五流水 MIPS CPU 实现

PB09210183 何春晖 PB09210174 林晗 PL09215001 切瑞

2012.06.18

1 总体设计

1.1 指令集

以标准 MIPS 指令集作为基准，提取出其中？31？条指令进行实现。

1.2 设计框图

最终实现如图 1。图中虚线为控制信号，实线为数据通路。

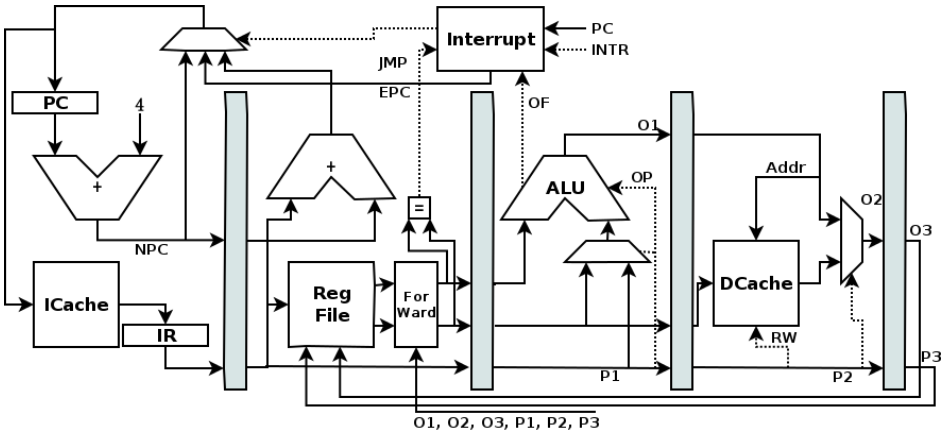

图 1: 五流水 MIPS CPU 框图

将要实现的 CPU 共分为五个阶段，各个从左往右阶段的功能依次为：

- 取指令：将指令从指令 Cache 中取出，并计算下一条指令的地址；
- 读操作数及转移：根据指令从寄存器中读出操作数，同时若为转移类指令，满足转移条件时设置 PC；
- 执行：根据指令进行计算，若为存取类指令，则计算地址；
- 访问存储器：若为存取类指令，对进行数据 Cache 进行读写；
- 写结果：将执行或存取的结果写回寄存器。

其中指令 Cache 和数据 Cache 实现时看作两个分立存储器。因此该 CPU 结构属哈佛结构。

全系统没有集中的控制器，而是将 IR 传递到各个阶段，由各个阶段自行根据需要译码。

此结构图与一般可查的图略有不同，原因将在后面予以解释。

2 部件设计

2.1 寄存器堆

2.2 运算器

2.3 中断模块

2.4 转发模块

为了克服流水线的数据相关，需要实现数据转发。

转发模块的原理很简单，就是在读操作数时，将欲读取的端口号与后面三个阶段的输出端口号相比。若出现相同，则选择相同的那个阶段的输出，否则才选择寄存器的值。

需要注意的是，MIPS 规定 0 号寄存器的值永远是零，对其写值全忽略。因此若欲读取的端口号为 0 时，总是选择寄存器的值（零）。

3 流水阶段的操作

3.1 取指令

此阶段要做的微操作为：

```
npc = pc + 4
if not jmp
    pc <- npc
    ir <- icache[pc]
else
    pc <- jmp_addr + 4
    ir <- icache[jmp_addr]
endif
```

3.2 读操作数及转移

此阶段要做的微操作为：

```
//读操作数
tmp_a = reg[ir[25:21]]
tmp_b = reg[ir[20:16]]
```

```
//转发
for_a = forward(tmp_a)
foe_b = forward(tmp_b)

//判断转移
r = for_a - for_b
if    (ir == beq and r == 0)
   or (ir == bne and r != 0) then
     jmp = 1
     jmp_addr = npc + SEXT(ir[15:0] << 2)
else if ir == jal or == j then
     jmp = 1
     jmp_addr = {npc[31:28], ir[25:0] << 2}
else if ir == jr then
     jmp = 1
     jmp_addr = for_a
else
     jmp = 0
endif
```

一般的设计是将转移放在执行阶段进行的，但是 MIPS 规定转移时只能有 **一个分支延迟槽**。故若在执行阶段才判断转移，流水线中已经有两条转移之后的指令，不满足要求，除非清空流水线。

将转移判断放在此阶段，**虽然增加了几个运算部件，但是却可以保证转移时流水线只有一条后面的指令，从而不需清流水线，从而提高效率**。

还要注意的是，若读存储器指令后紧接着对读入数据的运算指令，该运算指令无论怎么转发，都不能得到正确的值。这种情况 MIPS 称 **加载延迟**，并且禁止出现这种指令序列。因此这虽然是一个问题，但是不需要特殊处理。

3.3 执行

此阶段对指令译码并做相应运算。

3.4 访问存储器

此阶段需要判断是否为存取类指令。若不是，直接向下一级传递；否则，发读写令并选择结果。

3.5 写结果

此阶段将结果写回寄存器。

4 仿真测试

4.1 测试一：final_test

这是全指令测试程序。测试程序事先已经在 SPIM 虚拟机上运行，供仿真比较。
虚拟机结果、仿真结果如图 2、图 3。

```
User data segment [10000000]..[10040000]
[10000000]    000a0004  000a0004  0009fffe  0009fffe    . . . . . .
[10000010]    00000001  000a0003  000a0002  fff5fffc    . . . . . .
[10000020]    00000000  00000000  00500008  00014000    . . . . . P . . @ . .
[10000030]    00014000  00500008  00014000  00014000    . @ . . . . P . . @ . . . @ . .
[10000040]    00000000  000a0015  0009ffed  00000001    . . . . . .
[10000050]    000a0005  000a0004  00640000  00000000    . . . . . d . . . . .
[10000060]    00000001  00000000  00000000  00000000    . . . . . .
```

图 2: final_test 虚拟

Addr	+0	+1	+2	+3	+4	+5	+6	+7
000	000A0004	000A0004	0009FFFE	0009FFFE	00000001	000A0003	000A0002	FFF5FFFC
008	00000000	00000000	00500008	00014000	00014000	00500008	00014000	00014000
010	00000000	000A0015	0009FFED	00000001	000A0005	000A0004	00640000	00000000
018	00000001	00000000	00000000	00000000	00000000	00000000	00000000	00000000
020	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000

图 3: final_test 仿真

结果正确。

4.2 测试二：test

这个程序仅仅对 C 编译器会产生的指令进行了测试。重点考察了数据转发、分支延迟等情况。
虚拟机结果、仿真结果如图 4、图 5。

```
User data segment [10000000]..[10040000]
[10000000]    55aaaa55  aa5555aa  00005555  00003311    U . . U . U U . U U . . . 3 . .
[10000010]    ffffccef  ffffffff  55aa5555  55aaaaff    . . . . . U U . U . . . U
[10000020]    ab5554aa  56aaa954  2ad5552a  156aaa95    . T U . T . . V * U . * . . j .
[10000030]    00000001  00000000  0000900d  0000900d    . . . . . .
[10000040]    00001234  00001235  00005556  00004321    4 . . . 5 . . . V U . . ! C . .
[10000050]    00000001  00000000  00000000  00000000    . . . . . .
[10000060]..[1003ffff]  00000000
```

图 4: test 虚拟

Addr	+0	+1	+2	+3	+4	+5	+6	+7
000	55AAAA55	AA5555AA	00005555	00003311	FFFFCCEF	FFFFFFFF	55AA5555	55AAAAFF
008	AB5554AA	56AAA954	2AD5552A	156AAA95	00000001	00000000	0000900D	0000900D
010	00001234	00001235	00005556	00004321	00000001	00000000	00000000	00000000

图 5: test 仿真

结果正确。

4.3 测试三：C 程序

这组测试程序机器码均由 C 编译器产生[1]。测试内容分别为 fib（斐波那契数列）、heapsort（堆排序）、quicksort（快速排序）。

虚拟机结果、仿真结果如图 6、图 7、图 8、图 9、图 10、图 11。

```
User data segment [10000000]..[10040000]
[10000000]    00000015  00000000  00000000  00000000    . . . . . .
[10000010]..[1003ffff]  00000000
```

图 6: fib 虚拟

Addr	+0	+1	+2	+3	+4	+5	+6	+7
000	00000015	00000000	00000000	00000000	00000000	00000000	00000000	00000000
008	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000

图 7: fib 仿真

```
User data segment [10000000]..[10040000]
[10000000]    00000000  00000001  00000005  0000000f    . . . . . .
[10000010]    00000016  0000001c  00000030  0000003b    . . . . . 0 . . . ; . . .
[10000020]    00000050  00000058  0000005b  0000000b    P . . . X . . . [ . . . . . .
[10000030]    00000001  1000004c  1000004e  10000050    . . . . L . . . N . . . P . . .
```

图 8: heapsort 虚拟

Addr	+0	+1	+2	+3	+4	+5	+6	+7
000	00000000	00000001	00000005	0000000F	00000016	0000001C	00000030	0000003B
008	00000050	00000058	0000005B	0000000B	00000001	00000000	00000000	00000000

图 9: heapsort 仿真

```
User data segment [10000000]..[10040000]
[10000000]    00000008  0000002c  00000034  00000037    . . . . , . . . 4 . . . 7 . . .
[10000010]    0000003e  0000004a  0000004a  0000004a    > . . . J . . . J . . . J . . .
[10000020]    00000050  0000007b  0000000a  10000040    P . . . { . . . . . . . @ . . .
```

图 10: quicksort 虚拟

Addr	+0	+1	+2	+3	+4	+5	+6	+7
000	00000008	0000002C	00000034	00000037	0000003E	0000004A	0000004A	0000004A
008	00000050	0000007B	0000000A	00000000	00000000	00000000	00000000	00000000

图 11: quicksort 仿真

后面两个测试虚拟机的结果比仿真多出来的是数据段的初始内容，不影响结果的正确性。

[1]这是在编译原理实验构造的 C 编译器，仅仅支持 C 语言的一个很小的子集。

5 下载测试

使用 download2 模板代入并下载。代入时要注意的是，要根据情况删减不用的信号。否则，实验系统报读存错误。

下载结果与仿真相同，这里只说明中断的情况。

外部中断绑定在试验台的一个按键上，按键按下时触发中断。

首先载入 test 测试程序。为了看清触发过程，启动单拍调试。按下按键不放，同时单拍，如图 12，可见 PC 变为 0x404，进入中断服务程序。

图 12: 进入中断处理

继续单拍，直到返回原程序。读内存，如图 13 红字，可见运行正常。

6 总结

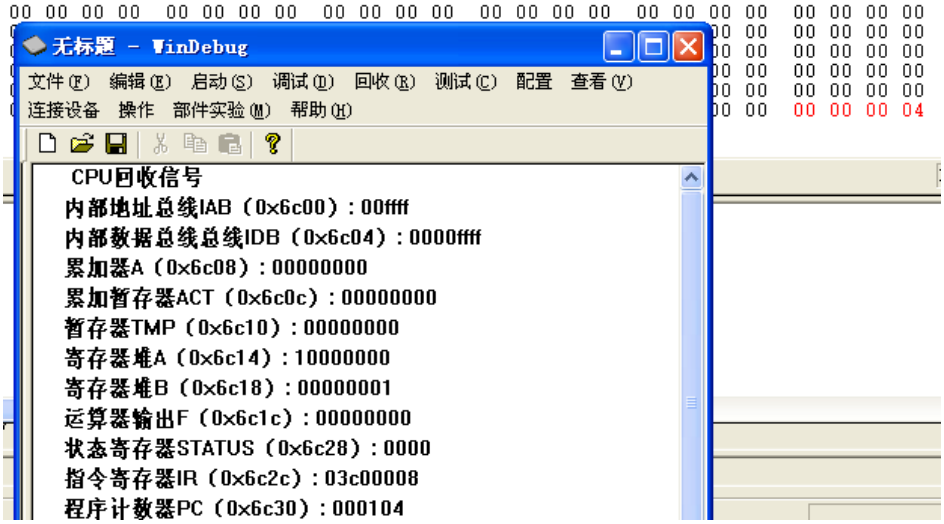
无标题 - WinDebug
文件(F) 编辑(E) 启动(S) 调试(D) 回收(R) 测试(C) 配置 查看(V)
连接设备 操作 部件实验(M) 帮助(H)
CPU回收信号
内部地址总线IAB（0x6c00）：00ffff
内部数据总线总线IDB（0x6c04）：0000ffff
累加器A（0x6c08）：00000000
累加暂存器ACT（0x6c0c）：00000000
暂存器TMP（0x6c10）：00000000
寄存器堆A（0x6c14）：10000000
寄存器堆B（0x6c18）：00000001
运算器输出F（0x6c1c）：00000000
状态寄存器STATUS（0x6c28）：0000
指令寄存器IR（0x6c2c）：03c00008
程序计数器PC（0x6c30）：000104

图 13: 中断处理结束