

**实践报告**



题目：流水硬连线控制器的设计

**学 院 ： 计算机学院**

**团队成员 ： 胡天翼 2018210547**

**张林羽觐 2018210309**

**马锐文2018212003**

**2020年 9 月 10 日**

**北京邮电大学课程设计报告**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **课程设计**  **名称** | 计算机组成原理课程设计 | | **学 院** | **计算机学院** | **指导教师** | **张杰** |
| **班 级** | **班内序号** | **学 号** | | **学生姓名** | **成绩** | |
| **2018211318** | **06** | **2018210547** | | **胡天翼** |  | |
| **2018211318** | **05** | **2018210309** | | **张林羽觐** |  | |
| **2018211318** | **22** | **2018212003** | | **马锐文** |  | |
|  |  |  | |  |  | |
| **课**  **程**  **设**  **计**  **内**  **容** | 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，该CPU要求：   1. 能够完成控制台操作：启动程序运行、读存储器、写存储器、读寄存器和写寄存器。 2. 能够执行教师给定的指令，完成规定的指令功能。 3. 在原指令基础上要求扩指 4. 具有修改PC指针功能（开始时任意指针）   团队分工：  胡天翼：指令与任意PC代码编写、设计测试程序、代码调试、协调团队工作  张林羽觐：控制台代码编写、控制台操作  马锐文：指令代码编写、控制台操作、设计测试程序 | | | | | |
| **学生课程设计报告** | （见后） | | | | | |
| **课**  **程**  **设**  **计**  **成**  **绩**  **评**  **定** | 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 | | | | | |

注：评语要体现每个学生的工作情况，可以加页。

目录

[1 实验目的 1](#_Toc50730731)

[2 实验环境 1](#_Toc50730732)

[3.实验要求 1](#_Toc50730733)

[3.1实现一 1](#_Toc50730734)

[3.2实现二 1](#_Toc50730735)

[4 设计与调试任务 2](#_Toc50730736)

[4.1 设计流水硬连线控制器 2](#_Toc50730737)

[4.2 编写控制器VHDL代码并调试 2](#_Toc50730738)

[4.3 将控制器代码下载到TEC-8实验台 2](#_Toc50730739)

[4.4 测试流水硬连线控制器 2](#_Toc50730740)

[4.5 整理出设计文件 2](#_Toc50730741)

[5 实验原理 3](#_Toc50730742)

[5.1 硬连线控制器的基本原理 3](#_Toc50730743)

[5.2 机器指令周期流程图设计 3](#_Toc50730744)

[5.3 执行一条机器指令的节拍电位数 3](#_Toc50730745)

[5.4 EPM7128器件的引脚 4](#_Toc50730746)

[5.5 从非流水硬连线控制器到流水硬连线控制器 5](#_Toc50730747)

[5.6流水硬连线控制器参考流程图 6](#_Toc50730748)

[6 实验步骤 7](#_Toc50730749)

[6.1 设计微操作时间表 7](#_Toc50730750)

[6.2 补充与分析 10](#_Toc50730751)

[6.3 编写代码 11](#_Toc50730752)

[6.4 模拟与仿真 16](#_Toc50730753)

[6.5 调试过程 17](#_Toc50730754)

[7 遇到的问题 20](#_Toc50730755)

[胡天翼 20](#_Toc50730756)

[马锐文 21](#_Toc50730757)

[张林羽觐 21](#_Toc50730758)

[8 实验日志 21](#_Toc50730759)

[8月3日 胡天翼记录 22](#_Toc50730760)

[8月7日 胡天翼记录 22](#_Toc50730761)

[8月15日 马锐文记录 22](#_Toc50730762)

[8月21日 马锐文记录 22](#_Toc50730763)

[8月31日 胡天翼记录 22](#_Toc50730764)

[9月1日 张林羽觐记录 23](#_Toc50730765)

[9月3日 张林羽觐记录 23](#_Toc50730766)

[9月5日 胡天翼记录 23](#_Toc50730767)

[9月7日 张林羽觐记录 23](#_Toc50730768)

[9 实验总结 24](#_Toc50730769)

[胡天翼 24](#_Toc50730770)

[马锐文 24](#_Toc50730771)

[张林羽觐 25](#_Toc50730772)

[10 附录 25](#_Toc50730773)

## 1 实验目的

* 理解并掌握计算机组成原理理论课程中学习的CPU各部件运行原理与机制，将理论知识运用于实践，自主设计一个流水硬连线控制器，加深对CPU各模块工作原理及相互联系的认识。
* 学会使用虚拟仿真软件Quartus II 与实验操作台TEC-8。
* 以小组为单位进行实验，加强团队沟通精神与协调合作的能力。
* 培养科学研究能力，取得设计和调试的实践经验。

## 2 实验环境

操作系统：Windows10

使用语言：VHDL

仿真软件：Quartus II 9.1

硬件平台：TEC-8实验平台

## 3.实验要求

### 3.1实现一

根据设计方案，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，该CPU 要求能够完成控制台操作：启动程序运行、读存储器、写存储器、读寄存器和写寄存器。

### 3.2实现二

仿真模拟向量测试。附加功能：

A 在原指令基础上要求扩指

B 具有修改PC指针功能（开始时任意指针）

## 4 设计与调试任务

### 4.1 设计流水硬连线控制器

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，该CPU要求：

1. 能够完成控制台操作：启动程序运行、读存储器、写存储器、读寄存器和写寄存器。
2. 能够执行教师给定的指令，完成规定的指令功能。
3. 在原指令基础上要求扩指
4. 具有修改PC指针功能（开始时任意指针）

### 4.2 编写控制器VHDL代码并调试

在Quartus Ⅱ下对硬连线控制器对设计方案进行编程和编译

### 4.3 将控制器代码下载到TEC-8实验台

将编译后的流水硬连线控制器下载到TEC-8实验台上的ISP器件EPM7128中去，使EPM7128成为一个流水硬连线控制器。

### 4.4 测试流水硬连线控制器

根据指令系统，编写检测流水硬连线控制器正确性的测试程序，并用测试程序对硬连线控制器在单拍方式下进行调试，直到成功。

### 4.5 整理出设计文件

在调试成功的基础上，整理出设计文件

## 5 实验原理

### 5.1 硬连线控制器的基本原理

硬连线控制器的基本原理，每个微操作控制信号S是一系列输入量的逻辑函数，即用组合逻辑来实现，

S = f(Im，Mi，Tk，Bj)

其中Im是机器指令操作码译码器的输出信号，Mi是节拍电位信号，Tk是节拍脉冲信号，Bj是状态条件信号。

在TEC-8实验系统中，节拍脉冲信号Tk(T1～T3)已经直接输送给数据通路。因为机器指令系统比较简单，省去操作码译码器，4位指令操作码IR4～IR7直接成为Im的一部分；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，因此SWC、SWB、SWA可以看作是Im的另一部分。Mi是时序发生器产生的节拍信号W1～W3；Bj包括ALU产生的进位信号C、结果为0信号Z等等。

### 5.2 机器指令周期流程图设计

设计微程序控制器使用流程图。设计硬连线控制器同样使用流程图。微程序控制器的控制信号以微指令周期为时间单位，硬连线控制器以节拍电位(CPU周期)为时间单位，两者在本质上是一样的，1个节拍电位时间和1条微指令时间都是从节拍脉冲T1的上升沿到T3的下降沿的一段时间。在微程序控制器流程图中，一个执行框代表一条微指令，在硬连线控制器流程图中，一个执行框代表一个节拍电位时间。

### 5.3 执行一条机器指令的节拍电位数

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，少数指令需要3个节拍电位W1、W2、W3。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。

对于一些控制台操作，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，可以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，可以用某些特殊的寄存器标志标。例如建立一个FLAG标志，当FLAG=0时，表示该控制台操作的第1个W1、W2；当FLAG=1时，表示该控制台操作的第2个W1、W2。

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。

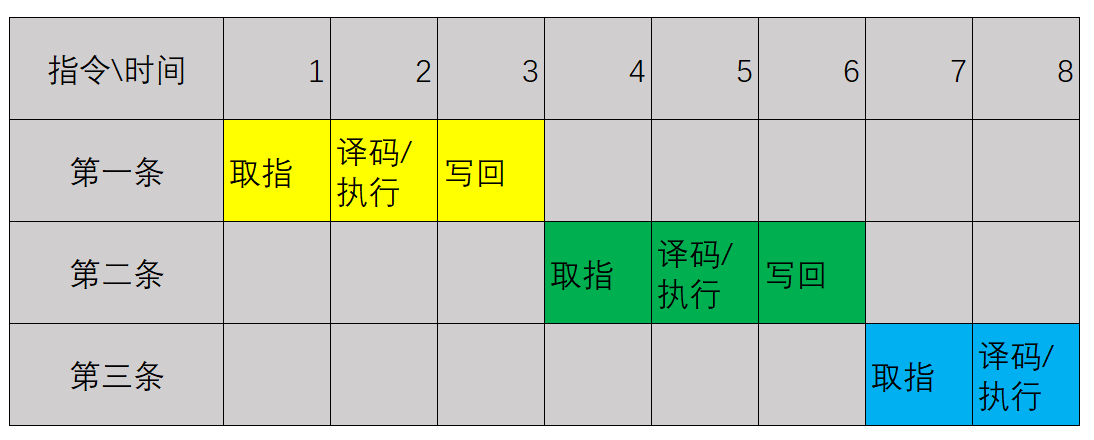
### 5.4 EPM7128器件的引脚

TEC-8实验系统中的硬连线控制器是用1片EPM7128器件构成的。为了使学生将主要精力集中在硬连线控制器的设计和调试上，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表中的规定。

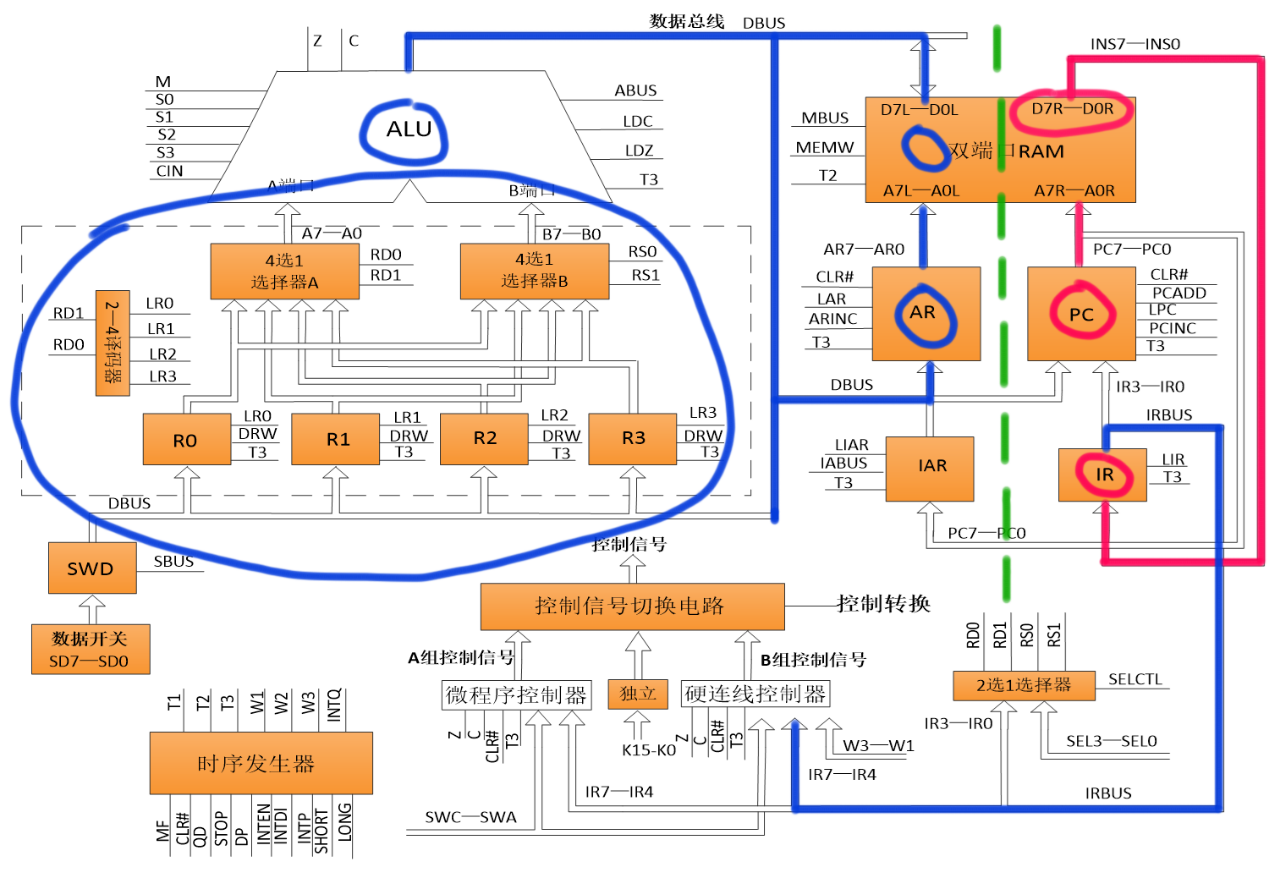
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 信号 | 方向 | 引脚号 | 信号 | 方向 | 引脚号 |
| CLR# | 输入 | 1 | MEMW | 输出 | 27 |
| T3 | 输入 | 83 | STOP | 输出 | 28 |
| SWA | 输入 | 4 | LIR | 输出 | 29 |
| SWB | 输入 | 5 | LDZ | 输出 | 30 |
| SWC | 输入 | 6 | LDC | 输出 | 31 |
| IR4 | 输入 | 8 | CIN | 输出 | 33 |
| IR5 | 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 |  |  |  |

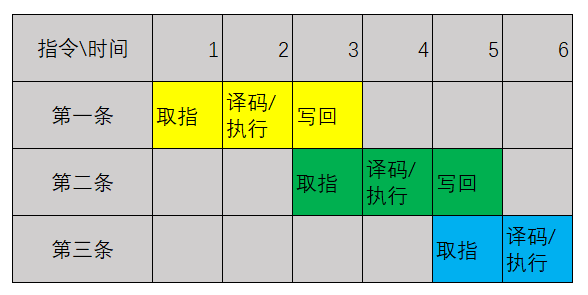
### 5.5 从非流水硬连线控制器到流水硬连线控制器

在硬布线控制器的设计中，指令有两种执行周期：①取指（W1）→译码/执行（W2）；②取指（W1）→译码/执行（W2）→写回（W3）.每条指令是顺序依次执行的，一条指令执行完成后才能进行第二条指令的取指。将其非流水硬连线控制器的原理绘制为时空图如下：



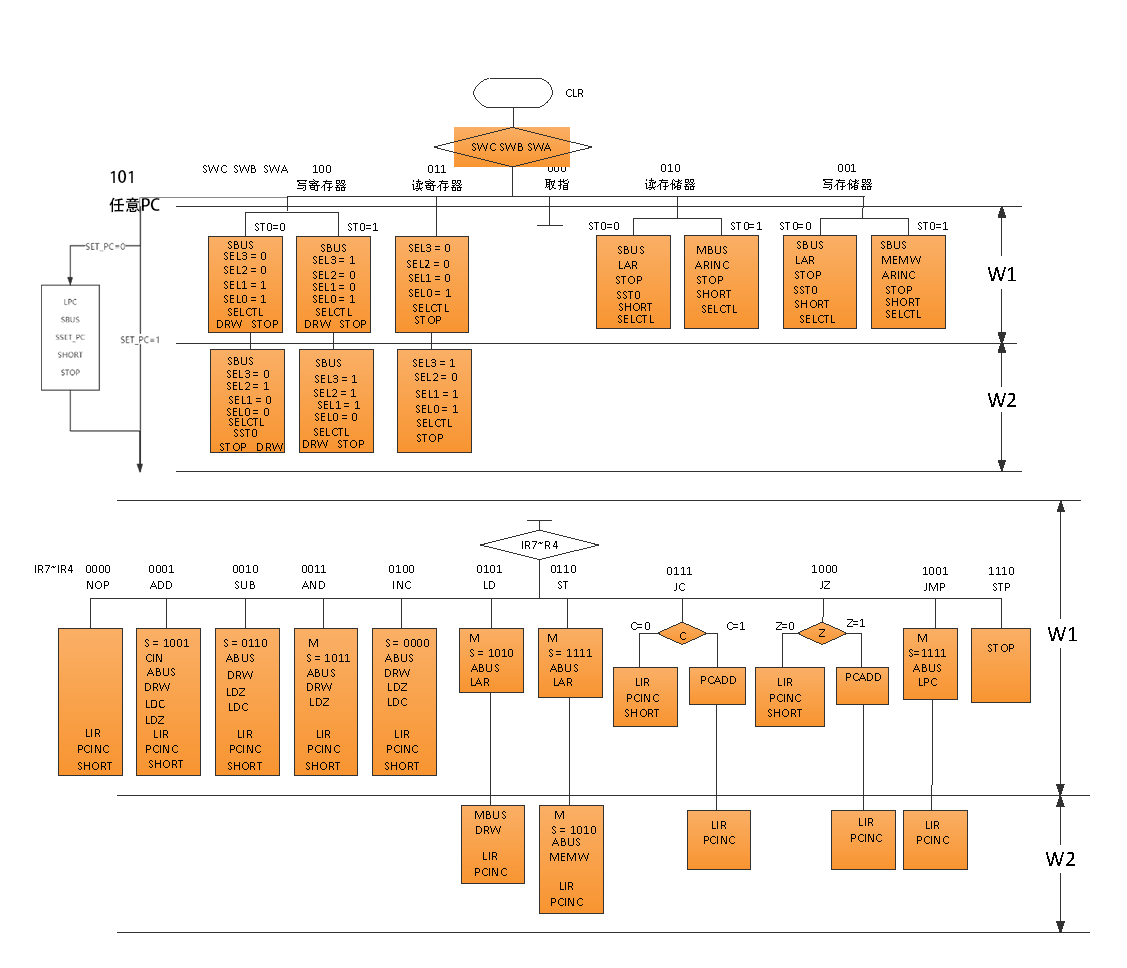
可以发现，在这其中有大量的空格，表明这些时空被浪费。通过观察数据通路，我们发现取指时仅使用双端口存储器的右侧端口、PC寄存器、IR寄存器以及双端口寄存器与相应的数据通路（图中的红线部分）；而译码、执行、写回时仅使用图中的蓝线部分的器件及数据通路（除JMP、JC、JZ指令）。因此取指是可以和执行部分同时进行，构成二级流水的。绘制为时空图如下





可以看到，在执行三条指令的情况下，流水硬连线控制器比非流水硬连线控制器快了3个时钟周期，大大提高了效率。

### 5.6流水硬连线控制器参考流程图



## 6 实验步骤

### 6.1 设计微操作时间表

#### 6.1.1 设计控制台操作

参考流程图，结合ST0的状态设计出每一种case，

例如写寄存器，SBUS = 1，SEL3 = ST0, SEL2 = W2 AND NOT(W1)…

微操作时间表如下：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 写寄存器 | | 读寄存器 | 读存储器 | | 写存储器 | |
| ST0 = 0 | ST0 = 1 |  | ST0 = 0 | ST0 = 1 | ST0 = 0 | ST0 = 1 |
| W1+W2 | W1+W2 | W1+W2 | W1 | W1 | W1 | W1 |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
| 0 \* (W1 + W2) | 1 \* (W1 + W2) | 0 \* W1 + 1 \* W2 |  |  |  |  |
| 0 \* W1 + 1 \* W2 | 0 \* W1 + 1 \* W2 | 0 \* (W1 + W2) |  |  |  |  |
| 1 \* W1 + 0 \* W2 | 1 \* W1 + 0 \* W2 | 0 \* W1 + 1 \* W2 |  |  |  |  |
| 1 \* W1 + 0 \* W2 | 1 \* W1 + 0 \* W2 | 1 \* (W1 + W2) |  |  |  |  |
| W1+W2 | W1+W2 |  |  |  |  |  |
| W1+W2 | W1+W2 |  | W1 | W1 | W1 | W1 |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  | W1 |
|  |  |  | W1 |  | W1 |  |
|  |  |  |  | W1 |  | W1 |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  | W1 | W1 | W1 | W1 |
| W1+W2 | W1+W2 | W1+W2 | W1 | W1 | W1 | W1 |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
| W2 |  |  | W1 |  | W1 |  |

#### 6.1.2 设计机器指令

主要参考流水硬连线控制器参考流程图，操作时间表，数据通路图，来分别进行具体的机器指令的设计。

比如：

ADD指令，

when IR7~IR4=0001，

在w1=1的区间内，s=1001，CIN，ABUS,DRW,LDZ,LDC,LIR,PCINC,SHORT=1;

微操作时间表如下：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 信号\指令 | NOP | ADD | SUB | AND | INC | LD | ST |
| SELCTL |  |  |  |  |  |  |  |
| ABUS |  | W1 | W1 | W1 | W1 | W1 | W1+W2 |
| M |  |  |  | W1 |  | W1 | W1+W2 |
| S |  | 1001 | 0110 | 1011 | 0000 | 1010 | 1111\*W1+1010\*W2 |
| SEL3 |  |  |  |  |  |  |  |
| SEL2 |  |  |  |  |  |  |  |
| SEL1 |  |  |  |  |  |  |  |
| SEL0 |  |  |  |  |  |  |  |
| DRW |  | W1 | W1 | W1 | W1 | W2 |  |
| SBUS |  |  |  |  |  |  |  |
| LIR | W1 | W1 | W1 | W1 | W1 | W2 | W2 |
| MBUS |  |  |  |  |  | W2 |  |
| MEMW |  |  |  |  |  |  | W2 |
| LAR |  |  |  |  |  | W1 | W1 |
| ARINC |  |  |  |  |  |  |  |
| LPC |  |  |  |  |  |  |  |
| PCINC | W1 | W1 | W1 | W1 | W1 | W2 | W2 |
| PCADD |  |  |  |  |  |  |  |
| CIN |  | W1 |  |  |  |  |  |
| LONG |  |  |  |  |  |  |  |
| SHORT | W1 | W1 | W1 | W1 | W1 |  |  |
| STOP |  |  |  |  |  |  |  |
| LDC |  |  | W1 |  | W1 |  |  |
| LDZ |  | W1 | W1 | W1 | W1 |  |  |
| SST0 |  |  |  |  |  |  |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 信号\指令 | JC,C=0 | JC,C=1 | JZ,Z=0 | JZ,Z=1 | JMP | STP |
| SELCTL |  |  |  |  |  |  |
| ABUS |  |  |  |  | W1 |  |
| M |  |  |  |  | W1 |  |
| S |  |  |  |  | 1111 |  |
| SEL3 |  |  |  |  |  |  |
| SEL2 |  |  |  |  |  |  |
| SEL1 |  |  |  |  |  |  |
| SEL0 |  |  |  |  |  |  |
| DRW |  |  |  |  |  |  |
| SBUS |  |  |  |  |  |  |
| LIR | W1 | W2 | W1 | W2 | W2 |  |
| MBUS |  |  |  |  |  |  |
| MEMW |  |  |  |  |  |  |
| LAR |  |  |  |  |  |  |
| ARINC |  |  |  |  |  |  |
| LPC |  |  |  |  | W1 |  |
| PCINC | W1 | W2 | W1 | W2 | W2 |  |
| PCADD |  | W1 |  | W1 |  |  |
| CIN |  |  |  |  |  |  |
| LONG |  |  |  |  |  |  |
| SHORT | W1 |  | W1 |  |  |  |
| STOP |  |  |  |  |  | W1 |
| LDC |  |  |  |  |  |  |
| LDZ |  |  |  |  |  |  |
| SST0 |  |  |  |  |  |  |

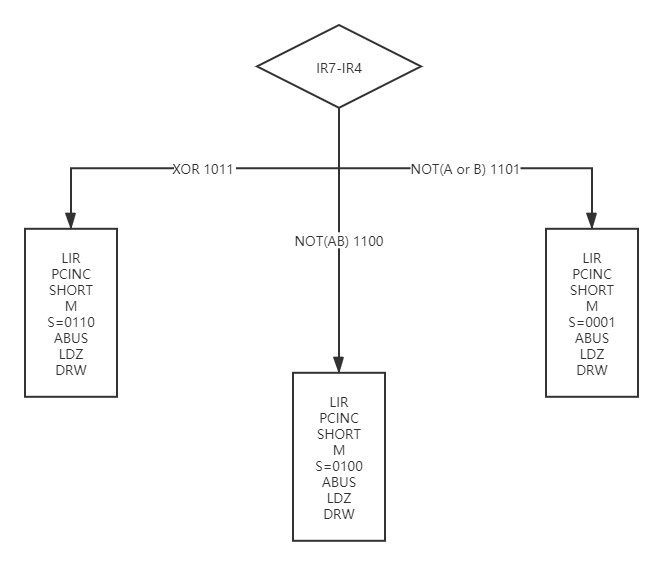
### 6.2 补充与分析

#### 6.2.1 考虑不能流水的特殊指令

指令集中一共有三条指令不能参与流水，分别是JC、JZ、JMP。原因在于：能够将指令流水的前提是指令在执行周期不改变PC寄存器与IR寄存器的值。但在JC、JZ、JMP中，为了实现PC跳转，需要修改PC寄存器的值。换句话说，此时的“下一条指令”的位置，不是PC在JC、JZ、JMP指令执行阶段的旧值，而是将指令中的操作数打入PC后的新值。因此需要多一个节拍，在PC的值更新后，再取指。

#### 6.2.2 扩充指令

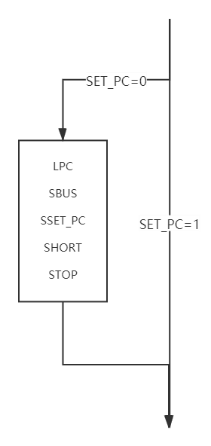
我们小组扩充了三条指令：异或、非与与非或



#### 6.2.3 任意PC

仿照指导书的流程图中ST0与SST0的设置方法，我增添了SET\_PC与SSET\_PC信号，此信号将在控制台置为101时起作用。SET\_PC判断当前是否还未执行过指令，可以设定执行指令的初始位置。SSET\_PC用于控制SET\_PC的翻转。在按下CLR时，将SET\_PC清零，表示此时还未开始执行指令，允许在进入时设置PC。

在指令执行的一开始，加入SET\_PC是否为零的检测。如是，进入设置任意PC的进程，LPC、SBUS信号打开，允许数据开关打入数据通路，再将数据存入PC寄存器中。于此同时，将SSET\_PC信号置1，此信号标志着SET\_PC需要翻转。而后在T3的下降沿时，将SET\_PC置1，这样之后的流程中将不再有设置PC这一步，而正常进入取指、译码、执行。原理图如下：



### 6.3 编写代码

#### 6.3.1定义库与输入输出信号

1. LIBRARY IEEE;
2. USE IEEE.STD\_LOGIC\_1164.ALL;
3. USE IEEE.STD\_LOGIC\_UNSIGNED.ALL;
5. ENTITY CPU IS
6. PORT (
7. --输入信号
8. CLR,T3,C,Z  : IN STD\_LOGIC;
9. W1,W2,W3    : IN STD\_LOGIC;
10. SW          : IN STD\_LOGIC\_VECTOR(2 DOWNTO 0);
11. IR          : IN STD\_LOGIC\_VECTOR(7 DOWNTO 4);

14. --输出信号
15. DRW     : OUT STD\_LOGIC;
16. PCINC   : OUT STD\_LOGIC;
17. LPC     : OUT STD\_LOGIC;
18. LAR     : OUT STD\_LOGIC;
19. PCADD   : OUT STD\_LOGIC;
20. ARINC   : OUT STD\_LOGIC;
21. SELCTL  : OUT STD\_LOGIC;
22. MEMW    : OUT STD\_LOGIC;
23. STOP    : OUT STD\_LOGIC;
24. LIR     : OUT STD\_LOGIC;
25. LDZ     : OUT STD\_LOGIC;
26. LDC     : OUT STD\_LOGIC;
27. CIN     : OUT STD\_LOGIC;
28. S       : OUT STD\_LOGIC\_VECTOR(3 DOWNTO 0);
29. M       : OUT STD\_LOGIC;
30. ABUS    : OUT STD\_LOGIC;
31. SBUS    : OUT STD\_LOGIC;
32. MBUS    : OUT STD\_LOGIC;
33. **SHORT**   : OUT STD\_LOGIC;
34. **LONG**    : OUT STD\_LOGIC;
35. SEL     : OUT STD\_LOGIC\_VECTOR(3 DOWNTO 0)
36. );
37. END CPU;

#### 6.3.2定义进程中使用的信号

ST0是分支信号,SST0是控制ST0翻转的信号，SET\_PC判断当前是否还未执行过指令，可以设定执行指令的初始位置。SSET\_PC用于控制SET\_PC的翻转。

1. ARCHITECTURE ARC OF CPU IS
2. SIGNAL ST0, SST0 : STD\_LOGIC;   --ST0是分支信号,SST0是控制ST0翻转的信号
3. SIGNAL SET\_PC, SSET\_PC    : STD\_LOGIC;   --SET\_PC判断当前是否还未执行过指令，可以设定执行指令的初始位置

#### 6.3.3 SET\_PC翻转控制

在SSET\_PC信号为1时，在当前QD结束的T3下降沿位置，将SET\_PC置1，表明PC已经被设置，不能再次被设置。

1. BEGIN
2. PROCESS (CLR, T3)
3. BEGIN
4. IF (CLR = '0') THEN
5. SET\_PC <= '0';
6. ELSE
7. IF (T3'EVENT AND T3 = '0' AND SSET\_PC = '1' ) THEN
8. SET\_PC <= '1';
9. END IF;
10. END IF;
11. END PROCESS;

#### 6.3.4 控制台判断

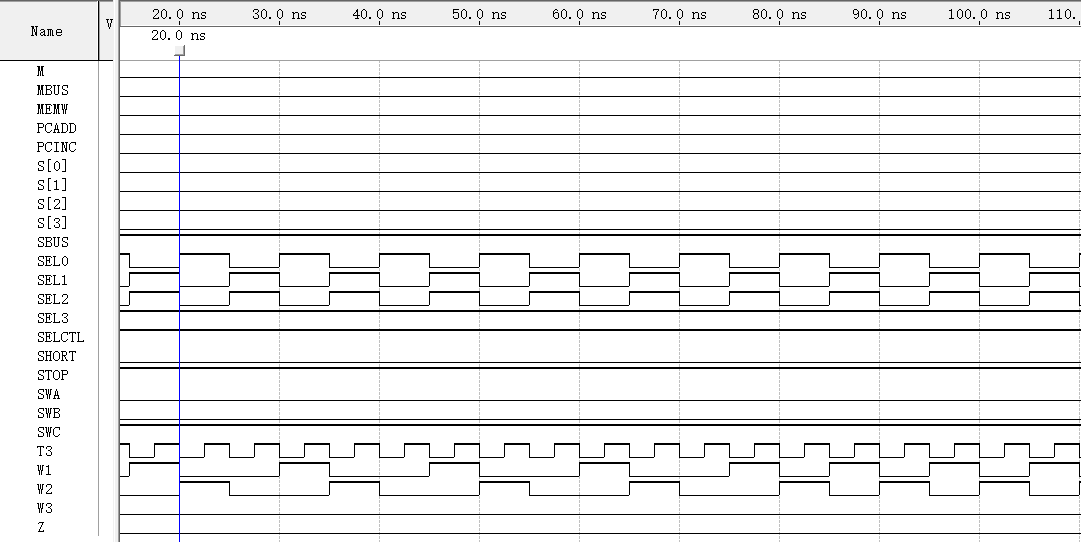
  （每种操作展示一条指令的代码，完整代码见附录）

1. PROCESS (SW, IR, W1, W2, W3, C, Z, CLR, T3)
2. BEGIN
3. --计算信号前，先将所有信号初始化为0
4. DRW     <= '0';
5. PCINC   <= '0';
6. LPC     <= '0';
7. LAR     <= '0';
8. PCADD   <= '0';
9. ARINC   <= '0';
10. SELCTL  <= '0';
11. MEMW    <= '0';
12. STOP    <= '0';
13. LIR     <= '0';
14. LDZ     <= '0';
15. LDC     <= '0';
16. CIN     <= '0';
17. S       <= "0000";
18. M       <= '0';
19. ABUS    <= '0';
20. SBUS    <= '0';
21. MBUS    <= '0';
22. **SHORT**   <= '0';
23. **LONG**    <= '0';
24. SEL     <= "0000";
25. SST0    <= '0';
26. SSET\_PC <= '0';
28. IF (CLR = '0') THEN
29. ST0 <= '0';
30. ELSE
31. IF (T3'EVENT AND T3 = '0' AND SST0 = '1' ) THEN --在T3下降沿时若SST0 = 1则翻转ST0
32. ST0 <= NOT ST0;
33. END IF;
34. CASE SW IS
35. WHEN "000" =>     --以初始PC为0的方式顺序执行指令
36. CASE IR IS
37. WHEN "0000" => --NOP
38. LIR <= W1;
39. PCINC <= W1;
40. **SHORT** <= W1;
42. WHEN "0001" => --ADD
43. S <= "1001";
44. CIN <= W1;
45. ABUS <= W1;
46. DRW <= W1;
47. LDC <= W1;
48. LDZ <= W1;
49. ………………
50. WHEN OTHERS =>    --未知指令当作NOP处理，跳至下一条
51. LIR <= W1;
52. PCINC <= W1;
53. **SHORT** <= W1;
54. END CASE;
55. WHEN "001" =>   --写寄存器
56. SELCTL <= W1;
57. **SHORT** <= W1;
58. SBUS <= W1;
59. STOP <= W1;
60. SST0 <= W1 AND (NOT ST0);
61. LAR <= W1 AND (NOT ST0);
62. ARINC <= W1 AND ST0;
63. MEMW <= W1 AND ST0;
65. WHEN "010" =>   --读存储器
66. SELCTL <= W1;
67. **SHORT** <= W1;
68. SBUS <= W1 AND (NOT ST0);
69. MBUS <= W1 AND ST0;
70. STOP <= W1;
71. SST0 <= W1 AND (NOT ST0);
72. LAR <= W1 AND (NOT ST0);
73. ARINC <= W1 AND ST0;
75. WHEN "011" =>   --写寄存器
76. SELCTL <= '1';
77. SEL(0) <= W1 OR W2;
78. STOP <= W1 OR W2;
79. SEL(3) <= W2;
80. SEL(1) <= W2;
82. WHEN "100" =>   --写存储器
83. SELCTL <= '1';
84. SST0 <= W2 AND (NOT ST0);
85. SBUS <= W1 OR W2;
86. STOP <= W1 OR W2;
87. DRW <= W1 OR W2;
88. SEL(3) <= (ST0 AND W1) OR (ST0 AND W2);
89. SEL(2) <= W2;
90. SEL(1) <= ((NOT ST0) AND W1) OR (ST0 AND W2);
91. SEL(0) <= W1;
92. WHEN "101" =>               --任意指定初始PC的执行方式
93. IF (SET\_PC = '0') THEN
94. SSET\_PC <= W1;   --设定PC初始后，指令执行过程中不再可以设定
95. LPC <= W1;      --LPC打开，使数据通路上的值可以打入PC寄存器
96. SBUS <= W1;     --SBUS打开，使数据开关的值可以打入数据通路
97. **SHORT** <= W1;    --设定PC只需要一拍，因此此时置**SHORT**为1
98. STOP <= W1;
99. ELSE
100. CASE IR IS
101. WHEN "0000" => --NOP
102. LIR <= W1;
103. PCINC <= W1;
104. **SHORT** <= W1;
106. …………
108. WHEN OTHERS =>      --未知指令当作NOP处理，跳至下一条
109. LIR <= W1;
110. PCINC <= W1;
111. **SHORT** <= W1;
112. END CASE;
113. END IF;
115. WHEN OTHERS => NULL;
116. END CASE;
117. END IF;
118. END PROCESS;
119. END ARC;

### 6.4 模拟与仿真

在quartus II上进行仿真，设置好T3、W1、W2的节拍信号，进行模拟仿真。

以写寄存器为例，将SWCBA设置为100（写寄存器），观察仿真结果：



DRW=1，SBUS=1，SELECT=1，STOP=1，SEL对应相应的写入寄存器序号，SEL=1001时写入R2，SEL=1110时写入R3。

对其他类型操作也进行相应的仿真，得到的结果与流程图中的预计结果相符。

### 6.5 调试过程

#### 6.5.1 调试程序

基本指令：

|  |  |  |  |
| --- | --- | --- | --- |
| 地址 | 程序指令 | 机器码 | 16进制 |
| 00H | LD R0,[R2] | 0101 0010 | 52 |
| 01H | INC R2 | 0100 1000 | 48 |
| 02H | LD R1,[R2] | 0101 0110 | 56 |
| 03H | ADD R0,R1 | 0001 0001 | 11 |
| 04H | JC 06H | 0111 0001 | 71 |
| 05H | AND R1,R0 | 0011 0100 | 34 |
| 06H | SUB R0,R2 | 0010 0010 | 22 |
| 07H | INC R1 | 0100 0100 | 44 |
| 08H | STA R0,[R1] | 0110 0100 | 64 |
| 09H | INC R3 | 0100 1100 | 4C |
| 0AH | JZ 0DH | 1000 0010 | 82 |
| 0BH | LD R2,[R3] | 0101 1011 | 5B |
| 0CH | JMP [R2] | 1001 1000 | 98 |
| 03H | ADD R0,R1 | 0001 0001 | 11 |
| 04H | JC 06H | 0111 0001 | 71 |
| 06H | SUB R0,R2 | 0010 0010 | 22 |
| 07H | INC R1 | 0100 0100 | 44 |
| 08H | STA R0,[R1] | 0110 0100 | 64 |
| 09H | INC R3 | 0100 1100 | 4C |
| 0AH | JZ 0DH | 1000 0010 | 82 |
| 0BH | LD R2,[R3] | 0101 1011 | 5B |
| 0CH | JMP [R2] | 1001 1000 | 98 |
| 0DH | INC R3 | 0100 1100 | 4C |
| 0EH | INC R3 | 0100 1100 | 4C |
| 0FH | SUB R0,R2 | 0010 0010 | 22 |
| 10H | LD R2,[R0] | 0101 1000 | 58 |
| 11H | ADD R3,R2 | 0001 1110 | 1E |
| 12H | LD R3,[R3] | 0101 1111 | 5F |
| 13H | OUT R0 | 1010 0000 | A0 |
| 14H | STP | 1110 0000 | E0 |

拓展指令：

|  |  |  |  |
| --- | --- | --- | --- |
| 15 | XOR R0,R2 | 1011 0010 | B2 |
| 16 | INC R1 | 0100 0100 | 44 |
| 17 | ST R0,[R1] | 0110 0100 | 64 |
| 18 | NOTAB R0,R2 | 1100 0010 | C2 |
| 19 | INC R1 | 0100 0100 | 44 |
| 1A | ST R0,[R1] | 0110 0100 | 64 |
| 1B | NOTA+B R0,R2 | 1101 0010 | D2 |
| 1C | INC R1 | 0100 0100 | 44 |
| 1D | ST R0,[R1] | 0110 0100 | 64 |
| 1E | STP | 1110 0000 | E0 |

#### 6.5.2 调试结果

将测试指令写入TEC-8后，按顺序执行，以下记录了程序执行顺序：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 寄存器初值 | R2=60H | R3=0FDH |  |  |  |
|  | 存储器初值 | [60H]=67H | [61H]=80H | [62H]=0FDH | [80H]=60 | [0FEH]=03H |
|  |  |  |  |  |  | [0FFH]=03H |
| 地址 | 程序指令 | R0 | R1 | R2 | R3 | 存储器 |
| 00H | LD R0,[R2] | 67 | 0 | 60 | FD |  |
| 01H | INC R2 | 67 | 0 | 61 | FD |  |
| 02H | LD R1,[R2] | 67 | 80 | 61 | FD |  |
| 03H | ADD R0,R1 | E7 | 80 | 61 | FD |  |
| 04H | JC 06H | C=0 |  |  |  |  |
| 05H | AND R1,R0 | E7 | 80 | 61 | FD |  |
| 06H | SUB R0,R2 | 86 | 80 | 61 | FD |  |
| 07H | INC R1 | 86 | 81 | 61 | FD |  |
| 08H | STA R0,[R1] | 86 | 81 | 61 | FD | [81]=86 |
| 09H | INC R3 | 86 | 81 | 61 | FE |  |
| 0AH | JZ 0DH | Z=0 |  |  |  |  |
| 0BH | LD R2,[R3] | 86 | 81 | 3 | FE |  |
| 0CH | JMP [R2] | 86 | 81 | 3 | FE |  |
| 03H | ADD R0,R1 | 07 | 81 | 3 | FE |  |
| 04H | JC 06H | C=1 |  |  |  |  |
| 06H | SUB R0,R2 | 04 | 81 | 3 | FE |  |
| 07H | INC R1 | 04 | 82 | 3 | FE |  |
| 08H | STA R0,[R1] | 04 | 82 | 3 | FE | [82]=4 |
| 09H | INC R3 | 04 | 82 | 3 | FF |  |
| 0AH | JZ 0DH | Z=0 |  |  |  |  |
| 0BH | LD R2,[R3] | 04 | 82 | 3 | FF |  |
| 0CH | JMP [R2] | 04 | 82 | 3 | FF |  |
| 03H | ADD R0,R1 | 86 | 82 | 3 | FF |  |
| 04H | JC 06H | C=0 |  |  |  |  |
| 05H | AND R1,R0 | 86 | 82 | 3 | FF |  |
| 06H | SUB R0,R2 | 83 | 82 | 3 | FF |  |
| 07H | INC R1 | 83 | 83 | 3 | FF |  |
| 08H | STA R0,[R1] | 83 | 83 | 3 | FF | [83]=83 |
| 09H | INC R3 | 83 | 83 | 3 | 0 |  |
| 0AH | JZ 0DH | Z=1 |  |  |  |  |
| 0DH | INC R3 | 83 | 83 | 3 | 1 |  |
| 0EH | INC R3 | 83 | 83 | 3 | 2 |  |
| 0FH | SUB R0,R2 | 80 | 83 | 3 | 2 |  |
| 10H | LD R2,[R0] | 80 | 83 | 60 | 2 |  |
| 11H | ADD R3,R2 | 80 | 83 | 60 | 62 |  |
| 12H | LD R3,[R3] | 80 | 83 | 60 | FD |  |
| 13H | OUT R0 |  |  |  |  |  |
| 14H | STP |  |  |  |  |  |
| 寄存器检测 | R0=80H | R1=83H | R2=60H | R3=FDH |  |  |
| 存储器检测 | [81H]=86H | [82H]=04H | [83H]=83H |  |  |  |

测试结果与预期一致。

## 7 遇到的问题

### 胡天翼

我在一开始编写任意PC时，思路是这样的：添加控制分支信号SET\_PC，SET\_PC若为0，则置PC，同时将SET\_PC置为1，则以后之后的执行过程中不再会置PC。代码如下：

1. IF (SET\_PC = '0') THEN
2. SET\_PC <= W1;
3. LPC <= W1;
4. SBUS <= W1;
5. **SHORT** <= W1;
6. STOP <= W1;

但将代码烧入TEC-8后，我发现运行结果与预期不符，LPC信号从未亮起，说明这个分支一直都没有进入过。经过反复排查，我猜测是一旦进入这个分支，更改SET\_PC后，引起相关信号变化，立刻进入了新的进程，此时SET\_PC为1，不进入置PC分支。换句话说，SET\_PC翻转速度太快，来不及写入PC就已经翻转了。为了解决这个问题我们需要延迟一拍翻转SET\_PC，我参考了SST0与ST0的实现方案，多加一个SSET\_PC信号，实现代码如下：

1. IF (SET\_PC = '0') THEN
2. SSET\_PC <= W1;
3. LPC <= W1;
4. SBUS <= W1;
5. **SHORT** <= W1;
6. STOP <= W1;

在置PC的同时，不立刻翻转SET\_PC，而置信号SSET\_PC为1，在下个T3下降沿时检测SSET\_PC，若为1则翻转SET\_PC，由此到达延迟翻转SET\_PC的效果，增添的代码如下：

1. PROCESS (CLR, T3)
2. BEGIN
3. IF (CLR = '0') THEN
4. SET\_PC <= '0';
5. ELSE
6. IF (T3'EVENT AND T3 = '0' AND SSET\_PC = '1' ) THEN
7. SET\_PC <= '1';
8. END IF;
9. END IF;
10. END PROCESS;

### 马锐文

（1）刚开始准备做流水型硬布线的时候，不清楚流水的意思，后来跟小组成员讨论后明白是取指和执行合并在一个周期里。

（2）在去实验室上机后，由于对tec-8机器的不熟悉，导致刚开始的操作会有很多基础错误，后来逐渐熟悉机器的操作后，就熟练了很多，并且由于硬件本身会有一定未知的错误，所以在选择机器的时候，也出现了一些错误。

（3）上机后也遇到了吞指令的问题，通过不断的更改测试程序，逐渐锁定一些会吞指令的操作，最开始尝试加nop操作，发现可以，后又尝试了更改代码的问题，最终解决。

### 张林羽觐

上机调试遇到了扩展指令无法被识别的情况，发现是开关置于微程序而非控制器的位置。

发现指示灯的状态与流程图上相异，原因是针脚对应关系设置不正确。

## 8 实验日志

### 8月3日 胡天翼记录

这是小组成员第一次正式的讨论，在QQ群内进行。我们三个人找出了介绍课与答疑课的录屏，一同观看，对课程设计的要点进行了总结与归纳。经过团队成员仔细讨论非流水与流水设计的流程图的区别，我们认为流水硬布线设计是在硬布线设计的基础上加以改进，并且改动的幅度并不大，我们可以实现。因此决定将课设设计的题目定为流水硬布线设计。

首先，我们决定先去理解硬布线设计的流程图中每个信号的含义，结合理论课学习的CPU组成知识，理解不同信号在TEC-8通路中所起的作用，以及额外的辅助信号ST0、SST0等。

### 8月7日 胡天翼记录

小组成员基本搞清了不同信号的作用与流水硬布线的整个流程，于是开始分工设计微操作时间表。张林羽觐负责控制台操作的信号，我和马锐文负责所有指令的信号。

### 8月15日 马锐文记录

小组成员按照分工完成操作时间表，然后统一输入输出命名，根据流程图和微操作时间表编写对应的指令的代码。

### 8月21日 马锐文记录

按照约定，小组成员于8.21日按时上交各自分工的代码，胡天翼负责统一代码和一些格式和错误的修改。

### 8月31日 胡天翼记录

终于返校了，一回到本部我们就立刻前往实验室进行尝试。不过，一开始并不顺利。在一开始，所有的控制台操作都无法正常进行。经过观察试验台的信号，我们发现select灯一直都没有亮，在按下CLR时，select灯反而会亮。经过仔细研究代码，我们发现搞错了按下CLR的判断。事实上，按下CLR时，CLR是低电平，不按时才是高电平；而我们原来将按下CLR的判断写成（CLR = ‘1’），因此不按CLR时反倒是一直处于清零状态，因而无法进行任何操作。解决这个问题后，写寄存器、读寄存器等操作可以正常进行了。

### 9月1日 张林羽觐记录

在单拍调试样例程序时，小组遇到了吞指令的问题，初步解决方法是在被跳过的指令处增加流水线停顿(气泡)，而后，我们修改了控制器程序的顺序与结构，成功解决了吞指令的问题。

### 9月3日 张林羽觐记录

在增加扩展指令后，吞指令的情况又出现了，于是我们选择修改样例程序上未使用的指令操作，经过了一遍又一遍的调试，准确无误地运行完样例程序了。

### 9月5日 胡天翼记录

解决了吞指令的问题后，加入扩展指令后，我开始思考如何加入任意PC功能。我注意到在流程图中的写寄存器、写存储器和读存储器中用上了分支信号ST0，可以进入不同的分支。这不正是我需要的吗！第一次进入指令执行时，需要设定PC的初始值；而后续进入指令执行，便不再经历设定PC的阶段，这恰好就是两条分支。于是我仿照ST0与SST0，设置了控制分支的SET\_PC与控制SET\_PC翻转的SSET\_PC，从而实现了任意PC。

### 9月7日 张林羽觐记录

我们进行了最后的测试，在更换机器后也能正常运行样例程序。

## 9 实验总结

### 胡天翼

在本次流水硬布线控制器的设计实验中，我结合了《计算机组成原理》理论课学习的CPU组成、TEC-8组成结构的基础知识和过去学习的VHDL语言编程知识，成功地完成了一次用高级语言对底层系统的模拟实践。实验过程中，我将理论知识应用于实践，得到了进一步的掌握与巩固。  
 在本次实验中，我们完成了一个具有流水功能得到控制器。通过这次实验练习，我对CPU中控制器的工作过程。一开始，我对如何构建一个完整的控制器毫无头绪，通过反复阅读PPT、实验指导书和查阅相关资料，理解控制器的工作过程，我终于对如何构建控制器有了初步的认识。在编写代码的过程中，我也因忘记了VHDL语言的部分基础知识，导致代码编写混乱，出现未知的BUG。通过请教同学老师，反复调试程序，我最终解决了若干问题，正确地完成了功能的实现。在调试任意PC的过程中，我因TEC-8呈现的信号与预期不符感到困惑。仔细观察TEC-8上出现的所有信号，分析控制器代码编写中出现的问题，我最终明白信号逻辑中存在的缺陷，调整代码后最终发出了正确的信号。  
 这次实验我总共花了将近两周的时间，虽然过程是艰辛的，但是结果确是让人回味的。最后，衷心地感谢所有给予我帮助的老师和同学们！

### 马锐文

每一次课程设计度让我学到了在平时课堂不可能学到的东西。所以我对每一次课程设计的机会都非常珍惜。通过这次的小组课设作业

（1）我对计算机组成原理的控制器的具体内容有了更深刻的理解，也更清楚了控制器的编码实现。

（2）体会到了尝试的重要，空想是没用的，需要实地尝试后才能更清楚具体的操作和实现是什么。比如遇到吞指令的时候，可以去修改测试程序来锁定错误的点，而更好的进行修改

（3）最好在做课设的过程中能够有记录的习惯，这样在写实验报告时能够比较完整的回忆起中间遇到的各种问题。并且在小组合作的时候，有记录的习惯的话，会更清楚的进行实验，而不是毫无头绪的乱做

### 张林羽觐

通过这次课程设计，我了解了如何在TEC-8真机上进行程序的调试，相较模拟器上的调试，我们需要进行繁琐地写入存储器/寄存器，对于计算机组成原理以及程序流水线也有进一步的理解。

## 10 附录

完整代码：

1. LIBRARY IEEE;
2. USE IEEE.STD\_LOGIC\_1164.ALL;
3. USE IEEE.STD\_LOGIC\_UNSIGNED.ALL;
5. ENTITY CPU IS
6. PORT (
7. --输入信号
8. CLR,T3,C,Z  : IN STD\_LOGIC;
9. W1,W2,W3    : IN STD\_LOGIC;
10. SW          : IN STD\_LOGIC\_VECTOR(2 DOWNTO 0);
11. IR          : IN STD\_LOGIC\_VECTOR(7 DOWNTO 4);

14. --输出信号
15. DRW     : OUT STD\_LOGIC;
16. PCINC   : OUT STD\_LOGIC;
17. LPC     : OUT STD\_LOGIC;
18. LAR     : OUT STD\_LOGIC;
19. PCADD   : OUT STD\_LOGIC;
20. ARINC   : OUT STD\_LOGIC;
21. SELCTL  : OUT STD\_LOGIC;
22. MEMW    : OUT STD\_LOGIC;
23. STOP    : OUT STD\_LOGIC;
24. LIR     : OUT STD\_LOGIC;
25. LDZ     : OUT STD\_LOGIC;
26. LDC     : OUT STD\_LOGIC;
27. CIN     : OUT STD\_LOGIC;
28. S       : OUT STD\_LOGIC\_VECTOR(3 DOWNTO 0);
29. M       : OUT STD\_LOGIC;
30. ABUS    : OUT STD\_LOGIC;
31. SBUS    : OUT STD\_LOGIC;
32. MBUS    : OUT STD\_LOGIC;
33. **SHORT**   : OUT STD\_LOGIC;
34. **LONG**    : OUT STD\_LOGIC;
35. SEL     : OUT STD\_LOGIC\_VECTOR(3 DOWNTO 0)
36. );
37. END CPU;
39. ARCHITECTURE ARC OF CPU IS
40. SIGNAL ST0, SST0 : STD\_LOGIC;   --ST0是分支信号,SST0是控制ST0翻转的信号
41. SIGNAL SET\_PC, SSET\_PC    : STD\_LOGIC;   --SET\_PC判断当前是否还未执行过指令，可以设定执行指令的初始位置
42. BEGIN
43. PROCESS (CLR, T3)
44. BEGIN
45. IF (CLR = '0') THEN
46. SET\_PC <= '0';
47. ELSE
48. IF (T3'EVENT AND T3 = '0' AND SSET\_PC = '1' ) THEN
49. SET\_PC <= '1';
50. END IF;
51. END IF;
52. END PROCESS;
54. PROCESS (SW, IR, W1, W2, W3, C, Z, CLR, T3)
55. BEGIN
56. --计算信号前，先将所有信号初始化为0
57. DRW     <= '0';
58. PCINC   <= '0';
59. LPC     <= '0';
60. LAR     <= '0';
61. PCADD   <= '0';
62. ARINC   <= '0';
63. SELCTL  <= '0';
64. MEMW    <= '0';
65. STOP    <= '0';
66. LIR     <= '0';
67. LDZ     <= '0';
68. LDC     <= '0';
69. CIN     <= '0';
70. S       <= "0000";
71. M       <= '0';
72. ABUS    <= '0';
73. SBUS    <= '0';
74. MBUS    <= '0';
75. **SHORT**   <= '0';
76. **LONG**    <= '0';
77. SEL     <= "0000";
78. SST0    <= '0';
79. SSET\_PC <= '0';
81. IF (CLR = '0') THEN
82. ST0 <= '0';
83. ELSE
84. IF (T3'EVENT AND T3 = '0' AND SST0 = '1' ) THEN --在T3下降沿时若SST0 = 1则翻转ST0
85. ST0 <= NOT ST0;
86. END IF;
87. CASE SW IS
88. WHEN "000" =>                        --以初始PC为0的方式顺序执行指令
89. CASE IR IS
90. WHEN "0000" => --NOP
91. LIR <= W1;
92. PCINC <= W1;
93. **SHORT** <= W1;
95. WHEN "0001" => --ADD
96. LIR <= W1;
97. PCINC <= W1;
98. **SHORT** <= W1;
99. S <= "1001";
100. CIN <= W1;
101. ABUS <= W1;
102. DRW <= W1;
103. LDC <= W1;
104. LDZ <= W1;


108. WHEN "0010" => --SUB
109. LIR <= W1;
110. PCINC <= W1;
111. **SHORT** <= W1;
112. S <= "0110";
113. ABUS <= W1;
114. DRW <= W1;
115. LDC <= W1;
116. LDZ <= W1;
118. WHEN "0011" => --AND
119. LIR <= W1;
120. PCINC <= W1;
121. **SHORT** <= W1;
122. S <= "1011";
123. M <= W1;
124. ABUS <= W1;
125. DRW <= W1;
126. LDZ <= W1;
128. WHEN "0100" => --INC
129. LIR <= W1;
130. PCINC <= W1;
131. **SHORT** <= W1;
132. S <= "0000";
133. ABUS <= W1;
134. DRW <= W1;
135. LDC <= W1;
136. LDZ <= W1;


140. WHEN "0101" => --LD
141. LIR <= W2;
142. PCINC <= W2;
143. S <= "1010";
144. M <= W1;
145. ABUS <= W1;
146. LAR <= W1;
147. MBUS <= W2;
148. DRW <= W2;
150. WHEN "0110" => --ST
151. LIR <= W2;
152. PCINC <= W2;
153. M <= W1 or W2;
154. S(3) <= '1';
155. S(2) <= W1;
156. S(1) <= '1';
157. S(0) <= W1;
158. ABUS <= W1 or W2;
159. LAR <= W1;
160. MEMW <= W2;
162. WHEN "0111" => --JC
163. LIR <= (W1 and (not C)) or (W2 and C);
164. PCINC <= (W1 and (not C)) or (W2 and C);
165. PCADD <= C and W1;
166. **SHORT** <= W1 and (not C);
168. WHEN "1000" => --JZ
169. LIR <= (W1 and (not Z)) or (W2 and Z);
170. PCINC <= (W1 and (not Z)) or (W2 and Z);
171. PCADD <= Z and W1;
172. **SHORT** <= W1 and (not Z);
173. WHEN "1001" => --JMP
174. LIR <= W2;
175. PCINC <= W2;
176. M <= W1;
177. S <= "1111";
178. ABUS <= W1;
179. LPC <= W1;
181. WHEN "1010" => --OUT
182. M <= W1;
183. S <= "1010";
184. ABUS <= W1;
185. LIR <= W1;
186. PCINC <= W1;
187. **SHORT** <= W1;
189. WHEN "1011" => --XOR
190. LIR <= W1;
191. PCINC <= W1;
192. **SHORT** <= W1;
193. M <= W1;
194. S <= "0110";
195. ABUS <= W1;
196. LDZ <= W1;
197. DRW <= W1;
199. WHEN "1100" => --OR
200. LIR <= W1;
201. PCINC <= W1;
202. **SHORT** <= W1;
203. M <= W1;
204. S <= "1110";
205. ABUS <= W1;
206. LDZ <= W1;
207. DRW <= W1;
208. when "1011" => --XOR
209. LIR <= W1;
210. PCINC <= W1;
211. **SHORT** <= W1;
212. M <= W1;
213. S <= "0110";
214. ABUS <= W1;
215. LDZ <= W1;
216. DRW <= W1;
218. when "1100" => --NOT(AB)
219. LIR <= W1;
220. PCINC <= W1;
221. **SHORT** <= W1;
222. M <= W1;
223. S <= "0100";
224. ABUS <= W1;
225. LDZ <= W1;
226. DRW <= W1;
228. when "1101" => --NOT(A+B)
229. LIR <= W1;
230. PCINC <= W1;
231. **SHORT** <= W1;
232. M <= W1;
233. S <= "0001";
234. ABUS <= W1;
235. LDZ <= W1;
236. DRW <= W1;
238. WHEN "1110" => --STP
239. STOP <= W1;
241. WHEN OTHERS =>      --未知指令当作NOP处理，跳至下一条
242. LIR <= W1;
243. PCINC <= W1;
244. **SHORT** <= W1;
245. END CASE;
246. WHEN "001" =>
247. SELCTL <= W1;
248. **SHORT** <= W1;
249. SBUS <= W1;
250. STOP <= W1;
251. SST0 <= W1 AND (NOT ST0);
252. LAR <= W1 AND (NOT ST0);
253. ARINC <= W1 AND ST0;
254. MEMW <= W1 AND ST0;
256. WHEN "010" =>
257. SELCTL <= W1;
258. **SHORT** <= W1;
259. SBUS <= W1 AND (NOT ST0);
260. MBUS <= W1 AND ST0;
261. STOP <= W1;
262. SST0 <= W1 AND (NOT ST0);
263. LAR <= W1 AND (NOT ST0);
264. ARINC <= W1 AND ST0;
266. WHEN "011" =>
267. SELCTL <= '1';
268. SEL(0) <= W1 OR W2;
269. STOP <= W1 OR W2;
270. SEL(3) <= W2;
271. SEL(1) <= W2;
273. WHEN "100" =>
274. SELCTL <= '1';
275. SST0 <= W2 AND (NOT ST0);
276. SBUS <= W1 OR W2;
277. STOP <= W1 OR W2;
278. DRW <= W1 OR W2;
279. SEL(3) <= (ST0 AND W1) OR (ST0 AND W2);
280. SEL(2) <= W2;
281. SEL(1) <= ((NOT ST0) AND W1) OR (ST0 AND W2);
282. SEL(0) <= W1;
283. WHEN "101" =>               --任意指定初始PC的执行方式
284. IF (SET\_PC = '0') THEN
285. SSET\_PC <= W1;   --设定PC初始后，指令执行过程中不再可以设定
286. LPC <= W1;      --LPC打开，使数据通路上的值可以打入PC寄存器
287. SBUS <= W1;     --SBUS打开，使数据开关的值可以打入数据通路
288. **SHORT** <= W1;    --设定PC只需要一拍，因此此时置**SHORT**为1
289. STOP <= W1;
290. ELSE
291. CASE IR IS
292. WHEN "0000" => --NOP
293. LIR <= W1;
294. PCINC <= W1;
295. **SHORT** <= W1;
297. WHEN "0001" => --ADD
298. LIR <= W1;
299. PCINC <= W1;
300. **SHORT** <= W1;
301. S <= "1001";
302. CIN <= W1;
303. ABUS <= W1;
304. DRW <= W1;
305. LDC <= W1;
306. LDZ <= W1;
308. WHEN "0010" => --SUB
309. LIR <= W1;
310. PCINC <= W1;
311. **SHORT** <= W1;
312. S <= "0110";
313. ABUS <= W1;
314. DRW <= W1;
315. LDC <= W1;
316. LDZ <= W1;
318. WHEN "0011" => --AND
319. LIR <= W1;
320. PCINC <= W1;
321. **SHORT** <= W1;
322. S <= "1011";
323. M <= W1;
324. ABUS <= W1;
325. DRW <= W1;
326. LDZ <= W1;
328. WHEN "0100" => --INC
329. LIR <= W1;
330. PCINC <= W1;
331. **SHORT** <= W1;
332. S <= "0000";
333. ABUS <= W1;
334. DRW <= W1;
335. LDC <= W1;
336. LDZ <= W1;
338. WHEN "0101" => --LD
339. LIR <= W2;
340. PCINC <= W2;
341. S <= "1010";
342. M <= W1;
343. ABUS <= W1;
344. LAR <= W1;
345. MBUS <= W2;
346. DRW <= W2;
348. WHEN "0110" => --ST
349. LIR <= W2;
350. PCINC <= W2;
351. M <= W1 OR W2;
352. S(3) <= '1';
353. S(2) <= W1;
354. S(1) <= '1';
355. S(0) <= W1;
356. ABUS <= W1 OR W2;
357. LAR <= W1;
358. MEMW <= W2;
360. WHEN "0111" => --JC
361. LIR <= (W1 AND (NOT C)) OR (W2 AND C);
362. PCINC <= (W1 AND (NOT C)) OR (W2 AND C);
363. PCADD <= C AND W1;
364. **SHORT** <= W1 AND (NOT C);
366. WHEN "1000" => --JZ
367. LIR <= (W1 AND (NOT Z)) OR (W2 AND Z);
368. PCINC <= (W1 AND (NOT Z)) OR (W2 AND Z);
369. PCADD <= Z AND W1;
370. **SHORT** <= W1 AND (NOT Z);
372. WHEN "1001" => --JMP
373. LIR <= W2;
374. PCINC <= W2;
375. M <= W1;
376. S <= "1111";
377. ABUS <= W1;
378. LPC <= W1;
380. WHEN "1010" => --OUT
381. M <= W1;
382. S <= "1010";
383. ABUS <= W1;
384. LIR <= W1;
385. PCINC <= W1;
386. **SHORT** <= W1;
388. WHEN "1011" => --XOR
389. LIR <= W1;
390. PCINC <= W1;
391. **SHORT** <= W1;
392. M <= W1;
393. S <= "0110";
394. ABUS <= W1;
395. LDZ <= W1;
396. DRW <= W1;
398. WHEN "1100" => --OR
399. LIR <= W1;
400. PCINC <= W1;
401. **SHORT** <= W1;
402. M <= W1;
403. S <= "1110";
404. ABUS <= W1;
405. LDZ <= W1;
406. DRW <= W1;
408. when "1011" => --XOR
409. LIR <= W1;
410. PCINC <= W1;
411. **SHORT** <= W1;
412. M <= W1;
413. S <= "0110";
414. ABUS <= W1;
415. LDZ <= W1;
416. DRW <= W1;
418. when "1100" => --NOT(AB)
419. LIR <= W1;
420. PCINC <= W1;
421. **SHORT** <= W1;
422. M <= W1;
423. S <= "0100";
424. ABUS <= W1;
425. LDZ <= W1;
426. DRW <= W1;
428. when "1101" => --NOT(A+B)
429. LIR <= W1;
430. PCINC <= W1;
431. **SHORT** <= W1;
432. M <= W1;
433. S <= "0001";
434. ABUS <= W1;
435. LDZ <= W1;
436. DRW <= W1;

439. WHEN "1110" => --STP
440. STOP <= W1;
442. WHEN OTHERS =>      --未知指令当作NOP处理，跳至下一条
443. LIR <= W1;
444. PCINC <= W1;
445. **SHORT** <= W1;
446. END CASE;
447. END IF;
449. WHEN OTHERS => NULL;
450. END CASE;
451. END IF;
452. END PROCESS;
453. END ARC;