第 2 章　JUC-II 微程序控制计算机

2.1 概述

本章介绍一个教学模型计算机的设计，是在作者设计的前一个版本的模型机基础上的改进和优化，为方便表达，本书将其命名为 OpenJUC-II。它的字长为 16 位，具有 38 条常用指

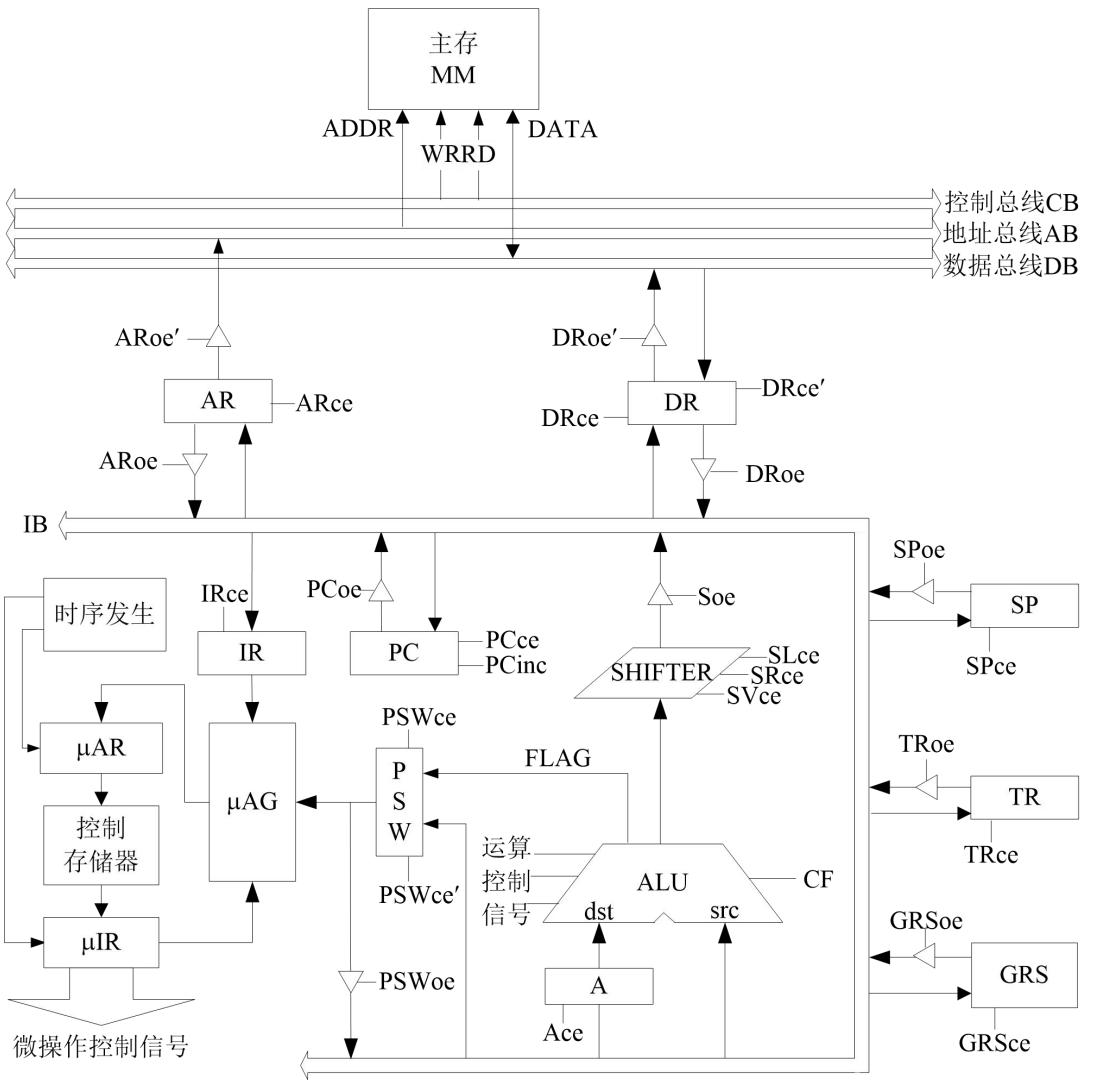


图 2.1 OpenJUC-II 教学模型机的数据通路

令、8 种基本寻址方式，采用微程序控制方式，主存寻址空间为 64K 字，外设与主存统一编址，采用向量中断机制，并且内置片上调试器以支持在线调试。该模型机已经在 FPGA 上实现，能够以 10MHz 的主频运行。

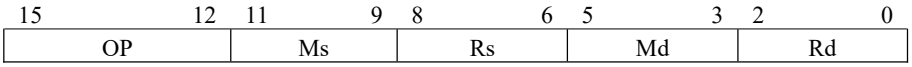
OpenJUC-II 模型计算机的数据通路如图 2.1。CPU 的字长为 16 位，包括运算器和控制器两个部分，各个部件通过 16 位内部总线 IB 相连；系统总线采用单总线结构，包括 16 位的数据总线 DB、16 位的地址总线 AB 和控制总线 CB。CPU 内部总线 IB 与系统总线之间通过 DR、AR 相联。

2.2 指令系统设计

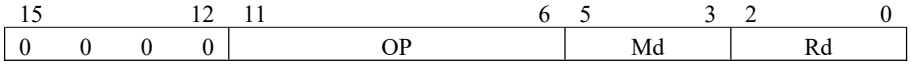
指令系统包括数据传送类指令、算术逻辑运算指令、移位指令、转移指令、子程序调用返回指令、输入输出指令等。在寻址方式上采用最典型的寻址方式，有立即寻址、直接寻址、间接寻址、寄存器寻址、寄存器间接寻址、寄存器自增间接寻址、变址寻址、相对寻址等 8 种寻址方式。指令操作的数据宽度是字，不能按字节操作。

2.2.1 指令格式及寻址方式

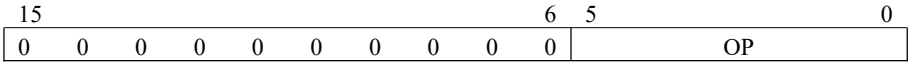
模型机的指令格式规整，所有的指令都可使用各种寻址方式（个别有限制的除外），所有的寄存器和存储单元都可同等对待。按照操作数个数的不同有三种指令格式：双操作数指令、单操作数指令和无操作数指令，如图 2.2 所示。



(a) 双操作数指令格式



(b) 单操作数指令格式



(c) 无操作数指令格式

图 2.2 指令格式

指令系统采用操作码扩展技术，当双操作数指令的操作码部分（$IR_{15\sim12}$）全为 0 时，表示扩展为单操作数指令，将双操作数指令的源操作数部分（$IR_{11\sim6}$）用作单操作数指令的操作码。扩展无操作数指令时，将 $IR_{15\sim6}$ 全为 0 作为无操作数指令的标志，最低 6 位用作指令操作码。

指令格式中的 Ms 代表源操作数的寻址方式，Md 代表目的操作数的寻址方式；Rs 和 Rd 分别表示的是源操作数和目的操作数的寄存器号。寻址方式的编码见表 2.1。除了立即寻址不应作为目的寻址方式外，目的操作数和源操作数具有相同的寻址方式。

表 2.1 寻址方式编码表

寻址方式	助记符	编码 M
寄存器寻址	Rn	000
寄存器间接寻址	(Rn)	001
寄存器自增间接寻址	(Rn)+	010
立即寻址	#imm	011
直接寻址	addr	100
间接寻址	(addr)	101
变址寻址	disp(Rn)	110
相对寻址	disp(PC)	111

当寻址方式为表 2.1 中的后面 5 种时，即 M=011~111 时，指令中还必须包含表示立即数、地址或偏移量的常数。常数的宽度均为一个字。因此根据目的操作数与源操作数寻址方式的不同，指令的总长度可能为单字、双字或三字。如果源和目的地址码均不包含常数，指令字长为一个字；如果源和目的地址码中有一个包含常数；指令字长为二个字；如果源和目的地址码中都包含常数；指令字长为三个字；如图 2.3 所示。

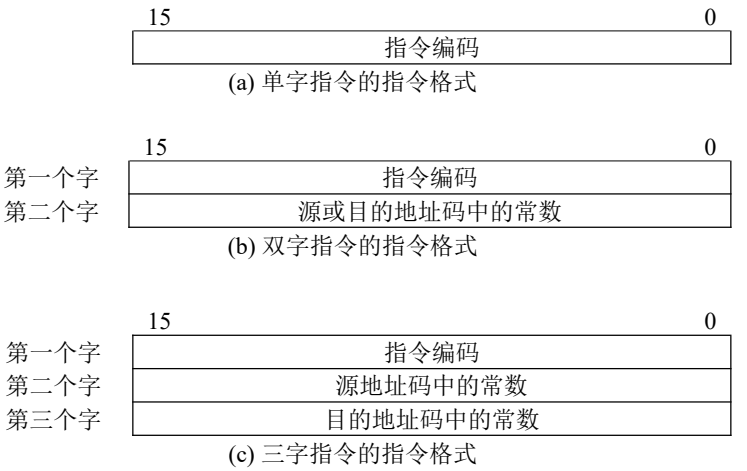


图 2.3 单字、双字和三字指令

2.2.2 指令类型

所有指令的指令编码见表 2.2。从指令功能上可分为以下几类。

1. 数据传送类指令

数据传送类指令有传送指令 MOV，入栈指令 PUSH、出栈指令 POP。

2. 算术逻辑运算指令

双操作数运算指令有加法指令 ADD、带进位的加法指令 ADDC、减法指令 SUB、带借位的减法指令 SUBB、比较指令 CMP，逻辑与指令 AND、逻辑或 OR、逻辑异或指令 XOR，

逻辑测试指令 TEST。

单操作数运算指令有加 1 指令 INC、减 1 指令 DEC、逻辑反指令 NOT。

表 2.2 指令编码表

指令助记符		指令编码																影响 PSW			
		F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0	S	Z	O	C
MOV	src, dst	0	0	0	1	源地址码						目的地址码						–	–	–	–
ADD	src, dst	0	0	1	0	源地址码						目的地址码						√	√	√	√
ADDC	src, dst	0	0	1	1	源地址码						目的地址码						√	√	√	√
SUB	src, dst	0	1	0	0	源地址码						目的地址码						√	√	√	√
SUBB	src, dst	0	1	0	1	源地址码						目的地址码						√	√	√	√
AND	src, dst	0	1	1	0	源地址码						目的地址码						√	√	×	×
OR	src, dst	0	1	1	1	源地址码						目的地址码						√	√	×	×
XOR	src, dst	1	0	0	0	源地址码						目的地址码						√	√	×	×
CMP	src, dst	1	0	0	1	源地址码						目的地址码						√	√	√	√
TEST	src, dst	1	0	1	0	源地址码						目的地址码						√	√	×	×
SAR	dst	0	0	0	0	0	0	0	0	0	1	目的地址码						×	×	×	√
SHL	dst	0	0	0	0	0	0	0	0	1	0	目的地址码						×	×	×	√
SHR	dst	0	0	0	0	0	0	0	0	1	1	目的地址码						×	×	×	√
ROL	dst	0	0	0	0	0	0	0	1	0	0	目的地址码						×	×	×	√
ROR	dst	0	0	0	0	0	0	0	1	0	1	目的地址码						×	×	×	√
RCL	dst	0	0	0	0	0	0	0	1	1	0	目的地址码						×	×	×	√
RCR	dst	0	0	0	0	0	0	0	1	1	1	目的地址码						×	×	×	√
JC	dst	0	0	0	0	0	0	1	0	0	0	目的地址码						–	–	–	–
JNC	dst	0	0	0	0	0	0	1	0	0	1	目的地址码						–	–	–	–
JO	dst	0	0	0	0	0	0	1	0	1	0	目的地址码						–	–	–	–
JNO	dst	0	0	0	0	0	0	1	0	1	1	目的地址码						–	–	–	–
JZ	dst	0	0	0	0	0	0	1	1	0	0	目的地址码						–	–	–	–
JNZ	dst	0	0	0	0	0	0	1	1	0	1	目的地址码						–	–	–	–
JS	dst	0	0	0	0	0	0	1	1	1	0	目的地址码						–	–	–	–
JNS	dst	0	0	0	0	0	0	1	1	1	1	目的地址码						–	–	–	–
JMP	dst	0	0	0	0	0	1	0	0	0	0	目的地址码						–	–	–	–
INC	dst	0	0	0	0	0	1	0	0	0	1	目的地址码						√	√	√	√
DEC	dst	0	0	0	0	0	1	0	0	1	0	目的地址码						√	√	√	√
NOT	dst	0	0	0	0	0	1	0	0	1	1	目的地址码						√	√	×	×
PUSH	dst	0	0	0	0	0	1	1	0	0	0	目的地址码						–	–	–	–
POP	dst	0	0	0	0	0	1	1	0	0	1	目的地址码						–	–	–	–
CALL	dst	0	0	0	0	0	1	1	0	1	0	目的地址码						–	–	–	–
HALT		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	–	–	–	–
NOP		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	–	–	–	–
RET		0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	–	–	–	–
RETI		0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	–	–	–	–
EI		0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	–	–	–	–
DI		0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	–	–	–	–

注：√ 表示指令设置 PSW 的该标志位；– 表示不影响；× 表示会影响、但没有意义

3. 移位指令

SAR：算术右移；

SHL、SHR：逻辑左移、右移；

ROL、ROR：循环左移、右移；

RCL、RCR：带进位的循环左移、右移；

4. 转移指令

这类指令有无条件转移指令 JMP，条件转移指令 JC、JNC、JO、JNO、JZ、JNZ、JS、JNS。条件转移指令测试 PSW 中的 SZOC 标志位，符合条件则程序转移，否则顺序执行。例如，JC 指令测试 CF 标志位，若 CF=1 转移，否则顺序执行；JNC 指令同样测试 CF 标志位，不同的是若 CF=0 转移，否则顺序执行。

5. 子程序和中断控制指令

包括子程序调用指令 CALL，子程序返回指令 RET；开中断指令 EI，关中断指令 DI，中断返回指令 RETI。

6. 其他指令

空操作指令 NOP，停机指令 HALT。停机指令主要用于模型机的调试。

2.3 运算器设计

2.3.1 多功能加减运算电路

根据指令系统的设计，算术运算指令包括加法（ADD）、带进位的加法（ADDC）、减法（SUB）、带借位的减法（SUBB）、加 1（INC）、减 1（DEC），为了实现这些运算指令的功能，设计多功能加减运算电路，如图 2.4。

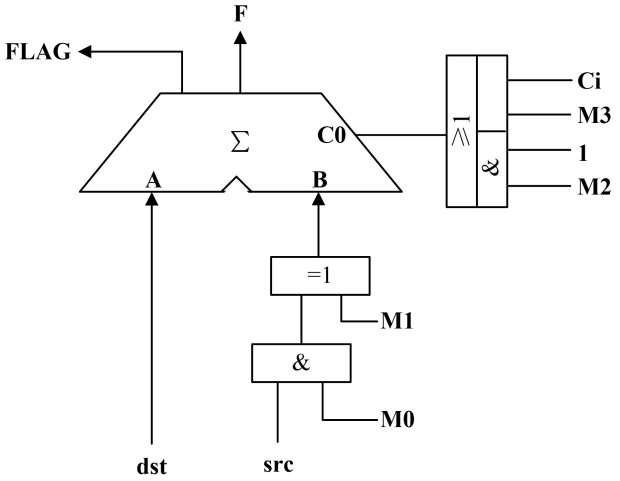


图 2.4 多功能加减运算电路

加法器的 C0 和 B 输入端都有数据选择，用运算控制信号选择输入数据。B 输入端数据选择用来选择源操作数，与门用来产生数据“0”，异或门用来实现可控取反。B 输入选择的

逻辑方程为

$$B = (src \cdot M0) \oplus M1$$

C0 数据选择用与或门实现，当进行带进位的加法或者带借位的减法运算时，选择 CF 送给 C0 输入端；当进行减法运算或加 1 运算时，将“1”送给 C0；其他运算时 C0 为 0。C0 输入选择的逻辑方程为

$$C0 = M2 + M3 \cdot Ci$$

通过控制 M0~M3 实现各种不同的运算。下面分析各种运算功能的实现。

（1）加法运算

运算控制信号 M3=0, M2=0, M1=0, M0=1，

则　B = src，C0 = 0

所以，F = A + B + C0 = dst + src

（2）减法运算

M3=0, M2=1, M1=1, M0=1，

则　$B = \overline{src}$，C0 = 1

所以，$F = dst + \overline{src} + 1 = dst - src$

（3）带进位的加法

M3=1, M2=0, M1=0, M0=1，

则　B = src，C0 = Ci

所以，F = A + B + C0 = dst + src + Ci

带进位的加法通常用于双倍字长的加法运算，用 n 位的加法器完成 2n 位的加法。先用 ADD 指令进行低 n 位的加法，进位存储在 PSW 中；然后用 ADDC 指令进行高 n 位的加法，同时把 PSW 中的 CF 标志位作为低位的进位 Ci 加进来。

（4）带借位的减法

M3=1, M2=0, M1=1, M0=1，

则　$B = \overline{src}$，C0 = Ci

所以，$F = dst + \overline{src} + Ci$

下面说明上式完成的就是带借位的减法运算。前面已经提到，减法运算有借位时 CF=0、没有借位时 CF=1，和常规的含义正好相反；假设以符号 borrow 表示真正含义的借位，那么上式可以表示为

$$F = dst + \overline{src} + CF = dst + \overline{src} + \overline{borrow} = dst + \overline{src} + 1 - borrow = dst - src - borrow$$

即带借位的减法。

（5）加 1

M3=0, M2=1, M1=0, M0=0

则　B = 0，C0 = 1

所以，F = dst + 1

（6）减 1

M3=0, M2=0, M1=1, M0=0

则　B = –1（即 111…1），C0 = 0

所以，F = dst – 1

（7）传送

M3=0, M2=0, M1=0, M0=0

则　B = 0，C0 = 0

所以，F = dst

这是多功能加减运算电路一个比较特别的功能，当 M3~M0 控制信号全为 0 时，dst 输入端的数据传送到 F 输出端。在某些指令的执行过程中会用到这个特性。

表 2.3 给出了多功能加减运算电路完整的功能列表。

表 2.3 多功能加减运算电路功能表

M3	M2	M1	M0	运算功能	对应指令
0	0	0	0	F=dst	无
0	0	0	1	F=dst+src	ADD
0	0	1	0	F=dst–1	DEC
0	0	1	1	$F=dst+\overline{src}$	无
0	1	0	0	F=dst+1	INC
0	1	0	1	F=dst+src+1	无
0	1	1	0	F=dst	无
0	1	1	1	F=dst–src	SUB
1	0	0	1	F=dst+src+Ci	ADDC
1	0	1	0	F=dst+Ci –1	无
1	0	1	1	$F=dst+\overline{src}+Ci$	SUBB
1	1	0	0	F=dst+Ci+1	无
1	1	0	1	F=dst+src+Ci+1	无
1	1	1	0	F=dst+Ci	无
1	1	1	1	F=dst–src+Ci	无

从该表很容易得出指令与 M0~M3 的逻辑关系如下。

M0　= ADD + SUB + ADDC + SUBB

M1 = DEC + SUB +SUBB

M2 = INC + SUB

M3 = ADDC + SUBB

2.3.2　运算结果的特征标志

运算器中需要将加法运算结果的一些特征保存下来，以便后续程序中使用，常见的特征标志有：

SF（Sign Flag）：符号标志。SF＝1 表示结果为负数，SF＝0 表示结果为正数；

ZF（Zero Flag）：零标志。ZF＝1 表示结果为零，ZF＝0 表示结果非零；

OF（Overflow Flag）：溢出标志。OF＝1 表示结果溢出，OF＝0 表示结果不溢出；

CF（Carry Flag）：进位标志。CF＝1 表示有进位，CF＝0 表示没有进位。

从前面已经知道，减法是转换成加法进行运算的，所以上面的特征标志都是针对补码加

法而言。溢出是指有符号补码加法溢出，如果是无符号数加法，OF 标志是没有意义的。CF 标志也是针对加法而言，如果是减法运算，CF＝0 表示有借位，CF＝1 表示没有借位，正好和加法运算相反。

这些特征标志通常存储在 CPU 的一个专用寄存器中，这个寄存器称为标志寄存器（FLAG）或程序状态字（PSW，Program Status Word）。除了上面的四个标志位外，CPU 的其他一些运行状态如允许中断标志也存放在 PSW 中。这些状态会影响后续程序的执行，属于控制信息，所以 PSW 通常被归类到控制器，但是其中的运算结果特征是由运算器产生的。

2.3.3 算术逻辑单元 ALU 设计

运算器除了完成算术运算，还要完成逻辑运算。根据指令系统的设计，逻辑运算指令包括逻辑与（AND）、或（OR）、非（NOT）、异或（XOR）。实现算术运算和逻辑运算的电路称为算术逻辑单元（ALU，Arithmetic Logic Unit）。

2.3.4 移位寄存器设计

移位操作分逻辑移位、算术移位和循环移位三大类，其中循环移位根据进位位是否一起参加循环，可分为不带进位循环移位和带进位循环移位两种。移位操作如图 2.5。

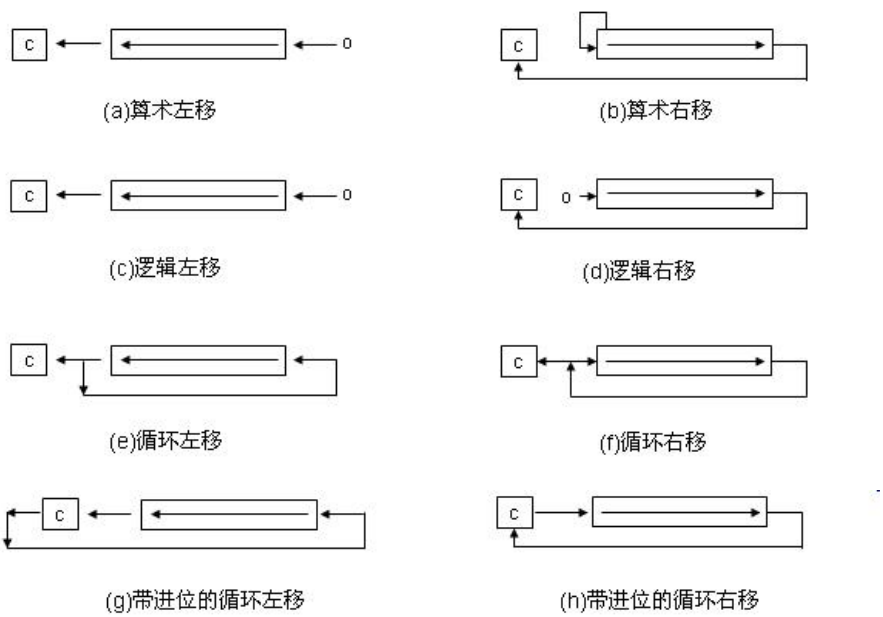


图 2.5 移位操作

从图 2.5 可以看出，除了左移和右移的不同，不同移位操作的区别主要是移入的数据来源不同。如果 16 位移位数据是 d，左移时移入最低位 d[0]的数据根据不同的的移位操作可取 0、最高位 d[15]或 PSW 中的 CF；右移时移入最高位 d[15]的数据根据不同的的移位操作可取 d[15] 自身、0、最低位 d[0]或 CF。因此可以用两个多路器选择左移和右移时的移入数据，

如图 2.6。

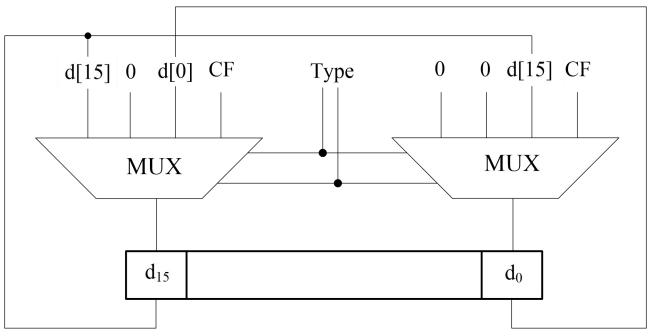


图 2.6 左移和右移移入数据的选择

图中多路器的选择信号 Type 连接指令寄存器 IR 的 8、7 两位，从表 2.2 指令编码表可以看出，当 $IR_{8\sim7}$ 分别为 00、01、10、11 时，对应的移位指令分别是：算术移位、逻辑移位、循环移位和带进位的循环移位。特别需要说明，和常规的移位寄存器不同，这里的移位寄存器不是对存储在寄存器中的内容进行移位，而是对输入数据进行移位。OpenJUC-II 的移位指令每次只移 1 位，对输入数据移位可以减少一次操作，不必先存储、再移位；如果希望移位指令可以指定移位的位数，那么就应该设计为对移位寄存器的内容进行移位。

2.3.5 运算器数据通路

运算器数据通路如图 2.7。

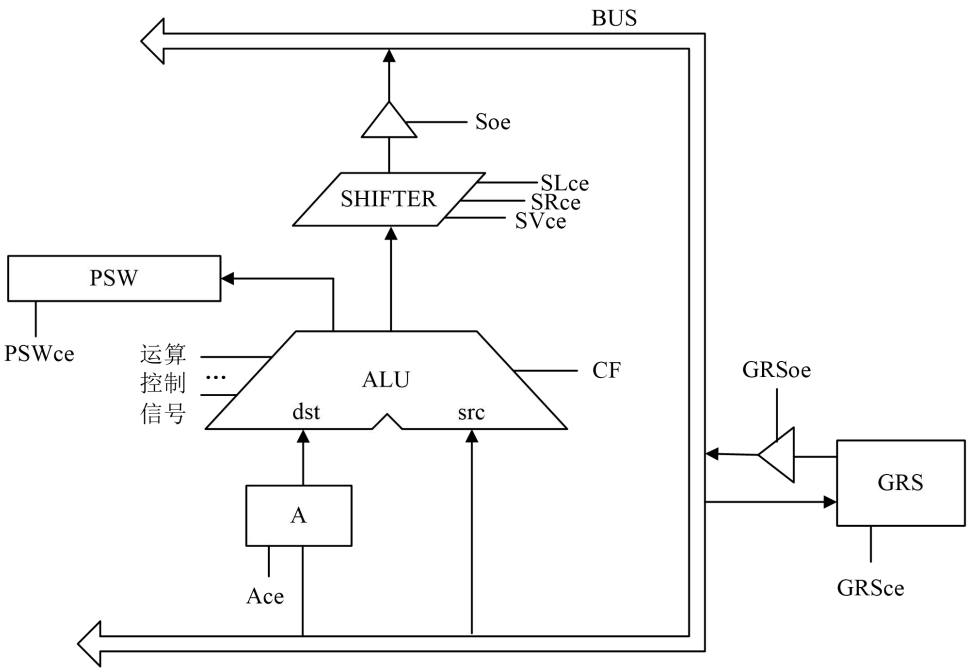


图 2.7 运算器数据通路

ALU 和移位寄存器的设计前面已经介绍，数据通路上还有几个模块：通用寄存器、A 暂存器、TR 暂存器和程序状态字 PSW。通用寄存器一共有 8 个，称为通用寄存器组；TR 用于暂存源操作数，A 用于暂存目的操作数，PSW 用来保存 ALU 运算结果的特征标志，即负标志 SF、零标志 ZF、溢出标志 OF 和进位标志 CF。它们的逻辑功能相同，都可以用数据寄存器来实现。除了功能部件，向总线输出的逻辑部件还要经过一个三态缓冲器。

下面做几点说明。

（1）移位类型与指令编码的关系

前已述及，算术移位、逻辑移位、循环移位和带进位的循环移位四种移位类型由端口 Type 选择，分别对应 00、01、10、11 四个编码。在硬件设计中，Type 来自指令寄存器 IR 的第 8 和第 7 位；对照表 2.2 指令编码表，这两位恰好可以用来区分四种不同的移位。由此可见指令编码的设计往往与硬件的设计有密切的关系。

（2）PSW 的 CF 标志位

从前面已经知道，PSW 的 SF、ZF、OF、CF 是 ALU 的运算结果特征标志，但是 CF 有些特殊，它还受移位操作的影响。从图 2.5 可以看出，所有的移位操作的移出数据都是送到 PSW 的 CF 标志位。也就是说，影响 CF 的有两个来源，一个源于 ALU 的运算结果，一个源于移位寄存器的移位输出。由图 2.5 可以看出，左移时最高位送 CF，右移时最低位送 CF。选择逻辑如图 2.8，SLce 有效时，选择移位寄存器的 d[15]送入 CF；SRce 有效时，选择移位寄存器的 d[0]送入 CF；SLce 和 SRce 都无效时，选择 ALU 产生的进位 Cout 送入 CF；SLce 和 SRce 不应该出现同时有效的情况，如果有这种情况出现，应保持 CF 不变，即 CF 送入 CF。

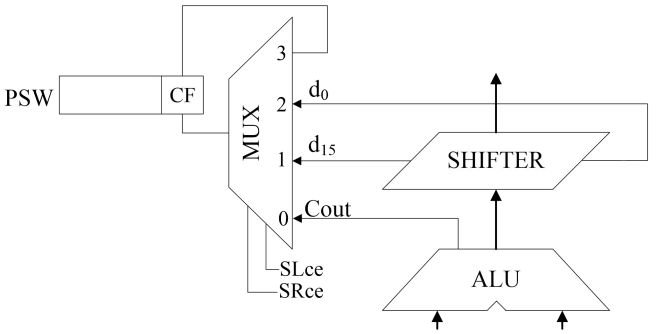


图 2.8 CF 选择逻辑

（3）通用寄存器组的地址

通用寄存器组的地址，也就是寄存器号，也是来自于指令寄存器。根据指令格式（见图 2.2），源寄存器号由 $IR_{8\sim6}$ 给出，目的寄存器号由 $IR_{2\sim0}$ 给出，在硬件设计中，它们由 SOF 信号选择。SOF 是控制器产生的一个信号，它表示当前是否处于取源操作数阶段。

2.4 微程序控制器设计

2.4.1 微程序控制器的基本组成

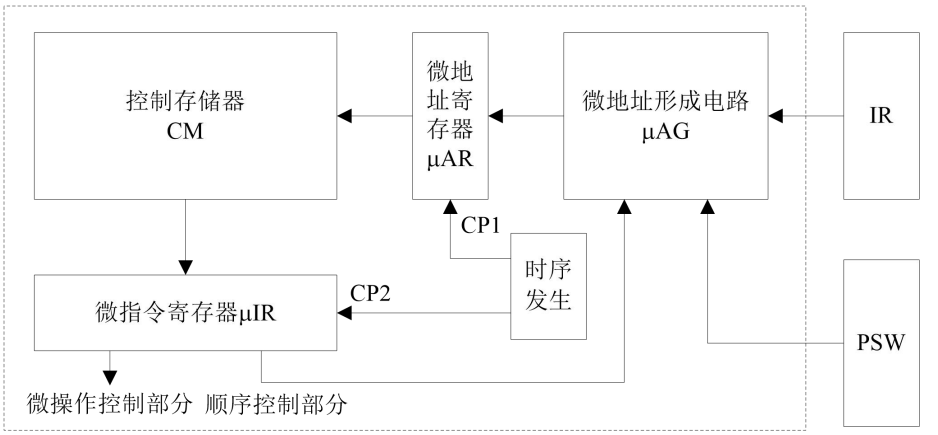


图 2.9 微程序控制器的基本组成

2.4.2 微指令寄存器μIR 和微指令译码

微指令寄存器μIR 保存从控存取出的微指令，可以分为微操作控制部分和顺序控制部分。微指令的顺序控制部分包括下址字段 NA 和转移方式字段 BM。下址字段的位数由控存容量决定，模型机的微程序占用了 256 个地址空间，考虑到给扩充留下余量，下址字段设计为 9 位。转移方式 BM 字段设计为 3 位，具体在后面微地址形成一节介绍。

微操作控制部分采用字段直接编码方法对微命令组合。根据微命令的相容、相斥性以及并行操作的需要，将微操作控制部分分为 8 个字段，见表 2.4。

表 2.4 OpenJUC-II 的微指令格式及编码

F0:XXoe（3 位）	F1:XXce（3 位）	F2:ALU（4 位）	F3:Shifter（2 位）	F4:AR（2 位）	F5:DR（2 位）	F6:PC（1 位）	F7:M&I（3 位）	F8:BM（3 位）	F9: NA（9 位）
0:NOP	0:NOP	0:NOP	0:NOP	0:NOP	0:NOP	0:NOP	0:NOP		
1:PCoe	1:PCce	1:ADD	1:SRce	1:ARoe′	1:DRoe′	1:PCinc	1:RD		
2:GRSoe	2:GRSce	2:ADDC	2:SLce	2:ARce	2: DRce′		2:WR		
3:Soe	3:IRce	3:SUB	3:SVce		3:DRce		3:PSWoe		
4:TRoe	4:TRce	4:SUBB					4:PSWce′		
5:ARoe	5:Ace	5:AND					5:STI		
6:DRoe	6:PSWce	6:OR					6:CLI		
7:SPoe	7:SPce	7:NOT					7:INTA		
		8:XOR							
		9:INC							
		A:DEC							

2.4.3 微地址寄存器和微地址的形成

机器复位时微地址寄存器μAR 的初始值决定了第一条微指令的地址，即取指令微程序的入口地址。OpenJUC-II 模型机取指令微程序的起始地址为 0，故μAR 复位时的初始值为 0。

后继微指令地址的形成，和当前微指令、机器指令以及 PSW 状态条件有关。OpenJUC-II 模型机的微地址形成采用下址字段与断定测试相结合的方法，其微指令格式如图 2.10 所示。

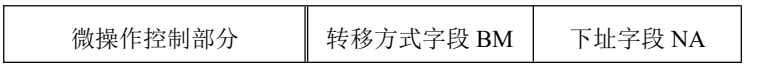


图 2.10 微指令格式

固定转移时，微地址直接由下址字段 NA 给出；根据测试结果转移时，微地址高位部分由下址字段 NA 的相应高位部分给出，微地址低位部分由测试结果给出。断定测试逻辑由硬件完成，测试条件主要来源于指令的操作码、寻址方式编码以及 PSW 中的标志位；在不同的场合有不同的测试条件，表 2.5 给出了转移方式字段的编码及微转移地址的形成方法。各种转移方式的设计原理在下面具体介绍。

表 2.5 微转移方式编码及微转移地址的形成方法

BM	操作	意义
0	$\mu AR = NA$	固定转移
1	$\mu AR_{8,6\sim0} = NA_{8,6\sim0}$，$\mu AR_7 = INTR \cdot IF$	根据是否有中断请求且是否允许中断产生两分支
2	$\mu AR_{8\sim2} = NA_{8\sim2}$， $\mu AR_1 = \overline{IR_{15}+IR_{14}+IR_{13}+IR_{12}}$， $\mu AR_0 = \overline{IR_{11}+IR_{10}+IR_9+IR_8+IR_7+IR_6}$	依据操作数的个数的三分支微转移。形成取源操作数、取目的操作数和执行阶段的微程序入口地址。如果是双操作数指令，则μAR_1=0；如果是单操作数指令，则μAR_1=1、μAR_0=0；如果是无操作数指令，则μAR_1=1、μAR_0=1。
3	$\mu AR_{8\sim1} = NA_{8\sim1}$， $\mu AR_0 = f_{\{OP,\ PSW(Z,O,S,\ C)\}}$	根据条件转移指令操作码和 PSW 的 ZF、OF、SF、CF 状态标志决定微地址，若满足条件μAR_0＝1，否则μAR_0＝0。
4	按操作码 OP 多路转移	按操作码 OP 形成多路微转移地址
5	$\mu AR_{8\sim3}$ =$NA_{8\sim3}$，$\mu AR_{2\sim0}$ =M	按寻址方式 M 形成多路微转移地址
6	保留	
7	$\mu AR_{8\sim1} = NA_{8\sim1}$， $\mu AR_0 = IR_5 + IR_4 + IR_3$	根据目的操作数寻址方式产生两分支：若 Md=000（寄存器寻址），则μAR_0＝0；否则μAR_0＝1。

1. 固定微转移（BM=0）

固定转移很容易理解，实现方法也很简单，就是将微指令中包含的下地址 NA 直接送给微地址寄存器μAR 就可以了，即μAR=NA。

2. 取指令阶段的多分支微转移（BM=2）

取指令结束时，将产生三个分支：如果是双操作数指令，转入取源操作数的微程序；如果是单操作数指令，转入取目的操作数的微程序；如果是无操作数指令，转入执行指令的微程序。在 2.2.1 已经介绍，模型机指令系统采用操作码扩展技术，指令的最高 4 位 $IR_{15\sim12}$ 不全为零，表示双操作数指令；如果 $IR_{15\sim12}$ 全为零，但 $IR_{11\sim6}$ 不全为零，表示单操作数指令；$IR_{15\sim6}$

全为零表示无操作数指令。下面简单分析一下这个三分支的微转移是如何实现的。

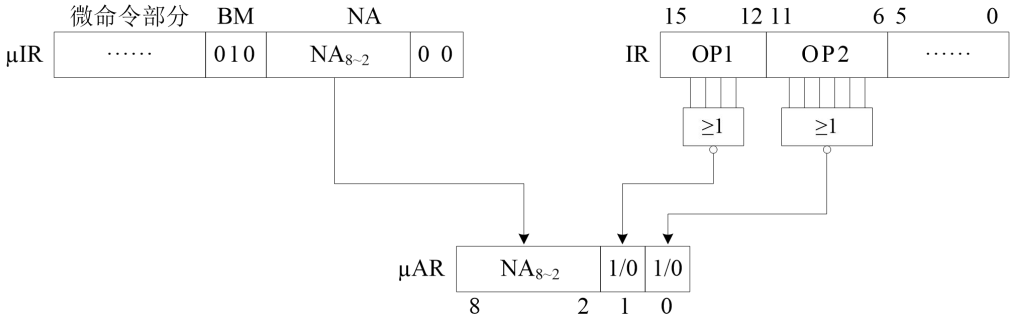


图 2.11 取指令结束时的三分支转移微地址生成

如图 2.11，微地址寄存器μAR 的高 7 位μAR_8~μAR_2由 NA 的高 7 位决定，而μAR_1和μAR_0由微地址形成逻辑依据 IR 产生，逻辑方程为：

$$\mu AR_{8\sim2} = NA_{8\sim2}$$

$$\mu AR_1 = \overline{IR_{15}+IR_{14}+IR_{13}+IR_{12}}$$

$$\mu AR_0 = \overline{IR_{11}+IR_{10}+IR_9+IR_8+IR_7+IR_6}$$

假如 NA=004H，如果是双操作数指令，$IR_{15\sim12}$不全为零，则μAR_1=0，而μAR_0可以为任意值，因此取源操作数微程序的入口地址为 004H 或 005H；如果是单操作数指令，$IR_{15\sim12}$全为零，但$IR_{11\sim6}$不全为零，则μAR_1=1、μAR_0=0，因此取目的操作数微程序的入口地址为 006H；如果是无操作数指令，$IR_{15\sim6}$全为零，则μAR_1=1、μAR_0=1，因此执行阶段微程序的入口地址是 007H。

3. 取操作数阶段的微程序分支（BM=5）

在取操作数阶段，需要根据不同的寻址方式执行不同的微程序。根据指令系统设计，寻址方式 M 占 3 位编码；源操作数寻址方式的编码占指令的 11~9 位，目的操作数寻址方式编码占指令的 5~3 位。见表 2.1 寻址方式编码。

取操作数阶段微程序转移地址的形成利用了寻址方式的编码，也就是指令中寻址方式的部分。如图 2.12，微转移地址由两部分拼接而成，高 6 位来自μIR 中 NA 的相应位，最低 3 位来自 IR 中的寻址方式编码 Ms 或 Md；根据寻址方式的不同，产生 8 个不同的微地址。

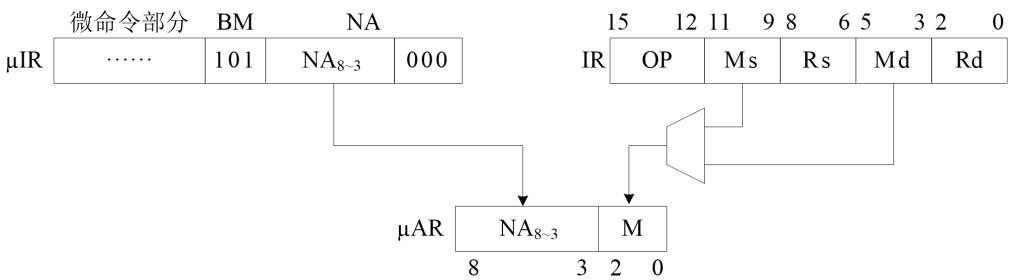


图 2.12 依据寻址方式的多分支微转移地址形成

相应的逻辑方程为：

$\mu AR_{8\sim3} = NA_{8\sim3}$

$\mu AR_{2\sim0} = M$

假如取源操作数的微程序安排自 008H 开始，即 NA=008H，则生成的微地址为 008H~00FH，分别对应着 8 种寻址方式。取目的操作数的微程序安排在 028H 开始，NA=028H，则生成的微地址为 028H~02FH。

4. 执行阶段的微程序分支

(1) 指令执行阶段微程序入口地址的形成（BM=4）

不同指令的执行阶段，对应着不同的微程序段。在进入指令执行阶段时，应根据机器指令的操作码产生微程序的入口地址。通常一条机器指令对应一段微程序，所以转移的分支将非常多，称为宽转移。

和前面用寻址方式编码参与微地址生成相类似，可以直接将操作码插入到微地址的适当位置。和前面方法不同的是，微地址的高位不是由下址字段 NA 给出，而是固定的常数。以双操作数指令为例，指令操作码是指令编码的 15~12 位，将其直接作为微地址的最低 4 位；假设需要将微程序入口分配在 040H~04FH，则微地址的高 5 位固定为 00100。如图 2.13。

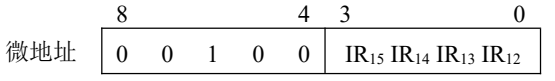


图 2.13 双操作数指令的微程序入口地址形成

每段微程序入口地址只占用 1 个地址单元，对于较长的微程序，可以通过固定转移的方法转到空闲的控存单元。对照表 2.2 指令编码表，可以得到各个双操作数指令的微程序入口地址，见表 2.6。

表 2.6 双操作数指令的微程序入口地址

指令		微程序入口地址	
助记符	操作码	二进制	十六进制
MOV	0001B	001000001B	041H
ADD	0010B	001000010B	042H
ADDC	0011B	001000011B	043H
SUB	0100B	001000100B	044H
SUBB	0101B	001000101B	045H
AND	0110B	001000110B	046H
OR	0111B	001000111B	047H
XOR	1000B	001001000B	048H
CMP	1001B	001001001B	049H
TEST	1010B	001001010B	04AH

OpenJUC-II 模型机的指令系统采用操作码扩展技术，分为双操作数指令、单操作数指令和无操作数指令三类，用类似的方法可以产生单操作数指令和无操作数指令的微程序入口地

址，如图 2.14 和图 2.15，各个指令具体的入口地址不再列表给出。

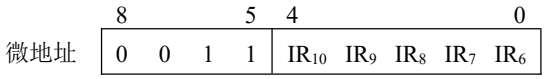


图 2.14 单操作数指令的微程序入口地址形成

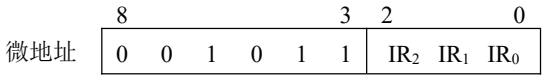


图 2.15 无操作数指令的微程序入口地址形成

(2) 执行过程中的微转移地址形成（BM=3）

条件转移指令的执行过程中需要测试 PSW 的标志位，以 JC 为例，若 CF=1，则程序转移，若 CF=0，顺序执行。实现程序转移的方法是将转移地址放入 PC，因此，在条件转移指令执行阶段的微程序中，微程序也需要有两个分支：如果机器指令的转移条件满足，将转移地址放入 PC；否则，不修改 PC 的值。

微地址的高位由下址字段 NA 直接给出，μAR_0 根据指令功能和 PSW 的标志位给出，如图 2.16。分析表 2.2 指令编码表，8 条条件转移指令由 IR 的 8、7、6 位区分；其中 IR_8 和 IR_7 区分所测试的标志位类型，IR_6 区分所测试的标志位为 1 还是为 0。因此用 IR_8 和 IR_7 选择标志位，IR_6 控制异或门决定是否取反。若转移指令的条件满足，μAR_0=1，否则μAR_0=0。

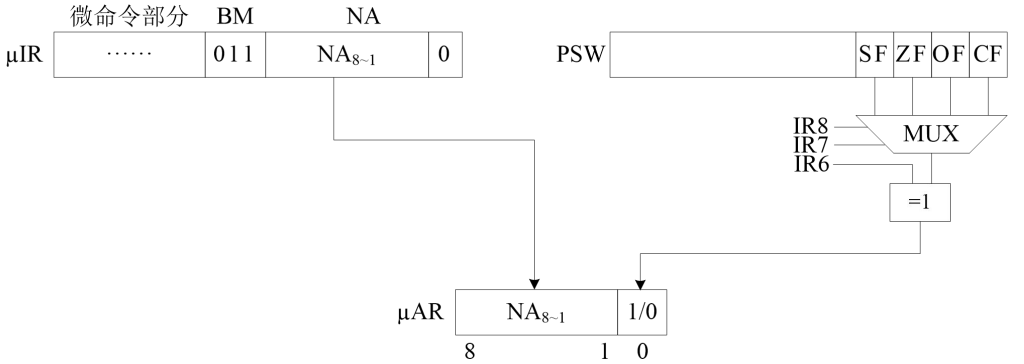


图 2.16 依据 PSW 的微转移地址形成

(3) 保存运算结果时的两分支微转移地址形成（BM=7）

有操作数指令的指令执行阶段最后还需要保存运算结果，此时，需要判断目的操作数是在寄存器中、还是在内存中，这就需要依据目的操作数寻址方式决定转移地址。根据寻址方式编码和指令格式，只有当 IR_5、IR_4 和 IR_3 都为零时，结果存入寄存器，其他情况均存入存储器。微地址的最低位由下式的逻辑给出。

$$\mu AR_0 = IR_5 + IR_4 + IR_3$$

如果目的操作数是在寄存器中，则$\mu AR_0 = 0$；如果目的操作数是在内存中，则$\mu AR_0 = 1$。微地址的高位由下址字段 NA 直接给出，微地址的形成方法如图 2.17 所示。

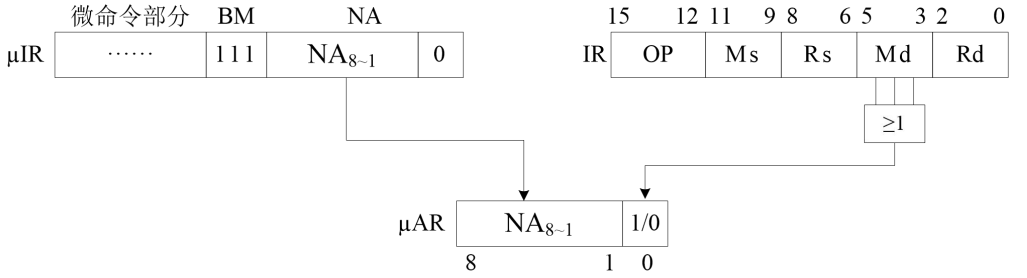


图 2.17 依据目的寻址方式的两分支微地址形成

微地址的高位由下址字段 NA 直接给出，假设 NA=050H，可知产生的两分支微地址分别是 050H（目的操作数在寄存器中）和 051H（目的操作数在内存中）。

(4) 指令执行结束时的中断检测（BM=1）

每条指令执行结束时，检测是否有中断请求（INTR 有效）并且 CPU 是否允许中断（IF 有效），如果是，则μAR_7 =1；否则μAR_7 =0。微地址的形成逻辑为：

$$\mu AR_{8,6\sim0} = NA_{8,6\sim0},$$

$$\mu AR_7 = INTR \cdot IF$$

假设 NA=000H，则两分支的微地址分别是 000H 和 080H，080H 是中断隐指令的微程序入口。

5. 控存地址空间分配

上面详细讨论了表 2.5 所列的七种微程序分支的设计原理和实现方法，除了 BM=4 的情况，其他六种情况的微地址均由 NA 和地址修改逻辑共同决定，OpenJUC-II 模型机的微地址分配如表 2.7。在后面微程序设计章节将看到微转移方式和下址字段的具体应用。

表 2.7 微程序地址空间分配

微程序	微地址
取指令	000H~003H
取源操作数的入口	004H 和 005H
取目标操作数的入口	006H
执行阶段的入口	007H
取源操作数微程序	008H～020H
（可用）	021H～027H
取目标操作数微程序	028H～03FH
（可用）	040H
双操作数指令的入口（最多 15 条，实际 10 条）	041H～04FH
执行结果存入目的操作数的微程序	050H～052H
（可用）	053H～057H
无操作数指令的入口（最多 8 条，实际 6 条）	058H～05FH
单操作数指令的入口（最多 31 条，实际 22 条）	061H～07FH
中断响应隐指令的入口	080H
（可用）	081H～1FFH

2.4.4 微程序控制时序

如图 2.18，时序系统有两个周期相等的信号 CP1 和 CP2；CP1 将μAG 形成的微指令地址打入μAR，启动了从控存读出微指令的操作；CP2 将控存输出的微指令打入微指令寄存器μIR，开始执行这条微指令。下一条微指令的读出，表示当前微指令执行结束，也就是在每个 CP1 出现时还应该保存上一条微指令的执行结果，因此 CP1 还作为 CPU 内部各个寄存器的时钟脉冲。

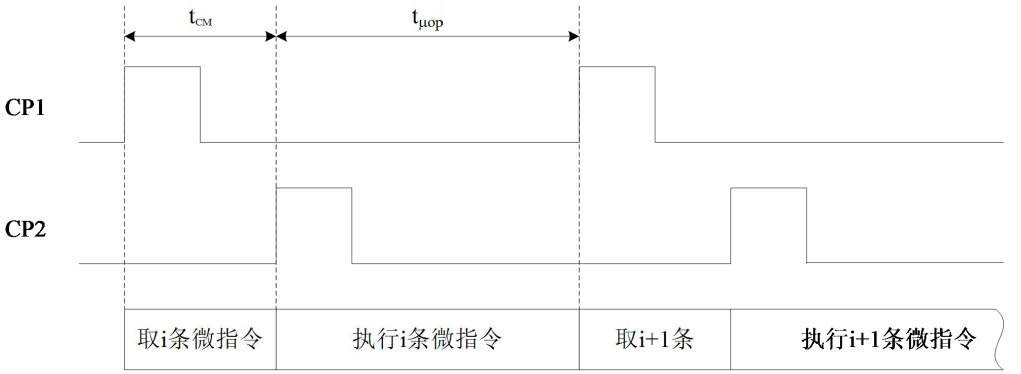


图 2.18 微指令的串行执行时序

模型机的时钟源由实验板的外部晶振提供，经过 FPGA 内部的锁相环调整为 10MHz 的系统时钟，每个微指令需要 2 个系统时钟周期，即微指令周期为 0.2μS。复位时 CP1 为高、CP2 为低，μAR 清零；开始运行后第一个系统时钟首先使 CP2 上升沿到来，将 000H 控存单元的微指令打入微指令寄存器μIR，开始执行这条微指令；下一个系统时钟产生 CP1 上升沿，将该微指令的执行结果保存到相关寄存器，同时将μAG 产生的下一条微指令地址打入μAR。也就是说，每个微指令周期 CP2 在先、CP1 随后，循环往复。

2.5 微程序设计

2.5.1 指令执行过程

微程序控制的工作过程就是实现机器指令的过程。一条指令的执行包括四个阶段，即取指令阶段 IF、取源操作数阶段 SOF、取目的操作数阶段 DOF 和执行阶段 EXE，如图 2.19。取指令结束后，根据指令的类型三分支转移，如果是双操作数指令，则先取源操作数，再取目的操作数，之后进入执行阶段；如果是单操作数指令，直接取目的操作数，之后进入执行阶段；如果是无操作数指令，则直接进入执行阶段。执行结束后，判断是否有中断请求，有且中断允许时执行中断隐指令，否则返回到取指令阶段，取下一条指令。

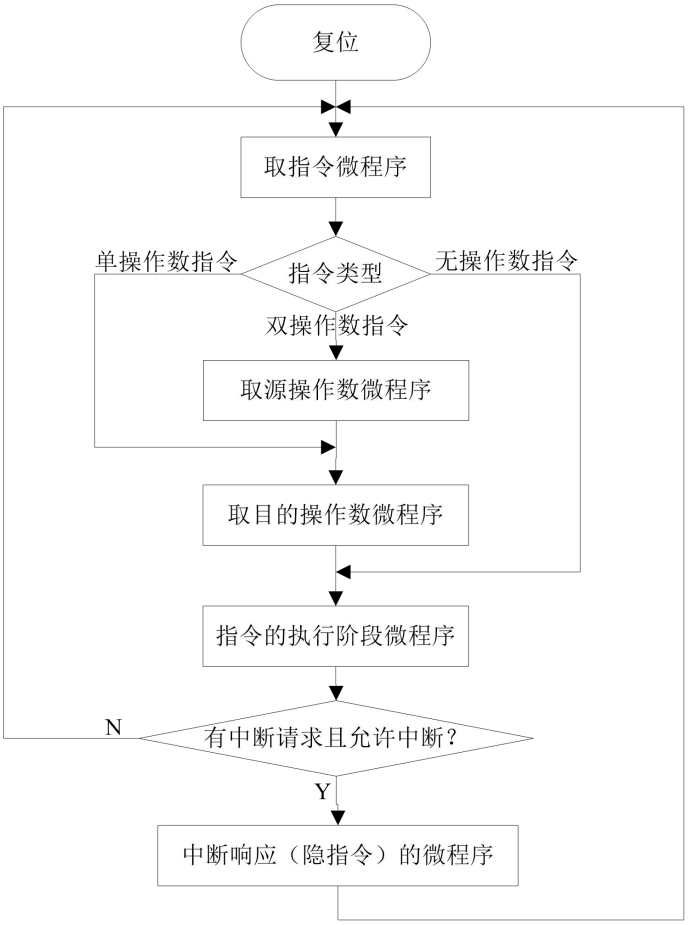


图 2.19 指令执行过程

2.5.2 微程序的设计方法

在微程序控制计算机的指令系统、数据通路及微指令的格式设计完成后，为了实现 CPU 指令系统的功能，必须编写每一条指令的微程序。由于指令的运行包括取指令、取操作数和执行三个阶段，因此，写微程序也包括三个方面，即取指令微程序、取操作数微程序和指令执行的微程序。

微程序设计的一般方法如下：

（1）画出微流程。根据 CPU 数据通路（见图 2.1）画出取指令微流程、取操作数微流程；根据每条指令的功能画出指令的执行微流程。

（2）将微流程翻译成微命令编码。按微指令格式及微命令的编码（见表 2.4），根据微流程的每一步填写微指令 F0－F7 字段的内容。

（3）分配微地址。根据微流程的每一步之间是否有转移以及何种转移，参照表 2.5 填写微指令 F8 字段（转移方式 BM）和 F9 字段（下地址 NA），确定每条微指令的地址。

2.5.3 取指令的微程序设计

按照上面介绍的微程序设计一般方法，分为三个步骤。

（1）画出取指令微流程

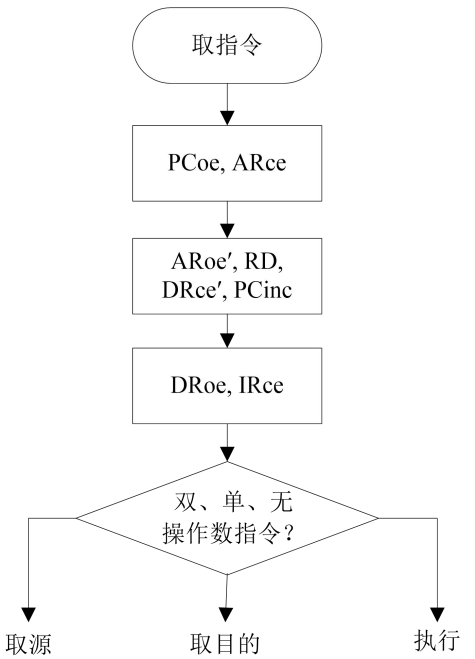


图 2.20 取指令微流程

取指令的目标是将存储器中的指令取到指令寄存器 IR 中。首先 PC 的内容送给 AR；然后 AR 的内容送到地址总线，给出读信号，将读出的指令送给 DR，同时 PC 加 1；最后 DR 的内容即指令送给 IR，完成取指令。取指阶段微流程如图 2.20。

（2）将微流程翻译成微命令编码

参照微指令编码表 2.4，根据上面取指令微流程填写微指令 F0~F7 字段，无操作字段填‘0’，见表 2.8。

表 2.8 取指令微程序的微命令编码

微地址(H)	微指令(H)	微指令字段(H)										微命令
		F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	
		1	0	0	0	2	0	0	0			PCoe, ARce
		0	0	0	0	1	2	1	1			ARoe′, RD, DRce′, PCinc
		6	3	0	0	0	0	0	0			DRoe, IRce

（3）分配微地址

表 2.8 中每条微指令的地址尚未确定，接下来就要根据微流程及控存的使用情况填写微指令的 F8、F9 字段。

根据微程序控制器的设计，在 CPU 复位时μAR=000H，所以第一条微指令地址是 000H。

从图 2.20 可以看出，取指令的过程中微程序没有分支，因此接下来的几条微指令地址可以依次为 001H、002H、003H，采用固定转移 BM=0（见表 2.5）即 F8＝0，转向下一地址，下地址 001H、002H、003H 由 F9 字段指定。

指令取到 IR 以后，根据指令包含操作数的个数要进行三分支转移。在 2.4.3 小节已经介绍了 BM＝2 的多分支转移的硬件逻辑，根据表 2.7 分配的控存地址空间，设置 NA＝004，可依次求得取源、取目和执行的入口分别是：004 或 005，006，007。

最后将微指令以 16 进制表示，完整的微程序如表 2.9。

表 2.9 取指令微程序

微地址(H)	微指令(H)	微指令字段(H)										微命令
		F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	
000	20080001	1	0	0	0	2	0	0	0	0	001	PCoe, ARce
001	00069002	0	0	0	0	1	2	1	1	0	002	ARoe′, RD, DRce′, PCinc
002	CC000003	6	3	0	0	0	0	0	0	0	003	DRoe, IRce
003	00000404	0	0	0	0	0	0	0	0	2	004	BM=2

表中的第一列对应每条微指令在控存中的微地址；第二列为每条微指令的 10 个字段按对应的位数展开为二进制拼接后形成的编码，并转换为 16 进制表示；微指令字段所包含的 10 列为每条微指令 10 个字段的微命令编码，0 代表空操作；最后一列对应每条微指令中有效的微命令。

对于像取指令这样以顺序执行为主的微程序，采用表格的形式比较简洁；但是在分支较多、转移频繁时，用表格不容易看清微指令的执行顺序。本书设计一种图符的形式表达微程序，如图 2.21，每条微指令用一个图块表示，图块的左上角是以十六进制表示的该微指令的微地址，右上角是微指令代码，中部是该微指令包含的微命令符号，右下角是下址字段的微地址（十六进制），左下角是微转移方式字段的值及简要说明。以流程图表示的取指令微程序如图 2.22。

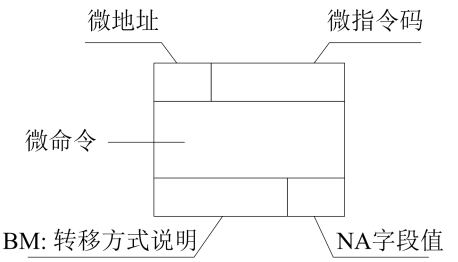


图 2.21 微指令图块

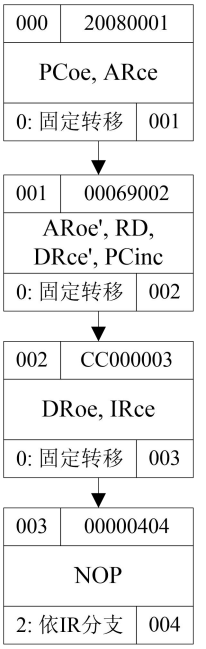


图 2.22 用图块表示的取指令微程序

2.5.4 取操作数阶段微程序设计

如果是双操作数指令，取指令结束后进入取源操作数阶段。取源操作数的微流程如图 2.23，入口地址为 004H 或 005H。在微地址为 004H 和 005H 的微指令中，根据不同的寻址方式 M 多路转移，该转移方式由 BM＝5 控制，微地址的形成逻辑见 2.4.3。设置 NA＝008H，可求得源操作数为寄存器寻址、寄存器间接、寄存器自增间接、立即、直接、间接、变址和相对寻址的微程序入口地址分别为 008H、009H、00AH、00BH、00CH、00DH、00EH 和 00FH。取到的源操作数放在 TR 暂存器中，取源操作数结束后固定转移到取目的操作数的入口 006H。除了 8 种寻址方式的微程序入口地址外，其它微指令的地址可安排在任何空闲的控存单元；为了紧凑地利用控存空间，图 2.23 已经分配好了各个微指令的地址，但图中大部分微指令的微命令部分和微指令编码部分是空白，留待同学们完成。

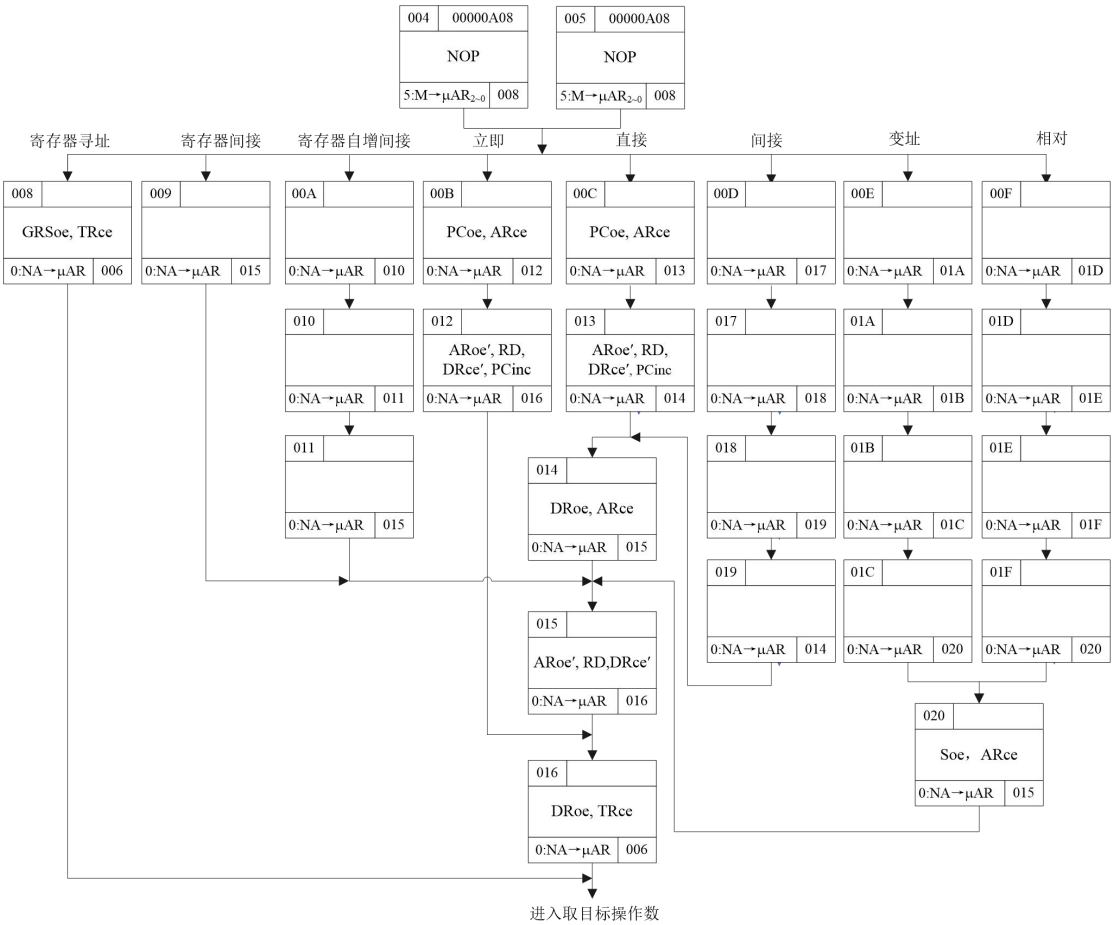


图 2.23 以流程图表示的取源操作数微程序

取目的操作数微流程与取源操作数微流程基本一致，只是每条微指令的下址字段 NA 设置有所不同，并且取出的目的操作数存放在 A 中，取完目的操作数后微地址固定转移到执行阶段的入口 007。

图 2.24 是以流程图表示的取目的操作数微程序，其入口地址为 006，取到的目的操作数放在 A 寄存器中。由 BM=5，NA=028H 控制转向目的操作数为寄存器寻址、寄存器间接、寄存器自增间接、直接、间接、变址和相对寻址的微程序入口地址分别为 028H、029H 、02AH、02CH、02DH、02EH 和 02FH。立即寻址不应作为目的操作数的寻址方式，所以 02BH 微指令什么都不做，仅仅是转到执行阶段的入口。取出的目的操作数存放在 A 中，取目的操作数结束后转移到执行指令的微程序入口 007H。

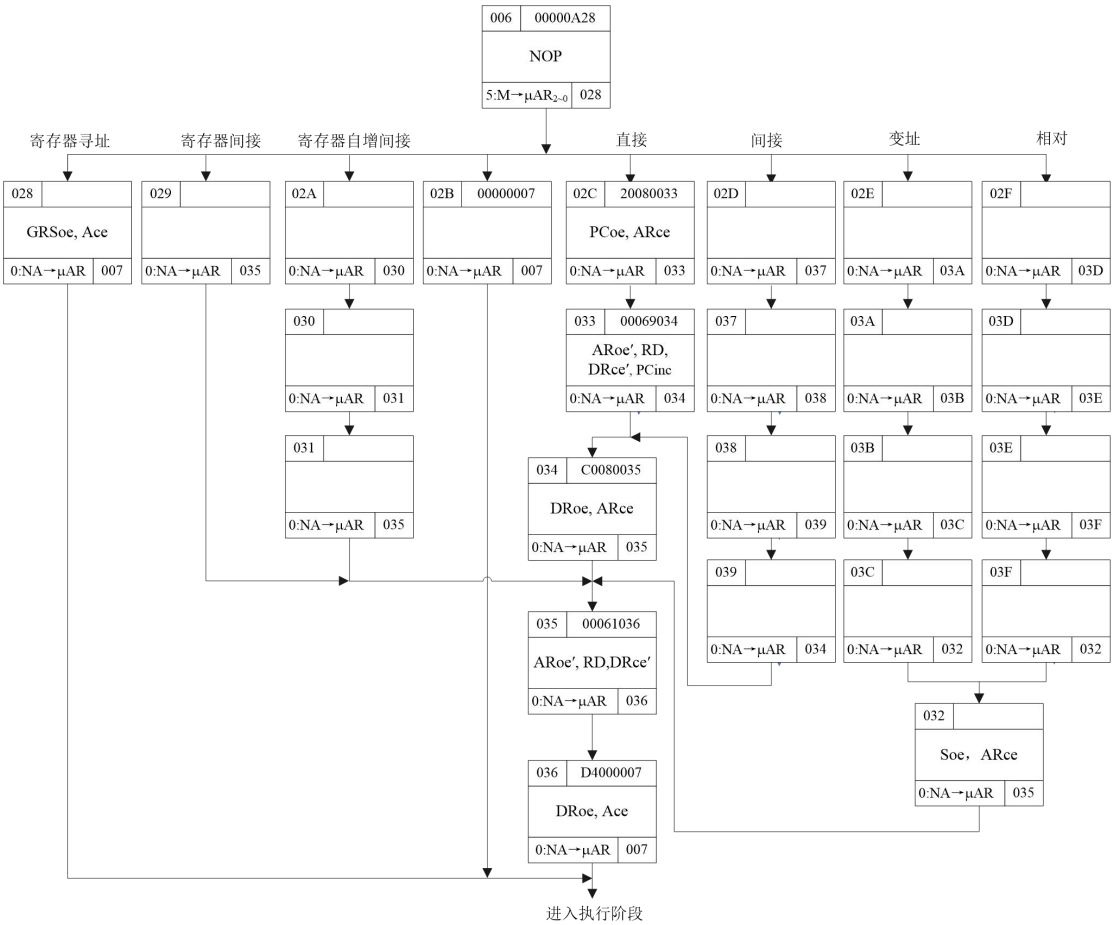


图 2.24 以流程图表示的取目的操作数微程序

2.5.5 执行阶段微程序设计举例

执行阶段执行相应指令的功能，所以每条指令对应有一段微程序。执行阶段总的入口是007H，该微指令是一条微转移指令，依据指令操作码实现宽转移，如图 2.25，各指令对应微程序入口地址形成方法在 2.4.3 小节已经介绍。

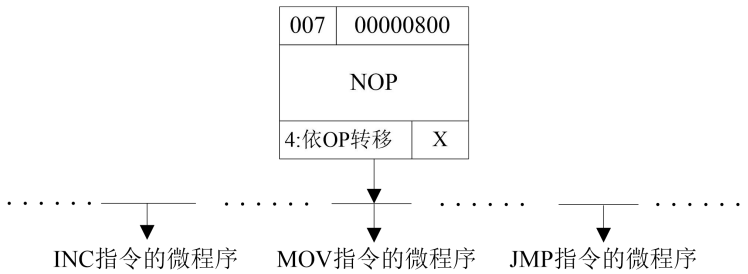


图 2.25 执行阶段入口的微转移

1. 保存结果的微程序

由于很多指令都需要保存运算结果，可以将其设计为公用的微程序。表 2.7 分配给它的微地址范围是 050H～052H，微程序见图 2.26。虽然目的操作数的寻址方式有 7 种，但操作数的存放位置只有两种，要么在寄存器中、要么在主存中。如果在主存中，在取目的操作数阶段也已经将地址保留在 AR 中，不需要重新根据目的寻址方式获得有效地址。050H 微指令将结果保存在寄存器中，051H 和 052H 微指令将结果保存在主存中。运算结果保存之后，该指令的执行就结束了，所以 BM=1，NA=000，检测是否有中断请求，若有中断请求并且 CPU 是否允许中断，转向 080H 中断响应隐指令，否则转向 000H 取下一条指令。

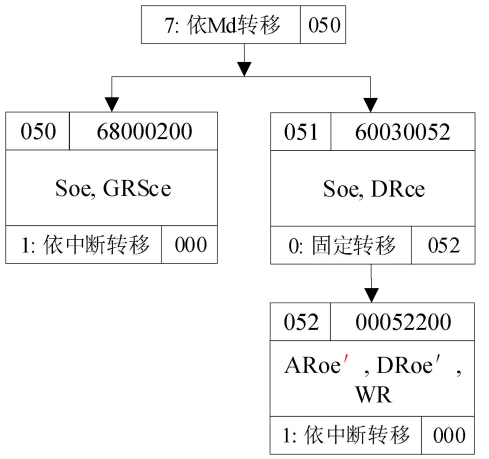


图 2.26 保存运算结果的微程序

2. 单操作数运算的微程序设计

单操作数运算指令有 3 条：INC、DEC、NOT，它们的微程序入口地址分别设计在 071H、072H、073H（见 2.4.3 图 2.14）。单操作数指令进入执行阶段时，目的操作数已经在 A 暂存器中，执行阶段只要控制 ALU 执行相应的运算功能，并且将运算结果保存在移位寄存器中，同时将运算结果的特征标志保存到 PSW 中。微程序见图 2.27。

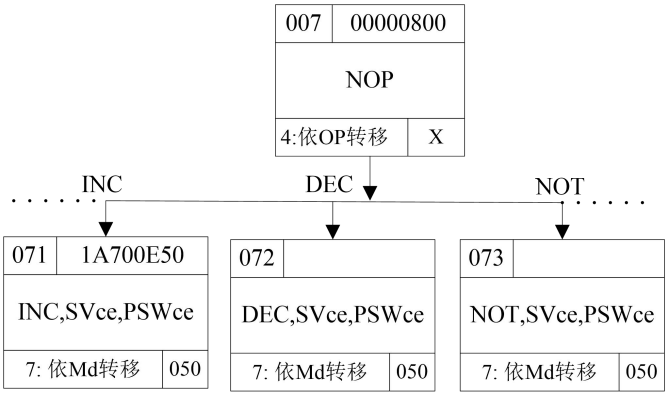


图 2.27 单操作数运算指令的执行阶段微程序

接下来要转移到图 2.26 的微程序，将移位寄存器中的结果保存到目的操作数所在的寄存器或存储单元，所以微转移方式 BM=7，NA=050H，从图 2.17 可知产生的两分支微地址是 050H 和 051H。

3. 转移指令的微程序设计

转移指令是单操作数指令，分为无条件转移和条件转移两种。无条件指令的功能是转向目的地址去执行，这个目的地址就是通过寻址方式得到的有效地址；转移指令只能使用内存寻址，不支持寄存器寻址。取操作数阶段结束以后，“目的操作数”存放在寄存器 A 中，操作数的有效地址在 AR 中；实际上“目的操作数”不是操作数，而是转移目的地址所在单元的指令码，对转移指令而言 A 中的“操作数”没有意义，转移指令只用 AR 中的内容作为转移地址。所以实现转移的微指令很简单，就是将 AR 的内容送给 PC，见表 2.10。

表 2.10 JMP 指令的微程序

微地址(H)	微指令(H)	微指令字段(H)										微命令
		F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	
070				0	0	0	0	0	0	1	000	ARoe, PCce

条件转移指令执行时根据转移条件是否满足两分支转移，条件满足时将 AR 的内容送 PC 实现转移，条件不满足时顺序执行，无需改变 PC。2.4.3 小节已经介绍 BM=3 时的微转移地址形成方法，转移条件的判断是由硬件完成的，所以不同条件转移指令的微程序是一样的，只是微程序入口地址不同，见图 2.28；图中微地址没有给出，读者可根据 2.4.3 小节图 2.14 单操作数指令执行阶段微程序入口地址的形成方法计算得出。图 2.28 中条件不满足和条件满足的微指令地址 026H 和 027H 仅仅是一个例子，原则上可以分配在其他空闲的控存单元，但要注意条件不满足的微指令地址一定要是偶数地址，而条件满足的微指令地址是奇数，并且后者比前者大 1，具体原因见 2.4.3 小节介绍 BM=3 时的微转移地址形成方法。

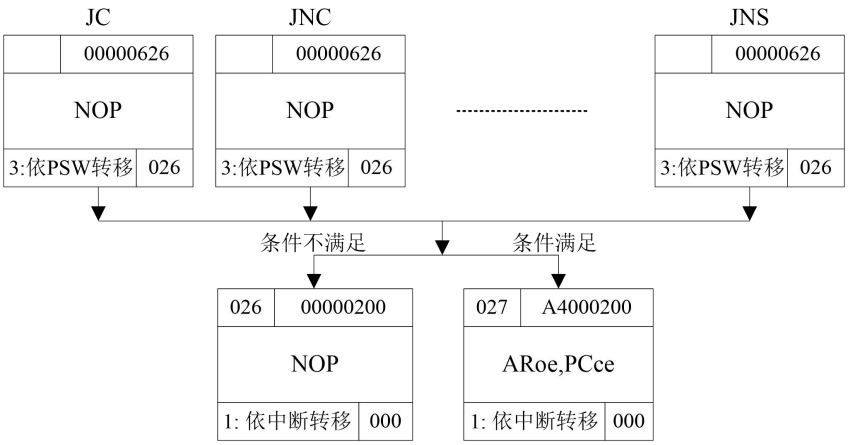


图 2.28 条件转移指令的微程序

4. 移位指令的微程序设计

从 2.3.4 移位寄存器和 2.3.5 运算器数据通路的硬件设计可以看出，不同移位指令移入数据来源的选择由硬件实现，移出数据位送到 PSW 的 CF 标志位也是由硬件实现。所以各个移位指令的微程序是相同的（但是微程序入口地址不同），只要控制左移或右移，以及将 CF 保存到 PSW 中，见表 2.11。读者可根据 2.4.3 小节图 2.14 计算得出 7 条移位指令的微程序入口地址。

表 2.11 移位指令的微程序

指令	微指令(H)	微指令字段(H)										微命令
		F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	
左移	18200E50	0	6	0	2	0	0	0	0	7	050	SLce, PSWce
右移	18100E50	0	6	0	1	0	0	0	0	7	050	SRce, PSWce

5. 双操作数指令的微程序设计

双操作数指令主要是算术运算和逻辑运算指令，此外还有一条数据传送 MOV 指令。微程序见图 2.29。

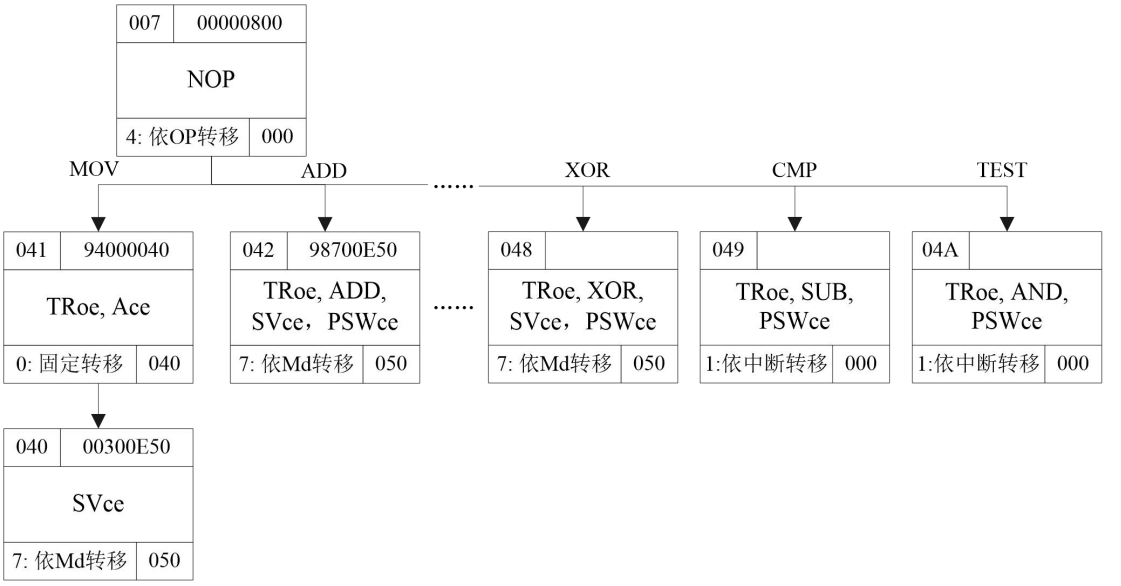


图 2.29 双操作数指令的微程序

(1) 双操作数运算指令的执行阶段

进入执行阶段时，源操作数已经取到 TR 暂存器中，目的操作数也已经取到 A 寄存器中。因此，需要将 TR 中的源操作数送到内部总线 IB 上，让 ALU 做相应指令的运算，并且将运算结果保存在移位寄存器中，同时将运算结果的特征标志保存到 PSW 中，这些操作在一条微指令中完成。需要保存结果的指令接下来转移到图 2.26 的微程序，将移位寄存器中的结果保存到目的操作数所在的寄存器或存储单元。注意 CMP 指令和 TEST 指令不需要保存结果，它们的 BM=1 结束指令的执行。

(2) MOV 指令的执行阶段

MOV 指令将源操作数复制到目的操作数，原本可以不需要经过 ALU，但为了共用图 2.26

保存结果的微程序，需要将 TR 中的源操作数传送到移位寄存器中，所以利用了 ALU 的传送功能（见 2.3.1 关于 ALU 的设计）。

6. 堆栈相关指令的微程序设计

和堆栈操作有关的指令主要有压栈指令 PUSH、出栈指令 POP、子程序调用指令 CALL、子程序返回指令 RET。

堆栈是主存储器中的一块连续的专用存储区域，它只能从一端存入或取出数据，遵循后进先出（LIFO）的规则。如果是从低地址一端操作，随着存入堆栈数据的增加，存放数据的单元地址减小，通常称作堆栈向上增长；如果是从高地址一端操作，随着存入堆栈数据的增加，存放数据的单元地址增大，通常称作堆栈向下增长。OpenJUC-II 的堆栈是向上增长的。

堆栈操作必须通过堆栈指示器 SP，它是 CPU 中的一个专用寄存器，用来指向栈顶元素。将数据压入堆栈时，首先将 SP 的内容减 1，使 SP 指向一个新的内存单元，然后将数据存入以 SP 内容为地址的内存单元；将数据从堆栈中取出时，首先以 SP 内容为地址取出该内存单元的数据，然后将 SP 的内容加 1，使 SP 指向新的栈顶。总之，SP 始终指向当前的栈顶元素。

表 2.12 给出了 PUSH 指令的微程序。在取操作数阶段，目的操作数已经存放到 A 暂存器中，也就是要放入堆栈的数据，因此首先通过 ALU 将 A 暂存器中的目的操作数传送到移位寄存器，这里利用了 ALU 电路的一个比较特别的功能，当运算控制信号全为 0 时，dst 输入端的数据传送到 F 输出端，所以 078H 这条微指令只需要给出移位寄存器的 SVce 信号。因为 079H 是 POP 指令的微程序入口，所以下一条微指令要放到空闲的控存单元，表 2.12 采用了 09BH（也可以是其他的微地址，只要不和其他微程序冲突），将移位寄存器的内容传送到 DR 寄存器，准备后面写入到堆栈。09CH~09EH 微指令将堆栈指针 SP 的内容减 1，并送给 AR 寄存器作为写入主存的地址。接下来要将 DR 的内容写入到以 AR 内容为地址的主存单元中，利用图 2.26 中保存结果的微指令，故直接转向 052H 微指令。

表 2.12 PUSH 指令的微程序

微地址(H)	微指令(H)	微指令字段(H)										微命令
		F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	
078	0030009B	0	0	0	3	0	0	0	0	0	09B	SVce
09B	6003009C	3	0	0	0	0	3	0	0	0	09C	Soe,DRce
09C	F400009D	7	5	0	0	0	0	0	0	0	09D	SPoe,Ace
09D	02B0009E	0	0	A	3	0	0	0	0	0	09E	DEC,SVce
09E	7C080052	3	7	0	0	2	0	0	0	0	052	Soe,ARce,SPce

7. HATL 指令

HALT 是停机指令，主要用于调试时避免程序跑飞。该指令的微程序只需一条微指令，实现原地踏步，见表 2.13。注意，停机指令执行后，只有复位才能使 CPU 重新工作。

表 2.13 HALT 指令的微程序

微地址(H)	微指令(H)	微指令字段(H)										微命令
		F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	
058	00000058	0	0	0	0	0	0	0	0	0	058	

2.6 主存储器

主存储器的字长是 16 位，CPU 地址总线也是 16 位，并且按字编址，不能按字节访问。因此主存空间为 64K×16 位。主存储器可以利用实验板上的 SRAM 芯片，如 DE2-115 开发板配有一片容量 1M×16 位的 SRAM 芯片。为了减少对实验板的依赖，也可以使用 FPGA 的片内存储器作为主存储器，容量可以根据片内 RAM 资源的多少而定，不一定要配满 128KB，满足教学实验的需要即可。

复位时，PC 的初始值为 0030H，即主程序的第一条指令放在 0030H 单元。SP 的初始值也为 0030H，堆栈的存储空间为 002FH~0008H。中断向量表的入口地址为 0000H，一共预留了 8 个中断向量的存储空间；IO 接口与主存统一编址，占用 FF00H~FFFFH 的地址空间。整个 64K 地址空间分配见表 2.14。

表 2.14 模型机地址空间分配

0000H~0007H	中断向量表
0008H~002FH	堆栈
0030H~FEFFH	程序及数据
FF00H~FFFFH	IO 接口

2.7 输入输出

2.7.1 概述

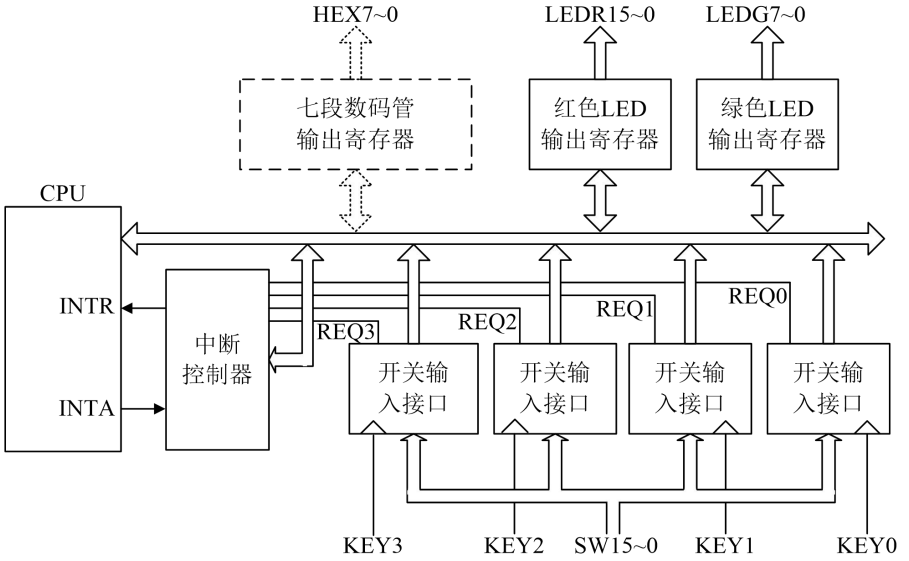


图 2.30 输入输出系统框图

输入输出系统设计了 4 个输入接口，绿色 LED 和红色 LED 输出接口以及 8 个七段数码管的输出接口（预留），如图 2.30。

输入输出接口和主存统一编址，占用主存空间的 FF00H~FFFFH 共 256 个地址，已经使用的接口地址见表 2.15。

表 2.15 接口地址表

接口地址	有效位数	输入接口寄存器
FF00H	8	中断屏蔽寄存器
FF01H	8	绿色 LEDG[7:0] 数据寄存器
FF02H	16	红色 LEDR[15:0] 数据寄存器
FF08H	16	输入数据寄存器 0
FF09H	1	输入状态寄存器 0
FF0AH	16	输入数据寄存器 1
FF0BH	1	输入状态寄存器 1
FF0CH	16	输入数据寄存器 2
FF0DH	1	输入状态寄存器 2
FF0EH	16	输入数据寄存器 3
FF0FH	1	输入状态寄存器 3

2.7.2 输出接口

基本输出接口可以用来连接实验板上的 LED 指示灯，如 DE2-115 实验板有 9 个绿色 LED 和 18 个红色 LED。模型机系统设计了 2 个数据寄存器作为输出接口， 1 个 8 位接口寄存器的输出端连接到 8 个绿色 LED，1 个 16 位接口寄存器的输出端连接到 16 个红色 LED，寄存器的输入端与系统数据总线相连，CPU 不仅可以输出数据改变 LED 的状态，也可以从寄存器读回数据，逻辑框图见图 2.31。

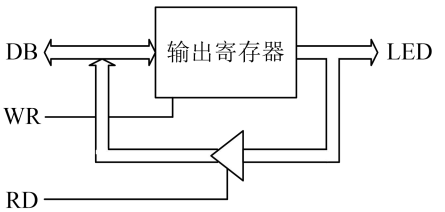


图 2.31 输出接口逻辑框图

2.7.3 输入接口

基本输入接口可以用来连接实验板上的拨动开关。如 DE2-115 实验板有 18 个拨动开关和 4 个按键，模型机系统设计了 4 个输入接口连接到这些拨动开关和按键。每个输入接口含有 1 个 16 位的数据寄存器和 1 个 1 位的状态寄存器，如图 2.32 所示。4 个输入接口的 READY 分别连接到 DE2-115 实验板的 KEY0~KEY3 四个按键，而 DATA 则共用 16 个拨动开关（最

高两位拨动开关 SW17~16 没有使用）。

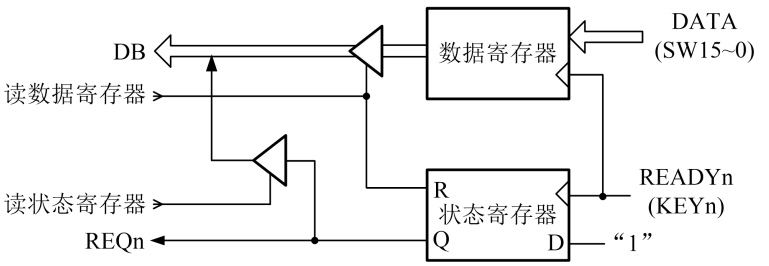


图 2.32 输入接口逻辑框图

当按下某一个 KEYn 按键时，将 16 个拨动开关作为数据打入该接口的数据寄存器，同时状态寄存器置 1；状态寄存器的输出可以由 CPU 通过数据总线读出，同时也可以作为中断请求 REQn 送给中断控制器；CPU 读该接口寄存器时，数据寄存器读信号打开三态门，将数据寄存器的内容输出到 DB 数据总线，同时将状态寄存器清零。

2.7.4 中断控制器

中断控制器的组成如图 2.33 所示。中断屏蔽寄存器的地址是 FF00H，因为模型计算机的外设与主存统一编址，故可以使用 MOV 指令访问 FF00H 地址实现对中断屏蔽寄存器写入或读出。

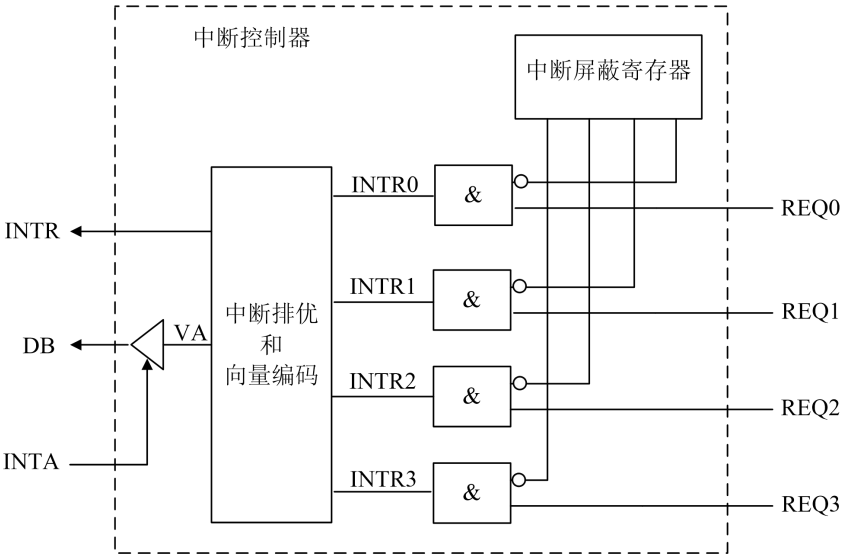


图 2.33 中断控制器组成框图

中断系统采用向量中断方式，中断向量表的首地址是 0000H，每个中断向量占用一个存储单元，存放该中断服务程序的入口地址。向量地址与中断源的对应关系见表 2.16。

表 2.16 中断向量地址

中断源	向量地址
INTR0	0000H
INTR1	0001H
INTR2	0002H
INTR3	0003H

2.7.5 CPU 对中断的支持

1. 硬件设计

CPU 内部有一个允许中断触发器 IF，用来控制是否响应中断。IF 可以由一个带有清 0 和置 1 端口的 D 触发器实现，开中断指令（EI）控制置 1 端，关中断指令（DI）和 CPU 复位信号控制清 0 端。

2. 中断隐指令的微程序设计

从 2.4.3 微地址形成一节知道，每条指令执行结束时，检测是否有中断请求（INTR 有效）并且 CPU 是否允许中断（IF 有效），如果是，CPU 响应中断，微程序转向 080H 中断隐指令的微程序入口。中断隐指令完成的操作是：

（1）保护 PC，即 PC 入栈；

（2）保护 PSW，即 PSW 入栈；

（3）中断响应信号 INTA 有效，从数据总线读中断向量地址 VA；

（4）根据 VA，取出中断服务程序的入口地址，送 PC；

（5）关中断。