北 京 邮 电 大 学

实 验 报 告

课程名称_计算机组成原理课程实验_

实验名称___CPU&程序中断实验____

_计算机科学与技术_学院_2020211305_班 姓名_马天成_

教师_靳秀国老师_ 成绩_____

_2022_年_06_月_02_日

目录

实验五：CPU 实验

实验目的

通过 TEC-8 执行由机器指令组成的简单程序，理解计算机如何取出指令、如何执行指令如何在一条指令执行结束后自动取出下一条指令并执行，牢固建立的计算机整机概念

实验要求

将简单程序进行译码，按指令格式汇编成二进制机器代码。完成控制台、时序部件、数据通路和微程序控制器之间的连线。 将程序机器代码利用控制台指令写入内存。根据程序的需要设置通用寄存器堆中相关寄存器的数据。单拍方式执行一遍程序，记录相关寄存器和存储器存储单元数据，与理论值比较分析。连续方式再次执行一遍程序，记录相关寄存器和存 储器存储单元数据，与理论值比较分析。

第一步：写寄存器

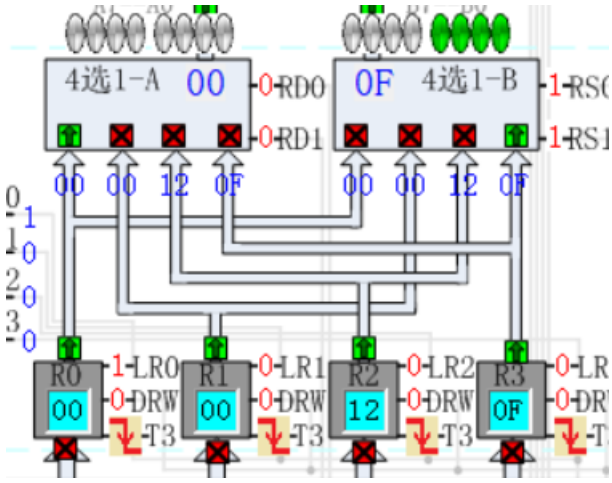
（控制台 100，写四个寄存器。不用写的写 00H，写的打数据开关，QD 单拍触发）

实验过程如下：

- 打开控制台为 100，转为单拍 DP 触发
- 初始 μ 为 000000，Nμ 为 000001，找不到相应功能模块，但下一条是取指。
- 按下 QD 触发，因为控制台方式是 100，转写寄存器操作，μ 变成 001001，Nμ 变成 001000，为写 R0 操作模块。
- 因为不用写 R0,，等价于写 00H，所以没有任何其他操作，按下 QD 触发， μ 变成 001000，Nμ 变成 001010，转到写 R1 操作模块。
- 同样，无需其他操作。不动数据开关，直接写入 00H，并按下 QD 触发，μ 变成 001010，Nμ 变成 001100，转到写 R2 操作模块。
- 此时将数据开关打成 12H,，然后按下 QD。12H 被写入 R2 寄存器，并按下 QD 触发打入，μ 变成 001100，Nμ 变成 000000，转到写 R3 操作模块。
- 此时将数据开关打成 12H,，然后按下 QD。12H 被写入 R2 寄存器，并按下 QD 触发打入，μ 变成 000000（代表停止），Nμ 变成 001001（因为还在写寄存器状态，所以又转回去了），

转到写 R0 操作模块。

经过一系列的操作过后，寄存器的值变成如下状态（实际上应该看 D 灯得知输入的数据，但为了验证答案就直接看寄存器了）



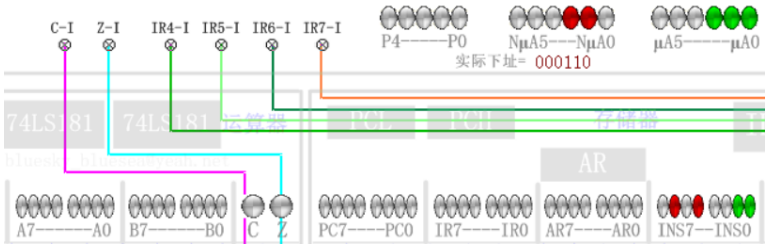
第二步：读寄存器

（控制台 011，读四个寄存器。QD 单拍触发，）

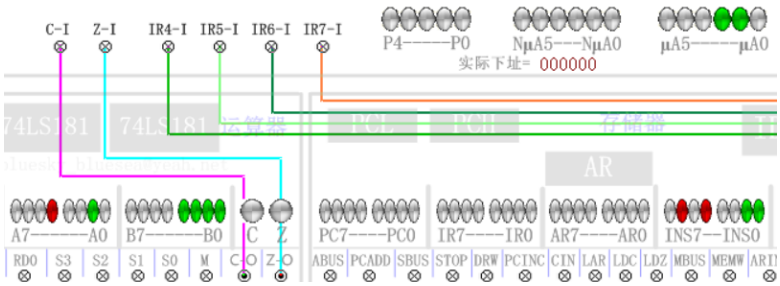
实验过程如下：

A7-A0：第一个寄存器的值
B7-B0：第二个寄存器的值

- 控制台 011，按下 QD， μ 为 000111，Nμ 为 000110，读出的值是前两个寄存器-R0，R1。下图可知，这俩都是 00H。



- 按下 QD， μ 为 000110，Nμ 为 000000（结束），读出的值是后两个寄存器-R2，R3。下图可知，是我们刚写入的 12H，0F。



第三步：写存储器

（控制台 001，写存储器，逐个读入，直到 clr 清零回到初始状态）

实验过程如下：

灯表示的数据：

D7-D0：实时表示数据通路的状态，数据开关改了就显示。

AR7-AR0：表示在 QD 按下时从数据开关（通路）获得的值要进入的地址。

- 打开控制台为 001，转为单拍 DP 触发。
- 按下 QD，因为控制台方式是 001，转写寄存器操作。初始 μ 为 000011，Nμ 为 000010，即准备进入写寄存器循环。这时候需要给出初始写入的地址。因为此时是打开 AR，使得数据开关的数据能通过数据通路打入 AR，进行初步的地址选择。
- 按下 QD，进入写寄存器循环。我们需要根据给出的程序，将相应的数据写到相应的 RAM 内存单元。每写完一个用 QD 单拍打入，并转下一个内存单元进行数据打入。AR 的值代表打入的地址。每次地址+1（体现在 AR 灯上）。
- 程序输入完成后，按下 clr 清零，地址值变成 00，μ 变成 000000，结束运行，所有变量回到初始状态。

写完是这样：

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0-	53	4C	57	21	86	68	4C	53	11	72	48	6A	31	A2	E0	85
1-	23	EF	00	00	00	00	00	00	00	00	00	00	00	00	00	00

第四步：读存储器

（控制台 010，写存储器，逐个读入，直到 clr 清零回到初始状态）

实验过程如下：

灯表示的数据：

D7-D0：实时表示数据通路的状态，显示当前地址取出的数值。

AR7-AR0：表示当前内存地址。

- 打开控制台为 010，转为单拍 DP 触发。
- 按下 QD，因为控制台方式是 010，转读寄存器操作。初始 μ 为 000101，Nμ 为 000100，即准备进入读存储器循环。这时候需要给出初始读出的地址。此时是打开 AR，使得数据开关的数据能通过数据通路打入 AR，进行初步的地址选择。

- 按下 QD，进入读寄存器循环。AR 的值代表地址，D 的值代表地址取出的值。每次地址+1（体现在 AR 灯上）。

- 检验内存单元完成（合适时即可）后，按下 clr 清零，地址值变成 00，μ 变成 000000，结束运行，所有变量回到初始状态。

经过检验，对应的地址里的值没有问题。

第五步：启动程序运行

（控制台 000，进入程序运行模式，取指读指运行。clr 清零回到初始状态）

实验前提描述：
此时寄存器的值已经存在，一串机器级程序也打进了内存中。
需要的任务：
从程序头部（程序在内存中的开始地址）执行，逐步往后取指，执行指令。

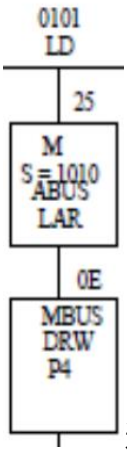
灯表示的数据：
最上部状态栏：控制信号，表达微指令。
P4-P1：周期状态，不同指令周期数不同，但 P1，P4 肯定有。
PC7-PC0：表示对应的程序指令地址。8 位对应 F*F 的内存空间大小。
IR7-IR0：指令数值，从内存中取出。但是只看高四位进行相关控制。
INS7-INS0：取出的指令（从内存取出转到 IR 寄存器进行相关操作）
D7-D0：实时表示数据通路的状态。
AR7-AR0：表示当前 AR 寄存器的状态。
A、B：代表左右寄存器的数值。

实验过程如下：
- 打开控制台为 000，转为单拍 DP 触发。先保证清零状态。

- 按下 QD，因为控制台方式是 000，转程序运行。初始 μ 为 000001，Nμ 为 100000，即准备进入运行。此时 P1 灯亮了。因为程序在 00H 处，所以 PC 不需要设置，清零状态就能用。

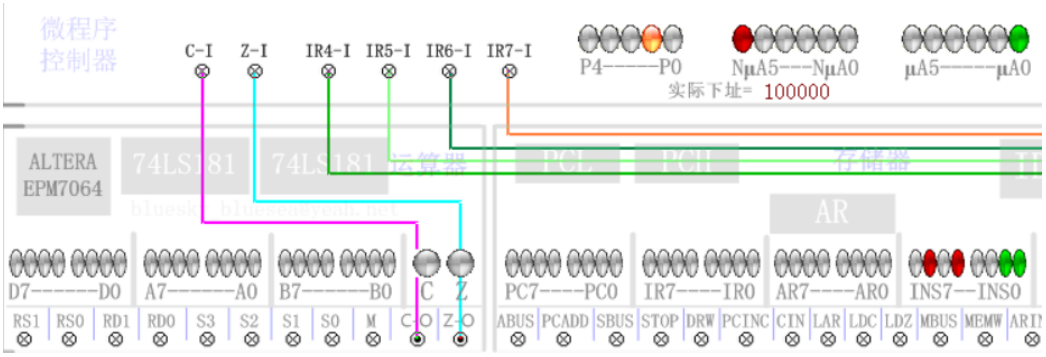
- 按下 QD，进入执行程序。P1 阶段，每次地址+1（体现在 AR 灯上）。

- <插入第一条指令执行的详细过程> LD 类型指令，IR7-IR4 开头表示



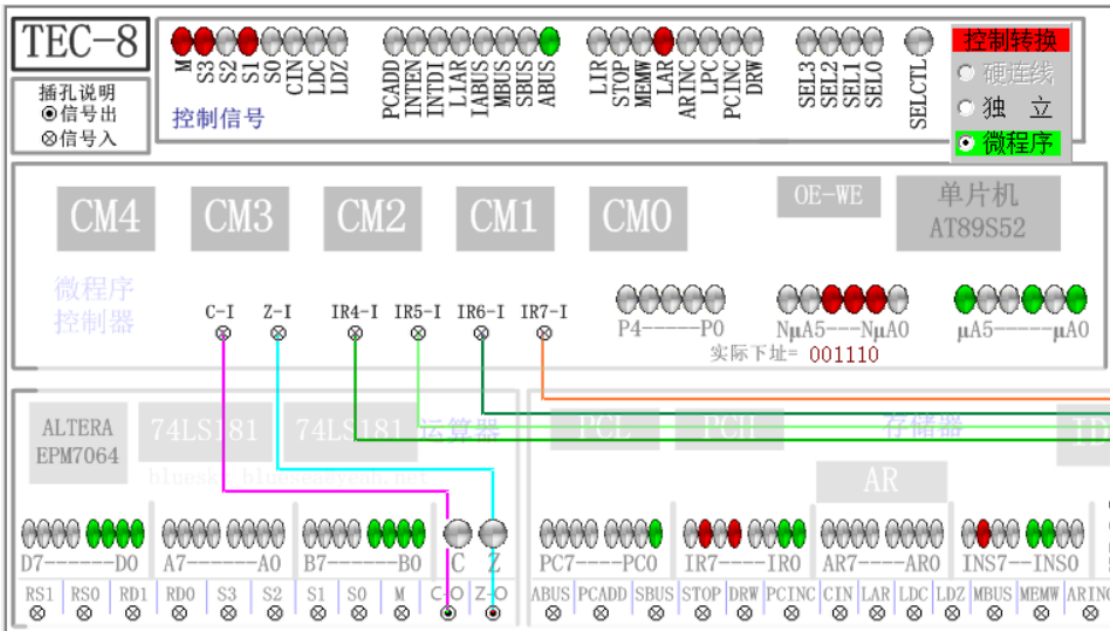
这就是当前指令会执行的步骤。

■ 按下 QD：



此图可知，AR=00H，00H 内存里的值 53 被拿出来放到了指令通路上。

■ 按下 QD：



IR 的值时 0101，即转到相应的程序进行。可看到 μ 的值是 25（16 进制），下地址是 0E。即对应上文指令：

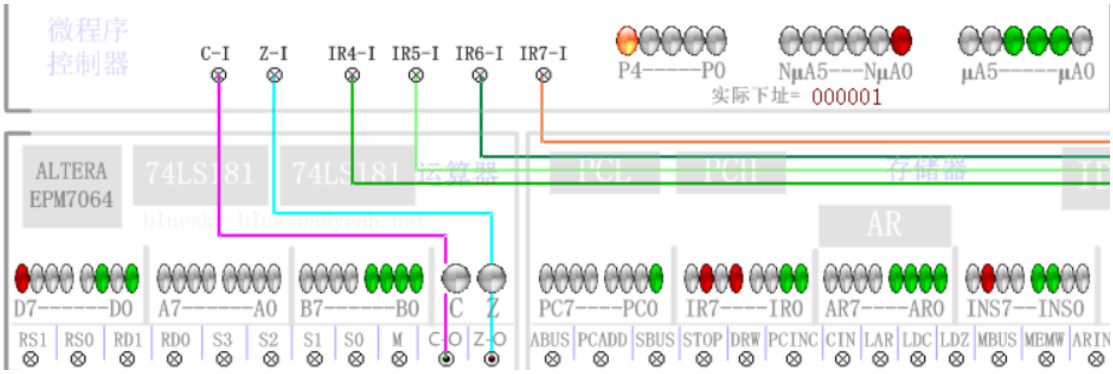
此时：AR：00H，数据通路上是 0FH（寄存器中取出，因为 ABUS 开）

此时 A 是左寄存器，取的是 R0；B 取的是 R3。

■ 按下 QD：

MBUS 指从内存中取出数据放到数据通路上。DRW 代表写进寄存器。

怎么看寄存器呢？看 IR 低八位：00 代表左取 R0，11 代表右取 R3。



细节：**P1 直接转到了 P4**。指令无 P2，P3。

此时数据通路上是 85H，即从 0FH 内存单元中取出，μ 也变成了 0EH，下地址也是 01H，即根据 PC 来决定下一条指令。再取值，执行指令。

- 之后就是一直按下 QD 跑程序了。这里不过多赘述。

一些细节问题：

INC：cin 置 1，哪里读出来回哪里，DRW 打开。

结束：12H 的内容是 00H，代表走到结束了。（但实际上走到了 13H，因为 12H 有数据）

实验结果：

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0-	53	4C	57	21	86	68	4C	53	11	72	48	6A	31	A2	E0	85
1-	23	EF	62	00	00	00	00	00	00	00	00	00	00	00	00	00

实验六：程序中断实验

实验目的

从硬件、软件结合的角度，模拟单级中断和 中断返回的过程； 通过简单的中断系统，掌握中断控制器、中断向量、中断屏蔽等概念；了解微程序控制器与中断控制器协调的基本 原理；掌握中断子程序和一般子程序的本质区别， 掌握中断的突发性和随机性。

实验要求

INT 为 1 转到微地址 11H，该微指令产生 INTDI 信号，禁止新的中断发生，产生 LIAR 信号保存当前地址 （断点寄存器），产生 STOP 信号，等待手动设置中断向量（数据 开关 SD7~SD0 设置中断地址），机器将中断 向量读到 PC 后，转到中服务程序继续执行。执行指 IRET，从中断地址返回，该指令产生 IABUS 信号，恢复断点地址，产生信号 LPC，将断点从数据总线装入 PC，恢复被中断的程序。

中断程序指令翻译：（查表）

00H	EI	1101 1111(DF)
01H	INC R0	0100 0000(40)
02H	INC R0	0100 0000(40)
03H	INC R0	0100 0000(40)
04H	INC R0	0100 0000(40)
05H	INC R0	0100 0000(40)
06H	INC R0	0100 0000(40)
07H	INC R0	0100 0000(40)
08H	INC R0	0100 0000(40)
09H	JMP [R1]	1001 0100(94)
45H	ADD R0 R0	0001 0000（10）

46H	EI	1101 1111 (DF)
47H	IRET	1011 1111 (BF)

写存储器，读存储器

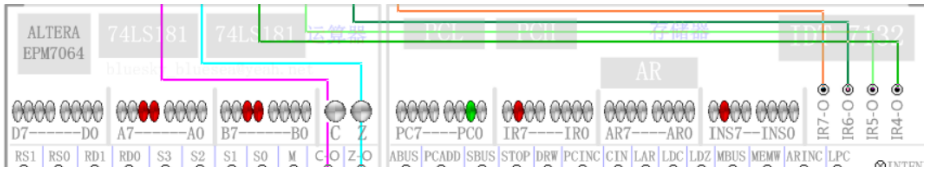
我们把程序打进内存中（00H 开始），然后转到程序执行模式执行。

执行程序

程序功能：在 00H-09H 是一个死循环，当程序连续执行且 R1 为 00H，程序会一直跳转到开头，执行 INCR0 指令。

按下 pause：程序进入中断，现在可以执行中断程序。这里我们执行 45H 开始的 ADD R0 R0（存到 R0 中）的指令。

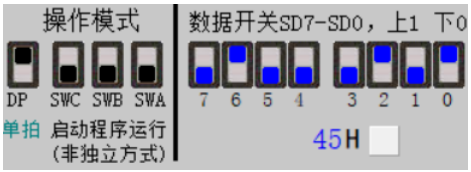
中断程序执行前：PC 为 02H（取指完下一条执行 02H，所以中断返回执行 02H）



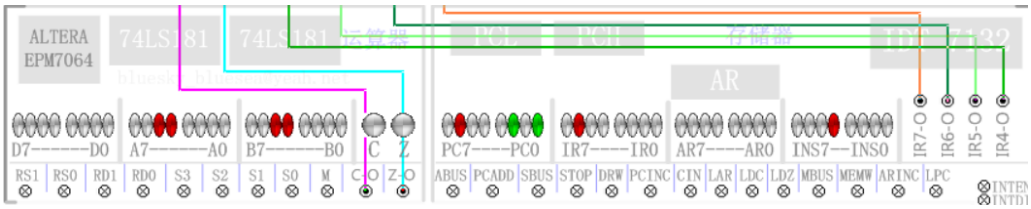
此时因为 IR 低 4 位是 0000，所以 A 和 B 都是 R0 寄存器的值。为 30H。

执行中断程序：输入中断程序 PC，单拍执行。

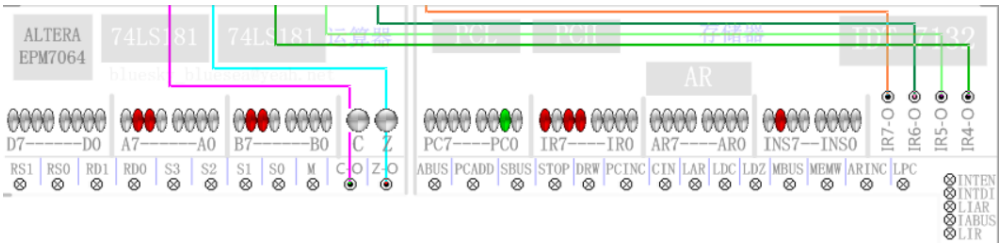
因为程序在 45H，所以数据开关打成 45H 输入。



此时 PC 也变成了我输入的，并处在 P1 取指周期。



中断返回：单拍执行到返回



这里很显然 PC 回到 02H 了。因为中断申请的时候这是需要执行的下一条指令，所以在当前指令执行完毕后，返回 02H。

也能观察到，寄存器 R0 的值变成了 60H，即 30H+30H。中断程序执行成功！

实验五六总结

通过这次实验，我了解了：

- 程序执行的步骤和细节，尤其是深入理解了整个指令执行各个部件的工作原理和使用方式（存储单元，各类寄存器，运算单元）；
- 理解了指令的执行方式，同理也可以推导水平型指令的工作原理；
- 也对中断的申请和释放有了深入的了解（虽然是单级中断，但是显然加个优先级就可以）；也对中断执行时间和中断返回时间以及此时的 PC 值有了深入的了解。
- 最主要的是通过对灯的观测和推导，深入理解了运算器在整个系统中的关键作用。所谓输出的控制信号也大多在这里体现（其中有一个 loadR0，判断 load 哪个寄存器，似乎是观测不到控制信号，在灯里也没有看到，所以应该是在指令高四位翻译的时候写死了）
- 唯二美中不足的我感觉就是程序只能写在 00H 开头以及 P 周期在单拍的时候表示不明确。这个确实让我在执行程序（微指令 01 时无法输入 PC；有是单拍 P 灯会非常诡异）时花费了很多心思思考和验证。最终发现是平台的问题。