北京邮电大学课程设计报告

<table>
<tr><td>课程设计名称</td><td colspan="2">计算机组成原理</td><td>学 院</td><td>计算机学院</td><td>指导教师</td><td>靳秀国</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>2020211313</td><td>4</td><td colspan="2">2020210723</td><td>吴家宝</td><td colspan="2"></td></tr>
<tr><td>2020211313</td><td>31</td><td colspan="2">2020212931</td><td>刘书恺</td><td colspan="2"></td></tr>
<tr><td>2020211313</td><td>16</td><td colspan="2">2020211838</td><td>王昱杰</td><td colspan="2"></td></tr>
<tr><td>2020211307</td><td>27</td><td colspan="2">2020211449</td><td>吴政隆</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
团队分工:
吴家宝：负责拓展功能，中断程序的实现与上板调试，大部分文档书写
刘书恺：负责拓展功能，流水程序的实现与上板调试，大部分文档书写
王昱杰：负责基础版的框架实现与仿真，尝试实现要求之外的任务，文档书写
吴政隆：负责测试程序的实现、调试与仿真（线上），文档书写</td></tr>
<tr><td>学生
课程设计
报告
(附页)</td><td colspan="6">见附页</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>

目录

1. 完成的任务

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.允修改PC指针功能（任意指针）

自选题目一 在必选题目基础上，完成流水硬连线控制器的设计根据设计方案，在TEC-8 进行组装、调试运行。

自选题目二 在必选题目基础上，设计实现带有中断功能的硬布线控制器。

2. 实验设备以及环境

实验平台：TEC-8 计算机组成与体系结构实验系统

可编程逻辑设计环境：Quartus II 9.0

编程语言：VHDL

3. 实验原理

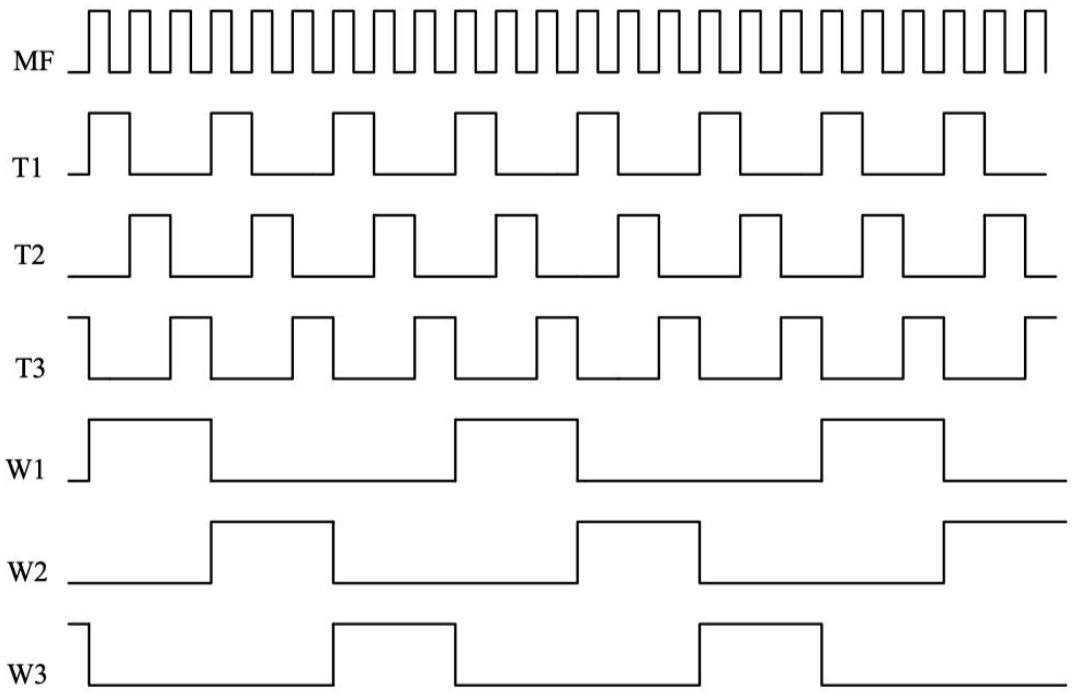
3.1 模型计算机时序信号

硬布线控制器原理：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，

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

其中 I_m 是机器指令操作码译码器的输出信号，M_i 是节拍电位信号，T_k 是节拍脉冲信号，B_j 是状态条件信号。在 TEC-8 中，节拍脉冲信号 $T_k(T_1 \sim T_3)$ 直接输送给数据通路。省略了操作码译码器，4 位指令操作码 IR4～IR7 直接成为 I_m 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 I_m 的另一部分。 M_i 是时序发生器产生的节拍信号 W1～W3；B_j 包括 ALU 产生的进位信号 C、结果为 0 信号 Z 等等。

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 个节拍脉冲 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 W 1、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。模型机的时序如下图所示:



3.2 组成模块 & 数据通路

下表为作为硬连线控制器时的EPM7128 引脚规定:

信号	方向	引脚号	信号	方向	引脚号
CLR#	输入	1	MEMW	输出	27

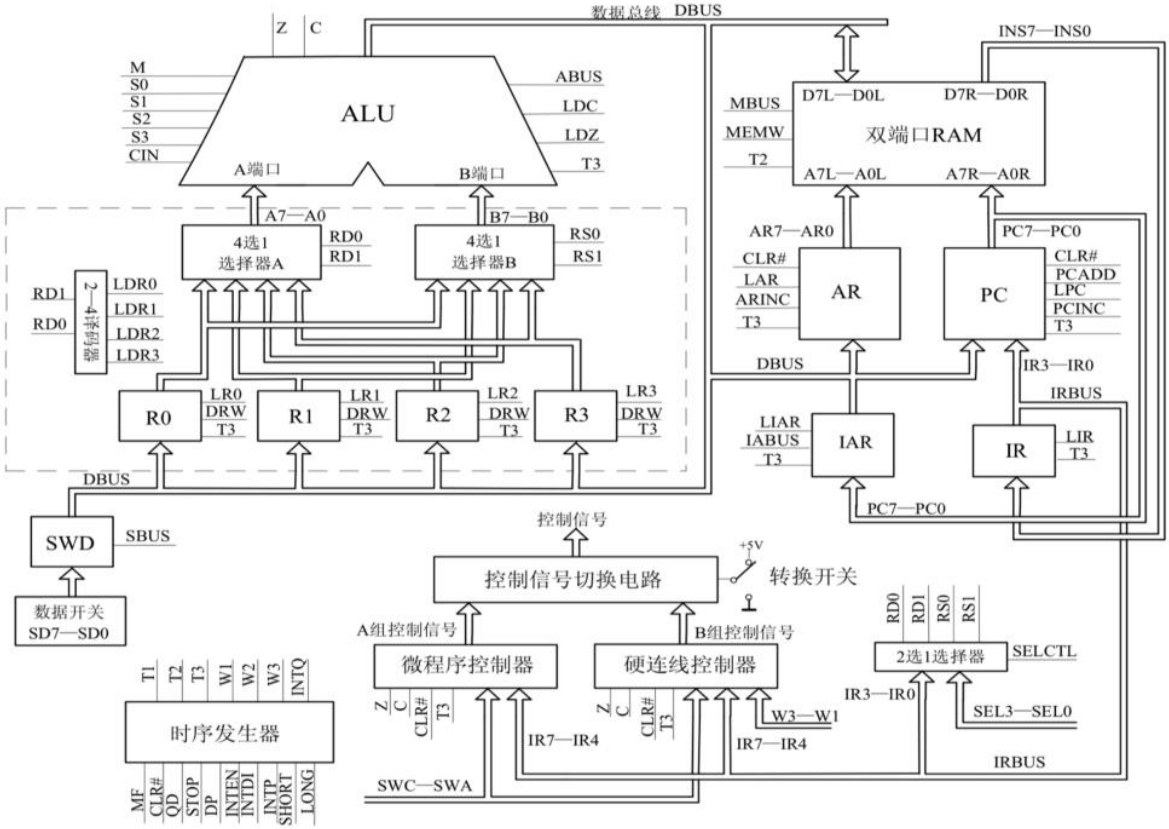
T3	输入	83	STOP	输出	28
SWA	输入	4	LIR	输出	29
SWB	输入	5	LDZ	输出	30
SWC	输入	6	LDC	输出	31
IR4	输入	8	CIN	输出	33
IR5	输入	9	S0	输出	34
IR6	输入	10	S1	输出	35
IR7	输入	11	S2	输出	36
W1	输入	12	S3	输出	37
W2	输入	15	M	输出	39
W3	输入	16	ABUS	输出	40
C	输入	2	SBUS	输出	41
Z	输入	84	MBUS	输出	44
DRW	输出	20	SHORT	输出	45
PCINC	输出	21	LONG	输出	46
LPC	输出	22	SEL0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

下表为各信号的说明:

信号	说明
CLR#	复位。
T3	节拍脉冲信号
SWC、SWB、SWA	操作模式选择。
IR7~IR4	指令寄存器的高四位。
W3~W1	节拍电位信号。
C	进位标志。
Z	结果为0 标志。
DRW	=1 时，在 T3 上升沿对 RD1、RD0 选中的寄存器进行写操作，将数据总线 DBUS 上的数 D7~D0 写入选定的寄存器。
PCINC	=1 时，在T3 的上升沿PC 加1。
LPC	=1 时，在 T3 的上升沿，将数据总线 DBUS 上的 D7~D0 写入程序计数器 PC。
LAR	=1 时，在 T3 的上升沿，将数据总线 DBUS 上的 D7~D0 写入地址寄存器 AR。

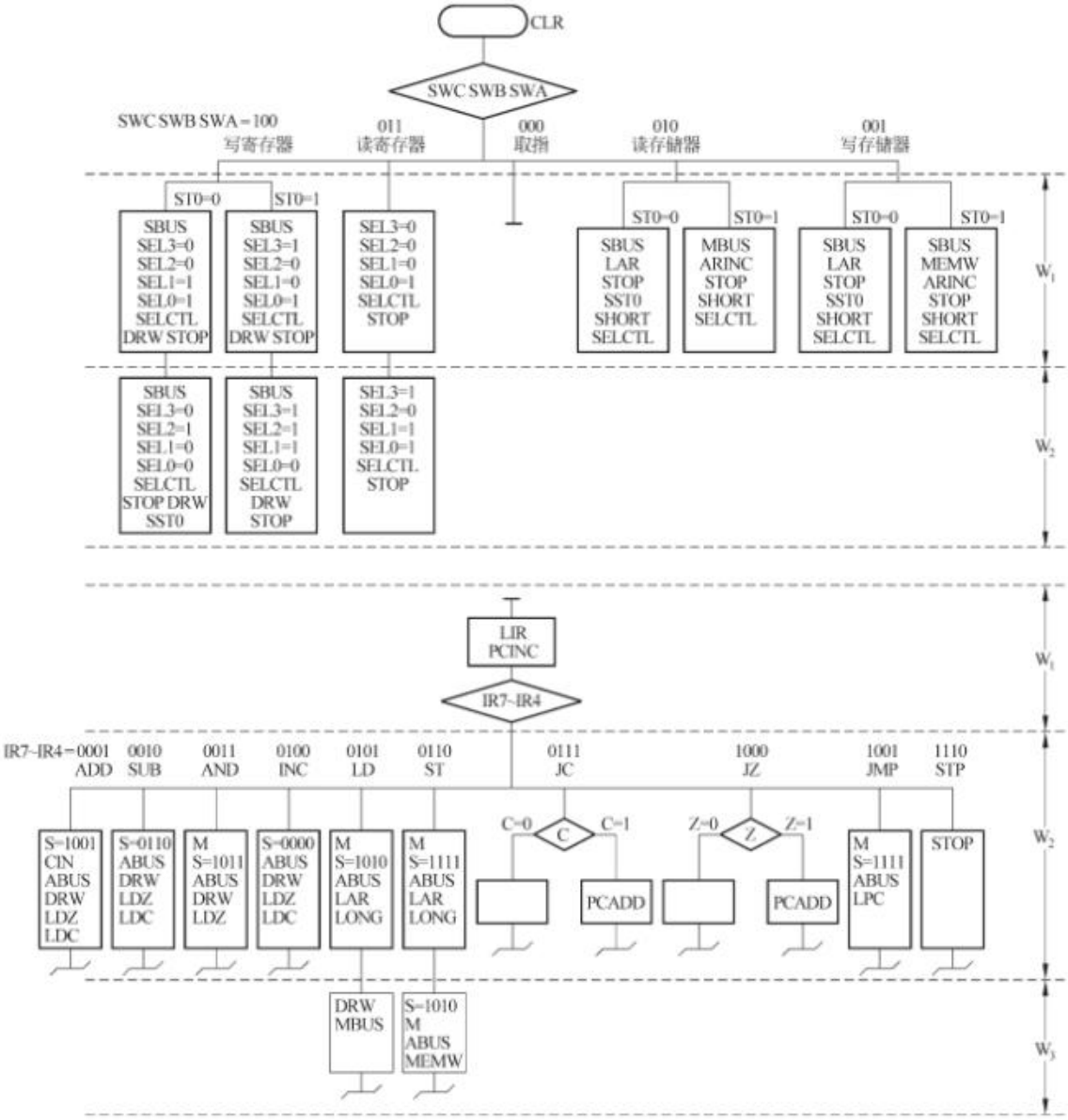
PCADD	=1 时，将当前的 PC 值加上相对转移量，生成新的 PC。
ARINC	=1 时，在T3 的上升沿，AR 加1。
SETCTL	=1 时，实验系统处于实验台状态。=0 时，实验系统处于运行程序状态。
MEMW	=1 时，在T2 为1 期间将数据总线DBUS 上的D7~D0 写入双端口RAM。写入的存储器单元由 AR7~AR0 指定。
STOP	=1 时，在 T3 结束后时序发生器停止输出节拍脉冲 T1、T2、T3。
LIR	=1 时，在 T3 的上升沿将从双端口 RAM 的右端口读出的指令 INS7~INS0 写入指令寄存器 IR。读出的存储器单元由 PC7~PC0 指定。
LDZ	=1 时，如果运算结果为0，在T3 的上升沿，将1 写入到Z 标志寄存器；如果运算结果不为0，将0 保存到Z 标志寄存器。
LDC	=1 时，在 T3 的上升沿将运算得到的进位保存到C 标志寄存器。
CIN	低位 74LS181 的进位输入。
M	运算模式：M=0 为算术运算；M=1 逻辑运算。
S3~S0	控制 74LS181 的运算类型。
ABUS	=1 时，将运算结果送数据总线 DBUS。
SBUS	=1 时，数据开关 SD7~SD0 的数送数据总线 DBUS。
MBUS	=1 时，将双端口 RAM 的左端口数据送到数据总线 DBUS。
SHORT	=1 时，指示时序发生器只产生一个节拍电位。
LONG	=1 时，指示时序发生器产生三个节拍电位。
SEL3~SEL2(RD1~RD0)	选择送ALU 的A 端口的寄存器
SEL1~SEL0(RS1~RS0)	选择送ALU 的B 端口的寄存器

下图为 TEC-8 模型计算机框图：



4. 题目分析与设计

4.1 任务一



我们程序设计参考了教科书上给出的流程图。经过简单分析我们认为程序主体部分可以由两个 case 语句进行实现：第一个 case 语句判断的是 SW，即控制台操作，不同的值对应着不同的操作，如读写寄存器。当 SW=000 时进入第二个 case 语句，判断的是 IR 高四位，这里代表对指令进行译码，根据不同的 IR 执行不同的操作。

执行不同的操作较为简单：我们只需要根据流程图中给出的信号与节拍进行赋值即可，例如 STOP 指令只需要一条 STOP<=W(2)指令即可。

程序编写的难点之处在于如何对 ST0 标志信号进行处理，因为这个信号与节拍脉冲信号紧

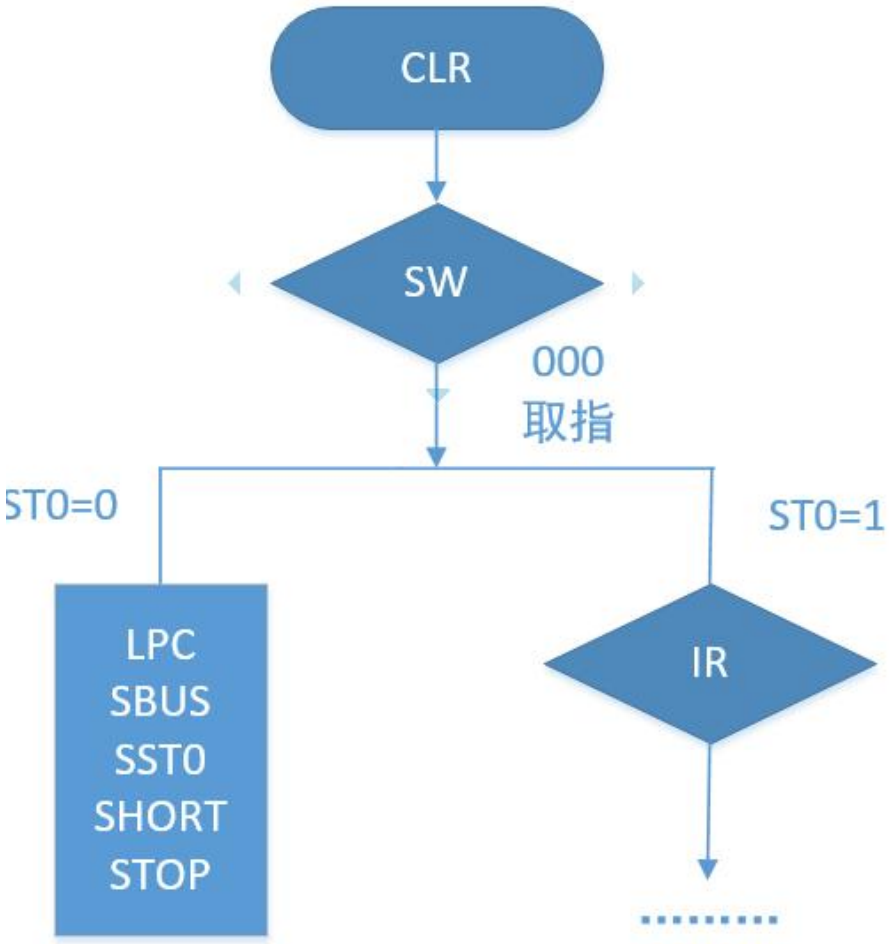
密联系，需要我们对整个程序运行时的时序关系有着清楚的认知。以写寄存器为例，我们简单地介绍一下处理方法:在按下 CLR 之后，我们会将 ST0 的值设为 0，这样我们第一次进入写寄存器操作时会进入左侧的流程。在左侧流程的 W2 周期里，我们会将 SST0 设为 1。此后我们便会检测：当 T3 处于下降沿并且 SST0 为一的时候，给 ST0 赋值为 1（至于为何是 T3 下降沿，我们会在后续开发日志中给出原因）。ST0 为 1，意味着我们下次一会进入右侧的流程，进而实现对 R2,R3 寄存器的更改，至此我们完成了写寄存器的操作。

此外，再简单地介绍一下 TEC-8 模型机是如何寻址的。当我们使 SELCTL=1 时，寄存器的选择便交由 SEL3-SEL0 决定，这个通常会在信号里给出，常用于控制台操作。而当 SELCTL=0 的时候，寄存器通常交由 IR 后四位进行选择，其中高两位为目的寄存器，低两位是源寄存器。

表 3: 指令系统

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

此外我们还实现了修改指令 PC 的功能，这一功能的实现同样运用到了 ST0 标志位，当第一次检测到 SW=000 时，会先允许用户通过数据开关向 PC 寄存器中写入特定内容，达到修改 PC 的效果。此后再检测到 SW=000 时，则会正常的进行 PC+1 和执行指令的操作。



4.2 流水线

流水线部分我们实现了一个简单的二级流水，即我们会在取值的时候同时执行指令，进而实现了缩短一条指令执行所需要的节拍电位，而修改方法也较为简单，只需要在执行指令的信号下面加入 LIR 和 PCINC 两条信号即可（表示 PC+1 和取指），但不同指令具体实现时又有些许的不同，下面将简单的分情况进行介绍：

1. 仅需一个节拍电位的操作（如 ADD）

这是最简单的一种情况，我们只需要在所有操作信号之后加上 LIR 和 PCINC 两条信号即可。

2. 需要两个节拍电位的操作（如 ST 和 LD）

这种情况比上述情况略微复杂一点，我们需要在执行的第二个周期里加入 LIR 和 PCINC 两条信号，如果加在第一个周期当中则会发生吞指现象，产生错误

3. 转移指令（如 JZ,JC）

无论是无条件转移指令还是有条件转移，需要关注的情况就是在发生转移的时候应该如何处理，我们给出方法是指令跳转时不进行取值，待指令跳转结束后进行取值操作（即 W2 周期），这样也可以有效地避免吞指现象

4.3 中断

4.3.1 新增的信号和标志

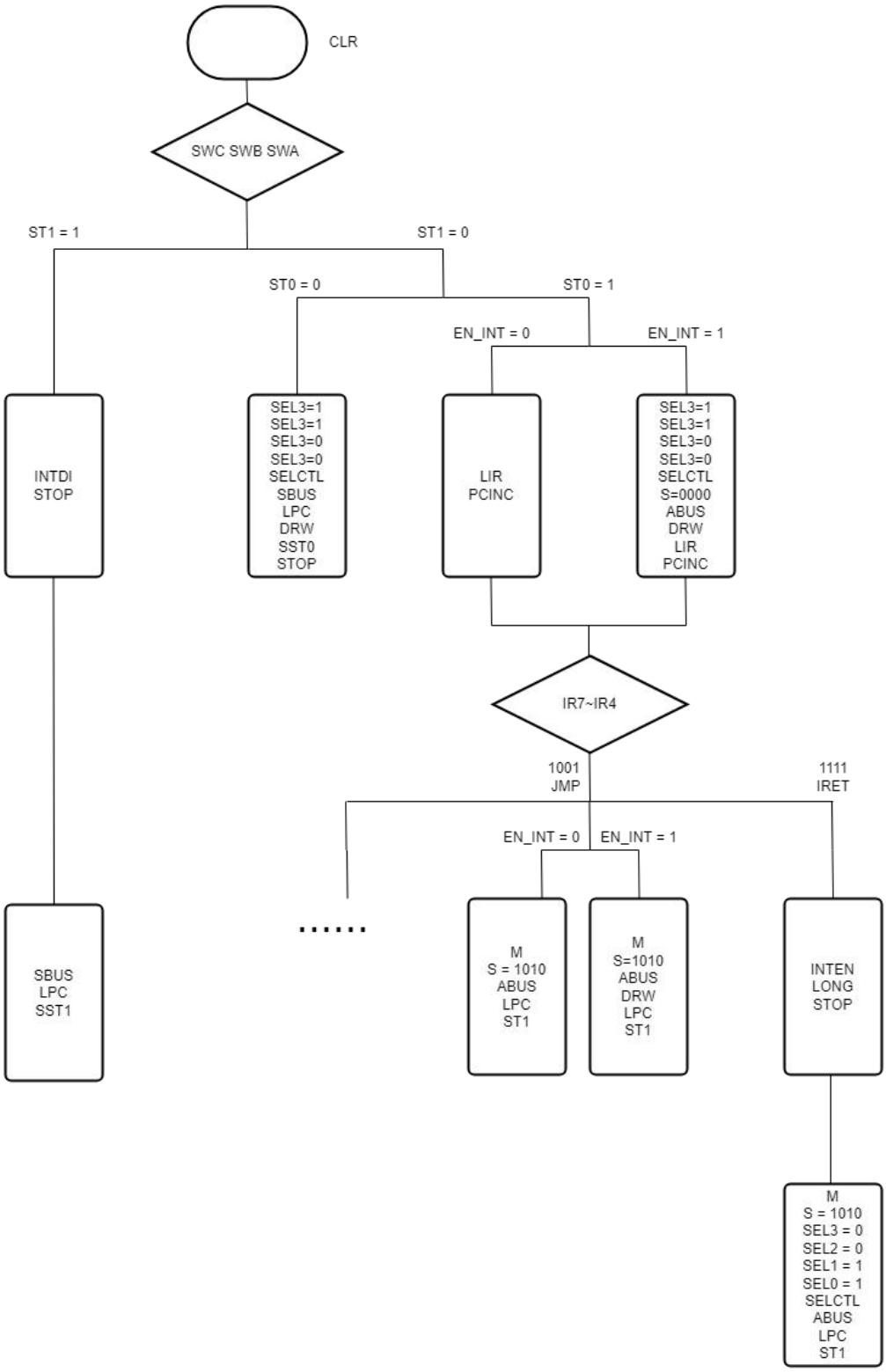
中断程序在基础版程序的基础上新增了一个输入信号 PULSE，表示中断脉冲信号。

程序中新增了一些标志，如表所示：

标志	说明
INT	中断标志
EN_INT	允许中断标志
ST1	中断周期标志
IDTDI	=1 时，置允许中断标志为 0，禁止 TEC-8 模型计算机响应中断请求
INTEN	=1 时，置允许中断标志为 1，允许 TEC-8 模型计算机响应中断请求

4.3.2 中断流程

中断部分的流程图如图所示：



中断时保存当前程序的地址

我们通过 R3 寄存器保存当前程序的地址。为了使程序运行过程中，R3 与 PC 同步，需要对取值阶段的信号以及程序的 JMP 指令做一些改动：

当进入取指阶段时，进行 PCINC 的同时利用 ALU 使 R3 自增一，保持 R3 与 PC 同步。

执行 JMP 指令时，预先设置 IR3-IR2 为 11，选中 R3 寄存器，此时再设置 DRW 为 1，即可将程序即将跳转的地址同时写入 PC 和 R3。

中断周期

按下 CLR 后，默认开中断，置允许中断标志 EN_INT 为 1，中断标志 INT 为 0。

在每个时钟周期的上升沿，根据 INTDI 和 INTEN 来设置允许中断标志 EN_INT。当程序检测到 PULSE 键被按下时，根据允许中断标志来设置 INT 的值。同时，在每条指令执行的最后一个节拍电位的 T3 上升沿，检测 INT，若 INT 为 1，则设置 ST1 为 1，表示即将进入中断周期。

进入中断周期后，首先通过 INTDI 关中断，并通过 STOP 信号暂停程序等待写入 PC 的值。再在 W2 周期的 T3 上升沿将 ST1 置为 0，标志中断周期结束，即将进入中断处理程序。

中断处理程序

在中断处理程序中，R3 不需与 PC 保持同步。由于我们只实现了单级中断，故可以根据允许中断标志 EN_INT 来判断是否正在执行中断处理程序。

IRET 指令用来结束中断处理程序，返回到主程序。在执行 IRET 指令的 W2 周期开中断，W3 周期将 R3 寄存器的值写入到 PC 中，表示返回到主程序，恢复断点。

5.测试程序与测试结果

在整个课设的过程中，我们为每个任务准备了不同的测试程序，其中一部分参考了大二下学期，计算机组成原理实验中的测试程序。最终为了验收，我们将所有测试程序融合为一个程序，用于展示我们所完成的所有功能。

测试程序	助记符	机器代码		寄存器 R0-R3
				00 00 16 13
02H	LD R0, [R3]	01010011	53	85 00 16 13
03H	INC R3	01001100	4C	85 00 16 14
04H	LD R1, [R3]	01010111	57	85 23 16 14
05H	SUB R0, R1	00100001	21	62 23 16 14
06H	JZ 0BH	10000110	86	62 23 16 14
07H	ST R0, [R2]	01101000	68	62 23 16 14
08H	INC R3	01001100	4C	62 23 16 15
09H	LD R0, [R3]	01010011	53	EF 23 16 15
0AH	ADD R0, R1	00010001	11	12 23 16 15
0BH	JC 0EH	0111 0010	72	
0CH	INC R2	01001000	48	
0DH	ST R2, [R2]	01101010	6A	
0EH	AND R0,R1	00110001	31	02 23 16 15
0FH	OR R0,R1	10110001	B1	23 23 16 15
10H	MOVE R2,R0	11001000	C8	23 23 23 15
11H	OUT R3	10100011	A3	23 23 23 15 DBUS=16H
12H	STP	11100000	E0	
13H	85H			
14H	23H			
15H	EFH			
16H	00H			
				00 20 00 00
20H	INC R0	0100 0000	40	循环程序 每次循环寄存器的值不同
21H	INC R0	0100 0000	40	
22H	INC R0	0100 0000	40	
23H	INC R0	0100 0000	40	
24H	INC R0	0100 0000	40	
25H	INC R0	0100 0000	40	
26H	INC R0	0100 0000	40	

```
27H       INC R0                    0100 0000   40
28H       INC R0                    0100 0000   40
29H       INC R0                    0100 0000   40
2AH       JMP [R1]                  1001 1101   9D

45H       ADD R0, R0                0001 0000   10
46H       IRET                      1111 0000   F0
```

这里对测试程序稍作解释：

其中 02H-16H 为流水线与基础版的测试程序，里面几乎包含了指令集中的所有指令（包含拓展的三条指令），同时由于程序并不是从 00H 开始运行，这也可以验证我们的修改 PC 指针的功能。其中最后一列给出了寄存器 R0-R3 值的变化。

而 20H-2AH,45H,46H 则为中断测试程序，主体程序为一个循环加以的程序，这样可以支持我们连续运行。测试过程中重点为观察中断程序结束后返回的 PC 值是否与进入中断时的 PC 值相同，如果相同，则证明我们保存断点并返回断点的功能可以正确执行。

5. 调试过程中遇到的问题与解决方案

5.1 在中断功能的测试中，按下 PULSE 按钮程序并不响应

通过逻辑笔的测试，按下 PULSE 按钮指示灯确实会发生变化，但程序并不响应该信号。起初我们认为是程序无法检测到 PULSE 信号的变化。在不断排查程序以及书写测试程序测试后，我们发现 PULSE 按钮仍无法响应，此时推断是数据通路的问题。通过与其他同学交流，我们得知要想接收到 PULSE 信号，需通过彩虹线进行连接。在正确连接后 PULSE 键仍无法很好地响应。最后，我们用彩虹线将开关连接，指定其中的某一个开关代替 PULSE 键，此时拨动开关，程序做出了正确的响应，解决了这一问题。

6.2 在中断功能的测试中，R3 寄存器无法与 PC 保持同步

查看代码后发现，此时的程序在未进入中断的 W1 周期中会发出取指信号和使 R3 寄存器自

增一的信号，接下来会进入判断 IR7-IR4 的 case 语句块，此时若指令寄存器中的值有效，则会对部分信号进行新的赋值，有可能会覆盖使 R3 自增一的信号，故 R3 无法与 PC 保持同步。

发现这一问题后，我们对代码进行了修改：在执行判断 IR7-IR4 的 case 语句块前，先判断当前所在的周期，若为执行指令的 W2 或 W3 周期，则进入 case 语句块；否则，不进入。代码逻辑修改后，该问题得到了解决。

6.3 在中断功能的测试中，JMP 指令执行结束后，R3 寄存器无法与 PC 保持同步

在对照数据通路推演后发现，这一问题是由于测试程序中的 JMP 指令译码错误。在基础版的程序中，JMP 的指令格式的后两位即 IR1-IR0 并不发挥作用。但在中断这一版程序中，当未进入中断时，若遇到 JMP 指令，不仅 PC 的值要发生跳转，R3 寄存器中的值同样需要发生跳转。此时就需要通过 IR3-IR2 选中 R3 寄存器，在 DRW 的作用下将要跳转到的程序地址写入 R3，实现与 PC 的同步。

故我们修改了测试程序中 JMP 指令的译码结果，该问题得到了解决。

6.4 写寄存器操作时，标志位 ST0 在第二个节拍脉冲时就发生变化

起初我们在用 verilog 语言编写程序，上板测试时会出现写完 R0 寄存器后立马跳转到写 R3 寄存器，我们通过仿真发现，造成这一错误的原因是 ST0 的值在第二个节拍电位中就变为了 1，所以导致了机器没有执行写 R1 寄存器的操作，而是变为执行写 R3 寄存器。后来通过查阅资料我们发现，造成这一错误的原因有两个：其一是因为比起检验节拍脉冲的下降沿，去检验 W(1) 的下降沿是更合适的操作，其二是因为 verilog 语言的高并发性，即使是在 always 语句块中仍然做不到顺序执行。因为第二个原因受限于我们的代码采用的不是组合逻辑，而是采用的顺序

执行，在权衡利弊的情况下，我们最终决定换用 VHDL 语言进行开发。有着 verilog 编写出的程序框架，我们其实很快速地便完成了 VHDL 语言编写的程序，并顺利解决了这一问题。

6. 开发日志

2022/8/22

上午全体组员参与计组课设理论课的教学，大致清楚了我们这门课需要我们完成的任务以及我们所需要具备的一些基础知识，同时老师也为我们提供了部分设计上的思路。下午线下参与的同学来到了实验室，听实验室老师详细地介绍了课程设计的任务以及要求，并初步确定了我们的实验箱编号。课程结束后，小组成员对接下来几天的计划进行了简单地讨论。为了学习更多的知识，扩宽自己的知识面，我们决定选用 verilog 进行开发，于是我们第一天的主要任务就定为了学习 verilog 语言。一天结束后，小组成员基本熟悉了 verilog 的语言，并且可以用 verilog 语法设计一些简单的小部件。

2022/8/23

上午小组成员对 TEC-8 模型机的数据通路进行了测试，主要参考了计组实验一以及实验二的方法，对读写寄存器、读写存储器的功能进行了测试。经测试后发现原来的实验箱数据通路存在诸多 bug，甚至无法从数据开关中读取到正确的数据，我们向老师反映了此问题，征得老师允许后我们更换了新的实验箱，重新进行了数据通路的测试。新实验箱经测试后功能正常。

下午与晚上我们便开始了对硬布线控制器程序的编写，基本产出了框架与大部分内容，但一些细节部分尚未给出合理的解决方案。

2022/8/24

上午小组成员针对昨天遗留的问题，通过在互联网上广泛地收集资料进行学习，最终成功地给出了合适的解决方案，最终我们产出了最初一版的代码。我们随即利用 Quartus II 9.0 自带的仿真工具对工程进行仿真，仿真结果也与我们的预期相符。仿真结束后，我们将工程烧录到芯片当中，准备载入测试程序进行测试。结果发现我们通过硬布线方式进行读写存储器与寄存器的操作并没有如我们预期般地运行，于是我们开始对信号进行一个个地排查。

到了下午我们找到问题所在：ST0 标志位比我们预期中更早地发生了改变。于是我们一步步反推原因，最终将原因锁定在 verilog 语言中 always 语句块中也是并发执行这一点上。小组成员在实验室对代码进行了反复的更改也没能给出合适地解决方案。但是在学习地过程中我们了解到 VHDL 的 process 语句块中的指令是顺序执行的，更加符合我们现有的代码框架，于是我们决定改用 VHDL 语言完成课程设计的任务。我们虽然没能用 verilog 语言顺利完成课设，但在解决 verilog 语句存在的问题以及调试过程中，也让我们对 TEC-8 模型机有了更深刻的认知。

2022/8/25

上午我们很快完成了任务一基础版的 VHDL 代码，有了昨天的经验，我们快速完成了仿真测试与上板烧录，并顺利通过了测试程序。但在测试过程中，有小组成员认为从数据开关写入测试程序的方法可以进行改进，我们可以写一个向存储器里“装载测试程序”的程序，并提出了相关思路。我们与老师讨论了这一思路的可行性，并最终决定让该小组成员做一下尝试，其余成员继续完成任务一的拓展与任务二三。在上午的最后，我们讨论出了任务一拓展的思路。

下午我们将任务一拓展的思路转化为了代码，并顺利通过了仿真测试与上板测试。于是我们马不停蹄地开始研究将原有程序转化为流水线的思路。最后顺利给出了流水线的运行流程图

以及代码。

2022/8/26

上午我们对昨天完成的流水线代码完成仿真与上板烧录测试，程序运行正常，测试结果也符合预期。但是目前的流水并不支持修改指令 PC 与拓指，我们决定将任务一拓展功能在任务二上也进行实现，于是在上午剩余时间里，我们完成了流水线拓展版的代码编写与测试。

下午我们准备开始实现中断功能，但是遗憾地发现芯片并没有连接 LIAR 与 IABUS 的管脚，这就意味着我们无法保存中断时的 PC 值，这让我们的工作一度陷入了停滞。最终在共同思考与探讨之后，我们意识到可以利用寄存器或内存来同步保存 PC 寄存器的值，最终我选择利用 R3 寄存器。这样就解决了 LIAR 寄存器不可使用的问题，顺着这一思路我们继续开展了我们的工作，并在当天完成了代码的编写。

晚上,,尝试做一个内存自动加载程序,思路是利用 vhdl signal 向量数组来保存数据,在 W1 周期通过不断循环 ALU 自增来得到对应 IR 值,在 W2 周期通告数据通路将数据打入内存,在 W3 周期将寄存器中的值清零并进入下一条指令的写入，从而实现将定义在源代码中的数据快捷的写入内存中,免去繁琐的控制台操作。然而 TEC8 的资源有限,利用 vhdl 自带的 Interger 类型,只能保存很少的数据,该程序陷入困境。

2022/8/27

上午我们对完成的中断程序进行上板烧录测试。在未进入中断前，程序无法按照预期执行。经排查我们发现是程序逻辑的问题以及 JMP 指令格式未进行合适的更改所致。修改后，程序可以正常执行了，但无法响应 PULSE 信号。起初我们认为是这一版程序无法检测到 PULSE 信号，便通过书写测试程序的方式来检验 PULSE 信号。在测试程序中，PULSE 信号仍没有效果。

下午我们继续思考该如何检测到 PULSE 信号。在与其他同学交流后我们发现，要通过彩虹线连接 PULSE，其才能发挥作用。在连接后，PULSE 信号仍无法很好地响应。于是，我们用彩虹线将开关连接，将其中的某一个开关替代 PULSE 信号。之后顺利通过了测试。

2022/8/28

上午,尝试对之前自动加载器程序进行优化:将保存 IR 的 interger 型改为 logic_vector,并自己编写了对应的加法器和比较器,然而这次编译提示超出"128"个计算单元，不断精简逻辑后能够正常编译通过,但在整个程序上板时，加法器不能正常运行(单独测试加法器注册),应该是再次遇到了 TEC8 性能瓶颈,无奈只能放弃改思路。

下午线下的同学再次来到实验室。共同梳理了整个程序的思路，并对程序再次进行了测试，没有遇到什么问题。之后一起商量了验收的相关事宜。

2022/8/29

上午我们通过腾讯会议与线上的同学进行验收的对接。之后各成员再次梳理整个课程设计的思路，为验收做最后的准备。

7. 心得体会

刘书恺:

与我而言，这七天我不仅仅学会的是设计一个硬布线控制器，更重要的是在不断调试的过程中，我不断加深了对 TEC-8 模型机的认知，越来越深刻地理解了一条指令、一个程序是如何一步步在数据通路中运行起来。印象较深的一个知识点为 SELCLT 的作用，当我们使 SELCTL=1

时，寄存器的选择便交由 SEL3-SEL0 决定，这个通常会在信号里给出，常用于控制台操作。而当 SELCTL=0 的时候，寄存器通常交由 IR 后四位进行选择，其中高两位为目的寄存器，低两位是源寄存器。

在调试过程中，多次运用了上学期计组实验的知识，弥补了上学期没有实际上机做实验的不足，进而将冯诺依曼体系结构实现了从理论到实践的转化。

在代码层面，最大的收获是加深了对硬件描述语言的理解。在之前的数电实验中，我一直缺乏对硬件描述语言的并发性完整的认识，而本次设计硬布线控制线，本质上就要与节拍电位与节拍脉冲打交道，对时序的控制非常重要。而正是因为缺乏对硬件语言并发性与时序正确的认识，导致我们在对标志信号进行处理时屡屡犯错，调试花了相当长的时间找出错误。

在硬件层面，最大的收获是掌握从硬件到软件的调试方法。刚开始在板子上运行不正确的时候，我们常常面对亮灯不正常的实验箱手足无措，到后面我们慢慢熟悉如何从信号灯反推数据通路的情况，再进一步分析是哪一个模块出现了问题，最后通过仿真找出 bug 所在

吴家宝：

本次课设令我收获很多。在前两年里，我所接触到的编程方面的学习基本都是软件方面的开发，仅在数字逻辑课程中初步体验了一小段时间的硬件开发。而本次课程设计的基础任务便是设计硬连线控制器，在我们将上学期计组的知识应用于实践的过程中，不仅丰富了我硬件开发的经历，同时使我进一步了解了计算机底层设计和组成原理。

在本次课设中，我主要负责的是程序代码的书写和调试工作。

代码编写方面，我们起初采用的是 Verilog 语言。在经过短时间的学习后便开始编写程序。在编写的过程中，由于对 Verilog 了解不深，我们的程序框架与 always 语句块的并发执行产生了冲突，小组成员反复更改后仍未解决这个问题。考虑到时间的紧迫，我们决定换回数字逻辑

课程上学过的 VHDL 语言，解决了这一问题。在编程的过程中，我在计组课程中学到的知识也得到了充分的利用，如数据的流向、ALU 的功能、流水线与中断的原理等等，同时对控制器运作过程中各个信号的作用也有了更加深刻的理解。

在调试工作中，我们首先通过仿真波形来检验程序的正确性，之后再进行上板烧录测试。每次烧录测试都需要手动重新设置寄存器和存储单元的值，在运行过程中不断单拍执行指令，观察相应的信号灯是否发生变化，是一项非常需要细致和耐心的工作。通过这几天的调试，我对大多数指令的执行过程都有了更深入的理解。

总的来说，这次课设是一个很好的实践机会，令我受益匪浅。

王昱杰:

本次实验的关键点在于 TEC-8 模型计算机的整体理解，以及硬布线控制器涉及到的各个控制信号，同时也对我们编写代码、调试代码的能力有更高的要求。第一次接触 TEC-8 还是在一年前的数字逻辑课上，之后计算机组成原理课程实验中,使用 TEC-8 在线仿真程序,相较线下课程,实验过程较为顺利,但感觉线上实验中对知识的理解程度与记忆速度都不如线下实验。线下提供了一块完整，有一些老化问题但却真实,可自定义的 TEC8 面板,我们可以以一种研究和探索的心态来完成这次课程设计，在刚开始基础版编程,调试的时候,由于对语言特性的不熟练,先自己写了许多小的测试程序,来验证每一步逻辑执行的正确性,同时也能发现一些开发板潜在的内部问题。此外,由于每次测试要通过控制台写内存,拨动开关,验证较为繁琐,我希望能通过烧录下载过程自动加载到内存,由于 TEC-8 没有内存直接写入引脚,我便尝试通过数据通路,ALU 运算,Signal 存储单元来综合设计程序进行写入,最终由于该电路板资源有限,我的预期功能并没有实现，但在设计程序,查阅资料的过程中,我也对 FPGA 与单片机有了更深入的理解,我期待能在后续微机接口课程中继续深入学习相关知识。

吴政隆：

由于疫情的原因，我本次参与计算机组成原理课程设计只能在线上进行。由于这门课程需要大量硬件上的操作，且非常重视工程上的设计、调试与实践。线上参与的形式使得我缺席了其它同学花费巨大时间精力的线下调试和验收。坦白地说，这样的形式使得我的工作量远小于其它同学，也影响了课程的教学学习效果。

幸运的是，小组同学每天都会通过在线文档同步硬件描述语言的代码以及实现流程图，借助曾经在数字逻辑课程中学习的 VHDL 知识，我能够通过代码理解其他同学的线下工作，并了解相关设计的思想和原理。同时，通过参与到文档和测试工作中，也让我一定程度弥补了线上学习的不足。

与此同时，我还参与到了线上虚拟仿真实验当中。这一系统具备从高级语言（C 语言）层面到逻辑门电路层面的模拟功能。通过教学模式+考核模式，让我能够在无法返校的情况下，通过虚拟仿真实验了解并掌握实验知识。同时，自顶向下的实验流程使得参与虚拟仿真实验十分有利于掌握整个计算机体系结构的层次和原理。

附录

任务一原码：

```
--指定 PC 指针
library ieee;
use ieee.std_logic_1164.all;

entity nopipe is
    port(
        CLR,CLK,C,Z: in std_logic;
        SW         : in std_logic_vector(2 downto 0);
        IR         : in std_logic_vector(7 downto 4);
        W          : in std_logic_vector(3 downto 1);
        DRW,PCINC,LPC,LAR,PCADD,ARINC,SELCTL,MEMW,STOP,
        LIR,LDZ,LDC,CIN,M,ABUS,SBUS,MBUS,
        SHORT,LONG : out std_logic;
        S,SEL      : out std_logic_vector(3 downto 0)
        );end nopipe;
 architecture behave of nopipe is
        signal ST0,SST0: std_logic;
    begin

        process(CLR,CLK,C,Z,SW,IR,W(1),W(2),W(3),ST0,SST0)
        begin
            DRW     <= '0';
            PCINC   <= '0';
            LPC     <= '0';
            LAR     <= '0';
            PCADD   <= '0';
            ARINC   <= '0';
            SELCTL  <= '0';
            MEMW    <= '0';
            STOP    <= '0';
            LIR     <= '0';
            LDZ     <= '0';
            LDC     <= '0';
            CIN     <= '0';
            M       <= '0';
            ABUS    <= '0';
            SBUS    <= '0';
            MBUS    <= '0';
            SHORT   <= '0';
```

```
LONG    <= '0';
SST0    <= '0';
S       <= "0000";
SEL     <= "0000";

if (CLR = '0') then
    ST0 <= '0';
else
    if ((CLK'event and CLK = '0') and SST0 = '1') then
        ST0 <= '1';
    end if ;

    case( SW ) is
        --写寄存器
        when "100" =>
        SBUS   <= W(1) or W(2);
        SEL(3) <= (ST0 and  W(1)) or (ST0 and  W(2));
        SEL(2) <= W(2);
        SEL(1) <= (not ST0 and W(1)) or (ST0 and W(2));
        SEL(0) <= W(1);
        SELCTL <= W(1) or W(2);
        DRW    <= W(1) or W(2);
        STOP   <= W(1) or W(2);
        SST0   <= W(2);

        --读寄存器
        when "011" =>
        SEL(3) <= W(2);
        SEL(2) <= '0';
        SEL(1) <= W(2);
        SEL(0) <= W(1) or W(2);
        SELCTL <= W(1) or W(2);
        STOP   <= W(1) or W(2);

        --读存储器
        when "010" =>
        SBUS   <= (not ST0) and W(1);
        LAR    <= (not ST0) and W(1);
        SST0   <= (not ST0) and W(1);

        STOP   <= W(1);
        SELCTL <= W(1);
        SHORT  <= W(1);
```

```
MBUS   <= W(1) and ST0;
ARINC  <= W(1) and ST0;

--写寄存器
when "001" =>
LAR    <= (not ST0) and W(1);
SST0   <= (not ST0) and W(1);--????

STOP   <= W(1);
SELCTL <= W(1);
SHORT  <= W(1);
SBUS   <= W(1);

MEMW   <= W(1) and ST0;
ARINC  <= W(1) and ST0;

--取指
when "000" =>
                                    if ST0 = '0' then
        LPC   <= W(1);
        SBUS  <= W(1);
        SHORT <= W(1);
        STOP  <= W(1);
        SST0  <= W(1);
        else
        LIR   <= W(1);
        PCINC <= W(1);
        case( IR ) is
            --ADD
            when "0001" =>
            S    <= W(2) & '0' & '0' & W(2);
            CIN  <= W(2);
            ABUS <= W(2);
            DRW  <= W(2);
            LDZ  <= W(2);
            LDC  <= W(2);

            --SUB
            when "0010" =>
            S    <= '0' & W(2) & W(2) & '0';
            ABUS <= W(2);
            DRW  <= W(2);
            LDZ  <= W(2);
            LDC  <= W(2);
```

```
                --AND
                when "0011" =>
                S    <= W(2) & '0' & W(2) & W(2);
                ABUS <= W(2);
                DRW  <= W(2);
                LDZ  <= W(2);

                --INC
                when "0100" =>
                S    <= '0' & '0' & '0' & '0';
                ABUS <= W(2);
                DRW  <= W(2);
                LDZ  <= W(2);
                LDC  <= W(2);

                --LD
                when "0101" =>
                S    <= W(2) & '0' & W(2) & '0';
                M    <= W(2);
                ABUS <= W(2);
                LAR  <= W(2);
                LONG <= W(2);
                DRW  <= W(3);
                MBUS <= W(3);

                --ST
                when "0110" =>
                S    <= ( W(2) or W(3)) &  W(2) & ( W(2) or W(3)) &
W(2);
                M    <= W(2) or W(3);
                ABUS <= W(2) or W(3);

                LAR  <= W(2);
                LONG <= W(2);
                MEMW <= W(3);

                --JC
                when "0111" =>
                PCADD <= W(2) and C;

                --JZ
                when "1000" =>
                PCADD <= W(2) and Z;
```

```
                    --JMP
                    when "1001" =>
                    S     <= W(2) & W(2) & W(2) & W(2);
                    M     <= W(2);
                    ABUS  <= W(2);
                    LPC   <= W(2);

                    --OUT
                    when "1010" =>
                    S    <= W(2) & '0' & W(2) & '0';
                    M  <= W(2);
                    ABUS <= W(2);

                    --OR
                    when "1011" =>
                    S    <= W(2) & W(2) & W(2) & '0';
                    M    <= W(2);
                    ABUS <= W(2);
                    DRW  <= W(2);
                    LDZ  <= W(2);

                    --MOVE
                    when "1100" =>
                    S    <= W(2) & '0' & W(2) & '0';
                    M  <= W(2);
                    ABUS <= W(2);
                    DRW  <= W(2);

                    --STP
                    when "1110" =>
                    STOP  <= W(2);

                    when others =>null;

                end case ;

            end if;

        when others =>null;

    end case ;

end if;
```

```
        end process;

    end behave;
```

任务二原码;

```
library ieee;use ieee.std_logic_1164.all;
 entity nopipe is
    port(
        CLR,CLK,C,Z: in std_logic;
        SW         : in std_logic_vector(2 downto 0);
        IR         : in std_logic_vector(7 downto 4);
        W          : in std_logic_vector(3 downto 1);
        DRW,PCINC,LPC,LAR,PCADD,ARINC,SELCTL,MEMW,STOP,
        LIR,LDZ,LDC,CIN,M,ABUS,SBUS,MBUS,
        SHORT,LONG : out std_logic;
        S,SEL      : out std_logic_vector(3 downto 0)
        );end nopipe;
 architecture behave of nopipe is
        signal ST0,SST0: std_logic;
    begin

        process(CLR,CLK,C,Z,SW,IR,W(1),W(2),W(3),ST0,SST0)
        begin
            DRW     <= '0';
            PCINC   <= '0';
            LPC     <= '0';
            LAR     <= '0';
            PCADD   <= '0';
            ARINC   <= '0';
            SELCTL  <= '0';
            MEMW    <= '0';
            STOP    <= '0';
            LIR     <= '0';
            LDZ     <= '0';
            LDC     <= '0';
            CIN     <= '0';
            M       <= '0';
            ABUS    <= '0';
            SBUS    <= '0';
            MBUS    <= '0';
            SHORT   <= '0';
            LONG    <= '0';
```

```
SST0    <= '0';
S       <= "0000";
SEL     <= "0000";

if (CLR = '0') then
    ST0 <= '0';
else
    if ((CLK'event and CLK = '0') and SST0 = '1') then
        ST0 <= '1';
    end if ;

    case( SW ) is
        --写寄存器
        when "100" =>
        SBUS   <= W(1) or W(2);
        SEL(3) <= (ST0 and  W(1)) or (ST0 and  W(2));
        SEL(2) <= W(2);
        SEL(1) <= (not ST0 and W(1)) or (ST0 and W(2));
        SEL(0) <= W(1);
        SELCTL <= W(1) or W(2);
        DRW    <= W(1) or W(2);
        STOP   <= W(1) or W(2);
        SST0   <= W(2);

        --读寄存器
        when "011" =>
        SEL(3) <= W(2);
        SEL(2) <= '0';
        SEL(1) <= W(2);
        SEL(0) <= W(1) or W(2);
        SELCTL <= W(1) or W(2);
        STOP   <= W(1) or W(2);

        --读存储器
        when "010" =>
        SBUS   <= (not ST0) and W(1);
        LAR    <= (not ST0) and W(1);
        SST0   <= (not ST0) and W(1);

        STOP   <= W(1);
        SELCTL <= W(1);
        SHORT  <= W(1);

        MBUS   <= W(1) and ST0;
```

```
ARINC  <= W(1) and ST0;

--写寄存器
when "001" =>
LAR    <= (not ST0) and W(1);
SST0   <= (not ST0) and W(1);--????

STOP   <= W(1);
SELCTL <= W(1);
SHORT  <= W(1);
SBUS   <= W(1);

MEMW   <= W(1) and ST0;
ARINC  <= W(1) and ST0;

--取指
when "000" =>   --取指
    if ST0 = '0' then
        LPC   <= W(1);
        SBUS  <= W(1);
        SHORT <= W(1);
        STOP  <= W(1);
        SST0  <= W(1);
    else
        case IR is
            when "0001" =>  --ADD
                S <= "1001";
                CIN <= W(1);
                ABUS <= W(1);
                DRW <= W(1);
                LDZ <= W(1);
                LDC <= W(1);

                LIR <= W(1);
                PCINC <= W(1);
                SHORT <= w(1);
            when "0010" =>  --SUB
                S <= "0110";
                ABUS <= W(1);
                DRW <= W(1);
                LDZ <= W(1);
                LDC <= W(1);

                LIR <= W(1);
```

```
        PCINC <= W(1);
        SHORT <= w(1);
    when "0011" =>  --AND
        M <= W(1);
        S <= "1011";
        ABUS <= W(1);
        DRW <= W(1);
        LDZ <= W(1);

        LIR <= W(1);
        PCINC <= W(1);
        SHORT <= w(1);
    when "0100" =>  --INC
        S <= "0000";
        ABUS <= W(1);
        DRW <= W(1);
        LDZ <= W(1);
        LDC <= W(1);

        LIR <= W(1);
        PCINC <= W(1);
        SHORT <= w(1);
    when "0101" =>  --LD
        M <= W(1);
        S <= W(1) & '0' & W(1) & '0';
        ABUS <= W(1);
        LAR <= W(1);
        MBUS <= W(2);
        DRW <= W(2);

        LIR <= W(2);
        PCINC <= W(2);
    when "0110" =>  --ST
        M <= W(1) or W(2);
        S <= '1' & W(1) & '1' & W(1);
        ABUS <= W(1) or W(2);
        LAR <= W(1);
        MEMW <= W(2);

        LIR <= W(2);
        PCINC <= W(2);
    when "0111" =>  --JC
        PCADD <= C and W(1);
```

```
    LIR <=  (W(1) and (not C)) or (W(2) and C);
    PCINC <= (W(1) and (not C)) or (W(2) and C);
    SHORT <= W(1) and (not C);
when "1000" =>  --JZ
    PCADD <= Z and W(1);

    LIR <=  (W(1) and (not Z)) or (W(2) and Z);
    PCINC <= (W(1) and (not Z)) or (W(2) and Z);
    SHORT <= W(1) and (not Z);
when "1001" =>  --JMP
    M <= W(1);
    S <= "1111";
    ABUS <= W(1);
    LPC <= W(1);

    LIR <= W(2);
    PCINC <= W(2);
when "1110" =>  --STP
    STOP <= W(1);

when "1010" =>  --OUT
    S    <= W(1) & '0' & W(1) & '0';
    M    <= W(1);
    ABUS <= W(1);

    LIR <= W(1);
    PCINC <= W(1);
    SHORT <= w(1);
when "1011" =>  --OR
    S    <= W(1) & W(1) & W(1) & '0';
    M    <= W(1);
    ABUS <= W(1);
    DRW  <= W(1);
    LDZ  <= W(1);

    LIR <= W(1);
    PCINC <= W(1);
    SHORT <= w(1);
when "1100" =>  --MOVE
    S    <= W(1) & '0' & W(1) & '0';
    M  <= W(1);
    ABUS <= W(1);
    DRW  <= W(1);
```

```
                                LIR <= W(1);
                                PCINC <= W(1);
                                SHORT <= w(1);
                            when others =>
                                LIR <= W(1);
                                PCINC <= W(1);
                                SHORT <= W(1);
                        end case;
                    end if;

                when others =>null;

            end case ;

        end if;

    end process;

end behave;
```

任务三原码：

```
library ieee;use ieee.std_logic_1164.all;
 entity nopipe is
    port(
        CLR,CLK,C,Z: in std_logic;
        SW         : in std_logic_vector(2 downto 0);
        IR         : in std_logic_vector(7 downto 4);
        W          : in std_logic_vector(3 downto 1);
        DRW,PCINC,LPC,LAR,PCADD,ARINC,SELCTL,MEMW,STOP,
        LIR,LDZ,LDC,CIN,M,ABUS,SBUS,MBUS,
        SHORT,LONG : out std_logic;
        S,SEL      : out std_logic_vector(3 downto 0)
        );end nopipe;
 architecture behave of nopipe is
        signal ST0,SST0: std_logic;begin

    process(CLR,CLK,C,Z,SW,IR,W(1),W(2),W(3),ST0,SST0)
    begin
        DRW     <= '0';
        PCINC   <= '0';
        LPC     <= '0';
        LAR     <= '0';
        PCADD   <= '0';
```

```
ARINC   <= '0';
SELCTL  <= '0';
MEMW    <= '0';
STOP    <= '0';
LIR     <= '0';
LDZ     <= '0';
LDC     <= '0';
CIN     <= '0';
M       <= '0';
ABUS    <= '0';
SBUS    <= '0';
MBUS    <= '0';
SHORT   <= '0';
LONG    <= '0';
SST0    <= '0';
S       <= "0000";
SEL     <= "0000";

if (CLR = '0') then
    ST0 <= '0';
else
    if ((CLK'event and CLK = '0') and SST0 = '1') then
        ST0 <= '1';
    end if ;

    case( SW ) is
        --写寄存器
        when "100" =>
        SBUS   <= W(1) or W(2);
        SEL(3) <= (ST0 and  W(1)) or (ST0 and  W(2));
        SEL(2) <= W(2);
        SEL(1) <= (not ST0 and W(1)) or (ST0 and W(2));
        SEL(0) <= W(1);
        SELCTL <= W(1) or W(2);
        DRW    <= W(1) or W(2);
        STOP   <= W(1) or W(2);
        SST0   <= W(2);

        --读寄存器
        when "011" =>
        SEL(3) <= W(2);
        SEL(2) <= '0';
        SEL(1) <= W(2);
        SEL(0) <= W(1) or W(2);
```

```
SELCTL <= W(1) or W(2);
STOP   <= W(1) or W(2);

--读存储器
when "010" =>
SBUS   <= (not ST0) and W(1);
LAR    <= (not ST0) and W(1);
SST0   <= (not ST0) and W(1);

STOP   <= W(1);
SELCTL <= W(1);
SHORT  <= W(1);

MBUS   <= W(1) and ST0;
ARINC  <= W(1) and ST0;

--写寄存器
when "001" =>
LAR    <= (not ST0) and W(1);
SST0   <= (not ST0) and W(1);--????

STOP   <= W(1);
SELCTL <= W(1);
SHORT  <= W(1);
SBUS   <= W(1);

MEMW   <= W(1) and ST0;
ARINC  <= W(1) and ST0;

--取指
when "000" =>   --取指
    if ST0 = '0' then
        LPC   <= W(1);
        SBUS  <= W(1);
        SHORT <= W(1);
        STOP  <= W(1);
        SST0  <= W(1);
    else
        case IR is
            when "0001" =>  --ADD
                S <= "1001";
                CIN <= W(1);
                ABUS <= W(1);
                DRW <= W(1);
```

```
        LDZ <= W(1);
        LDC <= W(1);

        LIR <= W(1);
        PCINC <= W(1);
        SHORT <= w(1);
    when "0010" =>  --SUB
        S <= "0110";
        ABUS <= W(1);
        DRW <= W(1);
        LDZ <= W(1);
        LDC <= W(1);

        LIR <= W(1);
        PCINC <= W(1);
        SHORT <= w(1);
    when "0011" =>  --AND
        M <= W(1);
        S <= "1011";
        ABUS <= W(1);
        DRW <= W(1);
        LDZ <= W(1);

        LIR <= W(1);
        PCINC <= W(1);
        SHORT <= w(1);
    when "0100" =>  --INC
        S <= "0000";
        ABUS <= W(1);
        DRW <= W(1);
        LDZ <= W(1);
        LDC <= W(1);

        LIR <= W(1);
        PCINC <= W(1);
        SHORT <= w(1);
    when "0101" =>  --LD
        M <= W(1);
        S <= W(1) & '0' & W(1) & '0';
        ABUS <= W(1);
        LAR <= W(1);
        MBUS <= W(2);
        DRW <= W(2);
```

```
    LIR <= W(2);
    PCINC <= W(2);
when "0110" =>  --ST
    M <= W(1) or W(2);
    S <= '1' & W(1) & '1' & W(1);
    ABUS <= W(1) or W(2);
    LAR <= W(1);
    MEMW <= W(2);

    LIR <= W(2);
    PCINC <= W(2);
when "0111" =>  --JC
    PCADD <= C and W(1);

    LIR <=  (W(1) and (not C)) or (W(2) and C);
    PCINC <= (W(1) and (not C)) or (W(2) and C);
    SHORT <= W(1) and (not C);
when "1000" =>  --JZ
    PCADD <= Z and W(1);

    LIR <=  (W(1) and (not Z)) or (W(2) and Z);
    PCINC <= (W(1) and (not Z)) or (W(2) and Z);
    SHORT <= W(1) and (not Z);
when "1001" =>  --JMP
    M <= W(1);
    S <= "1111";
    ABUS <= W(1);
    LPC <= W(1);

    LIR <= W(2);
    PCINC <= W(2);
when "1110" =>  --STP
    STOP <= W(1);

when "1010" =>  --OUT
    S    <= W(1) & '0' & W(1) & '0';
    M    <= W(1);
    ABUS <= W(1);

    LIR <= W(1);
    PCINC <= W(1);
    SHORT <= w(1);
when "1011" =>  --OR
    S    <= W(1) & W(1) & W(1) & '0';
```

```
                    M    <= W(1);
                    ABUS <= W(1);
                    DRW  <= W(1);
                    LDZ  <= W(1);

                    LIR <= W(1);
                    PCINC <= W(1);
                    SHORT <= w(1);
                when "1100" =>  --MOVE
                    S    <= W(1) & '0' & W(1) & '0';
                    M  <= W(1);
                    ABUS <= W(1);
                    DRW  <= W(1);

                    LIR <= W(1);
                    PCINC <= W(1);
                    SHORT <= w(1);
                when others =>
                    LIR <= W(1);
                    PCINC <= W(1);
                    SHORT <= W(1);
            end case;
        end if;

        when others =>null;

    end case ;

  end if;

 end process;
end behave;
```

内存加载器源码(失败):

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

 entity nopipe is
   port(
      CLR,CLK,C,Z: in std_logic;
```

```
        PULSE      : in std_logic;
        SW         : in std_logic_vector(2 downto 0);
        IR         : in std_logic_vector(7 downto 4);
        W          : in std_logic_vector(3 downto 1);
        DRW,PCINC,LPC,LAR,PCADD,ARINC,SELCTL,MEMW,STOP,
        LIR,LDZ,LDC,CIN,M,ABUS,SBUS,MBUS,
        SHORT,LONG : out std_logic;
        S,SEL      : out std_logic_vector(3 downto 0)
        );end nopipe;
architecture behave of nopipe is
   signal ST0, SST0, ST1, SST1     : std_logic;
              type IRs_sturct IS ARRAY(31 downto 0) of std_logic_vector(7 downto 0) ;
              signal IRs:IRs_sturct;
              --signal cnt:integer;
              signal lastPC:integer:=2;
              --signal PC:integer:=0;

   signal pcval :std_logic_vector(7 downto 0);
   signal pclen :integer;
   signal pc    :integer:=0;
   signal cnt   :std_logic_vector(7 downto 0);                begin
              IRs(0) <="00000000";
              IRs(1) <="00000001";
              IRs(2) <="00000011";
              IRs(3) <="00000111";
              IRs(4) <="00001111";
              IRs(5) <="00011111";
              IRs(6) <="00111111";
              IRs(7) <="01111111";
              pclen <= 7;

   process(CLR,CLK,PULSE,SW,IR,W(1),W(2),W(3),ST0,SST0,ST1,SST1)
                 function equal(sa,sb:std_logic_vector(7 downto 0)) return boolean is
                     variable a,b :std_logic_vector(7 downto 0);
          begin
                          a:=sa(7 downto 0); b:=sb(7 downto 0);
                              if a(0)=b(0) and a(1)=b(1)
                                  and a(2)=b(2) and a(3)=b(3)
                                  and a(4)=b(4) and a(5)=b(5)
                                  and a(6)=b(6) and a(7)=b(7) then
                                      return true;
                              else
                                      return false;
                              end if;
```

```
    end equal;

function inc(a:std_logic_vector(7 downto 0)) return std_logic_vector is
        variable res :std_logic_vector(7 downto 0);
begin
        res:=a(7 downto 0);
        if res(0)='0' then
                res(0):='1';return res;
        else res(0):='0';
        end if;

        if res(1)='0' then
                res(1):='1';return res;
        else res(1):='0';
        end if;

        if a(2)='0' then
                res(2):='1';return res;
        else res(2):='0';
        end if;

        if res(3)='0' then
                res(3):='1';return res;
        else res(3):='0';
        end if;

        if res(4)='0' then
                res(4):='1';return res;
        else res(4):='0';
        end if;

        if res(5)='0' then
                res(5):='1';return res;
        else res(5):='0';
        end if;

        if res(6)='0' then
                res(6):='1';return res;
        else res(6):='0';
        end if;

        if res(7)='0' then
                res(7):='1';return res;
        else res(7):='0';
```

```
            end if;
        return res;
        end inc;


    begin

DRW     <= '0';
PCINC   <= '0';
LPC     <= '0';
LAR     <= '0';
PCADD   <= '0';
ARINC   <= '0';
SELCTL  <= '0';
MEMW    <= '0';
STOP    <= '0';
LIR     <= '0';
LDZ     <= '0';
LDC     <= '0';
CIN     <= '0';
M       <= '0';
ABUS    <= '0';
SBUS    <= '0';
MBUS    <= '0';
SHORT   <= '0';
LONG    <= '0';
S       <= "0000";
SEL     <= "0000";


SST0    <= '0';
SST1    <= '0';

if (CLR = '0') then
                        cnt<="00000000";
else case( SW ) is
    --quick write
    when "000" =>
    cnt<=inc(cnt);
    cnt<=inc(cnt);
    cnt<=inc(cnt);
    cnt<=inc(cnt);
    if(cnt="00000000") then
```

```
                STOP <= '1';
            end if;
            pcval <= "00000011";
if(PC=pclen) then
                        STOP<='1';
                    end if;

                    if (W(1)='1' and not equal(cnt,"00000100")) then
                            cnt<=inc(cnt);
                            SHORT <= W(1);
                            S <= "0000";
    SELCTL <= W(1);
                            CIN <= '0';
                            ABUS <= '1';
                            DRW <= '1';
                    end if;

if(W(2)='1') then
                            LONG <='1';
                            S <= "1111";
                            CIN <= '0';
                            ABUS <= '1';
                            ARINC  <= '1';
                            MEMW   <= '1';
                            ARINC <= '1';
                    end if;

if(W(3)='1') then
                            S <= "0011";
                            ABUS <= '1';
                            DRW <= '1';
                            pc <= pc+1;
                    end if;

--write to memory
                    when "100" =>
    SBUS   <= W(1) or W(2);
    SEL(3) <= (ST0 and  W(1)) or (ST0 and  W(2));
    SEL(2) <= W(2);
    SEL(1) <= (not ST0 and W(1)) or (ST0 and W(2));
    SEL(0) <= W(1);
    SELCTL <= W(1) or W(2);
    DRW    <= W(1) or W(2);
    STOP   <= W(1) or W(2);
```

```
            SST0   <= W(2);

        --read from register
        when "011" =>
            SEL(3) <= W(2);
            SEL(2) <= '0';
            SEL(1) <= W(2);
            SEL(0) <= W(1) or W(2);
            SELCTL <= W(1) or W(2);
            STOP   <= W(1) or W(2);

        --read from memory
        when "010" =>
            SBUS   <= (not ST0) and W(1);
            LAR    <= (not ST0) and W(1);
            SST0   <= (not ST0) and W(1);

            STOP   <= W(1);
            SELCTL <= W(1);
            SHORT  <= W(1);

            MBUS   <= W(1) and ST0;
            ARINC  <= W(1) and ST0;

        --write to memory
        when "001" =>
            LAR    <= (not ST0) and W(1);
            SST0   <= (not ST0) and W(1);--????

            STOP   <= W(1);
            SELCTL <= W(1);
            SHORT  <= W(1);
            SBUS   <= W(1);

            MEMW   <= W(1) and ST0;
            ARINC  <= W(1) and ST0;

        when others =>null;
      end case ;
     end if;
  end process;
end behave;
```