4.47

A

```
void bubble_a(long* data, long count){
    long i, last;
    for(last = count - 1; last > 0; last -= 1){
        for(i = 0; i < last; i += 1){
            long* data_i = data + i;
            if(*(data_i + 1) < *(data_i)){
                long t = *(data_i + 1);
                *(data_i + 1) = *(data_i);
                *(data_i) = t;
            }
        }
    }
}
```

B

```
.pos 0
    irmovq stack, %rsp
    call main
    halt

# Array of 5 elements
.align 8
data:
  .quad 0x0000000000000003
  .quad 0x0000000000000002
  .quad 0x0000000000000005
  .quad 0x0000000000000001
  .quad 0x0000000000000004

main:
    irmovq data,%rdi
    irmovq 0x5,%rsi
    call bubble_a
    ret
```

(下一页是 bubble_a 的 y86-64 代码,再下一页是和它等价的 x86-64 代码)

```
bubble_a:
    irmovq 0x1,%r9
    rrmovq %rsi,%r8
    subq   %r9,%r8
    jmp    L1
L4:
    rromvq %rax,%r9    # lea    (%rdi,%rax,8),%rdx
    addq   %r9,%r9
    addq   %r9,%r9
    addq   %r9,%r9
    rrmovq %r9,%rdx
    addq   %rdi,%rdx
    mrmovq 0x8(%rdx),%rcx
    mrmovq (%rdx),%rsi
    rrmovq %rcx,%r9   # cmp    %rsi,%rcx
    subq   %rsi,%r9
    jge    L2
    rmmovq %rsi,0x8(%rdx)
    rmmovq %rcx,(%rdx)
L2:
    irmovq 0x1,%r9
    addq   %r9,%rax
    jmp    L3
L5:
    xorq   %eax,%eax
L3:
    rrmovq %rax,%r9  # cmp    %r8,%rax
    subq   %r8,%r9
    jl     L4
    irmovq 0x1,%r9
    subq   %r9,%r8
L1:
    irmovq 0x0, %r9  # test   %r8,%r8
    subq   %r9,%r8
    jg     L5
    retq

.pos 0x200
stack:
```

等价的 x86 的形式为

```
00000000004005b6 <bubble_a>:
   4005b6:   4c 8d 46 ff          lea    -0x1(%rsi),%r8
   4005ba:   eb 2b                jmp    4005e7 <bubble_a+0x31>
   4005bc:   48 8d 14 c7          lea    (%rdi,%rax,8),%rdx
   4005c0:   48 8b 4a 08          mov    0x8(%rdx),%rcx
   4005c4:   48 8b 32             mov    (%rdx),%rsi
   4005c7:   48 39 f1             cmp    %rsi,%rcx
   4005ca:   7d 07                jge    4005d3 <bubble_a+0x1d>
   4005cc:   48 89 72 08          mov    %rsi,0x8(%rdx)
   4005d0:   48 89 0a             mov    %rcx,(%rdx)
   4005d3:   48 83 c0 01          add    $0x1,%rax
   4005d7:   eb 05                jmp    4005de <bubble_a+0x28>
   4005d9:   b8 00 00 00 00       mov    $0x0,%eax
   4005de:   4c 39 c0             cmp    %r8,%rax
   4005e1:   7c d9                jl     4005bc <bubble_a+0x6>
   4005e3:   49 83 e8 01          sub    $0x1,%r8
   4005e7:   4d 85 c0             test   %r8,%r8
   4005ea:   7f ed                jg     4005d9 <bubble_a+0x23>
   4005ec:   f3 c3                repz retq
```

4.56

valP 会从 D_valP 到 E_valA 到 M_valA,这是已经实现了的,所以只需要考虑将 valC 传送到M_valE,这需要将 E_valC 也加入 ALU 中,产生 e_valE 存入 M_valE,需要使 aluA = E_valC, aluB = 0 (alu 默认执行加法)

还需要改动 SelectPC,D_bubble,E_bubble 的逻辑使得跳转错误时能够找到正确的分支,并且需要考虑到有两种跳转错误的方式

SelectPC 中的逻辑(使用 M 寄存器中的数据,因为这些在下一个时钟周期才会执行)

```
    M_icode == IJXX && M_ifun != UNCOND && M_valE < M_valA && !M_Cnd:M_valA;
    M_icode == IJXX && M_ifun != UNCOND && M_valE >= M_valA && M_Cnd:M_valE;
```

D/E_bubble 中的逻辑(使用这个时钟周期计算出的 e_Cnd 因为下一个时钟周期开始之前就要确定是否要插入气泡)

```
    ((E_icode == IJXX && E_ifun != UNCOND && E_valC < E_valA && !e_Cnd)
||
    (E_icode == IJXX && E_ifun != UNCOND && E_valC >= E_valA && e_Cnd))
```

5.13

A

图 5.13 的风格

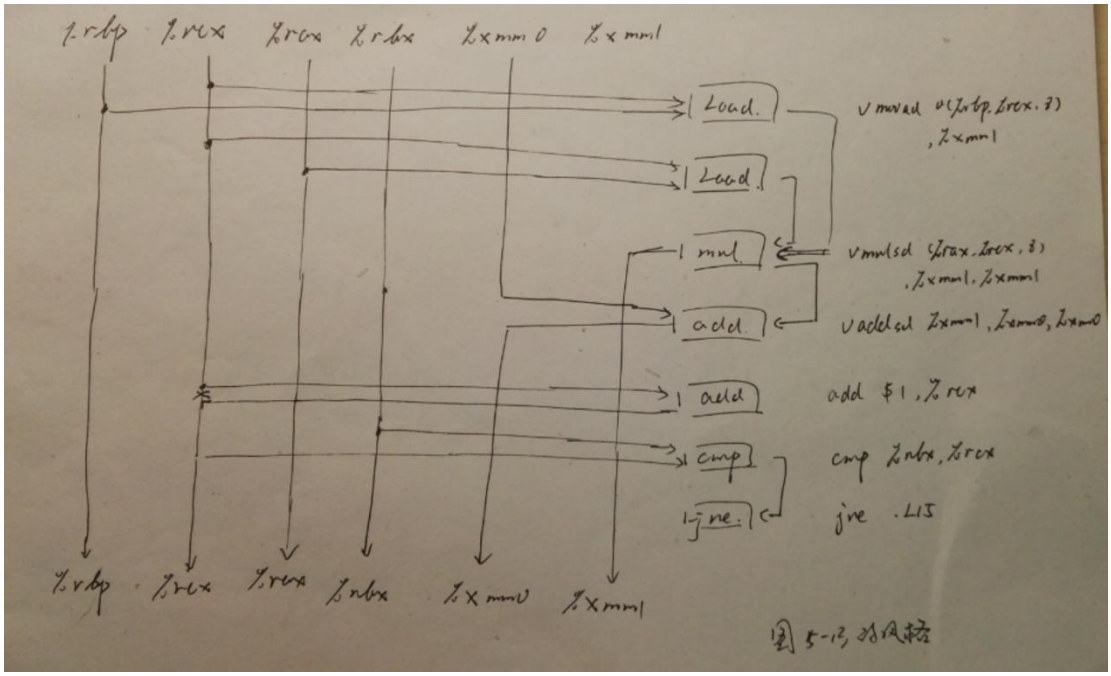
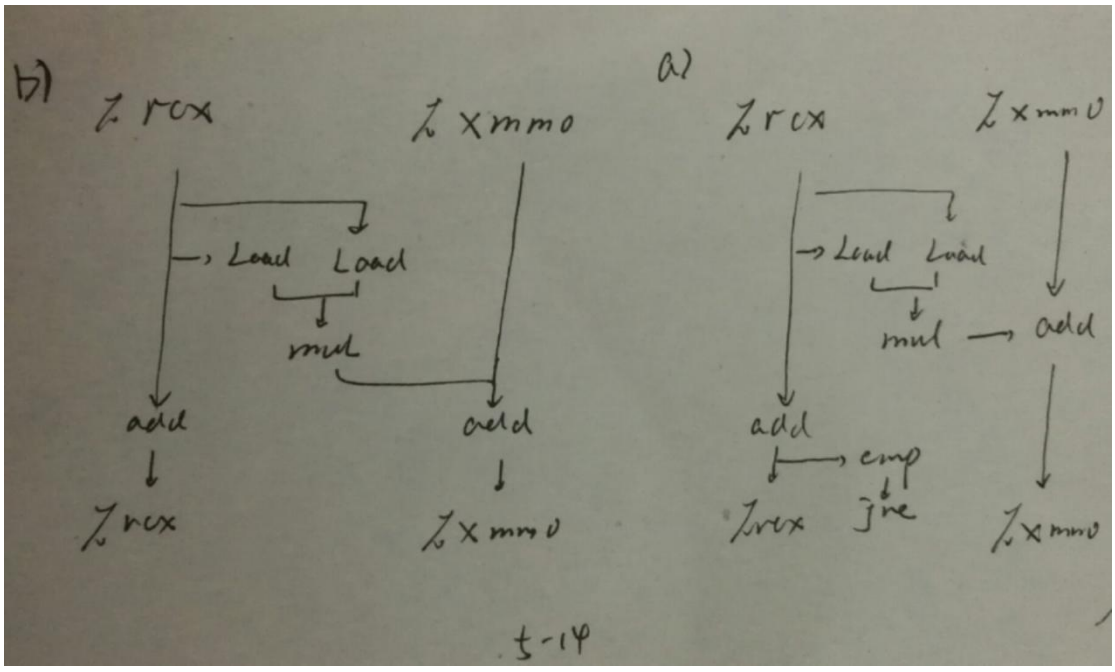


图 5.14 的风格(左面是 5.14b)的风格,右面是 5.14a)的风格)



对%xmm0 的操作形成一条关键路径

B

下界是 3.00(对%xmm 寄存器进行加法)

C

下界是 1.00(一个整数加法,同时也是两个整数 load 的时间)

D

因为乘法不在关键路径中,乘法的操作不构成数据相关,所以只会受到吞吐量的限制

5.14

```
void inner4(vec_ptr u, vec_ptr v, data_t *dest){
   long i;
   long length = vec_length(u);
   data_t *u_data = get_vec_start(u);
   data_t *v_data = get_vec_start(v);
   data_t sum = (data_t) 0;

   long limit = length - 5;
   for(i = 0; i < limit; i += 6){
      sum = sum + udata[i] * vdata[i]
                + udata[i + 1] * vdata[i + 1]
                + udata[i + 2] * vdata[i + 2]
                + udata[i + 3] * vdata[i + 3]
                + udata[i + 4] * vdata[i + 4]
                + udata[i + 5] * vdata[i + 5];
   }
   for( ; i < length; i += 1){
      sum = sum + udata[i] * vdata[i];
   }
   *dest = sum;
}
```

A

每计算一个值都需要两次 load,而整数只有两个加载的单元

B

在这个例子中,因为浮点数的加法从左向右结合,所以每一次循环中,关键路径上仍然有 6 个浮点加法,和不展开的情况是类似的