Lab Report on Lab 4

the Design of Y86 Pipeline Processor

16307130194 陈中钰

16 级 计算机科学技术学院

Contents

1 总体说明 2

2 界面实现及使用方法 4

3 总体设计 8

4 PC 选择和取指阶段 11

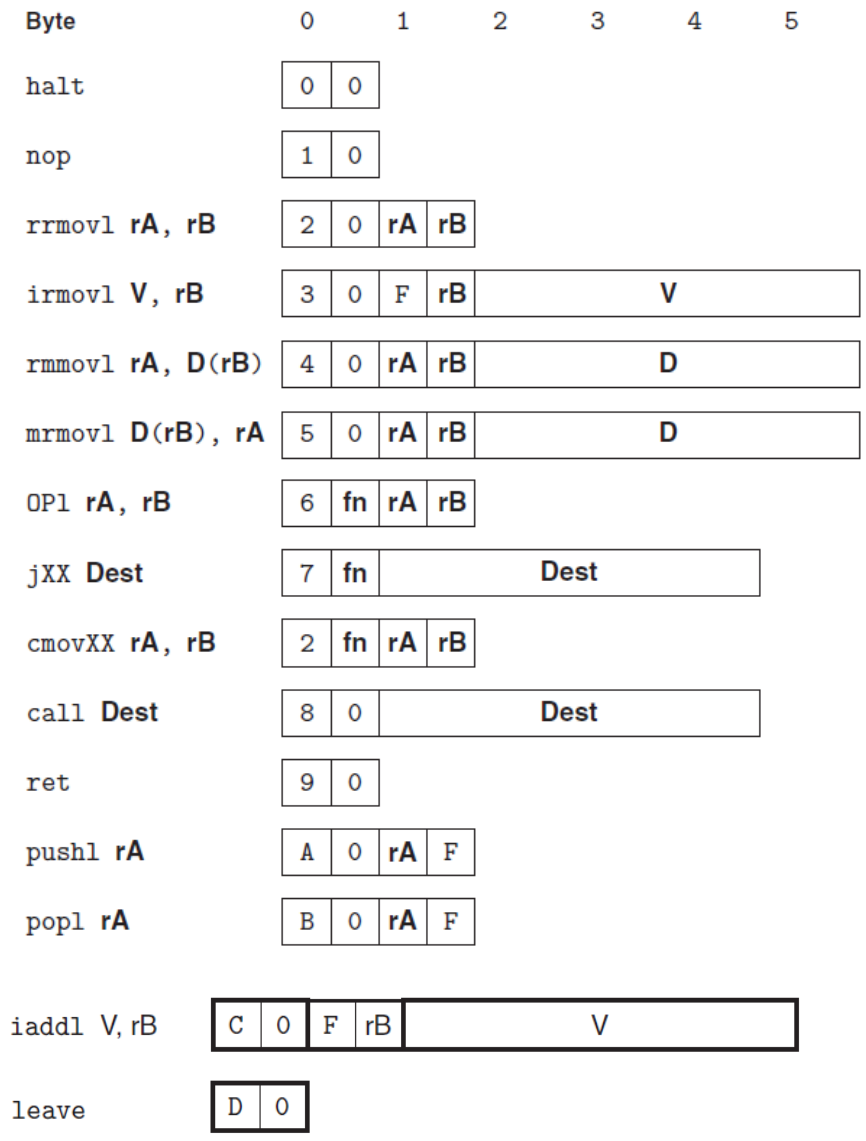
5 译码阶段 13

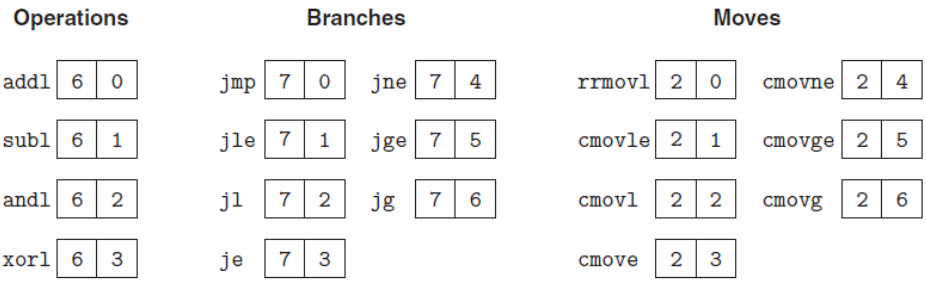
6 执行阶段 14

7 访存阶段和写回阶段 15

8 实验感想 15

1 总体说明

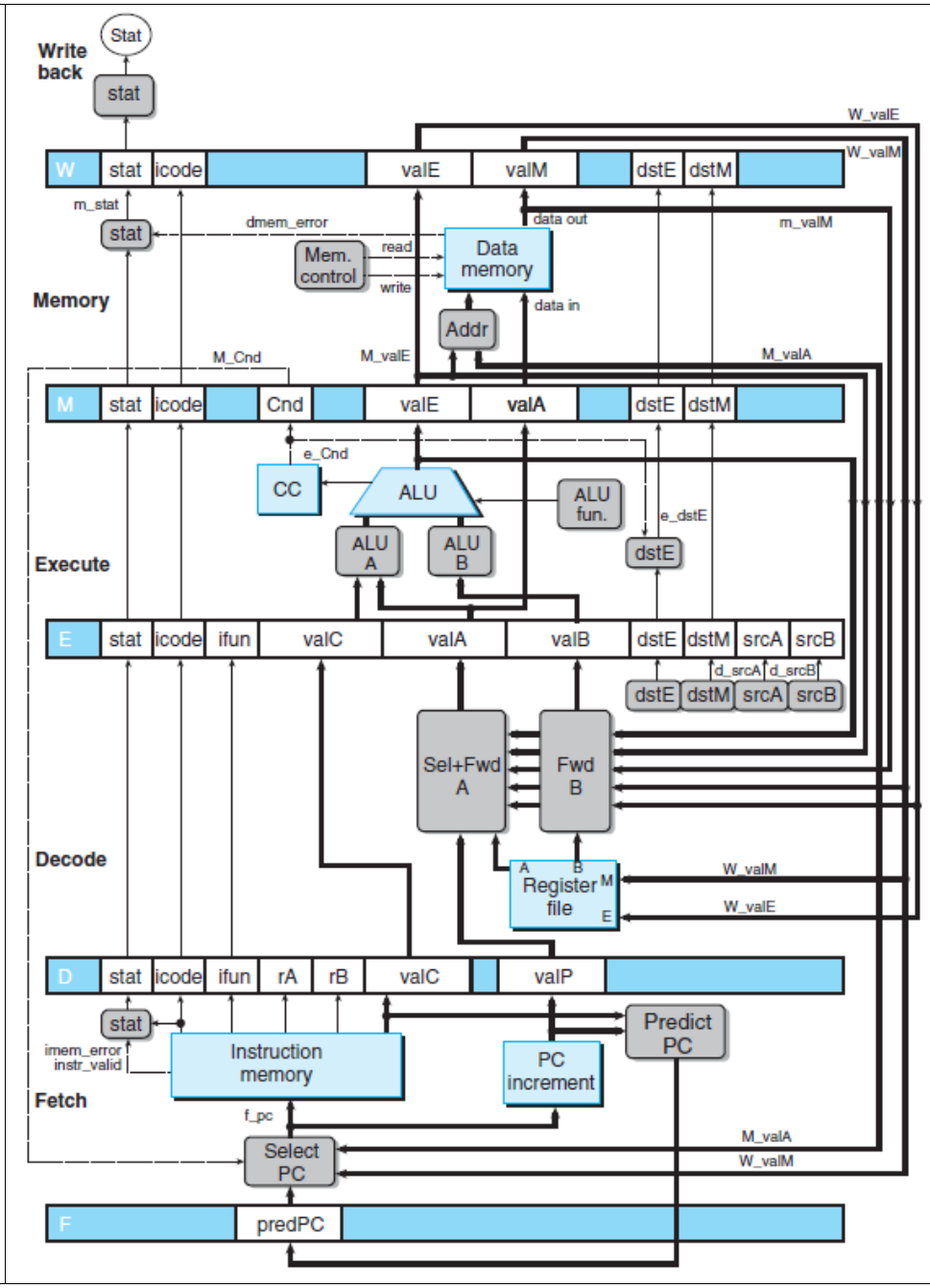
项目	内容
名称	Y86 Pipeline Processor
类型	PIPE
指令集	



寄存器文件								
	0	%eax	4	%esp	8	%e8	C	%e12
	1	%ecx	5	%ebp	9	%e9	D	%e13
	2	%edx	6	%esi	A	%e10	E	%e14

	3	%ebx	7	%edi	B	%e11	F	ENONE
字节序	Little Endian							
位宽	32 bits(4 B)							
读入格式	.yo 文件、二进制指令文件、十六进制指令文件							
结构	Von Neumann（程序指令存储器和数据存储器合并在一起的存储器结构）							
内存	134, 217, 728 B(0x5000000 B)							
储存方式	用 int 型数组存储指令和数据，数组的每个元素储存二进制数据中连续4bit 组成的十进制值							
语言	C, C++							
IDE	Microsoft Visual Studio Community 2015							
系统	Microsoft Windows 10 Enterprise(64 bit system, x64 processor)							
UI	Win32 Console Application							
资料	《深入理解计算机系统基础第3版》机械工业出版社，即 CSAPP							
	pipe-full.hcl 硬件控制语言文件							
	http://csapp.cs.cmu.edu/网站上的测试程序							
多文件	功能			文件				
	常用常量宏定义					macro.h		
	结构体定义（用于全局变量定义）					struct.h		
	全局变量的定义			global_variable.cpp		global_variable.h		
	五个阶段（取指阶段、译码阶段、执行阶段、访存阶段、写回阶段）的时序电路、组合电路操作函数			F_stage.cpp D_stage.cpp E_stage.cpp M_stage.cpp W_stage.cpp		F_stage.h D_stage.h E_stage.h M_stage.h W_stage.h		
	各类转换函数（如：int 型转 string 型，icode 名称读取函数等，用于制表）			convert.cpp		convert.h		
	主函数（运行 Y86处理器）			main.cpp				
	读取指令（二进制、十六进制指令）			input.cpp		input.h		
	制表（输出程序员可见状态）			output.cpp		output.h		
	UI 模拟器			simulator.cpp		simulator.h		
	流水线操作函数（处理器初始化、时钟上升沿函数、时序更新函数等）			pipeline.cpp		pipeline.h		
代码量	约2500行							

硬件结构



2 界面实现和使用方法

2.1 使用方法

把待运行的汇编代码转换成二进制或十六进制表示（程序运行时要选择对应的输入模式），允许有空格或其他非0~F 的字符存在（程序会自动忽略此类非法字符），大小写均可，保存在文本文件中。在 Visual Studio 编译运行 Y86_Pipeline 文件夹中的程序，或者直接运行 Y86_Pipeline.exe 可执行文件，按照程序输出的提示，输入操作对应的代码，最后手动输入代码地址路径，或者拖动代码文件到框内（会自动输入文件地址），点击运行即可运行程序。

2.2 注意事项

still mode 静态模式的输出方式下，只能处理1份指令，运行结束后需要关闭程序重新启动；rolling mode 滚动模式的输出方式可以处理多份指令
由于表格过长，静态模式的输出方式下可能会看不到整个表格，可以右击 dos 的窗口栏，在“属性“中把字体调小，或者把缓冲区高度调的足够大即可
待运行的代码文件需要放在纯英文、无空格路径中，否则可能无法正常运行
寄存器文件中有0x0~0xe 共14个寄存器，0xf 代表的才是无寄存器 RNONE
注意输入模式的选择和输入文件要对应，即二进制输入模式要对应输入二进制文件，否则无法正常运行
.yo 文件读入模式下，由于此 Y86的架构于 CMU 官网上的版本稍有不同，因此可能无法正常运行官网上的.yo 文件，但是所有的.yo 文件都已经根据这个版本修正过并提交了，可以直接使用所提交的.yo 文件
所有的.yo 文件都已经同时提交了，我验证过了，都是对的，所以真的如果不幸出了 bug，肯定不是 Y86本身的问题，所以还希望麻烦助教多试几遍~
速度设置范围是0~8000，由慢到块，小于4000的速度都是很慢的，建议设置8000的速度
请注意按照上文中的十六进制指令类型和格式、寄存器文件设置来设置测试代码

2.3 总体说明

项目	内容
主要文件	simulator.cpp, simulator.h,
功能	支持多文件读入，可连续处理多个指令文件
	可选择二进制、十六进制指令文件读取模式
	可选择输出保存到根目录下的 output.txt 文件
	可调节运行速度
	每个 cycle 能输出一个整齐的表格，包括所设置的参数、程序员可见状态、改变过的内存、组合电路操作等
	可选择滚动输出模式或静态输出模式（类似 Qt 界面，框架、参数名称在屏幕上不动，只有数据在跳动）
	多颜色文字

2.4 实现方法

2.4.1 彩色提示语

使用<Windows.h>头文件中的 SetConsoleTextAttribute()函数，设置将要输出的文本的字体颜色、粗细、背景颜色，并用 cout 输出提示语的 string。

颜色设置语句宏定义	`#define CYAN SetConsoleTextAttribute (GetStdHandle(STD_OUTPUT_HANDLE), FOREGROUND_INTENSITY | FOREGROUND_GREEN | FOREGROUND_BLUE)`
输出彩色语句	`BLUE; cout << "Y86 Pipeline@Zhongyu Chen" << endl; cout << "Serial number 16307130194" << endl; GREEN; cout << "Starting..." << endl;`

2.4.2 逻辑控制

通过输出提示语句，指示使用者输入操作代码，以进行程序运行参数的设置、运行程序等操作。

输出提示语及操作代码	`YELLOW;`
	`cout << "Choose output mode(default as 0)" << endl;`
	`cout << "0.rolling mode" << endl;`
	`cout << "1.still mode" << endl;`

输入操作代码	`WHITE;` `cin >> a;`

设置参数	`switch (a)`
	`{`
	`case 0:output = ROL; break;`
	`case 1:output = STI; break;`
	`default:output = 0;`
	`}`

根据参数运行程序	`if (output == STI)` `{`

2.4.3 打印表格

通过转换函数把数据值转换为 string 型，把诸如 reg、icode 的标号转换为对应的名字的 string，便于把若干个 string 连接成行，并在适当的位置添加框线、补上恰当的空格，并把每一行储存到 string 型数组中，最后输出。

对 string 进行处理，把表格每一行练成一个 string 储存到 string 数组中，能便于输出，但是却不能对 string 中特定名词设置颜色，因此有点单调。

在新的一行接上左框线及第一个参数名称、值，补上恰当的空格
接上第二个参数并补上空格
接上第三个参数并补上空格，加上右框线
index 后移，准备处理下一行
在各个阶段执行过程中会出现操作，把它们保存在 vector<string>的全局容器变量中，在适当的位置把对应阶段的操作添加到表格中去

2.4.4 输出到文件

定义输出文件流 fout，遍历表格 string 数组，输出到 fout。

2.4.5 滚动输出

遍历存有表格的 string 数组，逐行输出	`for (i = 0; i < p; i++)`
	`    cout << s[i] << endl;`
	`cout << endl << endl << endl;`

2.4.6 静态输出

添加<Windows.h>头文件，把光标设置在原点(0, 0)，创建一个缓冲区的手柄，遍历存有表格的 string 数组，把表格一行行地写入缓冲区中，最后再激活、显示缓冲区内容，使得整个表格同时出现，并且在不同的 cycle 之间，相同的变量会出现在固定的位置。由于人眼的视觉暂留，会发现表格框线、变量名称都是“静止”的，只有它们对应的值在跳动而已，产生类似 Qt 界面的效果。

变量定义	
设置光标位置	
把表格写入缓冲区中	
激活缓冲区的显示	

```
COORD coord;
DWORD bytes;
HANDLE hOutput = CreateConsoleScreenBuffer(
    GENERIC_WRITE,FILE_SHARE_WRITE,NULL,
    CONSOLE_TEXTMODE_BUFFER,NULL);
```

```
coord.X = 0;
coord.Y = 0;
bytes = 0;
for (i = 0; i < p; i++)
{
    coord.Y = i;
    WriteConsoleOutputCharacterA(hOutput,
    s[i].c_str(), s[i].size(), coord, &bytes);
}
SetConsoleActiveScreenBuffer(hOutput);
```

2.4.7 程序运行效果

```
Y86 Pipeline@Zhongyu Chen
Serial number 16307130194
Starting...
Features
1.Storing: Little Endian
2.Architecture: Von Neumann
3.Width: 32 bits
4.ROM: 134,217,728 B
5.Instructions: halt, nop, rrmovl, irmovl, rmmovl, mrmovl, cmovxx, opl, jx
Successful!
Do you want to exit or continue?(default as 0)
0.continue
1.exit
0
Continuing...
Please follow the instructions and input the number of the operation!
Choose input mode(default as 0)
0.input binary file
1.input hexadecimal file
0
Choose output mode(default as 0)
0.rolling mode
1.still mode(this mode can only run program once)
0
Input speed of running programs(0~8000, slow to fast)
8000
Do you want to save as output.txt?(default as 0)
0.view only
1.save as well
1
Input the route of the file
C:\Users\ECHOES\Desktop\lab4\test_cases\submitted\abs-asum-cmov.txt
Successful!
Type in any char to run

-------------------------CYCLE 1--------------------------
|parameter-----------------------------------------------|
|cycle  1          CPI    ∞          input  binary file  |
|speed  8000       output rolling    save   enable       |
|sequential update---------------------------------------|
|                                                        |
|                                                        |
|                                                        |
|register file-------------------------------------------|
|%eax   0          %ecx   0                              |
|%edx   0          %ebx   0                              |
|%esp   0          %ebp   0                              |
|%esi   0          %edi   0                              |
|CC------------------------------------------------------|
|ZF     0          SF     0          OF     0           |
```

2.4.8 程序表格效果

```
-------------------------CYCLE 7-------------------------
|parameter-----------------------------------------------
|cycle  7          CPI    1.00000000  input  .yo file
|speed  8000       output rolling     save   disable
|sequential update---------------------------------------
|M4[0xfc] <- 17
|R[%ebp] <- 256
|
|
|register file-------------------------------------------
|%eax   0          %ecx   0
|%edx   0          %ebx   0
|%esp   256        %ebp   256
|%esi   0          %edi   0
|CC------------------------------------------------------
|ZF     0          SF     0          OF     0
|Stat   SAOK-------------------------------------------
|W register----------------------------------------------
|bubble 0          stall  0
|stat   SAOK       valE   252        valM   0
|icode  ICALL      dstE   %esp       dstM   ENONE
|M register----------------------------------------------
|
|bubble 0          stall  0
|stat   SAOK       valE   248        valA   256
|icode  IPUSHL     dstE   %esp       dstM   ENONE
|Cnd    1
|E register----------------------------------------------
|valE <- 0 + 248 = 248
|bubble 0          stall  0
|stat   SAOK       dstE   %ebp       dstM   ENONE
|icode  IRRMOVL    valA   248        valB   0
|ifun   FJMP       srcA   %esp       srcB   ENONE
|valC   36
|D register----------------------------------------------
|valA <- m.valM = 0
|valB <- m.valM = 0
|bubble 0          stall  0
|stat   SAOK       valC   4          valP   0x2e
|icode  IIRMOVL    rA     ENONE      rB     %eax
|ifun   FNONE
|F register----------------------------------------------
|f.pc <- F.predPC = 0x2e
|icode:ifun <- M1[0x2e] = a:0 = IPUSHL:FNONE
|rA:rB <- M1[0x2f] = 0:f = %eax:ENONE
|valP <- 0x2e + 2 = 0x30
|
|bubble 0          stall  0
|f.pc   0x2e       predPC 0x2e
|changed memory------------------------------------------
|address           decimal           hexadecimal
|0xfc              17                0x11
-------------------------------------------------------
```

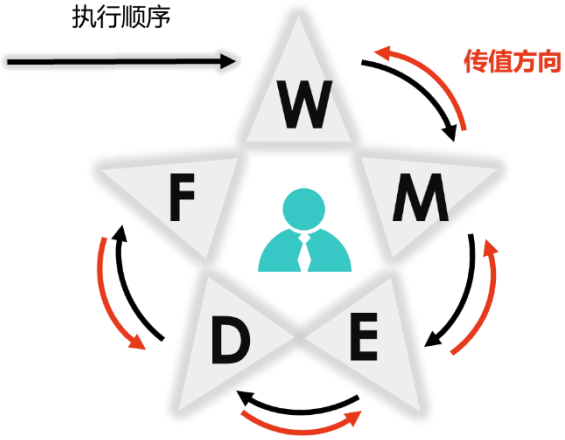
2.4.9 程序表格优点

实时显示 CPI 值、指令在时序更新以及每个周期中进行的操作、变化过内存、所有程序员可见状态的显示、运行模式的设置参数，总体上较整洁美观。

3 总体设计

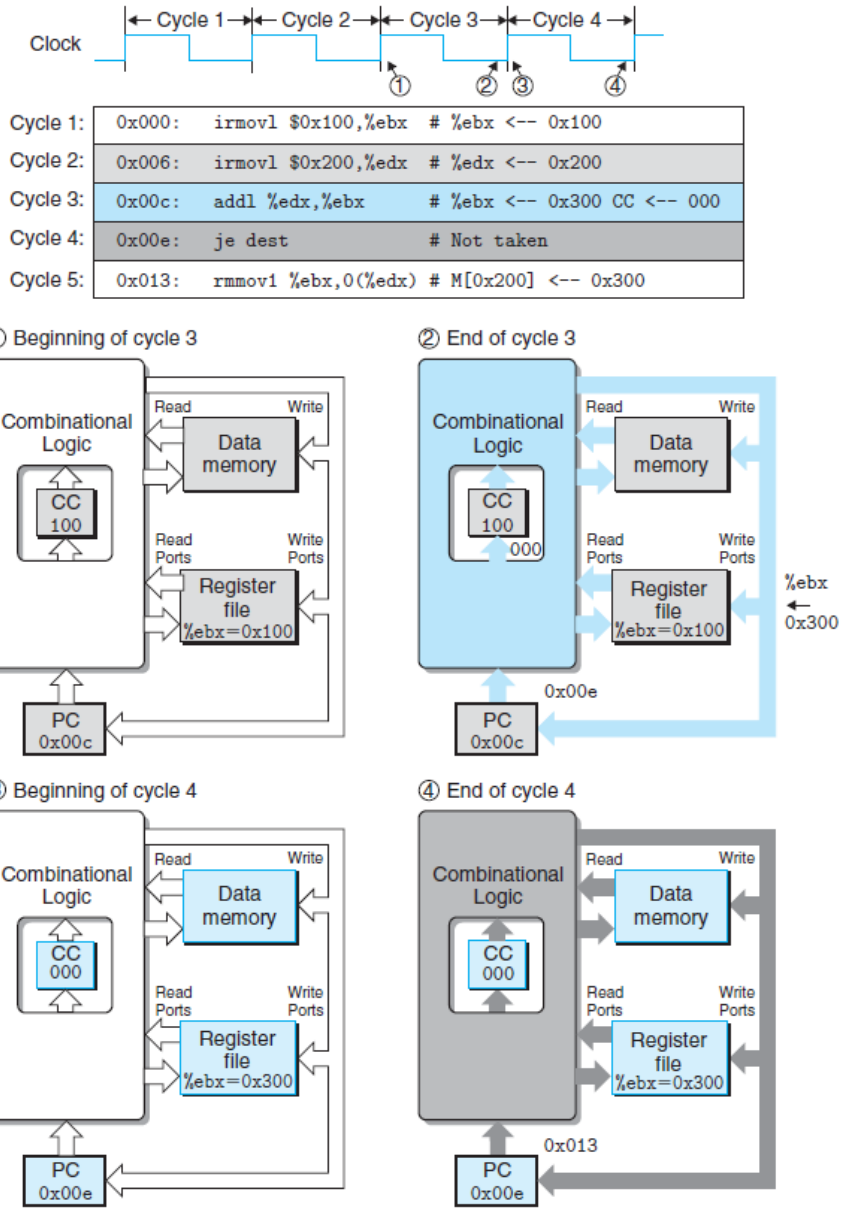
3.1 整体时序性

因为 C/C++语言是不具有时序性的，因此如何在使用 C/C++语言编程的情况下，保持时序性是最关键的部分。在流水线中，要达到每一条指令都往下一个阶段移动的效果，如果采取五个阶段的正向执行，那么前一阶段的结果会覆盖掉后一阶段要用到的数据，不可行；而如果采取反向执行，并且从 W stage 开始执行，这样既不会覆盖掉有用的数据，还能保持每条指令都进入到下一个 stage 中的特性，即能完美实现整体的时序性。



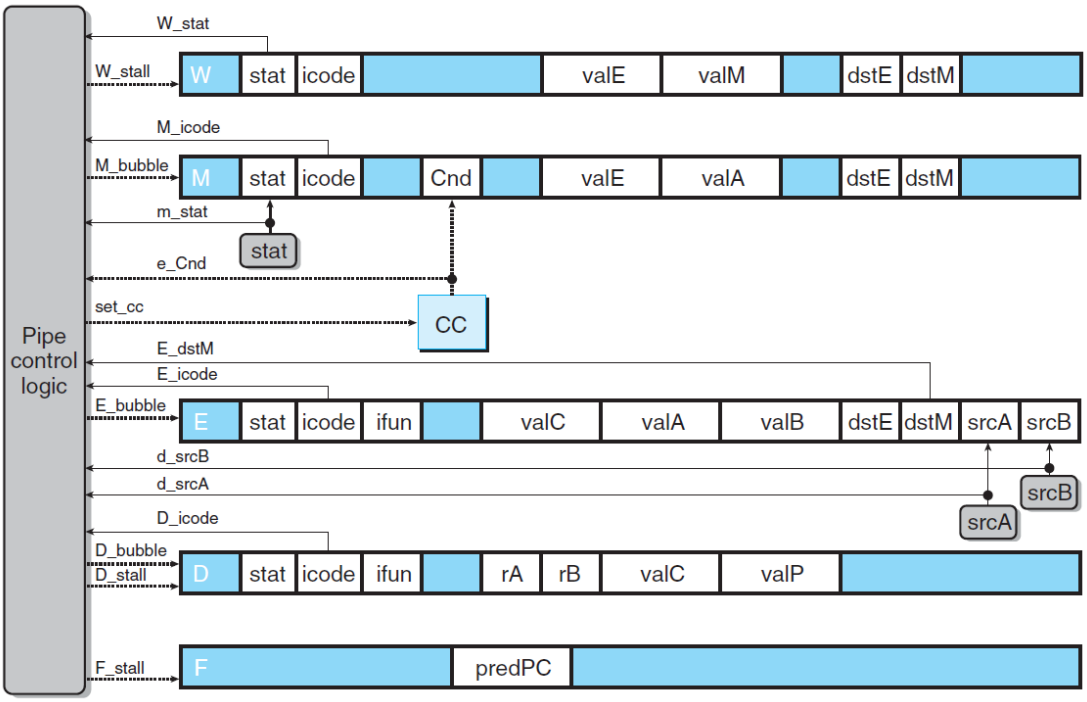
3.2 程序员可见状态时序性

除了每个阶段的流水线寄存器以外，还有条形码寄存器、寄存器文件、内存是具有时序性的。由于时序性，在每个时钟上升沿到来时，每个阶段都要首先更新流水线寄存器，然后再根据寄存器中新的值来运行并更新组合电路中的值；而另外三者的更新则需要在时钟上升沿到来时最先更新，因为在它们门口等着进去的值都是上一个 cycle 算好的值，因此要最先更新，否则会被这个 cycle 算出来的值覆盖掉。而按照书中的设计，它们之间的顺序是：先根据 Set CC 判断是否要更新 CC，接着根据 W.dstE、W.dstM 判断是否要更新寄存器文件，最后根据 mem.write 判断是否要写入内存。



3.3 控制逻辑

和 SEQ+不同的是，PIPE 在不同阶段同时运行5条指令，也就是上一条指令还未处理完，就要处理后面的几条指令了。因此 PIPE 的设计需要实现数据冒险、异常处理、加载冒险等异常情况的处理。所以还需要添加一个机制，能够检测出上述的异常，并且根据情况 stall 特定的阶段或者 bubble 特定的阶段，来消除异常。各个阶段的 stall、bubble 信号的更新都封装在 void control update();中，该部分可以在五个阶段的前面或者后面执行，均不影响程序的正常运行。



3.4 五个阶段

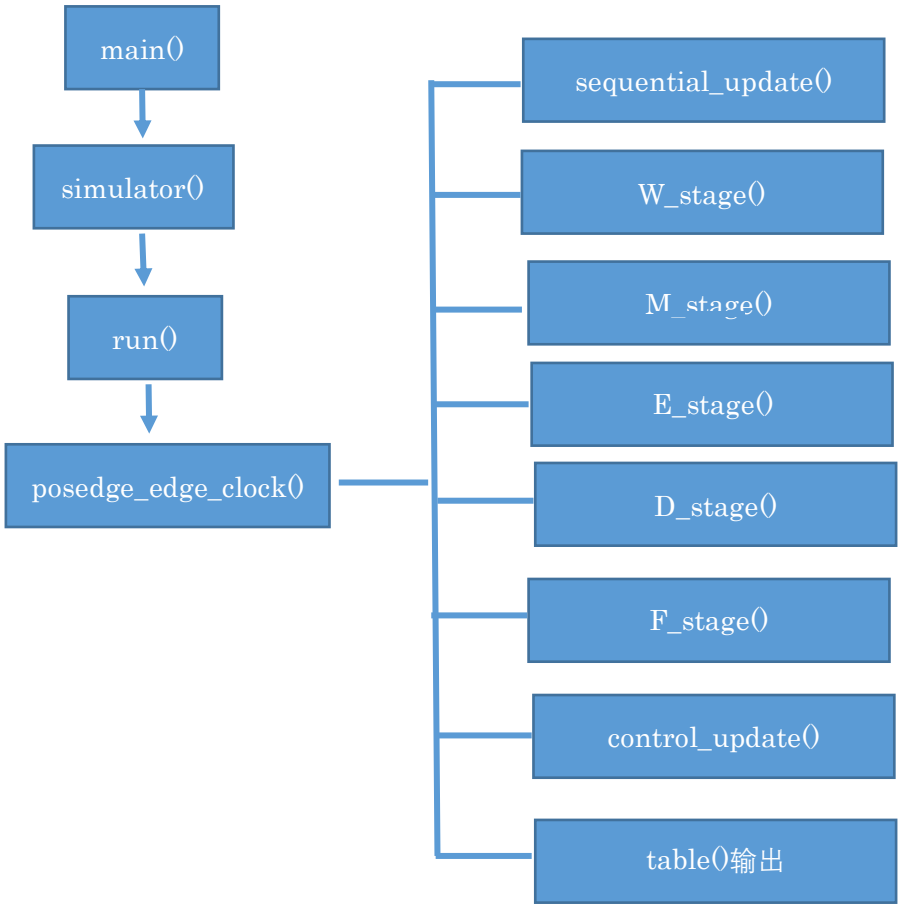
如上文所述，采取反向执行以保持时序性、流水线性，也就是先执行 W stage，最后执行 F stage。在执行某个阶段的时候，需要先更新流水线寄存器，再根据流水线寄存器中新的值来运算并更新该阶段的组合电路的值。这样子就能保持 Y86的时序性。

3.5 CPI

$$\mathrm{CPI} = \frac{C_i + C_b}{C_i} = 1.0 + \frac{C_b}{C_i}$$

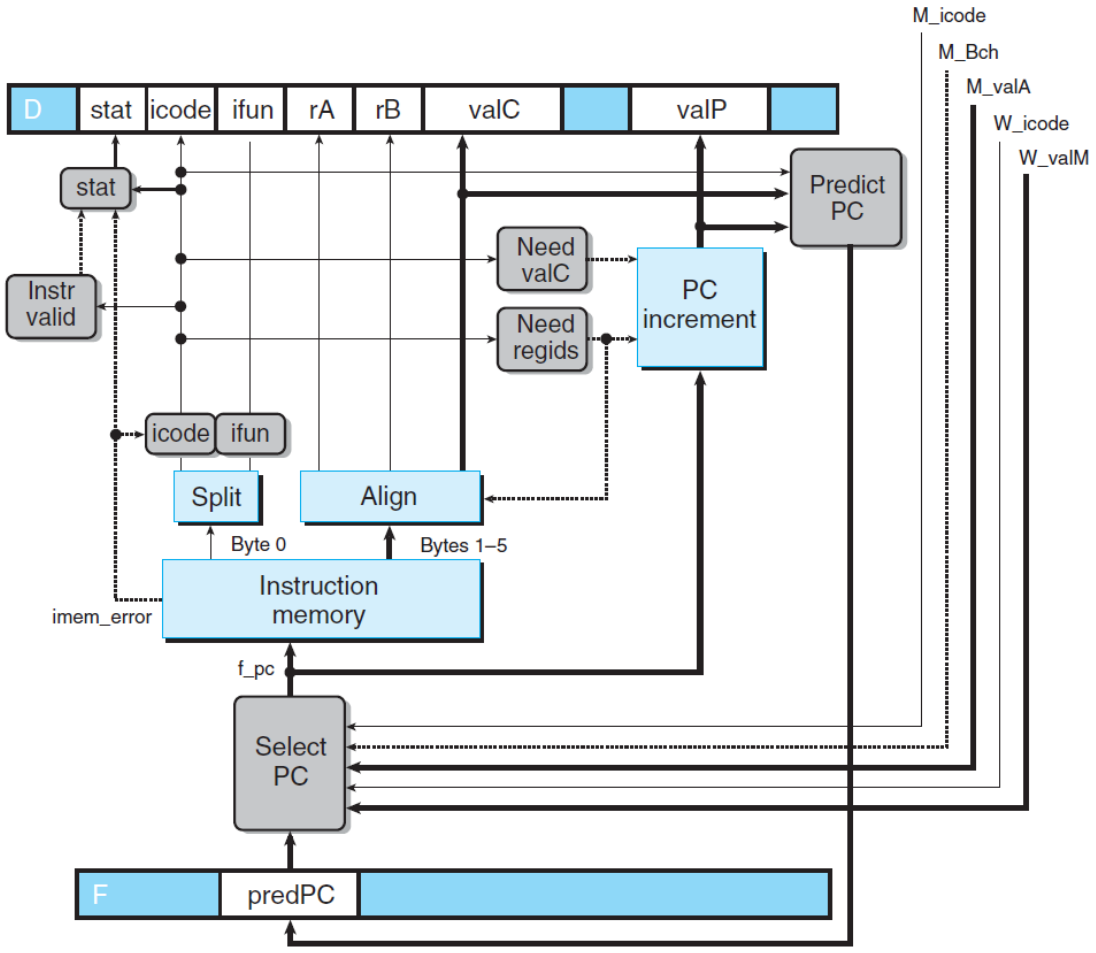
CPI 的计算采用上述公式。在每个周期实时显示当前周期运行结束后 CPI 的值。

3.6 Y86 PIPE 运行结构



4 PC 选择和取值阶段

4.1 硬件结构



4.2 变量定义

时序电路变量	定义在结构体 F 中，如 F.predPC
组合电路变量	定义在结构体 f 中，如 f.pc，f.stat 等

4.3 具体实现

总体上按照从下往上的顺序执行，先进行时序电路更新，再进行组合电路更新。以下按照执行顺序来阐述各部分的具体实现。

4.3.1 时序电路更新

F Stage 不可能被 bubble，而 stall 的时候 F 寄存器中的 predPC 不变；当没有 bubble、stall 的时候，predPC 被 PredictPC 更新。

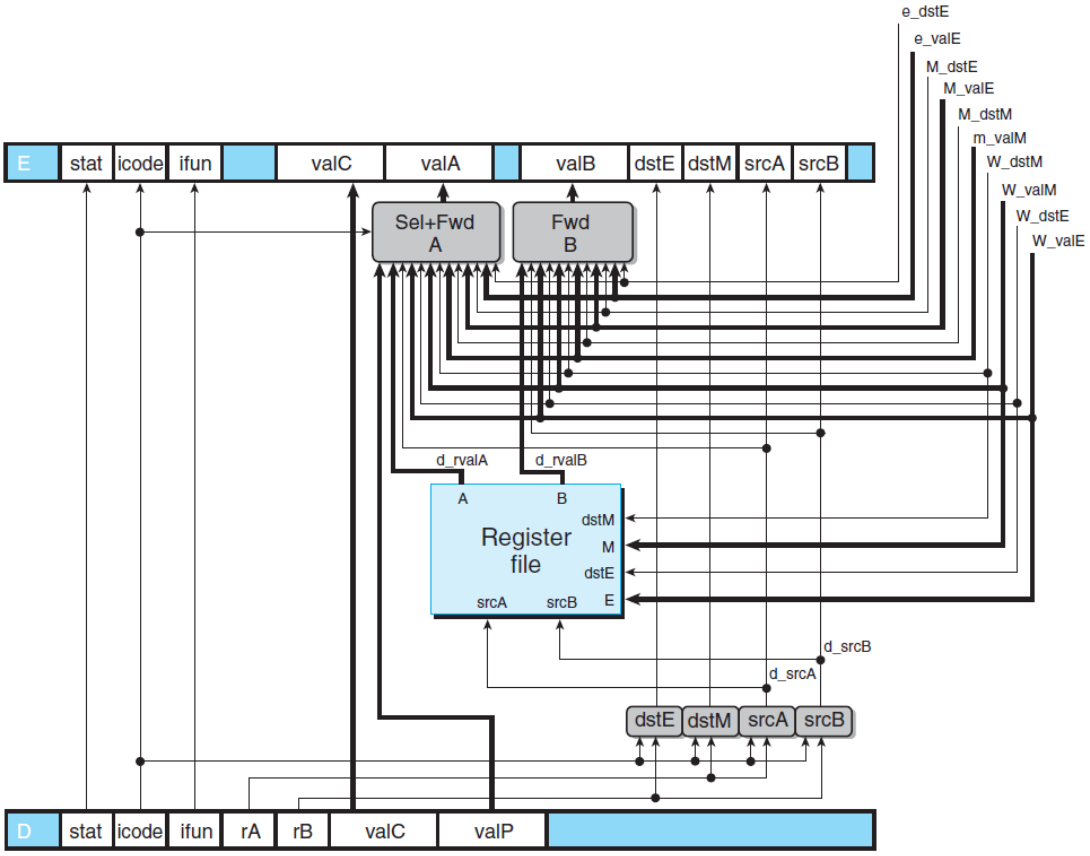
4.3.2 组合电路更新

SelecrPC 计算出 f.pc	1. 预测错误分支进入 M Stage，需要从 M.valA 读出该指令的 valP，运行不跳转的下一条指令
	2. ret 进入 W Stage，需要从 W.valM 读出返回地址，从返回处继续运行
	3. 其他情况取 F.predPC 的预测值，也就是运行下一条指令
Split	1. 从内存的 f.pc 处读出指令的 icode，ifun
	2. 检验是否越界
	3. 如果越界，icode、ifun 置为 INOP、FNONE
Instr	判断指令是否合法

valid	
Stat	对应设置 f.stat，如果内存越界，设为 SADR，非法指令设为 SINS，halt 设为 SHLT，其他正常设为 SAOK
Need rigids	当 f.icode 为 movl、push、pop、OPl 时，还需要从内存中读取 rA、rB，否则 rA、rB 设为空
Need valC	当 f.icode 为 irmovl、rmmovl、mrmovl、jxx、call 时，还需要从内存读取 valC
PC 增加	根据 Need rigids、Need valC 判断当前指令长度，从而获得下一条置零位置 valP
Predict PC	若为 jxx 或 call，预测下一条指令在 valC 处，否则下一条指令为 valP

5 译码阶段

5.1 硬件结构



5.2 变量定义

时序电路变量	流水线寄存器变量定义在结构体 D 中，如 D.stat，D.icode 等
	寄存器文件定义在 int reg[]中下标0x0~0xe 均对应寄存器，0xf 代表无寄存器
组合电路变量	定义在结构体 d 中，如 d.stat，d.valA 等

5.3 具体实现

总体上按照从下往上的顺序执行，先进行时序电路更新，再进行组合电路更新。以下按照执行顺序来阐述各部分的具体实现。

5.3.1 时序电路更新

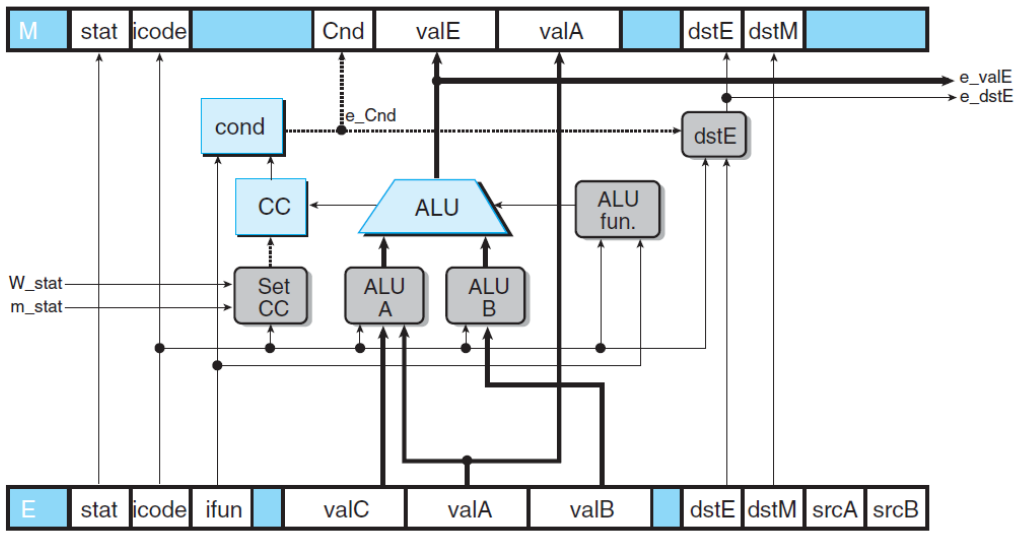
bubble	D.stat 更新为 SBUB，其他位置清0或者设置为无
stall	D 流水线寄存器中的值不变，使得 D 阶段重复上一个 cycle 中的操作，得到相同的结果，使得该指令“停住了“
其他情况	正常，D 流水线寄存器更新为从 F Stage 传过来的值

5.3.2 组合电路更新

直接传递	stat、icode、ifun、valC 可以直接传递到 E Stage
srcA	1. 若 icode 为 rrmovl、rmmovl、opl、push，则取 rA 的值
	2. 若 icode 为 pop、ret，则对象寄存器为%esp
	3. 其他情况下不需要寄存器
srcB	1. 若 icode 为 opl、rmmovl、mrmovl，则取 rB 的值
	2. 若 icode 为 push、pop、ret，则对象寄存器为%esp
	3. 其他情况下不需要寄存器
dstE	1. 如果 icode 为 rrmovl、irmovl、opl，则目标对象为 rB
	2. 若为 push、pop、call，则目标对象为%esp
	3. 其他情况不需要目标对象
dstM	1. 当 icode 为 mrmovl、pop 时，目标对象为 rA
	2. 其他情况下不需要目标对象
Sel+FwdA	根据情况判断是否要转发、转发的对象
Fwd B	根据情况判断是否要转发、转发的对象

6 执行阶段

6.1 硬件结构



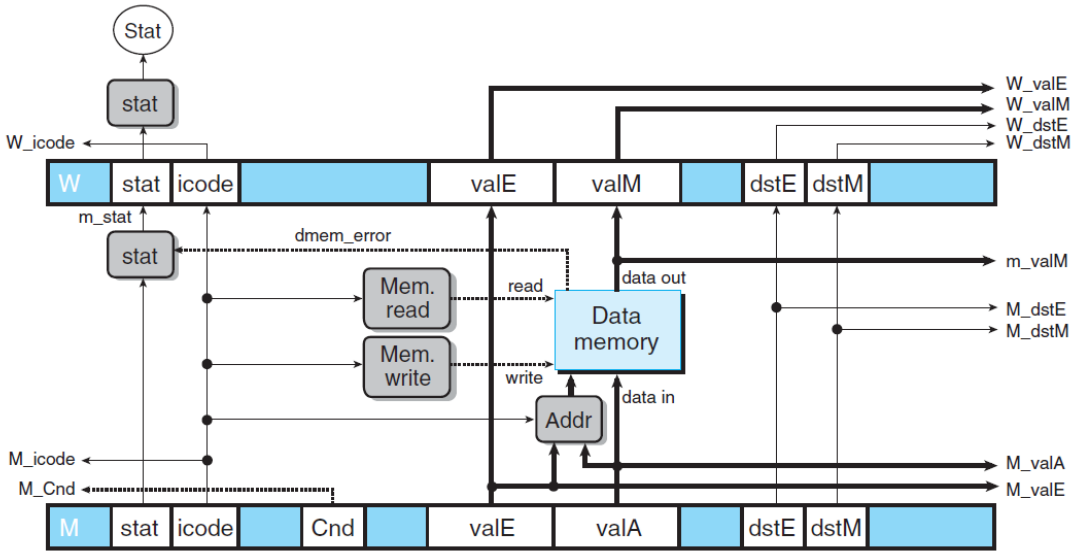
6.2 具体实现

更新流水线寄存器

aluA，aluB，alufun 的取值，并计算 set_cc
alu 运算，并根据 CC 的值获得 e.cnd
根据情况选择 dstE

7 访存阶段和写回阶段

7.1 硬件结构



7.2 具体实现

更新 W stage 的流水线寄存器，跟新 Stat
更新 M stage 的流水线寄存器
判断是否要读写 memory，选择出 address
如果要读 memory，则读取 memory
更新 m.stat

8 实验感想

8.1 经过了1个月的坚持，终于完成 Y86的设计以及界面的设计。在这个过程中，我对 Y86 PIPE 的设计和结构有了极其深厚的了解，掌握到了 PIPE 设计的精髓和特点。

8.2 第一次独立完成了约2500行、逻辑控制复杂的程序，我的能力得到了很大的锻炼和提升，有很强的成就感。

8.3 这次实验的完成效率过低，希望在未来能更多地锻炼自己的代码能力，提高写代码的效率。

8.4 看见过同学们多种多样炫酷的 UI，觉得自己的 dos 界面还是太弱了。在将来一定要更多地去探索这些未知领域，做出一些以前未曾做过的炫酷的事情！