Day2

Phase 1

汇编入门

指令	功能
`org`	设定程序在内存中的装载位置
`mov`	将第二参数所指定的值赋给第一个参数所指定的位置（可能是寄存器，也有可能是内存中的位置）
`add`	将第一参数与第二参数相加，结果存入第一参数
`cmp`	比较第一参数与第二参数，结果送入pswr
`je`	根据pswr中的标志位（也就是比较结果是否是相等），来进行跳转（不相等则不跳转）
`int`	触发软中断，调用中断服务程序
`hlt`	停机，使cpu进入低功耗状态

PS：`mov` 指令可以使用 `[xxx]` 来指示一个内存单元。当 `xxx` 是寄存器时，只能使用BX、BP、SI、DI。

PPS：`mov` 指令源数据与目的数据地址的位数必须相同，指示内存地址的格式时 `数据大小 [xxx]`

常用16位寄存器

代号	名字
AX	累加寄存器
CX	计数寄存器
DX	数据寄存器
BX	基址寄存器
SP	栈指针寄存器
BP	基址指针寄存器
SI	源变址寄存器
DI	目的变址寄存器

以上寄存器大部分都在上学期的计算机组成原理中学习过相关的知识

其中ACDB可以通过将X改为L或H来访问他们的高八位或者低八位数据

16位段寄存器有ES、CS、SS、DS、FS、GS，功能暂时不明确。

表中的寄存器加前缀E就使32位寄存器，注意访问16位寄存器访问的就是低16位。

Phase 2

分析代码

书上提供了显示一个字符的方法

```
mov ah, 0x0e
mov al, <字符>
mov bh, 0
mov bl, <color code>
int 0x10
```

截取一个部分源码

```
entry:
        MOV     AX,0            ; 初始化寄存器
        MOV     SS,AX
        MOV     SP,0x7c00
        MOV     DS,AX
        MOV     ES,AX

        MOV     SI,msg
putloop:
        MOV     AL,[SI]
        ADD     SI,1            ; 给SI加1
        CMP     AL,0
        JE      fin
        MOV     AH,0x0e         ; 显示一个文字
        MOV     BX,15           ; 指定字符颜色
        INT     0x10            ; 调用显卡BIOS
        JMP     putloop
fin:
        HLT                     ; 让CPU停止，等待指令
        JMP     fin             ; 无限循环

msg:
        DB      0x0a, 0x0a      ; 换行两次
        DB      "hello, world"
        DB      0x0a            ; 换行
        DB      0

        RESB    0x7dfe-$        ; 填写0x00直到0x001fe

        DB      0x55, 0xaa
```

注意到`putloop`从`msg`位置开始一个一个的输出字符，直到遇见0。

注意ascii码0x0a = 10，对应的时'\r'即换行。

注意这部分源码在经过nask编译之后并不符合软盘镜像的格式要求，需要使用`edimg`工具将他补全成一个软盘镜像。

按照作者的指导，nask，edimg之后所得到的镜像与day 1的`helloos.img`内容相同。

Phase 3

理解IPL概念。

机器启动后只会运行内存地址0x7c00-0x7dff的指令，所以我们把我们的系统引导程序装在到内存的这个位置，来让系统运行起来。而`org`命令，则指定了向内存中装载的地址。

Phase 4

makefile入门

在学习这门课之前，我早就学习过makefile的使用与编写。当我们准备以源码方式安装一个运行于`*nix`的应用程序时，常见的操作时这样的

```
$ ./configure
$ make
$ make install
```

这是我与makefile的first contact。

`./configure`根据系统特征自动生成makefile。而`make`则编译源码成为可以在本系统上执行的二进制文件，`make install`则将编译好的程序安装到系统当中（例如向`bin`文件夹中添加软链接等操作）

与makefile更为深入的接触是学习使用vim的时候，了解到了vim的`:make`命令原理，是通过调用make来实现的。想要使用make，就必须为项目创建makefile。

makefile的基础格式是

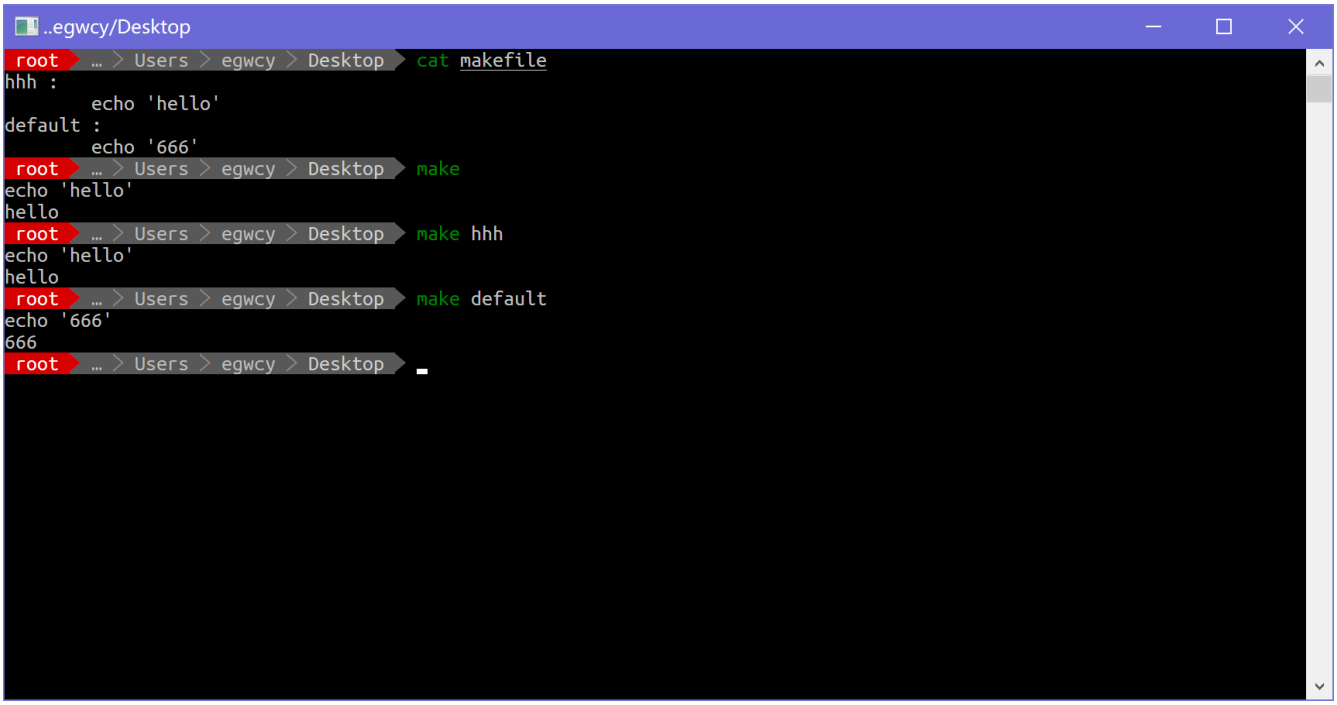
```
target ... : prerequisites ...
    command
```

当执行`make target`的时候，make工具会检查prerequisites是否有更新、target（如果是个文件）是否存在。

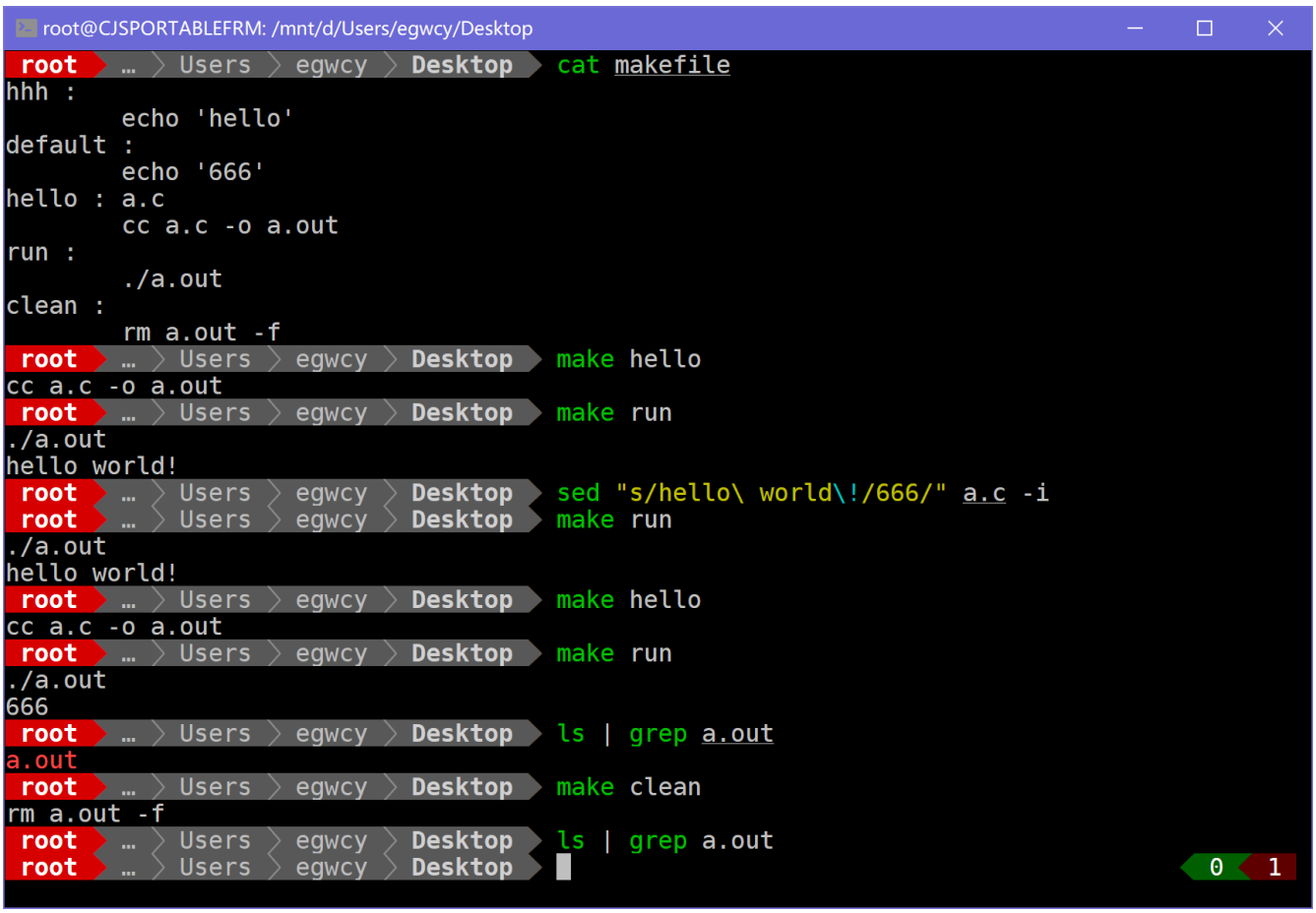
满足条件时，make将会执行一系列command。

有个地方必须注意：command前必须是一个`'\t'`，不可以是其他格式的缩进。这似乎是一个历史原因。

`make`命令后不跟参数的话，将会默认执行`make file`中的第一个命令。



我们尝试使用makefile来编译一个hello world程序吧



Windows下的makefile编制是大同小异的

Phase 5

关于第二天最后的那一页谈话的理解。指令也是数据。如果尝试执行一段不是指令的数据，机器会尝试以指令的方式来理解数据，造成的后果无法预料，而由于程序（指令序列）也是数据，显示这些数据对机器来说并不会产生什么问题，只是人可能无法阅读而已。

补充一小点

作者光盘提供的工具中的`asm.bat`生成了带日语注释的编译过程说明，我阅读了以`ips.nas`为源码生成的`ipl.lst`，截取一部分。（经过处理，删除了一些影响排版的注释）

```
     1 00000000                                 ; hello-os
     2 00000000                                 ; TAB=4
     3 00000000
     4                                                  ORG     0x7c00
     5 00007C00
     6 00007C00
     7 00007C00
     8 00007C00 EB 4E                                   JMP     entry
     9 00007C02 90                                      DB      0x90
    10 00007C03 48 45 4C 4C 4F 49 50 4C                 DB      "HELLOIPL"
    11 00007C0B 0200                                    DW      512
    12 00007C0D 01                                      DB      1
    13 00007C0E 0001                                    DW      1
    14 00007C10 02                                      DB      2
    15 00007C11 00E0                                    DW      224
    16 00007C13 0B40                                    DW      2880
    17 00007C15 F0                                      DB      0xf0
    18 00007C16 0009                                    DW      9
    19 00007C18 0012                                    DW      18
    20 00007C1A 0002                                    DW      2
    21 00007C1C 00000000                                DD      0
    22 00007C20 00000B40                                DD      2880
    23 00007C24 00 00 29                                DB      0,0,0x29
    24 00007C27 FFFFFFFF                                DD      0xffffffff
    25 00007C2B 48 45 4C 4C 4F 2D 4F 53 20 20           DB      "HELLO-OS   "
       00007C35 20
    26 00007C36 46 41 54 31 32 20 20 20                 DB      "FAT12   "
    27 00007C3E 00 00 00 00 00 00 00 00 00 00           RESB    18
       00007C48 00 00 00 00 00 00 00 00
    28 00007C50
    29 00007C50                                 ; プログラム本体
    30 00007C50
    31 00007C50                                 entry:
    32 00007C50 B8 0000                                 MOV     AX,0            ; レジスタ初
期化
    33 00007C53 8E D0                                   MOV     SS,AX
    34 00007C55 BC 7C00                                 MOV     SP,0x7c00
    35 00007C58 8E D8                                   MOV     DS,AX
    36 00007C5A 8E C0                                   MOV     ES,AX
    37 00007C5C
    38 00007C5C BE 7C74                                 MOV     SI,msg
    39 00007C5F                                 putloop:
    40 00007C5F 8A 04                                   MOV     AL,[SI]
```

```
    41 00007C61 83 C6 01                                ADD     SI,1            ; SIに1を足
す
    42 00007C64 3C 00                                   CMP     AL,0
    43 00007C66 74 09                                   JE      fin
    44 00007C68 B4 0E                                   MOV     AH,0x0e
    45 00007C6A BB 000F                                 MOV     BX,15           ; カラーコー
ド
    46 00007C6D CD 10                                   INT     0x10
    47 00007C6F EB EE                                   JMP     putloop
    48 00007C71                                 fin:
    49 00007C71 F4                                      HLT
    50 00007C72 EB FD                                   JMP     fin             ; 無限ループ
    51 00007C74
    52 00007C74                                 msg:
    53 00007C74 0A 0A                                   DB      0x0a, 0x0a      ; 改行を2つ
    54 00007C76 68 65 6C 6C 6F 2C 20 77 6F 72           DB      "hello, world"
       00007C80 6C 64
    55 00007C82 0A                                      DB      0x0a            ; 改行
    56 00007C83 00                                      DB      0
    57 00007C84
    58 00007C84 00 00 00 00 00 00 00 00 00 00           RESB    0x7dfe-$        ;
```

发现了一些小点

```
00007C00 EB 4E                                  JMP     entry
```

0x7c00位置是语句JMP entry。仔细看了一下，`EB`应该是`JMP`指令的操作码，而4E正好是这个2字节指令的下一个位置0x7c02到entry也就是0x7c50的差值，也就是说，`JMP`指令使用的是相对寻址方式。后面还有`JE`指令，也有相同的性质。