# 指令系统

## 指令概述

- 指令格式的基本形式


- 操作数寻址方式（以X86为例）
  - 立即数寻址，地址码字段就是操作数本身。这种方式取操作数快，但**操作数表示范围有限**（因为地址码字段位数）

```
MOV EAX, 2008H //EAX赋初值2008H
```

  - 直接寻址，操作数地址由地址码字段直接给出。这种方式**寻址范围受限于地址码字段位数**，且需要访问主存，速度较慢

```
MOV EAX, [2008H] @将2008H主存单元中的内容送入EAX中
```

  - 寄存器寻址，地址码字段表示通用寄存器的编号。这种方式获得操作数不需要访问主存，因为操作数在寄存器中

```
MOV EAX, ECX @将ECX中的内容送入EAX中
```

  - 间接寻址，地址码字段给出的是操作数的间接地址。这种寻址方式解决了**直接寻址方式寻址范围受限的问题**，但是**需要访问主存两次**，指令执行速度较低

```
MOV EAX, @2008H @ 2008H单元的内容为操作数的地址，需要继续利用操作数的地址从主存获得操作数并送入EAX
```

  - 寄存器间接寻址，这种寻址方式只需要访问一次主存，且可以解决寻址范围不足的问题

```
MOV EAX, [EBX] @EBX中存放操作数的地址，需要继续利用操作数的地址从主存获得操作数并送入EAX
```

  - 相对寻址，假设地址码字段为 $D$，则操作数有效地址 $EA = PC + 1 + D$
  - 变址寻址，变址寄存器内容变化，指令提供的地址码字段作为偏移量保持不变。这种寻址方式主要用于**访问线性表元素，如数组**

```
MOV EAX, 32[ESI] @变址寄存器ESI的值加上偏移量32形成操作数地址访问主存，并将结果送入EAX
```

  - 基址寻址，这个过程与变址寻址相反，基址寄存器中的内容不变，而地址码字段作为偏移量会变化
- 指令格式设计（**知道如何拓展操作码字段**）
- RISC：指令条数少，指令长度固定，需要大量寄存器，一个机器周期完成一条机器指令，CPU采用硬布线控制

- MIPS指令（RISC架构）

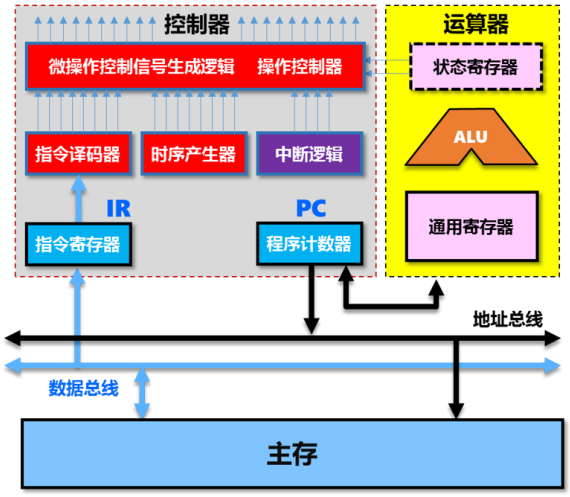
| | 31～26 | 25～21 | 20～16 | 15～11 | 10～06 | 05～00 |
|---|---|---|---|---|---|---|
| R型指令 | OP (6) = 0 | rs (5) | rt (5) | rd (5) | shamt (5) | funct (6) |
| I型指令 | OP (6) | rs (5) | rt (5) | Imm (16) | | |
| J型指令 | OP (6) | Address (26) | | | | |

图 5.24　MIPS 指令格式

# CPU设计

## CPU概述

- 功能：**程序控制（控制程序中指令执行的顺序）**、**操作控制（产生操作控制信号）**、时序控制、数据加工、中断处理
- 组成：运算器（算术逻辑运算单元和寄存器）、控制器




- 指令周期：将一条指令从取出到执行完成所需要的时间
  理解机器周期和时钟周期
- 数据通路：指令执行过程依次用到的功能部件的集合。
  - 总线型：冲突率高，并发性差，但成本低
  - 专用通路：并发性强，控制相对简单，但成本高

## 几种架构的CPU

### 单总线多周期CPU

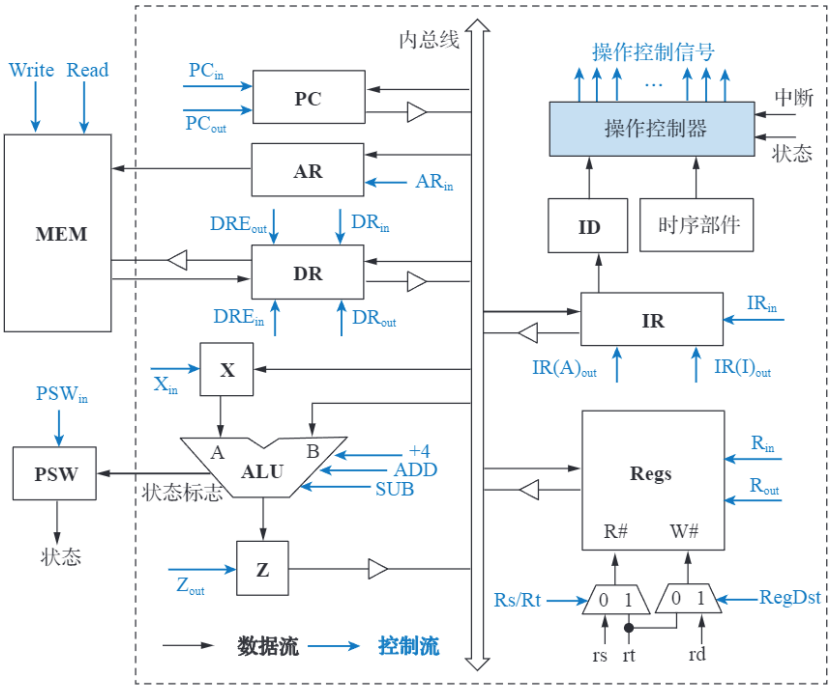



图 6.8　单总线结构的计算机框图

| 序号 | 控制信号 | 功能说明 |
|---|---|---|
| 1 | $PC_{in}$ | 控制 $PC$ 接收来自内总线的数据 |
| 2 | $PC_{out}$ | 控制 $PC$ 向内总线输出数据 |
| 3 | $AR_{in}$ | 控制 $AR$ 接收来自内总线的数据 |
| 4 | $DR_{in}$ | |
| 5 | $DR_{out}$ | |
| 6 | $DRE_{in}$ | 控制 $DR$ 接收从主存读出的数据 |
| 7 | $DRE_{out}$ | 控制 $DR$ 向主存输出数据 |
| 8 | $X_{in}$ | 控制暂存寄存器 $X$ 接收来自内总线的数据 |
| 9 | $PSW_{in}$ | 控制状态寄存器 $PSW$ 接收来自 $ALU$ 的状态 |
| 10 | $Z_{out}$ | 控制暂存寄存器 $Z$ 向内总线输出数据 |
| 11 | $IR_{in}$ | 控制 $IR$ 接收来自内总线的指令 |
| 12 | $IR(I)_{out}$或者$IR_imm_{out}$ | 控制指令中的立即数（符号扩展为32位）输出到内总线 |
| 13 | $IR(A)_{out}$ | 控制指令中的分支目标地址输出到内总线 |
| 14 | $Write$ | 存储器写命令 |
| 15 | $Read$ | |
| 16 | $R_{in}$ | 控制寄存器堆接收来自内总线的数据，写入 $W\#$ 端口对应的寄存器中 |

| 序号 | 控制信号 | 功能说明 |
| --- | --- | --- |
| 17 | $R_{out}$ | 控制寄存器堆输出 $R\#$ 端口对应的寄存器中的数据到内总线 |
| 18 | $Rs/Rt$ | 控制 $MUX$ 选择送入 $R\#$ 的寄存器编号，0 时送入 $rs$ 字段，1 时送入 $rt$ 字段 |
| 19 | $RegDst$ | 控制 $MUX$ 选择送入 $W\#$ 的寄存器编号，0 时送入 $rt$ 字段，1 时送入 $rd$ 字段 |

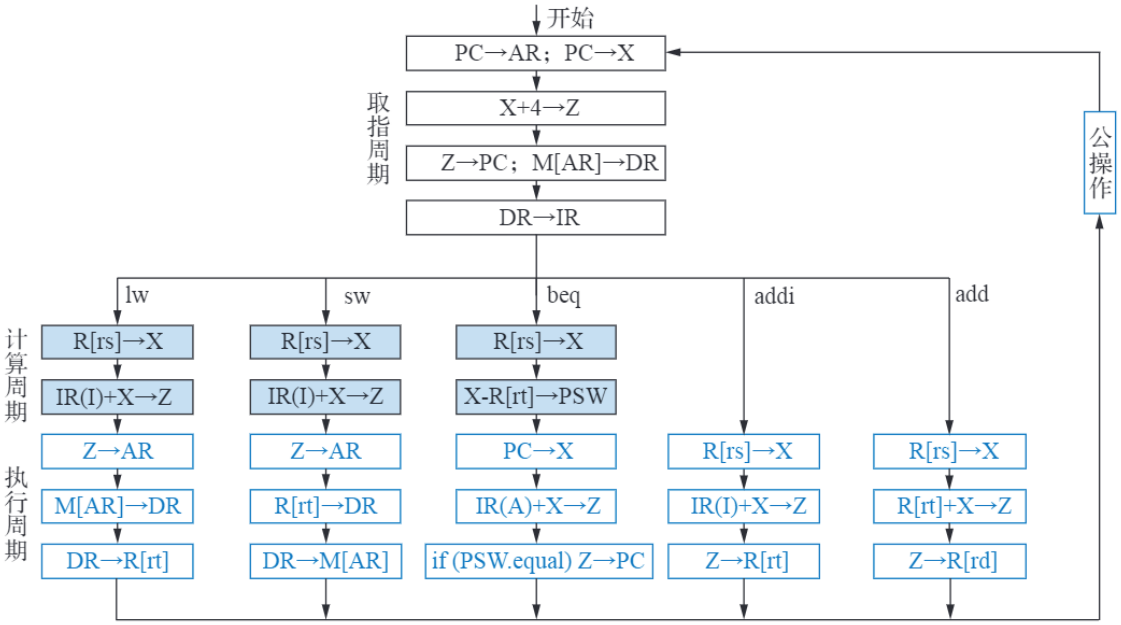



图 6.14　单总线结构数据通路指令方框图

**单周期MIPS CPU**

**一个时钟周期内完成一条指令的取出与执行操作，因此如果采用微程序控制器只需要一条微指令**

- 控制器是组合逻辑电路

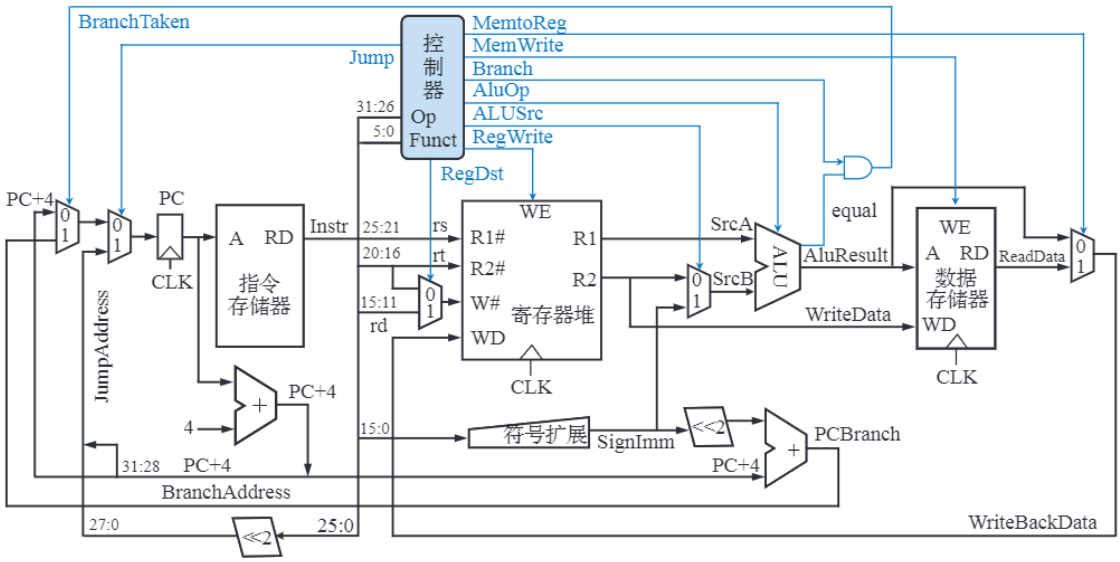



图 6.25　单周期 MIPS 处理器的数据通路高层视图

**多周期MIPS CPU**

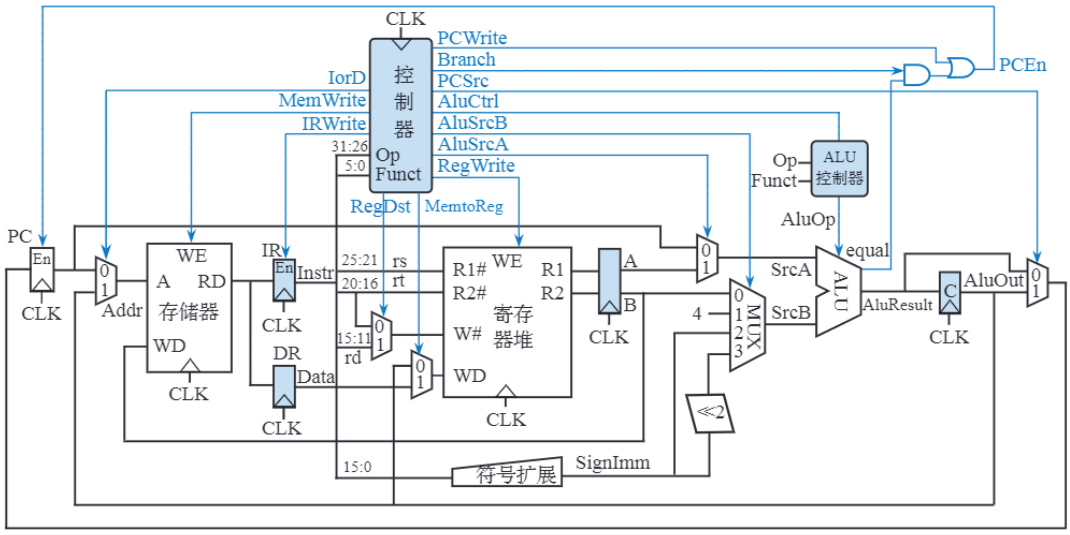



图 6.27　多周期 MIPS 处理器的数据通路高层视图

**数据通路特点**

- 不再区分指令存储器和数据存储器，指令和数据保存在同一存储器中
- $ALU$ 可以重复使用，不需要额外设置加法器（$PC+4$）
- 每个功能部件后增加了缓冲寄存器

表 6.10　多周期 MIPS 处理器的控制信号及功能

| 控制信号 | 信号分类 | 功能说明 |
|---|---|---|
| **Branch** | 指令译码信号 | 分支指令译码信号，beq、bne 指令需要产生类似的译码信号 |
| **RegDst** | 多路选择控制 | 写入目的寄存器选择，为 1 时写入 rd 寄存器，为 0 时写入 rt 寄存器 |
| **AluSrcA** | 多路选择控制 | 控制 ALU 的第一输入，为 0 时输入 PC 值，为 1 时输入寄存器 A 的值 |
| **AluSrcB** | 多路选择控制 | 控制 ALU 的第二输入，值为 0~3 时分别输入 B 值、常量 4、立即数、偏移地址 |
| **AluCtrl** | 多路选择控制 | ALU 控制器选择控制信号，为 0 时控制 ALU 做加法；为 1 时 ALU 的操作由指令功能译码决定，如 R 型指令由 funct 字段决定 |
| **MemToReg** | 多路选择控制 | 为 0 时写入暂存在 C 寄存器中的 ALU 运算结果，为 1 时写入 DR 寄存器的值 |
| **PCSrc** | 多路选择控制 | 控制程序计数器 PC 的输入，为 0 时输入 PC+4，为 1 时输入分支目标地址 |
| **IorD** | 多路选择控制 | 控制存储器地址输入，为 0 时输入指令地址，为 1 时输入寄存器 C 中的数据地址 |
| **RegWrite** | 功能部件控制 | 控制寄存器堆写操作，为 1 时数据需要写入指定寄存器，受时钟驱动 |
| **MemWrite** | 功能部件控制 | 控制数据存储器写操作，为 0 时进行读操作，为 1 时进行写操作，写受时钟驱动 |
| **PCWrite** | 功能部件控制 | 控制数据存储器写操作，为 0 时进行读操作，为 1 时进行写操作，写受时钟驱动 |
| **IRWrite** | 功能部件控制 | 控制指令寄存器写操作，为 1 时有效，受时钟驱动 |

表 6.11　MIPS 指令多周期数据通路及控制信号

| 指令类型 | 时钟周期 | 操作 | 控制信号（非零值） |
|---|---|---|---|
| 取指令 | *T*1 | M[PC] → IR　PC+4 → PC | IRWrite=PCWrite=1　AluSrcB=1 |
| | *T*2 | R[rs] → A; R[rt] → B; PC+Imm<<2 → C | AluSrcB=3 |
| R 型运算 | *T*3 | A op B → C | AluCtrl=AluSrcA=1 |
| | *T*4 | C → R[rd] | RegWrite=RegDst=1 |
| I 型运算 | *T*3 | A op imm → C | AluCtrl=AluSrcA=1　AluSrcB=2 |
| | *T*4 | C → R[rt] | RegWrite=1 |
| lw 指令 | *T*3 | A + imm → C | AluSrcA=1　AluSrcB=2 |
| | *T*4 | M[C] → DR | IorD=1 |
| | *T*5 | DR → R[rt] | MemToReg=RegWrite=1 |
| sw 指令 | *T*3 | A + imm → C | AluSrcA=1　AluSrcB=2 |
| | *T*4 | B → M[C] | IorD=MemWrite=1 |
| beq 指令 | *T*3 | if (A == B)　C → PC | AluSrcA=PCSrc=Branch=1 |

| 当前状态(现态) | | | | | 输入信号 | | | | | | | 下一状态（次态） | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| S3 | S2 | S1 | S0 | 现态10进制 | R_Type | LW | SW | BEQ | BNE | SYSCALL | ADDI | 次态10进制 | N3 | N2 | N1 | N0 |
| 0 | 0 | 0 | 0 | 0 | | | | | | | | 1 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 1 | 1 | 1 | | | | | | | 7 | 0 | 1 | 1 | 1 |
| 0 | 0 | 0 | 1 | 1 | | 1 | | | | | | 2 | 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 1 | 1 | | | 1 | | | | | 5 | 0 | 1 | 0 | 1 |
| 0 | 0 | 0 | 1 | 1 | | | | 1 | | | | 9 | 1 | 0 | 0 | 1 |
| 0 | 0 | 0 | 1 | 1 | | | | | 1 | | | 10 | 1 | 0 | 1 | 0 |
| 0 | 0 | 0 | 1 | 1 | | | | | | 1 | | 13 | 1 | 1 | 0 | 1 |
| 0 | 0 | 0 | 1 | 1 | | | | | | | 1 | 12 | 1 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 | 2 | | | | | | | | 3 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 1 | 3 | | | | | | | | 4 | 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 0 | 4 | | | | | | | | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 5 | | | | | | | | 6 | 0 | 1 | 1 | 0 |
| 0 | 1 | 1 | 0 | 6 | | | | | | | | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 7 | | | | | | | | 8 | 1 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 8 | | | | | | | | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 9 | | | | | | | | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 10 | | | | | | | | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 | 11 | | | | | | | | 12 | 1 | 1 | 0 | 0 |
| 1 | 1 | 0 | 0 | 12 | | | | | | | | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 13 | | | | | | | | 13 | 1 | 1 | 0 | 1 |

- 现代时序硬布线控制器设计的关键是画出所有指令执行过程的 $Morre$ 型 $FSM$，然后画出现代时序指令状态转移表，采用同步时序电路的设计方法设计电路
- 组成部分：$FSM$ 状态机（产生次态）、硬布线控制器（输出控制信号）、状态寄存器
- 控制信号只是状态的函数

| 微指令功能 | 状态 | 微指令地址 | IorD | PcSrc | AluSrcA | AluSrcB | MemToReg | RegDst | IrWrite | PcWrite | RegWrite | MemWrite | MemRead | BEQ | BNE | AluControl |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 取指令 | 0 | 0000 | 0 | 0 | 0 | 01 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 00 |
| 译码 | 1 | 0001 | 0 | 0 | 0 | 11 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 00 |
| LW1 | 2 | 0010 | 00 | 0 | 1 | 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 00 |
| LW2 | 3 | 0011 | 1 | 0 | 0 | 00 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 00 |
| LW3 | 4 | 0100 | 0 | 0 | 0 | 00 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 00 |
| SW1 | 5 | 0101 | 0 | 0 | 1 | 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 00 |
| SW2 | 6 | 0110 | 1 | 0 | 0 | 00 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 00 |
| R1 | 7 | 0111 | 0 | 0 | 1 | 00 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 10 |
| R2 | 8 | 1000 | 0 | 0 | 0 | 00 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 10 |
| BEQ | 9 | 1001 | 0 | 1 | 1 | 00 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 00 |
| BNE | 10 | 1010 | 0 | 1 | 1 | 00 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 00 |
| ADD1 | 11 | 1011 | 0 | 0 | 1 | 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 00 |
| ADD2 | 12 | 1100 | 0 | 0 | 0 | 00 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 00 |
| SYSCALL | 13 | 1101 | 1 | 0 | 0 | 00 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 11 |

| 机器指令译码信号 | | | | | | | 微程序入口地址 | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| R_Type | ADDI | LW | SW | BEQ | BNE | SYSCALL | 入口地址10进制 | S3 | S2 | S1 | S0 |
| 1 | | | | | | | 7 | 0 | 1 | 1 | 1 |
| | 1 | | | | | | 11 | 1 | 0 | 1 | 1 |
| | | 1 | | | | | 2 | 0 | 0 | 1 | 0 |
| | | | 1 | | | | 5 | 0 | 1 | 0 | 1 |
| | | | | 1 | | | 9 | 1 | 0 | 0 | 1 |
| | | | | | 1 | | 10 | 1 | 0 | 1 | 0 |
| | | | | | | 1 | 13 | 1 | 1 | 0 | 1 |

| 微指令功能 | 状态 | 微指令地址 | 微指令 | 十六进制 |
|---|---|---|---|---|
| 取指令 | 0 | 0000 | 00001001100100000001 | 13201 |
| 译码 | 1 | 0001 | 00011000000000010000 | 30010 |
| LW1 | 2 | 0010 | 00011000000000000011 | 60003 |
| LW2 | 3 | 0011 | 10000000001000000100 | 100204 |
| LW3 | 4 | 0100 | 00000100010000000000 | 8800 |
| SW1 | 5 | 0101 | 00110000000000000110 | 60006 |
| SW2 | 6 | 0110 | 10000000010000000000 | 100400 |
| R1 | 7 | 0111 | 00100000000001001000 | 40048 |
| R2 | 8 | 1000 | 00000010010001000000 | 4840 |
| BEQ | 9 | 1001 | 01100000000100000000 | C0100 |
| BNE | 10 | 1010 | 01100000000010000000 | C0080 |
| ADD1 | 11 | 1011 | 00110000000000001100 | 6000C |
| ADD2 | 12 | 1100 | 00000000100000000000 | 800 |
| SYSCALL | 13 | 1101 | 10000000000001101101 | 10006D |

- $P_0$ 为0时选择微指令字中的下址字段作为后续地址，为1时选择指令字对应的微程序入口地址作为后续地址
- 三大组成部分：控制存储器（存放微指令）、地址转移逻辑（确定要执行的下一条微指令的地址）、微地址寄存器

**单级中断与多级中断（根据PPT进程图来理解）**

- 关中断
- 保存断点（一般是下一条指令的地址）
- 中断识别与中断服务（一般还会保存现场）
- 恢复现场后开中断
- 中断返回，继续执行下一条指令

- 多周期MIPS CPU（采用微程序控制器）若要加入中断逻辑，需要增加中断隐指令的微程序，该微程序的功能是保存断点，修改PC值为中断程序入口地址，且微指令P字段需要增加一位用于中断判断，每条微程序最后一条指令中断判断位为1，如果当前有中断请求信号，需要分支跳转到中断隐指令对应的微程序