Overview of New Version Josh 0.03.1

Previous Josh version (0.03)

Only had two commands
1. "ver" to show the version of the Operating system
2. "exit" to reboot the operating system

Core Enhancement

This version includes "info" command to print hardware information of the system.

Command "info"

By using this command user can view following hardware details of the computer

Basic hardware details
1. Processor Vendor
2. Processor details
3. Total memory available
4. Number of Hard disk Drives attached
5. Number of serial ports available
6. Number of Parallel ports available

System tray indicators
1. Caps lock status
2. Num lock status

1. Processor Vendor

CPUID implicitly uses the eax register. The eax register should be loaded with a value specifying what information to return. CPUID should be called with EAX = 0 ,This returns the CPU's manufacturer ID string - a twelve character ASCII string stored in ebx, edx, ecx - in that order. The highest basic calling parameter is returned in eax.

The following are known processor manufacturer ID strings:
"AMDisbetter!" - Early engineering samples of AMD K5 processor
"AuthenticAMD" - AMD
"CentaurHauls" - Centaur
"CyrixInstead" - Cyrix
"GenuineIntel" - Intel
"TransmetaCPU" - Transmeta
"GenuineTMx86" - Transmeta
"Geode by NSC" - National Semiconductor
"NexGenDriven" - NexGen
"RiseRiseRise" - Rise
"SiS SiS SiS " - SiS
"UMC UMC UMC " - UMC
"VIA VIA VIA " - VIA

Following chunk of code explains how it is implemented in the kernel

```
_display_cpu_vendor:

mov si, strCpu
mov al, 0x01
int 0x21

mov eax, 0
cpuid
mov [strCpuVen], ebx
mov [strCpuVen+4], edx
mov [strCpuVen+8], ecx
mov si, strCpuVen
mov al, 0x01
int 0x21

call _display_endl
ret
---------------------------------

[SEGMENT .bss]
        strCpuVen       resb    96              ;buffer for cpu vendor string
```

2. Processor details

CPUID should be called with setting up eax register to eax=80000002h, 80000003h, 80000004h respectively. These return the processor brand string in eax, ebx,ecx and edx. CPUID must be issued with each parameter in sequence to get the entire 48-byte null-terminated ASCII processor brand string.

Following code explains how it is implemented in the kernel

```
_display_processor_brand:

mov si, strCpuBrand
mov al, 0x01
int 0x21

mov eax, 80000002h
cpuid
mov [strCpuId], eax
mov [strCpuId+4], ebx
mov [strCpuId+8], ecx
mov [strCpuId+12], edx
mov si, strCpuId
mov al, 0x01
int 0x21
```

```
mov eax, 80000003h
cpuid
mov [strCpuId], eax
mov [strCpuId+4], ebx
mov [strCpuId+8], ecx
mov [strCpuId+12], edx
mov si, strCpuId
mov al, 0x01
int 0x21

mov eax, 80000004h
cpuid
mov [strCpuId], eax
mov [strCpuId+4], ebx
mov [strCpuId+8], ecx
mov [strCpuId+12], edx
mov si, strCpuId
mov al, 0x01
int 0x21
call _display_endl
ret

------------------------------

[SEGMENT .bss]
strCpuId        resb    128             ;buffer for cpu brand string
```

3. Total memory available

Interrupt 15 should be called with ax=0xe801, this returns

CF clear if successful
AX = extended memory between 1M and 16M, in K (max 3C00h = 15MB)
BX = extended memory above 16M, in 64K blocks
CX = configured memory 1M to 16M, in K
DX = configured memory above 16M, in 64K blocks
CF set on error

To show total extended memory ax, bx register values should be added. These register values (ax in kB and bx in 64kB blocks). are converted to MB and added to convert ax(in kB), bits in that register are shifted left by 10 bits and bx(in 64kB), bits in that register are shifted left by 4 bits.

ax only shows memory between 1MB and 16 MB, therefore by calling this interrupt, it shows 1MB less than actual memory available. Therefore Interrupt 12 has been used for show the base memory that was previousy hidden from the interrupt 15. by just calling the interrupt 12 it will return the value into ax register

Following code explains how it is implemented in the kernel

```
 _display_ram_info:

call _display_base_memory

mov si, strMemo
mov al, 0x01
int 0x21

mov ax, 0xe801
int 0x15

shr ax, 10                    ;memory 1mb to 16 mb in KB converting to MB
shr bx, 4                     ;memory > 16mb in 64 KB blocks converting to MB
add ax, bx
mov dx, ax
call _hex2dec

call _display_space
mov si, strMb
mov al, 0x01
int 0x21

call _display_endl
ret

_display_base_memory:              ;display base memory <1 mb

mov si, strBaMemo
mov al, 0x01
int 0x21

int 0x12
mov dx, ax
call _hex2dec
call _display_space

mov si, strKb
mov al, 0x01
int 0x21
call _display_endl
ret
```

Note- The function "hex2dec" explanation documented at the end of this document.

4. Number of Hard disk Drives attached

When power is applied to the computer, the BIOS Data Area is created at memory location 0040:0000h with a typical size of 255 bytes. in offset 75h gives the number of Hard disk drives attached to the system

Following code explains how it is implemented in the kernel

```
_dispaly_num_of_hDD:        ; number of hard disk drives attached

mov si, strHDD
mov al, 0x01
int 0x21

push es
mov ax, 0x40
mov es, ax
mov ax, [es:75h]
add ax, 30h                 ;to convert value in to ascii
mov ah, 0x0e
int 0x10
pop es
call _display_endl
ret
```

5. Number of serial ports available

When power is applied to the computer, the BIOS Data Area is created at memory location 0040:0000h with a typical size of 255 bytes. in offset 10h gives the equipments available in the system

```
Bits 15-14 indicate the number of parallel ports installed
00b = 1 parallel port
01b = 2 parallel ports
03b = 3 parallel ports
Bits 13-12 are reserved
Bits 11-9 indicate the number of serial ports installed
000b = none
001b = 1 serial port
002b = 2 serial ports
003b = 3 serial ports
004b = 4 serial ports
Bit 8 is reserved
Bit 7-6 indicates the number of floppy drives installed
0b = 1 floppy drive
1b = 2 floppy drives
Bits 5-4 indicate the video mode
00b = EGA or later
01b = color 40x25
10b = color 80x25
11b = monochrome 80x25
```

Bit 3 is reserved
Bit 2 indicates if a PS/2 mouse is installed
0b = not installed
1b = installed
Bit 1 indicated if a math coprocessor is installed
0b = not installed
1b = installed
Bit 0 indicated if a boot floppy is installed
0b = not installed
1b = installed

By shifting 9 bits left and using "and" operation with e00 hexadecimal value, function can filter-out the serial ports available in the system

Following code explains how it is implemented in the kernel

```
_display_serial_ports:                ;display number of serial ports

mov si, strSerial
mov al, 0x01
int 0x21

push es
mov ax, 0x40
mov es, ax
mov ax, [es:10h]
and ax, 0xe00
shr ax, 9
add ax, 30h
mov ah, 0x0e
int 0x10
call _display_endl
pop es
ret
```

6. Number of parallel ports available

In number 5 stated that in offset "10h" bit 15 and 14 gives parallel ports available in the system
By shifting 14 bits left and using "and" operation with ffffc000 hexadecimal value, function can filter-out the parallel ports.

Following code explains how it is implemented in the kernel

```
_display_parallel_ports: ;display number of parallel ports

mov si, strParallel
mov al, 0x01
int 0x21
```

```
push es
mov ax, 0x40
mov es, ax
mov ax, [es:10h]
and ax, 0xffffc000
shr ax, 14
add ax, 30h
mov ah, 0x0e
int 0x10
pop es
ret
```

System tray indicators

When power is applied to the computer, the BIOS Data Area is created at memory location 0040:0000h with a typical size of 255 bytes. in offset 17h gives the Keyboard shift flags set 1

Bit 7 indicates if Insert is on or off
0b = Insert off
1b = Insert on
Bit 6 indicates if CapsLock is on or off
0b = CapsLock off
1b - CapsLock on
Bit 5 indicates if NumLock is on or off
0b = NumLock off
1b = NumLock on
Bit 4 indicates if ScrollLock is on or off
0b = ScrollLock off
1b = ScrollLock on
Bit 3 indicates if the Alt key is up or down
0b = Alt key is up
1b = Alt key is down
Bit 2 indicates if the Control key is up or down
0b = Control key is up
1b = Control key is down
Bit 1 indicates if the Left Shift key is up or down
0b = Left Shift key is up
1b = Left Shift key is down
Bit 0 indicates if the Right Shift key is up or down
0b = Right Shift key is up
1b = Right Shift key is down

Caps lock status

by shifting 6 bits left and using "and" operation with 40 hexadecimal value, function can filter-out the Caps lock status(indicates if CapsLock is on or off)

Following code explains how it is implemented in the kernel

```
_capslock_state:                ;state whether capslock enabled or disabled

mov si, strCapslock
mov al, 0x01
int 0x21

push es
mov ax, 0x40
mov es, ax
mov ax, [es:17h]
and ax, 40h
shr ax, 6
pop es

cmp ax, 0
je _state_off

call _state_on
ret
```

Num lock status

by shifting 5 bits left and using "and" operation with 20 hexadecimal value, function can filter-out the num lock status(indicates if numlock is on or off)

following code explains how it is implemented in the kernel

```
_numlock_state:                 ;state whether numlock enabled or disabled

mov si, strNumlock
mov al, 0x01
int 0x21

push es
mov ax, 0x40
mov es, ax
mov ax, [es:17h]
and ax, 20h
shr ax, 5
pop es
```

```
cmp ax, 0
je _state_off

call _state_on
ret
```

Function "hex2dec"

This is use to convert values in registers to decimal numbers in ascii and print the corresponding value. This is achieved by dividing the register value by 10 and remainder taken, this remainder is converted to decimal and print, after that again incrementing value of cx register and divide again. This happen until all the value is printed.

Code which is use to do the above scenario is implemented below

```
_hex2dec:

push ax          ; save AX
push bx          ; save CX
push cx          ; save DX
push si          ; save SI
mov ax,dx        ; copy number into AX
mov si,10        ; SI will be the divisor
xor cx,cx        ; clean up the CX

_non_zero:

xor dx,dx        ; clean up the DX
div si           ; divide by 10
push dx          ; push number onto the stack
inc cx           ; increment CX to do it more times
or ax,ax         ; end of the number?
jne _non_zero    ; if not go to _non_zero

_write_digits:

pop dx           ; get the digit off DX
add dl,0x30      ; add 30 to get the ASCII value
```

```
call _print_char          ; print
loop _write_digits        ; keep going till cx == 0

pop si                  ; restore SI
pop cx                  ; restore DX
pop bx                  ; restore CX
pop ax                  ; restore AX
ret

_print_char:
push ax                 ; save that AX register
mov al, dl
mov ah, 0x0E            ; BIOS teletype acts on newline!
 mov bh, 0x00
mov bl, 0x07
 int 0x10

pop ax                  ; restore that AX register
ret
```

References

[1] “Interrupt Jump Table,” [Online]. Available: http://www.ctyme.com/intr/int.htm. [Accessed: November 21, 2010].

[2] “Bios Central,” [Online]. Available: http://www.bioscentral.com/misc/bda.htm. [Accessed: November 19, 2010].

[3] “CPUID,” [Online]. Available: http://en.wikipedia.org/wiki/CPUID. [Accessed: November 22, 2010].

[4] “BDA-BIOS Data Area - PC Memory Map,” [Online]. Available: http://stanislavs.org/helppc/bios_data_area.html. [Accessed: November 19, 2010].

[5] “PC Assembly Language,” [Online]. Available: http://www.drpaulcarter.com/pcasm/. [Accessed: November 10, 2010].

[6] “MS-DOS, PC-BIOS AND FILE I/O (Part 3),” [Online]. Available: http://www.arl.wustl.edu/~lockwood/class/cs306/books/artofasm/Chapter_13/CH13-3.html#HEADING3-193/. [Accessed: November 10, 2010].

[7] “asm tutorials,” [Online]. Available: http://asm.sourceforge.net/resources.html#tutorials. [Accessed: November 11, 2010].

[8] “x86 Assembly Guide,” [Online]. Available: http://www.cs.virginia.edu/~evans/cs216/guides/x86.html. [Accessed: November 14, 2010].

[9] “BIOS interrupt call,” [Online]. Available: http://en.wikipedia.org/wiki/BIOS_interrupt_call. [Accessed: November 19, 2010].

[10] “hexadecimal to decimal using NASM,” [Online]. Available: http://efreedom.com/Question/1-2197484/Hexadecimal-Decimal-Using-NASM. [Accessed: November 22, 2010].