汇编代码及注释

```
.file  "lab.c"    ;打开文件
        .text     ;
        .globl      add2      ;声明 add2 要被连接器用到（.global 声明）
        .type       add2, @function      ;定义函数 add2
add2:
.LFB0:    ;函数的开头生成的标签
        .cfi_startproc       ;用在每个函数的开始，用于初始化一些内部数据
        endbr64       ;终止 64 位间接分支
        pushq     %rbp       ;将父函数的栈帧指针压栈
        .cfi_def_cfa_offset 16       ;此处距离 CFA 地址为 16 字节
        .cfi_offset 6, -16       ;把第 6 号寄存器原先的值保存在距离 CFA - 16 的位置
        movq      %rsp, %rbp       ;使 %rsp 和 %rbp 指向同一位置，将栈中保存的父栈帧的 %rbp 的值赋值给 %rbp，并且 %rsp 上移一个位置指向父栈帧的结尾处
        .cfi_def_cfa_register 6       ;从这里开始，使用 rbp 作为计算 CFA 的基址寄存器(前面用的是 rsp)
        movl      %edi, -4(%rbp)       ;约定%edi 为第一个参数，保存到-4(%rbp)的位置
        movl      %esi, -8(%rbp)       ;约定%esi 为第一个参数，保存到-8(%rbp)的位置
        movl      -4(%rbp), %edx       ;-4(%rbp)保存到%edx
        movl      -8(%rbp), %eax       ;-8(%rbp)保存到%eax
        addl %edx, %eax    ;计算 edx+eax
        popq      %rbp    ;恢复 rip 寄存器，将程序控制权交给调用者
        .cfi_def_cfa 7, 8    ;现在重新定义 CFA，它的值是第 7 号寄存器所指位置加 8 字节
        ret     ;弹出返回地址到 eip 中,从被调用函数返回到调用函数
        .cfi_endproc    ;函数结束
.LFE0:     ;函数的结尾生成的标签
        .size add2, .-add2    ;设置与 add2 有关的空间大小
        .globl      add3       ;声明 add3 要被连接器用到（.global 声明）
        .type       add3, @function       ;定义函数 add3
add3:
.LFB1:     ;函数的开头生成的标签
        .cfi_startproc    ;用在每个函数的开始，用于初始化一些内部数据
        endbr64    ;终止 64 位间接分支
        pushq     %rbp    ；将父函数的栈帧指针压栈
        .cfi_def_cfa_offset 16     ;此处距离 CFA 地址为 16 字节
        .cfi_offset 6, -16     ;把第 6 号寄存器原先的值保存在距离 CFA - 16 的位置
        movq      %rsp, %rbp    ;使 %rsp 和 %rbp 指向同一位置，将栈中保存的父栈帧的 %rbp 的值赋值给 %rbp，并且 %rsp 上移一个位置指向父栈帧的结尾处
        .cfi_def_cfa_register 6    ;    ;从这里开始，使用 rbp 作为计算 CFA 的基址寄存器(前面用的是 rsp)
        subq      $16, %rsp     ;申请 16 字节的栈空间
        movl      %edi, -4(%rbp)     ;约定%edi 为参数，保存到-4(%rbp)的位置
        movl      %esi, -8(%rbp)     ;约定%esi 为参数，保存到-8(%rbp)的位置
        movl      %edx, -12(%rbp)     ;约定%edi 为参数，保存到-12(%rbp)的位置
```

```
    movl    -12(%rbp), %edx     ;edx=[rbp-12]
    movl    -8(%rbp), %eax    ;eax=[rbp-8]
    movl    %edx, %esi    ;esi=edx
    movl    %eax, %edi    ;edi=eax
    call  add2      ；调用函数 add2
    movl    -4(%rbp), %edx     ;edx=[rbp-4]
    addl%edx, %eax      ;计算 edx+eax
    leave   ;执行完这两条指令后，ebp 恢复为旧的 ebp，即指向调用者的基址，esp 则指向返回地址。
    .cfi_def_cfa 7, 8   ；;现在重新定义 CFA，它的值是第 7 号寄存器所指位置加 8 字节
    ret   ;弹出返回地址到 eip 中,从被调用函数返回到调用函数
    .cfi_endproc     ;函数结束
.LFE1:      ;函数的结尾生成的标签
    .size add3, .-add3     ;设置与 add2 有关的空间大小
    .globl     main        ;声明 main 要被连接器用到（.global 声明）
    .type      main, @function     ;定义函数 main
main:
.LFB2:   ;函数的开头生成的标签
    .cfi_startproc   ;用在每个函数的开始，用于初始化一些内部数据
    endbr64   ;终止 64 位间接分支
    pushq    %rbp   ；将父函数的栈帧指针压栈
    .cfi_def_cfa_offset 16   ;此处距离 CFA 地址为 16 字节
    .cfi_offset 6, -16   ;把第 6 号寄存器原先的值保存在距离 CFA - 16 的位置
    movq    %rsp, %rbp     ;使 %rsp 和 %rbp 指向同一位置，将栈中保存的父栈帧的 %rbp 的值赋值给 %rbp，并且 %rsp 上移一个位置指向父栈帧的结尾处
    .cfi_def_cfa_register 6     ;从这里开始，使用 rbp 作为计算 CFA 的基址寄存器(前面用的是 rsp)
    subq      $16, %rsp    ;申请 16 字节的栈空间
    movl      $1, -8(%rbp)    ;-8(%rbp)的位置存 1
    movl      $2, -4(%rbp)    ;-4(%rbp)的位置存 2
    movl      -4(%rbp), %ecx       ;-4(%rbp)保存到%ecx
    movl      -8(%rbp), %eax       ;-8(%rbp)保存到%eax
    movl      $3, %edx     ;%edx 的位置存 3
    movl      %ecx, %esi    ;ecx=esi
    movl      %eax, %edi    ;edi=eax
    call  add3    ;调用 add3 函数
    leave   ;执行完这两条指令后，ebp 恢复为旧的 ebp，即指向调用者的基址，esp 则指向返回地址。
    .cfi_def_cfa 7, 8   ； 现在重新定义 CFA，它的值是第 7 号寄存器所指位置加 8 字节
    ret   ;弹出返回地址到 eip 中,从被调用函数返回到调用函数
    .cfi_endproc     ;函数结束
.LFE2:     ;函数的结尾生成的标签
    .size main, .-main
    .ident     "GCC: (Ubuntu 9.4.0-1ubuntu1~20.04.1) 9.4.0"    ;认证为 Ubuntu 系统里的 gcc
```

编译

```
    .section   .note.GNU-stack,"",@progbits      ;定义一个数据段，段名为 note.GNU-stack, progbits 为自定义数据段
    .section   .note.gnu.property,"a"   ;定义一个数据段，段名为 note.gnu.property, 缺省的标志为 a
    .align 8     ;按 8 字节对齐
    .long       1f - 0f    ;long 声明
    .long       4f - 1f    ;long 声明
    .long       5     ;long 声明
0:
    .string     "GNU"    ;使用 GNU 操作系统
1:
    .align 8
    .long       0xc0000002    ;声明一个占 32 位空间的数
    .long       3f - 2f    ;long 声明
2:
    .long       0x3    ;long 声明
3:
```

gdb 调试:

```
ubuntu@VM5750-for-linux:/home/ubuntu/桌面/未命名文件夹$ gcc lab.s -g -o lab
ubuntu@VM5750-for-linux:/home/ubuntu/桌面/未命名文件夹$ gdb lab
GNU gdb (Ubuntu 9.2-0ubuntu1~20.04.1) 9.2
Copyright (C) 2020 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from lab...
(gdb) b 21
Breakpoint 1 at 0x1140: file lab.s, line 21.
(gdb) b 30
Breakpoint 2 at 0x1141: file lab.s, line 30.
(gdb)
```

```
(gdb) i b
Num     Type           Disp Enb Address            What
1       breakpoint     keep y   0x0000000000001140 lab.s:21
2       breakpoint     keep y   0x0000000000001141 lab.s:30
(gdb) r
Starting program: /home/ubuntu/桌面/未命名文件夹/lab

Breakpoint 2, add3 () at lab.s:30
30              endbr64
(gdb) i r
rax            0x1                 1
rbx            0x5555555551a0      93824992235936
rcx            0x2                 2
rdx            0x3                 3
rsi            0x2                 2
rdi            0x1                 1
rbp            0x7fffffffdd20      0x7fffffffdd20
rsp            0x7fffffffdd08      0x7fffffffdd08
r8             0x0                 0
r9             0x7ffff7fe0d50      140737354009936
r10            0xf                 15
r11            0x2                 2
r12            0x555555555040      93824992235584
r13            0x7fffffffde10      140737488346640
r14            0x0                 0
r15            0x0                 0
rip            0x555555555141      0x555555555141 <add3>
eflags         0x202               [ IF ]
cs             0x33                51
ss             0x2b                43
ds             0x0                 0
es             0x0                 0
fs             0x0                 0
--Type <RET> for more, q to quit, c to continue without paging--
```

```
(gdb) p $rax=9
$4 = 9
(gdb) ir
Undefined command: "ir".  Try "help".
(gdb) i r
rax            0x9                 9
rbx            0x5555555551a0      93824992235936
rcx            0x2                 2
rdx            0x3                 3
rsi            0x2                 2
rdi            0x1                 1
rbp            0x7fffffffdd20      0x7fffffffdd20
rsp            0x7fffffffdd08      0x7fffffffdd08
```

```
(gdb) bt
#0  add2 () at lab.s:14
#1  0x0000555555555165 in add3 () at lab.s:44
#2  0x000055555555519a in main () at lab.s:72
(gdb) n
15              movl    %esi, -8(%rbp)
(gdb) n
16              movl    -4(%rbp), %edx
(gdb) n
17              movl    -8(%rbp), %eax
(gdb) n
18              addl    %edx, %eax
(gdb) n
19              popq    %rbp
(gdb) bt
#0  add2 () at lab.s:19
#1  0x0000555555555165 in add3 () at lab.s:44
#2  0x000055555555519a in main () at lab.s:72
(gdb) d 1
(gdb) i b
Num     Type           Disp Enb Address            What
2       breakpoint     keep y   0x0000555555555140 lab.s:21
(gdb) bt
```

其中查看函数调用堆栈这一环节前后做了三遍：
第一次因为设置断点设在了 add3 的程序段，所以查看的栈是在 add3 函数下查看的（也可能是我理解的不对）；第二次仅仅使用了 gdb 调试指令但没有启动程序，所以使用查看寄存器或者栈时提示"The program has no register now"以及"No stack"，最后一次才让 add2 显示出来了。

附加题
源程序：

```
int funtest(int n1, int n2, int n3, int n4, int n5, int n6, int n7, int n8, int n9)
{
    return n1 + n2 + n3 + n4 + n5 + n6 + n7 + n8 + n9;
}
int main()
{
    return funtest(1, 2, 3, 4, 5, 6, 7, 8, 9);
}
```

编译调试：

```
ubuntu@VM5750-for-linux:/home/ubuntu/桌面/未命名文件夹$ gcc -m64 -g funtest.c -o funtest64
ubuntu@VM5750-for-linux:/home/ubuntu/桌面/未命名文件夹$ gdb funtest64
GNU gdb (Ubuntu 9.2-0ubuntu1~20.04.1) 9.2
Copyright (C) 2020 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from funtest64...
```

```
(gdb) disassemble funtest
Dump of assembler code for function funtest:
   0x0000555555555129 <+0>:     endbr64
   0x000055555555512d <+4>:     push   %rbp
   0x000055555555512e <+5>:     mov    %rsp,%rbp
   0x0000555555555131 <+8>:     mov    %edi,-0x4(%rbp)
   0x0000555555555134 <+11>:    mov    %esi,-0x8(%rbp)
   0x0000555555555137 <+14>:    mov    %edx,-0xc(%rbp)
   0x000055555555513a <+17>:    mov    %ecx,-0x10(%rbp)
   0x000055555555513d <+20>:    mov    %r8d,-0x14(%rbp)
   0x0000555555555141 <+24>:    mov    %r9d,-0x18(%rbp)
   0x0000555555555145 <+28>:    mov    -0x4(%rbp),%edx
   0x0000555555555148 <+31>:    mov    -0x8(%rbp),%eax
   0x000055555555514b <+34>:    add    %eax,%edx
   0x000055555555514d <+36>:    mov    -0xc(%rbp),%eax
   0x0000555555555150 <+39>:    add    %eax,%edx
   0x0000555555555152 <+41>:    mov    -0x10(%rbp),%eax
   0x0000555555555155 <+44>:    add    %eax,%edx
   0x0000555555555157 <+46>:    mov    -0x14(%rbp),%eax
   0x000055555555515a <+49>:    add    %eax,%edx
   0x000055555555515c <+51>:    mov    -0x18(%rbp),%eax
   0x000055555555515f <+54>:    add    %eax,%edx
   0x0000555555555161 <+56>:    mov    0x10(%rbp),%eax
   0x0000555555555164 <+59>:    add    %eax,%edx
   0x0000555555555166 <+61>:    mov    0x18(%rbp),%eax
   0x0000555555555169 <+64>:    add    %eax,%edx
   0x000055555555516b <+66>:    mov    0x20(%rbp),%eax
   0x000055555555516e <+69>:    add    %edx,%eax
   0x0000555555555170 <+71>:    pop    %rbp
   0x0000555555555171 <+72>:    retq
End of assembler dump.
(gdb)
```

（上下为同一张图片）


```
(gdb) disassemble funtest
Dump of assembler code for function funtest:
   0x0000555555555129 <+0>:     endbr64
   0x000055555555512d <+4>:     push   %rbp
   0x000055555555512e <+5>:     mov    %rsp,%rbp
   0x0000555555555131 <+8>:     mov    %edi,-0x4(%rbp)
   0x0000555555555134 <+11>:    mov    %esi,-0x8(%rbp)
   0x0000555555555137 <+14>:    mov    %edx,-0xc(%rbp)
   0x000055555555513a <+17>:    mov    %ecx,-0x10(%rbp)
   0x000055555555513d <+20>:    mov    %r8d,-0x14(%rbp)
   0x0000555555555141 <+24>:    mov    %r9d,-0x18(%rbp)
   0x0000555555555145 <+28>:    mov    -0x4(%rbp),%edx
   0x0000555555555148 <+31>:    mov    -0x8(%rbp),%eax
   0x000055555555514b <+34>:    add    %eax,%edx
   0x000055555555514d <+36>:    mov    -0xc(%rbp),%eax
   0x0000555555555150 <+39>:    add    %eax,%edx
   0x0000555555555152 <+41>:    mov    -0x10(%rbp),%eax
   0x0000555555555155 <+44>:    add    %eax,%edx
   0x0000555555555157 <+46>:    mov    -0x14(%rbp),%eax
   0x000055555555515a <+49>:    add    %eax,%edx
   0x000055555555515c <+51>:    mov    -0x18(%rbp),%eax
   0x000055555555515f <+54>:    add    %eax,%edx
   0x0000555555555161 <+56>:    mov    0x10(%rbp),%eax
   0x0000555555555164 <+59>:    add    %eax,%edx
   0x0000555555555166 <+61>:    mov    0x18(%rbp),%eax
   0x0000555555555169 <+64>:    add    %eax,%edx
   0x000055555555516b <+66>:    mov    0x20(%rbp),%eax
   0x000055555555516e <+69>:    add    %edx,%eax
   0x0000555555555170 <+71>:    pop    %rbp
   0x0000555555555171 <+72>:    retq
End of assembler dump.
(gdb)
```

分析上图可知：函数调用时传递了 9 个参数，其中一个用栈，其他 8 个用寄存器（存入 rbp 寄存器的对应位置）

执行“`return n1 + n2 + n3 + n4 + n5 + n6 + n7 + n8 + n9;`”指令时，mov,add 指令依次执行

用 eax 寄存器存放需要新加入的数据（即 rbp 寄存器指定位置存放的数据），用 edx 存放当前加法的计算结果，依次对 eax,edx 中的数据作 add 操作，最终得到所需结果。