北京邮电大学课程设计报告

<table>
<tr><td>课程设计名称</td><td colspan="2">计算机组成原理课程设计</td><td>学 院</td><td>计算机</td><td>指导教师</td><td>杨秦</td></tr>
<tr><td>班 级</td><td>班内序号</td><td colspan="2">学 号</td><td>学生姓名</td><td colspan="2">成绩</td></tr>
<tr><td>305</td><td></td><td colspan="2">2022211683</td><td>张晨阳</td><td colspan="2"></td></tr>
<tr><td>305</td><td></td><td colspan="2">2022211124</td><td>梁维熙</td><td colspan="2"></td></tr>
<tr><td>305</td><td></td><td colspan="2">2022211122</td><td>龙文涛</td><td colspan="2"></td></tr>
<tr><td>305</td><td></td><td colspan="2">2022211637</td><td>廖轩毅</td><td colspan="2"></td></tr>
<tr><td>课程设计内容</td><td colspan="6">本次课程设计要求设计一个流水硬连线控制器，和 TEC-8 模型计算机的数据通路结合在一起，构成一个完整的 CPU。
我们小组实现了顺序模型处理器、存储器功能、寄存器功能，添加了 4 条额外指令、修改 PC 指针功能等功能。
张晨阳：负责调试代码、设计测试程序、撰写报告
梁维熙：负责编写完整代码、调试代码
龙文涛：负责确定时序信号表、整理资料、调试代码
廖轩毅：负责调试代码、设计测试程序、撰写报告</td></tr>
<tr><td>学生课程设计报告
(附页)</td><td colspan="6"></td></tr>
<tr><td>课程设计成绩评定</td><td colspan="6">遵照实践教学大纲并根据以下四方面综合评定成绩：
1、课程设计目的任务明确，选题符合教学要求，份量及难易程度
2、团队分工是否恰当与合理
3、综合运用所学知识，提高分析问题、解决问题及实践动手能力的效果
4、是否认真、独立完成属于自己的课程设计内容，课程设计报告是否思路清晰、文字通顺、书写规范
评语：

成绩：

指导教师签名：
年 月 日</td></tr>
</table>

注：评语要体现每个学生的工作情况，可以加页。

目录

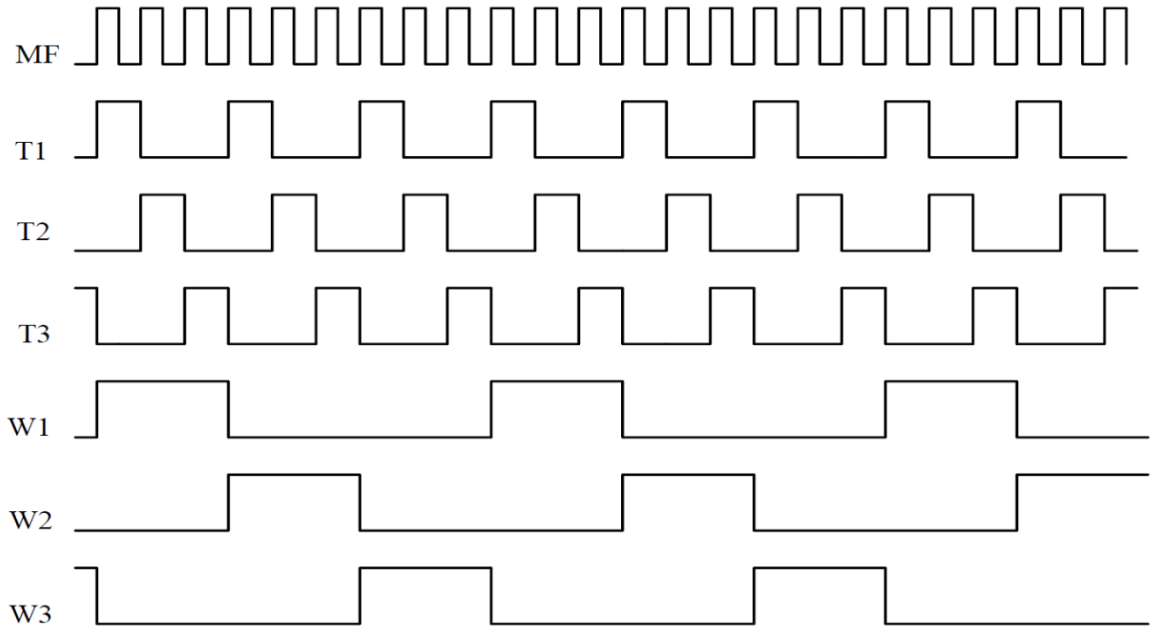
1 硬件环境

本次设计使用开发平台 Quartus Ⅱ 9.1 编写 VHDL 代码，采用搭载了 Altera EPM7128 芯片的 TEC-8 计算机硬件综合实验系统。

1.1 TEC-8 模型计算机时序信号

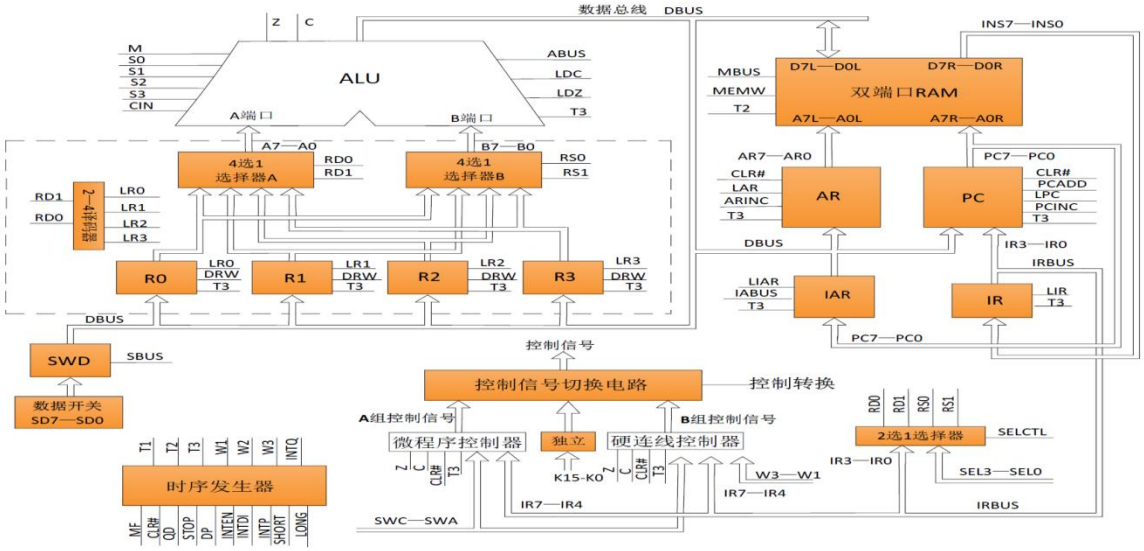
TEC-8 模型计算机主时钟 MF 频率为 1MHz，执行一条微指令需要 3 个节拍脉冲 T1、T2、T3。TEC-8 模型计算机的时序采用不定长的机器周期，大部分微指令包含 2 个机器周期 W1 和 W2，少数微指令包含 1 个机器周期 W1，或者 3 个机器周期 W1、W2、W3。

TEC-8 模型计算机的基本时序波形如下。



1.2 组成模块及数据通路

TEC-8 模型计算机框图如下所示。



1.3 EPM7128 芯片

TEC-8 计算机硬件综合实验系统中的硬连线控制器由 1 片 EPM7128 芯片构成。硬连线控制器和数据通路之间不采用接插线方式连接，在印制电路板上已经用印制导线进行了连接。这就要求硬连线控制器所需的信号的输出、输入信号的引脚号必须符合规定。

硬连线控制器 EPM7128 引脚的规定如下图所示。

信号	方向	引脚号	信号	方向	引脚号
CLR#	输入	1	MEMW	输出	27
T3	输入	83	STOP	输出	28
SWA	输入	4	LIR	输出	29
SWB	输入	5	LDZ	输出	30
SWC	输入	6	LDC	输出	31
IR4	输入	8	CIN	输出	33
IR5	输入	9	S0	输出	34
IR6	输入	10	S1	输出	35
IR7	输入	11	S2	输出	36
W1	输入	12	S3	输出	37
W2	输入	15	M	输出	39
W3	输入	16	ABUS	输出	40
C	输入	2	SBUS	输出	41
Z	输入	84	MBUS	输出	44
DRW	输出	20	SHORT	输出	45
PCINC	输出	21	LONG	输出	46
LPC	输出	22	SEL0	输出	48
LAR	输出	25	SEL1	输出	49
PCADD	输出	18	SEL2	输出	50
ARINC	输出	24	SEL3	输出	51
SELCTL	输出	52			

2 课题描述

2.1 设计任务

按照给定数据格式、指令系统和数据通路，根据所提供的器件要求，自行设计一个基于硬布线控制器的顺序模型处理器，硬件描述语言用 VHDL 或 Verilog 均可。

2.1.1 测试程序

测试程序 1 如下图所示。

地址	指令	二进制机器代码
00H	LD R0,[R3]	01010011
01H	INC R3	01001100
02H	LD R1,[R3]	01010111
03H	SUB R0,R1	00100001
04H	JZ 0BH	10000110
05H	ST R0,[R2]	01101000
06H	INC R3	01001100
07H	LD R0,[R3]	01010011
08H	ADD R0,R1	00010001
09H	JC 0CH	01110010
0AH	INC R2	01001000
0BH	ST R2,[R2]	01101010
0CH	AND R0,R1	00110001
0DH	OUTR2	10100010
0EH	STP	11100000
0FH	85H	10000101
10H	23H	00100011
11H	EFH	11101111

测试程序 2 如下表所示。

地址	指令	二进制机器代码
20H	LD R0，[R3]	01010011
21H	DEC R3	11111100
22H	LD R1，[R3]	01010111

23H	OR R0，R1	10110001
24H	OUT R0	10100000
25H	INC R3	01001100
26H	LD R0，[R3]	01010011
27H	NAND R0，R1	11000001
28H	OUT R0	10100000
29H	LD R0，[R3]	01010011
2AH	NOR R0，R1	11010001
2BH	OUT R0	10100000
2CH	STP	11100000
2DH	23H	00100011
2EH	DCH	11011100

2.1.2 指令系统格式

指令系统格式如下图所示。

名称	助记符	功能	指令格式		
			IR7 IR6 IR5 IR4	IR3 IR2	IR1 IR0
加法	ADD Rd, Rs	Rd ← Rd + Rs	0001	Rd	Rs
减法	SUB Rd, Rs	Rd ← Rd - Rs	0010	Rd	Rs
逻辑与	AND Rd, Rs	Rd ← Rd and Rs	0011	Rd	Rs
加 1	INC Rd	Rd ← Rd + 1	0100	Rd	XX
取数	LD Rd, [Rs]	Rd ← [Rs]	0101	Rd	Rs
存数	ST Rs, [Rd]	Rs → [Rd]	0110	Rd	Rs
C 条件转移	JC addr	如果 C=1，则 PC ← @ + offset	0111	offset	
Z 条件转移	JZ addr	如果 Z=1，则 PC ← @ + offset	1000	offset	
无条件转移	JMP [Rd]	PC ← Rd	1001	Rd	XX
输出	OUT Rs	DBUS ← Rs	1010	XX	Rs
异或	XOR Rd, Rs	Rd ← Rd xor Rs	1011	Rd	Rs
逻辑或	OR Rd, Rs	Rd ← Rd or Rs	1100	Rd	Rs
逻辑非	NOT Rd	Rd ← NOT Rd	1101	Rd	XX
停机	STP	暂停运行	1110	XX	XX

ALU 算数/逻辑运算功能表如下图所示。

工作方式 S3-S0	M=1 逻辑运算	M=0算术运算	
		CIN=1无进位	CIN=0有进位
0000	F=/A	F=A	F=A加1
0001	F=/(A+B)	F=A+B	F=(A+B)加1
0010	F=(/A)B	F=A+/B	F=(A+/B)加1
0011	F=0	F=-1(补码形式)	F=0
0100	F=/(AB)	F=A加A/B	F=A加A/B加1
0101	F=/B	F=(A+B)加A/B	F=(A+B)加A/B加1
0110	F=A⊕B	F=A减B减1	F=A减B
0111	F=A/B	F=(A/B)减1	F=A/B
1000	F=/A+B	F=A加AB	F=A加AB加1
1001	F=/(A⊕B)	F=A加B	F=A加B加1
1010	F=B	F=(A+/B)加AB	F=(A+/B)加AB加1
1011	F=AB	F=AB减1	F=AB
1100	F=1	F=A加A	F=A加A加1
1101	F=A+/B	F=(A+B)加A	F=(A+B)加A加1
1110	F=A+B	F=(A+/B)加A	F=(A+/B)加A加1
1111	F=A	F=A减1	F=A

2.1.3 数据通路

数据通路图请见 1.2 组成模块及数据通路部分。

2.2 基本功能

根据设计方案，在实验平台上进行组装、调试并运行成功。

2.3 附加功能

附加功能包括以下两个功能：

1、在原指令基础上要求扩指功能至少 3 条

2、修改 PC 指针功能（指针任意执行功能）

3 需求分析

所编写的程序需要实现以下功能：顺序模型处理器、存储器功能、寄存器功能、新增指令、修改 PC 指针。

3.1 顺序模型处理器

顺序模型处理器负责按顺序执行指令，这意味着指令的执行是按照程序中指令的先后顺序进行的。处理器需要从指令寄存器中获取指令，根据指令的操作码确定执行的具体操作，并生成相应的控制信号来完成指令的执行。处理器还需要管理时序信号，以确保每个指令在适当的时钟周期内完成。

3.2 存储器功能

存储器功能包括对数据和指令的读写操作。处理器需要能够从指定地址中读取或写入存储单元。

3.3 寄存器功能

寄存器功能包括对 R0、R1、R2、R3 寄存器的读写操作。处理器需要能够根据指令的要求读写特定的寄存器。

3.4 新增指令

在原指令的基础上，我们需要扩充四条指令：OR、NAND、NOR、DEC。

其中，OR 为逻辑或，NAND 为逻辑与非，NOR 为逻辑或非，DEC 为自减。各指令助记符及功能如下表所示。

名称	助记符	功能
逻辑或	OR Rd，Rs	Rd ← Rd or Rs
逻辑与非	NAND Rd，Rs	Rd ← not (Rd and Rs)
逻辑或非	NOR Rd，Rs	Rd ← not (Rd or Rs)
DEC	DEC Rd	Rd ← Rd - 1

3.5 修改 PC 指针

修改 PC 指针功能指可以指定 PC 的值，从而可以指定程序从哪里开始执行。

4 概要设计

4.1 硬连线控制器

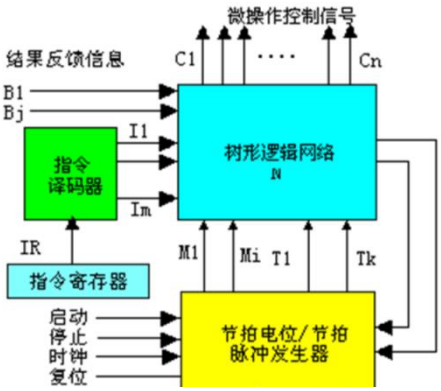
4.1.1 硬连线控制器基本原理

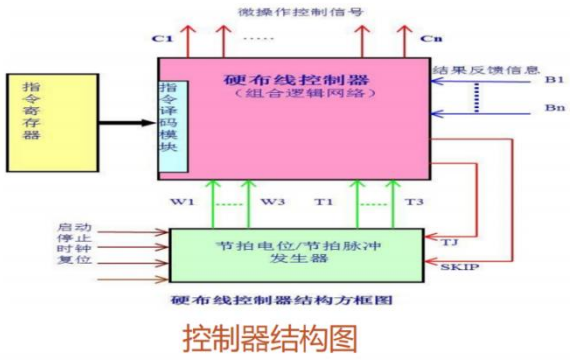
硬连线控制器也称硬布线控制器，把控制器看作产生控制信号的逻辑电路，由门电路和触发器构成。即：每个微操作控制信号（输出）是一系列由组合逻辑实现的输入信号的组合：

$$S = f(I_m, M_i, T_k, B_j)$$

其中，I_m是机器指令操作码译码器的输出信号，M_i是节拍电位信号，T_k是节拍脉冲信号，B_j是执行部件反馈信息，即状态条件信号。

如下图左：





控制器结构图

而在本实验系统中，如上图右：

T_k节拍脉冲信号直接输送给数据通路，即模型计算机框图中的T_1, T_2, T_3。

由于本系统机器指令系统较简单，省去了操作码译码器，4 位指令操作码 $IR4 \sim IR7$ 直接作为I_m的一部分；又由于 TEC-8 实验系统有控制台操作，控制台操作可以看作一些特殊的功能复杂的指令，比如 001 对应写存储器，010 对应读存储器等，因此 SWC、SWB、SWA 也可以看作是 I_m 的另一部分。

M_i 是时序发生器产生的节拍信号 $W1 \sim W3$。

B_j包括 ALU 产生的进位信号 C、零信号 Z 等等。

4.1.2 硬连线控制器 VS 微程序控制器

上文已经介绍了硬连线控制器，这里首先介绍一下微程序控制器。

微程序控制器要由控制存储器、微指令寄存器和地址转移逻辑三大部分组成，其中微指令寄存器分为地址寄存器和微命令寄存器两部分。如下图：

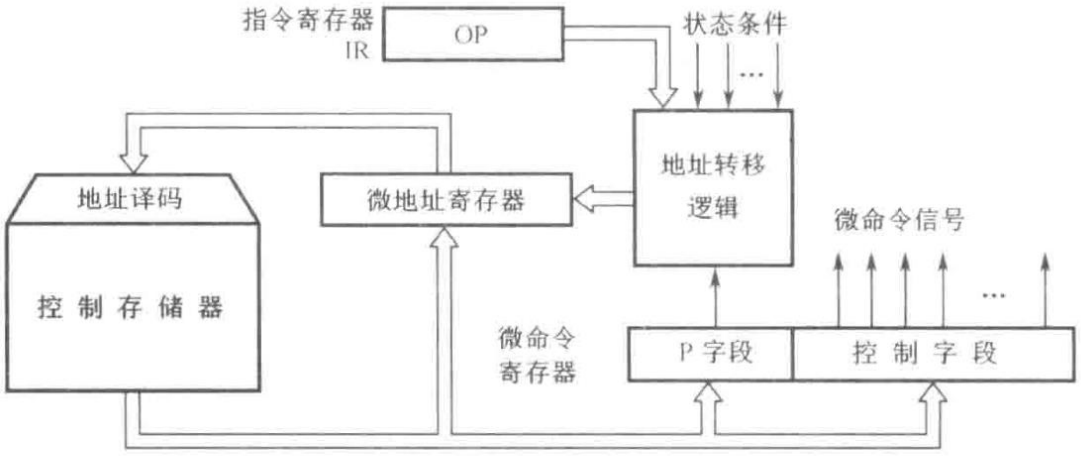


图 5.24 微程序控制器组成原理框图

通常来说，硬连线控制器速度快（控制器的速度取决于电路延迟），一旦设计完成后，就不能通过其他额外修改添加新功能（将控制部件视为专门产生固定时序控制信号的逻辑电路，用最少元件和取得最高速度作为设计目标）。

而微程序控制器具有规整性，灵活性，可维护性等优点，但由于微程序控制器采用了存储程序原理，所以每条指令都要从控制存储器中取一次，影响速度。

4.2 时序信号

4.2.1 节拍电位数

在 TEC-8 实验系统中，采用了可变节拍电位数来执行一条机器指令。

大部分指令的执行只需 2 个节拍电位 W1、W2，如：ADD、INC 等，少数指令需要 3 个节拍电位 W1、W2、W3，如：LD、ST。

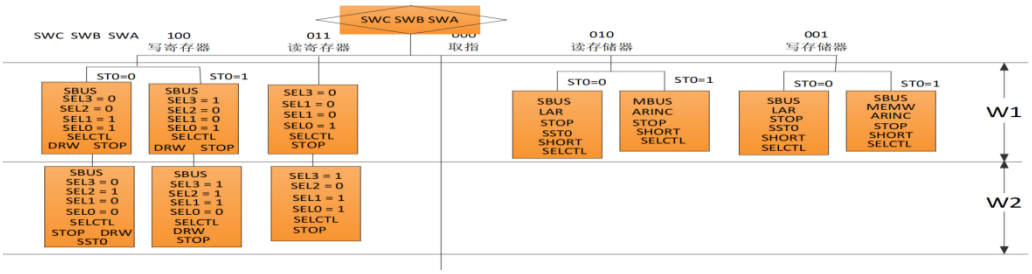
为了满足这种要求，在执行一条指令时除了产生完成指令功能所需的微操作控制信号外，对需要 3 个电位节拍的指令，还要求它在 W2 时产生一个信号 LONG。信号 LONG 送往时序信号发生器，时序信号发生器接到信号 LONG 后产生节拍电位 W3。

为了适应更为广泛的情况，TEC-8 的时序信号发生器允许只产生一个节拍电位 W1。当 1 条指令或者一个控制台在 W1 时，只要产生信号 SHORT，该信号送往时序信号

发生器，则时序信号发生器在 W1 后不产生节拍电位 W2，下一个节拍仍是 W1。

信号 LONG 和 SHORT 只对紧跟其后的第一个节拍电位的产生起作用。

4.2.2 ST0 标志位



对于一些控制台操作，如上图中的写寄存器操作，需要 4 个节拍电位才能完成规定的功能。为了满足这种情况，我们考虑将控制台操作化成两条机器指令的节拍。为了区分写寄存器操作的 2 个不同阶段，考虑用某些特殊的寄存器标志。我们建立了一个 ST0 标志，当 ST0=0 时，表示该控制台操作的第 1 个阶段；当 ST0=1 时，表示该控制台操作的第 2 个阶段。

举例来说，对于写存储器，当 ST0=0 时，表示这是写存储器的第一次操作，需要先写入 PC 的地址（第一阶段），随后 ST0 置 1，进入依次写入存储器的操作阶段（第二阶段）。

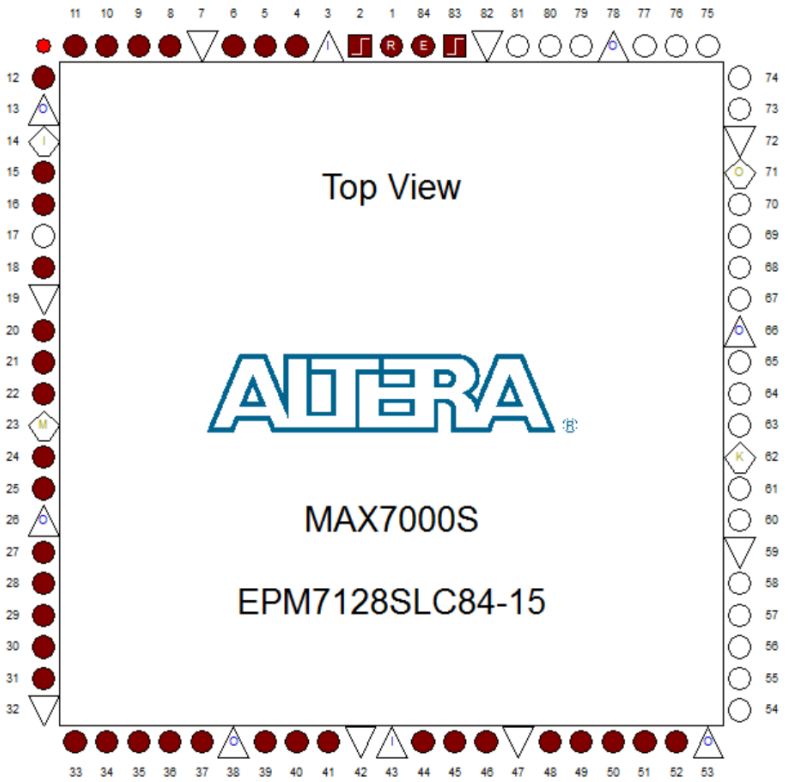
4.3 EPM7128 器件引脚

TEC-8 实验系统中的硬连线控制器是用 1 片 EPM7128 器件构成的。硬连线控制器和数据通路之间不采用接插线方式连接，在印制电路板上已经用印制导线进行了连接。这就要求硬连线控制器所需的信号的输出、输入信号的引脚号必须符合下图中的规定：

表 3.3 作为硬连线控制器时的 EPM7128 引脚规定

信号	方向	引脚号	信号	方向	引脚号
CLR#	输入	1	MEMW	输出	27
T3	输入	83	STOP	输出	28
SWA	输入	4	LIR	输出	29
SWB	输入	5	LDZ	输出	30
SWC	输入	6	LDC	输出	31
IR4	输入	8	CIN	输出	33
IR5	输入	9	S0	输出	34
IR6	输入	10	S1	输出	35
IR7	输入	11	S2	输出	36
W1	输入	12	S3	输出	37
W2	输入	15	M	输出	39
W3	输入	16	ABUS	输出	40
C	输入	2	SBUS	输出	41
Z	输入	84	MBUS	输出	44
DRW	输出	20	SHORT	输出	45
PCINC	输出	21	LONG	输出	46
LPC	输出	22	SEL0	输出	48
LAR	输出	25	SEL1	输出	49
PCADD	输出	18	SEL2	输出	50
ARINC	输出	24	SEL3	输出	51
SELCTL	输出	52			

我们的引脚连接图如下：



4.4 控制信号详解

本实验的控制信号详解如下。

名称	类型	功能
SWC, SWB, SWA	in std_logic	用于控制控制台的操作
CLR	in std_logic	拥有最高级别，用于重置
C, Z	in std_logic	进位信号和零标志位
W1, W2, W3, T3, QD	in std_logic	w1、w2、w3 是机器周期，T3 用它的下降沿定位
IRHIGH	in std_logic_vector (3 downto 0)	高四位的 IR
SELCTL	out std_logic	控制台信号
ABUS, M, CIN	out std_logic	ABUS 为 1 时允许 ALU 数据流出到 DBUS 上； M 用于控制 ALU 是逻辑运算还是算数运算； CIN 用于区分 S 码和 M 对应的操作是有进位操作或是无进位操作
S	out std_logic_vector (3 downto 0)	用于控制 ALU 产生不同的函数
SEL3, SEL2, SEL1, SEL0	out std_logic	用于 2-4 选择
DRW, SBUS	out std_logic	DRW 用于控制修改寄存器信号； SBUS 用于控制是否从手动开关读入数据至总线中
LIR	out std_logic	LIR 为 1 时允许指令送到指令寄存器
MBUS, MEMW	out std_logic	MBUS 为 1 时允许双端口存储器的数据送到数据总线； MEMW 为 1 时允许写入存储器
LAR, ARINC	out std_logic	LAR 为 1 时允许地址送入双端口存储器；

		ARINC 为 1 时允许地址自增
LPC, PCINC, PCADD	out std_logic	LRP 为 1 时允许将 DBUS 中的内容写入 PC 寄存器； PCINC 为 1 时允许 PC 自增； PCADD 用于控制 PC 的相对偏移
STOP	out std_logic	停机指令
LDC, LDZ	out std_logic	用于控制送进位信号和零标志位
LONG, SHORT	out std_logic	SHORT 置 1 时仅有 W1 周期； LONG 置 1 时则有 W1、W2、W3 周期

4.5 设计思路

硬连线控制器的常规 CPU 主要实现写寄存器、读寄存器、写存储器、读存储器、执行指令五项大功能，我小组采用分模块设计的思想来设计 VHDL 可执行程序，具体步骤如下：

4.5.1 初始化

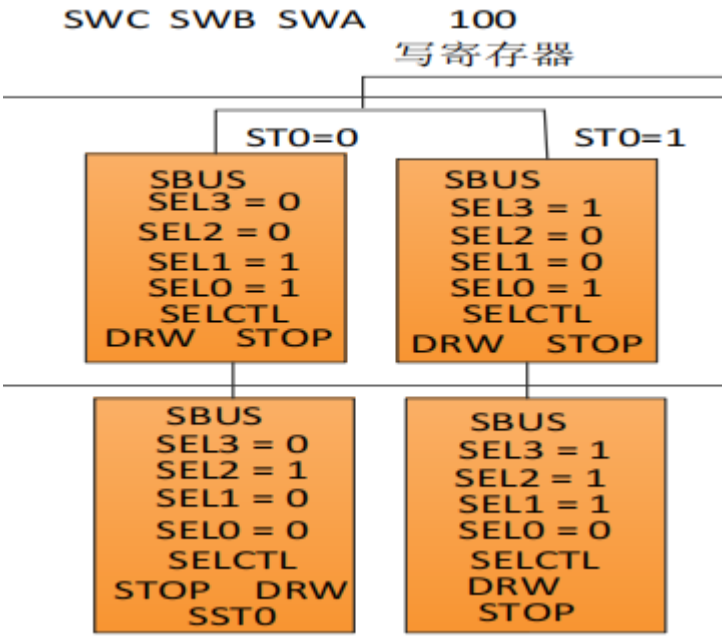
将输出信号初始化，每次启动进程，即每次有变量变化，都会启动进程，先将这些信号置为 0。

采用时序逻辑，在进程的最开始先把所有的输出信号置为 0，这样，就可以使得在进程结束之后没有被后面的程序所覆盖的信号直接为 0，不会成为错误信号干扰程序的进行。

对于 CLR，在我们的程序里，clr 是敏感信号的其中之一，故每次发生变化（按下 clr），都会启动进程，也就完成了所有元素的清零，并且在此时，需要将 ST0 置 0.

4.5.2 写寄存器

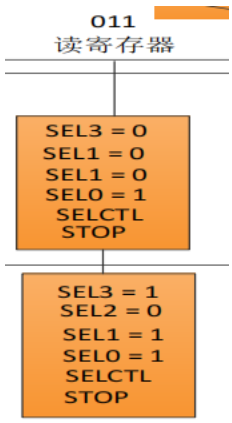
当 SWCBA='100'时，系统进入写寄存器操作。



在写寄存器操作中，由于写操作分为两个部分，即初始进入初始写操作部分和循环写操作部分。这种情况下，FLAG 标志 NEXTST0 和 STO 发挥了它们真正的作用，当 NEXTST0 为'1'时，表明可进入循环写操作模式，赋值 ST0 为有效位为'1'。

4.5.3 读寄存器

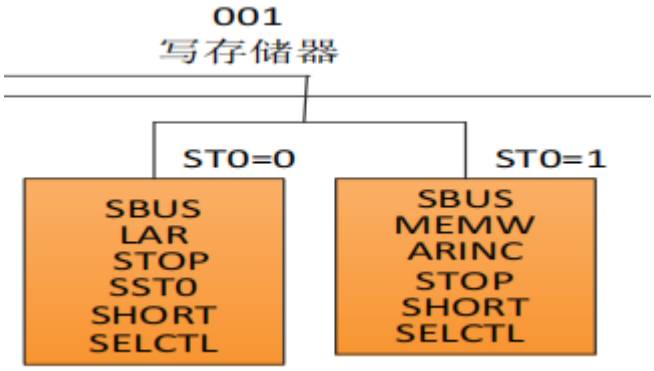
当 SWCBA='011'时，系统进入读寄存器操作。



在之前的计算机组成原理实验中我们知道，由于 TEC-8 上一共 4 个寄存器，所以通过 A7~A0 和 B7~B0 一次读两个的方式读出寄存器中的内容，所以在实际操中按两次 QD 即可。操作执行需要两个节拍脉冲，即 W1 和 W2。

4.5.4 写存储器

当 SWCBA='001'时，系统进入写存储器操作。

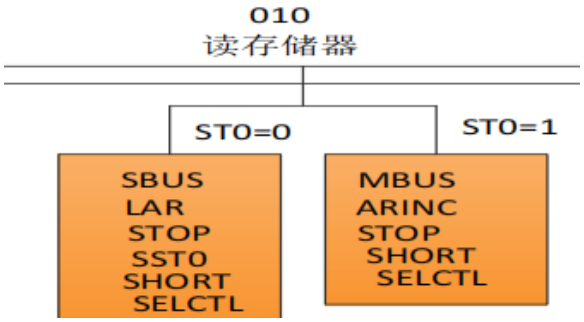


在该操作下存在初始写操作模式和循环写操作模式，即指定首地址、依次写入存储单元两个阶段，我们首先在 SD7~SD0 上设置存储器地址，按 QD 打入到地址寄存器 AR。之后不断在数据开关上设置数据，写入即可。

NEXTST0 的引入保证了在正确的节拍脉冲中开启，外部的 T3 下降沿又保证了 ST0 只有在 T3 下降沿才会改变。这样才能保证写存储器操作能够去按照正确的顺序进行执行。

4.5.5 读存储器

当 SWCBA='010'时，系统进入读存储器操作。



由于在读存储器操作中存在初始读操作模式和循环读操作模式，即指定首地址、依次读取存储单元两个阶段，因此在此处需要 ST0 和 NEXTST0。

此外，由于读操作为短操作，正常情况下控制台产生 W1 和 W2 两个节拍脉冲，为了终止 W2 节拍脉冲的产生，所以在 W1 节拍结束之后需要置 SHORT 为有效位‘1’,TEC-8 控制台在接到 SHORT 有效的信号时，将会终止产生 W2 脉冲信号，从而进入循环写操作模式中来。

4.5.6 取指操作

当 SWCBA='000'时，系统进入取指操作中。

在此操作中可选择执行 16 项计算机指令操作，分别为 ADD（加法）、SUB（减法）、AND（逻辑与）、INC（自增加 1）、LD（取数）、ST（存数）、JC（C 条件转移）、JZ（Z 条件转移）、JMP（无条件转移）、STP（停机）、OUT（输出）、OR（逻辑或）、NAND（与非）、NOR（或非）、DEC（自减）、空。

IR7~IR4 信号唯一标志了采用什么样的操作，在进入取指操作后，会根据 IR 的具体数值来进行那种动作。

其中，LD（取数）、ST（存数）需要在三个节拍电位中执行，而其余的操作须在两个节拍电位中执行完毕。因此，在 LD、ST 指令执行到 W2 节拍电位时，须将 LONG 输出信号置为有效位'1'，表示在产生完 W2 节拍信号后，TEC-8 实验平台还需要继续产生 W3 节拍电位，保证 LD、ST 操作能够完整地执行，其处理方法与标志 SHORT 信号有效，终止产生 W2 信号的方式相似。

5 团队分工

团队成员及分工情况如下表所示。

团队成员	分工情况	贡献度（总共 100）
张晨阳	调试代码、设计测试程序、撰写报告	25
梁维熙	编写完整代码、调试代码	25
龙文涛	确定时序信号表、整理资料、调试代码	25
廖轩毅	调试代码、设计测试程序、撰写报告	25

6 设计详解

6.1 ST0 标志功能

我们在 VHDL 可执行程序中设置有名为 NEXTST0 和 ST0 的 FLAG 标志，分别标志是否拥有第二阶段，以及进入第二阶段的标志。

我们根据硬连线控制器周期流程图中 NEXTST0 有效的情况，写出其对应的逻辑表达式，当逻辑表达式有效，即 NEXTST0 为有效'1'时，置 ST0 为有效位'1'，标志可以进入到第二阶段中。

注意：当 NEXTST0=1 时，由于只有在执行完所有 ST0=0 下的操作后，我们才能够转入对应的 ST0=1 的情况，这就要求我们把握置 ST0 为有效位‘1’的时机，我小组采用 T3 时刻下降沿置数的方式，能够保证程序按照正常的逻辑进行执行，不会出现抢先执行的情况，保证了程序能够有序地执行。

其核心代码如下：

```
1.    if (T3'event and T3 = '0') and NEXTST0 = '1' then
2.       ST0 <= '1';
3.    end if;
```

6.2 写寄存器

首先我们需要打开两个开关：SBUS 和 DRW。

其中 SBUS 置 1，允许我们将数据开关的值送到数据总线上。DRW 置 1，这样在 T3 的上升沿，对 RD1、RD0（SEL3、SEL2 选中的寄存器）进行写操作，将数据总线上的值写入选定的寄存器。

写寄存器的操作需要 4 个时钟周期，我们将其分成两条机器指令的时钟周期，通过 ST0 来区分。具体步骤如下：

第一个 W1 周期：

SEL3、SEL2 设为 00，表示选择寄存器 R0 进行写入。

SEL1、SEL0 设为 11，表示上一次操作的是寄存器 R3。

此时的数据总线上的值会写入 R0。

第一个 W2 周期：

SEL3、SEL2 设为 01，表示选择寄存器 R1 进行写入。

SEL1、SEL0 设为 00，表示上一次操作的是寄存器 R0。

在这个周期内，我们将 NEXTST0 标志置为 1。

在 T3 的下降沿，ST0 会从 0 变为 1，表示我们进入到第二个 W1、W2 周期。

接下来的周期与前面的操作类似，每次 SEL3、SEL2 指定当前要写入的寄存器，SEL1、SEL0 指定上一次写入的寄存器。

核心代码如下：

```
when "100" =>
    SELCTL <= W1 or W2;
    SBUS <= W1 or W2;
    STOP <= W1 or W2;
    NEXTST0 <= W2;
    DRW <= W1 or W2;

    SEL3 <= (ST0 and W1) or (ST0 and W2);
    SEL2 <= W2;
    SEL1 <= ((not ST0) and W1) or (ST0 and W2);
    SEL0 <= W1;
```

6.3 读寄存器

读取寄存器的操作需要两个时钟周期，我们将其分为两个节拍来进行。具体步骤如下：

W1 周期：

将 SEL3、SEL2 设为 00，表示选择寄存器 R0 进行读取。

将 SEL1、SEL0 设为 01，表示选择寄存器 R1 进行读取。

在这个周期内，R0 的值会被送往 ALU 的 A 端口，R1 的值会被送往 ALU 的 B 端口。我们可以通过观察指示灯 A7 到 A0 和 B7 到 B0 来看到 R0 和 R1 的值。

W2 周期：

将 SEL3、SEL2 设为 10，表示选择寄存器 R2 进行读取。

将 SEL1、SEL0 设为 11，表示选择寄存器 R3 进行读取。

在这个周期内，R2 的值会被送往 ALU 的 A 端口，R3 的值会被送往 ALU 的 B 端口。同样，我们可以通过观察指示灯 A7 到 A0 和 B7 到 B0 来看到 R2 和 R3 的值。

```
when "011" =>
    SELCTL <= W1 or W2;
    STOP <= W1 or W2;

    SEL0 <= W1 or W2;
```

```
6.      SEL1 <= W2;
7.      SEL2 <= '0';
8.      SEL3 <= W2;
```

6.4 写存储器

写存储器的操作可以分为两个步骤：指定首地址和依次写入存储单元。我们依然通过 ST0 来区分这两个步骤。

指定首地址（ST0 = 0）：

打开 SBU S，允许数据开关的值送到数据总线;

打开 LAR，将数据总线上的值送入地址寄存器 AR。这实现了指定首地址的操作;

将 NEXTST0 置为 1，表示我们即将进入 ST0 = 1，也就是写存储器的阶段。

写存储器（ST0 = 1）：

打开 MEMW，这样当前数据总线上的值就会存入指定的存储单元;

执行 ARINC，将地址寄存器 AR 的值+1，这样 AR 就指向下一个存储单元。

在下一个节拍中，可以向下一个存储单元写入值。

```
1. when "001" =>
2.     SBUS <= W1;
3.     STOP <= W1;
4.     SHORT <= W1;
5.     SELCTL <= W1;
6.     NEXTST0 <= W1;
7.
8.     LAR <= W1 and (not ST0);
9.     ARINC <= W1 and ST0;
10.    MEMW <= W1 and ST0;
```

6.5 读存储器

读存储器的操作可以分为两个步骤：指定首地址和依次读取存储单元。我们通过ST0 来区分这两个步骤。

指定首地址（ST0 = 0）：

打开 SBUS，使数据开关的值能够送到数据总线;

打开 LAR，将数据总线上的值送入地址寄存器 AR，实现指定首地址的操作;

将 NEXTST0 置为 1，表示我们即将进入 ST0 = 1，即读存储器的阶段。

读存储器（ST0 = 1）：

打开 MBUS，这样当前指定存储单元的值就会被读到数据总线上;

通过 D7 到 D0 指示灯，可以观察到读取的数据值;

执行 ARINC，将地址寄存器 AR 的值加一，使其指向下一个存储单元;

在下一个节拍中，读取下一个存储单元的值。

```
when "010" =>
    STOP <= W1;
    SHORT <= W1;
    SELCTL <= W1;
    NEXTST0 <= W1;

    SBUS <= W1 and (not ST0);
    LAR <= W1 and (not ST0);
    MBUS <= W1 and ST0;
    ARINC <= W1 and ST0;
```

6.6 取指操作

在以上四个模块的支持下，我们可以进行程序的执行。

首先指定起始 PC 地址，实现方式见 6.7.

我们通过 LIR、PCINC 来取出第一条指令，开始执行我们的程序。之后在每个机器指令结束时取出下一条指令，在下一个周期执行该指令。

其中涉及各指令操作码的编写，我们在设计程序前已经完成。

核心代码如下：

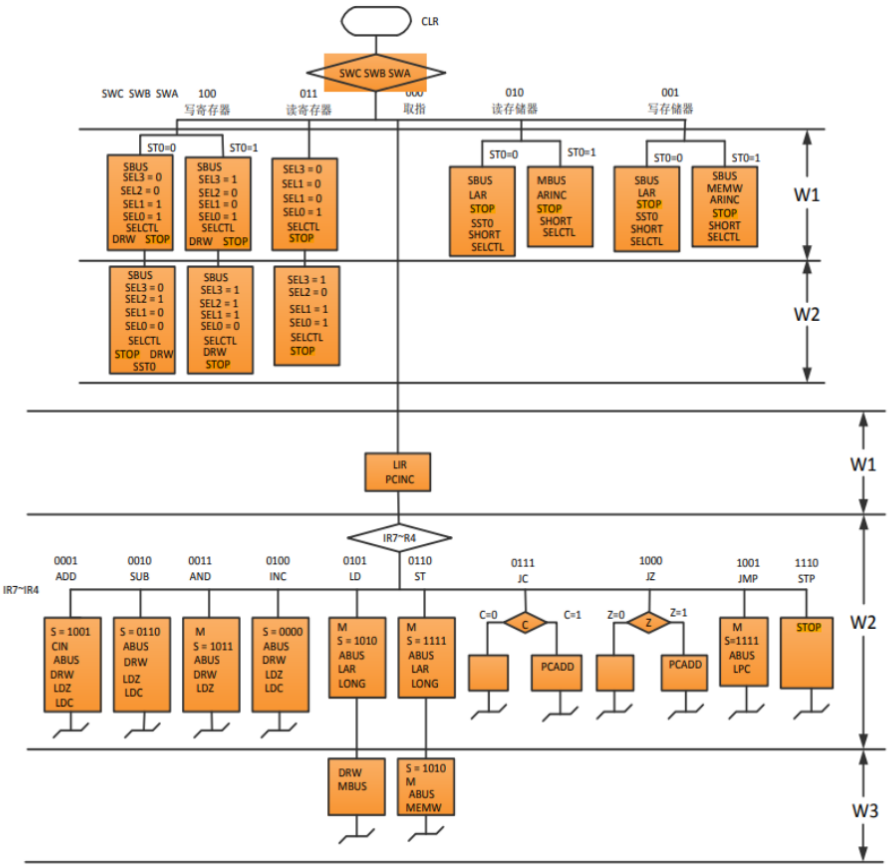
```
LIR <= W1;
PCINC <= W1;
case IRHIGH is
    --加法指令 ADD
    when "0001" =>
        S <= w2 & '0' & '0' & w2;   --F = A 加 B
        CIN <= W2;
        ABUS <= W2;
        DRW <= W2;
        LDZ <= W2;
        LDC <= W2;
    --减法指令 SUB
    when "0010" =>
        S <= '0' & w2 & w2 & '0';   --F = A 减 B
        ABUS <= W2;
        DRW <= W2;
        LDZ <= W2;
        LDC <= W2;
    -- AND 指令
    when "0011" =>
        M <= W2;
        S <= w2 & '0' & w2 & w2;    --F = AB
        ABUS <= W2;
        DRW <= W2;
        LDZ <= W2;
    --自增加一指令 INC
    when "0100" => --INC
        S <= '0' & '0' & '0' & '0'; -- F = A 加 1
        ABUS <= W2;
        DRW <= W2;
        LDZ <= W2;
        LDC <= W2;
    --取数指令 LD
```

```
34.     when "0101" => --LD
35.         M <= W2;
36.         S <= w2 & '0' & w2 & '0' ;  --F = B
37.         ABUS <= W2;
38.         LAR <= W2;
39.         LONG <= W2;
40.
41.         DRW <= W3;
42.         MBUS <= W3;
43.     --存数指令 ST
44.     when "0110" => --ST
45.         M <= W2 or W3;
46.         S <= (W2 or W3) & W2 & (W2 or W3) & W2; --W2: F = A W3: F = B
47.         ABUS <= W2 or W3;
48.         LAR <= W2;
49.         LONG <=W2;
50.
51.         MEMW <=W3;
52.     --有条件跳转 JC
53.     when "0111" => --JC
54.         PCADD <= W2 and C;
55.     --有条件跳转 JZ
56.     when "1000" => --JZ
57.         PCADD <= W2 and Z;
58.     --无条件跳转 JMP
59.     when "1001" => --JMP
60.         M <=W2;
61.         S <= W2 & W2 & W2 & W2; --F = A
62.         ABUS <= W2;
63.         LPC <=W2;
64.     --停机指令 STP
65.     when "1110" => --STP
66.         STOP <= W2;
67.     --展示数的指令 out
68.     when "1010" => -- out
69.         M <= W2;
70.         S <= W2 & '0' & W2 & '0';   --F = B
71.         ABUS <= W2;
72.     --接下来是扩展的 通过 ALU 的不同功能来拓展
73.     --或 or
74.     when "1011" => --或
75.         M <= W2;
76.         S <= W2 & W2 & W2 & '0';    --F = A + B
77.         ABUS <= W2;
```

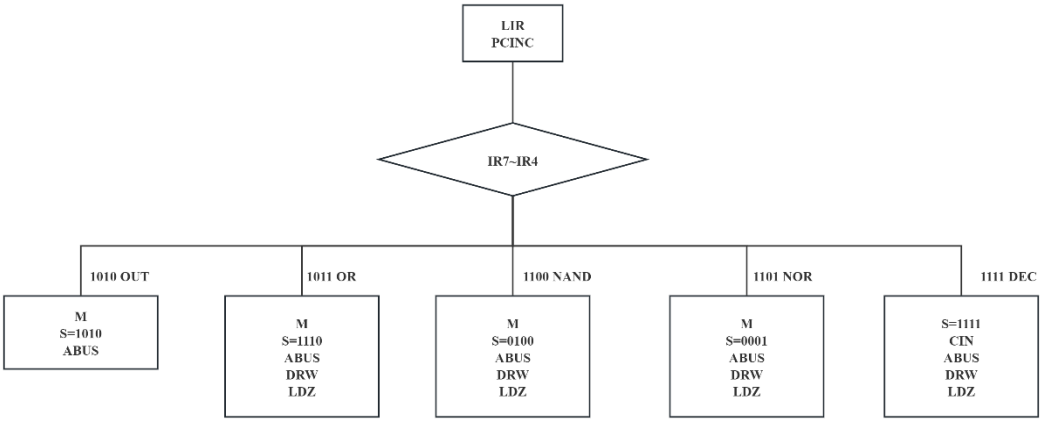
```
78.         DRW <= W2;
79.         LDZ <= W2;
80.     --与非
81.     when "1100" =>
82.         M <= '1';
83.         S <= '0' & W2 & '0' & '0';  --F = /(AB)
84.         ABUS <= W2;
85.         DRW <= W2;
86.         LDZ <= W2;
87.     --或非
88.     when "1101" =>
89.         M <= '1';
90.         S <= '0' & '0' & '0' & W2;  --F = /(A + B)
91.         ABUS <= W2;
92.         DRW <= W2;
93.         LDZ <= W2;
94.     --自减
95.     when "1111" =>
96.         CIN <= W2;
97.         S <= W2 & W2 & W2 & W2; --F = A 减 1
98.         ABUS <= W2;
99.         DRW <= W2;
100.        LDZ <= W2;
```

6.7 扩充指令

前十条指令，我们参考课程给出的硬布线控制器流程图，如下图：



在这张图的基础上，我们添加了 5 条指令：OUT（输出）、OR（逻辑或）、NAND（与非）、NOR（或非）、DEC（自减），其对应的操作码和控制信号如下：



6.8 修改 PC 指针

我们使用 ST0 来实现这个功能。

修改 PC 指针功能指的是我们可以执行任意指令，即：指定某个指令的地址开始执行。这与我们前文所说的分为两个阶段相似。

故我们设计了这样的实现：初始 ST0=0，置 SBUS 和 LPC 为 1，将数据开关上的值，即我们希望执行的指令的地址，存到 PC 里，而 PC 存储的是下一条指令的地址，这就完成了我们希望指定执行的功能。

同时我们将 NEXTST0 置为 1。这使得 ST0 也会置为 1，进入第二阶段开始按顺序执行我们的程序。

核心代码如下：

```
1. when "000" =>
2.     if ST0 = '0' then
3.         LPC <= W1; --这里是 PC 的指令拓展 为了可以置入 PC 的值
4.         SBUS <= W1;
5.         NEXTST0 <= W1;
6.         SHORT <= W1;
7.         STOP <= W1;
8.         SELCTL <= W1;
```

7 调试过程中的问题及讨论

7.1 USB 驱动识别问题

在实验室中使用下载器进行烧录工作时个人电脑无法正常识别，将 Altera USB-Blaster 识别为 USB-Blaster，在 Quartus II 中无法识别，无法正常进行烧录。

通过在设备管理器中安装 JTAG 服务，同时重新插入设备进行识别，然后在任务管理器中使用手动搜索功能进行寻找即可找到。

但是在寻找资料时发现了另一种无法识别设备的可能：文件的哈希值不在指定的目录文件中。如果是这种情况，需要在控制窗口中运行代码 shutdown.exe /r /o /f /t 00 来修改安全模式列表，禁用驱动程序强制名单来安装我们使用的下载器的驱动。

7.2 执行命令中出现意外信号

在使用设计的测试程序 2 进行调试的过程中，我们的硬连线控制器执行程序中第三个 OUT R0 指令的 W1 阶段时，本应为全 0 的 DBUS 中出现了数值为 20H。

测试程序 2 如下表所示。

地址	指令	二进制机器代码
20H	LD R0，[R3]	01010011
21H	DEC R3	11111100
22H	LD R1，[R3]	01010111
23H	OR R0，R1	10110001
24H	OUT R0	10100000
25H	INC R3	01001100
26H	LD R0，[R3]	01010011
27H	NAND R0，R1	11000001
28H	OUT R0	10100000
29H	LD R0，[R3]	01010011
2AH	NOR R0，R1	11010001
2BH	OUT R0	10100000
2CH	STP	11100000
2DH	23H	00100011

2EH	DCH	11011100

排查过程中，我们首先检查了代码，怀疑是编写代码时，在这一步打开了 SBUS，使得最初设置 PC 的起始值为 20H 的数据开关的值进入了 DBUS。但经过检查，我们发现并不是这个原因。

随后我们考虑逐步逆推，找到问题的根源：利用可修改起始地址的功能，从出现问题的地方逐步回退，直到复现这一问题。比如 OUT R0 指令的地址为 0x2B，我们依次从 2AH、29H、28H 等地址开始，执行到 2BH 处，观察是否出现数值为 20H 的问题。

通过这一方法我们发现：从执行程序中的 OR R0，R1 指令开始时，会导致该 OUT 指令的 W1 阶段中 DBUS 值为 20H。

在找到问题的稳定复现方法后，我们在网络上进行相关资料搜索，但是在查找的资料中未能够找到相关的解决办法。

在验收当天，老师指出为硬件方面的问题。

8 设计调试小结

8.1 测试程序 1

测试程序 1 如下图所示。

地址	指令	二进制机器代码
00H	LD R0,[R3]	01010011
01H	INC R3	01001100
02H	LD R1,[R3]	01010111
03H	SUB R0,R1	00100001
04H	JZ 0BH	10000110
05H	ST R0,[R2]	01101000
06H	INC R3	01001100
07H	LD R0,[R3]	01010011
08H	ADD R0,R1	00010001
09H	JC 0CH	01110010
0AH	INC R2	01001000
0BH	ST R2,[R2]	01101010
0CH	AND R0,R1	00110001
0DH	OUTR2	10100010
0EH	STP	11100000
0FH	85H	10000101
10H	23H	00100011
11H	EFH	11101111

8.1.1 测试 ALU 相关操作

利用 ALU 为组合逻辑器件的特性，实现在 W1 阶段即可直接判断计算结果是否为理想中的结果。同时，通过不同的指示灯来判断是否正确给对应的信号赋值。如 ADD R0，R1，在 W2 阶段，给 S3_0 置为 1001，CIN、ABUS、LDZ、LDC、CIN 置为 1，而这些灯在机器上均有相对应的指示灯进行展示，通过这些灯即可判断程序是否正确执行。

8.1.2 测试跳转操作

在测试程序中有 JZ、JC 操作，分别判断 Z、C 寄存器是否为 1 来判断是否进行跳转。所以在对应的跳转指令前使用相关的运算操作修改这两个寄存器，其中，在第一个跳转操作前我们让 Z 寄存器中置为 0，所以在执行操作时没有进行跳转操作；然后，在第二个跳转操作时，我们让 C 寄存器中值变化为 1，所以第二次跳转操作成功执行。

8.1.3 测试输出指令

在程序中使用 OUT 命令来输出对应寄存器中的值，如在 0DH 处，将 R2 寄存器中的值输出至 DBUS 中，同时由于 IR 指令的低四位自动进入 ALU 的 A 端口、B 端口的端口译码器中，所以我们可以看到两个端口的指示灯分别为相应寄存器中的值。然后在 W2 阶段将 B 端口的值输出至 DBUS 中。

8.1.4 测试停止指令

在实验机器上我们完成修改起始 PC 操作后，将 DP 开关置为 0。此时，我们再次给出 QD 信号，程序会自动运行至 STP 命令处停止；若不是，则停止指令出错。

8.2 测试程序 2

测试程序 2 如下表所示。

地址	指令	二进制机器代码
20H	LD R0，[R3]	01010011
21H	DEC R3	11111100
22H	LD R1，[R3]	01010111
23H	OR R0，R1	10110001
24H	OUT R0	10100000
25H	INC R3	01001100
26H	LD R0，[R3]	01010011
27H	NAND R0，R1	11000001
28H	OUT R0	10100000
29H	LD R0，[R3]	01010011
2AH	NOR R0，R1	11010001
2BH	OUT R0	10100000
2CH	STP	11100000
2DH	23H	00100011
2EH	DCH	11011100

8.2.1 测试自减操作

程序中先使用第一个数作为初始地址存入 R3 中等待使用，同时将第二个数写入低一位的地址等待使用。当进行取值操作时，进行自减操作使取数地址变动至第二个数相

对应的地址。且由于后续操作均将结果覆盖至 R0，所以我们再次将 R3 使用自增操作变换为第一个数的地址等待覆盖。

8.2.2 测试逻辑操作

在此处，我们使用两个互补的数作为测试程序的操作数，在后续进行逻辑操作时，方便立刻判断结果是否正确，如判断与非操作时，由于两个数为互补，所以其与结果必为全 0，进行非操作后结果为全 1。同时，通过不同的指示灯来判断是否正确给对应的信号赋值。

附件

附件 1 小组各成员心得总结

张晨阳心得总结

通过本次课程设计，我加深了对硬连线控制器的理解。

在计算机组成原理理论课的期末复习阶段时，我就对设计硬连线控制器的部分印象深刻。当时课程内容的步骤是：画出指令流程图、设计控制操作时序、进行微操作综合、电路实现。相关的题目我也进行了深入研究。而本次实践也让我对硬件更加感兴趣，特别是自己设计测试程序时，让我也体会到了硬件控制程序的乐趣。

在小组合作中，我们也分工明确，以较高的效率完成了本次课程设计，提高了沟通能力，加强了团队意识。

除此之外，这次课程设计也让我复习了上学期学习的 VHDL 语法，特别是 VHDL 中的一些并行操作，这与平时接触的程序语言很不同，让我印象深刻，也让我困扰了不久。

总之，这是一次收获很大的课程设计，让我对计算机组成原理的理解得到了很大的提高。

梁维熙心得总结

通过这次深入的实验探索，我对硬件描述语言的理解不仅停留在表面，而是达到了一个全新的层次。特别是对于 VHDL 这一特定领域，我有了更为深刻的认识，它与我们日常编程时所熟悉的高级语言之间的差异，如今在我心中刻画得更为清晰。这种差异不仅仅体现在语法结构上，更在于其独特的设计思路和实现机制，这让我意识到硬件编程的复杂性和独特魅力。

在实际操作中，我遇到了一些在软件设计中鲜少遇到的挑战，比如出现不明来历的信号。这些信号可能源自于电路设计的细微疏漏或是硬件资源的分配不当。面对这些棘手问题，我不得不从多个角度进行思考，结合电路原理、信号流向以及逻辑分析，逐步排查并定位问题所在。

在这个过程中，我的调试技能得到了显著提升。更重要的是，我培养了一种系统性的思维模式，能够从整体架构出发，逐层分解问题，再由局部到整体，逐步验证解决方案的有效性。这种能力的提升，不仅对当前的实验项目大有裨益，也将成为我未来学习中的宝贵的财富。

龙文涛心得总结

本次实验的目地是利用 Quartus 软件为硬连线控制器进行编程，在这个过程中使用的芯片为 Altera EPM7128。除了深入运用 VHDL（VHSIC 硬件描述语言）这一业界广泛采用的硬件描述语言进行编程外，我还借此机会初步踏入了 Verilog 的世界，探索了其语法结构、编程范式以及与 VHDL 之间微妙的区别与互补优势。Verilog 的灵活性和直观性让我对硬件设计语言的多样性有了更深的理解。

在这次的课程设计中，我深刻体会到了事先详尽的资料收集与整理对于后续程序设计及调试工作的重要性。只有拥有了足够的知识储备，才能够更快速地解决在实验过程中遇到的问题，并增强自己灵活应变的能力。我在本次课程设计中进行的调试工作就应用了这一部分的经验，而这也进一步锻炼了我的问题解决能力和创新思维。

在小组合作方面，我们本次的小组合作过程非常愉快，而这得益于清晰明了的事前分工。同时，这也使得我们能够以较高的效率完成本次的课程设计。高效且和谐的合作氛围不仅加速了项目的进度，也让我们在相互学习中不断成长，增强了个人在团队环境中的适应性和贡献度。这次课程设计的经验无疑增强了我的沟通合作能力，让我更加适应了适用于团队的工作流程。

总的来说，我从这次的课程设计中收获了许多宝贵的经验。无论是扎实的专业知识、敏锐的问题解决能力，还是高效的沟通合作技巧，都是未来职业生涯中不可或缺的财富希望在未来的某一天，我能够学以致用，将它们应用于各种类型的工作之中。

廖轩毅心得总结

本次课程设计的时间较为紧迫，任务较为繁重。但我们依旧靠团队的力量和个人的不懈努力，成功地完成了设计任务。

在计算机组成原理理论学习中，硬连线控制器的设计就是一个复杂而关键的部分。而在此次课程设计中，我通过绘制指令流程图和设计控制操作时序等，逐步掌握了硬连线控制器的设计流程。

在实践过程中，我深刻体会到了设计测试程序的重要性，锻炼了我的逻辑思维和问题解决能力。通过测试程序，我更加明确了 VHDL 与高级语言之间的差异，这种差异不仅体现在语法结构上，更在于其独特的设计思路和实现机制，这种差异让我深刻认识到硬件编程的复杂性及其独有的吸引力。它不仅拓宽了我对硬件编程的认识，也为我未来的学术探索和技术研究提供了宝贵的视角和深刻的洞见。

在实际操作中，我也遇到了一些在软件设计中不曾遇到的挑战，比如 D7~D0 出现不明来历的 20H。通过逐步逆推找到问题了根源，但查阅诸多的资料却依旧无法解决。这让我意识到硬件设计中问题的隐蔽性和复杂性以及自身知识的局限性。我意识到不仅在面对未知问题时需要更加细致和耐心的排查过程，更要努力学习更多知识，以面对未来更多可能出现的复杂问题。

综上所述，本次课程设计虽然面临时间紧迫和任务繁重的挑战，但通过团队的协作和个人的不懈努力，我们不仅成功完成了设计任务，而且在这一过程中获得了宝贵的经验。我加深了对硬连线控制器的理解，体验了硬件设计的乐趣与挑战，并在实践中锻炼了逻辑思维和问题解决能力。

附件 2 小组调试日志

7 月 1 日调试日志

今日，我们团队在计组设计项目中取得了显著的进展。我们不仅完成了控制器的时序信号表，而且明确了设计方向，即实现在基础要求上添加 OR（逻辑或）、DEC（自减）、NAND（逻辑与非）、NOR（逻辑或非）等指令，并能够调整程序计数器（PC）起始值的硬连线控制器。

为了确保设计的有效性和可扩展性，我们在设计初期就进行了深入的讨论和规划。我们团队的每位成员都发挥了自己的专长，共同推动项目向前发展。在此基础上，我们还完成了对 TEC-8 机器的熟悉，这是我们实验室中使用的一种教学和实验设备，它将帮助我们更好地理解电路的工作原理。同时，我们也对 Quartus II 软件进行了深入的学习和实践，这是我们后续进行硬件描述语言（HDL）编程和电路仿真的重要工具。

此外，负责代码实现的同学已经开始复习和掌握 VHDL（VHSIC 硬件描述语言）。VHDL 的掌握对于我们设计和实现复杂的数字电路至关重要。通过重拾 VHDL，我们的团队成员已经做好了充分的开发准备。

随着项目的不断深入，我们将继续优化我们的设计方案，确保每一个细节都能够满足项目的需求。我们相信，通过团队的共同努力和协作，我们将能够克服任何技术挑战，最终实现一个高效、可靠的数字逻辑设计。

7 月 2 日调试日志

今日，我们团队的主要任务集中在代码编写上。负责代码实现的同学在深入熟悉 VHDL 的基础上，在实验室中成功完成了代码的编写工作，并进行了初步调试。尽管时间紧迫，我们的团队成员依旧展现出了高效性，确保代码编写工作的顺利完成。

此外，负责报告撰写的同学已经开始着手编写项目报告。报告将详细记录我们的工作流程、技术细节、遇到的问题及其解决方案，以及项目的最终成果。编写报告不仅是对项目工作的总结，也是对团队成员工作成果的认可。我们希望通过这份报告清晰地展示我们的工作成果。

在接下来的工作中，我们将继续优化代码，确保其在 TEC-8 机器上的调试能够顺利进行。同时，我们也将进一步完善报告内容，确保报告的准确性和完整性。我们相信，通过团队成员的共同努力，我们能够克服任何挑战，按时完成项目任务，并取得令人满意的成果。

7 月 3 日调试日志

为了确保我们的项目能够顺利通过验收，我们完成了两个关键的测试程序。这些测试程序不仅验证了我们设计的硬布线控制器的功能，而且确保了其在实际应用中的可靠性和稳定性。

在测试过程中，我们对控制器进行了一系列的功能测试。我们对控制器的基本功能进行了测试，以进一步优化设计。

报告撰写方面，我们的工作也取得了显著的进展。报告中详细记录了我们的硬件环境、课题描述、需求分析、概要设计、团队分工、设计详解。我们还特别强调了在设计和调试过程中遇到的挑战以及我们的解决方案，这不仅展示了我们团队的创新能力和问题解决能力，也为未来的工作提供参考。

在接下来的工作中，我们将继续完善报告的剩余部分，并准备最终的验收演示。

7 月 4 日调试日志

今日，我们的项目进入了尾声。

所有成员在实验室进行了最后的调试和流程分析，为明天的验收做准备，撰写报告的同学也优化完成了本次报告。

我们相信，通过我们的努力和专业知识，我们的项目将能够顺利通过验收。