



32位MIPS CPU实验报告

计算机组成原理课程实验

计55班 张卡尔 2015011025

计51班 王士诚 2015011383

目录

1 实验内容	1
1.1 实验环境	1
1.2 实验目标	2
2 实验设计	2
2.1 整体设计	2
2.1.1 数据通路	2
2.1.2 设计思路	3
2.2 五级流水线设计	4
2.2.1 PC	4
2.2.2 IF取指	4
2.2.3 ID译码	4
2.2.4 EX	4
2.2.5 MEM	5
2.2.6 WB	5
2.3 CP0	5
2.4 CTRL模块	5
2.5 精确异常处理和中断	5
2.6 总线设计	6
2.7 串口	8
2.8 SRAM	8
2.9 VGA	8
3 实验结果	9
运行监控程序	9
图像显示	10
4 实验总结	11
选择合适的开发方式	11
合理安排时间	11
认真研究时序图，寻找潜在问题	11
使用调试工具	12

1 实验内容

1.1 实验环境

本实验使用的平台为Thinpad，其主要特点如下：

- 搭载Xilinx Artix-7系列FPGA：XC7A100T
- 100K LEs, 4.8Mb BRAM, 28nm制程
- 支持最新的Vivado设计工具
- 两组SRAM内存，每组4MB容量、32位数据线
- 8MB NOR Flash闪存
- 外围接口：SL811 USB, DM9000网卡, TFP410 DVI图像输出
- 板载控制芯片，支持远程或离线实验

本实验使用的编程环境为Vivado 2017.1，硬件描述语言为Verilog。

1.2 实验目标

本实验的目标是实现32位MIPS CPU，能够在CPU上运行32位监控程序¹，并支持通过串口与电脑上的32位Term程序²进行交互、支持VGA显示。具体来说，本实验实现了一下目标：

1. 五级流水线设计
2. 支持监控程序的指令和额外指令共46条
3. 支持HiLo寄存器操作
4. 支持CP0寄存器操作
5. 支持中断和异常处理
6. 支持串口通讯
7. 支持SRAM访问
8. 支持VGA显示
9. 支持Thinpad板上按钮输入
10. 多外设共用总线
11. 虚实地址转换（MMU）

2 实验设计

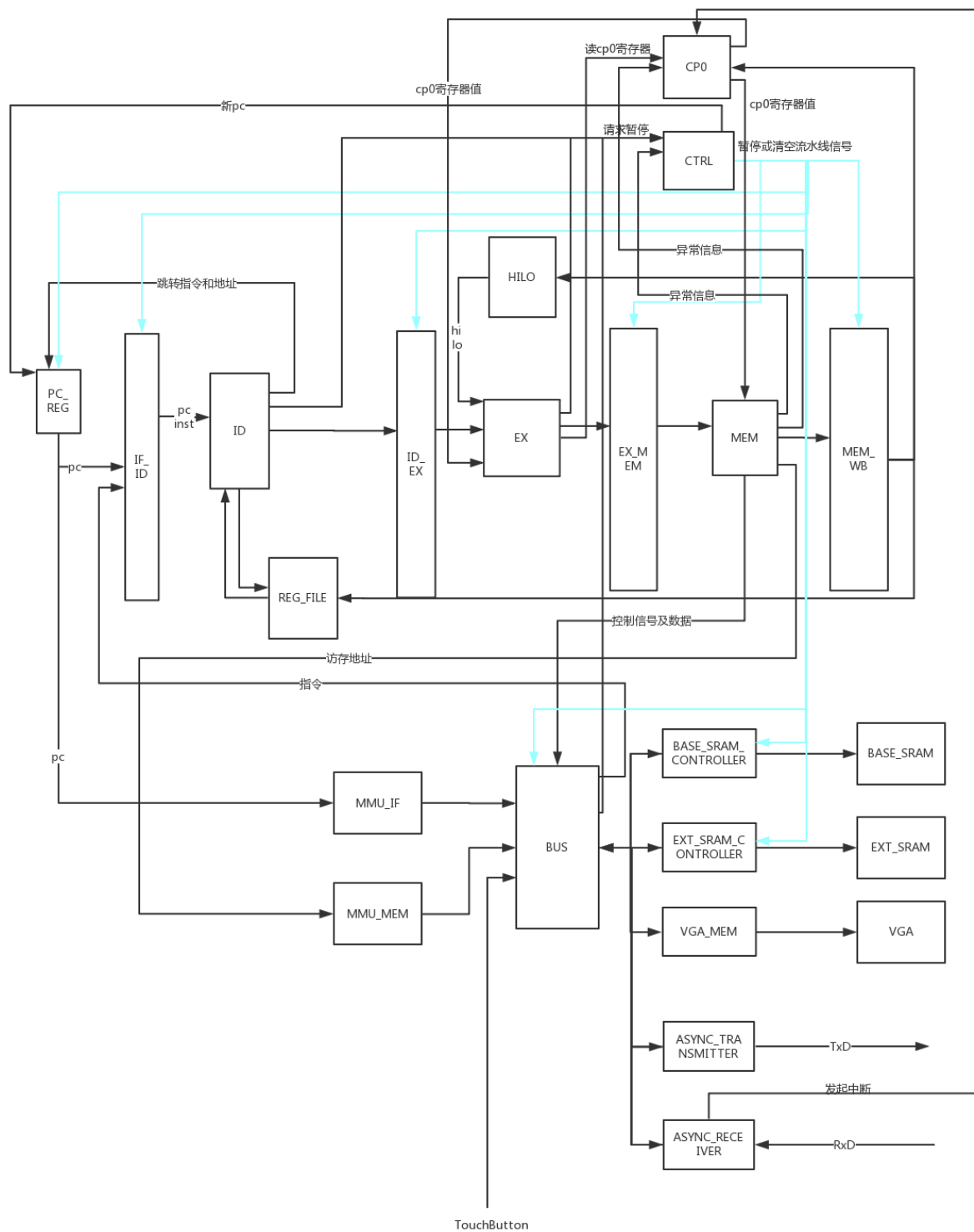
2.1 整体设计

2.1.1 数据通路

¹ 32位监控程序：<https://git.net9.org/wyl8899/Neptunus/tree/master/monitor>

² 32位Term程序：<https://git.net9.org/wyl8899/Neptunus/tree/master/term>

我们按照课件上介绍的32位MIPS流水CPU的设计方法，参考《自己动手写CPU》这本书，通过增量开发模式完成了CPU的设计，下图为最终的CPU数据通路图。



2.1.2 设计思路

我们采取了增量开发的模式，通过由简单到复杂、元件由少到多的顺序逐个实现指令集中的指令。在开发过程中根据新需求修改数据通路、增加接口和元件。使整个开发过程比较流畅和高效。

开发周期为：

分析需求 分析新指令所需的元件、接口、信号或者新功能所需的元件等，在旧有数据通路的基础上设计增量。

代码实现 在设计好新数据通路后，进行代码实现。代码实现过程中采用元件例化的思想，先设计出新元件的代码，然后在高层代码中进行例化并用信号线连接，使得设计、实现、仿真都十分清晰。

单元测试 在写完代码后写测试样例（由实现的新指令构成），进行仿真，比较结果正确与否进行修改。

集成 将测试通过的代码集成到原代码中。

使用了git进行版本控制和分支管理。

2.2 五级流水线设计

2.2.1 PC

PC是专用于指示程序执行位置的寄存器。

- 正常情况下，PC会自增4。
- 若译码阶段遇到跳转指令，PC会跳入分支目标地址。
- 当stall使能时，流水线暂停，PC的值保持不变。
- 当MEM模块捕捉到异常或ERET时，会清空流水线，PC被赋为异常处理程序的地址。

2.2.2 IF取指

取指模块接受PC输出的指令地址、从SRAM中读到的指令值，并负责在时钟上升沿时将其送到ID组合逻辑电路进行译码。如果流水线暂停，IF_ID寄存器会保留上一阶段的值。如果flush信号有效，则寄存器被清空，输出默认值，其效果等价于NOP指令。

2.2.3 ID译码

译码模块负责识别指令类型和各字段、读取寄存器模块中的值、产生流水线中的各种控制信号，并将操作数传递给EX模块，同时还进行分支判断处理、数据旁路。在精确异常处理中，译码模块还负责判断下一指令是否在延迟槽中。

2.2.4 EX

EX段为流水线的第三段，负责算术逻辑运算，主要实现了ALU功能，此外还将信号传入下一级。在EX中，会根据输入的运算类型和子类型信号做相应的运算，同时也会根据寄存器的读写控制信号向后传递相应的信号，在运算中还会检查异常情况并向后传递。

2.2.5 MEM

MEM作为访存阶段的控制逻辑，主要功能是根据输入控制信号确定访存指令，将相应的控制信号输入bus和MMU进行地址转换从而执行访存、访问串口和VGA等指令。

此外，MEM还作为精确异常的处理模块，根据ID和EXE阶段的输入异常信息及自己进行的检查，修改CP0中相关寄存器的字段，交由CP0进行异常处理。

需要注意的是，由于本实验所用的SRAM一个地址上的数据为32位，因此在读访存时需要去掉地址末两位，直接读取32位数据，再经选择器输出数据。在写访存时，需要将末两位地址转化为SRAM的BE位使能信号，从而写入某特定8位数据。

2.2.6 WB

MEM_WB寄存器负责写入寄存器，根据指令控制信号控制写普通寄存器、hilo寄存器和CP0寄存器的控制信号和地址、数据。

2.3 CP0

CP0作为协处理器负责相对独立与CPU进行系统控制。在我们实现的CPU中主要负责进行异常处理，以及作为MFC0、MTC0指令的硬件支持。

CP0中存有多组寄存器单元，负责存储系统配置、存储器状态、异常原因等信息，本CPU实现了如下寄存器：

- count 处理器计数周期
- compare 定时中断控制
- status 处理器状态和控制jicun
- cause 保存上一次异常原因
- epc 保存上一次异常的程序计数器
- config 配置寄存器
- prid 处理器标志和版本

在中断和异常处理中，CP0会根据cause寄存器和status寄存器等寄存器的相关字段进行判断，做出不同的处理。

2.4 CTRL模块

CTRL模块负责实现流水线暂停机制，其输入是来自IF、ID、EX、MEM模块的请求暂停信号stallreq。CTRL模块对暂停请求信号进行判断，然后输出暂停信号stall到PC和各个流水寄存器，从而控制PC的值和各个阶段的寄存器。在异常处理中，CTRL也负责给出PC异常处理例程入口地址，清除流水线。

2.5 精确异常处理和中断

本CPU实现异常处理的思路是：在流水线各个阶段收集异常信息，并传递到流水线的MEM阶段，在访存阶段进行统一处理：

- 译码阶段检查是否为系统调用异常、是否是ERET指令，是否是无效指令。
- 在执行阶段判断是否有自陷异常和移除异常
- 在访存阶段判断是否有中断发生。

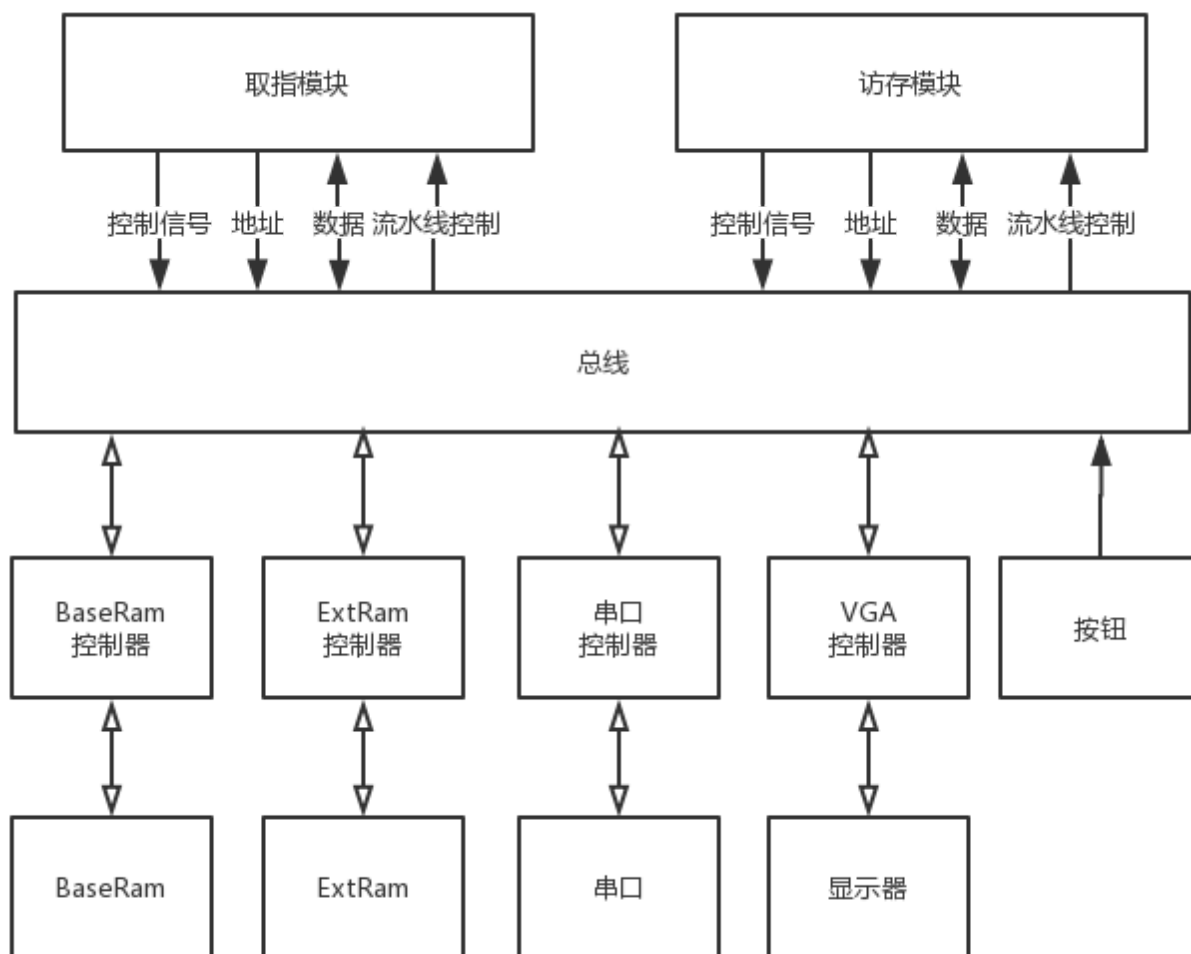
在访存阶段，将会结合CP0相关寄存器的值进行异常处理，如果需要处理，那么需要转移到异常对应的处理例程入口地址（ERET指令则转移到EPC寄存器中的地址），并清除流水线上除了回写阶段外的全部信息，修改CP0寄存器的值。

该过程需要ctrl模块配合完成。

关于异常例程地址，根据监控程序的要求，将各个异常的处理例程入口均设为0x80001180

异常类型	地址
0x00000001(中断异常)	0x80001180
0x00000008(系统调用)	0x80001180
0x0000000a(无效指令)	0x80001180
0x0000000c(溢出异常)	0x80001180
0x0000000d(自陷异常)	0x80001180
0x0000000e(返回异常)	EPC

2.6 总线设计



总线与取指模块、访存模块、两块SRAM的控制器、VGA的控制器和串口控制器连接。总线接受取指和访存的控制信号、地址和数据，产生对三种外设的控制信号，从而获取数据，并返回给CPU。当发生结构冲突时（如访存和取指都需要访问SRAM），总线负责仲裁主设备并暂停流水线。

本实验中地址映射如下：

地址	映射
0x8000.0000-0x800F.FFFF	监控程序代码
0x8010.0000-0x803F.FFFF	用户代码
0x8040.0000-0x807E.FFFF	用户数据
0x807F.0000-0x807F.FFFF	监控程序数据

0xBF00.03F8-0xBF00.03FC	串口
0xBD00.0000-0xBDFF.FFFF	VGA
0xBC00.0000	选择按钮

用户可以通过地址映射表编写程序，直接访问或修改外设数据。

2.7 串口

串口使用了助教提供的异步发送器和异步接受器代码，将其例化的元件接到了BUS上，由BUS处理访存指令，当访存指令的地址为0x1FD003F8和0x1FD003FC时，进行串口访问控制：

- 若地址为0x1FD003F8且为save指令，则TxD_start信号置高，TxD_data信号输入要写串口的数据。
- 若地址为0x1FD003F8且为load指令，则TxD_start信号置低，RxD_rdn信号置低，读取RxD_data信号到mem_data_o的低八位。
- 若地址为0x1FD003FC且为load指令，则RxD_rdn置高，TxD_start信号置低，mem_data_o[1:0]分别赋值RxD_dataready和TxD_ready。

串口的访问使用的是硬件中断，当检测到键盘输入是会引起中断，进入中断处理例程，从而读取到键盘的输入。

2.8 SRAM

根据SRAM的读写时序，为了减少取指阶段额外浪费的周期而提成CPI，本实验使用了纯组合逻辑方式设计了SRAM控制器。

- 读SRAM时，CE_N和OE_N均为低，WE_N为高，同时准备地址线，并把SRAM数据线置为高阻态等待获取SRAM数据。数据输出线直接和SRAM数据线相连。
- 写SRAM时，CE_N和WE_N为低，OE_N为高，同时准备地址线和数据线。由于准备数据线会有延迟，导致数据还未准备好就已经开始写入，因此WE_N在clk为低时才置低，为数据线保留半个周期的准备时间。

2.9 VGA

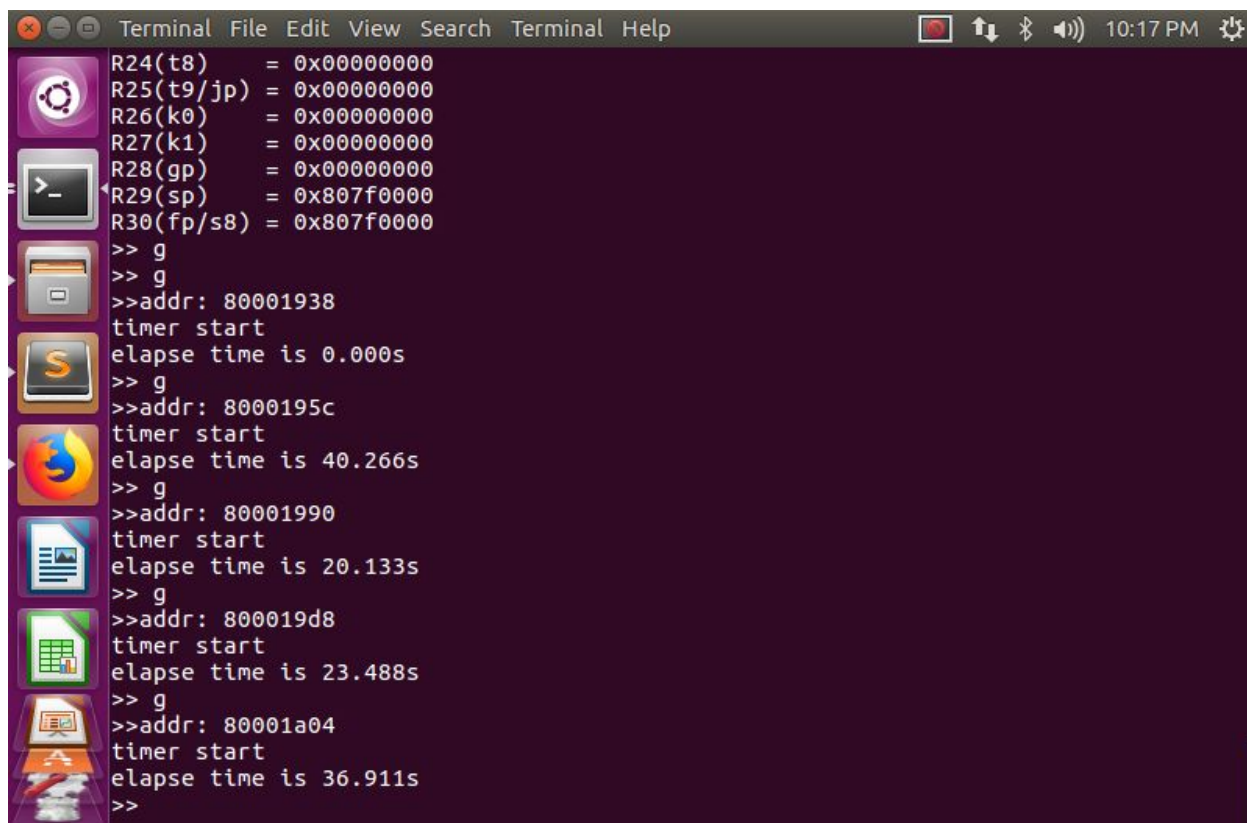


本实验使用640*480@60Hz VGA显示, 像素时钟为25.175MHz, 色彩为8bit, 分别为红色3位, 绿色3位, 蓝色2位。使用片内block ram作为显存, 顺序存储对应像素的色彩数据。block ram内一个地址上存32bit数据, 4个像素的数据存储在一个地址上, 一次性读出, 提高了图像的刷新速度。用户可以按照地址映射表编写程序, 直接修改某一像素上的色彩信息。

3 实验结果

3.1 运行监控程序

本实验CPU可以正常运行32位监控程序, 并在电脑上使用Term程序与监控程序进行交互。将性能测试程序拷入用户代码区, 并在Term中使用g指令运行, 得到如下结果:

A terminal window with a dark purple background and a sidebar of application icons on the left. The terminal displays MIPS assembly code and its execution output. The code includes register assignments for R24 through R30, followed by a loop of instructions: 'g', 'addr', 'timer start', and 'elapsed time'. The output shows the execution of these instructions, with addresses and elapsed times for each iteration. The time values are 0.000s, 40.266s, 20.133s, 23.488s, and 36.911s. The terminal window has a title bar with 'Terminal' and menu options like 'File', 'Edit', 'View', 'Search', 'Terminal', and 'Help'. The system tray on the right shows the time as 10:17 PM and various system icons.

```
Terminal File Edit View Search Terminal Help
R24(t8) = 0x00000000
R25(t9/jp) = 0x00000000
R26(k0) = 0x00000000
R27(k1) = 0x00000000
R28(gp) = 0x00000000
R29(sp) = 0x807f0000
R30(fp/s8) = 0x807f0000
>> g
>> g
>>addr: 80001938
timer start
elapsed time is 0.000s
>> g
>>addr: 8000195c
timer start
elapsed time is 40.266s
>> g
>>addr: 80001990
timer start
elapsed time is 20.133s
>> g
>>addr: 800019d8
timer start
elapsed time is 23.488s
>> g
>>addr: 80001a04
timer start
elapsed time is 36.911s
>>
```

3.2 图像显示

我们使用汇编语言自行编写了一个图像显示的小程序，在程序中，用户可以通过按钮来切换显示图片，实现类似PPT的效果。

首先将程序代码拷入用户代码区，再将三张图片的数据存入用户数据区。在Term上输入g指令运行程序，程序会先显示第一张图片，并循环等待按钮输入。当按钮0按下时，显示第二张图片。此时程序再次进入循环等待按钮1的输入。当按钮1按下时，显示第三章图片，程序退出。

图片显示效果如下：



4 实验总结

I. 选择合适的开发方式

我们发现许多16位的同学使用的开发模式是先设计支持所有功能的完整的数据通路再写代码。我们觉得这种思路并不十分有效率。尽早的代码实现可以尽早的显示出数据通路可能存在的问题，而且一些功能的实现基于现有的代码要比在纸面上设计要容易的多。所以可以选择先实现简单的数据通路，再在后期进行逐步修改的方式，当然这仍然需要对流水线数据流的认真分析和足够的单元测试。

II. 合理安排时间

因为32位挑战组的deadline较晚，所以我们组的进度有些拖后，导致后期在期末巨大压力下仍要完成计原实验，负担非常重，一些可以做的优化也没有完成。合理的时间安排和进度跟踪是必要的，在这里也向助教和老师建议加强进度的跟踪。

III. 认真研究时序图，寻找潜在问题

在实现SRAM读写的时候，我们的代码通过了仿真测试，但实际上板子时结果总是不对。我们花了很多时间在找bug，或者重写SRAM控制器。最终发现SRAM在写的时候出现了

时序问题：写使能打开时，数据线还未准备好，因此写入的数据是错误的。我们才意识到，硬件编程和软件编程不一样，即使逻辑上完全对，但是在时序上仍然可能会出问题，因此在编写代码前要认真研究硬件的时序图，保证各信号的时序关系正确。

IV. 使用调试工具

硬件调试比软件调试要难很多，因为编译很耗费时间，而且缺少合适的debug工具。在助教的指点下，我们使用的ila和vio逻辑分析仪，通过采样或单步观察信号变化的方法，明显提高了debug的效率。

本次实验工程量很大，需要面对很多困难，我们两个能坚持下来并成功跑通监控程序，还实现了VGA扩展，我们心里都很有成就感。在造CPU的过程中，我们学到了很多硬件编程和软件不一样的地方，学会了用硬件的角度思考问题，对流水线结构理解更深了，也学到了16位CPU组的同学学不到的东西（例如CP0寄存器和精确异常处理），收获很大。

最后感谢张宇翔助教能在我们遇到困难时不遗余力地帮助我们！