CS 2042 – Operating Systems

Programming Assignment

D M Sahabandu

100466H

Achieved Task

Introduce some new methods to the shell of Josh to display hardware information of the system and date & time of the system.

Overview

As the first step of this assignment I have read the given tutorial completely and manage to understand the execution behaviour of the Josh OS to a certain extent. It includes a shell which can take string arguments as inputs and call for appropriate functions. After understanding it then I started to build a new function which can show some hardware information of the system. To achieve that task first I understood that I have to do some background researches on the area. So I search information over the internet through some blogs of experts in this area and I successfully find out a way to take hardware information of the system using x86 Assembly language. Some of the useful information areas I found out are BIOS interrupt calls and CPUID instructions. Out of these methods I used BIOS interrupt calls and CPUID instructions to display hardware information of the system and the date and the time of the system. In that display method it shows processor information, RAM information and the system time. The complete procedure I followed to achieve this task using x86 assembly language is explained below in this document.

Procedure (Briefly)

I. Create the working directory for the entire project.
II. Download boot.asm and kernel-3.0.asm files.
III. Copy those files into the working directory.
IV. Edit the kernel code until it provides the above mentioned functionalities. (This step will be described later in this document)
V. Compile the boot.asm and kernel-3.0.asm codes

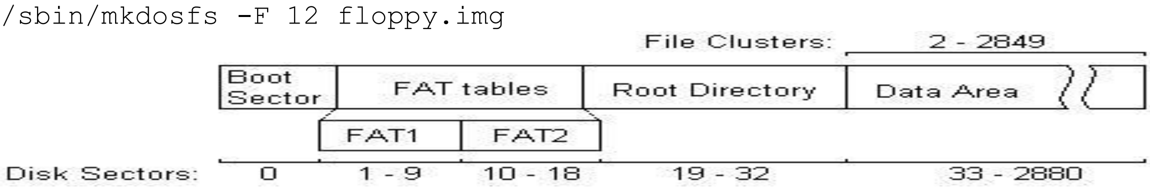
```
nasm boot.asm –f bin –o boot.bin
nasm kernel-3.0.asm -f bin -o kernel.bin
```

VI. Handle the errors and warnings in the edited code (Three warnings occurred and they were simply eliminated by adding colon at the end of the label)
VII. Create an image file size of 1.44MB (Because although I need to format the pen drive to fat12 format it is not allowed to format a device which is larger than 32MB into this file format)
VIII. Create the image of the floppy disk.

```
dd if=/dev/zero bs=512 count=2880 of=./floppy.img
```

IX. Format it in fat 12 file format.

```
/sbin/mkdosfs -F 12 floppy.img
```



X. Connect the loop device with floppy image.

```
sudo losetup /dev/loop1 ./floppy.img
```

XI. In order to mount the image, format the loop device.

```
sudo /sbin/mkdosfs -F 12 /dev/loop1
```

XII. Create separate directory to mount the device

```
sudo mkdir /media/floppy
```

XIII. Mount the floppy device

```
sudo mount -o loop /dev/loop1 /media/floppy/
```

XIV. Copy the boot.asm file into the boot sector of the floppy and kernel-3.0.asm file (edited version) into the data area of the floppy image.

```
sudo cp kernel.bin /media/floppy/
sudo dd if=./boot.bin of=/dev/loop1
```

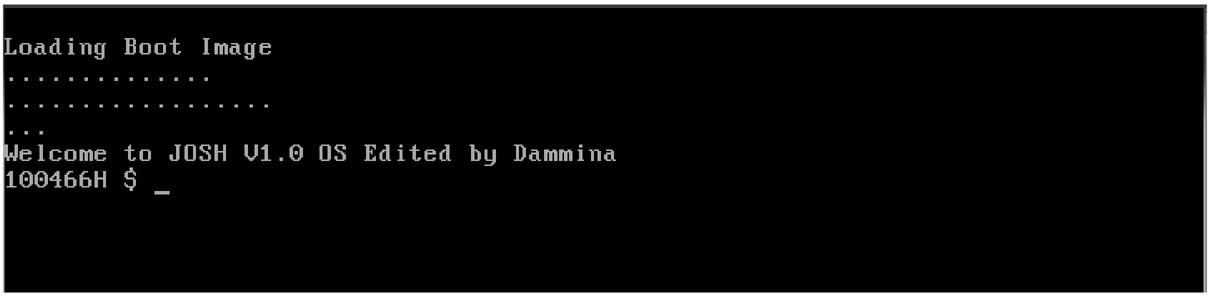
XV. Finally create a virtual machine using Oracle VM Virtual Box and provide the image as the first boot device and boot the new OS.

BIOS interrupt calls and CPUID instructions

I used CPUID instructions to retrieve the information about processor. In assembly language those instructions use no parameters. It only uses EAX registers. That register is loaded with the value specifying the information to return. First CPUID should be called with the value 0 in EAX. Because it will return the highest calling parameter that the CPU supports. To obtain extended function information CPUID should be called with bit 31 of EAX set. To determine the highest extended function calling parameter, call CPUID with `EAX = 80000000h`. BIOS interrupts also can be invoked by x86 Assembly language instructions. These functionalities were provided by BIOS help to store relevant data in a CPU register for our use.

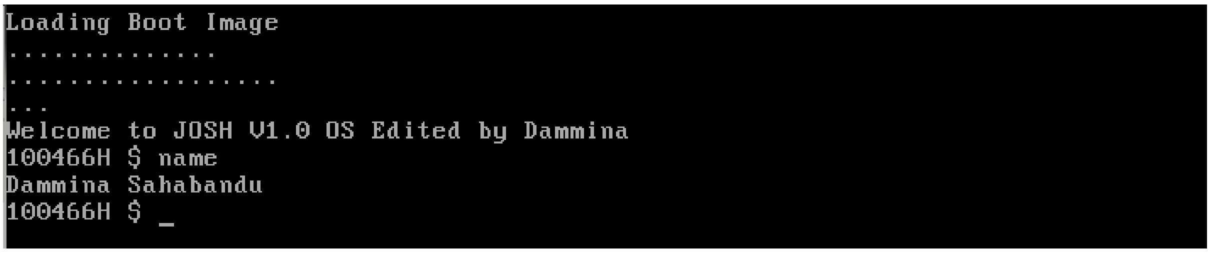
(Reference : http://en.wikipedia.org/wiki/CPUID)

Execution



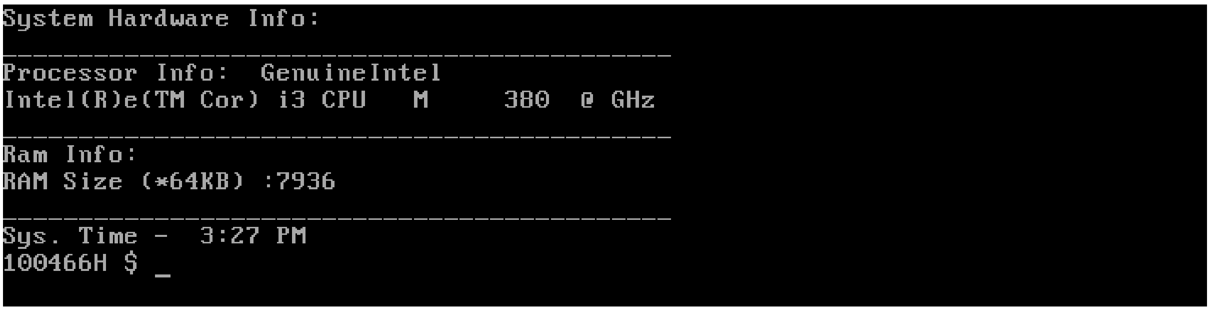
```
Loading Boot Image
..............
..................
...
Welcome to JOSH V1.0 OS Edited by Dammina
100466H $ _
```

Loading the boot image



```
Loading Boot Image
..............
..................
...
Welcome to JOSH V1.0 OS Edited by Dammina
100466H $ name
Dammina Sahabandu
100466H $ _
```

Print my name



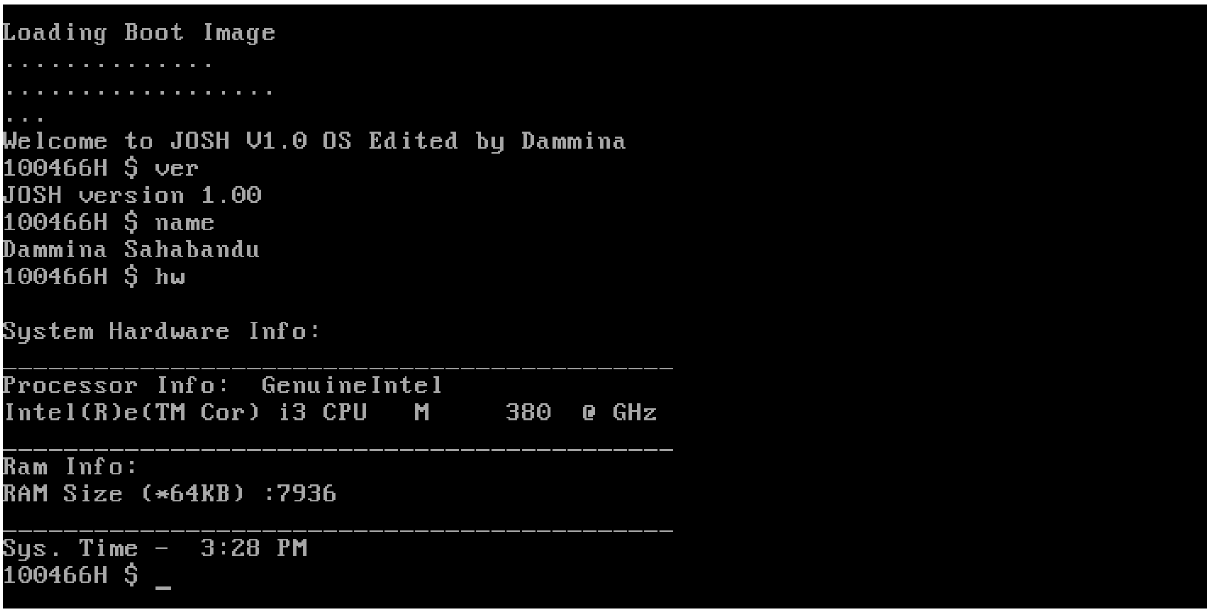
```
System Hardware Info:

_____________________________________________
Processor Info:  GenuineIntel
Intel(R)e(TM Cor) i3 CPU   M     380  @ GHz

_____________________________________________
Ram Info:
RAM Size (*64KB) :7936

_____________________________________________
Sys. Time -  3:27 PM
100466H $ _
```

Printing my PC hardware info



```
Loading Boot Image
..............
..................
...
Welcome to JOSH V1.0 OS Edited by Dammina
100466H $ ver
JOSH version 1.00
100466H $ name
Dammina Sahabandu
100466H $ hw

System Hardware Info:
__________________________________________
Processor Info:  GenuineIntel
Intel(R)e(TM Cor) i3 CPU   M     380  @ GHz
__________________________________________
Ram Info:
RAM Size (*64KB) :7936
__________________________________________
Sys. Time -  3:28 PM
100466H $ _
```

Complete view

The functions that have been added to the kernel

Processor Details

```
        ;mov si,ProcessorInfo                   /* Move the value of the ProcessorInfo variable to

                                                the register si*/

        call PrintMethod                        //Display the value of the si register in the kernel

        mov si,TAB

        call PrintMethod                        //Print a Tab after the ProcessorInfo value

        mov eax,0

        cupid                           /*save all the necessary information about the processor in

                                        to the registers*/

        mov [CPUvendor],ebx             /*move the value in the ebx register into the CPUvendor

                                        variable*/

        mov [CPUvendor+4],edx           /*move the value in the edx register in to the CPUvendor

                                        variables next 4(after the ebx value)*/

        mov [CPUvendor+8],ecx           // do the same thing for the ecx regiter

        mov si,CPUvendor

        call PrintMethod                //print the value of the register si

        call _display_endl

//repeat the same thing happened in the above code segment

        mov eax,80000002h

        cpuid

        mov [ProcessorType],eax

        mov [ProcessorType+4],ebx

        mov [ProcessorType+8],edx

        mov [ProcessorType+12],ecx

        mov si,ProcessorType

        call PrintMethod

//repeat the same thing happened in the above code segment

        mov eax,80000003h
        cpuid
        mov [ProcessorType2],eax
        mov [ProcessorType2+4],ebx
        mov [ProcessorType2+8],edx
        mov [ProcessorType2+12],ecx
        mov si,ProcessorType2
        call PrintMethod
        mov eax,80000004h
        cpuid
        mov [ProcessorType3],eax
        mov [ProcessorType3+4],ebx
        mov [ProcessorType3+8],edx
        mov [ProcessorType3+12],ecx
        mov si,ProcessorType3
        call PrintMethod
        call _display_endl
        mov si,EndOfLine
        call PrintMethod
```

RAM Details

```
;Ram
        call _display_endl
        mov si,RAMinfo
        call PrintMethod        //Display the value of the si register in the kernel
        mov si,TAB
        call PrintMethod        //Print a Tab after the RAMInfo value
        mov ax,0xE801           //set up the size information of the RAM into the ax register
        int 0x15        ; get ram size into registers       //calls the interrupt 15 to save the RAM size
                                                            Information to the registers
        call _display_endl

        mov si,RAMsize

        call PrintMethod        //Display the value of the si register in the kernel

        call _print_reg

        call _display_endl

        mov si,EndOfLine

        call PrintMethod
```

Display

```
[SEGMENT .data]

   WelcomeMessage   db  "Welcome to JOSH V1.0 OS Edited by Dammina", 0x00

        Prompt          db      "100466H $ ", 0x00              //my prompt

        cmdMaxLen               db      255                     ;maximum length of commands

        OSName                  db      "JOSH", 0x00    ;OS details//OS name

        MajorVersion            db      "1", 0x00

        MinorVersion            db      ".00", 0x00

        MyName                          db      "Dammina Sahabandu",0x00//my name

        TAB             db      "       ",0x00

        EndOfLine       db      "_________________________________________________",0x00

                                                        //separate sections by a line

        SystemHardware          db      "System Hardware Info:",0x00

        ProcessorInfo   db      "Processor Info:",0x00

        RAMinfo                 db      "Ram Info:",0x00

        RAMsize         db      "RAM Size (*64KB) :",0x00

        CPUvendor       db      "111111111111",0x00

        ProcessorType   db      "$$$$$$$$$$$$$$$$",0x00     //garbage value

        ProcessorType2 db       "$$$$$$$$$$$$$$$$",0x00     //garbage value

        ProcessorType3 db       "$$$$$$$$$$$$$$$$",0x00     //garbage value

        SystemTime                  db "Sys. Time - ",0x00      //system time

        space           db ", Time -",0x00
```

```
fmt_12_24 db 0                ; Non-zero = 24-hr format
    fmt_date        db 0, '/' ; 0, 1, 2 = M/D/Y, D/M/Y or Y/M/D
                                        ; Bit 7 = use name for months
                                        ; If bit 7 = 0, second byte = separator character
```

Variables Created

```
SystemHardware          db      "System Hardware Info:",0x00
ProcessorInfo   db      "Processor Info:",0x00
RAMinfo                 db      "Ram Info:",0x00
RAMsize         db      "RAM Size (*64KB) :",0x00
CPUvendor       db      "111111111111",0x00
ProcessorType   db      "$$$$$$$$$$$$$$$$",0x00
ProcessorType2 db       "$$$$$$$$$$$$$$$$",0x00
ProcessorType3 db       "$$$$$$$$$$$$$$$$",0x00
```

Internal Commands

```
cmdVer                  db      "ver", 0x00             ; internal commands
cmdExit                 db      "ext", 0x00
cmdName                         db      "name", 0x00
cmdHardware             db      "hw",0x00
```

Complete Code

```
;*****************start of the kernel code***************
[org 0x000]
[bits 16]

[SEGMENT .text]

;START #######################################################
   mov ax, 0x0100                        ;location where kernel is loaded
   mov ds, ax
   mov es, ax

   cli
   mov ss, ax                            ;stack segment
   mov sp, 0xFFFF                        ;stack pointer at 64k limit
   sti

   push dx
   push es
   xor ax, ax
   mov es, ax
   cli
   mov word [es:0x21*4], _int0x21        ; setup interrupt service
   mov [es:0x21*4+2], cs
   sti
   pop es
   pop dx
```

```
    mov si, WelcomeMessage   ; load message
    mov al, 0x01           ; request sub-service 0x01
    int 0x21

            call _shell                                      ; call the shell

    int 0x19              ; reboot
;END ########################################################

_int0x21:
    _int0x21_ser0x01:       ;service 0x01
    cmp al, 0x01           ;see if service 0x01 wanted
    jne _int0x21_end       ;goto next check (now it is end)

    _int0x21_ser0x01_start:
    lodsb                 ; load next character
    or  al, al            ; test for NUL character
    jz  _int0x21_ser0x01_end
    mov ah, 0x0E           ; BIOS teletype
    mov bh, 0x00           ; display page 0
    mov bl, 0x07           ; text attribute
    int 0x10              ; invoke BIOS
    jmp _int0x21_ser0x01_start
    _int0x21_ser0x01_end:
    jmp _int0x21_end

    _int0x21_end:
    iret

_shell:
```

```
_shell_begin:
;move to next line
call _display_endl

;display prompt
call _display_prompt

;get user command
call _get_command

;split command into components
call _split_cmd

;check command & perform action

; empty command
_cmd_none:
mov si, strCmd0
cmp BYTE [si], 0x00
jne     _cmd_ver                ;next command
jmp _cmd_done

; display version
_cmd_ver:
mov si, strCmd0
mov di, cmdVer
mov cx, 4
repe    cmpsb
jne     NameCommand                 ;next command
```

```
        call _display_endl
        mov si, OSName                  ;display version
        mov al, 0x01
int 0x21
        call _display_space
        mov si, txtVersion              ;display version
        mov al, 0x01
int 0x21
        call _display_space

        mov si, MajorVersion
        mov al, 0x01
int 0x21
        mov si, MinorVersion
        mov al, 0x01
int 0x21
        jmp _cmd_done

        ;display name
        NameCommand:
        mov si, strCmd0
        mov di, cmdName
        mov cx, 5
        repe    cmpsb
        jne     HardwareCommand                 ;next command

        call _display_endl
        mov si, MyName
        mov al, 0x01
        int 0x21
        jmp _cmd_done



        ;display hardware info
        HardwareCommand:
        mov si, strCmd0
        mov di,cmdHardware
        mov cx,2
        repe cmpsb
        jne     _cmd_exit
        call HardwareInformation
        jmp _cmd_done



        ; exit shell
        _cmd_exit:
        mov si, strCmd0
        mov di, cmdExit
        mov cx, 5
        repe    cmpsb
        jne     _cmd_unknown            ;next command

        je _shell_end                   ;exit from shell

        _cmd_unknown:
        call _display_endl
        mov si, msgUnknownCmd                   ;unknown command
        mov al, 0x01
int 0x21
```

```
        _cmd_done:

        ;call _display_endl
        jmp _shell_begin

        _shell_end:
        ret

PrintMethod:
        mov al, 0x01
        int 0x21
ret

HardwareInformation:
        call _display_endl
        call _display_endl
        mov si,SystemHardware
        call PrintMethod
        call _display_endl
        mov si,EndOfLine
        call PrintMethod
        call _display_endl
;Processor
        mov si,ProcessorInfo
        call PrintMethod

        mov si,TAB
        call PrintMethod
```

```
mov eax,0

cpuid

mov [CPUvendor],ebx

mov [CPUvendor+4],edx

mov [CPUvendor+8],ecx


mov si,CPUvendor

call PrintMethod

call _display_endl


mov eax,80000002h

cpuid

mov [ProcessorType],eax

mov [ProcessorType+4],ebx

mov [ProcessorType+8],edx

mov [ProcessorType+12],ecx

mov si,ProcessorType

call PrintMethod



mov eax,80000003h

cpuid

mov [ProcessorType2],eax

mov [ProcessorType2+4],ebx

mov [ProcessorType2+8],edx

mov [ProcessorType2+12],ecx

mov si,ProcessorType2

call PrintMethod


mov eax,80000004h
```

```
        cpuid
        mov [ProcessorType3],eax
        mov [ProcessorType3+4],ebx
        mov [ProcessorType3+8],edx
        mov [ProcessorType3+12],ecx
        mov si,ProcessorType3
        call PrintMethod

        call _display_endl
        mov si,EndOfLine
        call PrintMethod

;Ram
        call _display_endl

        mov si,RAMinfo
        call PrintMethod

        mov si,TAB
        call PrintMethod

        mov ax,0xE801
        int 0x15            ; get ram size into registers

        call _display_endl

        mov si,RAMsize
        call PrintMethod

        call _print_reg
        call _display_endl

        mov si,EndOfLine
        call PrintMethod

;Date and Time
        call _display_endl
        call _time_string

        mov si,SystemTime
        call PrintMethod

        mov si,TAB
        call PrintMethod

        mov si, BX
        call PrintMethod


ret


_time_string:
        pusha                       ;save all the registers

        mov di, bx                  ; Location to place time string

        clc                         ; For buggy BIOSes
        mov ah, 2                   ; Get time data from BIOS in BCD format
        int 1Ah
        jnc .read

        clc
        mov ah, 2                       ; BIOS was updating (~1 in 500 chance), so try again
        int 1Ah

.read:
        mov al, ch                      ; Convert hours to integer for AM/PM test
        call _bcd_to_dec
        mov dx, ax                      ; Save

        mov al, ch                      ; Hour
        shr al, 4               ; Tens digit - move higher BCD number into lower bits
        and ch, 0Fh                     ; Ones digit
        test byte [fmt_12_24], 0FFh
        jz .twelve_hr

        call .add_digit                 ; BCD already in 24-hour format
        mov al, ch
        call .add_digit
        jmp short .minutes

.twelve_hr:
        cmp dx, 0                       ; If 00mm, make 12 AM
        je .midnight

        cmp dx, 10                      ; Before 1000, OK to store 1 digit
        jl .twelve_st1
        cmp dx, 12                      ; Between 1000 and 1300, OK to store 2 digits
        jle .twelve_st2

        mov ax, dx                      ; Change from 24 to 12-hour format
        sub ax, 12
        mov bl, 10
        div bl
        mov ch, ah

        cmp al, 0                       ; 1-9 PM
        je .twelve_st1

        jmp short .twelve_st2           ; 10-11 PM

.midnight:
        mov al, 1
        mov ch, 2

.twelve_st2:
        call .add_digit                 ; Modified BCD, 2-digit hour
.twelve_st1:
        mov al, ch
        call .add_digit

        mov al, ':'                     ; Time separator (12-hr format)
        stosb

.minutes:
        mov al, cl                      ; Minute
        shr al, 4               ; Tens digit - move higher BCD number into lower bits
        and cl, 0Fh                     ; Ones digit
        call .add_digit
        mov al, cl
        call .add_digit

        mov al, ' '                     ; Separate time designation
        stosb

        mov si, .hours_string           ; Assume 24-hr format
        test byte [fmt_12_24], 0FFh
        jnz .copy

        mov si, .pm_string              ; Assume PM
        cmp dx, 12                      ; Test for AM/PM
        jg .copy

        mov si, .am_string              ; Was actually AM

.copy:
        lodsb                           ; Copy designation, including terminator
        stosb
        cmp al, 0
        jne .copy

        popa
        ret


.add_digit:
        add al, '0'                     ; Convert to ASCII
        stosb                           ; Put into string buffer
        ret



        .hours_string   db 'hours', 0
        .am_string      db 'AM', 0
        .pm_string      db 'PM', 0

_bcd_to_dec:
        pusha

        mov bl, al                      ; Store entire number for now

        and ax, 0Fh                     ; Zero-out high bits
        mov cx, ax                      ; CH/CL = lower BCD number, zero extended

        shr bl, 4               ; Move higher BCD number into lower bits, zero fill msb
        mov al, 10
        mul bl                          ; AX = 10 * BL

        add ax, cx                      ; Add lower BCD to 10*higher
        mov [.tmp], ax

        popa
        mov ax, [.tmp]                  ; And return it in AX!
        ret



        .tmp    dw 0
```

```
_print_reg:

        _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 our 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           ; no? Keep chuggin' away

        _write_digits:
        pop dx              ; get the digit off DX
        add dl,48             ; add 48 to get ASCII
        mov al, dl
```

```
    mov ah, 0x0e
    int 0x10
    loop _write_digits

    pop si          ; restore  SI
    pop cx          ; restore  DX
    pop bx          ; restore  CX
    pop ax          ; restore  AX
ret             ; End of procedure!

_get_command:
    ;initiate count
    mov BYTE [cmdChrCnt], 0x00
    mov di, strUserCmd

    _get_cmd_start:
    mov ah, 0x10            ;get character
    int 0x16

    cmp al, 0x00            ;check if extended key
    je _extended_key
    cmp al, 0xE0            ;check if new extended key
    je _extended_key

    cmp al, 0x08            ;check if backspace pressed
    je _backspace_key

    cmp al, 0x0D            ;check if Enter pressed
    je _enter_key
```

```
mov bh, [cmdMaxLen]         ;check if maxlen reached
mov bl, [cmdChrCnt]
cmp bh, bl
je      _get_cmd_start

;add char to buffer, display it and start again
mov [di], al                ;add char to buffer
inc di                             ;increment buffer pointer
inc BYTE [cmdChrCnt]  ;inc count

mov ah, 0x0E                ;display character
mov bl, 0x07
int 0x10
jmp     _get_cmd_start

_extended_key:              ;extended key - do nothing now
jmp _get_cmd_start

_backspace_key:
mov bh, 0x00                ;check if count = 0
mov bl, [cmdChrCnt]
cmp bh, bl
je      _get_cmd_start      ;yes, do nothing

dec BYTE [cmdChrCnt]  ;dec count
dec di

;check if beginning of line
mov     ah, 0x03            ;read cursor position
mov bh, 0x00
```

```
          int 0x10

          cmp dl, 0x00
          jne       _move_back
          dec dh
          mov dl, 79
          mov ah, 0x02
          int 0x10

          mov ah, 0x09              ; display without moving cursor
          mov al, ' '
mov bh, 0x00
mov bl, 0x07
          mov cx, 1                           ; times to display
int 0x10
          jmp _get_cmd_start

          _move_back:
          mov ah, 0x0E              ; BIOS teletype acts on backspace!
mov bh, 0x00
mov bl, 0x07
int 0x10
          mov ah, 0x09              ; display without moving cursor
          mov al, ' '
mov bh, 0x00
mov bl, 0x07
          mov cx, 1                           ; times to display
int 0x10
          jmp _get_cmd_start
```

```
        _enter_key:
        mov BYTE [di], 0x00
        ret

_split_cmd:
        ;adjust si/di
        mov si, strUserCmd
        ;mov di, strCmd0

        ;move blanks
        _split_mb0_start:
        cmp BYTE [si], 0x20
        je _split_mb0_nb
        jmp _split_mb0_end

        _split_mb0_nb:
        inc si
        jmp _split_mb0_start

        _split_mb0_end:
        mov di, strCmd0

        _split_1_start:                 ;get first string
        cmp BYTE [si], 0x20
        je _split_1_end
        cmp BYTE [si], 0x00
        je _split_1_end
        mov al, [si]
        mov [di], al
        inc si
inc di
jmp _split_1_start

_split_1_end:
mov BYTE [di], 0x00

;move blanks
_split_mb1_start:
cmp BYTE [si], 0x20
je _split_mb1_nb
jmp _split_mb1_end

_split_mb1_nb:
inc si
jmp _split_mb1_start

_split_mb1_end:
mov di, strCmd1

_split_2_start:                ;get second string
cmp BYTE [si], 0x20
je _split_2_end
cmp BYTE [si], 0x00
je _split_2_end
mov al, [si]
mov [di], al
inc si
inc di
jmp _split_2_start
```

```
_split_2_end:
mov BYTE [di], 0x00

;move blanks
_split_mb2_start:
cmp BYTE [si], 0x20
je _split_mb2_nb
jmp _split_mb2_end

_split_mb2_nb:
inc si
jmp _split_mb2_start

_split_mb2_end:
mov di, strCmd2

_split_3_start:                  ;get third string
cmp BYTE [si], 0x20
je _split_3_end
cmp BYTE [si], 0x00
je _split_3_end
mov al, [si]
mov [di], al
inc si
inc di
jmp _split_3_start

_split_3_end:
mov BYTE [di], 0x00
```

```
;move blanks
_split_mb3_start:
cmp BYTE [si], 0x20
je _split_mb3_nb
jmp _split_mb3_end

_split_mb3_nb:
inc si
jmp _split_mb3_start

_split_mb3_end:
mov di, strCmd3

_split_4_start:                         ;get fourth string
cmp BYTE [si], 0x20
je _split_4_end
cmp BYTE [si], 0x00
je _split_4_end
mov al, [si]
mov [di], al
inc si
inc di
jmp _split_4_start

_split_4_end:
mov BYTE [di], 0x00

;move blanks
_split_mb4_start:
cmp BYTE [si], 0x20
```

```
	je _split_mb4_nb
	jmp _split_mb4_end

	_split_mb4_nb:
	inc si
	jmp _split_mb4_start

	_split_mb4_end:
	mov di, strCmd4

	_split_5_start:				;get last string
	cmp BYTE [si], 0x20
	je _split_5_end
	cmp BYTE [si], 0x00
	je _split_5_end
	mov al, [si]
	mov [di], al
	inc si
	inc di
	jmp _split_5_start

	_split_5_end:
	mov BYTE [di], 0x00

	ret

_display_space:
	mov ah, 0x0E				; BIOS teletype

	mov al, 0x20
```

```
  mov bh, 0x00             ; display page 0
  mov bl, 0x07            ; text attribute
  int 0x10              ; invoke BIOS
       ret

_display_endl:
       mov ah, 0x0E         ; BIOS teletype acts on newline!
  mov al, 0x0D
       mov bh, 0x00
  mov bl, 0x07
  int 0x10
       mov ah, 0x0E         ; BIOS teletype acts on linefeed!
  mov al, 0x0A
       mov bh, 0x00
  mov bl, 0x07
  int 0x10
       ret

_display_prompt:
       mov si, Prompt
       mov al, 0x01
       int 0x21
       ret

[SEGMENT .data]
  WelcomeMessage   db  "Welcome to JOSH V1.0 OS Edited by Dammina", 0x00
       Prompt      db      "100466H $ ", 0x00
       cmdMaxLen          db      255                 ;maximum length of commands

       OSName             db      "JOSH", 0x00     ;OS details
```

```
    MajorVersion            db          "1", 0x00
    MinorVersion            db          ".00", 0x00
    MyName                              db          "Dammina Sahabandu",0x00

    TAB             db      "           ",0x00
    EndOfLine       db      "_______________________________________________",0x00
    SystemHardware          db          "System Hardware Info:",0x00
    ProcessorInfo   db      "Processor Info:",0x00
    RAMinfo                 db          "Ram Info:",0x00
    RAMsize         db      "RAM Size (*64KB) :",0x00
    CPUvendor       db      "111111111111",0x00
    ProcessorType   db      "$$$$$$$$$$$$$$$$",0x00
    ProcessorType2  db      "$$$$$$$$$$$$$$$$",0x00
    ProcessorType3  db      "$$$$$$$$$$$$$$$$",0x00
    SystemTime                    db "Sys. Time - ",0x00
    space           db ", Time -",0x00
fmt_12_24 db 0              ; Non-zero = 24-hr format
    fmt_date        db 0, '/' ; 0, 1, 2 = M/D/Y, D/M/Y or Y/M/D
                                        ; Bit 7 = use name for months
                                        ; If bit 7 = 0, second byte = separator character



    cmdVer                  db          "ver", 0x00                 ; internal commands
    cmdExit                 db          "ext", 0x00
    cmdName                             db          "name", 0x00
    cmdHardware             db          "hw",0x00

    txtVersion              db          "version", 0x00 ;messages and other strings
    msgUnknownCmd           db          "Unknown command or bad file name!", 0x00
```

```
[SEGMENT .bss]
        strUserCmd      resb    256             ;buffer for user commands
        cmdChrCnt       resb    1               ;count of characters
        strCmd0                 resb    256             ;buffers for the command components
        strCmd1                 resb    256
        strCmd2                 resb    256
        strCmd3                 resb    256
        strCmd4                 resb    256

;********************end of the kernel code********************
```