Y86 Simulator实验报告

15307130079 郭雨&15307130072 陈杨栋

**一、简述**

本次实验旨在研究如何流水线实现Y86指令。我们的分工方式是由郭雨实现程序内核的主要功能，由陈杨栋将内核嵌入UI界面并进行UI界面设计和编写。

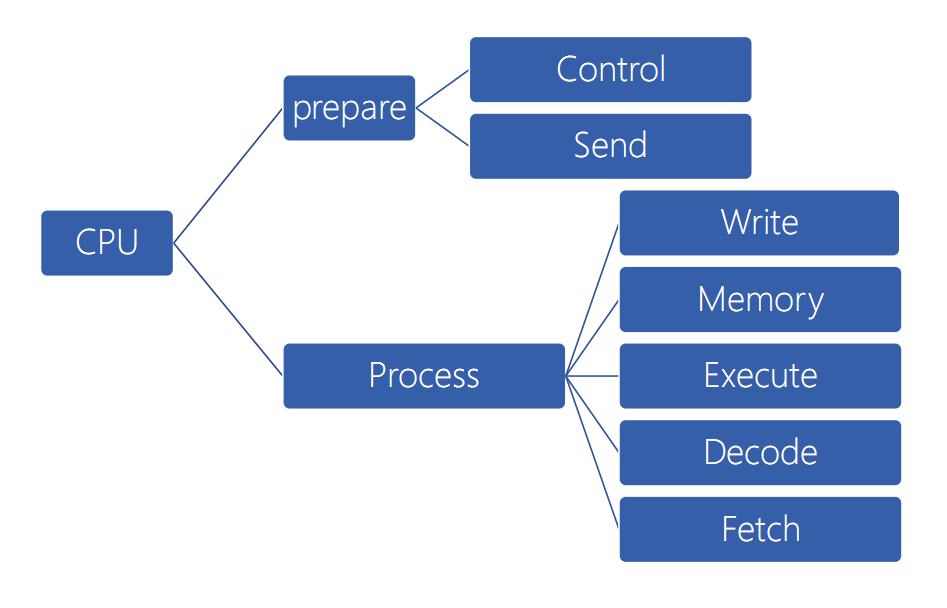
我们小组可以实现以下的Y86指令：

movl、cmov、addl、subl、addl、xorl、jmp、jle、jl、je、jge、jg、Call、pushl、popl、halt、ret、nop

**二、内核**

内核部分将课本中的HCL代码和Pipeline流水线图中的管道使用C++语言进行了实现。由于硬件逻辑代码具有一定的并行性，因此在代码顺序的排布上吃了一点苦头。

内核的结构比较普通，分为Prepare和Process两个部分，其中Prepare分为Control和Send，Process分为Write、Memory、Execute、Decode、Fetch。



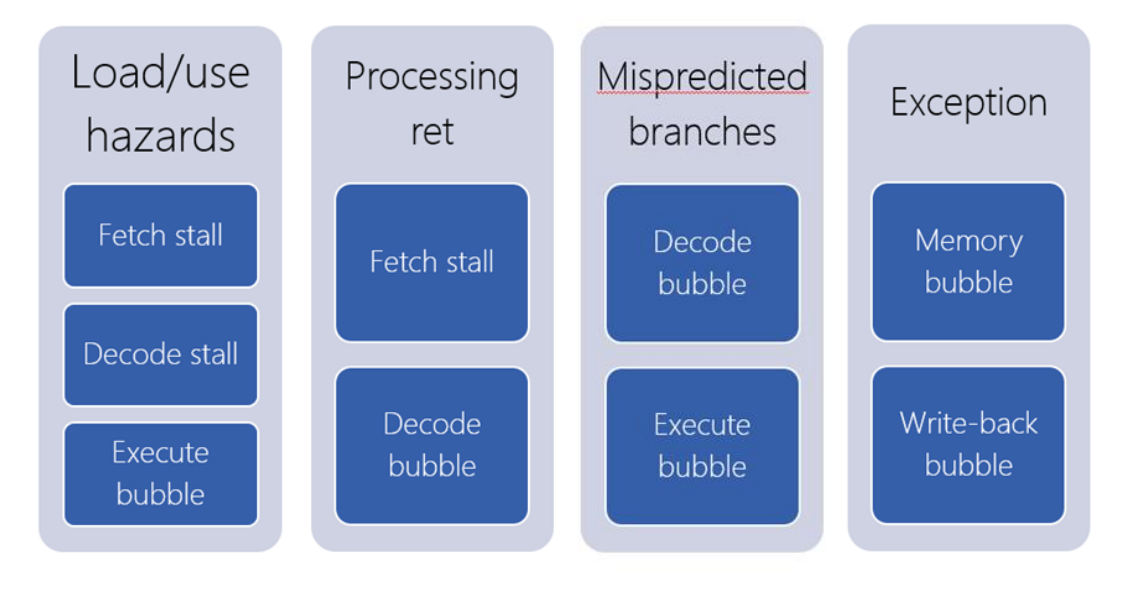
Control部分负责控制暂停和气泡信号，用于避免流水线冒险。流水线冒险可能出现的情况有四种，分别为：加载/使用冒险，ret语句、分支预测错误、程序异常。

加载/使用冒险的条件为(E\_icode == IMRMOVL || E\_icode == IPOPL) && (E\_dstM == d\_srcA || E\_dstM == d\_srcB)，应对方式是用bubble和stall机制，在Execute上插入一个气泡（nop），以一个延时的代价保证正确性。具体的措施是Fetch为stall，Decode为stall，Execute为bubble

ret语句的条件为D\_icode == IRET || E\_icode == IRET || M\_icode == IRET，由于书上没有提供较好的解决方法，只能采取插入三个气泡的方式。具体的措施是Fetch为stall，Decode为bubble。

分支预测错误的条件为E\_icode == IJXX && !e\_Cnd，由于两个错误的指令尚位于Fetch和Decode阶段，未造成修改，因此将错误指令直接变成气泡即可。处理方法是Decode为Bubble，Execute为Bubble，Fetch正常运行。

程序异常的条件即是状态码出现异常。我们需要把异常之后的指令的修改操作停止。由于修改只会出现在Memory、Write、Execute的条件码部分，只需要在这三个部分加上判断即可。



Send部分将数据传输到寄存器中，如果是stall，不修改寄存器的值，如果是bubble，无视输入值而将寄存器的值修改为bubble。

Process部分为五个流水线阶段，代码参考了书上的HCL代码，再加以拼接，这里没有太多亮点可以称述，具体实现可以阅读程序。

拼接过程中出现了一些优先级问题，如Control部分应当放在Send前，以Send作为时钟上升沿的标准，等等。

此外，代码里有一些if语句的条件，值得细细思考。例如，SelFwdA部分，选择传输到ValA的值，条件判断可能有多数满足，此时就会有一个优先级的问题，为了照顾顺序执行的原则，应当选择最靠近的阶段传输的值。然而同一阶段也会有多个值同时满足的情况，即同一个寄存器同时被内存取出的值和寄存器中读出的值修改。可以想到，这种情况是之前讨论过的“popl %esp”，此时我们需要做的是将栈顶的值赋给%esp，而不是将%esp减4。因此需要优先取valM，即内存中取出的值。

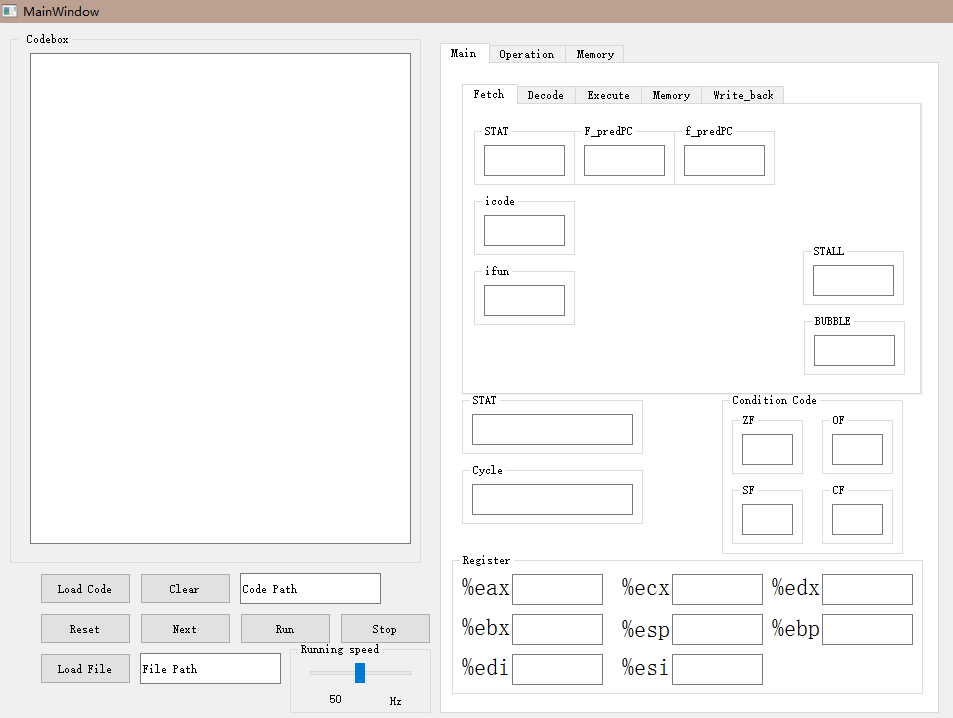
更宏观的，流水线的五个阶段，在硬件实现时应当是并行执行，但C++代码中我们没有选择并行，而是采取了WMEDF的反方向顺序执行。选择这么做的原因是为了照顾数据转发。由于现在的转发管道只有从靠后的阶段向靠前的阶段的方向，因此选择倒序执行，可以轻松满足转发逻辑。

此外，为了调试和展示的方便，我们额外输出了每一个阶段进行了什么微操作。这在UI中有所体现。

**三、UI**

为了使流水线一步一步更加简洁地呈现在使用者面前，我们自学并采用采用Qt工具库和开发软件编写了相应的UI。在初步学会使用Qt之后，我们面临的问题是如何将写好的内核程序和UI对接，我们采取的方案是将内核的所有功能以及相关数据封装到一个类并写在头文件里面（CPU.H）,由于Qt本身也支持c++语言，这样就只需要解决输入数据和输出数据的传输就可以了，而程序内核的工作可以和平时写的C++程序相同的跑起来而不必担心兼容性的问题。

下面介绍一下UI界面的使用方式



3

2

1

如图，有三个区域，1号区域是显示代码的文本框，2号区域是功能按键区域，3号区域是监视区域，大页签分为主监视区（主要显示各个变量以及个阶段状态的变化）、操作监视（主要显示每个阶段程序具体会进行什么操作）、内存监视区（包括程序使用的内存和程序自己保存位置的内存内存储的内容），在主监视区和操作监视区分为5个小页签区分pipeline 的5个阶段，对应显示每个阶段的对应情况，下面介绍按键功能。

Load Code功能是将右侧Code Path框内的文件路径读进来并写入1号区域，便于读入之后可以同步观察程序代码和pipeline流程，Clear则是对应的清空文本框。

Reset是重置所有状态，包括程序运行状态、pipeline状态和各阶段以及寄存器数值，全部置为未开始的状态。

Next是让指令运行下一步，Run是按照Running speed的频率正常执行程序，stop则是让程序停止执行，当程序顺序执行完毕后会有相应的弹窗提示。

Load File是二进制文件接口，对应旁边的File Path的路径的文件，加载之后程序就会准备就绪，可以点击重置键，然后根据需要选择单步执行或者直接执行。

**四、程序特色：**

一个比较友好、易于使用的界面

使用c++ map作为内存的存储容器，使程序可以支持较大范围的内存操作

使用Qt编写UI，完全兼容c++，使得内核和UI的对接十分方便。

在代码编写中，内核和ui的编写分开，都有着较为强大的适应性，内核可以单独工作，ui也可以直接通过其他版本的内核提供的接口迅速更换执行方式。同样，由于内核可以单独编译调试，大大降低了程序debug的难度，不必再使用较为复杂的Qt creator完成内核部分代码的调试。

核心使用c++编写使程序的运行效率更高。

提供了范围较广的运行频率调整功能，虽然没有执行到光标的功能，但也可以通过这个来迅速到达想要执行的指令位置，节约等待时间。

程序运行过程可以随时终止并且重置。

提供了加载汇编代码的功能，在监视流水线工作的同时可以在同一窗口下看到正在执行的y86程序段（如果有的话），而不需要再额外打开一个窗口。

Qt对c++拥有完全的兼容性以及良好的跨平台性能，经测试在windows和ubuntu环境下均可正常运行并实现全部功能，而不需要对代码进行修改，只需重新编译即可。

对绝大部分的符合语法要求的y86指令都能完整运行

使用Qt的页签功能对各个流程和各项监视参数进行了分割，可能没有直接在流水线图中那么直观，但保证了ui界面的条理性，使各项参数可以清晰的展示出来

**五、需要改进的地方：**

对超过10个Y86代码进行了测试，对所有发生的问题都进行了修正，但还不能称为测试了大量程序，无法100%保证内核的稳定性和内核与ui的完美对接。

Ui排布可能存在一定不合理之处。

没有引入cache

对于y86代码的载入存在一些格式问题，同时没有加入行号和高亮功能

没有对死循环的检验，考虑是否通过对cycle数的限制来终止死循环

没有加入对小端法和大端法存储的同时支持，默认使用了小端法

在ui的设计中，由于对于布局缺乏整体把控，后来还临时加上了一些东西，导致了排布不合理。ui只能说清晰，但谈不上美观。

由于Qt是速成的，并没有掌握其中一些精髓功能的使用，代码能用STL就用STL，可能因此忽略了Qt一些功能强大的工具库，无形中加大了编程复杂度，甚至因此放弃了一些本能实现的功能

原本添加了将binary code转化成assembly code并显示的功能，后来因为无法读出数组，某些特殊构造的代码无法直接翻译而放弃了，改为直接加载进 assembly code

使用了拖拽式编程，而不是使用纯代码设置，故窗口的适应性存在较大缺陷

**六、实验感想：**

整个过程中，我们为了能顺利做出正确的显示，仔细琢磨了书中对于pipeline的解释说明，加深了对第四章内容的理解与记忆。最开始的时候并没有想到自己可以做出这样的UI界面，但是通过找资料自行学习和软件自带的帮助文档，我们还是成功的做出来一个可以使用且界面相对友好的软件环境，认识到了其实有些事情真正着手去做之后其实没有那么难，只要找对了方法，一步一步来总是可以实现的。自己摸索的过程还是非常有趣的。