HW1/H3问题回答

我的Makefile使用说明

首先描述一下我的Makefile的使用说明. 题目虽然要求使用`gcc`和`clang`, 并且要有32位和64位的不同版本结果, 但我统一为一个文件, 而借助make命令后接参数来决定使用的编译器和目标机器位数.

以下是各种情况的编译命令(`-B`是覆盖):

- `gcc`, 32位: `make -B CC=gcc FLAGS=-m32`
- `gcc`, 64位: `make -B CC=gcc FLAGS=-m64`
- `clang`, 32位: `make -B CC=clang FLAGS=-m32`
- `clang`, 64位: `make -B CC=clang FLAGS=-m64`

这里有个小问题: 例如你在64位的Ubuntu上运行生成32位代码, 可能会找不到头文件. 可以通过命令`sudo apt install lib32z1 lib32z1-dev`来安装.

问题回答

接下来针对`gcc-32`, `gcc-64`, `clang-32`, `clang-64`分别注释.
因为有篇幅问题, 这里给出超链接gcc-32, gcc-64, clang-32, clang-64(点击标题也可跳转回来)
其中对`gcc-32`的注释是完全详细的版本, 对其他的注释则是增量注释(有不同之处才注释)

为了表明我确实完整地读过四个汇编代码, 这里先将一些主要特点(没有提到的特点可能是过于平凡或前面H1, H2有提到)用表格的形式给出.

	gcc-32	**gcc-64**	**clang-32**	**clang-6**
函数参数传递	栈(push为主)	寄存器(%rsi等)优先 设置%eax(参考资料1)	栈(mov为主)	寄存器(设置%e
函数返回值	%eax	%rax	%eax	%rax
特别的指令	X	`cltq`(符号扩展)等	X	用`mova` 用`movs`

	gcc-32	gcc-64	clang-32	clang-6
特别的编译	存在栈检查的操作 `printf("\n")`改为`putchar`	存在栈检查的操作 `printf("\n")`改为`putchar`	没有栈检查	没有栈

- 参考资料1

gcc, 32位

首先展示对gcc, 32位下的汇编码的注释.
我对每一句汇编代码都作了一定解释, 并且对相对完整的每块代码注释了其在源代码中的对应语句.(可能需要拖动滚动条才能看到右边的注释)

```
        .file   "sort.c"
        .text
        .globl  my_sort
        .type   my_sort, @function
my_sort:                                # 函数my_sort标签
.LFB2:
        .cfi_startproc
        pushl   %ebp                    # 保存基址寄存器
        .cfi_def_cfa_offset 8
        .cfi_offset 5, -8
        movl    %esp, %ebp              # 把栈顶寄存器值存入%ebp
        .cfi_def_cfa_register 5
        subl    $16, %esp               # 扩栈
        movl    $0, -12(%ebp)           # i = 0
        jmp     .L2                     # 跳转到.L2
.L6:
        movl    -12(%ebp), %eax         # %eax = i
        addl    $1, %eax                # %eax = i + 1
        movl    %eax, -8(%ebp)          # j = i + 1
        jmp     .L3                     # 跳转到.L3
.L5:
        movl    -12(%ebp), %eax         # %eax = i                              -------\
        leal    0(,%eax,4), %edx        # %edx = 4 * %eax                              |
        movl    8(%ebp), %eax           # %eax = nums                                  |
        addl    %edx, %eax              # %eax = nums + i                              |
        movl    (%eax), %edx            # %edx = nums[i]                               |
        movl    -8(%ebp), %eax          # %eax = j                                     |
        leal    0(,%eax,4), %ecx        # %ecx = 4 * %eax                              | if(nums[i] > nums[j])
        movl    8(%ebp), %eax           # %eax = nums                                  |
        addl    %ecx, %eax              # %eax = nums + j                              |
        movl    (%eax), %eax            # %eax = nums[j]                               |
        cmpl    %eax, %edx              # 比较nums[j]和nums[i]                          |
        jle     .L4                     # 如果nums[i] <= nums[j], 就跳转到.L4     -------/
        movl    -12(%ebp), %eax         # %eax = i                              -------\
        leal    0(,%eax,4), %edx        # %edx = 4 * %eax                              |
        movl    8(%ebp), %eax           # %eax = nums                                  |
        addl    %edx, %eax              # %eax = nums + i                              | tmp = nums[i]
        movl    (%eax), %eax            # %eax = nums[i]                               |
        movl    %eax, -4(%ebp)          # tmp = nums[i]                         -------/
        movl    -12(%ebp), %eax         # %eax = i                              -------\
        leal    0(,%eax,4), %edx        # %edx = 4 * %eax                              |
        movl    8(%ebp), %eax           # %eax = nums                                  |
        addl    %eax, %edx              # %edx = nums + i                              |
        movl    -8(%ebp), %eax          # %eax = j                                     |
        leal    0(,%eax,4), %ecx        # %ecx = 4 * %eax                              | nums[i] = nums[j]
        movl    8(%ebp), %eax           # %eax = nums                                  |
```

```
        addl    %ecx, %eax              # %eax = nums + j                               |
        movl    (%eax), %eax            # %eax = nums[j]                                |
        movl    %eax, (%edx)            # nums[i] = nums[j]                     -------/
        movl    -8(%ebp), %eax          # %eax = j                              -------\
        leal    0(,%eax,4), %edx        # %edx = 4 * %eax                               |
        movl    8(%ebp), %eax           # %eax = i                                      |
        addl    %eax, %edx              # %edx = nums + j                               | nums[j] = tmp
        movl    -4(%ebp), %eax          # %eax = tmp                                    |
        movl    %eax, (%edx)            # nums[j] = tmp                         -------/
.L4:
        addl    $1, -8(%ebp)            # j++
.L3:
        movl    -8(%ebp), %eax          # %eax = j
        cmpl    12(%ebp), %eax          # 比较n和j
        jl      .L5                     # 如果j < n跳转到.L5
        addl    $1, -12(%ebp)           # i++
.L2:
        movl    -12(%ebp), %eax         # %eax = i
        cmpl    12(%ebp), %eax          # 比较n和i
        jl      .L6                     # 如果i < n跳转到.L6
        nop                             # 空指令
        leave                           # 相当于pop %ebp
        .cfi_restore 5
        .cfi_def_cfa 4, 4
        ret                             # 返回
        .cfi_endproc
.LFE2:
        .size   my_sort, .-my_sort
        .section        .rodata
.LC0:
        .string "%d"
.LC1:
        .string " %d"
        .text
        .globl  main
        .type   main, @function
main:
.LFB3:
        .cfi_startproc
        leal    4(%esp), %ecx           # 加载有效地址，%ecx = %esp + 4
        .cfi_def_cfa 1, 0
        andl    $-16, %esp              # 置%esp的末四位成0，相当于栈指针跳到某地址为用户写的程序开了一段全新的栈
        pushl   -4(%ecx)                # 原栈顶(cfi程序的)入栈
        pushl   %ebp                    # %ebp入栈，保存基址寄存器
        .cfi_escape 0x10,0x5,0x2,0x75,0
        movl    %esp, %ebp              # %ebp = %esp，%ebp置为栈顶指针
```

```
        pushl   %ecx                    # %ecx入栈，保存返回地址
        .cfi_escape 0xf,0x3,0x75,0x7c,0x6
        subl    $36, %esp               # 扩栈，为main函数提供栈
        movl    %gs:20, %eax            #
        movl    %eax, -12(%ebp)         # 将%gs:20存入-12(%ebp)
        xorl    %eax, %eax              # %eax = 0，清零%eax
        subl    $8, %esp                # %esp -= 8，扩栈             -------\
        leal    -28(%ebp), %eax         # %eax = %ebp - 28，计算出n的地址      | scanf("%d", &n);
        pushl   %eax                    # %eax入栈，n的地址入栈                |
        pushl   $.LC0                   # .LC0地址入栈                        |
        call    __isoc99_scanf          # 调用scanf函数                       |
        addl    $16, %esp               # 函数调用结束，退栈           -------/
        movl    -28(%ebp), %eax         # 取出n，存入%eax              -------\
        sall    $2, %eax                # sizeof(int)*%eax                    |
        subl    $12, %esp               # 扩栈12字                            | malloc(sizeof(int)*n);
        pushl   %eax                    # %eax入栈                            |
        call    malloc                  # 调用malloc函数                       |
        addl    $16, %esp               # 调用函数结束，退栈                    |
        movl    %eax, -16(%ebp)         # 保存返回值                           |
        movl    $0, -24(%ebp)           # i = 0                       -------/
        jmp     .L8                     # 跳转到.L8
.L9:
        movl    -24(%ebp), %eax         # %eax = i
        leal    0(,%eax,4), %edx        # %edx = %eax * 4
        movl    -16(%ebp), %eax         # %eax = nums
        addl    %edx, %eax              # %eax += %edx，即取num+i
        subl    $8, %esp                # 扩栈                         -------\
        pushl   %eax                    # num+i入栈                           |
        pushl   $.LC0                   # "%d"入栈                            | scanf("%d", nums+i);
        call    __isoc99_scanf          # 调用scanf                           |
        addl    $16, %esp               # 退栈                         -------/
        addl    $1, -24(%ebp)           # i++
.L8:
        movl    -28(%ebp), %eax         # 取出n，存入%eax
        cmpl    %eax, -24(%ebp)         # 比较n和i
        jl      .L9                     # 若i<n就跳转到循环体.L9
        movl    -28(%ebp), %eax         # 取出n，存入%eax
        subl    $8, %esp                # 扩栈                         -------\
        pushl   %eax                    # %eax，即n，入栈                      |
        pushl   -16(%ebp)               # nums入栈                             | my_sort(nums, n);
        call    my_sort                 # 调用函数my_sort                      |
        addl    $16, %esp               # 退栈                         -------/
        movl    -16(%ebp), %eax         # %eax = nums
        movl    (%eax), %eax            # %eax = nums[0]
        subl    $8, %esp                # 扩栈                         -------\
        pushl   %eax                    # nums[0]入栈                         |
```

```
        pushl   $.LC0                   # "%d"入栈                          | printf("%d", nums[0]);
        call    printf                  # 调用printf                        |
        addl    $16, %esp               # 退栈                      -------/
        movl    $1, -20(%ebp)           # print的i = 1
        jmp     .L10                    # 跳转到.L10准备开始循环print
.L11:
        movl    -20(%ebp), %eax         # %eax = i
        leal    0(,%eax,4), %edx        # %edx = 4 * %eax
        movl    -16(%ebp), %eax         # %eax = nums
        addl    %edx, %eax              # %eax = nums + i
        movl    (%eax), %eax            # 即%eax = nums[i]
        subl    $8, %esp                # 扩栈                      -------\
        pushl   %eax                    # nums[i]入栈                       |
        pushl   $.LC1                   # " %d"入栈                         | printf(" %d", nums[i]);
        call    printf                  # 调用printf                        |
        addl    $16, %esp               # 退栈                      -------/
        addl    $1, -20(%ebp)           # i++
.L10:
        movl    -28(%ebp), %eax         # %eax = n
        cmpl    %eax, -20(%ebp)         # 比较n和i
        jl      .L11                    # 跳转到.L11
        subl    $12, %esp               # 扩栈                      -------\
        pushl   $10                     # 10(换行)入栈                       |
        call    putchar                 # 调用putchar                       | putchar('\n');
        addl    $16, %esp               # 退栈                      -------/
        movl    $0, %eax                # %eax = 0
        movl    -12(%ebp), %ecx         # 读出之前的%gs:20
        xorl    %gs:20, %ecx            # 和现在的%gs:20作比较，相等的话就会置ZF
        je      .L13                    # 检查返回地址和%ebp的值
        call    __stack_chk_fail        # 栈检查失败
.L13:
        movl    -4(%ebp), %ecx          # 前栈帧的%ebp
        .cfi_def_cfa 1, 0
        leave                           # 相当于pop %ebp
        .cfi_restore 5
        leal    -4(%ecx), %esp          # %esp = 恢复好%esp
        .cfi_def_cfa 4, 4
        ret                             # 返回
        .cfi_endproc
.LFE3:
        .size   main, .-main
        .ident  "GCC: (Ubuntu 5.4.0-6ubuntu1~16.04.12) 5.4.0 20160609"
        .section        .note.GNU-stack,"",@progbits
```

gcc, 64位

`gcc，64位`与`gcc，32位`的主要区别在于:

- 调用函数的参数传递借助`%rdi`，`%rsi`，`%rdx`，`%rcx`，`%r8`，`%r9`等寄存器, 非必要时候不使用栈
- 需要设置`%eax`以提供对向量寄存器参数使用情况的说明
- 需要使用`cltq`指令对32位数据扩展为64位数据

其余东西大同小异, 将以下是对不同之处的注释:

```
        .file   "sort.c"
        .text
        .globl  my_sort
        .type   my_sort, @function
my_sort:
.LFB2:
        .cfi_startproc
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        movq    %rdi, -24(%rbp)
        movl    %esi, -28(%rbp)
        movl    $0, -12(%rbp)
        jmp     .L2
.L6:
        movl    -12(%rbp), %eax
        addl    $1, %eax
        movl    %eax, -8(%rbp)
        jmp     .L3
.L5:
        movl    -12(%rbp), %eax
        cltq                                # 将32位%eax扩展为64位%rax
        leaq    0(,%rax,4), %rdx
        movq    -24(%rbp), %rax
        addq    %rdx, %rax
        movl    (%rax), %edx
        movl    -8(%rbp), %eax
        cltq                                # 将32位%eax扩展为64位%rax
        leaq    0(,%rax,4), %rcx
        movq    -24(%rbp), %rax
        addq    %rcx, %rax
        movl    (%rax), %eax
        cmpl    %eax, %edx
        jle     .L4
        movl    -12(%rbp), %eax
        cltq                                # 将32位%eax扩展为64位%rax
        leaq    0(,%rax,4), %rdx
        movq    -24(%rbp), %rax
        addq    %rdx, %rax
        movl    (%rax), %eax
        movl    %eax, -4(%rbp)
        movl    -12(%rbp), %eax
        cltq                                # 将32位%eax扩展为64位%rax
        leaq    0(,%rax,4), %rdx
```

```
        movq    -24(%rbp), %rax
        addq    %rax, %rdx
        movl    -8(%rbp), %eax
        cltq                                    # 将32位%eax扩展为64位%rax
        leaq    0(,%rax,4), %rcx
        movq    -24(%rbp), %rax
        addq    %rcx, %rax
        movl    (%rax), %eax
        movl    %eax, (%rdx)
        movl    -8(%rbp), %eax
        cltq                                    # 将32位%eax扩展为64位%rax
        leaq    0(,%rax,4), %rdx
        movq    -24(%rbp), %rax
        addq    %rax, %rdx
        movl    -4(%rbp), %eax
        movl    %eax, (%rdx)
.L4:
        addl    $1, -8(%rbp)
.L3:
        movl    -8(%rbp), %eax
        cmpl    -28(%rbp), %eax
        jl      .L5
        addl    $1, -12(%rbp)
.L2:
        movl    -12(%rbp), %eax
        cmpl    -28(%rbp), %eax
        jl      .L6
        nop
        popq    %rbp
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
.LFE2:
        .size   my_sort, .-my_sort
        .section        .rodata
.LC0:
        .string "%d"
.LC1:
        .string " %d"
        .text
        .globl  main
        .type   main, @function
main:
.LFB3:
        .cfi_startproc
        pushq   %rbp
```

```
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        subq    $32, %rsp
        movq    %fs:40, %rax
        movq    %rax, -8(%rbp)
        xorl    %eax, %eax
        leaq    -28(%rbp), %rax
        movq    %rax, %rsi                  # 为scanf传参(&n)                    -------\
        movl    $.LC0, %edi                 # 为scanf传参("%d")                          |
        movl    $0, %eax                    #                                    | scanf("%d", &n);
        call    __isoc99_scanf              # 调用scanf                         -------/
        movl    -28(%rbp), %eax             # %eax = n
        cltq                                # 扩展%eax的32位到%rax的64位
        salq    $2, %rax
        movq    %rax, %rdi                  # 为malloc传参                       -------\
        call    malloc                      # 调用malloc                         -------/ malloc(sizeof(int)*n)
        movq    %rax, -16(%rbp)
        movl    $0, -24(%rbp)
        jmp     .L8
.L9:
        movl    -24(%rbp), %eax
        cltq
        leaq    0(,%rax,4), %rdx
        movq    -16(%rbp), %rax
        addq    %rdx, %rax
        movq    %rax, %rsi                  # 为scanf传参(&n)                    -------\
        movl    $.LC0, %edi                 # 为scanf传参("%d")                          |
        movl    $0, %eax                    #                                    | scanf("%d", &n);
        call    __isoc99_scanf              # 调用scanf                         -------/
        addl    $1, -24(%rbp)
.L8:
        movl    -28(%rbp), %eax
        cmpl    %eax, -24(%rbp)
        jl      .L9
        movl    -28(%rbp), %edx
        movq    -16(%rbp), %rax             # %rax = nums
        movl    %edx, %esi                  # 为my_sort传参(n)                  -------\
        movq    %rax, %rdi                  # 为my_sort传参(nums)                       | my_sort(nums, n);
        call    my_sort                     # 调用my_sort                       -------/
        movq    -16(%rbp), %rax
        movl    (%rax), %eax
        movl    %eax, %esi                  # 为printf传参(nums[0])             -------\
        movl    $.LC0, %edi                 # 为printf传参("%d")                        | printf("%d", nums[0]);
        movl    $0, %eax                    #                                   |
```

```
        call    printf                  # 调用printf                    -------/
        movl    $1, -20(%rbp)
        jmp     .L10
.L11:
        movl    -20(%rbp), %eax
        cltq                            # 将32位%eax扩展为64位%rax
        leaq    0(,%rax,4), %rdx
        movq    -16(%rbp), %rax
        addq    %rdx, %rax
        movl    (%rax), %eax
        movl    %eax, %esi
        movl    $.LC1, %edi
        movl    $0, %eax
        call    printf
        addl    $1, -20(%rbp)
.L10:
        movl    -28(%rbp), %eax
        cmpl    %eax, -20(%rbp)
        jl      .L11
        movl    $10, %edi               # 为putchar传参('\n')            -------\ putchar('\n')
        call    putchar                 # 调用putchar                    -------/
        movl    $0, %eax
        movq    -8(%rbp), %rcx
        xorq    %fs:40, %rcx            #                               -------\
        je      .L13                    #                                      | 栈检验
        call    __stack_chk_fail        #                               -------/
.L13:
        leave
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
.LFE3:
        .size   main, .-main
        .ident  "GCC: (Ubuntu 5.4.0-6ubuntu1~16.04.12) 5.4.0 20160609"
        .section        .note.GNU-stack,"",@progbits
```

clang, 32位

特点在于:

- 调用函数时的传参用栈, 但是直接使用了`mov`而不是`push`
- 没有栈检查操作
- `printf("\n")`没有被改为`putchar('\n')`

其余的地方并没有实质上的不同, 只是一些语句的顺序, 组合的不同而已.

下面是对不同之处的注释, 里面包含了原编译出来注释, 考虑到那些注释本身是有一定意义的, 我没有将他们去掉.
其中关于函数的注释我只注释了main中的前几个, 这些大多相似.

```
        .text
        .file   "sort.c"
        .globl  my_sort                 # -- Begin function my_sort
        .p2align        4, 0x90
        .type   my_sort,@function
my_sort:                                # @my_sort
# %bb.0:
        pushl   %ebp
        movl    %esp, %ebp
        subl    $12, %esp
        movl    12(%ebp), %eax
        movl    8(%ebp), %ecx
        movl    $0, -4(%ebp)
.LBB0_1:                                # =>This Loop Header: Depth=1
                                        #     Child Loop BB0_3 Depth 2
        movl    -4(%ebp), %eax
        cmpl    12(%ebp), %eax
        jge     .LBB0_10
# %bb.2:                                #   in Loop: Header=BB0_1 Depth=1
        movl    -4(%ebp), %eax
        addl    $1, %eax
        movl    %eax, -8(%ebp)
.LBB0_3:                                #   Parent Loop BB0_1 Depth=1
                                        # =>  This Inner Loop Header: Depth=2
        movl    -8(%ebp), %eax
        cmpl    12(%ebp), %eax
        jge     .LBB0_8
# %bb.4:                                #   in Loop: Header=BB0_3 Depth=2
        movl    8(%ebp), %eax
        movl    -4(%ebp), %ecx
        movl    (%eax,%ecx,4), %eax
        movl    8(%ebp), %ecx
        movl    -8(%ebp), %edx
        cmpl    (%ecx,%edx,4), %eax
        jle     .LBB0_6
# %bb.5:                                #   in Loop: Header=BB0_3 Depth=2
        movl    8(%ebp), %eax
        movl    -4(%ebp), %ecx
        movl    (%eax,%ecx,4), %eax
        movl    %eax, -12(%ebp)
        movl    8(%ebp), %eax
        movl    -8(%ebp), %ecx
        movl    (%eax,%ecx,4), %eax
        movl    8(%ebp), %ecx
        movl    -4(%ebp), %edx
        movl    %eax, (%ecx,%edx,4)
```

```
        movl    -12(%ebp), %eax
        movl    8(%ebp), %ecx
        movl    -8(%ebp), %edx
        movl    %eax, (%ecx,%edx,4)
.LBB0_6:                                #   in Loop: Header=BB0_3 Depth=2
        jmp     .LBB0_7
.LBB0_7:                                #   in Loop: Header=BB0_3 Depth=2
        movl    -8(%ebp), %eax
        addl    $1, %eax
        movl    %eax, -8(%ebp)
        jmp     .LBB0_3
.LBB0_8:                                #   in Loop: Header=BB0_1 Depth=1
        jmp     .LBB0_9
.LBB0_9:                                #   in Loop: Header=BB0_1 Depth=1
        movl    -4(%ebp), %eax
        addl    $1, %eax
        movl    %eax, -4(%ebp)
        jmp     .LBB0_1
.LBB0_10:
        addl    $12, %esp
        popl    %ebp
        retl
.Lfunc_end0:
        .size   my_sort, .Lfunc_end0-my_sort
                                        # -- End function
        .globl  main                    # -- Begin function main
        .p2align        4, 0x90
        .type   main,@function
main:                                   # @main
# %bb.0:
        pushl   %ebp
        movl    %esp, %ebp
        subl    $40, %esp
        movl    $0, -4(%ebp)
        leal    .L.str, %eax            # %eax = "%d"的地址                    -------\
        movl    %eax, (%esp)            # 传参("%d")                            |
        leal    -8(%ebp), %eax          # %eax = &n                            | scanf("%d", &n);
        movl    %eax, 4(%esp)           # 传参&n                                |
        calll   __isoc99_scanf          # 调用scanf                            -------/
        movl    -8(%ebp), %ecx
        shll    $2, %ecx
        movl    %ecx, (%esp)
        movl    %eax, -24(%ebp)         # 4-byte Spill
        calll   malloc
        movl    %eax, -12(%ebp)
        movl    $0, -16(%ebp)
```

```
.LBB1_1:                                # =>This Inner Loop Header: Depth=1
        movl    -16(%ebp), %eax
        cmpl    -8(%ebp), %eax
        jge     .LBB1_4
# %bb.2:                                #   in Loop: Header=BB1_1 Depth=1
        movl    -12(%ebp), %eax
        movl    -16(%ebp), %ecx
        shll    $2, %ecx
        addl    %ecx, %eax
        leal    .L.str, %ecx            # %eax = "%d"的地址                    -------\
        movl    %ecx, (%esp)            # 传参("%d")                            |
        movl    %eax, 4(%esp)           # 传参nums + i                          | scanf("%d", nums+i);
        calll   __isoc99_scanf          # 调用scanf                            -------/
# %bb.3:                                #   in Loop: Header=BB1_1 Depth=1
        movl    -16(%ebp), %eax
        addl    $1, %eax
        movl    %eax, -16(%ebp)
        jmp     .LBB1_1
.LBB1_4:
        movl    -12(%ebp), %eax         # %eax = num                           -------\
        movl    -8(%ebp), %ecx          # %eax = n                                    |
        movl    %eax, (%esp)            # 传参num                               | my_sort(nums, n);
        movl    %ecx, 4(%esp)           # 传参n                                 |
        calll   my_sort                 # 调用my_sort                          -------/
        movl    -12(%ebp), %eax
        movl    (%eax), %eax
        leal    .L.str, %ecx
        movl    %ecx, (%esp)
        movl    %eax, 4(%esp)
        calll   printf
        movl    $1, -20(%ebp)
.LBB1_5:                                # =>This Inner Loop Header: Depth=1
        movl    -20(%ebp), %eax
        cmpl    -8(%ebp), %eax
        jge     .LBB1_8
# %bb.6:                                #   in Loop: Header=BB1_5 Depth=1
        movl    -12(%ebp), %eax
        movl    -20(%ebp), %ecx
        movl    (%eax,%ecx,4), %eax
        leal    .L.str.1, %ecx
        movl    %ecx, (%esp)
        movl    %eax, 4(%esp)
        calll   printf
# %bb.7:                                #   in Loop: Header=BB1_5 Depth=1
        movl    -20(%ebp), %eax
        addl    $1, %eax
```

```
        movl    %eax, -20(%ebp)
        jmp     .LBB1_5
.LBB1_8:
        leal    .L.str.2, %eax
        movl    %eax, (%esp)
        calll   printf
        movl    -4(%ebp), %ecx
        movl    %eax, -28(%ebp)         # 4-byte Spill
        movl    %ecx, %eax
        addl    $40, %esp
        popl    %ebp
        retl
.Lfunc_end1:
        .size   main, .Lfunc_end1-main
                                        # -- End function
        .type   .L.str,@object          # @.str
        .section        .rodata.str1.1,"aMS",@progbits,1
.L.str:
        .asciz  "%d"
        .size   .L.str, 3

        .type   .L.str.1,@object        # @.str.1
.L.str.1:
        .asciz  " %d"
        .size   .L.str.1, 4

        .type   .L.str.2,@object        # @.str.2
.L.str.2:
        .asciz  "\n"
        .size   .L.str.2, 2

        .ident  "clang version 10.0.1 "
        .section        ".note.GNU-stack","",@progbits
        .addrsig
        .addrsig_sym my_sort
        .addrsig_sym __isoc99_scanf
        .addrsig_sym malloc
        .addrsig_sym printf
```

clang, 64位

特点在于:

- 用`movabsq`取标签的地址(而不是`lea`)
- 参数传递借助`%rdi`, `%rsi`, `%rdx`, `%rcx`, `%r8`, `%r9`等寄存器

- 需要设置`%eax`以提供对向量寄存器参数使用情况的说明
- 用`movslq`进行符号扩展的`mov`(gcc-64则使用`mov`后接`clt`命令来完成)

其余东西大同小异, 需要补充注释的地方并不多:

```
        .text
        .file   "sort.c"
        .globl  my_sort                 # -- Begin function my_sort
        .p2align        4, 0x90
        .type   my_sort,@function
my_sort:                                # @my_sort
        .cfi_startproc
# %bb.0:
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset %rbp, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register %rbp
        movq    %rdi, -8(%rbp)
        movl    %esi, -12(%rbp)
        movl    $0, -16(%rbp)
.LBB0_1:                                # =>This Loop Header: Depth=1
                                        #     Child Loop BB0_3 Depth 2
        movl    -16(%rbp), %eax
        cmpl    -12(%rbp), %eax
        jge     .LBB0_10
# %bb.2:                                #   in Loop: Header=BB0_1 Depth=1
        movl    -16(%rbp), %eax
        addl    $1, %eax
        movl    %eax, -20(%rbp)
.LBB0_3:                                #   Parent Loop BB0_1 Depth=1
                                        # =>  This Inner Loop Header: Depth=2
        movl    -20(%rbp), %eax
        cmpl    -12(%rbp), %eax
        jge     .LBB0_8
# %bb.4:                                #   in Loop: Header=BB0_3 Depth=2
        movq    -8(%rbp), %rax
        movslq  -16(%rbp), %rcx         # 使用`movslq`进行带符号扩展的mov
        movl    (%rax,%rcx,4), %edx
        movq    -8(%rbp), %rax
        movslq  -20(%rbp), %rcx         # 使用`movslq`进行带符号扩展的mov
        cmpl    (%rax,%rcx,4), %edx
        jle     .LBB0_6
# %bb.5:                                #   in Loop: Header=BB0_3 Depth=2
        movq    -8(%rbp), %rax
        movslq  -16(%rbp), %rcx         # 使用`movslq`进行带符号扩展的mov
        movl    (%rax,%rcx,4), %edx
        movl    %edx, -24(%rbp)
        movq    -8(%rbp), %rax
        movslq  -20(%rbp), %rcx         # 使用`movslq`进行带符号扩展的mov
        movl    (%rax,%rcx,4), %edx
```

```
        movq    -8(%rbp), %rax
        movslq  -16(%rbp), %rcx         # 使用`movslq`进行带符号扩展的mov
        movl    %edx, (%rax,%rcx,4)
        movl    -24(%rbp), %edx
        movq    -8(%rbp), %rax
        movslq  -20(%rbp), %rcx         # 使用`movslq`进行带符号扩展的mov
        movl    %edx, (%rax,%rcx,4)
.LBB0_6:                                #   in Loop: Header=BB0_3 Depth=2
        jmp     .LBB0_7
.LBB0_7:                                #   in Loop: Header=BB0_3 Depth=2
        movl    -20(%rbp), %eax
        addl    $1, %eax
        movl    %eax, -20(%rbp)
        jmp     .LBB0_3
.LBB0_8:                                #   in Loop: Header=BB0_1 Depth=1
        jmp     .LBB0_9
.LBB0_9:                                #   in Loop: Header=BB0_1 Depth=1
        movl    -16(%rbp), %eax
        addl    $1, %eax
        movl    %eax, -16(%rbp)
        jmp     .LBB0_1
.LBB0_10:
        popq    %rbp
        .cfi_def_cfa %rsp, 8
        retq
.Lfunc_end0:
        .size   my_sort, .Lfunc_end0-my_sort
        .cfi_endproc
                                        # -- End function
        .globl  main                    # -- Begin function main
        .p2align        4, 0x90
        .type   main,@function
main:                                   # @main
        .cfi_startproc
# %bb.0:
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset %rbp, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register %rbp
        subq    $32, %rsp
        movl    $0, -4(%rbp)
        movabsq $.L.str, %rdi           # 用movabsq取字符串地址
        leaq    -8(%rbp), %rsi
        movb    $0, %al
        callq   __isoc99_scanf
```

```
        movslq  -8(%rbp), %rcx         # 使用`movslq`进行带符号扩展的mov
        shlq    $2, %rcx
        movq    %rcx, %rdi
        movl    %eax, -28(%rbp)         # 4-byte Spill
        callq   malloc
        movq    %rax, -16(%rbp)
        movl    $0, -20(%rbp)
.LBB1_1:                                # =>This Inner Loop Header: Depth=1
        movl    -20(%rbp), %eax
        cmpl    -8(%rbp), %eax
        jge     .LBB1_4
# %bb.2:                                #   in Loop: Header=BB1_1 Depth=1
        movq    -16(%rbp), %rax
        movslq  -20(%rbp), %rcx         # 使用`movslq`进行带符号扩展的mov
        shlq    $2, %rcx
        addq    %rcx, %rax
        movabsq $.L.str, %rdi           # 用movabsq取字符串地址
        movq    %rax, %rsi
        movb    $0, %al
        callq   __isoc99_scanf
# %bb.3:                                #   in Loop: Header=BB1_1 Depth=1
        movl    -20(%rbp), %eax
        addl    $1, %eax
        movl    %eax, -20(%rbp)
        jmp     .LBB1_1
.LBB1_4:
        movq    -16(%rbp), %rdi
        movl    -8(%rbp), %esi
        callq   my_sort
        movq    -16(%rbp), %rax
        movl    (%rax), %esi
        movabsq $.L.str, %rdi           # 用movabsq取字符串地址
        movb    $0, %al
        callq   printf
        movl    $1, -24(%rbp)
.LBB1_5:                                # =>This Inner Loop Header: Depth=1
        movl    -24(%rbp), %eax
        cmpl    -8(%rbp), %eax
        jge     .LBB1_8
# %bb.6:                                #   in Loop: Header=BB1_5 Depth=1
        movq    -16(%rbp), %rax
        movslq  -24(%rbp), %rcx         # 使用`movslq`进行带符号扩展的mov
        movl    (%rax,%rcx,4), %esi
        movabsq $.L.str.1, %rdi           # 用movabsq取字符串地址
        movb    $0, %al
        callq   printf
```

```
# %bb.7:                                #   in Loop: Header=BB1_5 Depth=1
        movl    -24(%rbp), %eax
        addl    $1, %eax
        movl    %eax, -24(%rbp)
        jmp     .LBB1_5
.LBB1_8:
        movabsq $.L.str.2, %rdi           # 用movabsq取字符串地址
        movb    $0, %al
        callq   printf
        movl    -4(%rbp), %ecx
        movl    %eax, -32(%rbp)         # 4-byte Spill
        movl    %ecx, %eax
        addq    $32, %rsp
        popq    %rbp
        .cfi_def_cfa %rsp, 8
        retq
.Lfunc_end1:
        .size   main, .Lfunc_end1-main
        .cfi_endproc
                                        # -- End function
        .type   .L.str,@object          # @.str
        .section        .rodata.str1.1,"aMS",@progbits,1
.L.str:
        .asciz  "%d"
        .size   .L.str, 3

        .type   .L.str.1,@object        # @.str.1
.L.str.1:
        .asciz  " %d"
        .size   .L.str.1, 4

        .type   .L.str.2,@object        # @.str.2
.L.str.2:
        .asciz  "\n"
        .size   .L.str.2, 2

        .ident  "clang version 10.0.1 "
        .section        ".note.GNU-stack","",@progbits
        .addrsig
        .addrsig_sym my_sort
        .addrsig_sym __isoc99_scanf
        .addrsig_sym malloc
        .addrsig_sym printf
```