

# 嵌入式技术基础

河北农业大学信息学院  
司永胜

[siyongsheng@foxmail.com](mailto:siyongsheng@foxmail.com)

# ARM公司简介

## 里程碑1 ----ARM成立

- **ARM**前身为艾康电脑(**Acorn**), 于**1978**年, 英国剑桥成立, 大学的孵化物。
- **1980**年代晚期, 苹果开始与艾康合作, 开发新版**ARM**核心。
- **1985**年, 艾康开发出全球第一款商用**RISC**处理器, 即**ARM1**, 针对于**PC**市场, 还没有嵌入式呢!!!
- **1990**年, 艾康财务危机, 受苹果和**VLSI**(最早做超大规模集成电路的公司)的投资, 成立独立子公司:**Advanced RISC Machines(ARM)**,**ARM**公司正式成立面世。

# ARM公司简介

## 里程碑2 -----嵌入式RISC处理器

- **1991年, ARM推出第一款嵌入式RISC处理器, 即ARM6。**
- **1993年,发布ARM7.**
- **1997年,发布ARM9TDMI,三星2440基于此内核。**
- **1999年,发布ARM9E,增强型ARM9**
- **2001年,ARMv6架构。**
- **2002年,发布ARM11微架构**

# ARM公司简介

## 里程碑3—Cortex系列

- **2004年**, 发布**ARMv7**架构的**Cortex**系列处理器, 同时推出**Cortex M3**。
- **2005年**, 发布**Cortex A8**处理器。
- **2007年**, 发布**Cortex M1**和**Cortex A9**
- **2009年**, 实现**Cortex A9**、发布**Cortex M0**
- **2010年**, 推出**Cortex-M4(F)**、成立**Linaro**(**ARM**公司牵头成立的公共组织, 专门做**ARM**处理器在**Linux**平台上的一些软件的开发和移植), 推出**Cortex-A15 MPcore**高性能处理器。

# ARM公司简介

## 里程碑4—64位处理器时代

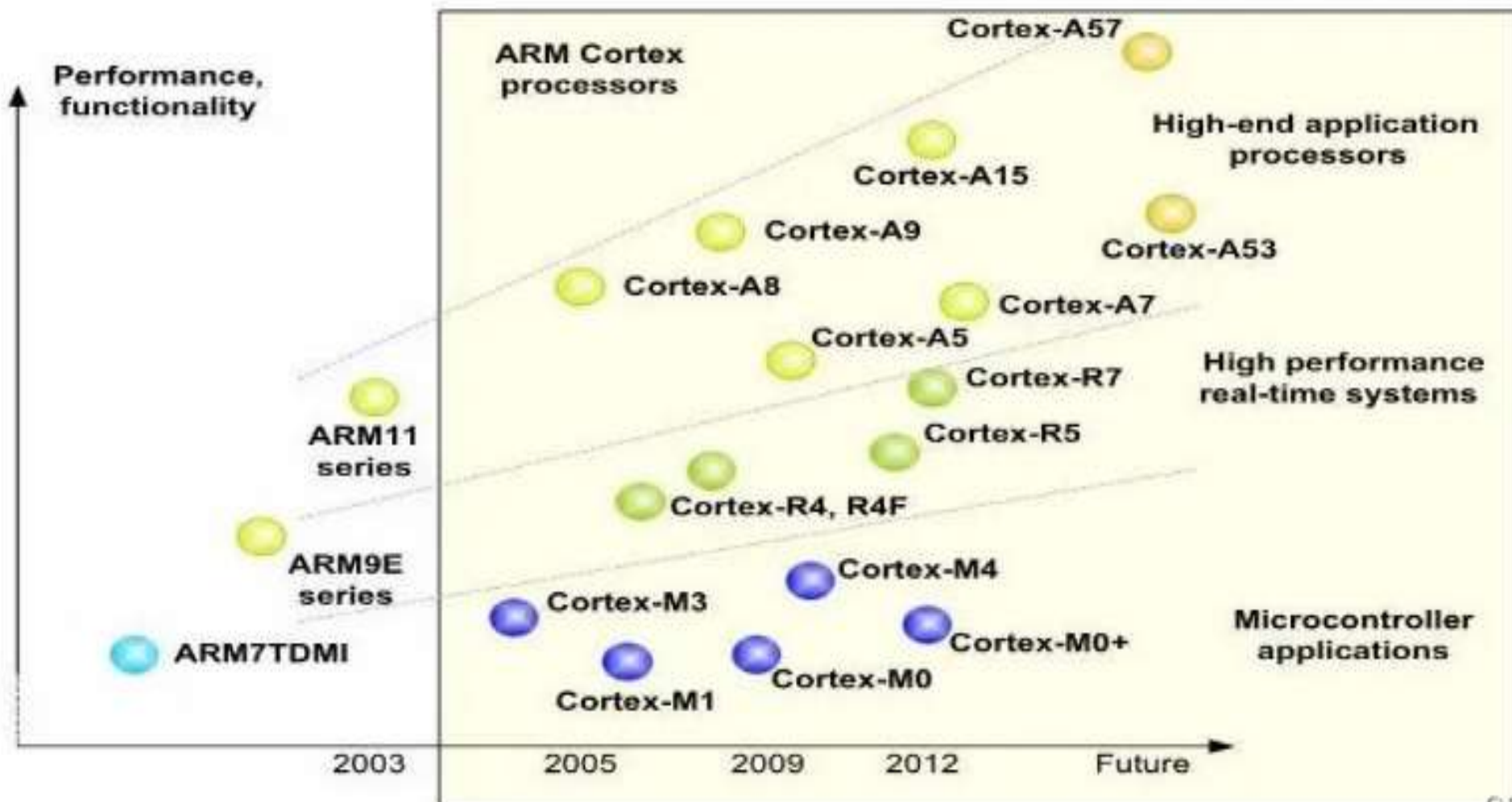
- **2011年**,推出**32位Cortex A7**处理器,**ARMv8**发布
- **2012年**,开始推出**64位**处理器。推出**Cortex M0+**、**ARM**首款**64位**处理器架构**Cortex A53**, **Cortex A57**架构。全球第一款**64位ARM**手机**iPhone5s**
- **2013年**,推出**32位Cortex A12**处理器架构
- **2014年**, 推出**Cortex-M7(F)**微控制器架构; **32位Cortex-A17**处理器架构
- **2015年**,推出**64位Cortex A35**、**Cortex A72**处理器架构。
- **2016年**, 推出**Cortex M23**、**Cortex-M33(F)**微控制器架构; **32位Cortex A32**处理器架构; **64位Cortex A73**处理器架构
- **2017年**,推出**64位Cortex A55**、**Cortex-A75**处理器架构。
- **2018年**, 推出微控制器**Cortex M35P**; **64位Cortex A76**处理器架构

# ARM公司简介

## ARM的商业模式：设计与生产分离

- **ARM**只负责设计**IC**，并且出卖自己的设计**IP**(只是产权)
- **ARM**自己不生产芯片，而是把设计**I**授权给其他半导体厂商来生产芯片。
- 另外也提供基于**ARM**架构的开发设计技术
- 软件工具，评估板，调试工具，应用软件，总线架构，外围设备单元，等等
- 大家知道华为的麒麟芯片吗？

# ARM系列的发展

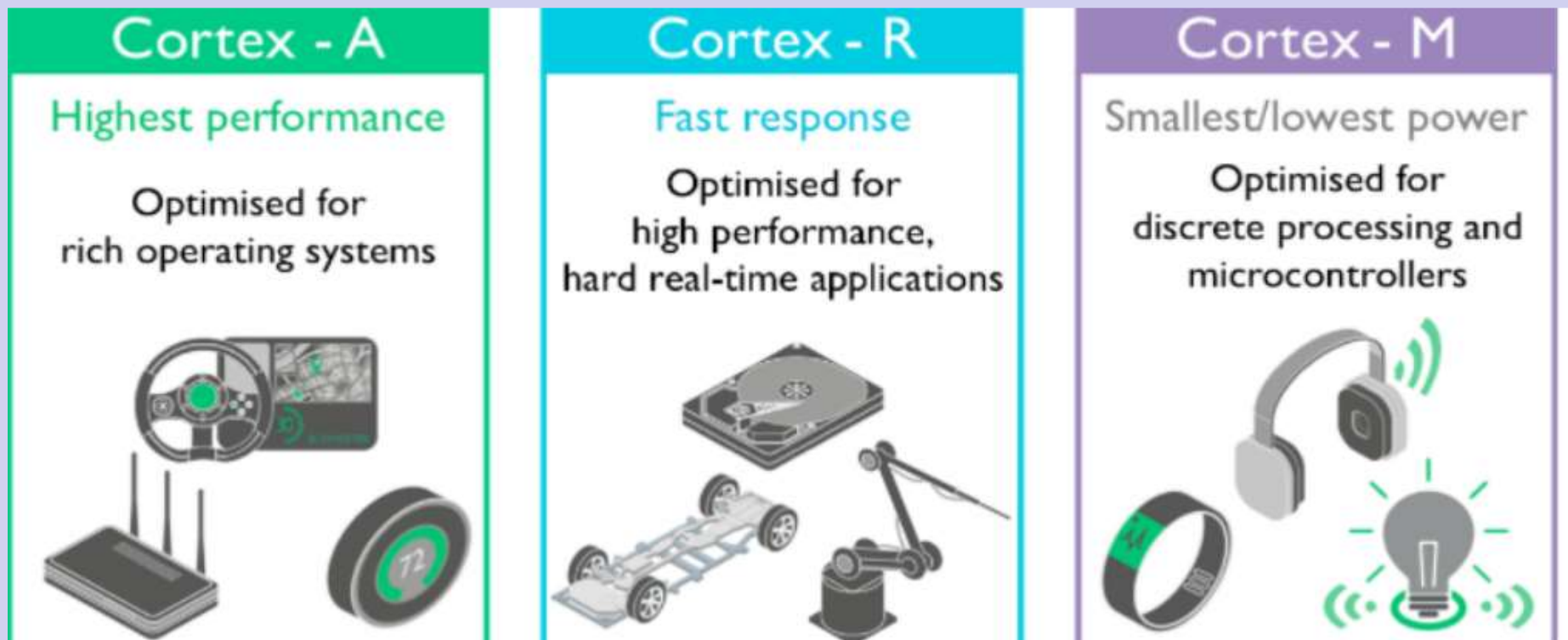


# ARM系列的发展

架构	处理器家族（代表）
ARMv1	ARM1
ARMv2	ARM2、ARM3等
ARMv3	ARM6、ARM7等
ARMv4	StrongARM、ARM7TDMI、ARM9TDMI等
ARMv5	ARM7EJ、ARM9E、ARM10E、Xscale等
ARMv6	ARM11、ARM Cortex-M等
ARMv7	ARM Cortex-A、ARM Cortex-M、ARM Cortex-R等
ARMv8	Cortex-A50、Cortex-A57、Cortex-A53等



# ARM系列的发展



**M**系列与**arm7**相似，不能跑操作系统（只能跑**ucos2\ucLinux**这些**RTOS**），偏向于控制方面,一般称为**MCU**。

**A**系列类似于**cpu**，与**arm9**和**arm11**相对应，面向尖端的基于虚拟内存的操作系统和用户应用。

**R**系列主要应用在对实时性要求高的场合。

# ARM系列的发展

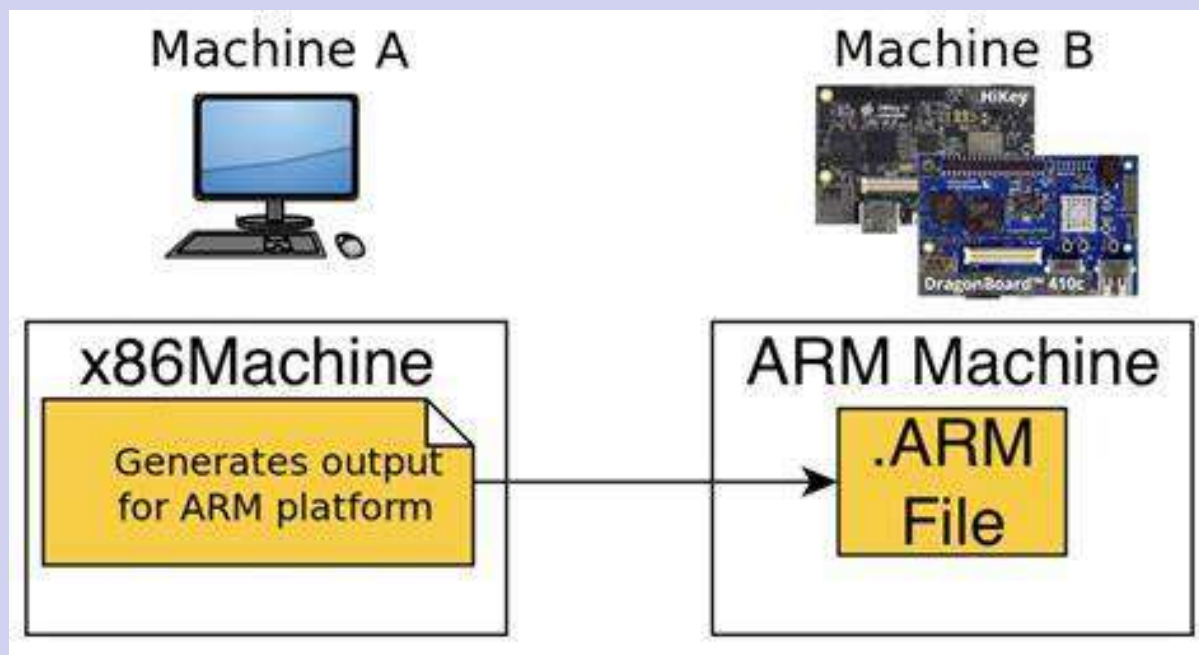
## 关于操作系统的一些说明

**M**系列和之前的**ARM7**没有**MMU**(内存管理单元)，一般称为**MCU**（微控制器），不能运行诸如**Linux**、**WinCE**等这些多用户多进程操作系统，因为运行这些系统需要**MMU**，才能给每个用户进程分配进程自己独立的地址空间。

**ucOS**、**ucLinux**这些精简实时的**RTOS**不需要**MMU**，可以在**ARM7**、**M**系列上运行。

**A**系列或之前的**ARM9**，带有**MMU**，可以运行诸如**Linux**等多用户多进程的操作系统。

# 交叉编译



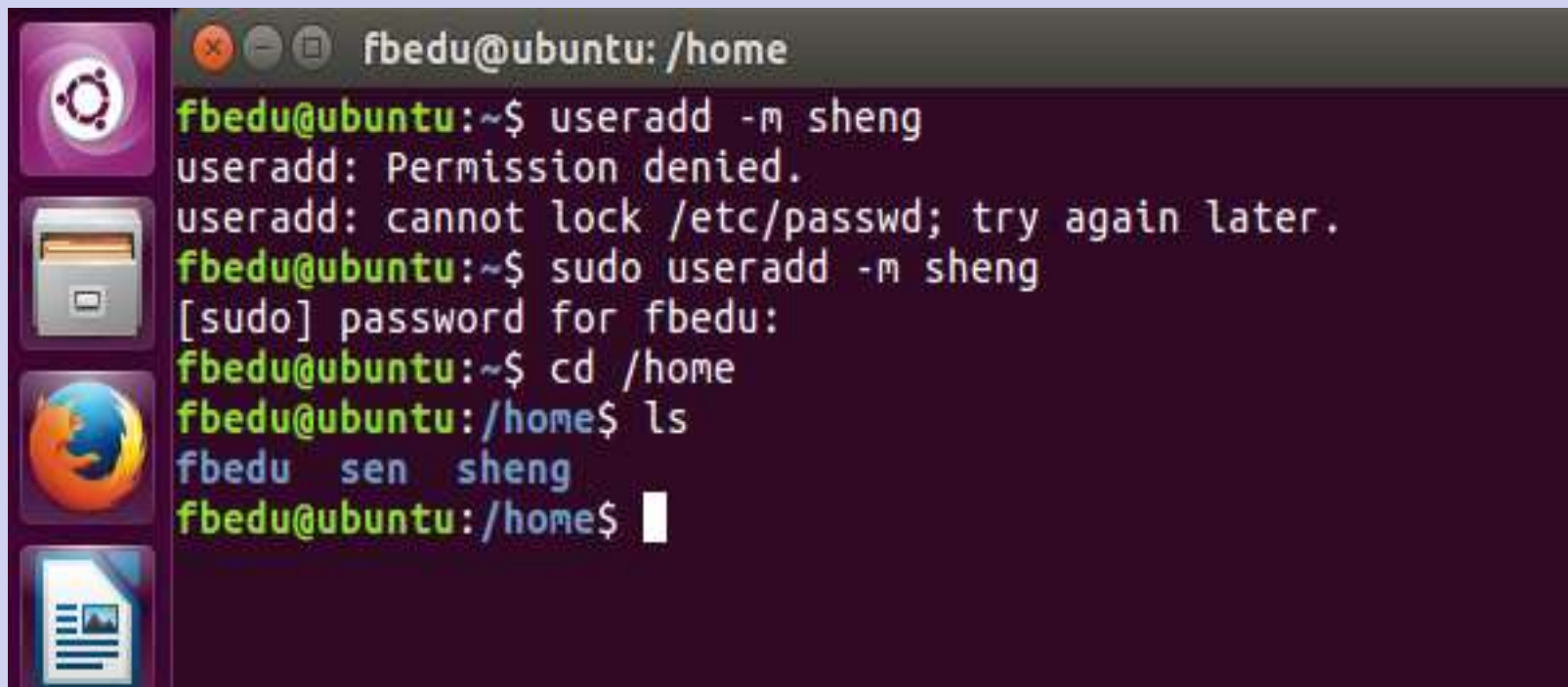
我们常用的计算机软件，都需要通过编译的方式把使用高级计算机语言编写的代码。在进行嵌入式系统的开发时，我们是在X86的体系结构的PC上进行编辑和编译代码，生成的代码却要在ARM的体系结构上运行，这一般说是不可行的，计算机的体系结构都不一样。但在交叉编译工具的帮助下，就可以实现，例如LinuxPC上的arm-linux-gcc编译器或ADS中的armcc编译器等。

# 基本Shell命令

- 目录信息查看命令 **ls**
- `ls -a` 显示目录所有文件及文件夹，包括隐藏文
- `ls -l` 显示文件的详细信息

# 基本Shell命令

## ■ 增加用户 **useradd**



```
fbedu@ubuntu: /home
fbedu@ubuntu:~$ useradd -m sheng
useradd: Permission denied.
useradd: cannot lock /etc/passwd; try again later.
fbedu@ubuntu:~$ sudo useradd -m sheng
[sudo] password for fbedu:
fbedu@ubuntu:~$ cd /home
fbedu@ubuntu:/home$ ls
fbedu  sen  sheng
fbedu@ubuntu:/home$
```

-m 自动建立用户的登入目录

# 基本Shell命令

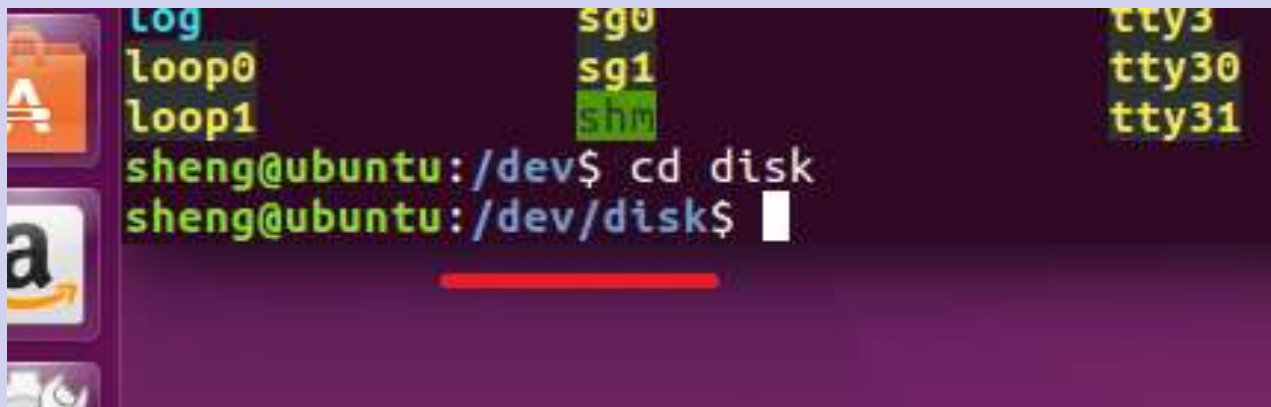
## ■ 设置用户密码passwd

```
fbedu@ubuntu:/home$ passwd sheng
passwd: You may not view or modify password information for sheng.
fbedu@ubuntu:/home$ su sheng
Password:

su: Authentication failure
fbedu@ubuntu:/home$
fbedu@ubuntu:/home$
fbedu@ubuntu:/home$ sudo passwd sheng
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
fbedu@ubuntu:/home$ su sheng
Password:
sheng@ubuntu:/home$
```

# 基本Shell命令

- 切换目录命令 **cd**
- `cd /` 切换到根目录
- `cd /home/sheng`



```
log          sg0          tty3
loop0        sg1          tty30
loop1        shm          tty31
sheng@ubuntu:/dev$ cd disk
sheng@ubuntu:/dev/disk$
```

A terminal window screenshot showing a user named 'sheng' at an 'ubuntu' machine. The prompt is 'sheng@ubuntu:/dev\$'. The user enters 'cd disk' and the prompt changes to 'sheng@ubuntu:/dev/disk\$'. The terminal has a dark purple background with yellow and green text. On the left, there are some application icons like a folder and a terminal icon. On the right, there are some system status indicators like 'log', 'sg0', 'tty3', 'loop0', 'sg1', 'tty30', 'loop1', 'shm', 'tty31'.

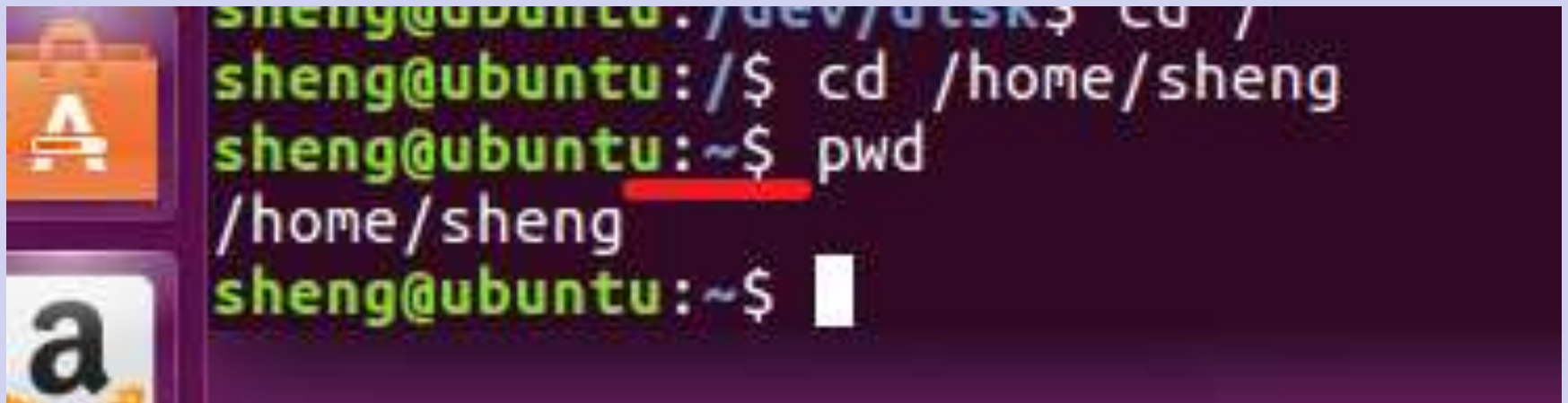
\$前的表示当前路径，但在当前用户目录下时候不显示



# 基本Shell命令

- 显示当前目录命令 **pwd**

例如：

A screenshot of a Linux terminal window. The prompt is 'sheng@ubuntu:~/dev/utsk\$'. The user enters 'cd /' and the prompt changes to 'sheng@ubuntu:/\$'. Then the user enters 'cd /home/sheng' and the prompt changes to 'sheng@ubuntu:~\$'. Finally, the user enters 'pwd' and the output is '/home/sheng'. The 'pwd' command and its output are highlighted with a red underline.

```
sheng@ubuntu:~/dev/utsk$ cd /  
sheng@ubuntu:/$ cd /home/sheng  
sheng@ubuntu:~$ pwd  
/home/sheng  
sheng@ubuntu:~$ █
```



# 基本Shell命令

- 显示当前操作系统和电脑信息 **uname**

全称是unix name

```
sheng@ubuntu:~$ uname
Linux
sheng@ubuntu:~$ uname -a
Linux ubuntu 4.15.0-29-generic #31~16.04.1-Ubuntu SMP Wed Jul 18 08:54:04 UTC 20
18 x86_64 x86_64 x86_64 GNU/Linux
```

# 基本Shell命令

- 清理屏幕命令 **clear**

假装清屏，鼠标往上滚动还是可以查看的

```
sheng@ubuntu:~$ uname
Linux
sheng@ubuntu:~$ uname -a
Linux ubuntu 4.15.0-29-generic #31~16.04.1-Ubuntu SMP Wed Jul 18 08:54:04 UTC 20
18 x86_64 x86_64 x86_64 GNU/Linux
```

# 基本Shell命令

- 查看文件内容命令 **cat**

```
sheng@ubuntu:~/linux/test$ ls
test.code-workspace  testprint  testprint.c
sheng@ubuntu:~/linux/test$ cat testprint.c
#include <stdio.h>
int main(int argc, char const *argv[])
{
    printf("Hello World!"); /* code */
    return 0;
}
sheng@ubuntu:~/linux/test$
```

# 基本Shell命令

## ■ 临时root权限sudo

```
sheng@ubuntu:~/linux/test$ cd /
sheng@ubuntu:/$ ls
bin      dev      initrd.img      lib64      mnt      root      snap      tmp      vmlinuz
boot     etc      initrd.img.old  lost+found  opt      run      srv      usr
cdrom    home    lib             media      proc     sbin     sys      var

sheng@ubuntu:/$ mkdir test
mkdir: cannot create directory 'test': Permission denied
sheng@ubuntu:/$ sudo mkdir test
[sudo] password for sheng:
sheng@ubuntu:/$ ls
bin      dev      initrd.img      lib64      mnt      root      snap      test      var
boot     etc      initrd.img.old  lost+found  opt      run      srv      tmp      vmlinuz
cdrom    home    lib             media      proc     sbin     sys      usr

sheng@ubuntu:/$ sudo rmdir test
sheng@ubuntu:/$ ls
bin      dev      initrd.img      lib64      mnt      root      snap      tmp      vmlinuz
boot     etc      initrd.img.old  lost+found  opt      run      srv      usr
cdrom    home    lib             media      proc     sbin     sys      var
```



# 基本Shell命令

创建和删除文件夹 **mkdir** **rmdir**

```
sheng@ubuntu:~/linux/test$ cd /
sheng@ubuntu:/$ ls
bin      dev      initrd.img      lib64      mnt      root      snap      tmp      vmlinuz
boot     etc      initrd.img.old  lost+found  opt      run      srv      usr
cdrom    home     lib             media      proc     sbin     sys      var

sheng@ubuntu:/$ mkdir test
mkdir: cannot create directory 'test': Permission denied
sheng@ubuntu:/$ sudo mkdir test
[sudo] password for sheng:
sheng@ubuntu:/$ ls
bin      dev      initrd.img      lib64      mnt      root      snap      test      var
boot     etc      initrd.img.old  lost+found  opt      run      srv      tmp      vmlinuz
cdrom    home     lib             media      proc     sbin     sys      usr

sheng@ubuntu:/$ sudo rmdir test
sheng@ubuntu:/$ ls
bin      dev      initrd.img      lib64      mnt      root      snap      tmp      vmlinuz
boot     etc      initrd.img.old  lost+found  opt      run      srv      usr
cdrom    home     lib             media      proc     sbin     sys      var
```

# 基本Shell命令

## 切换用户su

```
sheng@ubuntu:/$ su root
Password:
su: Authentication failure
sheng@ubuntu:/$ su root
Password:
su: Authentication failure
sheng@ubuntu:/$ su passwd root
No passwd entry for user 'passwd'
sheng@ubuntu:/$ sudo passwd root
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
sheng@ubuntu:/$ su root
Password:
root@ubuntu:/#
```

# 基本Shell命令

## 网络信息显示ifconfig

```
sheng@ubuntu:~$ ifconfig
ens33      Link encap:Ethernet  HWaddr 00:0c:29:a7:fa:2e
            inet addr:192.168.50.105  Bcast:192.168.50.255  Mask:255.255.255.0
            inet6 addr: fe80::89a:bfb0:2728:9f2c/64 Scope:Link
            UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
            RX packets:62 errors:0 dropped:0 overruns:0 frame:0
            TX packets:102 errors:0 dropped:0 overruns:0 carrier:0
            collisions:0 txqueuelen:1000
            RX bytes:23925 (23.9 KB)  TX bytes:12230 (12.2 KB)

lo         Link encap:Local Loopback
            inet addr:127.0.0.1  Mask:255.0.0.0
            inet6 addr: ::1/128 Scope:Host
            UP LOOPBACK RUNNING  MTU:65536  Metric:1
            RX packets:256 errors:0 dropped:0 overruns:0 frame:0
            TX packets:256 errors:0 dropped:0 overruns:0 carrier:0
            collisions:0 txqueuelen:1000
            RX bytes:20149 (20.1 KB)  TX bytes:20149 (20.1 KB)
```

# 基本Shell命令

系统重启命令 **reboot**

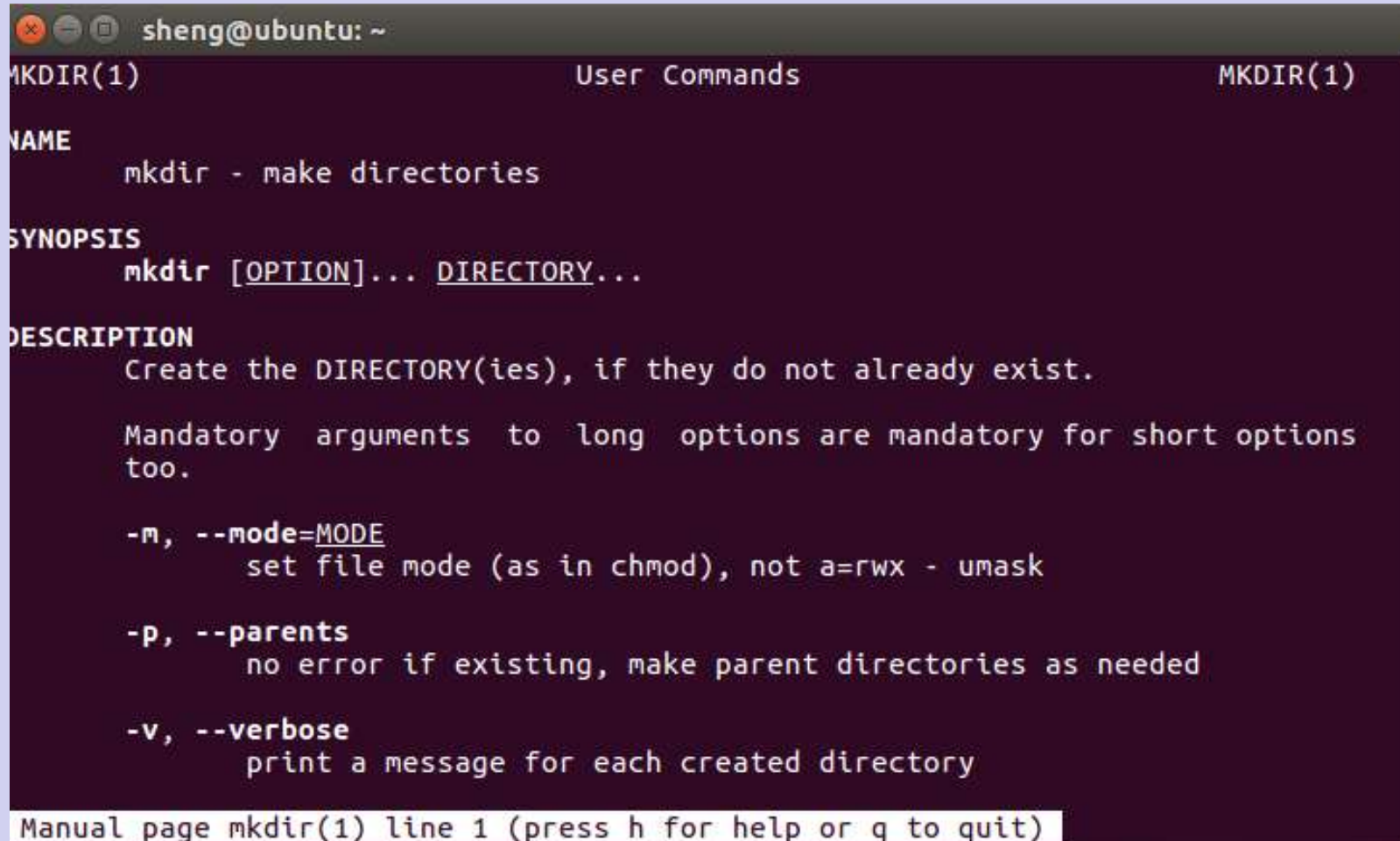
关机命令 **poweroff**



# 基本Shell命令

## 系统帮助命令man

```
sheng@ubuntu: ~$ man mkdir
```



```
sheng@ubuntu: ~
MKDIR(1)                                User Commands                                MKDIR(1)

NAME
    mkdir - make directories

SYNOPSIS
    mkdir [OPTION]... DIRECTORY...

DESCRIPTION
    Create the DIRECTORY(ies), if they do not already exist.

    Mandatory arguments to long options are mandatory for short options
    too.

    -m, --mode=MODE
        set file mode (as in chmod), not a=rwx - umask

    -p, --parents
        no error if existing, make parent directories as needed

    -v, --verbose
        print a message for each created directory

Manual page mkdir(1) line 1 (press h for help or q to quit)
```

# 基本Shell命令

## 查找文件命令find

```
sheng@ubuntu:~$ find -name testprint.c  
./linux/test/testprint.c  
sheng@ubuntu:~$
```

./的含义是当前文件夹

# 基本Shell命令

## 内容查找命令grep

```
sheng@ubuntu:~$ grep -nr "hello" /
```

```
sheng@ubuntu: ~
grep: /etc/cups/subscriptions.conf: Permission denied
grep: /etc/cups/subscriptions.conf.0: Permission denied
grep: /etc/cups/ssl: Permission denied
grep: /etc/.pwd.lock: Permission denied
grep: /etc/ssl/private: Permission denied
/usr/src/linux-headers-4.15.0-29/tools/build/Makefile.feature:77:      hello
\
/usr/src/linux-headers-4.15.0-29/tools/build/feature/Makefile:13:      test-h
ello.bin
\
/usr/src/linux-headers-4.15.0-29/tools/build/feature/Makefile:76:$(OUTPUT)test-h
ello.bin:
/usr/src/linux-headers-4.15.0-29/samples/kdb/Makefile:1:obj-$(CONFIG_SAMPLE_KDB)
+= kdb_hello.o
/usr/src/linux-headers-4.15.0-29/samples/Kconfig:62:      Build an example of ho
w to dynamically add the hello
/usr/src/linux-headers-4.15.0-29/include/net/dn_dev.h:56: * t2 - Rate limit time
r, min time between routing and hello messages
/usr/src/linux-headers-4.15.0-29/include/net/dn_dev.h:57: * t3 - Hello timer, se
nd hello messages when it expires
/usr/src/linux-headers-4.15.0-29/include/net/dn_dev.h:124:struct endnode_hello_m
essage {
/usr/src/linux-headers-4.15.0-29/include/net/dn_dev.h:139:struct rtnode_hello_me
ssage {
/usr/src/linux-headers-4.15.0-29/include/net/dn_dev.h:162:void dn_dev_hello(stru
```

参数种类很多 CTRL+C退出

# 基本Shell命令

文件夹大小查看命令 **du**

-s 仅显示总计，-h 以人类可读的方式

```
sheng@ubuntu:~$ ls
Desktop  Documents  Downloads  examples.desktop  linux  Music
sheng@ubuntu:~$ du -sh linux
92K      linux
```

磁盘空间检查命令 **df**

直接输入df即可

# 基本Shell命令

打开某个文件命令 **gedit**

# 基本Shell命令

文件类型查看命令 **file**

# 基本Shell命令

## 课堂演示

- 打开终端窗口, 使用 **pwd** 命令查看自己所在目录. **Pwd**
- 创建自己的账号, 并在 **/home** 下, 自动生成自己账号的文件夹 **useradd -m zhang**
- 给自己的账号设置密码 **Passwd zhang**
- 使用 **mkdir** 命令在自己的文件夹下创建 **hello** 目录
- 在自己文件夹下创建 **a.txt**、**b.txt**--- **touch a.txt**, 并用 **vi** 打开 **a.txt** 和 **b.txt**, 并在两个文件中分别输入 **Hello** 和 **World!** 并存盘退出。
- 将上述两个文件合并成一个文件 **ab.txt**-----**cat a.txt b.txt > ab.txt**
- 将 **ab.txt** 文件剪切到 **hello** 目录下。
- 用 **cat** 命令查看 **ab.txt** 文件内容
- 使用 **cd** 命令通过相对路径切换到根(**/**)目录
- 使用 **cd -** 切换到上次所在的目录
- 用 **ls** 命令以列表的形式查看当前目录下所有的文件(包括隐藏文件)
- 使用 **cd** 命令通过绝对路径方式切换到桌面目录下 **cd /home/zhen/Desktop**
- 使用 **rm** 命令删除 **ab.txt** 文件, 然后删除 **hello** 文件夹



# GCC编译器

- 在为Linux开发应用程序时，绝大多数情况下使用的都是C语言，因此几乎每一位Linux程序员面临的首要问题都是如何灵活运用C编译器。目前Linux下最常用的C语言编译器是GCC（GNU Compiler Collection），它是GNU项目中符合ANSI C标准的编译系统，能够编译用C、C++和Object C等语言编写的程序。GCC不仅功能非常强大，结构也异常灵活。
- Linux系统下的GCC（GNU C Compiler）是GNU推出的功能强大、性能优越的多平台编译器，是GNU的代表作品之一。gcc是可以在多种硬件平台上编译出可执行程序的超级编译器，其执行效率与一般的编译器相比平均效率要高20%~30%。



# GCC编译器

- 第一次编译
- 在学习使用GCC之前，下面的这个例子能够帮助用户迅速理解GCC的工作原理

```
#include<stdio.h>
```

```
int main(){
```

```
    printf("hello world!\n ");
```

```
    return 0;
```

```
}
```

## 终端演示

- 上面在编译的时候，为gcc的后面加入了选项-o进行新文件的重命名，如果不加入这个选项，那么新文件就会默认为a.out，如果再次编译其他的文件，同样不进行重命名的话，那么这里的a.out将会被覆盖掉。

# GCC编译器

- 第一次编译
- 在学习使用GCC之前，下面的这个例子能够帮助用户迅速理解GCC的工作原理

```
#include<stdio.h>
```

```
int main(){
```

```
    printf("hello world!\n ");
```

```
    return 0;
```

```
}
```

## 终端演示

- 上面在编译的时候，为gcc的后面加入了选项-o进行新文件的重命名，如果不加入这个选项，那么新文件就会默认为a.out，如果再次编译其他的文件，同样不进行重命名的话，那么这里的a.out将会被覆盖掉。

# GCC编译器

在使用**Gcc**编译器的时候，必须给出一系列必要的调用参数和文件名称。**Gcc**编译器的调用参数大约有**100**多个，其中多数参数根本就用不到，只介绍其中最常用的参数。

**GCC最基本的用法是：**

**gcc [options] [filenames]**

其中**options**就是编译器所需要的参数，**filenames**给出相关的文件名称。

- **-c**，只编译，不连接成为可执行文件，编译器只是由输入的**.c**等源代码文件生成**.o**为后缀的目标文件，通常用于编译不包含主程序的子程序文件。
- **-o output\_filename**，确定输出文件的名称为**output\_filename**，同时这个名称不能和源文件同名。如果不给出这个选项，**gcc**就给出预设的可执行文件**a.out**。
- **-g**，产生符号调试工具(**GNU**的**gdb**)所必要的符号资讯，要想对源代码进行调试，我们就必须加入这个选项。
- **-O**，对程序进行优化编译、连接，采用这个选项，整个源代码会在编译、连接过程中进行优化处理，这样产生的可执行文件的执行效率可以提高，但是，编译、连接的速度就相应地要慢一些。
- **-O2**，比**-O**更好的优化编译、连接，当然整个编译、连接过程会更慢。

# GCC编译器

在使用**Gcc**编译器的时候，必须给出一系列必要的调用参数和文件名称。**Gcc**编译器的调用参数大约有**100**多个，其中多数参数根本就用不到，只介绍其中最常用的参数。

**GCC最基本的用法是：**

**gcc [options] [filenames]**

其中**options**就是编译器所需要的参数，**filenames**给出相关的文件名称。

- **-c**，只编译，不连接成为可执行文件，编译器只是由输入的**.c**等源代码文件生成**.o**为后缀的目标文件，通常用于编译不包含主程序的子程序文件。
- **-o output\_filename**，确定输出文件的名称为**output\_filename**，同时这个名称不能和源文件同名。如果不给出这个选项，**gcc**就给出预设的可执行文件**a.out**。
- **-g**，产生符号调试工具(**GNU**的**gdb**)所必要的符号资讯，要想对源代码进行调试，我们就必须加入这个选项。
- **-O**，对程序进行优化编译、连接，采用这个选项，整个源代码会在编译、连接过程中进行优化处理，这样产生的可执行文件的执行效率可以提高，但是，编译、连接的速度就相应地要慢一些。
- **-O2**，比**-O**更好的优化编译、连接，当然整个编译、连接过程会更慢。

# GCC编译器

```
#include<stdio.h>
void main(){
    printf("hello world!\n ");
    long var=1;
    return 0;
}
```

编译测试

再用gcc helloworld.c -Wall -o helloworld 编译  
-Wall 尽可能多的产生警告信息

# VI或VIM

基本上vim可以分为三种模式，分别是一般模式、编辑模式和底行模式，这三种模式具体如下：

## 命令模式

您一进入vim 就是处于一般模式，该模式下只能输入指令，不能输入文字。這些指令可能是让光标移动的指令，也可能是删除指令或取代指令。

## 编辑模式

按i 就会进入编辑模式（插入模式），此时在状态列会有 **INSERT** 字樣。在该模式下才可以输入文字，按**Esc** 又会回到命令模式。

## 底行模式

按冒号“:” 就会进入底行模式，此时左下角会有一个冒号，等待输入命令。按**Esc**返回命令模式。

# VI或VIM

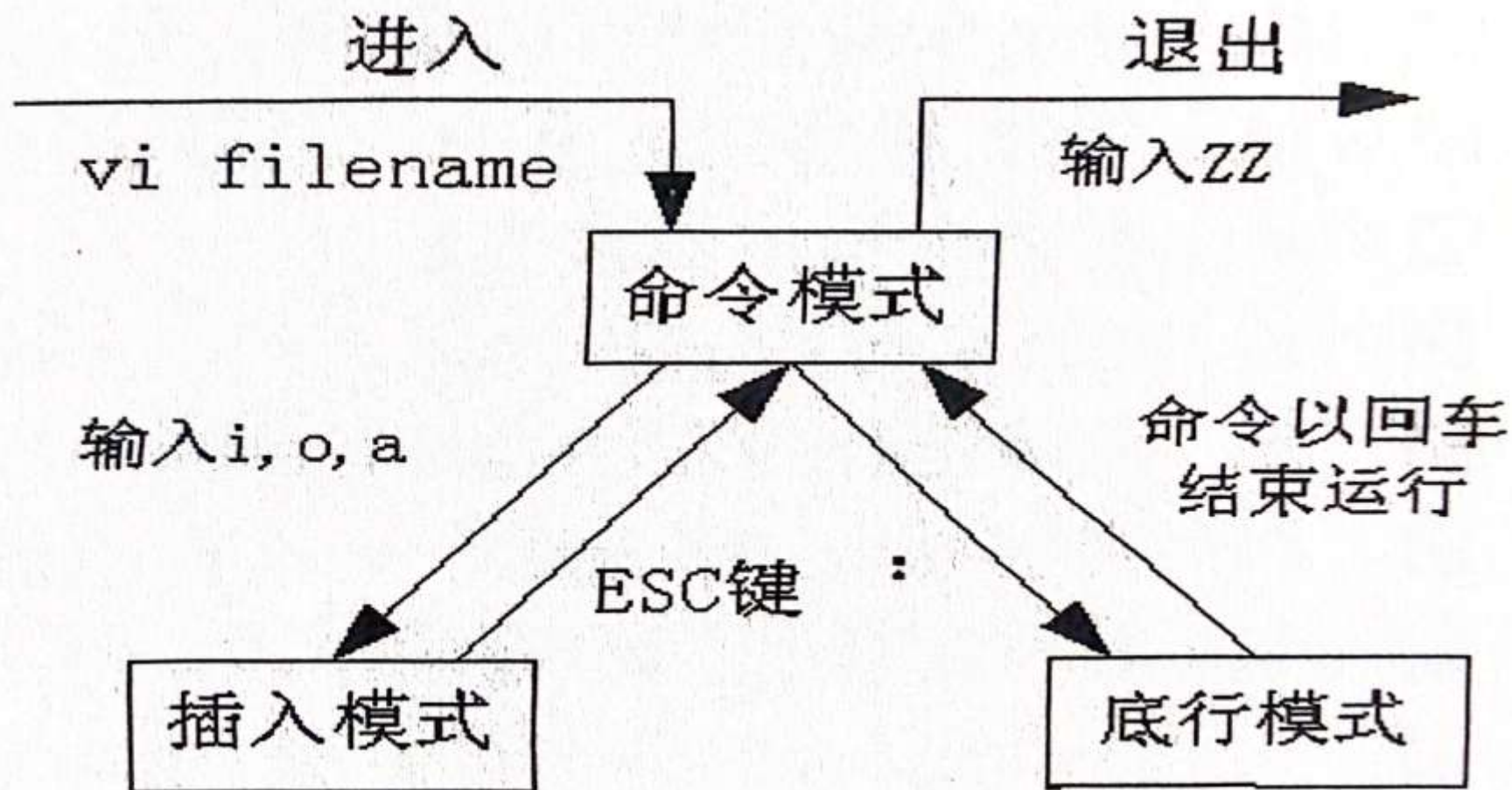


图 4.3 3 种模式的相互转换示意图

# GCC编译器

```
#include<stdio.h>
void main(){
    printf("hello world!\n ");
    long var=1;
    return 0;
}
```

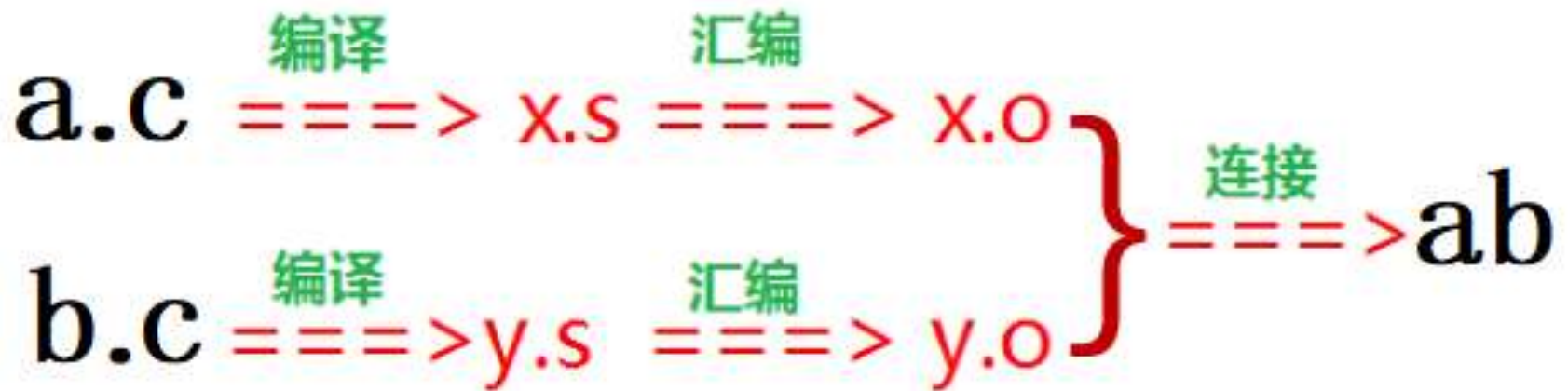
编译测试

再用gcc helloworld.c -Wall -o helloworld 编译  
-Wall 尽可能多的产生警告信息



# Makefile

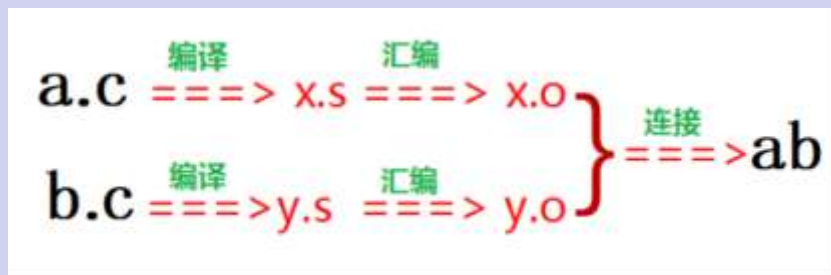
```
sheng@ubuntu:~$ gcc a.c b.c -o ab
sheng@ubuntu:~$ ls
ab  b.c  Documents  examples.desktop  helloworld.c  Pictures  Templates
a.c  Desktop  Downloads  helloworld        Music         Public    Videos
sheng@ubuntu:~$ ./ab
I am function_bsheng@ubuntu:~$
```



# Makefile

对于命令 `gcc a.c b.c -o ab`，假如我们修改了 `a.c` 文件的内容，那么需要重新汇编生成 `x.o` 文件，虽然没有修改 `B.c` 文件，此时 `gcc` 还是会默认再生成一次 `y.o` 文件，然后把 `x.o` 和 `y.o` 文件进行链接。

最好是如果 `y.o` 没有修改就不需要去重新生成一次了，直接使用旧的 `y.o` 文件。如果工程中文件非常多，很费时间。



# Makefile

**-c** 只激活预处理,编译,和汇编;生成 .o 的 obj 文件

```
sheng@ubuntu:~$ gcc a.c -c -o x.o  
sheng@ubuntu:~$ gcc b.c -c -o y.o  
sheng@ubuntu:~$ gcc x.o y.o -o ab
```

如果只修改了 **a.c** 的话，那我们只需要去重新生成 **x.o** 文件即可，然后再和旧的 **y.o** 重新连接在一起生成新的 **out** 就可以，第二条语句没有必要执行。

# Makefile

那如何判断文件是否被修改呢？通过判断文件的修改时间,这里不详细解释。

太麻烦了，好在有**makefile**

# Makefile

## Makefile的基本规则

```
1 target: prerequisites
2     command
3     ...
4     ...
5 =====
6 目标 :   依赖文件
7 [tab键] 命令
8     ...
9     ...
```

**target:**是目标文件，也可以是执行文件。

**prerequisites:** 就是要生成那个**target**所需要的文件或者目标。

**command:** 也就是**make**需要执行的命令。

# Makefile

## Makefile的基本规则

```
1 target: prerequisites
2     command
3     ...
4     ...
5 =====
6 目标 : 依赖文件
7 [tab键] 命令
8     ...
9     ...
```

- 当 prerequisites 比 target 新 =====> 执行 command

**target**是由一个或多个目标文件依赖于**prerequisites**中的文件，其生成规则定义在**command**中，而且只要**prerequisites**中有一个以上的文件比**target**文件更新的话，**command**所定义的命令就会被执行，这是**makefile**的最基本规则



# Makefile

```
1 ab: x.o y.o
2      gcc x.o y.o -o ab
3 x.o: a.c
4      gcc a.c -c -o x.o
5 y.o: b.c
6      gcc b.c -c -o y.o
7
```

红色框是**Tab**键

红色框是**Tab**键

# Makefile

```
sheng@ubuntu:~$ touch Makefile
sheng@ubuntu:~$ vi Makefile
sheng@ubuntu:~$ ls
ab    a.o  Desktop  Downloads  helloworld  Makefile
a.c  b.c  Documents examples.desktop helloworld.c Music
sheng@ubuntu:~$ make
make: 'ab' is up to date.
sheng@ubuntu:~$ vi Makefile
sheng@ubuntu:~$ vi a.c
sheng@ubuntu:~$ make
gcc a.c -c -o x.o
gcc x.o y.o -o ab
sheng@ubuntu:~$ ./ab
I am function_
new line!
sheng@ubuntu:~$
```

# Makefile

```
sheng@ubuntu:~$ vi Makefile
sheng@ubuntu:~$ vi a.c
sheng@ubuntu:~$ make
gcc a.c -c -o x.o
gcc x.o y.o -o ab
sheng@ubuntu:~$ ./ab
I am function_bnew line!
sheng@ubuntu:~$ vi a.c
sheng@ubuntu:~$ make
gcc a.c -c -o x.o
gcc x.o y.o -o ab
sheng@ubuntu:~$ ./ab
I am function_bnew line!
sheng@ubuntu:~$ vi b.c
sheng@ubuntu:~$ make
gcc b.c -c -o y.o
gcc x.o y.o -o ab
sheng@ubuntu:~$ ./ab
I am function_b
new line!
sheng@ubuntu:~$
```

# Makefile

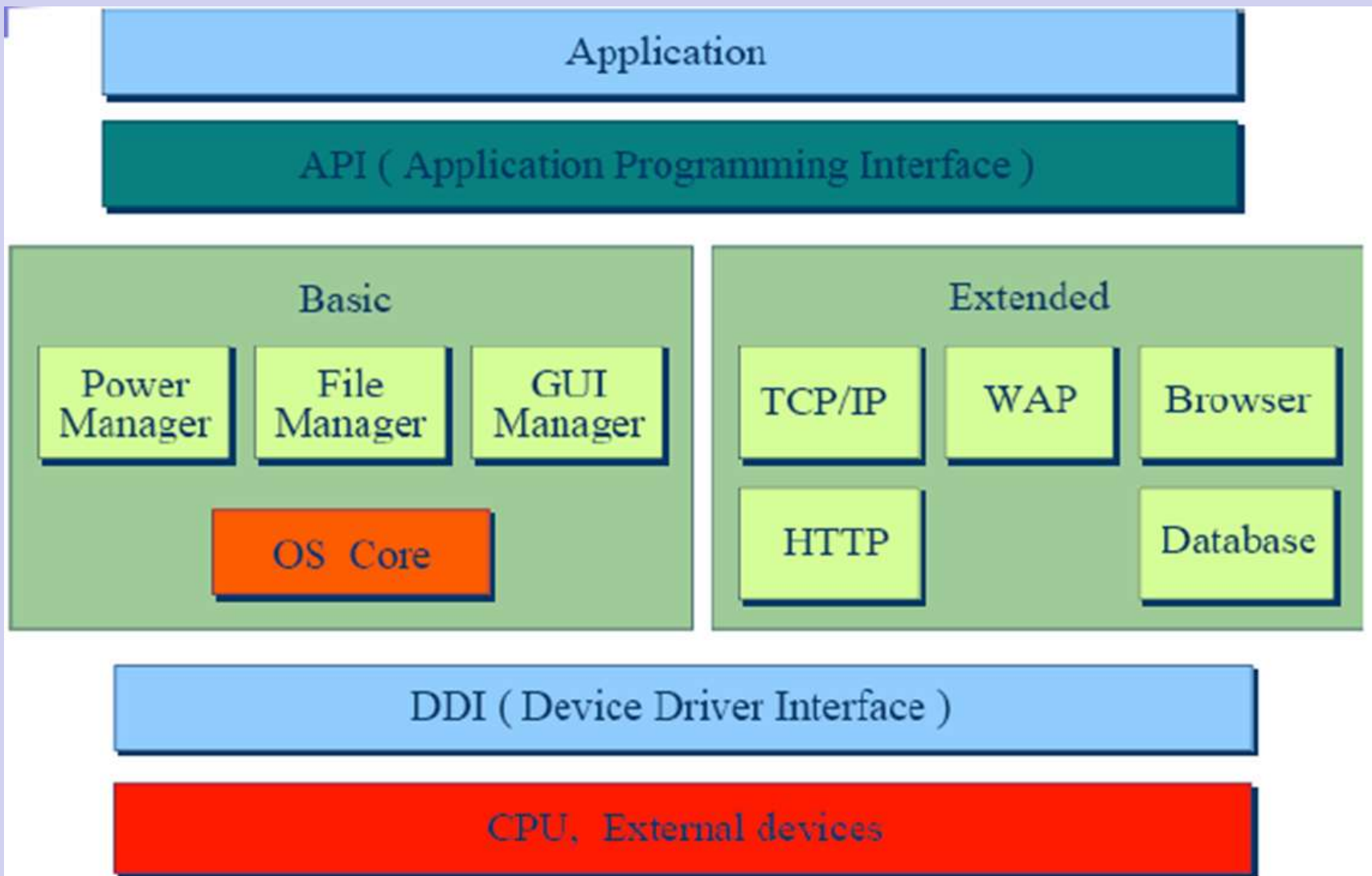
```
1 #include<stdio.h>
2
3 void function_b();
4
5 int main()
6 {
7     function_b();
8     return 0;
9 }
```

a.c

```
1 #include<stdio.h>
2
3 void function_b(void)
4 {
5     printf("I am function_b");
6 }
```

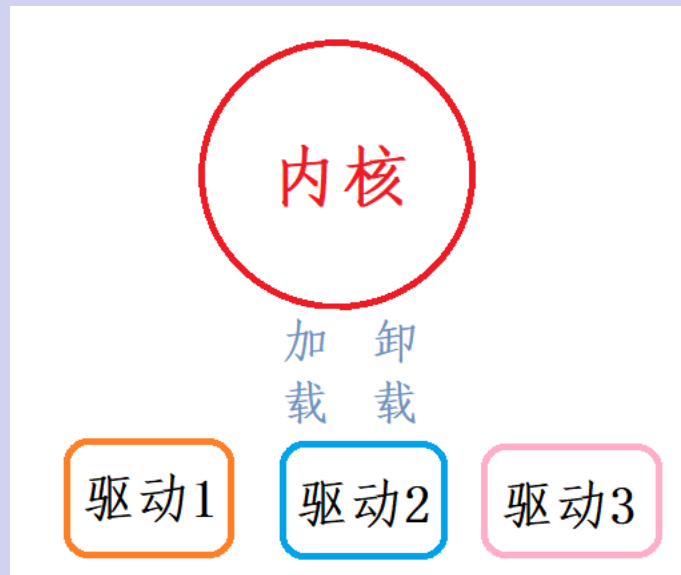
b.c

# 驱动开发



# 驱动开发

**Linux**中还有一个很重要的概念--**模块**。可在运行时添加到内核中的代码被称为模块。一般来说一个设备驱动总是被写成一个模块。





# 驱动开发

**printk ()** 是**Linux**内核中最广为人知的函数之一。它是我们打印消息的标准工具，通常也是追踪和调试的最基本方法。



# 驱动开发

```
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/init.h>
MODULE_LICENSE("GPL");
//入口函数，执行 insmod 时执行
static int __init hello_2_init (void)
{
    printk (KERN_INFO "Hello world\n");
    return 0;
}
//出口函数，执行 rmmod 时执行
static void __exit hello_2_exit (void)
{
    printk (KERN_INFO "Goodbye world\n");
}
module_init (hello_2_init);
module_exit (hello_2_exit);
```

# 驱动开发

这个最简单的内核模块只包含内核模块加载函数、卸载函数，用户可以通过编译生成 **ko** 后通过 **insmod** 命令加载到内核中，通过 **rmmod** 命令可以卸载这个模块，加载时输出 “**Hello World**”，卸载时输出 “**Goodbye World**”。

# 驱动开发

## 内核模块程序结构

### （1） 模块加载函数

当通过 **insmod** 或 **modprobe** 命令加载内核模块时，模块的加载函数会自动被内核执行，完成模块的相关初始化工作。

### （2） 模块卸载函数

当通过 **rmmod** 命令卸载某模块时，模块的卸载函数会自动被内核执行，完成与模块卸载函数中的退出功能。

### （3） 模块许可证声明

许可证声明描述内核模块的许可权限，如果不声明 **LICENSE**，模块被加载时，将收到内核被污染（**Kernel Tainted**）的警告。

# 驱动开发

## 内核模块程序结构

### 模块加载函数

当加载驱动模块时，内核会通过`module_init(xxx_init);`执行模块加载函数，完成模块加载函数中的初始化工作。一般模块加载函数形式如下：

```
1  static int __init xxx_init(void)
2  {
3      //驱动加载需要完成的任务
4
5      return 0;
6  }
```

# 驱动开发

```
1 static int __init xxx_init(void)
2 {
3     //驱动加载需要完成的任务
4
5     return 0;
6 }
```

\_\_init\* macro

\_\_init定义在:include/linux/init.h

```
#define __init __attribute__((__section__ (".init.text")))
#define __initdata __attribute__((__section__ (".init.data")))
```

# 驱动开发

```
1 static int __init xxx_init(void)
2 {
3     //驱动加载需要完成的任务
4
5     return 0;
6 }
```

**\_\_init**宏告知编译器，将变量或函数放在一个特殊的区域，这个区域定义在**vmlinux.lds**中。**\_\_init**将函数放在**".init.text"**这个代码区中，**\_\_initdata**将数据放在**".init.data"**这个数据区中。

标记为初始化的函数,表明该函数供在初始化期间使用

。在模块装载之后，模块装载就会将初始化函数扔掉。

这样可以将该函数占用的内存释放出来。

# 驱动开发

可接受的内核模块声明许可包括**GPL**、**GPL v2**。具体设置驱动的许可声明可使用函数

```
MODULE_LICENSE("GPL");
```

**GPL**最常用



# 驱动开发

## 驱动开发

# 驱动开发

**\*obj-m\***表示需要编译成模块的目标文件名集合（不是编译进内核）

获取相对路径必须在**pwd**前面加**shell**，然后把**shell pwd**当一个变量来引用，书写形式是：**\$(shell pwd)**符号“**:=**”的意思是将**\$(shell pwd)**表示为**INCDIR**

**-C** 选项的作用是指将当前工作目录转移到你所指定的位置，一般都是内核源代码目录

“**M=**”选项的作用是，当用户需要以某个内核为基础编译一个外部模块的话，需要在**make modules** 命令中加入

“**M=dir**”，程序会自动到你指定的**dir**目录中查找模块源码，将其编译，生成**KO**文件。

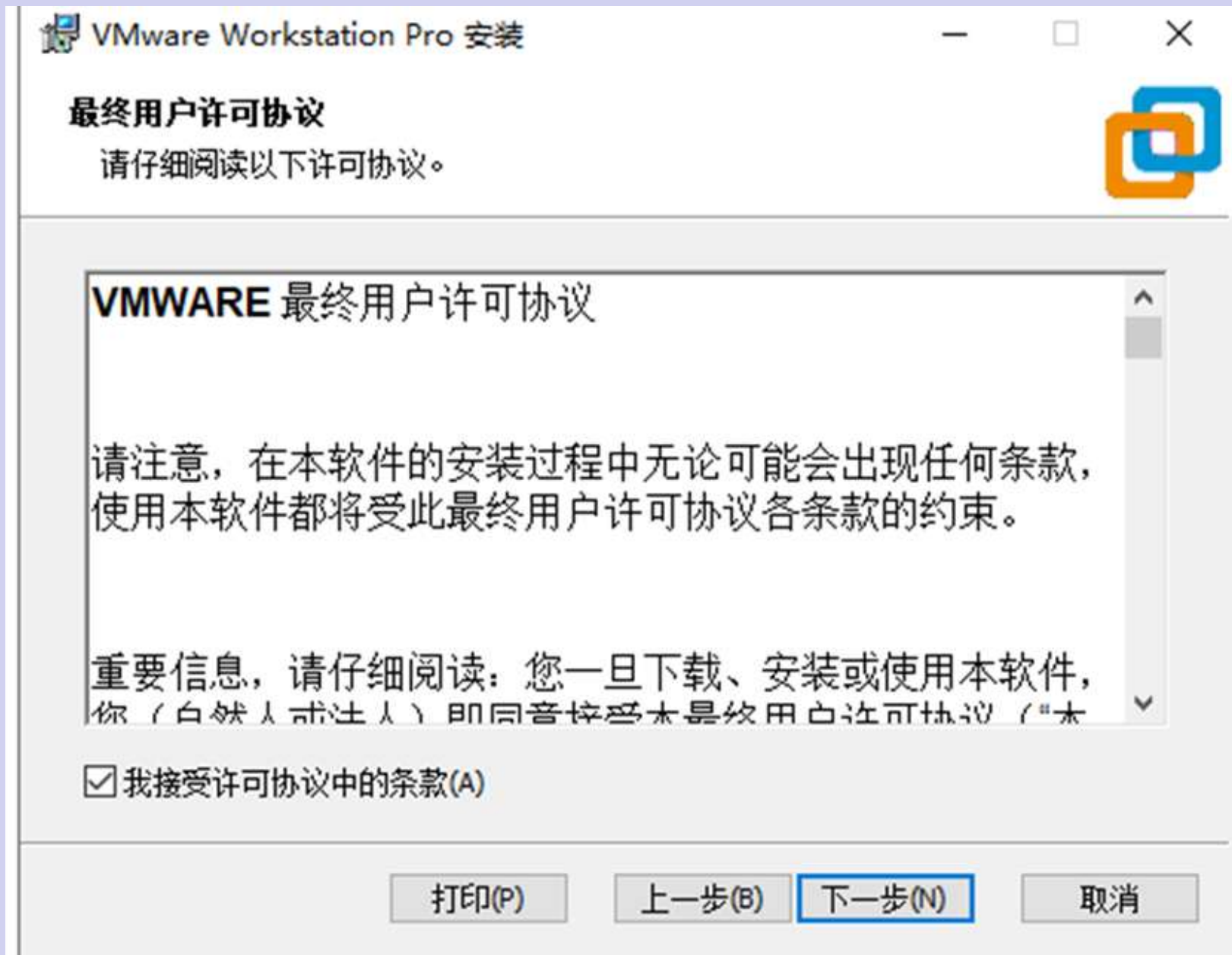
# 驱动开发

## 驱动开发

# 开发环境的搭建



# 开发环境的搭建



# 开发环境的搭建

VMware Workstation Pro 安装

## 自定义安装

选择安装目标及任何其他功能。



安装位置:

D:\VMWARE\

更改...

☒ 增强型键盘驱动程序(需要重新引导以使用此功能(E))  
此功能要求主机驱动器上具有 10MB 空间。

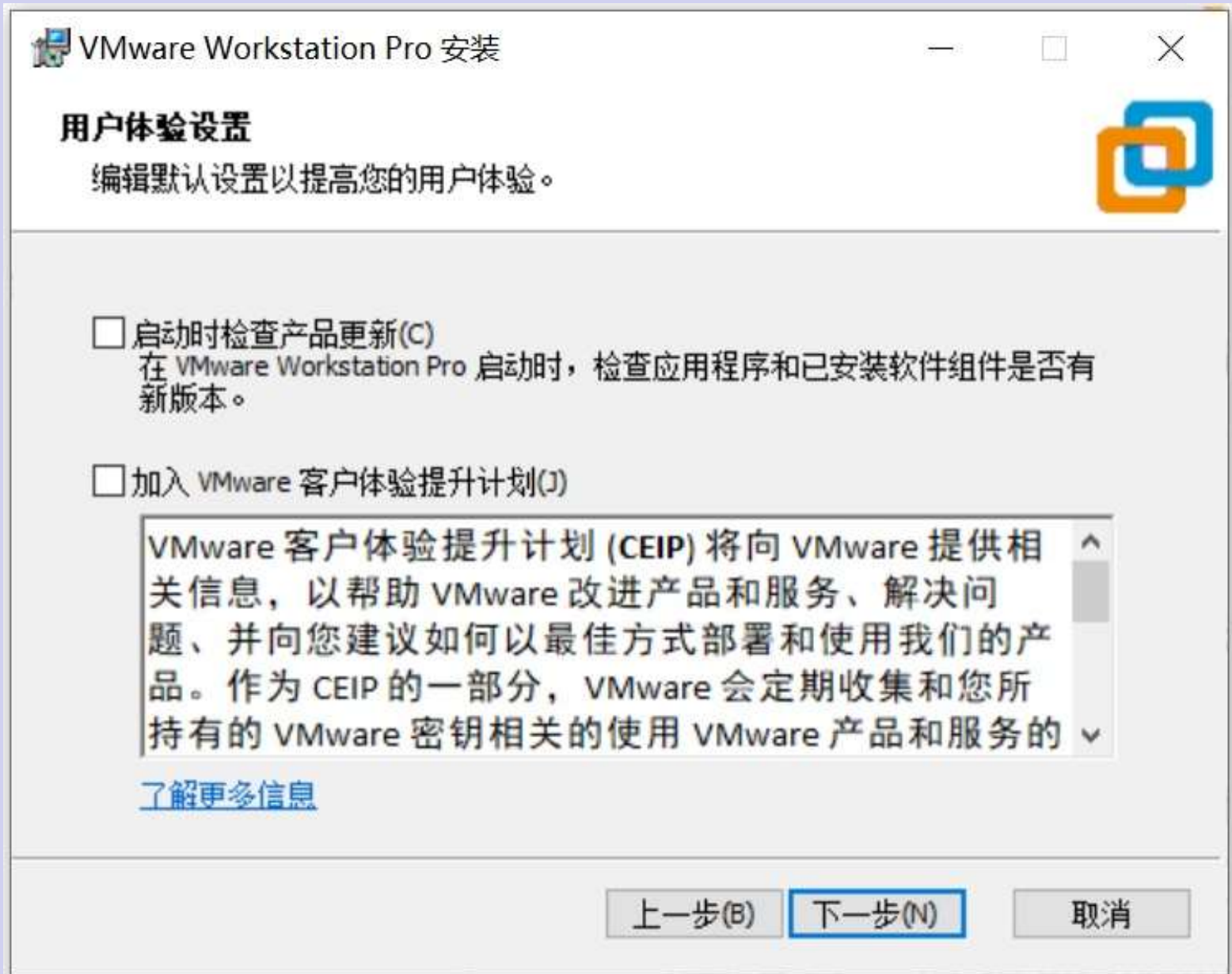
☒ 将 VMware Workstation 控制台工具添加到系统 PATH

上一步(B)

下一步(N)

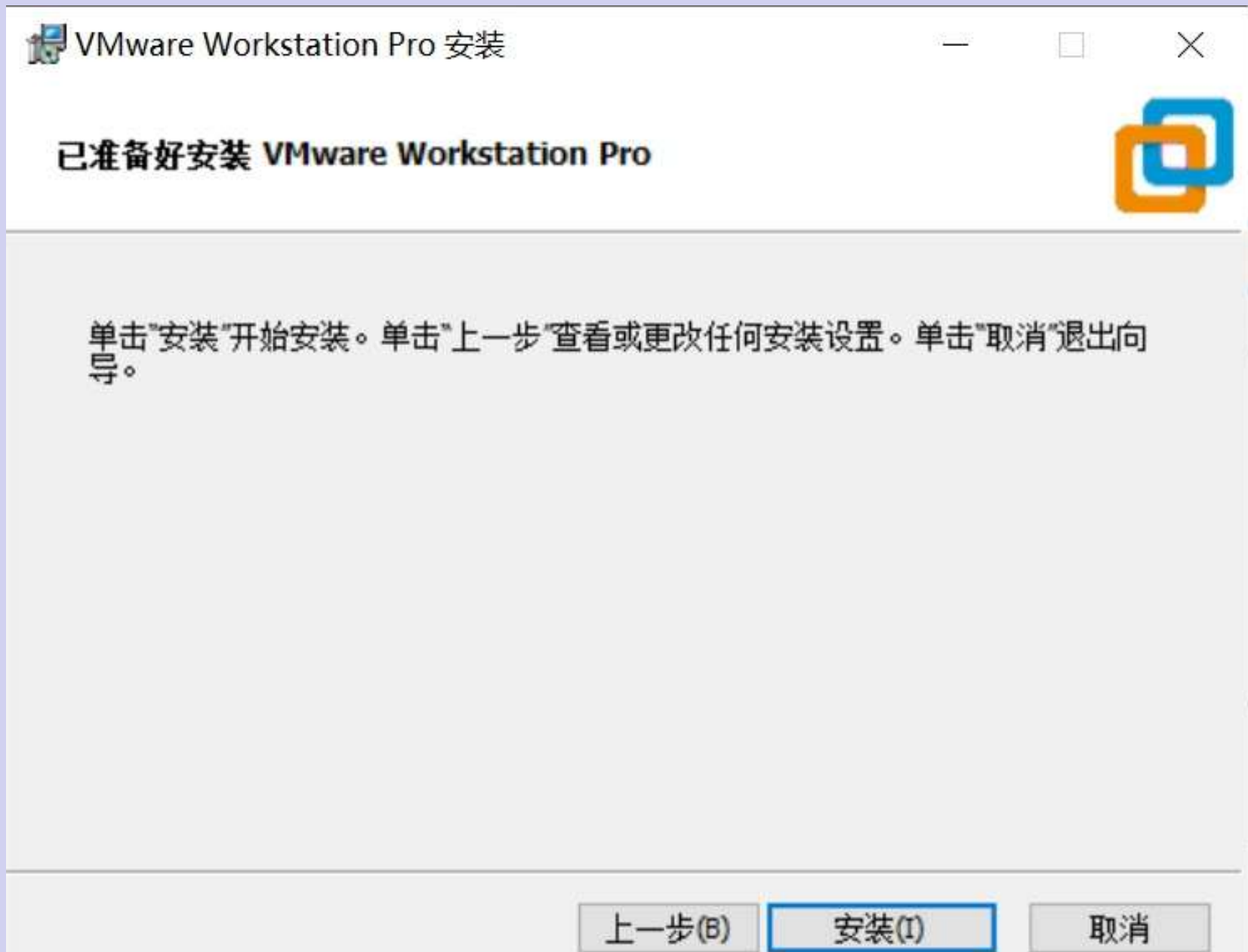
取消

# 开发环境的搭建



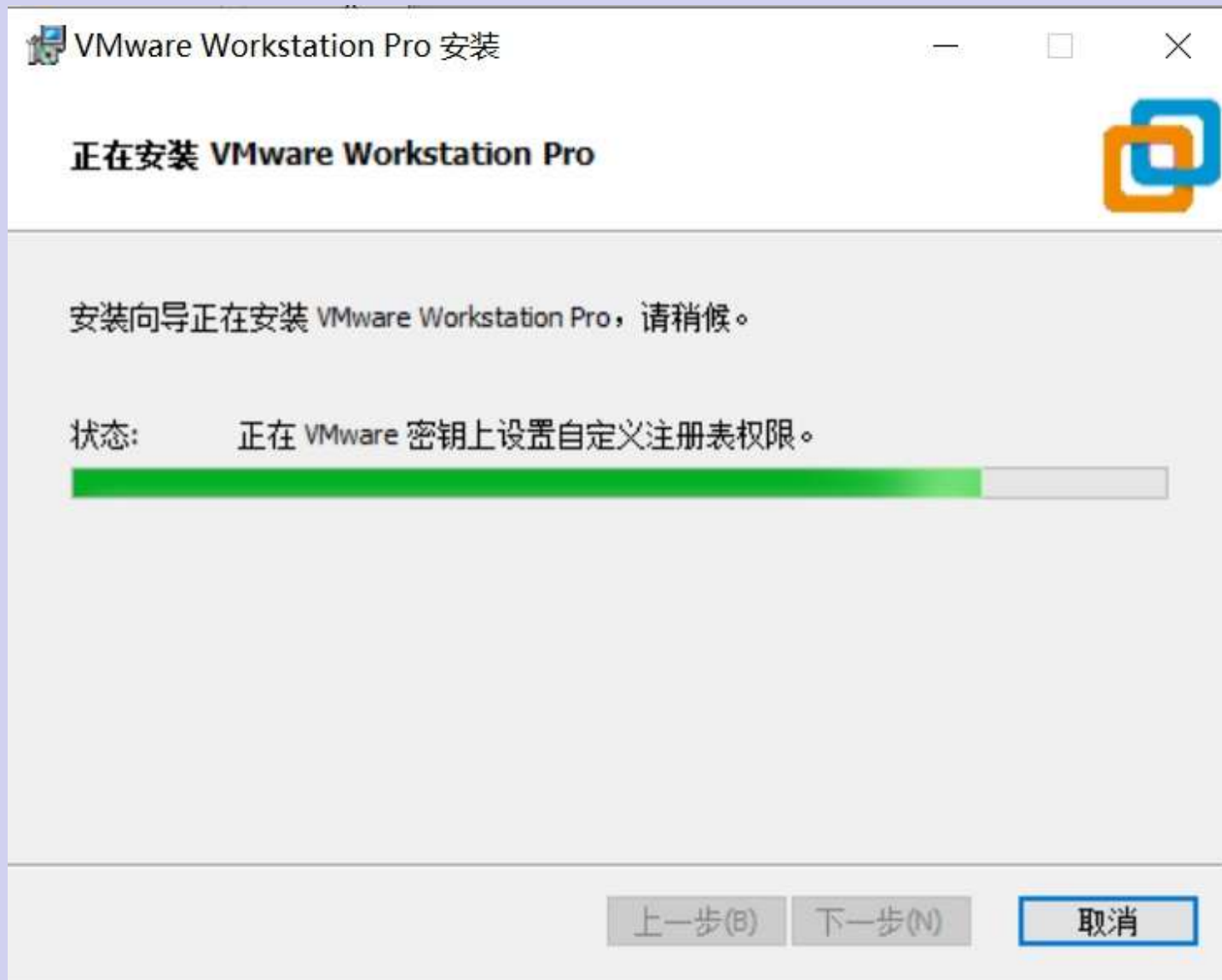


# 开发环境的搭建

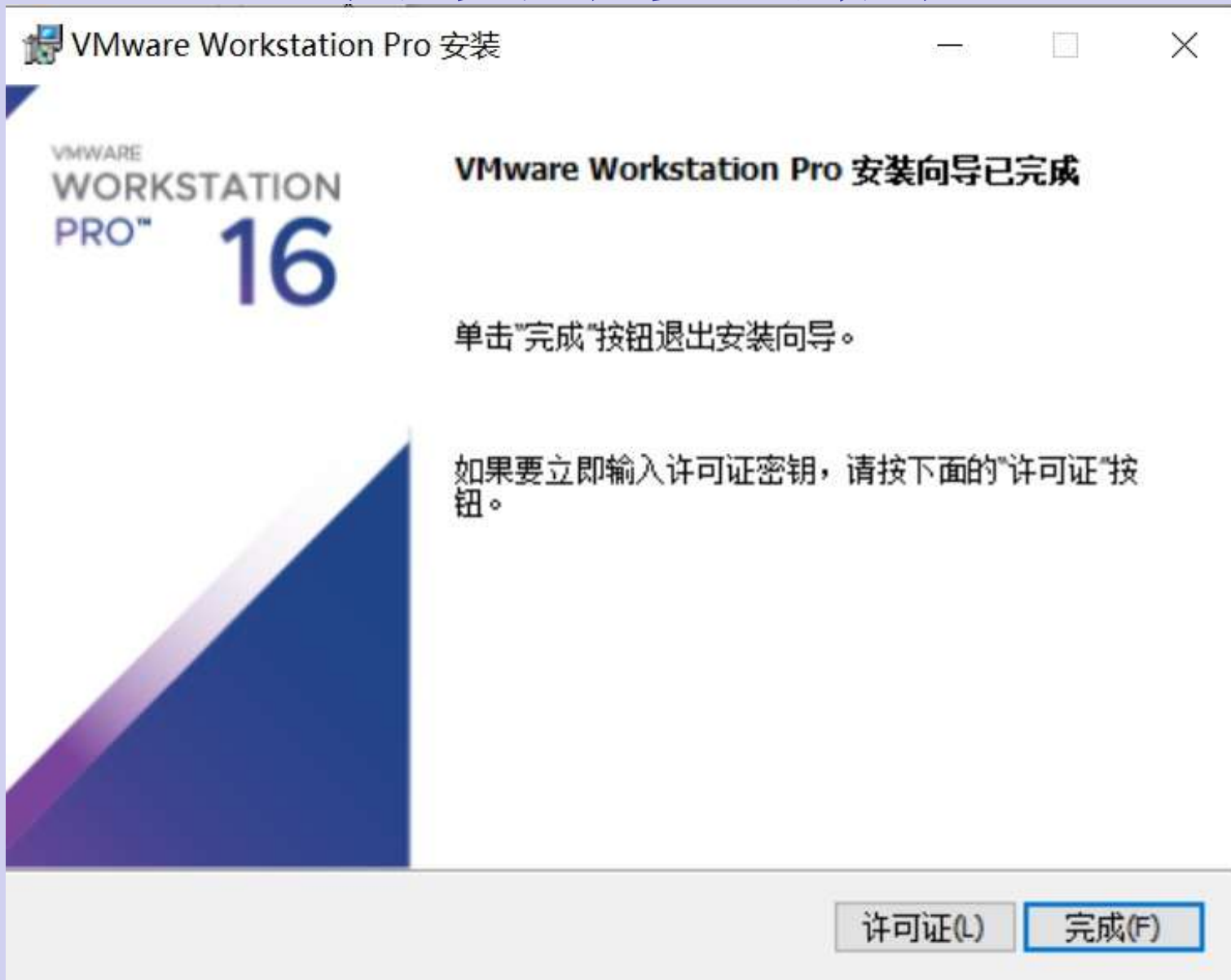


# 开发环境的搭建

# 开发环境的搭建



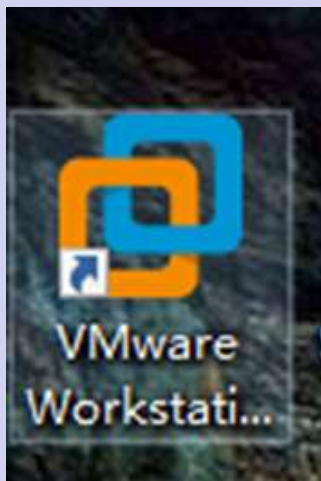
# 开发环境的搭建



# 开发环境的搭建

此电脑 > 软件 (E:) > 高级嵌入式综合实验平台 (实验箱) FB-EDU-EBS-A 资料 > DISK-Embedded > 02\_工具软件 > VMware-workstation-full-16.2.2

名称	修改日期	类型	大小
vmware 16 激活码.txt	2021/8/23 14:18	文本文档	1 KB
VMware-workstation-full-16.2.2-19200509....	2022/9/5 12:47	应用程序	630,195 KB



# 开发环境的搭建

WORKSTATION 16 PRO™



创建新的虚拟机



打开虚拟机



连接远程服务器

# 开发环境的搭建

此电脑 > 软件 (E:) > 高级嵌入式综合实验平台 (实验箱) FB-EDU-EBS-A 资料 > DISK-Embedded > 01\_系统镜像及源码 > 01-虚拟机镜像

名称	修改日期	类型	大小
 ubuntu16.04-origian ver.rar	2023/2/8 10:50	WinRAR archive	1,497,394 KB
 ubuntu16-complete.rar	2024/2/23 22:17	WinRAR archive	7,951,132 KB
 镜像说明.txt	2023/2/8 11:07	文本文档	1 KB

复制到一个单独文件夹下，然后解压。










涉及Linux的安装的，不要有中文、空格或其他符号



# 开发环境的搭建

共享查看

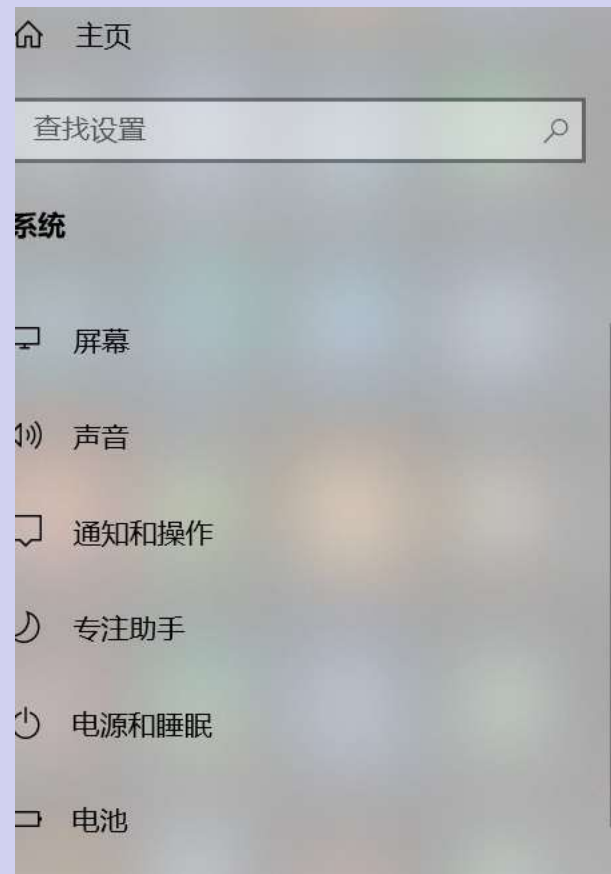
> 此电脑 > 软件 (E:) > ubuntu16-complete > ubuntu16.04

名称	修改日期	类型	大小
 mksSandbox.log	2024/2/27 20:17	文本文档	45 KB
 ubuntu16.04.nvram	2024/2/27 20:17	VMware 虚拟机非易...	9 KB
 ubuntu16.04.vmsd	2022/9/2 17:32	VMware 快照元数据	0 KB
 ubuntu16.04.vmx	2024/2/27 20:17	VMware 虚拟机配置	4 KB
 ubuntu16.04.vmx	2022/9/2 17:39	VMware 组成员	1 KB
 ubuntu16.04-cl1.vmdk	2024/2/27 20:17	VMware 虚拟磁盘文...	19,032,384 ...
 vm.scoreboard	2024/2/27 20:13	SCOREBOARD 文件	8 KB
 vmware.log	2024/2/27 20:17	文本文档	221 KB
 vmware-0.log	2024/2/23 22:06	文本文档	232 KB

# 开发环境的搭建



# 开发环境的搭建



## 关于

系统正在监控并保护你的电脑。

[在 Windows 安全中心中查看详细信息](#)

## 设备规格

设备名称	DESKTOP-IN9KTDA
处理器	Intel(R) Core(TM) i7-6500U CPU @ 2.50GHz 2.59 GHz
机带 RAM	8.00 GB (7.43 GB 可用)
设备 ID	1346A8A7-1BFD-4103-B63D-F1C0838C7FA4
产品 ID	00328-20090-00000-AA015
系统类型	64 位操作系统, 基于 x64 的处理器
笔和触控	没有可用于此显示器的笔或触控输入

复制

# 开发环境的搭建



www.jd.com

**intel-酷睿i7 6500u- [京东] 电脑办公,高科技,...**

来自百度 **intel 酷睿i7 6500u-**「京东」电脑办公,笔记本/**CPU**/鼠标/键盘/打印机,应有尽有,一站购「京东」品类全,折扣狠,送货快,省事又省心,享受购物就逛「JD.com」! 广告



英特尔

<https://www.intel.cn/content/www/cn/zh/products/sku/...>

**英特尔® 酷睿™ i7-6500U 处理器**

网页 1 天前 · 处理器编号 **i7-6500U** 光刻 14 nm **CPU** 规格 内核数 2 线程数 4 最大睿频频率 3.1

---

进一步探索

# 开发环境的搭建

虚拟机设置

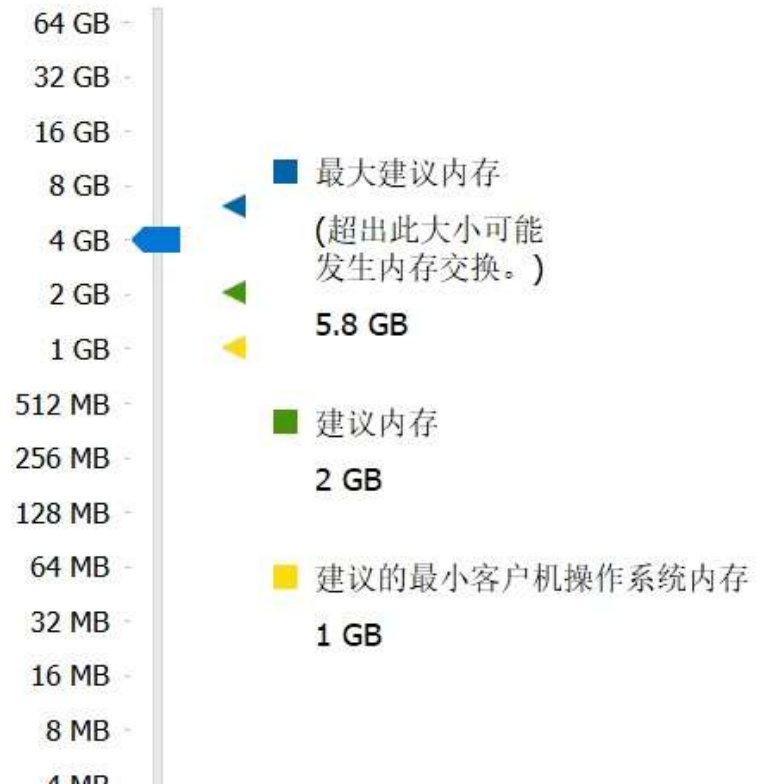
硬件 选项

设备	摘要
内存	4 GB
处理器	1
硬盘 (SCSI)	100 GB
CD/DVD (SATA)	自动检测
网络适配器	桥接模式 (自动)
USB 控制器	存在
声卡	自动检测
打印机	存在
显示器	自动检测

## 内存

指定分配给此虚拟机的内存量。内存大小必须为 4 MB 的倍数。

此虚拟机的内存(M):  MB



# 开发环境的搭建

虚拟机设置

硬件

选项

设备	摘要
内存	4 GB
处理器	1
硬盘 (SCSI)	100 GB
CD/DVD (SATA)	自动检测
网络适配器	桥接模式 (自动)
USB 控制器	存在
声卡	自动检测
打印机	存在
显示器	自动检测

处理器

处理器数量(P): 1

每个处理器的内核数量(C): 1

处理器内核总数: 1

虚拟化引擎

☐ 虚拟化 Intel VT-x/EPT 或 AMD-V/RVI(V)

☐ 虚拟化 CPU 性能计数器(U)

# 开发环境的搭建

虚拟机设置



硬件 选项

设备	摘要
内存	4 GB
处理器	1
硬盘 (SCSI)	100 GB
CD/DVD (SATA)	自动检测
网络适配器	桥接模式 (自动)
USB 控制器	存在
声卡	自动检测
打印机	存在
显示器	自动检测

设备状态

- ☐ 已连接(C)  
☒ 启动时连接(O)

网络连接

- ☒ 桥接模式(B): 直接连接物理网络  
☐ 复制物理网络连接状态(P)  
☐ NAT 模式(N): 用于共享主机的 IP 地址  
☐ 仅主机模式(H): 与主机共享的专用网络  
☐ 自定义(U): 特定虚拟网络

VMnet0

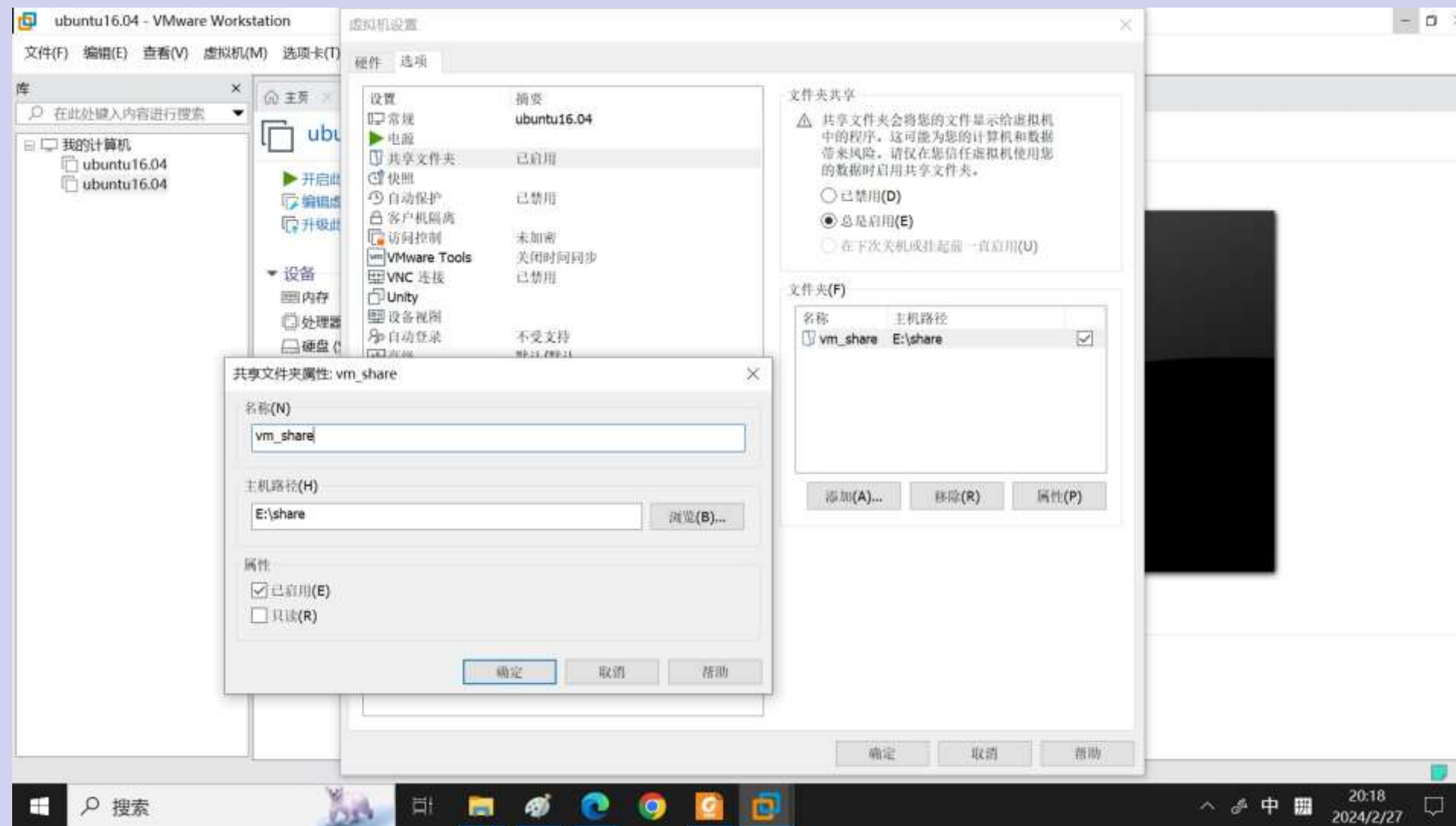
- ☐ LAN 区段(L):

LAN 区段(S)...

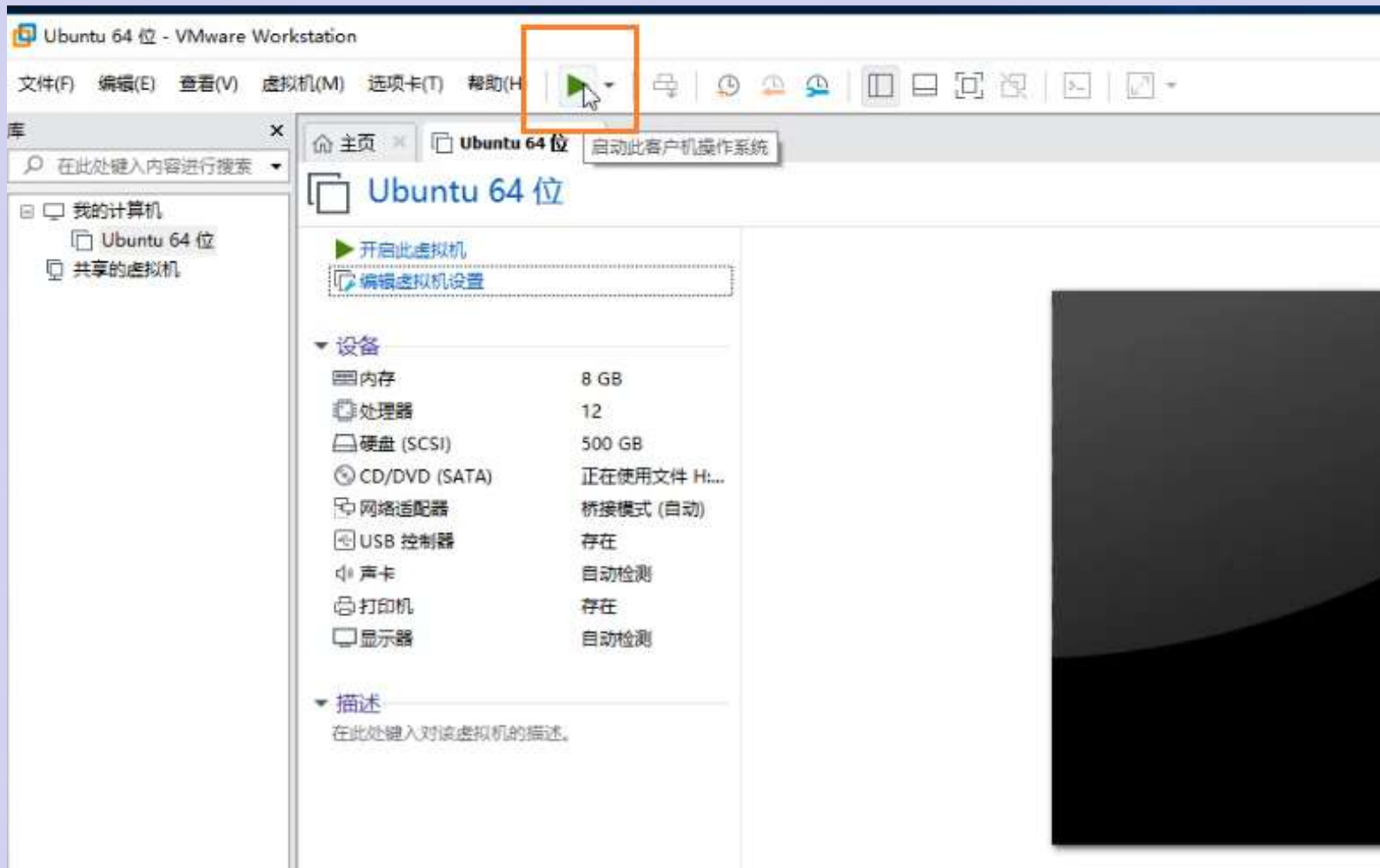
高级(V)...



# 开发环境的搭建

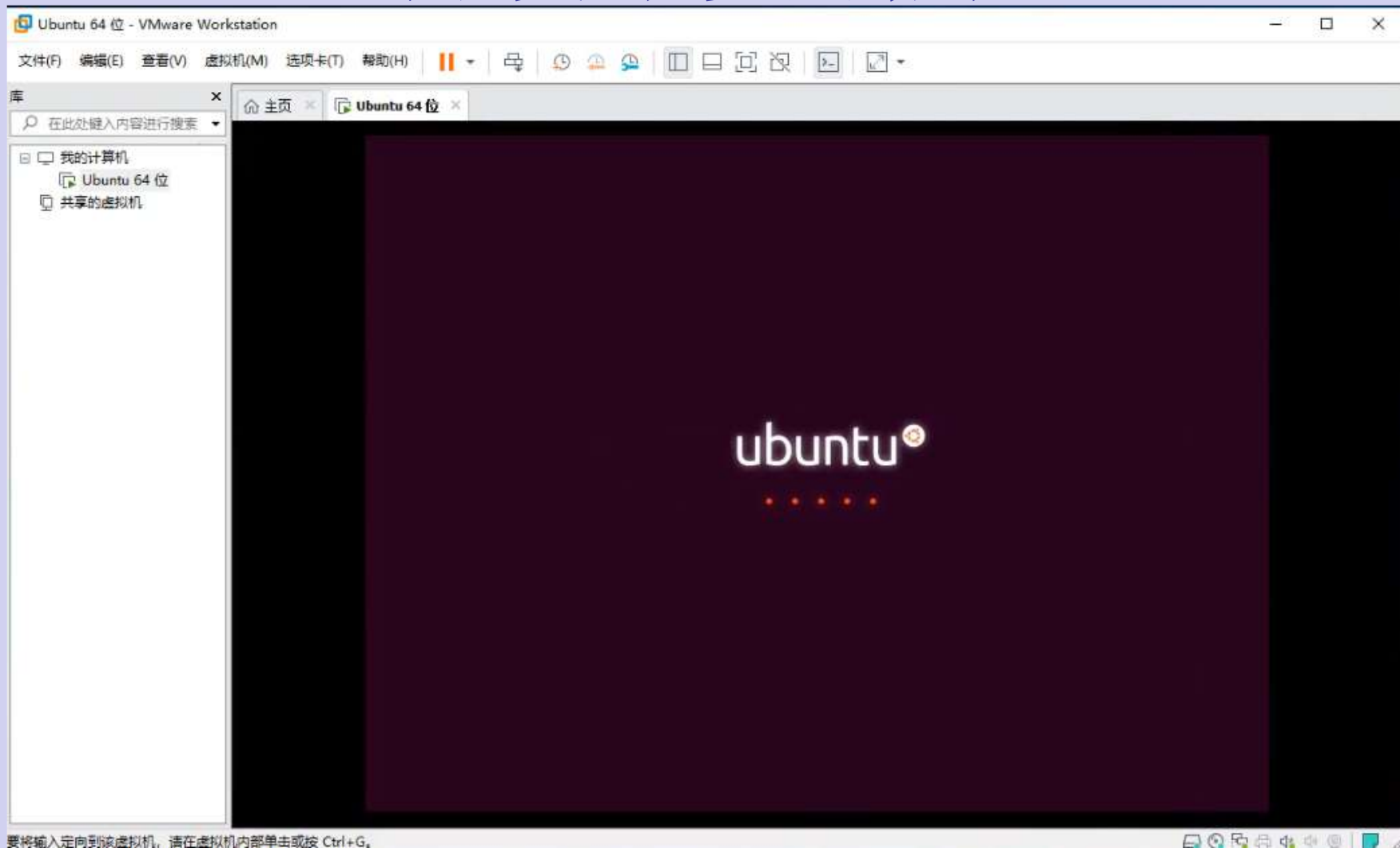


# 开发环境的搭建



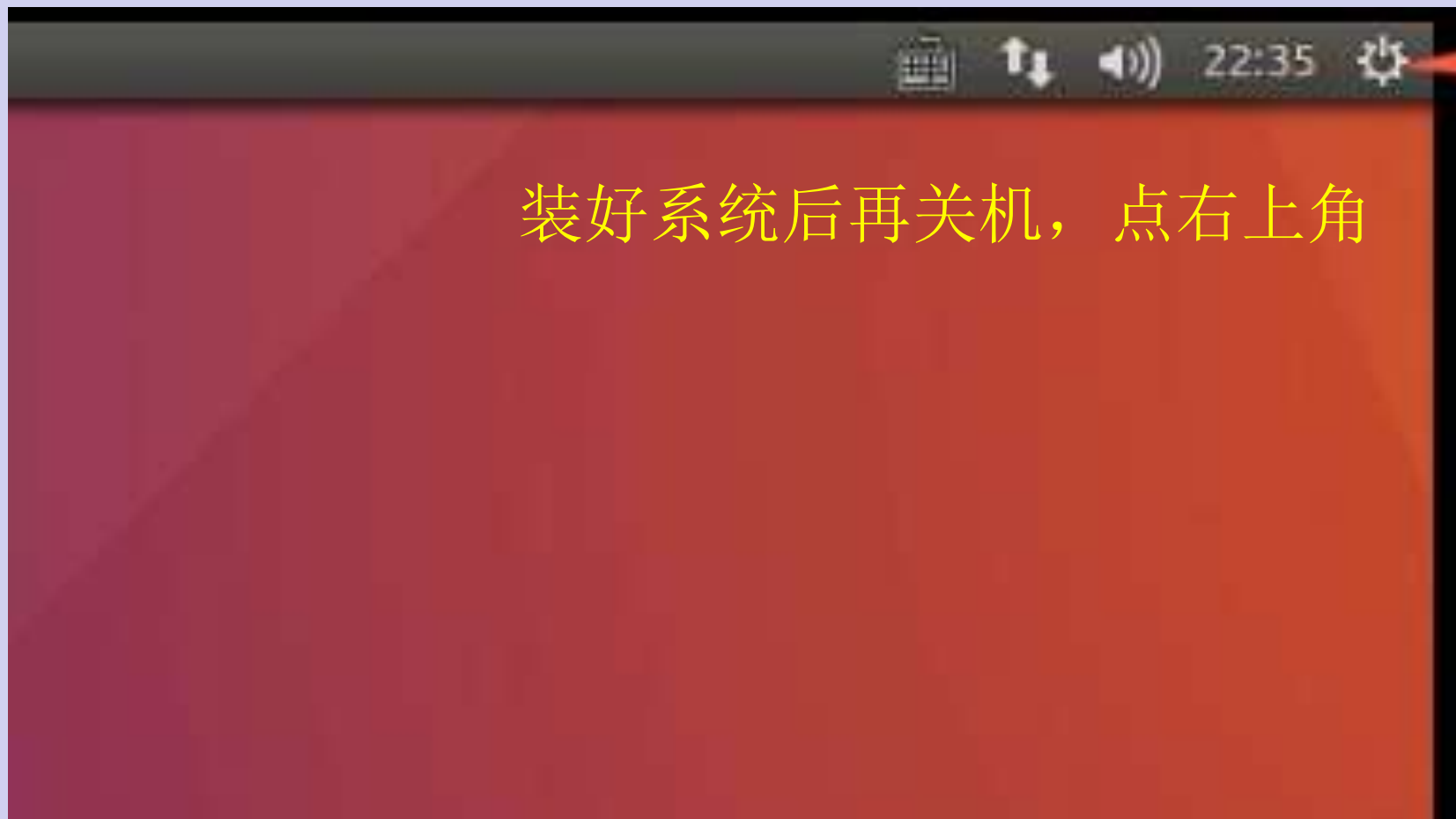
这时候相当于光驱中有安装光盘了，打开虚拟机即自动安装

# 开发环境的搭建

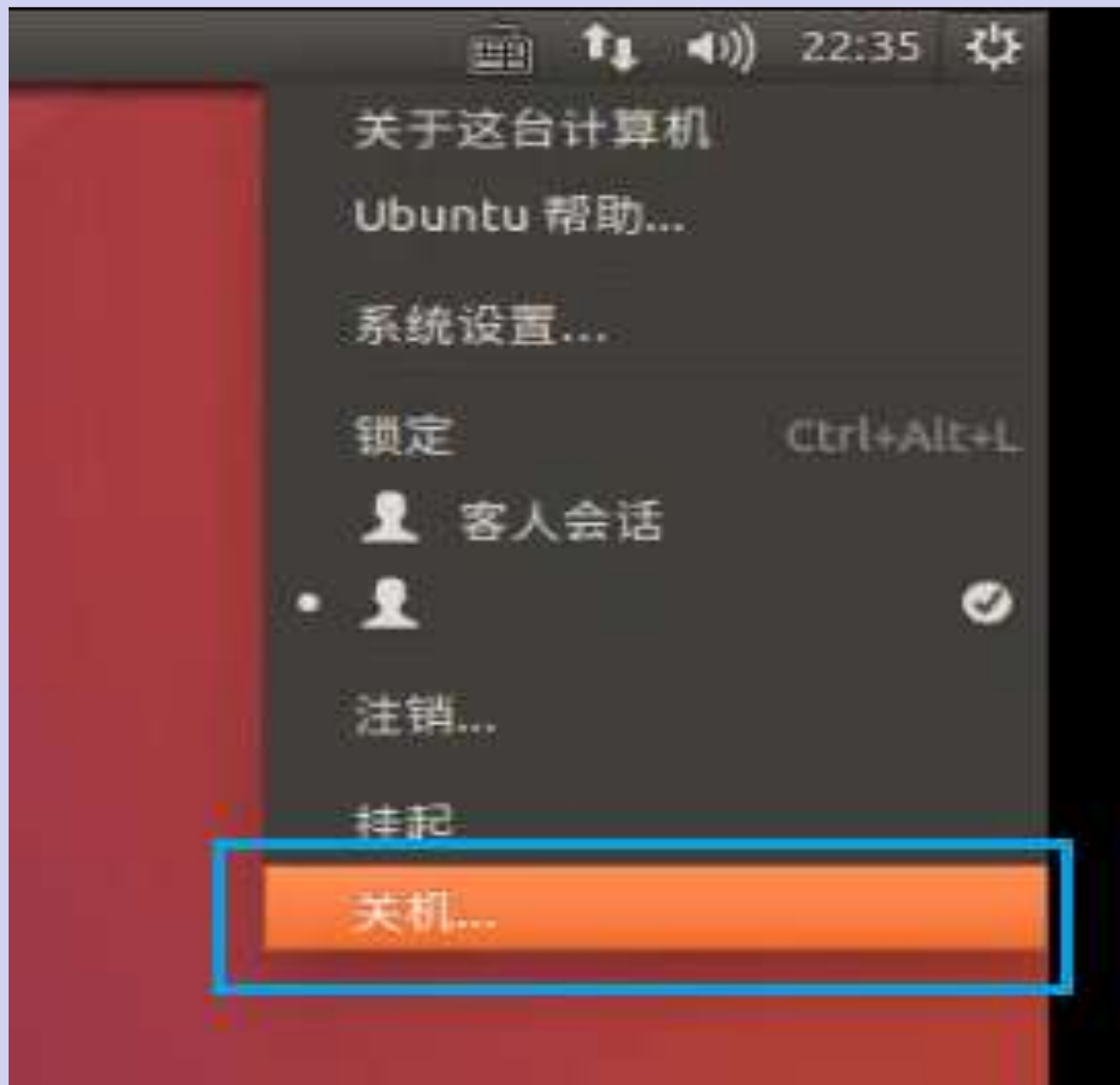


进入安装状态

# 开发环境的搭建



# 开发环境的搭建



# 开发环境的搭建



大家有时间搜索查询各个文件夹的作用

# 开发环境的搭建

可以查看一下虚拟机和主机的**IP**地址

首先我们在**windows**下按下快捷键**WIN+R**打开运行，输入**cmd**回车，随后在**cmd**中输入**ipconfig**命令



# VI的模式切换

- 进入命令模式：按下**Esc**键即可进入命令模式。
- 进入插入模式：在命令模式下，按下**i**、**a**、**o**等按键即可进入不同的插入模式。
- 进入底行模式：在命令模式下，按下**:**（冒号）即可进入底行模式

# 特殊符号

- ~ 帐户的 home 目录
- ; 分号 在命令与命令中间利用分号（; ）来隔开
- . 点号一个 .代表当前目录，两个 ..代表上层目录

更多的特殊符号解释，可查看

<https://cloud.tencent.com/developer/article/1454832>

# 特殊符号

- ~ 是用户的主目录,root用户的主目录是/root,普通用户的主目录是 “/home/普通用户名 ” 1、在root用户下, ~等同于/root 2、在普通用户下, ~ 等同于 /home/当前的普通用户名

更多的特殊符号解释, 可查看

<https://cloud.tencent.com/developer/article/1454832>

把WINDOWS的内容复制粘贴到vi:

在windows复制后, 在进入vi编辑器后,先切换成输入状态(即按下i键),再Shift+Insert

# 开发环境的搭建

## Windows和Ubuntu之间的文件互传

### 开启 Ubuntu 下的 FTP 服务

打开 Ubuntu 的终端窗口，然后执行如下命令来安装 FTP 服务：

```
sudo apt-get install vsftpd
```

等待软件自动安装，安装完成以后使用 VI 命令打开/etc/vsftpd.conf，命令如下：

```
sudo vi /etc/vsftpd.conf
```

打开 vsftpd.conf 文件以后找到如下两行：

```
local_enable=YES
```

```
write_enable=YES
```

确保上面两行前面没有“#”，有的话就取消掉，完成以后如图 4.1.1 所示：

```
27 # Uncomment this to allow local users to log in.
28 local_enable=YES
29 #
30 # Uncomment this to enable any form of FTP write command.
31 write_enable=YES
32 #
```

vsftpd.conf 修改

修改完 vsftpd.conf 以后保存退出，使用如下命令重启 FTP 服务：

```
sudo /etc/init.d/vsftpd restart
```

# 开发环境的搭建

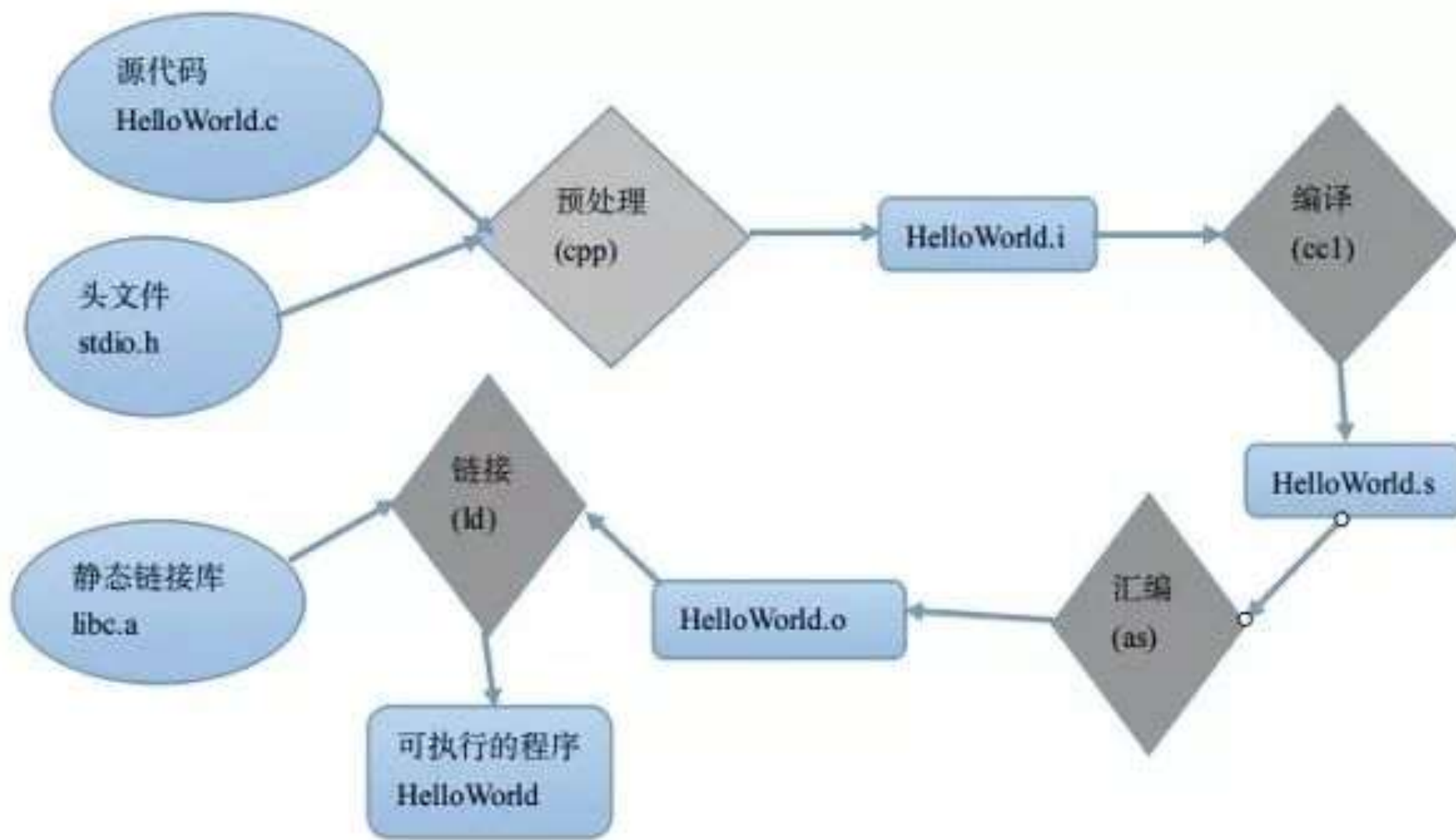
在Windows上安装Filezilla



# 开发环境的搭建

在Windows上安装Filezilla





# 开发环境的搭建

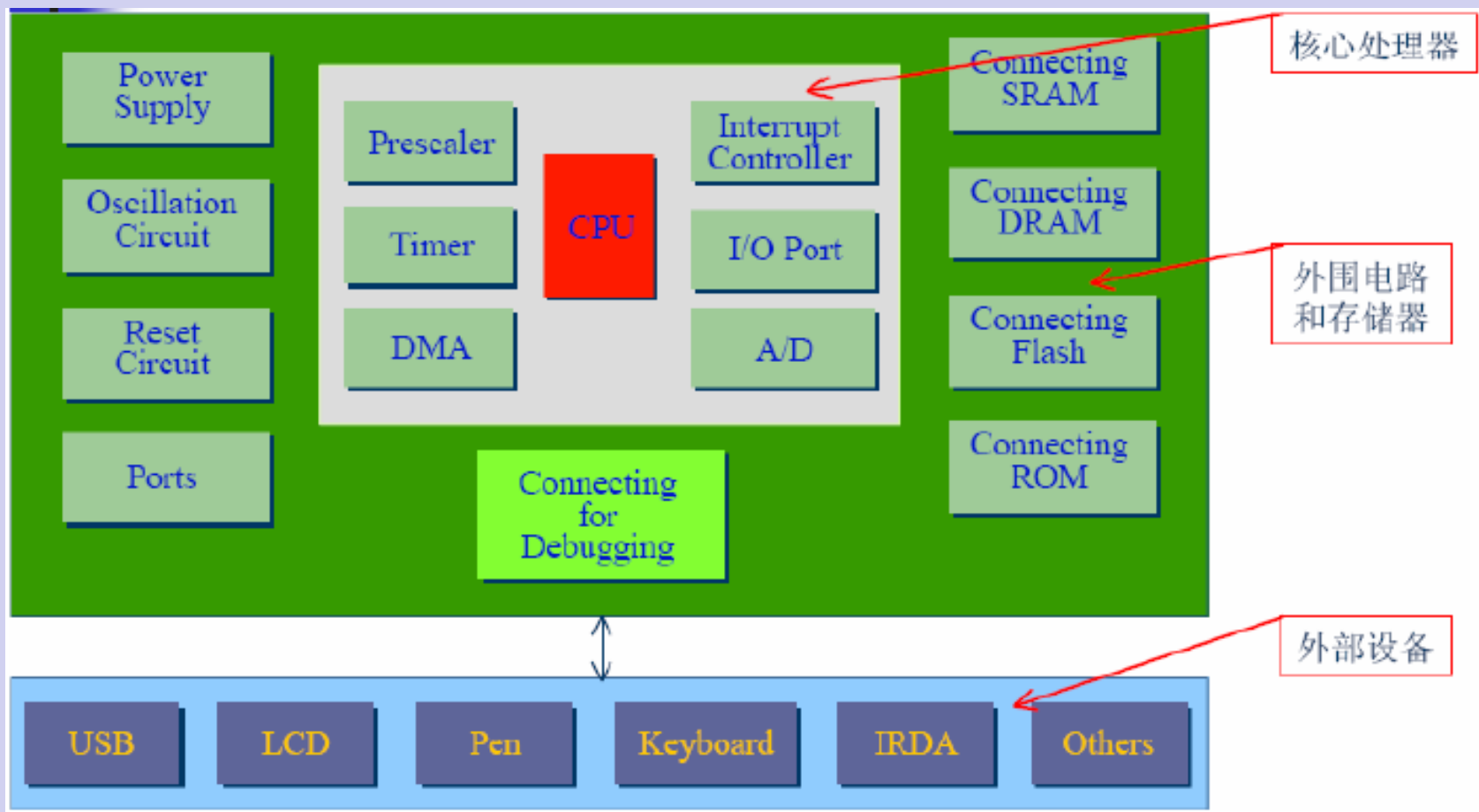


# 开发环境的搭建

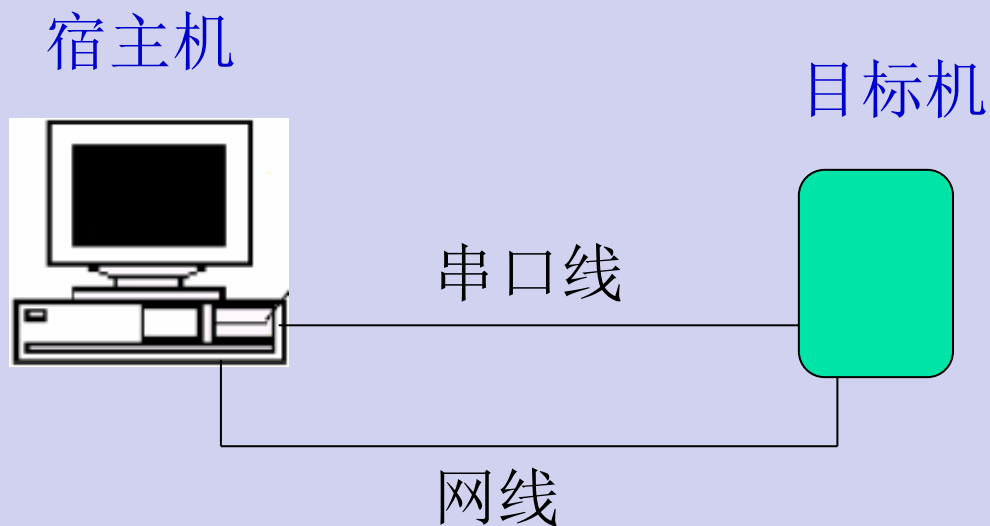
## 2.1 嵌入式系统的构架

API(Application Programming Interface)

## 2.2 嵌入式系统的硬件基本结构



## 3.4 嵌入式系统的开发模式



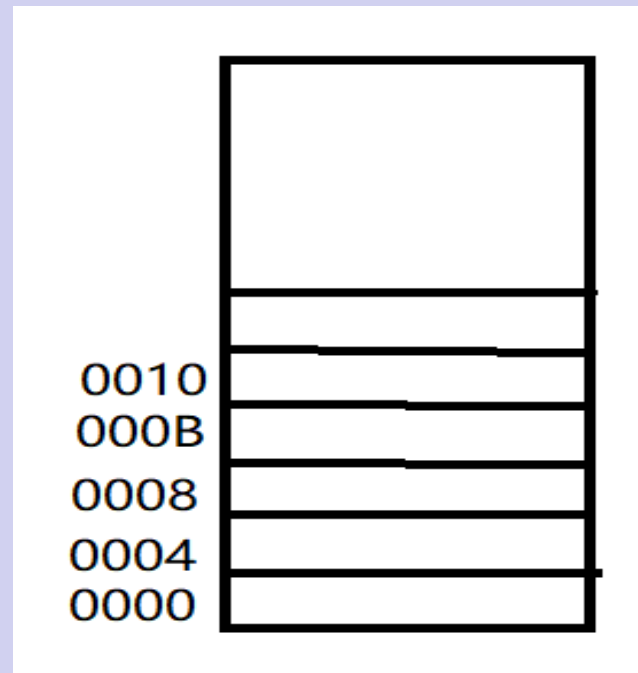
## 3.4 嵌入式系统的开发模式

- 嵌入式系统的软件使用交叉开发平台进行开发。
  - 系统软件和应用软件在主机开发平台上开发
  - 系统软件和应用软件在嵌入式硬件平台上运行。
- 宿主机 (Host) 是用来开发嵌入式软件的系统。
- 目标机 (Target) 是被开发的目的嵌入式系统。
- 交叉编译器 (Cross-compiler) 是进行交叉平台开发的主要软件工具。它是运行在一种处理器体系结构上, 但是可以生成在另一种不同的处理器体系结构上运行的目标代码的编译器。

# ARM编程模型

## 数据类型

1个**字节**8位，一个**字**4个字节，一个“**半字**”两个字节。  
对于ARM处理器，既可以对**字**操作，也可以对**半字**操作，也可以对**字节**进行操作做。那么这些数据字、字节、半字是怎么存储的呢？（指令实际上也是）



# 存储器格式

ARM体系结构可以用两种方法存储字数据，称之为**大端格式**和**小端格式**。

- **大端格式**：字数据的**高字节**存储在**低地址**中，而字数据的**低字节**则存放在**高地址**中。
- **小端格式**：与大端存储格式相反，在小端存储格式中，**低地址**中存放的是字数据的**低字节**，**高地址**存放的是字数据的**高字节**。



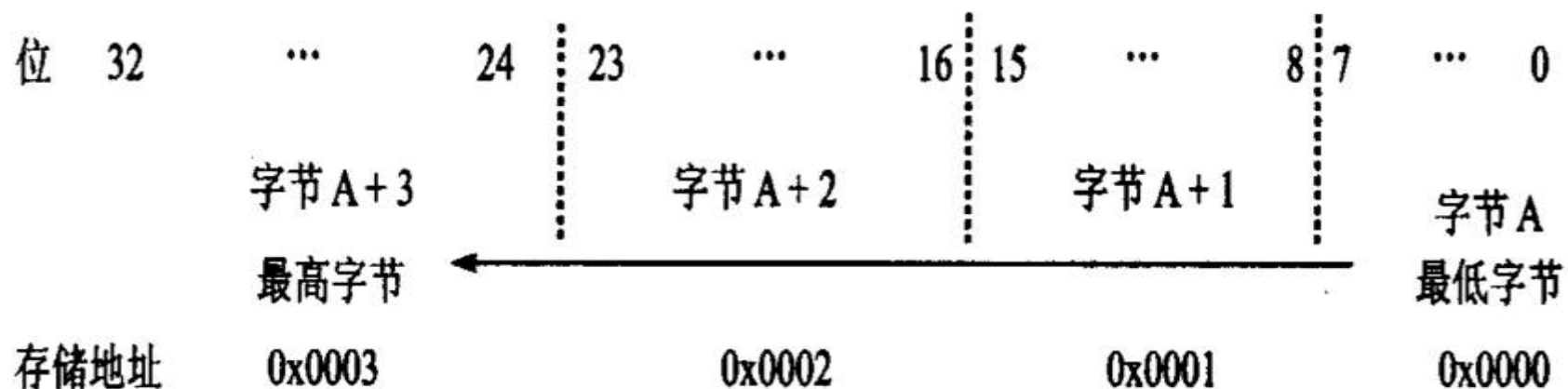


图 2.3 32 位字的小端存储格式

**小端格式：**低位字节放在字节A，高位字节放在字节A+3. 小端格式中，以最低位字节地址作为字地址

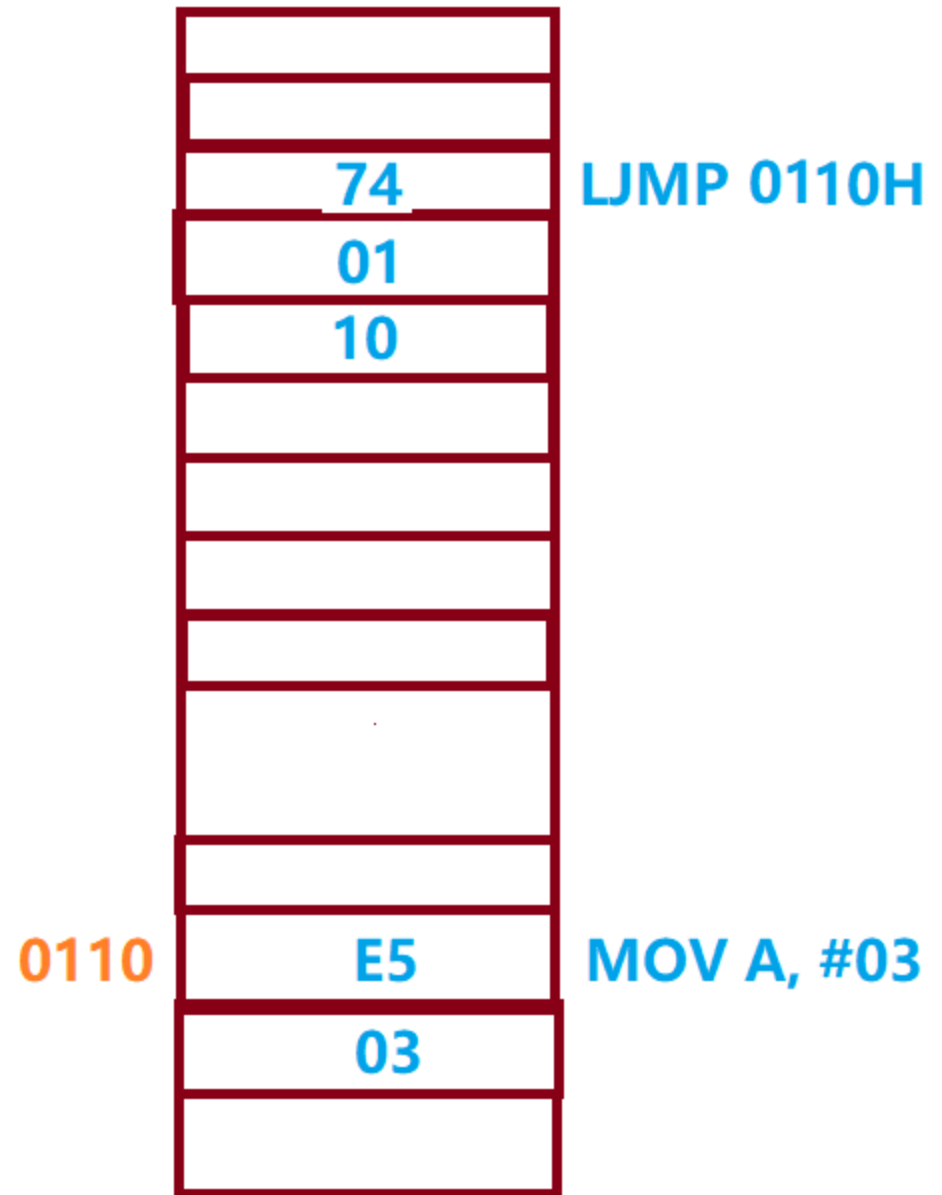
存储值	2A	36	47	8B	63	74	AC	9E	00	8F	23	66	C3	A8
地址	0x00 0D	0C	0B	0A	09	08	07	06	05	04	03	02	01	00
半字	半字7		半字6		半字5		半字4		半字3		半字2		半字1	
字					字3				字2				字1	

字1存储在0000地址中，字1和字2中的内容分别是什么数？

对字的操作，不能取地址在0001处的数据，这叫作**地址对准**。

**正确的地址入口叫地址对准**

# LJMP 0111H?



0007H

0006H

0005H

0004H

0003H

0002H

0001H

0000H

68

52

26

1A

03

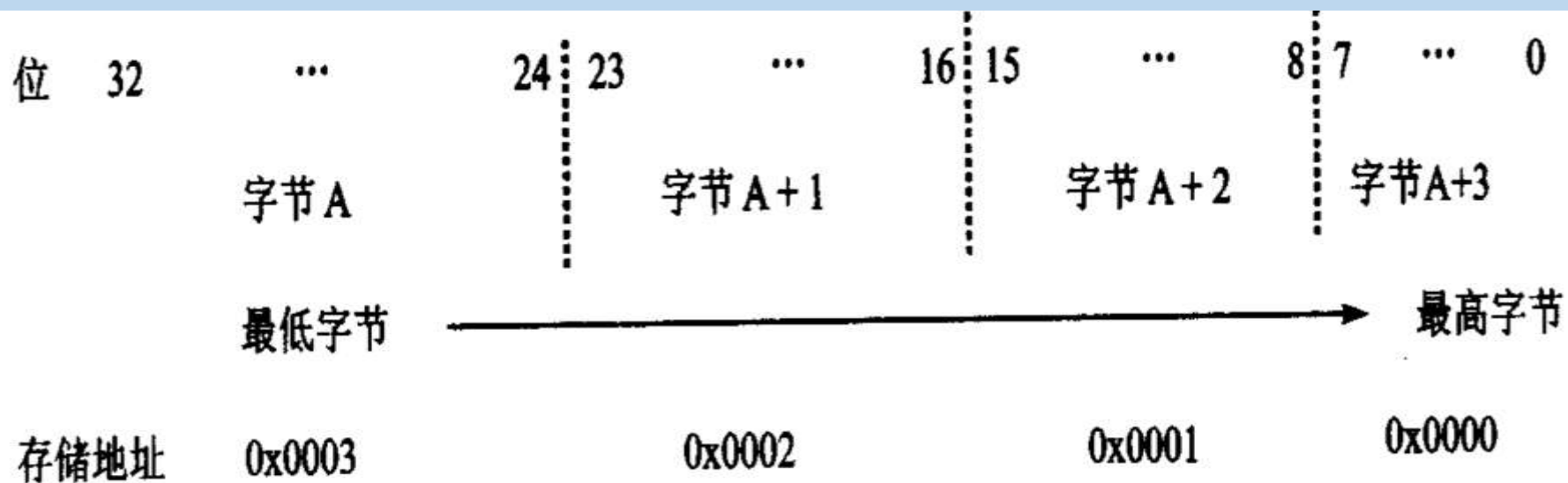


图 2.6 32 位字的大端存储格式

大端格式正好相反，略。

存储值	2A	36	47	8B	63	74	AC	9E	00	8F	23	66	C3	A8	
地址	0x00	0D	0C	0B	0A	09	08	07	06	05	04	03	02	01	00
半字	半字8		半字5		半字6		半字3		半字4		半字1		半字2		
字	字3				字2				字1						

**图 2.8 大端存储格式中字和半字的命名举例**

字一存储在0000地址中。

# 工作模式

ARM微处理器支持7种工作模式，分别为：

## 1、用户模式 (Usr)

用于正常执行程序-----大多数程序运行在用户模式，除了用户模式都叫**特权模式**。**特权模式可以访问更多资源**。51单片机只有一种模式，所有资源都可以用。

## 2、快速中断模式(FIQ)

用于高速数据传输

## 3、外部中断模式(IRQ)

用于通常的中断处理

# 工作模式

## 4. 管理模式 (svc)

操作系统使用的保护模式----进入OS后，进入管理模式

## 5. 数据访问终止模式(abt)

当数据或指令预取终止时进入该模式，可用于虚拟存储及存储保护。

## 6. 系统模式 (sys)

运行具有特权的操作系统任务。

## 7. 未定义指令中止模式 (und)

当未定义的指令执行时进入该模式，可用于支持硬件



# 工作模式

ARM微处理器的运行模式可以通过软件改变，也可以通过外部中断或异常处理改变。应用程序运行在用户模式下，当处理器运行在用户模式下时，某些被保护的系统资源是不能被访问的。

**注意：**用户模式下的应用程序不能修改实现模式转换

**51**也有中断，但不产生其他模式，因为它没有特权，所有资源在所有情况下共享

# 工作模式

除用户模式以外，其余的所有6种模式称之为非用户模式，或特权模式（Privileged Modes）；其中除去用户模式和系统模式以外的5种又称为异常模式（Exception Modes），常用于处理中断或异常，以及需要访问受保护的系统资源等情况。

# 异常和中断

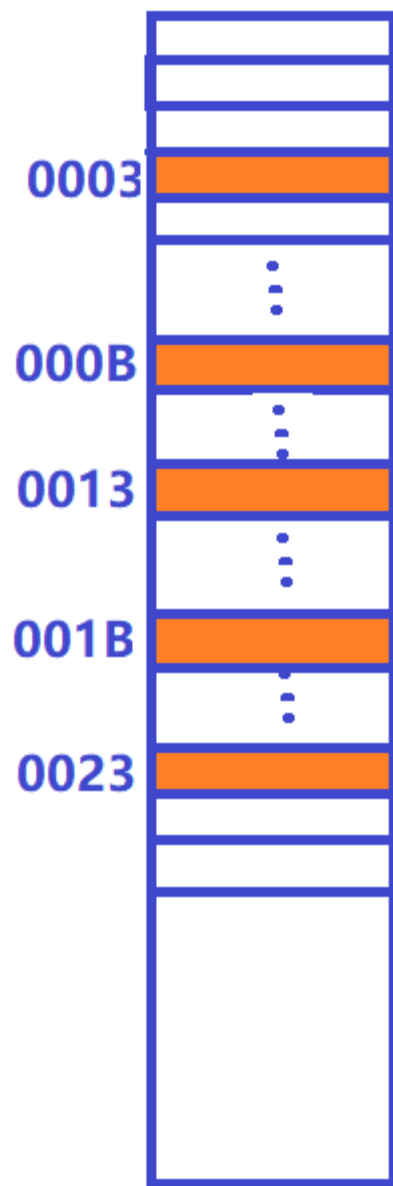
顺序执行

跳转执行

中断响应

可以预料到的事件

不能预期的事件



## ARM 体系异常分类:

- 复位异常 (Reset)
- 数据异常 (Data Abort)
- 快速中断异常 (FIQ)
- 外部中断异常 (IRQ)
- 预取异常 (Prefetch Abort)
- 软中断异常 (SWI)
- 未定义异常 (Undefined interrupt)

# 寄存器

ARM微处理器共有37个32位寄存器，其中31个为通用寄存器，6个为状态寄存器。但是这些寄存器不能被同时访问，具体哪些寄存器是可以访问的，取决ARM处理器的工作状态及具体的运行模式。但在任何时候，通用寄存器R14~R0、程序计数器PC、一个状态寄存器都是可访问的。

# 寄存器(ARM状态)

**在ARM工作状态下，任一时刻可以访问16个通用寄存器和一到两个状态寄存器。在非用户模式（特权模式）下，则可访问到特定模式分组寄存器，具体见下页图：**

# 寄存器(ARM状态)

ARM状态下的通用寄存器与程序计数器

System & User	FIQ	Supervisor	Abort	IRQ	Undefined
R0	R0	R0	R0	R0	R0
R1	R1	R1	R1	R1	R1
R2	R2	R2	R2	R2	R2
R3	R3	R3	R3	R3	R3
R4	R4	R4	R4	R4	R4
R5	R5	R5	R5	R5	R5
R6	R6	R6	R6	R6	R6
R7	R7	R7	R7	R7	R7
R8	 R8_fiq	R8	R8	R8	R8
R9	 R9_fiq	R9	R9	R9	R9
R10	 R10_fiq	R10	R10	R10	R10
R11	 R11_fiq	R11	R11	R11	R11
R12	 R12_fiq	R12	R12	R12	R12
R13	 R13_fiq	 R13_svc	 R13_abt	 R13_irq	 R13_und
R14	 R14_fiq	 R14_svc	 R14_abt	 R14_irq	 R14_und
R15 (PC)	R15 (PC)	R15 (PC)	R15 (PC)	R15 (PC)	R15 (PC)

ARM状态下的程序状态寄存器

CPSR	CPSR  SPSR_fiq	CPSR  SPSR_svc	CPSR  SPSR_abt	CPSR  SPSR_irq	CPSR  SPSR_und
------	--	--	--	--	--

 = 分组寄存器

# 通用寄存器

❖ 不分组寄存器 (The unbanked registers)

**R0-R7**

❖ 分组寄存器 (The banked registers)

**R8-R14**

❖ 程序计数器: **R15 (PC)**



# 不分组通用寄存器

**R0-R7**是不分组寄存器。这意味着在所有处理器模式下，访问的都是同一个物理寄存器。不分组寄存器没有被系统用于特别的用途，任何可采用通用寄存器的应用场合都可以使用未分组寄存器。

# 分组通用寄存器

## ❖ 分组寄存器R8-R12

1. FIQ模式分组寄存器R8-R12
2. FIQ以外的分组寄存器R8-R12

## ❖ 分组寄存器R13、R14

1. 寄存器**R13**通常用做堆栈指针SP
2. 寄存器**R14**用作子程序链接寄存器 (Link Register - LR) , 也称为LR, 指向函数的返回地址

# 程序计数器

**寄存器R15**被用作程序计数器，也称为**PC**。其值等于当前正在执行的指令的地址+8(因为在取址和执行之间多了一个译码的阶段)。

# 状态寄存器

■ CPSR

■ SPSR\_svc

■ SPSR\_abt

■ SPSR\_und

■ SPSR\_irq

■ SPSR\_fiq

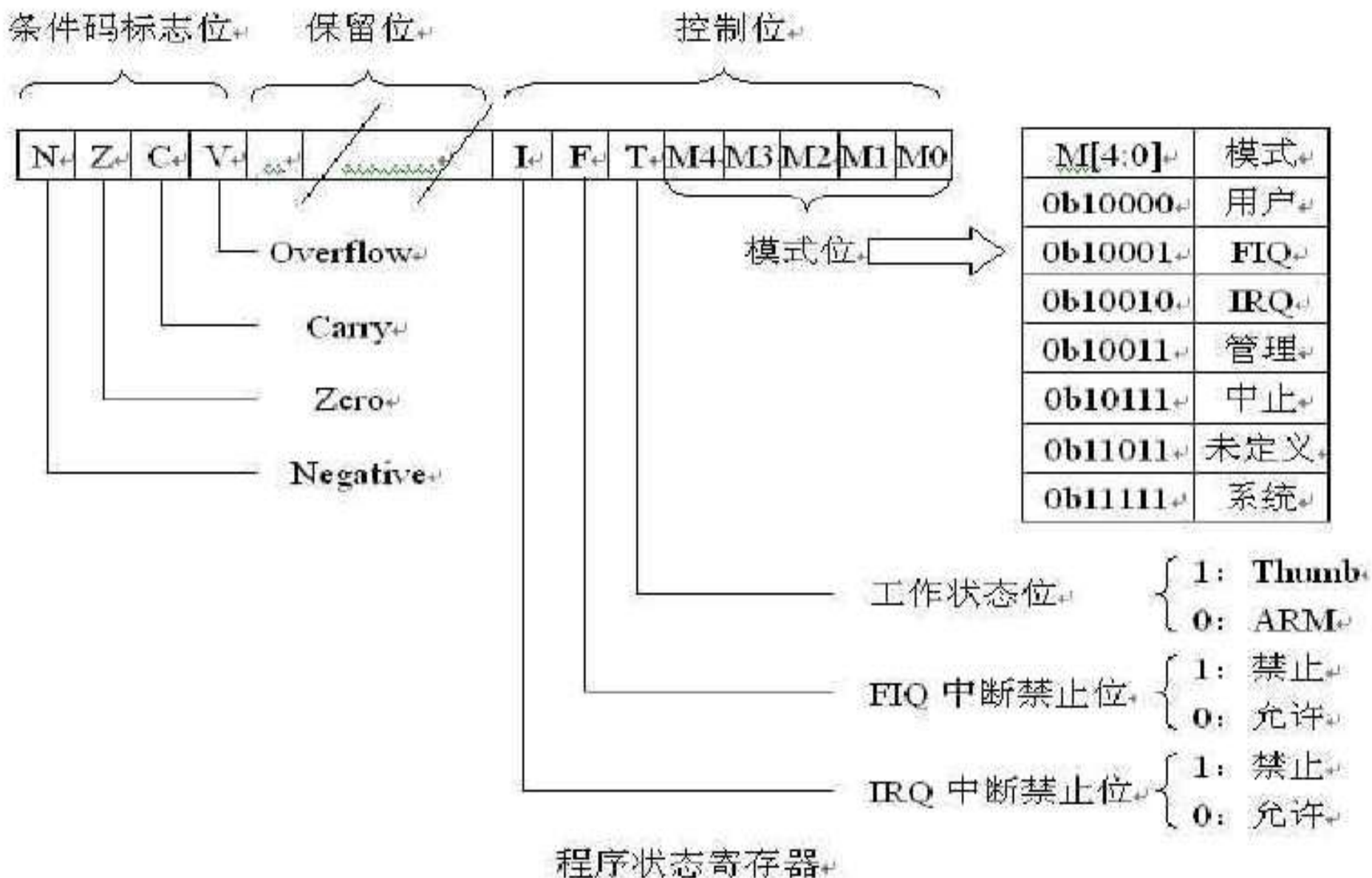
# 状态寄存器

ARM**所有工作模式**下都可以访问程序状态寄存器CPSR。CPSR包含条件码标志、中断禁止位、当前处理器模式以及其它状态和控制信息。

# 状态寄存器

CPSR在**每种异常模式**下都有一个对应的物理寄存器——程序状态保存寄存器SPSR。当异常出现时，SPSR用于保存CPSR的值，以便异常返回后恢复异常发生时的工作状态。

# CPSR/SPSR



- N: 运算结果的最高位反映在该标志位。对于有符号二进制补码，结果为负数时 $N=1$ ，结果为正数或零时 $N=0$
- Z: 指令结果为0时 $Z=1$  (通常表示比较结果“相等”)，否则 $Z=0$ ;
- C: 当进行加法运算 (包括CMN指令)，并且最高位产生进位时 $C=1$ ，否则 $C=0$ 。当进行减法运算 (包括CMP指令)，并且最高位产生借位时 $C=0$ ，否则 $C=1$ 。对于结合移位操作的非加法/减法指令，C为从最高位最后移出的值，其他指令C通常不变。



# CPSR/SPSR

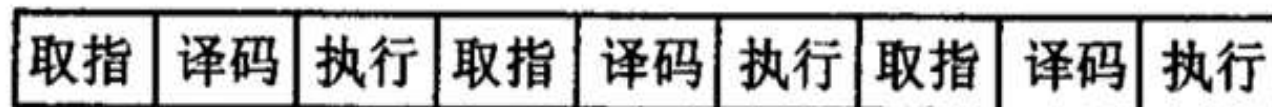
CPSR [4: 0]	模式	用途	可访问的寄存器
10000	用户	正常用户模式, 程序正常执行模式。	PC, R14~R0, CPSR
10001	FIQ	处理快速中断, 支持高速数据传送或通道处理。	PC, R14_fiq~R8_fiq, R7~R0, CPSR, SPSR_fiq
10010	IRQ	处理普通中断。	PC, R14_irq~R13_fiq, R12~R0, CPSR, SPSR_irq
10011	SVC	操作系统保护模式, 处理软件中断 (SWI)。	PC, R14_svc~R13_svc, R12~R0, CPSR, SPSR_svc
10111	中止	处理存储器故障、实现虚拟存储器和存储器保护。	PC, R14_abt~R13_abt, R12~R0, CPSR, SPSR_abt
11011	未定义	处理未定义的指令陷阱, 支持硬件协处理器的软件仿真。	PC, R14_und~R13_und, R12~R0, CPSR, SPSR_und
11111	系统	运行特权操作系统任务。	PC, R14~R0, CPSR

51系列顺序执行

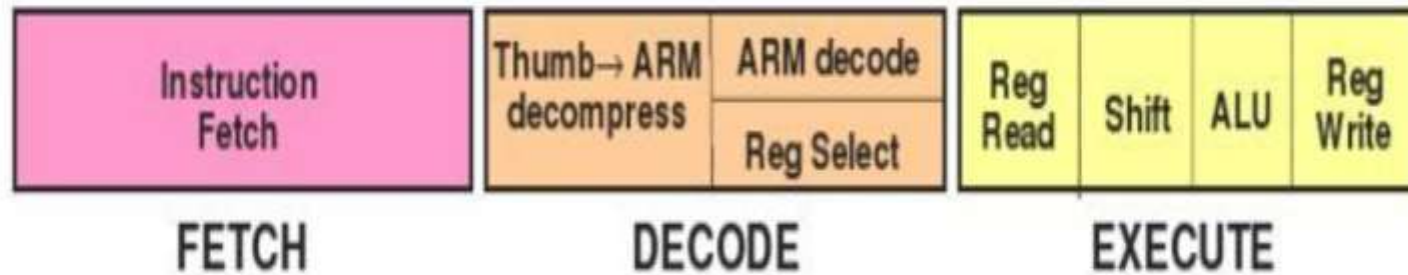
指令1

指令2

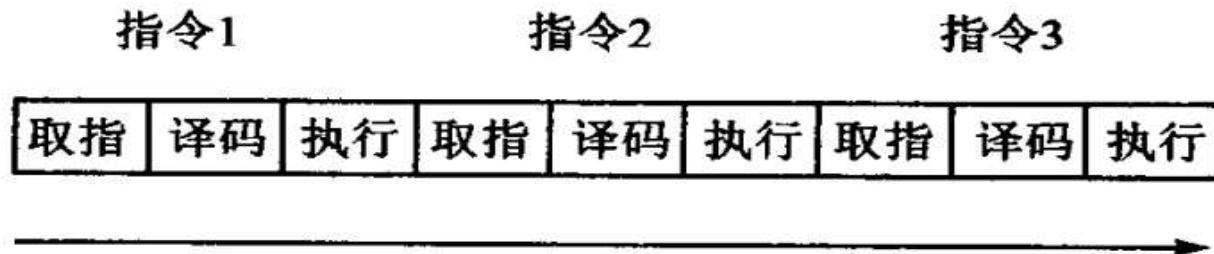
指令3



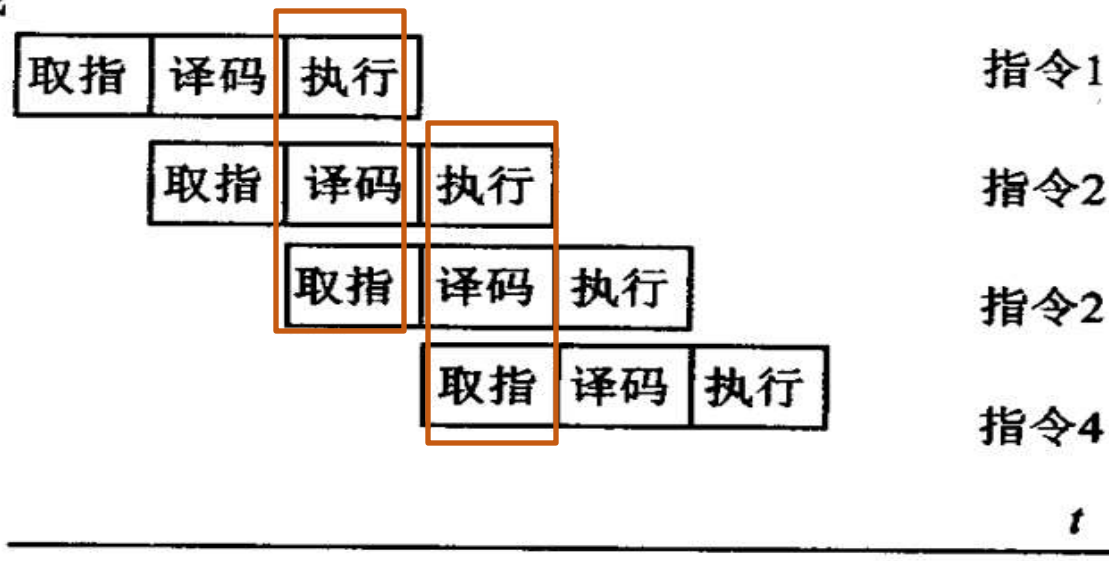
### The ARM7TDMI core and ARM7TDMI-S core pipeline



51系列顺序执行



ARM7的3级流水线



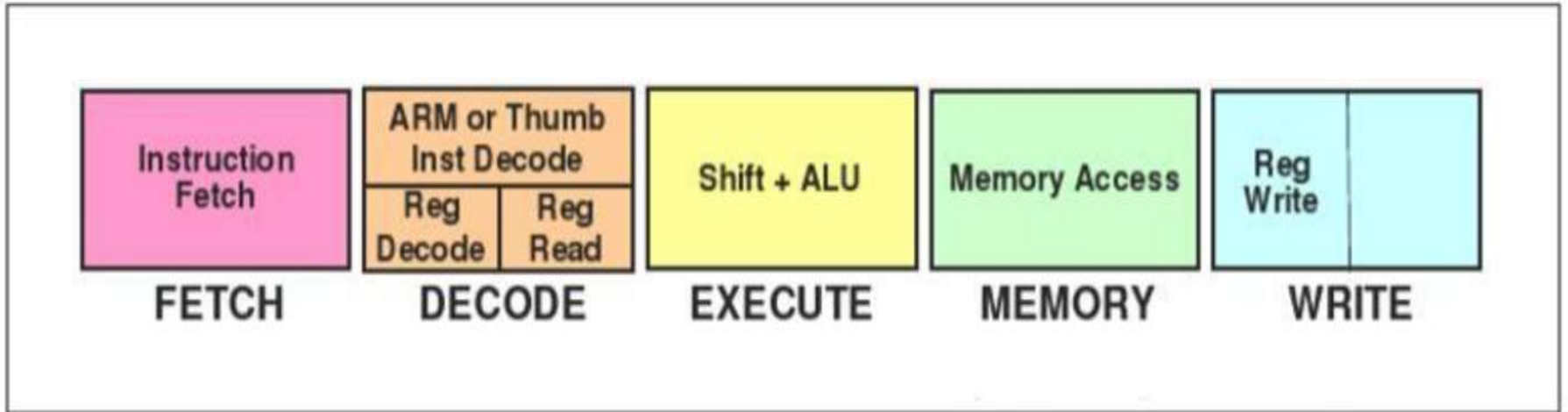
人们习惯把当前正在执行的指令作为参考点，但是，R15,也就是PC总是指向正在取指的指令。那么，**当前正在执行的指令的地址和PC值有什么关系呢？**

$$PC = \text{当前执行指令地址} + 8$$

PC不是指向正在运行的指令，而是---

**PC始终指向你要取得指令的地址。**

### The ARM9TDMI core pipeline



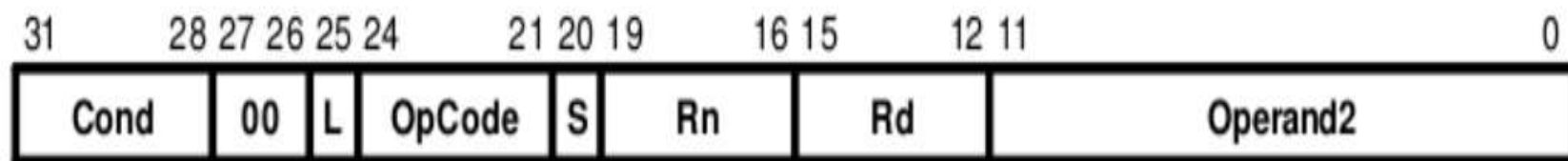
相比ARM7，ARM9采用了更高效的五级流水线设计，在取指令、译码、执行之后，又增加了LS1和LS2阶段，LS1负责加载和存储指令中制定的数据，LS2则负责提取、符号扩展，通过字节或半字加载命令来加载数据，但是LS1和LS2仅对加载（LDR）和存储命令(STR)有效，其他的指令是不需要执行这两个阶段的。



ARM9 的五级流水线

PC=当前执行指令地址+8 ?

# MOV



**op2**是占了**12**位，其中**bit11-bit8**是移位数(**rotate**),

**bit7-0**是一个**8**位的立即数(**imm**),

我们写指令的时候，也别难为汇编器，汇编器也要想办法按照 **$\text{imm} \gg (\text{rotate} * 2)$** 的方法，把你写的立即数变成这种格式。