Gardenia

——计算机系统大作业报告

F1124004 5110309058 陈志鹏
F1124004 5110309028 陈楠昕

目录

一．简介

这个项目是一个 CPU 及相应编译器、汇编器的设计，由陈志鹏和陈楠昕合作完成。CPU 是 Harvard 结构的 Tomasulo 设计，含 64 个寄存器，指令集为 32 位 RISC，并且实现了 Multiple Issue 及 Cache，用 verilog 描述；编译器和汇编器用 Java 实现。

二．ISA 设计

ISA 参考 MIPS 设计，但基于项目需要有适当修改。每条指令为 32 位，前 4 位为指令代码。
对于部分指令，最后一位表示该指令是否包含立即数，最后一位为 0 表示最后一个参数是寄存器，如果为 1 说明最后一个参数是立即数（Imm）。由于有 64 个寄存器，需 6 个二进制位表示一个寄存器。
对于 bgt 和 bnez，所要跳转的地址在汇编代码中用 label 表示，在二进制代码中用基于该指令的偏移量表示。
以下为汇编指令和对应二进制码，其中下标表示所占位数：

add	
汇编	add dst, src1, src2/Imm
二进制（无立即数）	1000 dst_6 $src1_6$ $src2_6$ 00 00000000
二进制（有立即数）	1000 dst_6 $src1_6$ Imm_{15} 1
mul	
汇编	mul dst, src1, src2/Imm
二进制（无立即数）	1001 dst_6 $src1_6$ $src2_6$ 00 00000000
二进制（有立即数）	1001 dst_6 $src1_6$ Imm_{15} 1
bgt	
汇编	bgt src1, src2, label
二进制	1010 $src1_6$ $src2_6$ $offset_{16}$
bnez	
汇编	bnez src1, label
二进制	1011 $src1_6$ $offset_{22}$
lw	
汇编	lw dst, src1, src2/Imm
二进制（无立即数）	1100 dst_6 $src1_6$ $src2_6$ 00 00000000
二进制（有立即数）	1100 dst_6 $src1_6$ Imm_{15} 1
sw	
汇编	sw dst, src1, src2/Imm
二进制（无立即数）	1101 dst_6 $src1_6$ $src2_6$ 00 00000000
二进制（有立即数）	1101 dst_6 $src1_6$ Imm_{15} 1

j	
汇编	j label
二进制	1110 $offset_{28}$
mv	
汇编	mv dst, src1/Imm
二进制（无立即数）	1111 dst_6 $src1_6$ 00000000 00000000
二进制（有立即数）	1111 dst_6 Imm_{21} 1
nop	
汇编	nop
二进制	00000000 00000000 00000000 00000000
halt	
汇编	halt
二进制	00010000 00000000 00000000 00000000

三．Compiler 设计

Compiler 设计是用 Java 描述的，支持的语法仅为样例程序中所用到的，仅支持将样例程序更改常数或者用完全相同的句法。Compiler 的词法分析使用 JFlex 完成，语法分析使用 JCup 完成，构建了 AST 然后转成汇编代码。

针对矩阵乘法，Compiler 采用了 Loop Unrolling 技术，将最内层循环完全展开。一开始打算一次最多 issue 26 条指令，Compiler 根据最内层循环体部分重新排列代码并分块，在中间插入 nop 指令表示一次 issue 的结束。后面改为一直 issue 直到需要 stall。

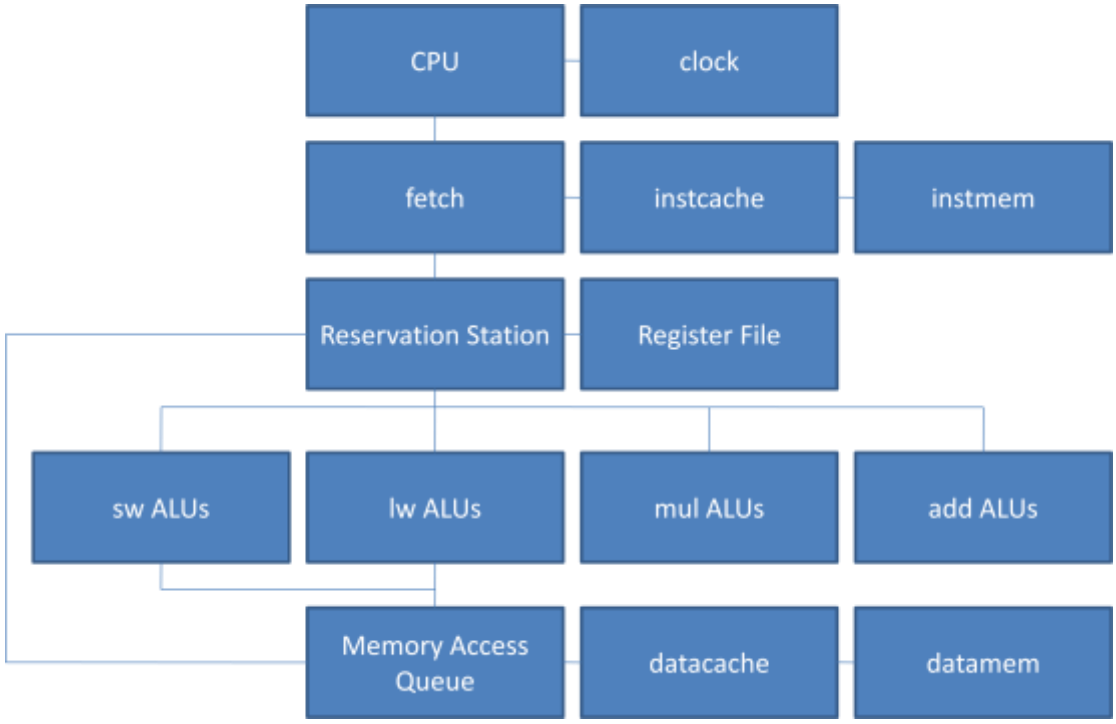
四．CPU 设计

CPU 设计为 Tomasulo，但将 EX 和 WB 两个阶段合为一个，即一旦指令完成就立刻把结果送向所需的运算单元。
CPU 包含以下几个文件：

CPU.v	整个 CPU 最高层模块
clock.v	以 2ps 为周期提供方波输出
fetch.v	从 instcache 获取指令，解码后放入 Reservation Station
RS.v	Reservation Station、Register File
ALU.v	加法器、乘法器
instcache.v	指令缓存
datacache.v	数据缓存
instmem.v	指令内存，从 input.bin 中读取程序；到内存取指令有 100 周

	期延迟
datamem.v	数据内存，从 ram_data.txt 中读取数据并将最后结果放入 cpures.txt；内存读写有 100 周期延迟
define.v	定义 timescale 和程序中所用到的常量

CPU 结构如下图所示：



1. CPU：
   包含 clock 和 fetch，将 clock 产生的时钟波形传给 fetch。
2. fetch：
   通过 instcache 获取指令，每次获取多条指令，解码后传给 Reservation Station，若 Reservation Station 满或遇到 bgt 指令操作数没有得到或 Instruction Cache Miss 则 stall。
3. Reservation Station：
   将 fetch 传来的解码后指令放入表中。CPU 中共有 96 个 lw 单元，32 个 add 单元，32 个 mul 单元，32 个 sw 单元，需要 8 个二进制位标记运算单元。表的每一项设计如下：
   add、mul、lw：

$Busy_1$	$Value1_{32}$	$Value2_{32}$	$Quest1_8$	$Quest2_8$	$Valid1_1$	$Valid2_1$

   Busy 表示该运算单元是否被在占用，Valid1 和 Valid2 表示该运算单元的两个操作数是否可用。Quest1 和 Quest2 表示操作数来自哪个运算单元，可由 Register Status 获知，若操作数是立即数或从寄存器中去出，则 Quest 无效。可用的操作数的值保存在 Value1 和 Value2 里面。
   sw：

$Busy_1$	$Value1_{32}$	$Value2_{32}$	$Value3_{32}$	$Quest1_8$	$Quest2_8$	$Quest3_8$	$Valid1_1$	$Valid2_1$	$Valid3_1$

   各二进制位意义同上。

在各个操作数都可用时，该运算单元就会执行，得出结果后，把结果送给需要的运算单元或寄存器，最后清零。

4. Register File：
   每个寄存器有一个 8 位的 Register Status，表示要写入该寄存器的运算单元。若没有运算单元要写入该寄存器，则 Register Status 为 01111111，表示寄存器的值有效。
5. Memory Access Queue：
   这个元件目的是防止 lw 和 sw 乱序执行，该元件实质是一个循环队列，会依次按照指令 issue 进入的顺序处理每个 lw 和 sw。

五．Cache 设计

Cache 共 8 个 Block，每个 Block 大小为 128Byte，用 Direct-Map 方式。
Instcache 是只读的，datacache 可读可写，用 write back 方式，如果 write miss，将 dirty 的 block 写入内存再把新 block 拿到 cache 并写入数据。

Cache 每次 miss 时从 Memory 拿两个相邻的 block 回来，Instcache 返回从请求地址开始的一个 block。

在最后一条指令 halt 被 issue 进入 Reservation Station 时，Reservation Station 发出一个信号，通知 datacache 将所有 dirty 的 block 写回 datamem 中，然后通知 datamem 将数据写入文件。

六．组内分工

陈志鹏：ISA 的设计，CPU 的总体设计，编写汇编器，设计并编写 RS、instmem、datamem、datacache 和 ALU，参与最后整合与调试，起草并修改报告
陈楠昕：设计并编写编译器、fetch 和 instcache，最后整合并调试，调整 CPU 参数，修改报告。

七．使用介绍

编译器和汇编器：编译运行 decency.compiler.Main 或运行 jar 文件(不需要加参数)，输入文件为 test.c，输出文件为 test.s 和 test.bin，其中 test.s 为汇编代码，test.bin 为二进制代码。
CPU：使用 modelsim 仿真 bourgeois 中的 CPU.v，输入文件为二进制代码 input.bin 和内存原数据 ram_data.txt，输出文件为 cpures.txt，为 CPU 仿真结束后内存数据。

八．测试结果

对于给出的样例数据的运行时间(其中 finish fetch 指的是最后一条指令 halt issue 的时间)：

```
# finish fetch in cycle 218
# ** Note: Data structure takes 6457724 bytes of memory
#          Process time 0.09 seconds
#          $finish    : datamem.v(48)
#    Time: 636 ps  Iteration: 2  Instance: /CPU/fet/rs/data/data
```

对于自己生成的 30*30 大小的矩阵乘法运行时间：

```
# finish fetch in cycle 329304
# ** Note: Data structure takes 4753740 bytes of memory
#          Process time 78.50 seconds
#          $finish    : datamem.v(48)
#    Time: 659410 ps  Iteration: 2  Instance: /CPU/fet/rs/data/data
```

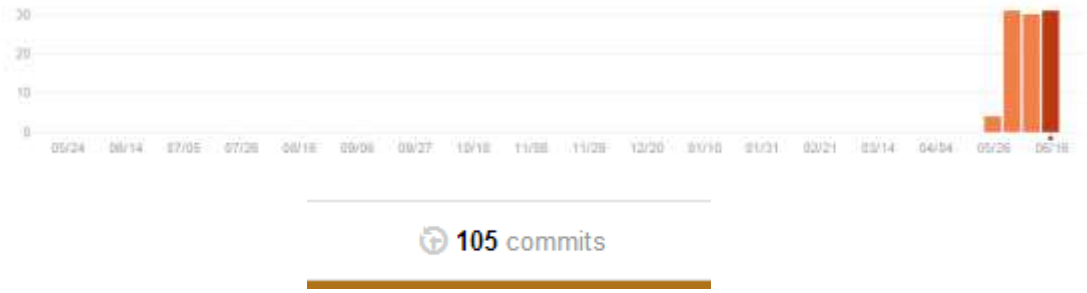
生成代码可见 makedata.cpp。

测试得出来的结论是运行速度很大程序上取决于 cache，调整运算单元个数很大程度上也能提高性能，因为瓶颈部分应该在于 Memory Access Queue，而这里的并行性很差，因此适当减少 lw 和 sw 单元增加其他单元应该能有效提高效率。但由于时间关系没有进行太多的尝试。

九．总结与致谢

在这个项目中，陈楠昕与陈志鹏并肩协力，共克千难万险，成功完成该项目。下面是 Github 上面的一些数据统计。

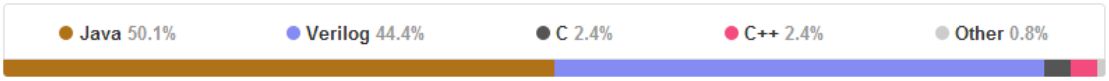
Commit activity



Commits（左为陈志鹏，右为陈楠昕）：

Additions（左为陈楠昕，右为陈志鹏）：

Language statics：



同时，非常感谢李青林提供的帮助。