 Y86 v2.0项目实验报告

—— 进程、线程与网络编程

何占魁 15307130175 何林洋 15307130240

1. 项目简介：
   1. 项目概览

Y86 v2.0项目旨在实现对上学期ICS课程项目**“基于Y86指令集的流水线CPU——Y86 v1.0”**的技术升级。一则对上学期的代码进行重构，优化代码风格；二则运用网络编程与并发编程知识，强化Y86内核，将其分拆为客户端与服务端，并使其多任务处理能力。

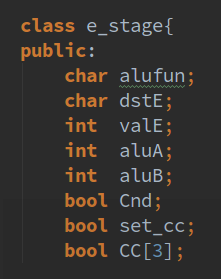
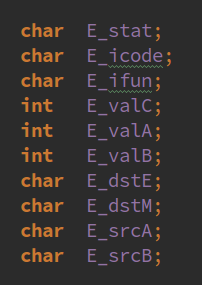
* 1. 基本开发信息
     1. 开发人员：何占魁、何林洋
     2. 开发平台：Ubuntu 16.04
     3. 开发语言：C++ 98，Qt 5.7
     4. 开发工具：CLion 2016.02, QtCreator 4.0.2, Git

1. 项目实现：
   1. 内核与UI代码重构

最初设想是基于已有代码，添加并发与网络功能。但事实上发现这份设想非常难实现，原因如下：

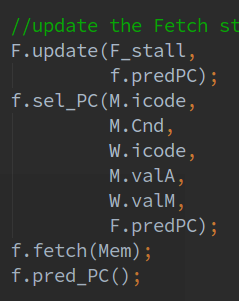
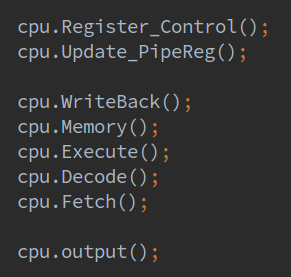
* + - 1. 代码可读性不佳
         1. 上学期对Y86理解不深，内核屡屡修改，层次凌乱
         2. Qt现学现用，框架和代码风格缺乏规范
      2. 代码可拓展性不佳
         1. 未作代码升级打算
         2. 实现思路与并发编程天然相悖

可拓展性差表现在代码与并发性思路的特点天然相悖，比如对于变量传递，v1.0对于流水线的每一个阶段设置一个定义该阶段所有变量的类，这使得变量之间的关系十分清楚，但对于希望变量共享的线程级并发来说，不同流水线阶段的变量传递麻烦，徒增编程的复杂性。

图一 v1.0变量定义示例 图二 v2.0变量定义示例

代码可拓展性差也表现在五阶段流水线运行比较混乱，再者此处流水线寄存器(F D E M W)的变量更新和流水线内部变量流动(f d e m w)混在一起，对于进程级并发来说会导致致命错误，这一点在下一节中提到。

图三 v1.0流水线更新示例 图四 v2.0流水线更新示例

* + - 1. 程序框架发生重大变化：
         1. v1.0只有本地运行程序，内核与UI均写在Qt程序中，代码相互纠缠。
         2. v2.0 分为基于原生C++的服务器与基于Qt的UI，服务器用以链接和运行CGI程序，UI接受动态内容并显示。
         3. 其框架变化如下：



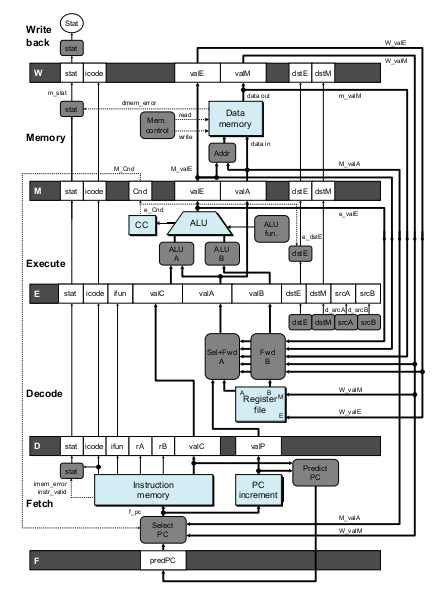
图五 v1.0和v2.0框架对比

* 1. 流水线内部线程级并发

因为代码重构阶段已经将线程级并发的准备工作准备充分，所以此处只需要对线程安全进行分析，然后写出相应并发例程。通过Debug再进行相应改进即可。

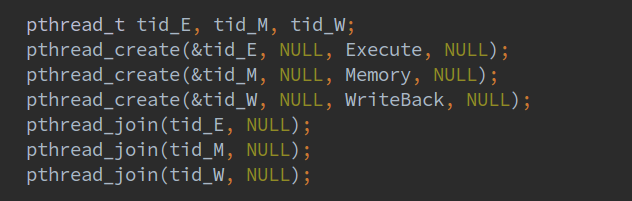
* + 1. 分析线程安全

根据Y86模拟器流水线图，可知Decode阶段和Fetch阶段有流水线寄存器的变量转发，因此这两个阶段的执行次序必须在Write、Memory和Execute之后，所以将这两个阶段提出来，不参与线程并发。



图六 Y86流水线示意图

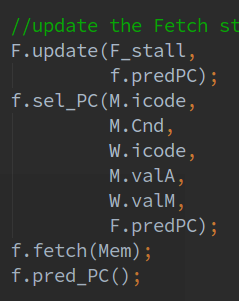
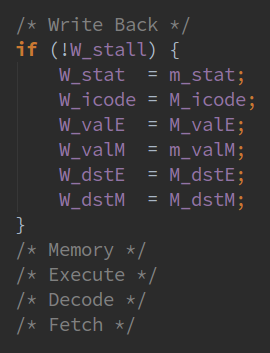
可以确定参与流水线并发的是Write、Memory和Execute 阶段，三者没有需要共同修改的变量，因此是线程安全的，不需要加互斥锁。



图七 线程并发代码节选

* + 1. 流水线寄存器更新

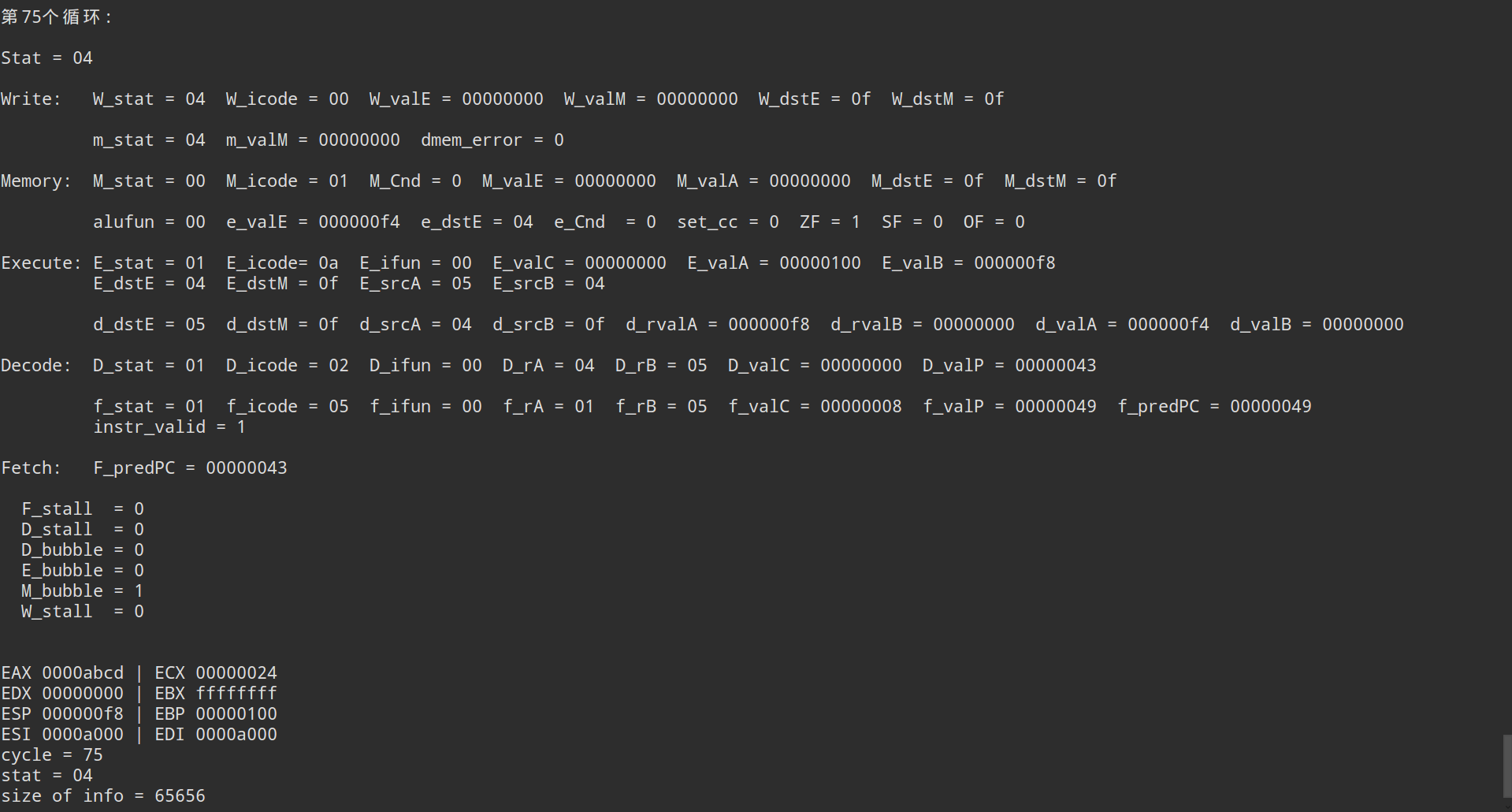
v1.0流水线寄存器的更新是和该流水线阶段更新先后进行，这在线程并发中会导致可能这个阶段变量的该周期初始值超前更新。因此，v2.0采用流水线寄存器集中更新机制，将其统一在一个函数中，在所有流水线阶段更新完毕后，再集中更新。

图八 v1.0的分散更新机制 图九 v2.0的集中更新机制

这里需要注意的一个关键是：集中更新必须按照W M E D F依次进行，否则会导致值传递直接从Fetch阶段直接“击穿”Write阶段

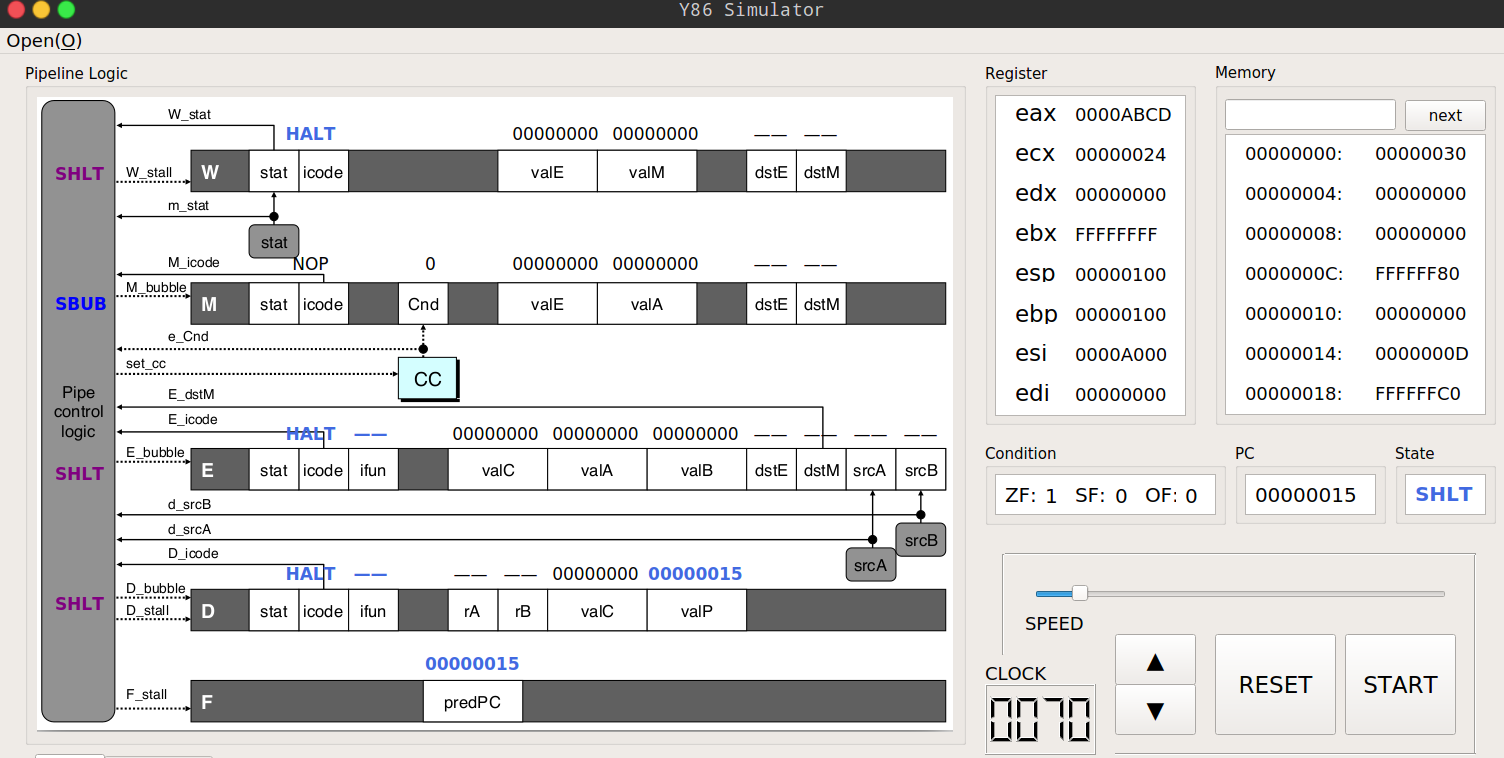
* 1. 服务器进程级并发
     1. 服务端运行命令行界面：



图十 服务端命令行界面示意图

服务端显示每个周期的**CPU状态、流水线寄存器值、流水线阶段值、流水线逻辑值、通用寄存器值**，以及与网络传输相关的**周期传输字节数。**

* + 1. 客户端界面

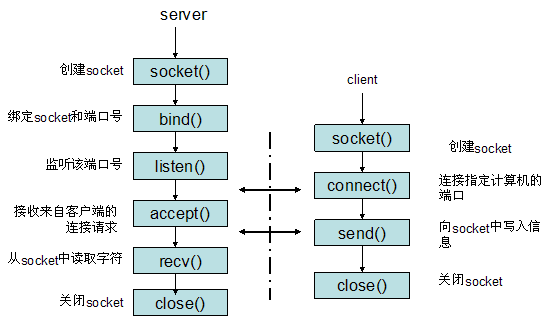


图十一 客户端界面示意图

客户端显示更加直观，显示服务器端的**除**流水线阶段值和周期传输字节数**之外的所有服务端参数**，并可通过按钮进行**自动运行、按步运行，**并支持**重置**和**回退**操作。

客户端的UI设计继承了上学期的UI外壳，内部完全重构。重构机制为“数据映射”。即用队列/链表的方式（这里是数组实现的队列），存储结构化的各周期参数，这里用数组实现队列，是考虑到可能存在的死循环和有限的存储空间的矛盾，对接受数据设置上限。也因此，可以轻松实现回退操作。

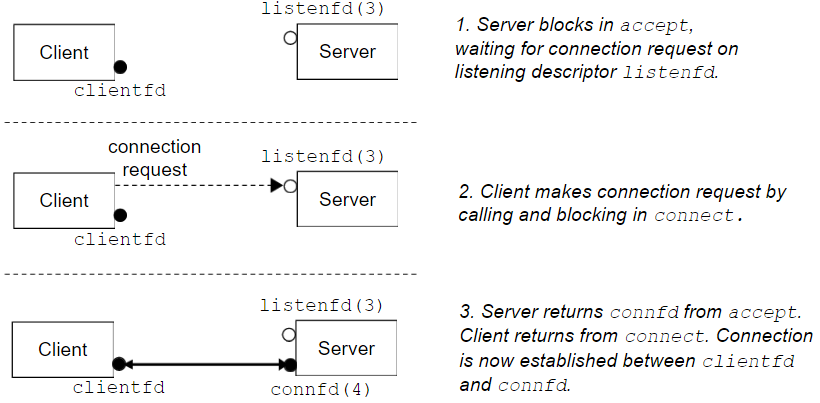
数据传输即用Socket接口，基于TCP协议进行数据传输，具体实现方法为传统的Socket编程方式，流程如下：



图十二 socket网络连接流程示意图

服务器端实现进程级别并发，可同时处理多个客户端的代码文件并实现并发执行和传输。这里选择进程级别并发，是考虑到fork()函数对数据空间的拷贝以及进程之间的变量不共享性，正好为不同请求的分头处理提供条件。

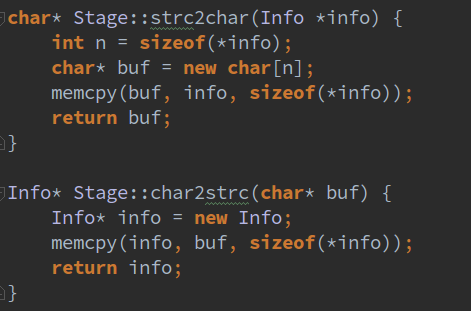
实现方式是利用fork()等函数和文件描述符操作连接不同的connfd和clientfd。



图十三 多进程并发流程示意图

* 1. 数据传输机制
     1. 字符数组与结构体转换

因为RIO读写包只提供字符数组的传输，而我们所传输的参数数量庞大，如果按照字符数组编码将非常复杂。于是在这里实现一种结构体和字符数组的互换函数。



图十三 字符数组和结构体的互换函数

实现思路为：利用字符数组和结构体底层实现的相似性，将字符数组/结构体的首指针提供给结构体/字符数组即可，比较巧妙地完成了参数编码和解析工作。

* + 1. 一个有趣的Y86数据传输机制

思路为“数据回传与显示异步”，其限制条件是**显示周期不会超前于传输周期**，其余时刻两者完全独立。这样的做法较为轻松地解决了**网络延时导致的控制周期不同步**问题。也使得UI控制逻辑和跳转功能的实现变得简单清晰。

对于这种机制的演示不方便用图片展示，可以看第三节提供的演示视频，视频中基于这种机制展示了一种有趣的“周期赛跑”现象。

但其存在的不足是这种方式的**非普适性**，该机制的设计前提是Y86的代码文件一经传输则不会被修改，因此可以做到异步显示，但未从技术上正面解决网络延时对控制的负面影响。



图十四 数据传输机制示意图

1. 项目演示：
2. 项目感想：
   1. 未实现工作

在这次开发中有许多点子，也有很多技术想要尝试，但由于时间紧迫而未能实现，略表遗憾，列在这里，希望有时间的时候回来看看，说不定一时冲动就把它完善了

* + 1. 安卓端开发

预想用安卓端和PC端的同时发送请求，展示服务器的多任务处理能力，但安卓端草草地写了界面之后，发现JAVA和C++后台通过socket传输数据不是一件简单的事情，如果写成HTTP协议传输json又太耗费时间，JAVA不熟练的我们煎熬了一个晚上没有做出来，这是最遗憾的一点。

* + 1. 服务端完善

希望最后再实现一下客户端对服务端的暂停操作，以及客户端是直接使用汇编器解析后传送字节码到服务器，希望实现传输代码文件，在服务端解析的功能。以及，在线程并发的内核中加入预线程化技术。

* 1. 我们的最后几句话

开发这个项目有很大的成就感，尤其是看到服务器和客户端开始互相传输数据的那一刻，非常开心。发现代码风格和技术是一个逐渐进步的过程，看到上一次的代码和这一次的变化，能感受到我们进步了，思路也更清晰了。相信我们以后会有更大的进步！也感谢这次有趣的项目任务！

何占魁 15307130175

何林洋 15307130240