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">计组课设</td><td>学 院</td><td>AI</td><td>指导教师</td><td>张杰</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>2020219108</td><td></td><td colspan="2">2020212271</td><td>唐子钰</td><td colspan="2"></td></tr>
<tr><td>2020219108</td><td></td><td colspan="2">2020212296</td><td>刘梓航</td><td colspan="2"></td></tr>
<tr><td>2020219108</td><td></td><td colspan="2">2020212299</td><td>马天驰</td><td colspan="2"></td></tr>
<tr><td>2020219111</td><td></td><td colspan="2">2020212261</td><td>黄安</td><td colspan="2"></td></tr>
<tr><td>课程设计内容</td><td colspan="6">简要介绍课程设计的主要内容，包括课程设计教学目的、基本内容、实验方法和团队分工等
主要内容：基于 TEC-8 和硬件语言完成顺序、流水和中断控制器的设计、组装、调试和测试。
目的：①融会贯通计组课程各章节的内容；②掌握熟悉硬连线、流水、中断原理；③学习掌握硬件描述语言的逻辑设计和调试。
方法：基于 quartus 和 vhdl 语言进行逻辑设计，然后编译下载后在 TEC-8 上组装、调试。
分工：见正文第三节。</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

一、实验环境

- PC 主机：使用 Quartus Ⅱ9.1，完成 VHDL 代码的逻辑设计、编程、编译和下载
- 元器件：Altera MAX7000 系列 CPLD 芯片 ：EPM7128
- 调试器件：数字示波器，万用表、逻辑测试笔

二、题目分析

1 任务目标

已完成
题目一：基于 Altera CPM7128 的硬连线控制器设计按照给定数据格式、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机
基本功能：根据设计方案，在 TEC-8 上进行组装、调试运行
附加功能：
a.在原指令基础上要求扩指至少三条
b.修改 PC 指针功能（任意指针）
题目二：完成流水硬连线控制器的设计根据设计方案，在 TEC-8 上进行组装、调试运行
题目三：基于 TEC-8 系统完成中断功能硬连线控制器设计。

2 实验原理-硬连线原理

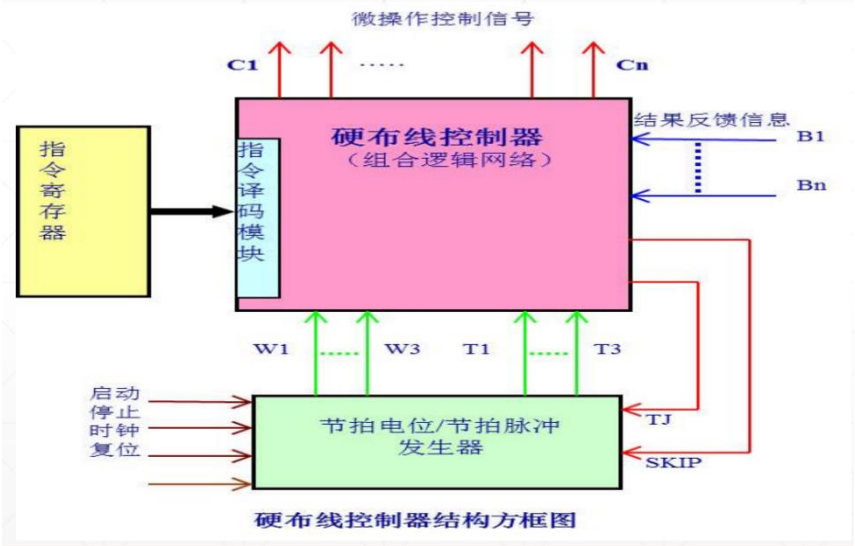
硬布线控制器是将控制部件做成产生专门固定时序控制信号的逻辑电路，产生各种控制信号，因而又称为组合逻辑控制器。这种逻辑电路以使用最少元件和取得最高操作速度为设计目标，因为该逻辑电路由门电路和触发器构成的复杂树型网络，所以称为硬布线控制器。

硬布线控制器的基本原理，归纳起来可叙述为：某一微操作控制信号 C 是指令操作码译码器输出 Im 、时序信号(节拍电位 Mi ，节拍脉冲 Tk)和状态条件信号 Bj 的逻辑函数，其数学描述为：

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

硬布线控制器相对于微程序控制器优点是：性能方面硬布线控制器由于无需从控制存储器中读取指令，延迟主要来自于电路延时，因此相对于微程序控制器有较大的提升。

硬布线控制器相对于微程序控制器缺点是：硬布线控制器的功能是通过组合逻辑门电路实现，设计实现复杂，且如需更改指令，必须重新设计电路。不适合应用于复杂指令系统中。



硬布线控制器结构方框图

3 前置知识

3.1 实验仪器介绍

▲TEC-8 实验系统

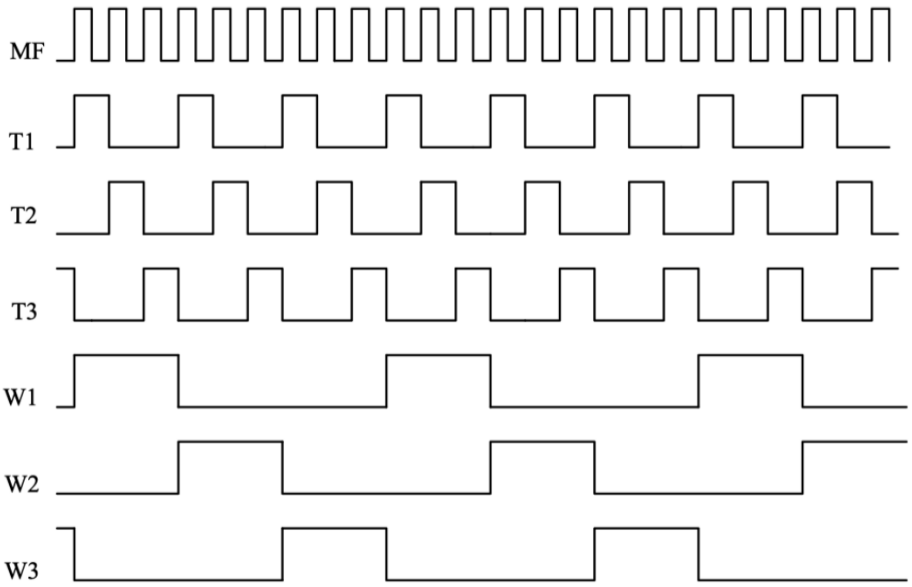
TEC-8 实验系统有如下特点：
1) 模型计算机采用 8 位字长;
2) 指令系统采用 4 位操作码;
3) 采用双端口存储器作为主存，实现数据总线和指令总线双总线体制，实现指令流水功能，体现出现代 CPU 设计思想;
4) 控制器采用微程序控制器、硬连线控制器和独立 3 种类型;

▲逻辑测试笔

TEC-8 实验系统上配置的逻辑测试笔在测试信号的电平时，红灯亮表示高电平，绿灯亮表示低电平，红灯和绿灯都不亮 表示高阻态。在测试脉冲个数时，首先按一次 Reset 按钮，使 2 个黄灯 D1、D0 灭，处于测 试初始状态。TEC-8 实验台上的逻辑测试笔最多能够测试 3 个连续脉冲。

红绿	测试结果	D	测试结果
00	高阻态	00	没有脉冲
10	高电平	01	1 个脉冲
01	低脉冲	10	2 个脉冲
		11	3 个脉冲

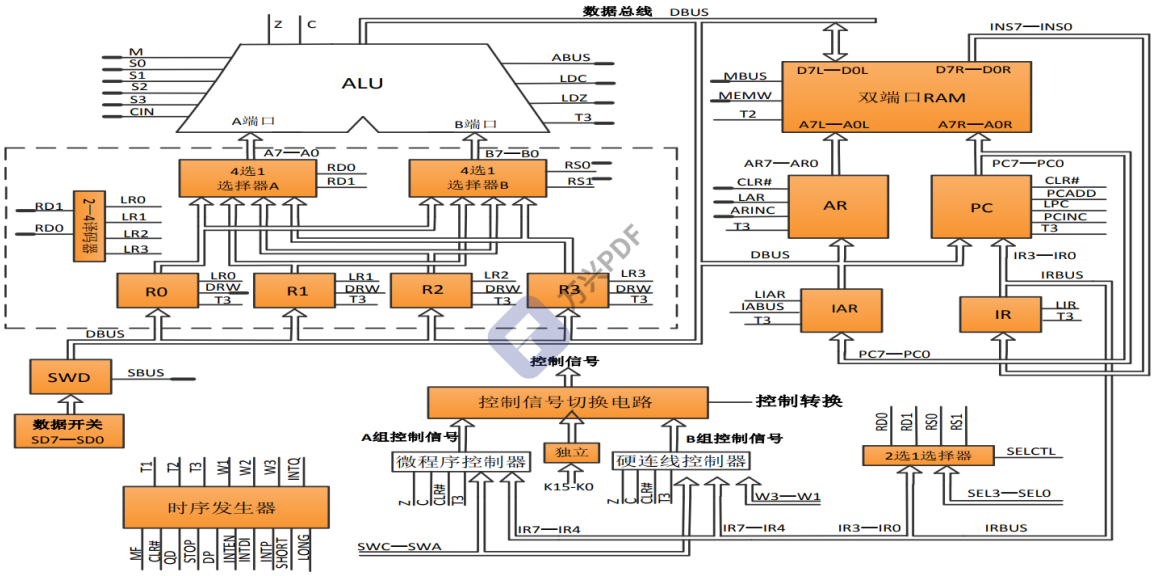
3.2 TEC-8 时序



TEC-8 时序系统，主时钟脉冲 MF 的频率是 1MHZ，T1、T2、T2 分别是三个节拍脉冲，频率均为 0.5MHZ；W1、W2、W3 分别为三个节拍电位，每个节拍电位中包括三个节拍脉冲，在 TEC-8 中执行一条**微指令**至少需要一个节拍电位，且 TEC-8 的指令系统采用不定长机器周期，指令周期在 1-3 个节拍电位不等。

3.3 TEC-8 模型机

TEC-8 的主要组成模块：时序发生器、算逻运 算单元 ALU、双端口寄存器组 7064、数据开关 SD、双端口存储器 7132、组合逻辑控制器、 微程序控制器、若干寄存器和若干选择器。



3.4 EPM7128 引脚

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			

注意，中断功能控制器中用到的 PULSE 和 MF 信号的引脚不直接与 TEC-8 实验台数据通路相连接，需要外接缆线才能将 PULSE 和 MF 的信号源与 EPM 引脚对接。本次课设对应引脚分别为 PULSE-61，MF-55。

3.5 各信号说明

信号	类型	说明
SWC、SWB、SWA	IN	选择不同的控制台模式
CLR	IN	复位,低电平有效
C	IN	进位标志
Z	IN	结果为零标志
IRH	IN	IRH7~IRH4，指令操作码
T3	IN	T3节拍脉冲
W3~W1	IN	节拍电位
M	OUT	算数/逻辑运算
S(S3~S0)	OUT	ALU选择方式
CIN	OUT	ALU低位进位输入
SEL3~SEL2	OUT	选择送 ALU 的 A 端口的寄存器
SEL1~SEL0	OUT	选择送 ALU 的 B 端口的寄存器
SELCTL	OUT	为1时处于控制台操作，为0时处于运行程序状态
DRW	OUT	为1时在T3上升沿将DBUS上的数据写入SEL3SEL2选中的寄存器
ABUS	OUT	为1时运算器结果送数据总线DBUS
SBUS	OUT	为1时将开关数据送数据总线DBUS
LIR	OUT	为1时将DBUS上的指令写入AR
MBUS	OUT	为1时将双端口RAM左端口数据送到DBUS
MEMW	OUT	为1时在T2为1期间将DBUS写入AR指定的存储器单元，为0时读存储器
LAR	OUT	为1时在T3的上升沿将DBUS的地址打入AR
LPC	OUT	为1时在T3的上升沿将DBUS的数据写入PC
LDC	OUT	为1时T3的上升沿保存进位
LDZ	OUT	为1时T3的上升沿保存结果为0标志
ARINC	OUT	为1时在T3的上升沿AR加1
PCINC	OUT	为1时在T3的上升沿PC加1
PCADD	OUT	PC加上偏移量
LONG	OUT	标志指令还需要第三个节拍电位W3
SHORT	OUT	标志指令不需要第二个节拍电位W2
STOP	OUT	为1时时序发生器在T3结束后停止
MF	IN	主时钟周期
PULSE	IN	实验台停止信号

三、团队分工

刘梓航	流水硬连线控制器代码设计和调试，顺序控制器流程图绘制，测试程序调试，顺序设计实验报告编写
马天驰	顺序硬连线控制器代码设计和调试，测试程序调试，实验报告编写，整合部分调试日志
唐子钰	主要负责控制器模型设计、中断流程图设计、译码表绘制、顺序中断源代码编写、组装调试、设备测试；负责文档的中断设计部分、整合以及一部分调试日志。
黄安	流水硬连线控制器代码设计和调试，流水控制器流程图绘制，测试程序调试，流水实验报告编写

四、设计详解

1 功能说明

CPU 要完成的功能分为两类，一类是控制台操作，即根据 SWCBA 的不同值选择不同的控制方式，即读写存储器和寄存器；二类是机器指令操作，即根据存储器中的 8 位机器指令码来执行不同的操作。后面在具体的设计中会结合小组成员绘制的控制器流程图说明各控制模式功能以及指令是如何实现的。

SWC-SWA	100	011	000	010	001
指令	读寄存器	写寄存器	取指令	读存储器	写存储器

名称	助记符	功能	指令格式		
			IR7~IR4	IR3~IR2	IR1~IR0
空指令	NOP	无	0000	XX[1]	XX
加法	ADD Rd, Rs	Rd ← Rd[2]+ Rs[3]	0001	Rd	Rs
减法	SUB Rd, Rs	Rd ← Rd - Rs	0010	Rd	Rs
逻辑与	AND Rd, Rs	Rd ← Rd ∧ 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 ← @[4]+ offset[5]	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
逻辑或	OR Rd, Rs	Rd ← Rd ∨ Rs	1011	Rd	Rs
比较	CMP Rd, Rs	Rd - Rs	1100	Rd	Rs
移动值	MOV Rd, Rs	Rd ← Rs	1101	Rd	Rs
停机	STOP	暂停运行	1110	XX	XX

[1] XX 代表随意值。

[2] Rs 代表源寄存器号。

[3] Rd 代表目的寄存器号。

[4] @ 代表当前 PC 的值。

[5] offset 是一个 4 位的补码有符号数。

2 硬件语言实现方式

我们有两种不同的方式来编写硬连线控制器的 VHDL 硬件语言。一种是行为描述方式，一种是数据流描述方式。前者即按照流程图的步骤根据判断条件依次进行 CASE、when 或者 if 分支完成整个程序；而后者则是所有的信号都在 ARCHITECTURE 中并行赋值，即使用逻辑表达式一次性实现所有信号。前者符合正常逻辑思考，简单清晰，只根据流程图便可实现程序，但代码较长；后者逻辑表达式较为复杂，代码短，另需译码表且不易调试。

代码 1 数据流描述方式

```
-- 操作模式
WRITE_REG<= '1' when SW= "100" else '0';
READ_REG <= '1' when SW= "011" else '0';
INS_FETCH<= '1' when SW= "000" else '0';
READ_MEM <= '1' when SW= "010" else '0';
WRITE_MEM<= '1' when SW= "001" else '0';
-- 操作码
ADD <= '1' when IR = "0001" and INS_FETCH = '1' and ST0 = '1' else '0';
SUB <= '1' when IR = "0010" and INS_FETCH = '1' and ST0 = '1' else '0';
-- 控制信号逻辑表达式合成
SBUS <= ((WRITE_REG or (READ_MEM and not ST0) or WRITE_MEM or (INS_FETCH and not ST0)) and W(1)) or (WRITE_REG and W(2));
DRW <= (WRITE_REG and (W(1) or W(2))) or ((ADD or SUB or AND_I or INC or OR_I or MOV) and W(2)) or (LD and W(3));
```

代码 2 行为描述方式

```
process(SWCBA,IRH,C,Z,W1,W2,W3,ST0)
    begin
    case SWCBA is
        when "100"=>--写寄存器
            STOP<=W1 or W2;
            SBUS<=W1 or W2;
            SELCTL<=W1 or W2;
            DRW<=W1 or W2;
            SST0<=(not ST0) and W2;
            SEL3<=ST0;
            SEL2<=W2;
            SEL1<=((not ST0)and W1)or(ST0 and W2);
            SEL0<=W1;
        when "011"=>--读寄存器

        when "010"=>--读存储器

        when "001"=>--写存储器

```

```
        when "000"=>--取指
                                    --第一阶段是输入首地址
            SBUS<=(not ST0) and W1;
            LPC<=(not ST0) and W1;
            SST0<=(not ST0) and W1;
            SHORT<=(not ST0) and W1;
            STOP<=(not ST0);--不能让 STOP<=(not ST0) and W1：W1 刚开始为 0，不能使按了 CLR 后停机。
            --下面是第二阶段，第一阶段只有 W1，第二阶段 W1 是 load 指令且 PC 自加。W2 后则执行指令。
            LIR<=ST0 and W1;
            PCINC<=ST0 and W1;
            case IRH is
                when "0001"=>--add

                when "0010"=>--sub
                 ...................
                when others=>NULL;
            end case;
        when others =>NULL;
    end case;
end process;
```

3 顺序硬连线控制器设计

3.1 顺序指令周期原理说明

根据指令操作码 SWC、SWB、SWA 的取值，我们用不同操作码表示不同的操作。对应关系如下表所示：

SWC-SWA	100	011	000	010	001
指令	读寄存器	写寄存器	取指令	读存储器	写存储器

由于多数的程序操作需要两拍节拍信号 W1、W2 即可执行完毕，而有些信号则需要一拍、三拍或四拍节拍信号，例如写寄存器指令需要连续写入四个寄存器而需要四拍节拍电位信号，而写存储器、读存储器指令因为需要持续进行读出/写入，则需要持续进入 W1 结拍，而不能进入 W2 结拍，用 SHORT 和 LONG 两种信号，指示当前拍的下一个节拍。SHORT 信号可以在 W1 结束后不进入 W2，而继续进行 W1；LONG 信号则可以在 W2 信号结束后进入 W3 而非返回 W1。

此外，对于读寄存器操作，需要四拍才可以执行完成，因此小组将该操作分为两个基本操作单元，即重复两次 W1、W2，以简化程序设计。为区分两个操作单元，小组成员设计了 FLAG 信号，即 ST0，用于指示第及阶段。ST0=0 表示第一阶段当前在第一个 W1、W2，ST0=1 则表示第二阶段当前在第二个 W1、W2。

对于读/写存储器操作，分为两个部分。第一个部分操作用于选定存储器地址，第二部分操作用于对选中地址进行读写，因此小组成员利用 ST0 来指示操作执行情况，ST0=0 表示选定地址，ST0=1 表示在进行读/写操作。根据要求，读写操作部分需要一直处于 W1 节拍，而不进入 W2，因此在 ST0=0 的操作进行完时，将 SHORT 信号置 1，不进入 W2，而是重新进入 W1。

对于取指令操作，同样分为两个阶段，第一个阶段用于选定初始指令地址，即修改任意 PC 指针；第二个阶段用于从输入地址开始，依次执行指令直到 STP。同样可以利用 ST0 信号来表示两个不同的阶段，即 ST0=0 表示输入初始指令地址，ST0=1 表示取指令并执行。在程序执行完毕之前将一直处于 ST0=1 状态。

顺序处理器设计

小组成员在设计硬件描述程序时采用了行为描述的风格，即基于指令流程图，根据流程图的判断条件和信号逻辑关系顺序写出程序并对相应信号进行赋值。代码的 ARCHITECTURE 部分分为两个进程，分别是根据 CLR、T3 时钟信号和 SST0 信号对 ST0 信号进行赋值的信号处理进程，以及根据输入指令信号选择不同指令进行执行的进程。在代码 Architecture 中，我们设计了三个信号（Signal），如下所示：

Signal	类型	功能
ST0	std_logic	FLAG 信号，=0 表示第一阶段，=1 表示第二阶段
SST0	std_logic	动态改变的信号，指示下一拍是第几个阶段，=1 表示下一拍为第二阶段，=0 表示下一拍为第一阶段，通过 SST0 来改变 ST0
SWCBA	std_logic_vector(2 downto 0)	操作码字段，控制执行不同操作

以下分别讲解两个进程的设计思路。

信号处理进程

在 T3 信号下降沿处，即一个节拍信号结束前，应通过 SST0 信号当前取值来判断接下来是否应该进入 W2 阶段。此外对于写寄存器操作，需要进行特殊判断防止其在第二阶段发生循环。因此小组成员的顺序控制器信号处理进程代码设计如下：

```
--处理 ST0 信号
process(CLR,T3)
begin
    if(T3'event and T3='0')THEN --下降沿
        if(SST0='1')THEN--检测指示下一阶段的信号
            ST0<='1';
        end if;
        if(W2='1' and SWCBA="100" and ST0='1')then--写寄存器的特判
            ST0<='0';
        end if;
    end if;
    if(CLR='0')then--重置检测
```

```
        ST0<='0';
    end if;
end process;
```

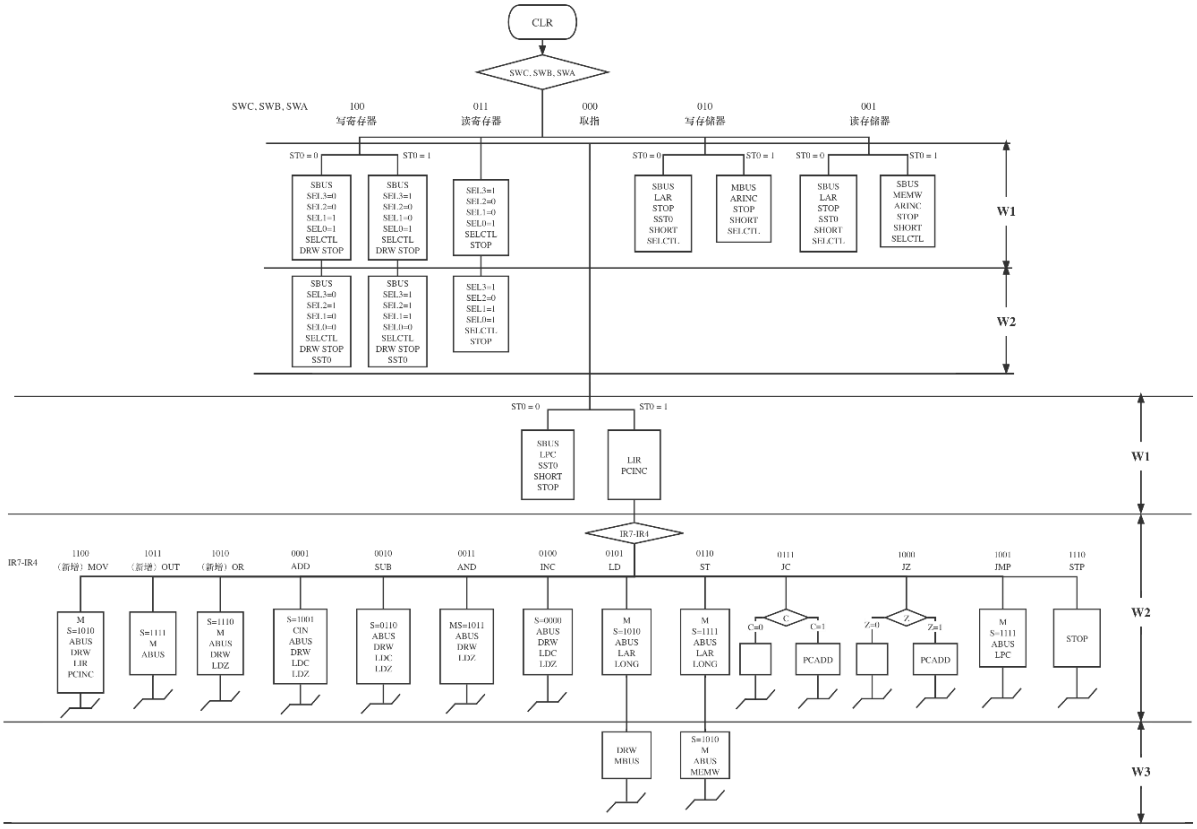
指令执行进程

针对 SWCBA 的不同取值，程序将进入对应的不同操作。在行为描述风格代码中，即可以根据 SWCBA 的取值使用 CASE-WHEN 语句进行分类设计。同时在 SWCBA 取 000，即取指令执行的操作中，也会使用 IR7-IR4 的四位指令码对具体指令进行标识，这里同样可以对 IR 使用 CASE-WHEN 语句。伪代码如下所示：

```
process(SWCBA,IRH,C,Z,W1,W2,W3,ST0)
    begin
                --初始化各个信号
                CASE SWCBA IS--对 SWCBA 取值进行分类
                        WHEN "011" IS --写寄存器
                                --指令对应的代码
                        WHEN "100" IS --读寄存器
                                --指令对应的代码
                        WHEN "001" IS --写存储器
                                --指令对应的代码
                        WHEN "010" IS --读存储器
                                --指令对应的代码
                        WHEN "000" IS --取指令
                                --第一阶段：输入指令地址
                                [对应赋值情况]
                                --第二阶段：顺序执行指令
                                [对应赋值情况]
                                CASE IR IS--对 IR7-IR4 取值进行分类
                                        WHEN "0001" IS
                                                --指令对应的代码
                                        .....省略
                                        WHEN OTHERS=>NULL
                                END CASE
                        WHEN OTHERS=>NULL
                END CASE
        END PROCESS
```

3.2 顺序指令周期流程图

周期流程图如下所示：



3.3 基础功能说明

3.3.1 读寄存器

按一次 CLR 进行清零（ST0=0）。调节操作码 SWC、SWB、SWA 分别为 011，将 DP 置 1，为单拍操作。读寄存器一共需要两个机器周期，分别显示 S0S1 和 S2S3 的值。在操作台上按一次 QD，SEL3-SEL0 显示为 0001，即 A7-A0，B7-B0 分别显示 R0，R1 当前的值。再按一次 QD，SEL3-SEL0 显示为 1011，即 A7-A0，B7-B0 分别显示 R2，R3 当前的值。

3.3.2 写寄存器

按一次 CLR 进行清零。调节操作码 SWC、SWB、SWA 分别为 100，将 DP 置 1，为单拍操作。在操作台上按一次 QD，并在数据输入端输入 8 位二进制码。按一次 QD，将当前数据存入 R0 寄存器内；在数据段再次输入 8 位数据，按一次 QD，将当前数据存入 R1 寄存器内。仿照以上步骤，将数据存入 R2，R3 内，每按一次 QD，进行一次存储操作。

3.3.3 读存储器

按一次 CLR 进行清零。调节操作码 SWC、SWB、SWA 分别为 010，将 DP 置 1，为单拍操作。读存储器共有两个阶段，第一个阶段用于确定初始地址，第二个阶段则循环对连续的存储单元进行读取。按一次 QD，进入地址选择阶段。此时 LAR 灯亮，在数据输入

端输入选定的存储器 8 位二进制地址，按一次 QD，执行 LAR，选中地址单元。此后每按一次 QD，读出一次当前存储器内所存的数据，并将 AR（指示选中的存储单元）加 1，选中下一个存储单元。数据在 TEC-8 控制台 DBUS 指示灯上显示。

3.3.4 写存储器

按一次 CLR 进行清零。调节操作码 SWC、SWB、SWA 分别为 001，将 DP 置 1，为单拍操作。写存储器共有两个阶段，第一个阶段用于确定初始地址，第二个阶段则循环对连续的存储单元进行写入。按一次 QD，进入地址选择阶段。此时 LAR 灯亮，在数据输入端输入选定的存储器 8 位二进制地址，按一次 QD，执行 LAR，选中地址单元。此后先在数据输入端输入数据，再按 QD，将会把输入的数据存入选中的存储单元中，并将 AR（指示选中的存储单元）加 1，选中下一个存储单元。

3.3.5 顺序执行指令

使用写存储器、写寄存器操作将指令码和数据预先存入 TEC-8 系统中，按一次 CLR 进行清零。调节操作码 SWC、SWB、SWA 分别为 000。按一次 QD，在数据输入端存入选中的指令地址，再按 QD 后，系统将从选中存储单元指令开始，一直顺序执行，直到 STOP 指令，执行结束。

3.4 扩充功能及指令说明

- OR

  与 AND 指令类似，IR7-IR4 为指令码，IR3-IR0 用于选寄存器。返回选中两个寄存器中数据按位或运算后的值

- MOV

  即寄存器间赋值，Rd<-Rs。

- OUT

  输出所选寄存器中的数值。

指令格式如下所示：

名称	助记符	功能	指令格式		
			IR7-IR4	IR3-IR2	IR1-IR0
逻辑或	OR Rd, Rs	Rd<-Rd∨Rs	1101	Rd	Rs
移动	MOV Rd Rs	Rd<-Rs	1100	Rd	Rs
输出	OUT [Rd]	DBUS<-Rd	1010	Rd	XX

3.5 译码表

	ADD	SUB	AND	INC	LD	ST	JC	JZ	JMP	OUT	IRET	MOV	OR	STOP	写寄存器	读寄存器	写存储器	读存储器	取指(仅W1)
DRW	W2	W2	W2	W2	W3							W2	W2		W1+W2				
PCINC												W2						ST0*W1	
LPC									W2									~ST0*W1	
LAR					W2	W2										~ST0*W1	~ST0*W1		
PCADD							C*W2	Z*W2											
ARINC																	ST0*W1	ST0*W1	
SELCTL																W1+W2	W1	W1	
MEMW						W3									W1+W2		ST0*W1		
LIR												W2							ST0*W1
LDZ	W2	W2	W2	W2									W2						
LDC	W2	W2	W2	W2															
CIN	W2																		
S	1001*W2	0110*W2	1011*W2	0000*W2	1010*W2	1111*W2+ 1010*W3 W2+W3			1111*W2	1111*W2		1010*W2	1110*W2						
M			W2		W2	W2+W3			W2	W2		W2	W2						
ABUS	W2	W2	W2	W2	W2	W2+W3			W2	W2		W2	W2						
SBUS					W3										W1+W2		W1	~ST0*W1	~ST0*W1
MBUS																		ST0*W1	
SHORT																	1	1	~ST0*W1
LONG					W2	W2													
SEL3															ST0	W2			
SEL2															W2 (~ST0*W1) +(ST0*W2)	0			
SEL1																W2			
SEL0															W1	W1+W2			
STOP														W2	W1+W2	W1+W2	W1	W1	~ST0*W1
SST0															~ST0*W2		~ST0*W1	~ST0*W1	~ST0*W1
SST1																			

3.6 测试程序

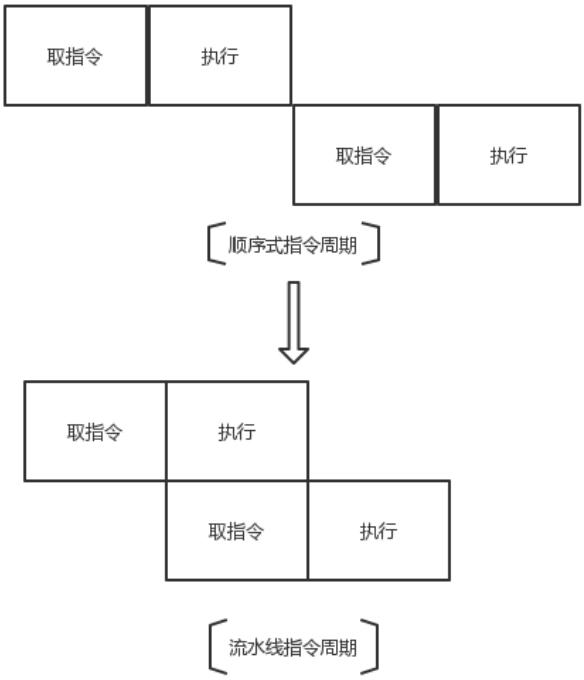
初值：R0、R1、R2、R3=00、00、60、FD；
[60H]、[61H]、[62H]、[80H]、[FE]、[FF]=67、80、FD、60、03、03
终值：R0-R3：80、83、60、FD；[81H]、[82H]、[83H]=86、04、83

测试指令及机器码			
地址	程序指令	2 进制机器码	16 进制机器码
00H	LD R0,[R2]	01010010	52
01H	INC R2	01001000	48
02H	LD R1,[R2]	01010110	56
03H	ADD R0,R1	00010001	11
04H	JC 06H	01110001	71
05H	AND R1,R0	00110100	34
06H	SUB R0,R2	00100010	22
07H	INC R1	01000100	44
08H	STA R0,[R1]	01100001	64
09H	INC R3	01001100	4C
0AH	JZ 0DH	10000010	82
0BH	LD R2,[R3]	01011011	5B
0CH	JMP [R2]	10011000	98
0DH	INC R3	01001100	4C
0EH	INC R3	01001100	4C
0FH	SUB R0,R2	00100010	22
10H	LD R2,[R0]	01011000	58
11H	ADD R3,R2	00011110	1E
12H	LD R3,[R3]	01011111	5F
13H	OUT R0	10100000	B0
14H	STP	11100000	E0

4 流水线硬布线控制器实现

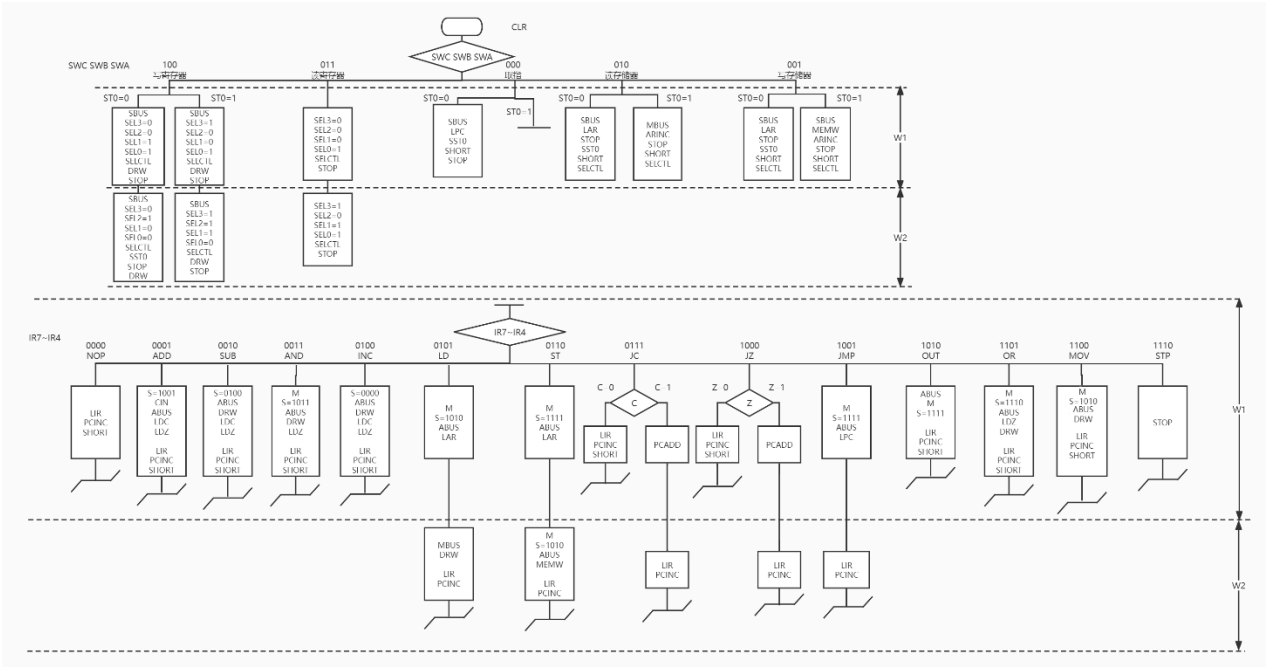
4.1 流水线指令周期原理说明

流水线指令周期与顺序式不同的是，顺序式的取指均在一独立节拍 W1 中完成，即执行每条指令时的第一个节拍电位 W1 都是用来执行 LIR、PCINC 读取指令，而流水线指令周期是将非流水线中单独为一个节拍的取指令 LIR、PCINC 转换到每条指令的执行节拍里，如下图所示：



从效率上看，对于连续执行大量相同的需要两个节拍的指令，流水线的总执行时间是顺序式的 50%，而流水线的层级越高，流水的效率相比顺序式就越高。

4.2 流水线指令周期流程图



4.3 译码表

	NOP	ADD	SUB	AND	INC	LD	ST	JC	JZ	JMP	OUT	IRET	MOV	OR	STOP	写寄存器	读寄存器	写存储器	读存储器	取指(仅W1)
DRW		ST0*W1	ST0*W1	ST0*W1	ST0*W1	W2							ST0*W1	ST0*W1		W1+W2				
PCINC	ST0*W1	ST0*W1	ST0*W1	ST0*W1	ST0*W1	W2	W2	ST0*W1*~CF+W2	ST0*W1*~Z+W2	W2	ST0*W1		ST0*W1	ST0*W1						
LPC										ST0*W1										~ST0*W1
LAR						ST0*W1													~ST0*W1	
PCADD							ST0*W1	ST0*C*W1	ST0*Z*W1											
ARINC																	W1+W2	ST0*W1	ST0*W1	
SELCTL																W1+W2		W1	W1	
MEMW							W2										ST0*W1	ST0*W1		
LIR	ST0*W1	ST0*W1	ST0*W1	ST0*W1	ST0*W1	ST0*W2	ST0*W2	ST0*W2	ST0*W2	ST0*W2	ST0*W1		ST0*W1	ST0*W1						
LDZ		ST0*W1	ST0*W1	ST0*W1	ST0*W1									ST0*W1						
LDC		ST0*W1	ST0*W1		ST0*W1															
CIN		ST0*W1																		
S		1001*ST0*W1	0110*ST0*W1	1011*ST0*W1	0000*ST0*W1	1010*ST0*W1	1111*ST0*W1+1010*W2						1010*ST0*W1	1110*ST0*W1						
M				ST0*W1		ST0*W1	ST0*W1+W2			1111*ST0*W1	ST0*W1		ST0*W1	ST0*W1						
ABUS		ST0*W1	ST0*W1	ST0*W1	ST0*W1	ST0*W1	ST0*W1+W2			ST0*W1	ST0*W1		ST0*W1	ST0*W1						
SBUS						W2										W1+W2		W1	~ST0*W1	~ST0*W1
MBUS																			ST0*W1	
SHORT	ST0*W1	ST0*W1	ST0*W1	ST0*W1	ST0*W1			ST0*W1*~CF	ST0*W1*~Z		ST0*W1		ST0*W1	ST0*W1				1	1	~ST0*W1
LONG																				
SEL3																ST0	W2			
SEL2																W2	0			
SEL1															(~ST0*W1)+(ST0*W2)	W1	W2			
SEL0																W1	W1+W2			
STOP															W2	W1+W2	W1+W2	W1	W1	~ST0*W1
SST0																~ST0*W2		~ST0*W1	~ST0*W1	~ST0*W1
SST1																				

4.4 基础功能说明

4.4.1 读写寄存

流水线的读、写寄存器和读、写存储器功能实现模式与顺序式的一致，详细见顺序式硬布线控制器基础功能说明

4.4.2 取指

在 ST0=1 时（初始 ST0=0，用于 PC 指针修改，具体见（4）），进入第一个 W1 节拍，初始指令执行，且在该指令执行的最后一节拍同时进行下一条指令的取指操作。特别的，在 TEC-8 实验系统中，由于按下 CLR 后，指令寄存器 IR 会被复位为 00H，所以程序执行的初始指令操作码均为 00H,因此我们将 00H 规定为空指令（NOP），执行该指令时，LIR、PCINC 信号打开，实现对当前 PC 指针对应的指令写入 IR,及 PC 指针加一。注意到，顺序式的取指进入程序操作是初始时利用第一个 W1 节拍，将 LIR、PCINC 信号打开，完成初始进入程序，而流水线的取指进入程序操作，是利用 NOP 指令，实现初始进入程序。

4.5 任意修改 PC 指针

在取指操作中，利用 ST0 信号的两个状态，分别完成 PC 指针修改和取指功能。初始 ST0=0,此时进入 PC 指针修改阶段，为一个 W1 节拍，LPC、SBUS 信号打开，实现对 PC 指针的修改，完成对程序首地址的输入，同时 SST0 置‘1’，即 ST0 信号在该 W1 节拍结束后 ST0=1,进入取指功能。注意到，该阶段没有 LIR 信号，因为我们可以利用 NOP 指令，完成初始进入程序操作。代码如下：

```
1.  WHEN "000" => --取值
2.      IF(ST0 ='0')THEN--任意修改 PC 指针，在开始的一个 W1 内
3.          LPC<=W1;
4.          SBUS<=W1;
5.          SST0<=W1;
6.          SHORT<=W1;
7.          STOP<='1';
```

5 中断硬连线控制器设计

5.1 目标功能

任务目标是基于 TEC-8 完成中断功能的硬连线控制器设计。

就结果而言，其应支持如下步骤：

①CPU 在正常状态下按下 pulse 按钮，发出中断请求；

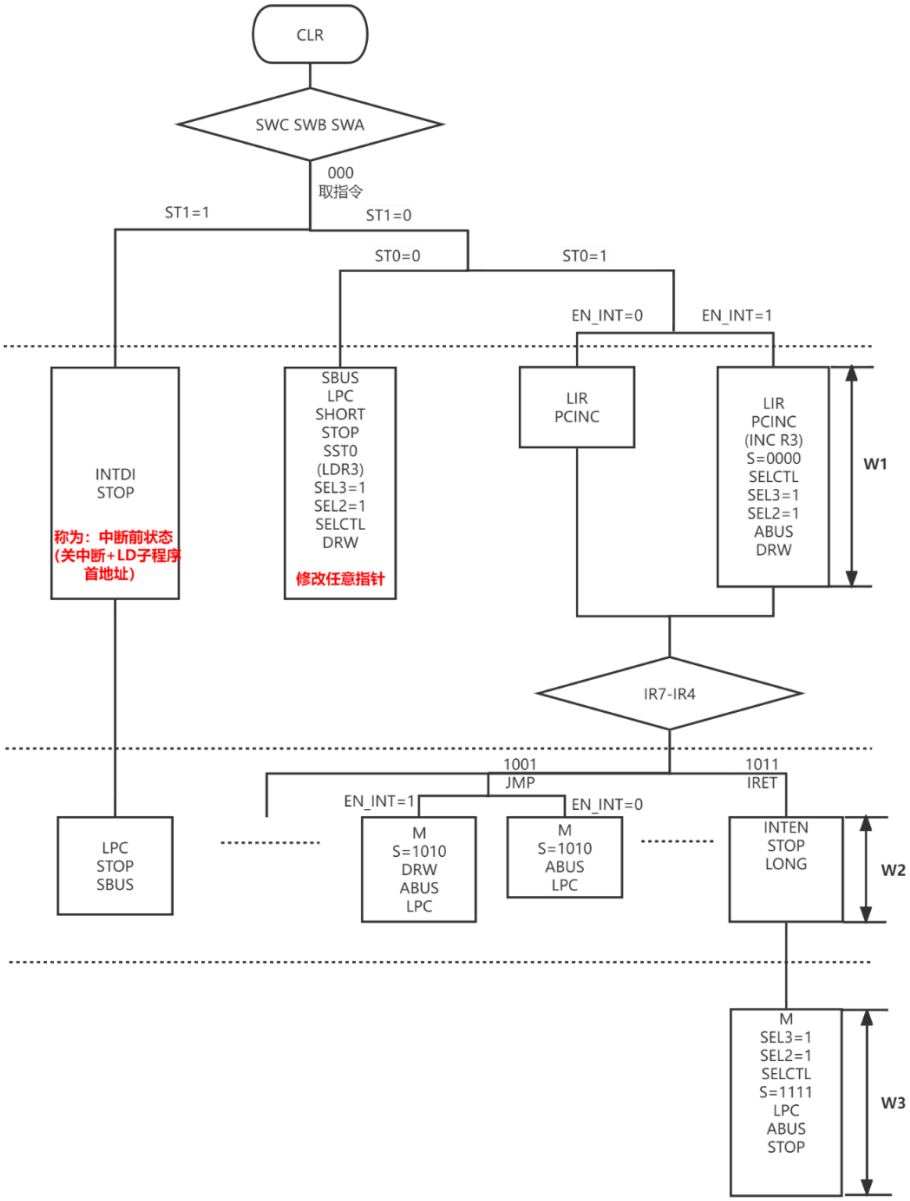
②CPU 允许仲昂的，中断响应，关中断，保存断点，获取中断服务程序入口地址；

③进入中断服务子程序，执行子程序指令；
④开中断，返回主程序断点；

就过程而言，本次实验较为复杂。原因是 TEC-8 实验台没有中断地址寄存器即 IAR 的引脚，这也就导致了我们没法使用 IAR 来临时保存断点即 PC 值。同时又由于 PC 寄存器中的数据没有办法输出到总线上，这也就导致了我们没法简单地用一个通用寄存器 Rd 来取代 IAR。经过老师的提醒，我们只能绕远路，使用一个通用寄存器如 R3 同步跟踪 PC 的值，即 PC 自增，我们 INC R3；PC JMP，我们也同步更新 R3。这样就能成功取代 IAR 寄存器的功能。当需要中断时，我们直接转到中断服务子程序，中断结束后，用 R3 的值替换 PC 然后继续同步跟踪即可。

5.2 中断功能流程概述

本中断功能控制器的具体设计思想参考了《TEC-8 组成原理指导书》中微程序中断实验以及一些其他资料。



我们首先将 CPU 控制器执行指令的阶段分为两种状态，一是目前 CPU 处于正常执行指令的状态简称为正常状态，即 EN_INT=1（允许中断）；二是 CPU 处于中断服务子程序执行指令的状态简称中断状态，即 EN_INT=0（禁止中断）。分状态的原因是正常状态下，我们需要同步更新 R3 以后续返回断点（见上流程图）；而在中断状态下则不需要跟踪 PC。

其次，在修改任意指针时除了 LPC 我们需要添加命令 LD R3，即将 SBUS 同时也载入到 R3 寄存器中。注意：在进入中断服务子程序后是不需要修改 PC 指针的，因此我们先判断 ST0，若为 0 那么一定是 CPU 处于正常状态下且下一步是需要修改任意 PC 指针。=1 才需要判断是否是中断状态 or 正常状态。

然后我们进入中断服务子程序前还有一个重要的步骤，该步骤中需要完成的指令有：关中断+LD 中断服务子程序首地址。因此我将过程利用 ST1=1 单独地标识了出来，且我定义该过程名为**中断前状态**。ST1=0 时即**非中断前状态。**关中断后 EN_INT 自动变为 0，后续执行指令时将自动辨识为中断状态。

最后是进入中断的方法，也就是如何进入中断前状态即使得 ST1=1。条件是需要我们在 CPU 正常状态下执行某条指令的最后一拍时，同时按下 PULSE 信号和 QD，此时程序将视为当前指令（执行完）发出了中断请求，下一拍将进入中断前状态，实际上具体体现到代码中是我设计了一个 SST1 信号来同步改变 ST1。注意：为了演示、调试方便，此处我有意设计为单拍执行且必须在最后一拍按下 PULSE 和 QD 后才进入中断前状态。如果是实际情况模型机执行指令速度过快，没办法人工 find 一个便于演示的断点（理论上如果测试程序是一个 loop 可以实现）；实际上如果是连续执行，由于人手按下 PULSE 信号滞留的时间一定远大于 CPU 模型机执行命令的时间，因此用 ST1 信号判断是否进入中断前状态对连续运行也是有效的。

中断控制器默认情况下 EN_INT=1、ST0=0、ST1=0（使用 CLR 将重置为默认），即我们重置后将自动开中断，不像微程序那样有专门支持开中断的指令。

5.3 新增信号

为了实现 8.2 中所描述的功能和流程图，我添加了如下信号。

中断新增信号	说 明
PULSE	TEC-8实验台停止信号
MF	TEC-8实验台时序逻辑主时钟脉冲
EN_INT	=1表示CPU允许中断，CPU处于正常状态。=0表示禁止中断，CPU处于中断状态。重置后为1。
INT	=1是中断发生中标志，实际上就是EN_INT=1时PULSE脉冲有效
INTDI	禁止中断信号（仅在关中断节拍有效）：用来置EN_INT为0，即禁止中断将持续到中断子程序结束即执行IRET命令（开中断）（CPU执行中断服务子程序时期）。
INTEN	允许中断信号（仅在开中断节拍有效）：用来置EN_INT为1，即允许中断将持续到中断前状态（关中断前）（CPU正常运行时期）。
ST1	=1表示CPU目前处于中断前状态，执行关中断+LD中断服务子程序首地址，准备进入中断服务子程序
SST1	=1表示CPU发出中断请求，仅在某条指令的最后一拍中断发生时（INT=1）有效

5.3.1 INTDI、INTEN、EN_INT、INT

需要说明的是 INTDI、INTEN 的说法源于微程序中的关中断指令信号和开中断指令信号，因此这两者是一个仅在一拍有效的临时信号，在其他时刻应当为 0。而 EN_INT 是一个指示标志，指示目前 CPU 是处于正常状态或者是中断状态，并非临时有效。因此我们设计的目标是使得 INTDI 有效时将 EN_INT 置为 0，同理 INTEN 有效时又将其置为 1，从而形成闭环。

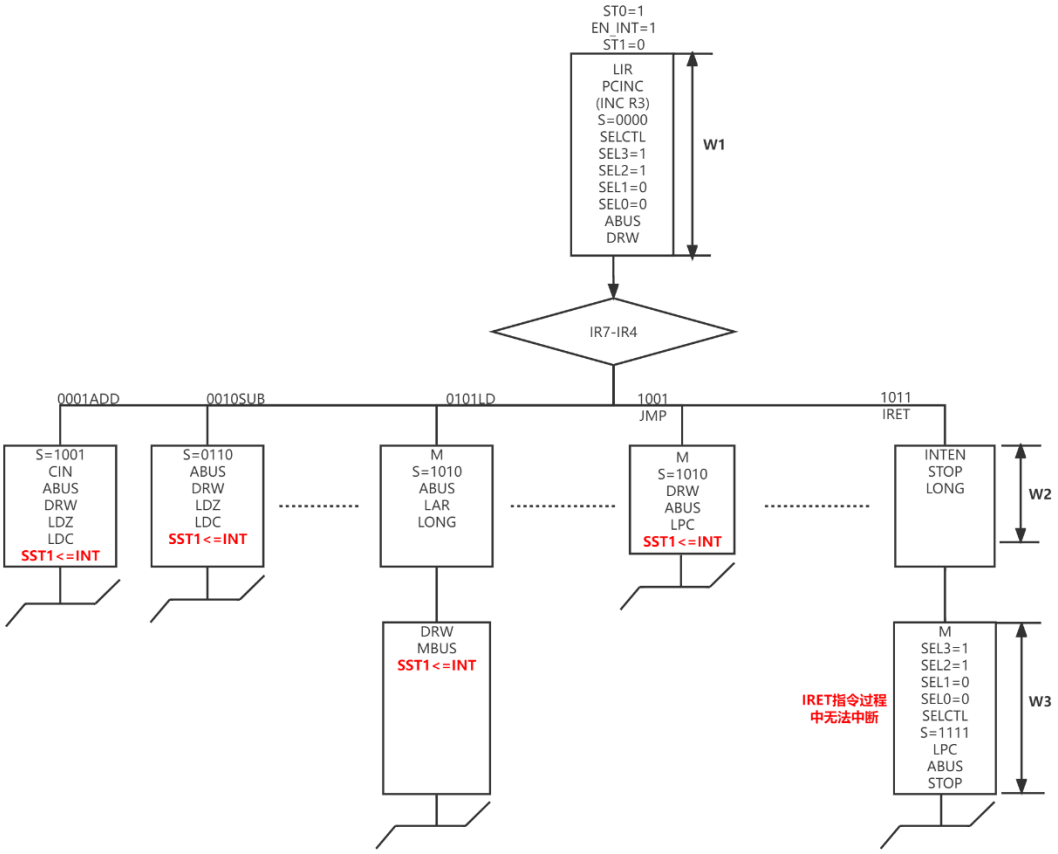
而 INT 信号则指示了 PULSE 信号，只有在允许中断（EN_INT=1）且 PULSE 信号（按下 PULSE 按钮）有效时，INT 有效。这几者的逻辑如下：

```
54. process(CLR,MF,EN_INT,PULSE)
55. begin
56.     if(MF'event and MF='1')then--每个主时钟上升沿检测是否发生状态改变
57.         EN_INT<=INTEN or((not INTDI) and EN_INT);--EN_INT=1 后只有在 INTDI 信号有效后才会变为 0。同理 EN_INT=0 后只有在 INTEN 信号有效后等于 1。
58.     end if;
59.     if(CLR='0')then
60.         EN_INT<='1';--重置后是允许中断，即正常 CPU 状态
61.     end if;
62.     INT<=EN_INT and PULSE;--指示 PULSE 信号
63. end process;
```

5.3.2 ST1 和 SST1

ST1 是用来指示下一拍进入中断前状态，且由 SST1 动态决定；可将 SST1 视为中断请求，而在本次中断控制器设计中，模型机发出中断请求必需的条件是，CPU 正常状态下执行某条指令最后一拍且单拍执行的同时按下 PULSE 信号使 INT 有效，从而发出中断请求即使 SST1=1。我们再并行一个进程在每拍结束时检测 SST1 从而动态改变 ST1，且结束中断前状态（ST1=0）则发生在中断前状态的 W2，在 W1 时已关中断（EN_INT=0）。如下：

```
64. process(T3,CLR)
65. begin
66.     if(T3'event and T3='0')then
67.         if(SST1='1')then--SST1=1 当且仅当执行某条指令最后一个拍结束时 INT=1（pulse 有效且 EN_INT 有效）
68.             ST1<='1';
69.         end if;
70.         if(ST1='1'and EN_INT='0' and W2='1')then--结束中断前状态：关中断后将 ST1 置 0
71.             ST1<='0';
72.         end if;
73.     end if;
74.     if(CLR='0')then
75.         ST1<='0';--重置后为非中断前状态
76.     end if;
77. end process;
```

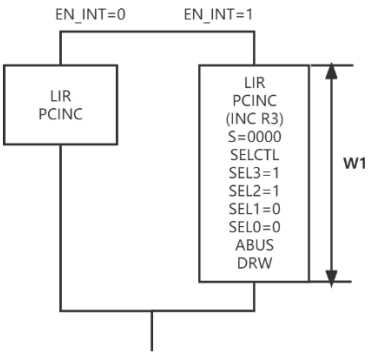


如图，SST1 在 CPU 正常执行指令状态下，将在每拍结束时由 INT 信号决定。由于中断功能控制器是基于顺序硬连线控制器设计的，因此在取指下除了中断功能流程图多了部分分支以及指令有些许改动，其他部分基本别无差异（SWCBA≠000 时一致）。

5.4 指令改动

中断控制器的设计涉及到了很多指令的改动，包括各指令周期的控制信号的改动以及指令格式的改动。在概述中已简单描述，这里更为系统地进行说明。

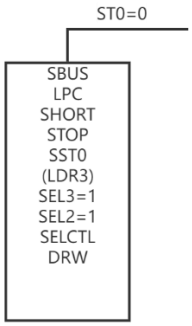
- PCINC 同步 INC R3



如图，在 CPU 正常状态下执行指令的第一拍需要 LIR 以及 PCINC，为了同步 R3 也自增，这里我们需要采用指令系统中 INC R3 的做法（TEC-8 数据总线和指令总线双总

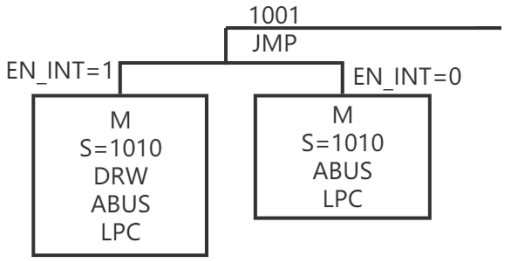
线机制保证了两操作能在同一拍进行），通过 SELCTL 激活二选一选择器，选中 R3 同时改变 ALU 的选择模式即 S=0000 使得 F=A+1 再使 DRW 有效，从而使得 PCINC 与 INC R3 同步进行。在中断状态下则不需要同步 R3。

- **LPC 同步 LD R3**



如图，在 CPU 正常状态下修改任意 PC 指针时，LPC 应与 LD R3 同步进行，因此只需使得 LR3 和 DRW 有效即可。

- **JMP 同步 Rd->R3**



从指令的格式入手，JMP Rd 的原始指令格式：1001Rdxx；更改后的指令格式为 1001 11Rd。

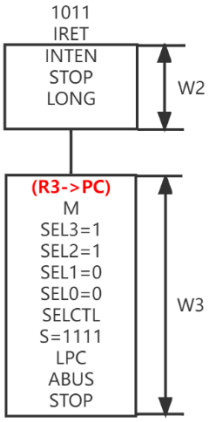
要实现 Rd->R3，Rd1、Rd0 只能为 11，因为我们要使 LR3 有效，以 LD R3，所以 JMP 指令的格式只能发生改变变为 **1001 11Rd**。从而 ALU A 端固定为 R3，因此 Rd 只能从 B 端出，因此 S 变为”1010”以输出 B 端，通过改变 JMP 指令格式，固定 R3 为 A 端且使得 LR3 有效，然后将 Rd 寄存器的值从 B 端输出到 ABUS，同时使得 LPC、DRW 有效，从而实现了同时 LD PC 和 LD R3。

为了方便，在中断功能控制器的指令中，所有 JMP 指令的指令格式都被更改为 100111Rd。

- **舍弃的指令**

本次设计在正常 CPU 状态下是不支持 **JC** 和 **JZ** 指令的，JC 和 JZ 指令跳变后同样要同步 R3，但跳变取决于 offset 即 IR3-IR0。我们不能通过正常的方法直接让 offset+R3->R3。理论上我们可以从双端口存储器左端读出指令，然后将指令和 00001111 相与从而提取出 IR3-IR0 即 offset，再将其存入又一个寄存器与 R3 相加再存入 R3，不过这样做太过于复杂，需要 6 拍甚至更多拍来完成，因此我暂时搁置了这部分的实现。不过在中断服务状态下是可以正常执行 JC 和 JZ 指令的。

- **IRET 指令**



IRET 指令需要完成两个功能：开中断和返回断点。前者使 INTEN 有效，后者实际上就是 R3->PC。ITRT 指令结束后由于开中断，EN_INT 在主时钟的上升沿被置为有效，从而下一个指令周期 CPU 将重新进入正常状态，且 PC 返回断点值，从而达到恢复断点的目的。

5.5 测试

	寄存器初值	R0=00H		R1=00H
地址	程序指令	机器码	16 进制	
00H	INC R0	01000000	40H	
01H	INC R0	01000000	40H	第二次循环第二拍同时按 PULSE 和 QD
02H	INC R0	01000000	40H	
03H	INC R0	01000000	40H	
04H	INC R0	01000000	40H	
05H	INC R0	01000000	40H	
06H	JMP R1	10011101	9DH	
07H				
08H				
09H	ADD R0 R0	00010000	10H	
0AH	IRET	10110000	B0H	
	检测寄存器	R0=10H	R1=00H	R3=02H

这里编写了一个简单的测试程序来检验中断功能控制器。我们首先将寄存器 R0、R1 置为 00H，然后在 CPU 正常状态下修改 PC 指针为 00H；顺序执行复数个 INC R0 指令，如果中断功能和同步功能没问题，R3 将同步自加，JMP 指令跳转到 R1 即 00H。然后我们在第二指令即 01H 的第二拍发出中断请求，此时的 PC 值为 02H。然后进入中断前状态执行关中断、LD 子程序首地址 09H，执行完 ADD R0 R0 后执行 IRET 指令开中断并返回断点。通过检测 R0、R1、R3 的值为 10H、00H、02H 验证了本小组设计的中断功能硬连线控制器的正确性，实现了中断功能。

5.6 译码表

	ADD	SUB	AND	INC	LD	ST	JC	JZ	JMP	OUT	IRET	DEC	OR	STOP	写寄存器	读寄存器	写存储器	读存储器	取指
DRW	W2	W2	W2	W2	W3				EN_INT*W2			W2	W2		W1+W2				~ST0*~ST1*W1+~ST1*ST0*EN_INT*W1
PCINC																			ST0*W1
LPC									W2		W3								~ST0*W1+ST1*W2
LAR					W2	W2											~ST0*W1	~ST0*W1	
PCADD							C*W2	Z*W2											
ARINC																	ST0*W1	ST0*W1	
SELCTL											W3					W1+W2	W1	W1	~ST0*~ST1*W1+~ST1*ST0*EN_INT*W1
MEMW						W3									W1+W2		ST0*W1		
LIR																			ST0*W1
LDZ	W2	W2	W2	W2								W2	W2						
LDC	W2	W2	W2	W2								W2							
CIN	W2																		
S	1001*W2	0110*W2	1011*W2	0000*W2	1010*W2	1111*W2+1010*W3			1010*W2	1111*W2	1111*W3	1111*W2	1110*W2						
M			W2		W2	W2+W3			W2	W2	W3		W2						
ABUS	W2	W2	W2	W2	W2	W2+W3			W2	W2	W3	W2	W2						~ST1*ST0*EN_INT*W1
SBUS															W1+W2		W1	~ST0*W1	~ST0*~ST1*W1+ST1*W2
MBUS					W3													ST0*W1	
SHORT																	1	1	~ST0*~ST1*W1
LONG					W2	W2					W2								~ST0*~ST1*W1+~ST1*ST0*EN_INT*W1
SEL3											W3			ST0	W2				~ST0*~ST1*W1+~ST1*ST0*EN_INT*W1
SEL2											W3			W2	0				~ST0*~ST1*W1+~ST1*ST0*EN_INT*W1
SEL1														(~ST0*W1)+(ST0*W2)	W2				
SEL0														W1	W1+W2				
STOP											W2+W3			W2	W1+W2	W1+W2	W1	W1	~ST0*~ST1*W1+ST1*(W1+W2)
SST0														~ST0*W2		~ST0*W1	~ST0*W1	~ST0*~ST1*W1	
SST1																			
INTEN											W2								
INTDI																			ST1*W1

5.7 中断设计总结

功夫不负有心人，最终我实现了支持中断功能的硬连线控制器。不过由于 TEC-8 实验台数据通路、引脚的局限性，以及时间、精力的欠缺，最终我们设计的控制器仍有些许不足。如 CPU 正常状态下不支持 JC 和 JZ 指令，以及寄存器仅支持 R0-R2 使用（实验条件硬性限制），以及并未实现保存现场和恢复现场：理论上可以在中断前状态和 IRET 指令那里自定义一些操作（写读存储器）来将 R0-R2 保存到存储器中然后再读出，不过太过麻烦我初步计算至少需要 11 拍来完成这个操作，这里由于时间原因就没再涉及。不过总体而言，我们从行为描述的角度，成功勾勒出整个中断功能硬连线控制器的模型和流程图，完成了任务目标。

附录

1 测试程序

1.1 测试程序执行过程

测试程序每一步流程以及相应寄存器存储器													
每一步流程	R0	R1	R2	R3	[81H]	[82H]	[83H]	[60H]	[61H]	[62H]	[80H]	[FEH]	[FFH]
初始	00H	00H	60H	FDH	00H	00H	00H	67H	80H	FDH	60H	03H	03H
LD R0, [R2]	67H	00H	60H	FDH	00H	00H	00H	R2 地址中数据存入 R0 寄存器					
INC R2	67H	00H	61H	FDH	00H	00H	00H	R2 寄存器数据加 1					
LD R1, [R2]	67H	80H	61H	FDH	00H	00H	00H	R2 地址中数据存入 R1 寄存器					
ADD R0, R1	E7H	80H	61H	FDH	00H	00H	00H	和存入 R0					
JC 06H	E7H	80H	61H	FDH	00H	00H	00H	C=0 不变					
AND R1, R0	E7H	80H	61H	FDH	00H	00H	00H	不变					
SUB R0, R2	86H	80H	61H	FDH	00H	00H	00H	R0 与 R2 差存入 R0					
INC R1	86H	81H	61H	FDH	00H	00H	00H	R1 加 1					
STA R0, [R1]	86H	81H	61H	FDH	86H	00H	00H	R0 中的数据存入[R1]					
INC R3	86H	81H	61H	FEH	86H	00H	00H	R3 加 1					
JZ 0DH	86H	81H	61H	FEH	86H	00H	00H	不变					
LD R2, [R3]	86H	81H	03H	FEH	86H	00H	00H	[R3]中的数据存入 R2					
JMP[R2]	86H	81H	03H	FEH	86H	00H	00H	跳转到 03 指令					
ADD R0, R1	07H	81H	03H	FEH	86H	00H	00H	C=1					
JC 06H	07H	81H	03H	FEH	86H	00H	00H	跳转到 06 指令					
SUB R0, R2	04H	81H	03H	FEH	86H	00H	00H	差存入 R0					

INC R1	04H	82H	03H	FEH	86H	00H	00H	R1 加 1
STA R0, [R1]	04H	82H	03H	FEH	86H	04H	00H	R0 中的数据存入[R1]
INC R3	04H	82H	03H	FFH	86H	04H	00H	R3 加 1
JZ 0DH	04H	82H	03H	FFH	86H	04H	00H	不变
LD R2, [R3]	04H	82H	03H	FFH	86H	04H	00H	[R3]中的数据存入 R2
JMP[R2]	04H	82H	03H	FFH	86H	04H	00H	跳转到 03 指令
ADD R0, R1	86H	82H	03H	FFH	86H	04H	00H	和存入 R0
JC 06H	86H	82H	03H	FFH	86H	04H	00H	不变
AND R1, R0	86H	82H	03H	FFH	86H	04H	00H	存入 R1
SUB R0, R2	83H	82H	03H	FFH	86H	04H	00H	差存入 R0
INC R1	83H	83H	03H	FFH	86H	04H	00H	R1 加 1
STA R0, [R1]	83H	83H	03H	FFH	86H	04H	83H	R0 中的数据存入[R1]
INC R3	83H	83H	03H	00H	86H	04H	83H	Z=1
JZ 0DH	83H	83H	03H	00H	86H	04H	83H	跳转到 0D 指令
INC R3	83H	83H	03H	01H	86H	04H	83H	R3 加 1
INC R3	83H	83H	03H	02H	86H	04H	83H	R3 加 1
SUB R0, R2	80H	83H	03H	02H	86H	04H	83H	R0 与 R2 的差存入 R0
LD R2, [R0]	80H	83H	60H	02H	86H	04H	83H	[R0]中的数据存入 R2
ADD R3, R2	80H	83H	60H	62H	86H	04H	83H	和存入 R3
LD R3, [R3]	80H	83H	60H	FDH	86H	04H	83H	存入 R3
OUT R0	80H	83H	60H	FDH	86H	04H	83H	显示 R0 的值

1.2 演示时测试程序

该部分是在验收展示时使用到的完整的测试程序，包括顺序、流水、中断硬连线控制器的测试。前两者顺序、流水使用的是 3.6 节中给出的测试程序，在附录的 1.1 节（上面）给出了该测试程序详细执行过程；中断控制器的测试程序也在 5.5 中给出；这里相当于一个汇总。

顺序和流水控制器测试程序初值

寄存器初值	存储器初值	
R2=60H	[60H]=67H	[80H]=60H
R3=0FDH	[61H]=80H	[0FEH]=03H
	[62H]=0FDH	[0FFH]=03H

测试指令及机器码			
地址	程序指令	2 进制机器码	16 进制机器码
00H	LD R0,[R2]	01010010	52
01H	INC R2	01001000	48
02H	LD R1,[R2]	01010110	56
03H	ADD R0,R1	00010001	11
04H	JC 06H	01110001	71
05H	AND R1,R0	00110100	34
06H	SUB R0,R2	00100010	22
07H	INC R1	01000100	44
08H	STA R0,[R1]	01100100	64
09H	INC R3	01001100	4C
0AH	JZ 0DH	10000010	82
0BH	LD R2,[R3]	01011011	5B
0CH	JMP [R2]	10011000	98
0DH	INC R3	01001100	4C
0EH	INC R3	01001100	4C
0FH	SUB R0,R2	00100010	22
10H	LD R2,[R0]	01011000	58
11H	ADD R3,R2	00011110	1E
12H	LD R3,[R3]	01011111	5F
13H	OUT R0	10100000	A0
14H	STP	11100000	E0
中断测试初值	R0=0;R1=16H		
16H	INC R0	01000000	40
17H	INC R0	01000000	40
18H	INC R0	01000000	40
19H	INC R0	01000000	40
1AH	INC R0	01000000	40
1BH	INC R0	01000000	40
1CH	JMP R1	10011101	9D
1DH			
1EH	ADD R0 R0	00010000	10
1FH	IRET	10110000	B0
扩指测试	R0=00H;R1=1FH;R2=F8H		初值
21H	OUT R1	10100100	A0
22H	MOV R0 R2	11000010	C2
23H	OR R0 R1	11010001	D1

地址 00H-14H 用来测试顺序和中断，赋予寄存器和存储器初值后连续拍执行，检查 CPU 停机后 R0-R3 以及[81H]-[83H]中的值是否正确。

地址 16H-1FH 用来测试中断控制器，具体方法在 5.5 中已给出。
地址 21H-23H 用来测试三条扩展指令。

2 计算机组成原理课程设计调试日志

8 月 23 日

一.今日进展

1. 上课了解要完成的任务
2. 利用课上 PPT 和之前计组数字逻辑材料复习 Quartus Ⅱ使用方法、VHDL 编程规范和语法格式、硬布线控制器基础知识

二.遇到问题与解决方法

问题：电脑段无法连接 TEC-8 实验箱
原因：TEC-8 实验箱内部芯片短路，更换试验箱

8 月 24 日

一.今日进展

1. 参考《TEC-8 组成实验指导书》上的硬连线流程图和网上的 VHDL 教程初步写好了最基础的顺序硬连线控制器的 VHDL 代码；经过语法排除和细节完善成功通过了 Quartus Ⅱ的编译。
2. 成功编译并下载基础的写寄存器代码，写入 epm7128 硬件中，达到预期目标，成功实现写入寄存器操作。
3. 将初步的顺序控制器 VHDL 代码编译好后，成功下载进硬件 epm7128 中。
4. 开始编写硬布线控制器流水模型处理器代码。

二.遇到问题与解决方法

问题一：写寄存器操作滞留在 ST0=1 这个阶段，导致第一次写完所有寄存器后再次运行写寄存器操作时，只能写 R2 和 R3 后两个寄存器。
原因：执行写存储器和读存储器操作时需要 ST0 一直为 1 以便 AR 寄存器自加方便操作，而写寄存器操作需要 ST0 在 ’0’ ’1 ’ 之间的循环转换，需要为写寄存器操作单独指定一个 ST0 赋值方案。
解决方案：在 T3 下降沿添加一个 IF 语句，判断是否为写寄存器操作，若是则将 ST0 赋值为‘0’。

问题二：在启动程序 SWCBA=’000’第一步修改任意 PC 指针时，发现程序由于未知原因跳过 SBUS，LPC 阶段；

原因：未知；有可能是 TEC-8 实验台自身问题。
解决方案：后续将代码进一步检查调试后不再出现该问题；

问题三：在启动程序 SWCBA=’000’操作，取指令并执行的步骤中，发现某些指令的信号赋值混乱，不符合流程图。
原因：程序在 case IR7-IR4 前进行了 if(W2=1)的判断，而某些指令中含有 W3，先行进行判断导致在 W3 的信号无法正常取得有效值。之所以进行 W2=1 的判断是为了防止在 W1 时也对 S3-S0 进行赋值。
解决方案：将 if(W2=’1’)的判断改为 if(W1=’0’)；

8 月 25 日

一.今日进展

1. 使用测试数据运行顺序硬布线控制器代码
2. 调试流水硬布线控制器代码
3. 复习中断相关知识，并尝试编写中断功能代码

二.遇到问题与解决方法

问题一：小组成员将编译好的流水处理器代码写入芯片当中，准备开始测试读写寄存器的基础功能。但调试过程中，读寄存器是 SEL3-SEL0 的指示灯并没有按照理想情况显示为 0001 和 1011 的循环，A7-A0 和 B7-B0 也自始至终不显示寄存器结果。
原因：在向芯片中写入一个曾经可以完整运行的代码后，我们发现此时实验台可以正常运行，说明不是 TEC-8 的问题。在回到 Quartus ii 界面检查所有环节后，我们发现是因为在将代码拷贝到实验室电脑后，并没有在新建工程下进行引角分配，导致芯片无法对信号进行处理。
解决方案：按照指导书上的引脚分配表进行分配后，成功运行了读、写存储器功能。

问题二：小组成员使用验收程序进行调试，将初始数据和指令全部写入试验台后，置 DP=1，从头开始单步执行。但在实验中发现程序中某一步 JC 跳转指令并没有实际发生理论应该发生的跳转，而是顺序执行了下去。
原因：经过代码分析，小组成员发现问题出在 JC 指令的前一步 ADD 指令上。在流程图设计时，小组成员由于疏忽，在 ADD 指令代码中并没有添加 LDC，即输入进位标志的代码，导致了 ADD 过程实际上并没有把进位返回给芯片，进而使得 JC 指令错误判断，并没有发生跳转。
解决方案：在代码中添加 JC<=W2 后，再次运行指令，成功跳转。

问题三：小组成员在运行测试程序时，执行完毕后检查存储器对应位置的数值，但发现在[81], [82], [83]这三个应该存有非零数据的存储单元中仍然存储的是 00H，其他内存单元的数值也都没有发生变化，说明指令执行过程中并没有把对应数值存入内存单元中。

原因：经过代码分析，小组成员发现 ST 指令对应的代码存在问题。代码中并没有给 LAR 赋值，导致执行将寄存器中数据写入存储器这一指令时，并没有将地址写入地址寄存器中，进而没有选中相应的存储器单元，导致存储到了错误位置。
解决方案：在代码中添加 LAR<=W2 完成赋值后，再次运行指令，运行完毕后检查存储器单元的数值均正确。

8 月 26 日

一.今日进展

1. 小组成员进一步调试流水控制器代码，编写测试程序依次调试解决了 JMP 和 JC 操作上的 BUG；最终完成流水控制器的设计、组装和测试；
2. 上午小组成员参考微程序中断实现的原理以及老师的提示（同步 R3 和 PC），初步模拟出中断功能硬布线控制器的模型，且初步画出一个添加了中断功能的流程图，计划在下午完成整个系统的设计以及代码的编写和调试；
3. 下午延续上午的工作，小组成员逐步完善了中断硬布线控制器模型的设计，同时编写 VHDL 行为描述结构进行同步调试以检验设计出来的系统每一小部分的中断功能。最后在傍晚调试成功，顺利完成中断功能控制器的设计和测试。

二.过程描述

上午：小组成员共同调试流水线代码，编写测试程序依次调试解决了 JMP 和 JC 操作上的 BUG，上午后半段时间小组成员参考微程序中断实现的原理以及老师的提示（同步 R3 和 PC），初步模拟出中断功能硬布线控制器的模型，且初步画出一个取值模式添加中断功能的流程图，计划在下午完成整个系统的设计以及代码的编写和调试。
下午：延续上午的工作，小组成员逐步完善了中断硬布线控制器模型的设计。在这个过程中小组成员同时编写行为描述结构 VHDL 语言进行同步调试，以检验设计出来的系统每一小部分的中断功能。每完成一部分流程图的设计就用代码复刻并在编译成功后写入 epm7128 硬件，一步步调试从而完善整个系统同时也验证了中断功能流程的可行性。具体而言，第一步小组成员参考微程序中断原理定义了 INT、EN_INT、INTDI、INTEN 等信号，同时引入主时钟脉冲 MF 和停止信号 PULSE，设计出各信号的赋值逻辑，比如 INT 只在 EN_INT 和 PULSE 同时有效时有效。EN_INT 有效表示当前处于正常执行指令周期，=0 则表示处于中断服务子程序周期等等。后续又初步完成了由按下 PULSE-->中断前状态（ST1=1:执行关中断指令+进入中断服务程序首地址）的逻辑；后续完成了 CPU 正常状态下同步 R3 自增的逻辑设计（具体包括 LPC 时同步 LR3 以及 PCINC 时同步 INC R3）；最后仔细思考后，放弃了 JC 和 JZ 命令（TEC-8 数据通路没有办法实现与 PC 的同步），同时考虑 JMP 的可行性，经过仔细思考，小组成员从指令格式入手使得执行 JMP 后能使 R3 和 PC 寄存器同步；

三.遇到问题与解决方法

问题一：在 CPU 正常执行指令状态时（非中断服务子程序），我们要保证 PC 和 R3 寄存器同步增加，其中某些信号如 DRW 和 ABUS 赋源为 W1 and EN_INT，但最后发现 DRW 和 ABUS 始终为 0；

原因：通过调试后发现后续的 case 会出现诸如 DRW<=W2 或者 ABUS<=W3 等信号赋源，从而导致在 W1 节拍，DRW、ABUS 等信号始终为 0。
解决方法：对 W1 和 W2、W3 分开判断运行。出现这种情况的原因是 process 进程内信号赋值仅取决于最后一次赋值；

问题二：pulse 按钮不起作用，信号灯始终为暗；
原因：没有外接彩色电缆，导致 TEC-8 实验台上的 PULSE 信号没有进入 epm7128 硬件；
解决方法：外接彩色电缆；

问题三：EN_INT 信号始终为 1，在设计中断功能控制器时，理论上关中断后在主时钟脉冲的上升沿 EN_INT 信号将会被置为 0；
原因：没有初始化 INTDI 和 INTEN，在没初始化 INTEN 之前，INTEN 由于未知原因始终为 1；我是如何可视化 INTEN 信号找到原因的？我使用了一种取巧的方法：在 SELCTL 无效时，SEL3-SEL0 相当于闲置的可视指示灯，利用这几个输出信号将其赋值为需要调试的信号 INTEN 即可。
解决方法：在 process 进程前面讲 INTEN 和 INTDI 置为 0；

问题四：在单独并行一个 process 进程处理 ST0/ST1 时，发现将判断 if(CLR=‘0’)放在带 T3 边缘（上升下降沿）判断的语句前时编译会报错，说无法在此时钟边缘上实现分配寄存器以及不将其值保持在时钟边缘之外。
原因：未知，小组成员猜测是由 VHDL 的语法决定了边缘检测需在朴素信号检测之前。
解决方法：先进行边缘检测再进行清零检测。

问题五：如何在 CPU 正常执行指令状态下实现 JMP 指令同步更新 PC 和 R3 寄存器的值；
分析：思考了很久在正常 CPU 执行状态下如何同步 R3 和 PC 值，我们不仅要使得 Rd->PC,还要同步更新 R3 即 Rd->R3。但是我们没有办法记录 Rd 寄存器的地址（因为它是由 JMP 指令中的 IR3-IR0 中的值决定），且 IR3、IR2 确定了 RD1、RD0（SELCTL 无效），从而确定了 LRd（d=0,1,2,3），我们要 LD R3 一定要使 DRW 有效且 LR3=1，这是初步思考。首先我考虑了多加一拍，但是我们要 LR3 一定要使总线上是 Rd 的数据，因此多加一拍毫无意义。使 SELCTL 有效呢？这个也不行，因为我们一定要保证 Rd0 和 Rd1 是 11，使得 LR3 有效，但我们无法确定 JMP Rd，Rd 是哪个寄存器，从而 SEL1 和 SEL0 将无法确定找到 Rd 寄存器。因此最终我们只能从原始指令 IR3-IR0 入手；山重水复疑无路，柳暗花明又一村。通过前面的分析，首先可以确定的是 Rd1、Rd0 只能为 11，因为我们要使 LR3 有效，以 LD R3，所以 JMP 指令的格式只能发生改变变为 100111Rd。从而 ALU A 端固定为 R3，因此 Rd 只能从 B 端出，因此 S 变为”1010”以输出 B 端，从而问题迎刃而解，通过改变 JMP 指令格式，固定 R3 为 A 端且使得 LR3 有效，然后将 Rd 寄存器的值从 B 端输出到 ABUS，同时使得 LPC 有效，从而实现了同步 LD PC 和 LD R3。
解决方法：改变 JMP 指令格式为 100111Rd，且改变 ALU 选择方式 S=1010 以输出 B 端，同时置 DRW 为 EN_INT 使得在正常状态下可以 LD R3。

问题六：在连续拍执行取指操作（SWCBA=000）时，CPU 清零重置后按 QD 直接跳过第一阶段的任意修改 PC 指针操作，即不会在 ST0=0 的 W1 节拍停机；
原因：在经过调试后发现，我们在 W1 令 STOP 信号有效，实际上在 W2 才停机，即在某一拍停机信号有效，下一拍才停机；我们在读寄存器指令操作时（SWCBA=011），令 STOP<=W1，在使用连续拍测试时发现实验台在 W2 停机，从而验证了该思想。
解决方法：我们直接令 STOP<=not ST0，原来是 STOP<=not ST0 and W1；这样做就能使 CPU 在 CLR 清零后还未产生时序信号时使得 STOP 有效，从而进入 W1 节拍后成功停机以修改任意 PC 指针。

8 月 27-8 月 31 日

8 月 27 日：小组成员已顺利完成三个任务目标即顺序、流水、中断控制器的设计与测试，后续计划开始编写报告文档，首先我们初步定制了一个报告的标题版框架，然后小组成员根据自己的分工各自开始编写自己所负责部分的报告文档，同时绘制流程图、译码表以及各种示意图和表格等。
8 月 28 日：小组成员继续编写文档，陆陆续续完成了各自的部分，由组长整合后顺利得到了课设报告的初稿，同时后续整理调试日志以及遇到的问题。
8 月 29 日：完成调试日志的整理，制作验收时的演示 PPT。
8 月 30 日：编写验收时的测试程序，并依次将顺序、流水、中断组装到 TEC-8 上进行调试与测试，最后成功运行，顺利通过测试。
8 月 31 日：成功完成验收。

3 心得体会

唐子钰

本次课设极大地弥补了我大二下学期缺乏实际上机进行实验的经验。

就本次课设达成的目标而言，本次课程设计，我个人独立完成了顺序硬连线控制器以及中断功能硬连线控制器的设计，成功绘制了中断功能流程图，然后基于 VHDL 分别完成了两个控制器的行为描述语言代码编写，并在 Quartus Ⅱ 9.1 平台上成功编译，组装到 TEC-8 调试并最终实现了指令系统以及控制模式所要求的功能。同时基于团队，在整个课设的时间段内，我们团队成员互相协调，保持沟通与信息交互，成功完成了三个目标，即顺序硬连线控制器、流水控制器以及中断功能控制器的设计。

就本次课设的个人收获而言，首先，由于课设是在大三开学前的小学期，随着这次机会我又一次复习了大量有关计算机组成原理和数字逻辑的理论知识，加深了对这些知识的印象；同时，本次课设使我进一步了解了硬连线控制器设计的原理和具体流程，加深了对流水、中断的理解；然后，通过实际体验在硬件上进行 VHDL 硬件描述语言的编写，我对 VHDL 的熟练程度大大增加，包括 VHDL 中的各种信号定义顺并赋值以及需要注意的语法等等。综合而言，本次课设给与我这么一个平台，我从无到有分别完成了顺序、中断控制器硬件描述 VHDL 语言的编写，先是从理论上即硬连线控制器流程图（包括中断）到实际的编程语言 VHDL 的编写，再是从 VHDL 完成编译到写入硬件 EPM7128，最终组装到 TEC-8 实验台上成功运行，通过自己写的硬连线控制器底层逻辑

成功实现各个指令以及中断功能，从理论和实践两方面验证了自己所设计硬连线控制器（即 VHDL 代码）和流程图的正确性，极大地加深了我对硬连线、信号赋值逻辑、硬件运行逻辑、流水中断原理、数据通路、CPU（包括运算器、时序发生器等）、时序（节拍与脉冲）、VHDL 硬件语言以及可编程器件底层逻辑（数据流）等的理解；只能说借着本次机会，不仅理论上大大加深了我对各种知识的理解，从实践的角度上来看，丰富了我对真正电子设计中对底层逻辑设计、硬件数据通路以及硬件(TEC-8)实践操作等的认识；可以说本次课设对于我个人而言收获颇丰。

就本次课设的过程而言，整体来说过程较为顺利，我们只用了 4 天时间就完成了顺序、流水以及中断功能控制器的设计包括在 TEC-8 实验台上的组装调试。对于我个人来说，印象最深刻的还是要数中断功能控制器流程图的理解与设计以及在 TEC-8 实验台上对中断控制器 VHDL 硬件代码的调试。其中不乏遇到很多问题 BUG，比如最开始初步设计好中断控制器发现不管怎么按 PULSE 按钮，INT 始终为 0，检查半天去问老师才发现是没有外接缆线；除此之外，在处理 JMP 指令同步 R3 时，我花了大量时间思考具体实现的方法，最终回到 CPU 模型机和数据通路通过逻辑反推去确定了 JMP 的 IR3、IR2 只能固定为 11 才有了后续通过更改 JMP 指令格式来同步 LD R3 的解决方法；还有很多在调试、实操以及测试中的问题这里就不再一一说明了（调试日志），最后只能说功夫不负有心人，最终我是完成了中断功能硬连线控制器的设计以及 VHDL 硬件语言行为描述代码的编写，实现了中断功能。

总结，本次课设给与我的提升不仅仅体现在个人学识、综合素养以及硬件实践操作的提高上，基于团队而言，我们小组进行了大量的交流，讨论，合理分工且都完成了各自分配的任务，对于我们整个团队来说，我们的个人能力、交流能力、团队协作、团队协商能力等都得到了提高。

刘梓航

知识掌握方面，我在上学期计算机组成原理的学习中仍然存在许多漏洞，特别是硬布线控制器方面的知识，由于课本中涉及较少，导致我对硬布线的原理掌握不透彻，特别是对其中“机器节拍”和“时钟”的概念理解不到位，导致了在开始设计的过程中，对每个节拍执行的操作流程并不熟悉。经仔细查阅上学期的课件和课本，我对硬布线控制器有了更加深入的认识，也使得我的计算机组成原理知识体系更为完善。

程序设计方面，由于本次试验程序需要用到 VHDL 语言，但距离我上一次使用 VHDL 已经过去了一年多，所以再次上手编写代码时有些困难。我在设计程序之前认为程序结构很简单，只需要对每一种情况对涉及的相关信号进行赋值即可。但实际上的设计过程要复杂得多。于是我找到数字逻辑课程讲授 VHDL 语言的部分进行重新学习。在编写过程中，经常会遇到找不到原因的报错，调试后发现经常出现某些信号的赋值发生了错误，情况没有考虑全面。于是我从头再来，更为谨慎地先列出每个信号的逻辑译码表，在化简确认后再进行编程，这样一来程序编写过程就变得顺畅了许多。

实践方面，由于上学期的计算机组成原理课程在线上进行，所有实验全部使用的仿真软件，所以许多实际操作中的细节被忽略了。在接触真实 TEC-8 操作台后，我发现了许多在仿真软件上无法模拟的情况。例如操作台并不能直接看到每个时候每个存储单元的数值，需要通过读取，从指示灯处来获得。此外在中断控制时，PULSE 信号的使用也和仿真软件中不同，在进行中断时，需要一直按住 PULSE 信号并按一下 QD 信号才会使得终端响应信号被正确赋值。此外在程序设计时，我们还明白了要尽量避免在某

一个时钟沿设计过多信号变化，这样有可能会导致试验设备发生故障。通过真实操作试验台进行调试，我也收获了许多将来会用到的实践经验。

黄安

通过本次课程设计，充分地对整个 TEC-8 模型计算机有了全面的认识，详细了解了各操作信号与时钟节拍的关系，也通过这课程设计，学习了硬布线控制器如何完成指令的操作流程，同时也提高了自身从 VHDL 语言设计到硬件部署调试的能力，在实验过程中，由于个人的一些粗心的错误，如 ST 指令代码缺少了 LAR 信号，ADD 指令代码缺漏了 LDC 信号，导致了调试时浪费了许多不必要的 debug 时间，甚至还一度编译错了测试程序的机器指令，使得在验证代码完整和正确性过程中，导致结果错误，从而陷入怀疑代码错误而对不存在的 bug 进行 debug 的陷阱中，虽然最后这些错误都及时被其他组员发现，但浪费的时间也使我非常惭愧，也认识到了在团队合作中，细心是一项必备要素，因为个人的粗心会拖累整个团队，同时也会给项目带来不必要的资源时间浪费。总的来说，本次课程实验使得我更深刻地理解了计算机各部分之间的配合及协作关系，也增加从软件到硬件的经验，更重要的是在团体合作中吸取了使我受益匪浅的教训。

马天驰

在这次计算机组成原理课程设计过程中，我们团队 4 人通力协作，充分发挥每个人的特长，顺利完成了顺序硬连线控制器、流水硬连线控制器以及中断功能的实现。在实验过程中我温习了硬连线控制器相关知识。上学期计算机组成原理实验要求完成微程序的仿真，这次我从硬连线的角度更加深入的理解了 CPU 原理。

此次实验一开始确实很蒙，但是我认真复习计算机组成原理 cpu 相关知识，以及 VHDL 语言语法规则之后，便发现眼前的困难只不过是纸老虎。硬连线控制器核心是控制信号的组合逻辑设计，只要掌握各种信号之间的关联。

此次实验同时也是对我们做事细心程度的考验，很多次调试遇到问题，其原因往往是某一个步骤由于粗心做错导致。比如在 TEC-8 调试顺序硬连线代码时发现昨天可以执行的命令今天却没有执行，仔细检查了代码后没有问题，后来发现是我们新建立了一个项目，只是复制了原来的代码并编译而没有配好引脚导致。还有我们在仿真平台上的运行结果与 TEC-8 上运行结果不一致，后来找到原因是我们代码设置的编译码与仿真平台不一致，但是测试时使用同一个指令。这些都很考验我们每一步操作的准确程度，不能有丝毫马虎大意。

此次实验很考验我们对细节的处理能力，尽管代码主体的架构并不是十分复杂，但是编写好代码之后仍然错误不断，很多错误的原因在于一些细节没有处理好，比如一开始为不同的模式设定同样的 FLAG 转换，但是后来发现不同模式的转换方式不一样，比如读写存储器操作，存入地址之后便一直为读写存储器不用转换状态，但是写寄存器需要在写 ROR1、R2R3 两个过程中来回切换。因此要采取不同的处理细节。

这次实验不仅强化了我对 CPU 控制器的理解，同时提高了我硬件的实验能力和编程能力，我在此次实验中受益匪浅。