A guide to game design

- Booting
- The game design
- Debugging

- Introduction to Simics
  - by Zhonghan Cheng

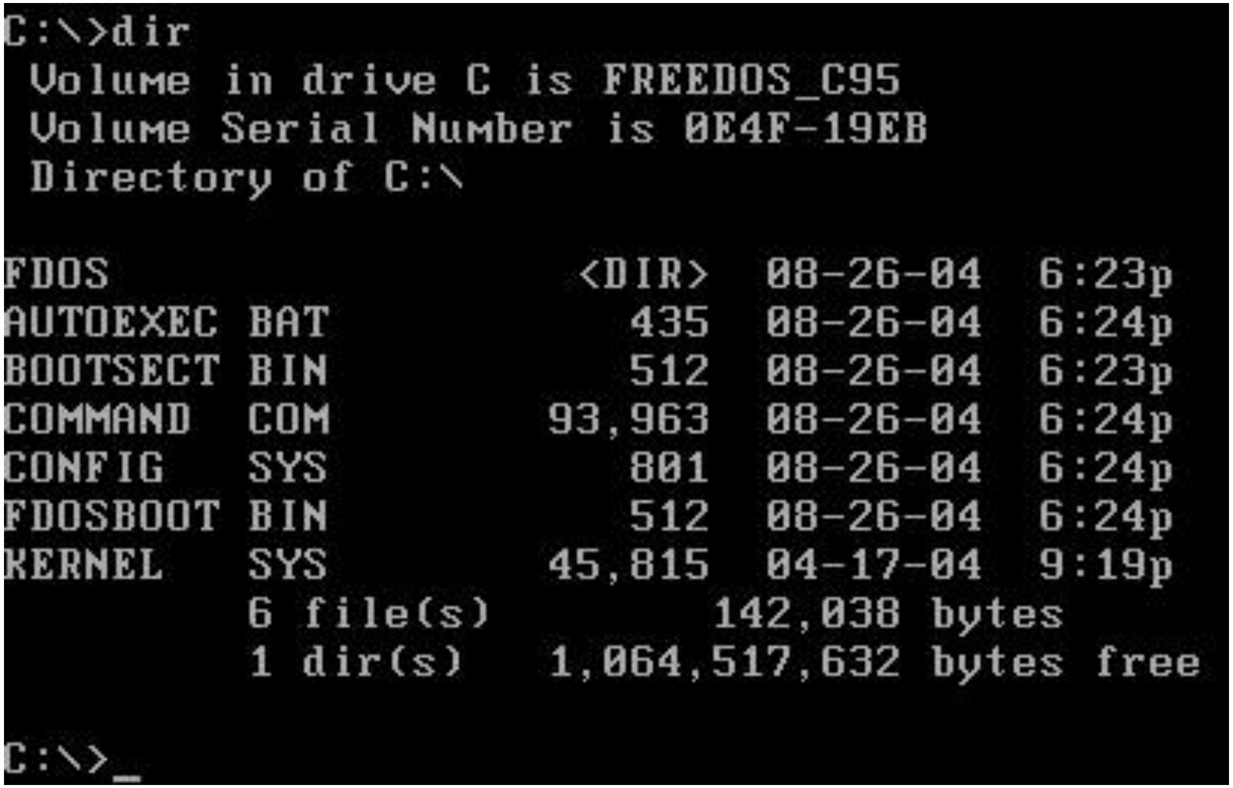
- Booting
  - recap - 8086 & 80386
  - BIOS booting
  - the bigger world

Real mode

- real address mode
- the only operating mode on 8086
- “direct” access for all the memory
- no protection, multitasking, or privilege level

Real mode (cont.)

- DOS runs in real mode



```
C:\>dir
 Volume in drive C is FREEDOS_C95
 Volume Serial Number is 0E4F-19EB
 Directory of C:\

FDOS                 <DIR>  08-26-04  6:23p
AUTOEXEC BAT           435  08-26-04  6:24p
BOOTSECT BIN           512  08-26-04  6:23p
COMMAND  COM        93,963  08-26-04  6:24p
CONFIG   SYS           801  08-26-04  6:24p
FDOSBOOT BIN           512  08-26-04  6:24p
KERNEL   SYS        45,815  04-17-04  9:19p
         6 file(s)         142,038 bytes
         1 dir(s)    1,064,517,632 bytes free

C:\>_
```

1MB addressing

- registers are 16-bit
- but 8086 has 20 address lines, supporting 2^{20}B = 1MB memory addressing
- first introduce segmentation:

  PA = (segment_base << 4) + offset

1MB addressing (cont.)

- instruction fetch:

  PA = (CS << 4) + IP

- date accessing:

  PA = (DS << 4) + AX/BX/CX/DX/SI/DI

- stack operation:

  PA = (SS << 4) + SP

- Why not "PA = (CS << 16) + IP" for 32-bit addressing?

Interrupt

- interrupt vector table
  - established by BIOS
  - starting from memory address 0
  - 256 entries, 4 bytes each
  - each entry specifies the address of the corresponding interrupt handler (CS:IP)

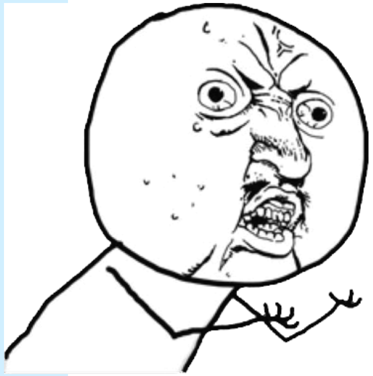
In real mode

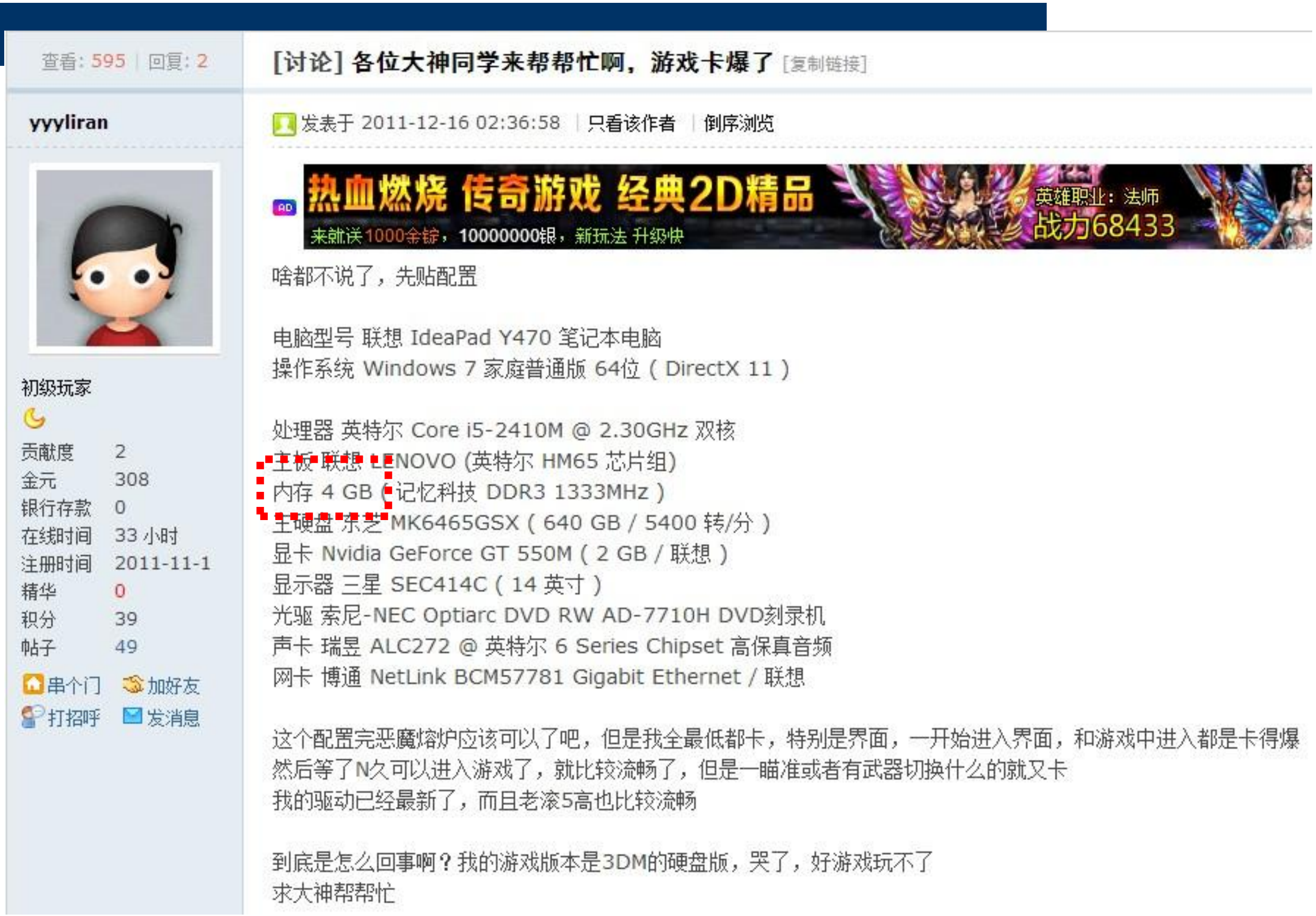
- we can do a lot of things
- 仙剑奇侠传 DOS版

李大娘：
少跟老娘鬼扯淡

1MB is enough?

- today





查看：595 | 回复：2

[讨论] 各位大神同学来帮帮忙啊，游戏卡爆了 [复制链接]

yyyliran

发表于 2011-12-16 02:36:58 | 只看该作者 | 倒序浏览

初级玩家

贡献度	2
金元	308
银行存款	0
在线时间	33 小时
注册时间	2011-11-1
精华	0
积分	39
帖子	49

串个门 加好友
打招呼 发消息

啥都不说了，先贴配置

电脑型号 联想 IdeaPad Y470 笔记本电脑
操作系统 Windows 7 家庭普通版 64位 (DirectX 11)

处理器 英特尔 Core i5-2410M @ 2.30GHz 双核
主板 联想 LENOVO (英特尔 HM65 芯片组)
内存 4 GB (记忆科技 DDR3 1333MHz)
主硬盘 东芝 MK6465GSX (640 GB / 5400 转/分)
显卡 Nvidia GeForce GT 550M (2 GB / 联想)
显示器 三星 SEC414C (14 英寸)
光驱 索尼-NEC Optiarc DVD RW AD-7710H DVD刻录机
声卡 瑞昱 ALC272 @ 英特尔 6 Series Chipset 高保真音频
网卡 博通 NetLink BCM57781 Gigabit Ethernet / 联想

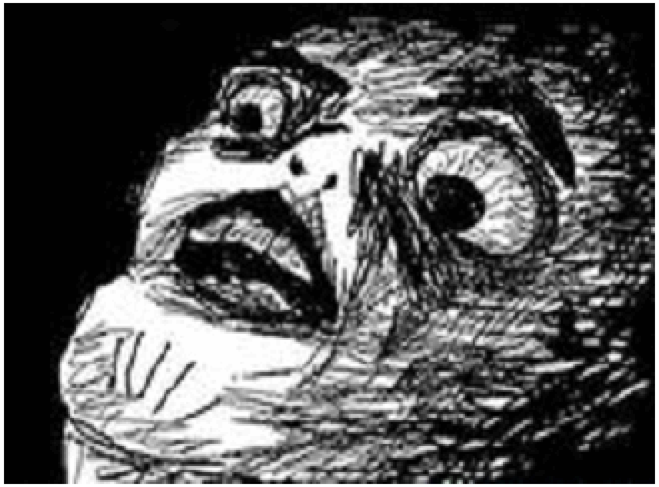
这个配置完恶魔熔炉应该可以了吧，但是我全最低都卡，特别是界面，一开始进入界面，和游戏中进入都是卡得爆
然后等了N久可以进入游戏了，就比较流畅了，但是一瞄准或者有武器切换什么的就又卡
我的驱动已经最新了，而且老滚5高也比较流畅

到底是怎么回事啊？我的游戏版本是3DM的硬盘版，哭了，好游戏玩不了
求大神帮帮忙

1MB is enough? (cont.)

- at that time

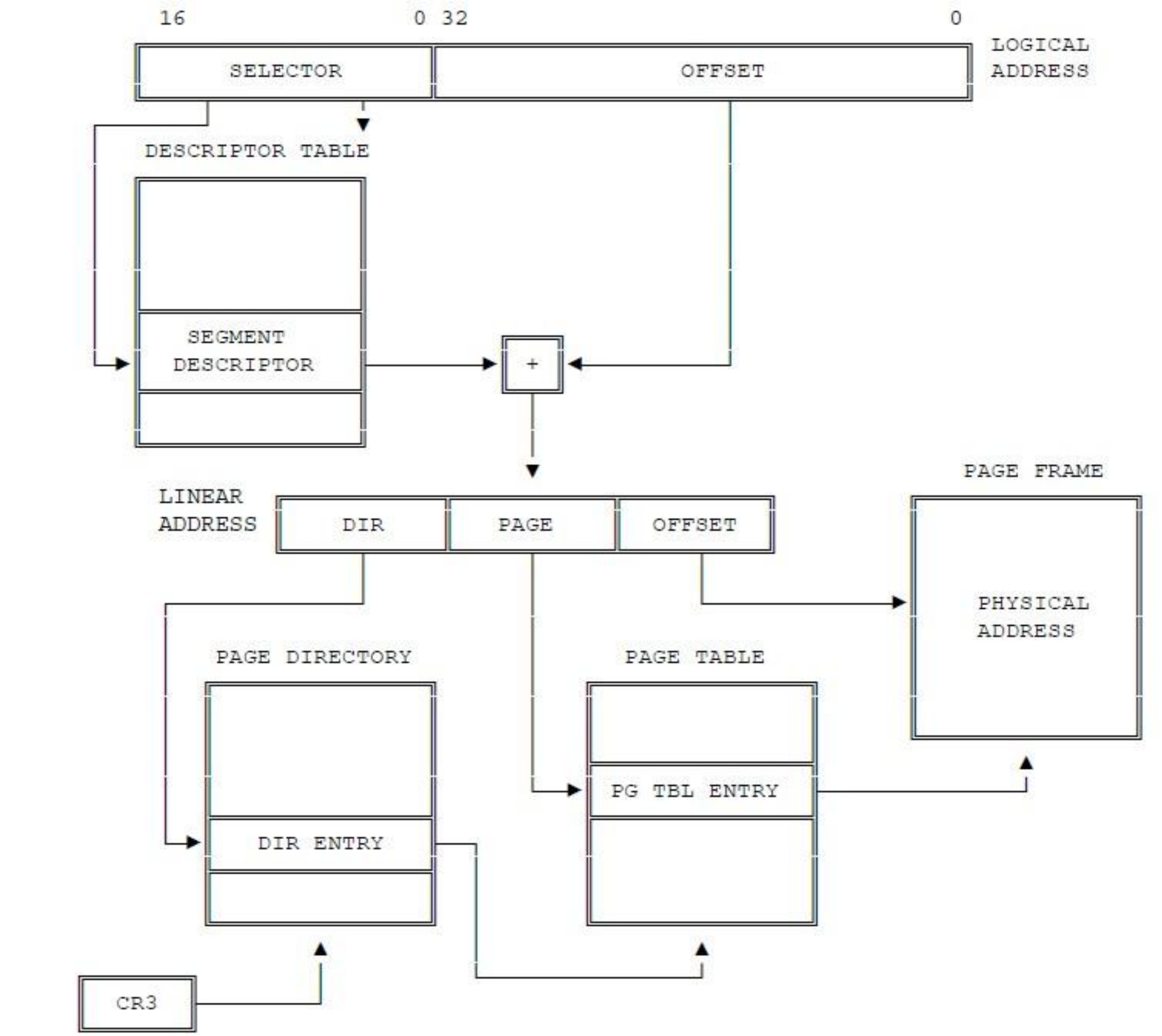
640K ought to be enough for anybody



Protected mode

- an operating mode provided by 80386
- Goal
  - 4GB addressing
  - protection
  - multitasking

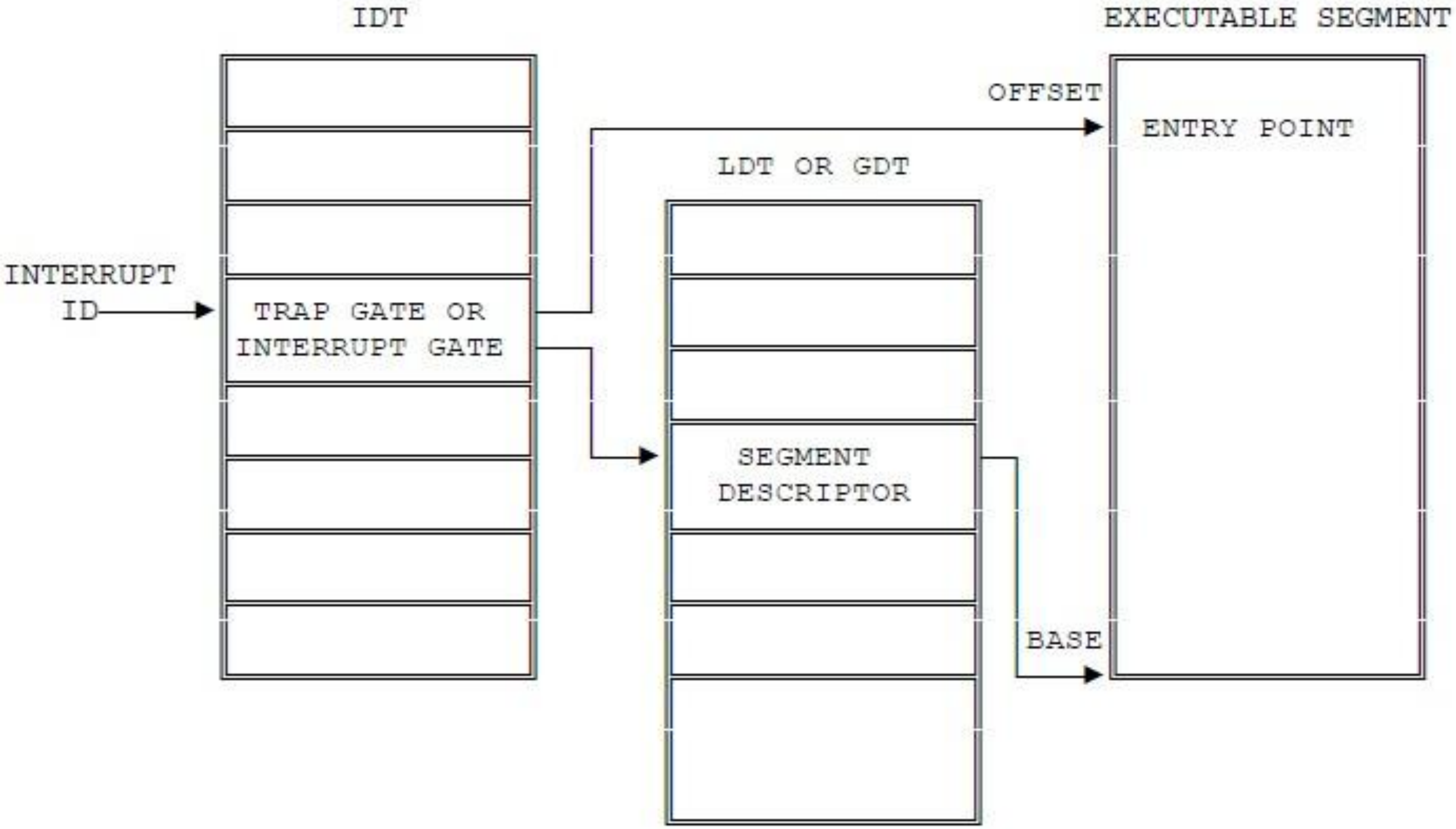
Recap – memory management



Interrupt

- Interrupts need protection, too.
  - User process cannot generate software interrupts casually to fool CPU.
- In 80386, interrupt handlers are called through gate descriptors.

Recap – Interrupt indexing



Furthermore

- virtual 8086 mode
- TSS for multitasking
- …

- We do not use them in Lab0.
- For details, refer to i386 manual if necessary.

- Booting
  - recap - 8086 & 80386
  - BIOS booting
  - the bigger world

BIOS

- Basic Input/Output System

BIOS (cont.)

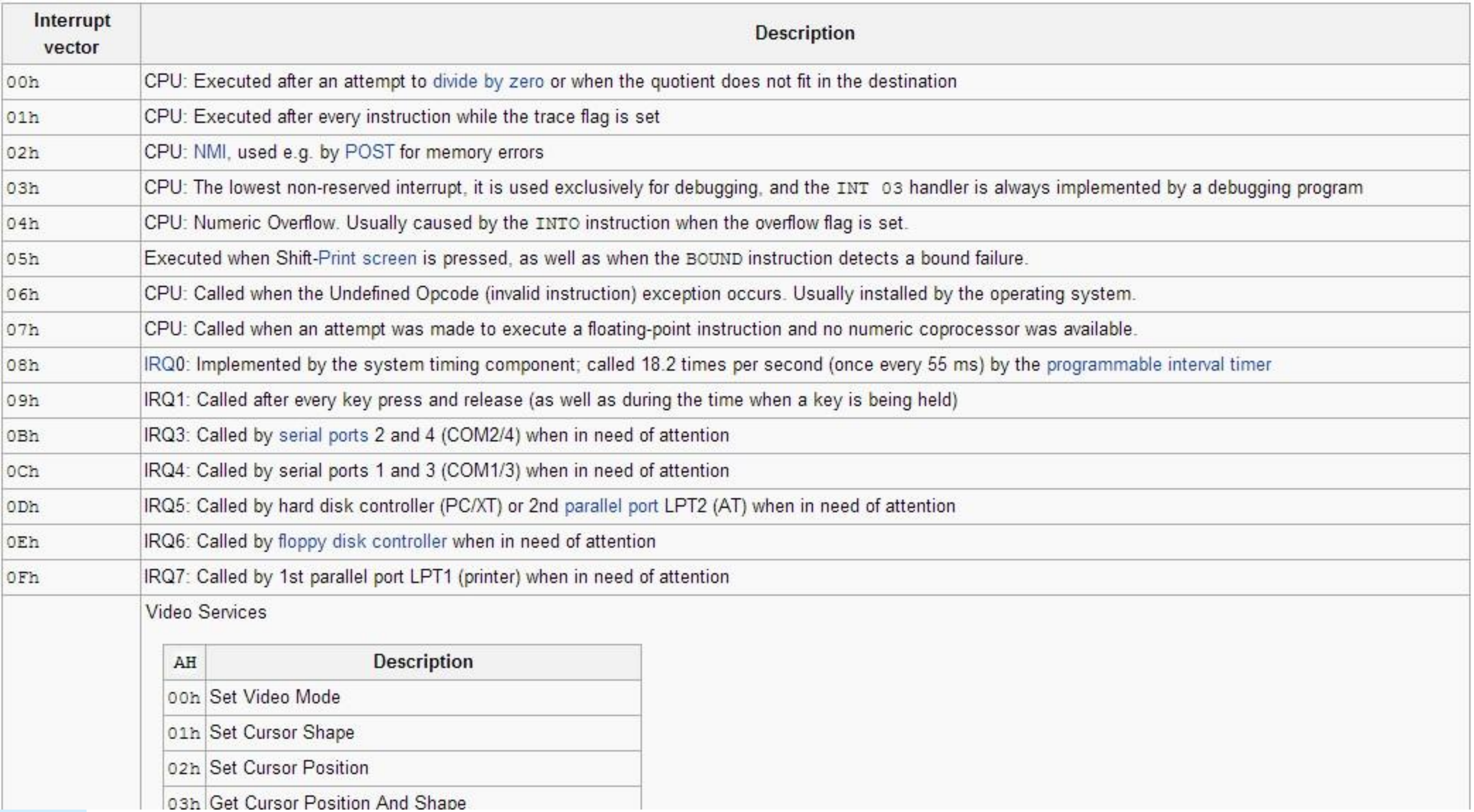
- BIOS provides
  - BIOS interrupts
  - CMOS configuration
  - power-on self-test (POST)
  - boot loader

BIOS interrupts

- Provides basic hardware control and IRQ handlers
  - print a character
  - read/write the disk
  - exceptions

BIOS interrupts (cont.)

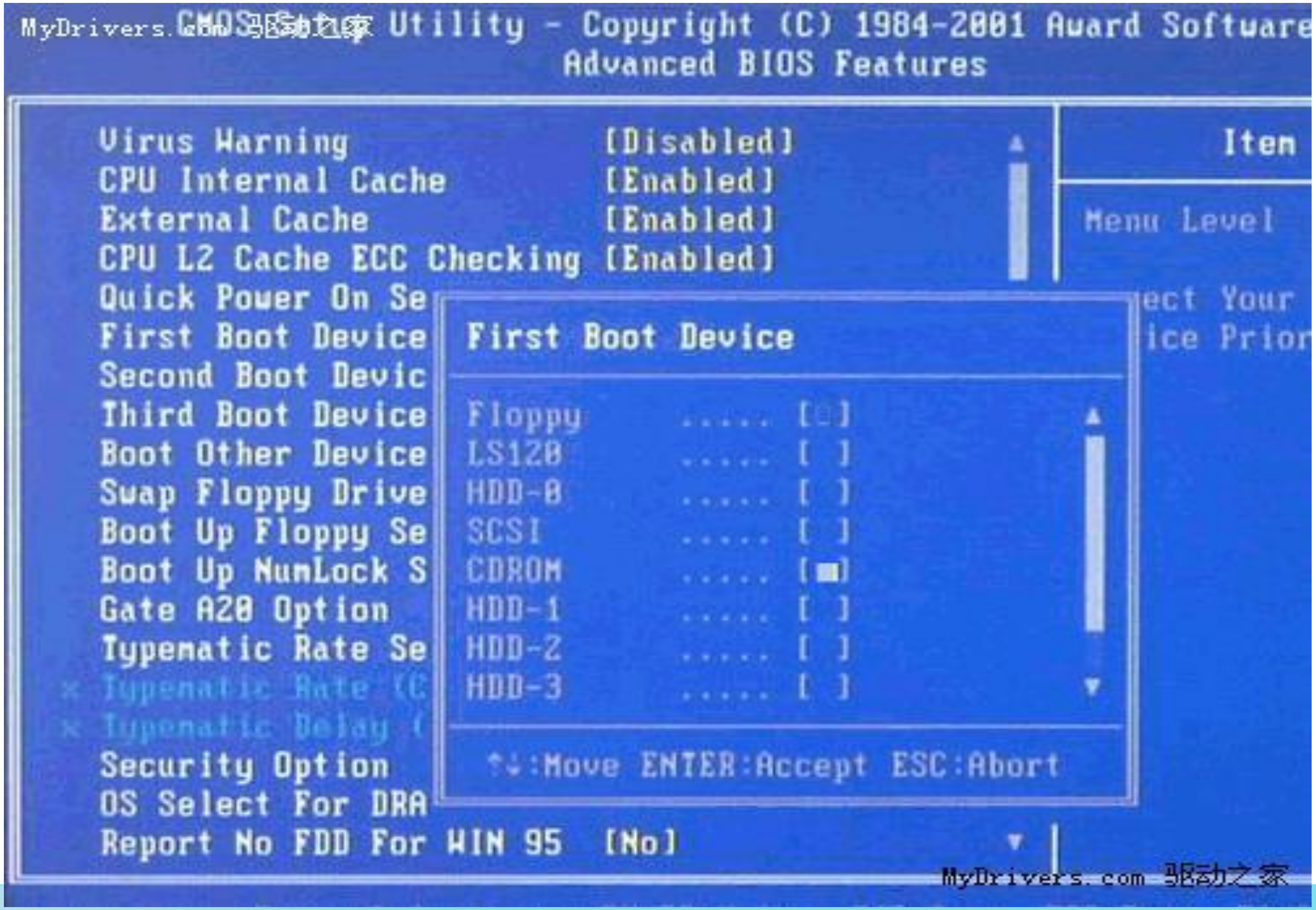
http://en.wikipedia.org/wiki/BIOS_interrupt_call



Interrupt vector	Description
00h	CPU: Executed after an attempt to divide by zero or when the quotient does not fit in the destination
01h	CPU: Executed after every instruction while the trace flag is set
02h	CPU: NMI, used e.g. by POST for memory errors
03h	CPU: The lowest non-reserved interrupt, it is used exclusively for debugging, and the `INT 03` handler is always implemented by a debugging program
04h	CPU: Numeric Overflow. Usually caused by the `INTO` instruction when the overflow flag is set.
05h	Executed when Shift-Print screen is pressed, as well as when the `BOUND` instruction detects a bound failure.
06h	CPU: Called when the Undefined Opcode (invalid instruction) exception occurs. Usually installed by the operating system.
07h	CPU: Called when an attempt was made to execute a floating-point instruction and no numeric coprocessor was available.
08h	IRQ0: Implemented by the system timing component; called 18.2 times per second (once every 55 ms) by the programmable interval timer
09h	IRQ1: Called after every key press and release (as well as during the time when a key is being held)
0Bh	IRQ3: Called by serial ports 2 and 4 (COM2/4) when in need of attention
0Ch	IRQ4: Called by serial ports 1 and 3 (COM1/3) when in need of attention
0Dh	IRQ5: Called by hard disk controller (PC/XT) or 2nd parallel port LPT2 (AT) when in need of attention
0Eh	IRQ6: Called by floppy disk controller when in need of attention
0Fh	IRQ7: Called by 1st parallel port LPT1 (printer) when in need of attention
	Video Services

AH	Description
00h	Set Video Mode
01h	Set Cursor Shape
02h	Set Cursor Position
03h	Get Cursor Position And Shape

CMOS configuration



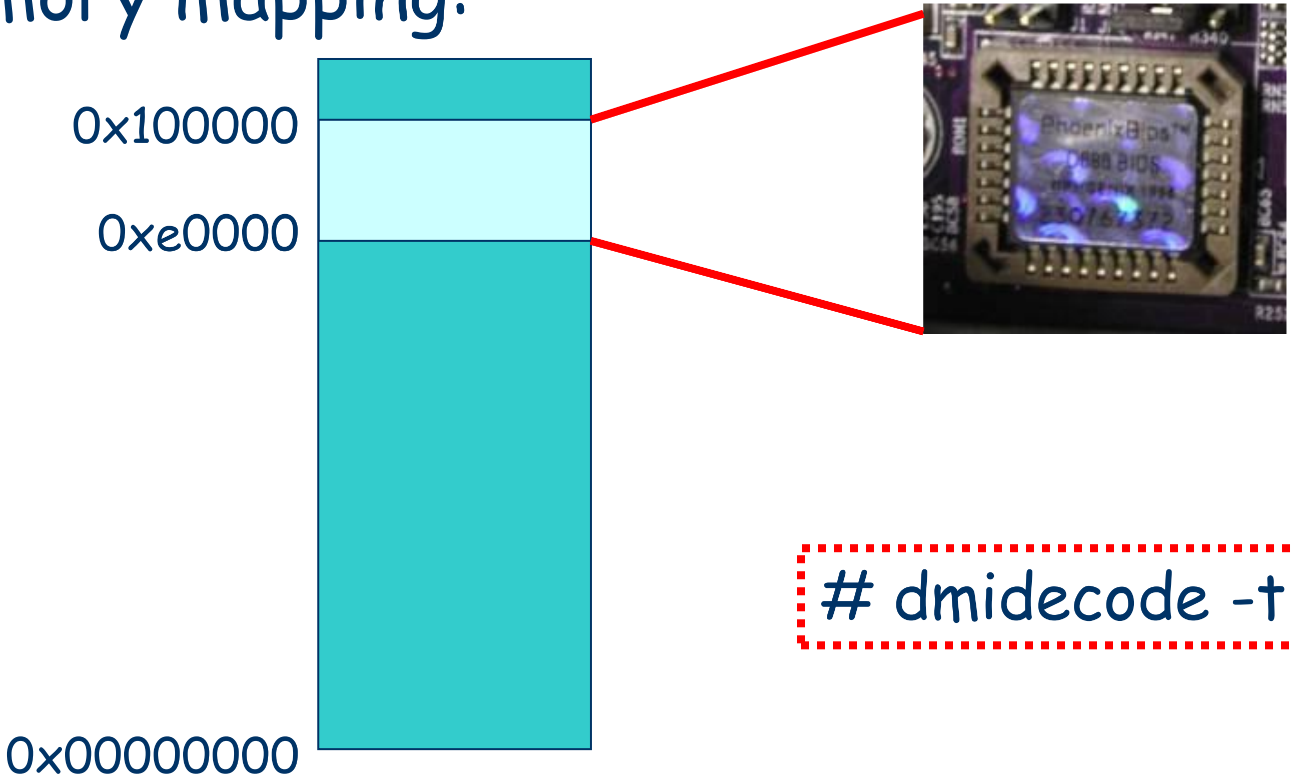
```
CMOS Setup Utility - Copyright (C) 1984-2001 Award Software
                  Advanced BIOS Features

  Virus Warning                [Disabled]      ▲ |    Item
  CPU Internal Cache           [Enabled]         |
  External Cache               [Enabled]         | Menu Level
  CPU L2 Cache ECC Checking    [Enabled]         |
  Quick Power On Se                                ect Your
  First Boot Device  First Boot Device             ice Prior
  Second Boot Devic
  Third Boot Device  Floppy     .....  [□]    ▲
  Boot Other Device  LS120      .....  [ ]
  Swap Floppy Drive  HDD-0      .....  [ ]
  Boot Up Floppy Se  SCSI       .....  [ ]
  Boot Up NumLock S  CDROM      .....  [■]
  Gate A20 Option    HDD-1      .....  [ ]
  Typematic Rate Se  HDD-2      .....  [ ]
x Typematic Rate (C  HDD-3      .....  [ ]    ▼
x Typematic Delay (
  Security Option     ↑↓:Move ENTER:Accept ESC:Abort
  OS Select For DRA
  Report No FDD For WIN 95  [No]               ▼
```

BIOS boot process

- When power on, BIOS works in real mode, with CS = 0xf000, IP = 0xfff0 initially
- PA = (CS << 4) + IP = 0xffff0
  - points to the first instruction
- Why in real mode, but not protected mode?
- Does the main memory (RAM) contain any effective instructions initially?

BIOS boot process (cont.)

- memory mapping:



0x100000

0xe0000

0x00000000

```
# dmidecode -t bios
```

BIOS boot process (cont.)

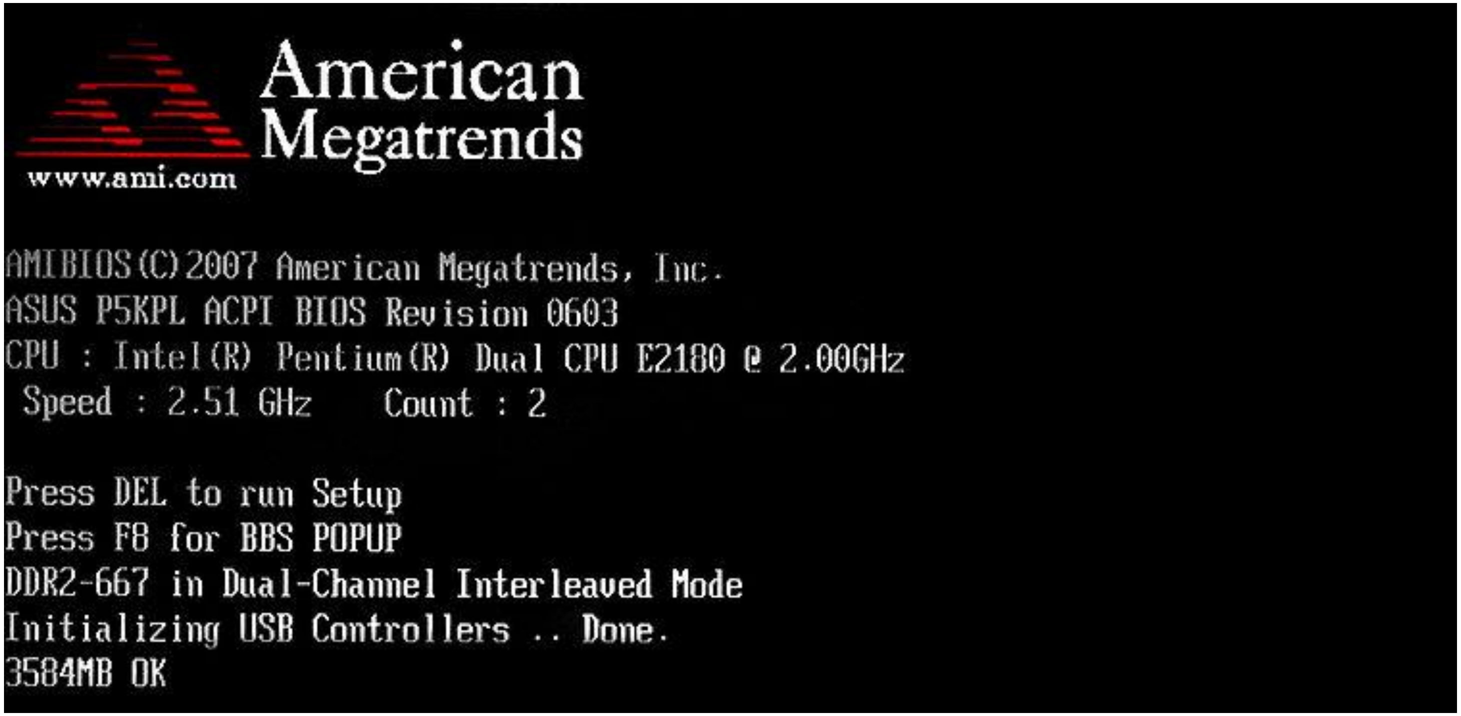
- POST
- Find the “boot device”

BIOS boot process (cont.)

- POST
  - test, identify, and initialize basic hardware
  - create interrupt vector table
  - if error happens, stop and print the error message
    - some BIOS cannot boot without a keyboard

BIOS boot process (cont.)

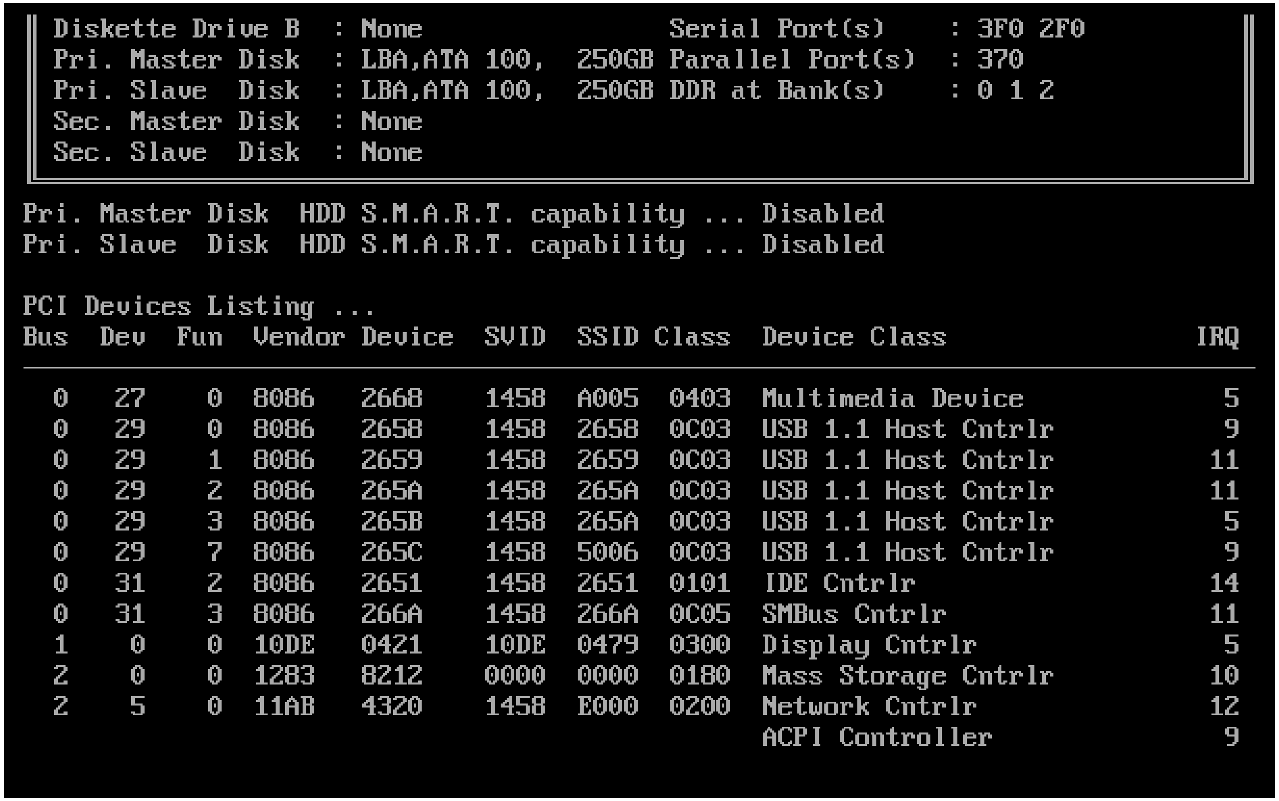
- POST



```
AMIBIOS(C)2007 American Megatrends, Inc.
ASUS P5KPL ACPI BIOS Revision 0603
CPU : Intel(R) Pentium(R) Dual CPU E2180 @ 2.00GHz
 Speed : 2.51 GHz     Count : 2

Press DEL to run Setup
Press F8 for BBS POPUP
DDR2-667 in Dual-Channel Interleaved Mode
Initializing USB Controllers .. Done.
3584MB OK
```

BIOS boot process (cont.)

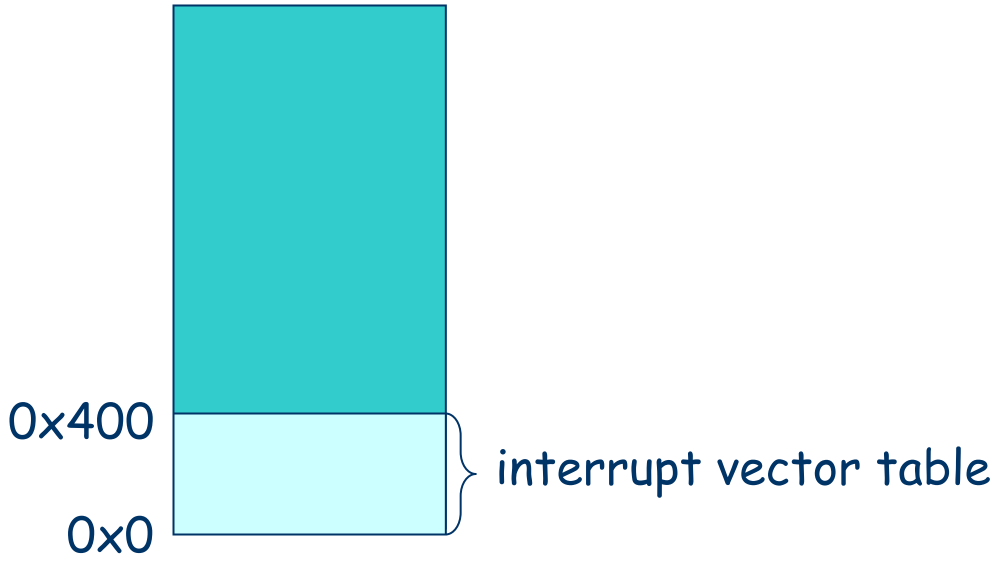
- POST



```
 ║ Diskette Drive B  : None                Serial Port(s)    : 3F0 2F0  ║
 ║ Pri. Master Disk  : LBA,ATA 100,  250GB Parallel Port(s)  : 370      ║
 ║ Pri. Slave  Disk  : LBA,ATA 100,  250GB DDR at Bank(s)    : 0 1 2    ║
 ║ Sec. Master Disk  : None                                             ║
 ║ Sec. Slave  Disk  : None                                             ║

Pri. Master Disk  HDD S.M.A.R.T. capability ... Disabled
Pri. Slave  Disk  HDD S.M.A.R.T. capability ... Disabled

PCI Devices Listing ...
Bus  Dev  Fun  Vendor Device  SVID  SSID Class  Device Class            IRQ
--------------------------------------------------------------------------------
  0   27    0  8086   2668    1458  A005  0403  Multimedia Device         5
  0   29    0  8086   2658    1458  2658  0C03  USB 1.1 Host Cntrlr       9
  0   29    1  8086   2659    1458  2659  0C03  USB 1.1 Host Cntrlr      11
  0   29    2  8086   265A    1458  265A  0C03  USB 1.1 Host Cntrlr      11
  0   29    3  8086   265B    1458  265A  0C03  USB 1.1 Host Cntrlr       5
  0   29    7  8086   265C    1458  5006  0C03  USB 1.1 Host Cntrlr       9
  0   31    2  8086   2651    1458  2651  0101  IDE Cntrlr               14
  0   31    3  8086   266A    1458  266A  0C05  SMBus Cntrlr             11
  1    0    0  10DE   0421    10DE  0479  0300  Display Cntrlr            5
  2    0    0  1283   8212    0000  0000  0180  Mass Storage Cntrlr      10
  2    5    0  11AB   4320    1458  E000  0200  Network Cntrlr           12
                                                ACPI Controller           9
```

BIOS boot process (cont.)

- POST
  - establish interrupt vector table



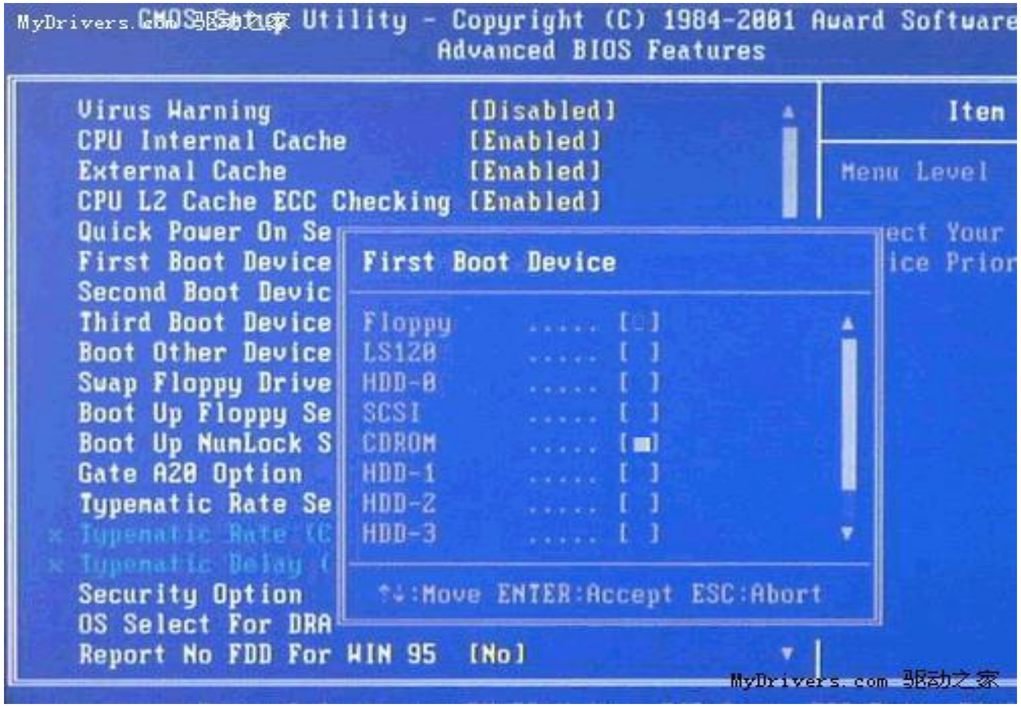
BIOS boot process (cont.)

- Find the “boot device”

```
uint8_t *mbr = (void *)0x7c00;
while (device != null) {
        // read 512 bytes from the beginning of device
        // to memory location 0x7c00
        dev_read(device, mbr, 0, 512);
        if (mbr[510] == 0x55 && mbr[511] == 0xaa)
                jump_to(mbr);
        else
                device = next_device();
}
print(“No bootable device!”);
```

BIOS boot process (cont.)

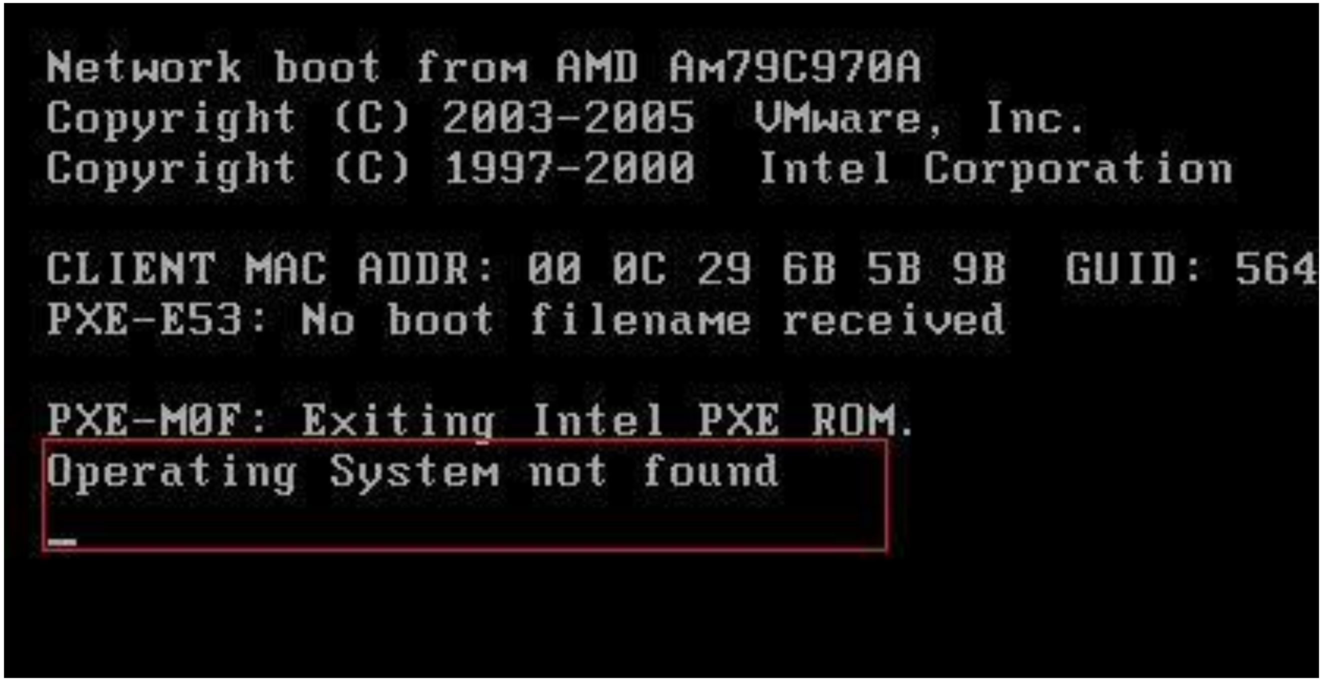
- The order of device checking is determined by CMOS.



```
          CMOS Setup Utility - Copyright (C) 1984-2001 Award Software
                         Advanced BIOS Features

   Virus Warning              [Disabled]          |      Item
   CPU Internal Cache         [Enabled]           |
   External Cache             [Enabled]           | Menu Level
   CPU L2 Cache ECC Checking  [Enabled]           |
   Quick Power On Se                                  ect Your
   First Boot Device | First Boot Device              ice Prior
   Second Boot Devic |
   Third Boot Device | Floppy     ..... [ ]
   Boot Other Device | LS120      ..... [ ]
   Swap Floppy Drive | HDD-0      ..... [ ]
   Boot Up Floppy Se | SCSI       ..... [ ]
   Boot Up NumLock S | CDROM      ..... [■]
   Gate A20 Option   | HDD-1      ..... [ ]
   Typematic Rate Se | HDD-2      ..... [ ]
 x Typematic Rate (C | HDD-3      ..... [ ]
 x Typematic Delay ( |
   Security Option   |  ↑↓:Move ENTER:Accept ESC:Abort
   OS Select For DRA
   Report No FDD For WIN 95   [No]
```

BIOS boot process (cont.)

- If no “bootable” device is found, print the error message.



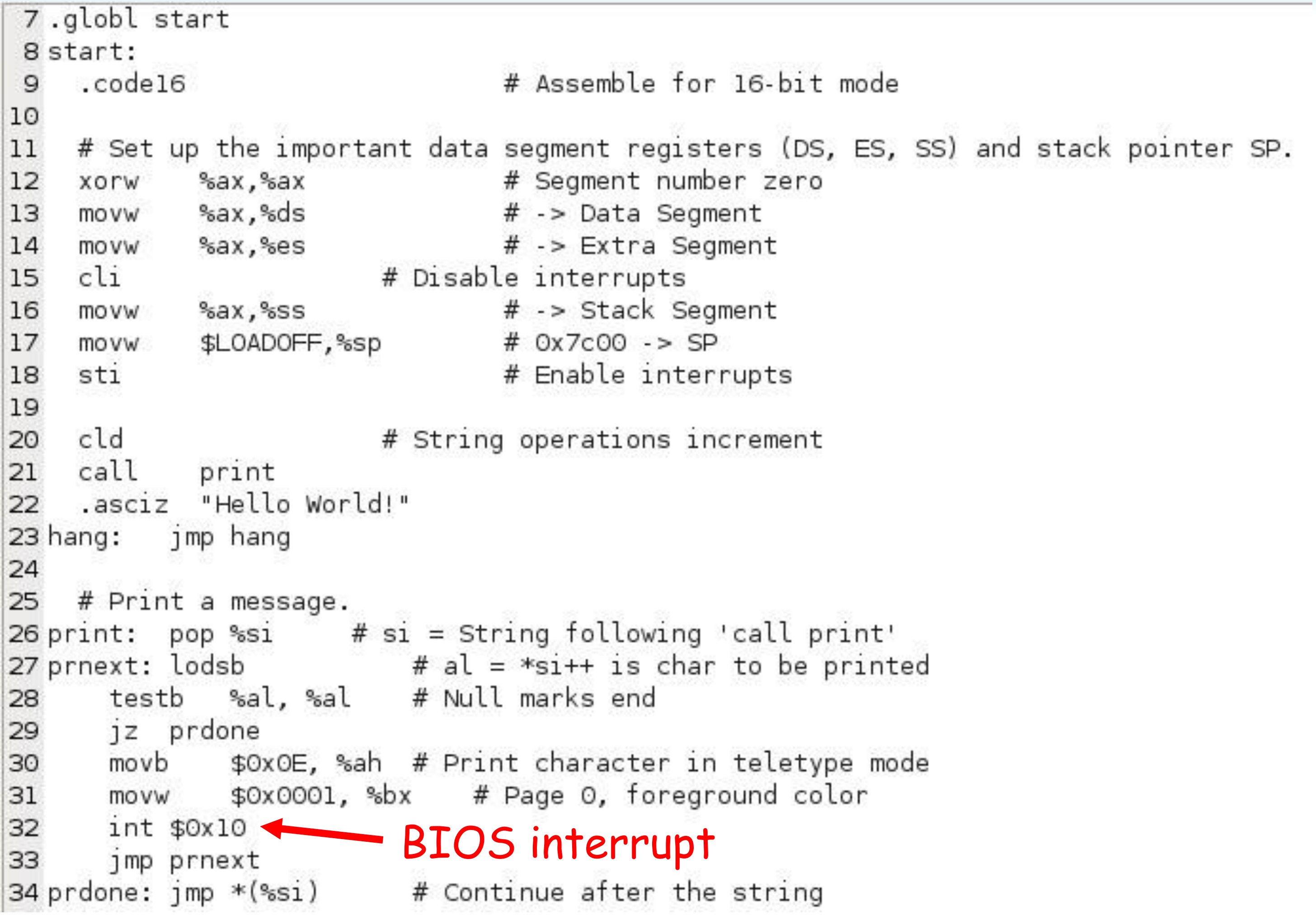
```
Network boot from AMD Am79C970A
Copyright (C) 2003-2005  VMware, Inc.
Copyright (C) 1997-2000  Intel Corporation

CLIENT MAC ADDR: 00 0C 29 6B 5B 9B  GUID: 564
PXE-E53: No boot filename received

PXE-M0F: Exiting Intel PXE ROM.
Operating System not found
_
```

- Booting
  - recap - 8086 & 80386
  - BIOS booting
  - the bigger world

When we are in MBR

- We are in real mode.
  - 16-bit environment
  - 1MB addressing
  - BIOS interrupts are available
- The computer is fully controlled by our code!
- We can do almost everything!
  - print a string
  - show the current time
  - read/write the disk
  - ...

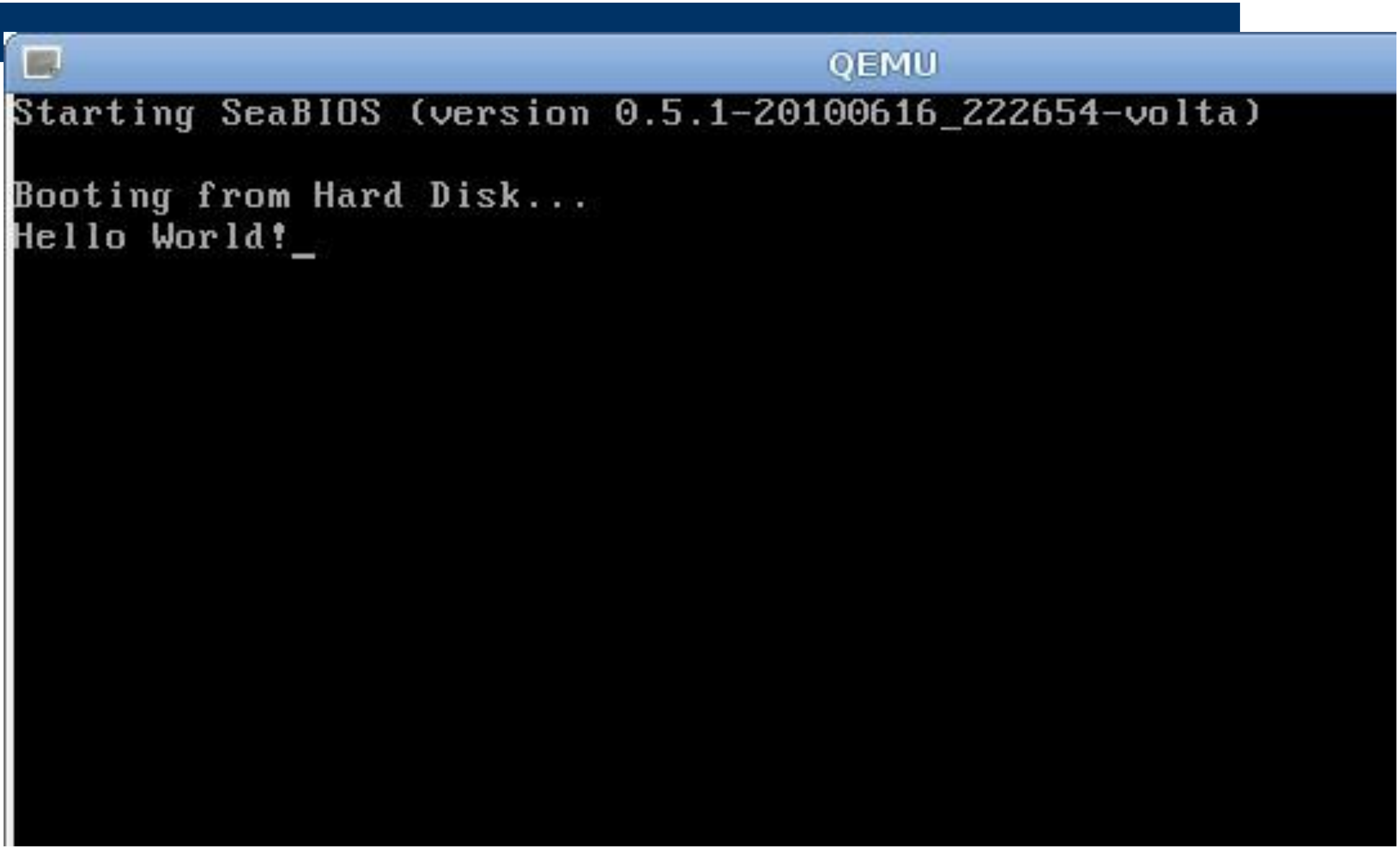
“Hello world” in MBR



```
.globl start
start:
  .code16                     # Assemble for 16-bit mode

  # Set up the important data segment registers (DS, ES, SS) and stack pointer SP.
  xorw    %ax,%ax             # Segment number zero
  movw    %ax,%ds             # -> Data Segment
  movw    %ax,%es             # -> Extra Segment
  cli                 # Disable interrupts
  movw    %ax,%ss             # -> Stack Segment
  movw    $LOADOFF,%sp        # 0x7c00 -> SP
  sti                         # Enable interrupts

  cld                 # String operations increment
  call    print
  .asciz  "Hello World!"
hang:   jmp hang

  # Print a message.
print:  pop %si     # si = String following 'call print'
prnext: lodsb           # al = *si++ is char to be printed
    testb   %al, %al    # Null marks end
    jz  prdone
    movb    $0x0E, %ah  # Print character in teletype mode
    movw    $0x0001, %bx    # Page 0, foreground color
    int $0x10
    jmp prnext
prdone: jmp *(%si)      # Continue after the string
```

BIOS interrupt

“Hello world” in MBR (cont.)



```
Starting SeaBIOS (version 0.5.1-20100616_222654-volta)

Booting from Hard Disk...
Hello World!_
```

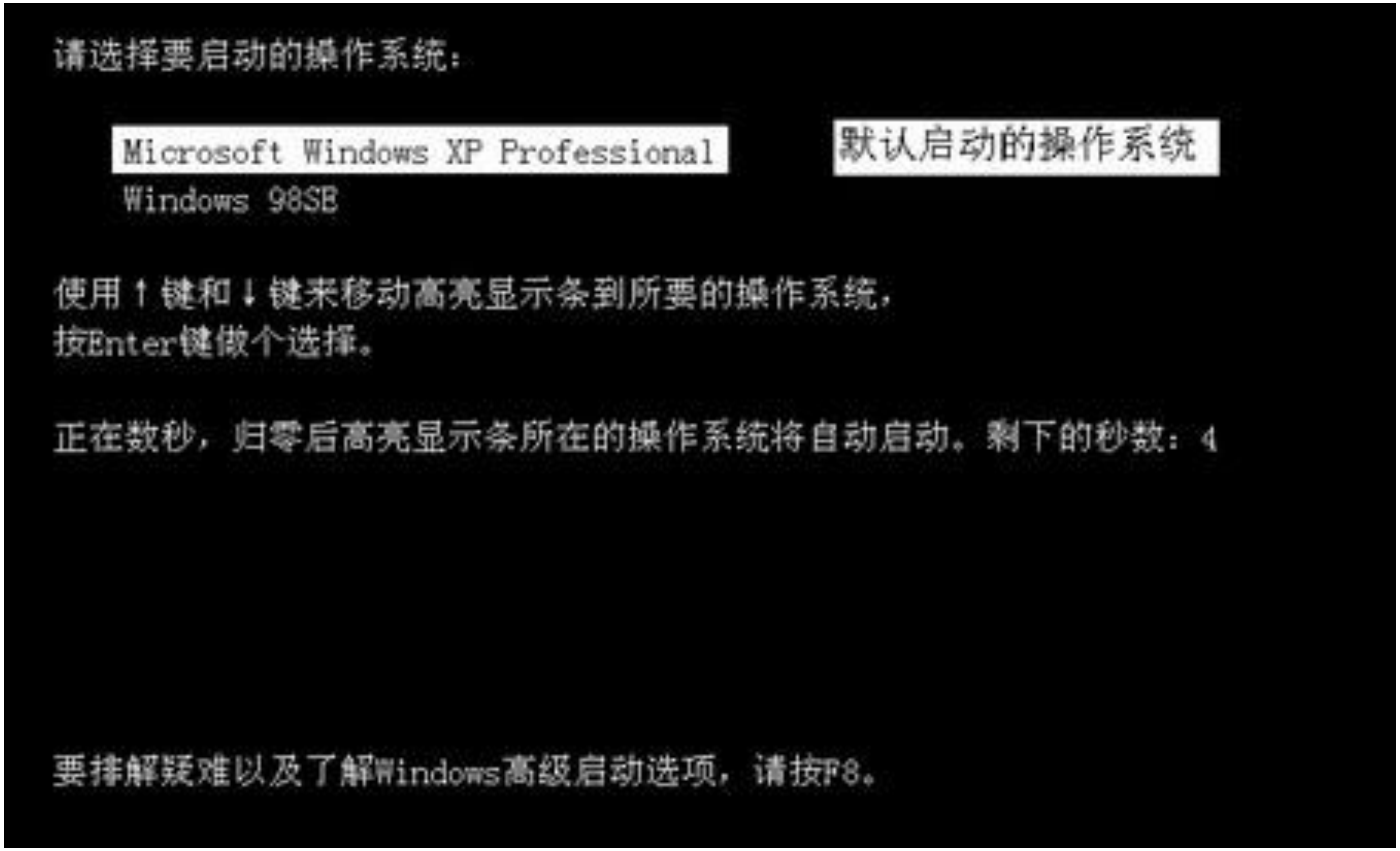
But…

- MBR is only 512 bytes
  - usually you cannot use all of them
  - contains partition table entries
- too small to contain an OS
- We can load more programs in MBR.

Boot sequence

- Minix
  - BIOS->MBR->MBR'->PBR->Boot->OS
- DOS
  - BIOS->MBR->VBR->OS
- Windows
  - BIOS->MBR->VBR->NTLDR / BOOTMGR->OS

“NTLDR” in Windows



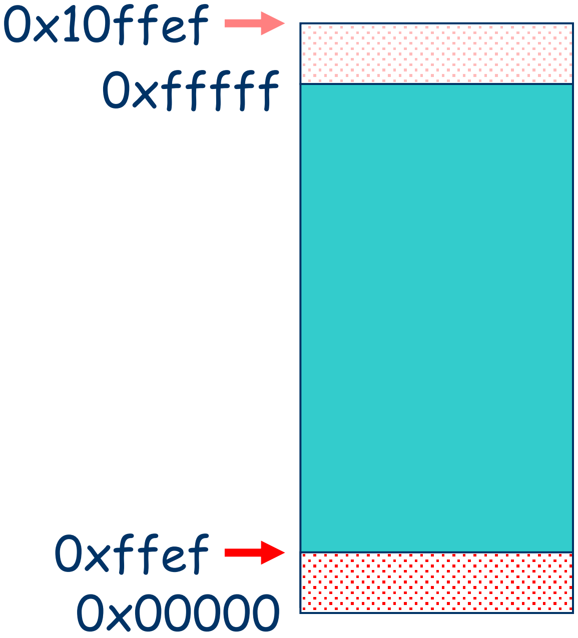
```
请选择要启动的操作系统：

    Microsoft Windows XP Professional          默认启动的操作系统
    Windows 98SE

使用↑键和↓键来移动高亮显示条到所要的操作系统，
按Enter键做个选择。

正在数秒，归零后高亮显示条所在的操作系统将自动启动。剩下的秒数：4



要排解疑难以及了解Windows高级启动选项，请按F8。
```

But… (cont.)

- we need
  - 32-bit environment
  - multitasking
  - protection
  - …
- Switch to protected mode!
  - before switching, GDT should be established
  - an interesting problem: A20 line

A20 line

- in 8086, PA = (CS << 4) + IP
- when CS = 0xffff, IP = 0xffff
  - PA = 0x10ffef
- only 20 address lines
  - wraparound



A20 line (cont.)

640K ought to be enough for anybody

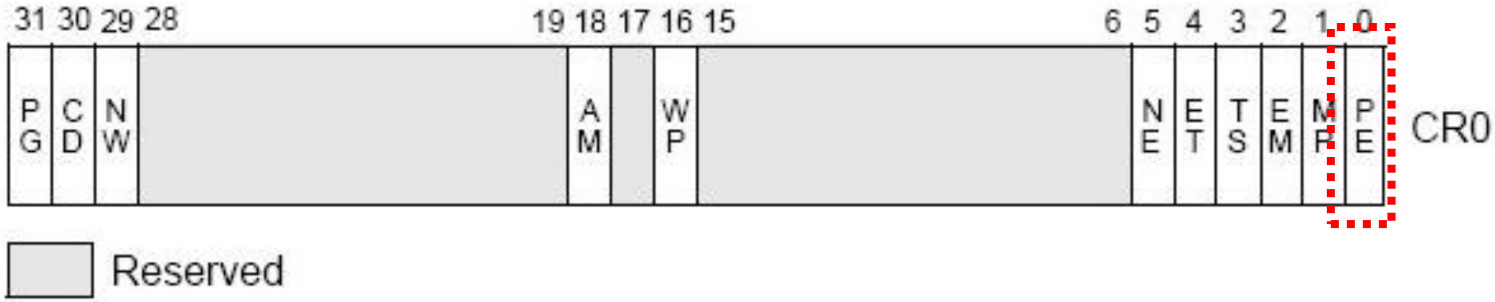
A20 line (cont.)

- some "2*" programs take this property
  - rather than accessing the bottom of memory directly
- i386 need to be compatible with them
- disable A20 line (the 21st address line) by default
  - always be zero
- before switching to protected mode, A20 line must be enabled

I_want_to_wrap _around.exe

Towards the bigger world

- switch to protected mode
  - PE = 1 in CR0



Bootblock

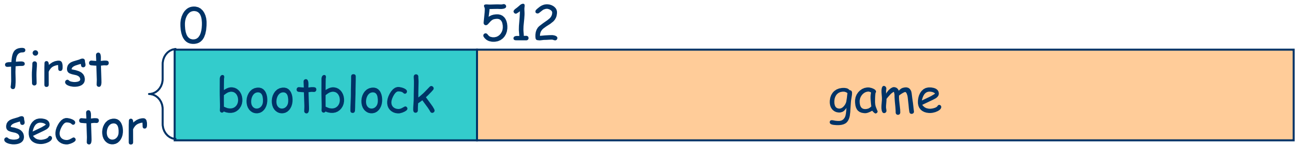
- In Lab0, bootblock does the following
  - disable interrupts
  - switch to graphic mode
  - enable A20 line
  - establish GDT
    - 4GB flat segments to “disable” segmentation
  - switch to protected mode
  - reset the segment registers (why?)
  - setup the stack
  - load the game into memory, then jump to it

Loading the game

- ELF file format
  - search the Internet for more details
- loaded to the physical address 0x100000

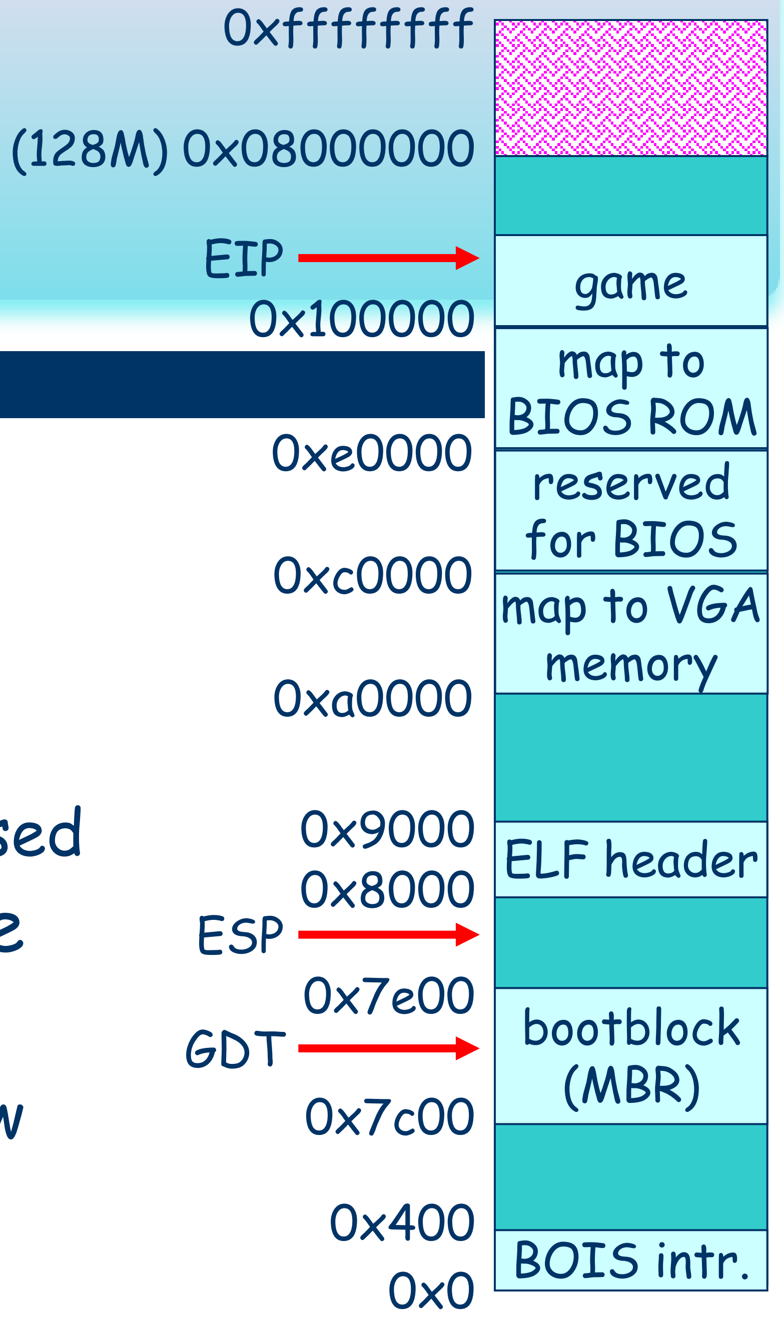
```
$(LD) $(LDFLAGS) -e game_init -Ttext 0x00100000 -o game $(OBJS)
```

- How does the bootblock (MBR) know where the game locates in disk?
  - it is pre-arranged – starts at the 2nd sector



Memory layout

- BIOS interrupts
  - cannot be used any longer
  - why?
- bootblock (MBR)
  - contains GDT currently used
- ESP points to somewhere around 0x8000
  - no protection for overflow
  - be careful!!!



Now it is the game's turn!

- initialize IDT
- initialize interrupt handles used in irq_handle()
- enable interrupts
- the game is running!
  - interrupt-driven

- The game design

FC games

A modern game

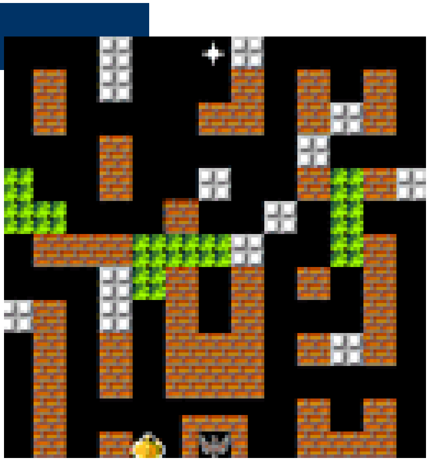
What is a game?

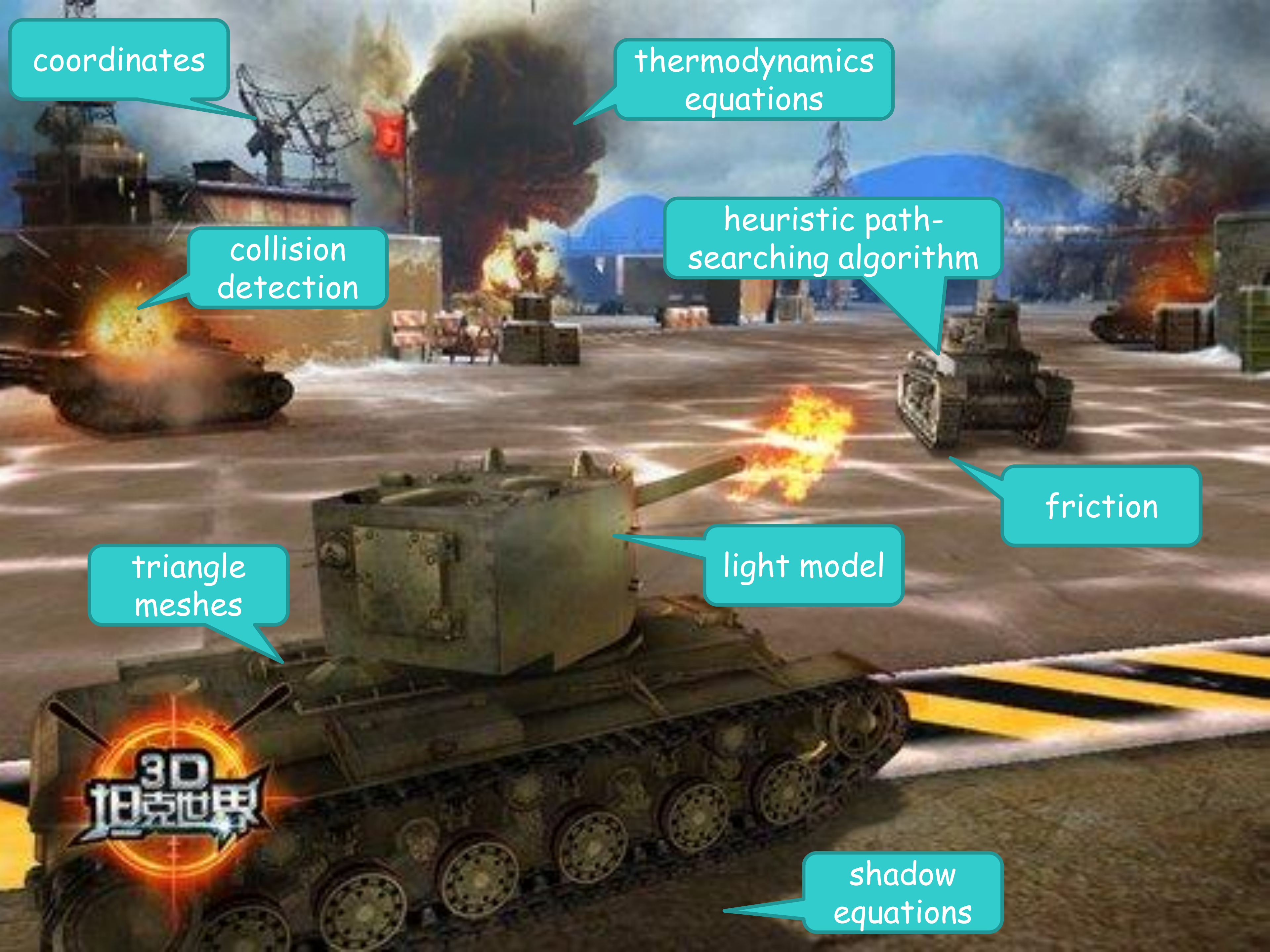
- Game = logic + display [+ sound + …]

- Logic determines how the states of objects change internally.
- Display shows the current state of the game to player.

All games are computed!

- Every object has its coordinates and properties.
- Objects interact with each other.
- Use physical laws to simulate our world.
  - collision, friction, light…
- The state of an object is computed according to its last state.





coordinates

thermodynamics equations

collision detection

heuristic path-searching algorithm

friction

light model

triangle meshes

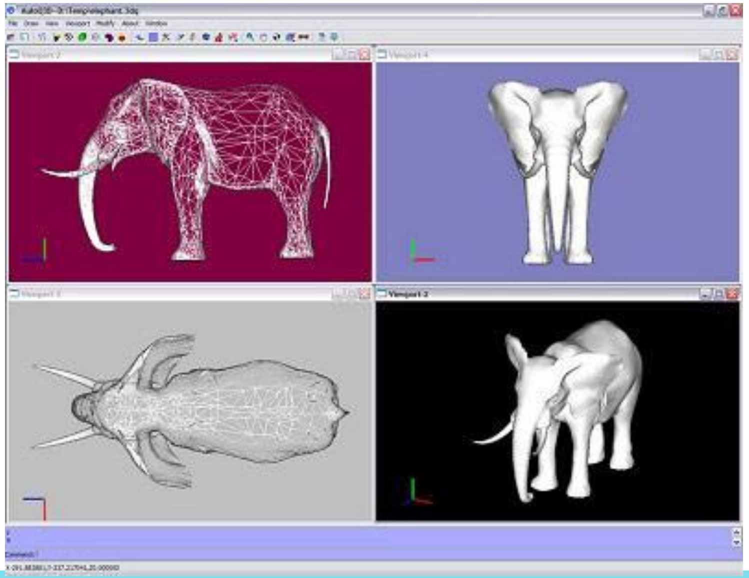
shadow equations

Logic implementation

- physical engine
  - gravity, friction, collision…
  - makes the game more real
- collision detection
  - every object has its shape
  - determines whether two objects have interaction
    - hit, miss
    - trigger other events
- AI
  - makes the game more insteresting
- Everything is computed!

Display the game

- Every object is shown pixel by pixel.
- What about 3D objects?
  - modeling -> effect processing -> transformation -> projection -> display
- Everything is computed!



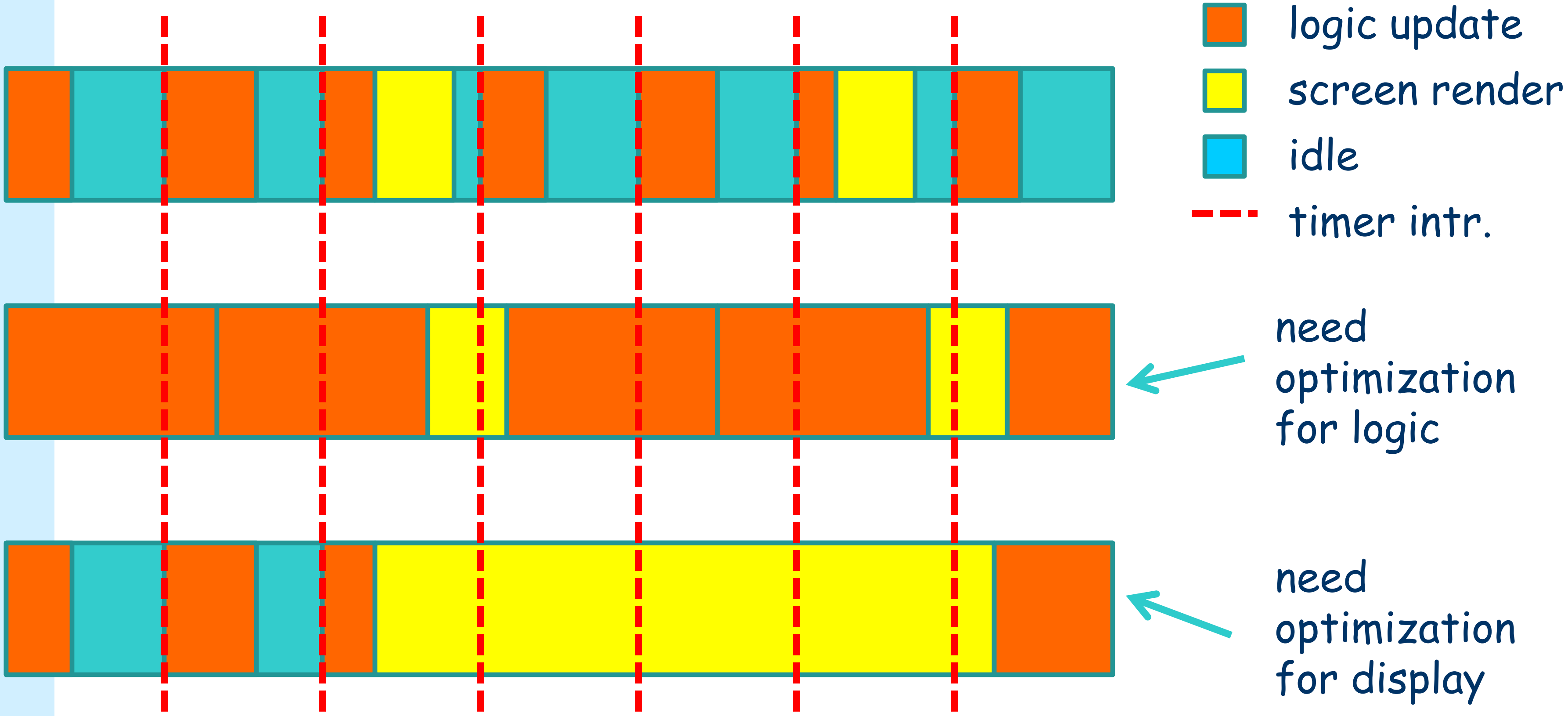
Primitive – draw_pixel()

- display a pixel on the screen
  - write a byte to video memory
  - memory-mapped I/O
    - 0xa0000~0xbffff
    - just like writing a byte to main memory
- The whole screen is drawn pixel by pixel.

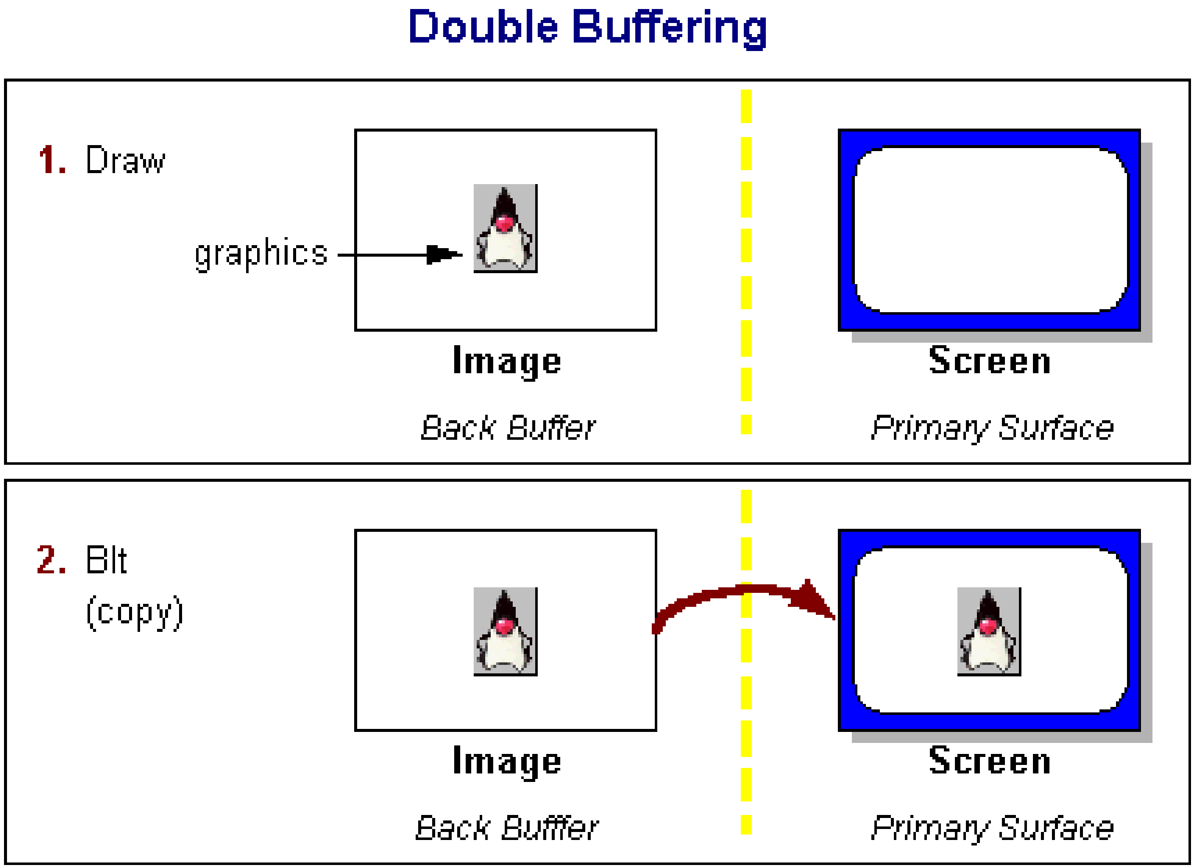
Object movement

- Logic
  - the coordinates are changed
- Display
  - erase the object at old position
    - erase = draw background
  - draw the object at new position

Frame-based rendering



Double buffering + partial update

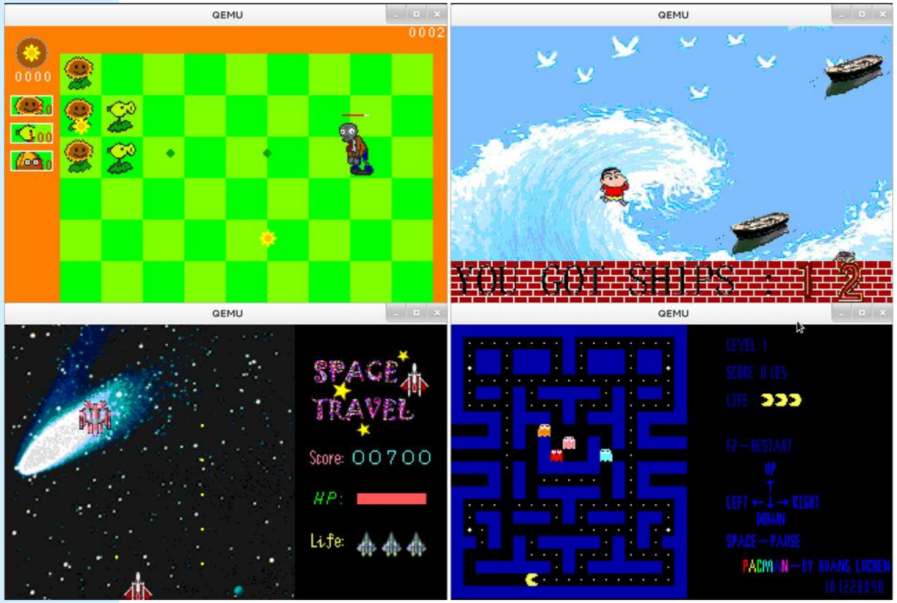


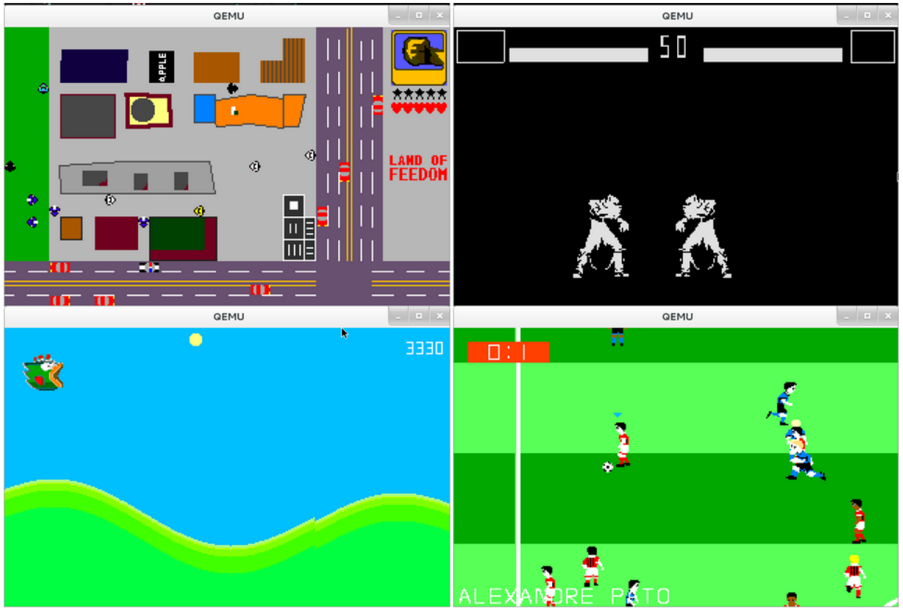
Develop your game

- program the logic
  - any game driven by timer interrupts is accepted
- interrupt-driven
  - timer interrupt
    - 100Hz timer
    - drives the logical time
  - keyboard interrupt
    - tell the game to capture the key pressed by player
    - update the game logic to interact with player

Have fun!

- http://cslab.nju.edu.cn/ics/index.php/Game
  - 10 Csers' OSLab0





- Debugging

Bugs

内存单元非法访问
系统内存不足
栈空间溢出
文件访问错误
堆空间溢出
找不到对象
磁盘空间已满
指针未初始化
不安全的类型转换
地址访问权限不
件路径未

What makes debugging hard?

- Fault --> Error --> Failure
  - fault – programmer's mistake
  - error – program runs in unexpected state
  - failure – fatal error, program cannot run any more
    - usually observable
- an example in Linux
  - fault - ???
  - error – a pointer is assigned NULL, which should not be
  - failure – dereferencing the pointer triggers segmentation fault

What makes debugging hard? (cont.)

- Fault --> Error --> Failure
  - fault may not activate error immediately
  - error may not trigger failure immediately
- When failure is triggered, it is far from the source of fault and error.
  - temporal & spatial
    - the further --> the higher cost
    - much further with context switching, hard to replay
  - more errors are activated
  - backtracking is difficult

What do they mean?

- Goal – expose fault & error as early as possible
  - once detected, trigger a failure
- the magical tools for debugging
  - -Wall & -Werror
  - assert
  - printk
  - GDB

-Wall & -Werror

- Do you hope that compiler helps you find your faults?
  - Wall – enable all warnings
  - Werror – treat warnings as compile errors
- This forces you to clear all warnings before the program runs.
  - Most of them may be potential errors.

-Wall & -Werror (cont.)

```
if(i >> 3 + 2 < 1000)

if(x = 0)
```

- Without –Wall, compilation succeeds.
- With –Wall, warning message is shown.
- With –Wall & -Werror, error message is shown, and compilation fails.
  - expose some faults in the compilation step
  - protect you against some silly mistakes

Assert

- set checkpoints
  - pass – continue to run
  - fail – stop and report a failure

- assert(cond_expr);
  - cond_expr is the condition to check

Assert (cont.)

- "must-not-reach" branch

```
if (…)
            …
else if (…)
            …

switch (x) {
            case …: …
            case …: …
            case …: …
}
```

```
if (…)
            …
else if (…)
            …
else
            assert(0);

switch (x) {
            case …: …
            case …: …
            case …: …
            default: assert(0);
}
```

Assert (cont.)

http://www.oschina.net/code/snippet_1156660_22107

- Without assertion

```
void delNode ( LinkList l , int n ) {
        LinkList p , q; int i = 1;
        p = q = l - > next ;
        if ( n == 1) { l - > next = q - > next ;
        } else {
                while ( i != n && q - > next != NULL ) {
                        p = q;
                        q = q - > next ;
                        i ++;
                }
                if ( i == n ) { p - > next = q - > next ; }
        }
}
```

Assert (cont.)

http://www.oschina.net/code/snippet_1156660_22107

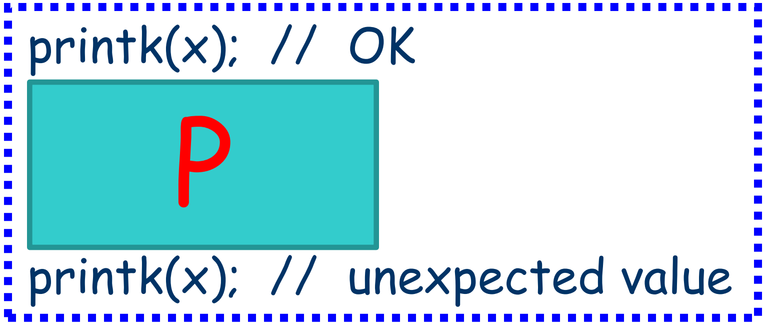
```
void delNode ( LinkList l , int n ) {
        LinkList p , q; int i = 1;
        assert(l != NULL && n > 0);
        p = q = l - > next ;
        assert(q != NULL);
        if ( n == 1) { l - > next = q - > next ;
        } else {
                while ( i != n && q - > next != NULL ) {
                        p = q;
                        q = q - > next ;
                        i ++;
                }
                if ( i == n ) { p - > next = q - > next ; }
        }
}
```

Assert (cont.)

- Assertion is VERY powerful.
  - convert errors into failures
    - expose errors immediately
    - shorten the distance between error and failure
  - protect you against activating more errors
- Assertions reflect your understanding of the program behavior.
  - What states are unexpected.
- With enough assertions, one can prove the program is correct.

Printk

- output some messages
  - detect whether some codes are reached
- display the value of variables
  - see whether the values are expected
- “binary search”
  - there must be an error related to x in P



printk(x); // OK

printk(x); // unexpected value

Printk (cont.)

- some useful macro

- __LINE__	an integer
- __FUNCTION__	a string
- __FILE__	a string

```
printk ( "reach line %d in function \"%s\", file \"%s\"\n" ,
        __LINE__ , __FUNCTION__ , __FILE__ ) ;
```

Printk (cont.)

- display more information before assertion fail

```
# define printa(cond, ...) \
        do { \
                if ( !(cond) ) { \
                        printk(__VA_ARGS__); \
                        assert(0); \
                } \
        } while(0)

printa(x >= 0, "x = %d\n", x) ;
```

GDB

- use GDB for remote debugging
  - see the instructions for Lab0 for more details
- the ultimate tool – can do “everything”
  - breakpoint
  - conditional breakpoint
  - watchpoint
  - stack backtrace
  - examine registers and memory locations
  - see “help” in gdb for more details

Suggestions for debugging

- Always enable –Wall & -Werror.
- Insert as many assertions as possible.
- Use printk to output debug information.
- Use GDB to make the execution flow clear.

- Interrupts & exceptions make the execution flow more complex.
  - Can you find the last instruction before interrupt/exception is triggered?

THE MOST IMPORTANT

机器总是对的!

未测试代码总是错的!

要怀疑机器，先怀疑自己!