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>>i)的最低位是 1 就对结果加一个 a<<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 没别储存 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

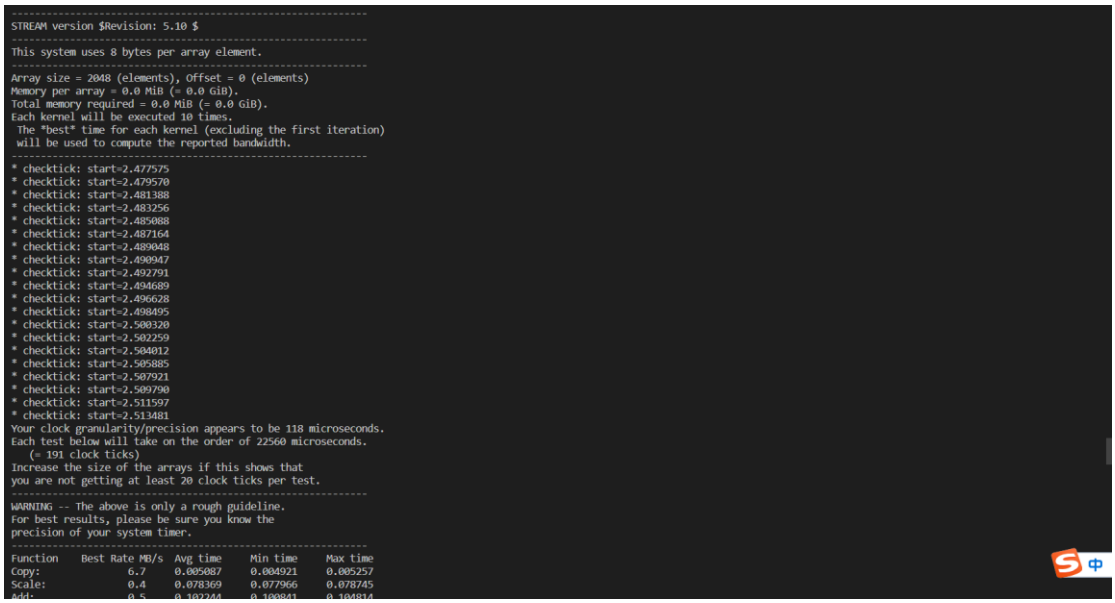
1. Test-cache 通过截图

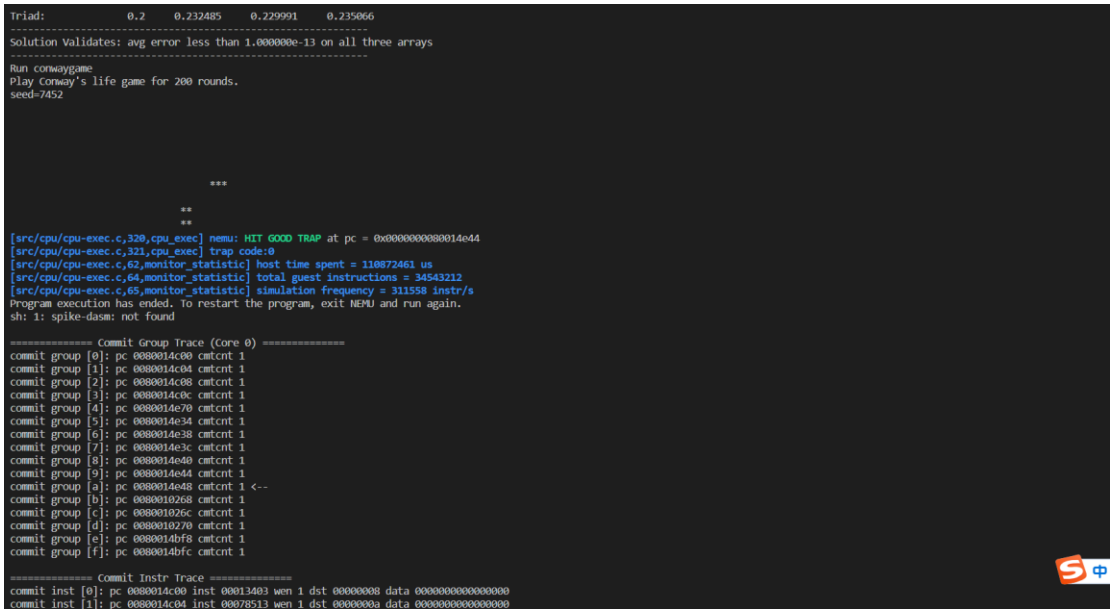
```
[OK] void (6ms)
[OK] reset (6ms)
[OK] fake load (75ms)
[OK] fake store (81ms)
[OK] naive (6ms)
[--] akarin~ (skipped)
[OK] strobe (6ms)
[OK] ad hoc (8ms)
[OK] pipelined (6ms)
[OK] memory cell (7ms)
[OK] memory cell array (9ms)
[OK] cmp: word (11ms)
[OK] cmp: halfword (19ms)
[OK] cmp: byte (43ms)
[OK] cmp: random (310ms)
[OK] memset (178ms)
[OK] memcpy (195ms)
[OK] load/store repeat (163ms)
[OK] backward memset (643ms)
[OK] backward load/store (784ms)
[OK] random step (4266ms)
[OK] random load/store (5304ms)
[OK] random block load/store (3667ms)
"std::sort": bingo!
[OK] std::sort (3441ms)
"std::stable_sort": bingo!
[OK] std::stable_sort (5185ms)
"heap sort": bingo!
[OK] heap sort (10393ms)
"binary search tree": bingo!
[OK] binary search tree (8077ms)
(info) 27 tests passed.
```

2. Test-lab3

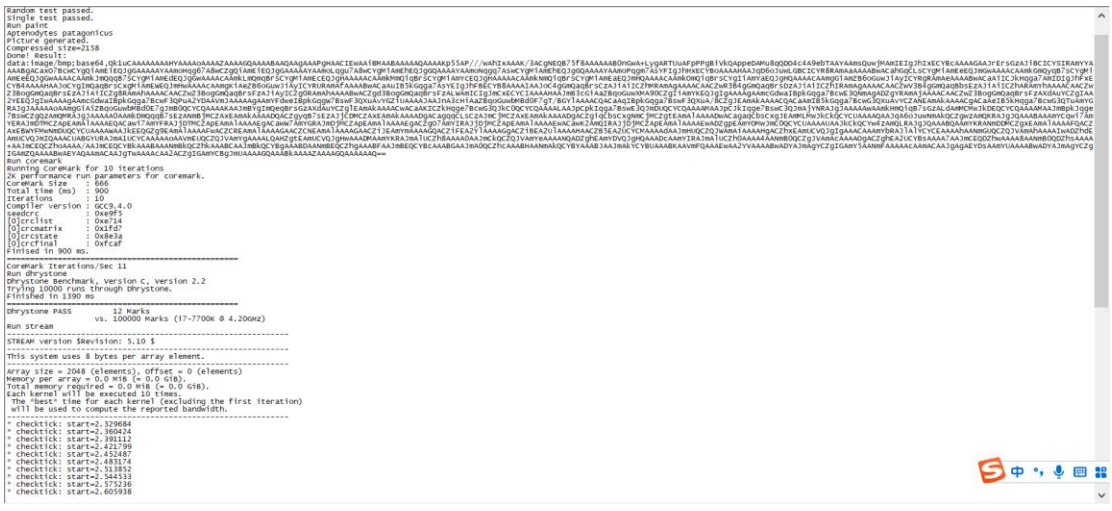
```
[WARNING] difftest store queue overflow
Random test passed.
Single test passed.
Run paint
Aptenodytes patagonicus
Picture generated.
Compressed size=2158
Done! Result:

uwjMAmIEIgJhIxECYBcAAAAGAAJrErsGzAJiBCICYSIRAmYYAAAABgACaxO7BcwCYgQiAmEiEQJgGAAAAAYAAmoHqg67A8wCZgQiAmEiEQJgGAAAAAYAAmoNqgq7AswCYgMiAmEhEQJgGQAAAAYAAmoPqg
m7AsYFIgJhHxECYBoAAAAHAAJqD6oJuwLGBCICYR8RAmAaAAAABwACahGqCLsCYgMiAmEeEQJgGwAAAACcAAmkGmQyqB7sCYgMiAmEeEQJgGwAAAACcAAmkJmQqqB7sCYgMiAmEcEQJgHAAAAAcCAAmkMmQiq
BrsCYgMiAmYcEQJgHAAAAAcCAAmkMmQiqBrsCYgMiAmEaEQJmHQAAAAcCAAmkOmQiqBrsCYgIiAmYaEQJgHQAAAAcAAmgGiAmZBGoGuwJiAyICYRgRAmAeAAAABwACaAiICJkHqga7AmIDIgJhFxECYB4AAAAHAAJoCYgImQaqBrsCxgMiAmEwEQJmHw
AAAAcAAmgKiAeZB6oGuwJiAyICYRURAmAfAAAABwACaAuIB5kGqga7AsYEIgJhFBECYB8AAAAIAAJoC4gGmQaqBrsCzAJiAiICZhMRAmAgAAAACAAC2wR3B4gGmQaqBrsDzAJiAiICZhIRAmAgAAAACAAC2wV3B4gGmQaqBbsEzAJiAiICZhARAmAYh
AAAACAAC2wZ3BogGmQaqBrsEzAJiAiICZg8RAmAhAAAACAAC2wZ3BogGmQaqBrsFzAJiAyICZg0RAmAhAAAABwACZgd3BogGmQaqBrsFzAlWAmICIgJmCxECYCIAAAAHAAJmB3cGiAaZBqoGuwXMA90CZgIiAmYKEQJgIgAAAgAAmcGdwaIBpkGqg
a7BcwE3QNmAgADZgYRAmAjAAAACAAC2wZ3BogGmQaqBrsFzAXdAuYCZgIAA2YEEQJgIwAAAAgAAmcGdwaIBpkGqgW7BswF3QXuAvYGZiUAAAAJAAJnA3cHiAaZBqoGuwbMBd0F7gT/BGYlAAAA
CQACaAqIBpkGqga7BswF3QXuA/8CZgJEAmAkAAAACQACaAmIB5kGqga7BcwG3QXuAvYCZANEAmAkAAAACgACaAeIB5kHqga7BcwG3QTuAmYGRAJgJAAAAAoAAmgGiAiZBqoGuwbMBd0E7gJmBi0QCYCQAAAAKAAJmBYgImQeqBrsGzAXdAuYCZglEAm
AkAAAACwACaAKICZkHqge7BcwG3QJkC0QCYCQAAAALAAJpCpkIqga7BswE3QJmDUQCYCQAAAAMAAJpCJkIqge7BswC3QJmAjYNRAJgJAAAAAwAAmkHmQiqB7sGZALdAmMCMwJkDEQCYCQAAAAMAAJmBpkJqge7BswCZgQzAmQMRAJgJAAAA0AAmkD
mQqqB7sEzANmBjMCZAxEAmAkAAADQACZgyqB7sEzAJjCDMCZAxEAmAkAAADgACagqqCLsCzAJmCjMCZAxEAmAkAAADgACZgiqCbsCxgHmCjMCZgtEAmAlAAAADwACagaqCbsCxgJEAmMLMwJkCkQCYCUAAAAQAAJqA6oJuwNmAkQCZgwzAmQKRA
JgJQAAABAAAmYCqwi7AmYERAJmDTMCZApEAmAlAAAAEQACawi7AmYFRAJjDTMCZApEAmAlAAAAEgACZgO7AmYIRAJjDjMCZApEAmAlAAAAEwACawCZmQIRAJjDjMCZApEAmAlAAAAEwAD7gpEAmYOPwJm
C0QCYCUAAAAUAAJkCkQCYw4zAmQLRAJgJQAAABQAAmYKRAmmDDMCZgxEAmAlAAAAFQACZAxEBwYFMwNmDUQCYCUAAAAWAAJkEEQGZg9EAmAlAAAAFwACZCREAmAlAAAAGAACZCNEAmAlAAAAGAACZiJEAmYmAAAAGQACZiFEA2YlAAAAGgACZiBEA2
UlAAAAHAACZB5EA2UCYCMAAAADAAJmHUQCZQJkAmAlAAAAHgACZhxEAmUCVQJgIgAAACAAAmYDRAJlAlYCYCEAAAAhAAmmGUQCZQJVAmYAhAAAAIwAD2hdEAmUCVQJmIQAAACUABGYURAJmA1UCYCAAAAAoAAmEUQCZQJVAmYgAAAALQAHZgtEAmUC
VQJgHwAADMAAmYRAJmAlUCZhAAAAAAAJmCkQCZQJVAmYeAAAHQADZghEAmYQVQJgIQAAADcAAmYIRAJmAlUCZhAAAAAAAAmB0QCZgJVAmAcAAAAQgACZghEA2UCYBsAAAA7AAJmCEQDZhwAAAABAAmB0QDZhsAAAA+AAJmCEQCZhoAAAA/AA
JmCEQCYBkAAAAAAmBkQCZhkAAABCAAJmBkQCYBgAAABDAAmBEQCZhgAABFAAJmBEQCYBcAAABGAAJmAQCZhcAAABHAAmAkQCYBYAAAB1AAJmAkYCYBUAAABKAAVmFQAAAEwAA2YVAAAABQAD2gIGAmYDAAHgAmY5AAmFAAAAAUAAmACAARg
AgACYAIAAmASAAJmFAAAAAUAA2YCBgJmAwADYAJmUAAAAcABGAEAARgUQAAAAUAA2YCBgJmAwADYAJmUAAAAGQAAABkAAAAZAAAAGQAAAAAQ==
Run coremark
Running CoreMark for 10 iterations
2K performance run parameters for coremark.
CoreMark Size    : 666
Total time (ms)  : 1090
Iterations       : 10
Compiler version : GCC9.4.0
seedcrc          : 0xe9f5
[0]crclist       : 0xe714
[0]crcmatrix     : 0x1fd7
[0]crcstate      : 0x8e3a
[0]crcfinal      : 0xfcaf
Finised in 1090 ms.
==========================================
CoreMark Iterations/Sec 9
Run dhrystone
Dhrystone Benchmark, Version C, Version 2.2
Trying 10000 runs through Dhrystone.
Finished in 2409 ms
==========================================
Dhrystone PASS        7 Marks
                vs. 100000 Marks (i7-7700K @ 4.20GHz)
```


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=2.477575
* checktick: start=2.479570
* checktick: start=2.481388
* checktick: start=2.483256
* checktick: start=2.485088
* checktick: start=2.487164
* checktick: start=2.489048
* checktick: start=2.490947
* checktick: start=2.492791
* checktick: start=2.494689
* checktick: start=2.496628
* checktick: start=2.498495
* checktick: start=2.500320
* checktick: start=2.502259
* checktick: start=2.504012
* checktick: start=2.505885
* checktick: start=2.507921
* checktick: start=2.509790
* checktick: start=2.511597
* checktick: start=2.513481
Your clock granularity/precision appears to be 118 microseconds.
Each test below will take on the order of 22560 microseconds.
(= 191 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 Rate MB/s Avg time Min time Max time
Copy: 6.7 0.005087 0.004921 0.005257
Scale: 0.4 0.078369 0.077966 0.078745


Triad: 0.2 0.232485 0.229991 0.235066
Solution Validates: avg error less than 1.000000e-13 on all three arrays
Run conwaygame
Play Conway's life game for 200 rounds.
seed=7452
[src/cpu/cpu-exec.c,320,cpu_exec] nemu: HIT GOOD TRAP at pc = 0x0000000080014e44
[src/cpu/cpu-exec.c,321,cpu_exec] trap code:0
[src/cpu/cpu-exec.c,62,monitor_statistic] host time spent = 110872461 us
[src/cpu/cpu-exec.c,64,monitor_statistic] total guest instructions = 34543212
[src/cpu/cpu-exec.c,65,monitor_statistic] simulation frequency = 311558 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 0080014c00 cmtcnt 1
commit group [1]: pc 0080014c04 cmtcnt 1
commit group [2]: pc 0080014c08 cmtcnt 1
commit group [3]: pc 0080014c0c cmtcnt 1
commit group [4]: pc 0080014e70 cmtcnt 1
commit group [5]: pc 0080014e34 cmtcnt 1
commit group [6]: pc 0080014e38 cmtcnt 1
commit group [7]: pc 0080014e3c cmtcnt 1
commit group [8]: pc 0080014e40 cmtcnt 1
commit group [9]: pc 0080014e44 cmtcnt 1
commit group [a]: pc 0080014e48 cmtcnt 1 <--
commit group [b]: pc 0080010268 cmtcnt 1
commit group [c]: pc 008001026c cmtcnt 1
commit group [d]: pc 0080010270 cmtcnt 1
commit group [e]: pc 0080014bf8 cmtcnt 1
commit group [f]: pc 0080014bfc cmtcnt 1
============= Commit Instr Trace =============
commit inst [0]: pc 0080014c00 inst 00013403 wen 1 dst 00000008 data 0000000000000000
commit inst [1]: pc 0080014c04 inst 00078513 wen 1 dst 0000000a data 0000000000000000

3. 上板通过截图


Random test passed.
Single test passed.
Run paint
Aptenodytes patagonicus
Picture generated.
Compressed size=2158
Done! Result:
Running CoreMark for 10 iterations
2K performance run parameters for coremark.
CoreMark Size : 666
Total time (ms) : 900
Iterations : 10
Compiler version : GCC9.4.0
seedcrc : 0xe9f5
[0]crclist : 0xe714
[0]crcmatrix : 0x1fd7
[0]crcstate : 0x8e3a
[0]crcfinal : 0xfcaf
Finished in 900 ms.
CoreMark Iterations/Sec 11
Run dhrystone
DPrystone Benchmark, Version C, Version 2.2
Trying 10000 runs through Dhrystone.
Finished in 1390 ms
DPrystone PASS 12 Marks
vs. 100000 Marks (17-7700k @ 4.2GHz)
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=2.329684
* checktick: start=2.360424
* checktick: start=2.391112
* checktick: start=2.421799
* checktick: start=2.452487
* checktick: start=2.483174
* checktick: start=2.513852
* checktick: start=2.544533
* checktick: start=2.575236
* checktick: start=2.605938

```
-------------------------------------------------------------
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=2.329684
* checktick: start=2.360424
* checktick: start=2.391112
* checktick: start=2.421799
* checktick: start=2.452487
* checktick: start=2.483174
* checktick: start=2.513852
* checktick: start=2.544533
* checktick: start=2.575236
* checktick: start=2.605938
* checktick: start=2.636626
* checktick: start=2.667313
* checktick: start=2.697991
* checktick: start=2.728669
* checktick: start=2.759346
* checktick: start=2.790049
* checktick: start=2.820751
* checktick: start=2.851429
* checktick: start=2.882106
* checktick: start=2.912809
Your clock granularity/precision appears to be 81 microseconds.
Each test below will take on the order of 17778 microseconds.
   (= 219 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 Rate MB/s  Avg time     Min time     Max time
Copy:            11.6     0.002829     0.002829     0.002829
Scale:            0.6     0.058366     0.058337     0.058421
Add:              1.1     0.047658     0.046766     0.049443
Triad:            0.4     0.119064     0.118144     0.120905
-------------------------------------------------------------
Solution validates: avg error less than 1.000000e-13 on all three arrays
-------------------------------------------------------------
Run conwaygame
Play Conway's life game for 200 rounds.
seed=6781



          **
         *  *
          **


Exit with code = 0
```