# CPU

CPU由
运算器（对数据加工：累加器ACC、乘商寄存器MQ、通用操作数寄存器X、算术逻辑单元ALU）和
控制器（协调控制计算机各部件执行程序的指令序列：控制单元CU、指令寄存器IR、程序计数器PC）组成。

控制器的基本功能包括：取指令、分析指令、执行指令、中断处理

一条指令会经过：取指周期FE，间址周期IND，执行周期EX，中断周期INT

**CPU的功能**有：
指令控制（程序顺序的控制），
操作控制（管理并产生由内存取出的每条指令的操作信号，把并送往对应部件并控制其按指令要求运行），
时间控制（为每条指令按时间允许提供应有的控制信号），
数据加工（读数据进行算数和逻辑运算），
中断处理（对异常清理和特殊请求进行处理）。

## 运算器的基本结构

算术逻辑单元：进行算术/逻辑运算。

通用寄存器组（AX、BX、CX、，R0、R1，SP等）：存放操作数。SP是堆栈指针，用于指示栈顶地址。

暂存寄存器：暂存从主存读来的数据，此数据不存放在通用寄存器（否则会破坏原有内容）。

程序状态字寄存器PSW：保留由算数逻辑运算指令或测试指令的结果而建立的各种状态信息。

移位器：对运算结果进行移位计算。

计数器：控制乘除法运算。

### 寄存器输入/输出到ALU

`ALU <= [R0 R1 R2 R3]`，如果直接用导线连接，相当于多个寄存器同时一直向ALU传输数据。解决方法：

1. 使用多路选择器：`ALU <=MUX= [R0 R1 R2 R3]`，控制信号选择一路输出。
2. 使用三态门：`ALU <=◁= [R0 R1 R2 R3]`，控制每一路是否输出。
3. CPU内部总线：`ALU <=总线= [R0 R1 R2 R3]`，将所有寄存器的IO端接到一条公共通路。并引入暂存寄存器、累加寄存器、PSW。

1、2方案性能高，寄存不存在数据冲突，但结构复杂，硬件量大，不易实现；
3方案解构简单、易实现，数据传输存在较多冲突，性能低。

## 控制器的基本结构

程序计数器PC：指明下条指令在主存中存放的地址，有自增功能。

指令寄存器IR：保存当前正在执行指令。

指令译码器：对操作码字段进行译码，想控制器提供特定的操作信号。

微操作信号发生器：根据IR内容（指令）、PSW内容（状态信息），产生控制整个计算机系统所需的各种控制信号。

时钟系统：产生时序信号，由统一时钟CLOCK分频得到。

存储器地址寄存器MAR：存放所要访问的主存单元的地址。

存储器数据寄存器MDR：存放向驻村写入的信息或从主存中读出的信息。

# 指令执行过程

指令周期、数据流、指令执行方案。

## 指令周期

指令周期是CPU从主存中每取出并执行一条指令的全部时间。
通常用若干机器周期（CPU周期）来表示。
一个机器周期包含若干时钟周期（CPU的最基本单位）。

**一个指令周期的完成内容**：
开始-》取指令`PC+1`-》对指令译码-》执行指令-》取下条指令`PC+1`
------<=======取指周期=======><=执行周期=>------------

指令周期内的机器周期数可以不等，每个机器内的节拍数也可以不等。

**指令周期的流程**：

- *取指周期*
- 有间接地址吗?
  - 是：间址周期
  - 否：执行周期
- 有中断吗?
  - 是：中断周期
  - 否：继续回到*取指周期*

## 指令周期的数据流

**取指周期**：

1. 当前指令地址送至存储器地址寄存器：`(PC)->MAR`
2. CU发出信号，经控制总线传到主存（读信号）：`1->R`
3. 将MAR所致主存中的内容经数据总线送入MDR：`M(MAR)->MDR`
4. 将MDR中的内容（指令）送入IR：`(MDR)->IR`
5. CU发出控制信号，形成下一条指令地址：`(PC)+"1"->PC`

**间址周期**：

1. 指令的地址码送入MAR：`Ad(IR)->MAR`或`Ad(MDR)->MAR`
2. CU发出控制信号，启动主存做读操作：`1->R`
3. MAR所指主存中的内容经数据总线送入MDR：`M(MAR)->MDR`
4. 将有效地址送到指令的地址码字段：`(MDR)->Ad(IR)`

**执行周期**:
不同指令的执行周期操作不同，没有统一的数据流向。

**中断周期**:
暂停当前任务去完成其他任务。为了可以回复当前任务，需要保存断点。一般用堆栈来保存断点，SP只想站定元素，进栈操作先修改指针，再存入数据。

1. CU控制将SP减1，修改后的地址送入MAR：`(SP)-1 -> SP, (SP)->MAR`
2. CU发出控制信号，启动主存做写操作：`1->W`
3. 将断点（PC内容）送入MDR：`(PC)->MDR`
4. CU控制将中断服务程序的入口地址（由向量地址形成部件产生）送入PC：`向量地址->PC`

### 指令执行方案

**单指令周期**:
所有指令的执行时间是相同的，指令周期取决于执行时间最长的指令的执行时间；
指令间串行运行。
*但是*本可以较短时间内完成的指令需要用最长指令周期的时间完成，降低整个系统的运行速度。

**多指令周期**:
不同指令用不同步骤执行，可以选用不同个数的时钟周期来完成不同指令间的执行过程；
指令见串行执行。
*但是*需要更复杂的硬件设计。

**流水线方案**:
每个始终周期启动一条指令，尽可能让多条指令同时运行，但各自在不同的执行步骤；
指令之间并行执行。

## 数据通路

数据通路是数据在功能部件之间传送的路径。其基本结构包括有：

1. CPU内部单总线方式
2. CPU内部多总线方式
3. 专用数据通路方式

### CPU内部单主线

**内部总线**是一个部件内部连接个寄存器以及运算部件之间的总线（例如CPU）；
**系统总线**是一台计算机系统各部件和各类I/O接口间互相联通的总吸纳。

**寄存器之间数据传送**:
e.g. 将PC送至MAR，实现传送操作的流程以及控制信号为：

```
(PC) -> Bus    # PCout有效，PC内容送总线
Bus -> MAR     # MARin有效，总线内容送MAR
# 可以写为：(PC) -> Bus -> MAR
```

**主存与CPU之间的数据传送**:
eg. CPU从主存读取指令，实现传送操作的历程以及控制信号为:

```
(PC) -> Bust -> MAR   # PCout和MARin有效，现行指令地址->MAR
1->R                  # CU发出读命令
MEM(MAR) -> MDR       # MDRin有效
MDR -> Bus -> IR      # MDRout和IRin有效，现行指令->IR
```

**算术运算或逻辑运算的数据传送**:
eg. 一条假发指令，未操作序列以及控制信号为:

```
Ad(IR) -> Bus -> MAR    # MDRout和MARin有效
1 -> R                  # CU发读命令
MEM(MAR) -> 数据线 -> MDR   # MDRin有效
MDR -> Bus -> Y     # MDRout和Yin有效，操作数->Y
(ACC) + (Y) -> Z    # ACCout和ALUin有效，CU向ALU发送加命令
Z -> ACC            # Zout和ACCin有效，结果->ACC
```

**例题**：以指令`ADD (R0), R1`的流程和控制信号为例

功能：`((R0))+(R1) -> (R0)`,
取指周期、间址周期、执行周期

各阶段指令流程:

取指周期：公共操作

| 时序 | 微操作 | 有效控制信号 |
| --- | --- | --- |
| 1 | (PC)->MAR | PCout, MARin |
| 2 | M(MAR)->MDR | MemR, MARout, MDRinE |
| 3 | (MDR)->IR | MDRout, IRin |
| 4 | 指令译码 | - |
| 5 | (PC)+1 -> PC | - |

间址周期：完成取数操作，被加数在主存中，加数已经在寄存器R1中

| 时序 | 微操作 | 有效控制信号 |
| --- | --- | --- |
| 1 | (R0)->MAR | R0out, MARin |
| 2 | M(MAR)->MDR | MemR, MARout, MDrinE |
| 3 | (MDR)->Y | MDRout, Yin |

执行周期：完成取数操作，被加数在主存中，加数已经放在寄存器R1中

| 时序 | 微操作 | 有效控制信号 |
| --- | --- | --- |
| 1 | (R1)+(Y) -> Z | R1out, ALUin, CU向ALU发ADD控制信号 |
| 2 | (Z)->MDR | Zout, MDRin |
| 3 | (MDR)->M(MAR) | MemW, MDRoutE, MARout |

### 专用通路结构

效率更多，成本更高，借助多路选择器与三态门。

**取指周期**:

```
(PC) -> MAR     # PCout, MARin
(MAR) -> 主存   # MARout
1 -> R          # 控制单元箱主存发送读命令
M(MAR) -> MDR   # MDRin
(MDR) -> IR     # MDRout, IRin
(PC)+1 -> PC
# (如果有控制信号): Op(IR) -> CU    # CUin
```

## 控制器设计

| - | 微程序控制器 | 硬布线控制器 |
| --- | --- | --- |
| 工作原理 | 微操作控制信号以微程序形式存储在控制存储器中，<br>执行指令时读出即可 | 微操作控制信号由组合逻辑电路<br>根据当前的指令码、状态和时序，即时产生 |
| 执行速度 | 慢 | 快 |
| 规整性 | 较规整 | 繁琐、不规整 |
| 应用场合 | CISC | RISC |
| 易扩充性 | 易扩充修改 | 困难 |

### 硬布线控制器

根据指令操作码、目前的机器周期、节拍信号、机器状态条件，即可确定现在这个接拍下应该发出哪些“微命令”。

控制单元CU会接收来自节拍发生器的信号T，并且可以用与门来设计电路，当T和指令信号同时为True时，则会发出对应微命令C。那么对于微操作`(PC) -> MAR, C = FE·T`。

**设计步骤**:

1. 分析每个阶段的微操作序列（取值、间址、执行、中断）
2. 选择CPU控制方式
    - 选择定长或不定长
3. 安排微操作时序
    - 先后顺序不得随意更改
    - 对于不同的被控对象，其微操作尽量在一个节拍完成
    - 时间较短的微操作，一个节拍内完成，并允许有先后顺序
4. 设计电路
    - 列出操作时间表
    - 写出微操作命令的最简表达式
    - 画出逻辑图

**特点**:
指令越多，设计和实现越复杂，因此一般用于RISC；
若扩充一条新指令，则控制器设计就要大改，扩充指令较困难。
由于使用纯硬件实现控制，执行速度快。未操作控制信号由组合逻辑电路即时产生。

## 微程序控制器

在微程序控制器中，微程序由微指令序列组成，*每种指令对应一个微程序*。

指令是对程序执行步骤的描述；
微指令是对指令执行步骤的描述。

若干微指令构成指令；若干指令构成程序。

**CU结构**:

1. 微地址形成部件
    - 微地址即微指令在CM中存放地址
    - 通过指令操作码形成对应为程序的第一条微指令的存放地址
2. 顺序逻辑
    - 根据机器标志和时序信息确定下一条微指令的存放地址
3. CMAR（μPC）
    - 明确接下来要执行的微指令的存放地址
4. 地址译码器
    - 将CMAR内的地址信息译码为电信号，控制CM读出微指令
5. 控制存储器CM
    - 存放所有机器指令对应的微程序（微指令序列）
    - 用ROM实现，按地址寻访。通常在CPU出厂时就把所有微程序写入。
6. CMDR（μIR）
    - 微指令寄存器，用于存放当前要执行的微指令。`CM(μPC)->μIR`

**微程序控制器的工作原理**:

- 指令周期 = 取指周期 -> 间址周期 -> 执行周期 -> 中断周期
- 取值周期、间址周期、中断周期的微指令序列通常是公用的；执行周期的微指令序列各不相同
- 取指周期的微指令序列固定从#开始存放；执行后期的微指令序列的存放根据指令操作码确定

微指令控制器解构：

- IR
- ->微地址形成部件
- ->顺序逻辑
    - <-标志
    - <-CLK
- ->CMAR
- ->地址译码
- ->控制存储器CM
- ->CMDR
    - 下地址->顺序逻辑
- ->CPU内部和系统总线的控制信号

## 微指令的设计

微命令按并行执行与否可分为：相容性微指令，互斥性微指令

有三种格式的微指令：

1. **水平型**：一条微指令定义多个可并行微指令
    - |<-操作控制->|<-顺序控制->|
    - 微程序短，执行速度快；
    - 但微指令长，编写微程序较麻烦
2. **垂直型**：一条微指令只定义一个微命令，由操作码字段规定具体功能
    - |<-微操作码->|<-目的地址->|<-源地址->|
    - 微指令短，简单规整，便于编写；
    - 但微程序长，执行速度慢，工作效率低
3. **混合型**：在垂直型的基础上增加一些不太复杂的并行操作。
    - 微指令较短，便于编写
    - 微程序补偿，执行速度快

**编码方式**：

1. **直接编码**：在微指令的操作控制字段中，每一位代表一个微操作命令。
    - 简单直观速度快，操作并行性好；
    - 但微指令字长过长，n个微指令就要微指令的操作字段由n位，控存容量极大。
2. **字段直接编码**：将微指令的控制字段分成若干"段"，每段经译码后发出控制信号。
    - 互斥性微指令分在同一段内，相容性的分在不同段；
    - 每个小段信息为不能太多；
    - 每个小段还要留出一个状态，用000表示不操作。
    - 可以缩短微指令字长；但要通过译码电路后在发出微命令，比直接编码慢。
3. **字段间接编码**：一个字段中某些微命令由另一个字段中的某些微命令来解释。
    - 可进一步缩短微命令字长；
    - 但削弱了微指令的并行控制能力，故通常作为字段直接编码的方式的一种辅助手段。

**微指令地址形成方式**：

1. 微指令的下地址字段指出：微指令格式中设置下一个地址字段，由微指令的下地址字段直接指出后续为指令单地址，这种方式又被称为**断定法**
2. 根据机器指令的操作码形成

3. 增量**计数器法**(CMAR)+1 -> CMAR
4. 分支转移：指明判别条件（转移方式），指明转移成功后的去向（转移地址）
5. 通过测试网络（顺序逻辑）
6. 由硬件产生微程序入口地址

## 微程序控制单元的设计

步骤：

1. 分析每个阶段的微操作序列
2. 写出对应机器指令的微操作命令以及节拍安排
    1. 写出每个周期所需要的微操作（参照硬布线）
    2. 补充微程序控制器特有的微操作：
        1. 取指周期：
           Ad(CMDR -> CMAR)
           OP(IR) -> 微地址形成部件 -> CMAR
        2. 执行周期：
           Ad(CMDR) -> CMAR
3. 确定微指令格式
    - 根据微操作个数决定采用何种编码方式，以确定微指令的操作控制字段的位数。
    - 根据CM中存储的微指令总数，确定微指令的顺序控制字段的位数。
    - 最后按操作控制字段位数和顺序控制字段位数就可以确定微指令字长。
4. 编写微指令码点
    - 根据操作控制字段的每一位代表的微操作命令，编写每一条微指令的码点。

**微程序设计的分类**：

1. 静态微程序设计
    - 程序无需改变，采用ROM存储
2. 动态微程序设计
    - 通过改变微指令和微程序改变机器指令，利于仿真，采用EPROM