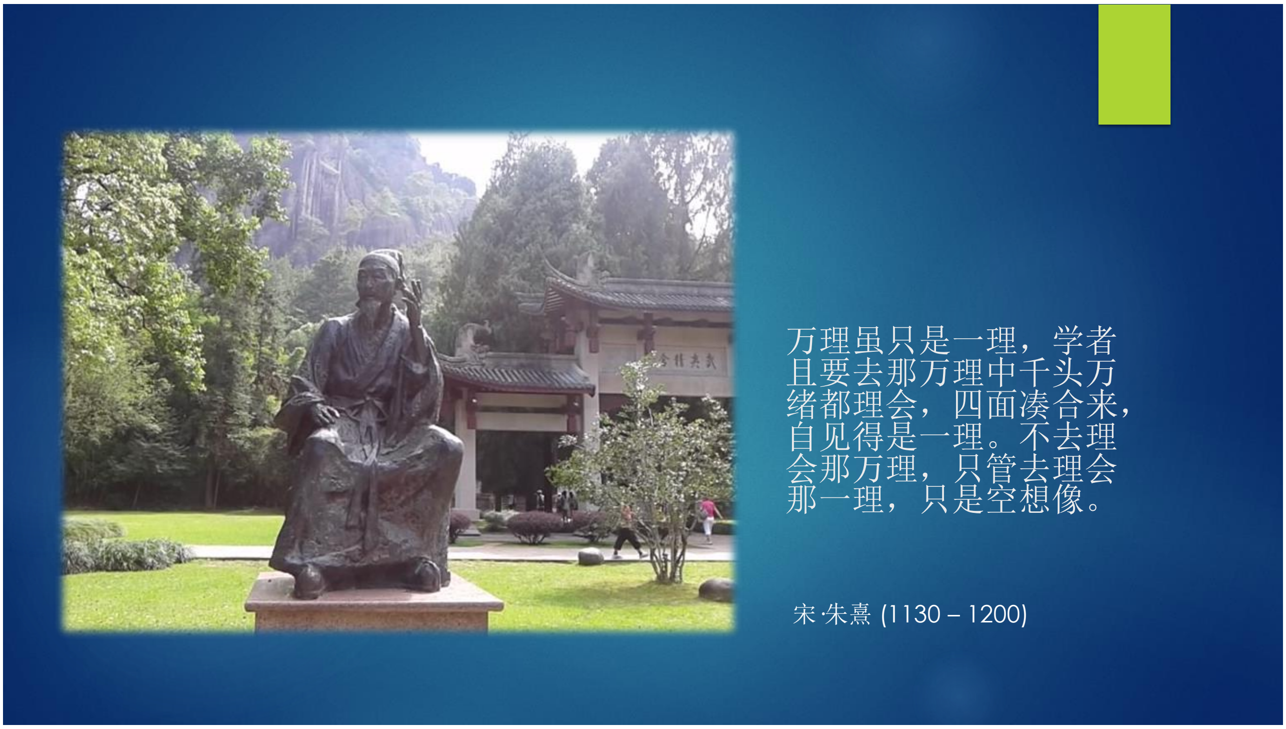
在调试器下理解ARMv8

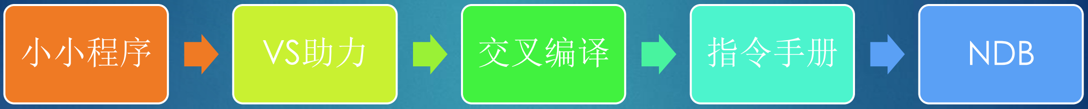
——应用层寄存器和指令

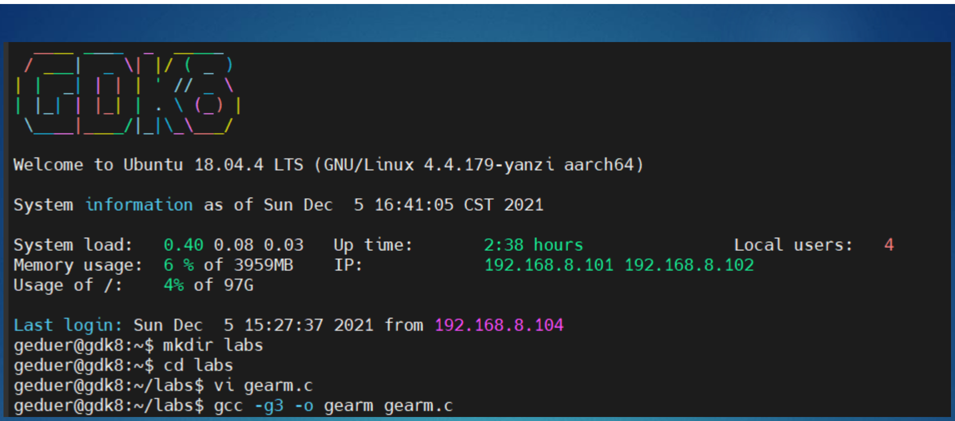
张银奎



万理虽只是一理，学者且要去那万理中千头万绪都理会，四面凑合来，自见得是一理。不去理会那万理，只管去理会那一理，只是空想像。

宋·朱熹 (1130 – 1200)





```
  ____ ____  _  _____
 / ___|  _ \| |/ ( _ )
| |  _| | | | ' // _ \
| |_| | |_| | . \ (_) |
 \____|____/|_|\_\___/

Welcome to Ubuntu 18.04.4 LTS (GNU/Linux 4.4.179-yanzi aarch64)

System information as of Sun Dec  5 16:41:05 CST 2021

System load:   0.40 0.08 0.03   Up time:       2:38 hours              Local users:   4
Memory usage:  6 % of 3959MB    IP:            192.168.8.101 192.168.8.102
Usage of /:    4% of 97G

Last login: Sun Dec  5 15:27:37 2021 from 192.168.8.104
geduer@gdk8:~$ mkdir labs
geduer@gdk8:~$ cd labs
geduer@gdk8:~/labs$ vi gearm.c
geduer@gdk8:~/labs$ gcc -g3 -o gearm gearm.c
```

```
#include <stdio.h>

int main(int argc, char* argv[])
{
    printf("hello gdk8(%d %s)\n", argc, argv[0]);
    return 0;
}
```

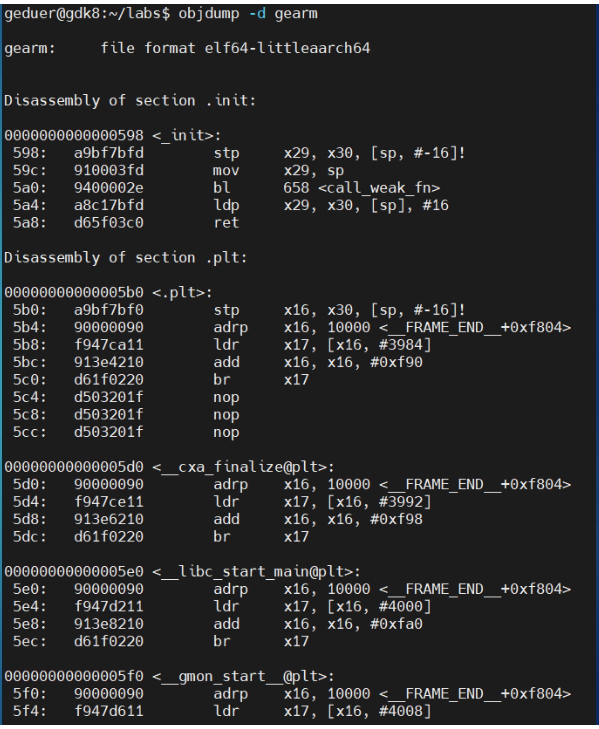
```
gcc -g3 -o gearm gearm.c
```

geduer@gdk8:~/labs$ gcc -g3 -o gearm gearm.c
geduer@gdk8:~/labs$ ./gearm

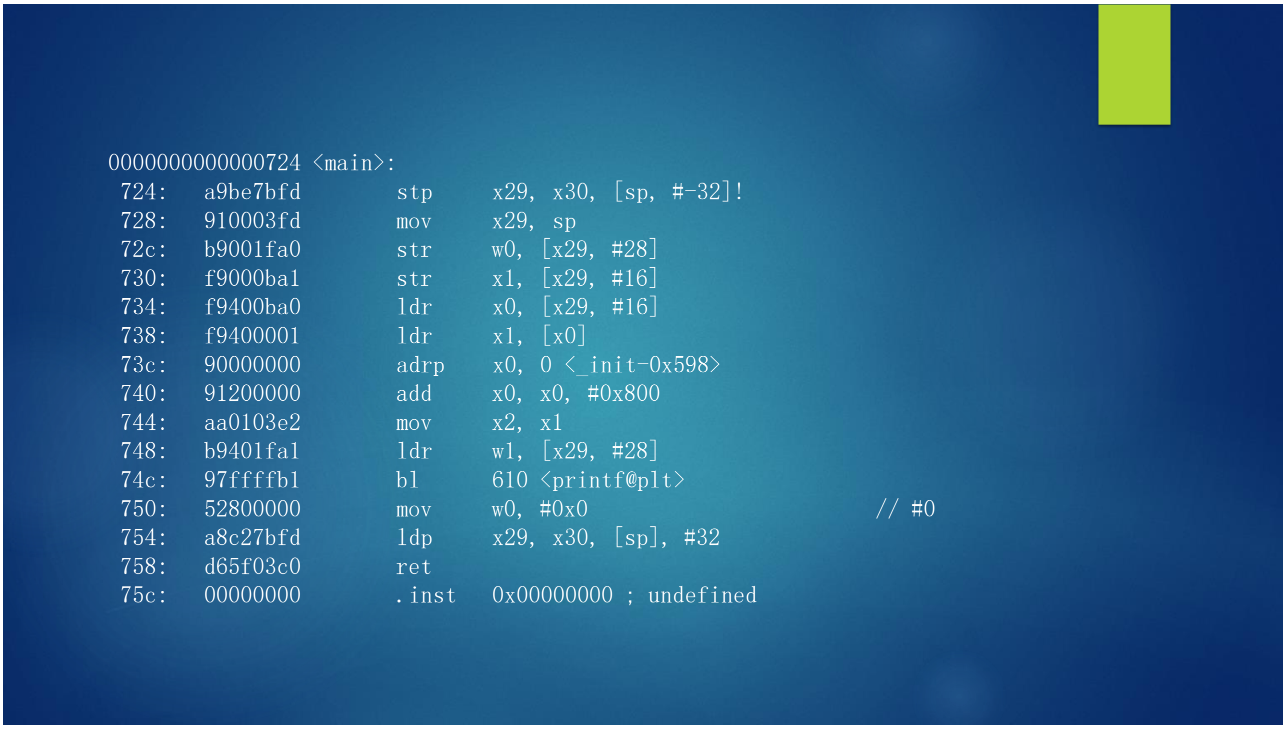
hello gdk8

objdump -d gearm

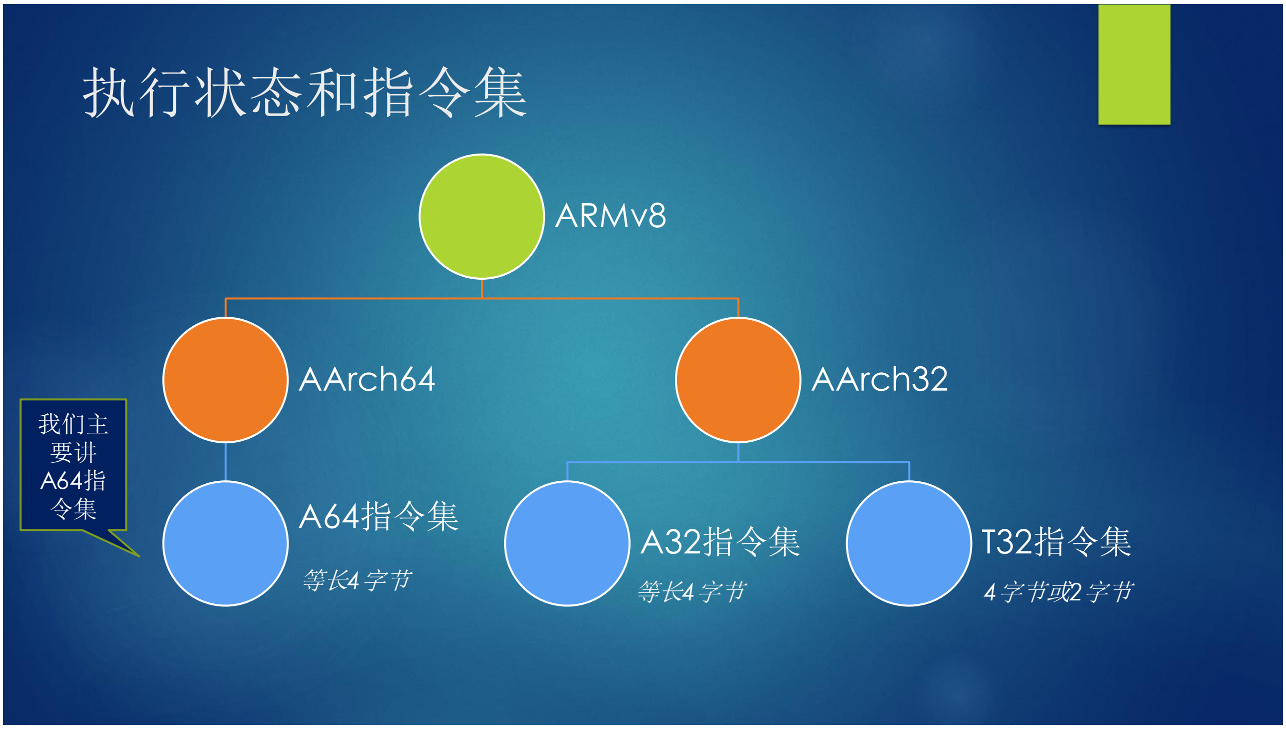
gearm: file format elf64-littleaarch64



```
geduer@gdk8:~/labs$ objdump -d gearm

gearm:     file format elf64-littleaarch64


Disassembly of section .init:

0000000000000598 <_init>:
 598:   a9bf7bfd        stp     x29, x30, [sp, #-16]!
 59c:   910003fd        mov     x29, sp
 5a0:   9400002e        bl      658 <call_weak_fn>
 5a4:   a8c17bfd        ldp     x29, x30, [sp], #16
 5a8:   d65f03c0        ret

Disassembly of section .plt:

00000000000005b0 <.plt>:
 5b0:   a9bf7bf0        stp     x16, x30, [sp, #-16]!
 5b4:   90000090        adrp    x16, 10000 <__FRAME_END__+0xf804>
 5b8:   f947ca11        ldr     x17, [x16, #3984]
 5bc:   913e4210        add     x16, x16, #0xf90
 5c0:   d61f0220        br      x17
 5c4:   d503201f        nop
 5c8:   d503201f        nop
 5cc:   d503201f        nop

00000000000005d0 <__cxa_finalize@plt>:
 5d0:   90000090        adrp    x16, 10000 <__FRAME_END__+0xf804>
 5d4:   f947ce11        ldr     x17, [x16, #3992]
 5d8:   913e6210        add     x16, x16, #0xf98
 5dc:   d61f0220        br      x17

00000000000005e0 <__libc_start_main@plt>:
 5e0:   90000090        adrp    x16, 10000 <__FRAME_END__+0xf804>
 5e4:   f947d211        ldr     x17, [x16, #4000]
 5e8:   913e8210        add     x16, x16, #0xfa0
 5ec:   d61f0220        br      x17

00000000000005f0 <__gmon_start__@plt>:
 5f0:   90000090        adrp    x16, 10000 <__FRAME_END__+0xf804>
 5f4:   f947d611        ldr     x17, [x16, #4008]
```



```
0000000000000724 <main>:
 724:   a9be7bfd        stp     x29, x30, [sp, #-32]!
 728:   910003fd        mov     x29, sp
 72c:   b9001fa0        str     w0, [x29, #28]
 730:   f9000ba1        str     x1, [x29, #16]
 734:   f9400ba0        ldr     x0, [x29, #16]
 738:   f9400001        ldr     x1, [x0]
 73c:   90000000        adrp    x0, 0 <_init-0x598>
 740:   91200000        add     x0, x0, #0x800
 744:   aa0103e2        mov     x2, x1
 748:   b9401fa1        ldr     w1, [x29, #28]
 74c:   97ffffb1        bl      610 <printf@plt>
 750:   52800000        mov     w0, #0x0                        // #0
 754:   a8c27bfd        ldp     x29, x30, [sp], #32
 758:   d65f03c0        ret
 75c:   00000000        .inst   0x00000000 ; undefined
```



执行状态和指令集

ARMv8

AArch64

AArch32

A64指令集

等长4字节

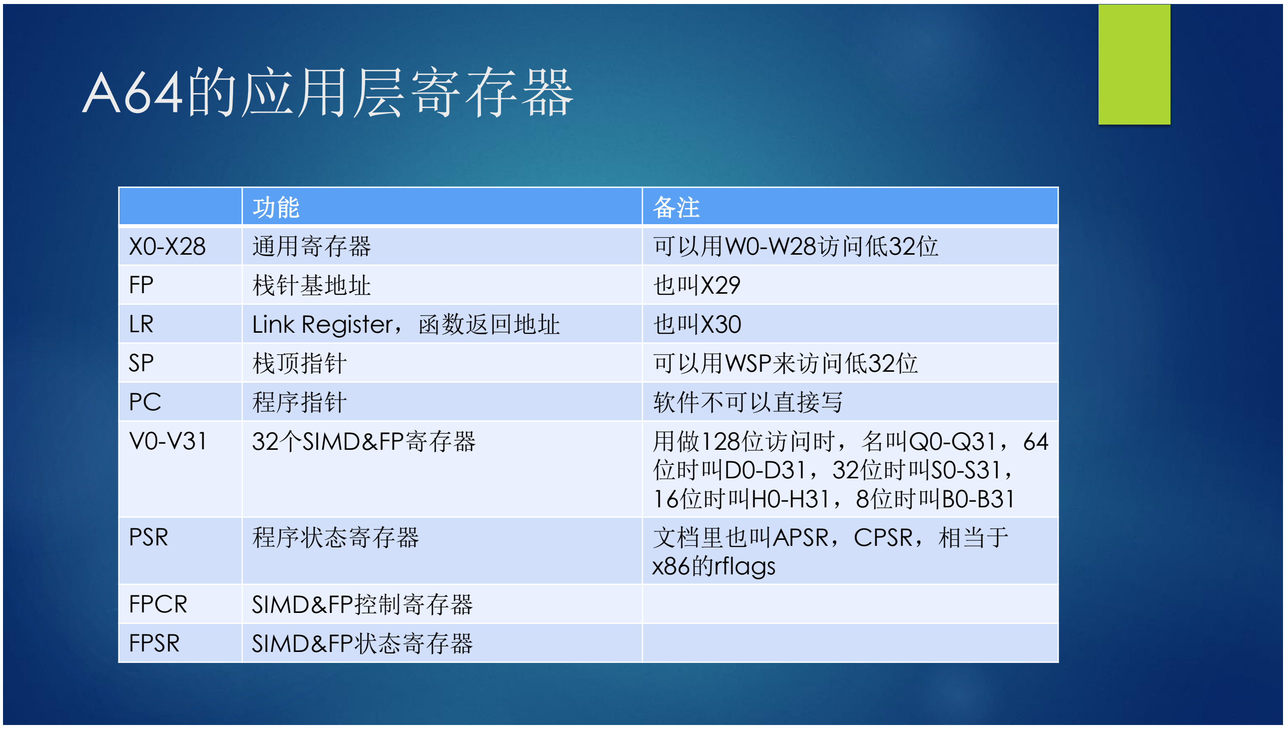
A32指令集

等长4字节

T32指令集

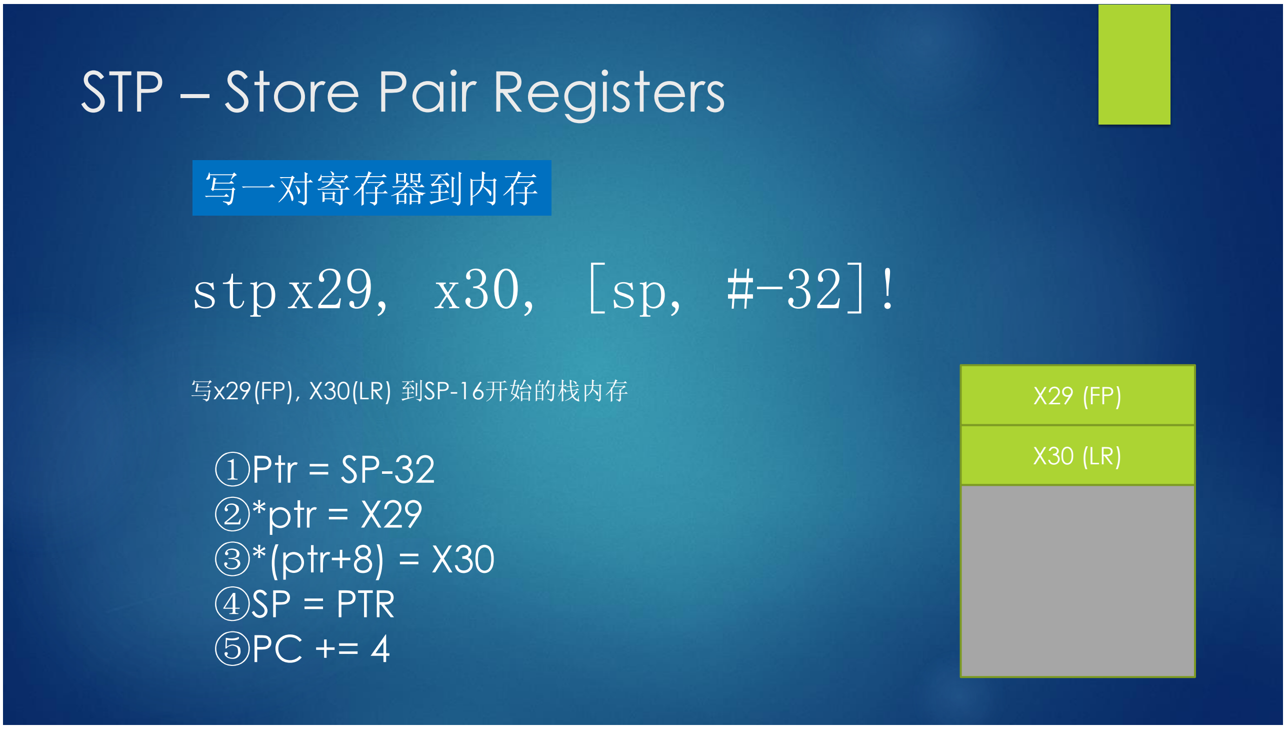
4字节或2字节

我们主要讲A64指令集



A64的应用层寄存器

	功能	备注
X0-X28	通用寄存器	可以用W0-W28访问低32位
FP	栈针基地址	也叫X29
LR	Link Register，函数返回地址	也叫X30
SP	栈顶指针	可以用WSP来访问低32位
PC	程序指针	软件不可以直接写
V0-V31	32个SIMD&FP寄存器	用做128位访问时，名叫Q0-Q31，64位时叫D0-D31，32位时叫S0-S31，16位时叫H0-H31，8位时叫B0-B31
PSR	程序状态寄存器	文档里也叫APSR，CPSR，相当于x86的rflags
FPCR	SIMD&FP控制寄存器	
FPSR	SIMD&FP状态寄存器	



STP – Store Pair Registers

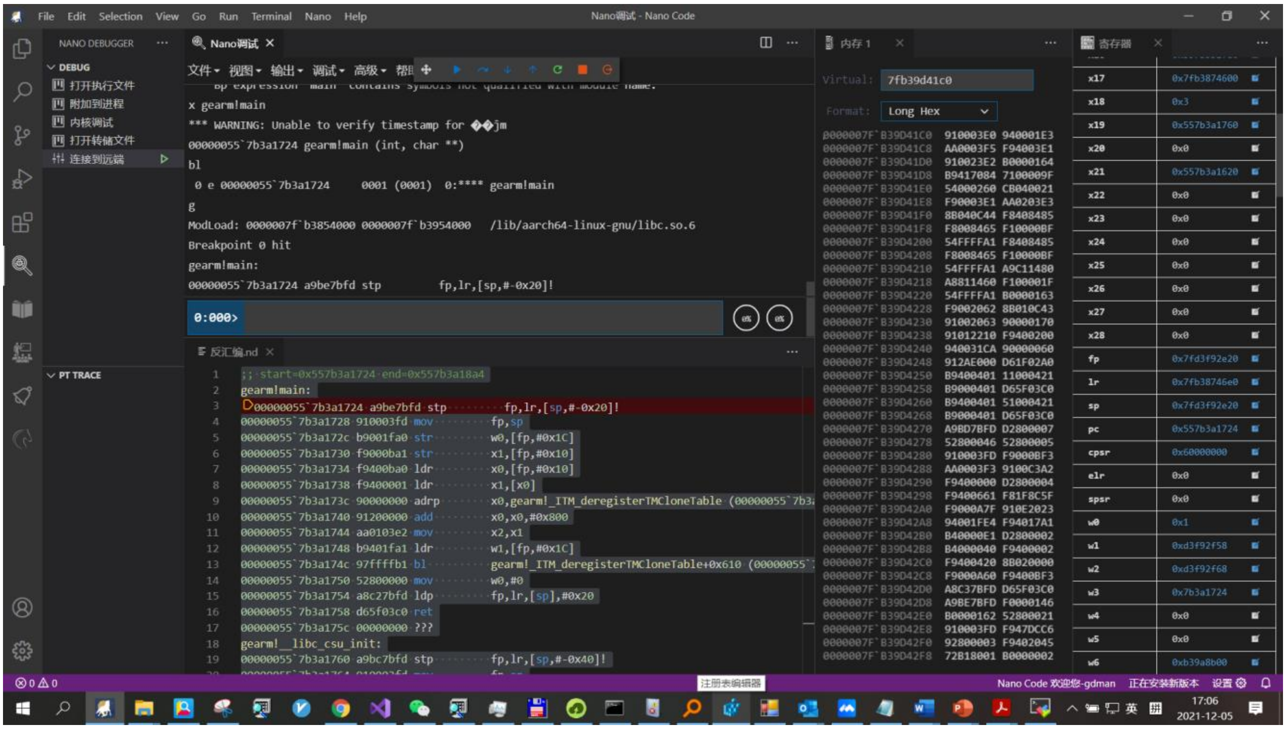
写一对寄存器到内存

```
stp x29， x30， [sp， #-32]!
```

写x29(FP), X30(LR) 到SP-16开始的栈内存

①Ptr = SP-32
②*ptr = X29
③*(ptr+8) = X30
④SP = PTR
⑤PC += 4

X29 (FP)
X30 (LR)

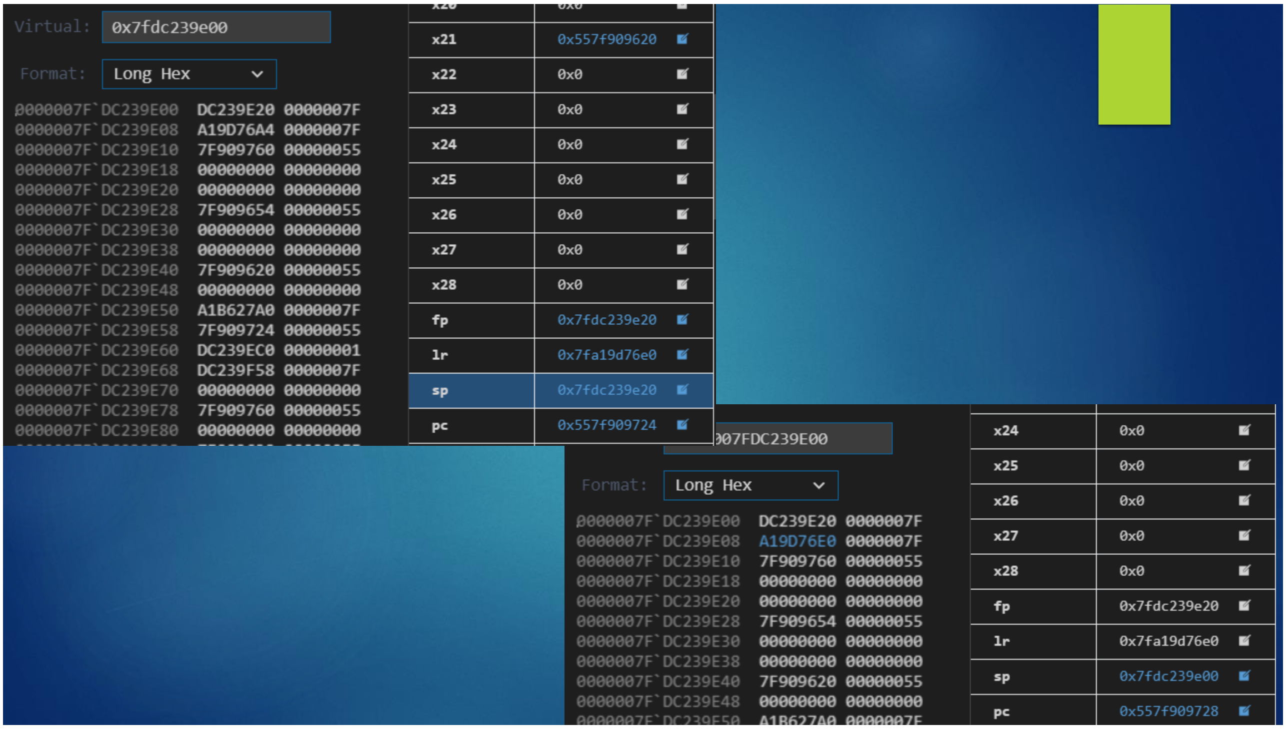


```
x gearm!main
*** WARNING: Unable to verify timestamp for ��jm
00000055`7b3a1724 gearm!main (int, char **)
bl
 0 e 00000055`7b3a1724     0001 (0001)  0:**** gearm!main
g
ModLoad: 0000007f`b3854000 0000007f`b3954000   /lib/aarch64-linux-gnu/libc.so.6
Breakpoint 0 hit
gearm!main:
00000055`7b3a1724 a9be7bfd stp         fp,lr,[sp,#-0x20]!
0:000>
```

```
;; start=0x557b3a1724 end=0x557b3a18a4
gearm!main:
00000055`7b3a1724 a9be7bfd stp         fp,lr,[sp,#-0x20]!
00000055`7b3a1728 910003fd mov         fp,sp
00000055`7b3a172c b9001fa0 str         w0,[fp,#0x1C]
00000055`7b3a1730 f9000ba1 str         x1,[fp,#0x10]
00000055`7b3a1734 f9400ba0 ldr         x0,[fp,#0x10]
00000055`7b3a1738 f9400001 ldr         x1,[x0]
00000055`7b3a173c 90000000 adrp        x0,gearm!_ITM_deregisterTMCloneTable (00000055`7b3
00000055`7b3a1740 91200000 add         x0,x0,#0x800
00000055`7b3a1744 aa0103e2 mov         x2,x1
00000055`7b3a1748 b9401fa1 ldr         w1,[fp,#0x1C]
00000055`7b3a174c 97ffffb1 bl          gearm!_ITM_deregisterTMCloneTable+0x610 (00000055`
00000055`7b3a1750 52800000 mov         w0,#0
00000055`7b3a1754 a8c27bfd ldp         fp,lr,[sp],#0x20
00000055`7b3a1758 d65f03c0 ret
00000055`7b3a175c 00000000 ???
gearm!__libc_csu_init:
00000055`7b3a1760 a9bc7bfd stp         fp,lr,[sp,#-0x40]!
```

```
Virtual: 7fb39d41c0
Format:  Long Hex
0000007F`B39D41C0  910003E0 940001E3
0000007F`B39D41C8  AA0003F5 F94003E1
0000007F`B39D41D0  910023E2 B0000164
0000007F`B39D41D8  B9417084 7100009F
0000007F`B39D41E0  54000260 CB040021
0000007F`B39D41E8  F90003E1 AA0203E3
0000007F`B39D41F0  8B040C44 F8408485
0000007F`B39D41F8  F8008465 F10000BF
0000007F`B39D4200  54FFFFA1 F8408485
0000007F`B39D4208  F8008465 F10000BF
0000007F`B39D4210  54FFFFA1 A9C11480
0000007F`B39D4218  A8811460 F100001F
0000007F`B39D4220  54FFFFA1 B0000163
0000007F`B39D4228  F9002062 8B010C43
0000007F`B39D4230  91002063 90000170
0000007F`B39D4238  91012210 F9400200
0000007F`B39D4240  940031CA 90000060
0000007F`B39D4248  912AE000 D61F02A0
0000007F`B39D4250  B9400401 11000421
0000007F`B39D4258  B9000401 D65F03C0
0000007F`B39D4260  B9400401 51000421
0000007F`B39D4268  B9000401 D65F03C0
0000007F`B39D4270  A9BD7BFD D2800007
0000007F`B39D4278  52800046 52800005
0000007F`B39D4280  910003FD F9000BF3
0000007F`B39D4288  AA0003F3 9100C3A2
0000007F`B39D4290  F9400000 D2800004
0000007F`B39D4298  F9400661 F81F8C5F
0000007F`B39D42A0  F9000A7F 910E2023
0000007F`B39D42A8  94001FE4 F94017A1
0000007F`B39D42B0  B40000E1 D2800002
0000007F`B39D42B8  B4000040 F9400002
0000007F`B39D42C0  F9400420 8B020000
0000007F`B39D42C8  F9000A60 F9400BF3
0000007F`B39D42D0  A8C37BFD D65F03C0
0000007F`B39D42D8  A9BE7BFD F0000146
0000007F`B39D42E0  B0000162 52800021
0000007F`B39D42E8  910003FD F947DCC6
0000007F`B39D42F0  92800003 F9402045
0000007F`B39D42F8  72B18001 B0000002
```

x17	0x7fb3874600
x18	0x3
x19	0x557b3a1760
x20	0x0
x21	0x557b3a1620
x22	0x0
x23	0x0
x24	0x0
x25	0x0
x26	0x0
x27	0x0
x28	0x0
fp	0x7fd3f92e20
lr	0x7fb38746e0
sp	0x7fd3f92e20
pc	0x557b3a1724
cpsr	0x60000000
elr	0x0
spsr	0x0
w0	0x1
w1	0xd3f92f58
w2	0xd3f92f68
w3	0x7b3a1724
w4	0x0
w5	0x0
w6	0xb39a8b00

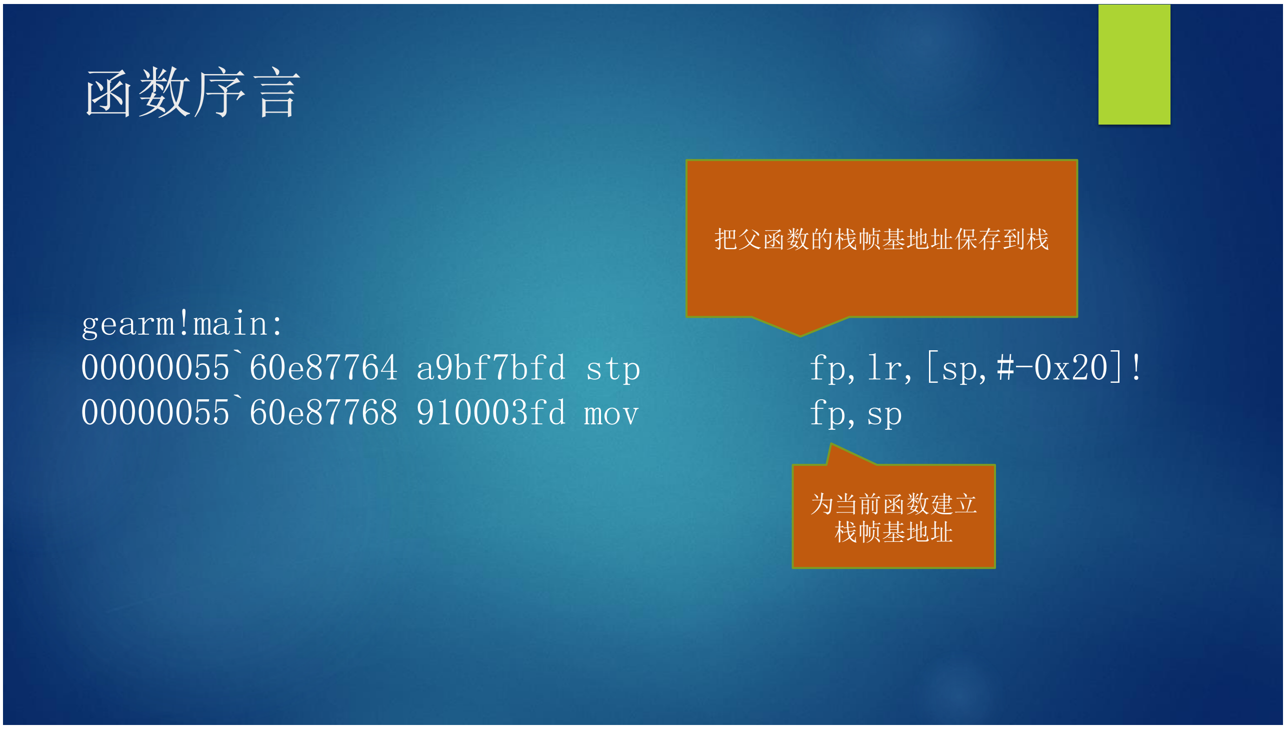


```
Virtual: 0x7fdc239e00
Format:  Long Hex
0000007F`DC239E00  DC239E20 0000007F
0000007F`DC239E08  A19D76A4 0000007F
0000007F`DC239E10  7F909760 00000055
0000007F`DC239E18  00000000 00000000
0000007F`DC239E20  00000000 00000000
0000007F`DC239E28  7F909654 00000055
0000007F`DC239E30  00000000 00000000
0000007F`DC239E38  00000000 00000000
0000007F`DC239E40  7F909620 00000055
0000007F`DC239E48  00000000 00000000
0000007F`DC239E50  A1B627A0 0000007F
0000007F`DC239E58  7F909724 00000055
0000007F`DC239E60  DC239EC0 00000001
0000007F`DC239E68  DC239F58 0000007F
0000007F`DC239E70  00000000 00000000
0000007F`DC239E78  7F909760 00000055
0000007F`DC239E80  00000000 00000000
```

x21	0x557f909620
x22	0x0
x23	0x0
x24	0x0
x25	0x0
x26	0x0
x27	0x0
x28	0x0
fp	0x7fdc239e20
lr	0x7fa19d76e0
sp	0x7fdc239e20
pc	0x557f909724

```
007FDC239E00
Format:  Long Hex
0000007F`DC239E00  DC239E20 0000007F
0000007F`DC239E08  A19D76E0 0000007F
0000007F`DC239E10  7F909760 00000055
0000007F`DC239E18  00000000 00000000
0000007F`DC239E20  00000000 00000000
0000007F`DC239E28  7F909654 00000055
0000007F`DC239E30  00000000 00000000
0000007F`DC239E38  00000000 00000000
0000007F`DC239E40  7F909620 00000055
0000007F`DC239E48  00000000 00000000
0000007F`DC239E50  A1B627A0 0000007F
```

x24	0x0
x25	0x0
x26	0x0
x27	0x0
x28	0x0
fp	0x7fdc239e20
lr	0x7fa19d76e0
sp	0x7fdc239e00
pc	0x557f909728

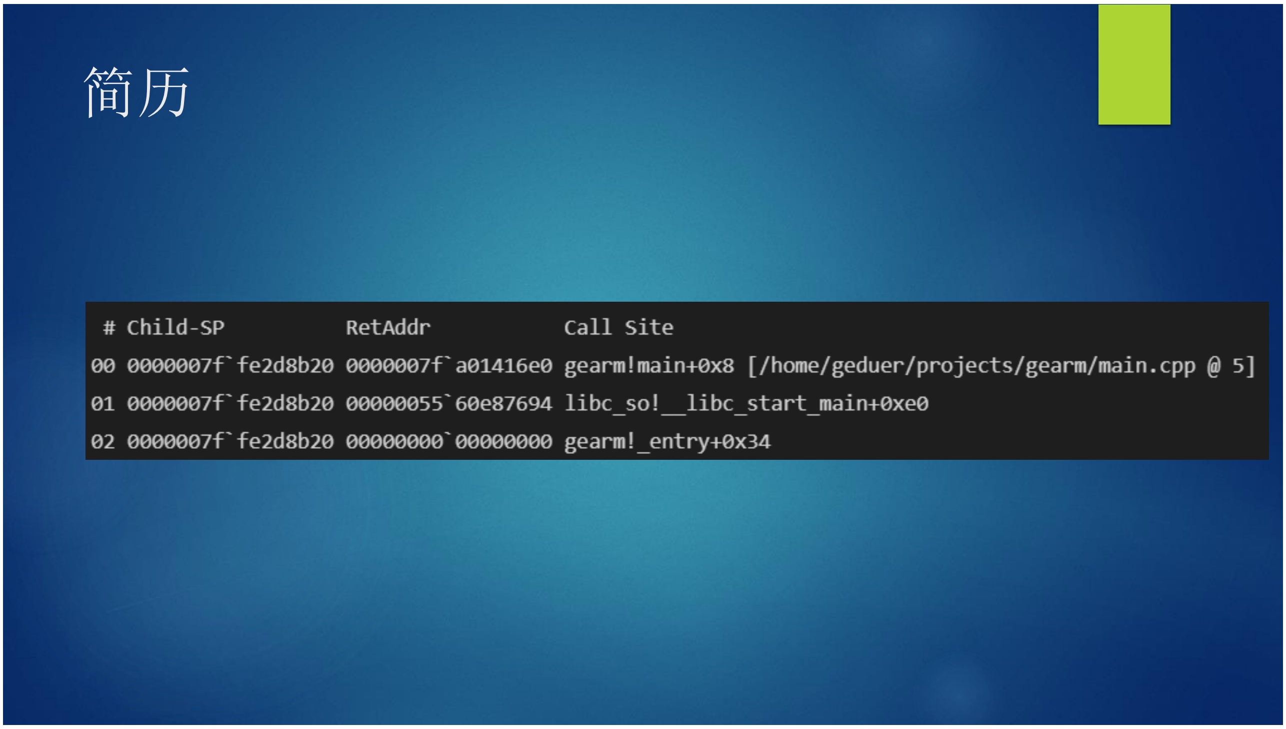


函数序言

把父函数的栈帧基地址保存到栈

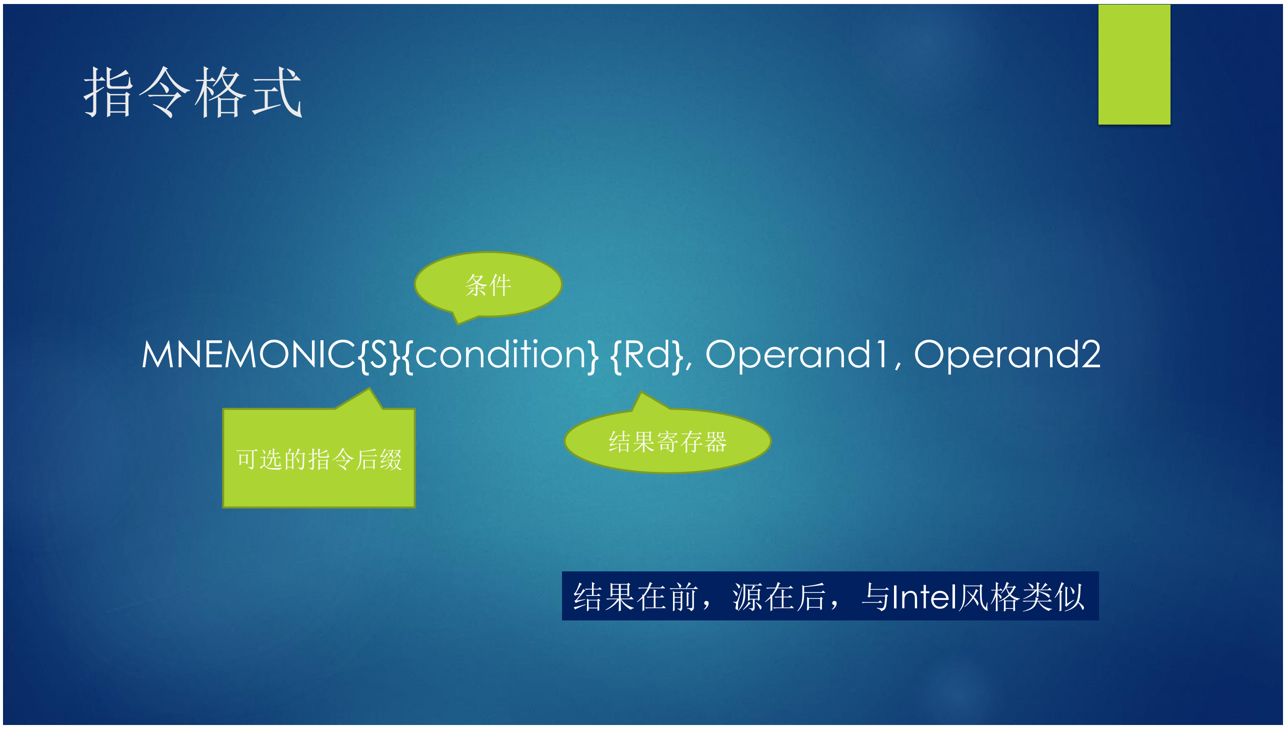
```
gearm!main:
00000055`60e87764 a9bf7bfd stp           fp, lr, [sp, #-0x20]!
00000055`60e87768 910003fd mov           fp, sp
```

为当前函数建立栈帧基地址



简历

```
 # Child-SP          RetAddr           Call Site
00 0000007f`fe2d8b20 0000007f`a01416e0 gearm!main+0x8 [/home/geduer/projects/gearm/main.cpp @ 5]
01 0000007f`fe2d8b20 00000055`60e87694 libc_so!__libc_start_main+0xe0
02 0000007f`fe2d8b20 00000000`00000000 gearm!_entry+0x34
```



指令格式

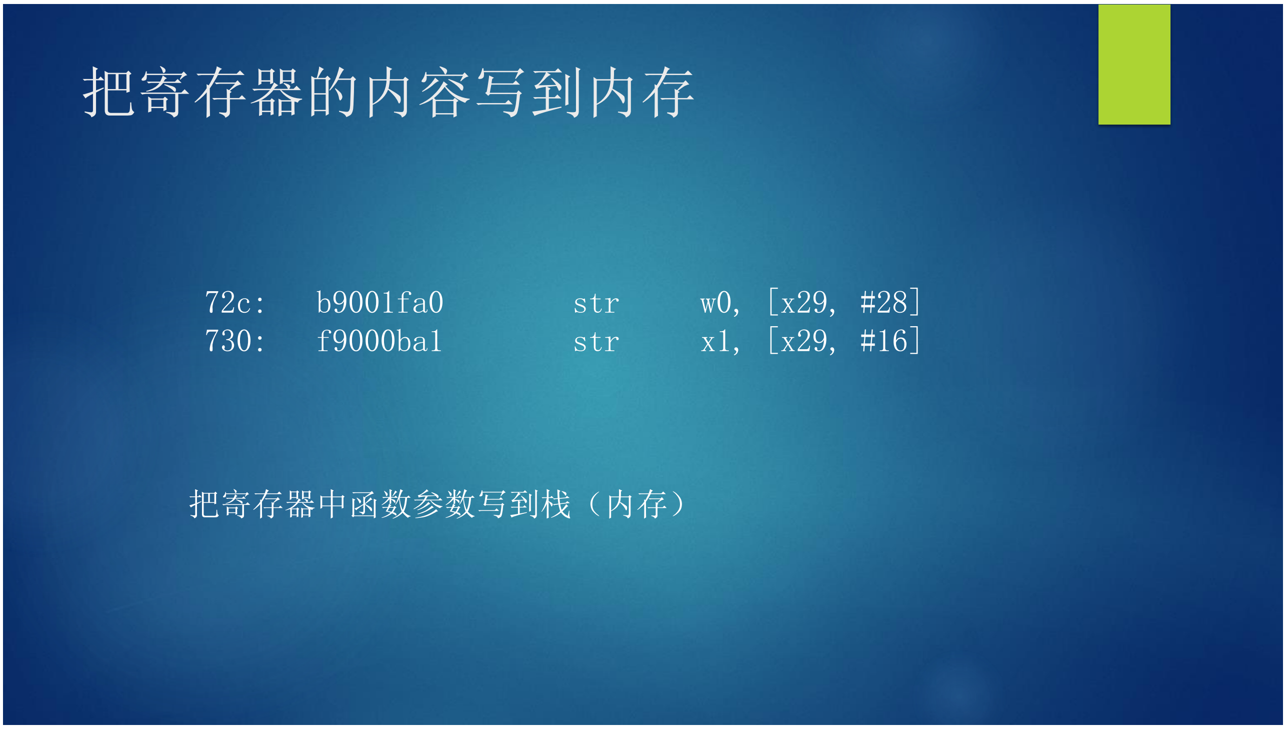
条件

MNEMONIC{S}{condition} {Rd}, Operand1, Operand2

可选的指令后缀

结果寄存器

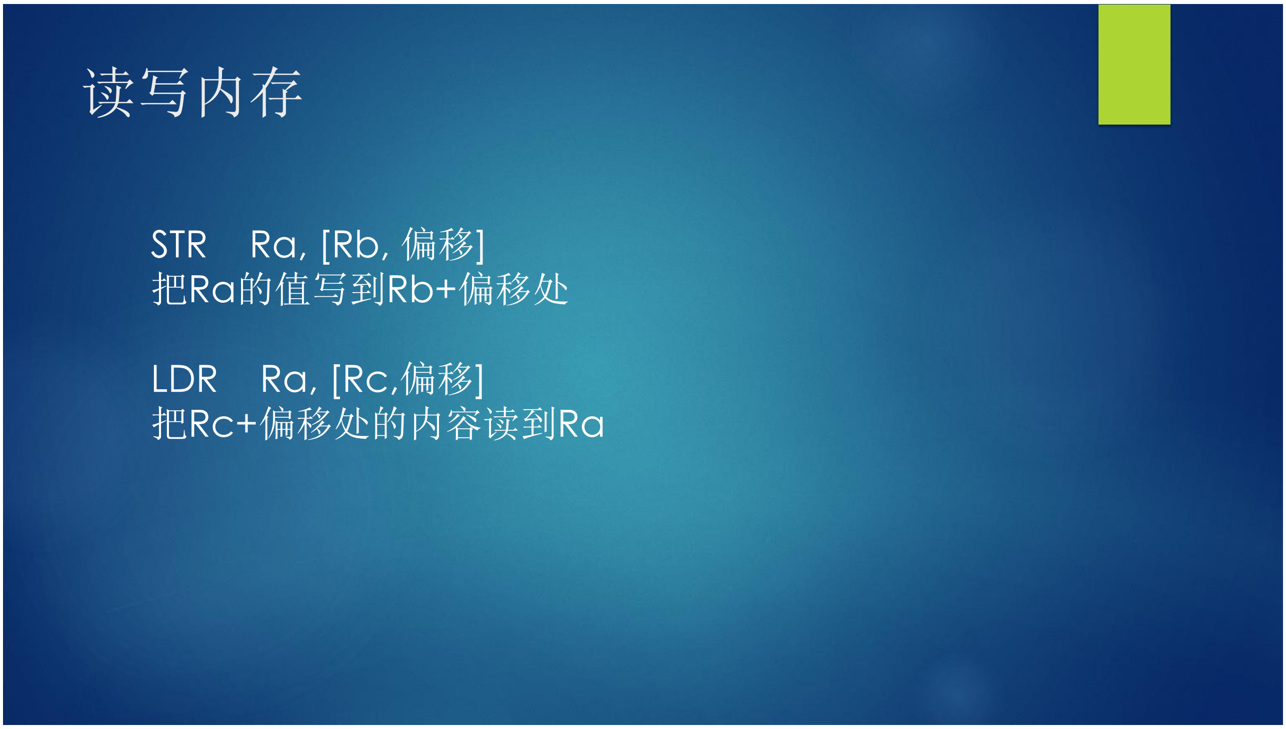
结果在前，源在后，与Intel风格类似



把寄存器的内容写到内存

```
72c:   b9001fa0         str     w0, [x29, #28]
730:   f9000ba1         str     x1, [x29, #16]
```

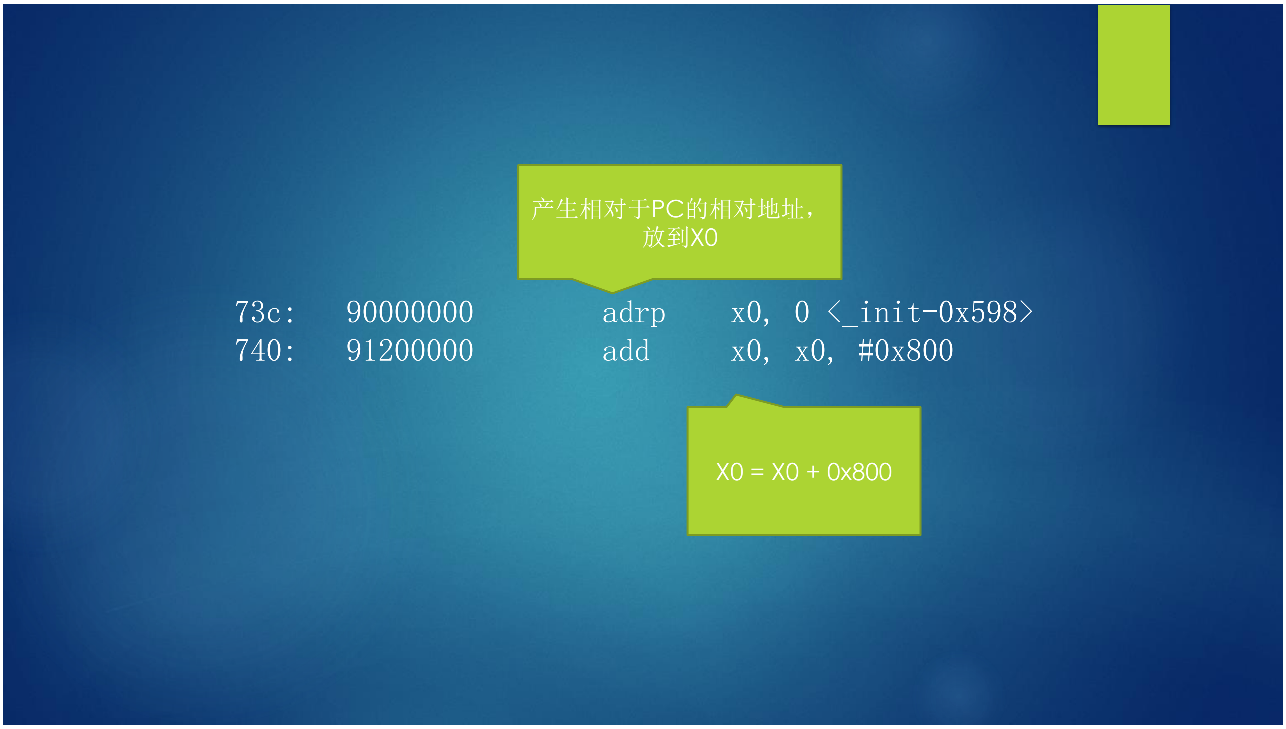
把寄存器中函数参数写到栈（内存）



读写内存

STR Ra, [Rb, 偏移]
把Ra的值写到Rb+偏移处

LDR Ra, [Rc,偏移]
把Rc+偏移处的内容读到Ra

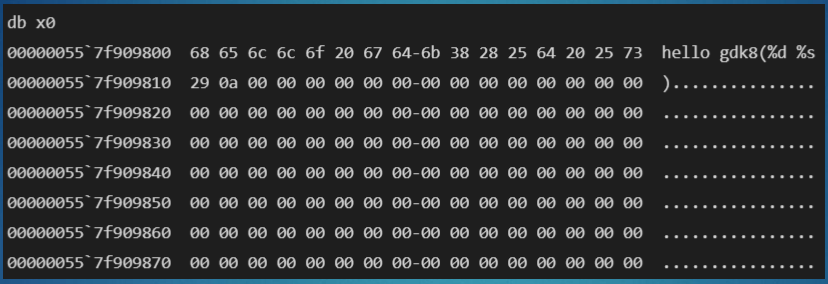


产生相对于PC的相对地址，
放到X0

```
73c:    90000000        adrp    x0, 0 <_init-0x598>
740:    91200000        add     x0, x0, #0x800
```

X0 = X0 + 0x800

两个字符串

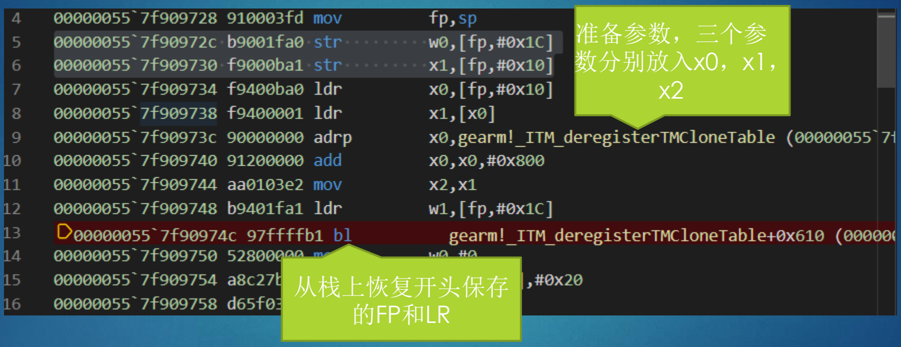


```
db x0
00000055`7f909800  68 65 6c 6c 6f 20 67 64-6b 38 28 25 64 20 25 73  hello gdk8(%d %s
00000055`7f909810  29 0a 00 00 00 00 00 00-00 00 00 00 00 00 00 00  )..............
00000055`7f909820  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ...............
00000055`7f909830  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ...............
00000055`7f909840  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ...............
00000055`7f909850  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ...............
00000055`7f909860  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ...............
00000055`7f909870  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ...............
```

```
int main(int argc, char* argv[])
{
        printf("hello gdk8(%d %s)\n", argc, argv[0]);
        return 0;
}
```

BL - Branch with Link

printf("hello gdk8(%d %s)\n", argc, argv[0]);

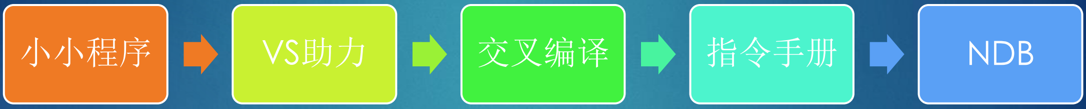


```
00000055`7f909728 910003fd mov          fp,sp
00000055`7f90972c b9001fa0 str          w0,[fp,#0x1C]
00000055`7f909730 f9000ba1 str          x1,[fp,#0x10]
00000055`7f909734 f9400ba0 ldr          x0,[fp,#0x10]
00000055`7f909738 f9400001 ldr          x1,[x0]
00000055`7f90973c 90000000 adrp         x0,gearm!_ITM_deregisterTMCloneTable (00000055`7f
00000055`7f909740 91200000 add          x0,x0,#0x800
00000055`7f909744 aa0103e2 mov          x2,x1
00000055`7f909748 b9401fa1 ldr          w1,[fp,#0x1C]
00000055`7f90974c 97ffffb1 bl             gearm!_ITM_deregisterTMCloneTable+0x610 (000000
00000055`7f909750 52800000 m            w0,#0
00000055`7f909754 a8c27b                ,#0x20
00000055`7f909758 d65f03
```

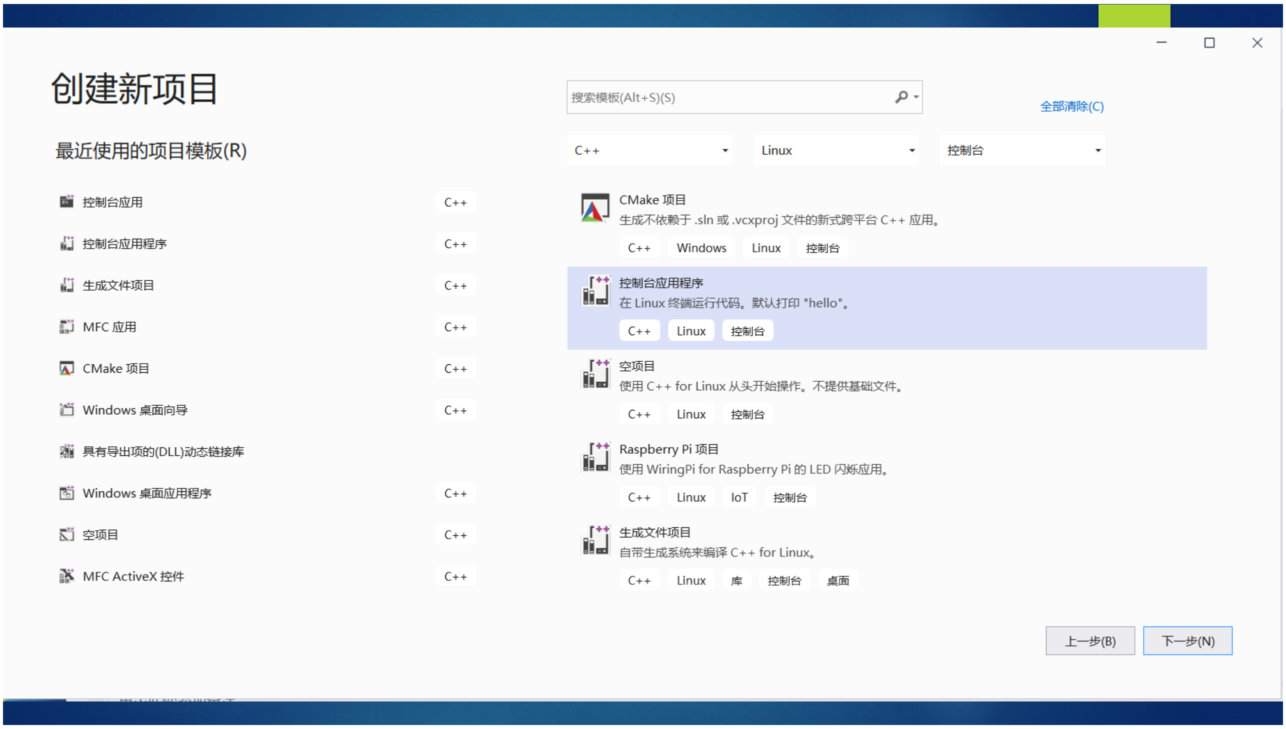
准备参数，三个参数分别放入x0，x1，x2

从栈上恢复开头保存的FP和LR

① LR = PC + 4
② PC = 子函数入口



小小程序 → VS助力 → 交叉编译 → 指令手册 → NDB



创建新项目

最近使用的项目模板(R)

控制台应用	C++
控制台应用程序	C++
生成文件项目	C++
MFC 应用	C++
CMake 项目	C++
Windows 桌面向导	C++
具有导出项的(DLL)动态链接库	
Windows 桌面应用程序	C++
空项目	C++
MFC ActiveX 控件	C++

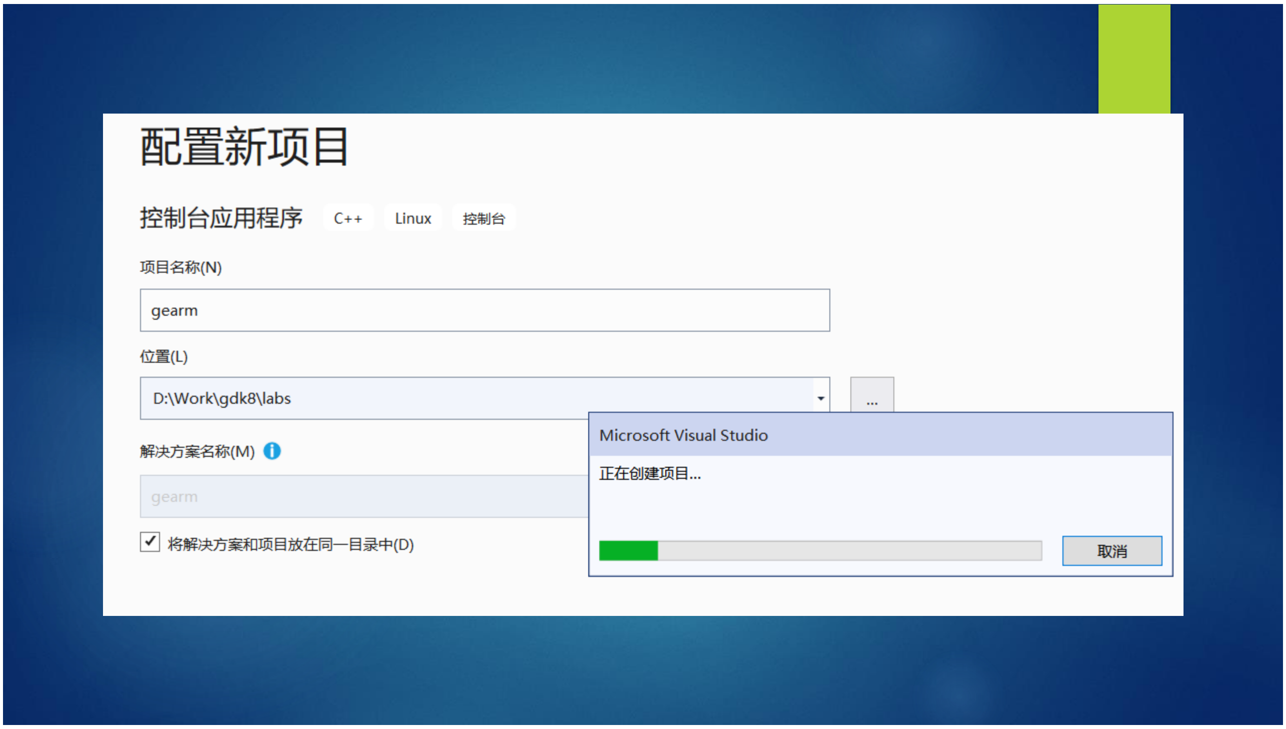
CMake 项目
生成不依赖于 .sln 或 .vcxproj 文件的新式跨平台 C++ 应用。
C++ Windows Linux 控制台

控制台应用程序
在 Linux 终端运行代码。默认打印 "hello"。
C++ Linux 控制台

空项目
使用 C++ for Linux 从头开始操作。不提供基础文件。
C++ Linux 控制台

Raspberry Pi 项目
使用 WiringPi for Raspberry Pi 的 LED 闪烁应用。
C++ Linux IoT 控制台

生成文件项目
自带生成系统来编译 C++ for Linux。
C++ Linux 库 控制台 桌面



配置新项目

控制台应用程序 C++ Linux 控制台

项目名称(N)

gearm

位置(L)

D:\Work\gdk8\labs

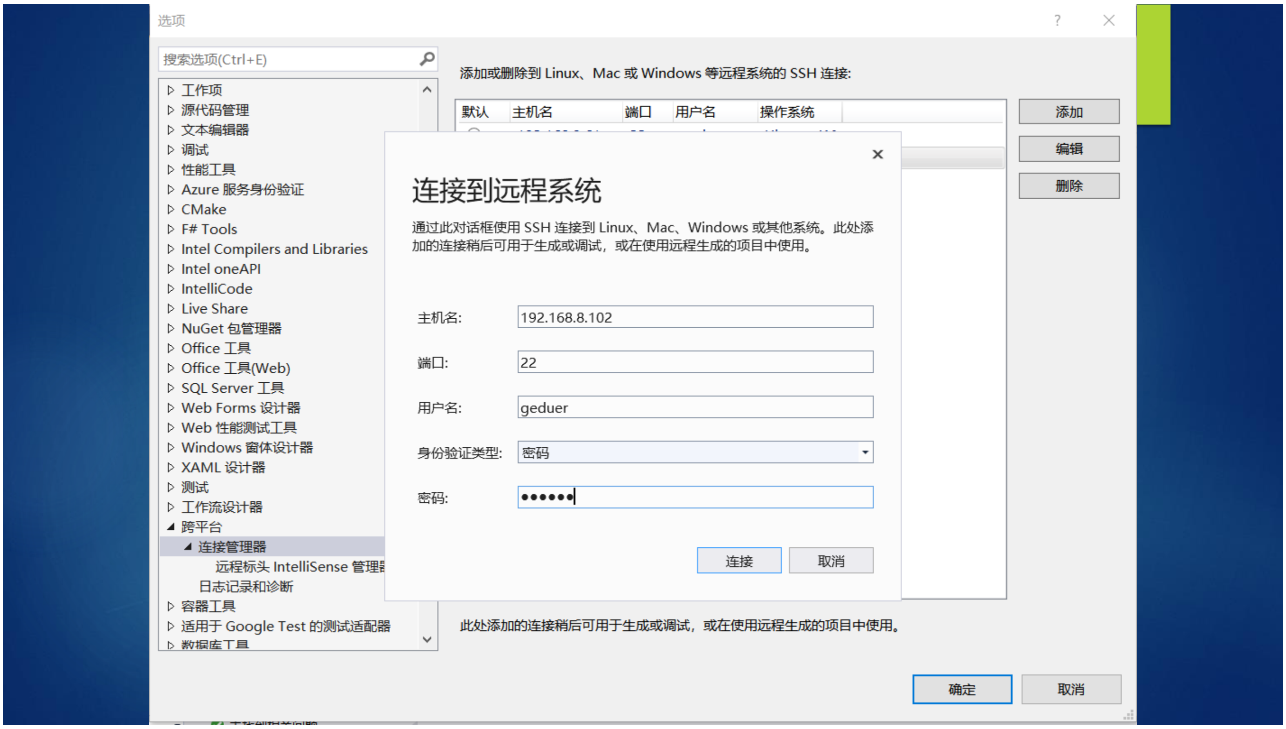
解决方案名称(M)

gearm

☑ 将解决方案和项目放在同一目录中(D)

Microsoft Visual Studio

正在创建项目...



添加或删除到 Linux、Mac 或 Windows 等远程系统的 SSH 连接:

默认	主机名	端口	用户名	操作系统

连接到远程系统

通过此对话框使用 SSH 连接到 Linux、Mac、Windows 或其他系统。此处添加的连接稍后可用于生成或调试，或在使用远程生成的项目中使用。

主机名: 192.168.8.102

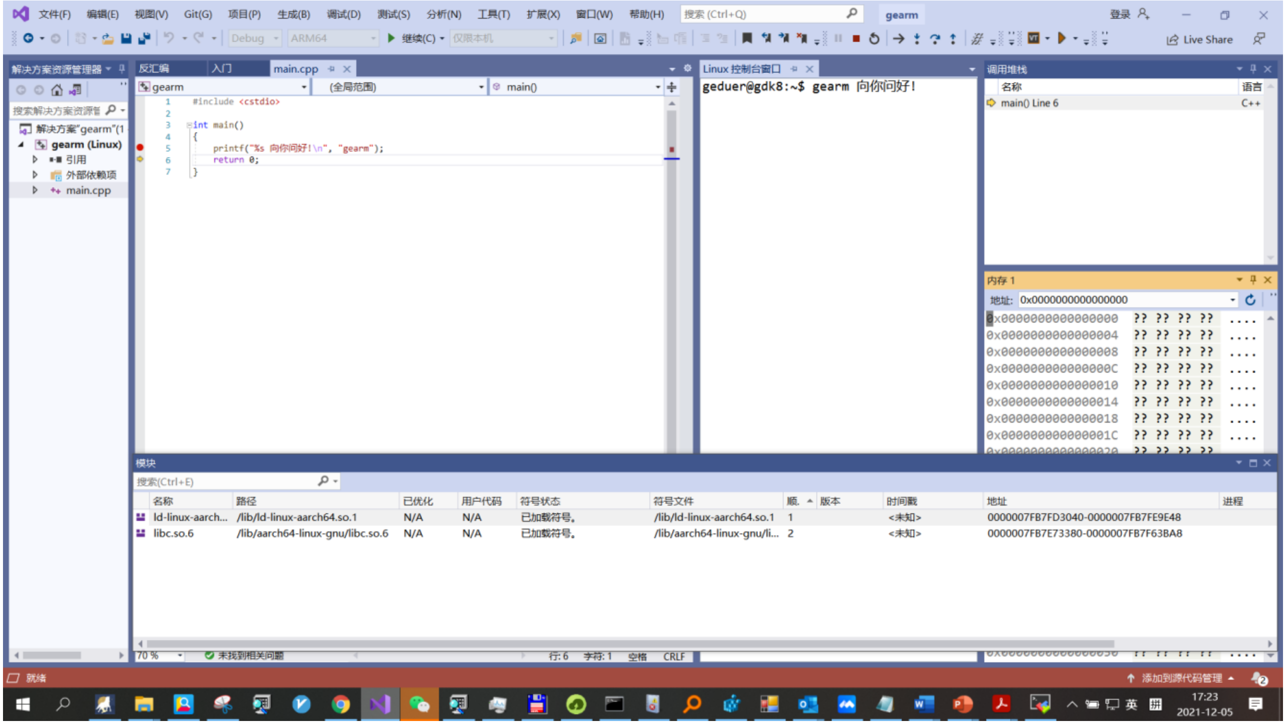
端口: 22

用户名: geduer

身份验证类型: 密码

密码: ••••••

此处添加的连接稍后可用于生成或调试，或在使用远程生成的项目中使用。



```cpp
#include <cstdio>

int main()
{
    printf("%s 向你问好!\n", "gearm");
    return 0;
}
```

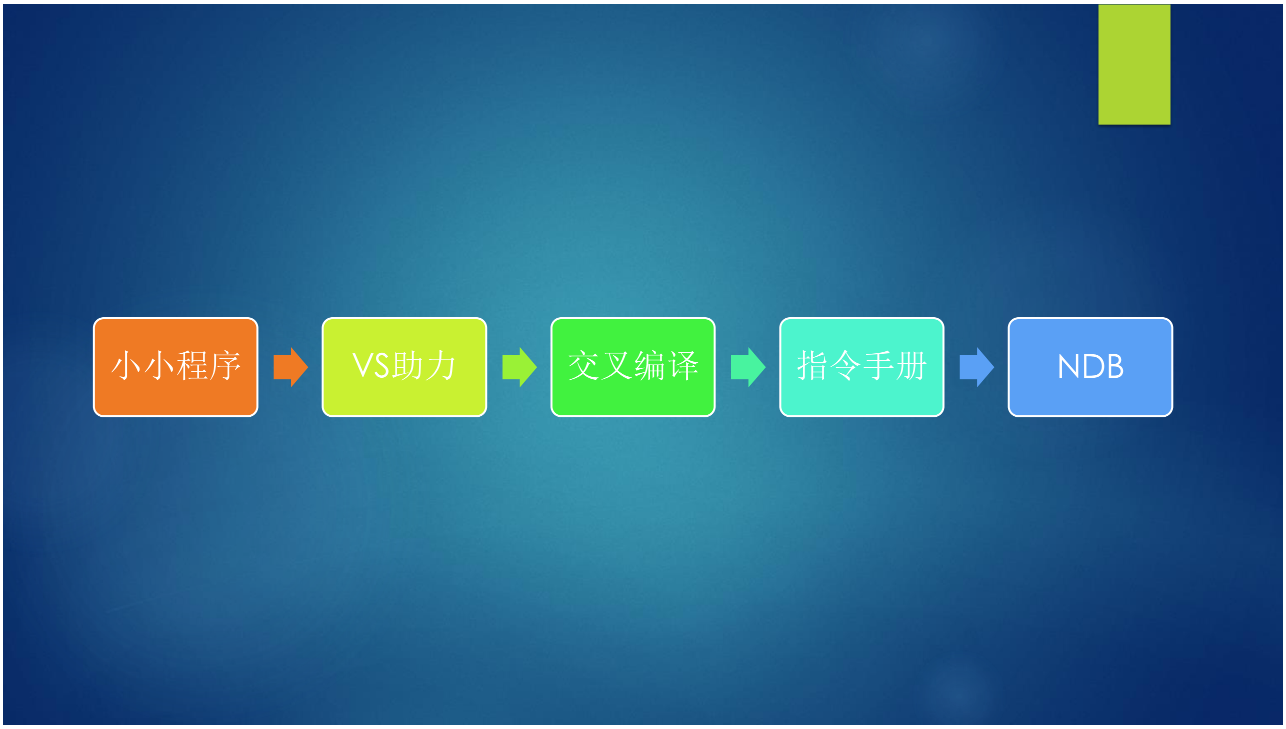
```
geduer@gdk8:~$ gearm 向你问好!
```

```
地址: 0x0000000000000000
0x0000000000000000  ?? ?? ?? ??  ....
0x0000000000000004  ?? ?? ?? ??  ....
0x0000000000000008  ?? ?? ?? ??  ....
0x000000000000000C  ?? ?? ?? ??  ....
0x0000000000000010  ?? ?? ?? ??  ....
0x0000000000000014  ?? ?? ?? ??  ....
0x0000000000000018  ?? ?? ?? ??  ....
0x000000000000001C  ?? ?? ?? ??  ....
```

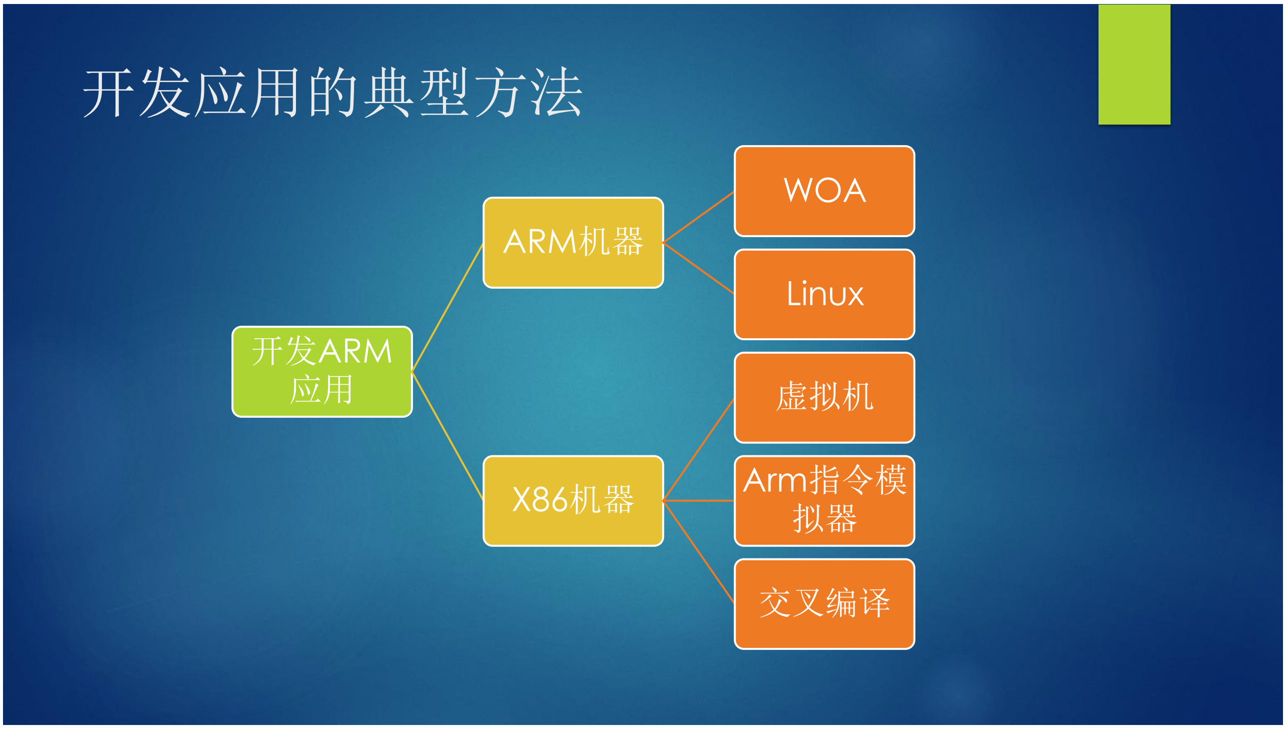
名称	路径	已优化	用户代码	符号状态	符号文件	顺.	版本	时间戳	地址	进程
ld-linux-aarch...	/lib/ld-linux-aarch64.so.1	N/A	N/A	已加载符号。	/lib/ld-linux-aarch64.so.1	1		<未知>	0000007FB7FD3040-0000007FB7FE9E48	
libc.so.6	/lib/aarch64-linux-gnu/libc.so.6	N/A	N/A	已加载符号。	/lib/aarch64-linux-gnu/li...	2		<未知>	0000007FB7E73380-0000007FB7F63BA8	

```
main():
0x0000005555555764 fd 7b bf a9          stp x29, x30, [sp, #-16]!
0x0000005555555768 fd 03 00 91          mov x29, sp
0x000000555555576c 00 00 00 90          adrp    x0, 0x5555555000
0x0000005555555770 01 c0 20 91          add x1, x0, #0x830
0x0000005555555770 01 c0 20 91          add x1, x0, #0x830
0x0000005555555774 00 00 00 90          adrp    x0, 0x5555555000
0x0000005555555778 00 e0 20 91          add x0, x0, #0x838
0x000000555555577c b5 ff ff 97          bl  0x5555555650 <printf@plt>
0x0000005555555780 00 00 80 52          mov w0, #0x0                // #0
0x0000005555555784 fd 7b c1 a8          ldp x29, x30, [sp], #16
0x0000005555555788 c0 03 5f d6          ret
```

VS从GDB获取，GDB风格



小小程序 → VS助力 → 交叉编译 → 指令手册 → NDB

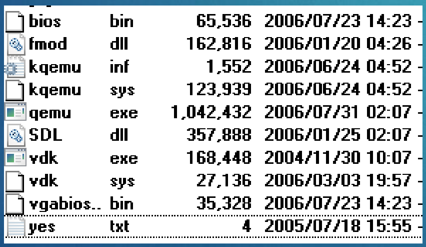


开发应用的典型方法

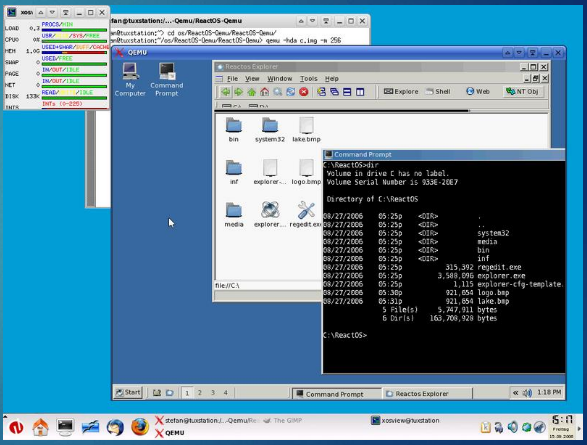
- 开发ARM应用
  - ARM机器
    - WOA
    - Linux
  - X86机器
    - 虚拟机
    - Arm指令模拟器
    - 交叉编译

QEMU

- Quick EMUlator
- 由法国人Fabrice Bellard开发
  - http://bellard.org/
- http://wiki.qemu.org



```
bios       bin      65,536  2006/07/23 14:23
fmod       dll     162,816  2006/01/20 04:26
kqemu      inf       1,552  2006/06/24 04:52
kqemu      sys     123,939  2006/06/24 04:52
qemu       exe   1,042,432  2006/07/31 02:07
SDL        dll     357,888  2006/01/25 02:07
vdk        exe     168,448  2004/11/30 10:07
vdk        sys      27,136  2006/03/03 19:57
vgabios..  bin      35,328  2006/07/23 14:23
yes        txt           4  2005/07/18 15:55
```



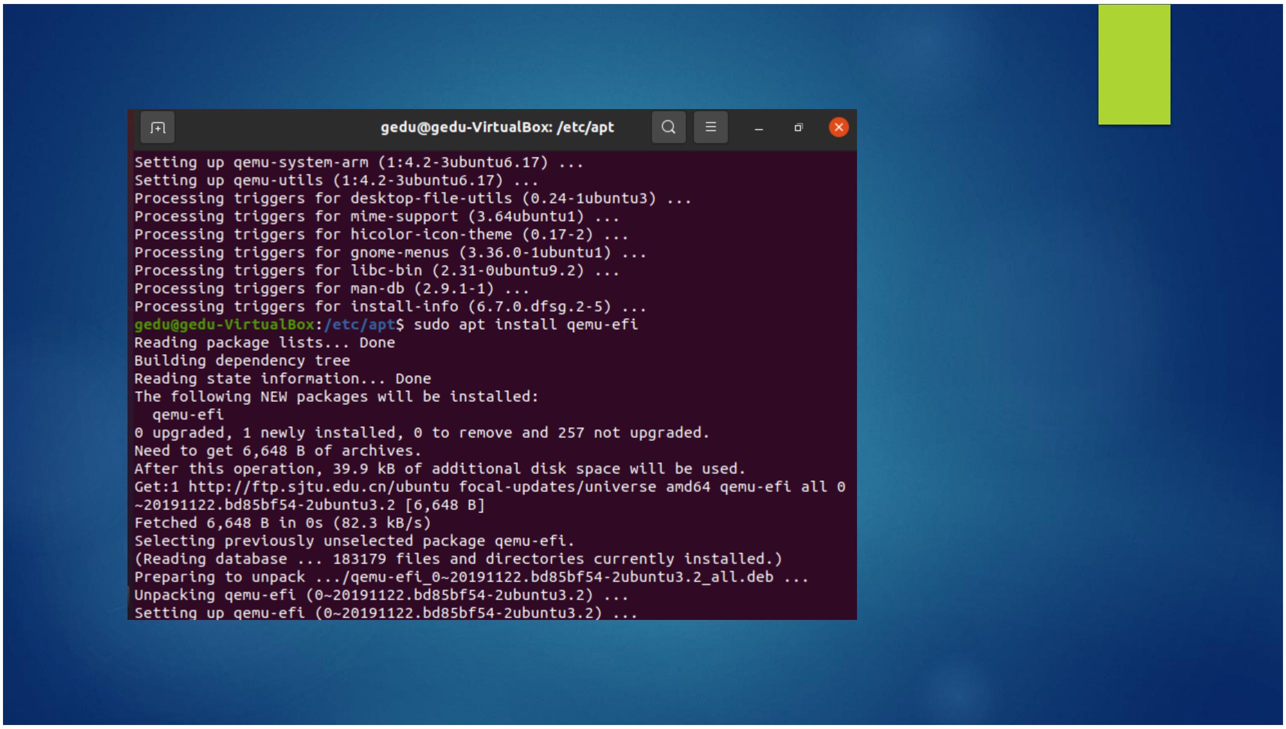
```
an@tuxstation:~> cd os/ReactOS-Qemu/ReactOS-Qemu/
an@tuxstation:~/os/ReactOS-Qemu/ReactOS-Qemu> qemu -hda c.img -m 256
```

```
C:\ReactOS>dir
 Volume in drive C has no label.
 Volume Serial Number is 933E-20E7

 Directory of C:\ReactOS

08/27/2006   05:25p    <DIR>          .
08/27/2006   05:25p    <DIR>          ..
08/27/2006   05:25p    <DIR>          system32
08/27/2006   05:25p    <DIR>          media
08/27/2006   05:25p    <DIR>          bin
08/27/2006   05:25p    <DIR>          inf
08/27/2006   05:25p           315,392 regedit.exe
08/27/2006   05:25p         3,588,096 explorer.exe
08/27/2006   05:25p             1,115 explorer-cfg-template.
08/27/2006   05:30p           921,654 logo.bmp
08/27/2006   05:31p           921,654 lake.bmp
               5 File(s)      5,747,911 bytes
               6 Dir(s)     163,708,928 bytes

C:\ReactOS>
```

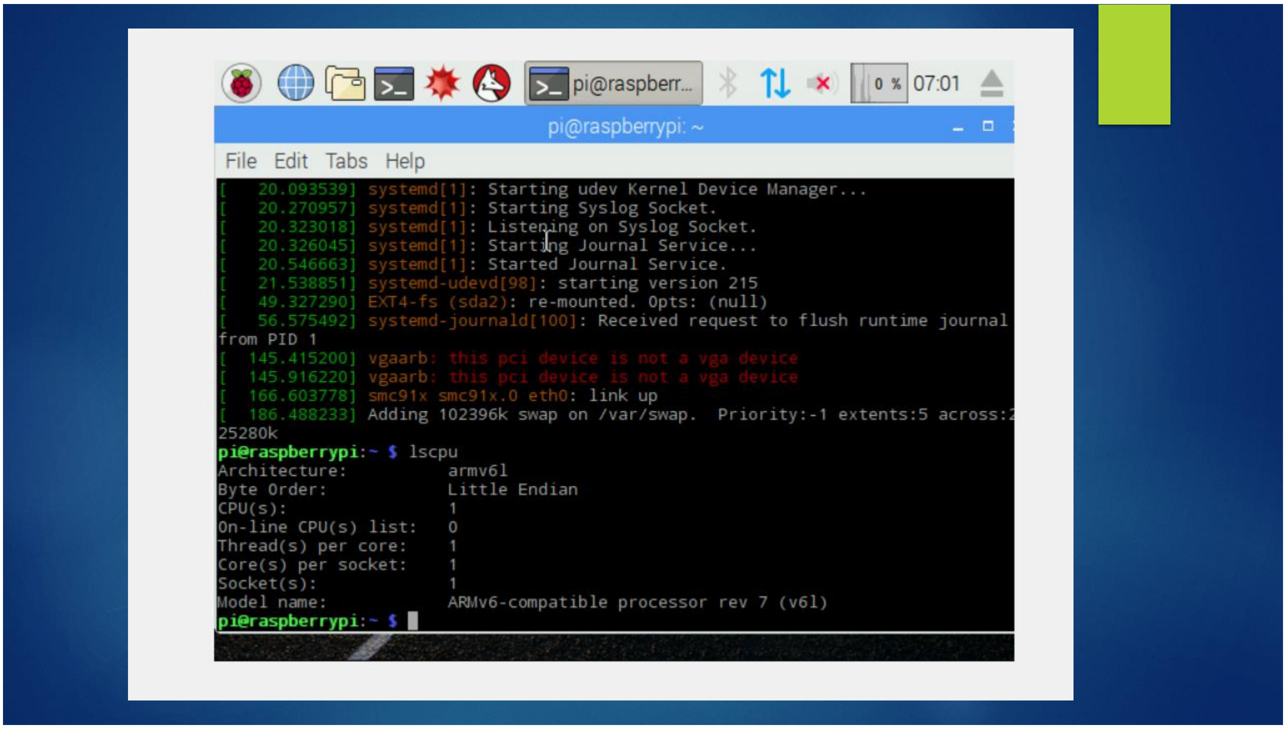
在Ubuntu中安装QEMU

- sudo apt-get install qemu-system-arm qemu-efi
- $ dd if=/dev/zero of=flash0.img bs=1M count=64
- $ dd if=/usr/share/qemu-efi/QEMU_EFI.fd of=flash0.img conv=notrunc
- $ dd if=/dev/zero of=flash1.img bs=1M count=64
- sudo qemu-system-aarch64 -m 1024 -cpu cortex-a57 -M virt -nographic -pflash flash0.img -pflash flash1.img -drive if=none,file=vivid-server-cloudimg-arm64-uefi1.img,id=hd0 -device virtio-blk-device,drive=hd0 -netdev type=tap,id=net0 -device virtio-net-device,netdev=net0,mac=$randmac

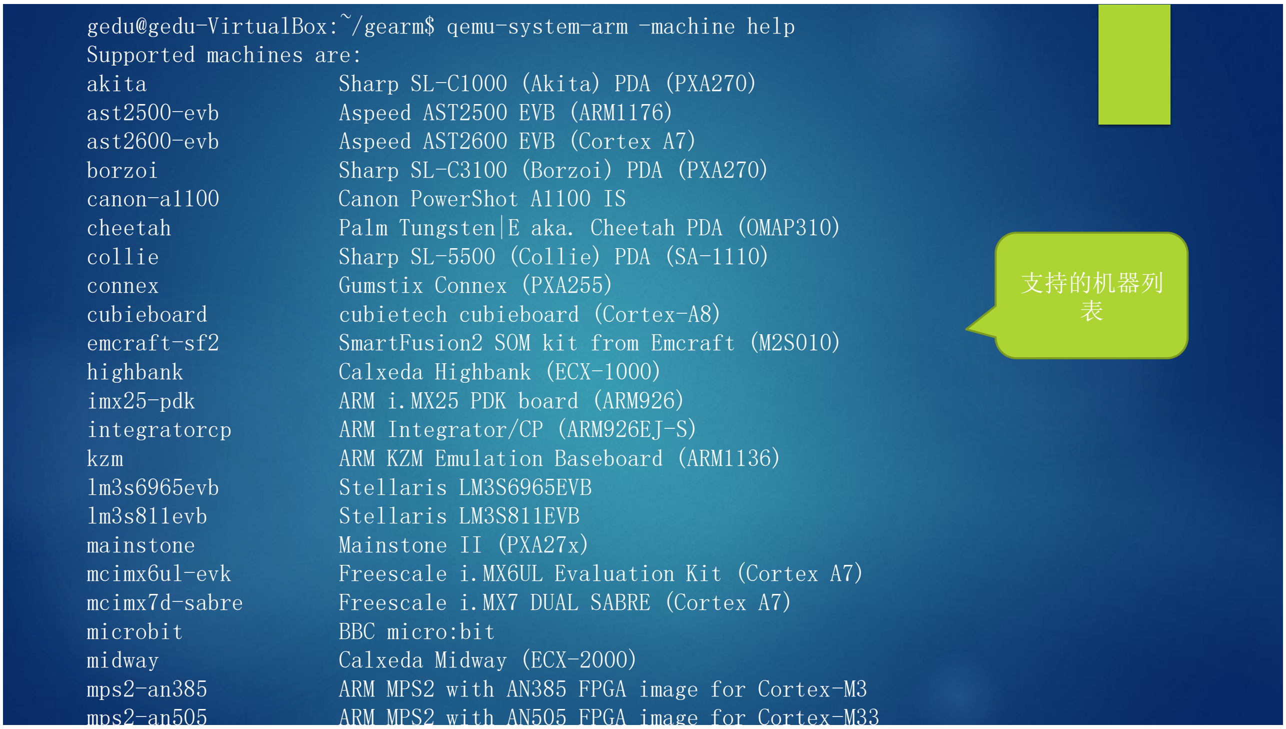
https://wiki.ubuntu.com/ARM64/QEMU



```
Setting up qemu-system-arm (1:4.2-3ubuntu6.17) ...
Setting up qemu-utils (1:4.2-3ubuntu6.17) ...
Processing triggers for desktop-file-utils (0.24-1ubuntu3) ...
Processing triggers for mime-support (3.64ubuntu1) ...
Processing triggers for hicolor-icon-theme (0.17-2) ...
Processing triggers for gnome-menus (3.36.0-1ubuntu1) ...
Processing triggers for libc-bin (2.31-0ubuntu9.2) ...
Processing triggers for man-db (2.9.1-1) ...
Processing triggers for install-info (6.7.0.dfsg.2-5) ...
gedu@gedu-VirtualBox:/etc/apt$ sudo apt install qemu-efi
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  qemu-efi
0 upgraded, 1 newly installed, 0 to remove and 257 not upgraded.
Need to get 6,648 B of archives.
After this operation, 39.9 kB of additional disk space will be used.
Get:1 http://ftp.sjtu.edu.cn/ubuntu focal-updates/universe amd64 qemu-efi all 0
~20191122.bd85bf54-2ubuntu3.2 [6,648 B]
Fetched 6,648 B in 0s (82.3 kB/s)
Selecting previously unselected package qemu-efi.
(Reading database ... 183179 files and directories currently installed.)
Preparing to unpack .../qemu-efi_0~20191122.bd85bf54-2ubuntu3.2_all.deb ...
Unpacking qemu-efi (0~20191122.bd85bf54-2ubuntu3.2) ...
Setting up qemu-efi (0~20191122.bd85bf54-2ubuntu3.2) ...
```

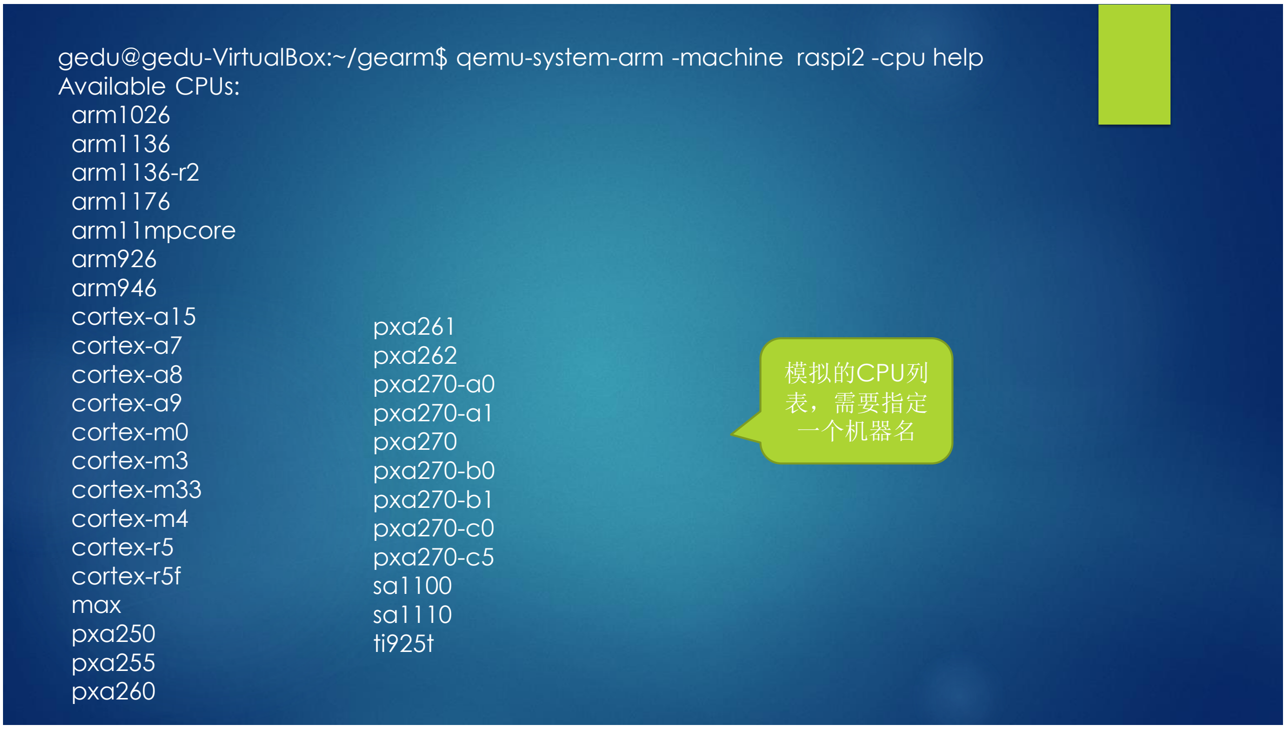


```
[   20.093539] systemd[1]: Starting udev Kernel Device Manager...
[   20.270957] systemd[1]: Starting Syslog Socket.
[   20.323018] systemd[1]: Listening on Syslog Socket.
[   20.326045] systemd[1]: Starting Journal Service...
[   20.546663] systemd[1]: Started Journal Service.
[   21.538851] systemd-udevd[98]: starting version 215
[   49.327290] EXT4-fs (sda2): re-mounted. Opts: (null)
[   56.575492] systemd-journald[100]: Received request to flush runtime journal
from PID 1
[  145.415200] vgaarb: this pci device is not a vga device
[  145.916220] vgaarb: this pci device is not a vga device
[  166.603778] smc91x smc91x.0 eth0: link up
[  186.488233] Adding 102396k swap on /var/swap.  Priority:-1 extents:5 across:2
25280k
pi@raspberrypi:~ $ lscpu
Architecture:          armv6l
Byte Order:            Little Endian
CPU(s):                1
On-line CPU(s) list:   0
Thread(s) per core:    1
Core(s) per socket:    1
Socket(s):             1
Model name:            ARMv6-compatible processor rev 7 (v6l)
pi@raspberrypi:~ $ 
```



```
gedu@gedu-VirtualBox:~/gearm$ qemu-system-arm -machine help
Supported machines are:
akita                Sharp SL-C1000 (Akita) PDA (PXA270)
ast2500-evb          Aspeed AST2500 EVB (ARM1176)
ast2600-evb          Aspeed AST2600 EVB (Cortex A7)
borzoi               Sharp SL-C3100 (Borzoi) PDA (PXA270)
canon-a1100          Canon PowerShot A1100 IS
cheetah              Palm Tungsten|E aka. Cheetah PDA (OMAP310)
collie               Sharp SL-5500 (Collie) PDA (SA-1110)
connex               Gumstix Connex (PXA255)
cubieboard           cubietech cubieboard (Cortex-A8)
emcraft-sf2          SmartFusion2 SOM kit from Emcraft (M2S010)
highbank             Calxeda Highbank (ECX-1000)
imx25-pdk            ARM i.MX25 PDK board (ARM926)
integratorcp         ARM Integrator/CP (ARM926EJ-S)
kzm                  ARM KZM Emulation Baseboard (ARM1136)
lm3s6965evb          Stellaris LM3S6965EVB
lm3s811evb           Stellaris LM3S811EVB
mainstone            Mainstone II (PXA27x)
mcimx6ul-evk         Freescale i.MX6UL Evaluation Kit (Cortex A7)
mcimx7d-sabre        Freescale i.MX7 DUAL SABRE (Cortex A7)
microbit             BBC micro:bit
midway               Calxeda Midway (ECX-2000)
mps2-an385           ARM MPS2 with AN385 FPGA image for Cortex-M3
mps2-an505           ARM MPS2 with AN505 FPGA image for Cortex-M33
```

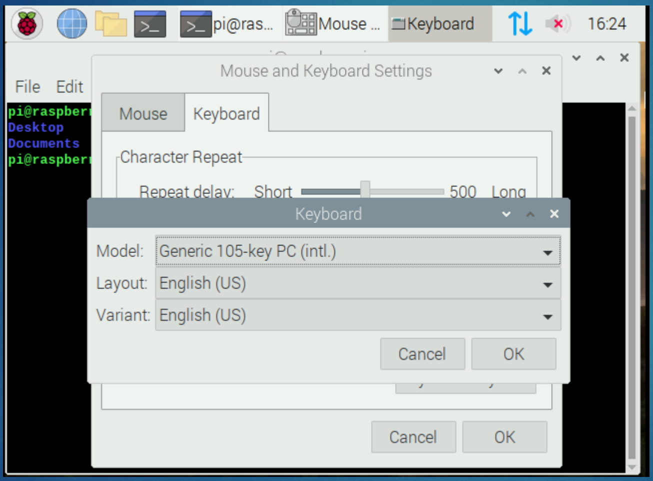
支持的机器列表



```
gedu@gedu-VirtualBox:~/gearm$ qemu-system-arm -machine  raspi2 -cpu help
Available CPUs:
 arm1026
 arm1136
 arm1136-r2
 arm1176
 arm11mpcore
 arm926
 arm946
 cortex-a15                pxa261
 cortex-a7                 pxa262
 cortex-a8                 pxa270-a0
 cortex-a9                 pxa270-a1
 cortex-m0                 pxa270
 cortex-m3                 pxa270-b0
 cortex-m33                pxa270-b1
 cortex-m4                 pxa270-c0
 cortex-r5                 pxa270-c5
 cortex-r5f                sa1100
 max                       sa1110
 pxa250                    ti925t
 pxa255
 pxa260
```

模拟的CPU列表，需要指定一个机器名

调整键盘布局



```
pi@raspberr
Desktop
Documents
pi@raspberr
```

Character Repeat

Repeat delay: Short 500 Long

Keyboard

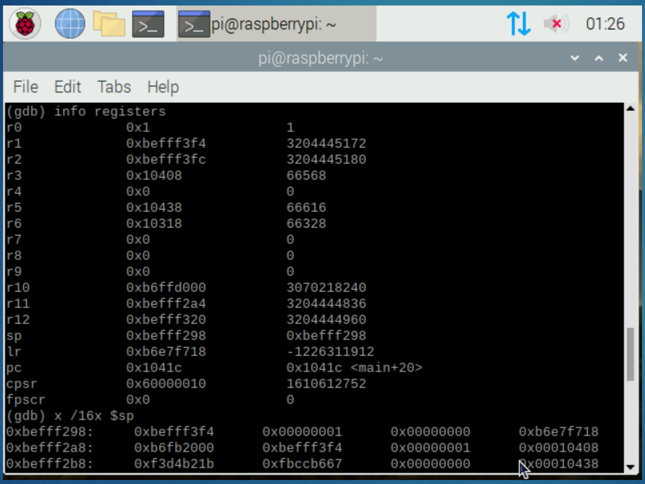
Model: Generic 105-key PC (intl.)

Layout: English (US)

Variant: English (US)

由 English (UK)改
为 English (US)
不然无法输入#

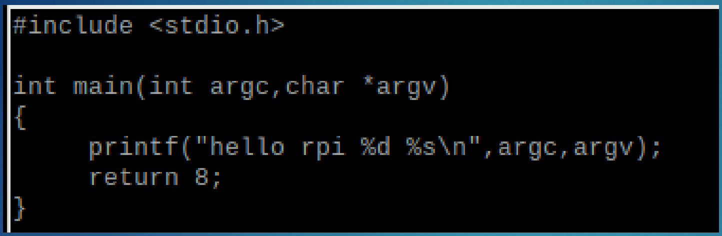
观察栈上的原始数据



```
(gdb) info registers
r0             0x1                 1
r1             0xbefff3f4          3204445172
r2             0xbefff3fc          3204445180
r3             0x10408             66568
r4             0x0                 0
r5             0x10438             66616
r6             0x10318             66328
r7             0x0                 0
r8             0x0                 0
r9             0x0                 0
r10            0xb6ffd000          3070218240
r11            0xbefff2a4          3204444836
r12            0xbefff320          3204444960
sp             0xbefff298          0xbefff298
lr             0xb6e7f718          -1226311912
pc             0x1041c             0x1041c <main+20>
cpsr           0x60000010          1610612752
fpscr          0x0                 0
(gdb) x /16x $sp
0xbefff298:     0xbefff3f4      0x00000001      0x00000000      0xb6e7f718
0xbefff2a8:     0xb6fb2000      0xbefff3f4      0x00000001      0x00010408
0xbefff2b8:     0xf3d4b21b      0xfbccb667      0x00000000      0x00010438
```

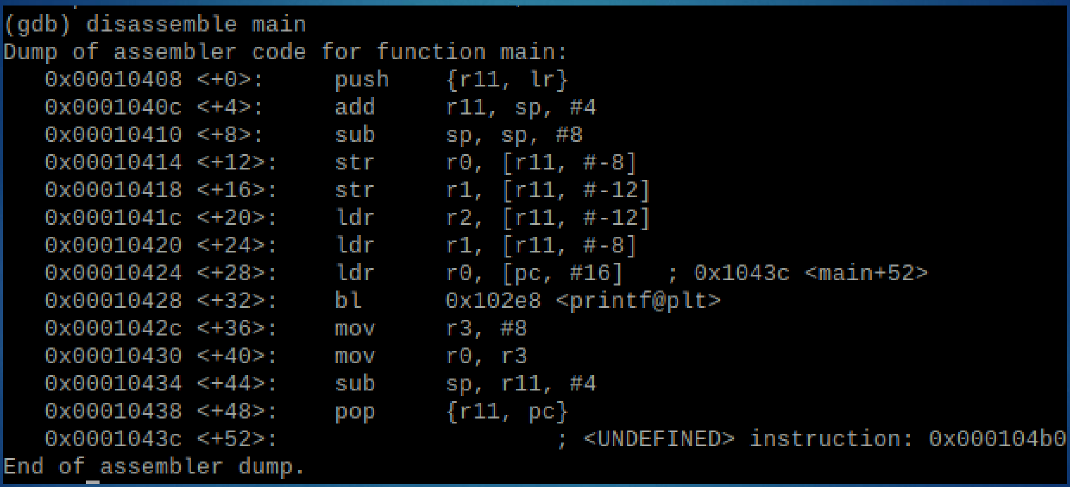
b __brk
b main
bt
x /32x $sp

Hello



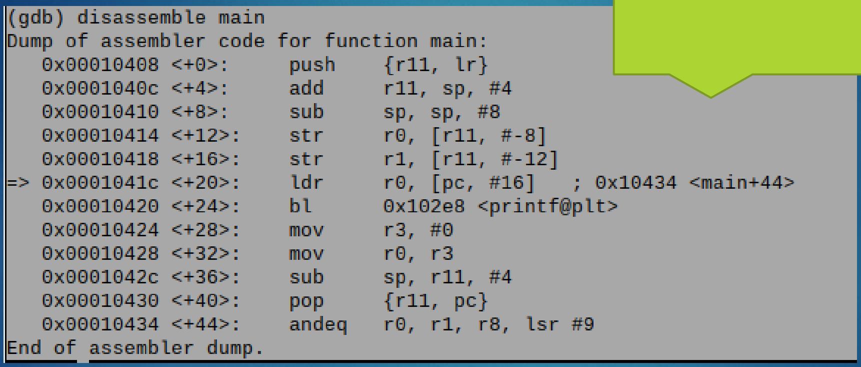
```
#include <stdio.h>

int main(int argc,char *argv)
{
      printf("hello rpi %d %s\n",argc,argv);
      return 8;
}
```

# gcc –g3 –o hello hello.c



```
(gdb) disassemble main
Dump of assembler code for function main:
   0x00010408 <+0>:     push    {r11, lr}
   0x0001040c <+4>:     add     r11, sp, #4
   0x00010410 <+8>:     sub     sp, sp, #8
   0x00010414 <+12>:    str     r0, [r11, #-8]
   0x00010418 <+16>:    str     r1, [r11, #-12]
   0x0001041c <+20>:    ldr     r2, [r11, #-12]
   0x00010420 <+24>:    ldr     r1, [r11, #-8]
   0x00010424 <+28>:    ldr     r0, [pc, #16]   ; 0x1043c <main+52>
   0x00010428 <+32>:    bl      0x102e8 <printf@plt>
   0x0001042c <+36>:    mov     r3, #8
   0x00010430 <+40>:    mov     r0, r3
   0x00010434 <+44>:    sub     sp, r11, #4
   0x00010438 <+48>:    pop     {r11, pc}
   0x0001043c <+52>:                    ; <UNDEFINED> instruction: 0x000104b0
End of assembler dump.
```

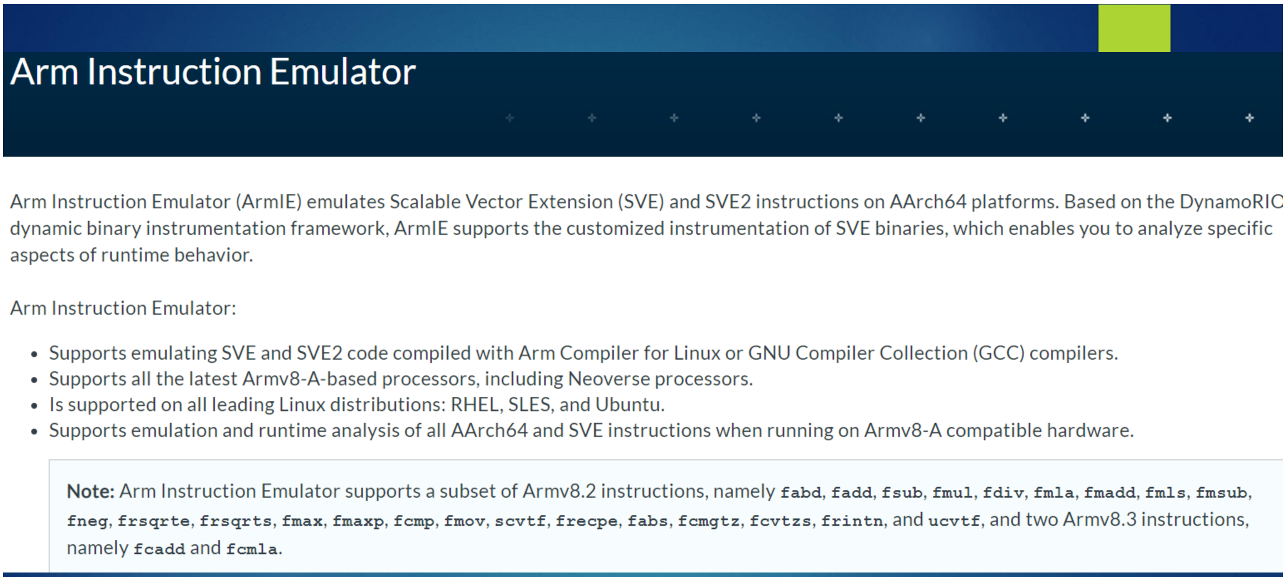
ARMv7的A32指令



```
(gdb) disassemble main
Dump of assembler code for function main:
   0x00010408 <+0>:     push    {r11, lr}
   0x0001040c <+4>:     add     r11, sp, #4
   0x00010410 <+8>:     sub     sp, sp, #8
   0x00010414 <+12>:    str     r0, [r11, #-8]
   0x00010418 <+16>:    str     r1, [r11, #-12]
=> 0x0001041c <+20>:    ldr     r0, [pc, #16]   ; 0x10434 <main+44>
   0x00010420 <+24>:    bl      0x102e8 <printf@plt>
   0x00010424 <+28>:    mov     r3, #0
   0x00010428 <+32>:    mov     r0, r3
   0x0001042c <+36>:    sub     sp, r11, #4
   0x00010430 <+40>:    pop     {r11, pc}
   0x00010434 <+44>:    andeq   r0, r1, r8, lsr #9
End of assembler dump.
```

安装Arm Instruction Emulator

- tar -xvzf <package_name>.tar.gz
- cd arm-instruction-emulator_21.0_Ubuntu-18.04_aarch64/
- sudo ./arm-instruction-emulator-21.0*_aarch64-linux-rpm.sh <option> <option>



Arm Instruction Emulator

Arm Instruction Emulator (ArmIE) emulates Scalable Vector Extension (SVE) and SVE2 instructions on AArch64 platforms. Based on the DynamoRIO dynamic binary instrumentation framework, ArmIE supports the customized instrumentation of SVE binaries, which enables you to analyze specific aspects of runtime behavior.

Arm Instruction Emulator:

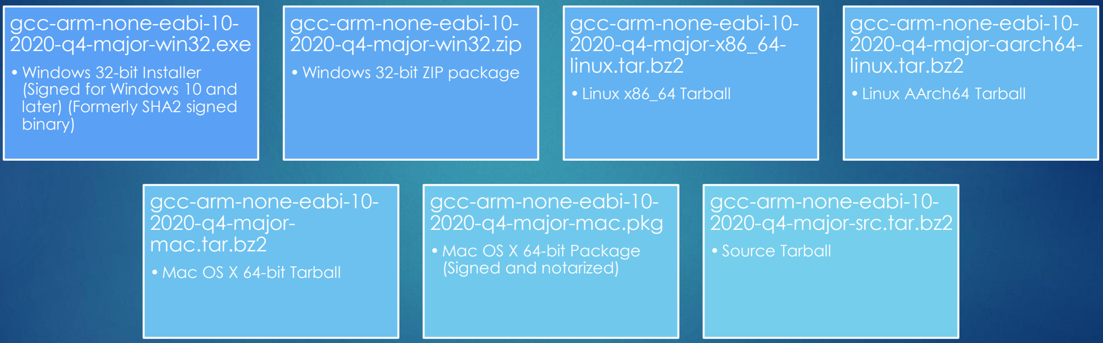
- Supports emulating SVE and SVE2 code compiled with Arm Compiler for Linux or GNU Compiler Collection (GCC) compilers.
- Supports all the latest Armv8-A-based processors, including Neoverse processors.
- Is supported on all leading Linux distributions: RHEL, SLES, and Ubuntu.
- Supports emulation and runtime analysis of all AArch64 and SVE instructions when running on Armv8-A compatible hardware.

Note: Arm Instruction Emulator supports a subset of Armv8.2 instructions, namely `fabd`, `fadd`, `fsub`, `fmul`, `fdiv`, `fmla`, `fmadd`, `fmls`, `fmsub`, `fneg`, `frsqrte`, `frsqrts`, `fmax`, `fmaxp`, `fcmp`, `fmov`, `scvtf`, `frecpe`, `fabs`, `fcmgtz`, `fcvtzs`, `frintn`, and `ucvtf`, and two Armv8.3 instructions, namely `fcadd` and `fcmla`.

- https://developer.arm.com/documentation/102190/2100/Get-started/Get-started-with-Arm-Instruction-Emulator?lang=en

gcc-arm-none-eabi

- GCC cross compiler for ARM Cortex-A/R/M processors



gcc-arm-none-eabi-10-2020-q4-major-win32.exe
- Windows 32-bit Installer (Signed for Windows 10 and later) (Formerly SHA2 signed binary)

gcc-arm-none-eabi-10-2020-q4-major-win32.zip
- Windows 32-bit ZIP package

gcc-arm-none-eabi-10-2020-q4-major-x86_64-linux.tar.bz2
- Linux x86_64 Tarball

gcc-arm-none-eabi-10-2020-q4-major-aarch64-linux.tar.bz2
- Linux AArch64 Tarball

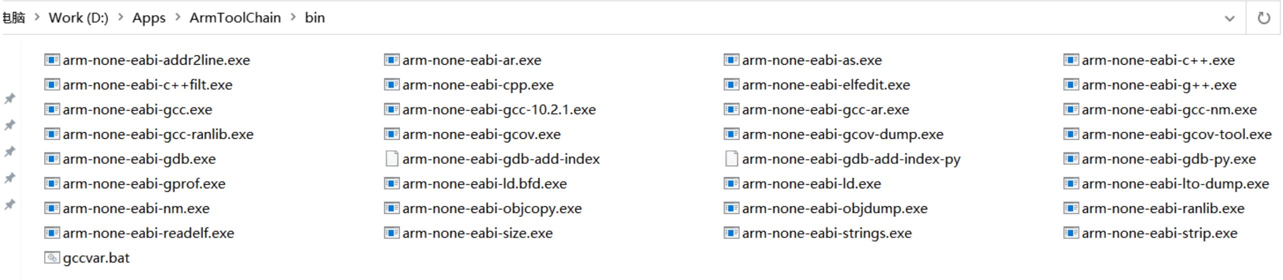
gcc-arm-none-eabi-10-2020-q4-major-mac.tar.bz2
- Mac OS X 64-bit Tarball

gcc-arm-none-eabi-10-2020-q4-major-mac.pkg
- Mac OS X 64-bit Package (Signed and notarized)

gcc-arm-none-eabi-10-2020-q4-major-src.tar.bz2
- Source Tarball

https://developer.arm.com/tools-and-software/open-source-software/developer-tools/gnu-toolchain/gnu-rm/downloads

Windows版本



电脑 › Work (D:) › Apps › ArmToolChain › bin

```
arm-none-eabi-addr2line.exe
arm-none-eabi-ar.exe
arm-none-eabi-as.exe
arm-none-eabi-c++.exe
arm-none-eabi-c++filt.exe
arm-none-eabi-cpp.exe
arm-none-eabi-elfedit.exe
arm-none-eabi-g++.exe
arm-none-eabi-gcc.exe
arm-none-eabi-gcc-10.2.1.exe
arm-none-eabi-gcc-ar.exe
arm-none-eabi-gcc-nm.exe
arm-none-eabi-gcc-ranlib.exe
arm-none-eabi-gcov.exe
arm-none-eabi-gcov-dump.exe
arm-none-eabi-gcov-tool.exe
arm-none-eabi-gdb.exe
arm-none-eabi-gdb-add-index
arm-none-eabi-gdb-add-index-py
arm-none-eabi-gdb-py.exe
arm-none-eabi-gprof.exe
arm-none-eabi-ld.bfd.exe
arm-none-eabi-ld.exe
arm-none-eabi-lto-dump.exe
arm-none-eabi-nm.exe
arm-none-eabi-objcopy.exe
arm-none-eabi-objdump.exe
arm-none-eabi-ranlib.exe
arm-none-eabi-readelf.exe
arm-none-eabi-size.exe
arm-none-eabi-strings.exe
arm-none-eabi-strip.exe
gccvar.bat
```

简单的死循环

bare metal

```
.globl _start

_start:
  b  main

main:
  b  main
```

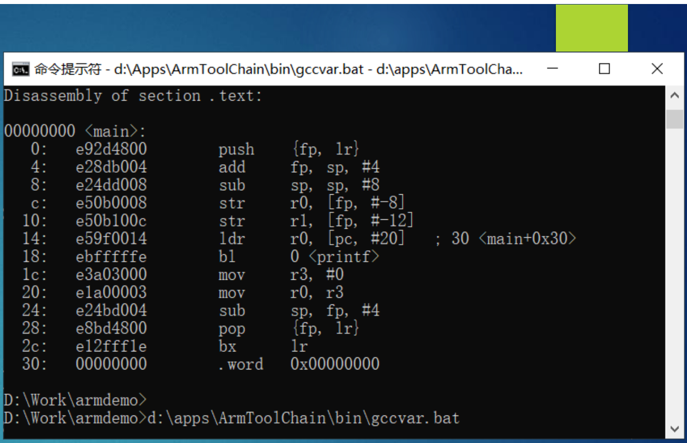
https://czak.pl/2020/04/24/bare-tinker-build-boot-hang.html

编译

```
arm-none-eabi-gcc -nostdlib -Ttext=0xff704000 main.S -o boot.elf
```

小试验

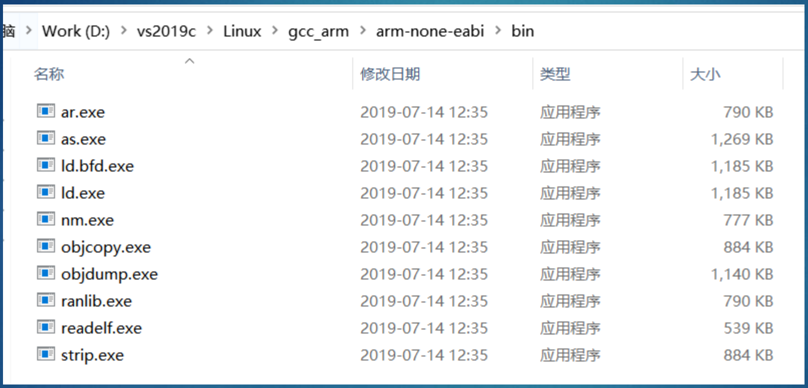
```
#include <stdio.h>

int main(int argc, char* argv[])
{
   printf("hello arm");
}
```


命令提示符 - d:\Apps\ArmToolChain\bin\gccvar.bat - d:\apps\ArmToolCha...

```
Disassembly of section .text:

00000000 <main>:
   0:   e92d4800        push    {fp, lr}
   4:   e28db004        add     fp, sp, #4
   8:   e24dd008        sub     sp, sp, #8
   c:   e50b0008        str     r0, [fp, #-8]
  10:   e50b100c        str     r1, [fp, #-12]
  14:   e59f0014        ldr     r0, [pc, #20]   ; 30 <main+0x30>
  18:   ebfffffe        bl      0 <printf>
  1c:   e3a03000        mov     r3, #0
  20:   e1a00003        mov     r0, r3
  24:   e24bd004        sub     sp, fp, #4
  28:   e8bd4800        pop     {fp, lr}
  2c:   e12fff1e        bx      lr
  30:   00000000        .word   0x00000000

D:\Work\armdemo>
D:\Work\armdemo>d:\apps\ArmToolChain\bin\gccvar.bat
```

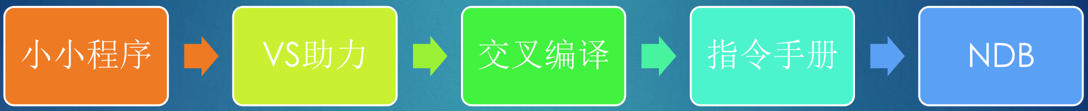
- d:\apps\ArmToolChain\bin\gccvar.bat
- D:\Work\armdemo>arm-none-eabi-gcc -c gearm.c
- D:\Work\armdemo>arm-none-eabi-objdump -d gearm.o

VS2019中集成的Arm工具链



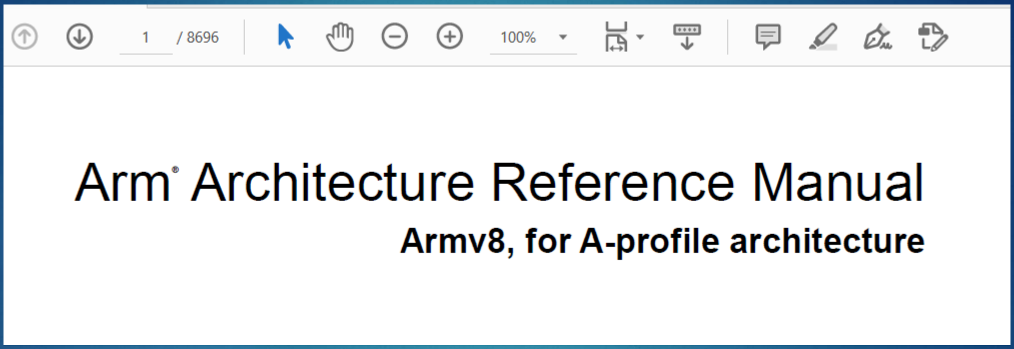
脑 › Work (D:) › vs2019c › Linux › gcc_arm › arm-none-eabi › bin

名称	修改日期	类型	大小
ar.exe	2019-07-14 12:35	应用程序	790 KB
as.exe	2019-07-14 12:35	应用程序	1,269 KB
ld.bfd.exe	2019-07-14 12:35	应用程序	1,185 KB
ld.exe	2019-07-14 12:35	应用程序	1,185 KB
nm.exe	2019-07-14 12:35	应用程序	777 KB
objcopy.exe	2019-07-14 12:35	应用程序	884 KB
objdump.exe	2019-07-14 12:35	应用程序	1,140 KB
ranlib.exe	2019-07-14 12:35	应用程序	790 KB
readelf.exe	2019-07-14 12:35	应用程序	539 KB
strip.exe	2019-07-14 12:35	应用程序	884 KB



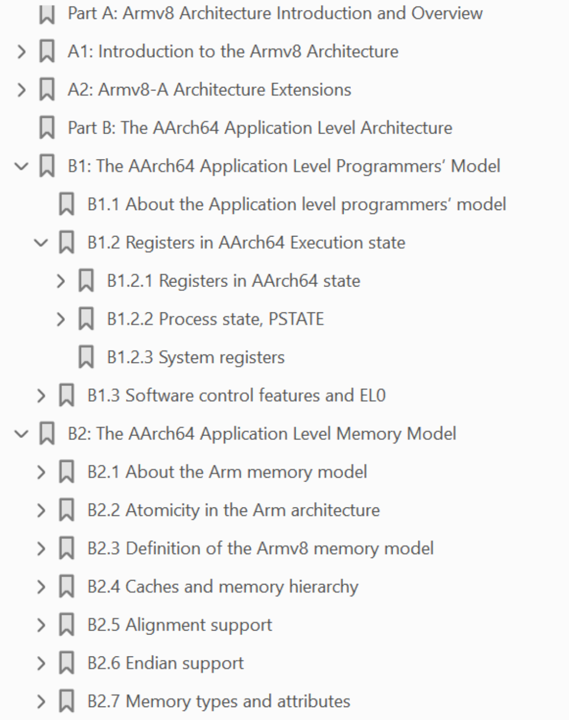
小小程序 → VS助力 → 交叉编译 → 指令手册 → NDB

22 July 2021版本，8696页



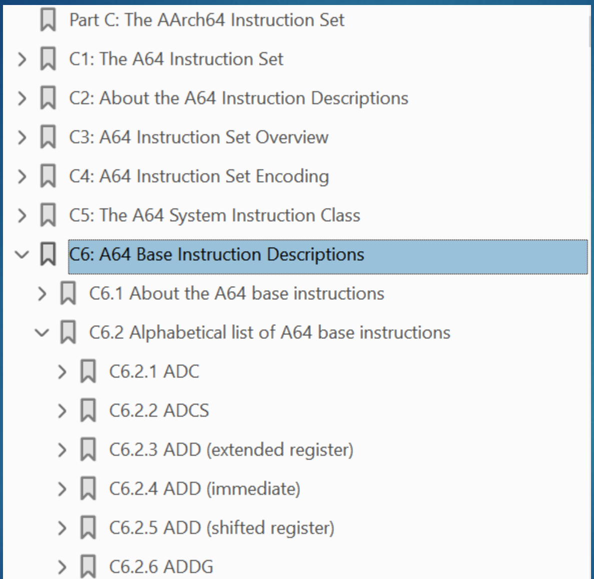
Arm® Architecture Reference Manual

Armv8, for A-profile architecture



- Part A: Armv8 Architecture Introduction and Overview
- A1: Introduction to the Armv8 Architecture
- A2: Armv8-A Architecture Extensions
- Part B: The AArch64 Application Level Architecture
- B1: The AArch64 Application Level Programmers' Model
  - B1.1 About the Application level programmers' model
  - B1.2 Registers in AArch64 Execution state
    - B1.2.1 Registers in AArch64 state
    - B1.2.2 Process state, PSTATE
    - B1.2.3 System registers
  - B1.3 Software control features and EL0
- B2: The AArch64 Application Level Memory Model
  - B2.1 About the Arm memory model
  - B2.2 Atomicity in the Arm architecture
  - B2.3 Definition of the Armv8 memory model
  - B2.4 Caches and memory hierarchy
  - B2.5 Alignment support
  - B2.6 Endian support
  - B2.7 Memory types and attributes

Part A， B为初级内容，大体相当于INTEL SDM的卷A



Part C: The AArch64 Instruction Set

C1: The A64 Instruction Set

C2: About the A64 Instruction Descriptions

C3: A64 Instruction Set Overview

C4: A64 Instruction Set Encoding

C5: The A64 System Instruction Class

C6: A64 Base Instruction Descriptions

C6.1 About the A64 base instructions

C6.2 Alphabetical list of A64 base instructions

C6.2.1 ADC

C6.2.2 ADCS

C6.2.3 ADD (extended register)

C6.2.4 ADD (immediate)

C6.2.5 ADD (shifted register)

C6.2.6 ADDG

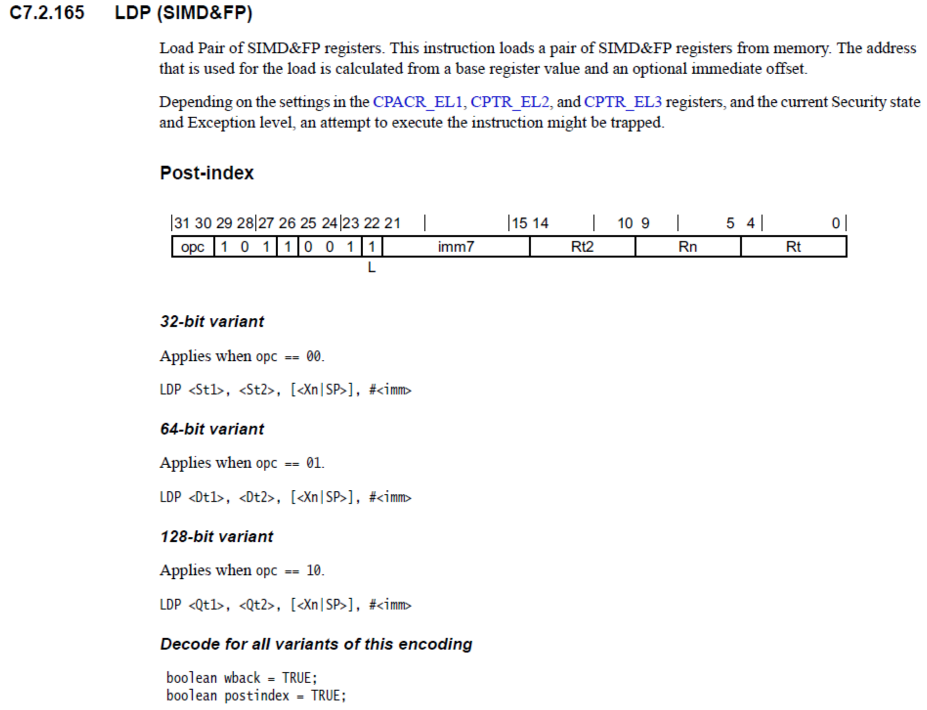
卷C为指令集，相当于Intel SDM的卷B

读写内存块（多个单元）- A32/T32

- LDM (load multiple) and STM (store multiple)
  - ldm r0, {r4,r5} 把r0指向的内存块读到R4和R5
  - stm r1, {r4,r5} 把r4和r5写到r1指向的内存块
- LDM后面的后缀
  - -IA (increase after), -IB (increase before), -DA (decrease after), -DB (decrease before)

LDP, LDPSW, STP – A64

- LDP – Load Pair
  - 从内存读一对数据到寄存器
- LDPSW – Load Pair with signed words
  - 从内存读一对数据，使用有符号逻辑
- STP – Store Pair
  - 写一对数据到内存



C7.2.165 LDP (SIMD&FP)

Load Pair of SIMD&FP registers. This instruction loads a pair of SIMD&FP registers from memory. The address that is used for the load is calculated from a base register value and an optional immediate offset.

Depending on the settings in the CPACR_EL1, CPTR_EL2, and CPTR_EL3 registers, and the current Security state and Exception level, an attempt to execute the instruction might be trapped.

Post-index

31 30	29 28 27	26	25 24 23	22	21 15	14 10	9 5	4 0
opc	1 0 1	1	0 0 1	1	imm7	Rt2	Rn	Rt
				L				

32-bit variant

Applies when opc == 00.

```
LDP <St1>, <St2>, [<Xn|SP>], #<imm>
```

64-bit variant

Applies when opc == 01.

```
LDP <Dt1>, <Dt2>, [<Xn|SP>], #<imm>
```

128-bit variant

Applies when opc == 10.

```
LDP <Qt1>, <Qt2>, [<Xn|SP>], #<imm>
```

Decode for all variants of this encoding

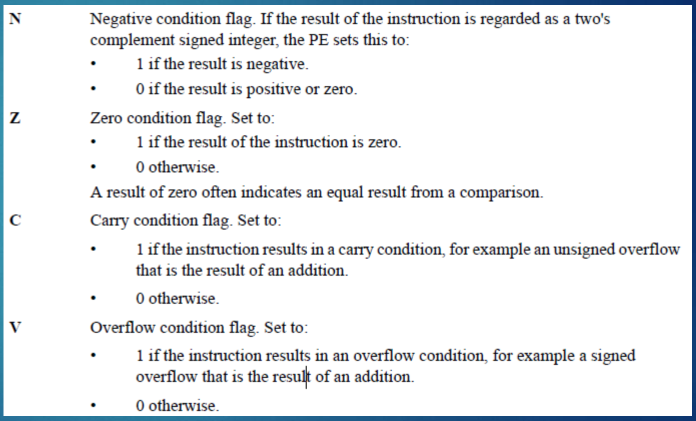
```
boolean wback = TRUE;
boolean postindex = TRUE;
```

联系

ARM	x86	典型用法
R0	EAX	函数返回值
R1-R5	EBX, ECX, EDX, ESI, EDI	General Purpose
R6-R10	–	
R11 (FP)	EBP	栈帧基地址
R12	–	Intra Procedural Call
R13 (SP)	ESP	栈指针
R14 (LR)	–	Link Register
R15 (PC)	EIP	程序指针
CPSR	EFLAGS	标志寄存器

PSR

- v8文档里称为PSTATE
- 相当于x86的标志寄存器RFLAGS
  - 条件标志位
  - 屏蔽标志位



N Negative condition flag. If the result of the instruction is regarded as a two's complement signed integer, the PE sets this to:
- 1 if the result is negative.
- 0 if the result is positive or zero.

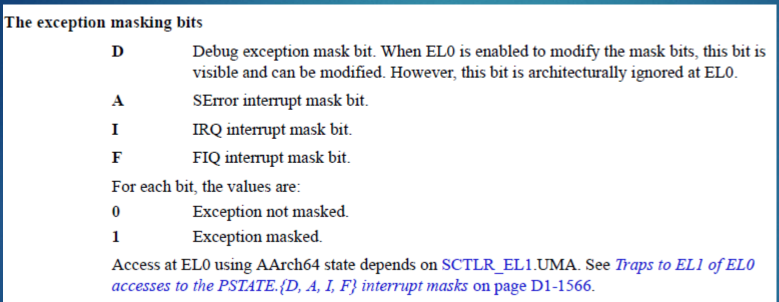
Z Zero condition flag. Set to:
- 1 if the result of the instruction is zero.
- 0 otherwise.

A result of zero often indicates an equal result from a comparison.

C Carry condition flag. Set to:
- 1 if the instruction results in a carry condition, for example an unsigned overflow that is the result of an addition.
- 0 otherwise.

V Overflow condition flag. Set to:
- 1 if the instruction results in an overflow condition, for example a signed overflow that is the result of an addition.
- 0 otherwise.

PSR的屏蔽标志位



The exception masking bits

D Debug exception mask bit. When EL0 is enabled to modify the mask bits, this bit is visible and can be modified. However, this bit is architecturally ignored at EL0.

A SError interrupt mask bit.

I IRQ interrupt mask bit.

F FIQ interrupt mask bit.

For each bit, the values are:

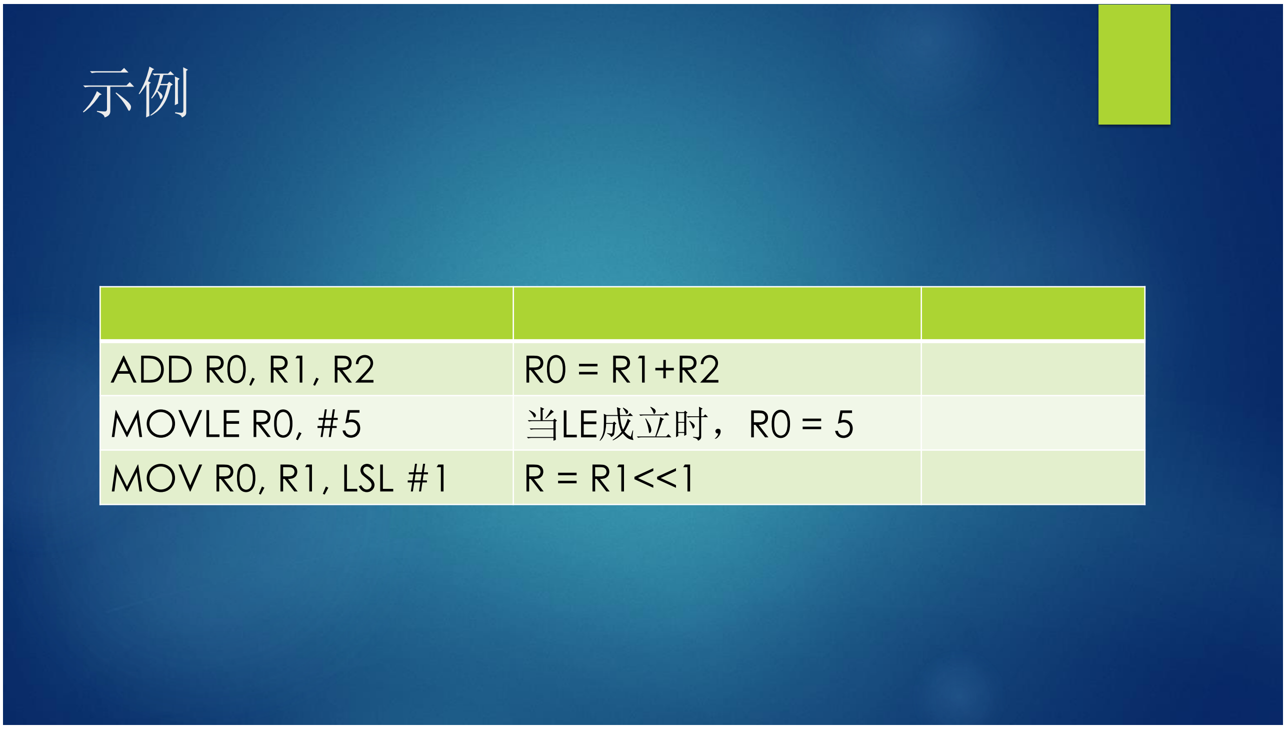
0 Exception not masked.

1 Exception masked.

Access at EL0 using AArch64 state depends on SCTLR_EL1.UMA. See *Traps to EL1 of EL0 accesses to the PSTATE.{D, A, I, F} interrupt masks* on page D1-1566.

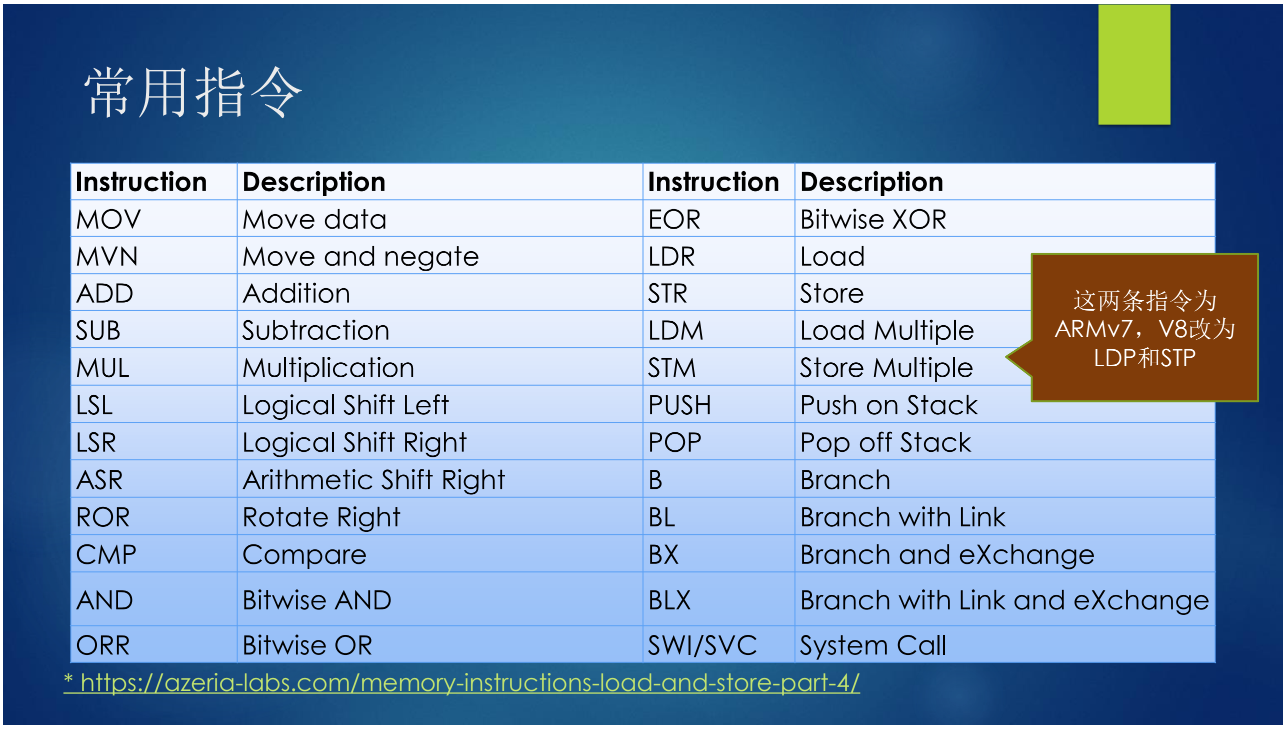
谓词执行

- Predicated Execution
- 消除分支的一种技术
- RISC中兴盛
- X86在引入了CMOV



示例

ADD R0, R1, R2	R0 = R1+R2	
MOVLE R0, #5	当LE成立时， R0 = 5	
MOV R0, R1, LSL #1	R = R1<<1	

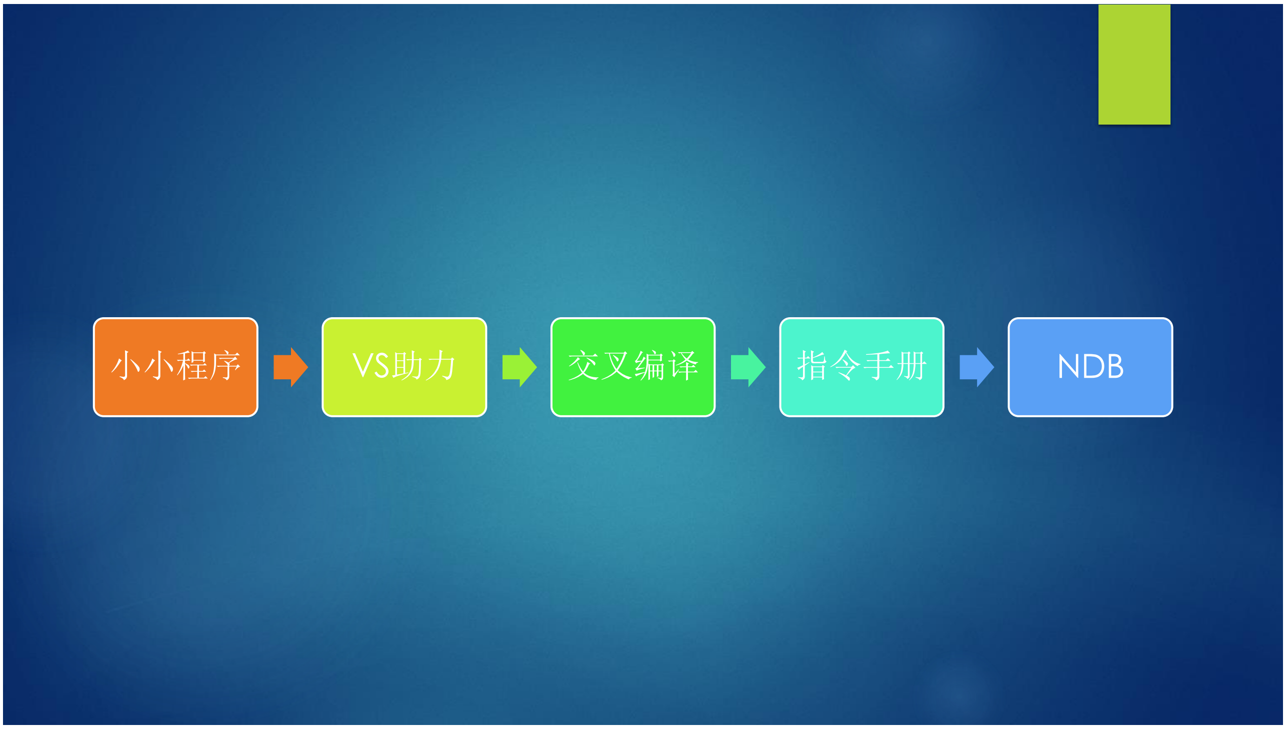


常用指令

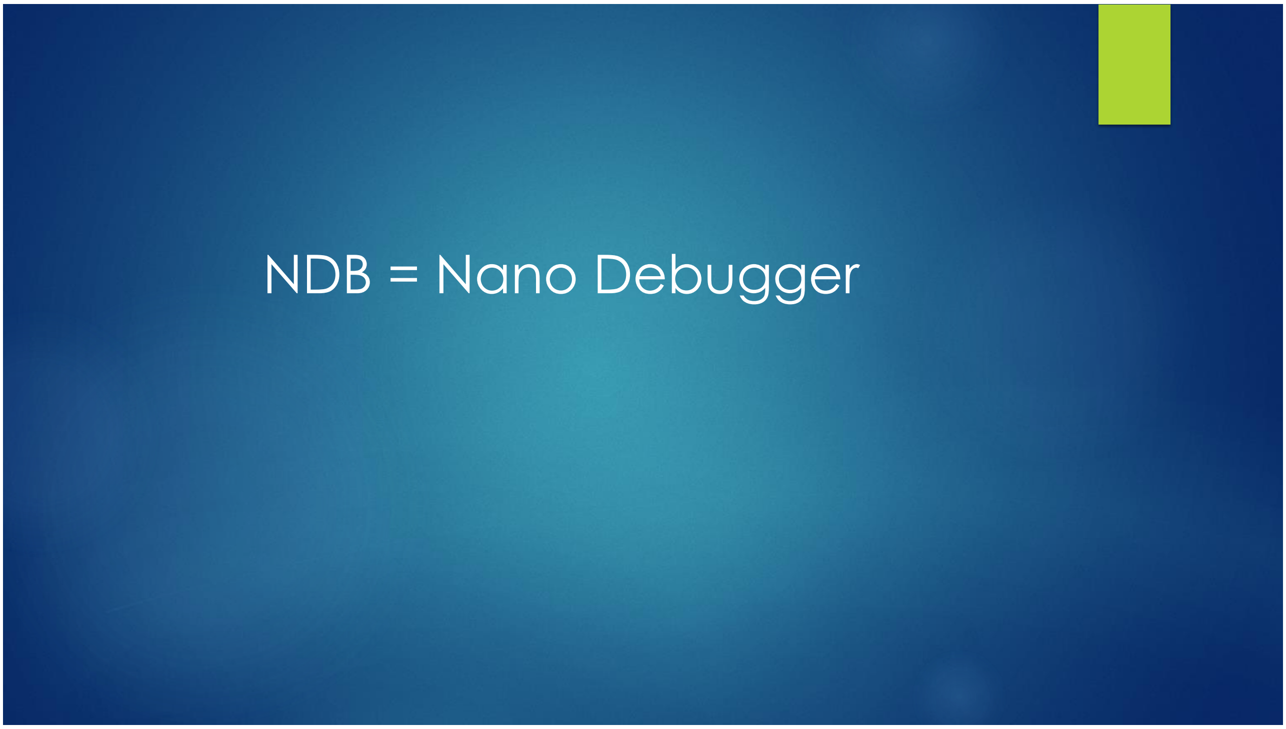
Instruction	Description	Instruction	Description
MOV	Move data	EOR	Bitwise XOR
MVN	Move and negate	LDR	Load
ADD	Addition	STR	Store
SUB	Subtraction	LDM	Load Multiple
MUL	Multiplication	STM	Store Multiple
LSL	Logical Shift Left	PUSH	Push on Stack
LSR	Logical Shift Right	POP	Pop off Stack
ASR	Arithmetic Shift Right	B	Branch
ROR	Rotate Right	BL	Branch with Link
CMP	Compare	BX	Branch and eXchange
AND	Bitwise AND	BLX	Branch with Link and eXchange
ORR	Bitwise OR	SWI/SVC	System Call

这两条指令为ARMv7，V8改为LDP和STP

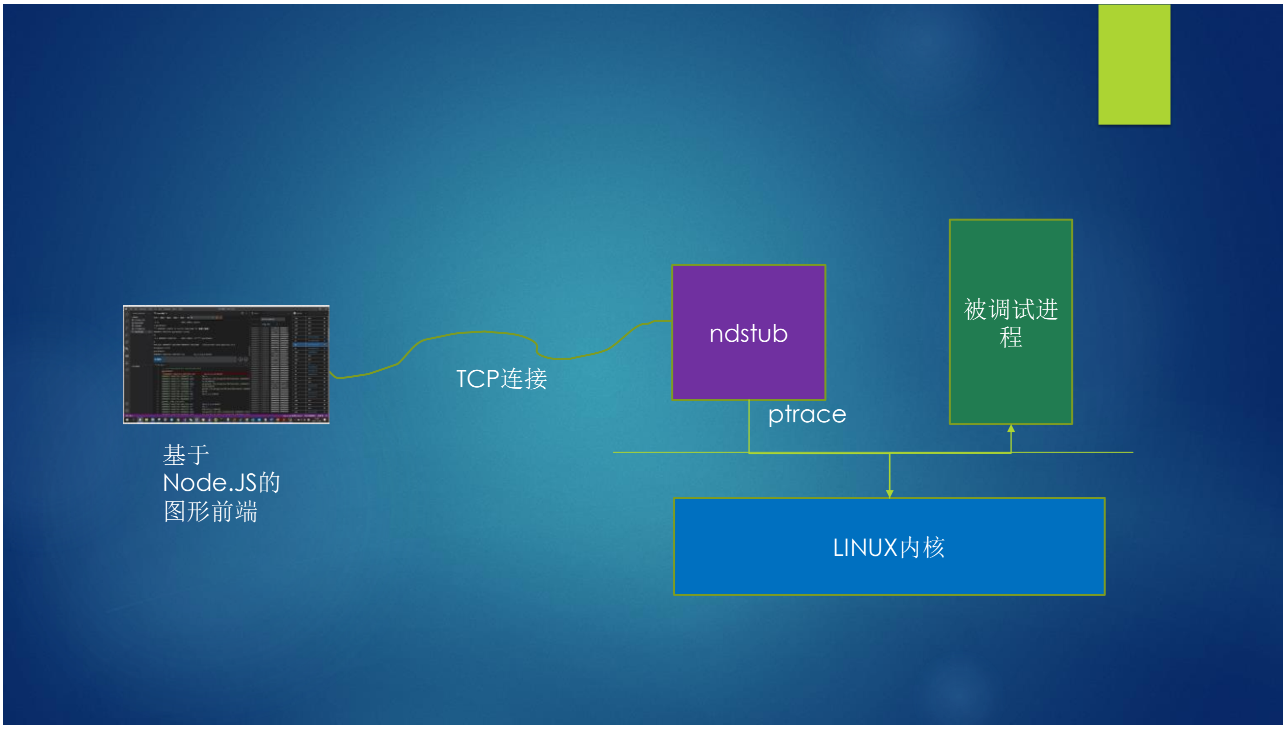
* https://azeria-labs.com/memory-instructions-load-and-store-part-4/

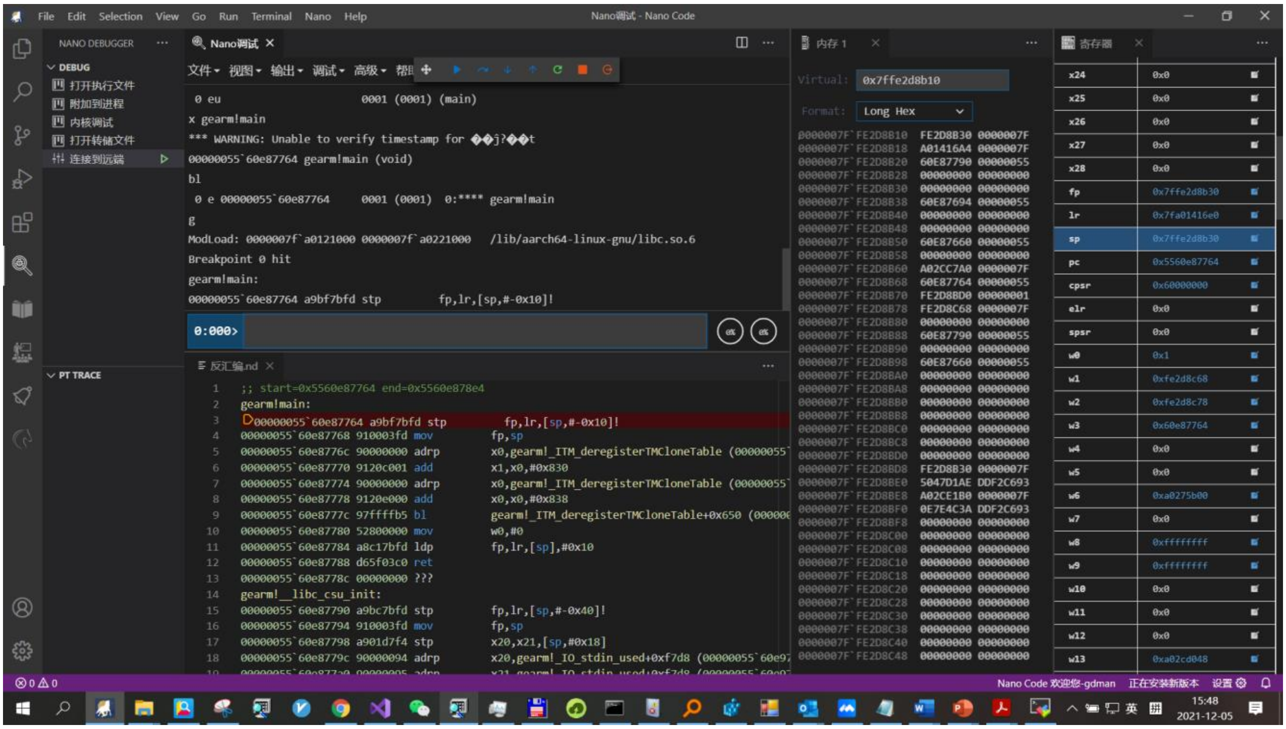


小小程序 → VS助力 → 交叉编译 → 指令手册 → NDB



NDB = Nano Debugger



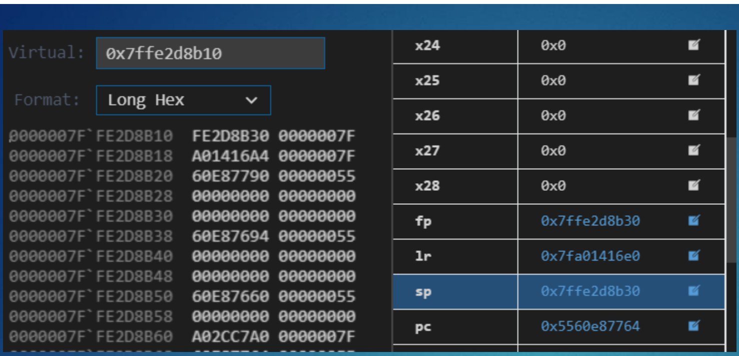


```
 0 eu                   0001 (0001) (main)
x gearm!main
*** WARNING: Unable to verify timestamp for ��j?��t
00000055`60e87764 gearm!main (void)
bl
 0 e 00000055`60e87764     0001 (0001)  0:**** gearm!main
g
ModLoad: 0000007f`a0121000 0000007f`a0221000   /lib/aarch64-linux-gnu/libc.so.6
Breakpoint 0 hit
gearm!main:
00000055`60e87764 a9bf7bfd stp         fp,lr,[sp,#-0x10]!
0:000>
```

```
;; start=0x5560e87764 end=0x5560e878e4
gearm!main:
00000055`60e87764 a9bf7bfd stp         fp,lr,[sp,#-0x10]!
00000055`60e87768 910003fd mov         fp,sp
00000055`60e8776c 90000000 adrp        x0,gearm!_ITM_deregisterTMCloneTable (00000055`
00000055`60e87770 9120c001 add         x1,x0,#0x830
00000055`60e87774 90000000 adrp        x0,gearm!_ITM_deregisterTMCloneTable (00000055`
00000055`60e87778 9120e000 add         x0,x0,#0x838
00000055`60e8777c 97ffffb5 bl          gearm!_ITM_deregisterTMCloneTable+0x650 (00000(
00000055`60e87780 52800000 mov         w0,#0
00000055`60e87784 a8c17bfd ldp         fp,lr,[sp],#0x10
00000055`60e87788 d65f03c0 ret
00000055`60e8778c 00000000 ???
gearm!__libc_csu_init:
00000055`60e87790 a9bc7bfd stp         fp,lr,[sp,#-0x40]!
00000055`60e87794 910003fd mov         fp,sp
00000055`60e87798 a901d7f4 stp         x20,x21,[sp,#0x18]
00000055`60e8779c 90000094 adrp        x20,gearm!_IO_stdin_used+0xf7d8 (00000055`60e97
00000055`60e877a0 90000095 adrp        x21,gearm!_IO_stdin_used+0xf7d8 (00000055`60e97
```

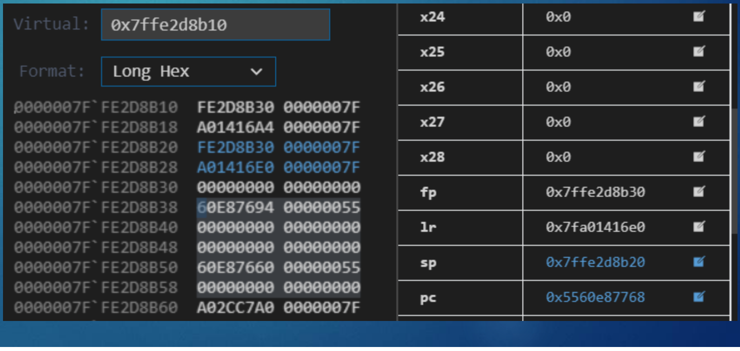
```
Virtual: 0x7ffe2d8b10
Format:  Long Hex
0000007F`FE2D8B10  FE2D8B30 0000007F
0000007F`FE2D8B18  A01416A4 0000007F
0000007F`FE2D8B20  60E87790 00000055
0000007F`FE2D8B28  00000000 00000000
0000007F`FE2D8B30  00000000 00000000
0000007F`FE2D8B38  60E87694 00000055
0000007F`FE2D8B40  00000000 00000000
0000007F`FE2D8B48  00000000 00000000
0000007F`FE2D8B50  60E87660 00000055
0000007F`FE2D8B58  00000000 00000000
0000007F`FE2D8B60  A02CC7A0 0000007F
0000007F`FE2D8B68  60E87764 00000055
0000007F`FE2D8B70  FE2D8BD0 00000001
0000007F`FE2D8B78  FE2D8C68 0000007F
0000007F`FE2D8B80  00000000 00000000
0000007F`FE2D8B88  60E87790 00000055
0000007F`FE2D8B90  00000000 00000000
0000007F`FE2D8B98  60E87660 00000055
0000007F`FE2D8BA0  00000000 00000000
0000007F`FE2D8BA8  00000000 00000000
0000007F`FE2D8BB0  00000000 00000000
0000007F`FE2D8BB8  00000000 00000000
0000007F`FE2D8BC0  00000000 00000000
0000007F`FE2D8BC8  00000000 00000000
0000007F`FE2D8BD0  00000000 00000000
0000007F`FE2D8BD8  FE2D8B30 0000007F
0000007F`FE2D8BE0  5047D1AE DDF2C693
0000007F`FE2D8BE8  A02CE1B0 0000007F
0000007F`FE2D8BF0  0E7E4C3A DDF2C693
0000007F`FE2D8BF8  00000000 00000000
0000007F`FE2D8C00  00000000 00000000
0000007F`FE2D8C08  00000000 00000000
0000007F`FE2D8C10  00000000 00000000
0000007F`FE2D8C18  00000000 00000000
0000007F`FE2D8C20  00000000 00000000
0000007F`FE2D8C28  00000000 00000000
0000007F`FE2D8C30  00000000 00000000
0000007F`FE2D8C38  00000000 00000000
0000007F`FE2D8C40  00000000 00000000
0000007F`FE2D8C48  00000000 00000000
```

x24	0x0
x25	0x0
x26	0x0
x27	0x0
x28	0x0
fp	0x7ffe2d8b30
lr	0x7fa01416e0
sp	0x7ffe2d8b30
pc	0x5560e87764
cpsr	0x60000000
elr	0x0
spsr	0x0
w0	0x1
w1	0xfe2d8c68
w2	0xfe2d8c78
w3	0x60e87764
w4	0x0
w5	0x0
w6	0xa0275b00
w7	0x0
w8	0xffffffff
w9	0xffffffff
w10	0x0
w11	0x0
w12	0x0
w13	0xa02cd048



```
Virtual: 0x7ffe2d8b10
Format: Long Hex

0000007F`FE2D8B10  FE2D8B30 0000007F
0000007F`FE2D8B18  A01416A4 0000007F
0000007F`FE2D8B20  60E87790 00000055
0000007F`FE2D8B28  00000000 00000000
0000007F`FE2D8B30  00000000 00000000
0000007F`FE2D8B38  60E87694 00000055
0000007F`FE2D8B40  00000000 00000000
0000007F`FE2D8B48  00000000 00000000
0000007F`FE2D8B50  60E87660 00000055
0000007F`FE2D8B58  00000000 00000000
0000007F`FE2D8B60  A02CC7A0 0000007F
```

x24	0x0
x25	0x0
x26	0x0
x27	0x0
x28	0x0
fp	0x7ffe2d8b30
lr	0x7fa01416e0
sp	0x7ffe2d8b30
pc	0x5560e87764



```
Virtual: 0x7ffe2d8b10
Format: Long Hex

0000007F`FE2D8B10  FE2D8B30 0000007F
0000007F`FE2D8B18  A01416A4 0000007F
0000007F`FE2D8B20  FE2D8B30 0000007F
0000007F`FE2D8B28  A01416E0 0000007F
0000007F`FE2D8B30  00000000 00000000
0000007F`FE2D8B38  60E87694 00000055
0000007F`FE2D8B40  00000000 00000000
0000007F`FE2D8B48  00000000 00000000
0000007F`FE2D8B50  60E87660 00000055
0000007F`FE2D8B58  00000000 00000000
0000007F`FE2D8B60  A02CC7A0 0000007F
```

x24	0x0
x25	0x0
x26	0x0
x27	0x0
x28	0x0
fp	0x7ffe2d8b30
lr	0x7fa01416e0
sp	0x7ffe2d8b20
pc	0x5560e87768

NDB调试gearm.out截图

NDB显示的A64寄存器

```
 x0=0000000000000001   x1=0000007fc27103b8   x2=0000007fc27103c8   x3=000000557b7c89a4
 x4=0000000000000000   x5=0000000000000000   x6=0000007fa8597b00   x7=0000000000000000
 x8=ffffffffffffffff   x9=000000003fffffff  x10=0000000020000000  x11=0000000000000000
x12=0000000000000000  x13=0000000000000000  x14=0000007fa861b1f8  x15=0000007fa861b150
x16=000000557b7f7d20  x17=0000007fa8463640  x18=0000000000000003  x19=000000557b7e02a8
x20=0000000000000000  x21=000000557b7c7130  x22=0000000000000000  x23=0000000000000000
x24=0000000000000000  x25=0000000000000000  x26=0000000000000000  x27=0000000000000000
x28=0000000000000000   fp=0000007fc2710280   lr=0000007fa8463720   sp=0000007fc2710280
 pc=000000557b7c89a4  psr=00000000 ---- EL0
ndstub!main:
00000055`7b7c89a4 a9bc7bfd stp         fp,lr,[sp,#-0x40]!
```

执行位置为Linux应用程序的main入口位置

切问而近思

欢迎关注格友公众号