目录

**第一部分 题目及题目分析3**

1.计算机组成原理课程设计题目3

2.实验目的3

3.题目分析3

**第二部分 实验环境5**

1.实验设备及编码环境5

2.实验硬件环境5

3.实验原指令系统9

**第三部分 团队分工10**

**第四部分 设计思路11**

1.基础硬布线控制器11

2.流水硬布线控制器13

3.流水与非流水区别14

4.中断硬布线控制器15

**第五部分 功能仿真及时序仿真17**

**第六部分 两种不同风格编码的分析18**

**第七部分 问题及解决20**

**第八部分 工作及调试日志21**

**第九部分 调试小结23**

**第十部分 个人总结及收获24**

**附录 VHDL源代码26**

## 第一部分 题目及题目分析

**1. 计算机组成原理课程设计题目**

**（1）基础题目：**

要求：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。

实现一、根据设计方案，在TEC-8上进行组装、调试并运行成功

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至少三条

b.修改PC指针功能（任意指针）

**（2）流水线题目（选做）：**

相较于基础题目增加实现三：实现指令执行过程中的流水功能。

**（3）中断题目（选做）：**

相较于基础题目增加实现三：实现指令执行过程中能够响应中断并且保存现场后进入中断服务程序，执行中断完成后能够返回中断点并且恢复现场。

**注：此处我们小组选择基础+流水线题目，并且实现扩充指令3条（OR XOR DEC），实现了修改PC指针的功能，能够通过功能仿真和时序仿真测试并且发现问题，能够成功运行样例程序并且得到正确结果。最终我们给出了中断的设计方案和实现代码，但并未实际测试。**

**2. 实验目的**

融会贯通计算机组成原理或计算机组成与系统结构课程各章教学内容，通过知识的综合运用，加深对CPU各模块工作原理及相互联系的认识。

掌握硬布线控制器的设计方法。

学习和运用当代EDA设计工具，掌握用EDA设计大规模复杂逻辑电路的方法。

培养科学研究能力，取得设计和调试的实践经验。

**3. 题目分析**

**（1）基础题目分析**

首先基础题目要求设计基于硬布线控制器的顺序模型处理器。我们所使用的实验仪器为TEC-8，其中硬布线控制器由1片EPM7128(U68)组成，产生TEC-8模型机所需的各种信号。那么入口应该是硬布线控制器，我们需要做的工作便是根据逻辑表达式给出不同的信号从而来实现控制其他部件。在这里我们需要使用Quartus Ⅱ将编译完成的控制逻辑程序烧录到该芯片中，将由此程序根据逻辑产生不同的控制信号来完成我们需要的操作。

对于仿真模拟向量测试，我们需要使用Quartus Ⅱ的模拟程序，进行功能仿真和时序仿真，具体步骤是将与理想情况下相符的输入信号设置好之后模拟输出信号来观察VHDL的程序逻辑本事是否存在问题，而功能仿真与时序仿真要分别进行，功能仿真仅验证逻辑正确性，而时序仿真则会考虑硬件芯片等的各种参数，会有时延的考虑，所以会更加精确。

对于附加功能a，在原先指令的基础上扩指至少三条，即当前基础指令并没有达到16条，所以可以交由我们自由编指，编指需要考虑的是整体各个部分的信号，比如运算器的信号或者存储器的信号等。

对于附加功能b，修改PC指针的功能即为改变PC计数器的值的功能，参考做CPU指令执行实验的PC修改方案，即给予SBUS信号，将用户的输入在LPC信号有效时打入PC计数器即可实现。

**（2）流水线题目分析**

流水线题目仅在基础题目的要求上增加指令流水功能，指令流水在课本上包括四个阶段，即取指令（IF）、指令译码（ID）、执行（EX）、结果写回（WB），我们所用的TEC-8模型计算机主时钟MF的频率为1MHz，可以产生3个及以下的机器周期，每个机器周期有三个节拍脉冲，在这里我们采用二级流水，即取指和执行流水。在流水线的设计中，PC的值会永远指向下一条指令的地址，在执行当前指令的最后一个节拍时将PCINC和LIR信号置为高电平，实现在执行当前指令时就把下一条指令取指并且PC+1，但是后面遇到了问题，比如取指慢于PC+1导致吞指令或者指令覆盖，解决办法在这里我们加入空节拍，之后做详细说明。

**（3）中断题目分析**

中断题目仅在基础题目的要求上增加中断的功能，中断的实现在理论上是需要中断地址寄存器IAR来保存当前PC的值，保存之后由用户给予中断服务程序的地址，LPC信号将地址置为新的PC值，程序转到中断服务程序执行，而保存现场（包括保存R0、R1、R2，R3以及 C 和 Z的值）和恢复现场都交给中断服务程序完成。而在我们此次实验中并没有IAR提供给我们保存PC的值，导致需要做一些处理来保存。我们小组给出的设计思路便是将值都存储到存储器中，但是需要由用户提供地址给存储器才能保存起来，同理恢复的时候也需要用户给出之前保存位置的地址，所以很繁琐，最后我们由于时间原因只给出了设计和实验代码，并没有实际测试。

## 第二部分 实验环境

**1. 实验设备及开发环境**

开发环境

实验设备采用TEC-8教学实验平台

编码环境使用Quartus Ⅱ VHDL语言

调试设备

逻辑笔

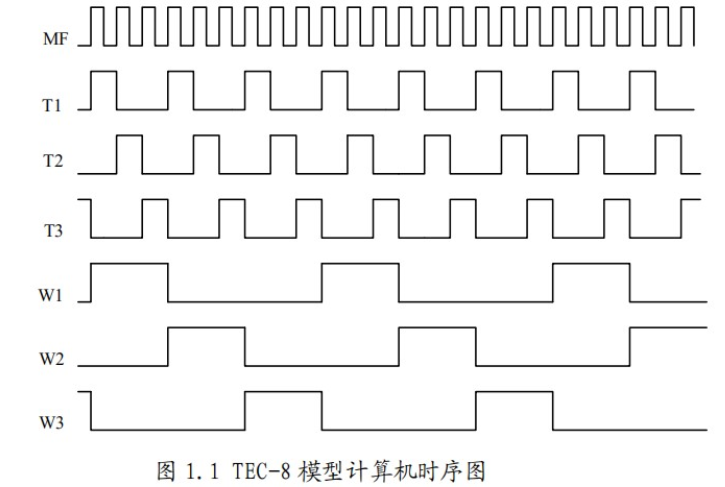
万用表

数字示波器

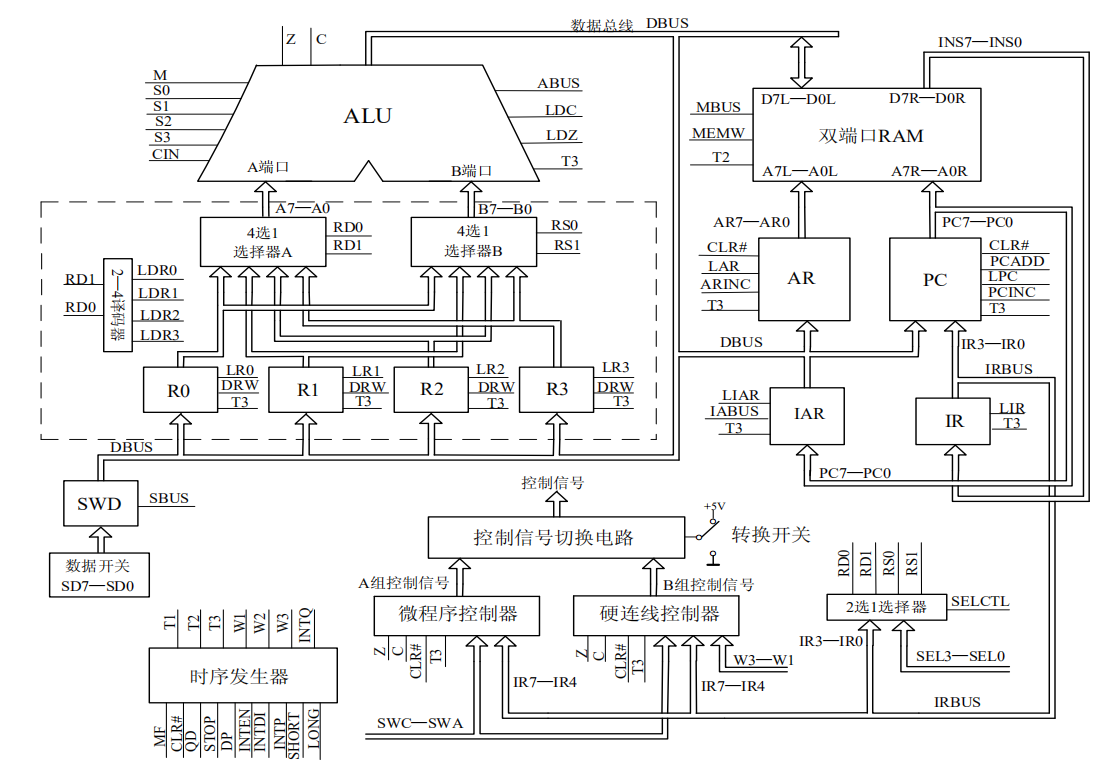
**2. 实验硬件环境**

**（1）模型计算机时序信号**

TEC-8 模型计算机主时钟 MF 的频率为 1MHz，执行一条微指令需要 3个节拍脉T1、T2、T3。TEC-8 模型计算机时序采用不定长机器周期，绝大多数指令采用 2个机器周期 W1、W2，少数指令采用一个机器周期 W1 或者 3个机器周期 W1、W2、W3。

****

**（2）TEC-8 模型计算机组成模块**

****

**时序发生器**产生节拍脉冲 T1、T2、T3，节拍电位 W1、W2、W3，以及中断请求信号 ITNQ。主时钟 MF 采用石英晶体振荡器产生的 1MHz 时钟信号。T1、T2、T3 的脉宽为 1 微秒。一个机器周期包含一组 T1、T2、T3。

**算术逻辑单元ALU** 对 A 端口的 8位数和 B 端口的 8 位数进行加、减、与、或和数据传送 5 种运算，产生 8 位数据结果、进位标志 C 和结果为 0 标志 Z。当信号 SBUS 为 1 时，将运算的数据结果送数据总线 DBUS。

**双端口寄存器组**向 ALU 提供两个运算操作数 A 和 B，保存运算结果。里面包含4个8位寄存器 R0、R1、R2、R3，4 选 1 选择器 A，4 选 1 选择器 B，2-4 译码器。4个寄存器通过 4 选 1 选择器向 ALU 的 A 端口提供 A 操作数，通过 4 选 1 选择器 B 向ALU 的 B 端口提供 B 操作数，2-4 译码器产生信号 LR0、LR1、LR2 和 LR3，选择保存运算数据结果的寄存器。

**8 位数据开关 SD7～SD0** 是双位开关，用于编制程序并把程序放入存储器，设置寄存器 R3～R0 的值。通过拨动数据开关 SD7～ SD0 得到的程序或者数据通过 SWD 送往数据总线 DBUS。

**双端口 RAM**是一种 2 个端口可同时进行读、写的存储器，2个端口各有独立的存储器地址、数据总线和读、写控制信号。在 TEC-8 中，双端口存储器的左端口是个真正的读、写端口，用于程序的初始装入操作，从存储器中取数到数据总线 DBUS，将数据总线DBUS 上的数写入存储器；右端口设置成只读方式，从右端口读出的指令INS7～INS0被送往指令寄存器 IR。

**程序计数器 PC** 由 2 片 GAL22V10 和 1片74244 组成，向双端口 RAM 的左端口提供存储器地址PC7～PC0，程序计数器 PC 具有 PC 复位功能，从数据总线 DBUS 上装入初始 PC 功能，PC 加 1 功能，PC 和转移偏量相加功能。

**地址寄存器 AR** 由 1 片GAL22V10 组成，向双端口RAM 的左端口提供存储器地址AR7～AR0。它具有从数据总线 DBUS 上装入初始 AR 功能和 AR 加 1 功能

**指令寄存器IR**的输出 IR7～IR4送往硬连线控制器、微程序控制器，IR3～IR0 送往 2 选 1 选择器。

**2选1选择器**由 1 片 74244 组成，用于在指令中的操作数 IR3～IR0 和控制信号SEL3～ SEL0 之间进行选择，产生目的寄存器编码 RD1、RD0，产生源寄存器编码 RS1、RS0。

**（3）数据流动方式**

**在进行数据运算操作时**，由 RD1、RD0 选中的寄存器通过 4 选 1 选择器 A 送往 ALU 的 A端口，由 RS1、RS0 选中的寄存器通过 4 选 1 选择器 B 送往 ALU 的 B 端口；信号 M、S3、S2、S1、S1 和 S0 决定 ALU 的运算类型，ALU 对 A 端口和 B 端口的两个数连同 CIN 的值进行算数逻辑运算，得到的数据运算结果在信号 ABUS 为 1 时送往数据总线 DBUS；在 T3 的上升沿，数据总线 DBUS 上的数据结果写入由 RD1、RD0 选中的寄存器。

**在寄存器之间进行数据传送操作时**，由 RS1、RS0 选中的寄存器通过 4 选 1 选择器 B 送往 ALU 的 B 端口；ALU 将 B 端口的数在信号 ABUS 为 1 时送往数据总线 DBUS；在 T3 的上升沿将数据总线上的数写入由 RD1、RD0 选中的寄存器。ALU 进行数据传送操作由一组特定的 M、S3、S2、S1、S0、CIN 的值确定。

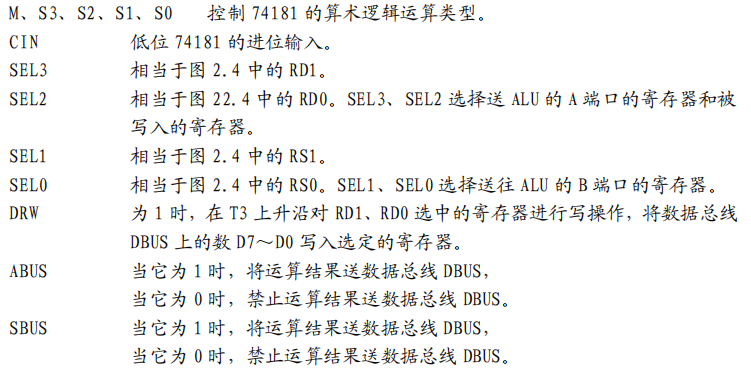
**在进行运算操作时**，由 RS1、RS0 选中的寄存器通过 4 选 1 选择器 B 送往 ALU 的 B 端口；由 RD1、RD0 选中的寄存器通过 4 选 1 选择器 A 送往 ALU 的 A 端口；ALU 对数 A 和 B 进行运算，运算的数据结果在信号 ABUS 为 1 时送往数据总线 DBUS；在 T3 的上升沿将数据总线上的数写入由 RD1、RD0 选中的寄存器。ALU 进行何种运算操作由 M、S3、S2、S1、S0、CIN 的值确定。

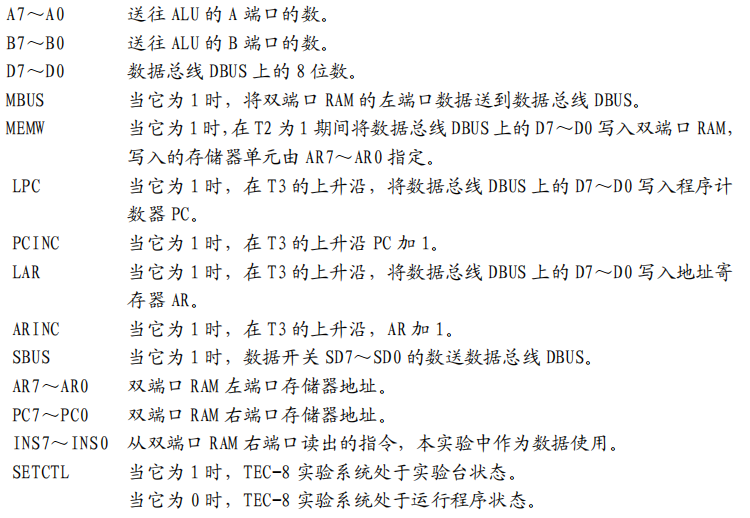
**在从存储器中取数操作中**，由地址 AR7～AR0 指定的存储器单元中的数在信号 MEMW 为0时被读出；在 MBUS 为 1 时送数据总线 DBUS；在 T3 的上升沿写入由 RD1、RD0 选中的寄存器。

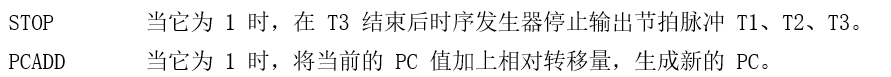
**在写存储器操作中**，由 RS1、RS0 选中的寄存器过 4 选 1 选择器 B 送 ALU 的 B 端口；ALU将 B 端口的数在信号 ABUS 为 1 时送往数据总线 DBUS；在 MEMW 为 1 且 MBUS 为 0 时，通过左端口将数据总线 DBUS 上的数在 T2 为 1 期间写入由 AR7～AR0 指定的存储器单元。在读指令操作时，通过存储器右端口读出由 PC7～PC0 指定的存储器单元的内容送INS7～INS0，当信号 LIR 为 1 时，在 T3 的上升沿写入指令寄存器 IR。

**数据开关 SD7～SD0 上的数在 SBUS为1时送到数据总线DBUS上**，用于给寄存器R0、R1、R2 和 R3，地址寄存器 AR，程序计数器 PC 设臵初值，用于通过存储器左端口向存储器写入测试程序。

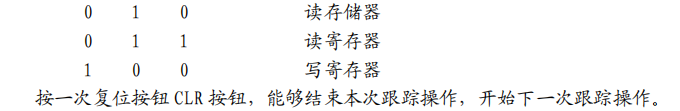
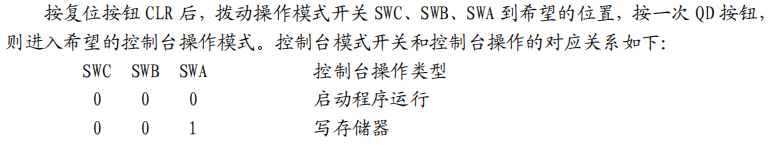
**（4）实验所需要的部分信号**

****

****



**（5）控制台操作**

****

**3. 实验原指令系统**

****

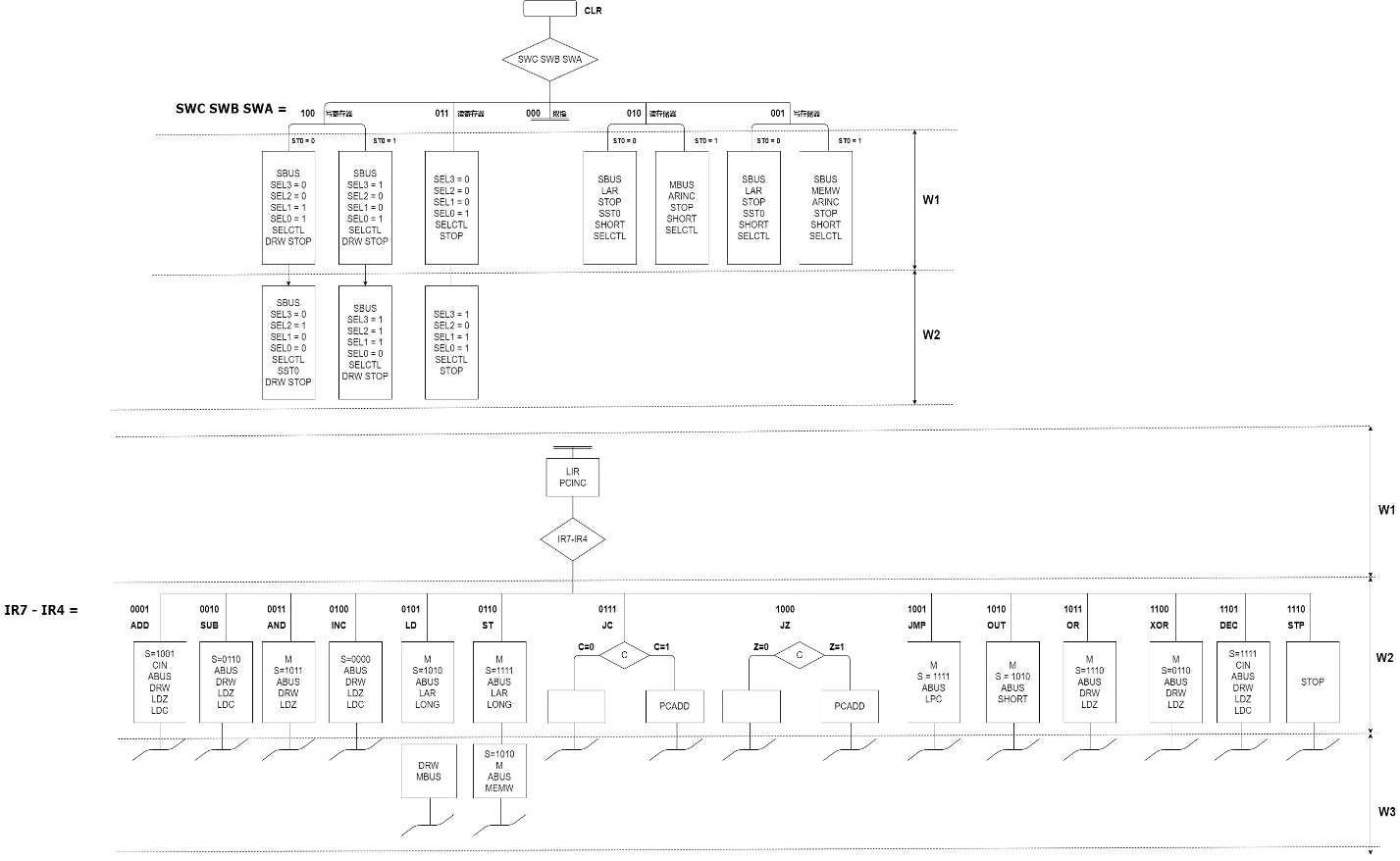
## 第三部分 团队分工

**此处脱敏处理。**

## 第四部分 设计思路

**1. 基础硬布线控制器**

**（1）硬布线控制器流程图**

****

**（2）组合逻辑译码表**

****

**（3）扩展指令集**

****

我们小组扩展三条指令OR XOR DEC分别为或、异或，和自减操作。不涉及存储器的读写操作。

**（4）设计说明**

控制台通过SWC SWB SWA选择功能。

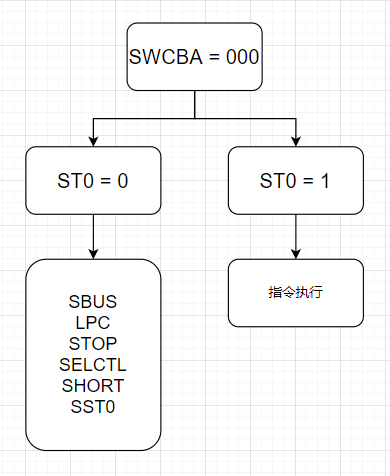
当SWC-SWA 为 100时为写寄存器，需要四个节拍分别写四个寄存器，但是由于只能产生最多3个节拍，所以使用两次W1 W2来写四个寄存器，使用标志位ST0区分当前到底处于第几次二节拍，当ST0 = 0时代表第一个W1和W2，当ST0 = 1时代表第二个W1和W2，而ST0置1的信号为SST0。SST0在第一个W2时为高电平，在W2的T3下降沿时若SST0为高电平则将ST0置1实现了四个节拍写寄存器。但是设计时没有考虑周到，应该在第二个W2处将ST0还原使得能够循环写寄存器。

当SWC-SWA 为 011时为写寄存器，需要两个节拍读四个寄存器，则使用W1和W2即可。

当SWC-SWA 为 010时为读存储器，首先需要一个节拍将用户输入的地址从SBUS送到AR寄存器，之后需要循环将AR计数器加一循环读存储器。此处设计依旧采用ST0作为标志位，当ST0为0时打入地址到AR寄存器，利用SST0在W1的T3下降沿将ST0置为1，之后ST0 = 1即为循环读存储器功能。

当SWC-SWA 为 001时为写存储器，首先需要一个节拍将用户输入的地址从SBUS送到AR寄存器，之后需要循环将AR计数器加一循环写存储器。此处设计依旧采用ST0作为标志位，当ST0为0时打入地址到AR寄存器，利用SST0在W1的T3下降沿将ST0置为1，之后ST0 = 1即为循环写存储器功能。

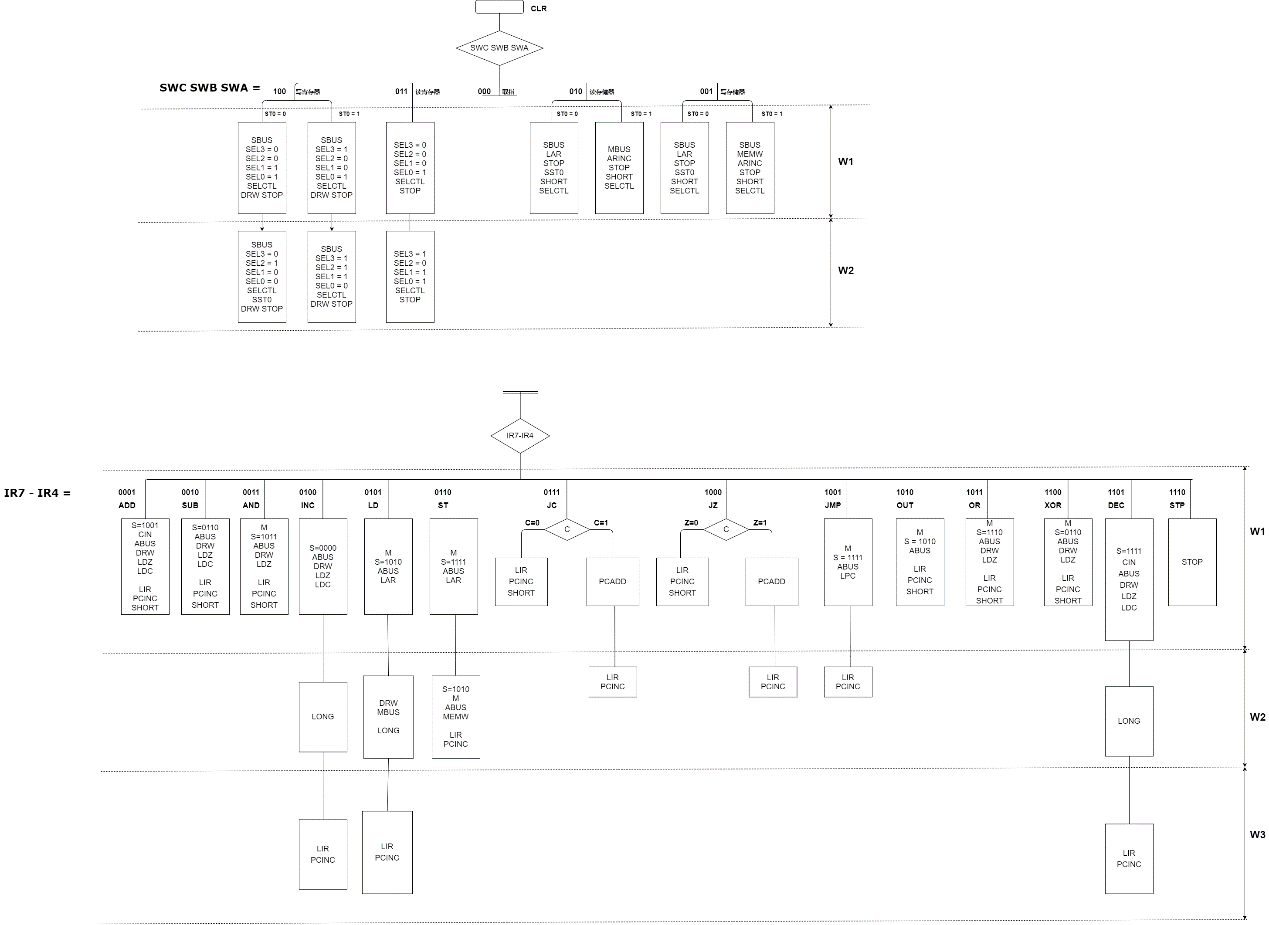
当SWC-SWA 为 000时为取值执行功能，在这里由于添加了修改PC指针的功能，所以设置标志位来区分当前是在执行还是修改PC指针。当ST0 = 0时，为修改PC指针功能，使用SBUS LPC等信号将输入打入PC计数器。当ST0 = 1时则为指令执行功能，此时将对IR7-4进行译码执行指令。



在指令执行过程中，第一个节拍W1用于取指令并且PC计数器+1，若为非流水则PC指向当前指令的地址，在第二个第三个节拍W2W3给出特定的信号执行指令，对于涉及到寄存器读写的指令的执行需要两个节拍W2和W3，在W2节拍给出LONG信号使得时序发生器给出第三个节拍。一般指令只需要一个节拍W2执行指令。

**2. 流水线硬布线控制器**

**（1）流水线硬布线控制器流程图**

****

**（2）组合逻辑译码表**



**注：由于指令集与基础硬布线控制器指令集一致，在这里不做说明。**

**（3）设计说明**

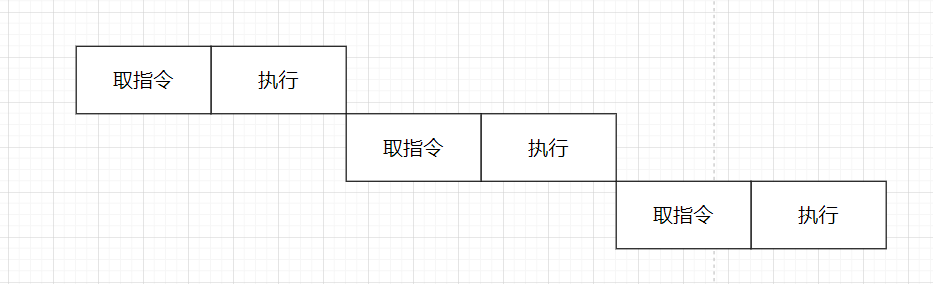
由于指令集与基础硬布线控制器指令集一致，在这里不做说明。

设计总体与硬布线控制器一致，这里指出不同的设计点。

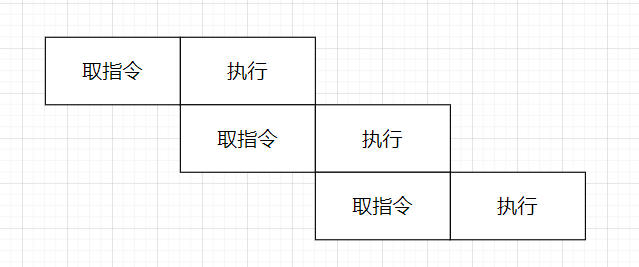
首先流水线要实现流水功能必须要在同一时间处理多项工作，在这里我们设计在执行的最后一个节拍取下一条指令并且完成PC加一的操作。即取指令和执行在时间上重叠，即实现流水。实现流水后不需要单独节拍去取指令，大大节省时间，但是由于实验台硬件运算速度问题，导致部分指令无法实现流水功能，在这里只能取消部分指令的流水来规避错误，这里取消INC、DEC以及LD指令的流水，后面问题与调试部分进行详细说明。

**3. 流水与非流水的区别**

**（1）非流水**

****

**（2）流水**



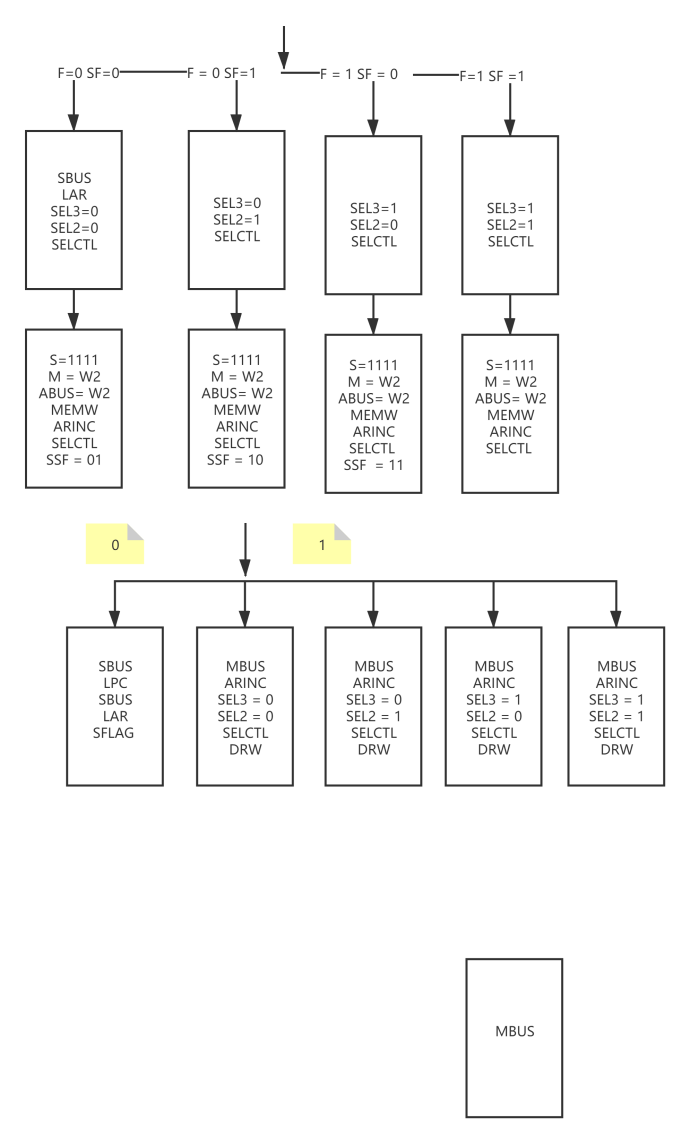
**（3）分析**

从执行时间上可以很清楚的看出，流水线可以在每条指令上节省一个节拍的时间，大大提高了效率，所以流水线是现代CPU的标志。

流水线的方式虽然能缩短程序执行时间，大大提高效率，但是需要考虑资源冲突等问题，或者还需要考虑硬件的问题，有的时候只能取消部分指令的流水来获得稳定的执行。

同时流水与非流水的一个不同点便是PC计数器的值，非流水版本下PC的值始终指向正在执行的指令地址，而在流水线版本PC的值指向下一条要执行指令地址。

**4. 中断硬布线控制器设计（并未实机测试）**



在上机之前，我们已经把中断的原理大致弄清楚了，实现中断分成接收中断信号，保存中断现场和，跳转中断服务程序，执行中断服务程序，还原中断现场几部分。

接收中断信号设一标志位FLAG来检测，如果收到了PULSE信号，这进入保存中断现场程序，并设置开启中断屏蔽。

保存中断现场的思路是将中断前的4个寄存器值、进位标志C、0标志Z保存下来，

但这里遇到第一个问题就是，程序需要将中断现场保存在存储器中，存储在存储器中需要指定一块地址，而程序无法预设保存中断现场的地址，所以在我看来，这地址只能SBUS LAR,手工输入。第二就是，怎么取出寄存器的值，这里用了片选信号SEL0~3,选择指定寄存器，再用S =1111，M=1 ，将寄存器数值输出到DBUS上，再用MEMW，将DBUS的值存在指定地址存储器中。设置两标志位保存C、Z，俩标志位与C、Z异或，即可取出其值。但是本机无法取出PC寄存器的数值，所以在当前程序无法保存PC的数值，思来想去也没想出办法，所以只能采用人工记录的方式。

完成中断现场保存后，进行跳转中断服务程序，手动输入中断向量，再跳转至中断向量处；执行中断程序，按照原程序执行。

还原中断现场，需要在中断程序末尾加入“1111”指令，载入IRET指令时，返回开始中断的地点，由于程序无法预设存储器地址，此时中断现场保存在存储器中，仍需要手动输入中断现场保存地址和中断地址。输入中断地址后，使用LPC指令返回发生中断值的位置。输入中断现场保存地址之后，可用指令MBUS将当前地址数据放上DBUS，再用片选信号选定寄存器，用DRW指令将DBUS内容写入寄存器中，这样寄存器的值就能被还原，可C、Z的还原又犯了难。

PC的取值，C、Z的还原，目前没有找到合适的办法，但是人工输入可以解决，不过需要这三都人工输入的话，那么这实在是一个让人无语的程序，但苦于找不到其他办法，只好作罢。

## 第五部分 功能仿真及时序仿真

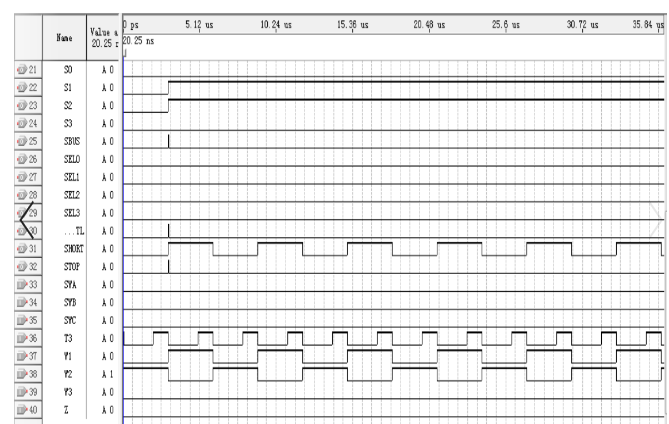
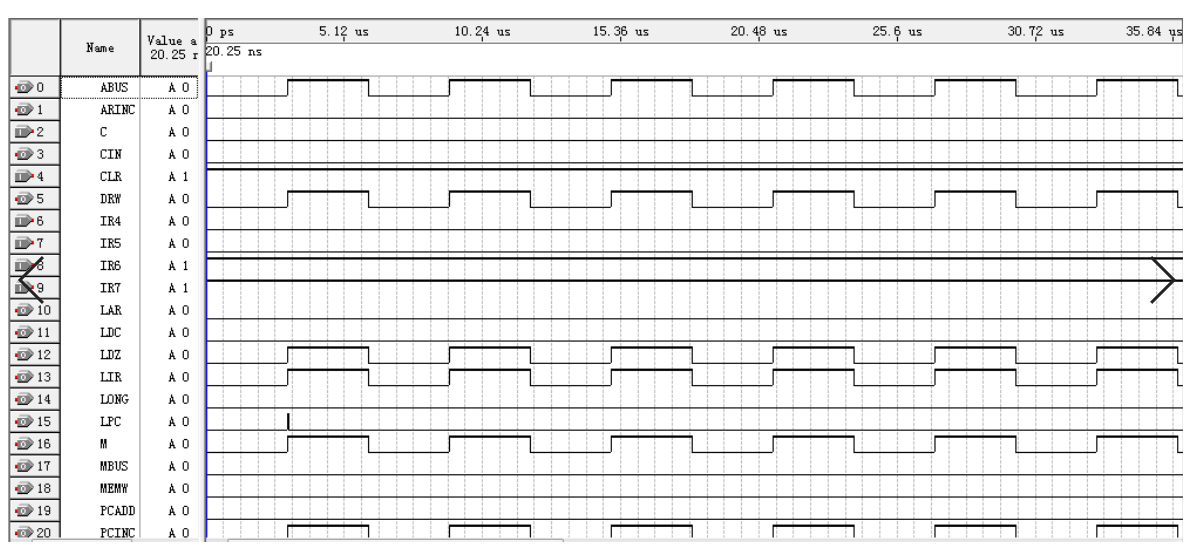
Quartus Ⅱ提供了仿真功能。能够对编译完成后的程序进行测试。

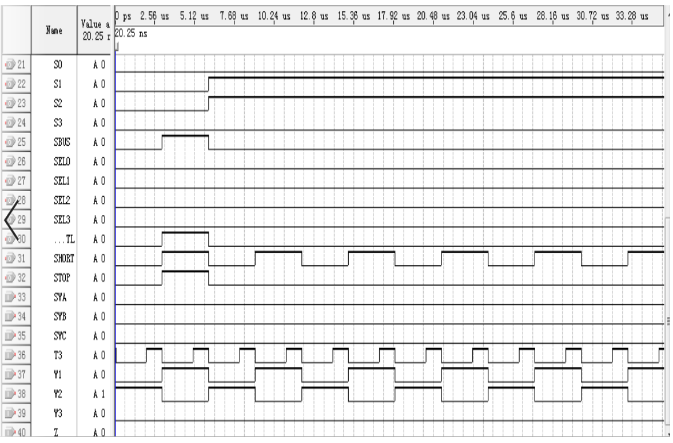
仿真分为功能仿真和时序仿真，其中功能仿真为了验证程序逻辑的正确性，而时序仿真不仅能验证正确性还能计算时延。

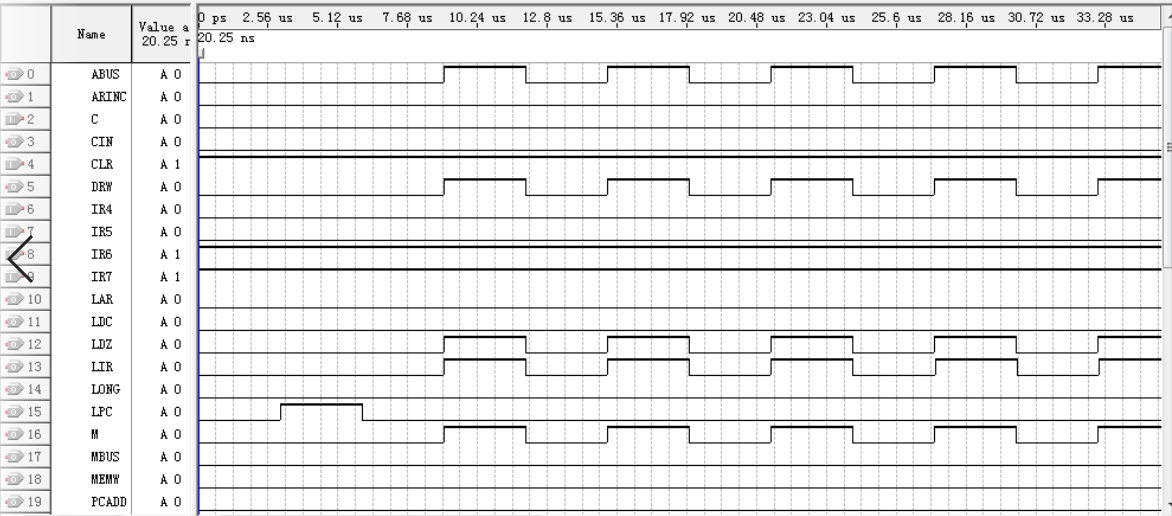
时序仿真得出的波形图比功能仿真的波形图更加精确可信。原因是：时序仿真还考虑到了各种芯片的实际参数。

此处以AND指令仿真测试为例进行对比。

**功能仿真**

****

**时序仿真**

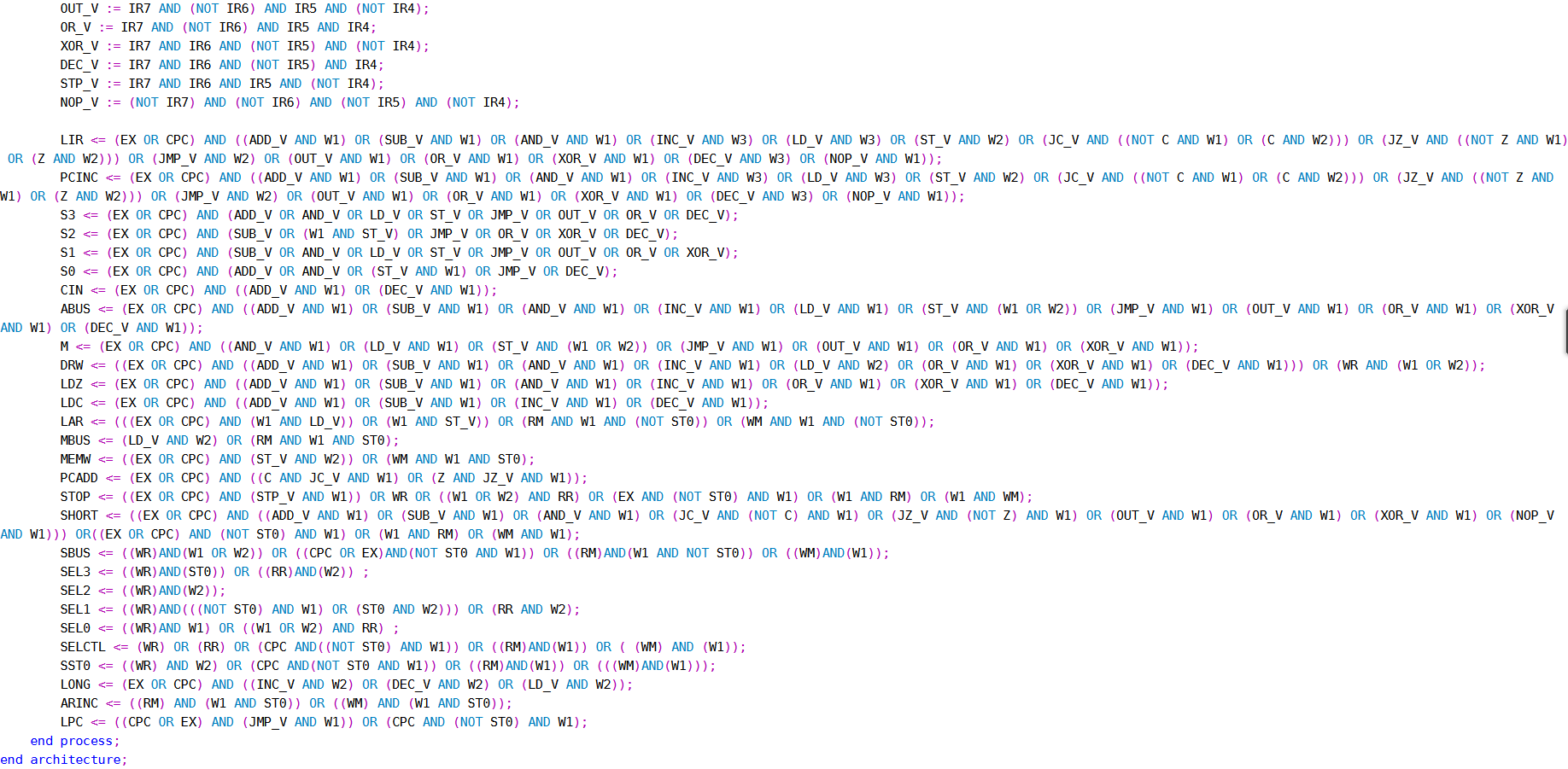


可以看到时序仿真得出的波形图比功能仿真得到的波形图更加精确，两者都能对程序逻辑的正确性进行验证。

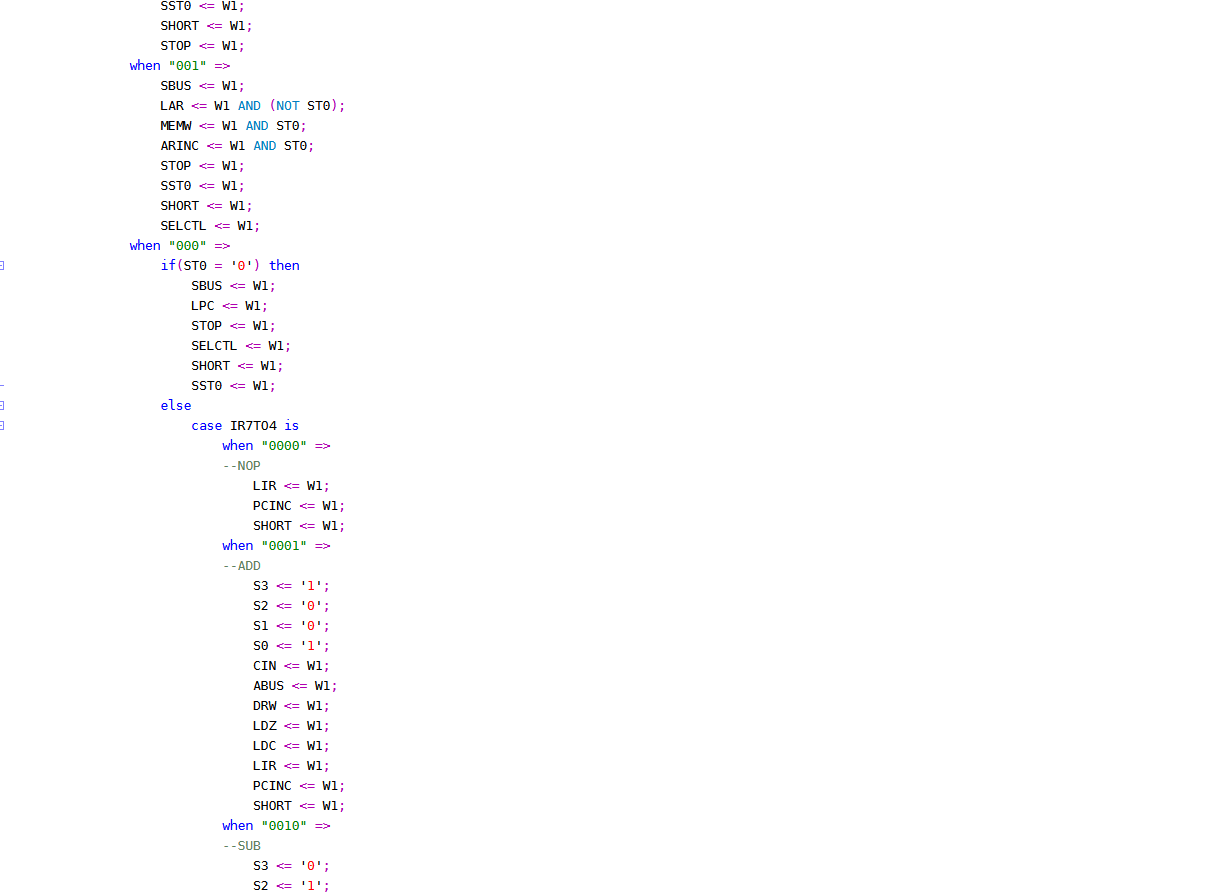
## 第六部分 两种不同编码风格的分析

我们小组在实验过程中使用了两种编码方式，并且都通过了测试，在这里给出不同编码风格的优缺点分析。

**第一种：组合逻辑方式编码（部分）**

****

**第二种：顺序逻辑方式编码（部分）**

****

可以很清楚的看出第一种组合逻辑编码方式没有考虑执行顺序，而是使用逻辑表达式来将对应的信号变为高电平低电平。第二种则是需要考虑到程序执行的流程，是按顺序编码的。

这里第一种组合逻辑编码方式优点为：逻辑严谨、减小了部分代码造成的时延、实验信号时序更加精确，是十分严谨的编码方式。它也有着缺点：不容易查错、不容易调试，必须要有组合逻辑表才能进行编码等等缺点。

这里第二种顺序编码方式优点为：容易查找错误点、便于调试，清晰易读且有着清晰的顺序。缺点为：可能会因为某些语句造成时延，并不是最严谨的方案。

我们在使用顺序逻辑编码时，遇到了很多问题，比如出现吞指令或者指令覆盖问题，但是如果使用组合逻辑编码方案，这些问题能得到改善，但依旧没有完全解决，这与硬件也有着关系。

## 第七部分 问题及解决

**问题1：第一个实验台的时序产生错误，W1,W2,W3灯会同时亮。**

**解决：**换了一个实验台就好了。

**问题2：在编码时没有设置CLR时将ST0恢复成 0，而是恢复成1，无法使用工作台。**

**解决：**修改代码重新烧录解决问题。

**问题3： 指令译码出现问题，解析出错误的功能。**

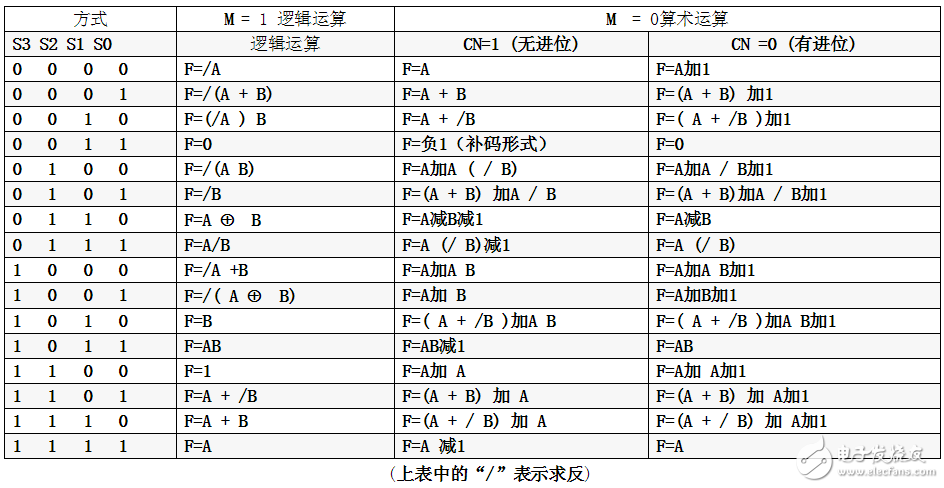
**解决：**发现是代码问题，在拼接指令高四位的信号时拼接顺序出错，修改后成功解析。

**问题4： 写寄存器和读存储器以及写存储器都出现问题，无法正常工作。**

**解决：**SST0信号没有写在代码中，导致标志位ST0无法置1，修改后成功。

**问题5： ALU运算器信号出错，导致运算出错。**

**解决：**寻找到74LS181运算功能表**，**改正错误的信号。

****

**问题6： 测试程序的LD后的JMP不会执行，而是会执行两遍JMP之后的指令。**

**解决：**首先发现问题错因是LIR和PCINC的本因是并行起作用，但是实际测试过程中发现先将PC加一，然后取指令，导致取到了下下条指令，从而吞掉JMP且重复执行两遍之后的指令，在INC和DEC指令也会出现此类问题。解决方案即为取消部分指令流水，在此取消INC、DEC和LD指令的流水，即加入空节拍延长流水。

**问题7：组合逻辑代码无法工作。**

**解决：**仔细检查逻辑译码表和代码，对比发现出现三处错误，修正代码即可。

## 第八部分 工作日志

**此处脱敏处理。**

## 第九部分 设计调试小结

首先在设计之前我们是必须要对问题进行分析，得到分析结果后才能知道我们的方向是什么，之后的设计和各项工作才能有条不紊的进行，问题分析是最重要的，拿到问题之后我们立马进行了讨论分析，很顺利的得到了我们想要的东西。这样的步调使得我们经过第二天的设计已经拿出了最终的设计方案，所以说拿到问题必须先分析问题再设计，这会使得我们的工作事半功倍。

调试是一个漫长的过程，我们需要综合各个方面去测试，去差错。这个时候，人越多是越好的，所以我们小组在调试阶段是全部到场进行调试，这样不仅能集思广益，而且能让每个组员都能够加深对设计的理解，这样就是每个人都有了收获，而且能够快速的解决问题。这无疑是我在这次实验过程中最大的收获。

设计时我们并没有遇到很多的问题，但是在调试时遇到了大量的问题，比如由于粗心导致的信号漏写，指令高四位拼接错误等等。这些很快被我们发现，最不容易被发现的却是硬件的问题，以为你此时并不知道是你自己的问题或者是实验台的问题，首先你不会去怀疑实验台，直到你无法找出自己的错误，这个过程十分漫长。比如有的实验台没有吞指令现象，但是我们的就有，而且十分频繁的出现，我们及时查找问题所在，找到LIR和PCINC的原因，及时询问老师，并且在小组成员的努力下寻找到一个稳定可行的解决方案，这无疑是十分高效且有成就感的工作。

## 第十部分 个人心得及收获

**此处脱敏处理。**

**VHDL源码：**

**基础版本：**

|  |
| --- |
| library ieee;  use ieee.std\_logic\_1164.all;  use ieee.std\_logic\_unsigned.all;  entity controller is  port(  CLR,SWA,SWB,SWC,IR4,IR5,IR6,IR7,W1,W2,W3,C,T3,Z:IN STD\_LOGIC;  DRW,PCINC,LPC,LAR,PCADD,ARINC,MEMW,STOP,LIR,LDZ,LDC,CIN,S0,S1,S2,S3,M,ABUS,SBUS,MBUS,SHORT,LONG,SEL0,SEL1,SEL2,SEL3,SELCTL:OUT STD\_LOGIC  );  end controller;  architecture controller\_arc of controller is  signal SWCBA:std\_logic\_vector(2 downto 0);  signal IR7TO4:std\_logic\_vector(3 downto 0);  signal ST0:std\_logic;  signal SST0:std\_logic;  begin  SWCBA <= SWC&SWB&SWA;  IR7TO4 <= IR7&IR6&IR5&IR4;    process(CLR,SWCBA,IR7TO4,W1,W2,W3,ST0,C,T3,Z,T3)  begin  DRW <= '0';  PCINC <= '0';  LPC <= '0';  LAR <= '0';  PCADD <= '0';  ARINC <= '0';  MEMW <= '0';  STOP <= '0';  LIR <= '0';  LDZ <= '0';  LDC <= '0';  CIN <= '0';  S0 <= '0';  S1 <= '0';  S2 <= '0';  S3 <= '0';  M <= '0';  ABUS <= '0';  SBUS <= '0';  MBUS <= '0';  SHORT <= '0';  LONG <= '0';  SEL0 <= '0';  SEL1 <= '0';  SEL2 <= '0';  SEL3 <= '0';  SELCTL <= '0';    if(CLR = '0') then  ST0 <= '0';  else  if(T3'EVENT AND T3 = '0') then  if(SST0 = '1') then  ST0 <= '1';  end if;  end if;    case SWCBA is  when "100" =>  SBUS <= W1 OR W2;  SEL3 <= ST0;  SEL2 <= W2;  SEL1 <= ((NOT ST0) AND W1) OR (ST0 AND W2);  SEL0 <= W1;  SST0 <= W2;  SELCTL <= '1';  DRW <= W1 OR W2;  STOP <= '1';  when "011" =>  SEL3 <= W2;  SEL2 <= '0';  SEL1 <= W2;  SEL0 <= W1 OR W2;  SELCTL <= '1';  STOP <= W1 OR W2;  when "010" =>  SBUS <= W1 AND (NOT ST0);  LAR <= W1 AND (NOT ST0);  MBUS <= W1 AND ST0;  ARINC <= W1 AND ST0;  SELCTL <= W1;  SST0 <= W1;  SHORT <= W1;  STOP <= W1;  when "001" =>  SBUS <= W1;  LAR <= W1 AND (NOT ST0);  MEMW <= W1 AND ST0;  ARINC <= W1 AND ST0;  STOP <= W1;  SST0 <= W1;  SHORT <= W1;  SELCTL <= W1;  when "000" =>  if(ST0 = '0') then  SBUS <= W1;  LPC <= W1;  STOP <= W1;  SELCTL <= W1;  SHORT <= W1;  SST0 <= W1;  else  LIR <= W1;  PCINC <= W1;  case IR7TO4 is  when "0001" =>  --ADD  S3 <= '1';  S2 <= '0';  S1 <= '0';  S0 <= '1';  CIN <= W2;  ABUS <= W2;  DRW <= W2;  LDZ <= W2;  LDC <= W2;  when "0010" =>  --SUB  S3 <= '0';  S2 <= '1';  S1 <= '1';  S0 <= '0';  ABUS <= W2;  DRW <= W2;  LDZ <= W2;  LDC <= W2;  when "0011" =>  --AND  S3 <= '1';  S2 <= '0';  S1 <= '1';  S0 <= '1';  M <= W2;  ABUS <= W2;  DRW <= W2;  LDZ <= W2;  when "0100" =>  --INC  S3 <= '0';  S2 <= '0';  S1 <= '0';  S0 <= '0';  ABUS <= W2;  DRW <= W2;  LDZ <= W2;  LDC <= W2;  when "0101" =>  --LD  S3 <= '1';  S2 <= '0';  S1 <= '1';  S0 <= '0';  M <= W2;  ABUS <= W2;  LAR <= W2;  LONG <= W2;  DRW <= W3;  MBUS <= W3;  when "0110" =>  --ST  S3 <= '1';  S2 <= W2;  S1 <= '1';  S0 <= W2;  LONG <= W2;  M <= W2 OR W3;  ABUS <= W2 OR W3;  LAR <= W2;  MEMW <= W3;  when "0111" =>  --JC  PCADD <= C AND W2;  when "1000" =>  --JZ  PCADD <= Z AND W2;  when "1001" =>  --JMP  S3 <= '1';  S2 <= '1';  S1 <= '1';  S0 <= '1';  M <= W2;  ABUS <= W2;  LPC <= W2;  when "1010" =>  --OUT  S3 <= '1';  S2 <= '0';  S1 <= '1';  S0 <= '0';  M <= W2;  ABUS <= W2;  when "1011" =>  --OR  S3 <= '1';  S2 <= '1';  S1 <= '1';  S0 <= '0';  M <= W2;  ABUS <= W2;  DRW <= W2;  LDZ <= W2;  when "1100" =>  --XOR  S3 <= '0';  S2 <= '1';  S1 <= '1';  S0 <= '0';  M <= W2;  ABUS <= W2;  DRW <= W2;  LDZ <= W2;  when "1101" =>  --DEC  S3 <= '1';  S2 <= '1';  S1 <= '1';  S0 <= '1';  CIN <= W2;  ABUS <= W2;  DRW <= W2;  LDZ <= W2;  LDC <= W2;  when "1110" =>  --STP  STOP <= W2;  when others =>    end case;  end if;  when others =>    end case;  end if;  end process;  end architecture; |

**流水线版本1：**

|  |
| --- |
| library ieee;  use ieee.std\_logic\_1164.all;  use ieee.std\_logic\_unsigned.all;  entity controller is  port(  CLR,SWA,SWB,SWC,IR4,IR5,IR6,IR7,W1,W2,W3,C,T3,Z:IN STD\_LOGIC;  DRW,PCINC,LPC,LAR,PCADD,ARINC,MEMW,STOP,LIR,LDZ,LDC,CIN,S0,S1,S2,S3,M,ABUS,SBUS,MBUS,SHORT,LONG,SEL0,SEL1,SEL2,SEL3,SELCTL:OUT STD\_LOGIC  );  end controller;  architecture controller\_arc of controller is  signal SWCBA:std\_logic\_vector(2 downto 0);  signal IR7TO4:std\_logic\_vector(3 downto 0);  signal ST0:std\_logic;  signal SST0:std\_logic;  begin  SWCBA <= SWC&SWB&SWA;  IR7TO4 <= IR7&IR6&IR5&IR4;    process(CLR,SWCBA,IR7TO4,W1,W2,W3,ST0,C,T3,Z,T3)  begin  DRW <= '0';  PCINC <= '0';  LPC <= '0';  LAR <= '0';  PCADD <= '0';  ARINC <= '0';  MEMW <= '0';  STOP <= '0';  LIR <= '0';  LDZ <= '0';  LDC <= '0';  CIN <= '0';  S0 <= '0';  S1 <= '0';  S2 <= '0';  S3 <= '0';  M <= '0';  ABUS <= '0';  SBUS <= '0';  MBUS <= '0';  SHORT <= '0';  LONG <= '0';  SEL0 <= '0';  SEL1 <= '0';  SEL2 <= '0';  SEL3 <= '0';  SELCTL <= '0';    if(CLR = '0') then  ST0 <= '0';  else  if(T3'EVENT AND T3 = '0') then  if(SST0 = '1') then  ST0 <= '1';  end if;  end if;    case SWCBA is  when "100" =>  SBUS <= W1 OR W2;  SEL3 <= ST0;  SEL2 <= W2;  SEL1 <= ((NOT ST0) AND W1) OR (ST0 AND W2);  SEL0 <= W1;  SST0 <= W2;  SELCTL <= '1';  DRW <= W1 OR W2;  STOP <= '1';  when "011" =>  SEL3 <= W2;  SEL2 <= '0';  SEL1 <= W2;  SEL0 <= W1 OR W2;  SELCTL <= '1';  STOP <= W1 OR W2;  when "010" =>  SBUS <= W1 AND (NOT ST0);  LAR <= W1 AND (NOT ST0);  MBUS <= W1 AND ST0;  ARINC <= W1 AND ST0;  SELCTL <= W1;  SST0 <= W1;  SHORT <= W1;  STOP <= W1;  when "001" =>  SBUS <= W1;  LAR <= W1 AND (NOT ST0);  MEMW <= W1 AND ST0;  ARINC <= W1 AND ST0;  STOP <= W1;  SST0 <= W1;  SHORT <= W1;  SELCTL <= W1;  when "000" =>  if(ST0 = '0') then  SBUS <= W1;  LPC <= W1;  STOP <= W1;  SELCTL <= W1;  SHORT <= W1;  SST0 <= W1;  else  case IR7TO4 is  when "0000" =>  --NOP  LIR <= W1;  PCINC <= W1;  SHORT <= W1;  when "0001" =>  --ADD  S3 <= '1';  S2 <= '0';  S1 <= '0';  S0 <= '1';  CIN <= W1;  ABUS <= W1;  DRW <= W1;  LDZ <= W1;  LDC <= W1;  LIR <= W1;  PCINC <= W1;  SHORT <= W1;  when "0010" =>  --SUB  S3 <= '0';  S2 <= '1';  S1 <= '1';  S0 <= '0';  ABUS <= W1;  DRW <= W1;  LDZ <= W1;  LDC <= W1;  LIR <= W1;  PCINC <= W1;  SHORT <= W1;  when "0011" =>  --AND  S3 <= '1';  S2 <= '0';  S1 <= '1';  S0 <= '1';  M <= W1;  ABUS <= W1;  DRW <= W1;  LDZ <= W1;  LIR <= W1;  PCINC <= W1;  SHORT <= W1;  when "0100" =>  --INC  S3 <= '0';  S2 <= '0';  S1 <= '0';  S0 <= '0';  ABUS <= W1;  DRW <= W1;  LDZ <= W1;  LDC <= W1;  LONG <= W2;  LIR <= W3;  PCINC <= W3;  when "0101" =>  --LD  S3 <= '1';  S2 <= '0';  S1 <= '1';  S0 <= '0';  M <= W1;  ABUS <= W1;  LAR <= W1;  DRW <= W2;  MBUS <= W2;  LIR <= W3;  LONG <= W2;  PCINC <= W3;  when "0110" =>  --ST  S3 <= '1';  S2 <= W1;  S1 <= '1';  S0 <= W1;  M <= W1 OR W2;  ABUS <= W1 OR W2;  LAR <= W1;  MEMW <= W2;  LIR <= W2;  PCINC <= W2;  when "0111" =>  --JC  LIR <= ((NOT C) AND W1) OR (C AND W2);  PCINC <= ((NOT C) AND W1) OR (C AND W2);  PCADD <= C AND W1;  SHORT <= (NOT C) AND W1;  when "1000" =>  --JZ  LIR <= ((NOT Z) AND W1) OR (Z AND W2);  PCINC <= ((NOT Z) AND W1) OR (Z AND W2);  PCADD <= Z AND W1;  SHORT <= (NOT Z) AND W1;  when "1001" =>  --JMP  S3 <= '1';  S2 <= '1';  S1 <= '1';  S0 <= '1';  M <= W1;  ABUS <= W1;  LPC <= W1;  LIR <= W2;  PCINC <= W2;  when "1010" =>  --OUT  S3 <= '1';  S2 <= '0';  S1 <= '1';  S0 <= '0';  M <= W1;  ABUS <= W1;  SHORT <= W1;  LIR <= W1;  PCINC <= W1;  when "1011" =>  --OR  S3 <= '1';  S2 <= '1';  S1 <= '1';  S0 <= '0';  M <= W1;  ABUS <= W1;  DRW <= W1;  LDZ <= W1;  LIR <= W1;  PCINC <= W1;  SHORT <= W1;  when "1100" =>  --XOR  S3 <= '0';  S2 <= '1';  S1 <= '1';  S0 <= '0';  M <= W1;  ABUS <= W1;  DRW <= W1;  LDZ <= W1;  LIR <= W1;  PCINC <= W1;  SHORT <= W1;  when "1101" =>  --DEC  S3 <= '1';  S2 <= '1';  S1 <= '1';  S0 <= '1';  CIN <= W1;  ABUS <= W1;  DRW <= W1;  LDZ <= W1;  LDC <= W1;  LONG <= W2;  LIR <= W3;  PCINC <= W3;  when "1110" =>  --STP  STOP <= W1;  when others =>  LIR <= W1;  PCINC <= W1;  SHORT <= W1;  end case;  end if;  when others =>    end case;  end if;  end process;  end architecture; |

**流水线版本2**

|  |
| --- |
| **library ieee;**  **use ieee.std\_logic\_1164.all;**  **use ieee.std\_logic\_unsigned.all;**  **entity controller is**  **port(**  **CLR,SWA,SWB,SWC,IR4,IR5,IR6,IR7,W1,W2,W3,C,T3,Z:IN STD\_LOGIC;**  **DRW,PCINC,LPC,LAR,PCADD,ARINC,MEMW,STOP,LIR,LDZ,LDC,CIN,S0,S1,S2,S3,M,ABUS,SBUS,MBUS,SHORT,LONG,SEL0,SEL1,SEL2,SEL3,SELCTL:OUT STD\_LOGIC**  **);**  **end controller;**  **architecture controller\_arc of controller is**  **signal SWCBA:std\_logic\_vector(2 downto 0);**  **signal IR7TO4:std\_logic\_vector(3 downto 0);**  **signal ST0:std\_logic;**  **signal SST0:std\_logic;**  **begin**  **process(CLR,SWCBA,IR7TO4,W1,W2,W3,ST0,C,T3,Z,T3)**  **variable WR:std\_logic;**  **variable RR:std\_logic;**  **variable RM:std\_logic;**  **variable WM:std\_logic;**  **variable EX:std\_logic;**  **variable CPC:std\_logic;**  **variable ADD\_V:std\_logic;**  **variable SUB\_V:std\_logic;**  **variable AND\_V:std\_logic;**  **variable INC\_V:std\_logic;**  **variable LD\_V:std\_logic;**  **variable ST\_V:std\_logic;**  **variable JC\_V:std\_logic;**  **variable JZ\_V:std\_logic;**  **variable JMP\_V:std\_logic;**  **variable OUT\_V:std\_logic;**  **variable OR\_V:std\_logic;**  **variable XOR\_V:std\_logic;**  **variable DEC\_V:std\_logic;**  **variable STP\_V:std\_logic;**  **variable NOP\_V:std\_logic;**  **begin**  **DRW <= '0';**  **PCINC <= '0';**  **LPC <= '0';**  **LAR <= '0';**  **PCADD <= '0';**  **ARINC <= '0';**  **MEMW <= '0';**  **STOP <= '0';**  **LIR <= '0';**  **LDZ <= '0';**  **LDC <= '0';**  **CIN <= '0';**  **S0 <= '0';**  **S1 <= '0';**  **S2 <= '0';**  **S3 <= '0';**  **M <= '0';**  **ABUS <= '0';**  **SBUS <= '0';**  **MBUS <= '0';**  **SHORT <= '0';**  **LONG <= '0';**  **SEL0 <= '0';**  **SEL1 <= '0';**  **SEL2 <= '0';**  **SEL3 <= '0';**  **SELCTL <= '0';**    **if(CLR = '0') then**  **ST0 <= '0';**  **else**  **if(T3'EVENT AND T3 = '0') then**  **if(SST0 = '1') then**  **ST0 <= '1';**  **end if;**  **end if;**  **end if;**  **WR := SWC AND (NOT SWB) AND (NOT SWA);**  **RR := (NOT SWC) AND SWB AND SWA;**  **EX := (NOT SWC) AND (NOT SWB) AND (NOT SWA) AND (ST0);**  **CPC := (NOT SWC) AND (NOT SWB) AND (NOT SWA) AND (NOT ST0);**  **RM := (NOT SWC) AND SWB AND (NOT SWA);**  **WM := (NOT SWC) AND (NOT SWB) AND SWA;**    **ADD\_V := (NOT IR7) AND (NOT IR6) AND (NOT IR5) AND IR4;**  **SUB\_V := (NOT IR7) AND (NOT IR6) AND IR5 AND (NOT IR4);**  **AND\_V := (NOT IR7) AND (NOT IR6) AND IR5 AND IR4;**  **INC\_V := (NOT IR7) AND IR6 AND (NOT IR5) AND (NOT IR4);**  **LD\_V := (NOT IR7) AND IR6 AND (NOT IR5) AND IR4;**  **ST\_V := (NOT IR7) AND IR6 AND IR5 AND (NOT IR4);**  **JC\_V := (NOT IR7) AND IR6 AND IR5 AND IR4;**  **JZ\_V := IR7 AND (NOT IR6) AND (NOT IR5) AND (NOT IR4);**  **JMP\_V := IR7 AND (NOT IR6) AND (NOT IR5) AND IR4;**  **OUT\_V := IR7 AND (NOT IR6) AND IR5 AND (NOT IR4);**  **OR\_V := IR7 AND (NOT IR6) AND IR5 AND IR4;**  **XOR\_V := IR7 AND IR6 AND (NOT IR5) AND (NOT IR4);**  **DEC\_V := IR7 AND IR6 AND (NOT IR5) AND IR4;**  **STP\_V := IR7 AND IR6 AND IR5 AND (NOT IR4);**  **NOP\_V := (NOT IR7) AND (NOT IR6) AND (NOT IR5) AND (NOT IR4);**    **LIR <= (EX OR CPC) AND ((ADD\_V AND W1) OR (SUB\_V AND W1) OR (AND\_V AND W1) OR (INC\_V AND W3) OR (LD\_V AND W3) OR (ST\_V AND W2) OR (JC\_V AND ((NOT C AND W1) OR (C AND W2))) OR (JZ\_V AND ((NOT Z AND W1) OR (Z AND W2))) OR (JMP\_V AND W2) OR (OUT\_V AND W1) OR (OR\_V AND W1) OR (XOR\_V AND W1) OR (DEC\_V AND W3) OR (NOP\_V AND W1));**  **PCINC <= (EX OR CPC) AND ((ADD\_V AND W1) OR (SUB\_V AND W1) OR (AND\_V AND W1) OR (INC\_V AND W3) OR (LD\_V AND W3) OR (ST\_V AND W2) OR (JC\_V AND ((NOT C AND W1) OR (C AND W2))) OR (JZ\_V AND ((NOT Z AND W1) OR (Z AND W2))) OR (JMP\_V AND W2) OR (OUT\_V AND W1) OR (OR\_V AND W1) OR (XOR\_V AND W1) OR (DEC\_V AND W3) OR (NOP\_V AND W1));**  **S3 <= (EX OR CPC) AND (ADD\_V OR AND\_V OR LD\_V OR ST\_V OR JMP\_V OR OUT\_V OR OR\_V OR DEC\_V);**  **S2 <= (EX OR CPC) AND (SUB\_V OR (W1 AND ST\_V) OR JMP\_V OR OR\_V OR XOR\_V OR DEC\_V);**  **S1 <= (EX OR CPC) AND (SUB\_V OR AND\_V OR LD\_V OR ST\_V OR JMP\_V OR OUT\_V OR OR\_V OR XOR\_V);**  **S0 <= (EX OR CPC) AND (ADD\_V OR AND\_V OR (ST\_V AND W1) OR JMP\_V OR DEC\_V);**  **CIN <= (EX OR CPC) AND ((ADD\_V AND W1) OR (DEC\_V AND W1));**  **ABUS <= (EX OR CPC) AND ((ADD\_V AND W1) OR (SUB\_V AND W1) OR (AND\_V AND W1) OR (INC\_V AND W1) OR (LD\_V AND W1) OR (ST\_V AND (W1 OR W2)) OR (JMP\_V AND W1) OR (OUT\_V AND W1) OR (OR\_V AND W1) OR (XOR\_V AND W1) OR (DEC\_V AND W1));**  **M <= (EX OR CPC) AND ((AND\_V AND W1) OR (LD\_V AND W1) OR (ST\_V AND (W1 OR W2)) OR (JMP\_V AND W1) OR (OUT\_V AND W1) OR (OR\_V AND W1) OR (XOR\_V AND W1));**  **DRW <= ((EX OR CPC) AND ((ADD\_V AND W1) OR (SUB\_V AND W1) OR (AND\_V AND W1) OR (INC\_V AND W1) OR (LD\_V AND W2) OR (OR\_V AND W1) OR (XOR\_V AND W1) OR (DEC\_V AND W1))) OR (WR AND (W1 OR W2));**  **LDZ <= (EX OR CPC) AND ((ADD\_V AND W1) OR (SUB\_V AND W1) OR (AND\_V AND W1) OR (INC\_V AND W1) OR (OR\_V AND W1) OR (XOR\_V AND W1) OR (DEC\_V AND W1));**  **LDC <= (EX OR CPC) AND ((ADD\_V AND W1) OR (SUB\_V AND W1) OR (INC\_V AND W1) OR (DEC\_V AND W1));**  **LAR <= (((EX OR CPC) AND (W1 AND LD\_V)) OR (W1 AND ST\_V)) OR (RM AND W1 AND (NOT ST0)) OR (WM AND W1 AND (NOT ST0));**  **MBUS <= (LD\_V AND W2) OR (RM AND W1 AND ST0);**  **MEMW <= ((EX OR CPC) AND (ST\_V AND W2)) OR (WM AND W1 AND ST0);**  **PCADD <= (EX OR CPC) AND ((C AND JC\_V AND W1) OR (Z AND JZ\_V AND W1));**  **STOP <= ((EX OR CPC) AND (STP\_V AND W1)) OR WR OR ((W1 OR W2) AND RR) OR (EX AND (NOT ST0) AND W1) OR (W1 AND RM) OR (W1 AND WM);**  **SHORT <= ((EX OR CPC) AND ((ADD\_V AND W1) OR (SUB\_V AND W1) OR (AND\_V AND W1) OR (JC\_V AND (NOT C) AND W1) OR (JZ\_V AND (NOT Z) AND W1) OR (OUT\_V AND W1) OR (OR\_V AND W1) OR (XOR\_V AND W1) OR (NOP\_V AND W1))) OR((EX OR CPC) AND (NOT ST0) AND W1) OR (W1 AND RM) OR (WM AND W1);**  **SBUS <= ((WR)AND(W1 OR W2)) OR ((CPC OR EX)AND(NOT ST0 AND W1)) OR ((RM)AND(W1 AND NOT ST0)) OR ((WM)AND(W1));**  **SEL3 <= ((WR)AND(ST0)) OR ((RR)AND(W2)) ;**  **SEL2 <= ((WR)AND(W2));**  **SEL1 <= ((WR)AND(((NOT ST0) AND W1) OR (ST0 AND W2))) OR (RR AND W2);**  **SEL0 <= ((WR)AND W1) OR ((W1 OR W2) AND RR) ;**  **SELCTL <= (WR) OR (RR) OR (CPC AND((NOT ST0) AND W1)) OR ((RM)AND(W1)) OR ( (WM) AND (W1));**  **SST0 <= ((WR) AND W2) OR (CPC AND(NOT ST0 AND W1)) OR ((RM)AND(W1)) OR (((WM)AND(W1)));**  **LONG <= (EX OR CPC) AND ((INC\_V AND W2) OR (DEC\_V AND W2) OR (LD\_V AND W2));**  **ARINC <= ((RM) AND (W1 AND ST0)) OR ((WM) AND (W1 AND ST0));**  **LPC <= ((CPC OR EX) AND (JMP\_V AND W1)) OR (CPC AND (NOT ST0) AND W1);**  **end process;**  **end architecture;** |