Yet Another Young and Naive CPU

倪昊斌[1]　陈子豪[2]

ACM Honored Class
Zhiyuan College
Shanghai Jiao Tong University

2015 年 6 月 19 日

[1]Student ID: 5122409037
[2]Student ID: 5130309305

Contents

Chapter 1

导言

本文是作者的计算机系统 1 课程设计的报告。作者实现了一个基于 MIPS 指令集的 5 级 pipeline 处理器，并且拥有以下特性：**三级缓存**，**fully-bypassing**, **2-level GAP branch predictor**，支持大部分常用 MIPS 指令，并且在实现过程中采用了模块化设计，本文接下来将会对处理器的基本架构和特性的设计按照模块逐一阐述。

Chapter 2

基本架构

我们综合考虑处理器的 5 个阶段以及实现难度后，将处理器大致分为以下 5 个模块。不同阶段之间的 pipeline register 将会分布在各个模块之中。

2.1 Program Counter

文件: program_counter.v

功能：PC 在每个时钟上升沿时将 PC+4。在遇到分支和跳转指令（预测）时改变 PC 的值实现跳转。在由于读取指令和内存操作时需要 stall 时保持 PC 值不变。

实现：若没有 stall 信号，在每次时钟上升沿将 PC+4 放入寄存器。如出现 target_enable 信号则修改 PC 的值。

2.2 Instruction Fetch

文件：inst_memory.v

功能：根据 PC 的值到缓存中读取指令。对于分支指令进行跳转预测。

实现：在时钟上升沿将 PC 的值输入一级缓存，如果出现 stall 则传递 inst_stall 信号。如果取出 branch 指令则交由 branch predictor 进行预测，将预测结果输出到 PC。

2.3 Instruction Decode & Write Back

文件：RegFile.v

功能：这一模块完成了对于当前指令的解码（判断指令类型、提取操作数），立即数位数补齐等工作，由于寄存器阵列在这一模块实现，写回寄存器的操作也在这个模块实现。这里采用了先写后读的设计，即如果需要写回的寄存器与需要读取的寄存器是同一寄存器的话，先进行写回操作在进行读取操作。

实现：先判断是否有数据需要写回寄存器，写完之后对输入指令进行解码，解码出运算数 A 和 B，如果是 I-type 的指令则 B 为补齐之后的立即数，同时把要写入内存的数据 wdata 向后传递。

2.4 Execution

文件：arithmetic_logic_unit.v

这一模块对 ID 阶段传过来的操作数进行相应的代数运算（包括内存地址计算），如果当前指令是条件跳转指令，此模块将会完成跳转条件判断以及跳转目标 pc 计算。所有操作都在时钟上升沿时完成。

功能：根据运算的指令进行计算（包括内存地址计算）。判断实际是否跳转，如果和预测结果不一致则修正预测的结果。

实现：在时钟上升沿，先进行分支指令比较结果的计算。如果与预测结果不符，flush 错误的指令并修正 PC。发送至 branch predictor 改变 predictor 状态。然后根据指令对运算数进行计算。

2.5 Memory Access

文件：data_memory.v

功能：由 ALU 得到地址，进行数据的读取或存储。

实现：在时钟上升沿根据指令将地址和数据 (如果是 sw 指令) 输入一级缓存，出现 stall 则传递 data_stall 信号。

Chapter 3

特性

3.1 三级缓存

文件：L1_cache.v, L2_cache.v, L3_cache.v

YAYNCPU 之所以能在 pipeline 和 tomasulo 的 CPU 之中以较为优秀的性能脱颖而出，一个重要原因是实现了高效的三级缓存机制，有效地减少了对于内存的访问次数以及因此而触发的延迟。三级缓存均分为独立的指令缓存和数据缓存，采用了 direct map 的寻址机制和 write through 的写策略。其一、二、三级的 block size 分别为 64B、128B 和 256B。这使得三级缓存和内存之间只需进行 4 次通讯（1 次指令，2 次数据和 1 次写入），对于性能有很大的提升。

YAYNCPU 缓存的实现也因模块化设计而框架清晰明了。首先指令缓存和数据缓存是由相同的模块分别实例化得到的。而每级缓存的设计也是一样的，仅在参数上有所不同：均从上级输入地址和写入数据/信号，当发生 cache miss 时发布 stall 信号并将地址等信息输出给下级，而当下级缓存返回结果后则立即开始向上级输出结果。

3.2 Fully-bypassing

文件：CPU.v

bypassing 的部分没有独立设置一个模块，而是在将各个模块连接在一起时将传进相应模块的运算数进行修改。作者实现了 5 级 pipeline 所能够实现的所有 bypassing，具体为，MA 到 EX，EX 到 EX，WB 到 MA，WB 到 EX。利用这些 bypassing，只有一些极个别情况需要 stall。考虑到指令分为 I-type 以及 R-type 指令，实现时会对指令类型进行分类讨论。

3.3 2-level GAP Branch Predictor

文件: branch_predictor.v

branch predictor 采用的是奔腾 Pro 处理器曾经采用的 GAP 设计, 与大多数 predictor 相同，

branch history register 记录的是之前两次跳转指令的结果，pattern history table 中每个 entry 中是一个 2bit 的自动机，初始状态为”10”。branch predictor 模块内嵌于 IF 的模块中，每次在 IF 阶段进行预测。

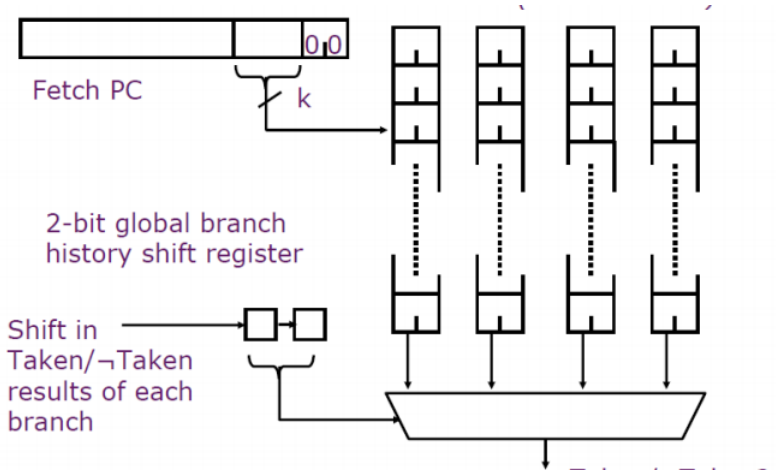

Figure 3.1: 2-level GAP branch predictor

Chapter 4

性能

由于使用了三级缓存，fully-bypassing 以及 GAP branch predictor，此处理器在运行 testbench 时展现出了非常高的性能，我们将一级缓存的 hit time 约定为 0，二级缓存的 hit time 约定为 3 个 cycle, 三级缓存的 hit time 约定为 5 个 cycle, miss time 即内存访问时间为 200 个 cycle，写入采用 write through 方法，对于内存中 100 个数相加的 testbench（总共 402 条指令），需要 1047 个 cycle，CPI 为 2.6，应该说性能是相当出色的。

本程序会在 instruction_memory.v 文件中读入指令文件 program.txt，并且已经预先将内存第 1 到 100 个 word 进行初始化（为 1-100）（注：也可以对 data_memory.v 进行略微修改后从文件读入用来初始化内存的数据），测试时只需运行 CPU.v 即可，最终结果将会写回内存的第 101 个 word。

testbench 的 MIPS 代码如下：

```
1                   addi $t3 $0 400
2                   loop:
3                   lw $t2 0($t1)
4                   addi $t1 $t1 4
5                   add $t0 $t0 $t2
6                   bne $t1 $t3 loop
7                   sw $t0 0($t1)
```

此处理器支持的指令为：

Table 4.1: 支持指令

指令	指令类型	opcode	使用示例
add	R-type	100000	add $d,$s,$t
addi	I-type	001000	addi $t,$s,C
addu	R-type	100001	addu $d,$s,$t
multu	R-type	011001	multu $d,$s,$t
xor	R-type	100110	xor $d,$s,$t
sll	R-type	000000	sll $d,$t,shamt
lw	I-type	100011	lw $t,C($s)
sw	I-type	101011	sw $t,C($s)
lui	I-type	001111	lui $t,C
beq	I-type	000100	beq $s,$t,C
bne	I-type	000101	bne $s,$t,C

Chapter 5

结语

通过实现这样的一个有一定挑战性的处理器，我们对于处理器构造的细节有了全面透彻的理解。在调试过程中，我们也遇到了不少问题，最终逐一克服，完成这样一个接近极致的 5 级 pipeline 处理器也给我们带来的巨大的成就感。最后，十分感谢计算机系统 1 课程为我们提供的这样一个深入理解处理器的机会，十分感谢幽默风趣、授课深入浅出的梁晓峣教授，同时十分感谢认真负责的叶冉助教。