#include <stdio.h>

#include <stdlib.h>

void func3(int c)

{

return c;

}

void func2(void)

{

int ii = 0;

func3(8);

return 2;

}

int func1(int a, int b)

{

int i = 0;

int j = 0;

i = i+a;

j = j+b;

func2();

return (i+j);

}

void main(void)

{

int ret = 0;

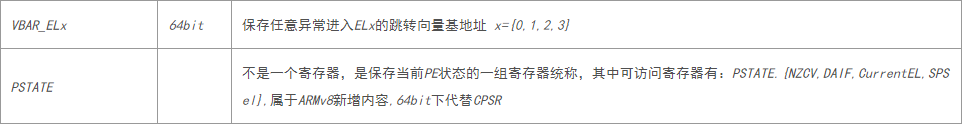
ret = func1(1, 2);

}

===========================================================

*AArch64*重要寄存器





X29：FP寄存器；

===========================================================

call1.o: file format elf64-littleaarch64

Disassembly of section .text:

0000000000000000 <func3>:

0: d10043ff sub sp, sp, #0x10

4: b9000fe0 str w0, [sp, #12]

8: d503201f nop

c: 910043ff add sp, sp, #0x10

10: d65f03c0 ret

0000000000000014 <func2>:

14: a9be7bfd stp x29, x30, [sp, #-32]!

18: 910003fd mov x29, sp

1c: b9001fbf str wzr, [x29, #28]

20: 52800100 mov w0, #0x8 // #8

24: 94000000 bl 0 <func3>

28: d503201f nop

2c: a8c27bfd ldp x29, x30, [sp], #32

30: d65f03c0 ret

0000000000000034 <func1>:

34: a9bd7bfd stp x29, x30, [sp, #-48]!

38: 910003fd mov x29, sp

3c: b9001fa0 str w0, [x29, #28]

40: b9001ba1 str w1, [x29, #24]

44: b9002fbf str wzr, [x29, #44]

48: b9002bbf str wzr, [x29, #40]

4c: b9402fa1 ldr w1, [x29, #44]

50: b9401fa0 ldr w0, [x29, #28]

54: 0b000020 add w0, w1, w0

58: b9002fa0 str w0, [x29, #44]

5c: b9402ba1 ldr w1, [x29, #40]

60: b9401ba0 ldr w0, [x29, #24]

64: 0b000020 add w0, w1, w0

68: b9002ba0 str w0, [x29, #40]

6c: 94000000 bl 14 <func2>

70: b9402fa1 ldr w1, [x29, #44]

74: b9402ba0 ldr w0, [x29, #40]

78: 0b000020 add w0, w1, w0

7c: a8c37bfd ldp x29, x30, [sp], #48

80: d65f03c0 ret

0000000000000084 <main>:

84: a9be7bfd stp x29, x30, [sp, #-32]! //拉伸栈空间32个字节，存放x29和x30

88: 910003fd mov x29, sp //将sp寄存器值赋给x29

8c: b9001fbf str wzr, [x29, #28] //读取wzr寄存器值存到x29+28内存地址处

90: 52800041 mov w1, #0x2 // #2 实参保存到w1寄存器

94: 52800020 mov w0, #0x1 // #1 实参保存到w0寄存器

98: 94000000 bl 34 <func1> //跳转到func1

9c: b9001fa0 str w0, [x29, #28] //读取w0寄存器地址放到x29+28内存地址处，w0寄存器存放的是子函数返回值

a0: d503201f nop

a4: a8c27bfd ldp x29, x30, [sp], #32 //将sp偏移32个字节的值存放到x29,x30中

a8: d65f03c0 ret

Disassembly of section .comment:

0000000000000000 <.comment>:

0: 43434700 .inst 0x43434700 ; undefined

4: 4828203a .inst 0x4828203a ; undefined

8: 20432643 .inst 0x20432643 ; undefined

c: 30303156 adr x22, 60635 <main+0x605b1>

10: 32303052 orr w18, w2, #0x1fff0000

14: 42303043 .inst 0x42303043 ; undefined

18: 5f323130 .inst 0x5f323130 ; undefined

1c: 38313032 ldsetb w17, w18, [x1]

20: 31303630 adds w16, w17, #0xc0d

24: 2e362029 usubl v9.8h, v1.8b, v22.8b

28: 00302e33 .inst 0x00302e33 ; NYI

==============================================================

栈就是指令执行时存放临时变量的内存空间，具有特殊的访问方式：

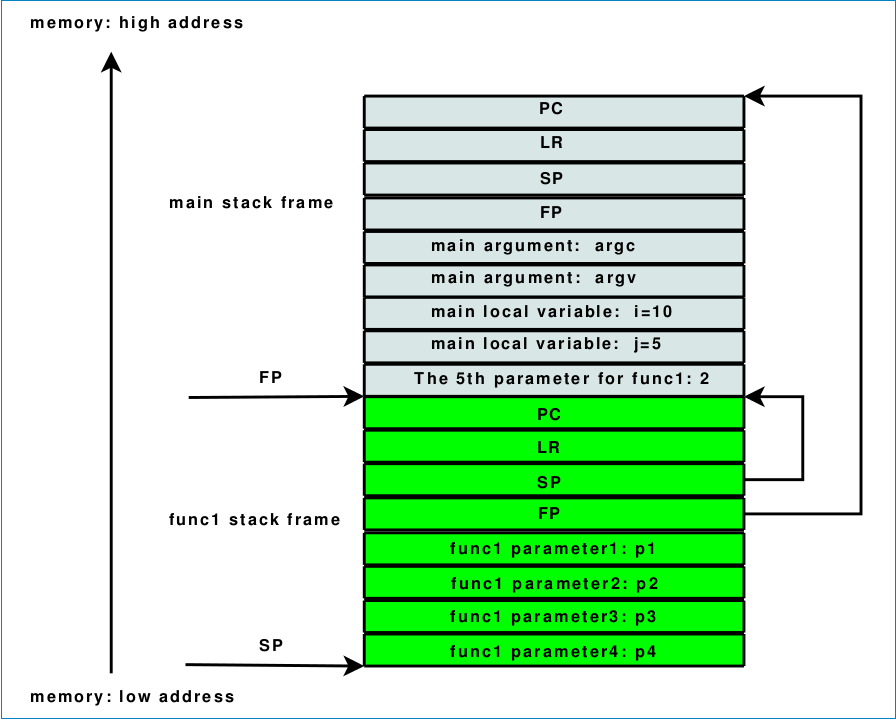
后进先出，Last In Out First.

栈是从高地址到低地址存储数据的，栈底在高地址，栈顶在低地址；

FP指向栈底

SP指向栈顶

[“stack frame是一个函数所使用的stack的一部分，所有函数的stack frame串起来就组成了一个完整的栈。stack frame的两个边界分别由FP和SP来限定”]



MOV

mov X1, X0；将寄存器X0的值传送到寄存器X1

STR

store register写入指令

将数据从寄存器中读出来，写入到内存中

STR(条件) 源寄存器, <存储器地址>

STR R0，[R1]，意思是R0->[R1],它把源寄存器写在前面，跟MOV/LDR都相反。

ldr

load register 读取指令

将数据从内存中读取出来，存到寄存器中，

LDR：通常都是作加载指令的，但是它也可以作伪指令

LDR r0,[r1] //将r1中的值存到r0中

LDR r1,[r2,#16] //将r2+16地址中的内容存到r1中

LDR r1,[r2],#4 //将r2地址中的内容存到r1中，同时r2=r2+4

ldp/stp

是ldr/str的衍生，可以同时读/写两个寄存器，ldr/str只能读写一个

sub sp, sp, #0x20; 拉伸栈空间32(20=2\*16)个字节

stp x0, x1, [sp, #0x10]; sp往上加16个字节，存放x0和x1

ldp x1, x0, [sp, #0x10]; 将sp偏移16个字节的值取出来，放入x1和x0

CMP

cmp是比较指令，cmp的功能相当于减法指令。

它不保存结果，只是影响相应的标志位

cmp 指令格式：cmp 操作对象1, 操作对象2

 mov al, 05h  
 mov bl, 08h  
 cmp al, bl   (al - bl = -3h)

bl:跳转指令

ret: 返回指令

ret {Xm}: 跳转到由Xm目标寄存器指定的地址处。

e: equal

ne: not equal

b: below

nb: not below

a: above

na: not above