A20总线

■产生背景：

1981年8月，IBM公司最初推出的个人计算机IBM PC使用的CPU是Intel8088。在该微机中地址线只有20根(A0 – A19)。在当时，内存RAM只有几百KB或不到1MB时，20根地址线已足够用来寻址这些内存。

在8086/8088中，有20根地址总线，所以可以访问的地址是2^20=1M，但由于8086/8088是16位地址模式，能够表示的地址范围是0-64K，所以为了在8086/8088下能够访问1M内存，Intel采取了分段的模式：16位段基地址:16位偏移。其绝对地址计算方法为：16位基地址左移4位+16位偏移=20位地址；

通过上述分段模式，能够表示的最大内存为：FFFFh:FFFFh=FFFF0h+FFFFh=10FFEFh=1M+64K-16Bytes（1M多余出来的部分被称做高端内存区HMA）。可是在8086/8088中，当程序给出超过1M（100000H-10FFEFH）的地址时，系统并不认为其访问越界而产生异常，而是自动从重新0开始计算，环绕到0FFEFh，也就是说系统计算实际地址的时候是按照对1M求模的方式进行的，这种技术被称为wrap-around。

1985年，当IBM公司引入AT机时，使用的是Intel 80286 CPU，具有24根地址线(A0-A23)，最高可寻址16MB，并且有一个与8088完全兼容的实模式运行方式。然而，在寻址值超过1MB时它却不能象8088那样实现地址寻址的环绕。

但是当时已经有一些程序是利用这种地址环绕机制进行工作的。为了实现完全的兼容性，IBM公司发明了使用一个开关(逻辑门)来开启或禁止100000h地址比特位。由于在当时的8042键盘控制器上恰好有空闲的端口引脚（输出端口P2，引脚P21），于是便使用了该引脚来作为与门控制这个地址比特位。该信号即被称为A20门。如果A20为零，则比特20及以上地址都被清除。从而实现了兼容性。毕竟A20Gate和键盘操作没有任何关系，在许多新型PC上存在着一种通过芯片来直接控制A20 Gate的BIOS功能。从性能上，这种方法比通过键盘控制器来控制A20Gate要稍微高一点。

当A20被禁止时：程序员给出100000H~10FFEFH间的地址，80286和8086/8088 的系统表现是一致的，即按照对1M求模的方式进行寻址，满足系统升级的兼容性问题；

当A20被开启时：程序员给出的100000H~10FFEFH间的地址，80286是访问的真实地址，而8086/8088是始终是按照对1M求模的方式进行的（这里注意，是始终）。

■A20与实模式，保护模式

实模式就是， 为了实现系统升级的兼容性，如80286的系统表现（包括80286以后的CPU）要与8086/8088 的系统表现一致，就需要80286 CPU访问100000H-10FFEFH之间的地址的时候， 按照对1M求模的方式进行， 无论A20地址线开启关闭与否， 这种内存访问情况 称为实模式。

保护模式就是， 以A20地址线开启为前提，80286 CPU访问100000H-10FFEFH之间的地址的时候， 是访问真实的内存地址，不是求模访问，如访问100001H，就是真真切切地 访问 0x 100001H，而不是求模的 0x000001H 地址， 这种内存访问情况称为保护模式；

如果A20Gate被禁止：对于80286来说，其地址为24bit，其地址表示为EFFFFF；对于80386极其随后的32-bit芯片来说，其地址表示为FFEFFFFF。这种表示的意思是如果A20Gate被禁止，则其第20-bit在CPU做地址访问的时候是无效的，永远只能被作为0；如果A20 Gate被打开：则其第20-bit是有效的，其值既可以是0，又可以是1。

所以，在保护模式下，如果A20 Gate被打开，则可以访问的内存则是连续的；如果A20Gate被禁止，则可以访问的内存只能是偶数段，因为是20位（从0始）总为零，所23~20位只能是0000、0010、0100、0110、1000、1010、1100、1110对应十六进制为0、2、4、6、8、A、C、E。对应的十六进制地址段是000000-0FFFFF，200000-2FFFFF，400000-4FFFFF…。

A20地址线并不是打开保护模式的关键，只是在保护模式下，不打开A20地址线，你将无法访问到所有的内存。

■简要整理

1.80286与8086兼容

2.80286处于实模式下时，防止用户程序访问到100000h~10FFEFh之间的内存（高端内存）

3.8086模式，A20关闭的情况下，访问超过1MB内存时，会自动回卷

4.8086模式下，A20打开的情况下，访问超过1MB内存，就真实的访问

5.保护模式下，A20关闭（始终为0），则用户的地址只能是：0 - (1MB-1), 2 - (3MB-1), 4 - (5MB-1)，我们可以这样设想，A20为个位数（以1MB为单位），如果它始终为0，你永远不可能让这个数变成奇数。

5.保护模式下，A20开启，则可以访问全地址，没有奇偶MB的问题。

■打开A20总线

在引导系统时，BIOS先打开A20总线来统计和测试所有的系统内存。而当BIOS准备将计算机的控制权交给操作系统时会先将A20总线关闭。所以操作系统必须使用适当的方法来开启它。但是由于各种兼容机所使用的芯片集不同，要做到这一点却是非常的麻烦。因此通常要在几种控制方法中选择。对A20信号线进行控制的常用方法是通过设置键盘控制器的端口值。

多数PC都使用键盘控制器（8042芯片）来处理A20Gate。从理论上讲，打开A20Gate的方法是通过设置8042芯片输出端口（64h）的2nd-bit，但事实上，当你向8042芯片输出端口进行写操作的时候，在键盘缓冲区中，或许还有别的数据尚未处理，因此你必须首先处理这些数据。

流程如下：

　1. 禁止中断；

　2. 等待，直到8042 Inputbuffer为空为止；

　3. 发送禁止键盘操作命令到8042Input buffer；

　4. 等待，直到8042 Inputbuffer为空为止；

　5. 发送读取8042 OutputPort命令；

　6. 等待，直到8042 Outputbuffer有数据为止；

　7. 读取8042 Outputbuffer，并保存得到的字节；

　8. 等待，直到8042 Inputbuffer为空为止；

　9. 发送Write 8042Output Port命令到8042 Input buffer；

　10. 等待，直到8042 Inputbuffer为空为止；

　11. 将从8042 OutputPort得到的字节的第2位置1（OR 2），然后写入8042 Input buffer；

　12. 等待，直到8042 Inputbuffer为空为止；

　13. 发送允许键盘操作命令到8042Input buffer；

　14. 打开中断。

下面代码是一个相关实现：

enable\_a20:

cli ; 1. 禁止中断；

call wait\_input\_empty; 2. 等待，直到8042 Inputbuffer为空为止；

movb $0xAD, %al; 3. 发送禁止键盘操作命令到8042Input buffer；

outb $0x64 #disableKeyboard

call wait\_input\_empty; 4. 等待，直到8042 Inputbuffer为空为止；

movb $0xD0, %al; 5. 发送读取8042 OutputPort命令；

outb $0x64 #command-read 8042 output port

call wait\_output\_full; 6. 等待，直到8042 Outputbuffer有数据为止；

inb $0x60 # got thevalue of 8042 output port and save it

pushb %al; 7. 读取8042 Outputbuffer，并保存得到的字节；

call wait\_input\_empty; 8. 等待，直到8042 Inputbuffer为空为止；

movb $0xD1, %al; 9. 发送Write 8042Output Port命令到8042 Input buffer；

outb $0x64 #command-write 8042 output port

call wait\_input\_empty; 10. 等待，直到8042 Inputbuffer为空为止；

popb %al

orb $0x02, %al #enable A20 Gate;andb $0xfd 关闭A20

outb $0x60; 11. 将从8042 OutputPort得到的字节的第2位置1（OR 2），然后写入8042 Input buffer；

call wait\_input\_empty; 12. 等待，直到8042 Inputbuffer为空为止；

movb $0xAE, %al; 13. 发送允许键盘操作命令到8042Input buffer；

outb $0x64 #enableKeyboard

sti; 14. 打开中断。

ret

wait\_input\_empty:

rp1: inb $0x64

testb %al, 0x02

jnz rp1

ret

wait\_output\_full:

rp2: inb $0x64

testb %al, 0x01

jz rp2

ret

以上描述的是一种和IBMPC完全兼容的，通过键盘控制器控制A20 Gate的方法。但是，正象我们在前面所提到的，A20 Gate与键盘操作完全没有关系，IBM之所以将A20Gate的功能控制放在键盘控制器上，完全是一种为了弥补Intel 80286与Intel8086/8088不完全兼容的缺陷，而采取的Hacker行为，所以在许多新型PC上存在着一种通过芯片来直接控制A20 Gate的BIOS功能，我们在RealMode下只需要调用BIOS中断就可以实现A20 Gate的控制功能。这个BIOS中断为 INT 15h, AX=2401h。被称为Fast A20。

movw $0x2401, %ax

int $0x15

MOV AX, 2401H

INT 15H

INT 15的2400，2401，2402命令被用来关闭，开启和返回A20线状态。

2400和2401（关闭、开启）命令返回状态：

CF = clear if success ＃如果成功则清空

AH = 0 ＃

CF = set on error ＃如果被设置则表明发生了错误

＃AH中保存的是状态码，01＝键盘控制器在安全模式，0x86＝功能不被支持

2042命令返回状态

CF = clear if success ＃如果成功则清空

＃AH中保存的是状态码，01＝键盘控制器在安全模式，0x86＝功能不被支持

＃AL表示A20线的当前状态，00＝关闭，01＝开启

＃CX值如果是0xffff则表明键盘控制器在0xc000次尝试中均没准备好。

CF = set on error ＃发生错误进CF会置位

关闭A20总线

push ax

mov ax, 0x2400

int 0x15

pop ax

开启A20总线：

push ax

mov ax, 0x2401

int 0x15

pop ax

检查A20总线

push ax

push cx

mov ax, 0x2402

int 0x15

pop cx

pop ax

操作 System Control Port A，这种最常见。

MCA, EISA and other systems can also control A20 via port 0x92.

Bits 0,1,3,6,7 seem to have the same meaning everywhere this port is implemented.

Bit 0 (w): writing 1 to this bit causes a fast reset (used to switch back to real mode; for MCA this took 13.4 ms).

Bit 1 (rw): 0: disable A20, 1: enable A20.

Bit 3 (rw?): 0/1: power-on password bytes (stored in CMOS bytes 0x38-0x3f or 0x36-0x3f) accessible/inaccessible. This bit can be written to only when it is 0.

Bits 6-7 (rw): 00: hard disk activity LED off, 01,10,11: hard disk activity LED on.

Bits 2,4,5 are unused or have varying meanings. (On MCA bit 4 (r): 1: watchdog timeout occurred.)

使用系统0x92端口

这个方法是十分危险的，因为它可以导致和其他硬件冲突并强制关机。

0x92端口位功能

Bit 0 - Setting to 1 causes a fast reset ＃为1时快速启动

Bit 1 - 0: disable A20, 1: enable A20 ＃为0时关闭A20线，为1时开启A20线

Bit 2 - Manufacturer defined ＃工厂定义

Bit 3 - power on password bytes. 0: accessible, 1: inaccessible ＃口令开机，0表示使用，1表示禁止

Bits 4-5 - Manufacturer defined ＃工厂定义

Bits 6-7 - 00: HDD activity LED off, 01 or any value is "on" ＃为00时表示HDD访问LED关闭，其他值为开启。

通过0x92端口开启A20总线的代码

push ax

in al,0x92

or al, 2;mov al, 2

out 0x92, al

pop ax

关闭A20总线

push ax

in al,0x92

and al, 0fdh ;mov al, 0

out 0x92, al

pop ax

AMI BIOS (不通用)

Bit 7 = 1: Weitek math coprocessor present

Bit 6 = 1: Floppy drive seek at boot disabled

Bit 5 = 1: System boot sequence A:,C: (otherwise C:,A:)

Bit 4 = 1: System boot CPU speed high

Bit 3 = 1: External cache enabled

Bit 2 = 1: Internal cache enabled

Bit 1 = 1: Fast gate A20 operation enabled

Bit 0 = 1: Turbo switch function enabled

■检测总线是否被打开

如果A20Gate被打开了，则在实模式下，程序员可以直接访问100000H～10FFEFH之间的内存，如果A20Gate被禁止，则在实模式下，若程序员访问100000H～10FFEFH之间的内存，则会被硬件自动转换为0H~0FFEFH之间的内存，所以我们可以利用这个差异来检测A20Gate是否被打开。

# This routine testswhether or not A20 is enabled. If so, it

# exits with zf = 0.

#

# The memory addressused, 0x200, is the int $0x80 vector, which

# should be safe.

A20\_TEST\_ADDR =4\*0x80

A20\_TEST\_LOOPS = 3

a20\_test:

pushw %cx

pushw %ax

xorw %cx, %cx

movw %cx, %fs # Lowmemory

decw %cx

movw %cx, %gs # Highmemory area

movw$A20\_TEST\_LOOPS, %cx

movw%fs:(A20\_TEST\_ADDR), %ax

pushw %ax

a20\_test\_wait:

incw %ax

movw %ax,%fs:(A20\_TEST\_ADDR)

call delay #Serialize and make delay constant

cmpw%gs:(A20\_TEST\_ADDR+0x10), %ax

loope a20\_test\_wait

popw%fs:(A20\_TEST\_ADDR)

popw %ax

popw %cx

ret

delay:

outb %al,$0x80

ret

检测程序2：

; NASM

[bits 16]

org 0x7c00

mov ax, cs

mov ds, ax

mov es, ax

call check\_a20

test ax, ax

mov ax, A20On

jnz Print ; Enabled

mov ax, A20Off

Print:

mov bp, ax

mov cx, 16

mov ax, 0x1301

mov bx, 0x000c

mov dl, 0

int 0x10

cli ; Shutdown

hlt

check\_a20:

push ds

push es

push di

push si

cli

xor ax, ax ; ax = 0

mov es, ax

not ax ; ax = 0xFFFF

mov ds, ax

mov di, 0x0500

mov si, 0x0510

mov al, byte [es:di]

push ax

mov al, byte [ds:si]

push ax

mov byte [es:di], 0x00

mov byte [ds:si], 0xFF

cmp byte [es:di], 0xFF

pop ax

mov byte [ds:si], al

pop ax

mov byte [es:di], al

mov ax, 0

je check\_a20\_\_exit

mov ax, 1

check\_a20\_\_exit:

pop si

pop di

pop es

pop ds

ret

A20On:

db "A20 is On "

A20Off:

db "A20 is Off "

times 510-($-$$) db 0

db 0x55

db 0xaa

（完）