实验二（选题一）说明文档

一 实验介绍

1 实验信息

本次实验名为“Tomasulo 算法实现与可视化”，实验内容分为三大部分，第一部分需要完成一个编译器；第二部分是实现各种微架构单元，主要包括 Load/Store 缓冲区，整数加法器、整数乘法器，还有各自的保留站，在此基础上实现一个虚拟机。第三部分是需要实现一个可视化的界面，展示流水线 CPU 的状态变化。最终实验评估包括实验报告和展示两部分。

本次实验的建议使用桌面版操作系统，由于实现的是模拟器，没有指令架构的要求。

高级语言采用 C/C++。

实验采取线上提交实验报告结合线下展示的模式。

文档接下来的章节包括：实验目的，背景知识，实验内容以及评分标准。

2 实验目的

本次实验属于创新实验，在课程内容的基础上有所提高。如果同学们找到代码和文档中的纰漏和错误之处，欢迎指出，并可视程度获得一定的加分。实验的代码量较少，但需要同学们花费一定的课外时间阅读文档和代码以及查阅一些额外的资料。实验中涉及到的流水线 CPU，Tomasulo 算法以及保留栈，本文档将会给出概要性的介绍。

本次实验的主要目的是让同学们：

- 深入了解和掌握高级流水线的知识，例如动态调度、动态分支预测、乱序执行等概念。
- 了解现代处理器针对标量处理器指令多发的预测优化

二 背景知识

本节将会概要地介绍标量处理器，流水线处理器，记分牌技术和 Tomasulo 算法。

1 标量处理机

标量数据表示和标量指令系统的处理机被称为标量处理机。它是一种最通用，也是使用最普遍的处理机。

而为了提高标量处理机的性能，也就是为了提高指令执行的速度，通常回采取以下几种途径:

1. 提高处理机的主频，事实上在科技发展和商业竞争的驱动下，处理机的主频每隔 2~3 年(现

在甚至更短)就要提高一个数量级了。

2. 采用更好的算法和设计更好的功能部件，例如采用精简指令系统计算机(Reduction Instruction Set Computer 简称 RISC)技术来减少执行指令的平均周期数，用多位的乘除法缩短乘除法的执行时间，设计新型的 ROM 运算部件等等。
3. 采用指令并行技术，就是多条指令并行地执行，有三种方法可以实现，第一是采用流水线技术(pipeline)，这种处理机称为流水线处理机或超流水线处理机(superpopelining)；第二是采用独立的功能部件来分担不同的操作和运算，诸如浮点加、乘除法、分支操作等等；第三是超长指令字技术(very long instruction word,简称 VLIW)。

而在这个实验里面，我们关心的是指令并行技术中的流水线技术的实现和改进。

2 流水线处理机

流水线是一种实现技术，可以从两个方面来开发处理机的并行性：

1. 空间并行性：设置多个独立的操作部件。如：多操作部件处理机、超标量处理机
2. 时间并行性：采用流水线技术。不增加或只增加少量硬件就能使运算速度提高几倍，如：流水线处理机、超流水线处理机。

流水线技术的一大问题就是冲突(Hazards)，其原因有三种相关指令而不能直接利用流水线来处理。这三种冲突是读——写冲突(read and write, RAW)、写——读冲突(write and read, WAR)、写——写冲突(write and write, WAW)。

3 记分牌技术（scoreboard）

记分牌技术是为了解决上述的 RAW/WAR/WAW 问题而提出来的，主要原理是把一个指令的执行分成了四部分,译码并检查、读操作(读取操作数)、执行指令、回写操作。记分牌分几个显示状态(status)：指令状态(Instruction status)、功能单元状态(Functional unit status)、寄存器结果状态(Register result status)。

4 Tomasulo 调度技术

Tomasulo 一个指令只有三个部分：Issue、Complete 和 Result。

Tomasulo 和 Scoreboard 技术的区别表

区别	Tomasulo	Scoreboard
发射方式	顺序发射	顺序发射
发射条件	没有结构冲突	没有结构冲突而且没有 WAW 冲突

取操作数的方式	假如操作数没有准备好，则监听公共总线，从总线上取得操作数，有 forwarding	等待直到所有操作数的寄存器可用，就是必须是寄存器都写回了，无 forwarding
取操作数	无	有
回写条件	写入公共总线，由操作部件，寄存器，后写缓冲站的源地址来确定接收总线上写回的数据	等待，直到没有 RAW/WAR 相关

三 实验内容

实验的主要内容是完成附带的 template.c 中的[TODO]内容，并且自行加入可视化部分，这部分在模板文件中没有给出，同学们可以根据自身的平台和习惯进行编写。

1 输入

虚拟机的输入文件是机器码文件，需要支持的指令集为：

表 1 汇编指令子集

指令	类型	操作码 (opcode)	功能码 (func)	功能
LW r1,r2,i	I	35		读内存(r1 = MEM[r2+i])
SW r1,r2,i	I	43		写内存(MEM[r2+i] = r1)
ADD r1,r2,r3	R	0	32	加(r1 = r2+r3)
ADDI r1,r2,i	I	8		立即数加(r1 = r2 + i)
SUB r1,r2,r3	R	0	34	减(r1 = r2 – r3)
AND r1,r2,r3	R	0	36	与(r1 = r2 & r3)
ANDI r1,r2,i	I	12		立即数与(r1 = r2 & i)
BEQZ r1,i	I	4		条件跳转(jump to *pc+1+i* if r1==0)
J addr	J	2		无条件跳转(jump to *pc+1+addr*)
HALT	J	1		停机
NOOP	J	3		空指令

这些指令组成了一个很小的汇编子集。在实验附带的模板中 template.c 是按照机器码来处理所有指令逻辑的，所以需要同学们对汇编代码做一个简单的翻译，无需真正实现一个编译器，具体实现内容请参考 template.c。

2 指令格式

所有的指令都是 32 位字长，格式如下：

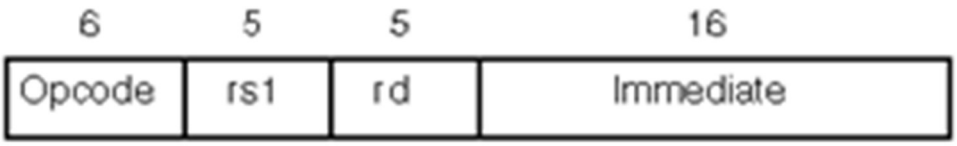

图 1 I 型指令格式

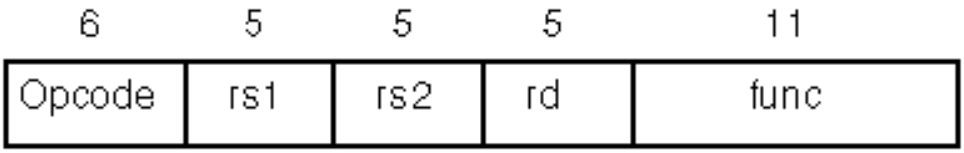

图 2 R 型指令格式

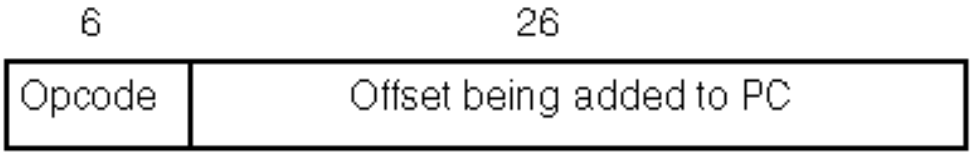

图 3 J 型指令格式

3 编写自己的汇编程序

除了实验提供的三个样例输入以外，你还可以使用上述汇编指令编写自己的程序，经汇编器翻译后在虚拟机上运行。

下面是一段求 Fibonacci 数列的程序：

```
addi r9,r9,1     ;set r9=1
addi r1,r1,10    ;set r1=10, r1 is counter
loop add r10,r8,r9   ;r10 = r8+r9
addi r8,r9,0     ;r8 = r9
addi r9,r10,0    ;r9 = r10
sw r10,r2,0      ;store r10, use r2 as offset
addi r2,r2,1     ;offset ++
addi r1,r1,-1    ;counter --
beqz r1,end      ;loop
j loop
end halt
```

3.1 书写规范

每行可以由如下部分组成：

- 可选的标号，例如上例中的 loop 和 end
- 指令名称
- 多个指令字段：寄存器、立即数，或标号
- 注释，由分号开始，到行末结束

3.2 如何得到机器码文件

你需要自行实现将汇编指令转换成机器码文件的汇编器。

例如，对上面的汇编程序样例，翻译之后生成的机器码文件的内容如下：

556335105

539033610

17387552

556269568

558432256

-1404436480

541196289

539099135

270532609

201326584

67108864

4 各组件功能.

4.1 整体结构

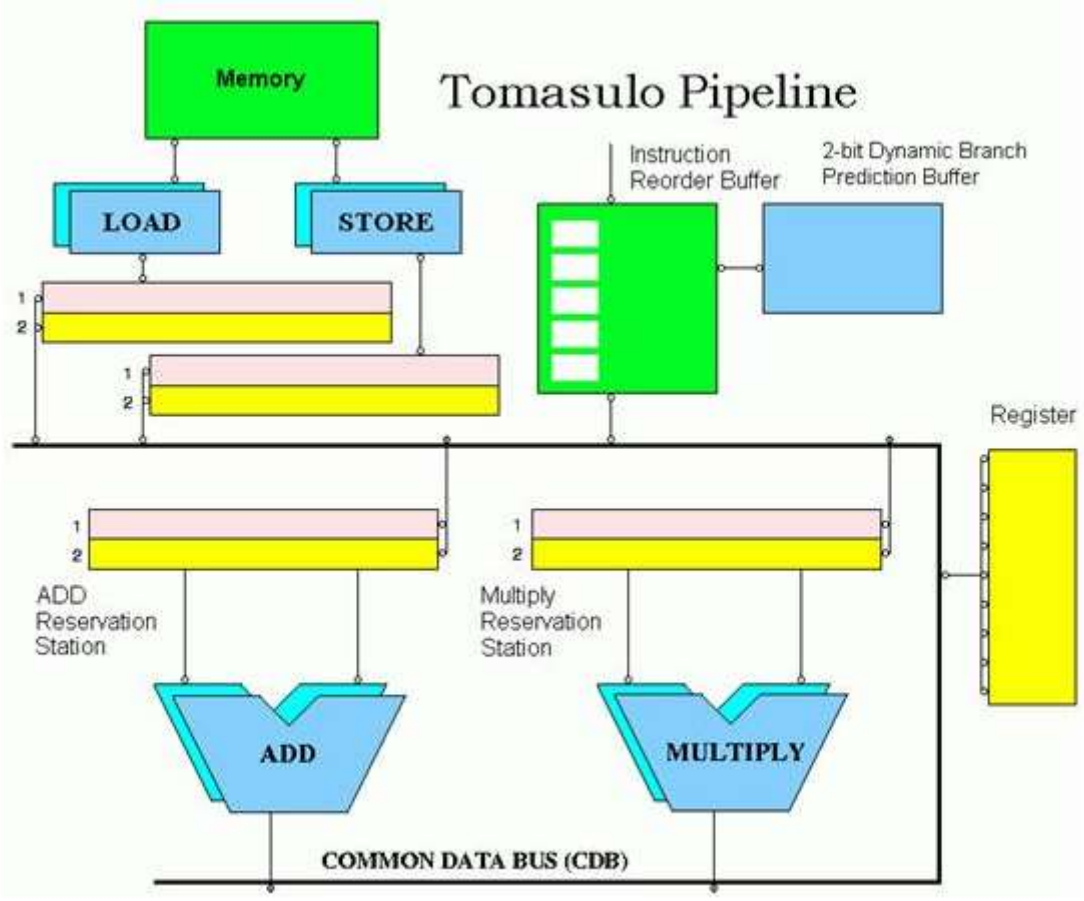

图 2 模拟器整体结构

我们拥有如下的执行单元：

• 两个 Load 缓冲区

• 两个 Store 缓冲区

• 两个整数 ALU 运算部件

本实验中不需要实现浮点运算，在算法上它和整数运算没有什么区别，区别只是在于执行时间的长短。

我们定义指令的执行时间如下：

```
/*
 * 不同操作所需要的周期数
 */
#define BRANCHEXEC  3   /* 分支操作 */
#define LDEXEC      2   /* Load     */
#define STEXEC      2   /* Store    */
#define INTEXEC     1   /* 整数运算 */
```

对于分支指令，在三个周期执行完成后，我们就知道了跳转的目标。在此之前，我们不知道分支程序将会转向哪里，但是你可以使用分支预测缓冲栈来进行分支预测。最初，你也可以假设所有的跳转条件都为假，于是只要顺序取指并在需要跳转的时候清空流水线就可以了。

4.3 保留栈

保留栈(Reservation Station)是 Tomasulo 算法的关键组件。在指令发射之后，将所需的寄存器的值(操作数)复制到保留栈的 Vj 和 Vk 字段中。如果所需的操作数还没有准备好，在 Qj 和 Qk 字段中记录需要的结果正在哪个运算单元中执行。对于那些只需要一个寄存器操作数的指令(例如 I 类型指令)，置 Qk=0，Vk=0。保留栈监视公共数据总线，等待所需结果的出现。

其格式如下：

```
typedef struct _resStation {    /* 保留栈的数据结构 */
    int instr;      /*    指令    */
    int busy;       /* 空闲标志位 */
    int Vj;         /* Vj, Vk 存放操作数 */
    int Vk;
    int Qj;         /* Qj, Qk 存放将会生成结果的执行单元编号 */
    int Qk;         /* 为零则表示对应的V有效 */
    int exTimeLeft;    /* 指令执行的剩余时间 */
    int reorderNum;    /* 该指令对应的先行指令窗口编号 */
}resStation;
```

图 4 保留站格式

4.4 再定序缓冲器（ROB）

要发射一条指令，ROB 中必须有一条空闲的项目。指令和执行指令的单元编号都保存在这里。当

结果生成之后，它也被临时保存在这里，并将结果有效标志置为真。对于 SW 指令，也将目标内存单元记录在这里(storeAddress 字段)。

ROB 是一个循环队列。在发射指令时，需要在队尾检查是否有空闲的项目。在提交指令结果时，需要在队首逐项进行。尽管指令的执行是乱序的，但提交结果必须是顺序进行的。如果队首的结果有效标志为真，将提交结果。

其格式如下：

```
typedef struct _reorderEntry {  /* ROB项的数据结构 */
    int busy;          /* 空闲标志位 */
    int instr;         /* 指令 */
    int execUnit;      /* 执行单元编号 */
    int instrStatus;   /* 指令的当前状态 */
    int valid;         /* 表明结果是否有效的标志位 */
    int result;        /* 在提交之前临时存放结果 */
    int storeAddress; /* store指令的内存地址 */
}reorderEntry;
```

图 5 ROB 项格式

4.5 寄存器状态

寄存器状态是寄存器的一部分。它表明寄存器中是否存放了有效的结果，或者它是否正在等待 ROB 中的某项提交结果。在发射指令时，我们需要根据寄存器的状态来填写保留栈的 Vj、Vk 和 Qj、Qk 字段。

```
typedef struct _regResultEntry {    /* 寄存器状态的数据结构 */
    int valid;         /* 1表示寄存器值有效，否则0 */
    int reorderNum;    /* 如果值无效，记录先行指令窗口中哪个项目会提交结果 */
}regResultEntry;
```

图 6 寄存器状态格式

4.6 分支预测缓冲区 BTB（选做）

最初，你可以假设所有的跳转条件都为假，于是预测跳转不会执行。在发现预测错误时，将流水线(保留栈和 ROB)清空。你也可以实现一个分支预测缓冲栈来进行更为精确的预测，使用 2-bit 的历史记录来进行预测。下面是历史信息的状态转换图：

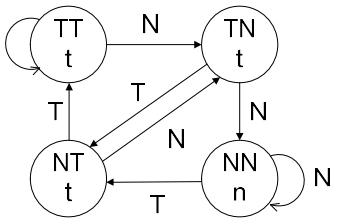

图 7 分支预测状态机

• TT、TN、NT、NN 2-bit 历史记录

• T 实际转移成功

• N 实际转移不成功

• t 预测转移成功

• n 预测转移不成功

如果当前 PC(正在取指的指令的 PC)与分支预测缓冲栈中的某个项目相符，那么说明正在读取的指令是分支指令。检查分支预测缓冲栈以对跳转进行预测。如果预测跳转，那么需要将 PC 修改为跳转目标并继续执行；如果预测不跳转，不对 PC 进行修改。

第一次执行一条分支指令时，将它添加到分支预测缓冲栈中去。此时它还没有历史记录，你可以尝试分别以 TT、TN、NT、NN 状态作为入口，分析它们的不同效果。

在向分支预测缓冲栈中加入新的项目时，你需要寻找一个空闲空间。如果缓冲栈已满，你必须覆盖一个已有的项目。这时你可以尝试使用不同的替换算法，例如随机选择一项替换，或者使用 LRU(最近最少使用)替换策略。

```
typedef struct _btbEntry{   /* 分支预测缓冲栈的数据结构 */
    int valid;          /* 有效位 */
    int branchPC;       /* 分支指令的PC值 */
    int branchTarget; /* when predict taken, update PC with target */
    int branchPred;     /* 预测：2-bit分支历史 */
}btbEntry;
```

图 8 分支预测缓冲区格式

4.7 虚拟机状态和内存模型

```
typedef struct _machineState{   /* 虚拟机状态的数据结构 */
    int pc;         /* PC */
    int cycles;     /* 已经过的周期数 */
    resStation reservation[NUMUNITS];       /* 保留栈 */
    reorderEntry  reorderBuf[RBSIZE];       /* ROB */
    regResultEntry regResult[NUMREGS];      /* 寄存器状态 */
    btbEntry  btBuf[BTBSIZE];               /* 分支预测缓冲栈 */
    int memory[MEMSIZE];        /* 内存   */
    int regFile[NUMREGS];       /* 寄存器 */
}machineState;
```

需要注意的是，寄存器值 regFile 和寄存器状态 regResult 是两个不同的结构。每当在程序中需要读取寄存器的值时，你都应该首先检查寄存器的状态标志位，以确定寄存器中保存的值是否是有效的。

内存是用一个很大的数组来实现的。为使程序的输出比较简洁，内存采用如下的管理方式：第 0～15 单元(每个单元为一个 32 位字)为数据段，从第 16 单元开始为代码段。机器代码从文件中读入后，写

入到代码段中，因此 CPU 在初始化以后，最初的 PC 值应该是 16，而不是 0。需要注意的是，你的汇编程序只能读写数据段范围内的内存，对超出这个范围的内存进行操作会导致执行出错甚至程序崩溃。

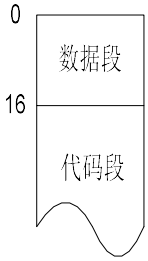

图 9 简单内存 管理

4.8 算法流程

Tomasulo 算法在每个时钟周期需要：

1. 首先检查位于 ROB 的栈顶的指令是否能够提交结果。如果能，检查该指令是否是条件跳转 beqz 或无条件跳转 j 指令，如果需要跳转，那么你需要清空 ROB 和保留栈中已经发射的所有的指令，并从正确的跳转目标处继续取指执行。

   如果该指令不是跳转指令，根据指令的内容对寄存器或内存进行必要的修改。指令的结果提交之后，将 ROB 中的该项释放。

2. 下一步，对所有已经发射的指令进行处理。指令的执行分为四个状态：发射(Issuing)，已发射但正在等待操作数的指令处于这个状态；执行(Executing)，正在运算部件中执行的指令处于这个状态；写结果(Writing Result)，已经完成执行，正在向保留栈和 ROB 中写回结果的指令处于这个状态；提交(Committing)，正在等待提交结果的指令处于这个状态。

   从内存中取出的指令在分配了一个 ROB 项和一个保留栈之后进入 Issuing 状态。在所有的操作数都准备好之前，它一直处于这个状态，准备好之后则进入执行状态。

   在指令执行时，保留栈有一个 ExTimeLeft 字段用于记录执行的剩余时间，实现对指令执行的行为模拟。在指令刚刚开始执行时，需要将指令执行所需的时间写入到这个字段中。以后的每个周期都将 ExTimeLeft 字段减 1，当它等于零时，指令执行完成，进入 Writing Result 状态。

   Writing Result 状态将指令的执行结果写回到该指令所在的 ROB 项中的一个临时存储字段中，并通过公共数据总线将这个结果发送到正在等待它的其他保留栈中去。注意，Writing Result 状态并没有修改寄存器或内存，这是因为到此时为止指令的执行一直都是乱序的，我们必须保证以正确的顺序来提交结果。在将结果写回到 ROB 中之后，进入 Committing 状态，此时可以释放相应的保留栈，下一条指令就可以占用空闲下来的运算部件了。

   ROB 是一个循环队列。处于 Committing 状态的指令必须在队列中等待，一直等到到达队首的时候才能够提交。指令提交结果时，可能会修改寄存器或内存(运算指令或内存指令)，也可能会导致流水线的清空(跳转指令)。

3. 最后，在每个时钟周期我们都要检查是否能够发射一条新的指令。要发射一条指令，首先在 ROB 中必须有空闲的项目，而且所需的运算单元必须有空闲的保留栈。指令的发射必须是顺序进行的。

这里给出的算法流程是为了给同学们做参考，不必完全按照这个流程走，请记住我们鼓励创新，不希望限制你的发挥，我们欢迎同学们重构代码，只需要在报告中说明即可。

4.9 HALT 和 NOOP

假设 halt 和 noop 指令都会进入整数运算单元来执行。这简化了指令执行时的操作，你可以像一般指令一样来处理它们，这也保证了所有的指令都以正确的顺序来提交。Noop 指令提交时，虚拟机不进行任何操作；Halt 指令提交时，虚拟机停机。

4.10 可视化界面示例

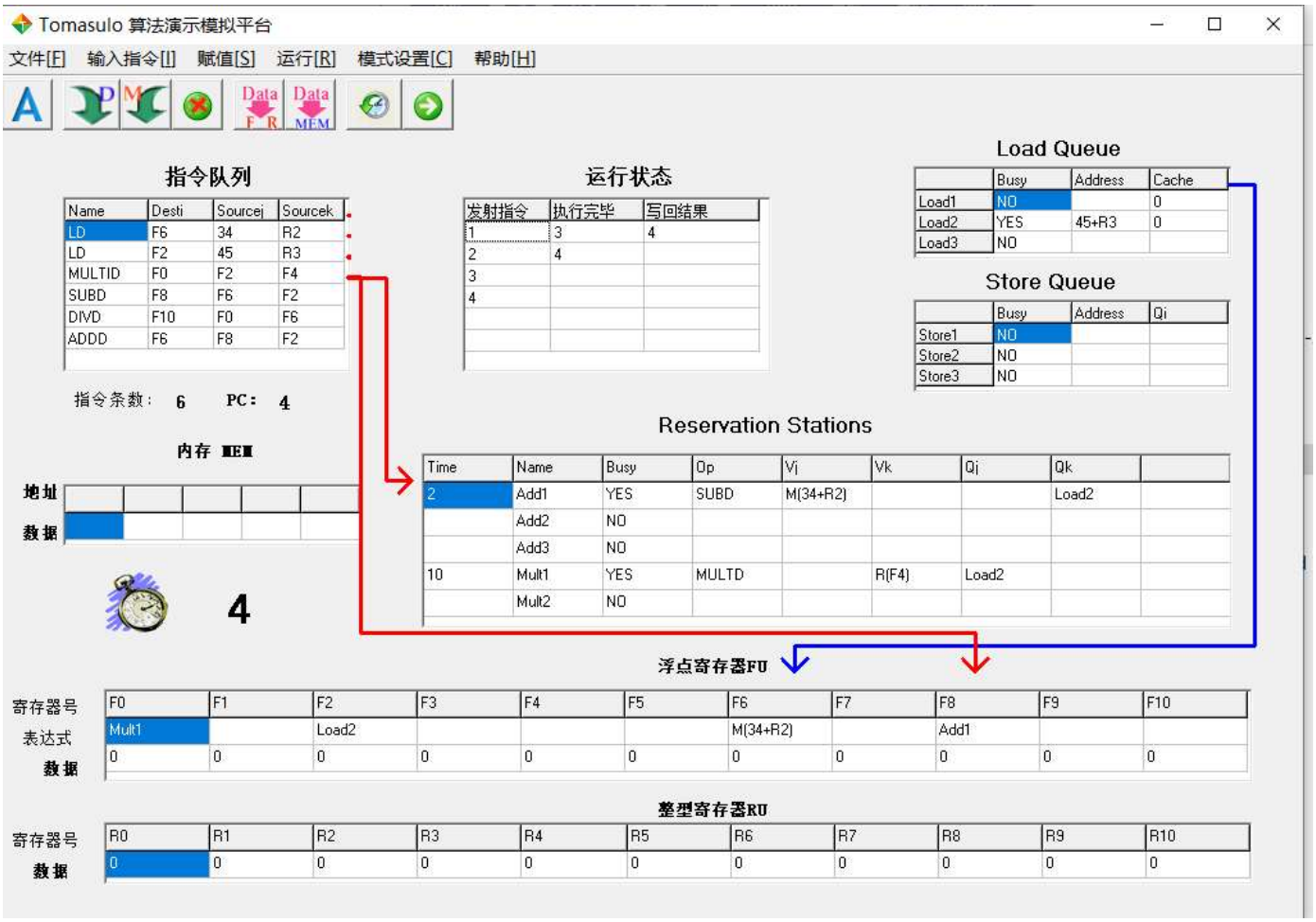

图 10 可视化界面示例

界面需要展示出当前时钟周期下处理器中所有状态，包括各功能部件中的内容，状态转移方向（控制流和数据流）。此处给出的界面仅作参考，不必包括其中的菜单栏等。

5 一些提示

5.1 模板文件

模板 template.c 中已经实现了主要的数据结构和程序框架，它的目的是减轻你的工作量，而不是限制你的发挥。如果需要，你可以定义自己的变量和函数。例如，如果你要实现分支预测的选作内容，就可能需要对 ROB 的数据结构进行修改。需要注意的是，不要修改模板中的输入输出部分，例如 printState()函数。你需要保证自己的输出与标准输出完全一致。

汇编器需要你自主实现，在指令格式已给出的情况下这不是一个很难解决的问题，需要注意的是如何处理带标号的指令（如输入中的 J LOOP）。

关于程序实现的细节，模板中已经包含了大量的注释，你可以参照这些注释来完成你的程序。

5.2 立即数和偏移量

I 类型指令和 J 类型指令中的立即数和偏移量分别是 16 bit 和 26 bit 的整数。为将它们转化为在程序中可以使用的 int 类型的数据，你需要使用 convertNum()和 convertNum26()两个函数。

5.3 C 语言的位运算

在对机器代码进行处理时，C 语言的位运算>>、<<、&、|、~、可能是很有用的。例如，如果需要检查整数类型变量 a 的第 3 位，可以使用(a>>3) & 0x1；要将 6 放到变量 a 的第 3~5 位，3 放到第 1、2 位，可以使用 a = (6<<3) | (3<<1)。

5.3 C 语言的位运算

你可以使用输出重定向将虚拟机的运行结果输出到文件中以方便 debug。使用如下命令：

simulator <输入文件名> > <输出文件名>

四 评价标准

本次实验的基础分共 15 分，加上选做加分 1 分，总分计算最高取 15 分。

1 评分细则

表 4-1 评分细则

任务	子项	评分/pts
Tomasulo 算法实现	基础功能部件实现正确	8
	实现分支预测缓冲区 BTB（选做）	1
编译器	11 种指令机器码翻译正确	2
可视化界面	线上报告中对模拟器和调度的设计说明	2
	线下演示清晰的展示出 Tomasulo 算法运行过程	2
	界面简洁美观	1

2 正确性评价

对于模拟器的正确性评价，我们在现场演示的时候选定一些样例输入，根据所实现的模拟器输出结果（各个功能部件的值：具体为寄存器、内存、保留站和 ROB 的状态）来给予正确性评价，同时根据界面展示的内容给予界面展示分数。对于选做内容 BTB，也需要界面展示做要求，同时需要在文档中详细说明设计。基础分数是 $Base$，加分是 $Bonus$，则最后总分的计算方式是：

$$Grade = Min\{15, Base + Bonus\}$$

3 实验展示

除了需要同学们线上提交一份报告，报告内容需要包括你的源码，还有关于实验的设计。此外在检查的时候需要和同学们约定线下展示的时间。

4 实验反馈

我们希望能从同学们这里得到关于本实验的反馈，关于实验内容、方式和目的等都可以，也欢迎同学在报告中交流关于本次实验的体会。