

计算机系统结构实验 - Archlab 报告

w1049

目录

1 实验简介	2
1.1 实验任务	2
1.2 实验用指令	2
2 实验过程	2
2.1 初始状态	2
2.2 指令替换	2
2.3 消除 Load-Use 冒险	2
2.4 循环展开	3
2.4.1 两路展开	3
2.4.2 八路展开	3
2.5 循环展开剩余部分处理	3
2.5.1 减少地址计算	3
2.5.2 优化跳转	3
2.5.3 迎合分支预测	4
2.5.4 进一步减少 Load-Use 冒险	4
2.6 去除 xor 指令	6
3 实验结果	6
3.1 可行性测试	6
3.2 CPE 及得分	7
A 附录	8
A.1 代码	8

1 实验简介

1.1 实验任务

本次实验基于 Y86-64 指令集模拟器，要求对 `ncopy.js` 和 `pipe-full.hcl` 进行修改，以使 `ncopy.js` 的运行速度尽可能地快。

我在不修改 `pipe-full.hcl` 的情况下，通过循环展开、指令替换、指令顺序调整等方式，得到了 $\text{AverageCPE} = 7.47$ 的结果。

1.2 实验用指令

实验中，在 `sim/pipe` 目录下主要使用了以下指令：

- 重新构建仿真器：

```
make clean && make psim VERSION=full
```

- 测试 `ncopy.js` 的正确性并计算 CPE：

```
make drivers && ./correctness.pl && ./benchmark.pl
```

- 检查代码长度：

```
../misc/yas ./ncopy.js && ./check-len.pl < ncopy.yo
```

2 实验过程

2.1 初始状态

初始状态测试得到 $\text{AverageCPE} = 15.18$ 。

2.2 指令替换

根据提示，助教为仿真器增加了 `iaddq` 指令，在重新构建仿真器后可用。代码中的 `irmovq`、`addq` 等指令可以使用 `iaddq` 替换，并去除对 `%r10` 的使用，节约指令条数。替换后得到 $\text{AverageCPE} = 12.70$ ，完整代码见附录代码 2。

2.3 消除 Load-Use 冒险

注意到 `mrmovq` 与 `mrmovq` 紧邻，存在 Load-Use 冒险，会造成一个周期的浪费，可以在两者之间插入一条无关指令以消除冒险。这一步得到的 $\text{AverageCPE} = 11.70$ ，完整代码见附录代码 3。

2.4 循环展开

通过循环展开可以减少 `src++`, `dst++`, `len--` 指令的数目, 达到减少 CPE 的目的。如果处理器支持多发射、乱序执行, 循环展开还为处理器的乱序提供条件; 但对于本次实验, 循环展开只有减少地址、长度计算指令的作用。

2.4.1 两路展开

首先尝试简单的两路展开, 观察循环展开是否有效。在两路展开的主循环里, 注意尽量避免 `mrmovq` 与 `rmmovq` 之间产生冒险。长度的计算与判断可能有多种方式, 我选择在最开始判断是否进入循环, 并把 `len -= 2` 放回原来的位置——循环结尾处, 计算的同时还能得到判断是否跳转的条件码, 省去 `andq len, len` 的步骤。

对于剩余部分 (`len < 2`), 只有 `len = 1` 与 `len = 0` 两种情况, 判断即可。在刚开始, 我们已经把 `len` 减去了 2, 此时再加上 1 就得到了 `len - 1`。在对最后一个元素进行复制时, 数据冒险是难以避免的, 不过只会发生一次, 影响不大。这一步得到的 `AverageCPE = 8.82`, `Score 33.5/60.0`, 效果非常明显。完整代码见附录代码 4。

2.4.2 八路展开

了解了循环展开的效果之后, 我们直接进行八路展开。如果能把剩余 0 至 7 个元素的情况全部单独写出来, 可以较大限度减少数据冒险, 但是这样肯定会超出 1000 bytes 的限制。尝试再加一个循环处理, 一个一个复制剩余元素。这么做的缺点是地址、长度计算指令较多。得到的结果是 `AverageCPE = 8.22`, `Score 45.7/60.0`。应该可以把剩余元素的循环也做展开, 进一步减少计算指令, 但我没有尝试。完整代码见附录代码 4。

2.5 循环展开剩余部分处理

2.5.1 减少地址计算

为了减少地址的计算, 可以预先写好对 0 至 7 单个元素的处理, 直接硬编码地址, 程序运行时依次复制 0、8、16……处的元素, 直到长度减为 0。同时, 为了减少数据冒险, 可以将 `mrmovq` 提前到上一次的两个传送指令之间; 这样做可能导致非法的地址访问, 不过非法的内容实际不会被使用。这一步得到的结果是 `AverageCPE = 8.01`, `Score 49.8/60.0`。完整代码见附录代码 6。

2.5.2 优化跳转

优化了地址计算后, 跳转还是一条计算指令 + 一条跳转指令的形式, 想要覆盖 0 至 7 需要把长度一点点减到 0。但是, 条件码只由计算指令设置, 跳转并不会改变条件码, 所以一次计算最多可以跳转三次 (大于、等于、小于), 更加节约计算指令。

我原本的思路是按照二进制的形式拆分剩余长度，先判断 `len` 是否大于等于 4，若是就同时复制前四个元素。在一次计算 + 三次跳转的前提下，我改为了计算 `len - 3`：

- `len - 3 = 0` 长度是 3，直接复制 3 个即可结束
- `len - 3 < 0` 长度可能为 0 1 2，再根据 `len` 与 1 的大小关系就可以区分开
- `len - 3 > 0` 长度可能为 4 5 6 7，先复制前 4 个元素，再判断长度与 6 的关系

在代码实现时，我感觉大块的复制（比如一次 4 个）更利于减少数据冒险，因为有足够多的指令用以改变顺序。因此我把 `R4,R3`（即前 4 个、前 3 个）的复制单独写了出来，而 `R7,R6,R5` 与 `R2,R1` 写到一起。我原想把更多的大块复制拆分出来，但受限于 1000 bytes，只能选了（我自认为，未比较测试）更有效的几组。

同时，有些跳转是不必要的，比如 `je` `jle` `jbe` 三者按顺序判断，则最后一次判断不必要，可以调整代码块的位置减少一些跳转；`jmp Done` 不如直接使用 `ret`。

我并没有测试过其他分组方式对效率的影响，只是按感觉划分，或许在测试数据下存在最优解（考虑 0 至 7 各数出现的次数、对平均值的影响）。经过此步，结果来到了 `AverageCPE=7.66`，`Score 56.8/60.0`，完整代码见附录代码 7。

2.5.3 迎合分支预测

查看 `pipe-full.hcl`，可以看到如下代码：

```

1 # Predict next value of PC
2 word f_predPC = [
3     f_icode in { IJXX, ICALL } : f_valC;
4     1 : f_valP;
5 ];

```

查看书中对 `valC` 和 `valP` 的定义（或是直接看“预测下一个 PC”小节），可知仿真器默认的分支预测策略是“总是预测分支发生”。我们可以修改代码来增加更高级的分支预测功能，或是迎合这种预测策略，把更易发生的情况放到分支中、调整指令顺序，使程序在测试数据下表现更好。

我选择不修改流水线的情况下迎合分支预测。我没有从数学、数据方面找出最优解，而是简单地测试了几种顺序，选择了表现较好的一种。这一步修改不多，结果是 `AverageCPE = 7.58`，`Score 58.4/60.0`，完整代码见附录代码 8。

2.5.4 进一步减少 Load-Use 冒险

上文说到在大块的复制中，很容易减少数据冒险；但对于 `R7,R6,R5` 这类，有一组 `mrmovq`, `rmmovq` 被多个 R 共用的情况，就难找到填充空隙的指令了。比如 `R2,R1` 部分的代码，其中存在两次 Load-Use 冒险：

```

1 R2:
2     mrmovq 8(%rdi), %r11
3     rmmovq %r11, 8(%rsi)    # 冒险

```

```

4    andq %r11, %r11
5    jle R2N1
6    iaddq $1, %rax
7 R2N1:
8 R1:
9    mrmovq (%rdi), %r12
10   rmmovq %r12, (%rsi)    # 冒险
11   andq %r12, %r12
12   jle R2N2
13   iaddq $1, %rax
14 R2N2:
15   # ret

```

是否有可能减少冒险呢？只能把目光转向 `jle` 指令。上文说到条件码只会由计算指令设置，不会被其他指令影响，所以可以把 `jle` 放到下一次复制的空隙中（或者相对地说，把 `mrmovq` 提前了）。对于 `R2` 来说，这样减少了一次数据冒险（第一次难以避免）；对于 `R1` 来说，数据冒险虽然减少了一次，但多出一 `jle`，并不能减少第一次数据冒险。

```

1 R2:
2    mrmovq 8(%rdi), %r11
3    rmmovq %r11, 8(%rsi)
4    andq %r11, %r11
5 R1:
6    mrmovq (%rdi), %r12
7    jle R2N1    # jle 在空隙中，R1 在此必须跳转
8    iaddq $1, %rax # R2 的计数指令
9 R2N1:
10   rmmovq %r12, (%rsi)
11   andq %r12, %r12
12   jle R2N2
13   iaddq $1, %rax
14 R2N2:
15   # ret

```

这样做是否会破坏程序的正确性呢？有可能，因为对于 `R1`，必须 `jle` 处跳转，防止执行 `R2` 用于计数的指令。好在 `R1` 本身也是由其他位置跳转而来，只要 `R1` 是由 `j1` 或 `je` 跳转而来，就可以保证在此处也会跳转。好消息是，这种修改涉及到 `R6, R5, R1`，它们确实满足条件：`jg` 跳转到的肯定是一组中的第一种情况，其他情况均是由 `j1` 或 `je` 跳转而来。

这一步只需要修改代码的最后部分。结果是 $\text{AverageCPE} = 7.55$ ，Score 59.0/60.0，完整代码见附录代码 9。

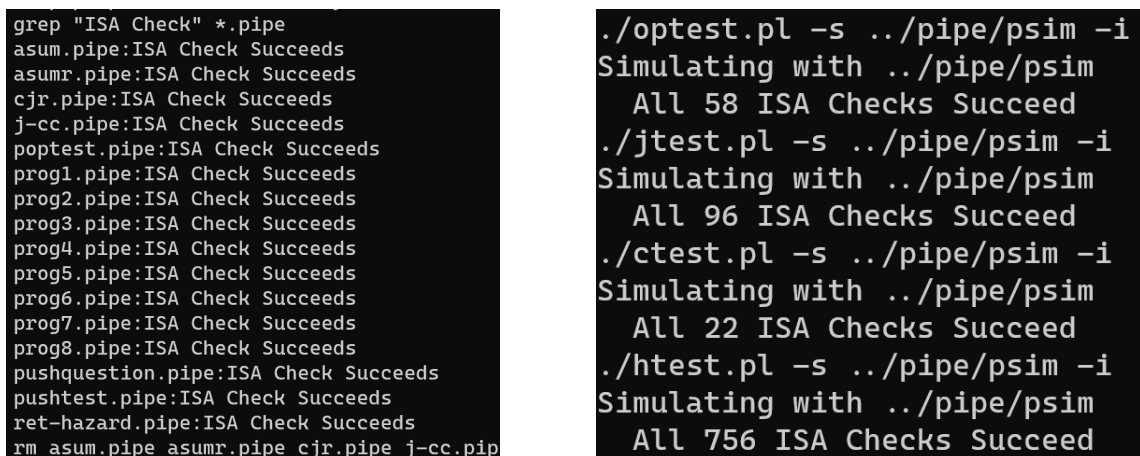
2.6 去除 xor 指令

最后，还有一个不怎么厚道的优化。第一条指令 xor 作用是置 0，但是仿真器默认寄存器的值是 0，所以这一条可以省去。这么做不太符合函数的本意，但是确实可以减少指令数，并且能通过仿真器的测试。由于每个测试数据都会用到这条指令，删去它的效果比较明显，结果是 $\text{AverageCPE} = 7.47$ ，Score 60.0/60.0。

3 实验结果

3.1 可行性测试

由于未修改流水线代码，前两个可行性测试当然可以通过，见图 1。



```

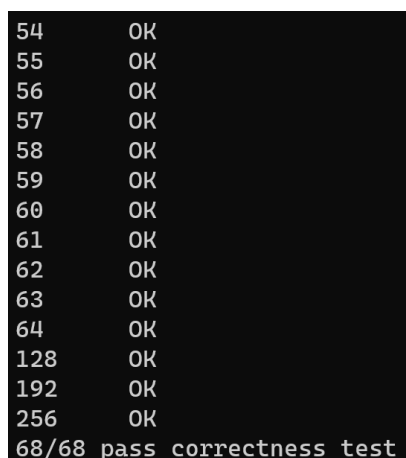
grep "ISA Check" *.pipe
asum.pipe:ISA Check Succeeds
asumr.pipe:ISA Check Succeeds
cjr.pipe:ISA Check Succeeds
j-cc.pipe:ISA Check Succeeds
poptest.pipe:ISA Check Succeeds
prog1.pipe:ISA Check Succeeds
prog2.pipe:ISA Check Succeeds
prog3.pipe:ISA Check Succeeds
prog4.pipe:ISA Check Succeeds
prog5.pipe:ISA Check Succeeds
prog6.pipe:ISA Check Succeeds
prog7.pipe:ISA Check Succeeds
prog8.pipe:ISA Check Succeeds
pushquestion.pipe:ISA Check Succeeds
pushtest.pipe:ISA Check Succeeds
ret-hazard.pipe:ISA Check Succeeds
rm asum.pipe asumr.pipe cjr.pipe j-cc.pip

./optest.pl -s ../pipe/psim -i
Simulating with ../pipe/psim
All 58 ISA Checks Succeed
./jtest.pl -s ../pipe/psim -i
Simulating with ../pipe/psim
All 96 ISA Checks Succeed
./ctest.pl -s ../pipe/psim -i
Simulating with ../pipe/psim
All 22 ISA Checks Succeed
./htest.pl -s ../pipe/psim -i
Simulating with ../pipe/psim
All 756 ISA Checks Succeed

```

图 1: 可行性测试 1、2

在代码编写过程中，我已经使用了 ./correctness.pl 保证代码的正确性，所以第三个正确性测试也可以通过，见图 2。



```

54      OK
55      OK
56      OK
57      OK
58      OK
59      OK
60      OK
61      OK
62      OK
63      OK
64      OK
128     OK
192     OK
256     OK
68/68 pass correctness test

```

图 2: 可行性测试 3

还可以查看最终汇编版本程序的长度，见图 3。

```
ncopy length = 986 bytes
```

图 3: 长度测试

3.2 CPE 及得分

最终版本的程序 AverageCPE = 7.47, Score 60.0/60.0, 见图4。

ncopy					
0	21		33	217	6.58
1	21	21.00	34	227	6.68
2	31	15.50	35	230	6.57
3	34	11.33	36	239	6.64
4	43	10.75	37	247	6.68
5	51	10.20	38	251	6.61
6	55	9.17	39	260	6.67
7	64	9.14	40	265	6.62
8	73	9.12	41	265	6.46
9	73	8.11	42	275	6.55
10	83	8.30	43	278	6.47
11	86	7.82	44	287	6.52
12	95	7.92	45	295	6.56
13	103	7.92	46	299	6.50
14	107	7.64	47	308	6.55
15	116	7.73	48	313	6.52
16	121	7.56	49	313	6.39
17	121	7.12	50	323	6.46
18	131	7.28	51	326	6.39
19	134	7.05	52	335	6.44
20	143	7.15	53	343	6.47
21	151	7.19	54	347	6.43
22	155	7.05	55	356	6.47
23	164	7.13	56	361	6.45
24	169	7.04	57	361	6.33
25	169	6.76	58	371	6.40
26	179	6.88	59	374	6.34
27	182	6.74	60	383	6.38
28	191	6.82	61	391	6.41
29	199	6.86	62	395	6.37
30	203	6.77	63	404	6.41
31	212	6.84	64	409	6.39
32	217	6.78	Average CPE 7.47		
			Score 60.0/60.0		

图 4: CPE 及得分情况

A 附录

A.1 代码

代码 1: 初始状态

```

1      # Loop header
2      xorq %rax, %rax      # count = 0;
3      andq %rdx, %rdx      # len <= 0?
4      jle Done             # if so, goto Done:
5
6 Loop:
7      mrmovq (%rdi), %r10 # read val from src...
8      rmmovq %r10, (%rsi) # ...and store it to dst
9      andq %r10, %r10      # val <= 0?
10     jle Npos             # if so, goto Npos:
11     irmovq $1, %r10
12     addq %r10, %rax       # count++
13 Npos:
14     irmovq $1, %r10
15     subq %r10, %rdx       # len--
16     irmovq $8, %r10
17     addq %r10, %rdi       # src++
18     addq %r10, %rsi       # dst++
19     andq %rdx, %rdx       # len > 0?
20     jg Loop              # if so, goto Loop:

```

代码 2: 指令替换

```

1      # Loop header
2      xorq %rax, %rax      # count = 0;
3      andq %rdx, %rdx      # len <= 0?
4      jle Done             # if so, goto Done:
5
6 Loop:
7      mrmovq (%rdi), %r10 # read val from src...
8      rmmovq %r10, (%rsi) # ...and store it to dst
9      andq %r10, %r10      # val <= 0?
10     jle Npos             # if so, goto Npos:
11     iaddq $1, %rax        # count++
12 Npos:
13     iaddq $-1, %rdx       # len--
14     iaddq $8, %rdi        # src++
15     iaddq $8, %rsi        # dst++
16     andq %rdx, %rdx       # len > 0?
17     jg Loop              # if so, goto Loop:

```

代码 3: 消除 Load-Use 冒险

```

1  xorq %rax, %rax      # count = 0;
2  andq %rdx, %rdx      # len <= 0?
3  jle Done             # if so, goto Done:
4
5  Loop:
6  mrmovq (%rdi), %r10   # read val from src...
7  iaddq $-1, %rdx       # len--
8  rmmovq %r10, (%rsi)   # ...and store it to dst
9  andq %r10, %r10      # val <= 0?
10 jle Npos             # if so, goto Npos:
11 iaddq $1, %rax        # count++
12 Npos:
13 iaddq $8, %rdi        # src++
14 iaddq $8, %rsi        # dst++
15 andq %rdx, %rdx      # len > 0?
16 jg Loop              # if so, goto Loop:

```

代码 4: 两路展开

```

1  xorq %rax, %rax      # count = 0;
2  iaddq $-2, %rdx      # len -= 2
3  jl Remain           # if len < 2, goto Remain
4
5  Loop:
6  mrmovq (%rdi), %r10   # read val from src...
7  mrmovq 8(%rdi), %r11   # read val from src...
8  rmmovq %r10, (%rsi)   # ...and store it to dst
9  rmmovq %r11, 8(%rsi)  # ...and store it to dst
10 andq %r10, %r10      # val <= 0?
11 jle Npos             # if so, goto Npos:
12 iaddq $1, %rax        # count++
13 Npos:
14 andq %r11, %r11       # val <= 0?
15 jle Npos2            # if so, goto Npos2:
16 iaddq $1, %rax        # count++
17 Npos2:
18 iaddq $16, %rdi       # src += 2
19 iaddq $16, %rsi       # dst += 2
20 iaddq $-2, %rdx       # len -= 2
21 jge Loop             # goto Loop:
22
23 Remain:
24 iaddq $1, %rdx        # len - 1
25 jl Done              # len < 1, goto Done
26 mrmovq (%rdi), %r10
27 rmmovq %r10, (%rsi)
28 andq %r10, %r10

```

```

29     jle Done
30     iaddq $1, %rax
31     # ret

```

代码 5: 八路展开

```

1     xorq %rax, %rax      # count = 0;
2     iaddq $-8, %rdx      # len -= 8
3     jl Remain           # if len < 8, goto Remain
4
5 Loop:
6     mrmovq (%rdi), %r10   # read val from src...
7     mrmovq 8(%rdi), %r11  # read val from src...
8     mrmovq 16(%rdi), %r12 # read val from src...
9     mrmovq 24(%rdi), %r13 # read val from src...
10    mrmovq 32(%rdi), %r14 # read val from src...
11    mrmovq 40(%rdi), %r9   # read val from src...
12    mrmovq 48(%rdi), %r8   # read val from src...
13    mrmovq 56(%rdi), %rcx  # read val from src...
14    rmmovq %r10, (%rsi)    # ...and store it to dst
15    rmmovq %r11, 8(%rsi)   # ...and store it to dst
16    rmmovq %r12, 16(%rsi)  # ...and store it to dst
17    rmmovq %r13, 24(%rsi)  # ...and store it to dst
18    rmmovq %r14, 32(%rsi)  # ...and store it to dst
19    rmmovq %r9, 40(%rsi)   # ...and store it to dst
20    rmmovq %r8, 48(%rsi)   # ...and store it to dst
21    rmmovq %rcx, 56(%rsi)  # ...and store it to dst
22    andq %r10, %r10       # val <= 0?
23    jle Npos              # if so, goto Npos:
24    iaddq $1, %rax        # count++
25 Npos:
26    andq %r11, %r11       # val <= 0?
27    jle Npos2             # if so, goto Npos2:
28    iaddq $1, %rax        # count++
29 Npos2:
30    andq %r12, %r12       # val <= 0?
31    jle Npos3             # if so, goto Npos3:
32    iaddq $1, %rax        # count++
33 Npos3:
34    andq %r13, %r13       # val <= 0?
35    jle Npos4             # if so, goto Npos4:
36    iaddq $1, %rax        # count++
37 Npos4:
38    andq %r14, %r14       # val <= 0?
39    jle Npos5             # if so, goto Npos5:
40    iaddq $1, %rax        # count++
41 Npos5:
42    andq %r9, %r9         # val <= 0?
43    jle Npos6             # if so, goto Npos6:

```

```

44     iaddq $1, %rax      # count++
45 Npos6:
46     andq %r8, %r8      # val <= 0?
47     jle Npos7          # if so, goto Npos7:
48     iaddq $1, %rax      # count++
49 Npos7:
50     andq %rcx, %rcx     # val <= 0?
51     jle Npos8          # if so, goto Npos8:
52     iaddq $1, %rax      # count++
53 Npos8:
54     iaddq $64, %rdi     # src += 8
55     iaddq $64, %rsi     # dst += 8
56     iaddq $-8, %rdx     # len -= 8
57     jge Loop           # goto Loop:
58
59 Remain:
60     iaddq $8, %rdx      # len
61     jle Done
62 RLoop:
63     mrmovq (%rdi), %r10
64     iaddq $8, %rdi      # src++
65     rmmovq %r10, (%rsi)
66     andq %r10, %r10
67     jle RN
68     iaddq $1, %rax
69 RN:
70     iaddq $8, %rsi      # dst++
71     iaddq $-1, %rdx     # len--
72     jg RLoop

```

代码 6: 减少地址计算

```

1     xorq %rax, %rax     # count = 0;
2     iaddq $-8, %rdx     # len -= 8
3     jl Remain          # if len < 8, goto Remain
4
5 Loop:
6     mrmovq (%rdi), %r10 # read val from src...
7     mrmovq 8(%rdi), %r11 # read val from src...
8     mrmovq 16(%rdi), %r12 # read val from src...
9     mrmovq 24(%rdi), %r13 # read val from src...
10    mrmovq 32(%rdi), %r14 # read val from src...
11    mrmovq 40(%rdi), %r9  # read val from src...
12    mrmovq 48(%rdi), %r8  # read val from src...
13    mrmovq 56(%rdi), %rcx # read val from src...
14    rmmovq %r10, (%rsi)   # ...and store it to dst
15    rmmovq %r11, 8(%rsi)  # ...and store it to dst
16    rmmovq %r12, 16(%rsi) # ...and store it to dst
17    rmmovq %r13, 24(%rsi) # ...and store it to dst

```

```

18    rmmovq %r14, 32(%rsi)      # ...and store it to dst
19    rmmovq %r9, 40(%rsi)      # ...and store it to dst
20    rmmovq %r8, 48(%rsi)      # ...and store it to dst
21    rmmovq %rcx, 56(%rsi)     # ...and store it to dst
22    andq %r10, %r10           # val <= 0?
23    jle Npos                  # if so, goto Npos:
24    iaddq $1, %rax            # count++
25 Npos:
26    andq %r11, %r11           # val <= 0?
27    jle Npos2                 # if so, goto Npos2:
28    iaddq $1, %rax            # count++
29 Npos2:
30    andq %r12, %r12           # val <= 0?
31    jle Npos3                 # if so, goto Npos3:
32    iaddq $1, %rax            # count++
33 Npos3:
34    andq %r13, %r13           # val <= 0?
35    jle Npos4                 # if so, goto Npos4:
36    iaddq $1, %rax            # count++
37 Npos4:
38    andq %r14, %r14           # val <= 0?
39    jle Npos5                 # if so, goto Npos5:
40    iaddq $1, %rax            # count++
41 Npos5:
42    andq %r9, %r9             # val <= 0?
43    jle Npos6                 # if so, goto Npos6:
44    iaddq $1, %rax            # count++
45 Npos6:
46    andq %r8, %r8             # val <= 0?
47    jle Npos7                 # if so, goto Npos7:
48    iaddq $1, %rax            # count++
49 Npos7:
50    andq %rcx, %rcx           # val <= 0?
51    jle Npos8                 # if so, goto Npos8:
52    iaddq $1, %rax            # count++
53 Npos8:
54    iaddq $64, %rdi            # src += 8
55    iaddq $64, %rsi            # dst += 8
56    iaddq $-8, %rdx            # len -= 8
57    jge Loop                  # goto Loop:
58
59 Remain:
60    iaddq $8, %rdx             # len
61    jle Done
62    mrmovq (%rdi), %r10
63    mrmovq 8(%rdi), %r11
64    rmmovq %r10, (%rsi)
65    andq %r10, %r10
66    jle RN1

```

```

67     iaddq $1, %rax
68 RN1:
69     iaddq $-1, %rdx
70     jle Done
71     mrmovq 16(%rdi), %r12
72     rmmovq %r11, 8(%rsi)
73     andq %r11, %r11
74     jle RN2
75     iaddq $1, %rax
76 RN2:
77     iaddq $-1, %rdx
78     jle Done
79     mrmovq 24(%rdi), %r13
80     rmmovq %r12, 16(%rsi)
81     andq %r12, %r12
82     jle RN3
83     iaddq $1, %rax
84 RN3:
85     iaddq $-1, %rdx
86     jle Done
87     mrmovq 32(%rdi), %r14
88     rmmovq %r13, 24(%rsi)
89     andq %r13, %r13
90     jle RN4
91     iaddq $1, %rax
92 RN4:
93     iaddq $-1, %rdx
94     jle Done
95     mrmovq 40(%rdi), %r9
96     rmmovq %r14, 32(%rsi)
97     andq %r14, %r14
98     jle RN5
99     iaddq $1, %rax
100 RN5:
101     iaddq $-1, %rdx
102     jle Done
103     mrmovq 48(%rdi), %r8
104     rmmovq %r9, 40(%rsi)
105     andq %r9, %r9
106     jle RN6
107     iaddq $1, %rax
108 RN6:
109     iaddq $-1, %rdx
110     jle Done
111     mrmovq 56(%rdi), %rcx
112     rmmovq %r8, 48(%rsi)
113     andq %r8, %r8
114     jle RN7
115     iaddq $1, %rax

```

```

116 RN7:
117     iaddq $-1, %rdx
118     jle Done
119     rmmovq %rcx, 56(%rsi)
120     andq %rcx, %rcx
121     jle RN8
122     iaddq $1, %rax
123 RN8:
124     # ret

```

代码 7: 优化跳转

```

1     xorq %rax, %rax      # count = 0;
2     iaddq $-8, %rdx      # len -= 8
3     jl Remain           # if len < 8, goto Remain
4
5 Loop:
6     mrmovq (%rdi), %r10   # read val from src...
7     mrmovq 8(%rdi), %r11  # read val from src...
8     mrmovq 16(%rdi), %r12 # read val from src...
9     mrmovq 24(%rdi), %r13 # read val from src...
10    mrmovq 32(%rdi), %r14 # read val from src...
11    mrmovq 40(%rdi), %r9   # read val from src...
12    mrmovq 48(%rdi), %r8   # read val from src...
13    mrmovq 56(%rdi), %rcx  # read val from src...
14    rmmovq %r10, (%rsi)    # ...and store it to dst
15    rmmovq %r11, 8(%rsi)   # ...and store it to dst
16    rmmovq %r12, 16(%rsi)  # ...and store it to dst
17    rmmovq %r13, 24(%rsi)  # ...and store it to dst
18    rmmovq %r14, 32(%rsi)  # ...and store it to dst
19    rmmovq %r9, 40(%rsi)   # ...and store it to dst
20    rmmovq %r8, 48(%rsi)   # ...and store it to dst
21    rmmovq %rcx, 56(%rsi)  # ...and store it to dst
22    andq %r10, %r10        # val <= 0?
23    jle Npos              # if so, goto Npos:
24    iaddq $1, %rax         # count++
25 Npos:
26    andq %r11, %r11        # val <= 0?
27    jle Npos2             # if so, goto Npos2:
28    iaddq $1, %rax         # count++
29 Npos2:
30    andq %r12, %r12        # val <= 0?
31    jle Npos3             # if so, goto Npos3:
32    iaddq $1, %rax         # count++
33 Npos3:
34    andq %r13, %r13        # val <= 0?
35    jle Npos4             # if so, goto Npos4:
36    iaddq $1, %rax         # count++
37 Npos4:

```

```

38     andq %r14, %r14      # val <= 0?
39     jle Npos5            # if so, goto Npos5:
40     iaddq $1, %rax       # count++
41 Npos5:
42     andq %r9, %r9        # val <= 0?
43     jle Npos6            # if so, goto Npos6:
44     iaddq $1, %rax       # count++
45 Npos6:
46     andq %r8, %r8        # val <= 0?
47     jle Npos7            # if so, goto Npos7:
48     iaddq $1, %rax       # count++
49 Npos7:
50     andq %rcx, %rcx      # val <= 0?
51     jle Npos8            # if so, goto Npos8:
52     iaddq $1, %rax       # count++
53 Npos8:
54     iaddq $64, %rdi      # src += 8
55     iaddq $64, %rsi      # dst += 8
56     iaddq $-8, %rdx      # len -= 8
57     jge Loop             # goto Loop:
58
59 Remain:
60     iaddq $5, %rdx       # len - 3
61     jg R4                # len > 3
62     jl Remain1           # R2 R1 R0
63     # je R3
64 R3:
65     mrmovq 16(%rdi), %r10
66     mrmovq 8(%rdi), %r11
67     mrmovq (%rdi), %r12
68     rmmovq %r10, 16(%rsi)
69     rmmovq %r11, 8(%rsi)
70     rmmovq %r12, (%rsi)
71     andq %r10, %r10
72     jle R3N1
73     iaddq $1, %rax
74 R3N1:
75     andq %r11, %r11
76     jle R3N2
77     iaddq $1, %rax
78 R3N2:
79     andq %r12, %r12
80     jle R3N3
81     iaddq $1, %rax
82 R3N3:
83     ret
84
85 Remain1:
86     iaddq $2, %rdx       # len - 1

```

```

87     jg R2
88     je R1
89     ret # jl Done, R0
90
91 R4:
92     mrmovq (%rdi), %r10
93     mrmovq 8(%rdi), %r11
94     mrmovq 16(%rdi), %r12
95     mrmovq 24(%rdi), %r13
96     rmmovq %r10, (%rsi)
97     rmmovq %r11, 8(%rsi)
98     rmmovq %r12, 16(%rsi)
99     rmmovq %r13, 24(%rsi)
100    andq %r10, %r10
101    jle R4N1
102    iaddq $1, %rax
103 R4N1:
104    andq %r11, %r11
105    jle R4N2
106    iaddq $1, %rax
107 R4N2:
108    andq %r12, %r12
109    jle R4N3
110    iaddq $1, %rax
111 R4N3:
112    andq %r13, %r13
113    jle R4N4
114    iaddq $1, %rax
115 R4N4:
116    iaddq $-1, %rdx      # len - 4
117    je Done
118    iaddq $-2, %rdx      # len - 6
119    je R6
120    jl R5
121    # jg R7
122
123 R7:
124    mrmovq 48(%rdi), %r10
125    rmmovq %r10, 48(%rsi)
126    andq %r10, %r10
127    jle R7N1
128    iaddq $1, %rax
129 R7N1:
130 R6:
131    mrmovq 40(%rdi), %r11
132    rmmovq %r11, 40(%rsi)
133    andq %r11, %r11
134    jle R7N2
135    iaddq $1, %rax

```



```

136 R7N2:
137 R5:
138     mrmovq 32(%rdi), %r12
139     rmmovq %r12, 32(%rsi)
140     andq %r12, %r12
141     jle R7N3
142     iaddq $1, %rax
143 R7N3:
144     ret
145
146 R2:
147     mrmovq 8(%rdi), %r11
148     rmmovq %r11, 8(%rsi)
149     andq %r11, %r11
150     jle R2N1
151     iaddq $1, %rax
152 R2N1:
153 R1:
154     mrmovq (%rdi), %r12
155     rmmovq %r12, (%rsi)
156     andq %r12, %r12
157     jle R2N2
158     iaddq $1, %rax
159 R2N2:
160     # ret

```

代码 8: 迎合分支预测

```

1     xorq %rax, %rax      # count = 0;
2     iaddq $-8, %rdx      # len -= 8
3     jl Remain           # if len < 8, goto Remain
4
5 Loop:
6     mrmovq (%rdi), %r10   # read val from src...
7     mrmovq 8(%rdi), %r11   # read val from src...
8     mrmovq 16(%rdi), %r12  # read val from src...
9     mrmovq 24(%rdi), %r13  # read val from src...
10    mrmovq 32(%rdi), %r14  # read val from src...
11    mrmovq 40(%rdi), %r9   # read val from src...
12    mrmovq 48(%rdi), %r8   # read val from src...
13    mrmovq 56(%rdi), %rcx  # read val from src...
14    rmmovq %r10, (%rsi)    # ...and store it to dst
15    rmmovq %r11, 8(%rsi)   # ...and store it to dst
16    rmmovq %r12, 16(%rsi)  # ...and store it to dst
17    rmmovq %r13, 24(%rsi)  # ...and store it to dst
18    rmmovq %r14, 32(%rsi)  # ...and store it to dst
19    rmmovq %r9, 40(%rsi)   # ...and store it to dst
20    rmmovq %r8, 48(%rsi)   # ...and store it to dst
21    rmmovq %rcx, 56(%rsi)  # ...and store it to dst

```

```

22     andq %r10, %r10      # val <= 0?
23     jle Npos             # if so, goto Npos:
24     iaddq $1, %rax       # count++
25 Npos:
26     andq %r11, %r11      # val <= 0?
27     jle Npos2            # if so, goto Npos2:
28     iaddq $1, %rax       # count++
29 Npos2:
30     andq %r12, %r12      # val <= 0?
31     jle Npos3            # if so, goto Npos3:
32     iaddq $1, %rax       # count++
33 Npos3:
34     andq %r13, %r13      # val <= 0?
35     jle Npos4            # if so, goto Npos4:
36     iaddq $1, %rax       # count++
37 Npos4:
38     andq %r14, %r14      # val <= 0?
39     jle Npos5            # if so, goto Npos5:
40     iaddq $1, %rax       # count++
41 Npos5:
42     andq %r9, %r9        # val <= 0?
43     jle Npos6            # if so, goto Npos6:
44     iaddq $1, %rax       # count++
45 Npos6:
46     andq %r8, %r8        # val <= 0?
47     jle Npos7            # if so, goto Npos7:
48     iaddq $1, %rax       # count++
49 Npos7:
50     andq %rcx, %rcx      # val <= 0?
51     jle Npos8            # if so, goto Npos8:
52     iaddq $1, %rax       # count++
53 Npos8:
54     iaddq $64, %rdi       # src += 8
55     iaddq $64, %rsi       # dst += 8
56     iaddq $-8, %rdx       # len -= 8
57     jge Loop             # goto Loop:
58
59 Remain:
60     iaddq $5, %rdx        # len - 3
61     jl Remain1           # R2 R1 R0
62     jg R4                # len > 3
63     # je R3
64 R3:
65     mrmovq 16(%rdi), %r10
66     mrmovq 8(%rdi), %r11
67     mrmovq (%rdi), %r12
68     rmmovq %r10, 16(%rsi)
69     rmmovq %r11, 8(%rsi)
70     rmmovq %r12, (%rsi)

```

```

71     andq %r10, %r10
72     jle R3N1
73     iaddq $1, %rax
74 R3N1:
75     andq %r11, %r11
76     jle R3N2
77     iaddq $1, %rax
78 R3N2:
79     andq %r12, %r12
80     jle R3N3
81     iaddq $1, %rax
82 R3N3:
83     ret
84
85 Remain1:
86     iaddq $2, %rdx      # len - 1
87     je R1
88     jg R2
89     ret # jl Done, R0
90
91 R4:
92     mrmovq (%rdi), %r10
93     mrmovq 8(%rdi), %r11
94     mrmovq 16(%rdi), %r12
95     mrmovq 24(%rdi), %r13
96     rmmovq %r10, (%rsi)
97     rmmovq %r11, 8(%rsi)
98     rmmovq %r12, 16(%rsi)
99     rmmovq %r13, 24(%rsi)
100    andq %r10, %r10
101    jle R4N1
102    iaddq $1, %rax
103 R4N1:
104    andq %r11, %r11
105    jle R4N2
106    iaddq $1, %rax
107 R4N2:
108    andq %r12, %r12
109    jle R4N3
110    iaddq $1, %rax
111 R4N3:
112    andq %r13, %r13
113    jle R4N4
114    iaddq $1, %rax
115 R4N4:
116    iaddq $-1, %rdx      # len - 4
117    jne R567
118    ret      # len = 0
119 R567:

```

```

120     iaddq $-2, %rdx      # len - 6
121     je R6
122     jl R5
123
124 R7:
125     mrmovq 48(%rdi), %r10
126     rmmovq %r10, 48(%rsi)
127     andq %r10, %r10
128     jle R7N1
129     iaddq $1, %rax
130 R7N1:
131 R6:
132     mrmovq 40(%rdi), %r11
133     rmmovq %r11, 40(%rsi)
134     andq %r11, %r11
135     jle R7N2
136     iaddq $1, %rax
137 R7N2:
138 R5:
139     mrmovq 32(%rdi), %r12
140     rmmovq %r12, 32(%rsi)
141     andq %r12, %r12
142     jle R7N3
143     iaddq $1, %rax
144 R7N3:
145     ret
146
147 R2:
148     mrmovq 8(%rdi), %r11
149     rmmovq %r11, 8(%rsi)
150     andq %r11, %r11
151     jle R2N1
152     iaddq $1, %rax
153 R2N1:
154 R1:
155     mrmovq (%rdi), %r12
156     rmmovq %r12, (%rsi)
157     andq %r12, %r12
158     jle R2N2
159     iaddq $1, %rax
160 R2N2:
161     # ret

```

代码 9: 进一步减少 Load-Use 冒险

```

1     xorq %rax, %rax      # count = 0;
2     iaddq $-8, %rdx      # len -= 8
3     jl Remain           # if len < 8, goto Remain
4

```

```

5 Loop:
6     mrmovq (%rdi), %r10      # read val from src...
7     mrmovq 8(%rdi), %r11     # read val from src...
8     mrmovq 16(%rdi), %r12    # read val from src...
9     mrmovq 24(%rdi), %r13    # read val from src...
10    mrmovq 32(%rdi), %r14    # read val from src...
11    mrmovq 40(%rdi), %r9     # read val from src...
12    mrmovq 48(%rdi), %r8     # read val from src...
13    mrmovq 56(%rdi), %rcx    # read val from src...
14    rmmovq %r10, (%rsi)      # ...and store it to dst
15    rmmovq %r11, 8(%rsi)     # ...and store it to dst
16    rmmovq %r12, 16(%rsi)    # ...and store it to dst
17    rmmovq %r13, 24(%rsi)    # ...and store it to dst
18    rmmovq %r14, 32(%rsi)    # ...and store it to dst
19    rmmovq %r9, 40(%rsi)     # ...and store it to dst
20    rmmovq %r8, 48(%rsi)     # ...and store it to dst
21    rmmovq %rcx, 56(%rsi)    # ...and store it to dst
22    andq %r10, %r10          # val <= 0?
23    jle Npos                 # if so, goto Npos:
24    iaddq $1, %rax           # count++
25 Npos:
26    andq %r11, %r11          # val <= 0?
27    jle Npos2                # if so, goto Npos2:
28    iaddq $1, %rax           # count++
29 Npos2:
30    andq %r12, %r12          # val <= 0?
31    jle Npos3                # if so, goto Npos3:
32    iaddq $1, %rax           # count++
33 Npos3:
34    andq %r13, %r13          # val <= 0?
35    jle Npos4                # if so, goto Npos4:
36    iaddq $1, %rax           # count++
37 Npos4:
38    andq %r14, %r14          # val <= 0?
39    jle Npos5                # if so, goto Npos5:
40    iaddq $1, %rax           # count++
41 Npos5:
42    andq %r9, %r9            # val <= 0?
43    jle Npos6                # if so, goto Npos6:
44    iaddq $1, %rax           # count++
45 Npos6:
46    andq %r8, %r8            # val <= 0?
47    jle Npos7                # if so, goto Npos7:
48    iaddq $1, %rax           # count++
49 Npos7:
50    andq %rcx, %rcx          # val <= 0?
51    jle Npos8                # if so, goto Npos8:
52    iaddq $1, %rax           # count++
53 Npos8:

```

```

54     iaddq $64, %rdi      # src += 8
55     iaddq $64, %rsi      # dst += 8
56     iaddq $-8, %rdx      # len -= 8
57     jge Loop             # goto Loop:
58
59 Remain:
60     iaddq $5, %rdx       # len - 3
61     jl Remain1           # R2 R1 R0
62     jg R4                # len > 3
63     # je R3
64 R3:
65     mrmovq 16(%rdi), %r10
66     mrmovq 8(%rdi), %r11
67     mrmovq (%rdi), %r12
68     rmmovq %r10, 16(%rsi)
69     rmmovq %r11, 8(%rsi)
70     rmmovq %r12, (%rsi)
71     andq %r10, %r10
72     jle R3N1
73     iaddq $1, %rax
74 R3N1:
75     andq %r11, %r11
76     jle R3N2
77     iaddq $1, %rax
78 R3N2:
79     andq %r12, %r12
80     jle R3N3
81     iaddq $1, %rax
82 R3N3:
83     ret
84
85 Remain1:
86     iaddq $2, %rdx       # len - 1
87     je R1
88     jg R2
89     ret # jl Done, R0
90
91 R4:
92     mrmovq (%rdi), %r10
93     mrmovq 8(%rdi), %r11
94     mrmovq 16(%rdi), %r12
95     mrmovq 24(%rdi), %r13
96     rmmovq %r10, (%rsi)
97     rmmovq %r11, 8(%rsi)
98     rmmovq %r12, 16(%rsi)
99     rmmovq %r13, 24(%rsi)
100    andq %r10, %r10
101    jle R4N1
102    iaddq $1, %rax

```

```

103 R4N1:
104     andq %r11, %r11
105     jle R4N2
106     iaddq $1, %rax
107 R4N2:
108     andq %r12, %r12
109     jle R4N3
110     iaddq $1, %rax
111 R4N3:
112     andq %r13, %r13
113     jle R4N4
114     iaddq $1, %rax
115 R4N4:
116     iaddq $-1, %rdx      # len - 4
117     jne R567
118     ret      # len = 0
119 R567:
120     iaddq $-2, %rdx      # len - 6
121     je R6
122     jl R5
123
124 R7:
125     mrmovq 48(%rdi), %r10
126     rmmovq %r10, 48(%rsi)
127     andq %r10, %r10
128 R6:
129     mrmovq 40(%rdi), %r11
130     jle R7N1
131     iaddq $1, %rax
132 R7N1:
133     rmmovq %r11, 40(%rsi)
134     andq %r11, %r11
135 R5:
136     mrmovq 32(%rdi), %r12
137     jle R7N2
138     iaddq $1, %rax
139 R7N2:
140     rmmovq %r12, 32(%rsi)
141     andq %r12, %r12
142     jle R7N3
143     iaddq $1, %rax
144 R7N3:
145     ret
146
147 R2:
148     mrmovq 8(%rdi), %r11
149     rmmovq %r11, 8(%rsi)
150     andq %r11, %r11
151 R1:

```

```
152     mrmovq (%rdi), %r12
153     jle R2N1
154     iaddq $1, %rax
155 R2N1:
156     rmmovq %r12, (%rsi)
157     andq %r12, %r12
158     jle R2N2
159     iaddq $1, %rax
160 R2N2:
161     # ret
```
