计算机组成原理大实验报告

计44 喻明理 2014011372

计44 程书宇 2014011391

计42 胡润邦 2014011302

目录

[1. 大实验概述 1](#_Toc469171071)

[1.1 实验目标 1](#_Toc469171072)

[1.2 实验硬件环境 1](#_Toc469171073)

[1.3 实验软件环境 1](#_Toc469171074)

[2. 数据通路及各模块设计 2](#_Toc469171075)

[2.3 RegFile模块 3](#_Toc469171076)

[2.4 EXE模块 4](#_Toc469171077)

[2.4 Mem模块 8](#_Toc469171078)

[2.5 WriteBack 模块 9](#_Toc469171079)

[4. 扩展功能各模块详述 13](#_Toc469171080)

[4.1 支持Flash 自启动 13](#_Toc469171081)

[4.2 支持扩展指令实现VGA 显示 14](#_Toc469171082)

[6. 一些有待改进的工作 17](#_Toc469171083)

[6.1 支持键盘的输入 17](#_Toc469171084)

[6.2 丰富VGA 的显示 17](#_Toc469171085)

# 1. 大实验概述

### 1.1 实验目标

1. 设计并实现5 段流水线CPU，支持MIPS16E 指令，包含25 条基本指令和5 条扩展

指令

2. 支持与Term 终端程序的串口通信

3. 支持Flash 自启动

4. 支持VGA 显示寄存器内容

### 1.2 实验硬件环境

1. FPGA 芯片: Spartan-3E 系列的 XC3S1200EFGG320 芯片，320 个管脚，120 万门，

FBGA 封装

2. CPLD 芯片: XC9500 系列的 XC95144XL-7TQ100 芯片，100 个管脚，144 个宏单元，

采用 TQFP 封装

3. SRAM : 型号为 IS61LV25616-10TI，电压需求为3.3V，每片存储容量为 256K\*16b

4. Flash : MT28F640J3，其数据线为16 位，地址线为23 位

5. 带VGA 输入的显示器

### 1.3 实验软件环境

1. FPGA 开发工具软件Xilinx ISE 14.4

2. FlashAndRam.exe 支持对flash，ram1，ram2 的读写

3. Term.exe 支持与监控程序的串口通信

4. Assembler.exe 将改写后的汇编版本监控程序转为二进制机器码

# 2. 数据通路及各模块设计

数据通路图终版

|  |  |
| --- | --- |
| 寄存器编号 | 对应寄存器 |
| 0000 | R0 |
| 0001 | R1 |
| 0010 | R2 |
| 0011 | R3 |
| 0100 | R4 |
| 0101 | R5 |
| 0110 | R6 |
| 0111 | R7 |
| 1000 | SP |
| 1001 | IH |
| 1010 | RA |
| 1011 | T |

2.1 InstructionFetch 模块

**2.1.1** 模块功能概述

1. 从PC 中读出指令地址

2. 从指令寄存器RAM2 中读出16 位指令，存入段寄存器

3. 向PC 中写入下一条指令的地址，下一条指令的地址有2 个选择，一个是当前PC 加

1，另一个是从ID 段计算得到的跳转地址，此处选择的控制信号同样来自ID 段。

4. 分支预测技术：对于之前执行过的跳转指令，保存其指令地址以及跳转到的指令地

址，从而实现一个简单的分支预测功能（在控制冲突部分详述）

5. 通过来自ALU段数据和挂起流水线支持对RAM2的读写操作（在结构冲突部分详述）

**2.1.2** 模块数据接口

接口名称 类型 说明

Clk In 50MHz 时钟输入

Rst In reset 按键输入

Start In 0 表示Flash 自启动阶段已经结束，可以正常运行

pc\_jump In ID 段计算的上一条指令正确的跳转地址

bypass\_pc\_jump In 0 表示ID 段判断给出的跳转地址可能有误，暂停一个周期

stopAll In 0 表示需要读写RAM2，暂停一个周期

sw\_in In 0 表示此时的MEM 段需要写内存，

lw\_in In 0 表示此时的MEM 段需要读内存

Addr In ALU 段给出的访存地址，判断是否需要读写RAM2

Data In ALU 段给出的存储数据，用于写入RAM2

data\_r Out 传递给MEM 段的RAM2 的数据

key\_signal In 0 表示键盘有按键输入

key\_value In 键盘输入的字符编码

key\_value\_r Out 传递给MEM 段的键盘输入

Pc Inout 当前指令地址

instruction Inout 当前的指令

Rpc Out 下一条指令的地址

2015-2016 秋季学期 大实验·实验报告 《计算机组成原理》

张浩天 高胜寒 王梓仲 7

predictErr Inout 0 表示预测错误，ID 段停止指令的译码

nop\_out Out 0 表示该条指令为空

ram2\_en Out RAM2 的使能

ram2\_oe Out RAM2 的读取使能

ram2\_we out RAM2 的写入使能

ram2addr Out RAM2 的地址线

ram2data Inout RAM2 的数据线

### 2.3 RegFile模块

#### 2.3.1 模块功能概述

1. 寄存器组包括8个通用寄存器和增加的T、IH、SP、RA寄存器

2. 根据控制信号和时钟将数据写入寄存器或从寄存器读数据

3. 将寄存器的值通过VGA在显示器上显示

#### 2.3.2 模块数据接口

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 类型 | 名称 | 长度 | 来源或去处 | 注释 |
| In | clk | 1 | 板子 |  |
|  | clk25 | 1 | 板子 |  |
|  | rst | 1 | 板子 |  |
|  | reg\_readno1 | 4 | ID | 读第一个寄存器的编号 |
|  | reg\_readno2 | 4 | ID | 读第二个寄存器的编号 |
|  | reg\_writeno | 4 | WB | 写寄存器的编号 |
|  | reg\_writedata | 16 | WB | 写入寄存器的内容 |
|  | reg\_we | 1 | WB | 是否写入 |
| out | id\_readdata1 | 16 | ID | 读出的第一个寄存器的内容 |
|  | id\_readdata2 | 16 | ID | 读出的第二个寄存器的内容 |
|  | id\_T | 16 | ID | T的内容 |
|  | id\_SP | 16 | ID | SP的内容 |
|  | id\_IH | 16 | ID | IH的内容 |
|  | id\_RA | 16 | ID | RA的内容 |
|  | vga\_r0 | 16 | VGA | 往VGA输出的寄存器内容（下同） |
|  | vga\_r1 | 16 | VGA |  |
|  | vga\_r2 | 16 | VGA |  |
|  | vga\_r3 | 16 | VGA |  |
|  | vga\_r4 | 16 | VGA |  |
|  | vga\_r5 | 16 | VGA |  |
|  | vga\_r6 | 16 | VGA |  |
|  | vga\_r7 | 16 | VGA |  |

### 2.4 EXE模块

#### 2.4.1 模块功能概述

接收ID段的操作数，根据控制信号执行制定的运算，结果写入寄存中。

#### 2.4.2 ALU操作号

|  |  |
| --- | --- |
| 操作号 | 操作 |
| 0000 | ADD |
| 0001 | SUB |
| 0010 | AND |
| 0011 | OR |
| 0110 | CMP |
| 1001 | NOT |
| 0100 | SLL |
| 0101 | SRA |
| 0111 | SLTUI |

#### 2.4.3 模块数据接口

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 类型 | 名称 | 长度 | 来源或去处 | 注释 |
| In | exe\_alu1\_operand1 | 16 | ID | ALU第一个操作数 |
|  | exe\_alu1\_operand2 | 16 | ID | ALU第二个操作数 |
|  | exe\_alu1\_opkind | 4 | ID | ALU运算类型 |
|  | exe\_regwrite | 1 | ID | 是否写入寄存器 |
|  | exe\_regdst | 4 | ID | 写入哪个寄存器 |
|  | exe\_memwrite | 1 | ID | 是否SW |
|  | exe\_memwritedata | 16 | ID | 写入内存的内容 |
|  | exe\_memtoreg | 1 | ID | 写回的内容是来自ALU的输出还是访存得到的数据 |
| out | mem\_regwrite | 1 | MEM | 是否需要写入寄存器 |
|  | mem\_regdst | 4 | MEM | 写入哪个寄存器 |
|  | mem\_wboraddr | 16 | MEM | ALU1算出的结果 |
|  | mem\_memwrite | 1 | MEM | 是否SW |
|  | mem\_memtoreg | 1 | MEM | 写回的内容是来自ALU的输出还是访存得到的数据 |
|  | pang\_mem\_memwrite | 1 | MEM | 是否SW传给旁路 |
|  | pang\_mem\_wboraddr | 16 | MEM | ALU1算出的结果 |
|  | pang\_mem\_memwritedata | 16 | MEM | 写入内存的内容 |
|  | pang\_mem\_memtoreg | 1 | MEM | 写回的内容是来自ALU的输出还是访存得到的数据 |
|  | if\_wboraddr | 16 | IF | ALU1算出的结果，直接传给IF段 |
|  | if\_memwrite | 1 | IF | 是否SW传给IF段 |
|  | if\_memwritedata | 16 | IF | 写入内容的内容传给IF段 |
|  | if\_flush\_from\_exe | 1 | IF | PC是否不能+1（此时在按MEM的要求读写指令地址区） |
|  | if\_id\_flush\_from\_exe | 1 | IF | PC是否不能+1 |
|  | id\_regwrite\_from\_exe | 1 | ID | 是否需要写入寄存器 |
|  | id\_regwritedata\_from\_exe | 16 | ID | 写入内存的内容 |
|  | id\_regdst\_from\_exe | 4 | ID | 写入哪个寄存器 |
|  | id\_isLW\_from\_exe | 1 | ID | 是否读取 |

2.2 InstructionDecode 模块

**2.2.1** 模块功能概述

1. 从指令操作码中译出所有控制信号、源寄存器目标寄存器编号，存入段寄存器

2. 从2 个源寄存器中读出2 个操作数A,B，并检查来自MEM 段的数据旁路，存入段寄

存器。

3. 对立即数进行符号扩展或0 扩展，存入段寄存器

4. 对于跳转指令，检查来自MEM 段的数据旁路，计算出正确的跳转地址，接回IF 段

给出PC 正确的跳转地址，检验分支预测的结果；同时检查来自ALU 段的旁路，判断

如果由于数据冲突本周期不能给出正确的跳转地址，向IF 段发出预测不可靠的信号。

**2.2.2** 模块数据接口

接口名称 类型 说明

clk In 50MHz 时钟输入

rst In reset 按键输入

start In 0 表示Flash 自启动阶段已经结束，可以正常运行

predictClr In 0 表示IF 段给出的这条指令预测错误，需要把它清空

stopAll In 0 表示需要读写RAM2，暂停一个周期

nop\_in In 0 表示这条指令为NOP 指令，直接传给下一段

instruction In IF 段读出的16 位指令

pc In 下一条指令的地址

regist In 寄存器

bypassExR In ALU 段的目标寄存器，对于跳转指令用于判断是否能给出可

靠的跳转地址。

bypassMemR In MEM 段的目标寄存器编号，用于数据旁路解决数据冲突

bypassMemdata In MEM 段的数据，用于数据旁路解决数据冲突

bypassBubble Out 传递给IF 段的控制信号，表示跳转指令的跳转地址

2015-2016 秋季学期 大实验·实验报告 《计算机组成原理》

张浩天 高胜寒 王梓仲 8

nop\_out Out 0 表示这条指令为NOP 指令，所有不访存且不写回的指令如

跳转指令均会传递0

npc Out 传递给IF 的新的跳转地址

lw Out 0 表示该指令需要读取存储器

sw Out 0 表示该指令需要写入存储器

wb Out 0 表示该指令需要写入寄存器

imOrry Out 0 表示ALU 段应使用源寄存器A 与立即数运算，1 表示应与源

寄存器B 运算

rx Out 源寄存器A 编号

ry Out 源寄存器B 编号

rz Out 目标寄存器编号

alu Out ALU 段的操作编号

im Out 0 扩展或符号扩展后的立即数

rxdata Out 源寄存器A 的数值

rydata Out 源寄存器B 的数值

2.3 Execute 模块

**2.3.1** 模块功能概述

1. 选择ALU 的第一个操作数：一个是段寄存器中保存的操作数A，另一个是旁路传递

的操作数

2. 选择ALU 的第二个操作数：第一次从段寄存器保存的操作数B 和旁路传递的操作数

选择，第二次从第一步的选择和立即数的符号扩展选择

3. ALU 根据控制信号，进行指定的运算，结果写入段寄存器

**2.3.2 ALU** 操作号

操作号 操作

000 x+y

001 x-y

010 x and y

011 x or y

100 x sll y

101 x sra y

110 比较，x=y 时为0，x/=y 时为1

2015-2016 秋季学期 大实验·实验报告 《计算机组成原理》

张浩天 高胜寒 王梓仲 9

**2.3.3** 模块数据接口

接口名称 类型 说明

clk In 50MHz 时钟输入

rst In reset 按键输入

start In 0 表示Flash 自启动阶段已经结束，可以正常运行

stopAll In 0 表示需要读写RAM2，暂停一个周期

nop\_in In 0 表示这条指令为NOP 指令，直接传给下一段

rx In 源寄存器A 的编号，用于完成数据旁路

ry In 源寄存器B 的编号，用于完成数据旁路

rz\_in In 目标寄存器编号，直接传给下一段

lw\_in In 访存控制信号，直接传给下一段

sw\_in In 访存控制信号，直接传给下一段

wb\_in In 写回控制信号，直接传给下一段

alu In 算术逻辑单元运算编号，具体操作参考运算编号表

im In 立即数

rxdata In 通过MEM/ID 旁路的源寄存器A 的数值

rydata In 通过MEM/ID 旁路的源寄存器B 的数值

imOrry In 用于立即数与源寄存器B 选择的控制信号

bypassMemR In MEM 段的目标寄存器编号，用于数据旁路解决数据冲突

bypassMemdata In MEM 段的数据，用于数据旁路解决数据冲突

rz\_out Out 传递给MEM 段的目标寄存器编号

lw\_out Out 传递给MEM 段的控制信号

sw\_out Out 传递给MEM 段的控制信号

wb\_out Out 传递给MEM 段的控制信号

nop\_out Out 0 表示这条指令为NOP 指令，所有不访存且不写回的指令如

跳转指令均会传递0

aludata Out ALU 的运行结果

exdata Out 源寄存器B 的经过旁路判断后的真实数据，为sw 操作存入存

储器的数值

### 2.4 Mem模块

#### 2.4.1 模块功能概述

1. 根据存储器控制信号，执行数据存储器RAM1 的读或写操作，地址和写入数据来自

上一个段寄存器

2. 根据ALU 给出的地址，判断是否为访问数据存储器RAM1，是否为访问用户程序区

RAM2，是否访问串口，是否访问VGA 模块，并执行相应操作.如果是访问用户程序区，则什么操作都不做。

3. 将读取存储器的数据或来自ALU 得到的结果写入下一个段间寄存器，用于

数据写回。

4. 在第一个状态变化的时候要用旁路传来的判断信号进行判断。

#### 2.4.2 模块数据接口

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 数据接口 | 类型 | 长度 | 来源（in）或去处(out) | 说明 |
| mem\_regwrite | in | 1 | exe/mem | 是否需要写入寄存器 |
| mem\_wboraddr | in | 16 | exe/mem | ALU1算出的结果 |
| mem\_regdst | in | 4 | exe/mem | 要写入的寄存器的编号 |
| mem\_memwrite | in | 1 | exe/mem | 指令是否是SW |
| mem\_memtoreg | in | 1 | exe/mem | 写回的内容是来自ALU的输出还是访存得到的数据 |
| mem\_inst | in | 16 | if | IF段读到的指令 |
| wb\_regwrite | out | 1 | wb/mem | 是否需要写入寄存器 |
| wb\_data | out | 16 | wb/mem | 写回寄存器的数据 |
| wb\_regdst | out | 4 | wb/mem | 写入哪个寄存器 |
| id\_regwrite\_from\_mem | out | 1 | id | 是否需要写入寄存器 |
| id\_regwritedata\_from\_mem | out | 16 | id | 写回寄存器的数据 |
| id\_regdst\_from\_mem | out | 4 | id | 写入哪个寄存器 |
| pang\_mem\_memwrite | in | 1 | exe | 指令是否是SW |
| pang\_mem\_wboraddr | in | 16 | exe | ALU1算出的结果 |
| pang\_mem\_memwritedata | in | 16 | exe | 写回寄存器的数据 |
| pang\_mem\_memtoreg | in | 1 | exe | 写回的内容是来自ALU的输出还是访存得到的数据 |
| ram1\_addr | out | 18 | 板子 | ram1的地址 |
| ram1\_data | inout | 16 | fontRom | ram1的数据 |
| ram1\_we | out | 1 | 板子 | RAM1 的输出使能 |
| ram1\_oe | out | 1 | 板子 | RAM1 的读取使能 |
| ram1\_en | out | 1 | 板子 | RAM1 的使能 |
| wrn | out | 1 | 板子 | 串口输出开关 |
| rdn | out | 1 | 板子 | 串口读取开关 |
| tbre | in | 1 | 板子 | 串口输出准备信号 |
| tsre | in | 1 | 板子 | 串口输出准备信号 |
| reset | in | 1 | 板子 | reset 按键输入 |
| clk25 | in | 1 | 板子 | 25MHz 时钟输入 |
| clk | in | 1 | 板子 | 50MHz时钟输入 |

### 2.5 WriteBack 模块

若指令为写回操作，则将数值写入目标寄存器。

**3.** 冲突处理

3.1 结构冲突

**3.1.1** 冲突原因

在实际运行监控程序的时候，需要考虑地址划分，0x4000 至0x7FFF 是用户程序区，

0x8000 至0xBEFF 为系统数据区，其中还有0xBF00 至0xBF03 为串口相关信息。由于当前程

序采用5 级流水线的方式运行，所以存在IF（指令寄存器）与MEM（数据寄存器）段同时

访问存储器的情况，导致发生结构冲突。

**3.1.2** 解决方案——分开存储

我们采取的解决方法是将指令存入RAM2，将数据存入RAM1。这样如果所有读取或存

储地址都在系统数据区的范围时，只要正常运行流水线，指令读取RAM2，数据访问RAM1，

并不会产生冲突。在此基础上，我们加入了串口的操作，串口在MEM 段实现，如果地址与

串口相关，则此时关闭数据寄存器的使能，打开串口的读或写，可以在一个周期内完成。这

样的情况下读写串口时并不影响数据存储器，更不会影响到指令存储器。

**3.1.3** 解决方案——暂停流水线

然而，在执行监控程序时，需要使用A 指令的操作改写指令寄存器，或U 指令读取指

令寄存器，所以在MEM 段可能需要访问指令寄存器。在MEM 段，如果地址指向了用户程

序区，说明必须访问指令寄存器，于是当前的IF 段必须暂停，且此后与此条指令相关的指

令译码等都会因此无效。

我们采取的方法是暂停整个流水线一个周期，在这一轮中指令寄存器完成访存操作，并

且将数据传给数据存储器，那么在下一轮开始的时候，数据存储器完成了对应指令应该完成

2015-2016 秋季学期 大实验·实验报告 《计算机组成原理》

张浩天 高胜寒 王梓仲 12

的事情，此后将可能需要的数据传给WB 段。

**3.1.4** 具体实现

具体实现方法是将ALU 段计算出来的地址和数据同时传给IF 段和MEM 段，MEM 段检

查地址是否指向用户程序区，如果是则给出全局挂起的信号（置0），表示暂停一个周期，在

下一个周期撤销全局挂起信号（置1），表示流水线恢复，并且如果之前是lw 操作则收取来

自IF 段的结果。IF 段也做地址的检查，如果是用户程序区则立即终止指令的读取，转而访

问用户程序区，此后如果是lw 操作则将结果传到MEM 段，接下来的一个周期，虽然ALU

段传来的地址和数据没有改变，但可以根据MEM 段给出的全局挂起信号得知之前的访问用

户程序区操作已经完成，需要重新开始读取指令，流水线各段恢复正常。

在暂停流水线的这一轮中，仅有IF 段在访存、MEM 段在修改全局的控制信号，而其它

阶段都会保持不执行操作，其阶段寄存器也不会被修改。这个这样的设计在结果上来看就是，

对于ID、ALU、WB 段来说，访问指令寄存器操作就像是普通的访存一样。

**3.1.5** 补充与总结

此后，我们加入了VGA 与键盘的读写（我们特殊规定了0xBF20、0xBF30 的地址），这些

操作也是在MEM 段完成，和访问串口一样，VGA 与键盘的操作不会对存储器产生影响。

至此我们解决了结构冲突的问题。

3.2 数据冲突

**3.2.1** 冲突原因

数据冲突是在运行汇编指令时时常发生的一种冲突，形成原因是之前指令的目标寄存器

在将结果储存回寄存器之前就被之后的指令所用到。

由于我们采用了5 段流水线操作，ID 段在一个周期的后半段的时候将会读取寄存器，而WB

段将会在一个周期的前半段存储寄存器，中间隔着ALU 与MEM 段，也就是说ID 段与ALU

或MEM 段的指令可能存在数据冲突，换句话说如果两条指令间相差超过2 条指令则不会发

生数据冲突，所以在这种情况下仅需检查相邻指令或相隔一条指令的数据冲突即可。

**3.2.2** 解决方案

我们的解决方案仅使用两条旁路，分别完成相邻指令和相隔指令，两条指令的旁路的目

标寄存器都来自ALU 与MEM 段间的寄存器，旁路数据都来自MEM 段的RAM1 数据线。对

于相邻指令，在ALU 段执行运算操作前获取目标寄存器和旁路数据，得到寄存器实际的数

2015-2016 秋季学期 大实验·实验报告 《计算机组成原理》

张浩天 高胜寒 王梓仲 13

据后再开始执行实际运算；对于相隔的指令，在ID 段先译码，获取目标寄存器和旁路数据，

得到寄存器实际的数据后存入ID 与ALU 段间的寄存器。

**3.2.3** 时序分析

考虑实际的时序逻辑，首先考虑一个周期的前半段，MEM 段目标寄存器的值保持不变，

此外需要获取段间寄存器传递的控制信号（是否写回寄存器、是否读取内存、是否写入内存），

如果是写入内存操作则不需要旁路（对于不需要旁路的指令在ID 阶段将目标寄存器设置为

1111，即永远不使用的寄存器）；如果是读取内存操作则需要旁路，MEM 段给出地址，打开

RAM1 的使能端，那么我们认为在周期后半段开始时MEM 的RAM1 数据线上已经得到了目

标寄存器所对应的正确数据；如果是非读取内存的写回寄存器操作，则直接将需要写回的数

据写到RAM1 的数据线上。

这样在一个周期的后半段，ID 段应当执行读取当前寄存器数值和返回实际PC 值（一般

是rpc，对于J 和B 指令按照寄存器的情况处理）的操作，此时正确的目标寄存器对应的旁

路数值已经可以取到，由此可以在寄存器与旁路数据中选择后传到ID 与ALU 段间的寄存器

和J 与B 指令的判断。在ALU 段应当执行运算操作，而此时正确的目标寄存器对应的旁路数

值也可以取到，在执行运算前先在ID 与ALU 段间的寄存器与旁路数据中做出选择后再执行

运算操作。综上，这样总可以保证到达ALU 与MEM 段间的寄存器的数值正确，即MEM 段

和WB 段得到的数据都是可靠的。

**3.2.4** 举例说明

举一个例子，如对同一个寄存器连续执行了三个指令，分别是LW，ADDIU 和SW。则当

SW 指令进入ID 段时LW 和ADDIU 段的结果均没有写回，ADDIU 在ALU 段，LW 在MEM 段，

此时ADDIU 与SW 均会得到来自LW 指令的旁路数据，这样ADDIU 能执行正确的运算，SW

会得到LW 时得到的寄存器数据。接下来SW 进入ALU 段，ADDIU 进入MEM 段，按照之前

的做法，ADDIU 会将写回的数据直接放到RAM1 数据线上，这样SW 在ALU 段可以得到ADDIU

后寄存器的值，最后SW 进入MEM 段可以将正确的值写进内存。

**3.2.5** 添加气泡的情况

但是上述的旁路存在一个严重的问题，就是旁路只能保证数据进入ALU 与MEM 的段间

寄存器是正确的，但由于没有从ALU 段到ID 段的旁路，如果在一个带写回的操作后紧跟着

一个J 或B 指令时会来不及完成（如果是相隔指令旁路数据就已经可以从MEM 段获取到）。

所以解决方案就是向IF 段发出J 或B 指令不可靠的信号，并清空ID 与ALU 段间的寄存器

（插入气泡），IF 段将会不更新PC 值（由于new PC 不可靠，也不可以更新），且保持IF 与

ID 段间的寄存器，重做该J 或B 指令。这样写回操作进入MEM 段，J 或B 操作仍在ID 段，

ALU 段为NOP，相当于在汇编代码中J 或B 指令前加入了NOP。

2015-2016 秋季学期 大实验·实验报告 《计算机组成原理》

张浩天 高胜寒 王梓仲 14

**3.2.6** 总结

所以综上，仅仅除了在写回操作后紧跟跳转指令需要浪费一个周期以外，其它情况均可

以连续执行。

3.3 控制冲突

**3.3.1** 冲突说明

控制冲突是指对于一条条件跳转指令，我们在ID 段也就是流水线的第二段执行完毕才

能给出正确的跳转地址，然而同一周期内IF 段需要将PC 置为这个跳转地址，并读出下一条

指令。显然，我们无法在同一周期内获得正确的跳转地址，除非让IF 段暂停一个周期，因此

会产生控制冲突。

**3.3.2** 解决方法

我们采取的解决方法是分支预测技术。根据程序运行的时间局部性和空间局部性，某一

条跳转指令很可能在不久之后被再次执行。那么只要我们在第一次执行时，存下它的跳转地

址，那么当第二次执行这条相同的指令时，就会有很大的概率再次跳去同一个地址，这样我

们便可以进行分支预测。

具体的实现方法是这样的：在InstructionFetch 这个模块中，维护一个跳转表，表的每一

项内容包括跳转指令的PC 和这条指令上次执行的跳转地址。在IF 执行的每个周期开始，将

PC 的值修改为上一周期预测的指令地址；当从指令存储器读出指令后，首先会判断是否为

跳转指令。如果是无条件跳转指令，那么直接计算跳转地址赋给PC 的预测寄存器，如果是

条件跳转指令，就遍历跳转表，依次与每一项的PC 比较，如果相同，就把上次的跳转地址

赋给PC 的预测寄存器。如果不是跳转指令，或者没有在跳转表中命中，那么就默认把当前

PC+1 赋给PC 的预测寄存器。

在同一周期内，ID 段已经把上一条指令正确的跳转地址计算好并旁路给了IF 段，在下

一个周期的开始，IF 段会首先比较这个正确的跳转地址和当前的PC 是否相同，如果相同，

表明当前这个PC 的预测是正确的，那么直接把PC 修改为上一周期预测的地址；如果发现

不同，说明当前这个PC 的预测是错误的，那么这一周期内ID 段正在执行的指令就是错误

的。IF 会立即向ID 段发出一个预测错误的信号（置0），当ID 段看到这个信号的时候，就会

把当前指令视为NOP 指令。同时IF 也会把PC 修改为那个旁路过来的正确的跳转地址，并

在这个周期给出正确的指令。当预测错误发生时，也会相应的修改或增加跳转表的表项。

2015-2016 秋季学期 大实验·实验报告 《计算机组成原理》

张浩天 高胜寒 王梓仲 15

**3.3.3** 实现成果

按照这个分支预测的实现，对于无条件跳转指令，不回耽误流水线的指令。对于条件跳

转指令，如果本次的跳转地址维持和上一次相同，那么预测结果正确，同样不回耽误流水线

的执行；如果是第一次出现的跳转指令或者跳转结果改变了，那么预测错误也只会耽误流水

线的一个周期。举例来说，对于一个10000 次的for 循环，只有第一次和最后一次跳转需要

耽误一个周期，中间的9998 次跳转都会直接执行，大大提高了流水线的效率。

# 4. 扩展功能各模块详述

### 4.1 支持Flash 自启动

#### 4.1.1 功能概述

由于RAM 断电后无法保存数据，因此需要每次上电后，利用FlashAndRam.exe 将监控

程序写入RAM2。为了避免这一麻烦的操作，将监控程序保存在Flash 中，每次上电后，由

CPU 程序控制将Flash 中的监控程序写入RAM2 后，自动开始执行监控程序。

#### 4.1.2 模块数据接口（FlashLoader 模块）

|  |  |  |  |
| --- | --- | --- | --- |
| 类型 | 名称 | 长度 | 注释 |
| in | clk | 1 |  |
|  | rst | 1 |  |
|  | start | 1 | 0表示自启动阶段结束，本模块停止工作 |
| inout | flashData | 16 | Flash16位数据线 |
| inout | flashAddr | 23 | Flash23位地址线 |
| out | flashByte | 1 | 操作模式 |
|  | flashVpen | 1 | 写保护 常置1 |
|  | flashCE | 1 | 使能 |
|  | flashOE | 1 | 读 |
|  | flashWE | 1 | 写 |
|  | flashRP | 1 | 0重置 1工作 |
|  | ram2\_en | 1 | ram2的使能 |
|  | ram2\_oe | 1 | Ram2读使能 |
|  | ram2\_we | 1 | Ram2写使能 |
|  | ram2addr | 18 | Ram2 18位地址线 |
| inout | ram2data | 16 | Ram2 16位数据线 |

#### 4.1.3 实现原理

首先利用 FlashAndRam.exe 将监控程序的机器码写入 Flash。在 Top 模块下新加入一个模块 FlashLoader。这个模块主要维护一个全局的控制信号 start。每次按下 rst 后，将 start 置 为 1，在 start 为 1 的情况下，流水线上各个模块都不会工作，只有 FlashLoader 这个模块会工作。该模块内部运行一个状态机，每个状态机的循环周期内，先从 Flash 指定地址读出一 条指令，然后将这条指令写入 RAM2，然后地址加 1。直到把整个监控程序都写入 RAM2 后， 该模块会把 start 改为 0。当 start 为 0 后，流水线各个模块开始工作，而 FlashLoader 模块停止工作。

### 4.2 支持扩展指令实现VGA 显示

#### 4.2.1 功能概述

将每个寄存器的编号(r0到r7，T)和每个寄存器的值，PC和指令的值传给VGA模块，然后在VGA上实时显示。

#### 4.2.2 模块数据接口（VGA 模块）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 数据接口 | 类型 | 长度 | 来源（in）或去处(out) | 说明 |
| hs | out | 1 | 板子 | 列场同步信号 |
| vs | out | 1 | 板子 | 行场同步信号 |
| oRed | out | 3 | 板子 | 三原色Red 输出 |
| oGreen | out | 3 | 板子 | 三原色Green 输出 |
| oBlue | out | 3 | 板子 | 三原色Blue 输出 |
| r0 | in | 16 | RegFile | 寄存器R0的值 |
| r1 | in | 16 | RegFile | 寄存器R1的值 |
| r2 | in | 16 | RegFile | 寄存器R2的值 |
| r3 | in | 16 | RegFile | 寄存器R3的值 |
| r4 | in | 16 | RegFile | 寄存器R4的值 |
| r5 | in | 16 | RegFile | 寄存器R5的值 |
| r6 | in | 16 | RegFile | 寄存器R6的值 |
| r7 | in | 16 | RegFile | 寄存器R7的值 |
| romAddr | out | 11 | fontRom | rom地址 |
| romData | in | 8 | fontRom | rom 数据 |
| pc | in | 16 | IF | pc的值 |
| cm | in | 16 | IF | 当前的指令 |
| tdata | in | 4 | RegFile | 寄存器T的值 |
| reset | in | 1 | 板子 | reset 按键输入 |
| CLK\_in | in | 1 | 板子 | 50MHz 时钟输入 |

#### 4.2.3 实现原理

增加一个VGA 模块, 与之对应的是一个存储相应信息的rom模块。通过在XY定位控制中，对于固定的位置进行分类讨论来选择相应的显示信息。每个寄存器分为4块，分别控制显示寄存器的编号，冒号，和8,8位的数字。

对于rom模块，由于每个字符包含10\*10 个像素，每个像素对应9 位的颜色信息。我们通过c 程序将对应的所有可能用到的字符或者数字信息的像素信息存在一个.mif 文件中。按照从0 到9 再到Z 的顺序，每个字符内部按照先行后列的顺序，线性存储在这个.mif 文件中，最终得到一个深度为2048 行，宽度为8位的二进制数据。这些数据通过.mif 文件导入到fontRom模块中。这个ROM 模块时用ISE 自带的ip\_core 生成的。最终会将这些像素信息存在片内。

**5.** 实验中遇到的困难

5.1 变量使用方法错误

为了在一定程度上简化代码量，我们在IF 和ID 模块中使用了少量的变量。然而我们遇

到的最初的bug 就是变量的初始化错误。在最初的代码中，我们如果在某个process 中使用

变量，就在这个process 下的begin 之前声明这个变量，并赋初值。然而经过长时间的测试，

我们发现这样的写法并不能保证在process 每次执行的时候都给变量赋初值。正确的写法是，

在process 内部的语句中，在使用变量之前显式的赋初值，这样才能保证每次process 使用

这个变量时，都已经被初始化了。在这一点上，不得不说硬件编程和软件编程体现出了很大

的相似性，因为vhdl 中的变量其实就是仿照软件中的变量来使用的，因此初值不正确对程

序的影响都是非常巨大的。吸取了这次教训后，我们仔细检查了所有变量包括所有信号的初

值，保证了程序的正常运行。

5.2 时钟同步问题

在我们设计的流水线的5 个模块中，由于时序设计的不同，在每个流水线的周期中，IF、

ID、MEM 三个模块都需要2 个50M 时钟周期，而EX 和WB 模块只需要1 个25M 的时钟周

期。因此在最初的设计中，我们将50M 的时钟接给IF、ID、MEM 三个模块，将50M 二分频

的时钟接给EX 和WB 模块。但是我们在后续的测试中，发现程序运行非常不稳定。同样的

程序，每次按下reset 后，程序运行的结果有可能会变化，比如流水线卡死的位置不同，或

者Term 的显示也不稳定。我们花了2 天时间仔细研究了各个模块，都没有发现会导致不确

定因素的地方。

最后，我们的一个组员机智的发现，很可能是那个二分频的时钟和原时钟不同步造成的。

当然这个误差在一个周期内可能非常非常微小，甚至小于ns 的量级。但随着流水线的执行，

这个误差可能会积攒的越来越大，最后导致接入不同时钟的模块不能同步工作，那么就会发

生各种各样无法预料的错误了。

我们改进的方法是，所有模块都接入50M 的时钟，对于只需要一个周期的EX 和WB 模

块，在模块内部自己记录这2 个周期的次序，只挑需要的周期工作。经过这样简单的修改

后，我们的程序终于能够稳定运行了。这个问题应该算是我们本次大作业遇到的最大困难，

连续2 天的调试几乎令我们筋疲力尽。但回过头来想想，其实硬件的运行应该是相对稳定

的，如果出现不稳定的情况，其实我们应该第一时间去检查时序，而检查时序的首先就应该

检查时钟。由于经验缺乏，我们把重点放在了各个模块的调试，完全忽略了时钟，因此我们

总结认为在调试过程中，通过宏观的分析来缩小问题的范围是非常重要的。

2015-2016 秋季学期 大实验·实验报告 《计算机组成原理》

张浩天 高胜寒 王梓仲 20

5.3 RAM 的访存和读写串口的时序问题

在大实验的最后阶段，我们已经完成了所有基础功能和扩展功能的实现。然而我们的主

频一直被限制在12.5M，这样的运行频率令我们所有的效率优化都显得没有任何意义。因此，

我们的功能重点就放在了如何提高主频上。

当我们提高主频时，根据Term 的异常显示和流水线卡死的位置，我们最终认为问题出

在对RAM 的访存和读写串口上。按照我们最初的设计，MEM 段一共利用2 个周期的4 个沿

来运行状态机，之所以这样设计是为了保证RAM 的访存和读写串口有足够的数据建立时间

和数据保持时间。然而，经过队友的进一步检查，发现这个设计的最大问题是，控制信号的

交接太紧。比如在一个写RAM 的周期的最后，当we 被拉高的时候，就把oe 给拉低了。这

样很可能会导致写RAM 失败。

为了解决这个问题，我们尝试做小范围的修改，然而我们不幸发现一旦修改这里的时序，

整个MEM 段的时序都需要修改，更严重的是，由于读写RAM 也发生在IF 段，因此这个模

块的时序也需要大改。因此我们在最终检查的前3 天，重新设计了IF 和MEM 模块的时序，

重写了这2 个模块的代码。修改后的时序设计中，我们只利用2 个周期的2 个沿，忽略了

RAM 访存的数据建立时间，保证控制信号的时序稳定。经过这次修改，我们的CPU 的主频

瞬间飞跃到了20M。之后又经过一些对process 的逻辑优化，我们终于实现了25M 的主频。

在大实验的最后关头，我们几乎重写了整个工程一半的代码。不得不说，这样的工作完

全是因为我们最初的时序设计有问题。如果我们在最初设计的时候，能够更加严谨的发现这

个问题，我们就可以省下发现bug，以及重写代码的这部分时间，大大提前大作业的完成时

间。因此，我们深切体会到了一个优秀的设计的重要性。

5.5 VGA 的发送问题

实验的最终阶段，如6.4 中所说，键盘与VGA 同时产生了问题，刚出现这样的问题时，

我们开始了VGA 的测试，通过汇编代码向VGA 输出信号，发现VGA 显示的字符与预期的不

符，又由于在VGA 与CPU 相连之前并没有发现模块的错误，所以我们认定是在MEM 段与

VGA 的接口发生了错误。

本次试验中，VGA 只有写操作，在MEM 段内通过短暂的周期内将vga\_signal 置0、恢复

1，在vga 模块中检测vga\_signal 的边沿。一开始，显示屏时常显示上一次测试的字符，所以

我们发现我们并没有给出数据准备时间。在修复了这个问题以后，虽然出错几率减小了，但

错误字符的问题仍然存在，特别是在按下键盘对应的R 字母（输出用户寄存器的中断程序）

时显示往往不正常。

于是我们尝试了vga 模块不再检测vga\_signal 的边沿，而是CPU 同步时钟的方法，错误

得以解决。

在最终的键盘和VGA 的调试中，虽然问题比较严重，但是由于之前在RAM 的问题上我

们积累了一定的经验，问题都很快就解决了。

# 6. 一些有待改进的工作

### 6.1 支持键盘的输入

在目前的体系中没有来得及添加对于键盘输入的处理。在加入对键盘输入的时候需要及时储存键盘变化的输入，但是可能在键盘输入时要加入对整个流水线的中断，可能要对流水线的结构做出改变，但如果中断流水线的话，只能在当前中断来获取键盘输入的字符，如果在中断的过程中按下键盘（虽然只有不到1ms 的时间），那么这时的按键输入就会被系统忽略，无法读取。这样的实现显然也存在一些不合理之处。比较合理的实现方法是利用ISE 的ip\_core 生成一个FIFO 模块，利用这个队列来缓存键盘输入的字符。之后在中断处理程序中，我们可以通过这个队列来读取最新的键盘输入，并且可以扩展为读取最近输入的若干个字符，这样就可以实现比较复杂的命令。而不是简单的‘R’这种

单字符的命令。

### 6.2 丰富VGA 的显示

我们在VGA模块只是在固定的XY位置输出固定的寄存器的值，可以在mif文件写入某些图片或者更多素材的图片内容，如果图片的大小很大的话可以在ram里面单独开出一块区域来实现VGA的更多显示。