Sorry, my compiler knowledge has been obsolete.

The current compiler is very smart to handle array arguments. Please forget my concern on code review meeting.

I use the following code to check our discussion.

```
1.  int test(int array[4])
2.  {
3.      array[0] = 0;
4.      array[1] = 1;
5.      array[2] = 2;
6.      array[3] = 3;
7.      return 0;
8.  }
9.
10. int main()
11. {
12.     int arr[4] = {3, 2, 1, 0};
13.
14.     test(arr);
15.
16.     return 0;
17. }
```

> arm-linux-gnueabi-gcc -O0 -g -o test-arm test.c
> arm-linux-gnueabi-objdump -dS test-arm > test-arm.asm

The following is assembly code

```
int test(int array[4])
{
    8444:    e52db004    push    {fp}        ; (str fp, [sp, #-4]!)
    8448:    e28db000    add     fp, sp, #0
    844c:    e24dd00c    sub     sp, sp, #12
    8450:    e50b0008    str     r0, [fp, #-8]        ⑥
    array[0] = 0;
    8454:    e51b3008    ldr     r3, [fp, #-8]        ⑦
    8458:    e3a02000    mov     r2, #0
    845c:    e5832000    str     r2, [r3]
    array[1] = 1;
    8460:    e51b3008    ldr     r3, [fp, #-8]
    8464:    e2833004    add     r3, r3, #4
    8468:    e3a02001    mov     r2, #1
    846c:    e5832000    str     r2, [r3]
    array[2] = 2;
    8470:    e51b3008    ldr     r3, [fp, #-8]
    8474:    e2833008    add     r3, r3, #8
    8478:    e3a02002    mov     r2, #2
    847c:    e5832000    str     r2, [r3]
    array[3] = 3;
    8480:    e51b3008    ldr     r3, [fp, #-8]
    8484:    e283300c    add     r3, r3, #12
    8488:    e3a02003    mov     r2, #3
    848c:    e5832000    str     r2, [r3]
    return 0;
    8490:    e3a03000    mov     r3, #0
}
    8494:    e1a00003    mov     r0, r3
    8498:    e28bd000    add     sp, fp, #0
    849c:    e8bd0800    ldmfd   sp!, {fp}
    84a0:    e12fff1e    bx      lr

000084a4 <main>:

int main()
{
    84a4:    e92d4800    push    {fp, lr}
    84a8:    e28db004    add     fp, sp, #4
    84ac:    e24dd018    sub     sp, sp, #24
    int arr[4] = {3, 2, 1, 0};
    84b0:    e59f3040    ldr     r3, [pc, #64]    ; 84f8 <main+0x54>
    84b4:    e24bc014    sub     ip, fp, #20                          ①
    84b8:    e893000f    ldm     r3, {r0, r1, r2, r3}                 ②
    84bc:    e88c000f    stm     ip, {r0, r1, r2, r3}                 ③

    test(arr);
    84c0:    e24b3014    sub     r3, fp, #20                          ④
    84c4:    e1a00003    mov     r0, r3                               ⑤
    84c8:    ebffffdd    bl      8444 <test>

    ......

```

```
54.     return 0;
55.     84e8:    e3a03000     mov    r3, #0
56. }
57.     84ec:    e1a00003     mov    r0, r3
58.     84f0:    e24bd004     sub    sp, fp, #4
59.     84f4:    e8bd8800     pop    {fp, pc}
```

①②③

在stack上分配arr[4]空间

fp - 20 = &arr[0]

④⑤

调用test()传递的是fp - 20,也就是arr[4]的首地址，不是我们担心的把整个arr[4]压栈(compiler足够智能了)

r0就受第一个参数

⑥

test()接收到的确实是arr[4]的首地址（在r0中），把r0赋值给临时变量[fp, #-8]

⑦

对arr[0]赋值。