A20 Line

From OSDev Wiki

The A20 Address Line is the physical representation of the 21st bit (number 20, counting from 0) of any memory access. When the IBM-AT (Intel 286) was introduced, it was able to access up to sixteen megabytes of memory (instead of the 1 MByte of the 8086). But to remain compatible with the 8086, a quirk in the 8086 architecture (memory wraparound) had to be duplicated in the AT. To achieve this, the A20 line on the address bus was disabled by default.

The wraparound was caused by the fact the 8086 could only access 1 megabyte of memory, but because of the segmented memory model it could effectively address up to 1 megabyte and 64 kilobytes (minus 16 bytes). Because there are 20 address lines on the 8086 (A0 through A19), any address above the 1 megabyte mark wraps around to zero. For some reason a few short-sighted programmers decided to write programs that actually used this wraparound (rather than directly addressing the memory at its normal location at the bottom of memory). Therefore in order to support these 8086-era programs on the new processors, this wraparound had to be emulated on the IBM AT and its compatibles; this was originally achieved by way of a latch that by default set the A20 line to zero. Later the 486 added the logic into the processor and introduced the A20M pin to control it.

For an operating system developer (or Bootloader developer) this means the A20 line has to be enabled so that all memory can be accessed. This started off as a simple hack but as simpler methods were added to do it, it became harder to program code that would definitely enable it and even harder to program code that would definitely disable it.

Contents

Keyboard Controller

The traditional method for A20 line enabling is to directly probe the keyboard controller. The reason for this is that Intel's 8042 keyboard controller had a spare pin which they decided to route the A20 line through. This seems foolish now given their unrelated nature, but at the time computers weren't quite so standardized. Keyboard controllers are usually derivatives of the 8042 (http://www.diakom.ru/el/elfirms/datashts/Smsc/42w11.pdf) chip. By programming that chip accurately, you can either enable or disable bit #20 on the address bus.

When your PC boots, the A20 gate is always disabled, but some BIOSes do enable it for you, as do some high-memory managers (HIMEM.SYS) or bootloaders (GRUB).

Testing the A20 line

Before enabling the A20 with any of the methods described below it is better to test whether the A20 address line was already enabled by the BIOS. This can be achieved by comparing, at boot time in real mode, the bootsector identifier (0xAA55) located at address 0000:7DFE with the value 1 MiB higher which is at address FFFF:7E0E. When the two values are different it means that the A20 is already enabled otherwise if the values are identical it must be ruled out that this is not by mere chance. Therefore the bootsector identifier needs to be changed, for instance by rotating it left by 8 bits, and again compared to the 16 bits word at FFFF:7E0E. When they are still the same then the A20 address line is disabled otherwise it is enabled.

The following code performs a check (not like described above -- more directly).

```
; The following code is public domain licensed

[bits 16]

; Function: check_a20
;
; Purpose: to check the status of the a20 line in a completely self-contai
;          The function can be modified as necessary by removing push's at
;          respective pop's at the end if complete self-containment is not
;
; Returns: 0 in ax if the a20 line is disabled (memory wraps around)
;          1 in ax if the a20 line is enabled (memory does not wrap arounc

check_a20:
    pushf
    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
    popf

    ret
```

Testing The A20 Line From Protected Mode

When in Protected Mode it's easier to test A20 because you can access A20's set memory addresses using any odd megabyte address and compare it to it's even megabyte neighbor.

```
[bits 32]

; Check A20 line
; Returns to caller if A20 gate is cleared.
; Continues to A20_on if A20 line is set.
; Written by Elad Ashkcenazi

is_A20_on?:

pushad
mov edi,0x112345  ;odd megabyte address.
mov esi,0x012345  ;even megabyte address.
mov [esi],esi     ;making sure that both addresses contain diffrent values
mov [edi],edi     ;(if A20 line is cleared the two pointers would point to
cmpsd             ;compare addresses to see if the're equivalent.
popad
jne A20_on        ;if not equivalent , A20 line is set.
```

```
ret                ;if equivalent , the A20 line is cleared.

A20_on:
; *your code from here*
```

Enabling

There are several sources that enable A20, commonly each of the inputs are or'ed together to form the A20 enable signal. This means that using one method (if supported by the chipset) is enough to enable A20. If you want to disable A20, you might have to disable all present sources. Always make sure that the A20 has the requested state by testing the line as described above.

Keyboard Controller

For the original method to enable the A20 line, some hardware IO using the Keyboard Controller chip (8042 chip) is necessary.

```
void init_A20(void)
{
   uint8_t   a;

   disable_ints();

   kyb_wait_until_done();
   kyb_send_command(0xAD);         // disable keyboard

   kyb_wait_until_done();
   kyb_send_command(0xD0);         // Read from input

   kyb_wait_until_done();
   a=kyb_get_data();

   kyb_wait_until_done();
   kyb_send_command(0xD1);         // Write to output

   kyb_wait_until_done();
   kyb_send_data(a|2);

   kyb_wait_until_done();
   kyb_send_command(0xAE);         // enable keyboard

   enable_ints();
}
```

or in assembly

```
;;
;; NASM 32bit assembler
;;

[bits 32]
[section .text]

enable_A20:
        cli

        call    a20wait
        mov     al,0xAD
        out     0x64,al

        call    a20wait
        mov     al,0xD0
        out     0x64,al

        call    a20wait2
        in      al,0x60
        push    eax

        call    a20wait
        mov     al,0xD1
        out     0x64,al

        call    a20wait
        pop     eax
        or      al,2
        out     0x60,al

        call    a20wait
        mov     al,0xAE
        out     0x64,al

        call    a20wait
        sti
        ret

a20wait:
        in      al,0x64
        test    al,2
        jnz     a20wait
        ret


a20wait2:
        in      al,0x64
        test    al,1
```

```
        jz      a20wait2
        ret
```

Fast A20 Gate

On most newer computers starting with the IBM PS/2, the chipset has a FAST A20 option that can quickly enable the A20 line. To enable A20 this way, there is no need for delay loops or polling, just 3 simple instructions.

```
in al, 0x92
or al, 2
out 0x92, al
```

As mentioned at the see also site (http://www.win.tue.nl/~aeb/linux/kbd/A20.html) , it would be best to do the write only when necessary, and to make sure bit 0 is 0, as it is used for fast reset. An example follows:

```
in al, 0x92
test al, 2
jnz after
or al, 2
and al, 0xFE
out 0x92, al
after:
```

However, the Fast A20 method is not supported everywhere and there is no reliable way to tell if it will have some effect or not on a given system. Even worse, on some systems, it may actually do something else like blanking the screen, so it should be used only after the BIOS has reported that FAST A20 is available. Code for systems lacking FAST A20 support is also needed, so relying only on this method is discouraged. Also, on some chipsets you might have to enable Fast A20 support in the BIOS configuration screen.

INT 15

Another way is to use the BIOS.

```
;FASM
use16
mov     ax,2403h                ;--- A20-Gate Support ---
int     15h
jb      a20_ns                  ;INT 15h is not supported
cmp     ah,0
jnz     a20_ns                  ;INT 15h is not supported

mov     ax,2402h                ;--- A20-Gate Status ---
int     15h
jb      a20_failed              ;couldn't get status
cmp     ah,0
jnz     a20_failed              ;couldn't get status
```

```
cmp     al,1
jz      a20_activated           ;A20 is already activated

mov     ax,2401h                ;--- A20-Gate Activate ---
int     15h
jb      a20_failed              ;couldn't activate the gate
cmp     ah,0
jnz     a20_failed              ;couldn't activate the gate

a20_activated:                  ;go on
```

If only one interrupt fails, you will have to use another method. (See below.)

Access of 0xee

On some systems reading ioport 0xee enables A20, and writing it disables A20. (Or, sometimes, this action only occurs when ioport 0xee is enabled.) And similar things hold for ioport 0xef and reset (a write causes a reset). The i386SL/i486SL documents say

The following ports are visible only when enabled, Any writes to these ports cause the action named. Name of Register Address Default Value Where placed Size FAST CPU RESET EFh N/A 82360SL 8 FAST A20 GATE EEh N/A 82360SL 8

Enable A20:

```
in al,0xee
```

Disable A20:

```
out 0xee,al
```

NOTE that it doesn't matter what AL contains when writing and AL is undefined while reading (to / from port 0xee)

Recommended Method

Because there are several different methods that may or may not be supported, and because some of them cause problems on some computers; the recommended method is to try all of them until one works in the "order of least risk". Essentially:

- Test if A20 is already enabled - if it is you don't need to do anything at all
- Try the BIOS function. Ignore the returned status.
- Test if A20 is enabled (to see if the BIOS function actually worked or not)
- Try the keyboard controller method.
- Test if A20 is enabled in a loop with a time-out (as the keyboard controller method may work slowly)
- Try the Fast A20 method last
- Test if A20 is enabled in a loop with a time-out (as the fast A20 method may work slowly)

- If none of the above worked, give up

See Also

External links

- http://www.win.tue.nl/~aeb/linux/kbd/A20.html