

课程报告

课程名称： 计算机组成原理

报告题目：基于MIPS指令系统的处理器设计

所在院系： 计算学部

所在专业： 人工智能

学生姓名： 刘子康

学生学号： 2022113416

选课时间： 2024年春季学期

评阅成绩：

目录

[一、引言 1](#_Toc170463581)

[1.1 MIPS指令系统概述 1](#_Toc170463582)

[1.2 MIPS指令集格式 1](#_Toc170463583)

[1.2.1 R型指令 1](#_Toc170463584)

[1.2.2 I型指令 1](#_Toc170463585)

[1.2.3 J型指令 2](#_Toc170463586)

[1.3 MIPS处理器中的寄存器 2](#_Toc170463587)

[二、指令格式设计 3](#_Toc170463588)

[2.1 add指令设计 3](#_Toc170463589)

[2.2 sub指令设计 3](#_Toc170463590)

[2.3 lw指令设计 3](#_Toc170463591)

[2.4 sw指令设计 3](#_Toc170463592)

[2.5 j指令设计 3](#_Toc170463593)

[三、微操作定义 3](#_Toc170463594)

[3.1取指阶段 4](#_Toc170463595)

[3.2执行阶段 4](#_Toc170463596)

[3.2.1 add指令微操作 4](#_Toc170463597)

[3.2.2 sub指令微操作 4](#_Toc170463598)

[3.2.3 lw指令微操作 5](#_Toc170463599)

[3.2.4 sw指令微操作 5](#_Toc170463600)

[3.2.5 j指令微操作 6](#_Toc170463601)

[3.3微操作总计 6](#_Toc170463602)

[四、节拍划分 7](#_Toc170463603)

[4.1取指微程序节拍划分 7](#_Toc170463604)

[4.2 add指令微程序节拍划分 7](#_Toc170463605)

[4.3 sub指令微程序节拍划分 8](#_Toc170463606)

[4.4 lw指令微程序节拍划分 8](#_Toc170463607)

[4.5 sw指令微程序节拍划分 8](#_Toc170463608)

[4.6 j指令微程序节拍划分 9](#_Toc170463609)

[五、处理器结构设计框图及功能描述 9](#_Toc170463610)

[5.1处理器结构设计框图 9](#_Toc170463611)

[5.2处理器功能描述 9](#_Toc170463612)

[5.2.1控制器 9](#_Toc170463613)

[5.2.2运算器 10](#_Toc170463614)

[5.2.3内部总线 10](#_Toc170463615)

[六、微程序设计及微指令序列 10](#_Toc170463616)

[6.1微程序控制单元 10](#_Toc170463617)

[6.2微指令编码方式 11](#_Toc170463618)

[6.3微指令序列地址形成 11](#_Toc170463619)

[6.3.1根据机器指令操作码形成 11](#_Toc170463620)

[6.3.2直接由微指令下地址字段给出 11](#_Toc170463621)

[6.4微指令字长及格式 11](#_Toc170463622)

[6.5微指令码点 12](#_Toc170463623)

[七、总结 13](#_Toc170463624)

# 

# 一、引言

## 1.1 MIPS指令系统概述

MIPS Microprocessor without Interlocked Pipeline Stages 无互锁流水线微处理器）是一种精简指令集（RISC）架构，由美国 MIPS计算机系统公司由斯坦福大学团队于1984年创立， 现为美普思科技）开发以其高效、简洁和强大的特性，在嵌入式系统、工作站和超级计算机等领域得到了广泛应用。

MIPS架构有多个版本，包括 MIPS I、II、III、IV，以及 MIPS V，这五个版本又分别分为 MIPS32/64 Release（即其32位/64位实现）。2021年3月，美普思科技宣布停止开发MIPS架构并加入RISC V基金会，未来的处理器设计将基于RISC V架构。

## 1.2 MIPS指令集格式

MIPS 32位处理器的指令格式分为R型、I型和J型。R型为寄存器型，即两个源操作数和目的操作数都是寄存器性；I型为操作数含有立即数；而J型特指转移类型指令。

## 1.2.1 R型指令

·OP：操作码（固定000000）

·rs、rt：两个源操作数寄存器编号

·rd：目的操作数寄存器编号

·shamt：位移量（执行移位操作时指明需要移动的次数）

·func：指令具体操作

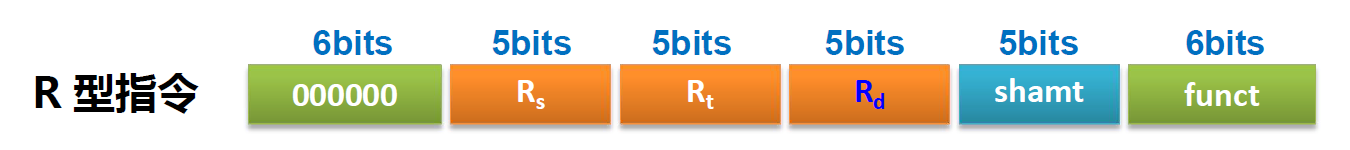


图1-1 R型指令格式

## 1.2.2 I型指令

·OP：操作码

·rs：第一个源操作数寄存器编号

·immediate：立即数，或load/store指令和分支指令的偏移地址

·rt：目的操作数所在的寄存器编号

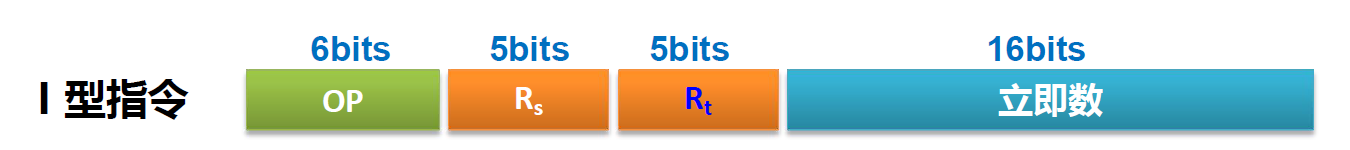


图1-2 I型指令格式

## 1.2.3 J型指令

·OP：操作码

·address：26位无条件转移地址（立即数）

图1-3 J型指令格式

## 1.3 MIPS处理器中的寄存器

MIPS处理器有32个32位通用寄存器R0~R31，如下表所示：

表1-1 MIPS处理器中的寄存器

|  |  |  |
| --- | --- | --- |
| 寄存器编号 | 别名 | 功能 |
| $0 | $zero | 常量寄存器，其值永远为0  Constant 0 |
| $1 | $at | 汇编暂存寄存器  Assembly Temporary |
| $2, $3 | $v0, $v1 | 用于存储子程序的返回值  Function Result |
| $4-$7 | $a0-$a3 | 子程序调用的前4个参数  Argument 1 to 4 |
| $8-$15 | $t0-$t7 | 临时变量寄存器  Unsaved Temporary |
| $16-$23 | $s0-$s7 | 变量寄存器  Saved Temporary |
| $24-$25 | $t8, $t9 | 临时变量寄存器  Unsaved Temporary |
| $26, $27 | $k0, $k1 | 保留给中断和自陷程序使用  Reserved for EXCEPTION |
| $28 | $gp | 全局指针  Pointer to Global Data |
| $29 | $sp | 堆栈指针  Stack Pointer |
| $30 | $fp | 帧指针  Frame Pointer |
| $31 | $ra | 函数返回地址  Return Address |

硬件上，这些寄存器并没有区别（除了$0以外），区分的目的是为了不同的编译器产生的代码可以正常的互相调用。

对于Linux 操作系统，位置无关代码的用户空间调用约定中还要求当调用函数时，$t9寄存器必须包含该函数的地址。这个约定源于MIPS的System V ABI补充规定。

# 二、指令格式设计

从R类指令、I类指令和J类指令中选择“add”、“sub”、“lw”、“sw”和“j”五种指令，用于设计一个基于MIPS指令系统的RISC处理器。

## 2.1 add指令设计

add指令的汇编形式为，实现32位整数加法功能，rs和rt寄存器分别存储被加数和加数，结果存储在rd寄存器中。

## 2.2 sub指令设计

sub指令的汇编形式为，实现32位整数减法功能，rs和rt寄存器分别存储被减数和减数，结果存储在rd寄存器中。

## 2.3 lw指令设计

lw指令的汇编形式为，该指令将offset进行位扩展后与rs寄存器中数据(base)相加获得访存地址，从该地址取出数据并存储在rt寄存器中。

## 2.4 sw指令设计

sw指令的汇编形式为，该指令将offset进行位扩展后与rs寄存器中数据(base)相加获得访存地址，从rt寄存器取出数据并存储在该地址处。

## 2.5 j指令设计

j指令的汇编形式为，该指令无条件跳转到26位立即数表示的目标地址。

# 三、微操作定义

以下基于CPU基本结构框架图与前文的MIPS寄存器来定义指令不同阶段的微操作。由于定义的五种指令均不存在间址和中断周期，所以指令周期仅包含取指周期和执行周期，将其分为五段指令流水线，分别是取指(IF)、译码&取操作数(ID)、执行(EX)、访存(MEM)和写回(WB)。

## 3.1取指阶段

从主存中读取指令到指令寄存器IR。

表3-1 取指阶段微操作

|  |  |  |
| --- | --- | --- |
| 微操作 | 描述 | 控制信号 |
| PC→MAR | 现行指令地址送至存储器地址寄存器 | PCo, MARi |
| 1→R | 向主存发送读命令，启动主存做读操作 | MemR=1 |
| M(MAR)→MDR | 将MAR（通过地址总线）所指的主存单元中的内容（指令）经数据总线读至MDR 内 | MARo, MDRi |
| MDR→IR | 将MDR的内容（指令）送至IR | MDRo, IRi |
| (PC)+1→PC | 形成下一条指令的地址 | +4 |

## 3.2执行阶段

## 3.2.1 add指令微操作

（1）取数：取操作数，将寄存器中数据暂存到Y中；

表3-2 add指令取数微操作

|  |  |  |
| --- | --- | --- |
| 微操作 | 描述 | 控制信号 |
| (R(IR[21~25]))→Y | 将rs寄存器中的被加数暂存到Y中 | R(IR[21~25])o, Yi |

（2）运算：将两个寄存器值相加（使用ALU），实现加法操作；

表3-3 add指令运算微操作

|  |  |  |
| --- | --- | --- |
| 微操作 | 描述 | 控制信号 |
| (Y)+R(IR[16~20])→Z | 将(Y)与rt寄存器中的操作数求和并送入Z，其中rt寄存器中的操作数直接通过Bus送到ALU | Yo, Ro, ALUi  “+”, Zi |

（3）写回：将结果存储到目标寄存器rd中。

表3-4 add指令写回微操作

|  |  |  |
| --- | --- | --- |
| 微操作 | 描述 | 控制信号 |
| (Z)→R(IR(11~15)) | 将暂存器Z中存储的运算结果写入目的寄存器rd中 | Ri，Zo |

## 3.2.2 sub指令微操作

（1）取数：取操作数，将寄存器rs中数据暂存到Y中；

表3-5 sub指令取数微操作

|  |  |  |
| --- | --- | --- |
| 微操作 | 描述 | 控制信号 |
| (R(IR[21~25]))→Y | 将rs寄存器中的被减数暂存到Y中 | R(IR(21~25))o, Yi |

（2）执行：从寄存器rs的值中减去寄存器rt的值（使用ALU）；

表3-6 sub指令运算微操作

|  |  |  |
| --- | --- | --- |
| 微操作 | 描述 | 控制信号 |
| (Y)-R(IR(16~20))→Z | 将(Y)与rt寄存器中的操作数求和并送入Z，其中rt寄存器中的操作数直接通过Bus送到ALU | Yo, Ro, ALUi  “+”, Zi |

（3）写回：将结果存储到目标寄存器rd中。

表3-7 sub指令写回微操作

|  |  |  |
| --- | --- | --- |
| 微操作 | 描述 | 控制信号 |
| (Z)→R(IR(11~15)) | 将暂存器Z中存储的运算结果写入目的寄存器rd中 | Ri，Zo |

## 3.2.3 lw指令微操作

（1）取数：取操作数，将基址寄存器rs中地址暂存到Y中；

表3-8 lw指令取数微操作

|  |  |  |
| --- | --- | --- |
| 微操作 | 描述 | 控制信号 |
| (R(IR[21~25]))→Y | 将rs寄存器中的地址暂存到Y中 | R(IR(21~25))o, Yi |

（2）运算：将基址寄存器rs的值与偏移量offset相加，得到数据内存地址；

表3-9 lw指令运算微操作

|  |  |  |
| --- | --- | --- |
| 微操作 | 描述 | 控制信号 |
| (Y)+Extended(IR(0~15))→Z | 将(Y)与offset偏移量求和并送入Z | Yo, ALUi  “+”，Zi |

（3）访存：从计算出的内存地址中读取数据；

表3-10 lw指令访存微操作

|  |  |  |
| --- | --- | --- |
| 微操作 | 描述 | 控制信号 |
| (Z)→MAR | 将暂存器Z中存储的运算地址送至MAR | Zo，MARi |
| 1→R | 向主存发送读命令，启动主存做读操作 | MemR=1 |
| M(MAR)→MDR | 将MAR所指的主存单元中的内容（指令）经数据总线读至MDR 内 | MDRi，MARo |

（4）写回：将读取到的数据存储到目标寄存器rt中。

表3-11 lw指令写回微操作

|  |  |  |
| --- | --- | --- |
| 微操作 | 描述 | 控制信号 |
| MDR→R(IR[16~20]) | 将MDR的数据放入目标寄存器rt中 | MDRo，Ri |

## 3.2.4 sw指令微操作

（1）取数：取操作数，将基址寄存器rs中地址暂存到Y中；

表3-12 sw指令取数微操作

|  |  |  |
| --- | --- | --- |
| 微操作 | 描述 | 控制信号 |
| (R(IR[21~25]))→Y | 将rs寄存器中的地址暂存到Y中 | R(IR(21~25))o, Yi |

（2）运算：将基址寄存器rs的值与偏移量offset相加，得到数据内存地址；

表3-13 lw指令运算微操作

|  |  |  |
| --- | --- | --- |
| 微操作 | 描述 | 控制信号 |
| (Y)+Extended(IR[0~15])→Z | 将(Y)与offset偏移量求和并送入Z | Yo, ALUi  “+”，Zi |

（3）访存：将数据寄存器rt的值写入到计算出的内存地址中。

表3-14 lw指令访存微操作

|  |  |  |
| --- | --- | --- |
| 微操作 | 描述 | 控制信号 |
| (Z)→MAR | 将Z中计算的地址送至存储器地址寄存器 | Zo, MARi |
| (R(IR[16~20]))→MDR | 将rt寄存器中的数据送至MDR | Ro |
| 1→W | 向主存发送写命令，启动主存做写操作 | MemW=1 |
| MDR→M(MAR) | 将MDR的数据写入主存 | MDRi, MDRo |

## 3.2.5 j指令微操作

（1）运算：计算跳转地址，并将其写入到程序计数器PC中，以便下一条指令从新地址开始执行。

表3-19 j指令运算微操作

|  |  |  |
| --- | --- | --- |
| 微操作 | 描述 | 控制信号 |
| {PC, IR[0~25]}→PC | 计算得到跳转后的PC | PCi |

## 3.3微操作总计

设计该基于MIPS 指令系统的处理器共需要16个微操作，如下表所示：

表3-20 所用微操作总计

|  |  |
| --- | --- |
| 序号 | 微操作 |
| 0 | PC→MAR |
| 1 | 1→R |
| 2 | M(MAR)→MDR |
| 3 | MDR→IR |
| 4 | (PC)+1→PC |
| 5 | (R(IR[21~25]))→Y |
| 6 | (Y)+R(IR[16~20])→Z |
| 7 | (Z)→R(IR[11~15]) |
| 8 | (Y)-R(IR[16~20])→Z |
| 9 | (Y)+Extended(IR[0~15])→Z |
| 10 | (Z)→MAR |
| 11 | MDR→R(IR[16~20]) |
| 12 | (R(IR[16~20]))→MDR |
| 13 | 1→W |
| 14 | MDR→M(MAR) |
| 15 | {PC, IR[0~25]}→PC |

# 四、节拍划分

安排微操作节拍应按照以下3点原则：

①对于次序不可改变的微操作，应注意其先后顺序；

②被控制对象不同的微操作应尽量安排在同一个节拍内，以节省时间；

③所占时间不长的微操作应安排在同一个节拍内完成，且允许它们有先后次序。

## 4.1取指微程序节拍划分

微程序控制的所有控制信号均来自微指令，而微指令又存于控制存储器中，因此想要完成系列微操作，必须先将微指令从控制存储器中读出，即必须线给出这些微指令的地址。除第一条微指令外，其余微指令地址均由上一条微指令的下地址字段直接给出，因此在每一条微指令后增加一个将其下地址字段送至CMAR的微操作；对于最后一条微指令，其后续微指令的地址由微地址形成部件形成。

取指阶段各指令微操作相同，故节拍划分也一致，且所有微指令均由T的上升沿打入CMDR中。

表4-1 取指阶段节拍划分

|  |  |  |
| --- | --- | --- |
| 节拍 | 微操作 | 说明 |
| T0 | PC→MAR, 1→R | 微指令1 |
| T1 | Ad(CMDR)→CMAR | 微指令2 |
| T2 | M(MAR)→MDR, (PC)+1→PC | 微指令3 |
| T3 | Ad(CMDR)→CMAR | 微指令4 |
| T4 | MDR→IR, OP(IR)→微地址生成部件（编码器） | 微指令5 |
| T5 | OP(IR)→微地址生成部件→CMAR | 微指令6 |

其中(PC)+1→PC操作也可安排在T2节拍，因为PC→MAR后PC内容就可修改，并且其与MDR→IR操作无先后顺序要求。

## 4.2 add指令微程序节拍划分

表4-2 add指令微程序节拍划分

|  |  |  |  |
| --- | --- | --- | --- |
| 节拍 | 微操作 | 说明 | 阶段 |
| T0 | (R(IR[21~25]))→Y | 微指令7 | 取数 |
| T1 | Ad(CMDR)→CMAR | 微指令8 |
| T0 | (Y)+R(IR[16~20])→Z | 微指令9 | 运算 |
| T1 | Ad(CMDR)→CMAR | 微指令10 |
| T0 | (Z)→R(IR[11~15]) | 微指令11 | 写回 |
| T1 | Ad(CMDR)→CMAR | 微指令12 |

## 4.3 sub指令微程序节拍划分

表4-3 sub指令微程序节拍划分

|  |  |  |  |
| --- | --- | --- | --- |
| 节拍 | 微操作 | 说明 | 阶段 |
| T0 | (R(IR[21~25]))→Y | 微指令13 | 取数 |
| T1 | Ad(CMDR)→CMAR | 微指令14 |
| T0 | (Y)-R(IR[16~20])→Z | 微指令15 | 运算 |
| T1 | Ad(CMDR)→CMAR | 微指令16 |
| T0 | (Z)→R(IR[11~15]) | 微指令17 | 写回 |
| T1 | Ad(CMDR)→CMAR | 微指令18 |

## 4.4 lw指令微程序节拍划分

表4-4 lw指令微程序节拍划分

|  |  |  |  |
| --- | --- | --- | --- |
| 节拍 | 微操作 | 说明 | 阶段 |
| T0 | (R(IR[21~25]))→Y | 微指令19 | 取数 |
| T1 | Ad(CMDR) → CMAR | 微指令20 |
| T0 | (Y)+Extended(IR[0~15])→Z | 微指令21 | 运算 |
| T1 | Ad(CMDR) → CMAR | 微指令22 |
| T0 | (Z)→MAR, 1→R | 微指令23 | 访存 |
| T1 | Ad(CMDR) → CMAR | 微指令24 |
| T2 | M(MAR) → MDR | 微指令25 |
| T3 | Ad(CMDR) → CMAR | 微指令26 |
| T0 | MDR→R(IR[16~20]) | 微指令27 | 写回 |
| T1 | Ad(CMDR) → CMAR | 微指令28 |

## 4.5 sw指令微程序节拍划分

表4-5 sw指令微程序节拍划分

|  |  |  |  |
| --- | --- | --- | --- |
| 节拍 | 微操作 | 说明 | 阶段 |
| T0 | (R(IR[21~25]))→Y | 微指令29 | 取数 |
| T1 | Ad(CMDR) → CMAR | 微指令30 |
| T0 | (Y)+Extended(IR[0~15])→Z | 微指令31 | 运算 |
| T1 | Ad(CMDR) → CMAR | 微指令32 |
| T0 | (Z)→MAR, (R(IR[16~20]))→MDR, 1→W | 微指令33 | 访存 |
| T1 | Ad(CMDR) → CMAR | 微指令34 |
| T2 | MDR →M(MAR) | 微指令35 |
| T3 | Ad(CMDR) → CMAR | 微指令36 |

## 4.6 j指令微程序节拍划分

表4-6 j指令微程序节拍划分

|  |  |  |  |
| --- | --- | --- | --- |
| 节拍 | 微操作 | 说明 | 阶段 |
| T0 | (R(IR[21~25]))→Y | 微指令37 | 取数 |
| T1 | Ad(CMDR) → CMAR | 微指令38 |
| T0 | {PC, IR[0~25]}→PC | 微指令39 | 运算 |
| T1 | Ad(CMDR) → CMAR | 微指令40 |

# 五、处理器结构设计框图及功能描述

## 5.1处理器结构设计框图

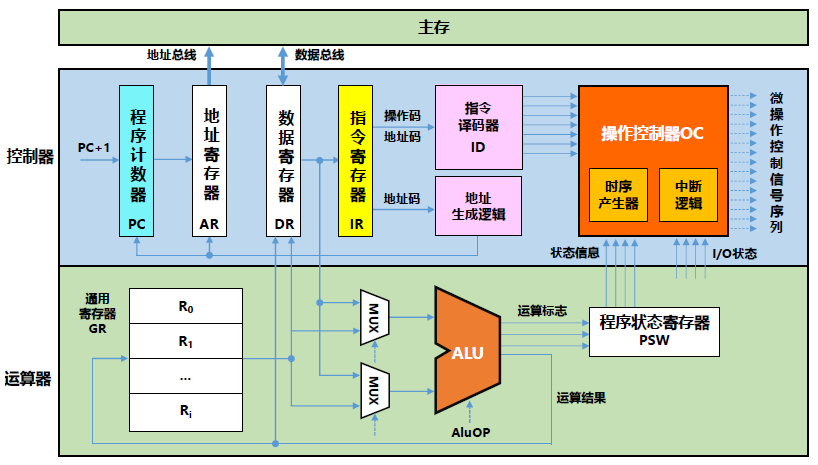


图5-1 简单CPU模型

## 5.2处理器功能描述

## 5.2.1控制器

（1）CU（控制单元）

·从指令寄存器接收指令，进行译码，并产生相应的控制信号。

·控制处理器内部各部件的协同工作，如数据通路的建立和关闭、寄存器的读写等。

·处理中断和异常，根据中断向量或异常类型跳转到相应的处理程序。

（2）PC（程序计数器）

·存放现行指令的地址，维护指令执行的顺序，通过增量或跳转来更新其值，以指向下一条要执行的指令的地址，通常具有计数功能。

（3）IR（指令寄存器）

·从指令存储器读取并暂时存储当前欲执行的指令

（4）MAR（存储器地址寄存器）

·用于存放将被访问的存储单元的地址。

（5）MDR（存储器数据寄存器）

·用于存放欲存入存储器中的数据或最近从存储器中读出的数据。

## 5.2.2运算器

（1）寄存器

·包含32个32位通用寄存器R0~R31，用于存储数据和中间结果。

·接收来自ALU或数据总线的数据，并根据控制信号进行读写操作。

（2）ALU（算术逻辑单元）

·运算器的核心部分，接收来自寄存器文件的两个操作数，并根据指令执行算术或逻辑运算操作。

·将运算结果输出到寄存器文件或数据总线。

## 5.2.3内部总线

·连接处理器内部各部件，用于传输数据和内存地址，以便在不同部件之间交换信息。

# 六、微程序设计及微指令序列

## 6.1微程序控制单元

根据微程序设计思想，一条机器指令对应一个微程序。

由于任何一条机器指令的取指令操作是相同的，因此将取指令操作的命令统一变成一个微程序，这个微程序只负责将指令从主存单元中去除送至指令寄存器中，如前文3.1所示。

图6-1为微程序控制单元的基本组成，其中控制存储器（简称控存）用来存放全部微程序；CMAR是控存地址寄存器，用来存放欲读出的微指令地址；CMDR是控存数据寄存器，用来存放从控存读出的微指令；顺序逻辑用于控制微指令序列，具体为控制形成下一条微指令（即后续微指令）的地址，其输入与微指令地址形成不见（与IR相连）、微指令的下地址字段有关。

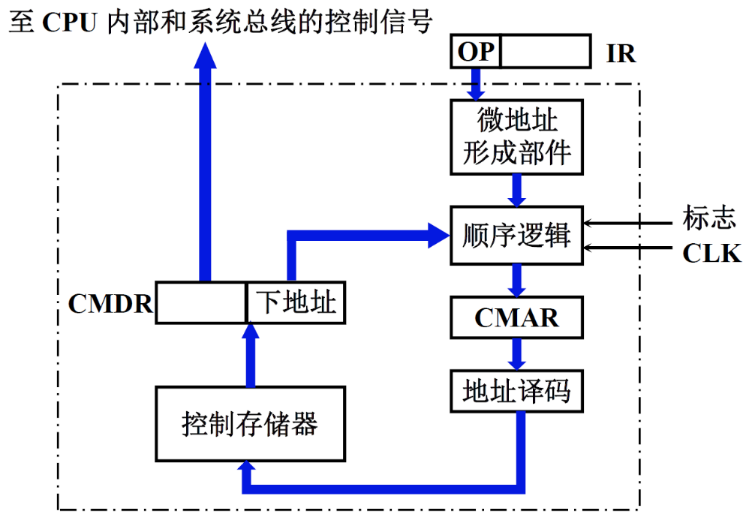


图6-1 微程序控制单元的基本组成

## 6.2微指令编码方式

由上文，共有16+2（Ad(CMDR) → CMAR和OP(IR)→微地址生成部件→CMAR）=18个微操作、40条微指令。微操作数量较少，故微指令采用直接编码方式，在操作控制字段中，按顺序每一位代表一个微操作命令，若为“1”则表示控制信号有效，为“0”则表示控制信号无效。

## 6.3微指令序列地址形成

根据上述分析，可采取由指令的操作码和微指令的下地址字段两种方式形成后续微指令的地址。

## 6.3.1根据机器指令操作码形成

当机器指令取至指令寄存器IR后，第一条微指令的地址由操作码经微地址形成部件形成，即机器指令微程序的首地址。

## 6.3.2直接由微指令下地址字段给出

微指令的顺序控制字段（下地址字段）直接指出后续微指令的地址。

## 6.4微指令字长及格式

根据直接编码方式，18个微操作对应18位操作控制字段，40条微指令对应6位顺序控制字段（下地址字段），则微指令字长至少取24位。

然而可以发现在40条微指令中有20条微指令是为了控制将后续微指令的地址打入CMAR的操作，因此实际上是每两个时钟周期才能取出并执行一条微指令。如果将CMDR的下地址字段Ad(CMDR)直接接到控制存储器的地址线上，并由下一个时钟周期的上升沿将该地址单元的内容（微指令）读到CMDR中，便能做到在一个时钟周期内读出并执行一条微指令。

因此上述操作可以省去20条微指令和2个微操作，则这五条机器指令共有16个微操作和20条微指令，故操作控制字段取16位，下地址字段取5位，其微指令格式设计如下图6-2所示。

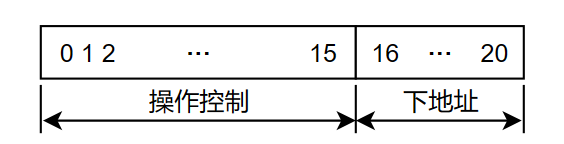


图6-2 微指令格式

其中，第0位表示控制 PC→MAR

第1位表示控制 1→R

第2位表示控制 M(MAR)→MDR

第3位表示控制 MDR→IR

第4位表示控制 (PC)+1→PC

第5位表示控制 (R(IR[21~25]))→Y

第6位表示控制 (Y)+R(IR[16~20])→Z

第7位表示控制 (Z)→R(IR[11~15])

第8位表示控制 (Y)-R(IR[16~20])→Z

第9位表示控制 (Y)+Extended(IR[0~15])→Z

第10位表示控制 (Z)→MAR

第11位表示控制 MDR→R(IR[16~20])

第12位表示控制 (R(IR[16~20]))→MDR

第13位表示控制 1→W

第14位表示控制 MDR→M(MAR)

第15位表示控制 {PC, IR[0~25]}→PC

## 6.5微指令码点

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 微程序  名称 | 微指令地址  （八进制） | 微指令（二进制码） | | | | | | | | | | | | | | | | | | | | |
| 操作控制字段 | | | | | | | | | | | | | | | | 顺序控制字段 | | | | |
|  |  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| 取指 | 01 | 1 | 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 1 |
| 02 |  |  | 1 |  | 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 1 |  |
| 03 |  |  |  | 1 |  |  |  |  |  |  |  |  |  |  |  |  | × | × | × | × | × |
| add指令 | 04 |  |  |  |  |  | 1 |  |  |  |  |  |  |  |  |  |  |  |  | 1 |  | 1 |
| 05 |  |  |  |  |  |  | 1 |  |  |  |  |  |  |  |  |  |  |  | 1 | 1 |  |
| 06 |  |  |  |  |  |  |  | 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |
| sub指令 | 07 |  |  |  |  |  | 1 |  |  |  |  |  |  |  |  |  |  |  | 1 |  |  |  |
| 10 |  |  |  |  |  |  |  |  | 1 |  |  |  |  |  |  |  |  | 1 |  |  | 1 |
| 11 |  |  |  |  |  |  |  | 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |
| lw指令 | 12 |  |  |  |  |  | 1 |  |  |  |  |  |  |  |  |  |  |  | 1 |  | 1 | 1 |
| 13 |  |  |  |  |  |  |  |  |  | 1 |  |  |  |  |  |  |  | 1 | 1 |  |  |
| 14 |  | 1 |  |  |  |  |  |  |  |  | 1 |  |  |  |  |  |  | 1 | 1 |  | 1 |
| 15 |  |  | 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 1 | 1 | 1 |  |
| 16 |  |  |  |  |  |  |  |  |  |  |  | 1 |  |  |  |  |  |  |  |  |  |
| sw指令 | 17 |  |  |  |  |  | 1 |  |  |  |  |  |  |  |  |  |  | 1 |  |  |  |  |
| 20 |  |  |  |  |  |  |  |  |  | 1 |  |  |  |  |  |  | 1 |  |  |  | 1 |
| 21 |  |  |  |  |  |  |  |  |  |  | 1 |  | 1 | 1 |  |  | 1 |  |  | 1 |  |
| 22 |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 1 |  |  |  |  |  |  |
| j指令 | 23 |  |  |  |  |  | 1 |  |  |  |  |  |  |  |  |  |  | 1 |  | 1 |  |  |
| 24 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 1 |  |  |  |  |  |

# 七、总结

此次项目设计了一个简单的基于MIPS指令系统的RISC处理器，包含2个R型指令——add指令和sub指令，2个I型指令——lw指令和sw指令，1个J型指令——j指令，以及32个32位通用寄存器R0~R31。该处理器使用微程序设计方式，共有16个微操作和20条微指令。