

# 上海大学 计算机学院

## 《计算机组成原理实验》报告九

姓名 孔馨怡 学号 22122128

时间 周一 9-11 机位 17 指导教师 顾惠昌

---

实验名称：程序转移机制

### 一、实验目的

1. 学习实现程序转移的硬件机制。
2. 掌握堆栈寄存器的使用。

### 二、实验原理

#### 1. 程序转移

在任何程序段的内部，执行流程有顺序、分支、循环三种，而程序段之间又有相互调用（例如：调用子程序、中断服务、子程序返回、进程调度、任务切换……），看似很复杂，其实计算机硬件用非常简单的技术解决了这些问题。

分支和循环总是可以相互替代，所以也常说程序段内的执行流程其实仅有顺序和转移两种，而程序段之间的调用也只是把执行流程转移到了另外一个程序段上。所以，任何复杂的程序流程，在硬件实现机制上只有两种情况：顺序执行和转移。硬件实现这两种情况的技术很简单：

对 PC 寄存器的自动加 1 功能实现程序顺序执行。

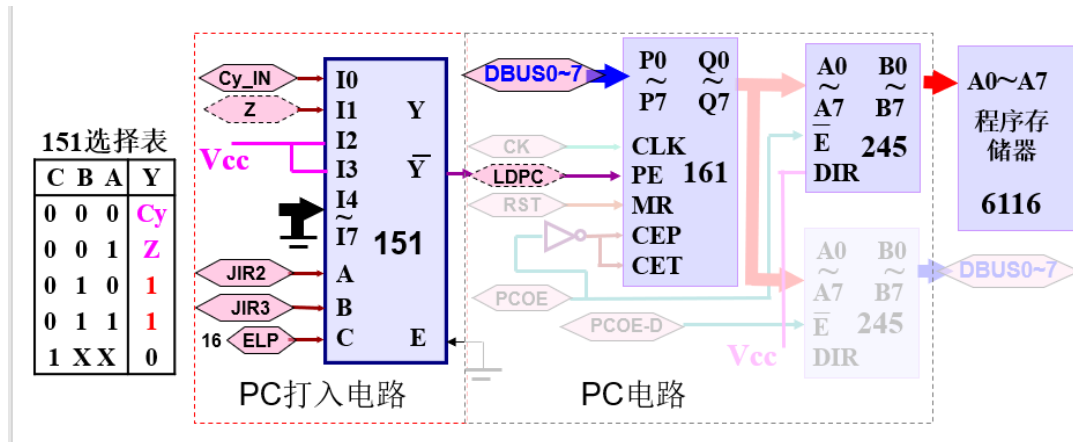
对 PC 寄存器的打入初值功能实现程序转移。

当转移目标为本段内未执行过的指令时就形成分支；当转移目标是本段内执行过的指令时就形成循环；当转移目标为其他段的指令时就形成段间调用。可见：转移操作决定于“给 PC 赋值”，而转移类型决定于“所赋的值同当前指令的关系”。

## 2. 实验箱系统的程序转移硬件机制

当 LDPC 有效（即：LDPC=0）时，如果此时 DBUS 上的值就是转移的目标地址，则此目标地址被打入 PC（即：PC 被赋新值），从而实现程序的转移。

若 LDPC 为 0 是附带条件的, 就形成“条件转移”。实验箱依靠“PC 打入电路”实现“有进位”时转移和“计算结果为零”时转移, 以及无条件转移。

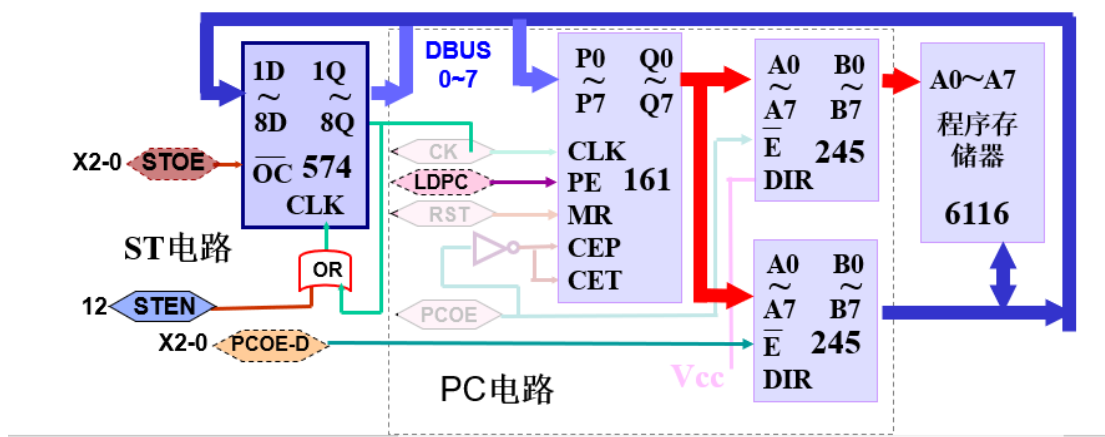


### 3. ST 寄存器结构和子程序调用与返回控制信号

实验箱子程序调用和返回的结构由 PC 电路和 ST 电路组成。

(1) 当调用子程序时, PC 的当前值 (即: 断点地址) 经下面的 245 送上 DBUS, 进入 ST 保存。然后给 PC 打入子程序入口地址, 该子程序入口地址是由调用指令自身携带的目标地址, 至此转子程序过程完成。

(2) 当子程序返回 (RET) 时, 返回指令开启 ST 的输出, 并给出 PC 打入信号 (无条件转移), 于是 ST 保存的断点经由 DBUS 打入 PC, 实现子程序返回。



### 三、实验内容

#### 1. 实验任务一：试用手动方式实现子程序调用转移过程。

假设调用子程序指令的下一条指令存放在 11H 单元，子程序的入口地址为 22H。

实验任务二：试用手动方式实现子程序返回转移过程。假设调用子程序指令的下一条指令存放在 11H 单元，子程序的入口地址为 22H 。

##### (1) 实验步骤、实验现象

- 前期准备和连线
- 按照实验需求，以及实验箱功能设置连线，并提前设置初始状态（按照下表）

接口	信号孔	功能	有效电平	初始状态
K0	X0	寄存器输出选择	----	0
K1	X1	寄存器输出选择	----	0
K2	X2	寄存器输出选择	----	0
K5	ELP	将数据打入 PC 寄存器	低电平有效	1
K7	STEN	启用 ST 存储器 将数据打入 ST 寄存器	低电平有效	1
K8	AEN	启用 A 寄存器 将数据打 A 寄存器	低电平有效	1
• • •	• • •	• • •	• • •	• • •

提前设置初始状态							
K23	K22	K21	K20	K19	K18	K17	K16
0	0	0	0	0	0	0	0

八位扁平线连接		
J1	J2	J3
连接	连接	

- 打开电源，按 TV/ME 三次设置进入手动模式
- 将 11H 立即数写入 PC 寄存器

按照下面表格设置立即数 11H，作为准备传入的立即数

11H							
K23	K22	K21	K20	K19	K18	K17	K16
0	0	0	1	0	0	0	1

**分析：**

要写入 PC 寄存器，所以将 ELP 启用；

11H 从立即数传入，所以选择 IN\_OE 外部输入门， $X2X1X0 = 000$ ；

即按照下表设置：

K8	K7	K5	K2	K1	K0
AEN	STEN	ELP	X2	X1	X0
1	1	0	0	0	0

●按下 STEP 键, **观察实验现象如下：**

PC 寄存器黄灯亮起, IN 门红灯亮起，而后 PC 显示 11，录入成功。

●将 22H 写入 A 寄存器

按照下面表格设置立即数 22H，作为准备传入的立即数

22H							
K23	K22	K21	K20	K19	K18	K17	K16
0	0	1	0	0	0	1	0

**分析：**

不写入 PC 寄存器，所以将 ELP 关闭；

要写入 A 寄存器，所以将 A 寄存器启用；

22H 从立即数传入，所以选择 IN\_OE 外部输入门， $X2X1X0 = 000$ ；

即按照下表设置：

K8	K7	K5	K2	K1	K0
AEN	STEN	ELP	X2	X1	X0
1	1	0	0	0	0

●按下 STEP 键, **观察实验现象如下：**

A 寄存器黄灯亮起, IN 门红灯亮起，而后 A 显示 22，录入成功。

●将 PC 寄存器数据写入 ST 寄存器

**分析：**

不写入 PC 寄存器，所以将 ELP 关闭；

要写入 ST 寄存器，所以将 STEN 启用；

数据从 PC 寄存器传入，所以选择 PC\_OE PC 寄存器， $X2X1X0 = 011$ ；

即按照下表设置：

K8	K7	K5	K2	K1	K0
AEN	STEN	ELP	X2	X1	X0
1	0	1	0	1	1

●按下 STEP 键, **观察实验现象如下：**

ST 寄存器黄灯亮起, PC 红灯亮起，而后 ST 显示 11，录入成功。

●将 A 寄存器数据写入 PC 寄存器

**分析：**

写入 PC 寄存器，所以将 ELP 启用；

不写入 ST 寄存器，所以将 STEN 关闭；

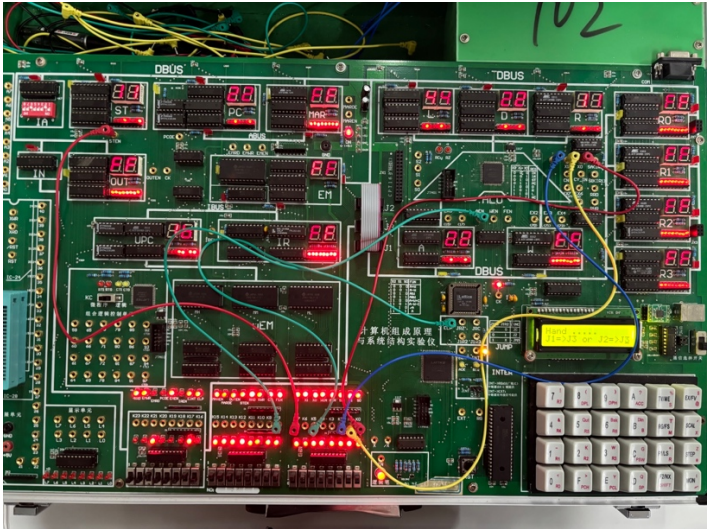
数据从 A 寄存器传入，所以选择 D\_OE 直通门， $X2X1X0 = 100$ ；

即按照下表设置：

K8	K7	K5	K2	K1	K0
AEN	STEN	ELP	X2	X1	X0
1	1	0	1	0	0

●按下 STEP 键, 观察实验现象如下:

PC 寄存器黄灯亮起, D 红灯亮起, 而后 PC 显示 22, 录入成功。



●将 ST 寄存器数据写入 PC 寄存器

分析:

写入 PC 寄存器, 所以将 ELP 启用;

不写入 ST 寄存器, 所以将 STEN 关闭;

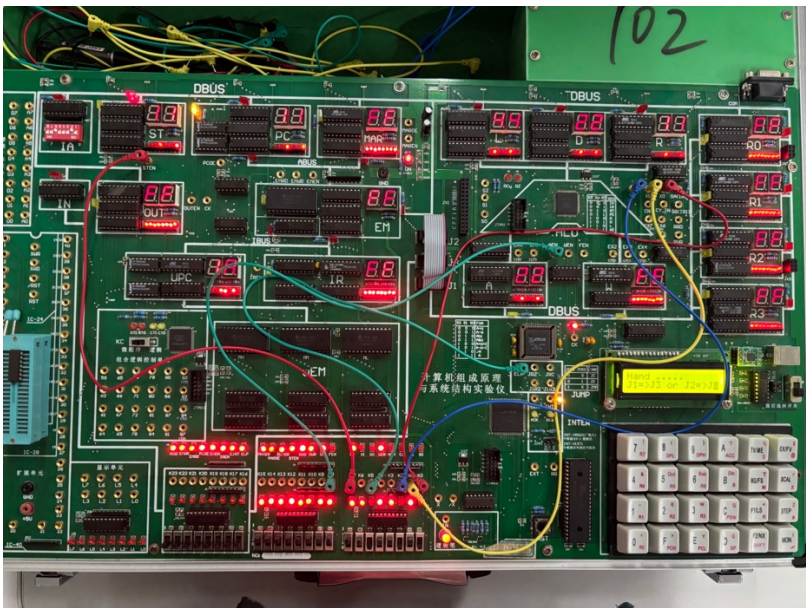
数据从 ST 寄存器传入, 所以选择 ST\_OE 堆栈寄存器,  $X2X1X0 = 010$ ;

即按照下表设置:

K8	K7	K5	K2	K1	K0
AEN	STEN	ELP	X2	X1	X0
1	1	0	0	1	0

●按下 STEP 键, 观察实验现象如下:

PC 寄存器黄灯亮起, ST 寄存器红灯亮起, 而后 PC 显示 11, 录入成功。





## (2) 数据记录、分析与处理

对应的分析、实验现象、数据记录均在 (1) 中体现 此处略。(为了呈现清晰明了, 与实验步骤一同叙述)

## (3) 实验结论

成功用手动方式实现子程序调用转移过程

成功用手动方式实现子程序返回转移过程。假设调用子程序指令的下一条指令存放在 11H 单元, 子程序的入口地址为 22H 。

2. 实验任务二: 编程实现 OUT 寄存器交替显示  $8N$  和  $8N+1$  ( $N$  为 1, 3, 5, 7, 9...), 当  $8N$  大于 FFH 时, OUT 显示 EE. 交替时间为 0.5s 一次。(实验箱的工作频率为: 114.8Hz。)

分析: 编写一个延时子程序, 实现 1 秒钟的时间延时。

编写一个主程序, 实现在 OUT 寄存器中交替显示。

### (1) 实验步骤

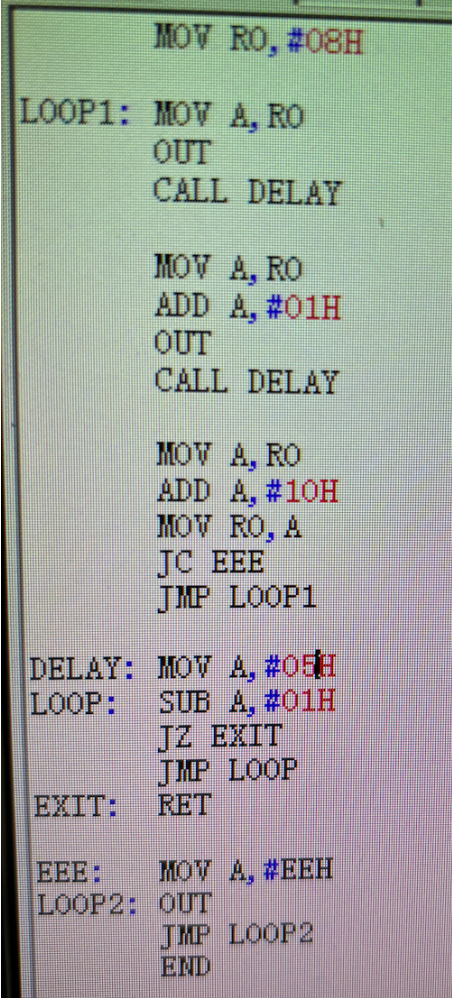
- 1、根据分析编写后缀为 asm 的程序文件:

```
MOV R0, #08H
LOOP1: MOV A, R0
OUT
CALL DELAY

MOV A, R0
ADD A, #01H
OUT
CALL DELAY

MOV A, R0
ADD A, #10H
MOV R0, A
JC EEE
EMP LOOP1

DELAY: MOV A, #05H
LOOP: SUB A, #01H
JZ EXIT
JMP LOOP
```



```
MOV R0, #08H
LOOP1: MOV A, R0
OUT
CALL DELAY

MOV A, R0
ADD A, #01H
OUT
CALL DELAY

MOV A, R0
ADD A, #10H
MOV R0, A
JC EEE
JMP LOOP1

DELAY: MOV A, #05H
LOOP: SUB A, #01H
JZ EXIT
JMP LOOP

EXIT: RET

EEE: MOV A, #EEH
LOOP2: OUT
JMP LOOP2
END
```

```
EXIT:  RET

EEE:   MOV A, #EEH
LOOP2: OUT
      JMP LOOP2
      END
```

2、打开软件，点击连接通信口，将 asm 文件导入进来，然后点击编译，进行运行，观察实验箱和软件内结构图各寄存器到变化。

(2) 实验现象、数据记录、分析与处理

编写逻辑和程序分析如下：

程序		分析
	MOV R0, #08H	初始值
LOOP1:	MOV A, R0	赋值 A
	OUT	输出 8N
	CALL DELAY	子程序：延时开始
	MOV A, R0	赋值 A
	ADD A, #01H	加一操作
	OUT	输出 8N+1
	CALL DELAY	子程序：延时开始
	MOV A, R0	赋值 A
	ADD A, #10H	N+2 的操作 <a href="#">详细查看解释 1</a>
	MOV R0, A	存储新的值
	JC EEE	完成“是否大于 FF 的操作” <a href="#">详细查看解释 2</a>
	EMP LOOP1	没有进位继续循环
DELAY:	MOV A, #05H	延时开始 <a href="#">详细查看解释 3</a>
LOOP:	SUB A, #01H	减一操作
	JZ EXIT	为 0 结束
	JMP LOOP	否则继续
EXIT:	RET	子程序结束
EEE:	MOV A, #EEH	结束程序开始
LOOP2:	OUT	输出 EE
	JMP LOOP2	循环完成假停机 <a href="#">详细查看解释 4</a>
END		END 结束

### 解释 1:

交替显示  $8N$  和  $8N+1$  ( $N$  为 1, 3, 5, 7, 9...), 当  $N+2$  时, 实际加了  $8*2=16$ , 在 16 进制中就是 10。

所以在  $8N$ 、 $8N+1$  交替显示时, 依次为 08、09、18、19、28、29、...、E8、E9、F8、F9。

### 解释 2:

当  $8N+1$  为 F9 时, 下一次的相加将会导致运算器进位, 因为超出了 FF 的最大值, 所以我们判断是否大于 FF 的操作就是先加 10H, 然后判断进位, 如果进位标志没有亮, 则说明还要继续; 否则就是一件大于 FFH 了, 结束循环。

### 解释 3:

**如何计算延时所需的次数和时间:**

根据频率 114.8Hz, 再根据公式: 频率\*周期=1, 求出周期, 然后可以得出 1s 内会需要多少次

周期  $T = 1/f = 1/114.8 = 0.0087108 \text{ s}$  ;

那么一秒就需要 11 次。(只是此时为了在实验中具体调整, 所以设置为 0.5s 的延时)

### 解释 4:

在程序结束以后其实机组还会继续跳转, 出现各寄存器的小屏幕闪过的现象, 此时在程序最后设置一个死循环, 来完成假停机, 这样看起来就像停机了, 不会乱闪。

### (3) 实验结论

成功利用 CP226 汇编语言程序集成开发环境编写程序完成交替显示  $8N$  和  $8N+1$  ( $N$  为 1, 3, 5, 7, 9...), 当  $8N$  大于 FFH 时, OUT 显示 EE. 交替时间为 0.5s 一次。

## 四、建议和体会

1. 在做类似任务一这样手动进行数据的输入, 寄存器的写入等的操作时, 每次录入数据之后要记得将写入/输入的开关关闭, 并检查黄灯是否亮起, 以免刚刚录入的数据被重新写了覆盖了。
2. 在做类似任务一这样手动进行数据的输入输出时, 要检查有几个红灯, 也就是检查到底有几个输出, 若有两个及以上的输出, 那么输出的数据可能就不是自己想要的了。
3. 在做任务二的时候多多注意实验箱和电脑软件上结构图/逻辑分析等的状态变化, 数据变化等的信息, 在逻辑图上可以查看数据/指令的走向, 便于理解。
4. 千万不要弄错了逻辑/微指令的选择, 不仅仅是电脑软件上选择好就可以了, 还要注意实验箱上有没有选择正确, 如果发生了类似的情况的话, 注意观察数据的变化, 是否有通路没有数据, 来检查有无错误。
5. 在做 CP226 汇编语言程序集成开发环境下编写程序时, 要多多熟悉开发的环境, 例如 asm 文件的编写是不能有错误的, 装载的设备是否正确等的问题。
6. 拼写问题: 例如 JMP 不要写成 JUMP,



很多时候拼写问题不易检查出来，但是影响到整个实验  
如果在装载后看到电脑提示：不认识某标志符或者是某某指令是无效的 就很有可能是拼写出了问题

- 7. END 在 asm 文件中只能出现一次 就算有些程序跑不到 END 也只能写一次 它代表了程序到此就结束了
- 8. 程序在跑完想要的操作时

五、思考题

若要求 11 和 55 各显示 50 次后停机，应该如何修改程序？

答：

行数	指令	操作数
1	MOV	W,#50H
2	LOOP1:	
3	MOV	A,#11H
4	OUT	
5	CALL	DELAY
6	MOV	A,#55H
7	OUT	
8	CALL	DELAY
9	SUB	W,#01H
10	JZ	STOP
11	JUMP	LOOP1
12	DELAY:	
13	MOV	A,#11H
14	LOOP:	
15	SUM	A,#01H
16	JZ	EXIT
17	JUMP	LOOP
18	EXIT	RET
19		END