

南京農業大學

本科生课程设计

结题报告

题 目: 基于 TEC-5 指令的五级流水 CPU 的设计与实现

课 程: 计算机组成原理与系统结构课程设计

姓 名: 黄锦鹤

班 级: 计科 171

学 号: 19217124

指导教师: 赵力

2019 年 9 月 21 日

南京农业大学信息科技学院

1、主要功能

- 1) 完成 TEC-5 运算器设计与测试
- 2) 完成存储器设计与测试
- 3) 完成 TEC-5 基本指令集的数据通路
- 4) 完成微程序控制器设计与测试
- 5) 完成流水线系统的设计与测试

2、组队情况

小组成员：

范毛烨：元件的测试和修改，指令的测试，毛刺现象的解决，控制冒险元件的设计。

黄锦鹤：框架搭建，时序，旁路，多选器和寄存器的设计。

成永康：段寄存器，控制器，指令存储器，数据存储器的设计。

3、创新之处

在 TEC-5 上实现了流水线系统，并且很好的解决了冒险问题。

目录

摘要	错误!未定义书签。
关键词	错误!未定义书签。
Abstract	错误!未定义书签。
Key words	错误!未定义书签。
1 整体设计	5
1.1 设计方案	5
1.2 数据通路图	7
2 指令系统	7
2.1 指令格式	7
2.2 指令的分类和功能	7
2.2.1 算术逻辑运算指令	7
2.2.2 数据传输类指令	7
2.2.3 程序控制类指令	7
2.3 指令设计	8
2.4 微指令设计	9
3 流水线计算机设计	10
3.1 流水段寄存器设计	10
3.2 存储器设计	11
3.2.1 指令存储器设计	11
3.2.2 数据存储器	12
3.3 微程序控制器	13
2.5 数据选择器	13
2.6 旁路单元（定向传送单元）	14
3.4 时序信号控制 时序发生器	16
3 测试	18
4 遇到的问题与处理	27
4.1 Proteus 74ls181 存在问题	27
4.2 毛刺现象	29
5 心得体会	32

基于 TEC-5 指令的五级流水计算机的设计与实现

计算机科学与技术专业学生 黄锦鹤

指导教师 赵力

摘要：本次实验基于 2019 年的计算机组装与原理课程设计，以及《计算机组装与设计——硬件软件接口》一书的 MIPS 流水线结构图实现了基于流水线结构和 TEC-5 指令的 8 位 CPU 的功能，基本上完成了在硬件仿真平台 PROTEUS 上实现 TEC-5 模型机的功能。在

本次课设上最终完成的系统由运算器、控制器、时钟脉冲信号发生器、双端口通用寄存器堆和指令数据存储器等组成，能够在手动模式和自动模式下执行，手动输入数据之后可切换到自动模式执行指令，在自动模式下能以流水线的方式自动执行 ADD, SUB 等 7 条指令。

关键词：TEC-5; 硬件仿真; 8 位计算机; 流水线;

Design and Implementation of 8-bit Computer Based on Electrical Schematic

Student majoring in Network Project Jinhe Huang
Tutor ZHAO Li

Abstract: This experiment is based on the 2019 computer composition principle course design, and the MIPS pipeline structure diagram of "Computer Composition and Design - Hardware Software Interface", which realizes the function of 8-bit CPU based on pipeline structure and TEC-5 instruction. The function of implementing the TEC-5 model machine on the hardware simulation platform Proteus was completed. The final system completed in this lesson consists of an arithmetic unit, a controller, a clock pulse generator, a dual-port general-purpose register file, and an instruction data memory. It can simulate TEC-5 in manual mode and automatic mode, and manually input it. After the data can be switched to the automatic mode execution instruction, in the automatic mode, 7 instructions such as ADD, SUB, etc. can be automatically executed and executed in a pipelined manner.

Key words: TEC-5; hardware simulation; 8-bit computer; pipeline;

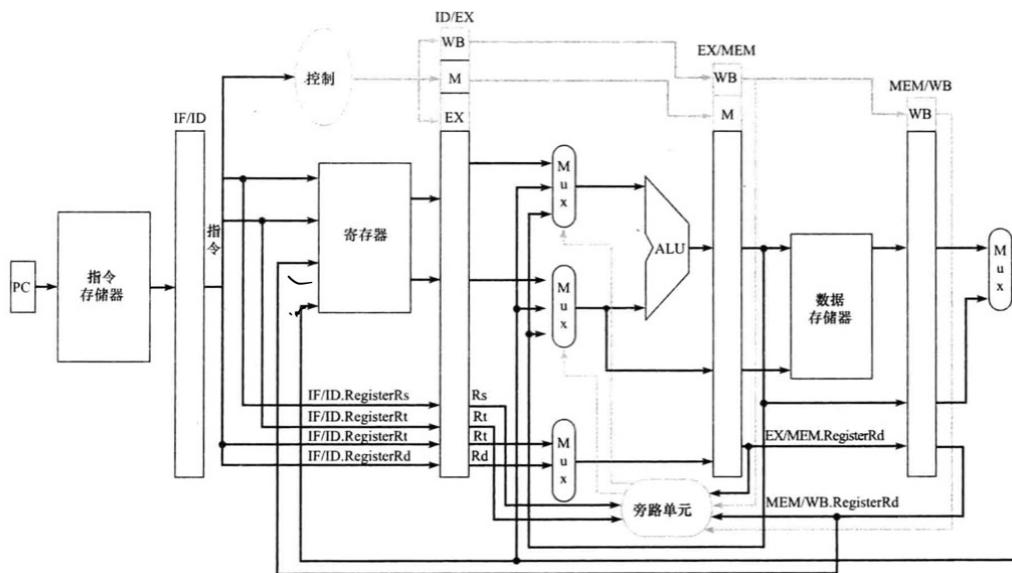
引言

1 整体设计

1.1 设计方案

在流水线设计前，对单周期 CPU 和 TEC-5 进行分析，确定了将操作分为取指令 (IF)、译码与访问寄存器 (ID)、执行 (EX)、访存 (MEM)、写回 (WB) 五段。在设计时先不确定执行时间最长的阶段，先搭建数据通路再来设计时钟周期与时序发生单元。

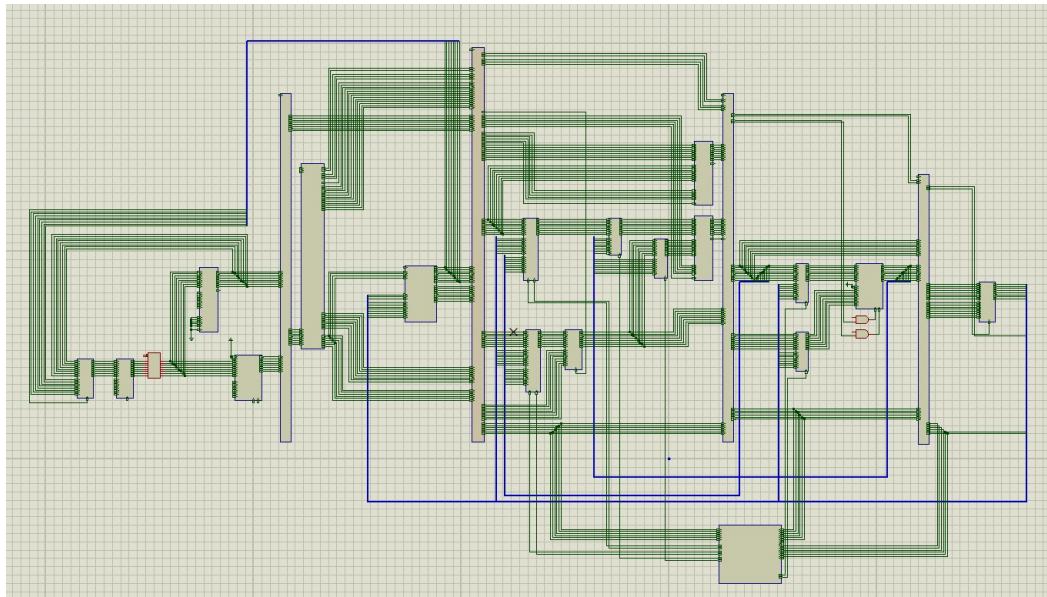
设计初期我们确定使用《计算机组成与设计 硬件软件接口中》MIPS32 流水线模型搭建一个原型。



不同与此模型的是，此模型是 8 位机，同时写回默认是 RD，没有 RT 操作数，保持与 TEC-5 指令相同的操作方式，保留了立即数接口，但没有 TEC-5 的总线结构，并删除了符号拓展器件。此外，我们采用了类似微程序控制器的思想，将某条指令译码后需要的各阶段微操作存在一个单元里，并将产生的操作信号送入第一个流水段寄存器中，而未使用的信号随着下一个流水段周期送入下一级流水段寄存器中，实现了控制信号的逐级传输。

在旁路设计中，与《计算机组成与设计 硬件软件接口中》MIPS32 流水线模型不同的是，我们引入了数据存储器向执行段操作数的一条旁路，解决了 LOAD 指令引发的数据冒险问题，并使得流水线不需要插入 NOP 指令（空指令）来阻塞。使得流水线理论 CPI 接近 1。

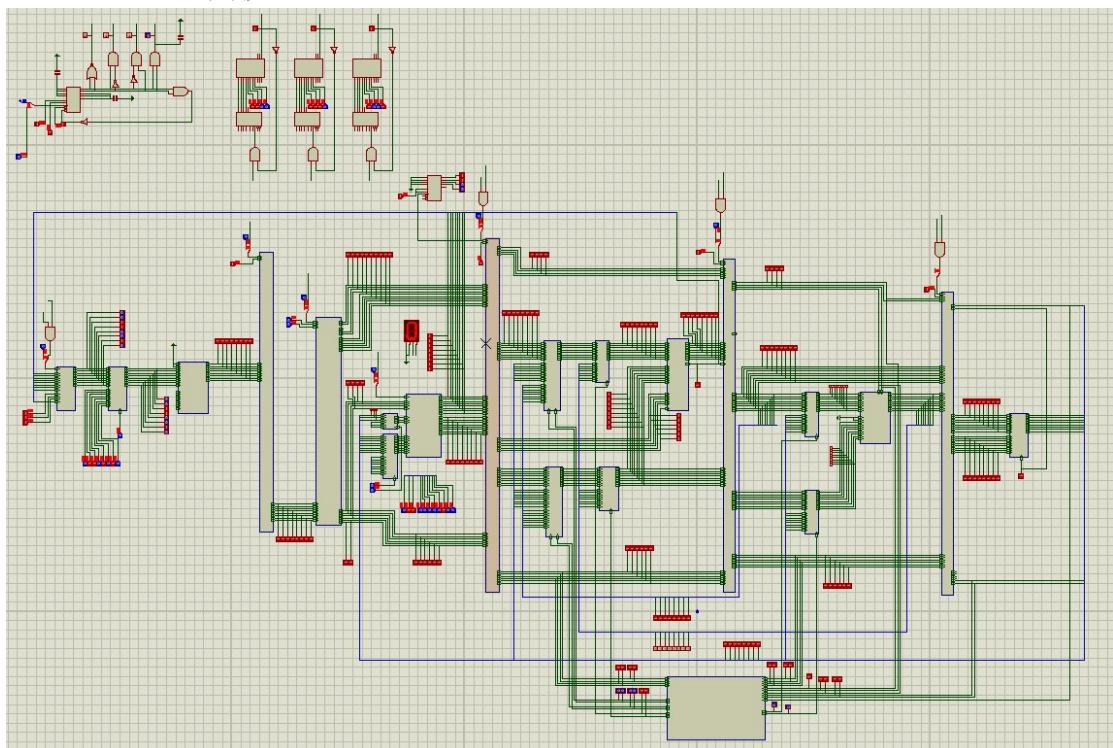
在搭建完基础的数据通路与设计的旁路（定向前送路径）后，完成了一个如下图不包含器件具体实现的设计原型。



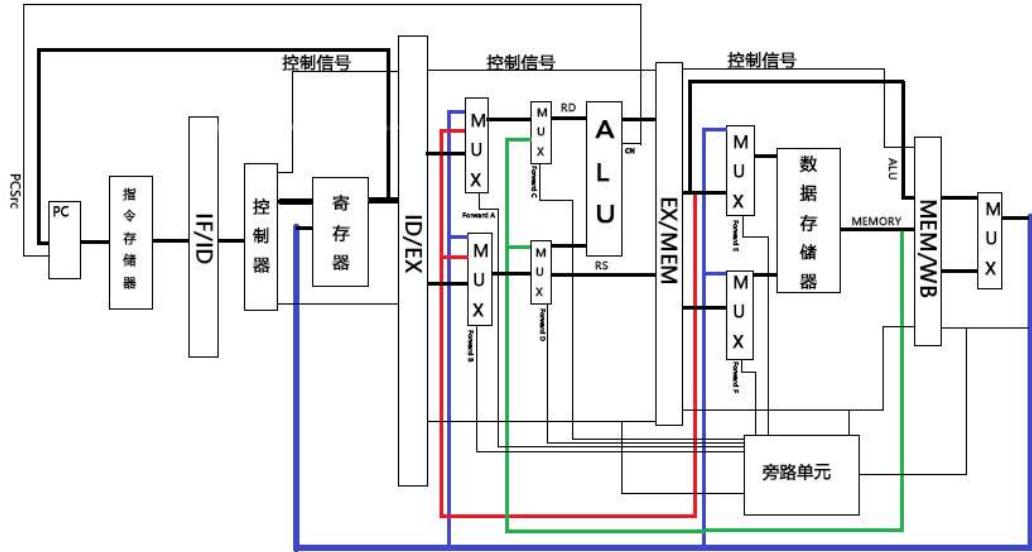
在此原型上，我们初步实现了各个封装的子图包括 ALU、寄存器、指令存储器、数据存储器、控制器、数据多选器、流水段寄存器、旁路单元。

通过对这个模型分析，TEC-5 指令能够实现的立即数输入只有四位，仅能采用分高四位或第四位输入数据，在数据写入时由于内存和寄存器的组织结构为 8 位一个单位，会导致写入的某个单元高四位或低四位数据丢失，因此我们舍弃了在 8 位机上实现立即数。此外我们舍弃了 EX 阶段地址运算的器件，按照 TEC-5 的 JC 指令，PC 的地址来源来源于寄存器堆，由此我们将 EX 段的条件转移提前到 JC 段，并避免了一类控制冒险（此时 JC 指令位于 IF/ID 段寄存器中，下一条指令并未写入流水段寄存器）。

之后对时序发生单元进行设计，模拟流水线的执行流程并添加测试组件，最终完成了如下的仿真模型。



1. 2 数据通路图



2 指令系统

2. 1 指令格式

TEC-5 系统总共涉及到有 8 条指令，这 8 条指令的格式如下图所示。

由于整体数据通路与 TEC-5 不同，我们选取了能够体现流水线设计的前六条指令，并在设计时添加了 NOP 指令用于流水线阻塞。

名称	助记符	功能	指令格式						
			IR7	IR6	IR5	IR4	IR3	IR2	IR1 IR0
加法	ADD Rd, Rs	Rd + Rs → Rd	0	0	0	0	Rs1	Rs0	Rd1 Rd0
减法	SUB Rd, Rs	Rd - Rs → Rd	0	0	0	1	Rs1	Rs0	Rd1 Rd0
逻辑与	AND Rd, Rs	Rd & Rs → Rd	0	0	1	0	Rs1	Rs0	Rd1 Rd0
存数	STA Rd, [Rs]	Rd → [Rs]	0	0	1	1	Rs1	Rs0	Rd1 Rd0
取数	LDA Rd, [Rs]	[Rs] → Rd	0	1	0	0	Rs1	Rs0	Rd1 Rd0
条件转移	JCR3	若 C=1 则 R3 → PC	0	1	0	1	1	1	× ×
停机	STP	暂停执行	0	1	1	0	×	×	× ×
输出	OUT Rs	Rs → DBUS	0	1	1	1	Rs1	Rs0	× ×

2. 2 指令的分类和功能

2. 2. 1 算术逻辑运算指令

包含加法(ADD)、减法(SUB)、逻辑与(AND)三条指令。

2. 2. 2 数据传输类指令

包含存数(STA)、取数(LDA)两条指令。

2. 2. 3 程序控制类指令

包含条件转移(JC)

2.3 指令设计

标识符	操作数	IR0	IR1	IR2	IR3	RS1	RS0	RD1	RD0
ADD	R0,R0	0	0	0	0	0	0	0	0
ADD	R0,R1	0	0	0	0	0	1	0	0
ADD	R0,R2	0	0	0	0	1	0	0	0
ADD	R0,R3	0	0	0	0	1	1	0	0
ADD	R1,R0	0	0	0	0	0	0	0	1
ADD	R1,R1	0	0	0	0	0	1	0	1
ADD	R1,R2	0	0	0	0	1	0	0	1
ADD	R1,R3	0	0	0	0	1	1	0	1
ADD	R2,R0	0	0	0	0	0	0	1	0
ADD	R2,R1	0	0	0	0	0	1	1	0
ADD	R2,R2	0	0	0	0	1	0	1	0
ADD	R2,R3	0	0	0	0	1	1	1	0
ADD	R3,R0	0	0	0	0	0	0	1	1
SUB	R0,R0	0	0	0	1	0	0	0	0
SUB
AND	R0,R0	0	0	1	0	0	0	0	0

AND
STA	R0,[R0]]	0	0	1	1	0	0	0	0
STA
LDA	[R0],R 0	0	1	0	0	0	0	0	0
LDA
JC	R0	0	1	0	1	0	0	X	X

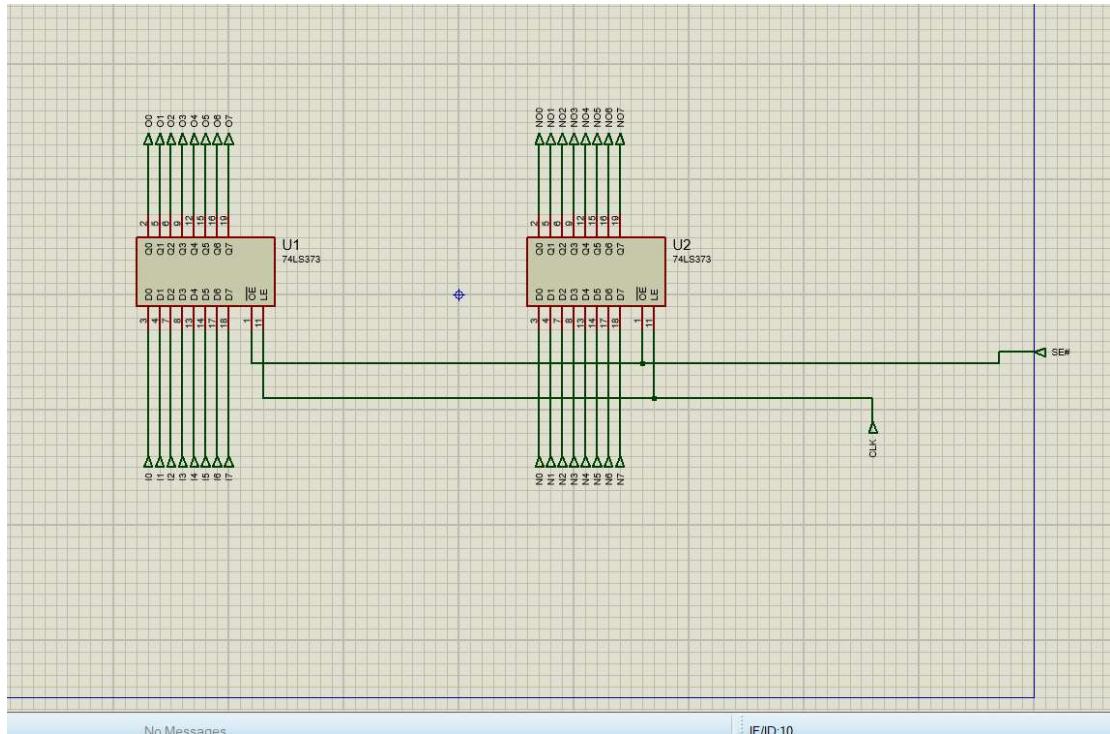
2.4 微指令设计

微指令格式

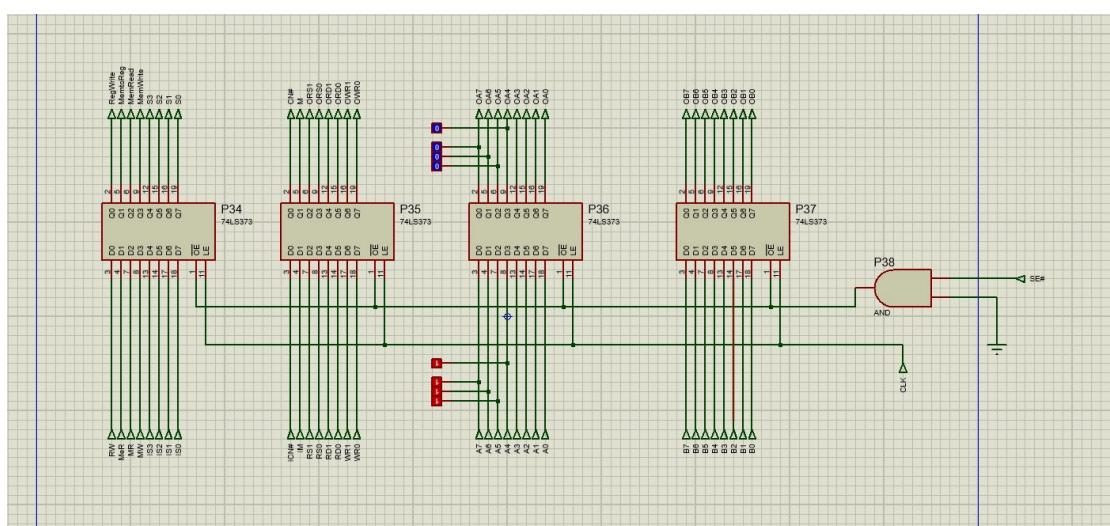


3 流水线计算机设计

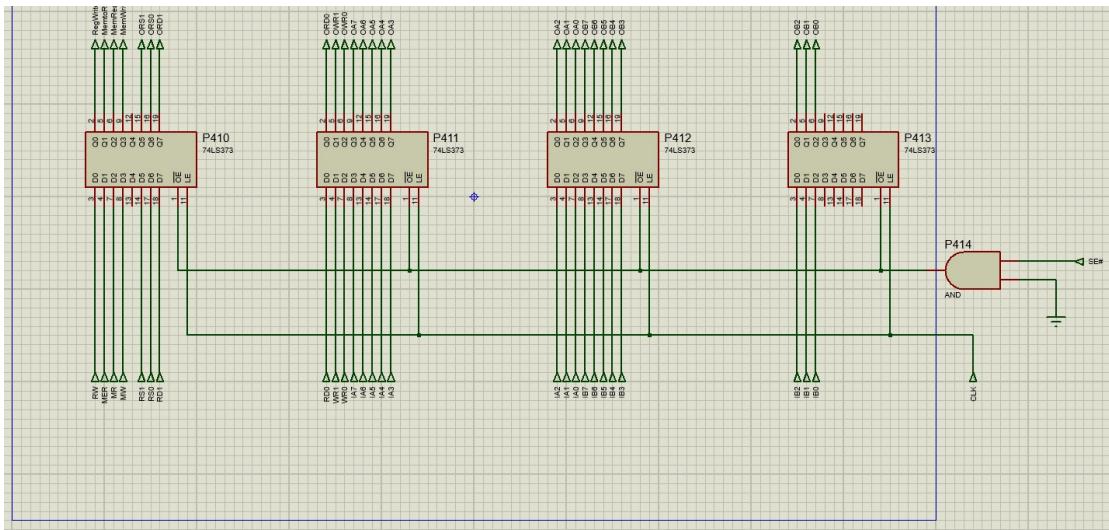
3.1 流水段寄存器设计



对于第一段流水线的流水段寄存器：要保护起来的是从指令存储器取出的指令。



第二段流水线流水段寄存器：保护的是从微指令控制器译码出来的 10 位微指令和从双端口寄存器取出来的左右操作数

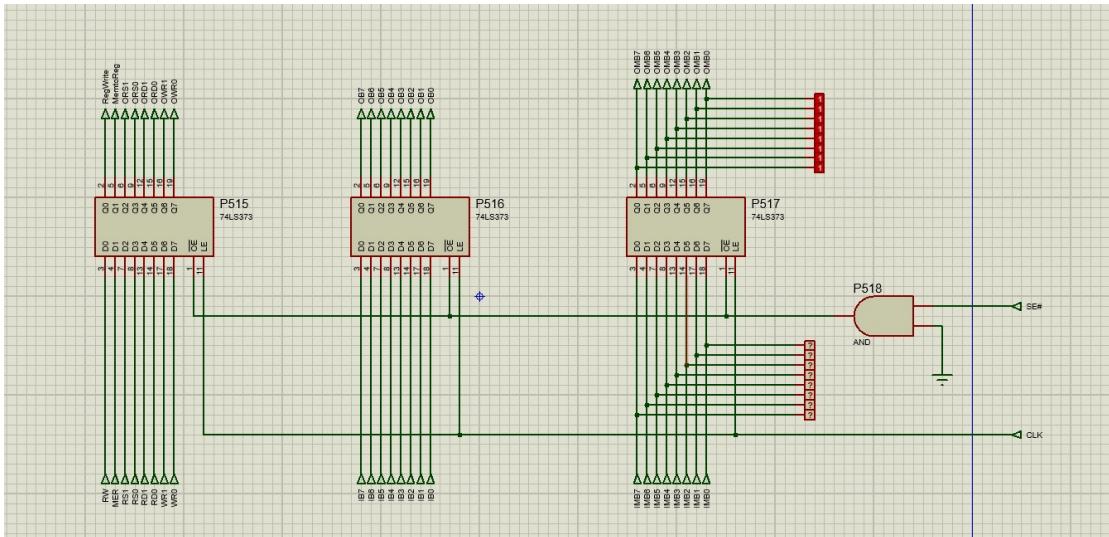


第三段流水线流水段寄存器：保护的是微指令和从 ALU 直接计算出的结果，同时保护旁路输出的上条指令做出的正确结果，避免读取脏数据。

例如：ADD R1,R2

ADD R1,R3

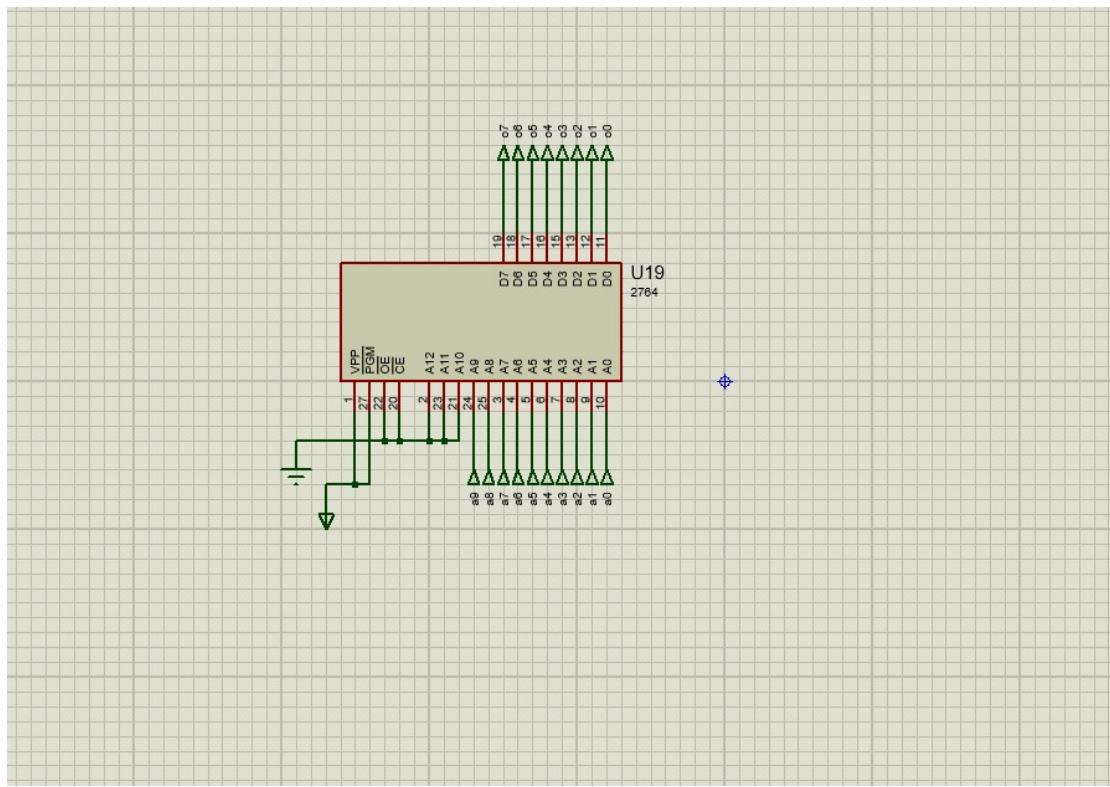
在流水线中，这样的话第二条指令读取到的 R1 的数据是脏数据，而不是第一条 ADD 所做出的 R1+R2 的正确数据，而此时，正确数据通过上一条流水段之后的旁路传输回来。



第四段流水线流水段寄存器：保护的是微指令和从主存中取的结果以及在存取数指令中直传的数据

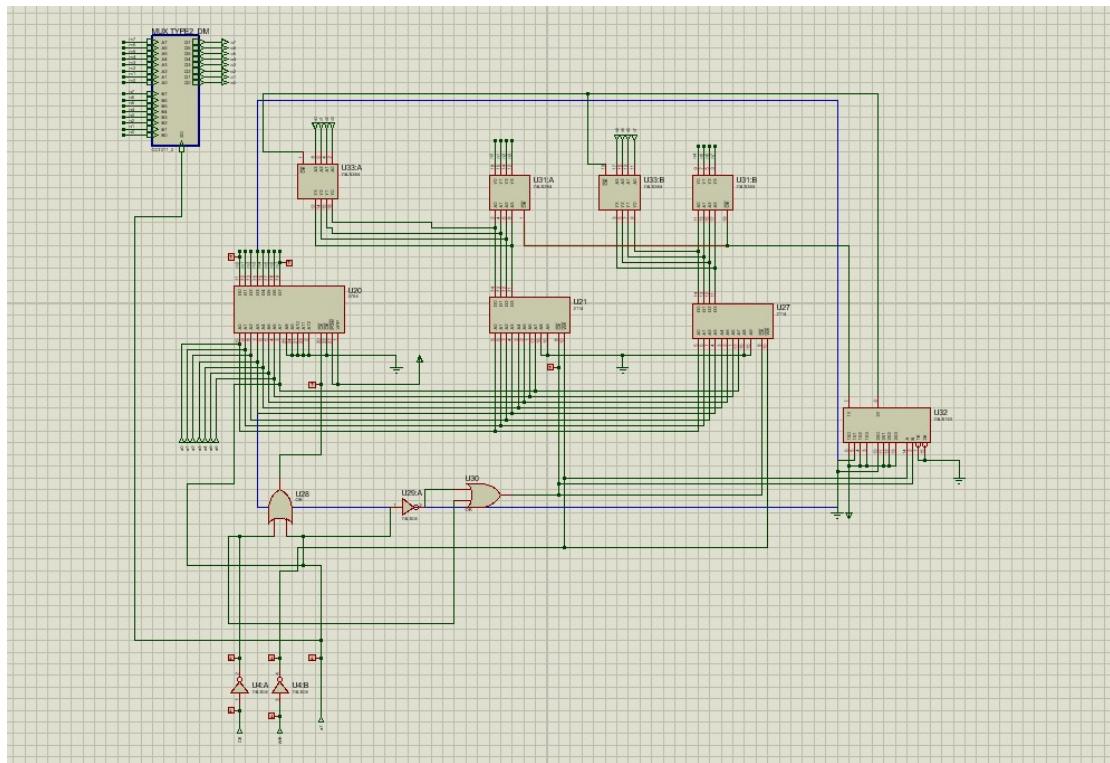
3.2 存储器设计

3.2.1 指令存储器设计



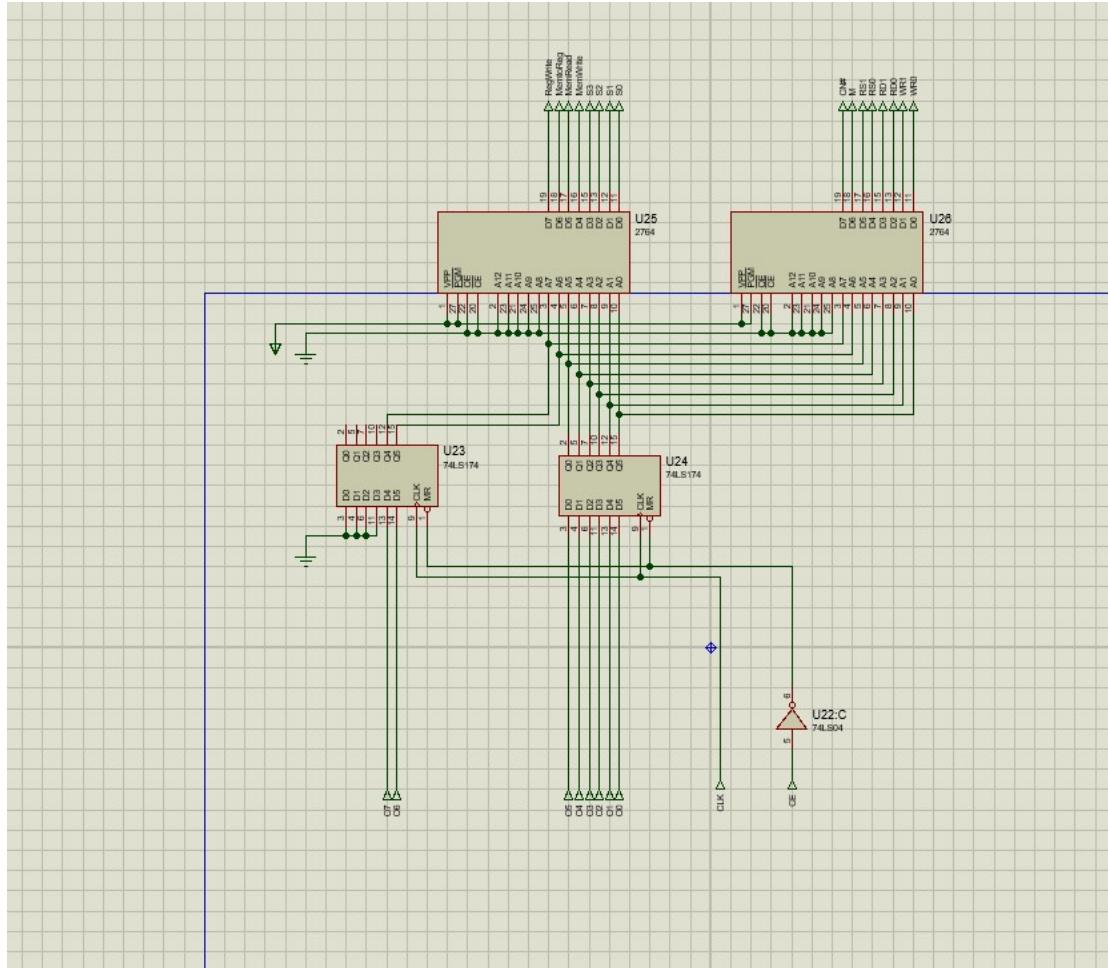
就是一个 2764 的只读存储器，要完成的程序提前烧写进去，按下开机按钮，直接自动一条条取出运行。

3.2.2 数据存储器



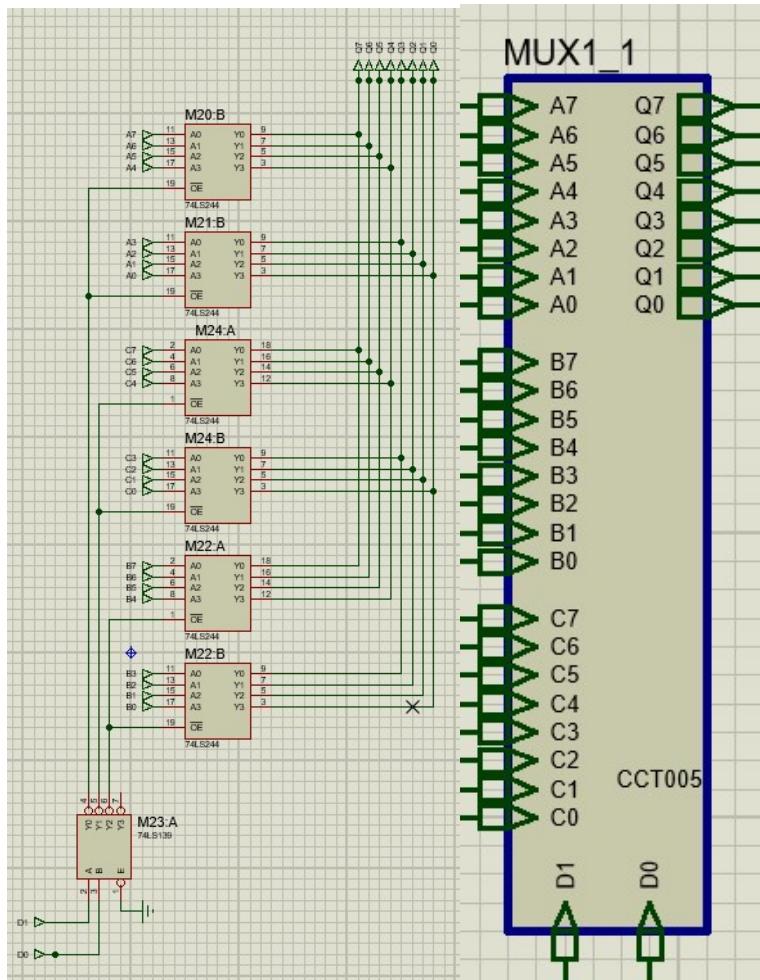
首先，我们的数据存储器中的数据是 8 位的，并且共有 256 个单元，我们将这 256 个单元进行了一个对半划分，即前 128 个单元用作只读单元，即提前在 2764 中烧写进数据，以便于直接读取，而不需要人工输入给寄存器，实现完全的自动挡驾驶，后 128 个单元采用两片 2114 制成一个可读写的存储器，以便于进行程序中的存取操作。而两片的片选位即采用 a7 的信号来控制。

3.3 微程序控制器



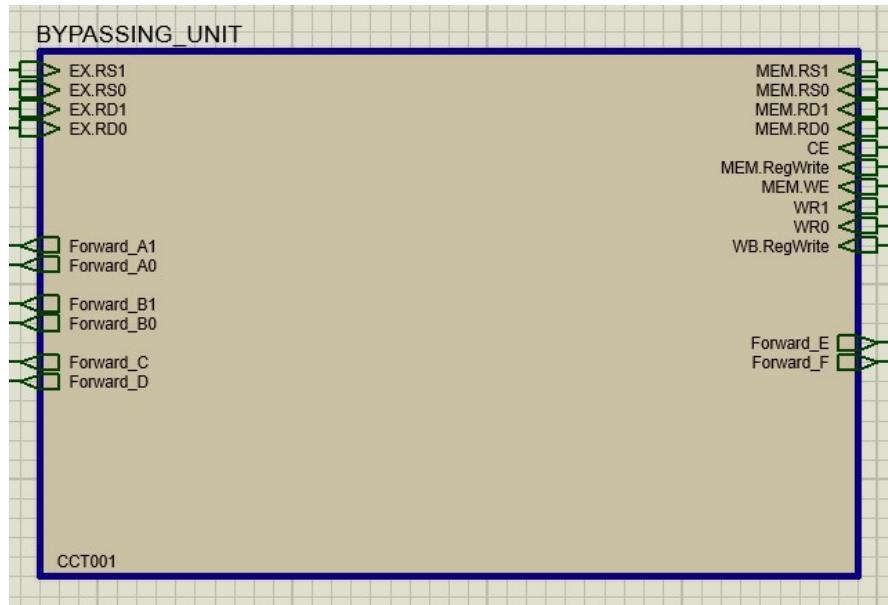
将设计好的微指令写入 2764 存储器，并预留出 RS1,RS0,RD1,RD0 的位置，凑成 16 位微指令，这样更方便用指令进行译码，取出微指令，同时流入流水段寄存器

2.5 数据选择器



对于三选一多选器通过 D0, D1 (对于二选一通过 D) 选择数据的通路，主要运用于数据来源的选择，包括 PC、旁路选择、手动控制台与自动控制台的切换、写回段 ALU 运算结果与内存读出结果的选择等。具体实现通过译码或者选择逻辑使得需要选择的数据通过三态门 74ls244，其它通路的三态门保持高阻态。

2.6 旁路单元（定向传送单元）



对 EX 段、MEM 段、WB 段的 RD、RS、RegWrite、WE、CE 等信号进行逻辑运算，产生旁路信号（ForwardA-F）控制 EX 段和 MEM 段的操作数据来源。

旁路单元的信号非 0 时对应操作为

Forward A=10	执行段 RD 来源于上一条 ALU 的结果
Forward B=10	执行段 RS 来源于上一条 ALU 的结果
Forward C=1	执行段 RD 来源于访问段内存中读出的数据
Forward D=1	执行段 RS 来源于访问段内存中读出的数据
Forward A=01	执行段 RD 来源于写回段写回的数据
Forward B=01	执行段 RS 来源于写回段写回的数据
Forward E=1	访存段 RD 来源于写回段写回的数据
Forward F=1	访存段 RS 来源于写回段写回数据

EX 段旁路选择逻辑为

if A=(EX.RD=MEM.RD and MEM.RegWrite=1 and CE=0) Forward A=10

if B=(EX.RS=MEM.RD and MEM.RegWrite=1 and CE=0) Forward B=10

if C=(EX.RD=MEM.RD and MEM.RegWrite and CE=1) Forward C=1

```

if D=(EX.RS=MEM.RD and MEM.RegWrite and CE=1) Forward D=1

if (EX.RD=WB.RD and WB.RegWrite and A=0 and C=0) Forward A=01

if (EX.RS=WB.RD and WB.RegWrite and B=0 and D=0) Forward B=01

否则 ForwardA=00, ForwardB=00、ForwardC=0、ForwardD=0 (数据不来源于旁路)

```

MEM 段旁路逻辑为

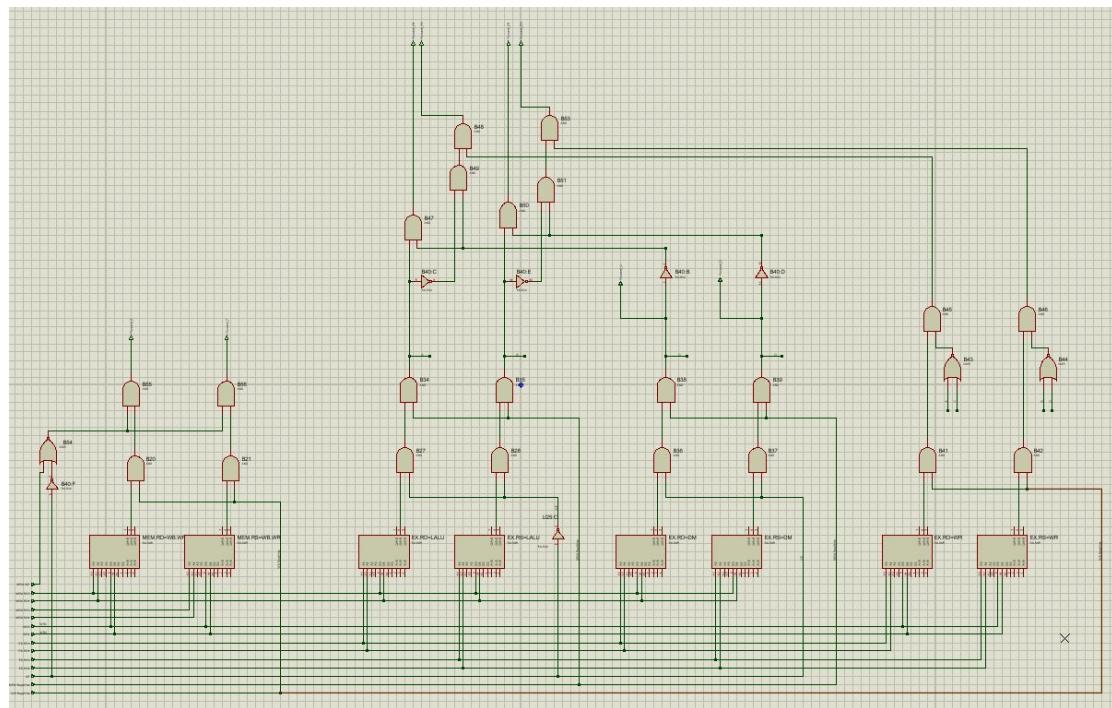
```

if (MEM.RD=WB.RD and WB.RegWrite and (CE=1 or WE=1)) Forward E=1

if (MEM.RS=WB.RD and WB.RegWrite and (CE=1 or WE=1)) Forward F=1

```

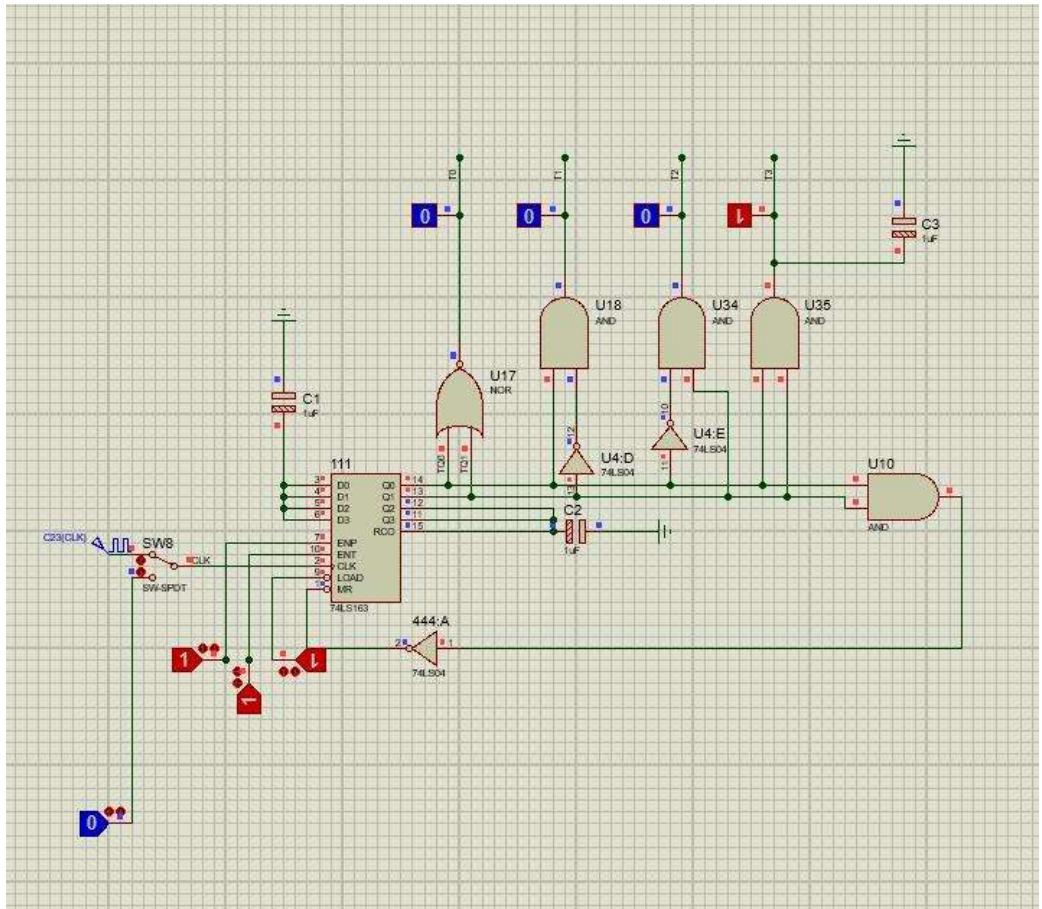
具体实现如下图



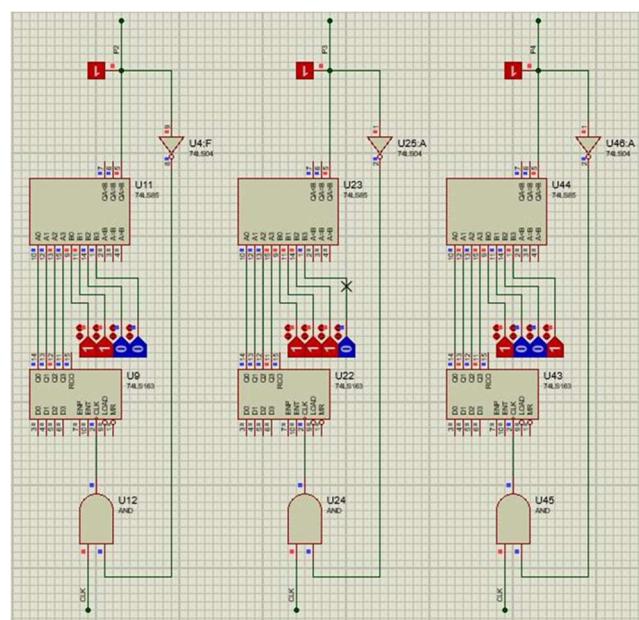
先通过比较器比较当前操作的 RD 或者 RS 是否等于某个阶段的 RD，然后根据上述逻辑通过逻辑器件产生旁路信号。

3.4 时序信号控制 时序发生器

通过 74ls163 设置四进制产生四个同频率的脉冲信号，T0, T1, T2 控制不同期间的时钟，
T3 为控制流水段寄存器的时钟，从 T0 到 T3，一条指令从一个阶段推进到另一个阶段



为了解决在执行第一条指令时除第一个流水段寄存器后面的流水段寄存器不可开始工作，需要按照一定的时间逐步工作以防止脏数据的读入以及 PC 开机读入两次的错误等，需要器件控制工作的时间，因此我们通过 74ls163 计数并将计数结果与比较器相连，当计数大于设置的工作开始时间才产生一个使能信号，并通过这个使能信号锁死 74ls163 的计数，使得流水线正常工作。



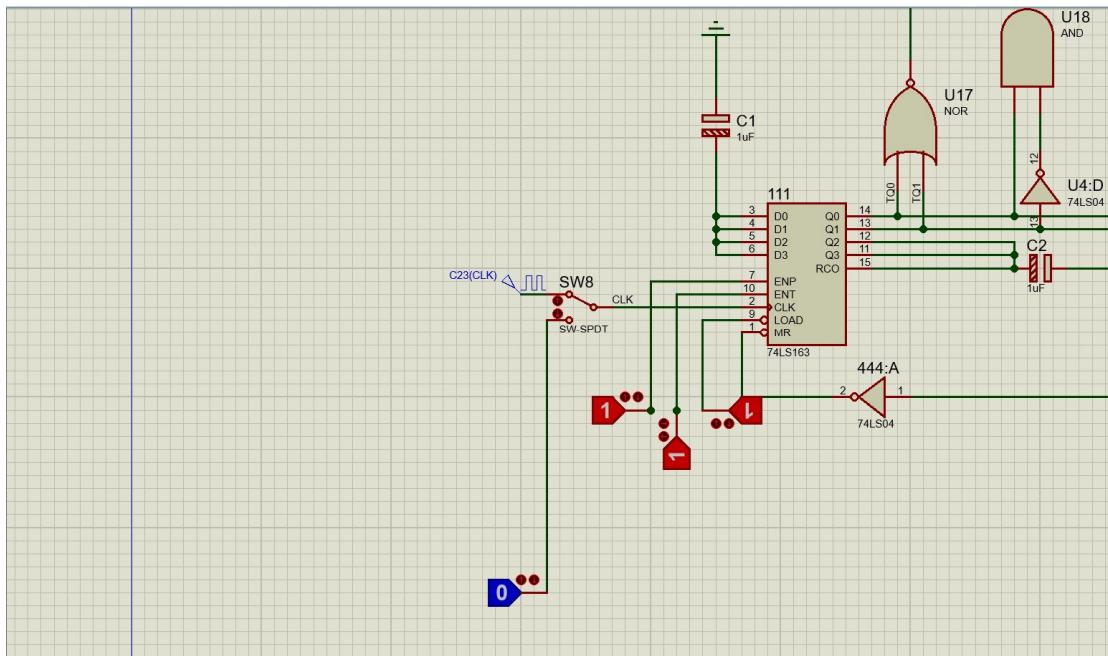
3 测试

3. 1 单指令执行步骤

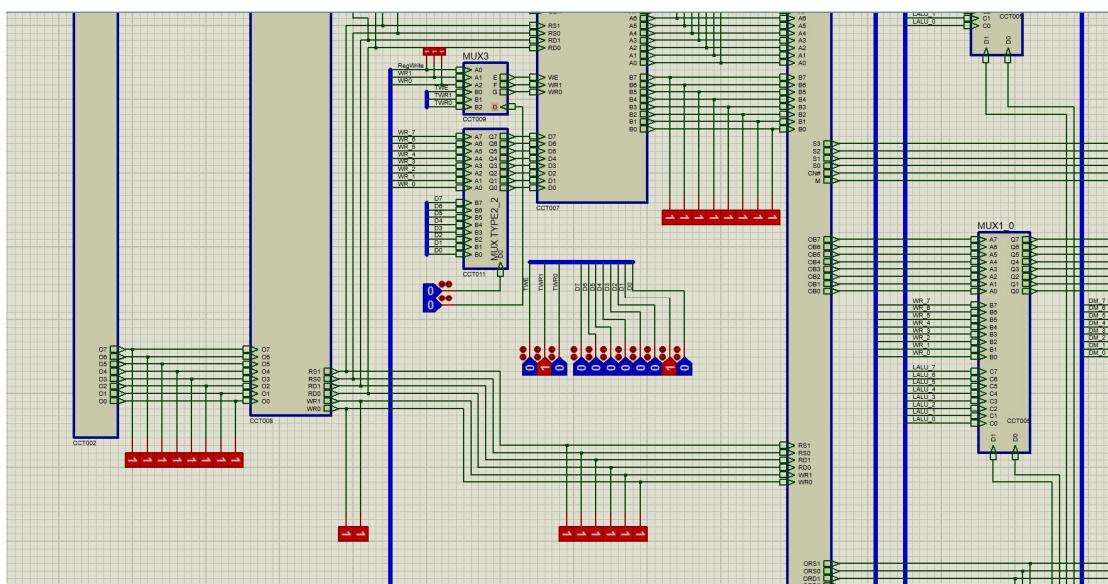
我们以 ADD R3,R2 指令为例测试流水线的运算情况

流水线的各部分操作都是自动执行的，指令也是预先烧入指令寄存器的，在开始这些操作之前，请将各控制信号按下面的说明提前设置好。

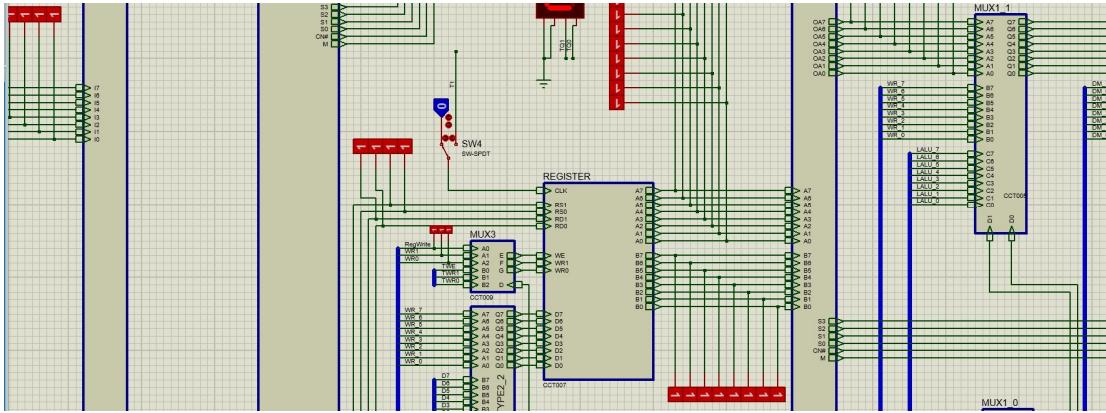
SWB 是主时钟，测试指令前将其掷为手动控制页面用于手动控制



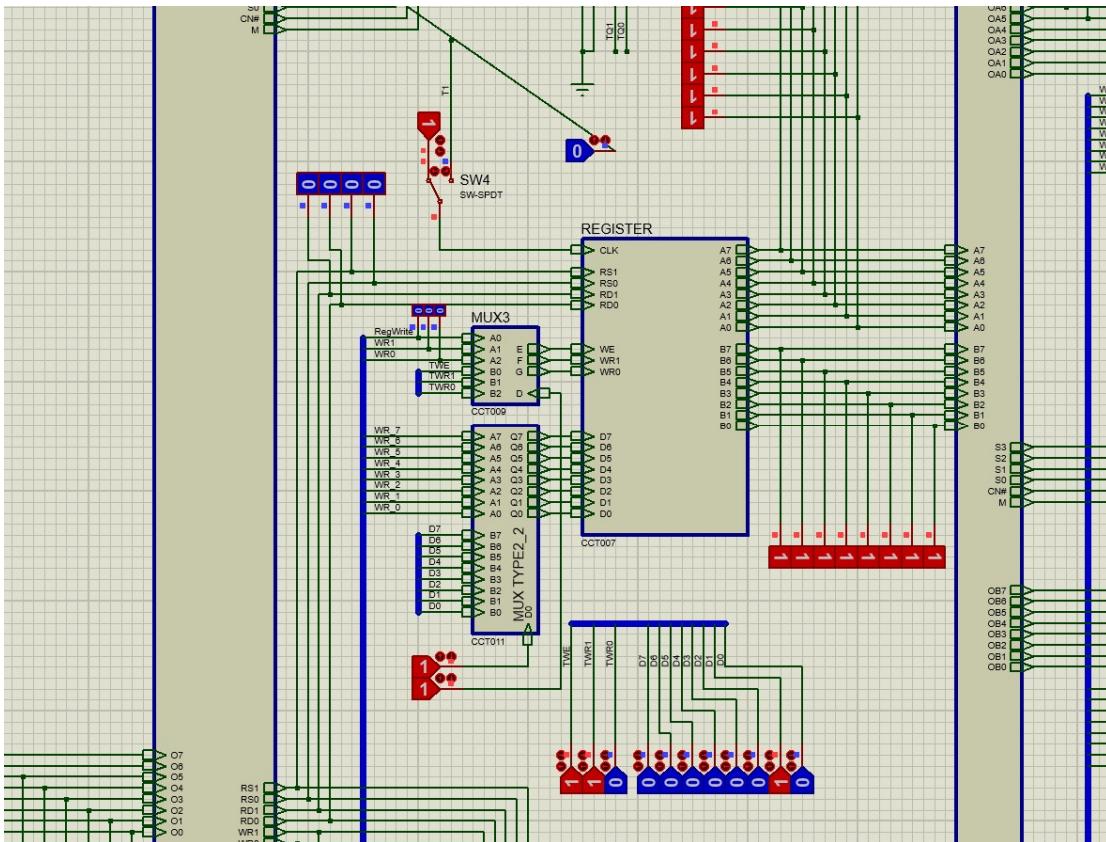
写寄存器操作如下：



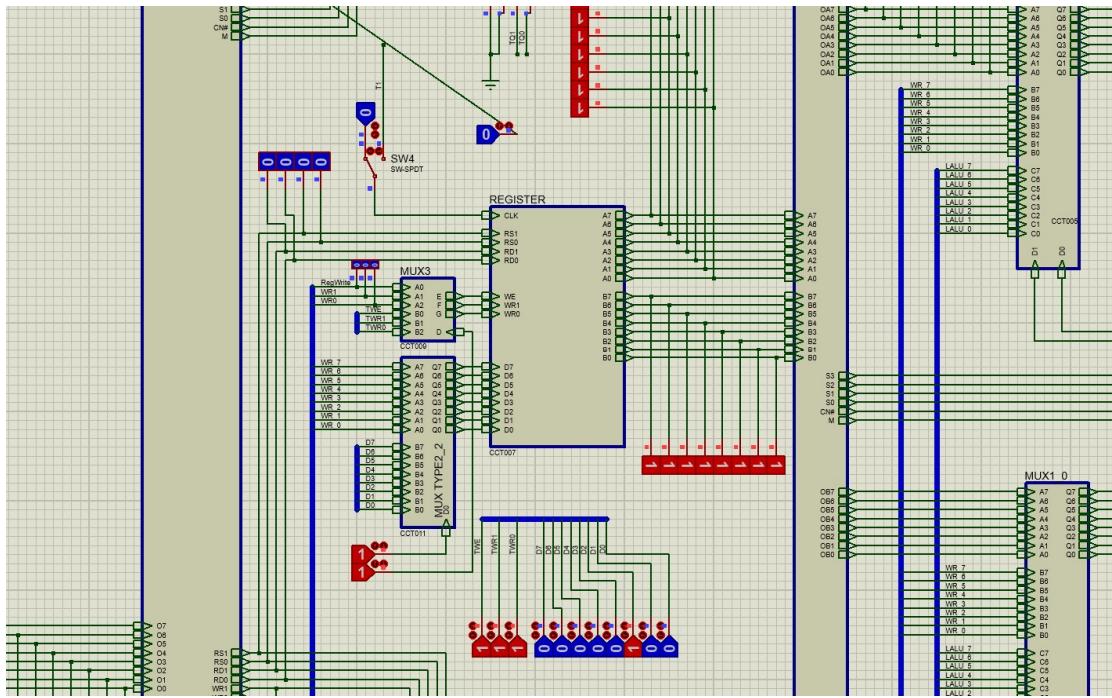
D0-D7 为写入寄存器的数据，TWR1,TWR0 为寄存器的选通信号，TWE 为写入信号，高电平时右边数据写入相应寄存器。



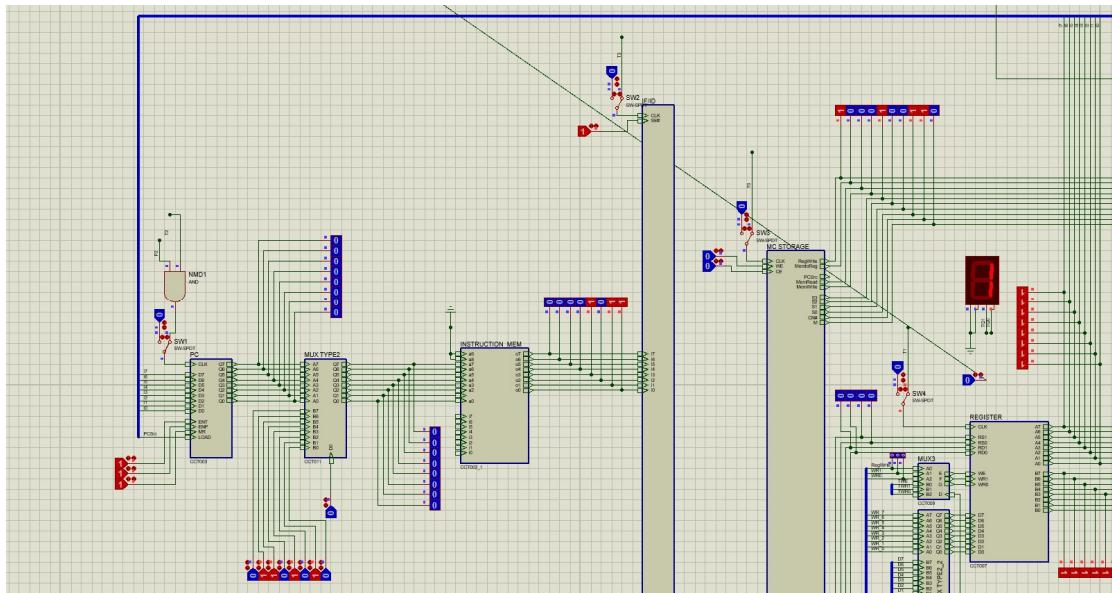
SW4 为控制寄存器的时钟，可以选择手动控制或用总时钟控制。



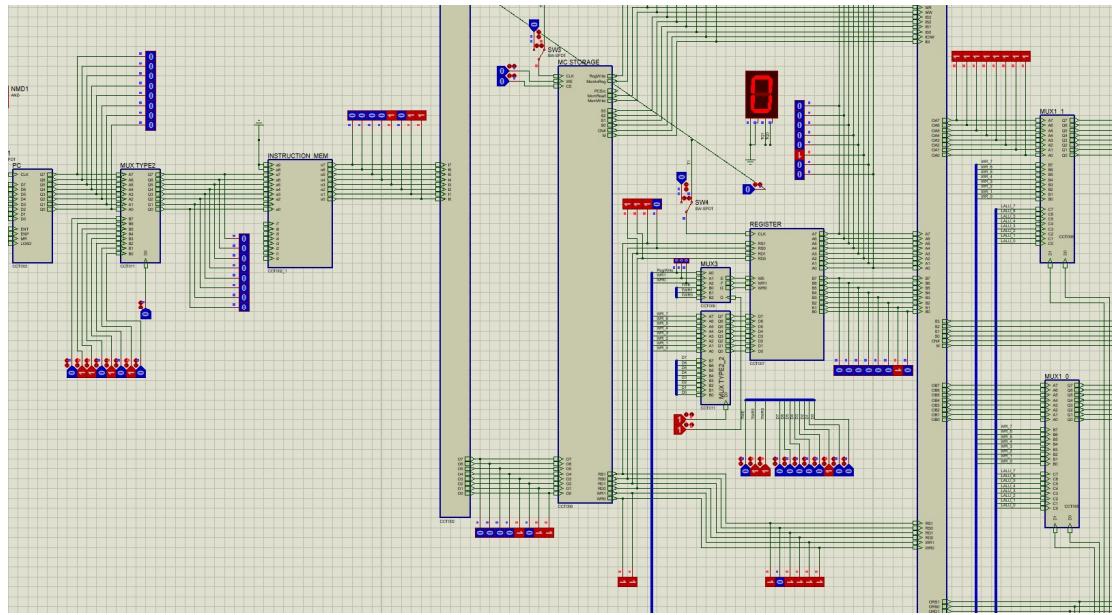
写入寄存器 R2 操作如上，此时 TWE 信号为 1，数据线上数据为 2，点击 SW4 手动触发高电平写入数据。再置为低电平。以相同方式将数据 4 写入 R3



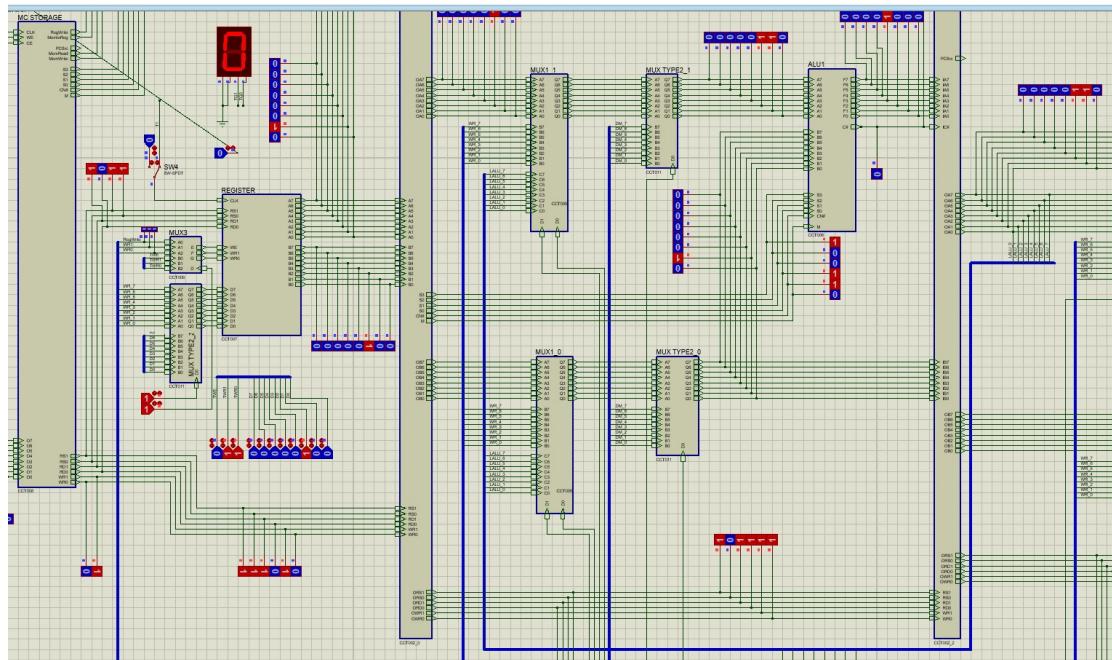
将主时钟的手动控制信号拉至此便于控制，第一个周期时：



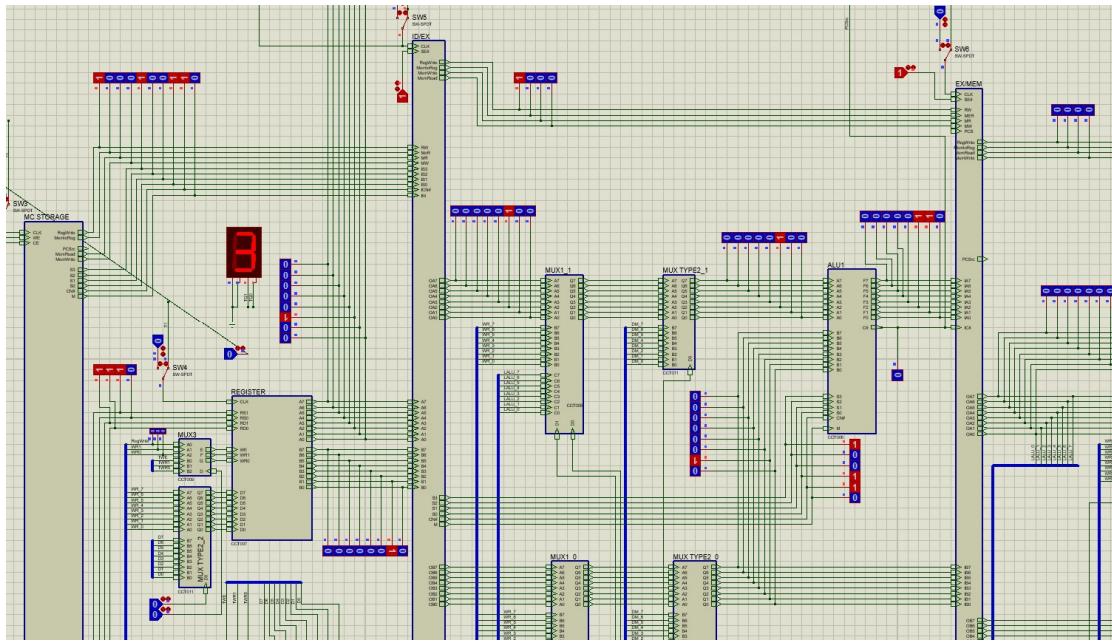
右边 LED 灯从 0 读到 3 时为一个 CPU 时钟周期



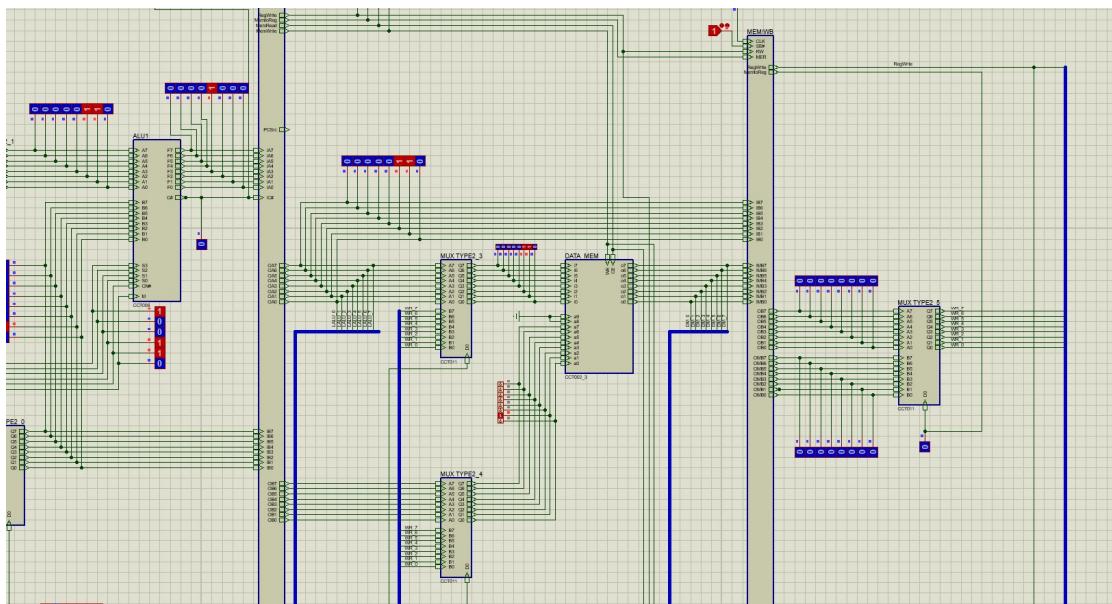
显然，当第一个周期过去时，CPU 已经将 ADD R3,R2 的机器码 00001011（可以参考上文的指令表）从指令存储器中取出。



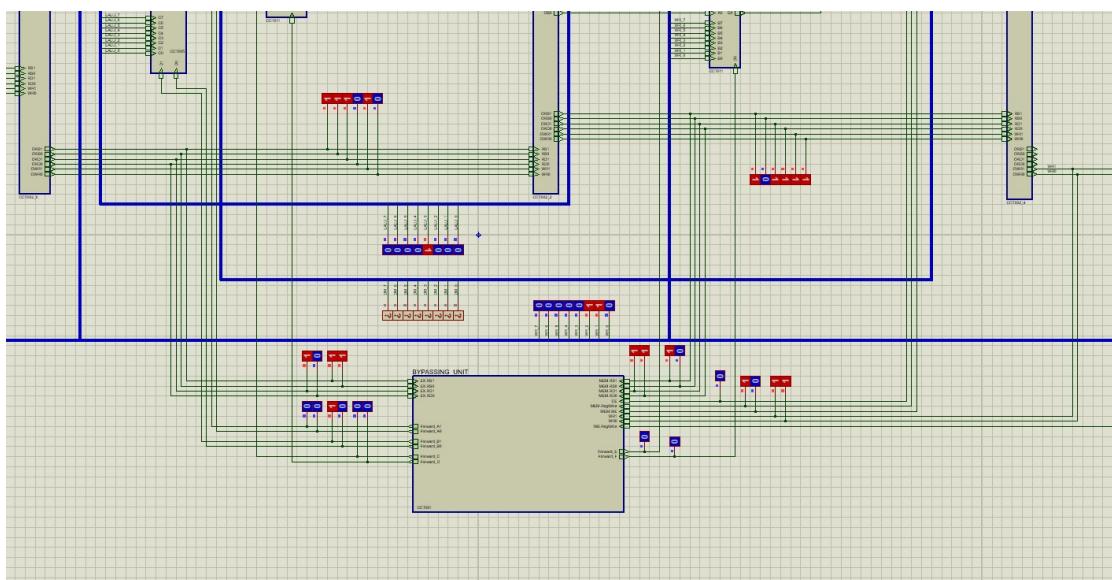
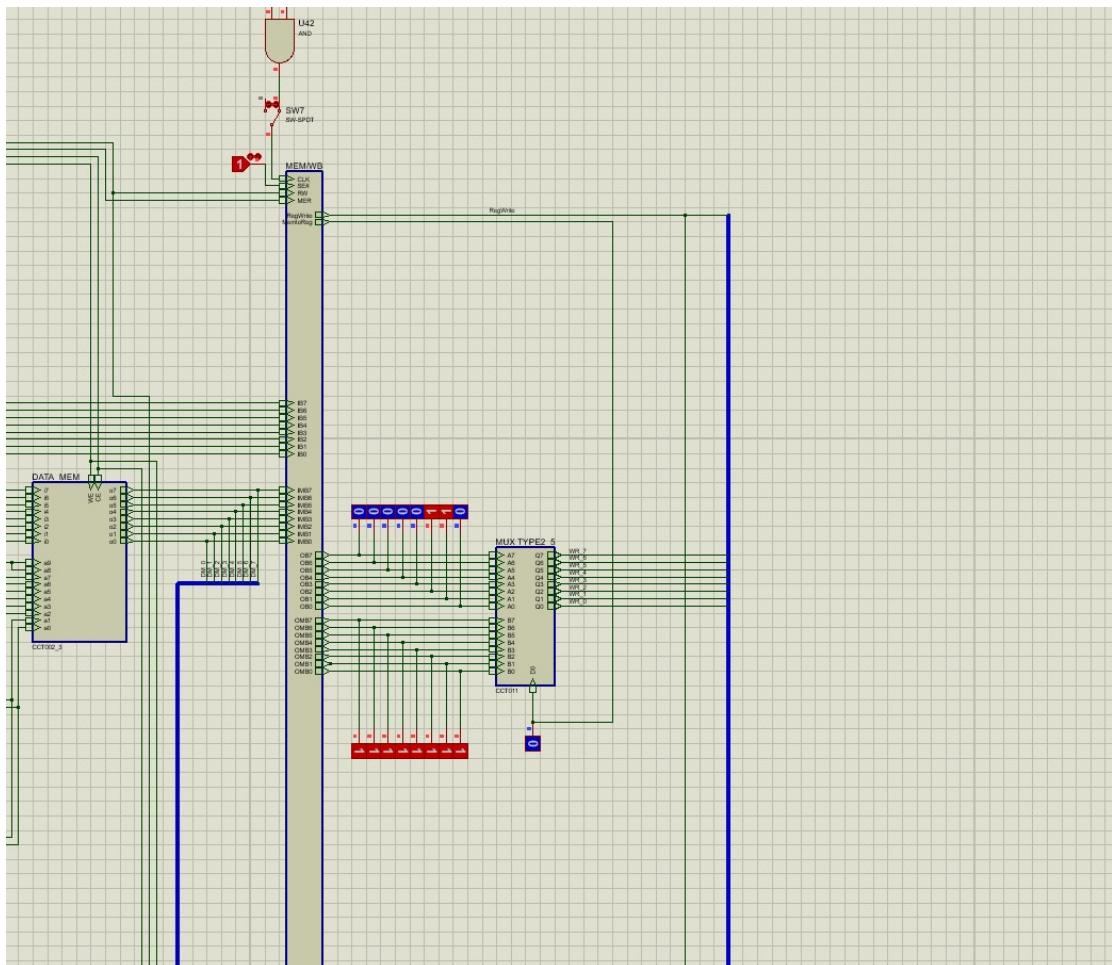
到第二个周期结束时，指令运行到译码段，由图可看出 RS 值为 10， RD 值为 11，即寄存器 R2 与 R3。数据线上 A0-A7 为 2，B0-B7 为 4，即 R2 中数据为 2，R3 中为 4.

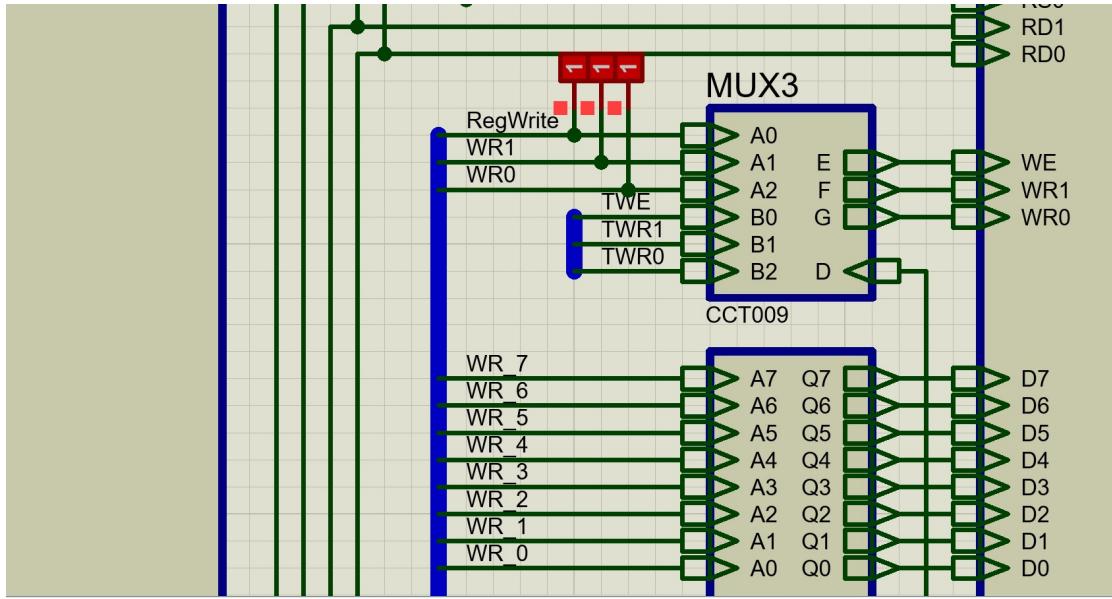


到第三个周期结束时，指令运行到执行段，ALU 上 A0-A7 为 4, B0-B7 为 2，即 R2, R3 数据传入 ALU，ALU 输出 F0-F7 为 6.



第四个周期结束时，指令执行到访存段，由于 ADD 为 R 型指令，无需访存，所以数据从 EX/MEM 段寄存器通过旁路直接写入下个段寄存器 MEM/WB，如图可见。





第五个周期结束时，指令执行到写回段，如图可见写入选择器写回数据为 6，写回寄存器地址为 11，即 R3

3. 2 多指令流水

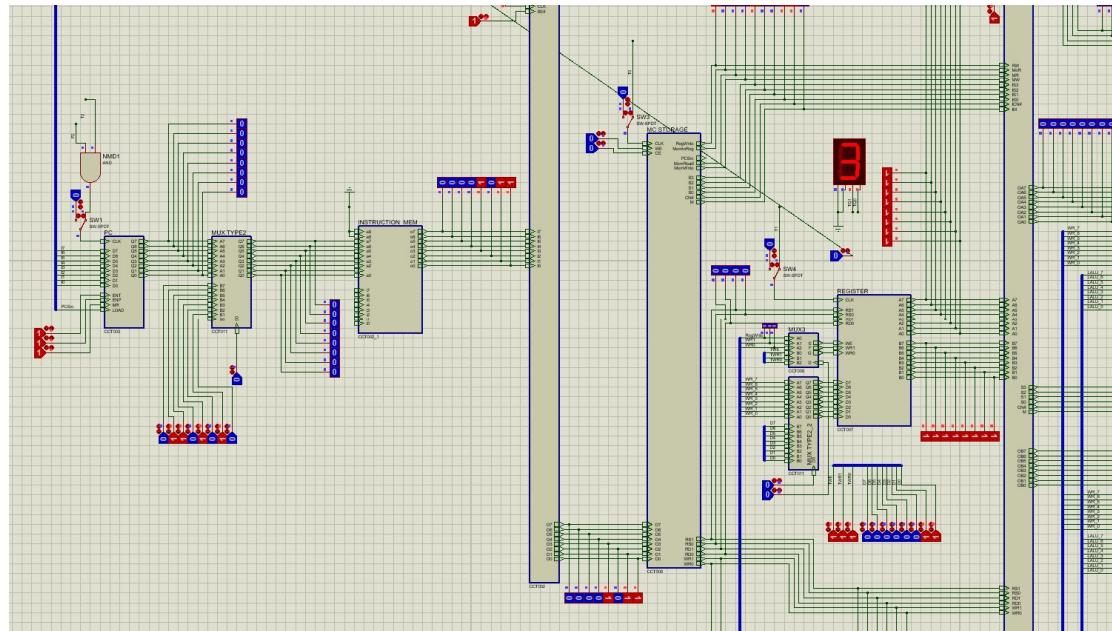
本段我们测试的程序如下：

```

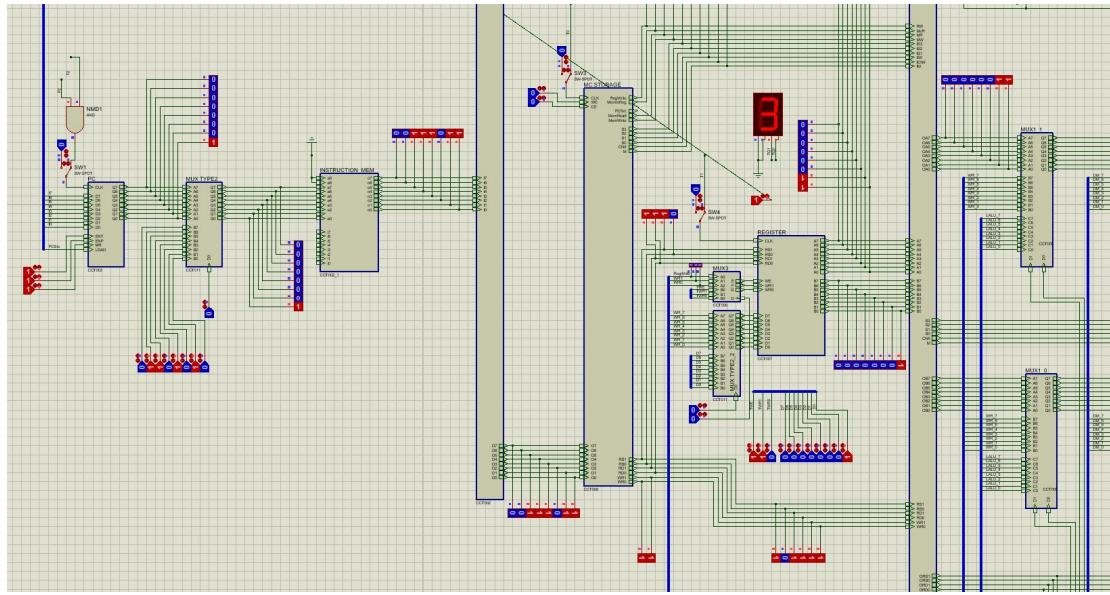
ADD R3, R2
STA R3, [R2]
SUB R3, R2
LDA R3, [R2]

```

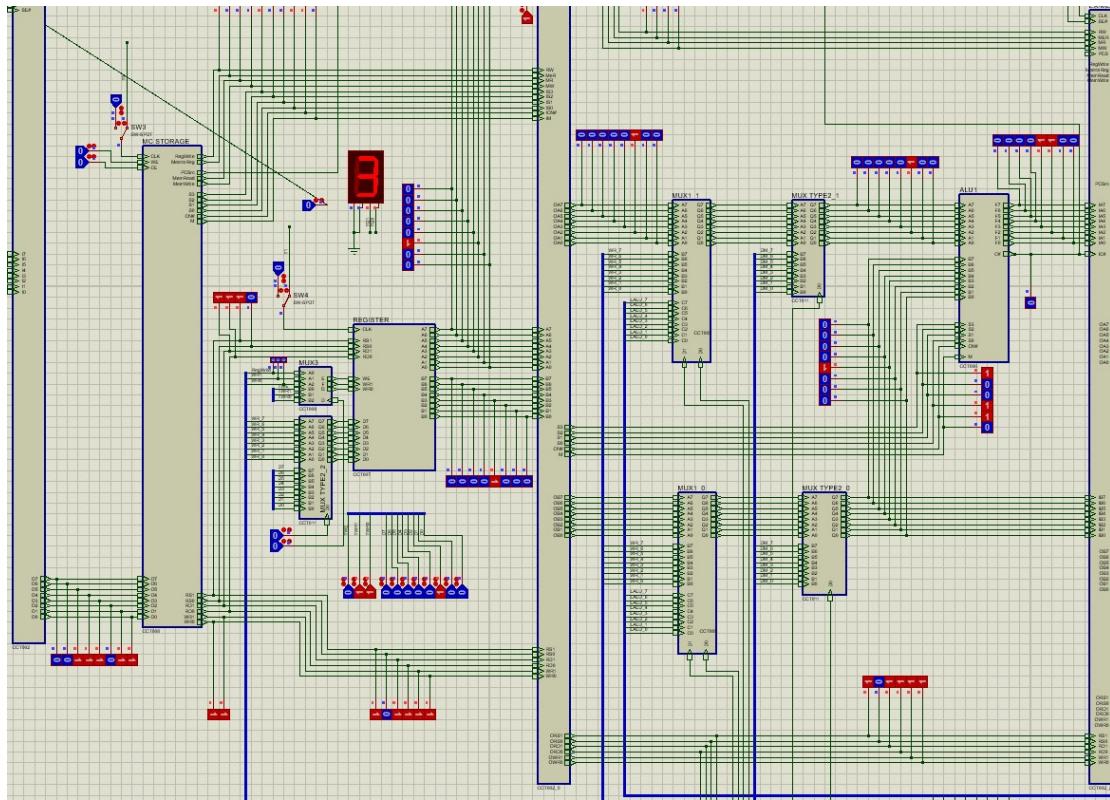
开始设置同上，这次我们将 R3 设置为 4，R2 设置为 8。



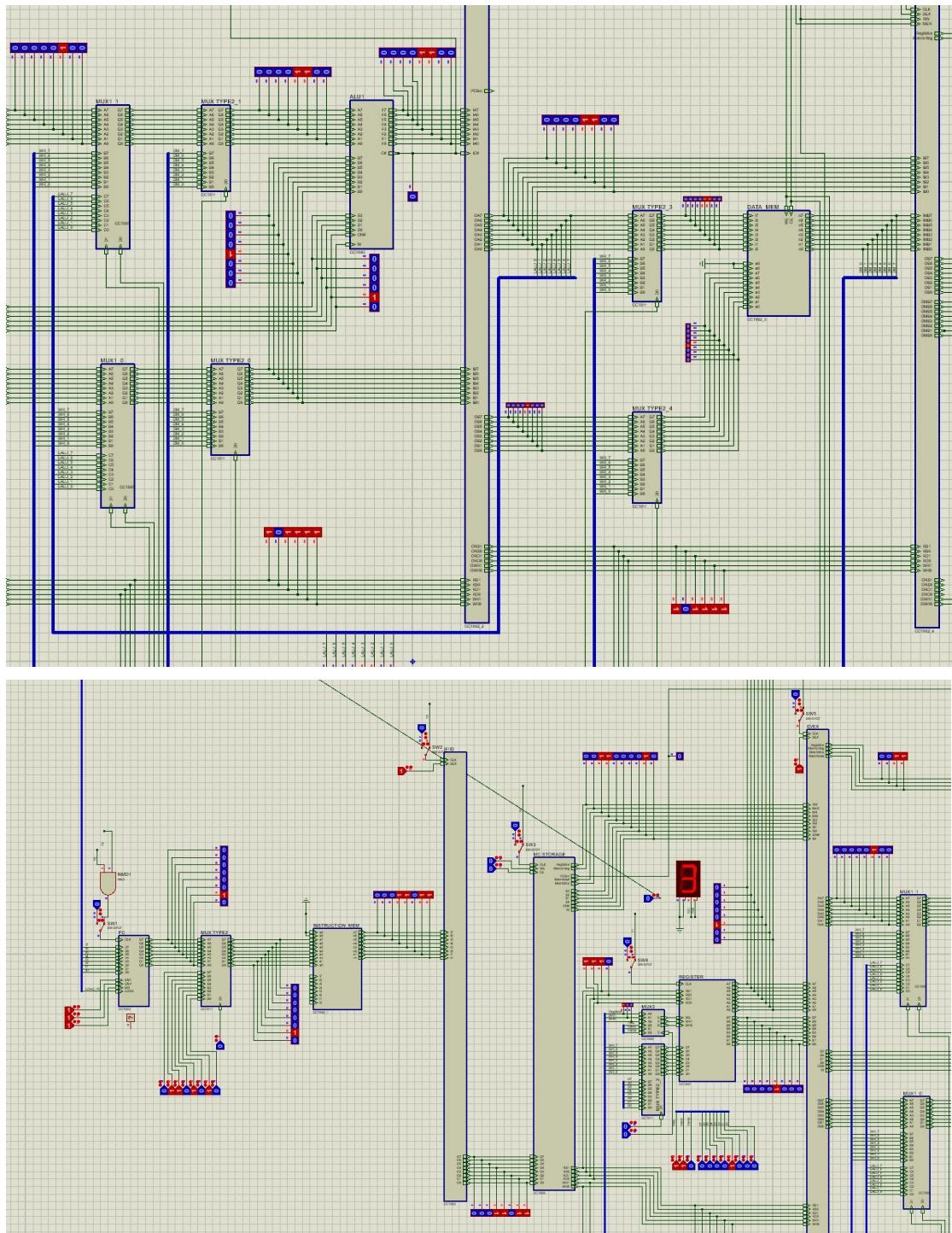
周期一，CPU 将 ADD R3, R2 机器码取出



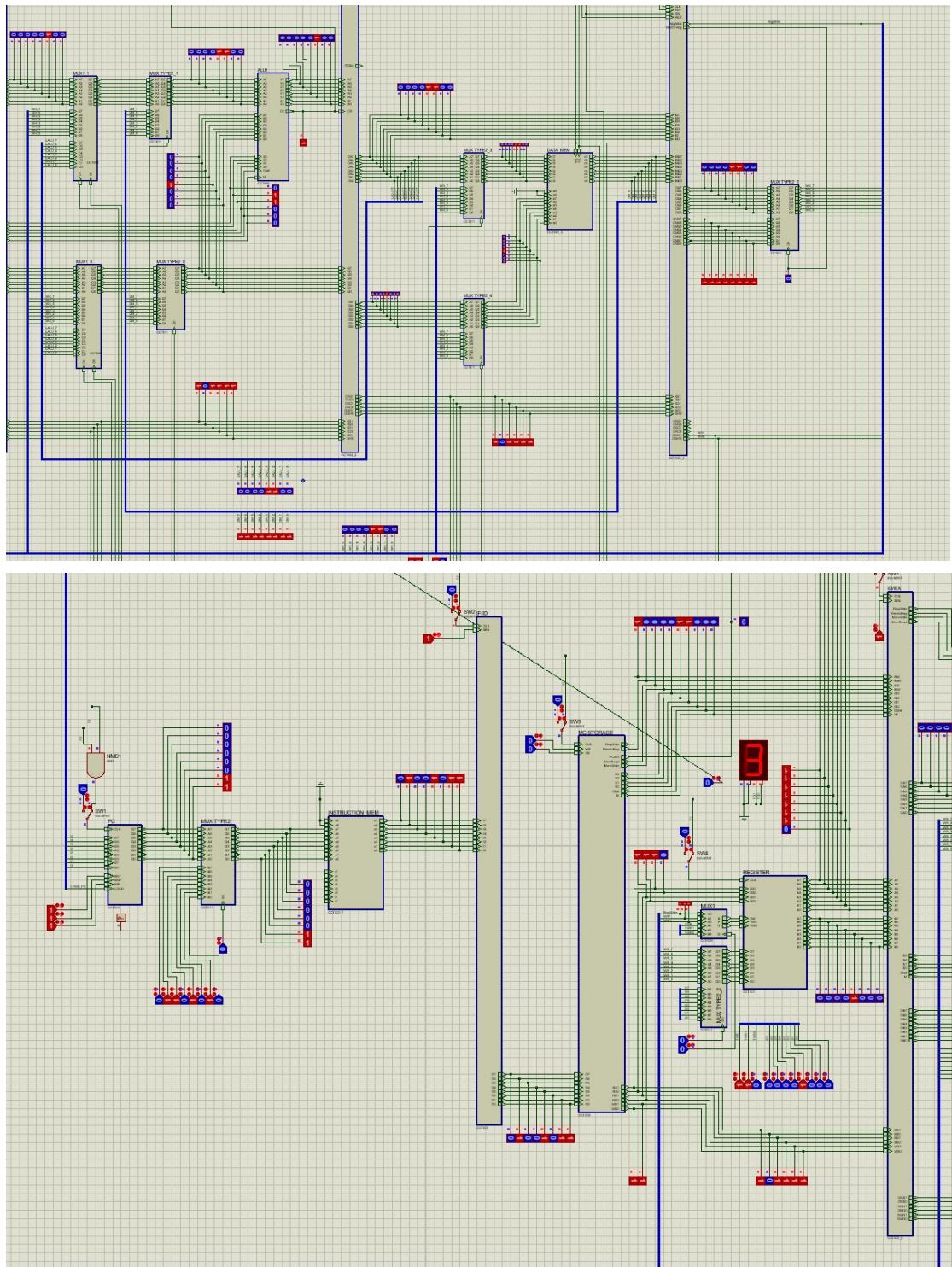
周期二，ADD R3, R2 指令完成译码的同时将 STA R3, [R2] 的机器码 00111011 从指令存储器中取出



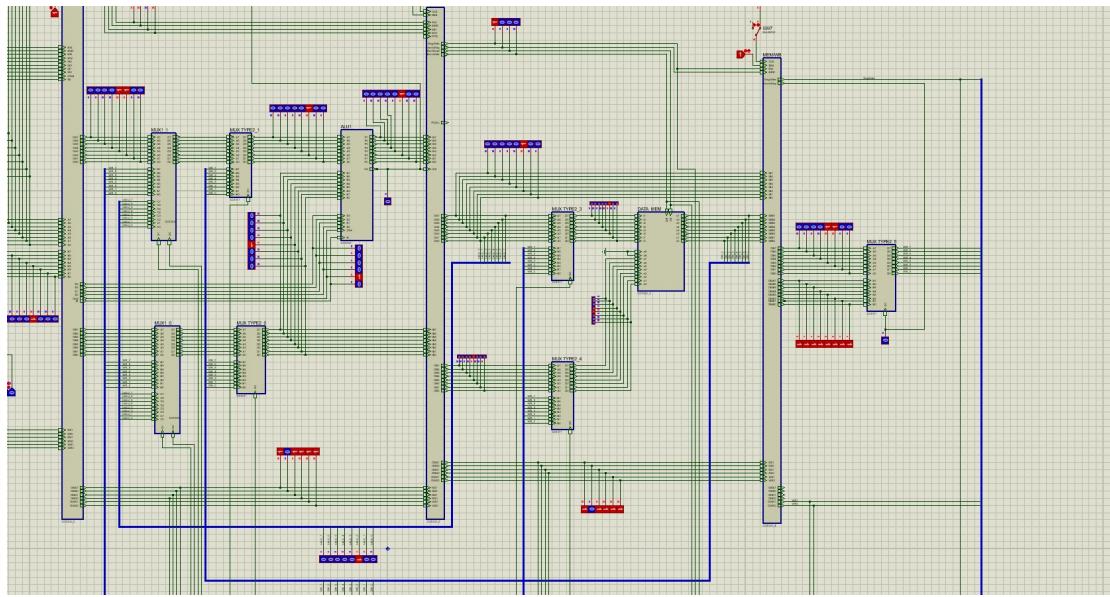
周期三，ADD R3, R2 指令执行完毕将数据送入访存段，STA R3, [R2] 译码同时完成，SUB R3, R2 指令的机器码同时也取出。



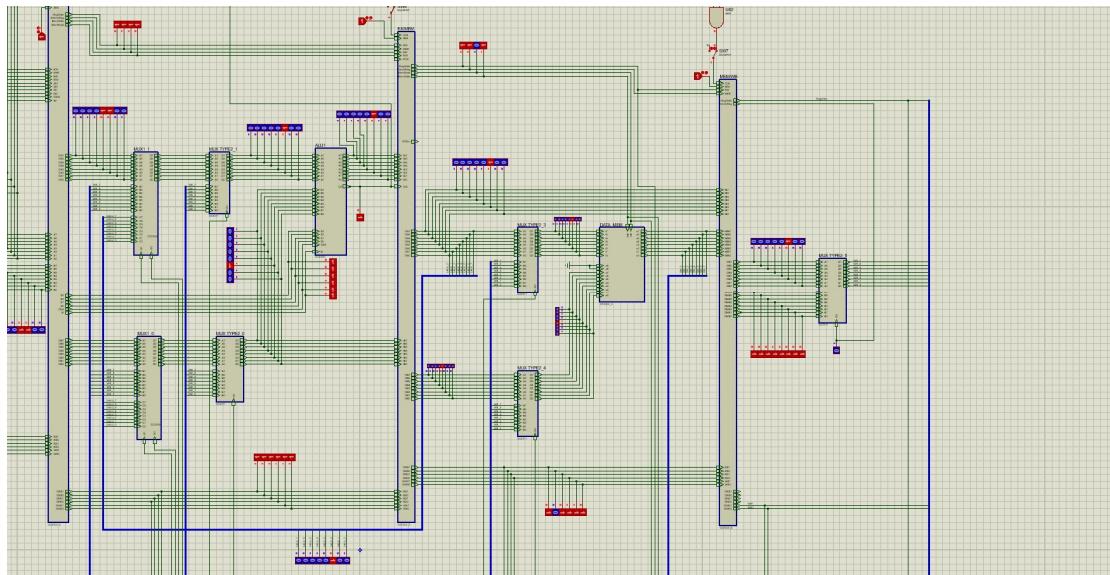
周期四，ADD R3, R2 指令无需访存，通过段寄存器直接送入写回段，STA R3, [R2] 无需使用 alu，通过段寄存器直接送入访存段，SUB R3, R2 指令的机器码开始译码，最后一个指令 LDA R3, [R2] 进行取指操作。



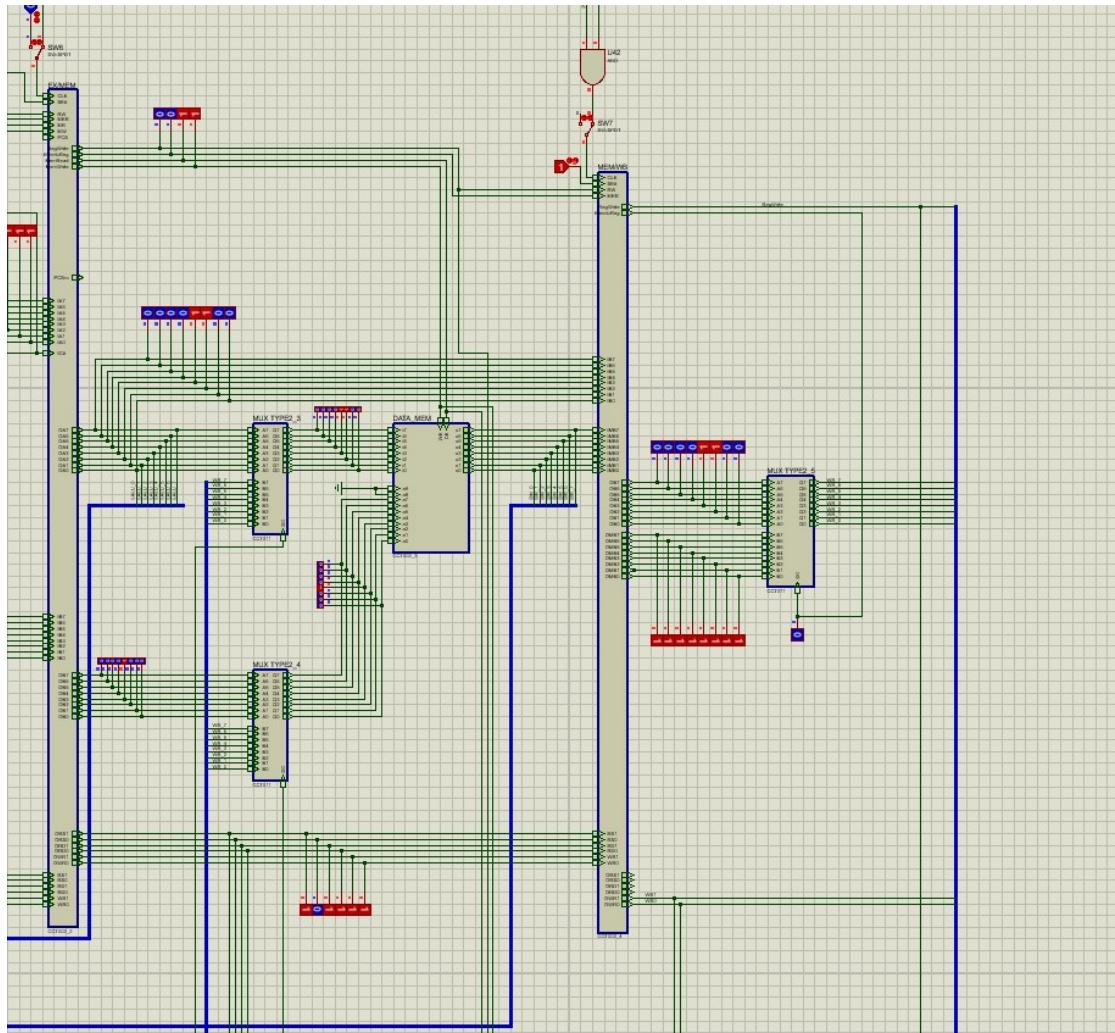
周期五，ADD R3, R2 指令结果写回寄存器（实际上在写回段前已经写回，这点我们在 3.3 中详细分析），STA R3, [R2] 将前一条指令存入 R3 中的结果 12 写入内存 00001000 中，SUB R3, R2 指令开始执行并且通过 alu 计算得到结果 4 (R3=12, R2=8, 相减结果为 4)，后一个指令 LDA R3, [R2] 进行译码操作并将操作数送入下一个段寄存器。



周期六, ADD R3, R2 指令执行完毕, STA R3, [R2]进入写回段, 由于无需写回, 数据线上数据不变。SUB R3, R2 指令进入访存段直接通过, LDA R3, [R2]进入执行段也同样直接通过。



周期七, STA R3, [R2]指令执行完毕。SUB R3, R2 指令进入写回段写回寄存器 R3, LDA R3, [R2]进入访存段读取地址为 00001000 的数据（即之前存入的数据）。

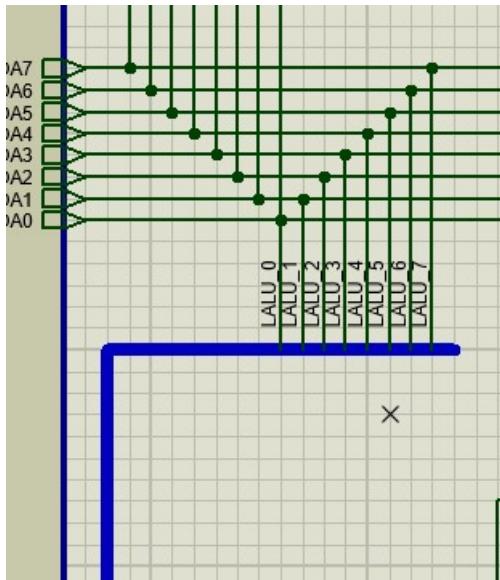


周期八，前面指令都执行完毕，LDA R3, [R2]进入写回段将读取的数据写回寄存器 R3 中。
至此，四条指令全部执行完毕。

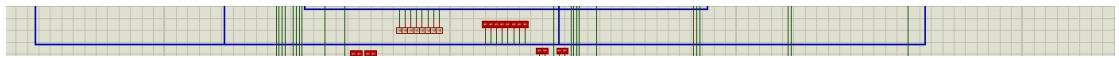
3.3 冒险情况的解决

3.1.1 数据冒险

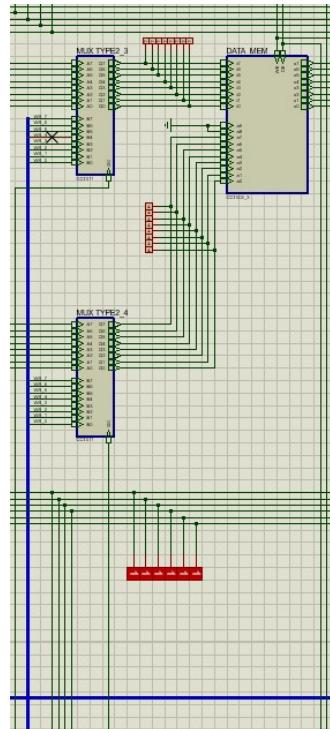
本设计采用了旁路机制（定向传送）来解决数据冒险，在此我们可将数据冒险分为三类，第一类是 EX 段操作数来自于 MEM 段上一条指令 ALU 结果。



。第二类是 EX 段操作数来自于 WB 段两条前的 ALU 或者访存结果。这两类冒险可以直接通过旁路解决，分别在 MEM 段 ALU 输出和 WB 段写回寄存器的数据搭建两条旁路将数据送回 EX 段。

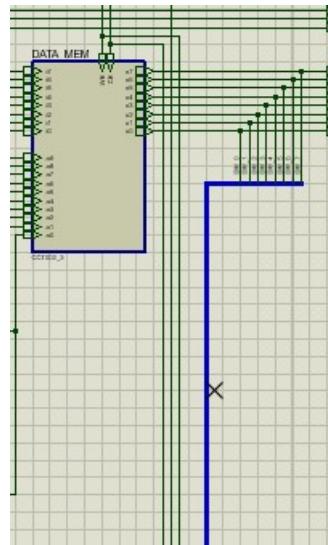


第三类是当 LOAD 指令紧跟 STA 指令，通过搭建 WB 段向写回数据向访存段的旁路可以解决这一类数据冲突。



此外，当 LOAD 指令紧跟 ADD 指令的这种情况下，因为两条指令存在时间回溯性和某些原因，在参照的 MIPS32 流水线模型中无法用旁路来解决，需要使用 NOP 指令堵塞流水线，但在我们的模型中，可以通过内存的输出与 EX 段操作数之间搭建旁路，使得在 EX 段周期结束前获得访存得到的数据且不需要阻塞流水。

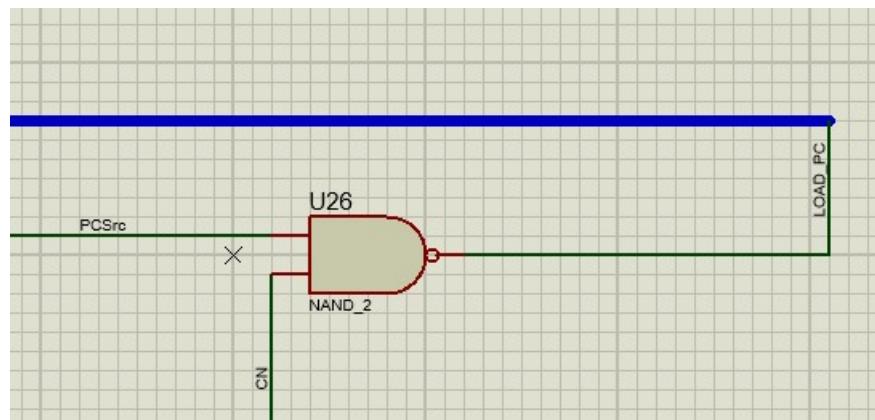
线，大大的提高了流水线 CPI。



但这种做法有一种问题在于，按这种方法写回到 EX 的数据，在 ALU 算出结果后此结果在 MEM 段被 WB 段的数据替换，因此需要在旁路单元的逻辑判断中屏蔽掉当 MEM 不访存但 WB 将数据传入 MEM 段的情况 1。

3.1.2 控制冒险

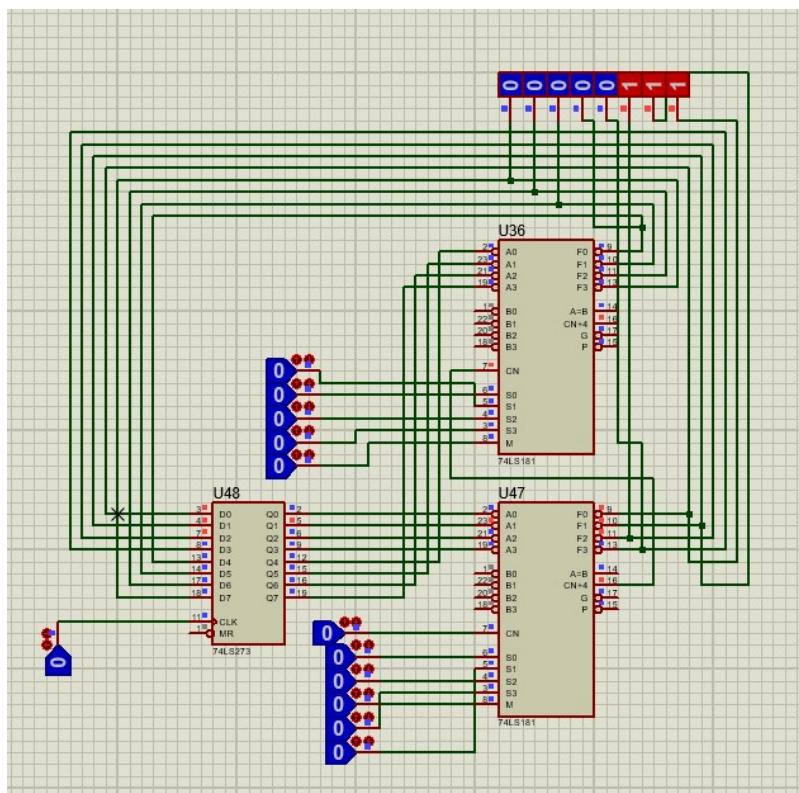
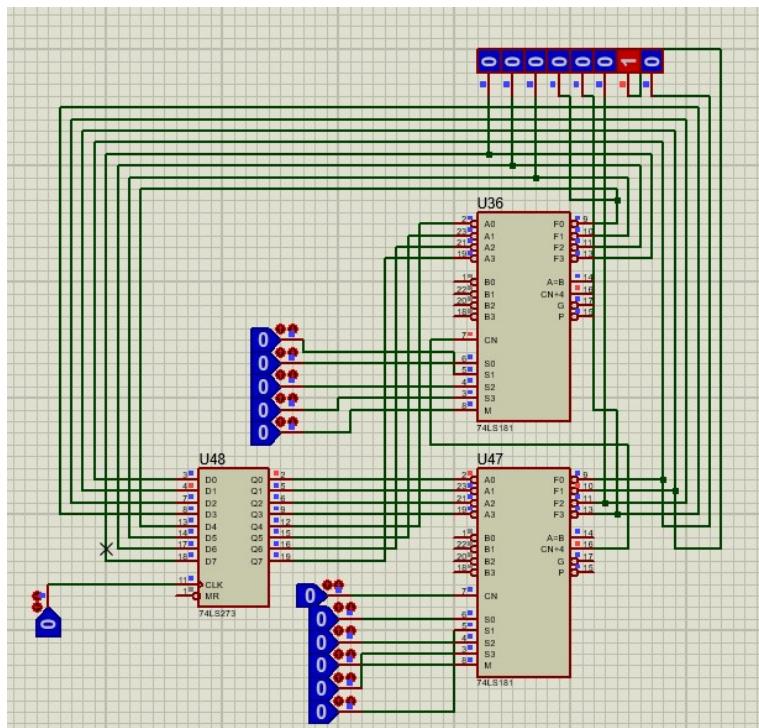
主要解决由 JC 指令引发的控制冒险，由于结构比较简单，只需要将 JC 指令的执行提前到 ID 段，这时可获得上一条指令的 CN 信号，通过 CN 与 PCSrc 相与得到正确的结果。当指令不为 JC 指令时，PCSsrc 设置为高电平，此时无论 CN 的信号是否为高电平跳转都不会被触发。

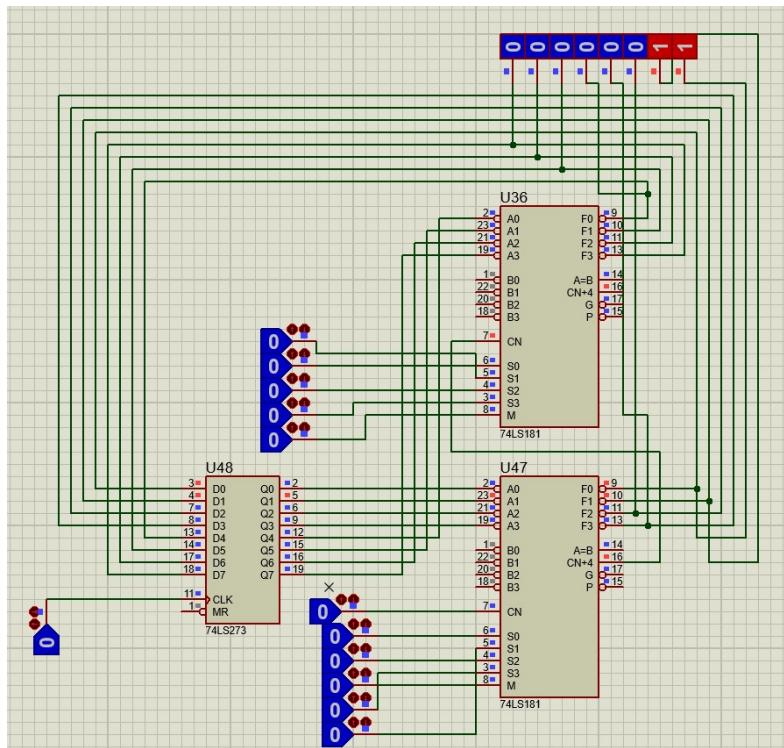
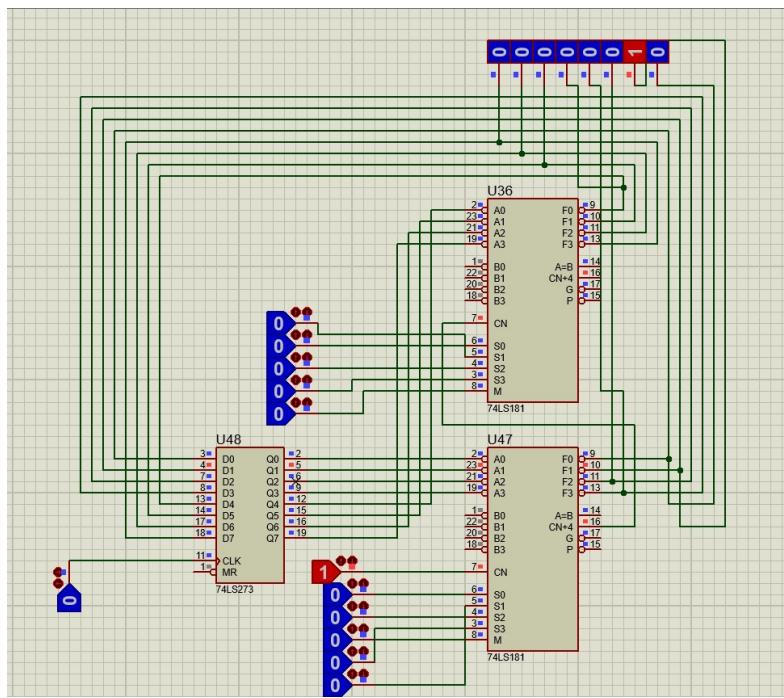


4 遇到的问题与处理

4.1 Proteus 74ls181 存在问题

为了验证这个错误，我们设计了一个测试电路



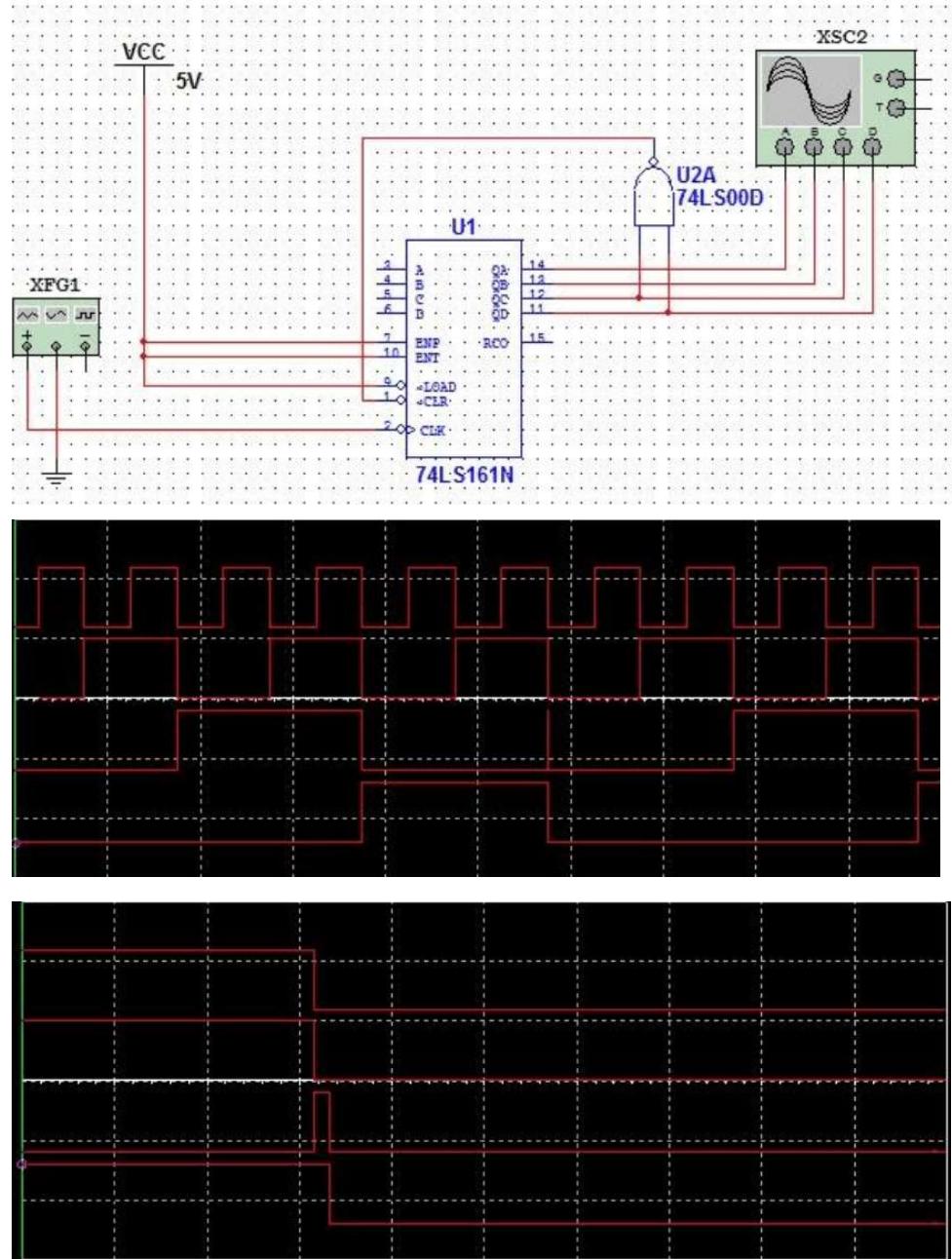


此电路设计为两个级联的 741s181 执行 A 口加 1 的操作并将结果存在 741s244 中，给一个上升沿后又将结果送入 741s181 的 A 口。当执行 $2+1$ 后结果变为 7，此时，手动将 CN 信号翻转一次又可以得到正确的结果。更多的测试表明这当 74LS181 的 A 或者 B 口的数据为奇数时会发生这种错误，在其它组的项目在测试中也发现了这个问题。

4.2 毛刺现象

在测试中我们遇到一个匪夷所思的问题——当没有上升沿来时流水段寄存

器自动读入了数据，导致在一个流水段周期中寄存器工作了两次，并且这两次是连续发生的，直接观察测试的信号并不能发现错误是如何产生。经过老师的细心指点，这个错误的读入可能是由于毛刺现象产生的，计数器的多个时钟进行与运算时产生了一个短暂且我们不需要的上升沿，由于 74ls244 是边沿触发，这个短暂的上升沿并不能在测试中通过 logic probe 观察信号看出，因此我们通过在流水段寄存器设计一个计数器，发现尽管并没有时钟，计数的数值在某个时刻规律的多自增了 1，因为我们猜测是由于毛刺现象导致了流水段寄存器错误的读入。

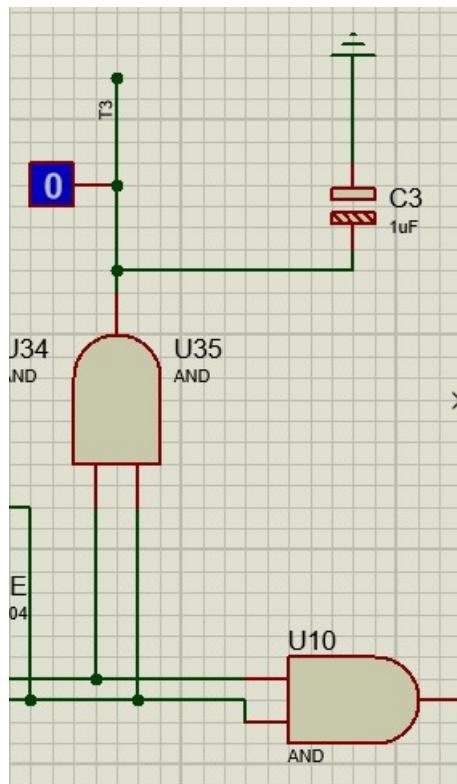


以上图片演示了错误信号的产生，黑色两图从上到下分别为接入示波器 A、B、C、D 口的输入波形，发现 C 口产生了一个短暂的突起，这个突起就是毛刺，放大后发现这个上升沿极其短暂，因此直接通过 logic probe 无法观测出。

信号在器件内部通过连线和逻辑单元时，都有一定的延时。延时的大小与连线的长短和逻辑单元的数目有关，同时还受器件的制造工艺、工作电压、温度等条件的影响。信号的高低电平转换也需要一定的过渡时间。由于存在这两方面因素，多路信号的电平值发生变化时，在信号变化的瞬间，组合逻辑的输出有先后顺序，并不是同时变化，往往会出现一些不正确的尖峰信号，这些尖峰信号称为“毛刺”。如果一个组合逻辑电路中有“毛刺”出现，就说明该电路存在“冒险”。

数字电路中常将毛刺定义为采样间越过逻辑门限一次以上的任何跳变，主要是指电路输出波形中含有时间很短有规律或没有规律的脉冲而又对设计没有用处或产生其他影响，一般都要考虑去除毛刺。

通过示波器对多个输出波形分析我们找到了毛刺的发生原因，是 T3 前与门的延迟导致了毛刺的发生。



毛刺的消除方法常有

- 1、修改设计法: a、代数法，在产生冒险现象的逻辑表达式上，加上冗余项或乘上冗余因子；b、卡诺图法：将卡诺图中相切的圈用一个多余的圈连接起来。
- 2、选通法：在电路中加入选通信号，在输出信号稳定后，选通允许输出，从而产生正确输出。
- 3、滤出法：由于冒险脉冲是一个非常窄的脉冲，可以在输出端接一个几百微法的电容，将其滤出掉。

为了快速解决问题并不进行过多的修改，我们选择了滤除法，在 T3 前与门的输入口加入了一个小电容滤掉了毛刺。通过测试，修改后流水段寄存器工作正

常，毛刺问题被成功解决。

5 心得体会

在完成这次计算机组成原理课程设计之前，我们已经有三次参与共同开发项目，第一次是编写了一个数独小游戏，那时只是第一次合作没有暴露出问题，第二次是数据库的课设，大家各自写各种的最后花了很多时间赶工做出来一个还算凑合的作品，第三次是暑假实践做游戏时，由于各自做各自的，分工不清晰，前期讨论不充分，拿到不经过充分的交流就开始做，导致最后的答辩及其失败，项目完成度也特别低。

这次我们尝试了软件工程的开发方法并且在开学前就加强了沟通，与之前失败的经历相比我们投入了更多精力在前期准备上，首先根据资料上的流水线结构结合 TEC-5 计算机模型搭建了一个框架原型，这个原型仅仅是含有没有内容的模块和线路，这样做的好处的一方面是可以让我们在搭建的过程中对流水线有更好的认识，另一方面是可以按模块的拼装，并且每个模块可以单独测试，避免了之后调试的很多麻烦。在确定实现的功能后对各个模块进行了简化，仅仅保留了需要的接口，感觉就像 CPU 的制造一样，采用屏蔽不需要的器件来简化以避免当需要添加功能时发现没有接口的尴尬。

在操作中证明了这样的设计方法在处理规模稍大的项目时是及其有效的，各个封装的子图使得结构工整，飞线的少量使用让数据的流向变得直观。尽量保持项目与数据通路图排布相似，这样可以让我们发现错误更快更及时。最后在我们的调试中，有效避免了一些结构上的错误，减轻了许多压力。

除了开发的方法以外我觉得这次更重要的是心态的培养。倘若对一些稍微复杂的系统有畏难的情节，那么必然无法交出满意的答卷，这次是我第一次尝试稍难的课题，尽管不是很成功，但是感受到了心态上没有这么畏难了，在面对错误时也没有这么浮躁，由于时间原因一些错误还来不及解决，也算是有遗憾吧。

当然这次课设告诉了我要永远对项目保持一颗敬畏之心，你永远不要试图认为逻辑是对的就是对的，这次我们就发现了 Proteus 的一些错误和我们从未遇见的毛刺现象，要时刻保持学习，用知识才能解决问题，而不是仅仅依靠自己的经验和一些技巧。

最后要感觉成永康同学和范毛烨同学的辛勤劳动和配合，感谢赵力老师对我们知识的灌溉、热情的帮助，解决我们的疑难杂症并发现了电路中的毛刺现象，还有所有提供帮助的同学。

尽管完成度还不算很高，但这次宝贵的经验让我面对课设的信心有了提升，希望自己能够面对接下来的课设的考验。