

第三次实验报告

20307130112 马成

一、 实现乘除法器

我分别用了三个模块来实现功能，乘法器 `multi` 可以处理 `MUL MULW` 除法器 `div` 处理 `DIV REM DIVW REMW` 除法器 `divu` 处理 `REMUW DIVU REMU DIVUW`，这些处理器都是用 64 周期进行计算，用于需要恢复初始阶段，每一次一个模块进行处理后需要一个周期进行初始化（如果不是连续请求一个模块就没事）

1. 乘法器：首先给出一个计数器 `i`，因为我们处理 64 位的数据，`i=0` 认为是初始化状态，每次如果 $(b \gg i)$ 的最低位是 1 就对结果加一个 $a \ll i$ ，最后当 `i=65` 的时候表示已经计算结束，`data_ok` 置为 1
2. 除法器 `divu`：`divu` 用于无符号除法。首先给出一个计数器 `i`，`i=0` 初始状态的时候将 `rem` 余数设为 0，开始的时候每一次将 `rem` 和 `result` 左移 1 位并加上 `a[64-i]` 的值，如果 `b > rem` 就将 `result+1`。最后可以同时得到商和余数
3. 除法器 `div`：`div` 用于有符号除法，首先给出一个计数器 `i`，`i=0` 初始状态的时候将 `rem` 余数设为 `a` 的 64 位符号位拓展，开始的时候每一次将 `result` 左移一位，`rem` 左移 1 位并加上 `a[64-i]` 的值，如果 `rem` 和 `b` 异号则两者相减，否则两者相加，如果得到的结果没有改变符号，则 `result+1`。最后可以同时得到商和余数
4. 对于 `divw, divuw, remw, remuw` 将两者的 `src` 分别处理好即可，不用修改 `div` 和 `divu` 的内容进行判断
5. 这里新增了一种状况就是 `execute` 阶段有出现了阻塞的可能，遇到乘除法的时候需要让 `decode` 和 `fetch` 阶段不动，等待 `execute` 部分 `data_ok` 为 1 的时候，具体的操作和实验二中 `mem` 部分的类似即可。

二、 Cache

1. 我实现的 `cache` 是二路组相连的结构，一共用 8 个 `cache_set` 每一个 `cache_set` 有两个 `cache_line`。在实现中，分别使用了两个 `ram` 分别储存 `data` 和 `meta`，`data` 使用了 $2^8=256$ 个储存器，每 16 个储存器表示一个 `cache_line` 的数据，`meta` 我是一个 `cache_set` 放一个 `meta`，这个 `meta` 可以同时纪录这两个 `cache_line` 的数据。
2. 替换策略就是如果有空块就优先替换空块，在 `meta` 中有一个 `u1` 的参数用于纪录应该被替换的 `cache_line` 的编号，每一个 `cache` 从主存取数据的时候将这个参数取反，由此实现交替替换。
3. 在 `cache` 中我一共有 6 个状态，他们分别是，并且使用时序逻辑考虑对于他们的转化 `COMPARETAG, READY, FETCH, WRITEBACK, FINAL, WRITEREADY`。
`COMPARETAG` 阶段主要用于比较所需要的 `tag` 和 `cache` 中已有的 `cache_line` 是否匹配，如果有匹配状态到 `ready` 阶段，否则根据块是否脏到 `writeback` 或者 `fetch` 阶段即可。`Fetch` 就是向主存取值的阶段，`cresp.last` 之前一直保持这个状态即可。`writeback` 的时候是写回阶段，在 `cresp.last` 之前保持状态，写回后进入 `fetch` 阶段即可。`Fetch` 后进入 `final` 阶段，这是必然命中直接取出对应值即可。然后进入 `ready` 阶段。为了让值真正的写入 `data_ram` 中再退出，需要写的指令多加一个 `WRITEREADY` 阶段。随后给出 `data_ok` 即可。

三、通过截图

1. Test-cache 通过截图

```
[OK] void (7ms)
[OK] reset (12ms)
[OK] fake load (95ms)
[OK] fake store (95ms)
[OK] naive (7ms)
[--] akarin~ (skipped)
[OK] strobe (10ms)
[OK] ad hoc (9ms)
[OK] pipelined (9ms)
[OK] memory cell (10ms)
[OK] memory cell array (6ms)
[OK] cmp: word (15ms)
[OK] cmp: halfword (31ms)
[OK] cmp: byte (59ms)
[OK] cmp: random (320ms)
[OK] memset (217ms)
[OK] memcpy (1248ms)
[OK] load/store repeat (193ms)
[OK] backward memset (666ms)
[OK] backward load/store (761ms)
[OK] random step (4140ms)
[OK] random load/store (5599ms)
[OK] random block load/store (3387ms)
"std::sort": bingo!
[OK] std::sort (6079ms)
"std::stable_sort": bingo!
[OK] std::stable_sort (8931ms)
"heap sort": bingo!
[OK] heap sort (18459ms)
"binary search tree": bingo!
[OK] binary search tree (12471ms)
(info) 27 tests passed.
```

2. Test-lab3

[illegible]

```

Dhrystone PASS                2 Marks
vs. 100000 Marks (i7-7700K @ 4.20GHz)

Run stream

-----
STREAM version $Revision: 5.10 $
-----
This system uses 8 bytes per array element.
-----
Array size = 2048 (elements), Offset = 0 (elements)
Memory per array = 0.0 MiB (= 0.0 GiB).
Total memory required = 0.0 MiB (= 0.0 GiB).
Each kernel will be executed 10 times.
The "best" time for each kernel (excluding the first iteration)
will be used to compute the reported bandwidth.
-----
* checktick: start=6.067862
* checktick: start=6.071985
* checktick: start=6.076045
* checktick: start=6.079955
* checktick: start=6.083698
* checktick: start=6.087604
* checktick: start=6.091731
* checktick: start=6.095690
* checktick: start=6.099713
* checktick: start=6.103688
* checktick: start=6.107559
* checktick: start=6.111435
* checktick: start=6.115553
* checktick: start=6.119667
* checktick: start=6.123668
* checktick: start=6.127789
* checktick: start=6.131991
* checktick: start=6.135826
* checktick: start=6.139949
* checktick: start=6.143945
Your clock granularity/precision appears to be 290 microseconds.
Each test below will take on the order of 89001 microseconds.
(= 306 clock ticks)
Increase the size of the arrays if this shows that
you are not getting at least 20 clock ticks per test.
-----
WARNING -- The above is only a rough guideline.
For best results, please be sure you know the
precision of your system timer.

```

```
Function      Best Rat MB/s     Avg time       Min time       Max time
Copy:        0.7         0.044599       0.044087       0.044961
Scale:       0.3         0.124320       0.123701       0.124819
Add:        0.2         0.250806       0.247526       0.254293
Triad:       0.1         0.436925       0.415687       0.476504
```

```
solution Validates: avg error less than 1.000000e-13 on all three arrays
```

```
Run Conwaygame
Play Conway's life game for 200 rounds.
seed=16304
```

```
***
***               ***
**                **
**                **
***              ***
***             ***
***            ***
***           ***
***          ***
***         ***
***        ***
***       ***
***      ***
***     ***
***    ***
***   ***
***  ***
*** ***
****
```

```
[src/cpu/cpu-exec.c,320,cpu_exec] nemu: HIT GOOD TRAP at pc = 0x000000008001e44d
[src/cpu/cpu-exec.c,321,cpu_exec] trap code:0
[src/cpu/cpu-exec.c,62,monitor_statistic] host time spent = 72964978 us
[src/cpu/cpu-exec.c,64,monitor_statistic] total guest instructions = 34539647
[src/cpu/cpu-exec.c,65,monitor_statistic] simulation frequency = 473372 instr/s
Program execution has ended. To restart the program, exit NEMU and run again.
```

```
sh: 1: spike-dasm: not found
```

```
===== Commit Group Trace (Core 0) =====
```

```
commit group [0]: pc 0080010268 cntcnt 1
commit group [1]: pc 008001026c cntcnt 1
commit group [2]: pc 0080010270 cntcnt 1
commit group [3]: pc 0080014bf8 cntcnt 1
commit group [4]: pc 0080014dfc cntcnt 1
commit group [5]: pc 0080014cf0 cntcnt 1
commit group [6]: pc 0080014ce4 cntcnt 1
commit group [7]: pc 0080014ceb cntcnt 1
commit group [8]: pc 0080014cc0 cntcnt 1
commit group [9]: pc 0080014c70 cntcnt 1
commit group [a]: pc 0080014e34 cntcnt 1
commit group [b]: pc 0080014e38 cntcnt 1
commit group [c]: pc 0080014e3c cntcnt 1
commit group [d]: pc 0080014e40 cntcnt 1
commit group [e]: pc 0080014e44 cntcnt 1
```


1