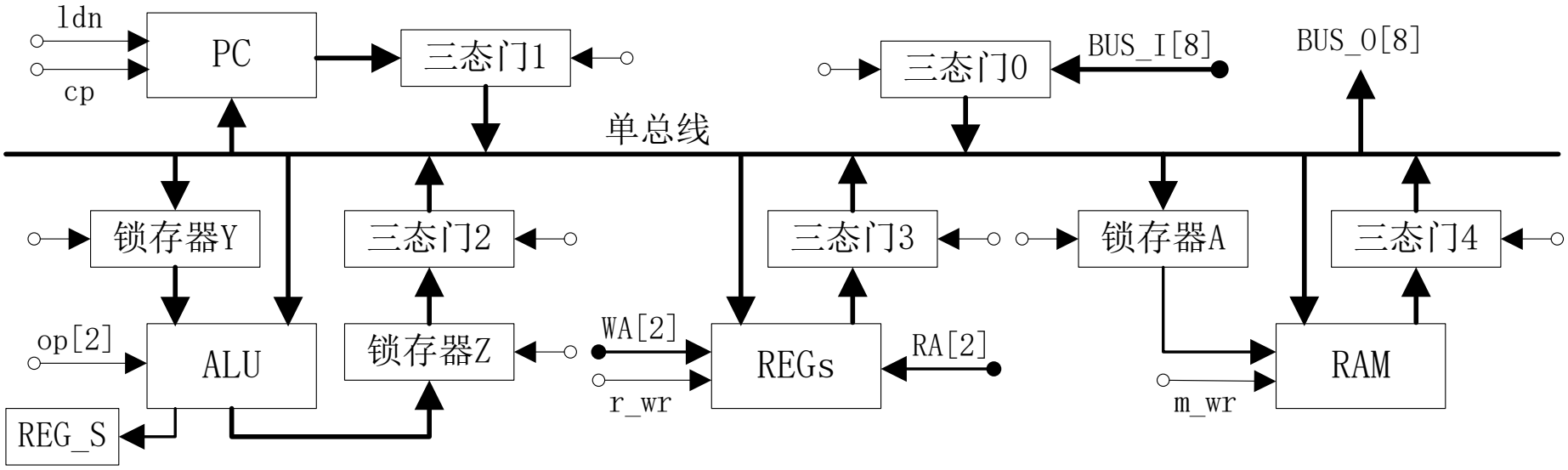
- CPU数据通路：指CPU内部传递数据的物理通道，以及通道上的部件。

- 单总线结构的数据通路：指所有部件通过同一个总线传递数据，即所有部件的数据入端（简称入端）、数据出端（简称出端）均连接在同一个总线上。为保证数据传输的正确性，单总线通路要求同时只能有一个部件输出数据、可以有多个部件接收数据，多个数据传递操作必须串行实现。
- 本实验要求的是设计单总线结构数据通路，数据宽度为8位，包括ALU、寄存器文件、RAM、计数器4个功能部件。

- 数据通路可以采用如图下所示的方案，其中，三态门（记为TSL）是依据总线操作特性（同时只有一个部件能发送数据）而设置的；锁存器是为解决部件的多个端口在单总线上数据接收冲突而设置的；**REG_S为状态寄存器**，存放关系运算所需的标志位（如ZF）；其它部件与总线的数据输入、数据输出接口记为**BUS_I、BUS_O**，输入、输出分开便于观察实验结果。



- ALU的操作控制信号线(2根)为op[2]，REGs的写地址信号线(2根)、读地址信号线(2根)分别为WA[2]、RA[2]，REGs、RAM的写操作控制信号线分别为r_wr、m_wr，
- 程序计数器的数据引脚有入端D、出端Q，控制引脚有置数ldn（低电平有效)、时钟CP、同步/异步清零CLRN（低电平有效)等。计数器的真值表如下：CLRN=0时，Q=0（清零）；CLRN=1、LDN=0、CP上升沿时，Q=D（置数）；CLRN=1、LDN=1、CP上升沿时，Q=Q+1（计数）；其余情况Q保持不变。



2、指令功能的实现

- 要求实现取数、加法、条件转移指令的功能：

  取数： LD -- RD←[(RS)]；

  加法： ADD -- RD←(RD)+(RS)及RD←(RD)+M[(RS)]；

  条件转移：JZ -- ZF=1时，PC←(PC)+disp；

  ZF=0时，PC←(PC)+1。

（1）指令格式参数的产生

- 取指令阶段的任务：实现IR←[(PC)]、PC←(PC)+1，执行指令阶段的任务是实现指令约定的操作功能。理论上，指令格式的参数OPER(操作类型)、RS、RD、disp均来自IR，ZF来自状态寄存器；
- 本实验中，为简化实验难度，指令格式的参数通过引脚输入产生，如参数OPER隐含约定（每次仿真测试一种指令、ADD指令用引脚op[2]表示），RS、RD用引脚RS[2]、RD[2]产生，disp用引脚BUS_I[8]产生，ZF状态用引脚ZF产生。（指令操作的参数与部件的引脚不一定都是一一对应的，如RA[2]可能来自RS[2]或RD[2]，WA[2]只能来自RD[2]）。

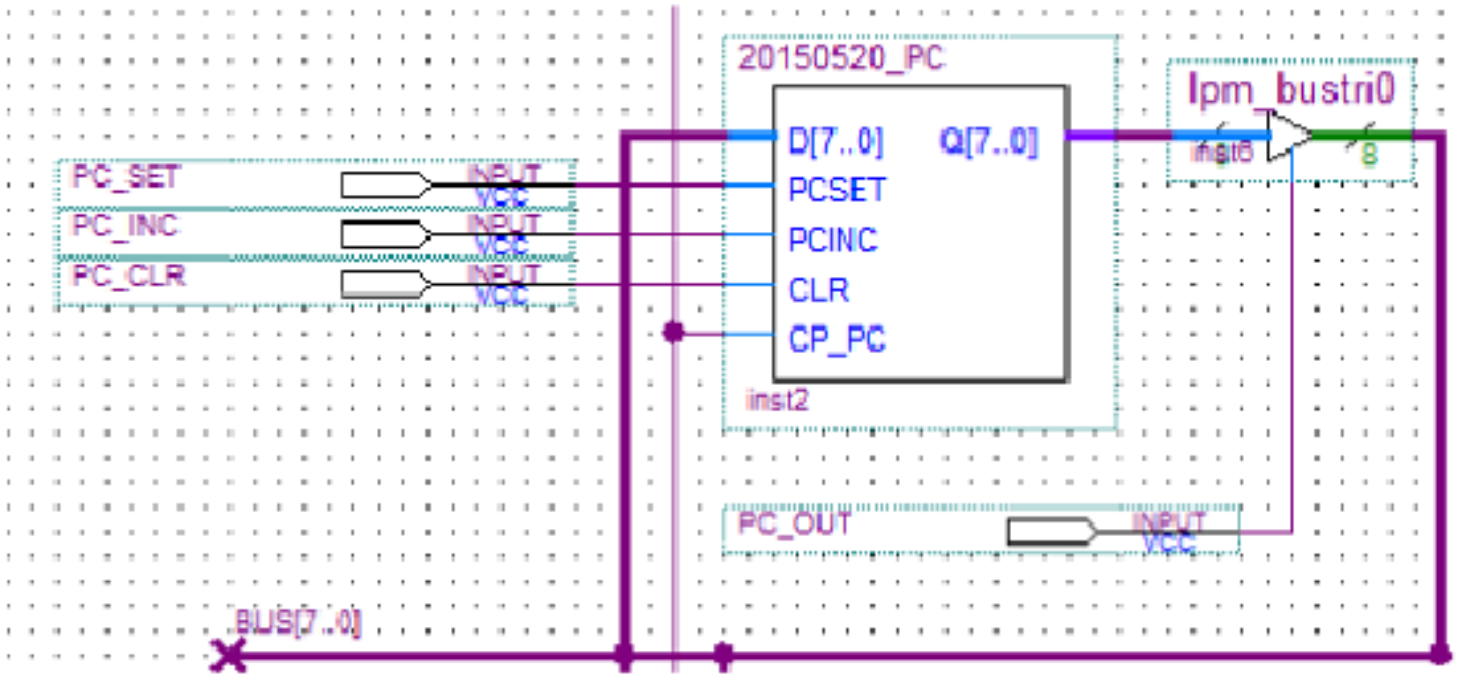
（2）指令功能的实现

- 每个指令的功能可表示为一个微操作步序列，每个微操作步由一个或几个可同时实现的微操作组成。只要按序实现某微操作步序列中各微操作的功能，即可实现该指令的功能。
- 通过控制相关部件（给出控制信号）来实现微操作的功能，如锁存器Z→R1的数据传送操作：控制信号如下：三态门2控制信号=1(有效)、WA[2]=01、r_wr=1(有效)。
- 指令功能→微操作步序列的分解，需要基于数据通路，使用CPU的4种基本操作实现。分别为寄存器间数据传送、ALU运算、MEM读、MEM写，基于单总线通路，第1种基本操作可用一个微操作实现，后3种均需用二个微操作实现。
- 根据取数LD、加法ADD、条件转移JZ指令的功能约定，可以分别写出指令执行时的微操作步序列。

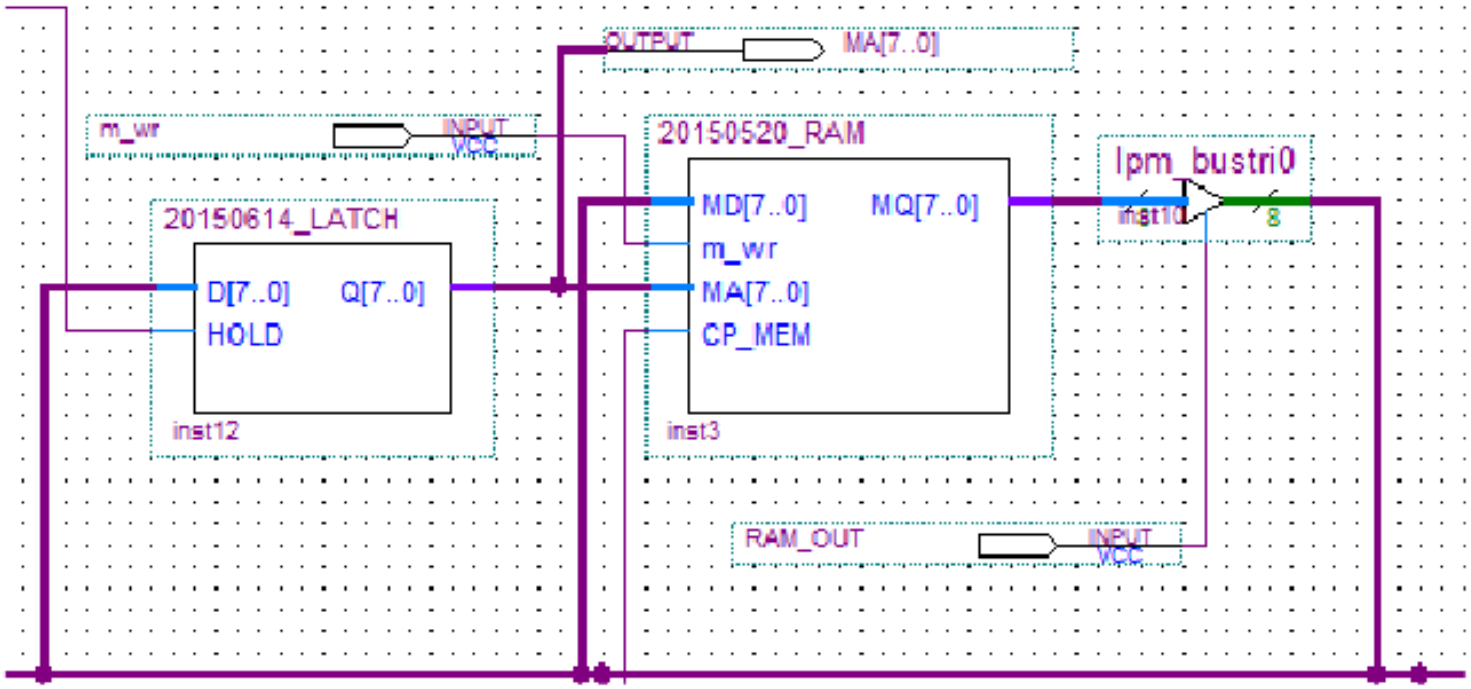
◆ 实验步骤与实现方案

（1）数据通路设计

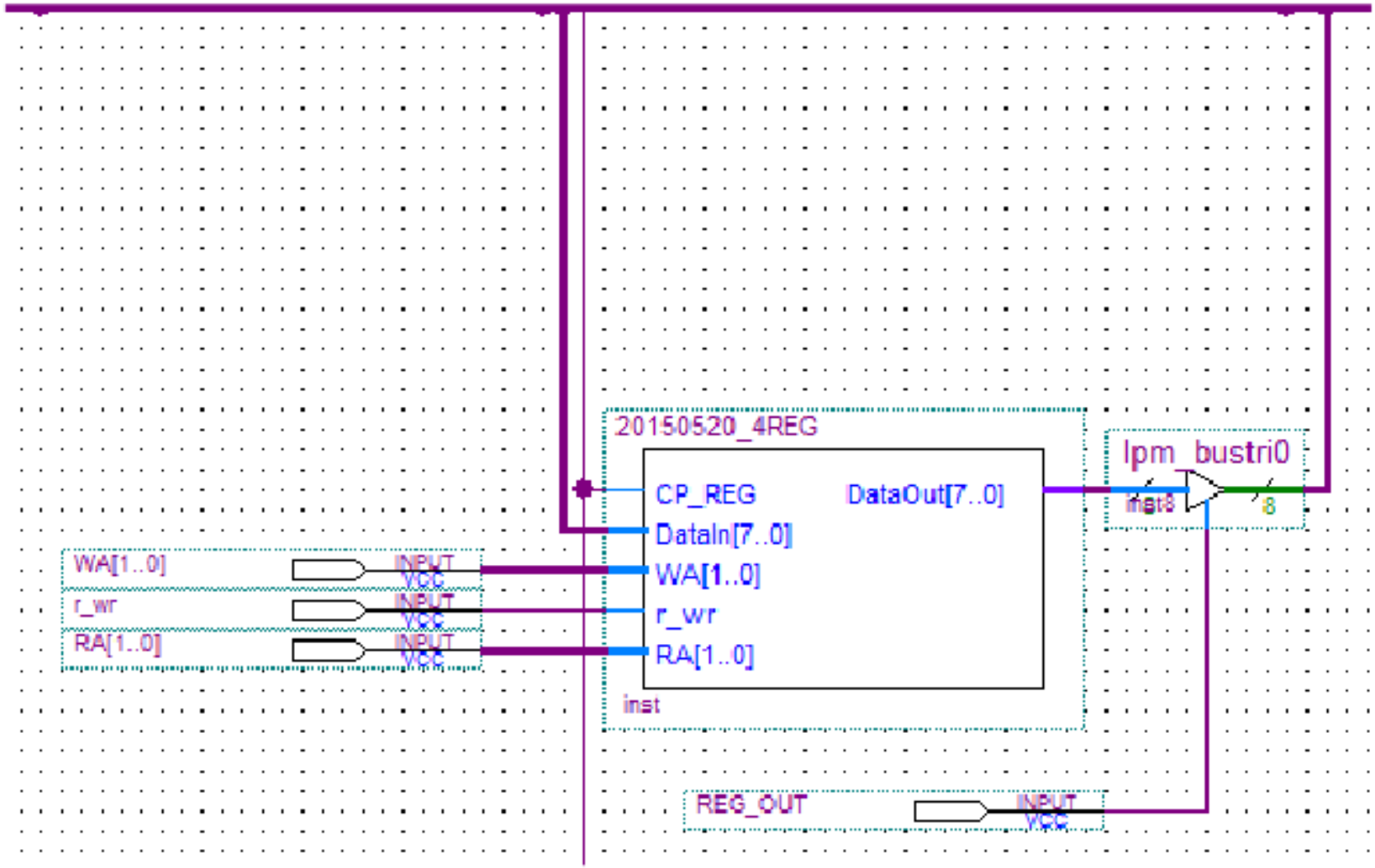
程序计数器 PC：在输出端增加三态门，实现 PC 与总线间的分时段隔离，避免信号之间的冲突。



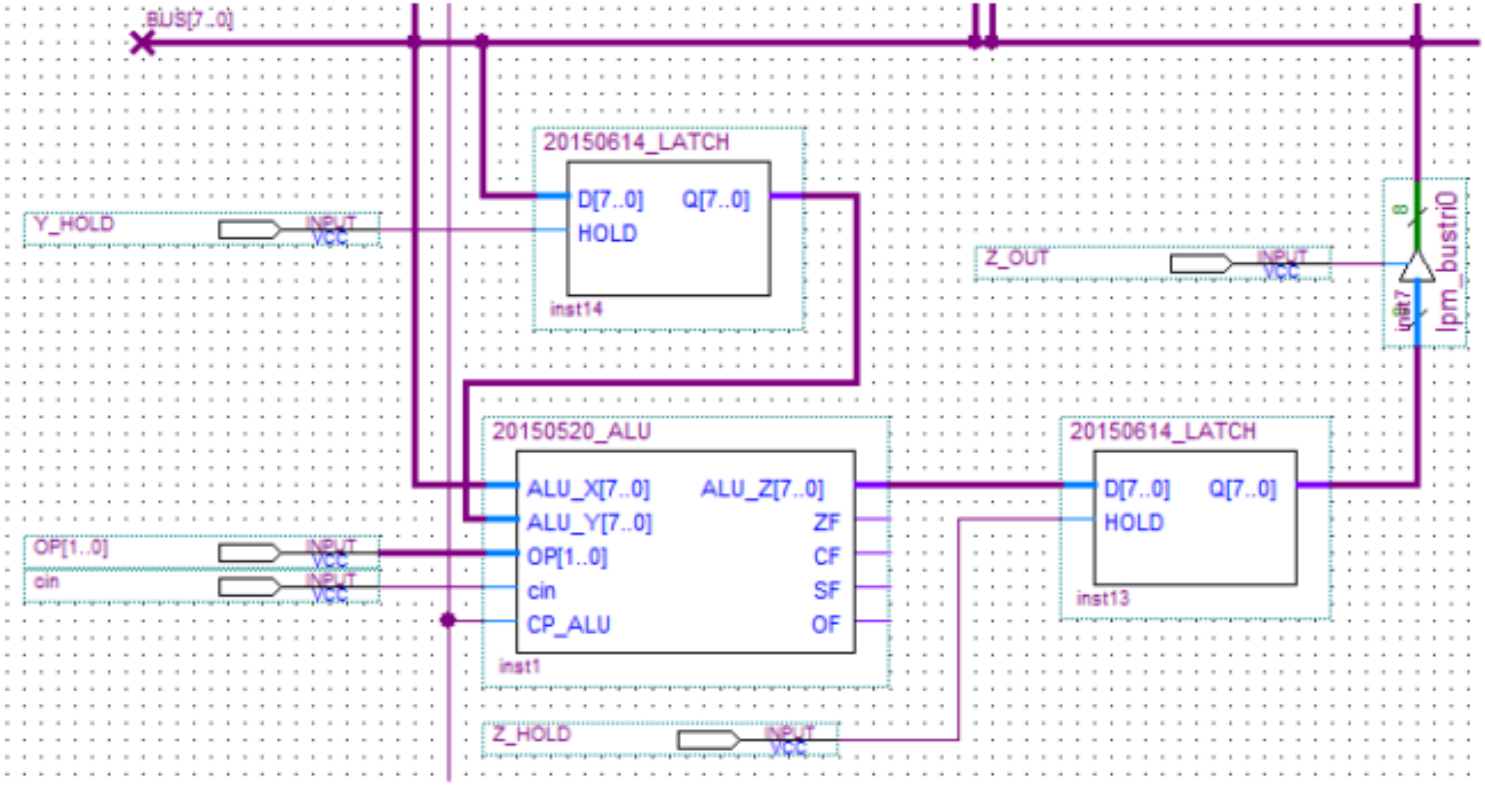
DRAM：除在输出端增加三态门以外，还在地址输入端增设地址锁存器，其目的是为保证数据存取时地址信号的稳定。



寄存器组：在输出端增加三态门。



ALU：由于 ALU 是组合逻辑器件，不能保存信息，因此要求输入信号保持稳定，为此，在数据输入端增加锁存器。ALU 的运算结果也需要暂存，因此输入端也设置了锁存器，通过三态门控制与总线的连接通断。



另外，外部数据输入接口也通过三态门挂载到总线上。

（2）各部件的控制信号：

控制信号名称	功能描述
程序计数器控制信号	
PC_SET	PC 置数
PC_INC	PC 自增
算术逻辑单元控制信号	
Y_HOLD	操作数锁存器 锁存
Z_HOLD	运算结果锁存器 锁存
OP[2]	ALU 操作类型
寄存器组控制信号	
WA[2]	寄存器写地址
RA[2]	寄存器读地址
r_wr	寄存器写使能
DRAM 控制信号	
RAM_ADDR_HOLD	地址锁存器 锁存
三态门控制信号（以下信号同时至多一个有效）	
PC_OUT	程序计数器输出
Z_OUT	运算结果输出
REG_OUT	寄存器输出
RAM_OUT	RAM 输出
BUS_IN	外部输入允许

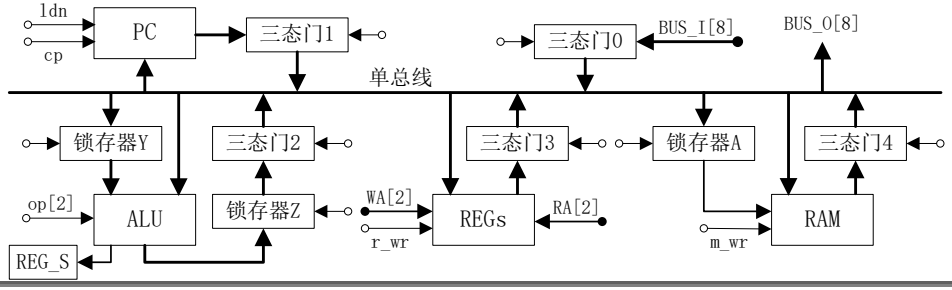
（3）指令微操作设计

取数 LD ： RD←[(RS)]

1）RS→RA，REG_OUT，RAM_ADDR_HOLD，REG_OUT（；关闭REG三态门）
（；置RS读地址）
2）RD→WA，r_wr，RAM_OUT
（；置RD写地址）
3）RAM_OUT，RAM_ADDR_HOLD，结束

加法 ADD：RD←(RD)+[(RS)]

1）RD→RA，REG_OUT，Y_HOLD，REG_OUT
2）RS→RA，REG_OUT，RAM_ADDR_HOLD，REG_OUT
3）RAM_OUT，ADD→OP，Z_HOLD，RAM_OUT，RAM_ADDR_HOLD，Y_HOLD
4）RD→WA，r_wr，Z_OUT
5）Z_OUT，结束



加法 ADD：RD←(RD)+ (RS)

1）RD→RA，REG_OUT，Y_HOLD，REG_OUT
2）RS→RA，REG_OUT，ADD→OP，Z_HOLD，REG_OUT，Y_HOLD
3）RD→WA，r_wr，Z_OUT
4）Z_OUT，结束

条件转移 JZ（ZF=1）：PC←(PC)+disp

1）PC_OUT，Y_HOLD，PC_OUT

2）BUS_IN，ADD→OP，Z_HOLD，BUS_IN，Y_HOLD

3）Z_OUT，PC_SET，Z_OUT，结束

条件转移 JZ（ZF=0）：PC←(PC)+1

1）PC_INS

（4）指令操作数设定

寄存器地址	寄存器数值
2H	**50**
3H	**100**

DRAM 地址	DRAM 数值
100	**180**

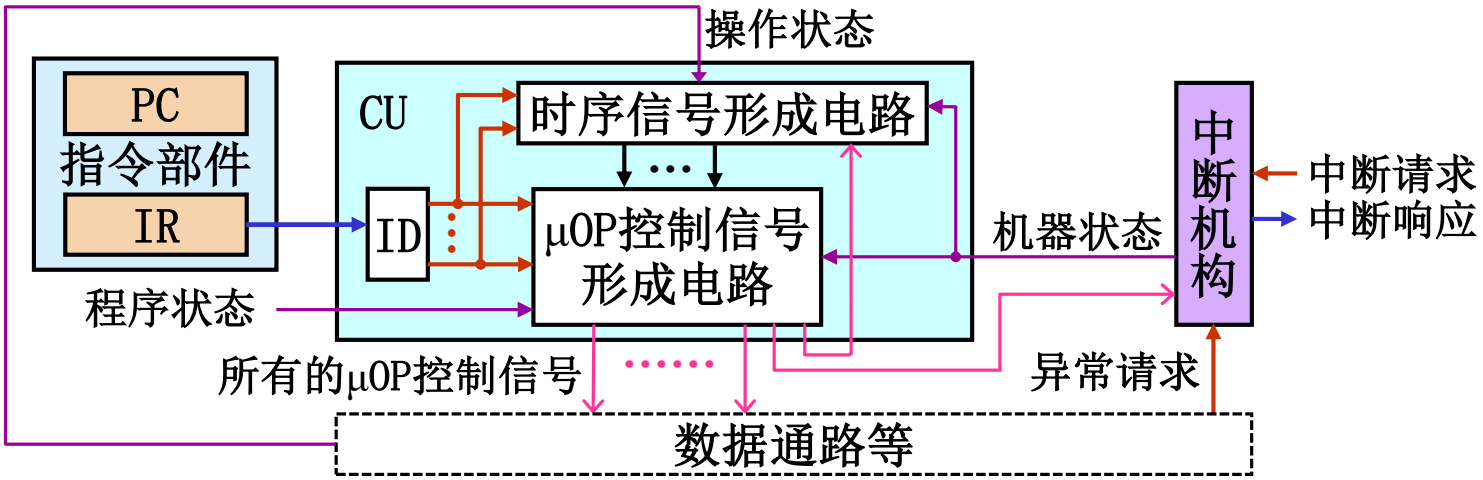
（5）实验结果显示（仿真波形图）

§5.3 控制器的组成

一、控制器的基本结构

*功能：指令控制、操作控制、时间控制、异常及中断处理

*基本组成：由指令部件、控制单元CU、中断机构组成



*工作原理：循环地、有序地产生工作流程所需的μOP控制信号

指令控制　时间控制　操作控制　指令周期或中断周期

*控制器类型：

	硬布线控制器	微程序控制器
μOP控制信号的描述	有限状态机	微程序
μOP控制信号的产生	组合逻辑电路 (与时序信号相关)	微指令执行部件 (与时序信号无关)
时序信号的周期	1个(指令周期＋中断周期) 如：MIPS指令周期＝5T_C	1个微指令周期
说明：(1)某指令μOP序列的步数＝该指令周期的状态数 (2)1个微程序等价于1条指令的μOP序列 (3)1条微指令周期相当于1个μOP的时延 (4)时序信号的单位为时钟周期		

二、时序信号的形成

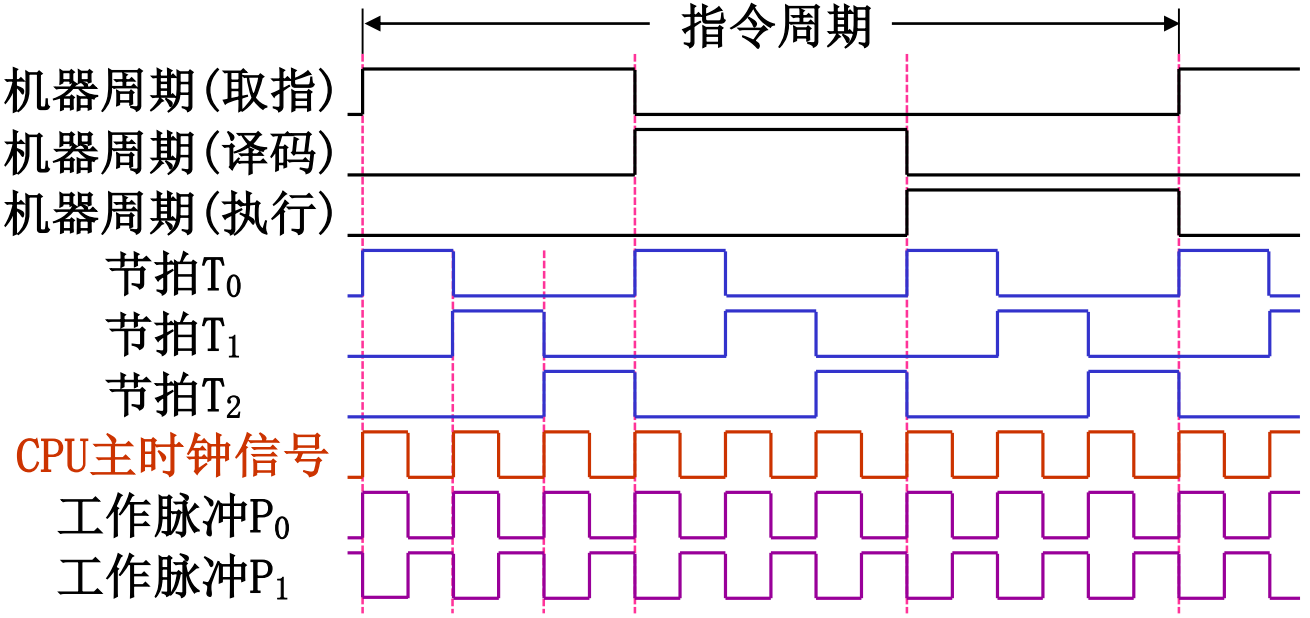
*时序信号：表示时长、次序，信号间无间隙及重叠

（1个μOP）　（μOP序列）

1、时序系统的组织

*时序信号类型：机器周期、节拍、工作脉冲

划分依据—　基本功能、　μOP、　同步脉冲(如电位-脉冲制)



*早期计算机的时序系统：三级时序(机器周期/节拍/工作脉冲)

信号的个数—按最复杂情况设置

例：指令周期＝$\sum_{i=1}^{x}$机器周期$_i$，机器周期＝$\sum_{i=1}^{m}$节拍$_i$，

节拍＝$\sum_{i=1}^{k}$工作脉冲$_i$

注：机器周期的个数需多设置1个(用于中断周期)

信号的周期—有定长、变长2种类型，

应用：不同指令周期的x、m不同(性能好)，

k为常数(与指令类型无关)

信号表示的功能—可表示时间次序、操作步骤

(易理解)　(可简化电路)

例：时序信号有5个，add指令中wb_r的表示方法

用t4表示时--GPRs的RegWr＝(add+ori)·t4+lw·t5+

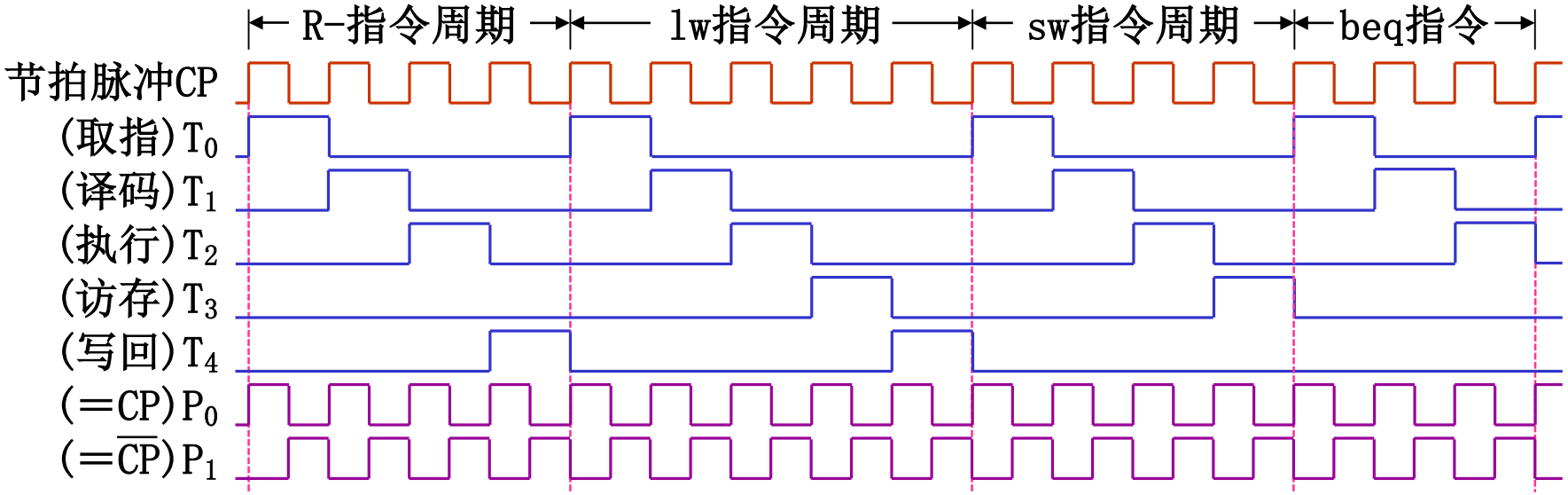
用t5表示时--GPRs的RegWr＝(add+ori+…)·t5

*现代计算机的时序系统：节拍/工作脉冲两级时序

信号的个数—按最复杂情况设置

信号的周期—常采用变长类型

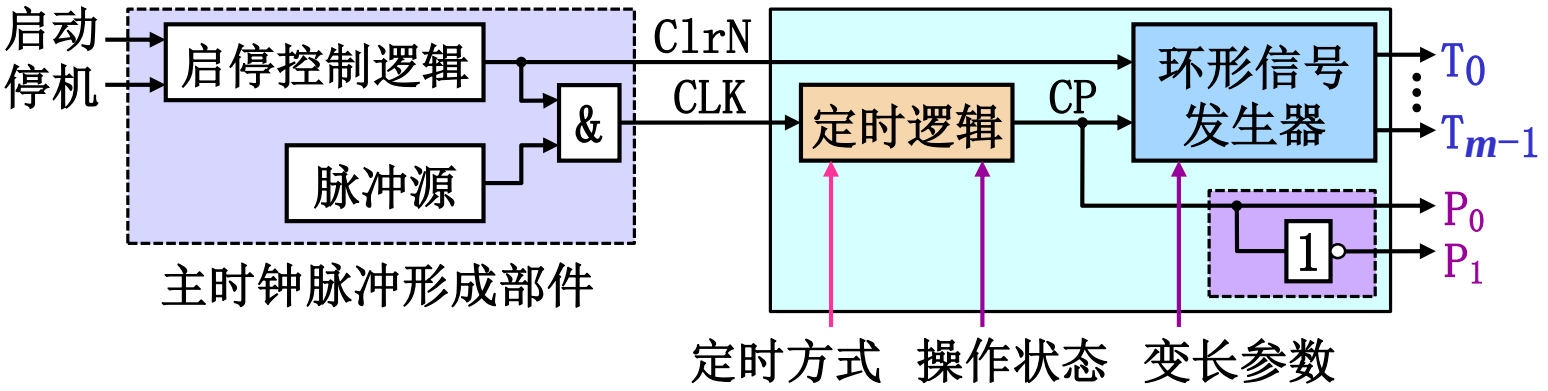
例：支持7条MIPS指令的时序系统，每个μOP在一个CP内完成



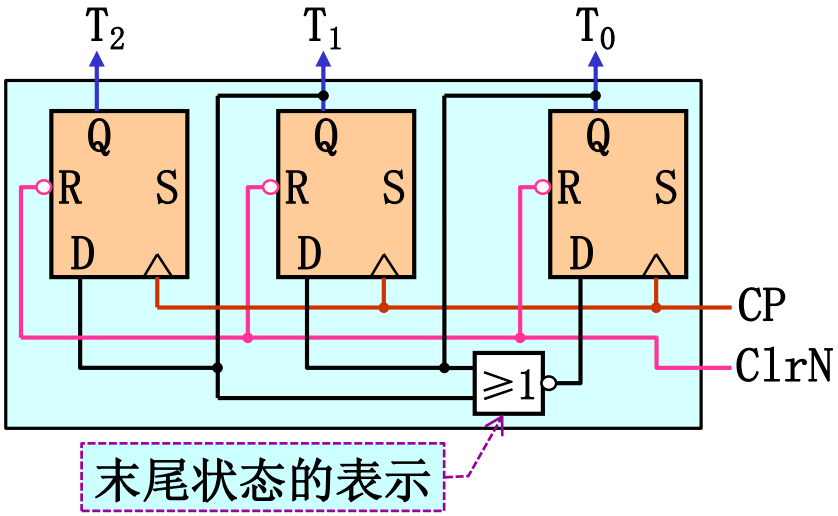
2、时序信号形成电路的组成

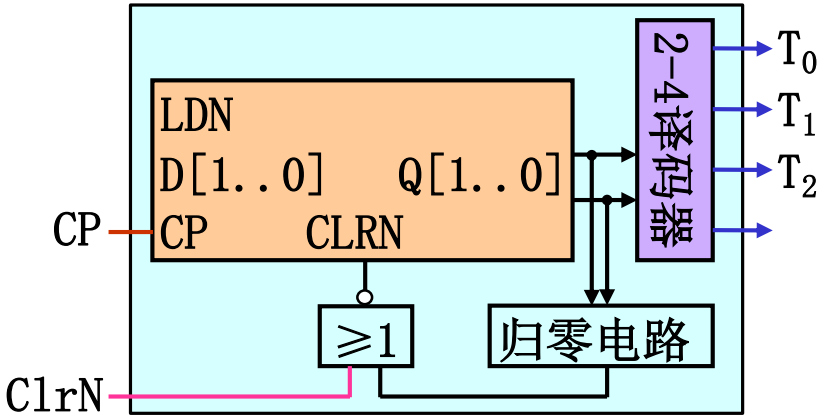
***时序信号形成电路组成：** 环形信号发生器、定时逻辑

（产生节拍、工作脉冲）（控制信号时长）



***信号发生器组成：** 有移位REG、计数器＋译码器2种





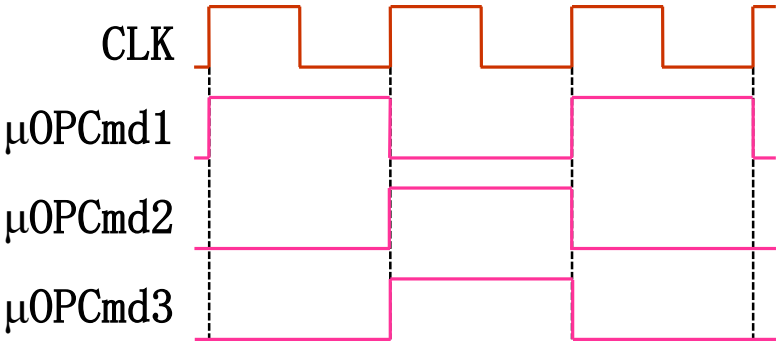
3、 μOP的定时方式　　--又称控制器的控制方式

指μOP序列中各μOP时长的控制方法，即CP时长的控制方式

*同步控制方式：

各μOP的时序只受统一的基准时钟信号(主时钟脉冲信号)控制

定时原理—μOPCmd发出与时钟信号同步，节拍周期$T_{CP}=1T_C$



定时逻辑— CP与时钟信号CLK同步，即CP＝CLK

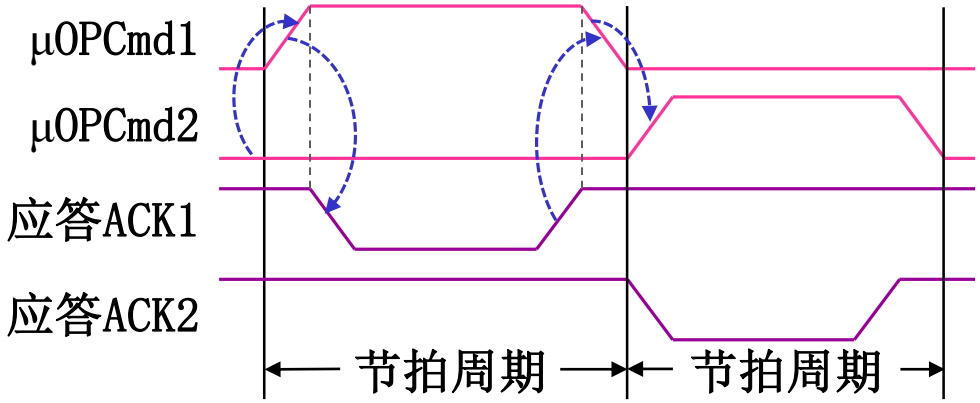
特点— 控制简单、时间浪费大，适合CPU内部的μOP定时(时延相近)

*异步控制方式：　--又称应答方式或握手方式

各μOP的时序只受专门的联络信号(应答信号)控制

定时原理—μOPCmd串行发出，收到应答信号时本μOP完成，

节拍周期 $T_{CP} = T_{收到应答} - T_{发出命令}$



定时逻辑— CP与应答信号同步，即CP＝ACK_i

特点— 时间浪费小、控制复杂，适合CPU与外部的μOP定时

(时延相差较大)

*联合控制方式：　--又称半同步方式

各μOP的时序受基准时钟信号及联络信号的共同控制

定时原理—基础为同步控制方式，支持异步控制方式；

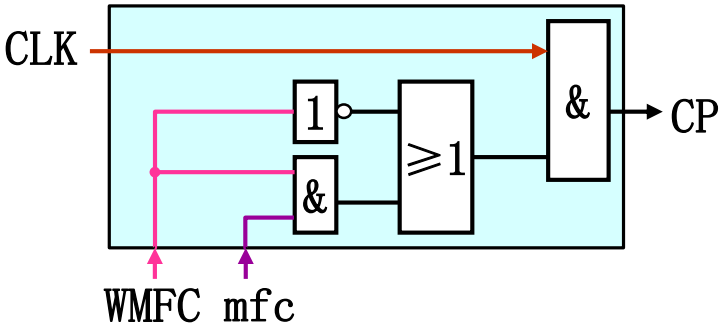
$T_{CP}=kT_C$，整数 $k=1$(同步时)或>1(异步时)

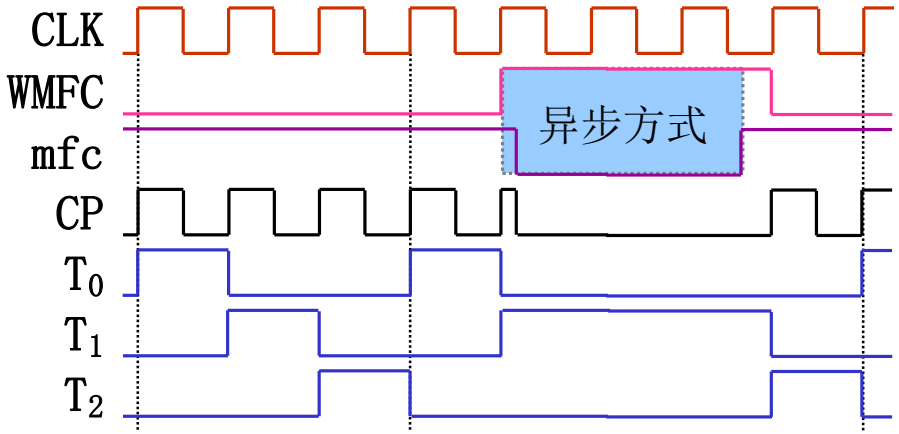
控制方式转换的实现— 延长节拍信号CP，用μOPCmd控制

定时逻辑— 以MEM读写μOP为例，设信号为mfc及WMFC(0为同步)

则 WMFC＝0时CP＝CLK，WMFC＝1时CP＝mfc·CLK；

即 $CP=(\overline{WMFC}+WMFC\cdot mfc)\cdot CLK$





应用方法—用μOP设置方式

特点—可有效处理μOP时延不同问题，适合CPU的所有μOP定时

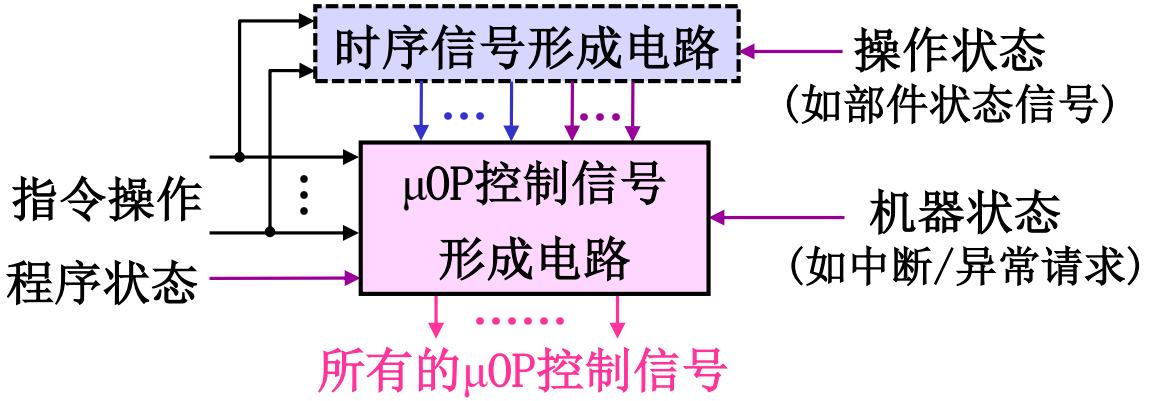
三、uOP控制信号的形成

*CU的任务：循环地、有序地产生工作流程所需的μOP控制信号

实现—时序信号形成电路　μOP控制信号形成电路

* μOP控制信号形成电路组成：

输入—指令操作、程序/机器状态、时序信号



输出—所有的μOP控制信号，来自状态转换图

内部逻辑—组合逻辑(硬布线控制器)，存储逻辑(微程序控制器)

实质：　编码器　　微主机

(每个输出都是输入的函数)　(输出由执行微指令产生)

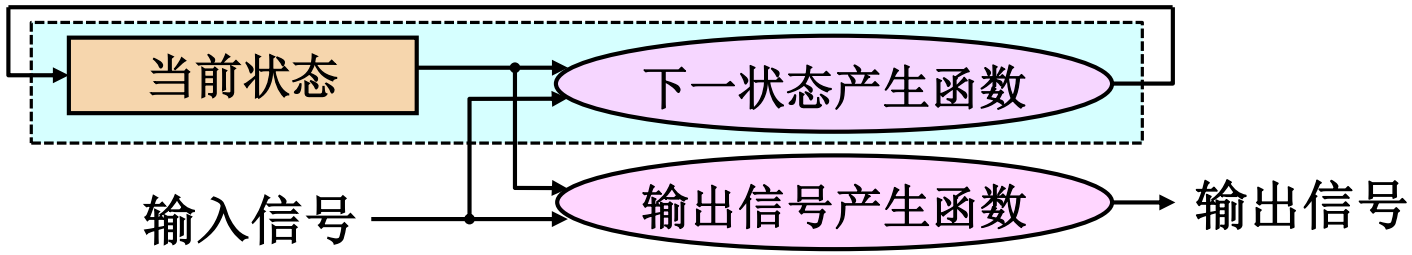
§6.3 硬布线控制器的设计

*硬布线CU的实现要求：

两级时序信号，组合逻辑形成μOP控制信号

*硬布线CU的实现方法：

有限状态机(FSM)模型— 当前状态＋2个函数



FSM模型的应用—

时序信号形成电路

当前状态：所有的时序信号，用状态部件保存

下一状态产生函数：环形信号发生器的内部逻辑

输出信号产生函数：μOP控制信号形成电路的内部逻辑

一、CU的设计步骤

第1步—形成状态转换图　　--基于数据通路

根据各指令功能需求，列出并汇总

要求：注明状态转换条件(OP及寻址方式、程序状态、机器状态)

第2步—组织时序系统　　--基于状态转换图

确定时序信号个数(含功能)、各种时序信号序列、定时方式

(最长路径[2级]/步骤)　　(变长周期)　　(联合方式)

要求：每种信号序列都需包含适用条件(同状态转换条件)

第3步—设计时序信号形成电路　--基于时序信号序列

(1)确定每个时序信号的下一状态产生函数

(2)实现信号表示、下一状态函数、复位逻辑、定时逻辑

(触发器)　　(组合逻辑)　　(末尾状态表示)　(CP与CLK关系)

第4步—设计μOP控制信号形成电路 --基于状态图及信号序列

(1)列出μOPCmd的使用时间表：

画出使用时间表(行为所有μOPCmd、列为所有时序信号)，

给每个状态打上时间戳(状态转换条件用时序信号表示)，

将每个状态的每个μOPCmd的转换条件填入表中

(2)获得μOPCmd的逻辑表达式(按行汇总、化简)

(3)实现μOPCmd的逻辑表达式(每个μOPCmd一个电路)

第5步—整合成CU

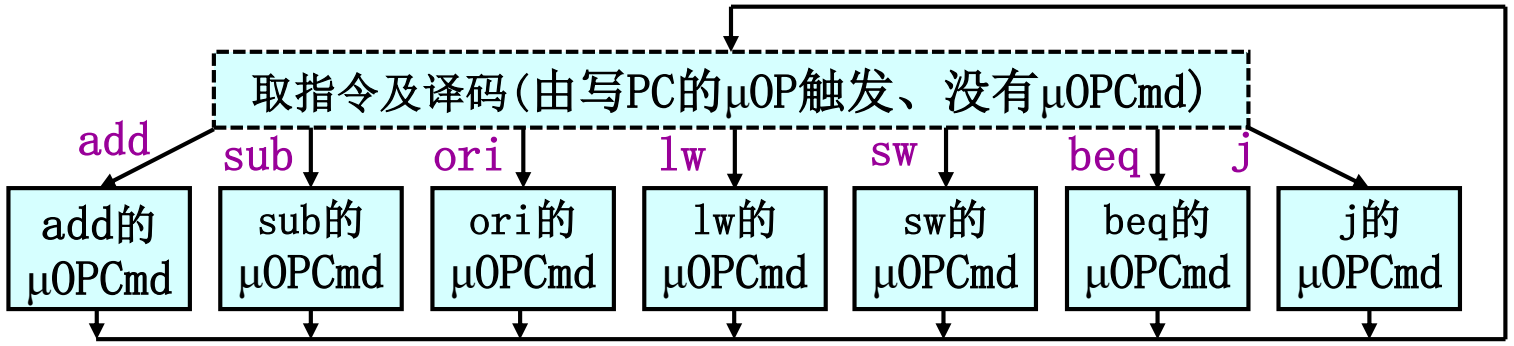
连接ID、时序信号形成电路、 μOP控制信号形成电路

说明： ID设计未提及(太简单)

二、单周期CU的设计

*设计背景：支持7条MIPS指令的数据通路

*形成状态转换图：μOPCmd序列仅1步，状态转换条件为操作码



*组织时序系统：0个节拍信号(路径长度为1)、2个工作脉冲，无时序信号序列，同步方式定时

*设计时序信号形成电路：

(1)无节拍信号，下一状态＝当前状态

(2)无需信号发生器、P0＝CP、P1＝$\overline{CP}$，CP＝CLK

*指令执行过程的状态转换图：

	Branch	Jump	Extctr	ALUBsrc	ALUctr	RegAsrc	RegDsrc	RegWr	MemRd	MemWr
add	0	0	×	1	00	1	1	1	0	0
sub	0	0	×	1	01	1	1	1	0	0
ori	0	0	0	0	10	0	1	1	0	0
lw	0	0	1	0	00	0	0	1	1	0
sw	0	0	1	0	00	×	×	0	0	1
beq	1	0	×	1	11	×	×	0	0	0
j	0	1	×	×	×	×	×	0	0	0

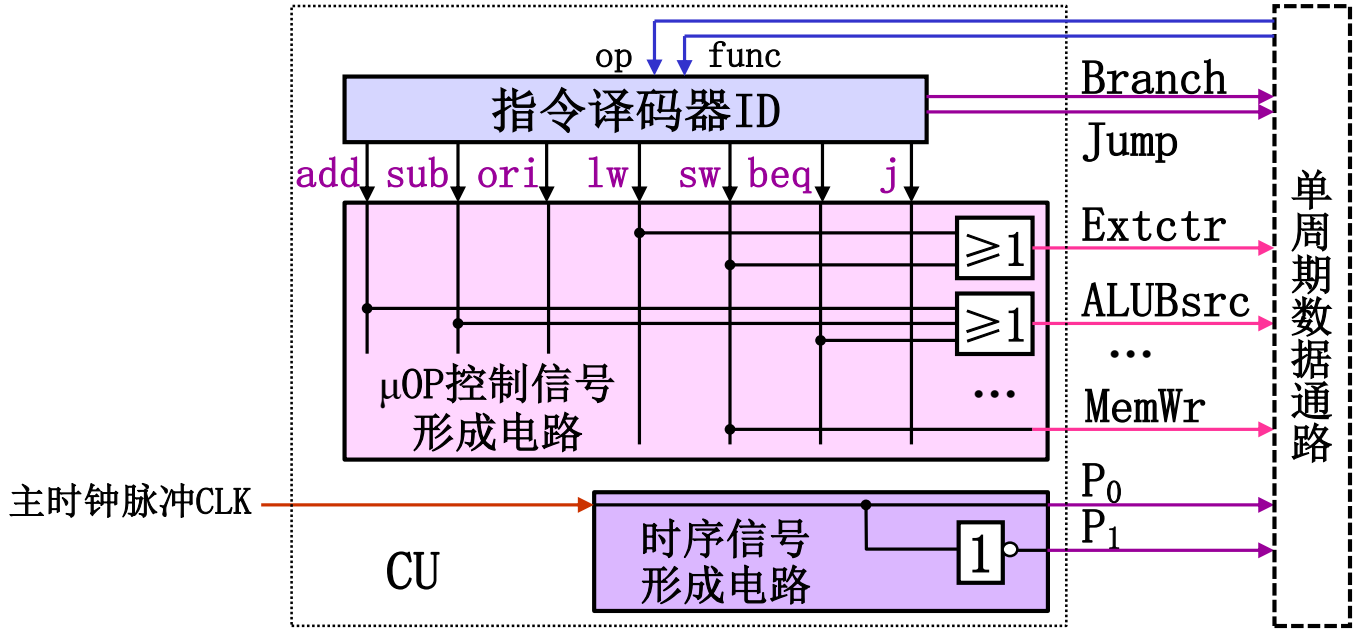
*设计μOP控制信号形成电路：

(1)μOPCmd使用时间表只有1列，无时间戳

Extctr	lw＋sw	RegAsrc	add＋sub
ALUBsrc	add＋sub＋beq	RegDsrc	add＋sub＋ori
ALUctr	[1]=ori＋beq, [0]=sub＋beq	RegWr	add＋sub＋ori＋lw
MemRd	lw	MemWr	sw

(2)(3)Extctr＝lw＋sw，ALUctr[1]＝ori＋beq，MemRd＝lw，…

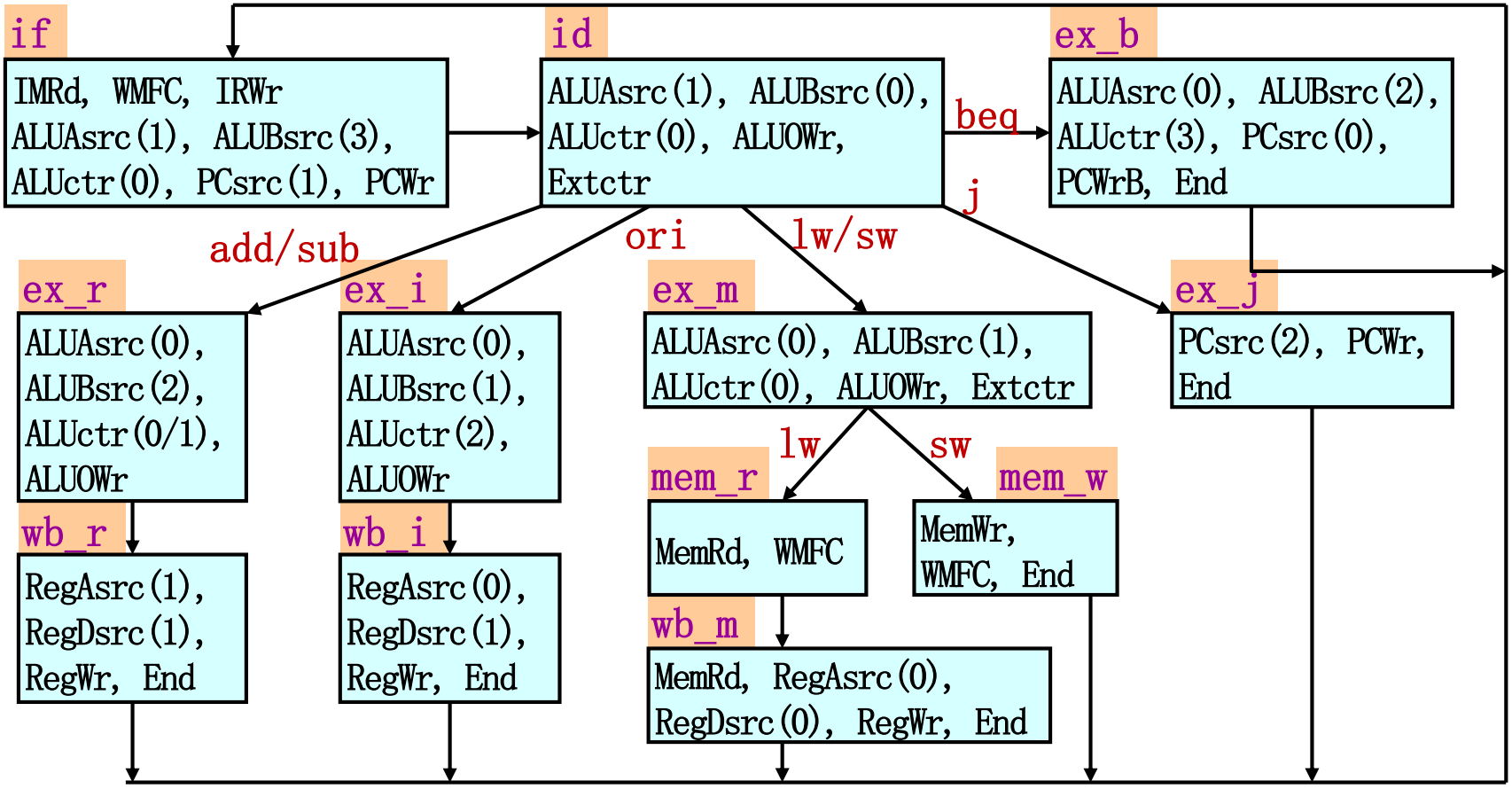
*整合成CU：连接相关电路



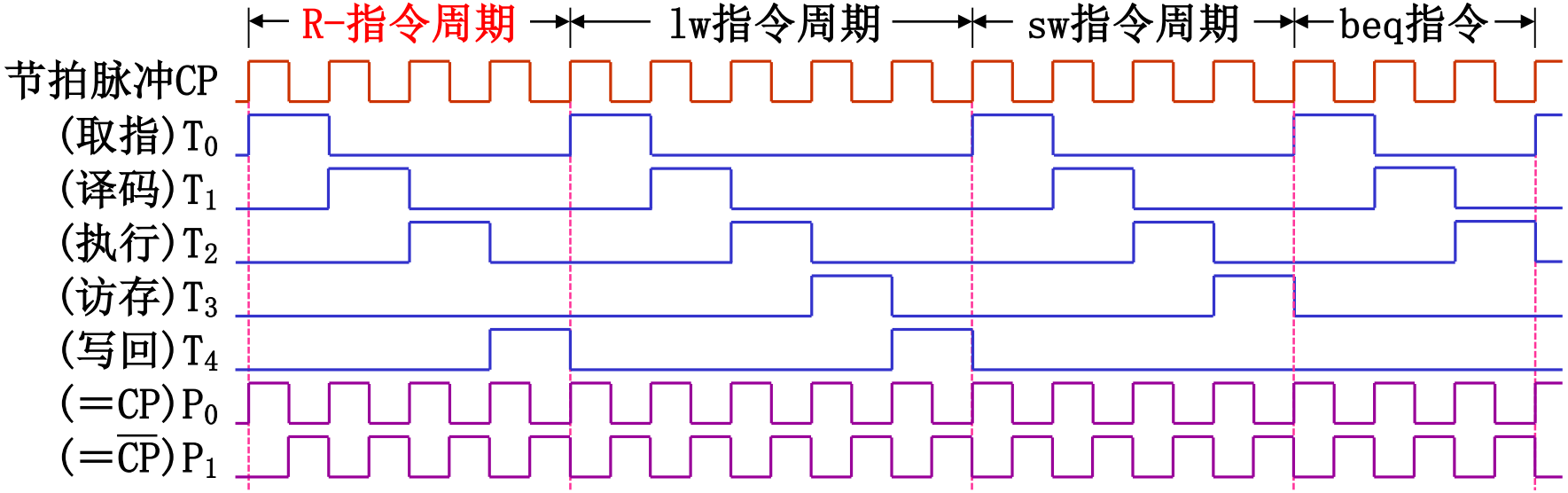
三、多周期CU的设计

*设计背景：支持7条MIPS指令的数据通路，且MEM时延可变

*形成状态转换图：μOPCmd序列≤5步，状态转换条件为操作码



*组织时序系统：5个节拍信号(表示操作步骤)、2个工作脉冲，
4种时序信号序列，联合方式定时



*设计时序信号形成电路：

(1)确定下一状态产生函数：

$T_1=T_0$，$T_2=T_1$，$T_3=(lw+sw)\cdot T_2$，

$T_4=(add+sub+ori)\cdot T_2+lw\cdot T_3$，

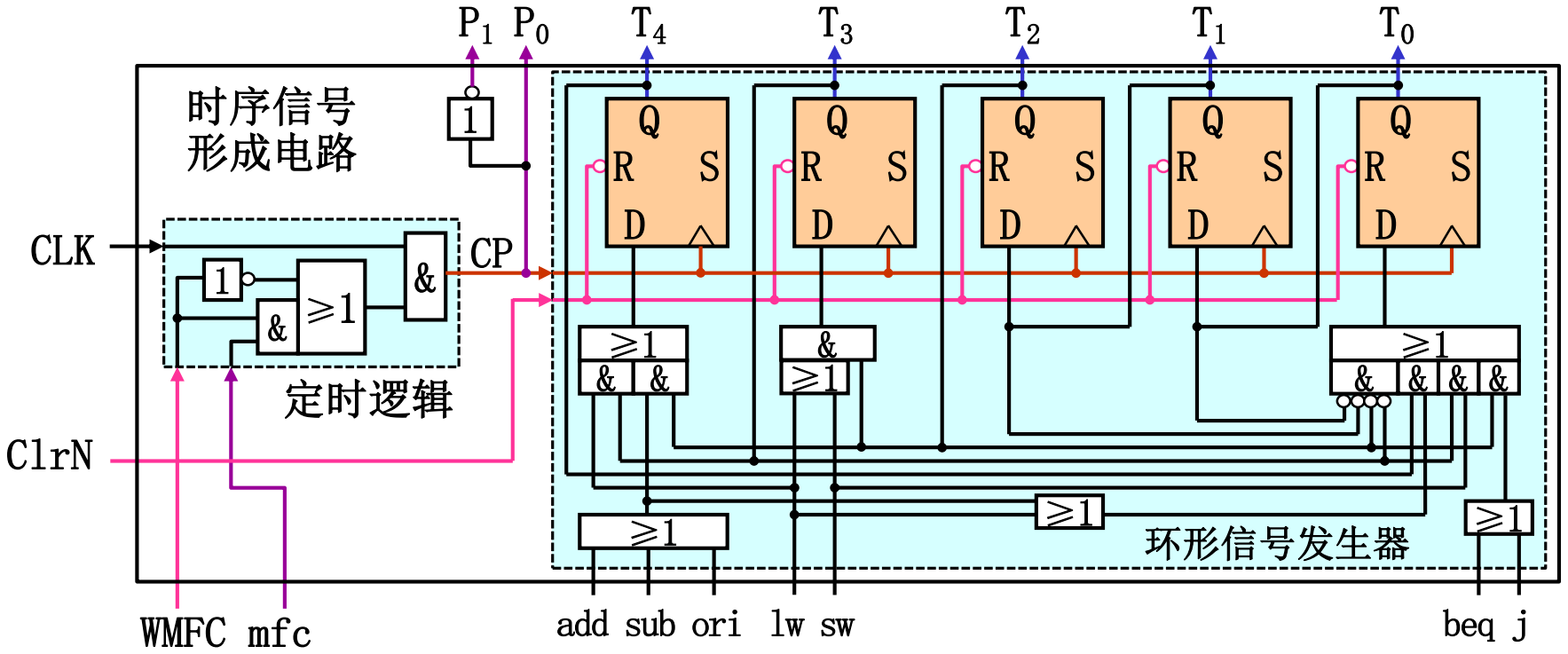
$T_0=(add+sub+ori+lw)\cdot T_4+sw\cdot T_3+(beq+j)\cdot T_2+\overline{T_0}\cdot\overline{T_1}\cdot\overline{T_2}\cdot\overline{T_3}$

(2)电路实现：信号表示(5个触发器)，

下一状态产生函数(5个组合逻辑电路)，

复位逻辑(复位触发器)，

定时逻辑(同步/异步分类定时)



*设计μOP控制信号形成电路：

(1)μOPCmd使用时间表有17行5列，状态转换条件为操作码

	T_0	T_1	T_2	T_3	T_4
PCsrc	All(1)		beq(0)+j(2)		
PCWr	All		j		
PCWrB			beq		
IMRd	All				
IRWr	All				
RegAsrc					add|sub(1)+ori|lw(0)
RegDsrc					add|sub|ori(1)+lw(0)
RegWr					add|sub|ori|lw
Extctr		All	lw|sw		
ALUAsrc	All(1)	All(1)	add|sub|ori|lw|sw|beq(0)		
ALUBsrc	All(3)	All(0)	add|sub|beq(2)+…		
ALUctr	All(0)	All(0)	add|lw|sw(0)+sub(1)+…		
ALUOWr		All	add|sub|ori|lw|sw		
MemRd				lw	lw
MemWr				sw	
WMFC	All			lw|sw	
End			beq|j	sw	add|sub|ori|lw

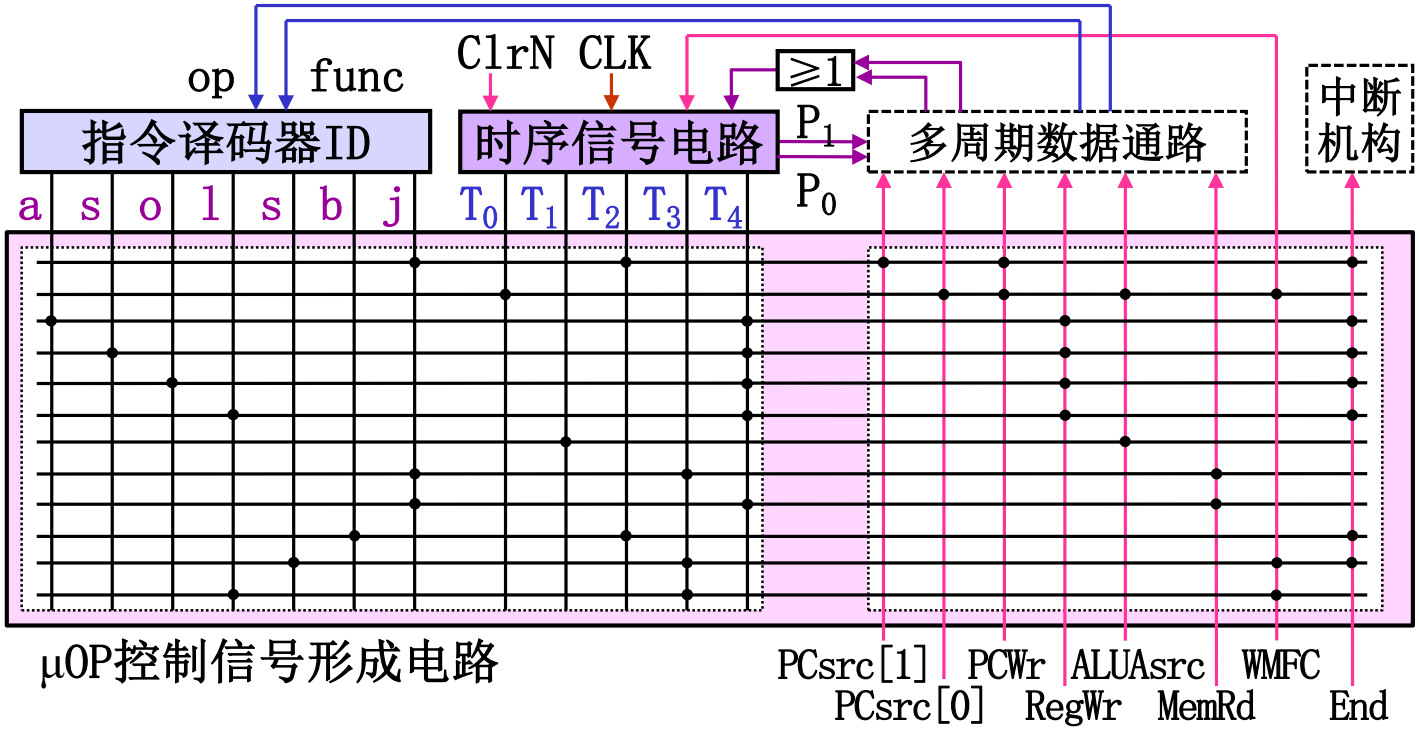
(2)按行汇总使用时间表，μOPCmd的逻辑表达式为：

$PCsrc[1]=T_2\cdot j$，$PCsrc[0]=T_0$，$PCWr=T_0+T_2\cdot j$，

$RegWr=T_4\cdot(add+sub+ori+lw)$，…，

$End=T_2\cdot(beq+j)+T_3\cdot sw+T_4\cdot(add+sub+ori+lw)$

(3)用组合逻辑电路实现每个μOPCmd



作业5-3：P237— 17、19

§6.4 微程序控制器的设计

一、微程序控制思想

*微程序控制思想：　　—类似于存储程序工作方式

①每条指令的执行过程(μOPCmd序列)都用微程序表示，所有微程序都存放在专用MEM中

②CU自动、逐条取出并执行微指令(产生μOP控制信号)

简化CU的实现

*相关术语：

微命令—部件的操作控制信号　　(～1个μOPCmd)

微指令—用格式及编码表示、可同时执行的一组微命令

微程序—实现特定功能的微指令序列　(～μOPCmd序列)

控制存储器(CS)—专用于存放微程序的ROM，按微地址访问

微指令周期—从CS中取出并执行一条微指令的时间


*CPU工作流程的微程序结构：

微程序种类—取指、中断等公用微程序，各指令功能微程序

控制存储器CS

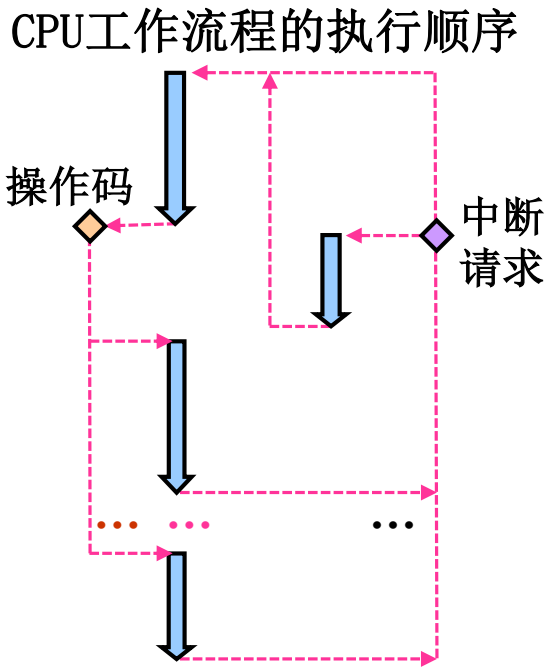
	…	…
M	微命令	M+1
M+1	微命令	M+2
M+2	微命令	***
N	…	…
…	微命令	M
P	微命令	P+1
P+1	微命令	P+2
P+2	微命令	M或N
…	…	…
Q	微命令	Q+1
Q+1	微命令	M或N
	…	…

取指微程序

中断微程序

add指令微程序

j指令微程序



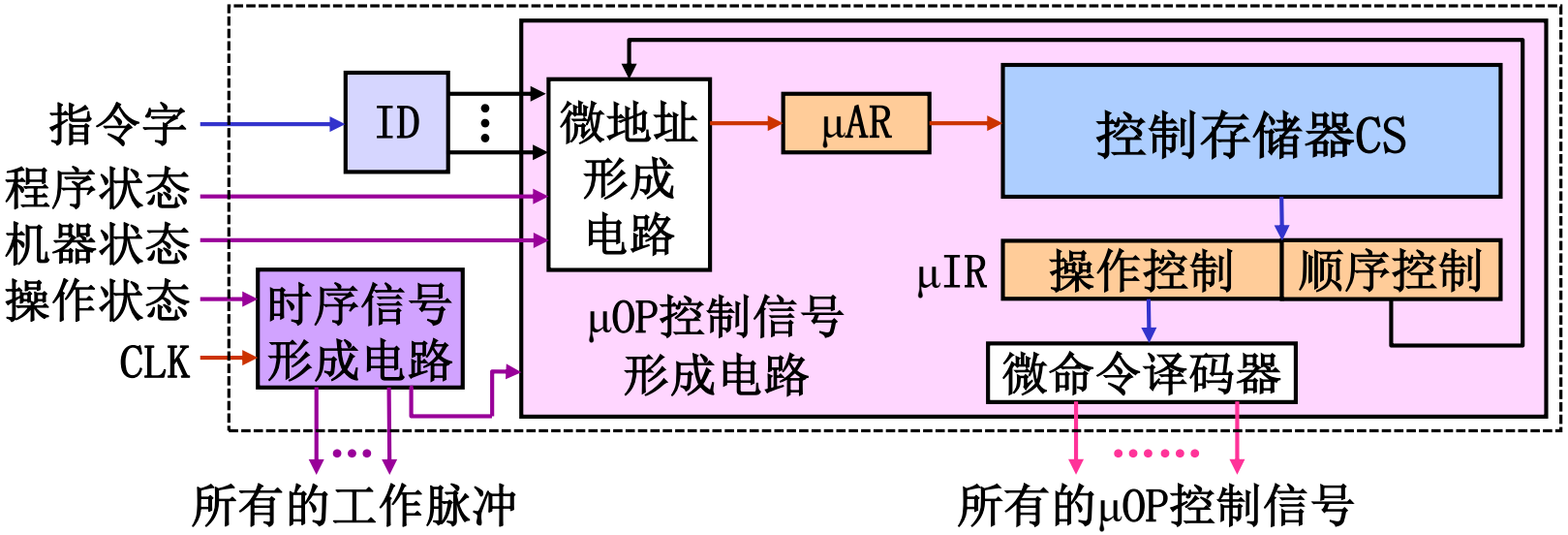
微指令格式—

微程序结构—最后一条微指令为跳转型，其余为顺序型

（微程序的功能所决定）

二、微程序CU的组成与工作原理

*基本组成：结构与硬布线CU相同，内部电路有差别



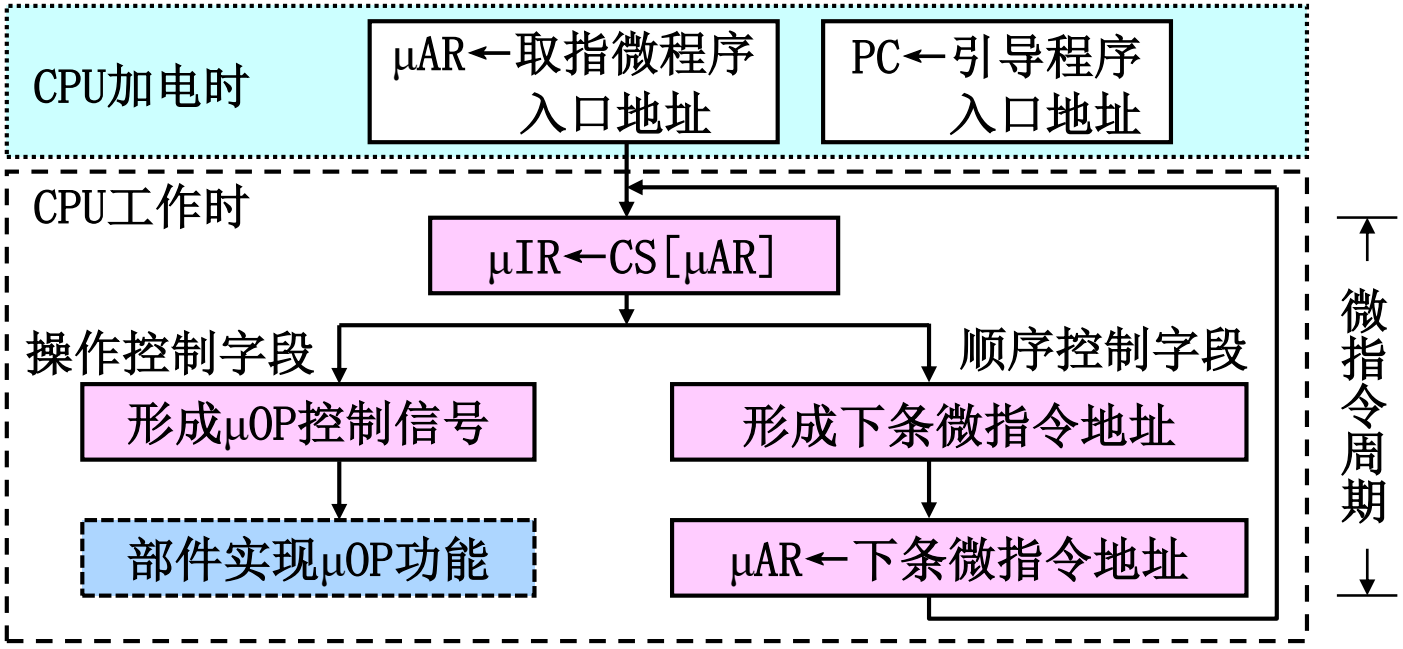
μOP控制信号形成电路—微指令部件＋CS （～组合逻辑电路）

μAR、μIR、微命令译码器、微地址形成电路

时序信号形成电路—一级时序 （～两级时序）

信号的周期：1个微指令周期，比μOPCmd多1个工作脉冲

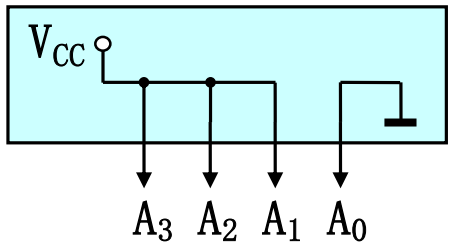
*工作原理：循环地取出并执行微指令



微指令周期—取微指令、执行微指令　　（～硬布线CU的节拍）

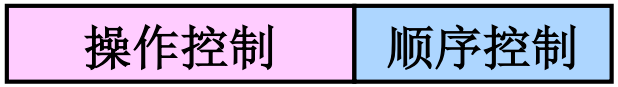
μAR初值—加电时由硬件产生，还需产生PC(微程序中需使用)

如：地址1110的产生电路为



三、微指令格式

*微指令格式：定长指令字结构 ←控制简单、执行速度快



1、微指令编码方式 --操作控制字段的编码

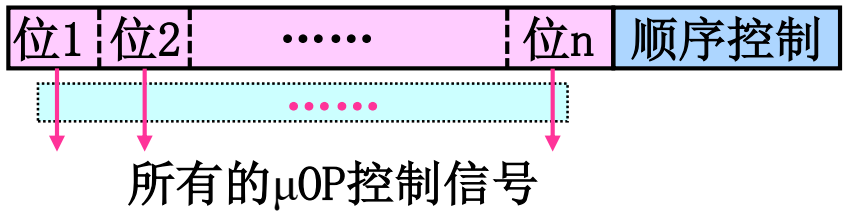
所有微命令的编码方式，决定了微命令译码器的组成

*直接编码方式：

编码—位编码，每个微命令用一位编码表示；

字段总长度＝n位 （n为所有微命令个数）

μOP控制信号形成—直接形成



*字段直接编码方式：

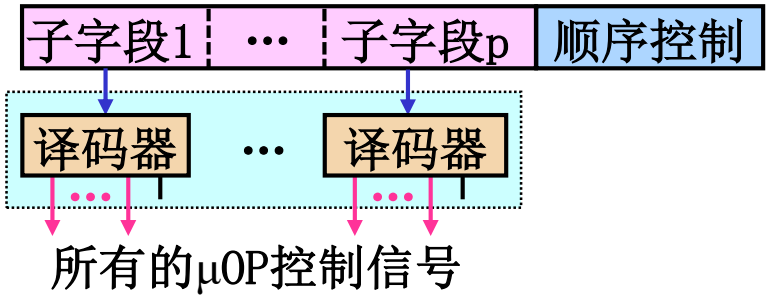
编码—子字段编码，微命令用某个 子字段的编码表示；

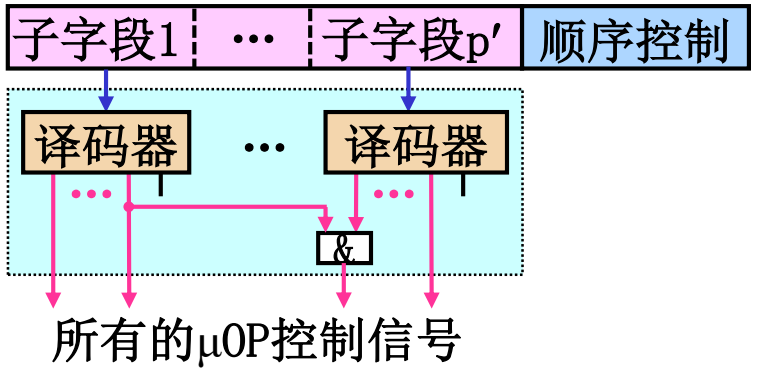
子字段中的微命令须互斥(同时最多一个有效)，

子字段长度=⌈$\log_2$(定义的微命令数+1)⌉

所有微命令都无效时

μOP控制信号形成—各子字段单独译码





*字段间接编码方式：

编码—子字段编码，微命令用多个 子字段的编码表示

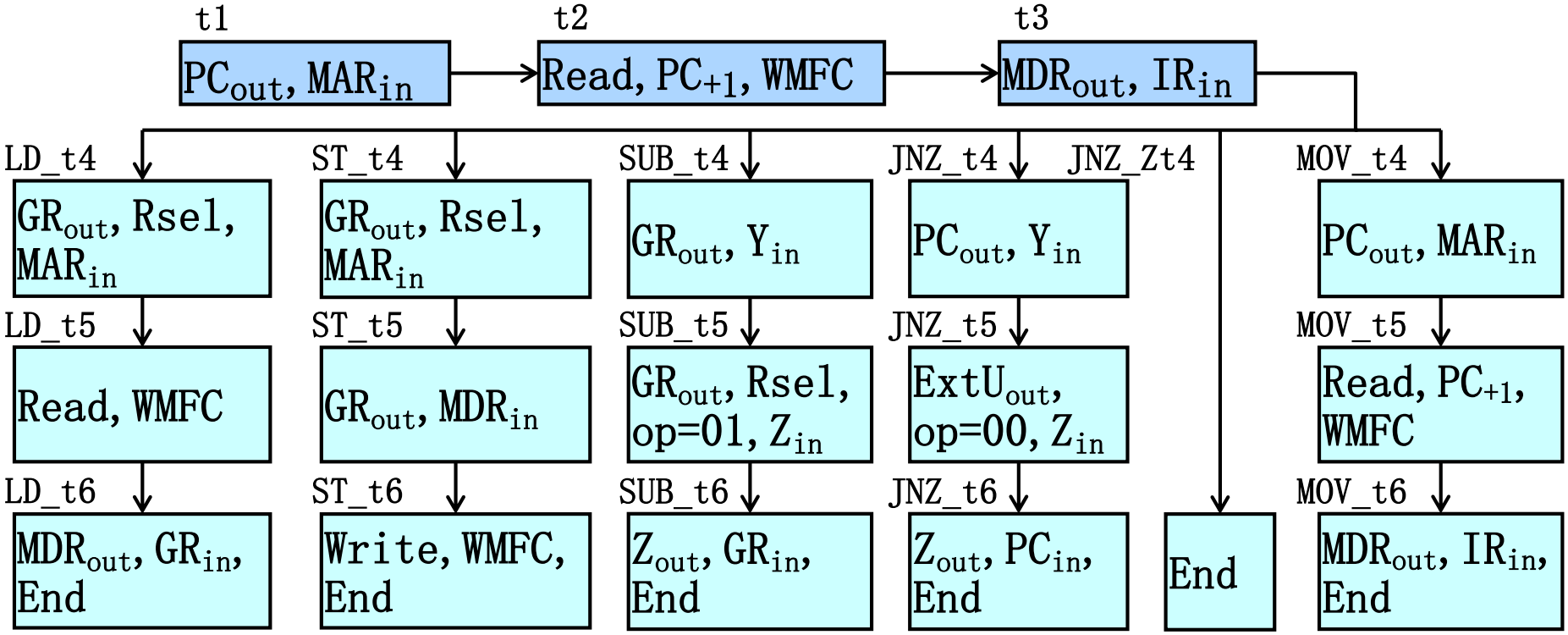
(如子字段p′的码7及子字段1的码1)

子字段中的微命令须互斥，子字段个数较少(p′<p)

μOP控制信号形成—子字段单独译码+逻辑电路组合

例1—对单总线结构的Demo_IS数据通路(支持5条指令)，若用微程序控制方式实现CU，请设计微指令格式的操作控制字段。

解：(1)所有指令执行的状态转换图(μOPCmd序列)如下：



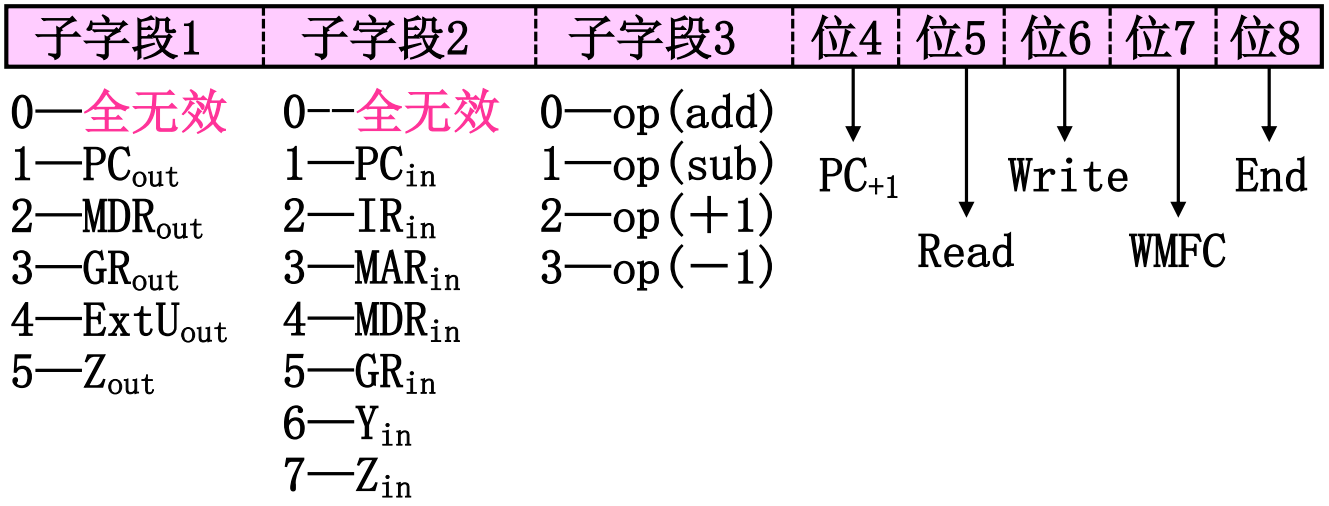
(2)微命令互斥性分析：（基于同一状态）

- PC_{out}、MDR_{out}、GR_{out}、$ExtU_{out}$、Z_{out}互斥(分时输出)
- PC_{in}、IR_{in}、MAR_{in}、MDR_{in}、GR_{in}、Y_{in}、Z_{in}互斥(1个接收者)

  注：Z_{in}不使用总线通路，可能不互斥，本例互斥
- op、PC_{+1}、Read、Write、WMFC、End中(暂不考虑Rsel)，

  op单独编码(00~11)，其余均独立编码(偷懒)

(3)操作控制字段的编码：3个子字段＋5位，共13位



2、微地址形成方式　　--顺序控制字段的编码

形成下条微命令地址的方式，决定了微地址形成电路的组成

*顺序控制字段的组成：方式位＋地址参数

*微地址形成方式的种类：(微指令寻址方式)

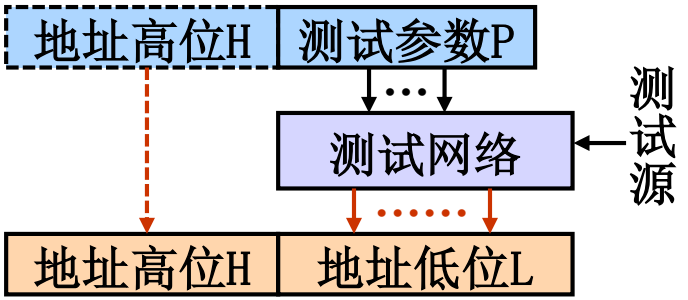
计数器法—μAR＝(μAR)＋1，地址参数隐含，适于顺序型

下址法—μAR＝(地址参数)，适于顺序型、跳转型

测试网络法—

μAR＝*f*(测试源,测试参数)，

适于(多路)分支型



硬件产生法—μAR＝固定地址，适于硬件初始化

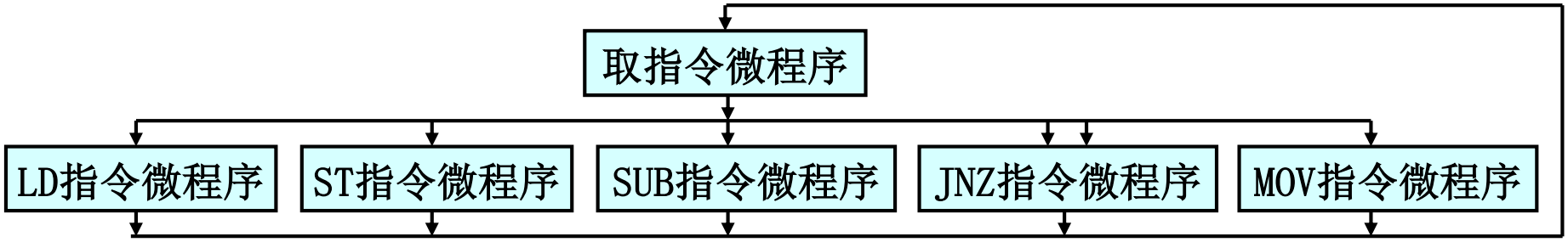
*微地址形成方式的常见应用：

增量法—计数器法＋测试网络法，地址参数长度不同

断定法—　下址法＋测试网络法，地址参数长度相同

例2—对单总线结构的Demo_IS数据通路(支持5条指令)，若用微程序控制方式实现CU，请设计微指令格式的顺序控制字段。

解：(1)微程序结构如下，共有19条微指令，有顺序、跳转、多路分支3种指令寻址方式



(2)采用断定法形成微地址：

下址法中，下址值的位数为 $\lceil \log_2 19 \rceil = 5$位，

测试网络法中，测试方法仅有1种，测试源为操作码及ZF

(3)顺序控制字段的编码：1位方式位F＋5位地址参数P

F＝0—下址法，P为下址值
F＝1—测试网络法，P空闲

3、微指令格式

采用定长指令字结构，有水平型、垂直型2种风格

***水平型微指令格式：** --多个μOP并行(水平)，注重指令功能

采用定长编码，宜采用断定法寻址，微指令字长较长

操作码(多个μOP)	方式位0	下地址	
操作码(多个μOP)	方式位x	地址高位	测试参数

***垂直型微指令格式：** --多个μOP串行(垂直)，注重指令字长

采用变长编码，宜采用增量法寻址，微指令字长较短

顺序型：

μOP类型	源地址	目的地址	子功能f

转移型：

转移类型a	转移条件	地址参数
转移类型b	目标地址	

***常用微指令格式：** 水平型格式 (CPU的重点是性能)

执行μOP能力强(指令速度快)，代码效率低(数量有限)

四、微程序控制单元的设计

1、设计步骤

第1步—列出所有的μOPCmd序列　　--基于数据通路

根据各指令功能需求列出(同硬布线控制器)

第2步—设计微指令格式　　--基于微程序结构

确定微指令格式类型(常为水平型)，

进行操作控制字段、顺序控制字段的编码

第3步—编制微程序　　--基于微指令格式

确定各微程序在CS中的位置，转换所有μOPCmd序列

第4步—设计相关电路　　--基于微指令格式

包括微命令译码器、微地址形成电路

2、设计举例

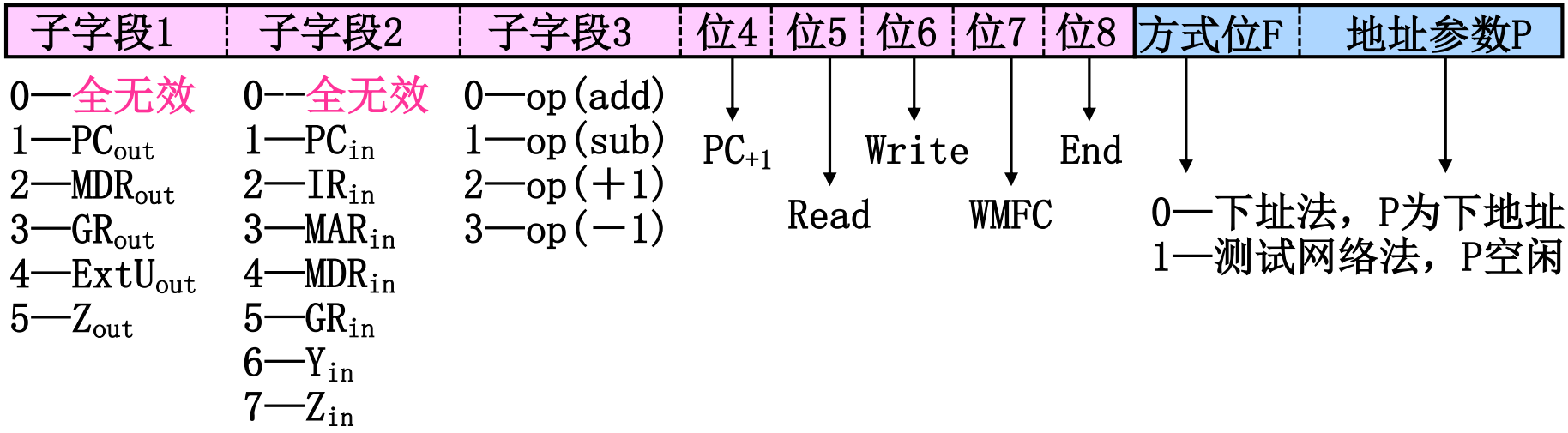
*设计背景：单总线结构的Demo_IS数据通路(支持5条指令)

*列出所有的μOPCmd序列：5条指令(课件P74)

*设计微指令格式：采用水平型格式

操作控制字段编码(课件P75)

顺序控制字段编码(课件P77)



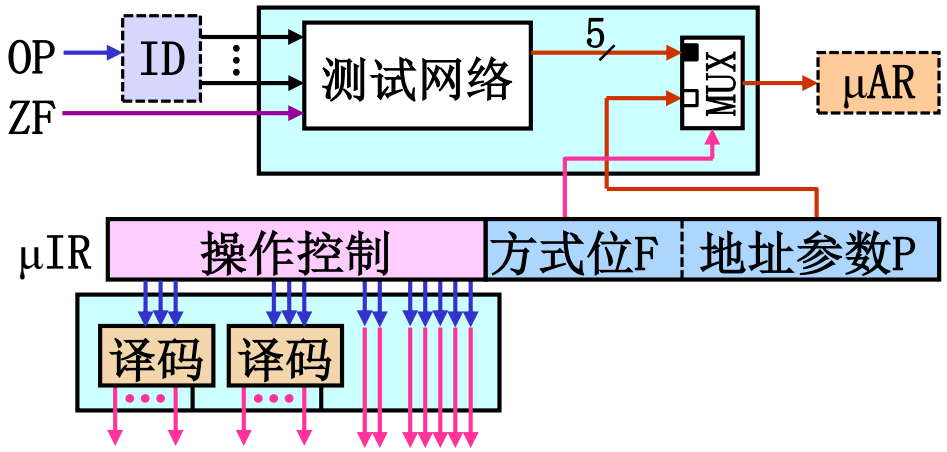
74 75 77

*编制微程序：取指、LD、ST、SUB、JNZ、MOV连续存放

地址	控制存储器CS	
00000	001 011 00 00000 000001	取指微程序
00001	000 000 00 11010 000010	
00010	010 010 00 00000 100000	
00011	011 011 00 00000 000100	LD指令微程序
00100	000 000 00 01010 000101	
00101	010 101 00 00001 000000	
00110	011 011 00 00000 000111	ST指令微程序
00111	011 100 00 00000 001000	
01000	000 000 00 00111 000000	

地址	控制存储器CS	
01001	011 110 00 00000 001010	SUB指令微程序
01010	011 111 01 00000 001011	
01011	101 101 00 00001 000000	
01100	001 110 00 00000 001101	JNZ指令微程序
01101	100 111 00 00000 001110	
01110	101 001 00 00001 000000	
01111	000 000 00 00001 000000	
10000	001 011 00 00000 010001	MOV指令微程序
10001	000 000 00 11010 010010	
10010	011 101 00 00001 000000	

*设计相关电路：微命令译码器、微地址形成电路



OP	ZF	$A_4A_3A_2A_1A_0$	
LD		0 0 0 1 1	A_4=
ST		0 0 1 1 0	A_3=
SUB		0 1 0 0 1	A_2=
JNZ	0	0 1 1 0 0	A_1=
JNZ	1	0 1 1 1 1	A_0=
MOV		1 0 0 0 0	

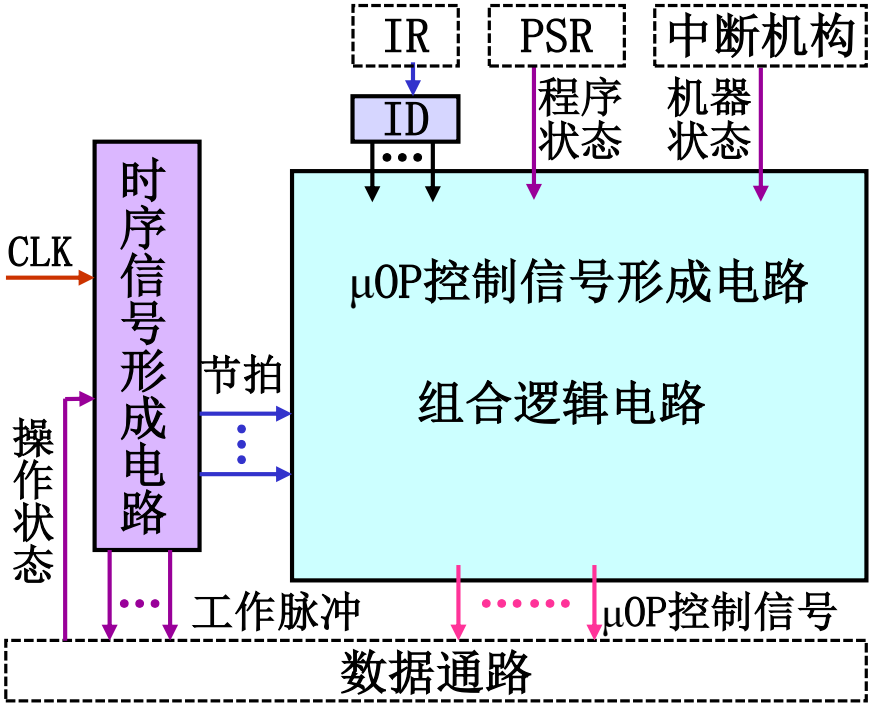
3、硬布线、微程序控制器组成对比

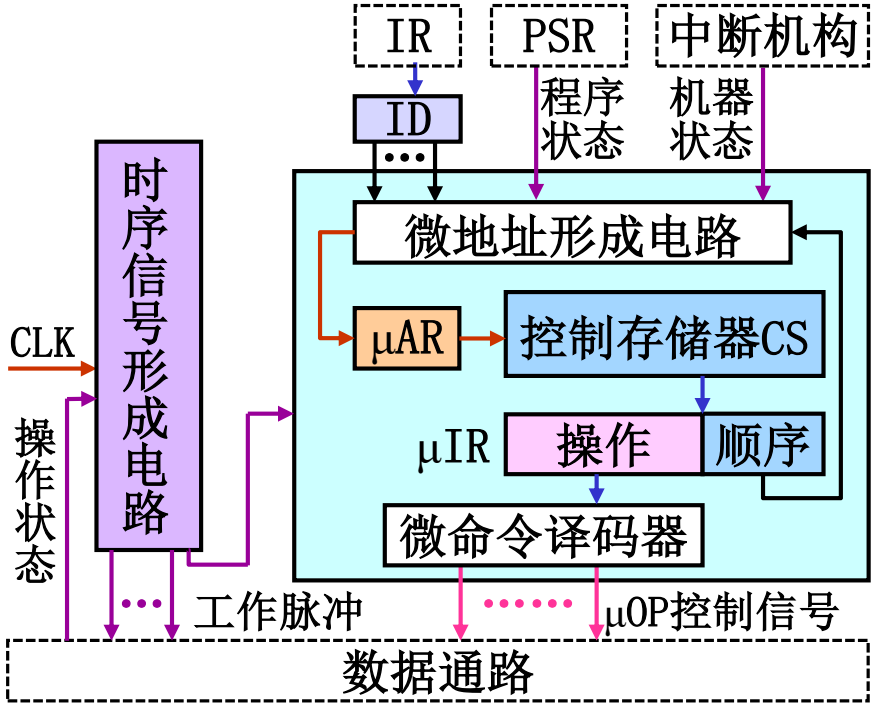
CU组成—ID、时序信号形成电路、μOP控制信号形成电路

ID组成—按指令格式译码，输出OP、寻址方式信号

时序信号形成电路组成— 2级时序，1级时序

μOP控制信号形成电路组成— 组合逻辑电路，微主机





作业5-4：P237— 23

§6.6 异常及中断的处理

一、异常及中断的基本概念

*事件(Event)：改变程序正常执行顺序的特殊情况

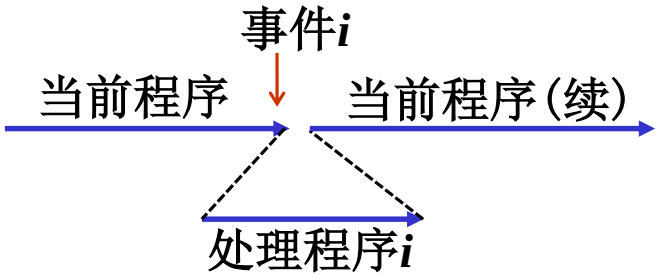
(指令约定的顺序)

↑

CPU必须处理

*事件类型：异常(Exception)、中断(Interrupt) ←按发生位置分类

*事件处理方法：执行相应的处理程序



1、异常　　--内部异常(程序性异常)

由CPU内部执行指令所引起的意外事件（如除零、断点）

*处理时机：立即处理

*异常分类：按报告及返回方式分

	故障(Fault)	陷阱(Trap)	终止(Abort)
报告结果	可能修复①的异常	预先安排②的异常	不可修复的异常
返回方式	当前指令 或 终止程序 (可修复的)　(无法修复的)	下条指令	终止程序 或 关机 (能/否定位到程序)
示例	缺页，除零、溢出③	单步，调用、断点	无效表、硬件故障
检测时机④	随时	指令结束时	随时

说明：①指程序本身可以捕获，而Abort只能由系统捕获

②安排有两种方式(设置触发条件、执行特殊指令)；

③溢出常用作可选择的异常(用陷阱指令INTO触发)；

④处理时机常通过检测时机控制(一检测到就处理)

如 add R1,R2,R3 或 add R1,R2,R3
…　　INTO
…

2、中断　　--外部中断

同步事件指与指令执行相关

由CPU外部的设备产生的请求事件，常称为异步事件

*中断分类：根据事件的紧急程度分

可屏蔽中断—可暂不处理的中断，如键盘中断、打印机中断

不可屏蔽中断—须立即处理的中断，如MEM校验错、电源故障

*处理时机：不可屏蔽中断—当前指令周期结束时(便于实现)

可屏蔽中断—可为多个指令周期后

即两个指令周期之间，如$T_{指令}$=5/2GHz=2.5ns <<设备要求(ms级)

*可屏蔽的实现方法：

状态表示—状态REG中设置“中断允许”标志IF

操作实现—指令系统提供开中断(IF←1)、关中断指令

*返回方式：下条指令或终止程序(不可屏蔽中断才有)

※异常及中断的内涵：有事件类型、程序控制流改变2个方面

(不同系统的定义不同)

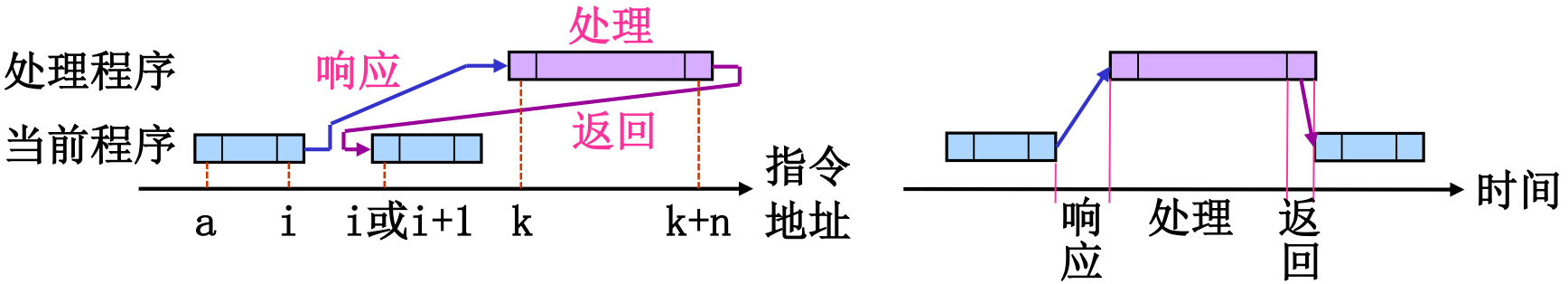
二、异常及中断的处理过程

*事件处理过程：

响应—从当前程序转到处理程序的过程

处理—执行处理程序的过程

返回—从处理程序返回到当前程序的过程



*术语：响应、处理、返回、异常处理程序、中断服务程序等

*异常/中断的处理需求：

①同时有多个事件(多种)，但同时只能处理1个

②不同事件的紧急程度、检测时机、响应操作不同

(源于处理时机)

1、异常及中断的响应

*任务：①保存断点及程序状态，②关中断，　←与所选事件无关

③识别事件类型并转入处理程序　←须选择事件

*保存断点及程序状态：　--硬件实现(软件无法在指令周期内处理)

目标—事件返回时，当前程序能继续执行

断点—事件的返回地址，便于返回的实现(直接恢复保存值)

实现—断点、程序状态(PSR)用后援寄存器保存，

异常类型用异常类型寄存器保存　←影响响应操作

(中断类型由中断源[外设]负责保存[在CPU外部])

*关中断：　--硬件实现

目标—响应过程不被新事件(指可屏蔽中断)打断

实现—使IF＝0，应先保存程序状态(不改变返回时的IF)

*识别事件类型并转入处理程序：　　--硬件实现(部分可软件实现)

子任务—识别事件类型(最紧急的)、

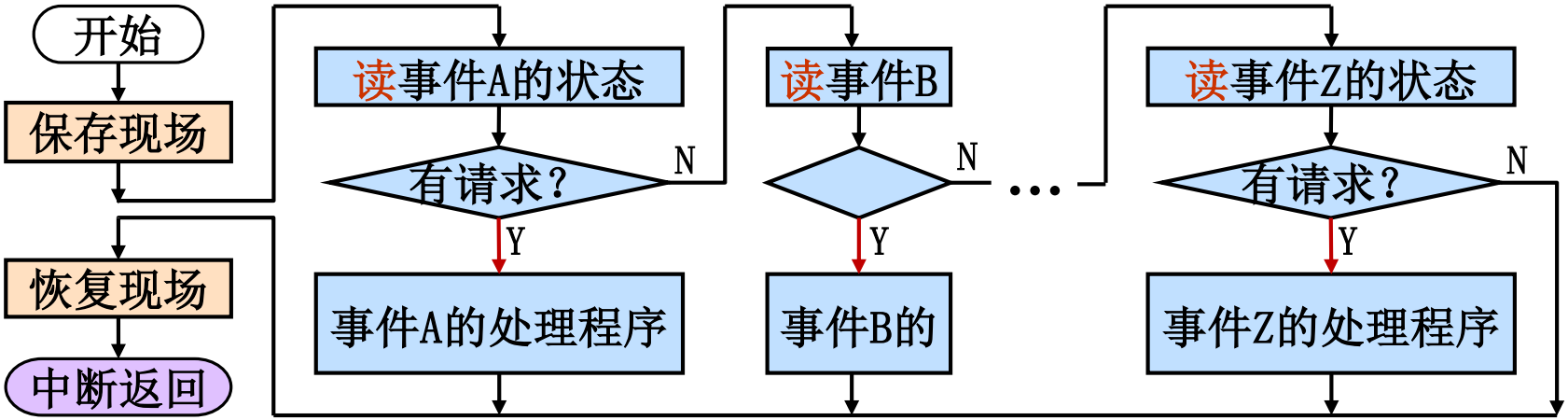
获取相应处理程序的入口地址、

PC←获取的入口地址

实现策略—向量方式、非向量方式

非向量方式—所有事件共用一个处理程序，入口地址固定

实现：PC←入口地址，其余由软件(处理程序)完成



事件紧急程度：由查找顺序决定

向量方式—每个事件有一个处理程序，

各入口地址保存在中断向量表IVT中

IVT：由OS管理，放在主存中(CPU中用REG保存首址)

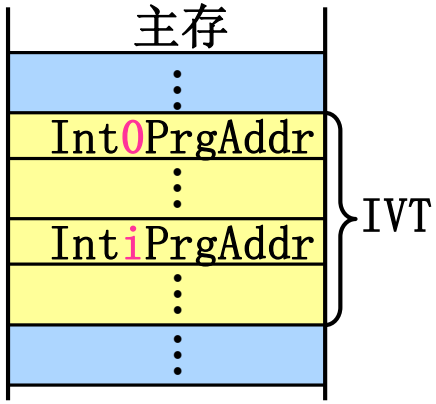
实现：判优、查表、PC←入口地址

判优逻辑：查找相关REG、选择事件(*i*)

(异常类型REG、中断源的状态REG)

查表逻辑：形成表项地址、访存

(首地址＋*i*×表项长度)



2、异常及中断的返回

*任务：恢复断点及程序状态

*实现：处理程序中用专用指令实现

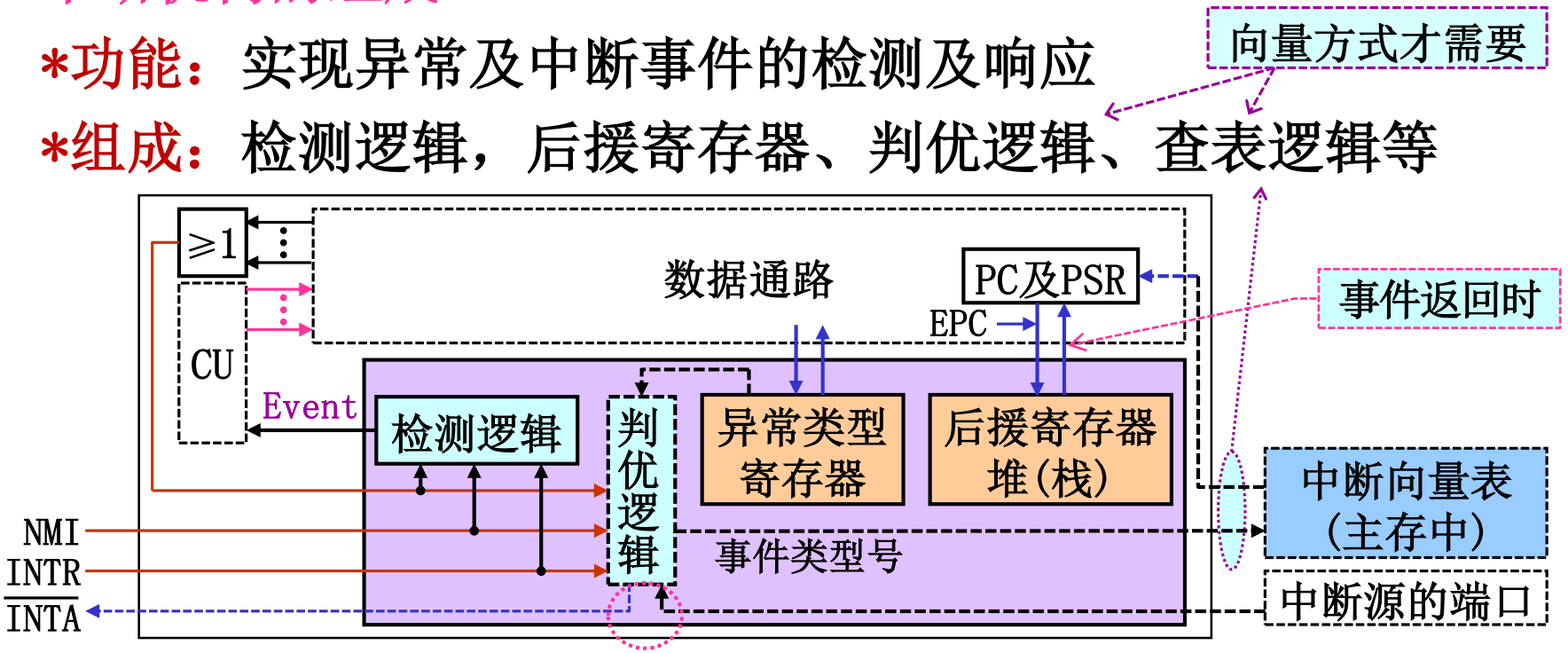
└→指令功能为 PC及PSR←(后援寄存器)

(硬件不知何时返回)

3、中断机构的组成

*功能：实现异常及中断事件的检测及响应

*组成：检测逻辑，后援寄存器、判优逻辑、查表逻辑等



检测逻辑—分类检测，以实现检测时机及响应方法控制

判优逻辑—统一判优、分类获取事件类型号，

中断类型号获取需进行外部操作(由$\overline{\text{INTA}}$控制)

其他逻辑—响应结束前撤销事件(清除相应REG)

三、支持异常处理的CPU设计

*设计背景：支持7条MIPS指令的数据通路及控制单元

支持溢出、非法操作码异常的处理

*MIPS的要求：响应采用非向量方式(入口地址为80000180H)，

事件类型由硬件获得(异常及状态REG中)

*异常检测的设计：异常逻辑＝OF＋ErrCode(非法操作码信号线)

*异常响应的设计：EPC (例外程序寄存器)，CAUSE(导致中断和异常的原因寄存器)

响应部件设置—用EPC保存断点，用Status保存程序状态，

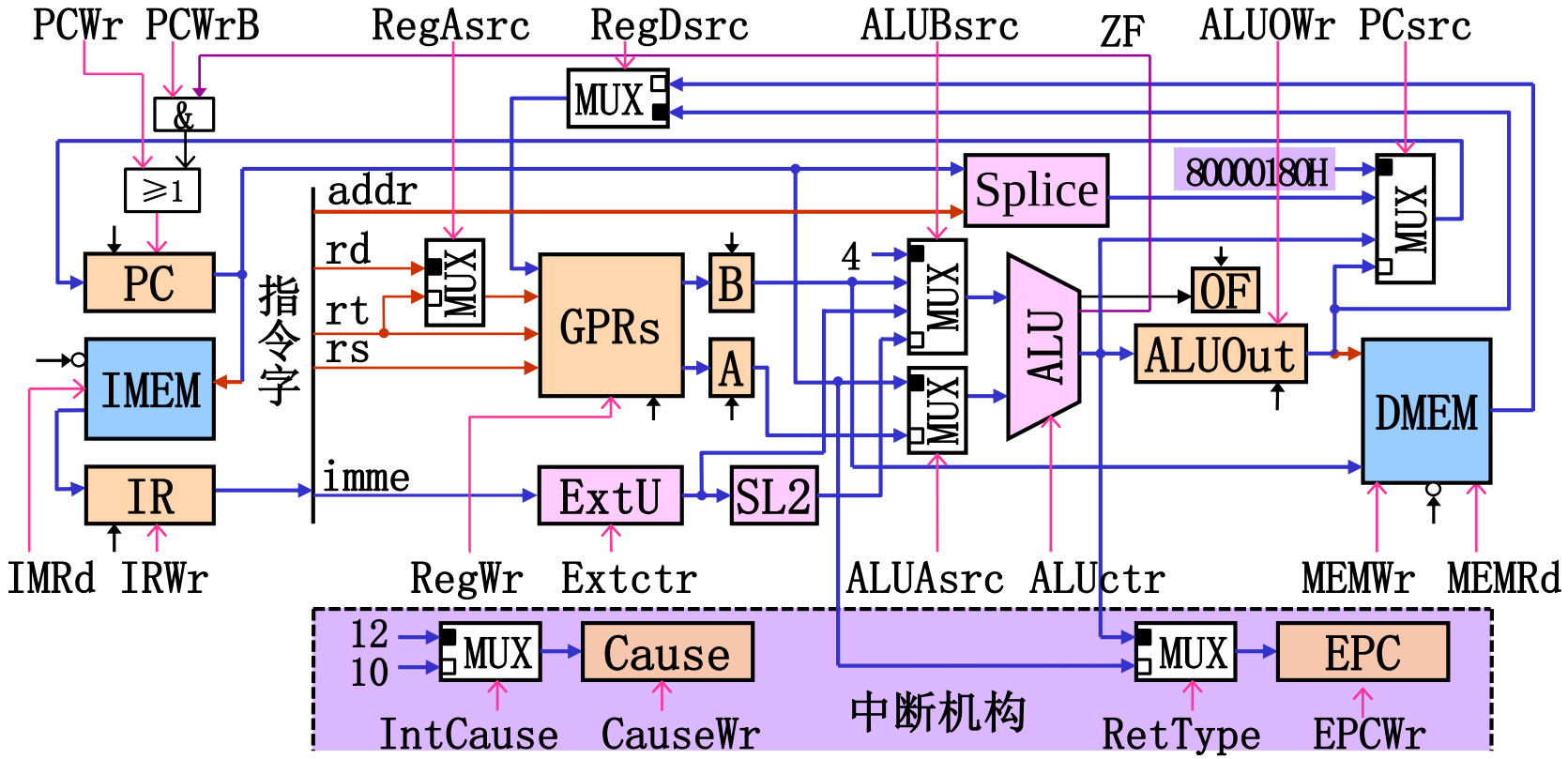
用Cause保存异常类型(分别为12、10)

响应的μOP需求—暂不考虑Status的操作

溢　　出：EPC←(PC)－4、Cause←12、PC←8000 0180H

非法操作码：EPC←(PC)－4、Cause←10、PC←8000 0180H

*数据通路的设计：增设EPC、Cause及相应路径(输出路径略)



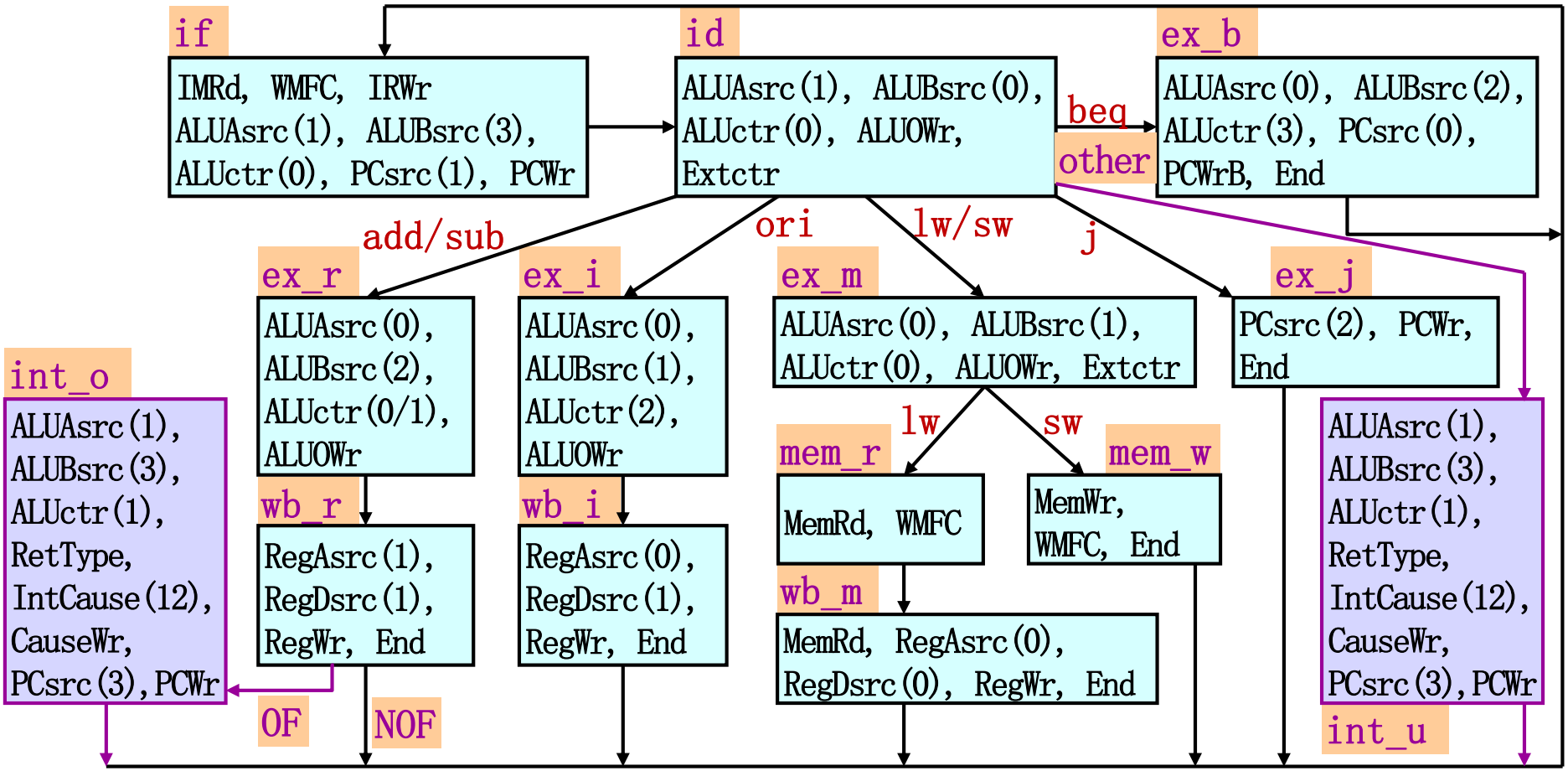
*异常响应过程的组织：都可在一个节拍内完成

响应μOPCmd—ALUAsrc=1、ALUBsrc=3、ALUctr=1、RetType=1，IntCause=12或10、CauseWr，PCsrc=3、PCWr

响应时间—立即处理，即指令周期中一检测到就响应

指令执行过程状态转换图—

在状态wb_r、id可分别检测到溢出、ErrCode异常



*CU的设计：增加一个时序信号(由Event触发产生)，其余类似

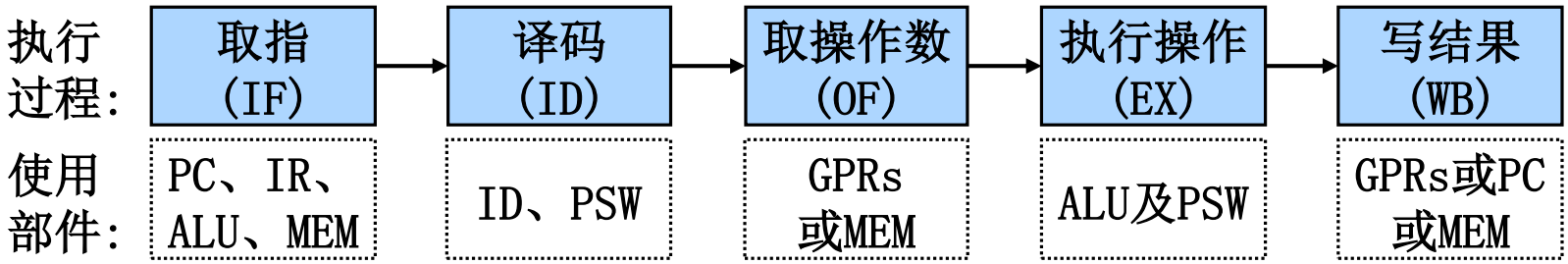
作业5-5：P238— 25

§6.7 指令流水线技术

*并行性：包括同时性、并发性

一、指令流水线概述

*指令执行过程的部件使用分析：



特征—部件很少重复使用(ALU、MEM)，重复使用的频率很低

└←不冲突的操作除外(GPRs/读写)

*指令执行性能的优化：

重叠指令执行过程，缩短指令周期的平均值

(如IF+ID、OF+EX+WB)

1、指令流水线的概念

*基本思想：指令执行过程分为多个阶段，←基础(同多周期)

每个阶段使用专门部件实现，←改进(段分离)

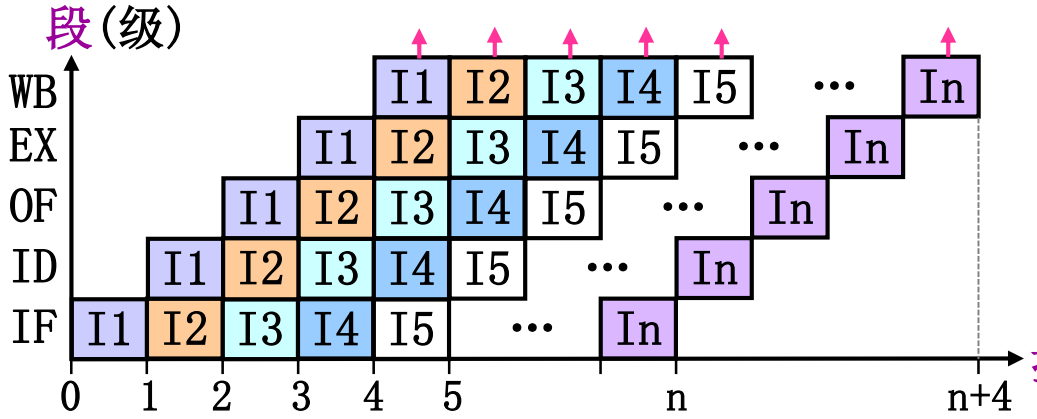
每条指令可依次通过各个阶段←效果(段并行)

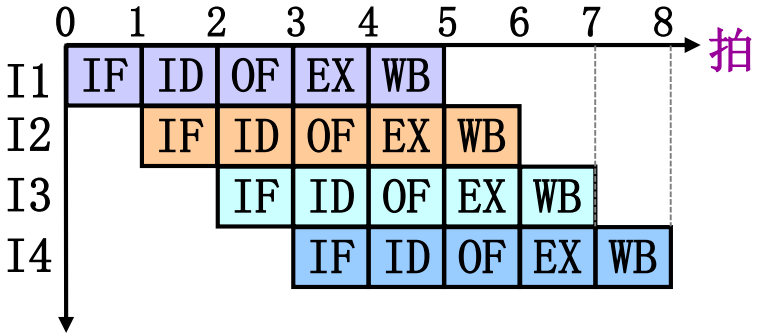
*基本组成：多个段(功能段)按序组成

取指(IF) → 译码(ID) → 取数(OF) → 执行(EX) → 写回(WB) ←适于CISC

取指(IF) → 译码(ID) → 执行(EX) → 访存(MEM) → 写回(WB) ←适于RISC

*工作原理：每条指令按序通过各段，不同指令执行过程重叠





程序执行时间—$T_{串行}=n\cdot(m\Delta t)$，$T_{流水}=m\Delta t+(n-1)\Delta t$

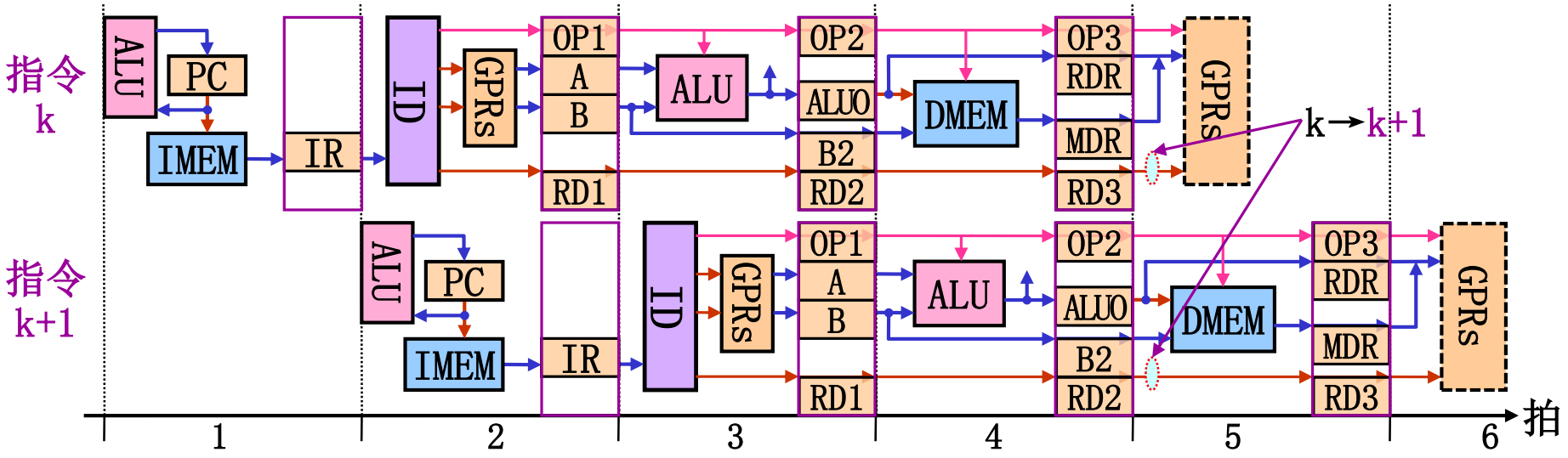
2、流水线组成的基本要求

(1) 各个段的操作相互独立　　←重叠的基础

要求：各个段的源数据来自时序部件、结果存到时序部件

实现：增设段间寄存器

└←依据：后续段需要使用的每个数据/地址/命令

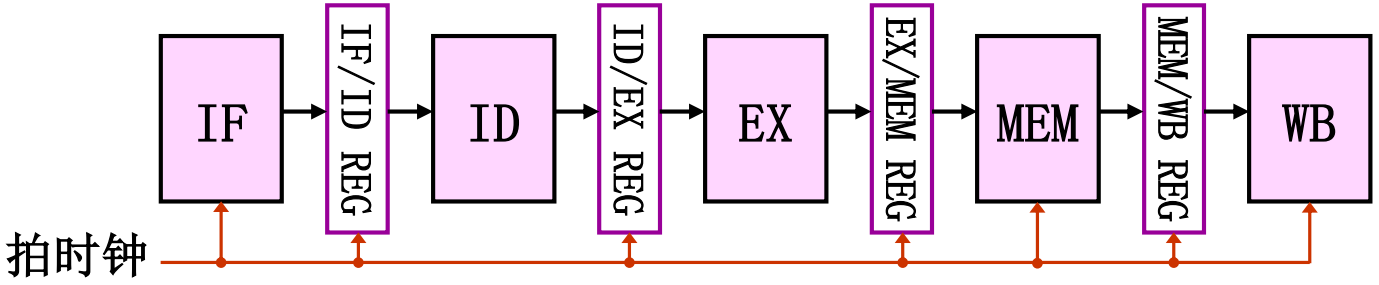


(2) 各个段的操作同步

←重叠的保证

要求：段间寄存器同时写入

实现：设置公共的时钟信号



拍长：拍长＝max{段*i*操作时间}，故各段时延尽量接近

(3) 各个段的操作无冲突

冲突：流水线因某些原因无法正确执行后续指令的现象，又称冒险(Hazard)

冒险的种类：结构冒险、数据冒险、控制冒险

例如— 部件复用　OPD源-目相关　分支指令

实现：增设部件及控制器，处理各种冒险(稍后讨论)

3、流水线的性能

假设一流水线有m段、拍长为Δt，共执行n条指令

*吞吐率：单位时间内完成/输出的指令条数或结果数量

时机吞吐率— $T_P=\dfrac{n}{T_{流水}}=\dfrac{n}{m\Delta t+(n-1)\Delta t}$

最大吞吐率— 当$n>>m$时，$T_{P\,max}=1/\Delta t$，即拍长的倒数

*加速比：流水方式相对于串行方式的速度比

$$S=\frac{串行方式执行时间}{流水方式执行时间}=\frac{n\cdot m\Delta t}{m\Delta t+(n-1)\Delta t}=\frac{nm}{m+n-1}$$

最大加速比— 当$n>>m$时，$S_{max}=m$，即流水线段数

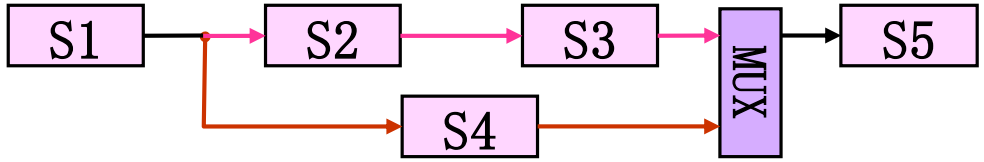
*效率：部件使用时间与整个执行时间的比值(平均值)

$$E=\frac{n个任务所占时空区}{n个任务总的时空区}=\frac{n\cdot m\Delta t}{m(m+n-1)\Delta t}=\frac{n}{m+n-1}=\frac{S_P}{m}=T_P\cdot\Delta t$$

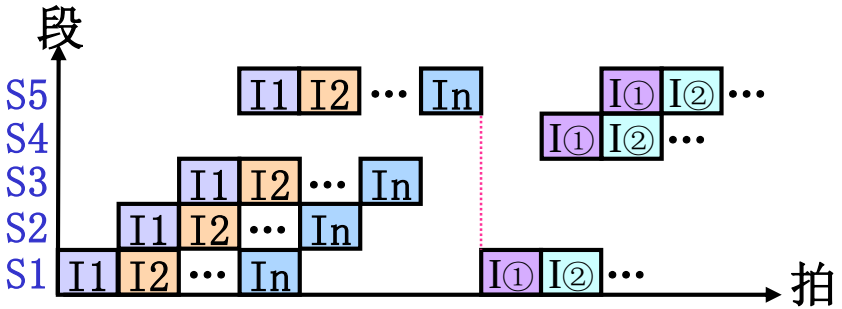
最高效率— 当$n>>m$时，$E_{max}=1$

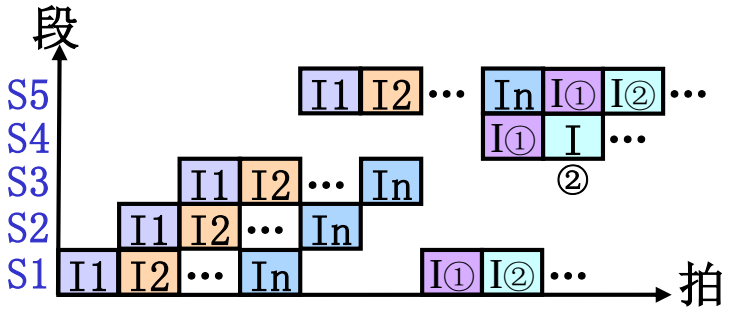
4、流水线的分类　　--即属性

*按功能分类：单功能流水线、多功能流水线

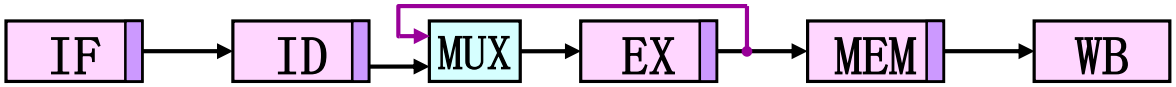


*按工作方式分类：静态流水线、动态流水线





*按结构分类：线性流水线、非线性流水线



*按流入/流出次序分类：顺序流水线、乱序流水线

*按处理的数据类型分类：标量流水线、向量流水线

二、指令流水线的冒险处理

1、结构冒险

由于争用硬件资源，引起流水线停顿的现象

如：MIPS通路中，冯·诺依曼结构、(PC)＋4实现、数据路径等

*处理策略：（2种）

①重复设置部件—性能好(线性流水线)，适于高频率冲突

②分时使用部件—成本低(非线性流水线)，适于低频率冲突

*重复设置部件策略的实现：

①增设部件—每个部件只能使用一次

如1：采用哈佛结构、增设Adder、互连采用点点结构

②固定部件使用时间—每个部件只在一个段使用

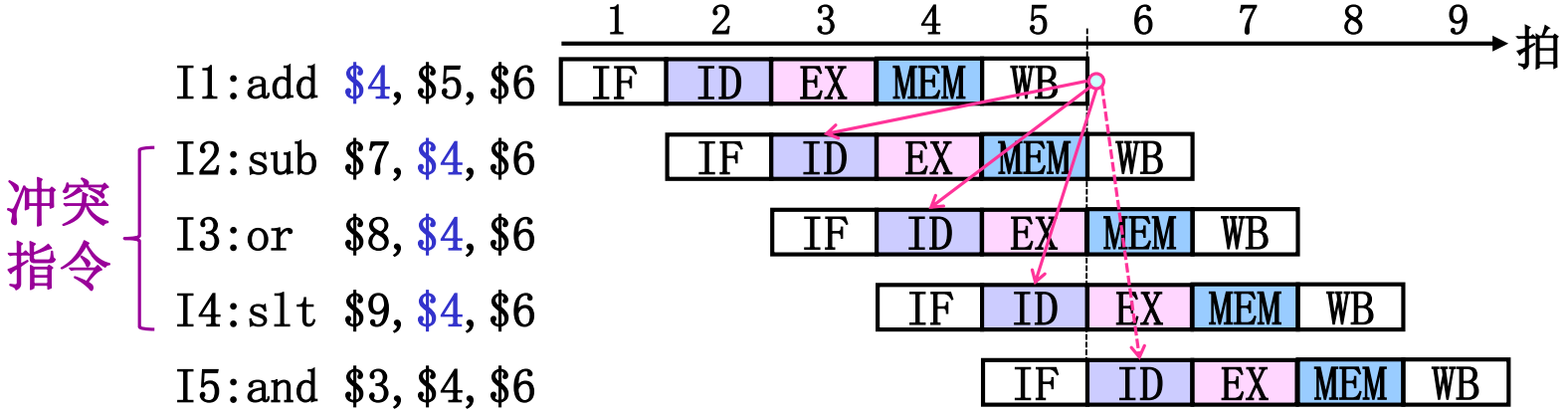
如2：只在WB段写GPRs，否则lw、add指令会冲突

*指令周期的长度：取决于最后一个操作所在段的位置

2、数据冒险

由于指令所需数据不可用，引起流水线停顿的现象

*冒险类型：写后读(Read After Write, RAW)冒险



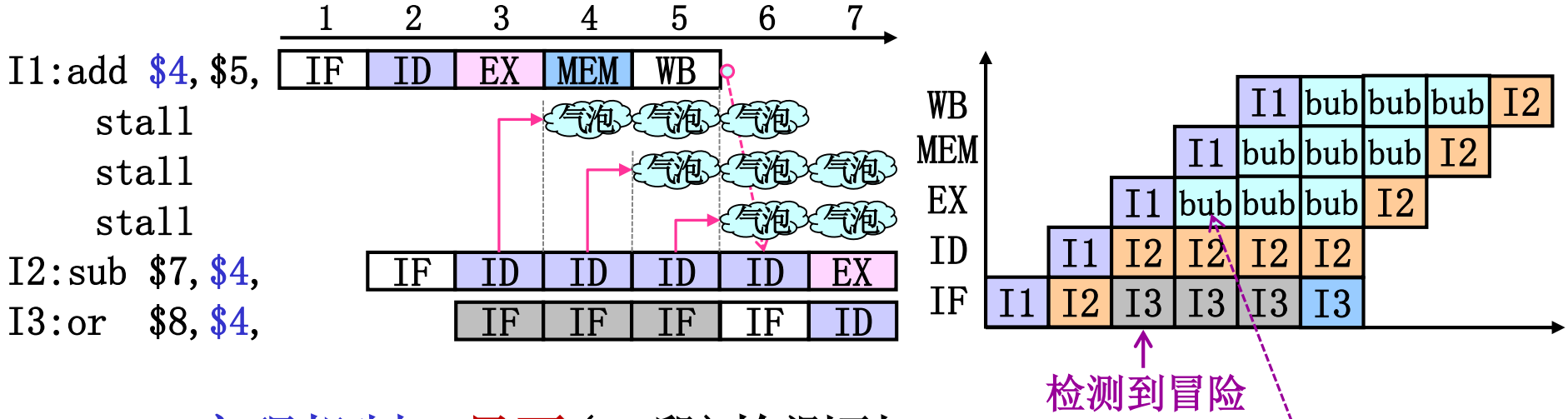
*处理方法：阻塞法、转发法、乱序执行法

(一起等) (抄近路取) (当事人等)

←小结

*阻塞法：使冲突指令及其后续指令停顿，直到RAW冒险消除

停顿方法—插入气泡(nop指令的译码输出)



实现机制—只要(ID段)检测到RAW，

就暂停IF段、使ID段产生气泡

封锁时钟信号→┘ └←写入ID/EX寄存器

停顿拍数—数据读-可读的间隔拍数

例1：MIPS流水线中，写GPRs放在WB段，在下一拍才能够读出所写的数据。现有如下MIPS指令序列：

```
I1: add $4, $5, $6      ; $4←$5＋$6
I2: sub $7, $4, $5      ; $7←$4－$5
I3: or  $8, $4, $7      ; $8←$4 | $7
I4: sw  $6, 20($4)      ; M[$4＋20]←$6
I5: lw  $9, 20($8)      ; $9←M[$8＋20]
```

问：①哪些指令之间存在RAW冒险？

②采用阻塞法处理RAW冒险，指令序列的执行时间为多少拍？

解：①RAW冒险有：I1-I2、I1-I3、I1-I4，I2-I3，I3-I5

②I1-I2冒险停3拍，I1-I3、I1-I4停0拍(随I1-I2自动消除)；

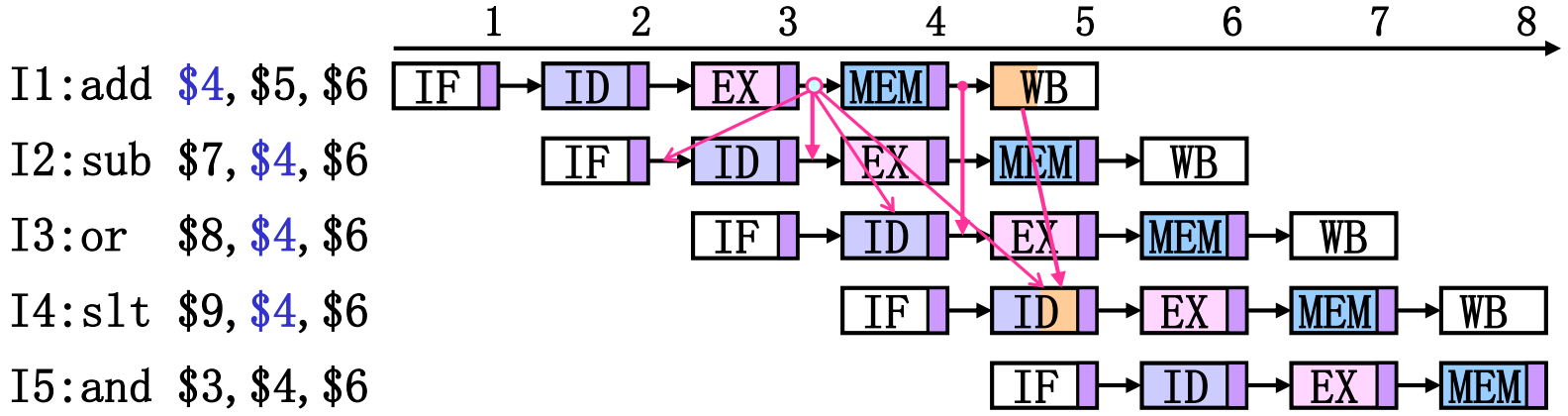
I2-I3冒险停3拍；I3-I5冒险停2拍；

指令序列执行时间＝[5Δ*t*＋(5－1)Δ*t*]＋(3＋3＋2)Δ*t*

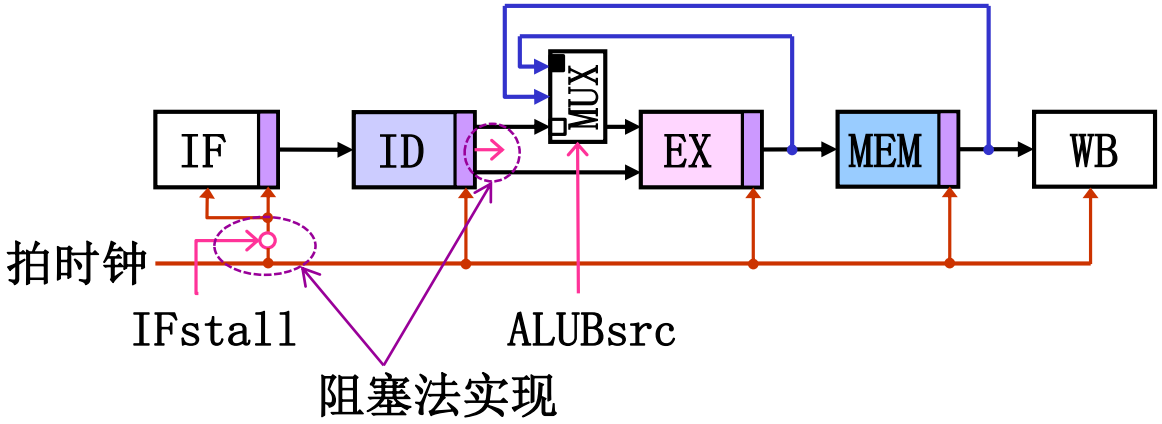
＝17Δ*t*

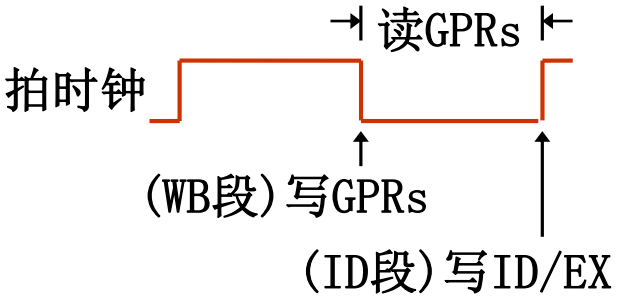
*转发法：使冲突指令可直接从数据产生段获取数据，来消除冒险

获取时机—可在使用时获取(等价于较早的段获取)



实现机制—增加转发线路、同一拍中提前写





停顿拍数—能转发时为0拍，否则为阻塞法停顿拍数

例2：续例1，采用转发法处理RAW冒险，写GPRs放在前半拍，①设置有EX→EX、MEM→EX的转发线路，②仅设置EX→EX的转发线路，指令序列的执行时间分别为多少拍？

解：RAW冒险有：I1-I2、I1-I3、I1-I4，I2-I3，I3-I5，

①前半拍写GPRs可消除I1-I4冒险，

EX→EX线路可消除I1-I2、I2-I3冒险，

MEM→EX线路可消除I1-I3、I3-I5冒险，

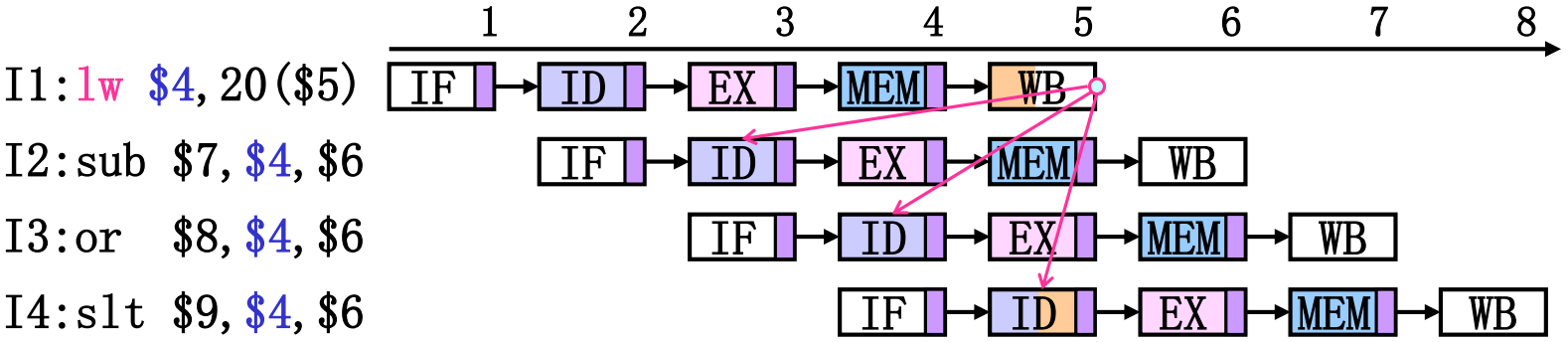
指令序列执行时间＝[5Δ*t*＋(5－1)Δ*t*]＋0Δ*t*＝9Δ*t*

②I1-I4、I1-I2、I2-I3冒险无需停顿，

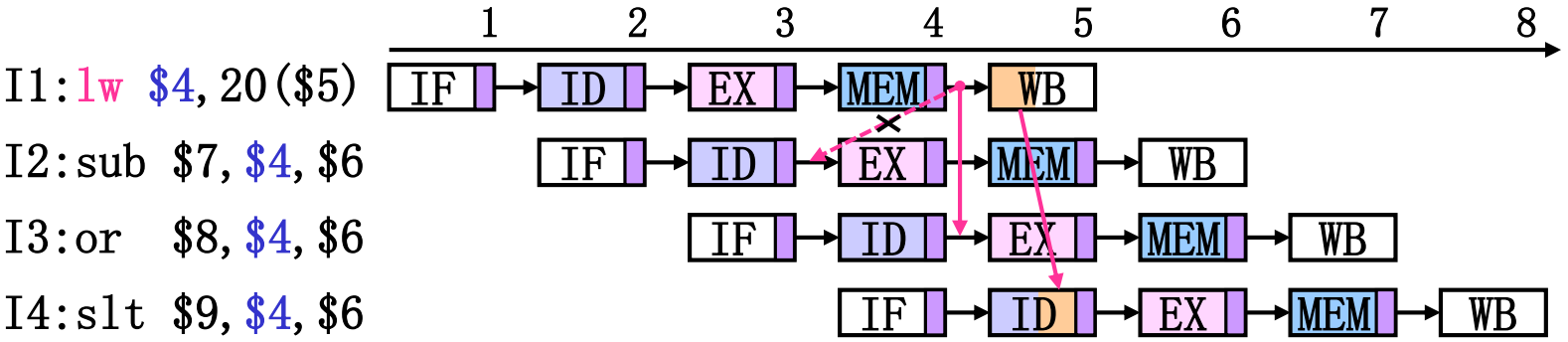
I1-I3、I3-I5冒险只能采用阻塞法处理，各停顿2拍

指令序列执行时间＝[5Δ*t*＋(5－1)Δ*t*]＋(2＋2)Δ*t*＝13Δ*t*

load-use冒险—由lw指令引起的、紧邻lw指令的RAW冒险



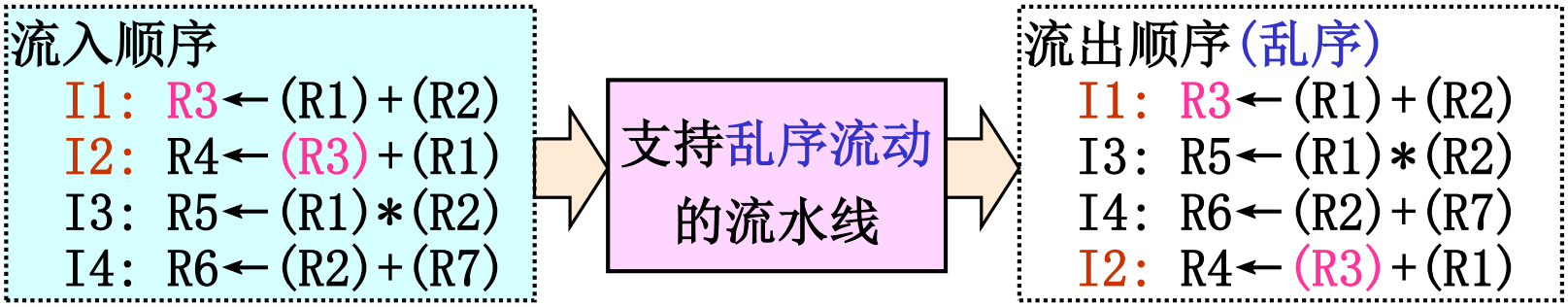
非紧邻lw指令的冒险处理：转发法



load-use的冒险处理：阻塞法，

或软件方法(插入nop指令)

*乱序执行法：只停顿冲突指令，后续无RAW冒险的指令可先执行



实现机制—增加指令窗口、采用动态调度方法

(OPD的就绪的才能正常流动)

停顿拍数—0拍

新增冒险类型—读后写(Write After Read, WAR)冒险、

写后写(Write After Write, WAW)冒险

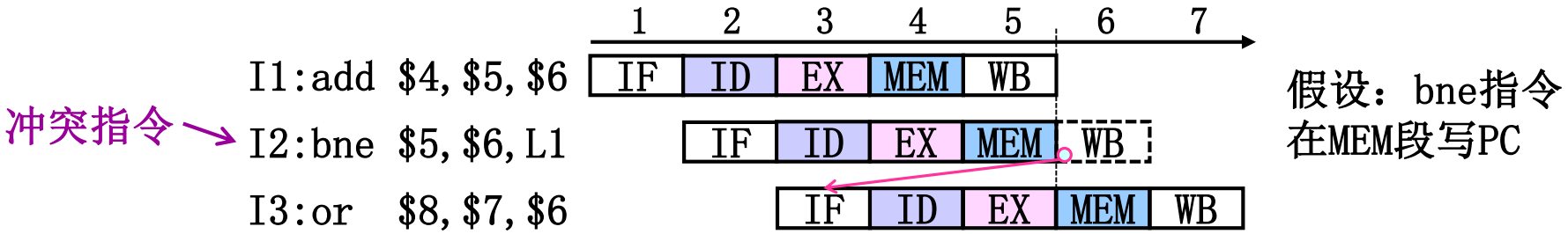
冒险处理方法：动态调度方法

进入顺序(示例)
I1: R3←(R1)+(R2)
I2: R4←(R3)+(R5)
I3: R5←(R1)*(R2)
I4: R4←(R2)+(R6)

3、控制冒险　　--分支冒险

由于指令执行顺序改变，引起流水线停顿的现象

└←下条指令地址≠(PC)＋1

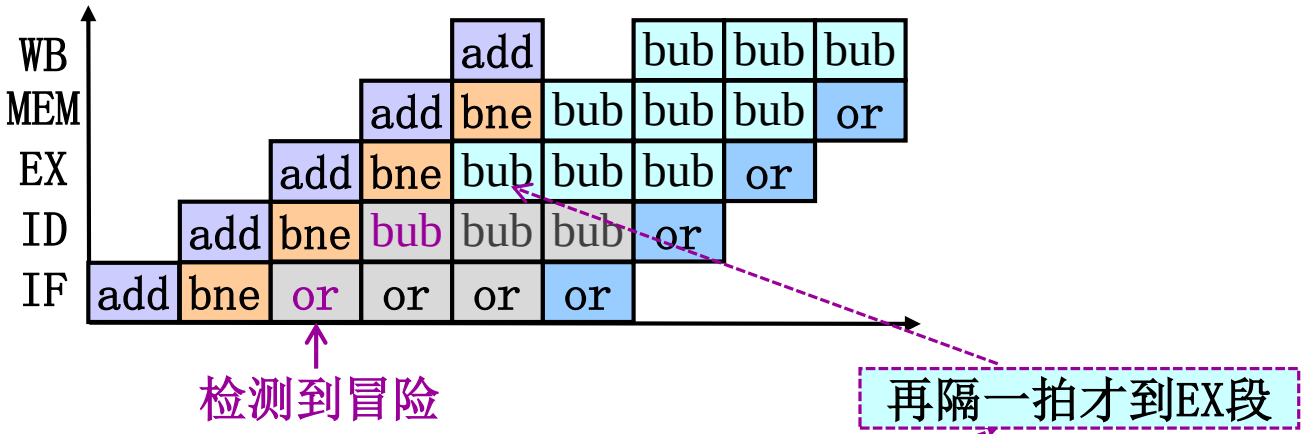


*处理方法：阻塞法、分支预测法、延迟分支法

*阻塞法：使分支指令之后的指令停顿，直到控制冒险消除

（RAW冒险中冲突指令也停顿）

停顿方法—插入气泡



实现机制—一旦(ID段)检测到控制冒险，

就立即暂停IF段、ID段下拍起每拍都产生气泡，直到冒险消除

（否则bne信息就被冲掉了）

停顿拍数—分支指令从ID段到PC可用的间隔拍数

性能优化方法—尽早判断是否转移，←利于不转移时

尽早计算分支目标地址←利于转移时

例3： MIPS流水线中，有EX段→EX段转发线路，bne指令在MEM段写PC。现有如下MIPS指令序列：

```
      addi $4, $5, 100        ; I1: $4←$5＋100
L1:   add  $8, $6, $7         ; I2: $8←$6＋$7
      sw   $8, 20($6)         ; I3: M[$6＋20]←$8
      addi $5, $5, 1          ; I4: $5←$5＋1
      bne  $5, $4, L1         ; I5: $5≠$4时PC←L1
      addi $9, $9, 10         ; I6: $9←$9＋10
```

问：①哪些指令之间存在RAW冒险？

②采用阻塞法处理控制冒险，指令序列的执行时间为多少拍？

解： ①RAW冒险有：I2-I3、I4-I5 （I1-I5不存在冒险）

②I2-I3、I4-I5冒险可用转发法处理，停0拍；

控制冒险用阻塞法处理时，I5每次时流水线停3拍；

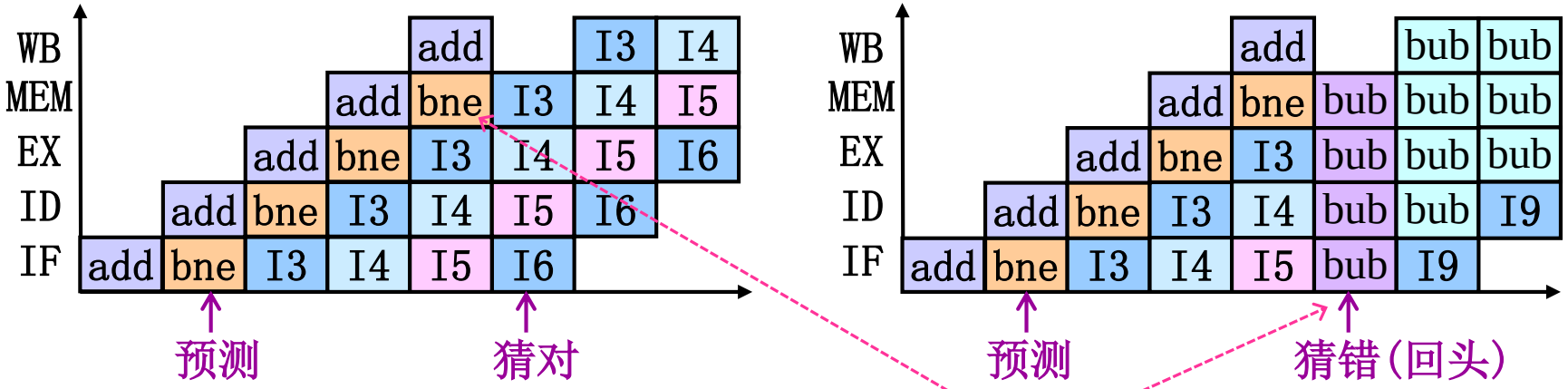
指令序列执行时间＝[5Δt＋(402－1)Δt]＋3Δt ×100

＝706Δt

*分支预测法：预测转移方向，并执行该方向的指令，

猜对时继续执行后续指令，

猜错时回头执行另一方向上的指令



停顿拍数—猜对时≥0拍(IF/ID)，猜错时＝阻塞法＋1拍

实现机制—IF段或ID段预测，猜对时不写PC，

猜错时清空流水线(分支指令已实现写PC)

预测方法—静态预测、动态预测(根据该指令的转移历史)

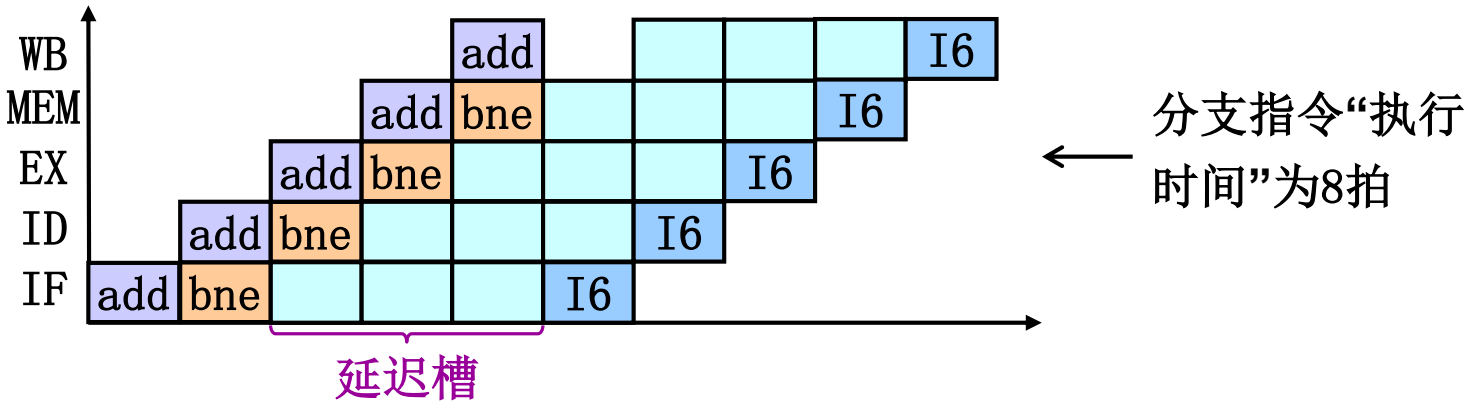
└→所需硬件：BTB、更新逻辑

运用：动态预测＋静态预测(首次执行时)

*延迟分支法：延迟槽中的指令总是被执行

（逻辑上延长分支指令的执行时间）

延迟槽—分支指令执行完前，可流入流水线的指令位置



停顿拍数—延迟槽中指令全为nop指令时，＝阻塞法；

延迟槽中指令含有用指令时，＜阻塞法

实现机制—软件实现（编译时重排序指令序列）

└←分支指令前无相关性指令移入延迟槽

使用场合—延迟槽大小＝1条指令时，

否则常用分支预测方法（两者不兼容）

※指令流水线技术小结：

基本思想—各操作串行---→各操作重叠(操作的是不同指令)

基本组成—

要求：操作分离、操作同步、操作无冲突

实现：段间REG，公共拍时钟，部件＋控制器

操作冲突的处理—

类型：结构冒险、数据冒险、控制冒险

处理：①部件不复用，同一部件在同一拍使用

②阻塞法、转发法、乱序执行法

③阻塞法、分支预测法、延迟分支法

作业5-5：P238— 27、29

三、指令流水线的并行技术

*指令级并行性表示：IPC(Instructions Per Cycle)，IPC×CPI ＝1

*超级流水线技术：增加流水线级数(段数) ←CPI＝1

└→缩短T_C(执行过程相同)

发展过程—级数不宜太多(486→PⅢ→P4→Core)

*多发射流水线技术：同时流动(执行)多条指令 ←CPI＜1

实现—增加指令打包、冒险处理环节

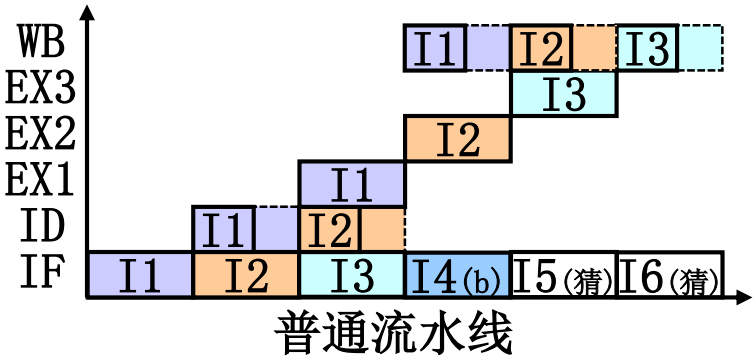
用推测技术优化→┘ └→先后及同时产生的

类型—超标量、超长指令字VLIW

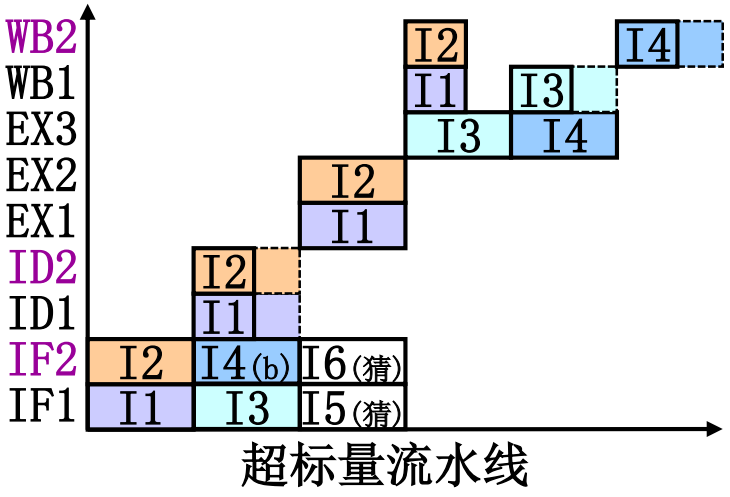
(指令数可变) (指令数固定)

*超标量流水线：硬件完成指令打包、冒险处理

关键技术—动态调度、分支预测、推测执行 ←增加IPC



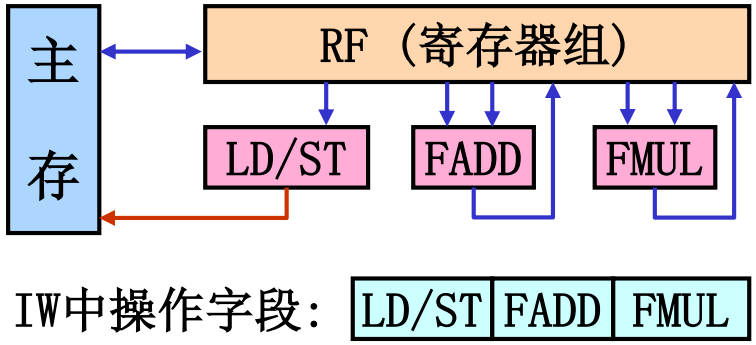
普通流水线

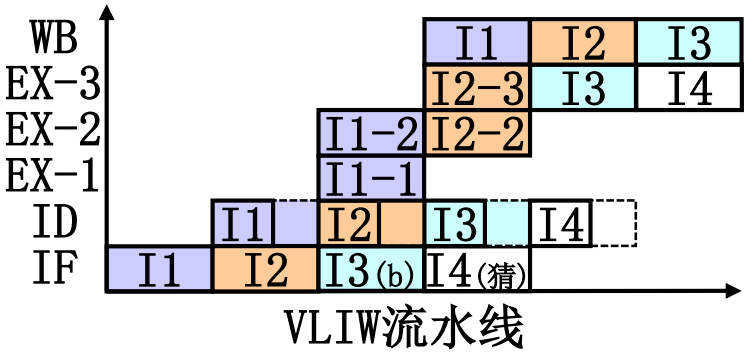


超标量流水线

*VLIW流水线：软件完成指令打包、冒险处理

关键技术—静态调度(重排序＋阻塞)、延迟分支 ←增加IPC





VLIW流水线