

3130101796 蔡武威

仅一台 macbook

macbook

1. 生成了 Thumb 指令还是 ARM 指令

- C 代码

[illegible]

- 通过编译参数改变，使得其编译成 ARM 指令或 thumb 指令。

➤ 默认为 ARM 指令

```
/Volumes/cross-compilation/bin/arm-none-linux-gnueabi-gcc hello.c -o hello.a
```

- 加上参数-mthumb 编译为 thumb 指令

```
/Volumes/cross-compilation/bin/arm-none-linux-gnueabi-gcc hello.c -mthumb -o hello.t
```

- ## ● 汇编代码情况

用 `arm-linux-gnueabi-objdump -d` 命令查看汇编码

ARM

```

0000842c <main>:
 842c:    e92d4800    push    {fp, lr}
 8430:    e28db004    add     fp, sp, #4
 8434:    e59f000c    ldr     r0, [pc, #12] ; 8448 <main+0x1c>
 8438:    ebffffcd    bl      8374 <_init+0x44>
 843c:    e3a03000    mov     r3, #0
 8440:    e1a00003    mov     r0, r3
 8444:    e8bd8800    pop     {fp, pc}
 8448:    00008528    .word   0x00008528

```

➤ Thumb

```

0000842c <main>:
 842c:    b580    push    {r7, lr}
 842e:    af00    add     r7, sp, #0
 8430:    4b03    ldr     r3, [pc, #12] ; (8440 <main+0x14>)
 8432:    1c18    adds   r0, r3, #0
 8434:    f7ff ef9e    blx    8374 <_init+0x44>
 8438:    2300    movs   r3, #0
 843a:    1c18    adds   r0, r3, #0
 843c:    46bd    mov     sp, r7
 843e:    bd80    pop     {r7, pc}
 8440:    00008520    .word   0x00008520

```

对比上述汇编可知，ARM 和 Thumb 编译的结果最大的不同是 32 位与 16 位的区别，且 Thumb 的指令数比 ARM 多，但整体上，ARM 汇编结果比 Thumb 的大（ls -l 的结果如下）

```

[→ ESlab2 ls -l
total 40
-rwxr-xr-x  1 scn3  staff  5725  4  3 13:11 hello.a
-rw-r--r--  1 scn3  staff   320  4  3 13:00 hello.c
-rwxr-xr-x  1 scn3  staff  5692  4  3 13:11 hello.t

```

2. 对于 ARM 指令，能否产生条件执行的指令

● C 代码

```

7
8 #include<stdio.h>
9 int main(void) {
10     int a = 2;
11     int b = 3;
12     if (a > b)
13         printf("a is larger");
14     else
15         printf("b is larger");
16     return 0;
17 }

```

● ARM 指令

```

0000842c <main>:
842c: e92d4800 push    {fp, lr}
8430: e28db004 add     fp, sp, #4
8434: e24dd008 sub     sp, sp, #8
8438: e3a03002 mov     r3, #2
843c: e50b300c str     r3, [fp, #-12]
8440: e3a03003 mov     r3, #3
8444: e50b3008 str     r3, [fp, #-8]
8448: e51b200c ldr     r2, [fp, #-12]
844c: e51b3008 ldr     r3, [fp, #-8]
8450: e1520003 cmp     r2, r3
8454: da000003 ble     8468 <main+0x3c>
8458: e59f3024 ldr     r3, [pc, #36] ; 8484 <main+0x58>
845c: e1a00003 mov     r0, r3
8460: ebffffc3 bl      8374 <_init+0x44>
8464: ea000002 b       8474 <main+0x48>
8468: e59f3018 ldr     r3, [pc, #24] ; 8488 <main+0x5c>
846c: e1a00003 mov     r0, r3
8470: ebffffbf bl      8374 <_init+0x44>
8474: e3a03000 mov     r3, #0
8478: e1a00003 mov     r0, r3
847c: e24bd004 sub     sp, fp, #4
8480: e8bd8800 pop     {fp, pc}
8484: 00008568 .word   0x00008568
8488: 00008574 .word   0x00008574

```

观察 8454 的 ble 可知，能产生条件执行的指令。

3. 设计 C 的代码场景，观察是否产生了寄存器移位寻址。

- C 代码


```
ESlab2 — scn3@scn3deMacBook-Pro — ..esktop/ESlab2 — zsh — ...

8468:      e3530000      cmp     r3, #0
846c:      08bd8008      popeq   {r3, pc}
8470:      e12fff33      blx     r3
8474:      e8bd8008      pop     {r3, pc}
8478:      00010608      .word   0x00010608
847c:      00000000      .word   0x00000000

00008480 <main>:
8480:      e52de004      push    {lr}           ; (str lr, [sp, #-4]!)
8484:      e24dd00c      sub     sp, sp, #12
8488:      e59f0034      ldr     r0, [pc, #52]   ; 84c4 <main+0x44>
848c:      e28d1004      add     r1, sp, #4
8490:      e1a0200d      mov     r2, sp
8494:      ebffffcb      bl      83c8 <_init+0x50>
8498:      e59d2000      ldr     r2, [sp]
849c:      e59d1004      ldr     r1, [sp, #4]
84a0:      e0821001      add     r1, r2, r1
84a4:      e0821201      add     r1, r2, r1, lsl #4
84a8:      e1a01201      lsl     r1, r1, #4
84ac:      e58d1004      str     r1, [sp, #4]
84b0:      e59f0010      ldr     r0, [pc, #16]   ; 84c8 <main+0x48>
84b4:      ebffffc0      bl      83bc <_init+0x44>
84b8:      e3a00000      mov     r0, #0
84bc:      e28dd00c      add     sp, sp, #12
84c0:      e8bd8000      pop     {pc}
84c4:      000085a8      .word   0x000085a8
84c8:      000085b0      .word   0x000085b0

000084cc <__libc_csu_fini>:
84cc:      e12fff1e      bx      lr
```

观察 84a4 可知，产生了寄存器移位寻址。

4. 设计 C 的代码场景，观察一个复杂的 32 位数是如何装载到寄存器的。

- C 代码


```
ESlab2 — scn3@scn3deMacBook-Pro — ..esktop/ESlab2 — -zsh — 92x30

8408: e3530000 cmp r3, #0
840c: 08bd8008 popeq {r3, pc}
8410: e59f3010 ldr r3, [pc, #16] ; 8428 <frame_dummy+0x2c>
8414: e3530000 cmp r3, #0
8418: 08bd8008 popeq {r3, pc}
841c: e12fff33 blx r3
8420: e8bd8008 pop {r3, pc}
8424: 000105a4 .word 0x000105a4
8428: 00000000 .word 0x00000000

0000842c <main>:
842c: e92d4800 push {fp, lr}
8430: e28db004 add fp, sp, #4
8434: e24dd008 sub sp, sp, #8
8438: e59f3020 ldr r3, [pc, #32] ; 8460 <main+0x34>
843c: e50b3008 str r3, [fp, #-8]
8440: e59f301c ldr r3, [pc, #28] ; 8464 <main+0x38>
8444: e1a00003 mov r0, r3
8448: e51b1008 ldr r1, [fp, #-8]
844c: ebffffc8 bl 8374 <_init+0x44>
8450: e3a03000 mov r3, #0
8454: e1a00003 mov r0, r3
8458: e24bd004 sub sp, fp, #4
845c: e8bd8800 pop {fp, pc}
8460: 12345678 .word 0x12345678
8464: 00008544 .word 0x00008544

00008468 <__libc_csu_fini>:
8468: e12fff1e bx lr
```

观察可知，32 位数先被存在了 8460 处，然后用 ldr 指令将之装入寄存器 r3。

5. 写一个 C 的多重函数调用的函数，观察并分析

- C 代码

```
ESlab2 — vim function.c — vim — vim function.c — 79×27
4   > Mail:
5   > Created Time: 日 4/ 3 15:54:10 2016
6   *****/
7
8   #include<stdio.h>
9
10  int g(int x, int y) {
11      return x * y;
12  }
13
14  int f(int x) {
15      int y;
16      scanf("%d", &y);
17      return g(x,y);
18  }
19
20  int main(void) {
21      int a;
22      scanf("%d\n", &a);
23      printf("%d", f(a));
24      return 0;
25  }
~
<top/ESlab2/function.c [FORMAT=unix] [TYPE=C] [POS=10,18][40%] 03/04/16 - 15:59
```

- ARM 指令


```
ESlab2 — scn3@scn3deMacBook-Pro — ..esktop/ESlab2 — -zsh — 93x30

84e4:      e1a03000      mov     r3, r0
84e8:      e1a00003      mov     r0, r3
84ec:      e24bd004      sub     sp, fp, #4
84f0:      e8bd8800      pop     {fp, pc}
84f4:      0000862c      .word   0x0000862c

000084f8 <main>:
84f8:      e92d4810      push    {r4, fp, lr}
84fc:      e28db008      add     fp, sp, #8
8500:      e24dd00c      sub     sp, sp, #12
[ 8504:      e59f203c      ldr     r2, [pc, #60] ; 8548 <main+0x50> ]
8508:      e24b3010      sub     r3, fp, #16
850c:      e1a00002      mov     r0, r2
8510:      e1a01003      mov     r1, r3
8514:      ebffffab      bl      83c8 <_init+0x50>
8518:      e59f402c      ldr     r4, [pc, #44] ; 854c <main+0x54>
851c:      e51b3010      ldr     r3, [fp, #-16]
8520:      e1a00003      mov     r0, r3
8524:      ebffffe1      bl      84b0 <f>
8528:      e1a03000      mov     r3, r0
852c:      e1a00004      mov     r0, r4
8530:      e1a01003      mov     r1, r3
8534:      ebffffa0      bl      83bc <_init+0x44>
8538:      e3a03000      mov     r3, #0
853c:      e1a00003      mov     r0, r3
8540:      e24bd008      sub     sp, fp, #8
8544:      e8bd8810      pop     {r4, fp, pc}
8548:      00008630      .word   0x00008630
854c:      0000862c      .word   0x0000862c
```

```
ESlab2 — scn3@scn3deMacBook-Pro — ..esktop/ESlab2 — -zsh — 93×36
847c:      00000000      .word  0x00000000

00008480 <g>:
8480:      e52db004      push   {fp}           ; (str fp, [sp, #-4]!)
8484:      e28db000      add    fp, sp, #0
8488:      e24dd00c      sub    sp, sp, #12
848c:      e50b0008      str    r0, [fp, #-8]
8490:      e50b100c      str    r1, [fp, #-12]
8494:      e51b3008      ldr    r3, [fp, #-8]
8498:      e51b200c      ldr    r2, [fp, #-12]
849c:      e0030392      mul    r3, r2, r3
84a0:      e1a00003      mov    r0, r3
84a4:      e28bd000      add    sp, fp, #0
84a8:      e8bd0800      pop    {fp}
84ac:      e12fff1e      bx     lr

000084b0 <f>:
84b0:      e92d4800      push   {fp, lr}
84b4:      e28db004      add    fp, sp, #4
84b8:      e24dd010      sub    sp, sp, #16
84bc:      e50b0010      str    r0, [fp, #-16]
84c0:      e59f202c      ldr    r2, [pc, #44]   ; 84f4 <f+0x44>
84c4:      e24b3008      sub    r3, fp, #8
84c8:      e1a00002      mov    r0, r2
84cc:      e1a01003      mov    r1, r3
84d0:      ebfefefc      bl     83c8 <_init+0x50>
84d4:      e51b3008      ldr    r3, [fp, #-8]
84d8:      e51b0010      ldr    r0, [fp, #-16]
84dc:      e1a01003      mov    r1, r3
84e0:      ebfefef6      bl     8480 <g>
84e4:      e1a03000      mov    r3, r0
84e8:      e1a00003      mov    r0, r3
84ec:      e24bd004      sub    sp, fp, #4
84f0:      e8bd8800      pop    {fp, pc}
84f4:      0000862c      .word  0x0000862c
```

观察和分析可知：

- a) 调用时的返回地址在 lr 链接寄存器中，在执行 bl 指令时，会自动将下一条指令地址传入 lr 寄存器。
 - b) 传入的参数在 r0 和 r1 中。
 - c) 在函数堆栈中，本地变量存在返回地址和 fp 之后。
 - d) r0-r3 是 caller 保存而 r4 是 callee 保存。
6. 尝试写 C 的表达式来编译得到 MLA 指令。
- C 代码


```
ESlab2 — scn3@scn3deMacBook-Pro — ..esktop/ESlab2 — -zsh — 92x30

846c: 08bd8008 popeq {r3, pc}
8470: e12fff33 blx r3
8474: e8bd8008 pop {r3, pc}
8478: 00010600 .word 0x00010600
847c: 00000000 .word 0x00000000

00008480 <main>:
8480: e52de004 push {lr} ; (str lr, [sp, #-4]!)
8484: e24dd014 sub sp, sp, #20
8488: e59f0030 ldr r0, [pc, #48] ; 84c0 <main+0x40>
848c: e28d100c add r1, sp, #12
8490: e28d2008 add r2, sp, #8
8494: e28d3004 add r3, sp, #4
8498: ebffffca bl 83c8 <_init+0x50>
849c: e59f0020 ldr r0, [pc, #32] ; 84c4 <main+0x44>
84a0: e59d3004 ldr r3, [sp, #4]
84a4: e59d100c ldr r1, [sp, #12]
84a8: e59d2008 ldr r2, [sp, #8]
84ac: e0213192 mla r1, r2, r1, r3
84b0: ebffffc1 bl 83bc <_init+0x44>
84b4: e3a00000 mov r0, #0
84b8: e28dd014 add sp, sp, #20
84bc: e8bd8000 pop {pc}
84c0: 000085a4 .word 0x000085a4
84c4: 000085ac .word 0x000085ac

000084c8 <__libc_csu_fini>:
84c8: e12fff1e bx lr

000084cc <__libc_csu_init>:
```

在 84ac 处出现了 mla。

7. 尝试写 C 的表达式来编译得到 BIC 指令。

- C 代码

```
ESlab2 — vim bic.c — vim — vim bic.c — 92×30
```

```
1 /*****  
2 > File Name: bic.c  
3 > Author:  
4 > Mail:  
5 > Created Time: 日 4/ 3 16:22:02 2016  
6 *****/  
7  
8 #include<stdio.h>  
9 int main(void) {  
10     int x, y;  
11     scanf("%d%d", &x, &y);  
12     printf("%d", x & (~y));  
13     return 0;  
14 }
```

```
~/Desktop/ESlab2/bic.c [FORMAT=unix] [TYPE=C] [POS=13,5][92%] 03/04/16 - 16:24  
"bic.c" 14L, 354C
```

- ARM 指令（编译参数加上-O）

```
ESlab2 — scn3@scn3deMacBook-Pro — ..esktop/ESlab2 — -zsh — 92x30

8468:      e3530000      cmp     r3, #0
846c:      08bd8008      popeq  {r3, pc}
8470:      e12fff33      blx    r3
8474:      e8bd8008      pop    {r3, pc}
8478:      000105f4      .word  0x000105f4
847c:      00000000      .word  0x00000000

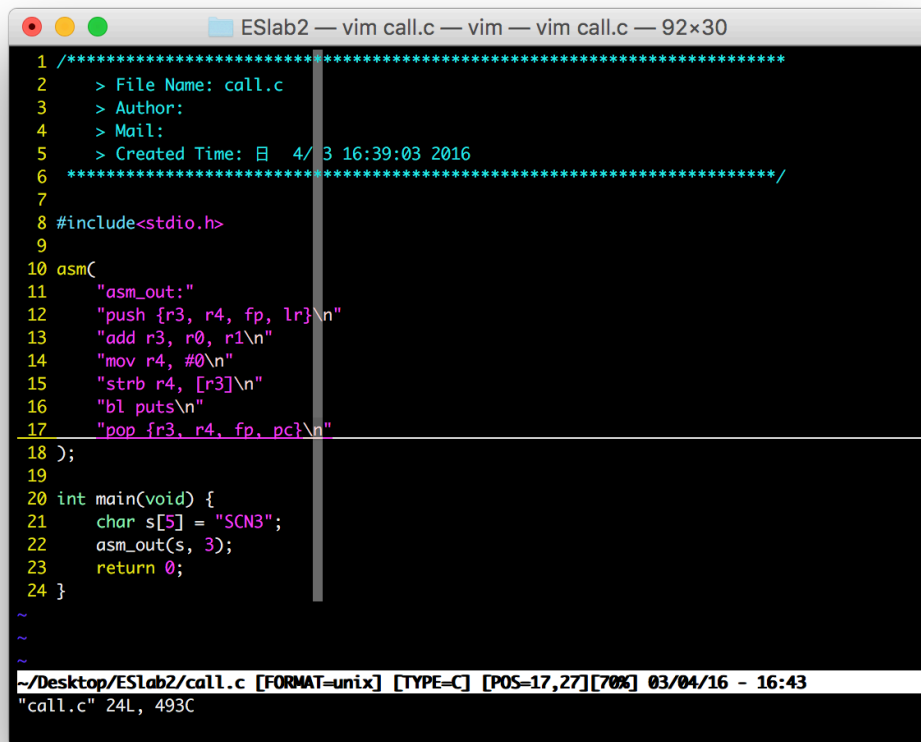
00008480 <main>:
8480:      e52de004      push   {lr}          ; (str lr, [sp, #-4]!)
8484:      e24dd00c      sub    sp, sp, #12
8488:      e59f0024      ldr     r0, [pc, #36] ; 84b4 <main+0x34>
848c:      e28d1004      add    r1, sp, #4
8490:      e1a0200d      mov    r2, sp
8494:      ebffffcb      bl     83c8 <_init+0x50>
8498:      e59f0018      ldr     r0, [pc, #24] ; 84b8 <main+0x38>
849c:      e89d000a      ldm     sp, {r1, r3}
84a0:      e1c31001      bic     r1, r3, r1
84a4:      ebffffc4      bl     83bc <_init+0x44>
84a8:      e3a00000      mov     r0, #0
84ac:      e28dd00c      add    sp, sp, #12
84b0:      e8bd8000      pop     {pc}
84b4:      00008598      .word  0x00008598
84b8:      000085a0      .word  0x000085a0

000084bc <__libc_csu_fini>:
84bc:      e12fff1e      bx      lr

000084c0 <__libc_csu_init>:
84c0:      e92d47f0      push   {r4, r5, r6, r7, r8, r9, sl, lr}
84c4:      e1a08000      mov     r8, r0
```

在 84a0 处出现了 bic 指令。

8. 编写一个汇编函数，接受一个整数和一个指针做为输入，指针所指应为一个字符串，该汇编函数调用 C 语言的 `printf()` 函数输出这个字符串的前 `n` 个字符，`n` 即为那个整数。在 C 语言写的 `main()` 函数中调用并传递参数给这个汇编函数 来得到输出。
 - 代码如下（用“puts”输出字符串）



```
1 /*****
2  > File Name: call.c
3  > Author:
4  > Mail:
5  > Created Time: 日 4/ 3 16:39:03 2016
6  *****/
7
8 #include<stdio.h>
9
10 asm(
11     "asm_out:"
12     "push {r3, r4, fp, lr}\n"
13     "add r3, r0, r1\n"
14     "mov r4, #0\n"
15     "strb r4, [r3]\n"
16     "bl puts\n"
17     "pop {r3, r4, fp, pc}\n"
18 );
19
20 int main(void) {
21     char s[5] = "SCN3";
22     asm_out(s, 3);
23     return 0;
24 }
~
~
~
~/Desktop/ESlab2/call.c [FORMAT=unix] [TYPE=C] [POS=17,27][70%] 03/04/16 - 16:43
"call.c" 24L, 493C
```

四、扩展内容
无