
西安电子科技大学
人工智能学院

# 计算机组成与体系结构

## 第6章 中央处理器(CPU)

# 本章第2次课重点

- 控制器结构
- 硬布线控制器设计
- **微操作序列→微命令序列**

# 控制器结构

# 6.1.5 控制器的组成

## 控制器应完成的任务：

- 产生微命令（即控制信号）。
- 按节拍产生微命令。

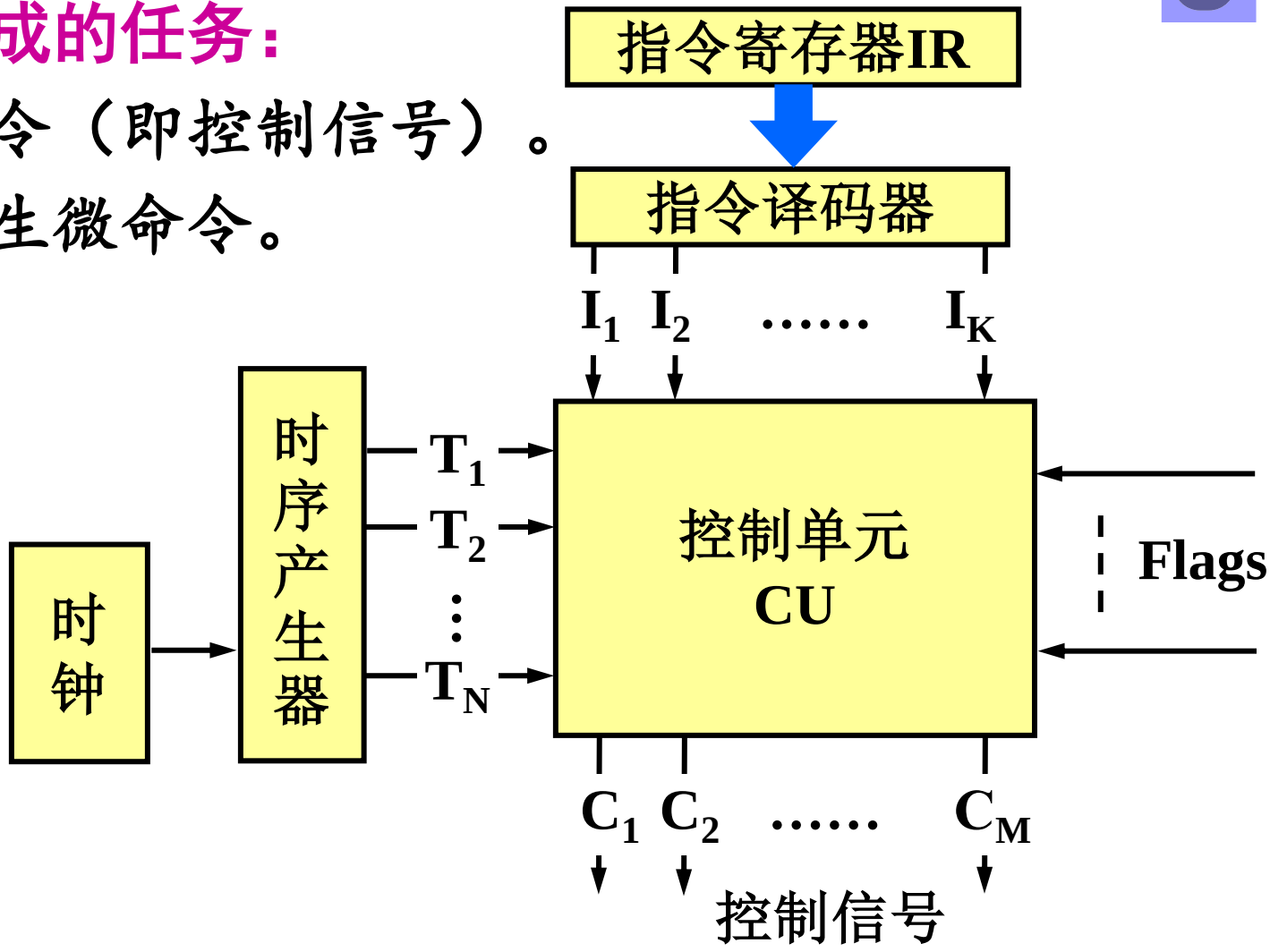



图6.6 控制单元模型

# 6.1.5 控制器的组成

- 设计者在设计控制器之前需要做以下工作：
  - ✓ 定义计算机基本硬件组成和基本指令系统；
  - ✓ 基于定义的硬件结构，针对每条指令，描述CPU完成的微操作；
  - ✓ 确定控制单元应该完成的功能，即何时产生何种微命令。
- 两种设计控制器的通用方法：
  - ✓ 硬布线控制（hardwired control）设计法
  - ✓ 微程序控制（microprogrammed control）或微码控制（microcoded control）设计法

# 硬布线控制器设计

# 6.2 硬布线控制器设计

- 硬布线控制器设计法将控制单元看作一个顺序逻辑电路（sequential logic circuit）或有限状态机（finite-state machine），它可以产生规定顺序的控制信号，这些信号与提供给控制单元的指令相对应。
- 它的设计目标是：使用最少的元器件，达到最快的操作速度。
- 设计示例
  - ✓ RISC-V系统控制单元设计（单周期实现）
  - ✓ 类x86系统控制单元设计（多周期实现）

图6.6

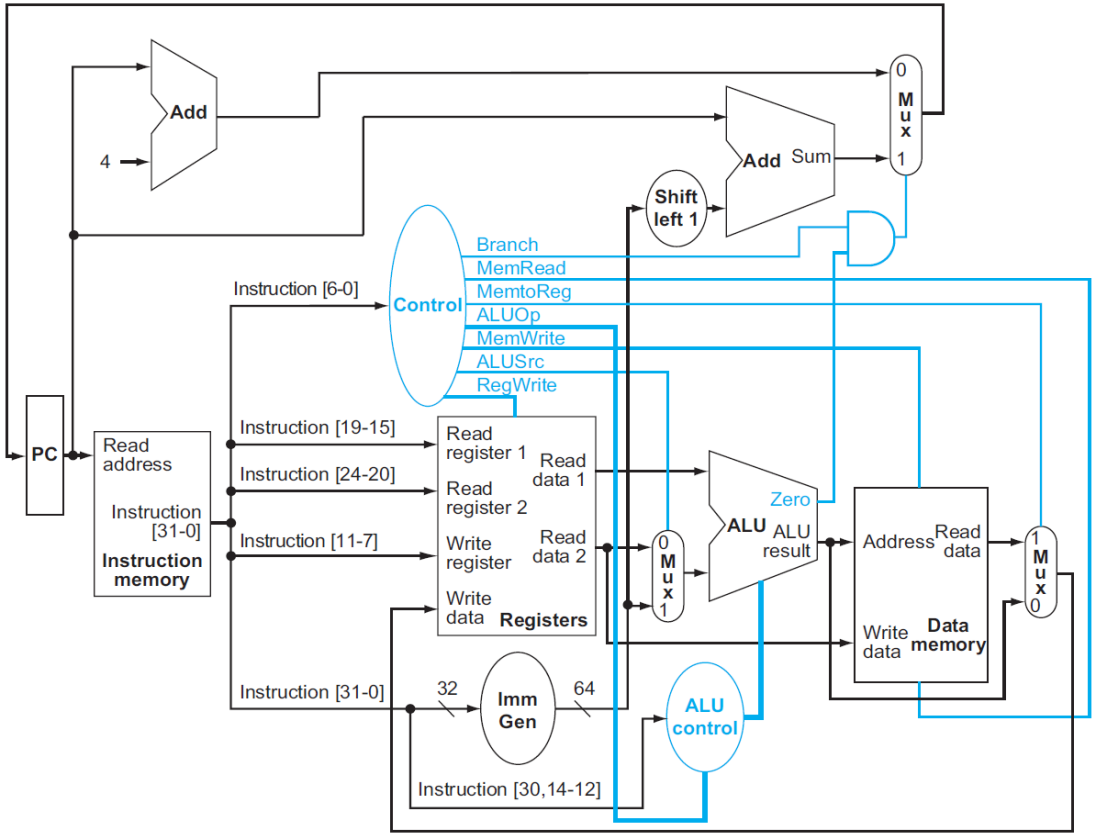
# RISC-V系统控制单元设计

# 6.2.1 RISC-V系统控制单元设计

- 图中控制单元（Control）的输入是指令操作码I[6：0]，输出是表中支持7条指令执行的控制信号。实现对访存指令ld和sd、算术逻辑指令add、sub、and和or、条件分支指令beq的支持。
- 表上半部是与四个指令类（覆盖7条指令）对应的输入信号I[6:0]编码，每列一个指令类，它们决定了输出控制信号是否有效。表下半部给出了与四种操作码对应的输出控制信号。

表6.2 控制单元的单周期实现逻辑真值表

| 输入或输出 | 信号名称 | R型 | ld | sd | beq |
|---|---|---|---|---|---|
| 输入 | I[6] | 0 | 0 | 0 | 1 |
| | I[5] | 1 | 0 | 1 | 1 |
| | I[4] | 1 | 0 | 0 | 0 |
| | I[3] | 0 | 0 | 0 | 0 |
| | I[2] | 0 | 0 | 0 | 0 |
| | I[1] | 1 | 1 | 1 | 1 |
| | I[0] | 1 | 1 | 1 | 1 |
| 输出 | ALUSrc | 0 | 1 | 1 | 0 |
| | MemtoReg | 0 | 1 | x | x |
| | RegWrite | 1 | 1 | 0 | 0 |
| | MemRead | 0 | 1 | 0 | 0 |
| | MemWrite | 0 | 0 | 1 | 0 |
| | Branch | 0 | 0 | 0 | 1 |
| | ALUOp1 | 1 | 0 | 0 | 0 |
| | ALUOp0 | 0 | 0 | 0 | 1 |



# 6.2.1 RISC-V系统控制单元设计

- 分析真值表可以看出，每个控制信号是由一个与或逻辑确定的。图6.7是控制单元的硬布线实现，采用的是可编程逻辑阵列PLA（由一个与门阵列之后紧跟一个或门阵列构成）。
- 如果使用了128个可能的操作码中的大多数，并且需要产生更多的控制信号，那么图中门的数目就会大得多，每个门也会有更多的输入。
- 所有控制信号在单周期内产生。

表6.2 控制单元的单周期实现逻辑真值表

| 输入或输出 | 信号名称 | R型 | ld | sd | beq |
|---|---|---|---|---|---|
| 输入 | I[6] | 0 | 0 | 0 | 1 |
| | I[5] | 1 | 0 | 1 | 1 |
| | I[4] | 1 | 0 | 0 | 0 |
| | I[3] | 0 | 0 | 0 | 0 |
| | I[2] | 0 | 0 | 0 | 0 |
| | I[1] | 1 | 1 | 1 | 1 |
| | I[0] | 1 | 1 | 1 | 1 |
| 输出 | ALUSrc | 0 | 1 | 1 | 0 |
| | MemtoReg | 0 | 1 | x | x |
| | RegWrite | 1 | 1 | 0 | 0 |
| | MemRead | 0 | 1 | 0 | 0 |
| | MemWrite | 0 | 0 | 1 | 0 |
| | Branch | 0 | 0 | 0 | 1 |
| | ALUOp1 | 1 | 0 | 0 | 0 |
| | ALUOp0 | 0 | 0 | 0 | 1 |

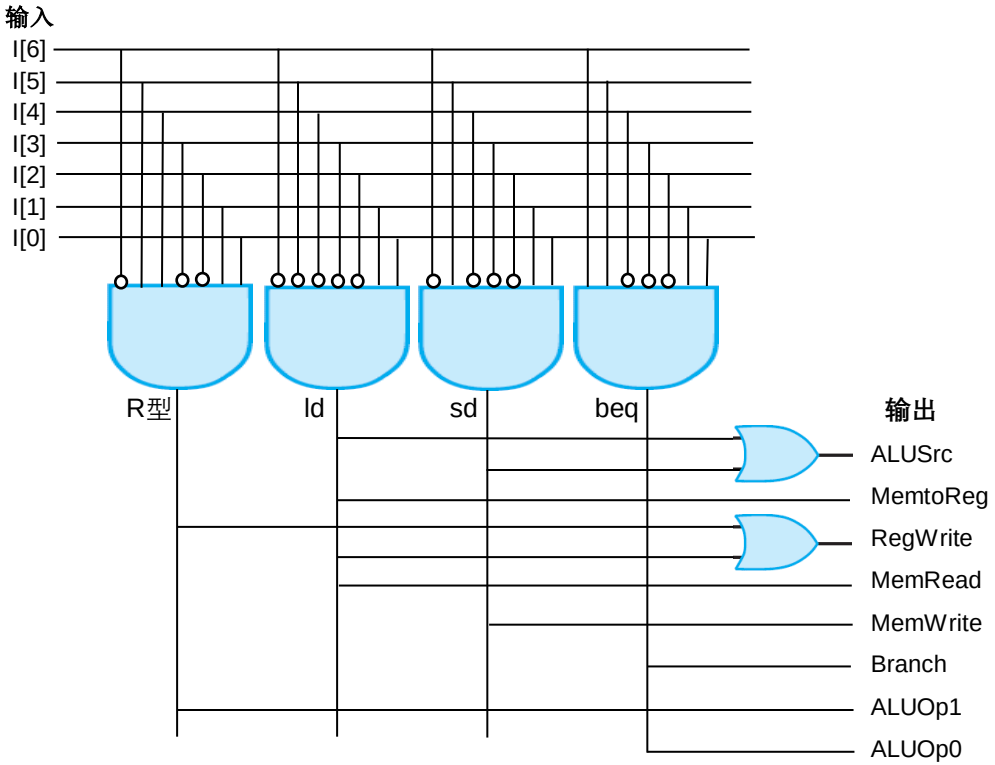



图6.7 控制单元的结构化实现

# 类x86系统控制单元设计

# 6.2.2 类x86系统控制单元设计

- 针对图6.3和图6.4结构确定的系统，其硬布线控制器可采用图6.6结构
- 所有控制信号在多周期内产生。

图6.6

- 两种设计方法：
  - ✓ 采用一级时序，即只利用节拍信号
  - ✓ 采用两级时序，即利用节拍和CPU周期两种时间信号

# 6.2.2 类x86系统控制单元设计

方法1：一级时序

图6.3

实现指令SUB R0,（X）功能的微操作序列：

T1: AR←PC ;取指令阶段

T2: DR←Memory[AR]，Mread

PC←PC+I

T3: IR←DR

T4: AR←IR（地址字段） ;执行指令阶段

T5: DR←Memory[AR]，Mread

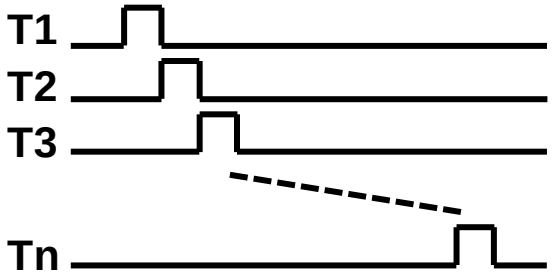
T6: AR←DR

T7: DR←Memory[AR]，Mread

T8: Y←R0

T9: Z←Y - DR

T10: R0←Z




一级时序

# 6.2.2 类x86系统控制单元设计

方法2：两级时序

图6.3

实现指令SUB R0,（X）功能的微操作序列：

M1：　　;取指CPU周期

T1：AR←PC

T2：DR←Memory[AR]，Mread

PC←PC+I

T3：IR←DR

方法1

采用两个CPU周期

M2：　　;执行CPU周期

T1：AR←IR(地址字段)

T2：DR←Memory[AR]，Mread

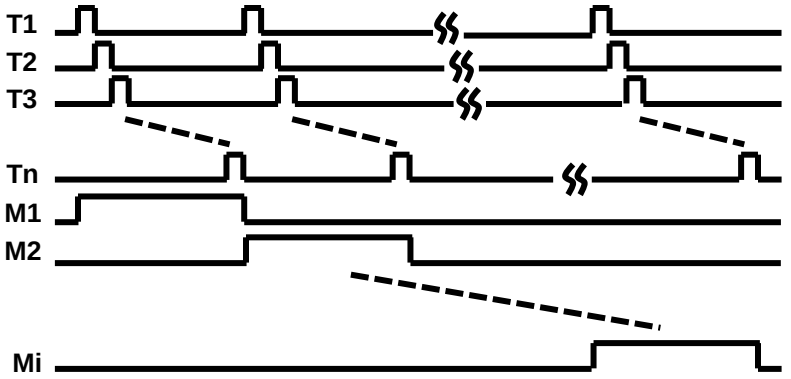
T3：AR←DR

T4：DR←Memory[AR]，Mread

T5：Y←R0

T6：Z←Y - DR

T7：R0←Z




二级时序

# 6.2.2 类x86系统控制单元设计

方法2：两级时序

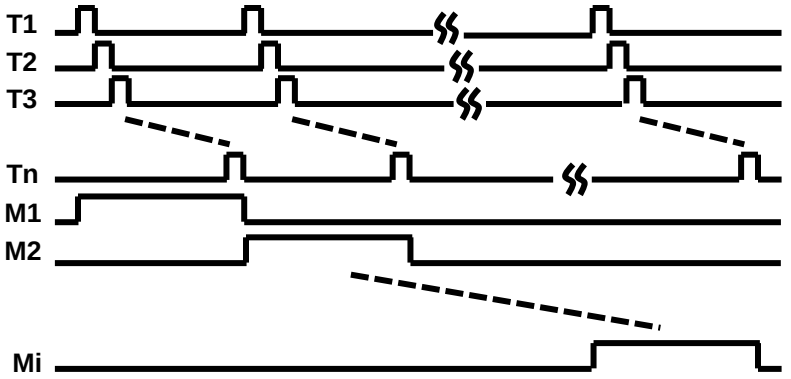
图6.3

实现指令SUB R0,（X）功能的微操作序列：

M1：　　;取指CPU周期
　T1：AR←PC
　T2：DR←Memory[AR]，Mread
　　PC←PC+I
　T3：IR←DR
M2：　　;取数CPU周期
　T1：AR←IR（地址字段）
　T2：DR←Memory[AR]，Mread
　T3：AR←DR
　T4：DR←Memory[AR]，Mread
M3：　　;执行CPU周期
　T1：Y←R0
　T2：Z←Y - DR
　T3：R0←Z

方法2

采用三个CPU周期




二级时序

# 6.2.2 类x86系统控制单元设计

【随堂练习】两级时序实现指令SUB R0,(R1)功能的微操作序列

M1: ;取指CPU周期

T1: AR←PC

T2: DR←Memory[AR], Mread

PC←PC+I

T3: IR←DR

M2: ;执行CPU周期

T1: AR←R1

T2: DR←Memory[AR], Mread

T3: Y←R0

T4: Z←Y - DR

T5: R0←Z

# 微命令序列

# 6.2.2 类x86系统控制单元设计

- 微命令是微操作实现所需的控制信号
- 微命令序列是何时产生何种控制信号的描述
- 微操作序列 ⟺ 微命令序列
- 微命令序列是控制器设计的依据

# 6.2.2 类x86系统控制单元设计

针对图6.3和图6.4设计的

图6.3

- 控制信号：

$PC_{in}$为程序计数器的锁存输入控制信号；

$PC_{out}$为程序计数器的输出允许控制信号；

PC+1为程序计数器的自动增量（如自动加1）控制信号；

$IR_{in}$为指令寄存器的锁存输入控制信号；

$IR_{out}$为指令寄存器的输出允许控制信号；

$SP_{in}$为指令寄存器的锁存输入控制信号；

$SP_{out}$为指令寄存器的输出允许控制信号；

SP+1为堆栈指示器的自动增量（如自动加n）控制信号；

SP-1为堆栈指示器的自动减量（如自动减n）控制信号；

$Ri_{in}$为通用寄存器Ri（0≤i≤n-1）的锁存输入控制信号；

$Ri_{out}$为通用寄存器Ri（0≤i≤n-1）的输出允许控制信号；

# 6.2.2 类x86系统控制单元设计

图6.3

- 控制信号：

$Y_{in}$为暂存器Y的锁存输入控制信号；
$Z_{out}$为暂存器Z的输出允许控制信号；
$AR_{in}$为地址寄存器向CPU内部总线的锁存输入控制信号；
$AR_{out}$为地址寄存器面向系统总线的输出允许控制信号；
$DRI_{in}$为双端口数据寄存器面向CPU内部总线的锁存输入控制信号；
$DRI_{out}$为双端口数据寄存器面向CPU内部总线的输出允许控制信号；
$DRS_{in}$为双端口数据寄存器面向系统总线的锁存输入控制信号；
$DRS_{out}$为双端口数据寄存器面向系统总线的输出允许控制信号；

Mread为从主存储器读出信息的读控制信号；
Mwrite为将信息写入到主存储器的写控制信号；
IOread为从I/O设备输入信息的读控制信号；
IOwrite为将信息写入到I/O设备的写控制信号；

# 6.2.2 类x86系统控制单元设计

图6.3

- 控制信号：

  ADD为加载至ALU的加法运算控制信号；
  SUB为加载至ALU的减法运算控制信号；
  AND为加载至ALU的逻辑与运算控制信号；
  OR为加载至ALU的逻辑或运算控制信号；
  SHL为加载至ALU的逻辑左移控制信号；
  SHR为加载至ALU的逻辑右移控制信号；
  ROL为加载至ALU的循环左移控制信号；
  ROR为加载至ALU的循环右移控制信号；
  ……

# 6.2.2 类x86系统控制单元设计

- 公操作取指周期

| 节拍 | 微操作序列 | 微命令序列 |
|---|---|---|
| T1 | AR←PC | $PC_{out}$，$AR_{in}$ |
| T2 | DR←Memory[AR] | $AR_{out}$，Mread，$DRS_{in}$ |
| T3 | PC←PC+I，IR←DR | PC+1，$DRI_{out}$，$IR_{in}$ |

- 指令执行周期举例：
  - ① MOV R0, X
  - ② MOV (R1), R0
  - ③ ADD R1, R0
  - ④ SUB R0, (X)
  - ⑤ IN R0, P
  - ⑥ JZ offs
  - ⑦ POP R0
  - ⑧ CALL (X)

# 6.2.2 类x86系统控制单元设计

➢指令执行周期举例：

（1）MOV R0, X

| 节拍 | 微操作序列 | 微命令序列 |
|---|---|---|
| T1 | AR←IR（地址字段） | $IR_{out}$，$AR_{in}$ |
| T2 | DR←Memory[AR] | $AR_{out}$，Mread，$DRS_{in}$ |
| T3 | R0←DR | $DRI_{out}$，$R0_{in}$ |

（2）MOV (R1), R0

| 节拍 | 微操作序列 | 微命令序列 |
|---|---|---|
| T1 | AR←R1 | $R1_{out}$，$AR_{in}$ |
| T2 | DR←R0 | $R0_{out}$，$DRI_{in}$ |
| T3 | Memory[AR]←DR | $AR_{out}$，$DRS_{out}$，Mwrite |

# 6.2.2 类x86系统控制单元设计

- 指令执行周期举例：

（3）ADD R1, R0

| 节拍 | 微操作序列 | 微命令序列 |
|---|---|---|
| T1 | Y←R0 | $R0_{out}$，$Y_{in}$ |
| T2 | Z←R1+Y | $R1_{out}$，ADD |
| T3 | R1←Z | $Z_{out}$，$R1_{in}$ |

（4）SUB R0, (X)

| 节拍 | 微操作序列 | 微命令序列 |
|---|---|---|
| T1 | AR←IR（地址字段） | $IR_{out}$，$AR_{in}$ |
| T2 | DR←Memory[AR] | $AR_{out}$，Mread，$DRS_{in}$ |
| T3 | AR←DR | $DRI_{out}$，$AR_{in}$ |
| T4 | DR←Memory[AR] | $AR_{out}$，Mread，$DRS_{in}$ |
| T5 | Y←R0 | $R0_{out}$，$Y_{in}$ |
| T6 | Z←Y－DR | $DRI_{out}$，SUB |
| T7 | R0←Z | $Z_{out}$，$R0_{in}$ |

# 6.2.2 类x86系统控制单元设计

- 指令执行周期举例：

（5）IN R0, P

| 节拍 | 微操作序列 | 微命令序列 |
|---|---|---|
| T1 | AR←IR（地址字段） | $IR_{out}$，$AR_{in}$ |
| T2 | DR←IO[AR] | $AR_{out}$，IOread，$DRS_{in}$ |
| T3 | R0←DR | $DRI_{out}$，$R0_{in}$ |

（6）JZ offs

| 节拍 | 微操作序列 | 微命令序列 |
|---|---|---|
| | ZF=1： | |
| T1 | Y←IR（地址字段） | $IR_{out}$，$Y_{in}$ |
| T2 | Z←PC+Y | $PC_{out}$，ADD |
| T3 | PC←Z | $Z_{out}$，$PC_{in}$ |

# 6.2.2 类x86系统控制单元设计

➢指令执行周期举例：

（7）POP R0

| 节拍 | 微操作序列 | 微命令序列 |
|---|---|---|
| T1 | AR←SP | $SP_{out}$，$AR_{in}$ |
| T2 | DR←Memory[AR] | $AR_{out}$，Mread，$DRS_{in}$ |
| T3 | R0←DR，SP←SP+n | $DRI_{out}$，$R0_{in}$，SP+1 |

（8）CALL (X)

| 节拍 | 微操作序列 | 微命令序列 |
|---|---|---|
| T1 | SP←SP－n，DR←PC | SP-1，$PC_{out}$，$DRI_{in}$ |
| T2 | AR←SP | $SP_{out}$，$AR_{in}$ |
| T3 | Memory [AR]←DR | $AR_{out}$，$DRS_{out}$，Mwrite |
| T4 | AR←IR(地址字段) | $IR_{out}$，$AR_{in}$ |
| T5 | DR←Memory[AR] | $AR_{out}$，Mread，$DRS_{in}$ |
| T6 | PC←DR | $DRI_{out}$，$PC_{in}$ |

# 微命令的生成

# 6.2.2 类x86系统控制单元设计

## $PC_{OUT}$信号：

- $PC_{out}$出现在：
  - 取指周期（定义为第1个CPU周期M1）的T1节拍 取指
  - 指令JZ offs执行周期（假设为第2个CPU周期M2）的T2节拍（当ZF=1时） JZ
  - 指令CALL (X)执行周期的T1节拍 CALL
  - ……
- 生成$PC_{out}$的逻辑表达式为：
  - 两级时序

    $PC_{out}$ = M1·T1 + M2·T2·JZ(相对寻址)·(ZF=1) +
    M2·T1·CALL(间接寻址) + ……
  - 一级时序

    $PC_{out}$ = T1 + T5·JZ(相对寻址)·(ZF=1) +
    T4·CALL(间接寻址) + ……

# 6.2.2 类x86系统控制单元设计

**$AR_{in}$信号：**

- $AR_{in}$出现在：
  - 取指周期的T1节拍 取指 MOV
  - 指令 MOV R0,X 和指令 MOV (R1),R0 执行周期的T1节拍
  - 指令 SUB R0,(X) 执行周期的T1和T3节拍 SUB
  - 指令 IN R0,P 和指令 OUT P,R0 执行周期的T1节拍 IN
  - 指令 PUSH R0 执行周期的T2节拍
  - 指令 POP R0 执行周期的T1节拍 POP
  - 指令 CALL (X) 执行周期的T2和T4节拍 CALL
  - 指令 RET 执行周期的T1节拍
  - ……
- 生成$AR_{in}$的逻辑表达式为：

# 6.2.2 类x86系统控制单元设计

**$AR_{in}$信号：**

- $AR_{in}$出现在：
- 生成$AR_{in}$的逻辑表达式为：
  - 两级时序

    $AR_{in}$＝M1·T1＋M2·T1·MOV(源操作数直接寻址+目的操作数寄存器间接寻址)＋M2·(T1+T3)·SUB(源操作数间接寻址)＋M2·T1·(IN(直接寻址)＋OUT(直接寻址))＋M2·T2·PUSH＋M2·T1·POP＋M2·(T2+T4)·CALL(间接寻址)＋M2·T1·RET＋……
  - 一级时序

    $AR_{in}$＝T1＋T4·MOV(源操作数直接寻址+目的操作数寄存器间接寻址)＋(T4+T6)·SUB(源操作数间接寻址)＋T4·(IN(直接寻址)+OUT(直接寻址))＋T5·PUSH＋T4·POP＋(T5+T7)·CALL(间接寻址)＋T4·RET＋……

# 6.2.2 类x86系统控制单元设计

**Mread信号：**

- Mread出现在：
  - 取指周期的T2节拍
  - 在指令 MOV R0,X 执行周期的T2节拍
  - 指令 SUB R0,(X) 执行周期的T2和T4节拍
  - 指令 POP R0 执行周期的T2节拍
  - 指令 CALL (X) 执行周期的T5节拍
  - 指令 RET 执行周期的T2节拍
  - ……
- 生成Mread的逻辑表达式为：
  - 两级时序

Mread ＝ M1·T2＋M2·T2·MOV(源操作数直接寻址)
＋M2·(T2+T4) ·SUB(源操作数间接寻址)
＋M2·T2·POP＋M2·T5·CALL(间接寻址)
＋M2·T2·RET＋……

# 6.2.2 类x86系统控制单元设计

**Mread信号：**

- Mread出现在：
  - 取指周期的T2节拍
  - 在指令 MOV R0,X 执行周期的T2节拍
  - 指令 SUB R0,(X) 执行周期的T2和T4节拍
  - 指令 POP R0 执行周期的T2节拍
  - 指令 CALL (X) 执行周期的T5节拍
  - 指令 RET 执行周期的T2节拍
  - ……
- 生成Mread的逻辑表达式为：
  - 一级时序

$$\begin{aligned} Mread = {} & T2 + T5 \cdot MOV(\text{源操作数直接寻址}) \\ & + (T5+T7) \cdot SUB(\text{源操作数间接寻址}) \\ & + T5 \cdot POP + T8 \cdot CALL(\text{间接寻址}) \\ & + T5 \cdot RET + \cdots\cdots \end{aligned}$$

# 6.2.2 类x86系统控制单元设计

由控制单元产生并加载到CPU内外的全部控制信号均可用下述形式表述：

第n个节拍

第m个CPU周期

指令译码器的第j个输出

两级时序：$C_i = \sum (M_m \cdot T_n \cdot I_j \cdot F_k)$

第k个CPU内部状态标志或CPU外部请求信号

在执行指令$I_j$时，若状态$F_k$满足要求，则在第m个CPU周期$M_m$的第n个节拍$T_n$，控制单元发出$C_i$控制命令，即在$M_m$、$T_n$、$I_j$和$F_k$同时有效时，$C_i$有效。

一级时序：$C_i = \sum (T_n \cdot I_j \cdot F_k)$

# 6.2 硬布线控制器设计

图6.6

## 小结：

- 每个控制信号的逻辑表达式就是一个与或逻辑方程式。
- 用一个与或逻辑电路就可以实现该控制信号的生成。
- 将所有控制信号的与或逻辑电路组合在一起就构成了硬布线控制单元。
- 时间信息（单周期实现不需要）、指令信息、状态信息是硬布线控制单元的输入，控制信号是硬布线控制单元的输出。
- 采用硬布线法设计控制器的特点：
  - 一旦完成了控制器的设计，改变控制器行为的唯一方法就是重新设计控制单元 → 修改不灵活
  - 在现代复杂的处理器中，需要定义庞大的控制信号逻辑方程组 → 与或组合电路实现困难

    ⇨ 微程序设计法

# 两种控制器设计方法比较

# 6.4 微程序控制器与硬布线控制器的比较

- 硬布线控制器
  - ✓ 速度快
  - ✓ 当计算机系统复杂时，设计困难
  - ✓ 一旦实现，不可修改和扩充
  - ✓ 常用于RISC处理器控制器的实现
- 微程序控制器
  - ✓ 设计简单化、规范化
  - ✓ 功能可修改、可扩充
  - ✓ 实现成本低，出错概率小
  - ✓ 比硬布线控制器速度慢
  - ✓ 常用于CISC处理器控制器的实现