子程序的实现细节

源代码 main.c

```c
#include <stdio.h>

struct tuple5{
    int a0;
    int a1;
    int a2;
    int a3;
    int a4;
};

int func1(int a, int b){
    if(a > b){
        int c;
        c = a - b;
        return c;
    }else{
        int d;
        int c = 1;
        d = b - a;
        return d;
    }
    return 0;
}

struct tuple5 func2(int a, int b){
    struct tuple5 t;
    if(a > b){
        t.a0 = t.a1 = t.a2 = t.a3 = t.a4 = a;
    }else{
        t.a0 = t.a1 = t.a2 = t.a3 = t.a4 = b;
    }
    return t;
}

struct tuple5 func3(int a, int b){
    struct tuple5 t;
    if(a < b){
        t.a0 = t.a1 = t.a2 = t.a3 = t.a4 = a;
    }else{
        t.a0 = t.a1 = t.a2 = t.a3 = t.a4 = b;
    }
    return t;
}

int main(){
    int a = 0;
    a = func1(1, 2);
    struct tuple5 t1;
    t1 = func2(2, 3);
    struct tuple5 t2;
```

```
    t2 = func3(2, 3);
    printf("%d\n", a);
    printf("%d\n", t1.a0+t2.a0);
    return 0;
}
```

使用 -O0 选项

```
gcc -O0 -S main.c -o o0.s
```

```
    .file   "main.c"
    .text
    .globl  func1
    .def    func1;  .scl    2;  .type   32; .endef
    .seh_proc   func1
func1:
    pushq   %rbp
    .seh_pushreg    %rbp
    movq    %rsp, %rbp
    .seh_setframe   %rbp, 0
    subq    $16, %rsp
    .seh_stackalloc 16
    .seh_endprologue
    movl    %ecx, 16(%rbp)
    movl    %edx, 24(%rbp)
    movl    16(%rbp), %eax
    cmpl    24(%rbp), %eax
    jle .L2
    movl    16(%rbp), %eax
    subl    24(%rbp), %eax
    movl    %eax, -12(%rbp)
    movl    -12(%rbp), %eax
    jmp .L3
.L2:
    movl    $1, -4(%rbp)
    movl    24(%rbp), %eax
    subl    16(%rbp), %eax
    movl    %eax, -8(%rbp)
    movl    -8(%rbp), %eax
.L3:
    addq    $16, %rsp
    popq    %rbp
    ret
    .seh_endproc
    .globl  func2
    .def    func2;  .scl    2;  .type   32; .endef
    .seh_proc   func2
func2:
    pushq   %rbp
    .seh_pushreg    %rbp
    movq    %rsp, %rbp
    .seh_setframe   %rbp, 0
    subq    $32, %rsp
    .seh_stackalloc 32
```

```
    .seh_endprologue
    movq    %rcx, 16(%rbp)
    movl    %edx, 24(%rbp)
    movl    %r8d, 32(%rbp)
    movl    24(%rbp), %eax
    cmpl    32(%rbp), %eax
    jle .L5
    movl    24(%rbp), %eax
    movl    %eax, -16(%rbp)
    movl    -16(%rbp), %eax
    movl    %eax, -20(%rbp)
    movl    -20(%rbp), %eax
    movl    %eax, -24(%rbp)
    movl    -24(%rbp), %eax
    movl    %eax, -28(%rbp)
    movl    -28(%rbp), %eax
    movl    %eax, -32(%rbp)
    jmp .L6
.L5:
    movl    32(%rbp), %eax
    movl    %eax, -16(%rbp)
    movl    -16(%rbp), %eax
    movl    %eax, -20(%rbp)
    movl    -20(%rbp), %eax
    movl    %eax, -24(%rbp)
    movl    -24(%rbp), %eax
    movl    %eax, -28(%rbp)
    movl    -28(%rbp), %eax
    movl    %eax, -32(%rbp)
.L6:
    movq    16(%rbp), %rcx
    movq    -32(%rbp), %rax
    movq    -24(%rbp), %rdx
    movq    %rax, (%rcx)
    movq    %rdx, 8(%rcx)
    movl    -16(%rbp), %eax
    movl    %eax, 16(%rcx)
    movq    16(%rbp), %rax
    addq    $32, %rsp
    popq    %rbp
    ret
    .seh_endproc
    .globl  func3
    .def    func3;  .scl    2;  .type   32; .endef
    .seh_proc   func3
func3:
    pushq   %rbp
    .seh_pushreg    %rbp
    movq    %rsp, %rbp
    .seh_setframe   %rbp, 0
    subq    $32, %rsp
    .seh_stackalloc 32
    .seh_endprologue
    movq    %rcx, 16(%rbp)
    movl    %edx, 24(%rbp)
    movl    %r8d, 32(%rbp)
    movl    24(%rbp), %eax
    cmpl    32(%rbp), %eax
```

```
    jge .L9
    movl    24(%rbp), %eax
    movl    %eax, -16(%rbp)
    movl    -16(%rbp), %eax
    movl    %eax, -20(%rbp)
    movl    -20(%rbp), %eax
    movl    %eax, -24(%rbp)
    movl    -24(%rbp), %eax
    movl    %eax, -28(%rbp)
    movl    -28(%rbp), %eax
    movl    %eax, -32(%rbp)
    jmp .L10
.L9:
    movl    32(%rbp), %eax
    movl    %eax, -16(%rbp)
    movl    -16(%rbp), %eax
    movl    %eax, -20(%rbp)
    movl    -20(%rbp), %eax
    movl    %eax, -24(%rbp)
    movl    -24(%rbp), %eax
    movl    %eax, -28(%rbp)
    movl    -28(%rbp), %eax
    movl    %eax, -32(%rbp)
.L10:
    movq    16(%rbp), %rcx
    movq    -32(%rbp), %rax
    movq    -24(%rbp), %rdx
    movq    %rax, (%rcx)
    movq    %rdx, 8(%rcx)
    movl    -16(%rbp), %eax
    movl    %eax, 16(%rcx)
    movq    16(%rbp), %rax
    addq    $32, %rsp
    popq    %rbp
    ret
    .seh_endproc
    .def    __main; .scl    2;  .type   32; .endef
    .section .rdata,"dr"
.LC0:
    .ascii "%d\12\0"
    .text
    .globl  main
    .def    main;   .scl    2;  .type   32; .endef
    .seh_proc   main
main:
    pushq   %rbp
    .seh_pushreg    %rbp
    movq    %rsp, %rbp
    .seh_setframe   %rbp, 0
    subq    $96, %rsp
    .seh_stackalloc 96
    .seh_endprologue
    call    __main
    movl    $0, -4(%rbp)
    movl    $2, %edx
    movl    $1, %ecx
    call    func1
    movl    %eax, -4(%rbp)
```

```
    leaq    -32(%rbp), %rax
    movl    $3, %r8d
    movl    $2, %edx
    movq    %rax, %rcx
    call    func2
    leaq    -64(%rbp), %rax
    movl    $3, %r8d
    movl    $2, %edx
    movq    %rax, %rcx
    call    func3
    movl    -4(%rbp), %eax
    movl    %eax, %edx
    leaq    .LC0(%rip), %rcx
    call    printf
    movl    -32(%rbp), %edx
    movl    -64(%rbp), %eax
    addl    %edx, %eax
    movl    %eax, %edx
    leaq    .LC0(%rip), %rcx
    call    printf
    movl    $0, %eax
    addq    $96, %rsp
    popq    %rbp
    ret
    .seh_endproc
    .ident  "GCC: (GNU) 10.2.0"
    .def    printf; .scl    2;  .type   32; .endef
```

在函数 `func1()` 中有 `if` 语句, 可以看到, `if` 语句中定义的变量的空间是在 `if` 语句里分配的, 不同子句中的变量, 不管同名与否, 不会公用空间 (比如两个子句中的 `int c`, 一个在 `-12(%rbp)`, 一个在 `-4(%rbp)`)

```
    cmpl    24(%rbp), %eax
    jle .L2
    movl    16(%rbp), %eax
    subl    24(%rbp), %eax
    movl    %eax, -12(%rbp)
    movl    -12(%rbp), %eax
    jmp .L3
.L2:
    movl    $1, -4(%rbp)
    movl    24(%rbp), %eax
    subl    16(%rbp), %eax
    movl    %eax, -8(%rbp)
    movl    -8(%rbp), %eax
.L3:
```

函数 `func2()` 要返回一个比较大的结构体, 这里选择了直接对调用时的左值进行赋值 (先取了局部变量的地址, 然后装到 `rcx` 里, 在函数里对 `rcx` 所指向的空间进行赋值) 而没有用多余的地址保存返回值

```
.L6:
    movq    16(%rbp), %rcx
    movq    -32(%rbp), %rax
    movq    -24(%rbp), %rdx
    movq    %rax, (%rcx)
    movq    %rdx, 8(%rcx)
    movl    -16(%rbp), %eax
    movl    %eax, 16(%rcx)
    movq    16(%rbp), %rax
    addq    $32, %rsp
    popq    %rbp
    ret
```

`main()`函数要调用两个返回较大结构体的函数, 可以看到编译器对此没有什么特殊的处理

```
leaq    -32(%rbp), %rax
movl    $3, %r8d
movl    $2, %edx
movq    %rax, %rcx
call    func2
leaq    -64(%rbp), %rax
movl    $3, %r8d
movl    $2, %edx
movq    %rax, %rcx
call    func3
```

使用`-O2`选项

```
gcc -O2 -S main.c -o o2.s
```

```
    .file   "main.c"
    .text
    .p2align 4
    .globl  func1
    .def    func1;  .scl    2;  .type   32; .endef
    .seh_proc   func1
func1:
    .seh_endprologue
    movl    %ecx, %r8d
    movl    %edx, %eax
    subl    %edx, %r8d
    subl    %ecx, %eax
    cmpl    %edx, %ecx
    cmovg   %r8d, %eax
    ret
    .seh_endproc
    .p2align 4
    .globl  func2
    .def    func2;  .scl    2;  .type   32; .endef
    .seh_proc   func2
func2:
    .seh_endprologue
    cmpl    %r8d, %edx
```

```
    movq    %rcx, %rax
    cmovl   %r8d, %edx
    movl    %edx, (%rcx)
    movl    %edx, 4(%rcx)
    movl    %edx, 8(%rcx)
    movl    %edx, 12(%rcx)
    movl    %edx, 16(%rcx)
    ret
    .seh_endproc
    .p2align 4
    .globl  func3
    .def    func3;  .scl    2;  .type   32; .endef
    .seh_proc   func3
func3:
    .seh_endprologue
    cmpl    %r8d, %edx
    movq    %rcx, %rax
    cmovg   %r8d, %edx
    movl    %edx, (%rcx)
    movl    %edx, 4(%rcx)
    movl    %edx, 8(%rcx)
    movl    %edx, 12(%rcx)
    movl    %edx, 16(%rcx)
    ret
    .seh_endproc
    .def    __main; .scl    2;  .type   32; .endef
    .section .rdata,"dr"
.LC0:
    .ascii "%d\12\0"
    .section    .text.startup,"x"
    .p2align 4
    .globl  main
    .def    main;   .scl    2;  .type   32; .endef
    .seh_proc   main
main:
    subq    $40, %rsp
    .seh_stackalloc 40
    .seh_endprologue
    call    __main
    movl    $1, %edx
    leaq    .LC0(%rip), %rcx
    call    printf
    movl    $5, %edx
    leaq    .LC0(%rip), %rcx
    call    printf
    xorl    %eax, %eax
    addq    $40, %rsp
    ret
    .seh_endproc
    .ident  "GCC: (GNU) 10.2.0"
    .def    printf; .scl    2;  .type   32; .endef
```

在函数 `func1()` 中有 `if` 语句, 可以看到 `-O2` 中, 并没有给局部变量分配内存空间, 仅仅利用了寄存器

```
func1:
    .seh_endprologue
    movl    %ecx, %r8d
    movl    %edx, %eax
    subl    %edx, %r8d
    subl    %ecx, %eax
    cmpl    %edx, %ecx
    cmovg   %r8d, %eax
    ret
```

在函数`func2()`中要返回一个比较大的结构体, 这时处理的基本思路还是直接对调用者中要被赋值的局部变量的空间进行数据填充, 但是做了许多优化, 没有使用额外的空间, 也把指令精简了许多

```
func2:
    .seh_endprologue
    cmpl    %r8d, %edx
    movq    %rcx, %rax
    cmovl   %r8d, %edx
    movl    %edx, (%rcx)
    movl    %edx, 4(%rcx)
    movl    %edx, 8(%rcx)
    movl    %edx, 12(%rcx)
    movl    %edx, 16(%rcx)
    ret
```

`main()`函数要调用两个返回较大结构体的函数, 可以看到编译器做了很大的优化

```
main:
    subq    $40, %rsp
    .seh_stackalloc 40
    .seh_endprologue
    call    __main
    movl    $1, %edx
    leaq    .LC0(%rip), %rcx
    call    printf
    movl    $5, %edx
    leaq    .LC0(%rip), %rcx
    call    printf
    xorl    %eax, %eax
    addq    $40, %rsp
    ret
```

把`-O0`和`-O2`对比可以发现, `-O2`的优化程度远远大于`-O0`