# 第四章：

P4.47：

A：



B：

# 从地址 0 开始执行

    .pos 0

    irmovq stack, %rsp      # Set up stack pointer

    call main               # Execute main program

    halt                    # Terminate program

# 四个元素的数组

    .align 8

array:

    .quad 0x000000000abc

    .quad 0x0000000000bc

    .quad 0x00000000000c

    .quad 0x000000000001

# 主函数

main:

    irmovq array,%rdi

    irmovq $4,%rsi

    call bubble\_b           # bubble(array, 4)

    ret

# void bubble\_b(long \*data, long count)

# data in %rdi, count in %rsi

bubble\_b:

    irmovq $1, %r8           # 常数 1

    irmovq $8, %r9           # 常数 8

    rrmovq %rsi, %rax

    subq %r8, %rax          # last = %rax = count - 1

    je Done                 # last == 0 -> jmp done

Loop1:

    xorq %rcx, %rcx         # i = %rcx = 0

Loop2:

    rrmovq %rcx, %rdx       # %rdx = i

    addq %rdx, %rdx         # %rdx = 2 \* i

    addq %rdx, %rdx         # %rdx = 4 \* i

    addq %rdx, %rdx         # %rdx = 8 \* i

    addq %rdi, %rdx         # %rdx = data + 8 \* i

    mrmovq (%rdx), %r10     # %r10 = data[i]

    addq %r9, %rdx          # %rdx = data + 8 \* i + 8

    mrmovq (%rdx), %rbx     # %rbx = data[i + 1]

    rrmovq %rbx, %r11       # %r11 = data[i + 1]

    subq %r10, %rbx         # %rbx = data[i + 1] - data[i]

    jge Test1               # data[i + 1] - data[i] > 0 -> do nothing

    rmmovq %r10, (%rdx)     # data[i + 1] = data[i]

    subq %r9, %rdx          # %rdx = %rdx - 8

    rmmovq %r11, (%rdx)     # data[i] = data[i + 1]

Test1:

    addq %r8, %rcx          # i++

    rrmovq %rcx, %r12       # %r12 = %rcx = i

    subq %rax, %r12         # i - last

    jl Loop2                # i < last -> jmp Loop2

    subq %r8, %rax          # last--

    jg Loop1                # last > 0 -> jmp Loop1

Done:

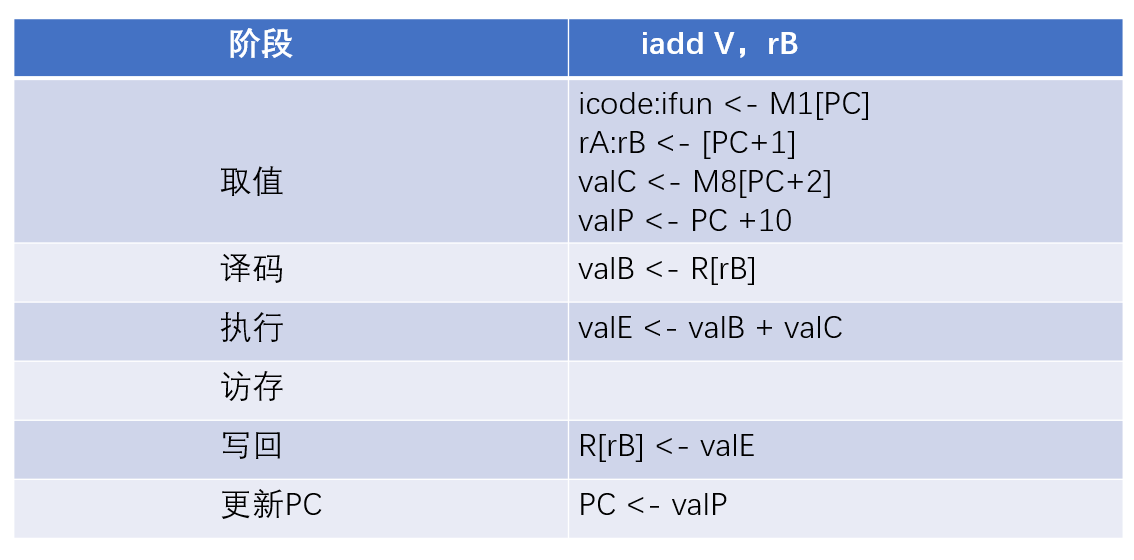
    ret                     # Return

# 栈从 0x200 向低地址生长

    .pos 0x200

stack:

P4.51：



P4.59：

答：4.47性能更好。

假设data[i+1]>data[i]的概率为50%

4.47、4.48、4.49三者分别执行的指令数：2.5，7，6

三者分别需要的时钟周期：3.5，7，6

# 第五章：

P5.15：

答：void inner6(vec\_ptr u, vec\_ptr v, data\_t \*dest) {

    long i;

    long length = vec\_length(u);

    long limit = length - 5;

    data\_t \*udata = get\_ver\_start(u);

    data\_t \*vdata = get\_vec\_start(v);

    data\_t sum0 = (data\_t)0;

    data\_t sum1 = (data\_t)0;

    data\_t sum2 = (data\_t)0;

    data\_t sum3 = (data\_t)0;

    data\_t sum4 = (data\_t)0;

    data\_t sum5 = (data\_t)0;

    for (i = 0; i < limit; i += 6) {

        sum0 = sum0 + udata[i] \* vdata[i];

        sum1 = sum1 + udata[i + 1] \* vdata[i + 1];

        sum2 = sum2 + udata[i + 2] \* vdata[i + 2];

        sum3 = sum3 + udata[i + 3] \* vdata[i + 3];

        sum4 = sum4 + udata[i + 4] \* vdata[i + 4];

        sum5 = sum5 + udata[i + 5] \* vdata[i + 5];

    }

    for (; i < length; i++) {

        sum0 = sum0 + udata[i] \* vdata[i];

    }

    \*dest = sum0 + sum1 + sum2 + sum3 + sum4 + sum5;

}

原因：只有两个加载单元，一个时钟周期只能加载两个值，CPE 最低只能到 1.00。

P5.19：

void psum\_4\_1a(float a[], float p[], long n) {

    long i;

    float tmp0, tmp1, tmp2, tmp3 = 0;

    for (i = 0; i < n - 3; i += 4) {

        tmp0 = tmp3 + a[i];

        tmp1 = tmp0 + a[i + 1];

        tmp2 = tmp1 + a[i + 2];

        tmp3 = tmp2 + a[i + 3];

        p[i] = tmp0;

        p[i + 1] = tmp1;

        p[i + 2] = tmp2;

        p[i + 3] = tmp3;

    }

    for (; i < n; i++) {

        tmp3 += a[i];

        p[i] = tmp3;

    }

}

# 第七章：

P7.7：

double x;

void f()

{

    //x = -0.0;

}

P7.11：

为bss文件留出空间。