$@: 目标的名字

$^: 构造所需文件列表所有所有文件的名字

$<: 构造所需文件列表的第一个文件的名字

$?: 构造所需文件列表中更新过的文件

输出的文件格式

反汇编所有代码段

```
uart.bin: start.o clock.o uart.o main.o
```

目标文件　依赖文件　代码段链接地址

```
arm-linux-ld -Ttext 0xD0020010 -o uart.elf $^
```

$^所有依赖文件

将所有依赖文件链接成elf文件，–Ttext 0xd0020010告诉连接器连接成的文件加载到RAM的0xd0020010处执行

链接工具

复制文件内容到另一文件

```
arm-linux-objcopy -O binary uart.elf $@
```

二进制　$@目标文件

将调试生成的uart.elf可执行文件转换成二进制文件uart .bin

反汇编工具

```
arm-linux-objdump -D uart.elf > uart.dis
```

对调试生成的elf文件进行反汇编，用于调试错误，生成.dis文件

```
%.o: %.c
```

目标生成.o文件，每个.o文件依赖于对应的.c文件

```
arm-linux-gcc -c $< -o $@ -nostdlib
```

–c参数将对源程序.c进行预处理、编译、汇编操作，生成.o文件

将目录下所有源程序.c文件预处理生成.o文件

-o参数是指定输出格式为.o文件

```
%.o:%.S
```

目标生成.o文件，每个.o文件依赖于对应的.S文件

$<：目标文件对应依赖文件名字

```
arm-linux-gcc -c $< -o $@ -nostdlib
```

$@：目标文件名字

将目录下所有源程序.s文件预处理生成.o文件

-nostdlib: 不连接系统标准启动文件和标准库文件，只把指定文件传递给链接器

```
clean:
    rm *.o *.elf *.bin *.dis
```

清除所有.o .elf .dis .bin文件

-g	可执行程序包含调试信息，目的是为了给 GDB 工具调试程序使用
-o	指定输出文件名，如不指定 -o，默认输出文件名为 a.out
-c	只编译不链接，产生 .o 文件，不产生可执行文件
-C	将当前的工作目录切换到指定目录中
-f	编译用其它文件名书写的 Makefile
-D	执行 gcc 过程中给程序中添加宏定义
-Wall	编译后显示所有警告
-L	指定链接的第三方库所在的目录
-nostdlib	不连接系统标准启动文件和标准库文件，只把指定的文件传递给链接器。


```
%.o: %.c
```

等同于下面的写法。

```
f1.o: f1.c
f2.o: f2.c
```

使用匹配符%，可以将大量同类型的文件，只用一条规则就完成构建。看到这你可能一头雾水，到底怎么用别急，往下看。