计算机组成原理课程设计

报告

(2021/2022 第二学期------第 17 周)

指导教师：许建龙　　杨东鹤

班级：计算机科学与技术 21（4）班

姓名：陈昊天

学号：2021329600006

一、目的和要求

深入了解计算机各种指令的执行过程，以及控制器的组成，指令系统微程序设计的具体知识，进一步理解和掌握动态微程序设计的概念；完成微程序控制的特定功能计算机的指令系统设计和调试。

要求清楚懂得以下内容：

（1） TEC-2机的功能部件及其连接关系；

（2） TEC-2机每个功能部件的功能与具体组成；

（3） TEC-2机支持的指令格式；

（4） TEC-2机的微指令格式，AM2910芯片的用法；

（5） 已实现的典型指令的执行实例，即相应的微指令与其执行次序的安排与衔接；

（6） 要实现的新指令的格式与功能。

二、实验环境

1. TEC-2机一台
2. 电脑一台
3. TEC-2模拟软件一套

三、具体内容

1 运算器知识

Am2901 的控制与操作

为了控制Am2901运算器按我们的意图完成预期的操作功能,就必须向其提供相应的控制信号和数据。

控制信号包括:

选择ALU的八种运算（三种算术、五种逻辑运算）功能中我们所要求的一种。这可通过提供三位功能选择码I5 I4 I3实现。如表2.1所示。

选择送入ALU的两个操作数据 R和S的组合关系（实际来源）。表上已标明, R从D和A中选择, S从A、B和Q中选择, 再考虑到两边还均可选"0"值, 则我们可以从这许多可能组合中选取最有用的8种组合, 即A、Q组合, A、B组合, O、Q组合, 0、B组合, 0、A组合, D、A组合, D、Q组合, D、O组合 , 并用I2 I1 I0三位操作数选择码控制二组多路选通门选取其一, 具体规定如表2.1所示。

选择运算结果或有关数据以什么方式送往何处的处理方案, 这主要指通用寄存器组和Q寄存器执不执行接收操作或移位操作, 以及向芯片输出信息Y提供的是什么内容。这是通过 I8 I7 I6三位结果选择码来控制三组相应的选择门电路实现的, 具体规定如表2.1所示。

表2.1

			I8-6		I5-3	I2-0	
0	0	0	F→Q	F	R+S	A	Q
0	0	1	无	F	S-R	A	B
0	1	0	F→B	A	R-S	0	Q
0	1	1	F→B	F	R∨S	0	B
1	0	0	F/2→B Q/2→Q	F	R∧S	0	A
1	0	1	F/2→B	F	/R∧S	D	A
1	1	0	2F→B 2Q→Q	F	R⊙S	D	Q
1	1	1	2F→B	F	R⊙S	D	0
			寄存器结果选择	Y输出选择	运算功能选择	R	S

外部的数据包括:

(1)通过D接收外部送来的数据, 这已经在ALU的操作数来源选择中解决。

(2)应正确给出芯片的最低位的进位输入信号Cn。

(3)关于左右移位操作过程中的RAM3、RAM0、Q3和Q0的处理, 左移操作时RAM3与Q3为输出, RAM0和Q0为输入；相反, 右移操作时，RAM0和Q0为输出, RAM3和Q3为输入, 这是由I8和I7共同控制的。这几个外部信息的接收与送入，需要在 Am2901芯片之外用另外的电路解决。

(4)四个标志位的值的接收与记忆电路, 需在Am2901芯片之外实现。

(5)当执行通用寄存器组的读操作时，由外部送入的A地址选中的通用寄存器的内容送往A端口,由B地址选中的通用寄存器的内容送往B端口，B地址还用作通用寄存器的写入控制。由于有16个通用寄存器, 故A、B地址均由4位组成。当A、B地址给出同一数值时, 则将选中同一通用寄存器，此时A、B端口同时输出同一寄存器的内容。

(6)如有通用寄存器组的写操作时, 结果将写入由B地址选择的寄存器中。

(7)芯片的输出数据Y可以从AUL的运算结果F或A口的数据二者之中选择其一。

运算器的控制与操作

关于数据, 主要是D15-D0这16位数据输入, 这可以用教学计算机上的16位数据开关直接提供。

关于控制信号, 当我们不考虑教学计算机的其它功能部件, 尤其是不提供控制器支持时，就只能用实验机上的两个12位的微型开关SW2和SW1来向运算器提供全部的控制信号, 包括I8-I0, SST, A地址、B地址、SCi, SSH共24位。其具体安排如图5.1所示。

图5.1 微型开关的控制功能

这样就可以借助开关、按钮、指示灯来操作和控制运算器的运算操作，观察运算器的运算结果，这就是我们称之为脱机的运算器实验方式。用这种方式学习、理解运算器的组成与运行机制是足够简便直观的。

当我们把运算器部件接入教学机整机之内运行时, 就不能靠微型开关来为其提供控制信号，而改由控制器来给出前面刚提到的这24位控制信号。在微程序控制方式下, 这些控制信号是由一个被称之为微指令寄存器PLR的相应的24位直接提供的，它与24位的微型开关提供的信号是互斥关系，可用二选一逻辑来指明选用它们中的哪24位控制信号。二选一逻辑的选择控制信号就是教学机上设置的FS4功能选择开关。当其状态为"1"(开关向上拨)，选通微型开关的信号作为控制信号, 为"0"时, 选通控制器的PLR的24位信号作为控制信号, 如图5.2所示。

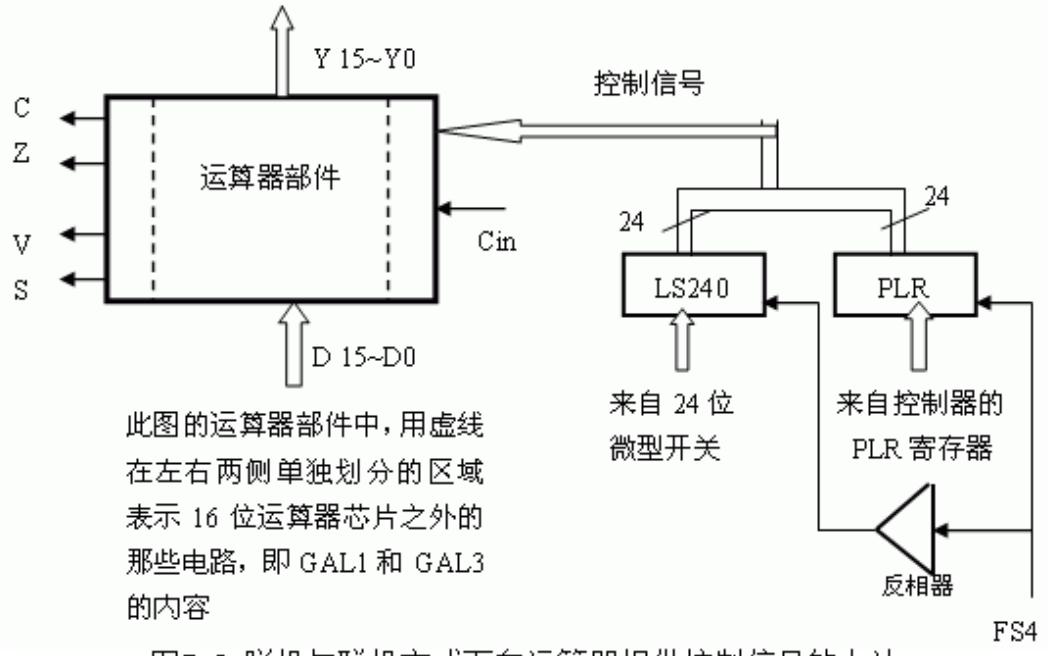


图5.2 脱机与联机方式下向运算器提供控制信号的办法

LS240器件与PLR器件的输出均有三态控制逻辑, 当选通信号为低电位时, 器件输出有效, 否则输出为高阻态。

2 控制器知识

控制器的功能、组成概述

控制器部件是计算机的五大功能部件之一，其作用是向整机每个部件(包括控制器部件本身)提供协同运行所需要的控制信号。计算机的最本质的功能是连续执行指令，而每一条指令往往又要分成几个执行步骤才得以完成。由此又可以说，计算机控制器的基本功能，是依据当前正在执行的指令和它所处的执行步骤，形成(或称得到)并提供出在这一时刻整机各部件要用到的控制信号。

执行一条指令，要经过读取指令、分析指令、执行指令所规定的处理功能三个阶段完成，控制器还要保证能按程序中设定的指令运行次序，自动地连续执行指令序列。

为此，控制器组成中，必须有一个能提供指令在内存中的地址的部件，通称程序计数器(PC)，服务于读取指令，并接收下条要执行的指令的地址。

还要有一个能保存读来的指令内容的部件，通称指令寄存器(IR)，以提供本指令执行的整个过程中要用到的指令本身的主要信息。

控制器的第三个组成成分，是脉冲源、启停控制逻辑，指令执行的步骤标记线路，它标记出每条指令的各执行步骤的相对次序关系。

控制器的第四个、也是控制器设计中最费力的一个组成成分，是全部时序控制信号的产生部件，它依据指令内容、指令的执行步骤(时刻)，也许还有些别的什么条件信号，来形成并提供出当前各部件本时刻要用到的控制信号。计算机整机各硬件系统，正是在这些信号控制下协同运行，产生予期的执行结果，也就是执行一条又一条的指令。

依据前述控制器的最后两个组成成分的具体组成与运行原理的不同，通常把控制器区分为微程序的控制器和组合逻辑(硬布线)的控制器两大类。教学计算机系统中，分别设计并实现了这两种控制器，为教学提供了比较理想的实例，也为实验人员开辟了选择使用与进行实验的广阔天地。图1.1给出了控制器组成与其在整机中地位的示意表示。

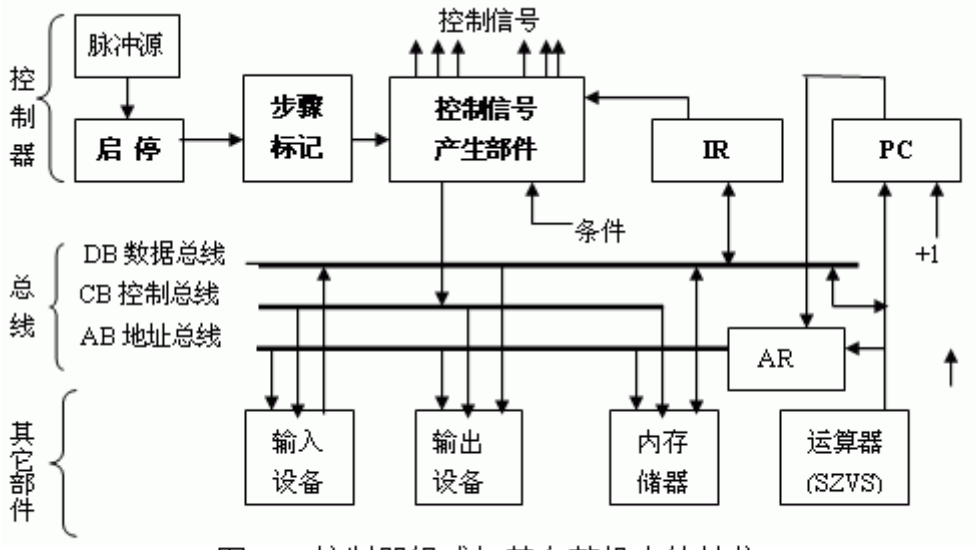


图1.1 控制器组成与其在整机中的地位

微程序控制器的设计与实现

1. 基本组成部分

基本组成指的是该控制器的必不可少的组成部分，如图3.4所示。它与微程序设计的最基本的原理直接相关，是课程的重点内容。
从图3.1可以看出该控制器的基本组成部件及它们相互间的逻辑关系。

最核心的部分是控制存储器，用于存放教学机的微程序，由56位组成，用7片74LS6116随机读写的8位×2048字的内存芯片实现。通常控存都是用ROM芯片实现，把厂家设计好的微程序固化在里边，仅提供读操作功能，可靠性更高些。教学机要支持动态微程序设计，即允许实验人员把自己设计的微程序写进控存，我们只能用可读写存储器支持这一要求。这还带来一个新的问题，即在实验机加电启动时，首先必须把已设计好的53条机器指令用到的微程序调入控存，这个问题将到该控制器的辅助组成部分去进一步讲解。正常执行微程序时，该控存将依据Am2910提供给它的10位地址，在读写命令W/R(为高是读操作)控制下读出相应单元的一条微指令。

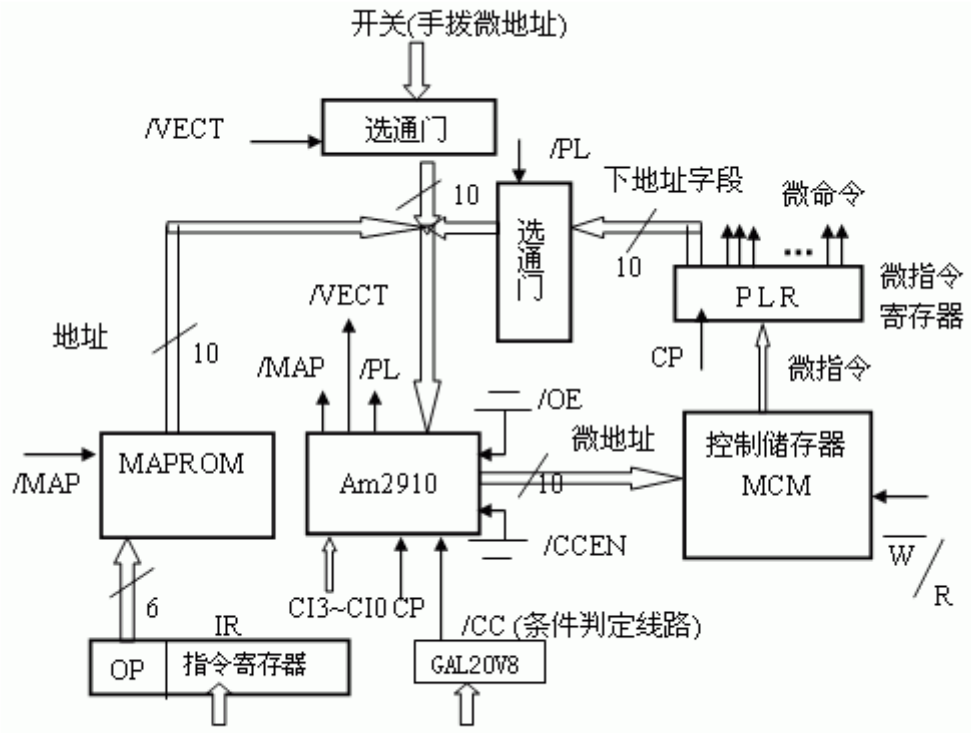


图 3.1 微程序控制器的基本组成

读出的微指令将通过CP脉冲，打入该控制器的第二个组成部件——微指令寄存器PLR中。前边已提到过，一条微指令的内容通常由下地址和控制信号(微命令)两部分组成。下地址可能又被分成几个小字段，它们被用于控制、形成、或直接提供下一条微指令的地址，是供给控制器部件本身使用的。微命令也由若干个小字段组成，提供对计算机其它功能部件，如运算器、主存、输入/输出设备的控制信号。在第二章，我们已看到用于控制运算器的许多控制信号。在本小节稍后部分，将详细介绍教学计算机的微指令的格式和每个字段的内容及其控制功能。

该控制器的第三个组成成分，是微程序定序器Am2910器件及其配套电路。

在本教学计算机的实现中，已把Am2910的/OE端接地，使其输出Y11-Y0总保持有效(实用Y9-Y0共10位)。

把/CCEN接地，使Am2910的条件判断结果只取决于/CC。

把CI接电源，使微指令地址 +1总是执行。

用/VECT信号把通过水平板上的开关给出的10位微指令的手拨地址接通到Am2910的D输入端。

分别用/MAP和/PL两个信号选通MAPROM和微指令下地址字段的输出送到Am2910的D输入端，从而形成D输入端的3选1逻辑功能。

与Am2910配套的电路，主要包括MAPROM和用于形成/CC信号逻辑值的条件判定线路。MAPROM被用作为指令微地址映射部件，它变换指令的操作码为该指令对应的微程序段入口地址，由两片74LS2716 ROM芯片组成，其地址为指令的操作码，对应单元中存放相应微程序段的入口地址，执行读操作，并用/MAP选通读出的信息，解决的是指令功能分支问题。

关于/CC条件码的形成问题，需解决指明判定条件和完成条件判定两个方面的需求。要判断的条件相当多，是通过微指令中下地址字段中的SCC(3位)和SC(1位)两个子字段指明的，其具体规定如表3.1所示,采用专门的硬件电路，即一片Gal20v8器件实现表中所规定的功能。

表3.1

SCC	SC	/CC	SCC	SC	/CC
0		0	7	IR10-8	
1		1		0	/C
2	SC=0	/FS1		1	/Z
3	SC=0	/FS2		2	/V
4	SC=0	/FS3		3	/S
5	SC=0	/WAIT		4	C
2	SC=1	/C		5	Z
3	SC=1	/Z		6	V
4	SC=1	/V		7	S
5	SC=1	/S			
6		/INT			

表中的FS1、FS2和FS3是水平板上的3个功能开关，用于选择教学机执行不同的操作功能，其具体规定如表3.2所示

表3.2

FS1	FS2	FS3	FS4	功　能
X	X	X	1	脱机运算器部件实验(X 代表该位可取任意值)
0	0	0	0	装入微程序
0	0	1	0	执行微程序
0	1	0	0	存储器写(单步)
0	1	1	0	存储器读(单步)
1	0	0	0	存储器读(连续)
1	0	1	0	从 0 地址连续执行程序
1	1	0	0	从指定地址单指令执行程序
1	1	1	0	从指定地址连续执行程序

/WAIT是教学处于单步执行时，用于单步控制线路的等待状态(等待按下STEP微型按键)。

C、Z、V、S或它们的取反值/C、/Z、/V、/S是运算器中的四个状态标志位。当SCC的3位微码为111,即十进制编码值为7时，通过条件转移指令的指令操作码 IR的第10-8位选择它们，以形成条件码/CC的值。

当SC为1时，通过SCC三位编码的2、3、4和5状态选择/C、/Z、/V、/S形成条件码/CC的值，用于非条件转移指令所用的微指令中。

/INT为中断请求信号，低电平有效，在每条指令结束时，判有无中断请求，以确定转中断处理还是执行下一条指令。

采用专门的硬件电路，即一片Gal20v8器件实现表3.2所规定的功能。

微指令字下地址字段中还有一个子字段CI3-CI0，用于给出Am2910的命令码，它与/CC的取值、Am2910内部的R/C的内容是否为零等一起，共同决定Am2910芯片内部的操作和形成下一条微指令地址的具体办法。

自行设计新指令的微程序

所谓新指令，是指教学机支持的64条基本指令中未实现、留给学生自行设计与实现的11条机器指令，即指令汇总表中最后的11条指令。这11条指令的情况是:

—— 6位操作码已定，为D4、D8、DC、E0、E4、E8、EC、F0、F4、F8和FC，这是按8位长度的16进制方式给出的。其最低两位，可用于选择C、Z、V、S四个标志位作条件转移指令的判别条件。

—— 这11条指令的微程序段的入口地址已定为100h，110h，120h,...1FEh。这是由MAPROM器件的内容限定的,这些内容已写好在该器件的相应单元中。

—— 这11条指令没有相应的汇编语句名，执行的功能也未定义。但在使用时，必须使其指令格式与已实现的53条指令的格式类同，如要用C、Z、V、S作为判别条件，只能用指令寄存器的第9,8两位编码加以标明，作为写入用的寄存器编号只能通过IR7-IR4标明等等。使用不当，目前已给出的硬件可能无法直接支持。
设计新指令的微程序段将涉及以下几个问题:

—— 选定指令格式及功能，包括确定要用的操作码，指令中其它字段的内容分配与使用，本指令要实现的具体功能。

—— 按新指令的功能与格式，设计该指令的执行过程，即分成几步完成，每一步要实现的详细操作细节，各步之间的衔接次序等。

—— 将每一步中的操作，用一条微指令实现，即具体设计每条微指令各字段的具体编码值，既包括控制码的各字段，也包括下地址字段，形成下地址用到的条件码等等。

—— 将设计好的微码，装入控制存储器的相应单元。

—— 设计一个使用新、旧指令的用户程序，检查程序运行的正确性，以确定新指令是否正确执行，对新指令的执行过程仔细调试，直到得到满意的结果。

这一过程中的向控存中装入新指令的微码有两种方法，一是通过水平板上的开关与按键直接拨入，具体操作方法参见教学计算机补充材料；二是在程序中用LDMC指令指令实现自动装入，其具体操作步骤介绍如下。

作为例子，最简单的方法，是抄一条现有指令作为新指令予以实现。例如，操作码选D4，指令格式选为D4 DR，SR,,实现DR+SR→DR的功能。它就是ADD DR，SR那条指令，差别仅是操作码由04变为D4。查指令汇总表，D4操作码的微指令的入口地址应为100h,故将1Ch地址中的微指令(实现ADD DR，SR的操作)中的内容复制到100h单元，就完成了这条新指令的微程序设计的过程。看下面一个程序例子。

```
<A800
800: MOV R8, 240 ；为指令的目的寄存器赋初值
MOV R9， 360 ；为指令的源寄存器赋初值
MOV  R1， 900 ；微码在内存的首地址
MOV R2, 1 ；微指令条数
MOV R3， 100 ；微码在控存中的首地址
LDMC  ；用R1,R2,R3作为参数,装入微码
D489 ；新指令的二进制执行码
RET
<E900
900：  0029...0301....b090....0088
```

以上用到的数值均为16进制。在监控命令工作时，输入均用16进制数，且都不能跟h字符。

该程序的功能是将240和360两个16进制形式的整数分别送入R8和R9。用新指令 (机器码为D489：操作码为D4，DR选R8,SR选R9) 完成两个寄存器的内容相加，结果写入R8。

该程序当中的4条指令，实现的是装入新指令的微码。微码在内存的首地址为900，四个字的内容为0029，0301，b090，0088，是控存1CH单元的内容，可以用监控程序的E命令键入。

该程序可以用监控程序的命令打入。倒数第2行的D489是新指令的机器码，不能在A命令方式下打入。具体操作过程，可以在A命令方式下，先在此处打入任何一条单字指令，例如，MOV R0，R0。整个程序输入后，再将该单元的内容用E命令改为D489，该程序运行过程中，在为R8,R9赋值后，接着装入新指令的微码，再执行新指令，最后返回监控程序以结束该程序的执行过程。

该程序运行结束后，用R命令检查程序的执行结果，R8的值应变为05A0。

从这个例子可以得出以下几个结论:

—— 新旧指令可以用在同一程序中；
—— 新指令在每次教学机重新加电后，至少得重新装入一次对应的微码； 仅在装入相应微码后，新指令才能执行，即已将新指令追加到教学的指令系统中；
—— 新指令无汇编码(因汇编程序实现在前，新指令实现在后)，故在程序中，只能通过机器码使用新指令；否则必须去扩展有关的汇编程序。
—— 装入新的微指令与使用新指令变得非常容易，同学的精力就可以全部集中到微程序设计方面来。但必须想到，新、旧指令的微程序之间存在着如下协调与配合关系:
前边的例子中，只设计了新指令的具体执行功能，执行前的取指过程和执行后的判中断、与下条指令的衔接等均使用了原微程序的有关内容。从同学学习微程序设计的角度看，取指过程与每条指令完成后的相应处理是公用于所有指令的，而且比较简单，看懂原来的实现方法与细节，以及与每条指令执行过程的衔接方式，也就达到了深入掌握的程度，故一般不必在自己设计的微程序中考虑这一部分内容。若有的学生想在自己的微程序中实现自己设计的取指等处理过程，必须保证在新旧指令衔接时不出现矛盾。最简单的方法，是在自己的多条新指令中，有几条指令有自己设计的取指与后续处理，它们不能与原有指令的微程序段正确衔接。但有一条新指令用原有指令的取指处理完成取指过程，有另一条新指令用原有指令的后续处理完成判中断，保证能正确与原有指令的取指过程衔接。当用这样的两条指令“夹”起的其它新指令序列出现在任何程序中时，每条指令均将能正确衔接执行。
设计不同格式与功能的新指令的执行步骤，以及每一步中的微指令字的各字段的编码，是学习计算机微程序设计的重点，也是学懂计算机指令执行过程的核心内容。

3 提示

微指令

PC: Program Counter (程序计数器)。程序开始时PC指向指令的第二个字，即第一个参数

AR: Address Register (地址寄存器)

MEM: Memory (当出现MEM的时候，就是在对"AR"存的地址取值)

AR相当于"指针"，MEM相当于"值"

Q: Q Register (用于计算)

注意：用G命令运行用于测试微程序的汇编程序后，汇编程序本身可能会发生变化，需要重新编辑。这通常是由于微指令设计存在问题。

微码

下地址：一般只在条件转移时用到，就是CC#=0或CC#=CND时使用，而且固定为0010100100

CI3-0 | SCC SC | SST | /MI0 REQ /WE | MI8-0 | SCi | SSH | DC1 | DC2

	完成功能	R/C 内容	R/C 操作	使能信号	/CCEN			
					/CC 高		/CC 低	
					Y 输出	堆栈	Y 输出	堆栈
0	初始化			PL	0	清除	0	清除
1	条件转微子			PL	μ PC		D	压入
2	指令功能分支			MAP	D		D	
3	条件转移			PL	μ PC		D	
4	入栈 R/C 装数			PL	μ PC	压入	μ PC	压入
6	转中断向量			VECT	μ PC		D	
8	R/C	非零	-1	PL	F		F	
	非零循环	零		PL	μ PC	弹出	μ PC	弹出
10	条件返回			PL	μ PC		F	弹出
14	顺序执行			PL	μ PC		μ PC	
15	三路转移	非零	-1	PL	F		μ PC	
		零		PL	D	弹出	μ PC	弹出

CI3-0：只会用到条件转移和顺序执行

CI3-0 | SCC SC | SST | /MI0 REQ /WE | MI8-0 | SCi | SSH | DC1 | DC2

	SCC	/CC		SCC	/CC
0		0	7	IR10-8	
1		1		0	/C
2	SC=0	/FS1		1	/Z
3	SC=0	/FS2		2	/V
4	SC=0	/FS3		3	/S
5	SC=0	/WAIT		4	C
2	SC=1	/C		5	Z
3	SC=1	/Z		6	V
4	SC=1	/V		7	S
5	SC=1	/S			
6		/INT			

SCC SC：一般为0，或设置CND条件码时使用

CI3-0 | SCC SC | SST | /MI0 REQ /WE | MI8-0 | SCi | SSH | DC1 | DC2

SST 编码			状态位输入				说明
B34	B33	B32	C	Z	V	S	
0	0	0	C	Z	V	S	四位标志位的值保持不变
0	0	1	CY	F=0	OV	F3	接收 ALU 的标志位输出值
0	1	0	IB7	IB6	IB5	IB4	恢复标志位原现场值
0	1	1	0	Z	V	S	置 C 为 “0”，另三个标志不变
1	0	0	1	Z	V	S	置 C 为 “1”，另三个标志不变
1	0	1	RAM0	Z	V	S	右移操作，另三个标志不变
1	1	0	RAM15	Z	V	S	左移操作，另三个标志不变
1	1	1	Q0	Z	V	S	联合右移，另三个标志不变

SST：一般为0，加减操作时为001接收ALU标志位输出

CI3-0 | SCC SC | SST | /MI0 REQ /WE | MI8-0 | SCi | SSH | DC1 | DC2

/MIO	REQ	/WE	操作功能	/MIO	REQ	/WE	操作功能
0	0	0	存储器写	0	1	1	I／O读
0	0	1	存储器读	1	0	X	不操作
0	1	0	I／O写	1	1	X	装入微码

/MIO REQ /WE：一般为不操作，读取MEM时为存储器读，写入MEM时为存储器写

/WE 设为1 把Y送到内部总线好让DC接收

CI3-0 | SCC SC | SST | /MI0 REQ /WE | MI8-0 | SCi | SSH | DC1 | DC2

			I8-6		I5-3	I2-0	
0	0	0	F→Q	F	R+S	A	Q
0	0	1	无	F	S-R	A	B
0	1	0	F→B	A	R-S	0	Q
0	1	1	F→B	F	R∨S	0	B
1	0	0	F/2→B Q/2→Q	F	R∧S	0	A
1	0	1	F/2→B	F	/R∧S	D	A
1	1	0	2F→B 2Q→Q	F	R⊙S	D	Q
1	1	1	2F→B	F	R⊗S	D	0
			寄存器结果选择	Y 输出选择	运算功能选择	R	S

MI8-0：重点。F为ALU输出

F->Q：输出送Q寄存器

F->B：输出送B口

Y的内容会被送到内部数据总线，DC1和DC2，送AR和送MEM时使用

R和S选中的内容会参与运算

指令中的DR位于B口，SR位于A口

CI3-0 | SCC SC | SST | /MI0 REQ /WE | MI8-0 | SCi | SSH | DC1 | DC2

SCi编码（B11B10）	00	01	10	11
Cin 取值	0	1	C	TCLK 方波

SCi：进位设置，一般为0，PC+1->PC时设置为1

CI3-0 | SCC SC | SST | /MI0 REQ /WE | MI8-0 | SCi | SSH | DC1 | DC2

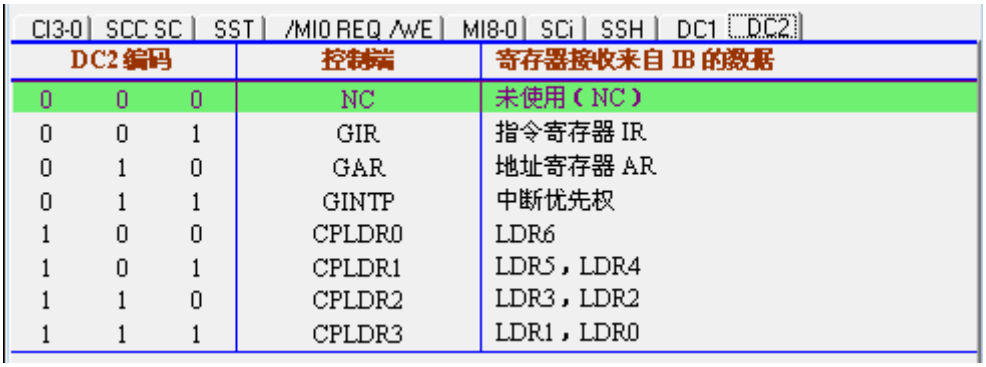
2 位控制码		左移		右移		说明
SSH 码编码		RAM0	Q0	RAM15（RAM7）	Q15（Q7）	
B9	B8					
0	0	0	X	0	X	通用寄存器逻辑位移
0	1	C	X	C	X	通用寄存器与 C 循环移
1	0	Q15（Q7）	/F15 /F7	CY	RAM0	原码除（左移）乘（右移）
1	1	X	X	F15（F7）或 OVR	RAM0	右移用于补码乘法

SSH：一般为0

CI3-0 | SCC SC | SST | /MI0 REQ /WE | MI8-0 | SCi | SSH | DC1 | DC2

DC1 编码			控制端	送往内部总线 IB 的数据
0	0	0	/SWTOIB	开关手拨数据
0	0	1	/RTOIB	运算器的输出
0	1	0	/ITOIB	指令的低 8 位
0	1	1	/FTOIB	状态寄存器
1	0	0	/INTA	中断向量
1	0	1	NC	未使用
1	1	0	/EI	转用于开中断
1	1	1	/DI	转用于开中断

DC1：一般为0，需要写入MEM时设置为001



CI3-0 | SCC SC | SST | /MIO REQ /WE | MI8-0 | SCi | SSH | DC1 | DC2

DC2 编码			控制端	寄存器接收来自 IB 的数据
0	0	0	NC	未使用（NC）
0	0	1	GIR	指令寄存器 IR
0	1	0	GAR	地址寄存器 AR
0	1	1	GINTP	中断优先权
1	0	0	CPLDR0	LDR6
1	0	1	CPLDR1	LDR5，LDR4
1	1	0	CPLDR2	LDR3，LDR2
1	1	1	CPLDR3	LDR1，LDR0

DC2：一般为0，需要写入AR时设置为010

A口、B口：需要使用SP、PC、IP（R4-R6）等寄存器时设置为寄存器的地址，分别为0100、0101、0110

SA=0 A口的值来自A口的字段

SA=1 A口的值来自SR的字段

SB=0 B口的值来自B口的字段

SB=1 B口的值来自DR的字段

4 加法指令

指令格式：`D5DRSR, DISP` 双字指令（控存入口100H）

功能： `[DR]=[DR] +[[SR]+DISP]`

设计

注意，DISP是一个数值，测试时手动指定。一个该指令的例子：`D501 0005`。01表示DR=R0, SR=R1。R0和R1需要测试时自行指定。

中括号相当于C语言的解引用，获得地址上的数值

微程序:

```
SR->AR              AR存了SR的值，SR的值是一个地址。AR使当前MEM等于AR存的地址上的数值
MEM->Q              MEM送Q，此时Q=[SR]
PC->AR, PC+1->PC    AR存了PC的值，PC的值是第一个参数的地址，即DISP所在的地址。
                    AR使当前MEM等于DISP所在的地址上的值，MEM就是DISP本身。
                    PC递增（存疑：实际上，后面没有用到PC，不递增也可以）
Q+MEM->AR           MEM=DISP，Q+MEM为[SR]+DISP，AR使MEM的值为[SR]+DISP这个地址的值
MEM->Q              Q变为[[SR]+DISP]
DR->AR              意义与第一行类似，此时MEM=[DR]
MEM+Q->Q            Q变为[DR] +[[SR]+DISP]
Q->MEM, CC#=0       AR不变，MEM还是[DR]，Q送[DR]。CC# = /CC = CC反，为0表示程序正常结束。
```

微指令对应微码：

```
SR->AR              0000 0E00 90C0 0082
MEM->Q              0000 0E00 00F0 0000
PC->AR, PC+1->PC    0000 0E00 A0B5 5402
Q+MEM->AR           0000 0E00 10E0 0002
MEM->Q              0000 0E00 00F0 0000
DR->AR              0000 0E00 90B0 000A
MEM+Q->Q            0000 0E01 00E0 0000
Q->MEM, CC#=0       0029 0300 1020 0010
```

测试

1 输入微码

将微码输入到由0900H开始的内存单元中。

用E命令输入微码，回车后输入微程序。每个数值间以空格分开，输入完毕后按回车键。

```
E0900
0000 0E00 90C0 0082 0000 0E00 00F0 0000 0000 0E00 A0B5 5402 0000 0E00 10E0
0002 0000 0E00 00F0 0000 0000 0E00 90B0 000A 0000 0E01 00E0 0000 0029 0300
1020 0010
```

2 编写用于加载微码的汇编指令

```
A0800
MOV R1, 900
MOV R2, 8
MOV R3, 100
LDMC
RET
```

```
A0800           从0800地址开始输入汇编语句
MOV R1, 900     微码从0900地址开始
MOV R2, 8       有8条微指令
MOV R3, 100     控存为100H，对应操作码为D4~D7，见附录一最后几行(如图)
LDMC            装入微码
RET             汇编语句结束，子程序返回
```

C8	110010××	JP ADR	32H	92H
CC	110011××	CALL ADR	33H	94H
D0	110100××	LDMC	34H	98H
D4	110101××	(未用)	35H	100H
D8	110110××	(未用)	36H	110H
DC	110111××	(未用)	37H	120H
E0	111000××	(未用)	38H	130H
E4	111001××	(未用)	39H	140H
E8	111010××	(未用)	3AH	150H
EC	111011××	(未用)	3BH	160H
F0	111100××	(未用)	3CH	170H
F4	111101××	(未用)	3DH	180H
F8	111110××	(未用)	3EH	190H
FC	111111××	(未用)	3FH	1A0H

```
>A0800
0800: MOV R1, 900
0802: MOV R2, 8
0804: MOV R3, 100
0806: LDMC
0807: RET
0808:
>
```

3 运行用于加载微码的汇编指令

```
G0800
```

4 编写用于测试我们做好的微程序的汇编指令

```
A0820
MOV R0, 0001
MOV [1080], R0
MOV R0, 1070
MOV [1090], R0
MOV R0, 1050
MOV [1070], R0
MOV R0, 0001
MOV [1075], R0
MOV R0, 1080
MOV R1, 1090
NOP
NOP
RET
```

前面的一大堆MOV操作，为了给内存地址为1080，1090，1070，1075的内存上赋值

然后给R0, R1赋值。R0，R1作为我们指定的DR, SR，在我们编写的微程序运行前赋好初值

```
MOV R0, 0001 //R0=0001
MOV [1080], R0 //[1080]=R0
......
MOV R0, 1080
MOV R1, 1090
```

关于为什么这么赋值：

```
我们的功能：[DR]=[DR]+[[SR]+DISP]
我们的实际指令：D501 0005
D5：操作码
01：DR=R0，SR=R1
0005：DISP=0005

[DR]=[DR]+[[SR]+DISP]
DR=1080
[DR]=[1080]=0001
SR=1090
[SR]=[1090]=1070
DISP=0005
[SR]+DISP=1070+0005=1075
[1075]=0001
最后[DR]=[DR]+[[SR]+DISP]
       =0001+0001
       =0002
即[1080]=0002
这是我们预期得到的结果
```

两个NOP空操作是为了给我们编写的微程序（双字指令）留位置。我们自定义的指令 `D501 0005` 不能直接用A命令写在内存里，会报错。

```
>A0820
0820: MOV R0, 0001
0822: MOV [1080], R0
0824: MOV R0, 1070
0826: MOV [1090], R0
0828: MOV R0, 1050al TEC-2 By Guiheng Zhou, Jun. 2005, Sun Yat-sen University
082A: MOV [1070], R0
082C: MOV R0, 0001
082E: MOV [1075], R0
0830: MOV R0, 1080
0832: MOV R1, 1090
0834: NOP
0835: NOP
0836: RET
0837:
>
```

5 用E命令直接编辑两个NOP

把NOP编辑为我们的双字指令。

```
E0834
D501 0005
```

此时可以用U0820来查看汇编指令。可以看到，0834和0835已经更改为D501和0005

```
>U0820
0820: 2C00 0001  MOV    R0,     0001
0822: 3400 1080  MOV    [1080], R0
0824: 2C00 1070  MOV    R0,     1070
0826: 3400 1090  MOV    [1090], R0
0828: 2C00 1050  MOV    R0,     1050
082A: 3400 1070  MOV    [1070], R0
082C: 2C00 0001  MOV    R0,     0001
082E: 3400 1075  MOV    [1075], R0
0830: 2C00 1080  MOV    R0,     1080
0832: 2C10 1090  MOV    R1,     1090
0834: D501       DW     D501
0835: 0005       NOP
0836: AC00       RET
0837: 0000       NOP
0838: 0000       NOP
0839: 0000       NOP
>
```

6 运行用于测试我们做好的微程序的汇编指令

```
G0820
```

D命令查看结果

```
D1080
```

1080地址处应为0001+0001 = 0002

```
>G0820
>D1080
1080    0002  0000  0000  0000  0000  0000  0000  0000  ................
1088    0000  0000  0000  0000  0000  0000  0000  0000  ................
1090    1070  0000  0000  0000  0000  0000  0000  0000  .p..............
1098    0000  0000  0000  0000  0000  0000  0000  0000  ................
10A0    0000  0000  0000  0000  0000  0000  0000  0000  ................
10A8    0000  0000  0000  0000  0000  0000  0000  0000  ................
10B0    0000  0000  0000  0000  0000  0000  0000  0000  ................
10B8    0000  0000  0000  0000  0000  0000  0000  0000  ................
10C0    0000  0000  0000  0000  0000  0000  0000  0000  ................
10C8    0000  0000  0000  0000  0000  0000  0000  0000  ................
10D0    0000  0000  0000  0000  0000  0000  0000  0000  ................
10D8    0000  0000  0000  0000  0000  0000  0000  0000  ................
10E0    0000  0000  0000  0000  0000  0000  0000  0000  ................
10E8    0000  0000  0000  0000  0000  0000  0000  0000  ................
10F0    0000  0000  0000  0000  0000  0000  0000  0000  ................
>
```

5 传送指令

指令格式：D8××，ADDR1，ADDR2　三字指令（控存入口110H）

功能：　[ADDR1]←[ADDR2]

设计

微程序：

```
PC+1->AR            0000 0E00 90B0 5402
MEM->AR             0000 0E00 10F0 0002
MEM->Q              0000 0E00 00F0 0000
PC->AR,PC+1->PC     0000 0E00 A0B5 5402
MEM->AR             0000 0E00 10F0 0002
Q->MEM, CC#=0       0029 0300 1020 0010
```

测试

```
E0900
0000 0E00 90B0 5402 0000 0E00 10F0 0002 0000 0E00 00F0 0000 0000 0E00 A0B5
5402 0000 0E00 10F0 0002 0029 0300 1020 0010

A0800
MOV R1, 900
MOV R2, 6
MOV R3, 110
LDMC
RET

G0800

A0820
MOV R0, 0001
MOV [1020], R0
NOP
NOP
NOP
RET

E0824
D800 1010 1020

G0820
D1010
```

1010应为0001

```
>D1010
1010    0001  0000  0000  0000  0000  0000  0000  0000  ................
1018    0000  0000  0000  0000  0000  0000  0000  0000  ................
1020    0001  0000  0000  0000  0000  0000  0000  0000  ................
1028    0000  0000  0000  0000  0000  0000  0000  0000  ................
1030    0000  0000  0000  0000  0000  0000  0000  0000  ................
1038    0000  0000  0000  0000  0000  0000  0000  0000  ................
1040    0000  0000  0000  0000  0000  0000  0000  0000  ................
1048    0000  0000  0000  0000  0000  0000  0000  0000  ................
1050    0000  0000  0000  0000  0000  0000  0000  0000  ................
1058    0000  0000  0000  0000  0000  0000  0000  0000  ................
1060    0000  0000  0000  0000  0000  0000  0000  0000  ................
1068    0000  0000  0000  0000  0000  0000  0000  0000  ................
1070    1050  0000  0000  0000  0000  0001  0000  0000  .P..............
1078    0000  0000  0000  0000  0000  0000  0000  0000  ................
1080    0002  0000  0000  0000  0000  0000  0000  0000  ................
>
```

6 转移指令

判断两个通用寄存器内容是否相等，若相等则转移到指定目的地址（IP+DISP），否则顺序执行。

指令格式：`E1DRSR，DISP` 双字指令（控存入口130H，DISP为相对转移地址偏移量）**指令格式有改动，后续说明**

功能：`if DR=SR goto IP+DISP else 顺序执行`。

设计

分析：

在本例中，需要进行条件转移。当 DR == SR 时，DR - SR = 0，标志位 Z = 1。此时设法让CC# = Z，微程序继续运行，实现 goto 的后续功能。

否则，DR != SR，标志位 Z = 0，CC# = Z = 0，**微程序提前结束**，运行整个微程序之后的汇编指令，实现顺序执行的功能。

为了让CC# = Z，查表发现需要用到 IR10-8，也就是指令格式中E后面的数字。**为了让CC# = Z，需要让E后面的数字为5。同时控存入口变成140H**

E0	111000××	(未用)	38H	130H
E4	111001××	(未用)	39H	140H
E8	111010××	(未用)	3AH	150H
EC	111011××	(未用)	3BH	160H
F0	111100××	(未用)	3CH	170H
F4	111101××	(未用)	3DH	180H
F8	111110××	(未用)	3EH	190H
FC	111111××	(未用)	3FH	1A0H

所以测试用的指令格式为：`E501，0003`

5：置条件码为标志位 Z

01：指定DR，SR为 R0, R1

0003：DISP=0003

微程序及其微码

```
DR-SR                    0000 0E01 9190 0088
PC->AR,PC+1->PC,CC#=CND  0029 03E0 A0B5 5402
IP+MEM->PC,CC#=0         0029 0300 30D6 5000
```

测试

1 输入微码

将微码输入到由0900H开始的内存单元中。

用E命令输入微码，回车后输入微程序。每个数值间以空格分开，输入完毕后按回车键。

```
E900
0000 0E01 9190 0088 0029 03E0 A0B5 5402 0029 0300 30D6 5000
```

```
>E900
0900    0000:0000  0000:0E01  0000:9190  0000:0088  0000:0029
0905    0000:03E0  0000:A0B5  0000:5402  0000:0029  0000:0300
090A    0000:30D6  0000:5000
>
```

2 编写用于加载微码的汇编指令

```
A0800
MOV R1, 900
MOV R2, 3
MOV R3, 140
LDMC
RET
```

```
A0800          从0800地址开始输入汇编语句
MOV R1, 900    微码从0900地址开始
MOV R2, 3      有3条微指令
MOV R3, 140    控存为140H，对应操作码为E4~E7 我们使用E5
LDMC           装入微码
RET            汇编语句结束，子程序返回
```

```
>A0800
0800: MOV R1, 900
0802: MOV R2, 3
0804: MOV R3, 140
0806: LDMC
0807: RET
0808:
>
```

3 运行用于加载微码的汇编指令

```
G0800
```

4 测试条件转移的功能

编写用于测试我们做好的微程序的汇编指令

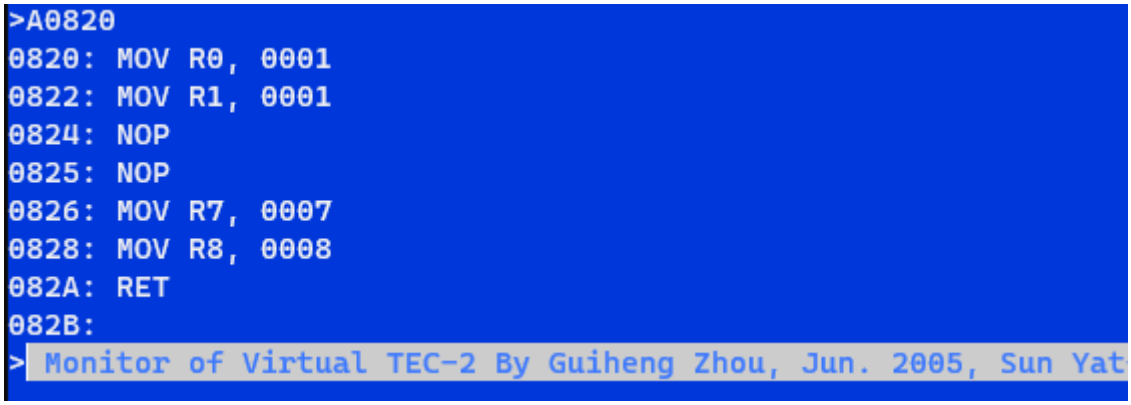
```
A0820
MOV R0, 0001
MOV R1, 0001
NOP
NOP
MOV R7, 0007
MOV R8, 0008
RET
```

R7和R8为了测试goto和顺序执行

DISP在第5步设置为0003

如果发生goto，那么表现为跳过`MOV R7, 0007`，直接执行`MOV R8, 0008`

否则，顺序执行，两条`MOV`都执行



用E命令直接编辑两个NOP

把NOP编辑为我们的双字指令。

```
E0824
E501 0003
```

此时可以用U0820来查看汇编指令。

```
>E0824
0824    0000:E501  0000:0003
>U0820
0820: 2C00 0001  MOV    R0,     0001
0822: 2C10 0001  MOV    R1,     0001
0824: E501       DW     E501
0825: 0003       NOP
0826: 2C70 0007  MOV    R7,     0007
0828: 2C80 0008  MOV    R8,     0008
082A: AC00       RET
082B: 0000       NOP
082C: 0000       NOP
082D: 0000       NOP
082E: 0000       NOP
082F: 0000       NOP
0830: 0000       NOP
0831: 0000       NOP
0832: 0000       NOP
0833: 0000       NOP
>
```

运行用于测试我们做好的微程序的汇编指令

R命令查看寄存器内容

```
R
```

G0820运行程序

```
G0820
```

R命令再次查看寄存器内容

```
R
```

```
>R
R0=0000  R1=090C  R2=0000  R3=0143  SP=FFFF  PC=0800  IP=0807  R7=0000  R8=0000
R9=0000 R10=0000 R11=0000 R12=0000 R13=0000 R14=0000 R15=0000   F=01001111
0800: 2C10 0900  MOV    R1,     0900
>G0820
>R
R0=0001  R1=0001  R2=0000  R3=0143  SP=FFFF  PC=0820  IP=082A  R7=0000  R8=0008
R9=0000 R10=0000 R11=0000 R12=0000 R13=0000 R14=0000 R15=0000   F=01001111
0820: 2C00 0001  MOV    R0,     0001
>
```

可以发现，R7不变，而R8变化，说明goto起作用。

5 测试顺序执行的功能

编写用于测试我们做好的微程序的汇编指令

```
A0820
MOV R0, 0001
MOV R1, 0002
NOP
NOP
MOV R7, 0007
MOV R8, 0008
RET
```

R7和R8为了测试goto和顺序执行

DISP在第5步设置为0003

如果发生goto，那么表现为跳过`MOV R7, 0007`，直接执行`MOV R8, 0008`

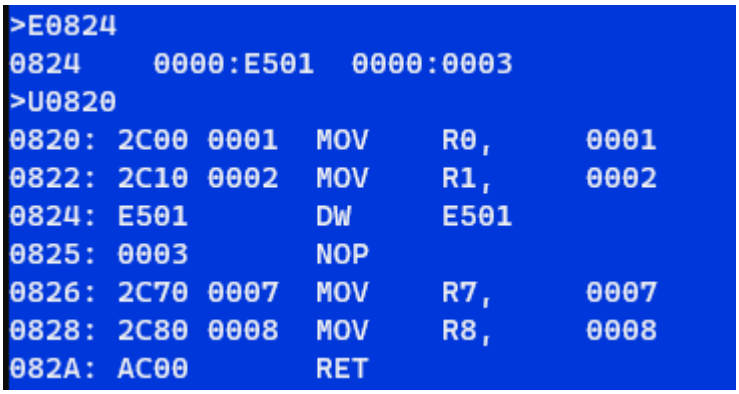
否则，顺序执行，两条`MOV`都执行

R0 != R1，预期效果为顺序执行

把NOP编辑为我们的双字指令。

```
E0824
E501 0003
```

此时可以用U0820来查看汇编指令。



R命令查看寄存器内容

```
R
```

G0820运行程序

```
G0820
```

R命令再次查看寄存器内容

```
R
```

```
>R
R0=0001  R1=0001  R2=0000  R3=0143  SP=FFFF  PC=0820  IP=082A  R7=0000  R8=0008
R9=0000 R10=0000 R11=0000 R12=0000 R13=0000 R14=0000 R15=0000   F=01001111
0820: 2C00 0001  MOV     R0,      0001
>G0820
>R
R0=0001  R1=0002  R2=0000  R3=0143  SP=FFFF  PC=0820  IP=082A  R7=0007  R8=0008
R9=0000 R10=0000 R11=0000 R12=0000 R13=0000 R14=0000 R15=0000   F=00011111
0820: 2C00 0001  MOV     R0,      0001
>
```

可以发现，这次R7也发生变化。说明完成了顺序执行。亦可以在程序运行前将R8清零，会发现程序运行后R8变为0008

四、实验心得

计算机组成原理课程设计是我进入大学以来接触的最困难的课程之一，仅次于 ACM 竞赛。主要问题是遇到的坑太多。从实验三：微程序控制器实验开始，堪称到处都是问题。例如，（1）微程序的微指令怎么来的？它的语法是什么？为什么PC可以直接写在编码里而ADDR1不行？（2）微码是怎么来的？56个二进制变成十六进制怎么有16个？如果微码不小心写错了怎么调试？（3）如何测试编好的微程序？微控存是什么？为什么指令的操作码是这个？为什么要有NOP？

如果解决了全部的这些问题，意味着课程设计已经进入尾声了。这次课程设计就是在边挖坑边填坑的过程中进行。

在课程设计中，75%的内容都是自己看书+看博客自学。下面列出参考资料：

《计算机组成原理实验指导书》

《TEC - 2 计算机组成原理实验系统 简明实验操作手册》

计算机组成原理课设Tec-2 01-基础入门
https://blog.csdn.net/qq_54869075/article/details/125249488

计算机组成原理课设Tec-2 02-实例分析
https://blog.csdn.net/qq_54869075/article/details/125262669

计算机组成原理课设Tec-2 03-指令设计
https://blog.csdn.net/qq_54869075/article/details/126899159

TEC-2 微程序设计与测试
https://www.cnblogs.com/daix6/p/4461895.html

TEC-2几条微指令的微码说明 & TEC-2微程序运行测试步骤
https://www.cnblogs.com/joyeecheung/p/3687773.html

TEC-2机微程序设计实验
https://blog.csdn.net/qq_43594913/article/details/106735166

《计算机组成原理》课程设计报告——TEC-2实验系统——微程序设计
https://blog.csdn.net/weixin_43272781/article/details/106862614