

# 实验四 微程序控制器实验

## 一、实验目的

- 1、理解“微程序”设计思想，了解“指令-微指令-微命令”的微程序结构。
- 2、掌握微程序控制器的结构和设计方法。

## 二、实验内容

设计一个“最简版本”的 CPU 模型机：利用时序发生器来产生 CPU 的预定时序，通过微程序控制器的自动控制，在数据通路中完成唯一的 CPU 功能——程序跳转。

## 三、实验器件

- 1、ROM 存储器（2764）和计数器（74LS163、74LS192）。
- 2、D 触发器（74LS74）、寄存器（74LS273）及移位寄存器(74LS194)。

## 四、实验原理

如图 2-34 所示，本实验构造了一个“最简版本”的 CPU，仅仅由时序发生器（图中黄框部分）、微程序控制器（图中红框部分）和数据通路组成。

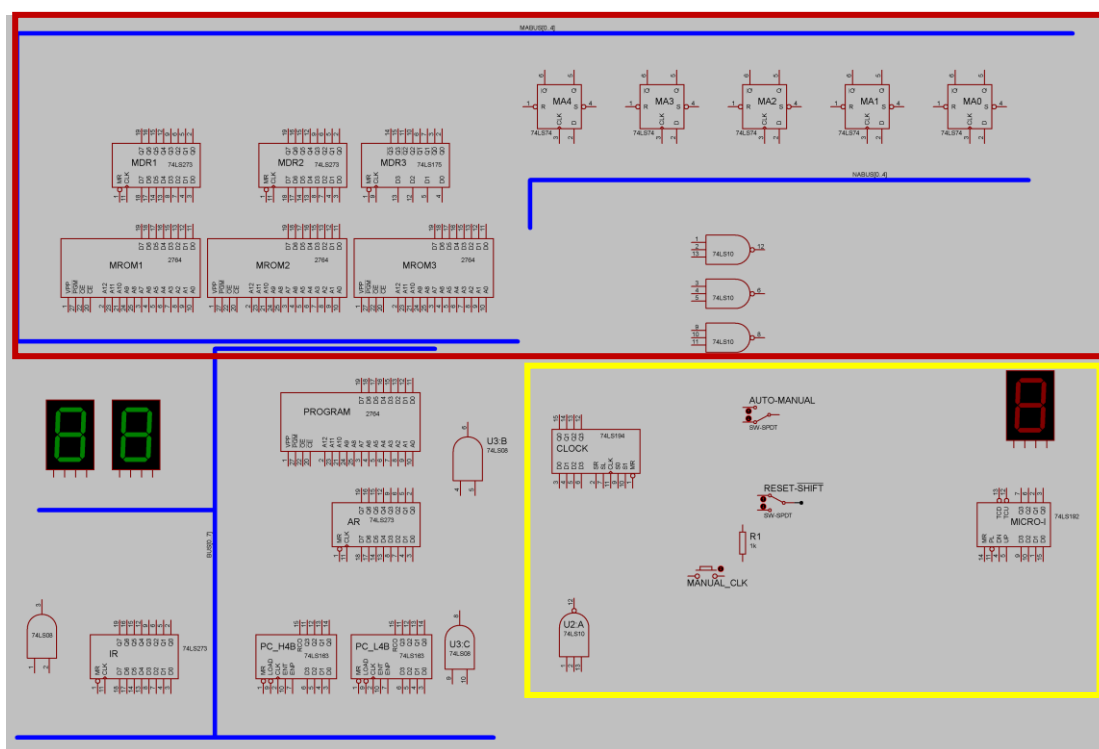


图 2-34 微程序控制器版本的 CPU

数据通路由三个部件组成：指令寄存器 IR（74LS273）、程序计数器 PC（74LS163 级联）、程序存储器 PROGRAM（ROM）及其地址寄存器 AR（74LS273）。所有部件都并联挂在单条的 8 位总线 BUS 上，通过数码管显示总线 BUS 信息。

其中，程序计数器 PC 是一个八位递增计数器，由两个同步二进制计数器 74LS163 级联构成。74LS163 的逻辑功能如表 2-13 所示：D<sub>0</sub>D<sub>1</sub>D<sub>2</sub>D<sub>3</sub> 为并行输入端；Q<sub>0</sub>Q<sub>1</sub>Q<sub>2</sub>Q<sub>3</sub> 为并行输出端；ENT、ENP 为递增使能端；LOAD 为置数端；MR 为清零端；CLK 为时钟输入端；RCO 为进位输出端。当低四位 74LS163 输出端 Q<sub>3</sub>-Q<sub>0</sub> 溢出后，则 RCO=1 送到高四位 74LS163 的 ENT 和 ENP 端，允许高四位 74LS163 在 CLK 上升沿自加 1 一次（仅允许一次）。注意：74LS163 的加载和自加 1 功能都必须满足 CLK 端上升沿跳变的条件才能实现。

表 2-13 计数器 74LS163 逻辑功能表

MR	LOAD	ENT	ENP	CLK	功能	Q <sub>0</sub> Q <sub>1</sub> Q <sub>2</sub> Q <sub>3</sub>
0	×	×	×	×	清除	0 0 0 0
1	0	×	×	↑	加载	Q <sub>0</sub> Q <sub>1</sub> Q <sub>2</sub> Q <sub>3</sub> = D <sub>0</sub> D <sub>1</sub> D <sub>2</sub> D <sub>3</sub>
1	1	0	×	×	保持	Q <sub>0</sub> Q <sub>1</sub> Q <sub>2</sub> Q <sub>3</sub>
1	1	×	0	×	保持	Q <sub>0</sub> Q <sub>1</sub> Q <sub>2</sub> Q <sub>3</sub>
1	1	1	1	↑	自加 1	{Q <sub>0</sub> Q <sub>1</sub> Q <sub>2</sub> Q <sub>3</sub> } 状态码+1

因为本次实验的数据通路简单，实现的功能有限。所以本实验的 CPU 只能使用四条基本指令：空指令 NOP、停机指令 HLT，以及直接寻址的跳转指令 JMP1 和间接寻址的跳转指令 JMP2，如表 2-14 所示。

表 2-14 微程序版 CPU 指令格式

指令	OP 码 I <sub>7</sub> I <sub>6</sub> I <sub>5</sub>	机器语言程序示例	指令功能说明
NOP	0 0 0	00000000; NOP	“空”指令，不执行任何操作
HLT	1 1 1	11100000; HLT	硬件停机
JMP1	0 0 1	00100000; JMP1 xxxxxxx; addr1	直接寻址：程序跳转一次到地址 addr1 执行 addr1 → PC;
JMP2	0 1 0	01000000; JMP2 xxxxxxx; addr1	间接寻址：程序跳转二次到地址 addr2 执行 [addr1] = addr2, addr2 → PC;

根据表 2-14 所示的 CPU 指令格式，可以编写机器语言（即二进制数据）形式的程序，存放在存储器 PROGRAM 中。程序按地址顺序存放（可以通过程序计数器 PC 递增寻址），每一个存储器单元地址上存储 8 位二进制数据，例如下表所示的这段程序：

存储器地址	数据(B)	汇编助记符	存储器地址	数据(B)	汇编助记符
00H	00100000	JMP1, 06H	06H	00000010	NOP/Addr
01H	00000110		07H	11100001	HLT
02H	11101010	HLT	08H	01000000	JMP2, [06H]
03H	00001010	NOP/Addr	09H	00000110	
04H	00000000	NOP	0AH	11111111	（未定义）
05H	00000000	NOP	0BH	11111111	（未定义）

表 2-14 所示的每一条指令，在其指令周期中都经历了取指周期和执行周期两个阶段，如图 2-35 所示。从图中可以看出，所有指令的取指周期都是一样的，执行周期则各有不同，甚至没有（例如 NOP 指令）。归纳起来所有指令在数据通路上出

现的操作只有两种：取指周期中的指令流 ROM->IR，执行周期中单次或多次重复出现的数据流 ROM->PC。

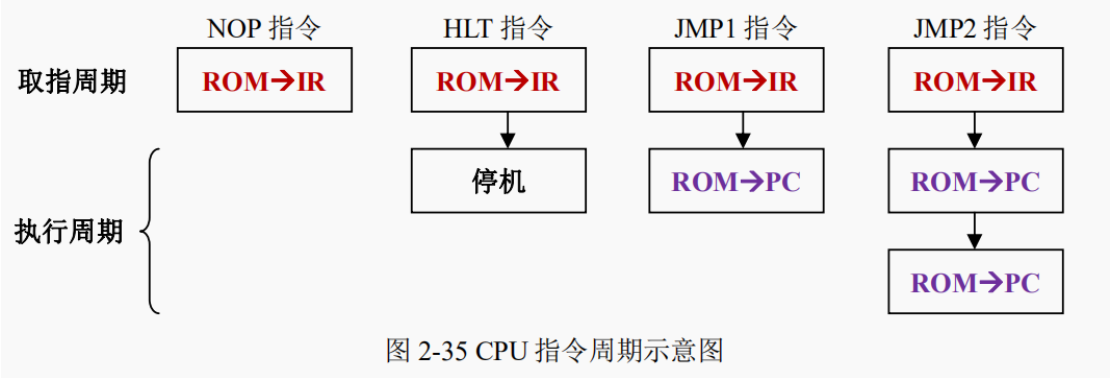


图 2-35 CPU 指令周期示意图

指令流（ROM->IR）是从存储器 PROGRAM 取出指令，经过总线 BUS 流向指令寄存器 IR，数据流（ROM->PC）是从存储器 PROGRAM 取出数据，经过总线 BUS 流向程序计数器 PC。无论是指令流还是数据流，信息都是先从一个部件打到总线 BUS，再从总线 BUS 打到另一个部件的过程。因为信息从源部件打出到总线 BUS 的操作必须先于信息从总线 BUS 打入目标部件的操作。所以为了保证上述操作的先后次序，指令流和数据流内部都可以分为两个周期 T1 和 T2。在 T1 周期，源部件（例：程序存储器 PROGRAM）的信息打入总线；在 T2 周期，总线的信息打入目标部件（例如指令寄存器 IR 或者程序计数器 PC）。

表 2-15 数据通路的微操作信号列表

电平触发信号	功能	边沿触发信号	功能
$\overline{OE}$	从存储器 PROGRAM 读出数据		
$\overline{LDPC}$	加载程序计数器 PC		
LDAR	加载地址寄存器 AR	$AR\_CLK=LDAR \cdot T1$	$\rightarrow AR$
LDIR	加载指令寄存器 IR	$IR\_CLK = LDIR \cdot T2$	$\rightarrow IR$
PC_INC	程序计数器 PC 递增或被加载	$PC\_CLK=PC\_INC \cdot T2$	$PC+1/\rightarrow PC$

数据通路上设计了一系列的微操作信号用来控制各个部件，如表 2-15 所示。其中，边沿触发信号是由电平触发信号与 T1 或 T2 周期对应的节拍信号逻辑“与”获得，确保其在 T1 或 T2 周期的开始可以产生上升沿跳变。在指令流或数据流的路径上，上述微操作信号组合起来，在 T1 或 T2 周期内完成特定的功能。如表 2-16 所示：

表 2-16 指令流/数据流的微操作信号列表

		有效的微操作信号		功能
指令流 ROM→IR	T1	$\overline{OE}$ , $AR\_CLK$ (LDAR)	$PC \rightarrow AR$ , $ROM \rightarrow BUS$	
	T2	$\overline{OE}$ , $IR\_CLK$ (LDIR), $PC\_CLK$ (PC_INC)	$BUS \rightarrow IR$ , $PC+1$	
数据流 ROM→PC	T1	$\overline{OE}$ , $\overline{LDPC}$ , $AR\_CLK$ (LDAR)	$PC \rightarrow AR$ , $ROM \rightarrow BUS$	
	T2	$\overline{OE}$ , $\overline{LDPC}$ , $PC\_CLK$ (PC_INC)	$BUS \rightarrow PC$	

具体分析一条 CPU 指令从取指到执行的全过程，其中最关键的问题是如何令 CPU 根据所取的指令，运行不同的指令周期，以及判断应该执行指令流，还是数据流。本实验运行“微程序”设计原理来解决这个问题：指令周期本身可以看作是一个“任务”，而其中的指令流（ROM->IR）或数据流（ROM->PC）可以看成是一条“指令”。若干条“指令”组合成不同的“程序”，就可以解决不同的“任务”。

上述的“指令”就是微指令，组成的“程序”就是微程序。本实验的微指令结构如图 2-36 所示：微指令字长 24 位，通过下址转移方式确定后续运行的微指令，即微指令的 1-5 位表示下一条微指令地址 [uA4, uA0]。而微指令的 6-7 位留给判断字段 P<sub>x</sub>（其中 P<sub>2</sub> 位空缺），P<sub>1</sub>=1，表示本微指令是取指周期的微指令；P<sub>1</sub>=0，表示本微指令是执行周期的微指令。微指令的 8-24 位则是微命令字段，微命令即是数据通路中电平触发的微操作信号（详见上表 2-15）。某位置“1”，表示该位微操作信号有效；反之，置“0”则表示该位微操作信号无效。

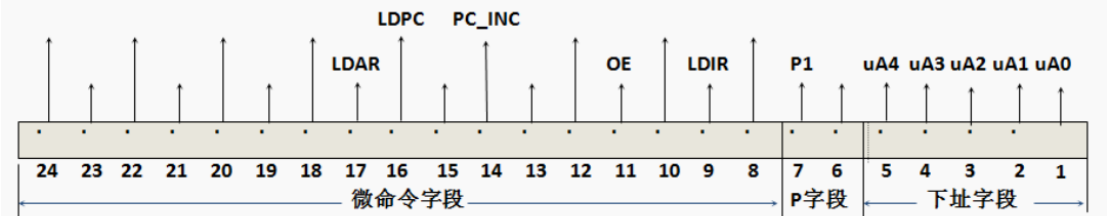


图 2-36 微指令结构图

因此，前述图 2-35 所示的指令周期中的一个方框在空间上对应一条特定的微指令，在时间上对应该微指令的周期，即微指令周期。图 2-35 中共有四种微指令，如表 2-17 所示。

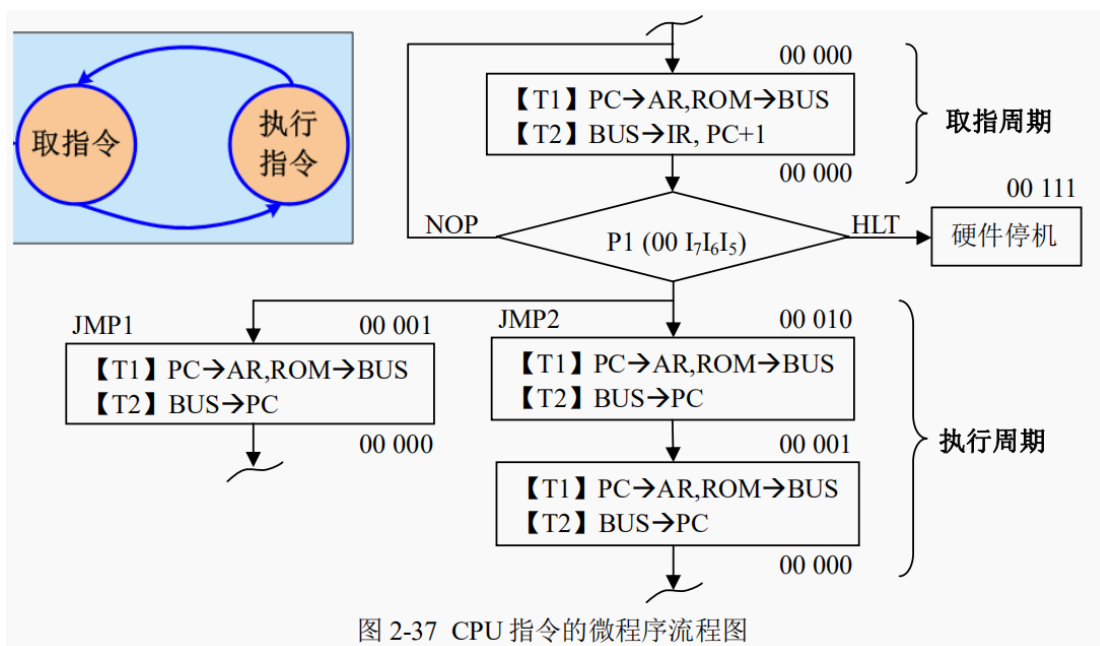
表 2-17 微指令代码表

Addr	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
00000	0	0	0	0	0	0	0	1	0	0	1	0	0	1	0	1	0	1	0	0	0	0	0	0
00001	0	0	0	0	0	0	0	1	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0
00010	0	0	0	0	0	0	0	1	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0	1
00111	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

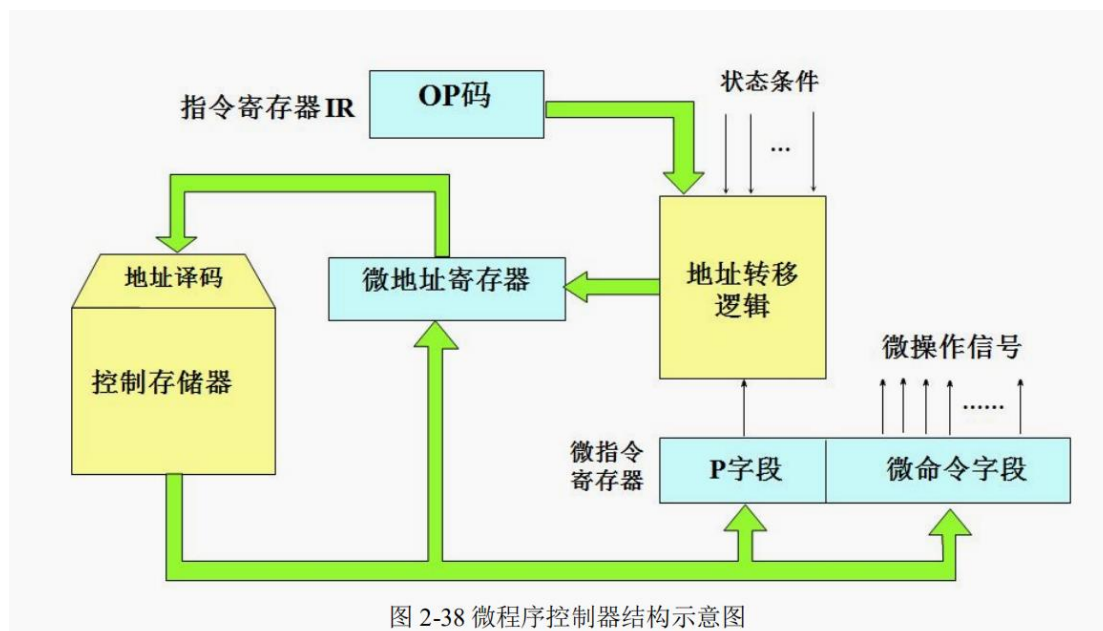
将 CPU 指令周期示意图 2-35 用“微程序”设计原理阐述，可以得到 CPU 指令的微程序流程图（图 2-37 所示）。图中每一个方框在时间上表示一个微指令周期，包括了 T<sub>1</sub>（源部件->总线）和 T<sub>2</sub>（总线->目标部件）两个周期；在空间上每一个方框表示一条微指令，通过一系列微操作信号的执行使得数据从某个源部件经过总线 BUS 到达另一个目标部件。图中每个方框的右上方是当前微指令的地址，右下方是下一条微指令的地址（简称“下址”）。

如图 2-37 所示，微程序流程图最上方首先执行的方框是所有 CPU 指令都需要的取指微指令，即指令流（ROM->IR）。取出指令后，P<sub>1</sub> 菱形框表示指令译码及地址转移：CPU 指令集总共有四条不同的指令，必须根据指令的 OP 码的 I7I6I5 位判断取出的是哪一条 CPU 指令，从而选择菱形框之下的不同路径（入口地址 [00I7I6I5]）继续执行。菱形框之下的每条路径都对应不同 CPU 指令的执行周期，其中每个方框就是一条执行微指令，即数据流（ROM->PC）。值得注意的是，NOP 指令和 HLT 指令比较特殊，只有取指周期，没有执行周期。NOP 指令的 OP 码是 000，取指后译码得到的执行周期第一条微指令地址仍为[00000]，即直接返回下一条指令的取指周期。而 HLT 指令的 OP 码是 111，译码后直接令硬件停机。

在所有路径末尾，最后一条微指令的下址[uA4-uA0]都必须是取指微指令地址 [00000]，即一条 CPU 指令结束后必须返回取指周期，准备取出下一条 CPU 指令。CPU 的运行过程就是不断的循环：取指令、执行指令、取指令、执行指令……。如图 2-37 所示。



负责执行上述微程序流程图 2-37 的 CPU 部件就是微程序控制器，其结构由控制存储器、微指令寄存器、微地址寄存器、和地址转移逻辑电路组成，如图 2-38 所示。CPU 启动或重启后，微地址寄存器清零，控制存储器从地址[00000]开始读出取指周期的第一条微指令（注：取指微指令必须从控制存储器首地址[00000]开始）。如前述图 2-36 所示，微指令结构包括了控制字段、下址字段 uA4-uA0 和判断字段 (P1)。控制字段直接锁存形成该微指令执行的微命令信号（微操作信号），下址字段锁存微地址寄存器，待到下一个微指令周期开始即可取出下一条微指令。若当前微指令是取指周期末尾微指令，则 P1=1，启动地址转移逻辑，根据指令寄存器 IR 中的 OP 码修改微地址寄存器，转向该指令执行周期第一条微指令。



微程序控制器电路的控制存储器字长 24 位，由 3 个 2764 芯 MROM1-3 组成；微指令代码表 2-17 中的四条微指令都存储在控制存储器中，控制存储器的输出端则连接着微指令寄存器 MDR1-MDR3（由两个 74LS273 寄存器和一个 74LS175



触发器组成)。在 T1 节拍, 控制存储器的当前微指令打入微指令寄存器锁存, 送往数据通路执行。然后, 部分电平触发的微操作信号 (LDIR、PC\_INC) 再与 T2 节拍组合, 产生边沿触发信号 (IR\_CLK、PC\_CLK), 在 T2 周期执行“总线->目标部件”的操作。

如图 2-40 所示,微地址寄存器字长五位(MA4-MA0), 由触发器 74LS74 组成, 其输入端通过 NMABUS 总线连接当前微指令的下址字段[uA4-uA0], 其输出端则通过控制存储器的地址总线 MABUS 送到控制存储器的地址端 A4-A0。而微程序控制器结构示意图 2-38 中的地址转移逻辑其实就是微程序流程图 2-37 中的菱形框 P1, 其逻辑电路由三个三路与非门实现。在取指周期末尾, 微指令下址字段本来[00000]; 然而 P1=1, 启动地址转移逻辑, 根据指令寄存器 IR 的 OP 码 (I7I6I5 位) 生成信号  $\overline{\text{SET\_MAX}}=0$ , 强制把微地址寄存器 MA4-MA0 置位为[00I7I6I5], 即该指令执行周期的第一条微指令地址。

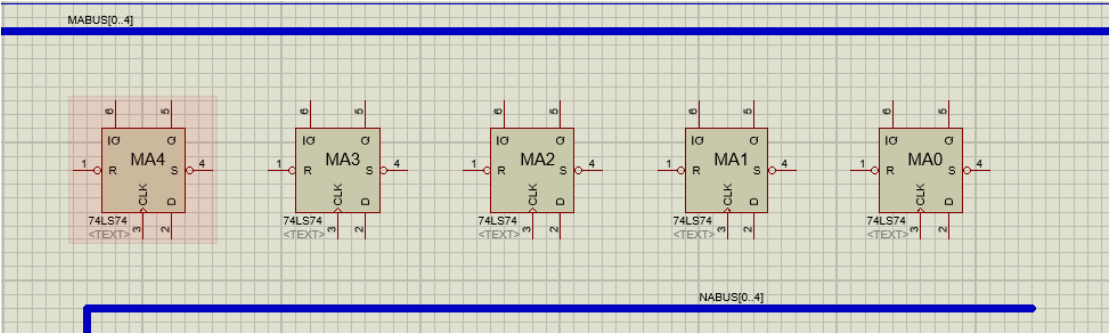
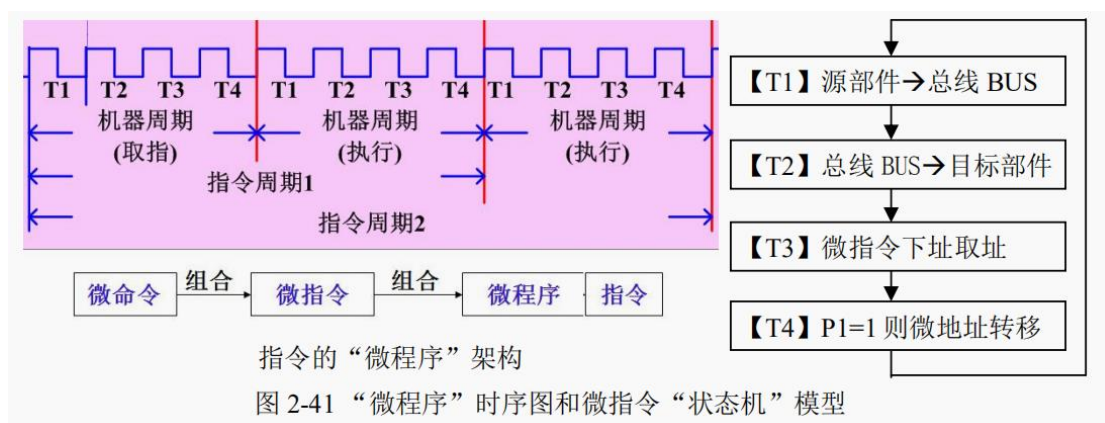


图 2-40 微地址寄存器和地址转移逻辑

微程序的地址转移过程需要在微指令周期增加 T3 和 T4 两个周期。在 T3 周期, 当前微指令的下址字段[uA4-uA0]通过 NMABUS 总线打入微地址寄存器 MA4-MA0 (如图 2-40 所示), 进而通过地址总线 MABUS 送往控制存储器地址端, 使其输出下一条微指令。在 T4 周期, 若当前微指令是执行周期微指令, 则 P1=0, 无任何操作; 若当前微指令是取指周期微指令, 则 P1=1, 触发地址转移逻辑, 重置微指令地址, 跳转到当前指令的执行周期入口。

因此, 如图 2-41 (右) 所示, 一条微指令的运行过程可以看成是一个简化的 Moore 状态机: 4 个周期 {T1, T2, T3, T4} 分别对应四个状态【Tx】, 每个状态【Tx】中完成相应的任务, 状态“T1-> T4”的一次循环即是微指令从取指、判断 (分支) 到执行的全过程,

综上所述, 一条 CPU 指令即是一段微程序, 包含了若干条微指令 (其中至少有一条取指微指令)。每条微指令又由一系列并行输出的微命令信号构成。从时间的角度来看, 如图 2-41 (左) 所示: 每个指令周期都包含了若干等长的微指令周期 (即机器周期), 其中必有一个取指微指令周期; 每个微指令周期又包含了等长的四个 Tx 节拍。在每个微指令周期中, 微指令“状态机”周而复始在四个状态【Tx】间顺序转移, 如图 2-41 (右) 所示。在每个状态【Tx】中, 状态信号 Tx 与当前微指令寄存器并行输出的微命令信号结合, 在数据通路上完成一系列的微操作, 实现状态【Tx】方框中所描述的任务。



本实验的时序发生器电路中 CLK 为全系统的基准时钟信号，可以由方波信号 AUTO\_CLK 提供（双击信号源可以自行选择方波信号频率），或者通过人工操作按键 MANUAL\_CLK 手动步进；移位寄存器 74LS194 用来作为节拍生成器：当 HLT=1 且 RESET=0，则 74LS194 状态{S0,S1}={1,0}；SR=T4，74LS194 是循环右移模式。以 CLK 为基准时钟，74LS194 周而复始地发送节拍信号 T1-T4。每个节拍信号 Tx 对应上图 2-40（右）所示“状态机”的状态【Tx】，时钟 CLK 驱动其状态转移：T1->T2->T3->T4->T1->.....“T1->T4”的一次循环称为一个 CPU 周期（即微指令周期）。

时序发生器还提供了硬件电路实现 HLT 指令的停机功能（即“断点”）。当指令寄存器 IR 的 OP 码 I7I6I5=111 的时候，硬件电路生成  $\overline{\text{HLT}}=0$ 。此时，74LS194 状态{S0,S1}={0,0}，工作模式是保持，输出{T1,T2,T3,T4}={0,1,0,0}，停机在 HLT 指令的取指周期 T2 节拍上。

此外，为了观测微程序的运行，时序发生器提供了一个由 T1 上升沿驱动的 74LS192 计数器 MICRO-I，通过数码管显示当前运行第几条微指令。

启动仿真后，CPU 初始化过程如下所示：

- 1) 时钟 CLK 接在 MANUAL\_CLK 端，令 RESET=1，则  $\overline{\text{CLR}}=0$ ，清零微地址寄存器 Max (x=1, 2, 3) 和指令寄存器 IR。此时， $\overline{\text{HLT}}=1$ ，74LS194 状态{S0,S1}={1,1}，工作模式是送数。
- 2) 手动按钮 MANUAL\_CLK  $\neg$ ，令 CLK 端上升沿跳变，节拍 {T1,T2,T3,T4}={1,0,0,0}。
- 3) 令 RESET=0，74LS194 恢复循环右移模式，进入第一条指令的取指周期节拍时序。

跳出 HLT 指令“断点”的重启过程则有所简化，只需要上述步骤 1 和 3，且 74LS194 恢复循环右移模式（跳出“断点”）后进入的是 HLT 指令后续下一条指令的取指周期。

## 五、实验步骤

- 1) 根据表 2-17 微指令代码表可以编写下列微程序，编译并生成三个 HEX 文件，分别烧写到图 2-34 所示的控制存储器 EPROM1、EPROM2 及 EPROM3（切记勿写错存储器！）。

EPROM1 烧写内容	EPROM2 烧写内容	EPROM3 烧写内容
ORG 0000H	ORG 0000H	ORG 0000H
DB 00000001B	DB 00100101B	DB 01000000B
DB 00000001B	DB 10100100B	DB 00000000B
DB 00000001B	DB 10100100B	DB 00000001B
DB 00000000B	DB 00000000B	DB 00000000B
DB 00000000B	DB 00000000B	DB 00000000B
DB 00000000B	DB 00000000B	DB 00000000B
DB 00000000B	DB 00000000B	DB 00000000B
DB 00000000B	DB 00000000B	DB 00000000B
END	END	END

2) 问题：在写入控制存储器的微指令代码表中，地址[00001]和[00010]的两条执行周期微指令执行的微操作完全一样。请问，可否合并这两条微指令？若不能，原因是什么？

编译如下所示的机器语言源程序，生成 HEX 文件烧写到图 2-34 所示的程序存储器 PROGRAM 中（编译和烧写 asm 文件的方法参见“存储器实验：ROM 批量导入数据”）。

ORG 0000H

```

DB 00100000B ; JMP1,06H
DB 00000110B
DB 11101010B ; HLT
DB 00001010B ; NOP/Addr


DB 00000000B
DB 00000000B
DB 00000010B ; NOP/Addr
DB 11100001B ; HLT

DB 01000000B ; JMP2,[06H]
DB 00000110B

```

END

1、启动仿真前，时钟信号 CLK 接在 MANUAL\_CLK 端；启动仿真，使能复位信号 RESET=1，然后手动按钮 MANUAL\_CLK，令时钟信号 CLK 上升沿跳变，初始节拍{T1,T2,T3,T4}={1,0,0,0}；最后恢复复位信号 RESET=0，初始化过程完成。

2、手动按钮 MANUAL\_CLK，令时钟信号 CLK 形成  脉冲，单步执行上述机器语言程序。在 JMP1 或 JMP2 指令的指令周期中，对照微程序流程图 2-37，观察每一条微指令的作用，以及单步执行的结果（例如寄存器 AR、IR、PC 及总线 BUS 上的数据）。

3、时钟信号 CLK 改接在 AUTO-CLK 信号源（主频 10Hz），程序会自动运行到 HLT 指令“断点”暂停。查看“断点”处的微指令周期数指示，以及寄存器 AR、IR、PC 及总线 BUS 上的数据。然后，使能复位信号 RESET“0->1->0”，跳出“断点”，进入 HLT 指令的后续下一条指令继续运行。



实验 2：新指令 JMP3

增加一条二次间接寻址的跳转指令 JMP3，如下所示。请补充微程序流程图 2-37 及微指令代码表 2-17（新增微指令地址[00011]），实现下述 JMP3 指令的功能。

指令	OP 码	机器语言程序	指令注释
JMP3	011	01100000; JMP3 xxxxxxxx; addr1	二次间址：程序跳转三次到地址 addr3 执行 [addr1] =addr2, [addr2]= addr3, addr3→ PC;

编译如下所示的机器语言源程序，生成 HEX 文件烧写到图 2-34 所示的程序存储器 PROGRAM 中（编译和烧写 asm 文件的方法参见“存储器实验：ROM 批量导入数据”）。

```
ORG    0000H

        DB    00100000B ; JMP1,06H
        DB    00000110B
        DB    11101010B ; HLT
        DB    00001010B ; NOP/Addr

        DB    01100000B ; JMP3,[[0BH]]
        DB    00001011B
        DB    00000010B ; NOP/Addr
        DB    11100001B ; HLT

        DB    01000000B ; JMP2,[06H]
        DB    00000110B
        DB    11100000B ; HLT
        DB    00000011B ; NOP/Addr
```

END

参照实验 1 所述的初始化、手动单步执行的方法，单步执行上述机器语言程序。在 JMP3 指令的指令周期中，对照其微程序流程图，观察每一条微指令的作用，以及单步执行的结果（例如寄存器 AR、IR、PC 及总线 BUS 上的数据）。

参照实验 1 所述的自动运行以及跳出“断点”的方法，自动运行上述机器语言程序到 HLT 指令“断点”暂停。查看“断点”处的微指令周期数指示，以及寄存器 AR、IR、PC 及总线 BUS 上的数据。

问题：在本实验程序中，有部分地址标示“NOP/[ADDR]”，为何相同代码会有不同的执行效果？执行到该处，在什么情况下是不执行任何操作？在什么情况下是程序跳转？

## 六、思考题

1、微程序版本 CPU 最多有多少条微指令？最多有多少条 CPU 指令？微指令和 CPU 指令的容量分别由什么因素限定？

2、请问微程序控制器“状态机”可否提升效率，减少到三个状态{T1,T2,T3}？即微指令周期可否减少到只用 T1、T2、T3 三个节拍即可完成一条微指令从取指到执行的全过程？

3、在本次实验的 CPU 模型机上增加两个 74LS173 寄存器 R1 和 R2，以及一个连接总线 BUS 的 8 位拨码开关，扩展 CPU 指令集，增加下述 MOV/SET 指令及相应的微指令：

汇编助记符	功能	I <sub>7</sub> I <sub>6</sub> I <sub>5</sub> I <sub>4</sub>	I <sub>3</sub> I <sub>2</sub>	I <sub>1</sub> I <sub>0</sub>
MOV RA, RB;	(RB)→RA	0110	RA	RB
SET RA, IMM;	IMM→RA	0011	RA	x/x
		IMM		

（注：IMM 是由拨码开关输入的 8 位立即数；RA 和 RB 是在指令“功能”描述中的逻辑寄存器，可以对应 R0 或 R1 寄存器。）

4、在思考题 3 的电路基础上，参考上述“运算器实验”，增加 74LS181 运算器电路，扩展 CPU 指令集，增加下述 ADD/SUB/AND/OR/XOR 指令及相应的微指令：

汇编助记符	功能	I <sub>7</sub> I <sub>6</sub> I <sub>5</sub> I <sub>4</sub>	I <sub>3</sub> I <sub>2</sub>	I <sub>1</sub> I <sub>0</sub>
ADD RA, RB;	(RA) + (RB)→RA	1101	RA	RB
SUB RA, RB;	(RA) - (RB)→RA	1100	RA	RB
AND RA, RB;	(RA) ∧ (RB)→RA	1110	RA	RB
OR RA, RB;	(RA) ∨ (RB)→RA	1111	RA	RB
XOR RA, RB;	(RA) ⊕ (RB)→RA	1011	RA	RB