26th September 2021

New Kernel

Hi,
A new modified/remastered source code has been added as NEW KERNEL [https://github.com/pritamzope/OS/tree/master/NEW%20KERNEL] to GitHub(Link [https://github.com/pritamzope/OS]) by separating each start to end procedure.
Compilation scripts are replaced with Makefile.
Assembly code is replaced with NASM assembly rather than GNU AS.

Requirements:

```
$ sudo apt-get install make nasm gcc xorriso qemu qemu-system-x86 qemu-system-i386 virtualbox
```

Step by Step Order:-

1. **Console:** simple console for printings and basic string operations(Console [https://github.com/pritamzope/OS/tree/master/NEW%20KERNEL/Console])
2. **GDT:** loading of Global Descriptor Table(GDT [https://github.com/pritamzope/OS/tree/master/NEW%20KERNEL/GDT])
3. **IDT:** Loading of Interrupt Descriptor Table with 8259 PIC(IDT [https://github.com/pritamzope/OS/tree/master/NEW%20KERNEL/IDT])
4. **TSS:** Loading of Task State Segment(TSS)(TSS [https://github.com/pritamzope/OS/tree/master/NEW%20KERNEL/TSS])
5. **Keyboard:** PS/2 Keyboard (Keyboard [https://github.com/pritamzope/OS/tree/master/NEW%20KERNEL/Keyboard])
6. **Terminal:** A simple terminal for support of help, cpuid, echo commands with backtracking(Terminal [https://github.com/pritamzope/OS/tree/master/NEW%20KERNEL/Terminal])
7. **Timer:** Simple Programmable Timer (Timer [https://github.com/pritamzope/OS/tree/master/NEW%20KERNEL/Timer])
8. **Mouse:** PS/2 Mouse working on console (Mouse [https://github.com/pritamzope/OS/tree/master/NEW%20KERNEL/Mouse])
9. **FPU:** Floating Point Unit (FPU [https://github.com/pritamzope/OS/tree/master/NEW%20KERNEL/FPU])
10. **Memory Info:** Memory information such as finding available memory on the system, memory map (Memory Info [https://github.com/pritamzope/OS/tree/master/NEW%20KERNEL/Memory%20Info])
11. **Physical Memory Manager:** A bitmap allocator Physical Memory Manager(PMM [https://github.com/pritamzope/OS/tree/master/NEW%20KERNEL/Physical%20Memory%20Manager])
12. **KHeap:** Kernel Heap management, kmalloc, kcalloc, kfree (KHeap [https://github.com/pritamzope/OS/tree/master/NEW%20KERNEL/KHeap])
13. **Paging:** Simple single Paging (KHeap [https://github.com/pritamzope/OS/tree/master/NEW%20KERNEL/KHeap])
14. **ATA read/write:** An ATA drive(Harddisk) read/write oprations. writing Hello World and simple struct example to drive (ATA [https://github.com/pritamzope/OS/tree/master/NEW%20KERNEL/ATA])
15. **BIOS32:** Calling BIOS Interrupts an get their results (BIOS32 [https://github.com/pritamzope/OS/tree/master/NEW%20KERNEL/BIOS32])
16. **VGA:** A simple VGA graphics with mouse control (VGA [https://github.com/pritamzope/OS/tree/master/NEW%20KERNEL/VGA])
17. **VESA VBE:** VESA BIOS Extensions Graphics (VESA VBE [https://github.com/pritamzope/OS/tree/master/NEW%20KERNEL/VESA%20VBE])
18. **Bitmap Text:** Drawing a Bitmap ASCII characters text using VESA BIOS Extensions Graphics (Bitmap Text [https://github.com/pritamzope/OS/tree/master/NEW%20KERNEL/Bitmap%20Text])

Posted 26th September 2021 by PritamZope

1 View comments

https://github.com/pritamzope/OS/tree/master/Global_Descriptor_Table
[https://github.com/pritamzope/OS/tree/master/Global_Descriptor_Table]

Setting up Interrupt Descriptor Table in both x86 assembly and in C.

https://github.com/pritamzope/OS/tree/master/Interrupt_Descriptor_Table
[https://github.com/pritamzope/OS/tree/master/Interrupt_Descriptor_Table]

Posted 1st November 2019 by PritamZope

4 View comments

15th June 2019

OS using xlang compiler

xlang is the high level language compiler for Intel x86. Download and install it from following link.

https://github.com/pritamzope/xlang [https://github.com/pritamzope/xlang]

In the program, high-level language features with inline assembly.
Here's the simple bootloader program that uses BIOS interrupts with some key demo using xlang.

bootloader.x

```
asm{
  "[bits 16]",
  "[org 0x7c00]"
}
//global character to print
char pchar;pchar = 0;
//gotoxy positions
char x_pos, y_pos;
x_pos = 0; y_pos = 0;
//enter key
char enter_key; enter_key = 0x1c;
//global array
char vbuffer[32];
//bootloader starting point
void start()
{
  char ch;
  clear_screen();
  //print each alphabet from A to Z
  for(ch = 'A'; ch <= 'Z'; ch++){
    pchar = ch;
    print_char();
  }
  //goto next line
  goto_newline();
  //assign Type: characters to array vbuffer
  vbuffer[0] = 'T';
  vbuffer[1] = 'y';
  vbuffer[2] = 'p';
  vbuffer[3] = 'e';
  vbuffer[4] = ':';
  vbuffer[5] = 0;
  //print whole array
  print_array();
```

```
void print_array()
{
  char i;
  for(i = 0; i < 32; i++){
    pchar = vbuffer[i];
    if(pchar == 0){
      break;
    }else{
      print_char();
    }
  }
}
/*get key by calling read_key()
 if key is enter_key then goto nextline
 otherwise print that character
*/
void input_test()
{
  short key;
  char key_code;
  while(1){
    key = read_key();
    pchar = (char)key;
    key = key & 240;
    key_code = key;
    if(key_code == enter_key){
      goto_newline();
    }else{
      print_char();
    }
  }
}
void goto_newline()
{
  y_pos++;
  x_pos = 0;
  gotoxy();
}
//call BIOS interrupt to read key
//result is always in eax register, so no return statement
short read_key()
{
  asm{
    "\tmov ax,0x00",
    "\tint 0x16"
  }
}
//BIOS interrupt to goto x,y pos
void gotoxy()
{
  asm{    "\tmov ah, 0x02",
    "\tmov bh, 0x00",
    "\tmov dl, %" ["=m"(x_pos):],
    "\tmov dh, %" ["=m"(y_pos):],
    "\tint 0x10"
  }
}
//print character using BIOS interrupt
void print_char()
{
  asm{
    "\txor bx, bx",
    "\tmov bl, 10",
    "\tmov al, %" ["=m"(pchar):],
    "\tmov ah, 0x0E",
    "\tint 0x10"
  }
}
void clear_screen()
{
```

```
}
//define bootloader signature
asm{
   "times (510 - ($ - $$)) db 0x00",
   "dw 0xAA55"
}
```

Compile :
Compile above code using xlang compiler, run following command on terminal and produce .asm assembly file.

```
xlang -S bootloader.x
```

Assemble :
Assemble the generated bootloader.asm file using NASM.

```
nasm -fbin bootloader.asm
```

Run on QEMU :
Run bootloader filr on qemu.

```
qemu-system-x86_64 bootloader
```

See more examples here

https://github.com/pritamzope/xlang/tree/master/examples
[https://github.com/pritamzope/xlang/tree/master/examples]

Posted 15th June 2019 by PritamZope

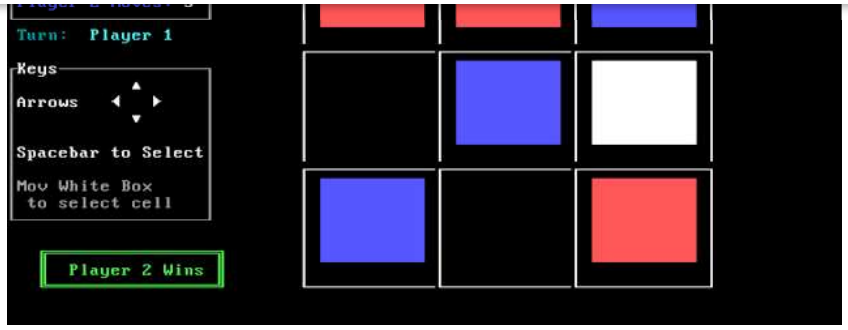
2 View comments

Tic-Tac-Toe Game

7th April 2019

Tic-Tac-Toe Game in kernel using Boxes.
How to Play :
Use arrow keys(UP,DOWN,LEFT,RIGHT) to move white box between cells and press SPACEBAR to select that cell.
RED color for player 1 Box and BLUE color for player 2 box.
See Turn for which player has a turn to select cell.

https://github.com/pritamzope/OS/tree/master/Tic-Tac-Toe
[https://github.com/pritamzope/OS/tree/master/Tic-Tac-Toe]



[https://3.bp.blogspot.com/-3hSxM8sC9Is/XKnn0R4kUEI/AAAAAAAABHc/XzP6RK4WQzcpRJ-m7mCyRkt4UbpJNLJcQCLcBGAs/s1600/tic_tac_toe.png]

Posted 7th April 2019 by PritamZope

1 View comments

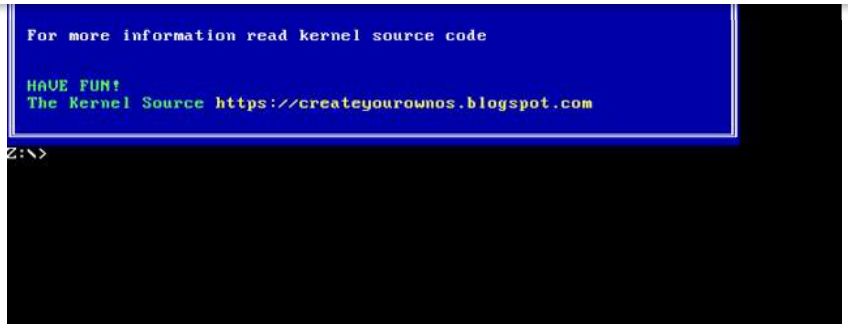
1st April 2019

GUI

Creating 16-Bit application GUI in C such as drawing Boxes and creating DOSBox GUI.

https://github.com/pritamzope/OS/tree/master/GUI
[https://github.com/pritamzope/OS/tree/master/GUI]

[https://1.bp.blogspot.com/-UReYNn5TCzE/XKH_4ojR_7I/AAAAAAAABG4/owsm9GcsDwMfwvGvP-CPXNUuyc1OecTBACLcBGAs/s1600/box_demo_kernel.png]



[https://3.bp.blogspot.com/-cPqc3tIqt_E/XKH_4vPVLLI/AAAAAAAABG8/1ttv3Ea8JjM7JhNGw4JyUf6wPXd9kDfpACLcBGAs/s1600/dosbox_gui.png]

Posted 1st April 2019 by PritamZope

0 Add a comment

31st March 2019

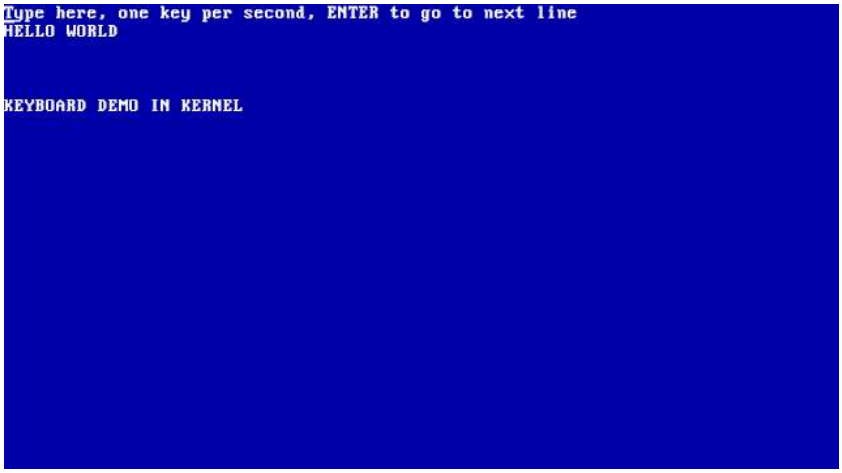
Keyboard I/O in Kernel

Handling Keyboard I/O using port numbers and in/out instructions in C.
Where one key is handled per second, change time of **sleep(timer_count)** function called in **test_input()** function.
test_input() function reads keycode from I/O, gets its corresponding ASCII code and displays it.

https://github.com/pritamzope/OS/tree/master/Kernel/Keyboard
[https://github.com/pritamzope/OS/tree/master/Kernel/Keyboard]

To Run kernel in VirtualBox :

1) Click on **New**
2) Enter **Name, Type: Other, Version: Other/Unknown**
3) Memory size : keep it as it is
4) Click next until **File location and size** dialog, select **64.00 MB** size
5) Click create
6) Goto **Settings -> System -> Acceleration** and disable Enable VT-x/AMD-V or any other graphics card support
7) Goto **Settings -> Storage** select ISO image of kernel.
8) Click OK and Start machine



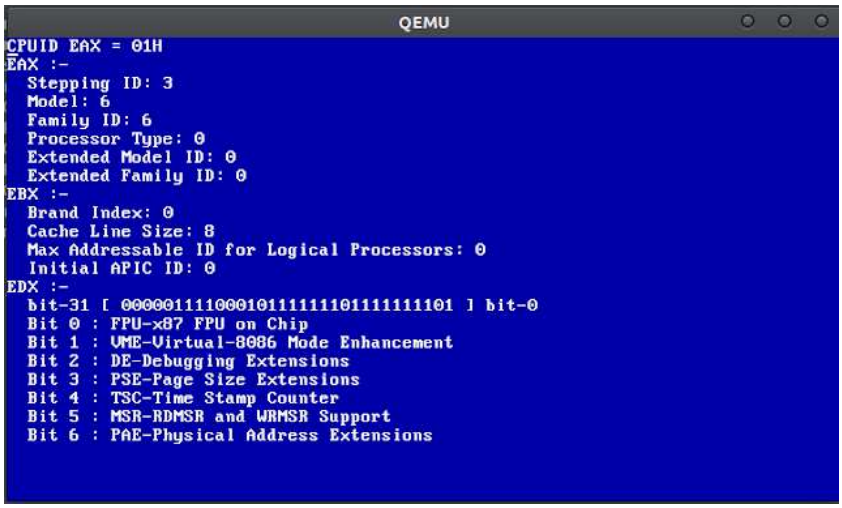
[https://2.bp.blogspot.com/-7ncqD5Rp5-E/XKB37nGstII/AAAAAAAABGY/HfVL37GuuFYyF8hM8IlbufsmBjPSbhCHgCLcBGAs/s1600/keyboard_demo.png]

25th March 2019

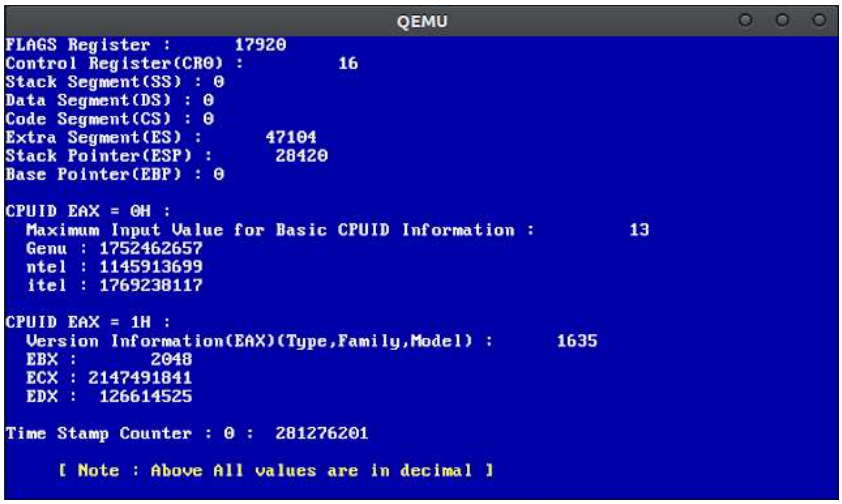
CPU Information

Read CPU information using CPUID instruction in Assembly as well as C Kernel code. See Intel® 64 and IA-32 Architectures Software Developer's Manual [https://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developer-instruction-set-reference-manual-325383.pdf] for about information about each instruction used.(see cpuid, lahf, ...)

https://github.com/pritamzope/OS/tree/master/CPUInfo
[https://github.com/pritamzope/OS/tree/master/CPUInfo]



[https://1.bp.blogspot.com/-VH1KHG7Qg44/XMbRcui9naI/AAAAAAAABIs/tp43MOIJQmM00ZaNzUjfkR9-95PihCY-wCLcBGAs/s1600/c_cpuinfo.png]
CPU Info in C



[https://4.bp.blogspot.com/-bzKr_wYqoXY/XJj3h3RtNmI/AAAAAAAABF4/dCVHDu6k_u843go1-zYeK2v8UOlYShTqQCLcBGAs/s1600/cpuinfo.png]
CPU Info in Assembly Code

Posted 25th March 2019 by PritamZope

30th December 2018 Graphics at low level in OS

Graphics

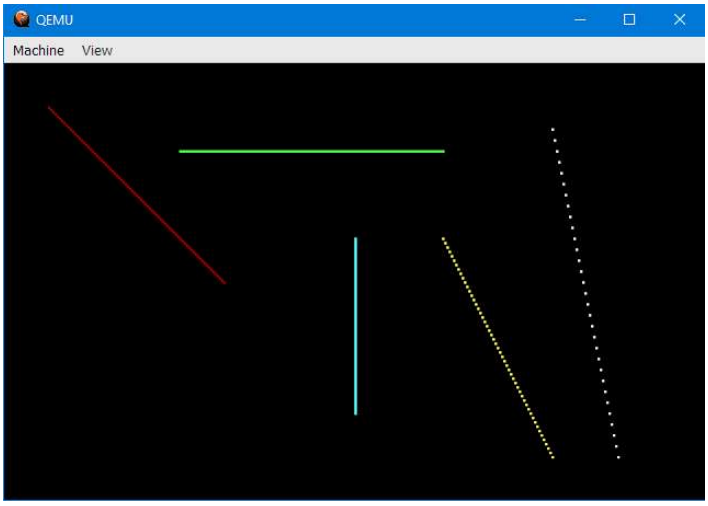
https://github.com/pritamzope/OS/tree/master/Graphics
[https://github.com/pritamzope/OS/tree/master/Graphics]

https://github.com/pritamzope/OS/tree/master/VGA
[https://github.com/pritamzope/OS/tree/master/VGA]

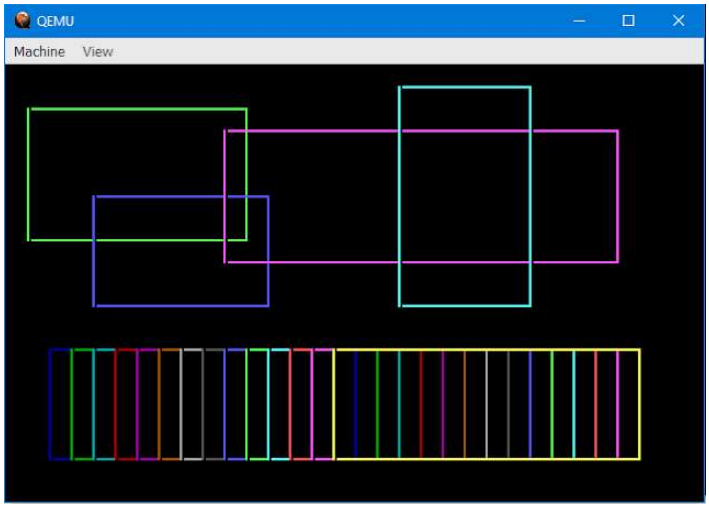
vga_hello.asm : This source is same as we did in Kernel.c source. It points to VGA addreess 0xB800, with colors 4 bits, and 80*20 resolution font.

color_change.asm : Background colors changes as any key is pressed.

rotate_text.asm : Rotate the text to the right position with changing fore and back color by incrementing VGA index as any key is pressed.



[https://2.bp.blogspot.com/-Tto-dMkodBA/XCjXCQLp87I/AAAAAAAABDs/pTzbxc6_8kIQLZQixNlevyl9Ik0HCM04ACEwYBhgL/s1600/dda_line_draw.png]



[https://4.bp.blogspot.com/-K0bD8h0aEUI/XCjXDzofhBI/AAAAAAAABDw/GwMpHGgdzH4YjiQmURao2t5lu-t8Jpo5wCEwYBhgL/s1600/rectangle.png]

Posted 30th December 2018 by PritamZope

9th August 2018

Create Kernel In C

Go to following links to understand how to create a kernel in C and assembly.

Create Your Own Kernel In C [https://www.codeproject.com/Articles/1225196/Create-Your-Own-Kernel-In-C]

Create Your Own Kernel [https://www.c-sharpcorner.com/article/create-your-own-kernel/]

Watch Video Here [https://www.youtube.com/watch?v=4hJDOvwbTZs&t=3228s]

https://github.com/pritamzope/OS/tree/master/Kernel [https://github.com/pritamzope/OS/tree/master/Kernel]

Download Code kernel.zip [https://www.dropbox.com/s/9frh7k3uzg67kdt/kernel.zip?dl=0]

There are 3 parts of it. each extending the previous one.

[https://1.bp.blogspot.com/-urZ8pLeGSAo/W2vtkiukg_I/AAAAAAAAA-c/slF_Rse6oDcfdyAGExmsTHc1kkQ3VbT5wCLcBGAs/s1600/kernel_3_output_2.png]

Posted 9th August 2018 by PritamZope

7 View comments

21st February 2017

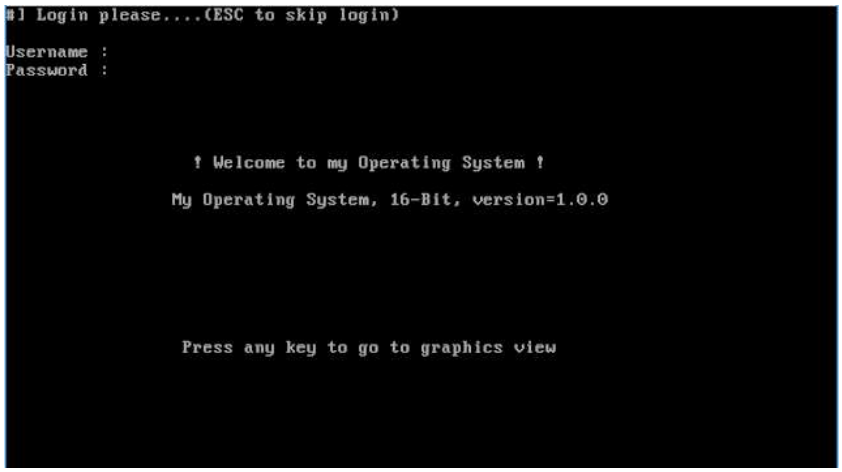
Creating Your Own Operating System

Creating Your Own Operating System

In this blog we will write our own bootloader using 16-bit assembly language to create our own

Download ISO images [https://www.dropbox.com/s/a5jxp9k0dw02vu0/iso_images.zip?dl=0]

[https://2.bp.blogspot.com/-StK4XCJLVPA/WKyUPuK_HOI/AAAAAAAAAXs/yWlxIiYPt0IvjCw67JHneEIDYR9rPqIKACLcB/s1600/extended_os_1.png]



[https://1.bp.blogspot.com/-3xSXpTOSwWU/WKyUPqkb0DI/AAAAAAAAAXo/eos8S2uGww0W_alE6xIGFMBh3CD9GFJGgCLcB/s1600/extended_os_2.png]

[https://3.bp.blogspot.com/-PQKIwwOx9iE/WKyUPmg0OSI/AAAAAAAAAXw/-3wTAUC_RJQyj0btQzFS2Vyjk6YKzFzrwCLcB/s1600/extended_os_3.png]

Writing an operating system is the most complicated task in the world of programming.

The first part of operating system is the Bootloader.
Bootloader is a piece of program that runs before any operating system is running.
it is used to boot other operating systems, usually each operating system has a set of bootloaders specific for it.
Bootloaders usually contain several ways to boot the OS kernel and also contain commands for debugging and/or modifying the kernel environment.

We will create 3 stage OS. First is to just display message on the screen with colors, second is to take input from user, and third for drawing.
The bootloaders are generally written in 16-bit assembly(also called Real mode), then the bits can be extended to 32-bit(Protected mode).
So the bootloaders must be written in 16-bit assembly.
Before you move to next, you must have some knowledge about 16-bit assembly language.

Requirements

You need an assember that can convert your assembly instructions into raw binary format and an simulator to view.
I am using **Nasm** assember and **Qemu** simulator.

For Linux, type following commands to install nasm & qemu(Quick Emulator)

sudo apt-get install nasm
sudo apt-get install qemu qemu-system-x86_64

For Windows,download them from following sites and install them.

- http://www.nasm.us/ [http://www.nasm.us/]
- https://qemu.weilnetz.de/ [https://qemu.weilnetz.de/]

Coding

Starting of coding always comes with printing Hello World, isn't it ?
Here's the code that prints Hello World! on screen.
(file = hello_world.asm)

hello_world.asm

```
[bits 16]         ; tell assembler that working in real mode(16 bit mode)
[org 0x7c00]      ; organize from 0x7C00 memory location where BIOS will load us


start:            ; start label from where our code starts


   xor ax,ax          ; set ax register to 0
   mov ds,ax          ; set data segment(ds) to 0
   mov es,ax          ; set extra segment(es) to 0
   mov bx,0x8000


   mov si, hello_world            ; point hello_world to source index
   call print_string                ; call print different color string function



   hello_world db  'Hello World!',13,0


print_string:
   mov ah, 0x0E          ; value to tell interrupt handler that take value from al & print it

.repeat_next_char:
```

```
.done_print:
    ret                     ;return

    times (510 - ($ - $$)) db 0x00    ;set 512 bytes for boot sector which are necessary
    dw 0xAA55                                   ; boot signature 0xAA & 0x55
```

OK now, what the hell is this ?

[bits 16] : This line tells the assember you are working in 16-bit real mode.
it will convert assembly data to 16-bit binary form.

[org 0x7c00] : This is assember directive. 0x7c00 is the memory location where BIOS will load us.

xor ax,ax
mov ds,ax
mov es,ax
mov bx,0x8000

First setting the registers to zero such as ax,ds,es which we will used further .
Then we will copy memory location 0x8000 to bx register
because we want to perform operations/instructions because we are loaded at 0x7c00 memory location.
we need memory location above it.

hello_world db 'Hello World!',13,0 : this line defines the string with label hello_world,
where 13 is New line and 0 is end of string.

mov si, hello_world
call print_string

Pointing first character of hello_world string to source index(si) register and then call print_string function.
Copying 0x0E to ah register.
this will tell to interrupt handler that take value/ASCII character from al & print it using int 0x10.

AH = 0x0E
AL = character
BH = page
BL = color (graphics mode)
int 0x10

.repeat_next_char : Label for continue to loop until end of string occurs.

lodsb : This instruction loads the first character from si to al register using ASCII code.

Then we will compare whether al contains 0 or not,if not then print it and jump to loop,
otherwise jump to .done_print.

int 0x10 : This is BIOS video interrupt which takes char value from al register & print it.

times (510 - ($ - $$)) db 0x00 : A boot sector always be a 512 byte. starting with address 0x00.
because on hard drive,there are only 512 bytes of sectors.

dw 0xAA55 : This is the magic number of bootable device.
This line is boot signature that makes our code to bootable code.
it defines word 0xAA & 0x55.
These are last two bytes of our first sector.
because of this number,BIOS loads us at 0x7c00 location when computer starts.

For Linux,Type following command to compile file

nasm -f bin hello_world.asm -o myos.bin

Once file is compiled successfully and myos.bin file is created, then run it in qemu.

Consider i have file in C:\Users\Pritam\Documents\temp folder.

```
nasm.exe -f bin "C:\Users\Pritam\Documents\temp\hello_world.asm"
                                    -o "C:\Users\Pritam\Documents\temp\myos.bin"
```

and to run in qemu,

```
"C:\Program Files\qemu\qemu-system-x86_64.exe"  "C:\Users\Pritam\Documents\temp\myos.bin"
```

where i have installed qemu.

Here i have created .bin file, but you can also create .iso file.

Once it successfully prints Hello World! then attach Secondary device/USB drive and boot .bin/.iso in it.
You can use dd command on linux or can use rufus software on windows.
Output of above program is

[https://3.bp.blogspot.com/-7-BBXRWCcFY/WKyUQKqy2ZI/AAAAAAAAAX4/_jtim6FGQkc7Xa1NS1lkE7ASs-uwCdbUACEw/s1600/hello_world_os.png]

Without BIOS :

Without BIOS

In above program we used BIOS interrupt.
To display HelloWorld on the screen without BIOS see following link :

HelloWorld_without_BIOS [https://github.com/pritamzope/OS/tree/master/Bootloader/HelloWorld_without_BIOS]

Watch video here for how to implement above procedure for both Linux & Windows :

Create Your Own Operating System (OS)

To print a string on screen at specific location or to set cursor at specific location use following actions.

AH = 0x02
BH = page
DH = row
DL = column

e.g:

```
mov ah,0x02        ; set value for change to cursor position
mov bh,0x00        ; page
mov dh,0x06        ; y cordinate/row
mov dl,0x05        ; x cordinate/col
int 0x10

mov si, hello_world
call print_string
```

To get input first set ax to 0x00 and call int 0x16.
To display character which has been input,mov ah to 0x0E and call int 0x10.
It will store char value to al register & key code to ah register.

e.g:

```
inputLoop:
mov ax,0x00
int 0x16

cmp ah,0x1C          ; compare input is enter(1C) or not
je .inputLoop
cmp al,0x61          ; compare input is character 'a' or not
je exitLoop

mov ah,0x0E          ;display input char
int 0x10
exitLoop:
ret
```

As described above that every sector has size only 512 bytes,

so if you write code which is taking more than 512 bytes it will not work or assembler will give you an error.
So to use more memory, you need to load/read next sector into main memory.
To load/read sectors in main memory,

AH = sector number(1,2,3 etc)[1 is already taken by our bootloader]
AL = number of sectors to read
DL = type of memory from where to read(0x80 is for hard drive/USB drive)
CH = cylinder number
DH = head number
CL = sector number
BX = memory location where to jump after loaded
int 0x13 = Disk I/O interrupt

Then jump to your memory location(label in assembly).

e.g:

```
; load second sector from memory

mov ah, 0x02              ; load second stage to memory
mov al, 1                 ; numbers of sectors to read into memory
mov dl, 0x80              ; sector read from fixed/usb disk
mov ch, 0                 ; cylinder number
mov dh, 0                 ; head number
mov cl, 2                 ; sector number
mov bx, _OS_Stage_2       ; load into es:bx segment :offset of buffer
int 0x13                  ; disk I/O interrupt

jmp _OS_Stage_2           ; jump to second stage
```

For clearing the screen,copy 0x13 to ax & call video interrupt.

AX = color
DI = x & y cordinates(y=320 for next line(320*200 display))
[ES:DI] = value of x,y cordinates & color(AX)[segment :offset]

Here's the complete 3 stages OS code

Extended Operating System

```
[bits 16]         ; tell assembler that working in real mode(16 bit mode)
[org 0x7c00]      ; organize from 0x7C00 memory location where BIOS will load us

start:            ; start label from where our code starts

   xor ax,ax          ; set ax register to 0
   mov ds,ax          ; set data segment(ds) to 0
   mov es,ax          ; set extra segment(es) to 0
   mov bx,0x8000

   mov ax,0x13        ;clears the screen
   int 0x10           ;call bios video interrupt

   mov ah,02          ;clear the screen with big font
   int 0x10           ;interrupt display

   ;set cursor to specific position on screen
   mov ah,0x02        ; set value for change to cursor position
   mov bh,0x00        ; page
   mov dh,0x06        ; y cordinate/row
   mov dl,0x09        ; x cordinate/col
   int 0x10

   mov si, start_os_intro            ; point start_os_intro string to source index
   call _print_DiffColor_String      ; call print different color string function

   ;set cursor to specific position on screen
   mov ah,0x02
   mov bh,0x00
   mov dh,0x10
   mov dl,0x06
   int 0x10

   mov si,press_key                  ; point press_key string to source index
   call _print_GreenColor_String     ; call print green color string function

   mov ax,0x00        ; get keyboard input
   int 0x16           ; interrupt for hold & read input

   ;////////////////////////////////////////////////////////////
   ; load second sector into memory

   mov ah, 0x02                ; load second stage to memory
   mov al, 1                   ; numbers of sectors to read into memory
   mov dl, 0x80                ; sector read from fixed/usb disk
   mov ch, 0                   ; cylinder number
   mov dh, 0                   ; head number
   mov cl, 2                   ; sector number
   mov bx, _OS_Stage_2         ; load into es:bx segment :offset of buffer
   int 0x13                    ; disk I/O interrupt

   jmp _OS_Stage_2             ; jump to second stage


   ;////////////////////////////////////////////////////////////
   ; declaring string datas here
   start_os_intro db 'Welcome to My OS!',0
```

```
    display_text db '! Welcome to my Operating System !', 0

    os_info db 10, 'My Operating System, 16-Bit, version=1.0.0',13,0

    press_key_2 db 10,'Press any key to go to graphics view',0

    window_text db 10,'Graphics in OS.....', 0
    hello_world_text db 10,10, '    Hello World!',0
    login_label db '#] Login please....(ESC to skip login)', 0


    ;////////////////////////////////////////////////////////
    ; defining printing string functions here

    ;****** print string without color

print_string:
    mov ah, 0x0E          ; value to tell interrupt handler that take value from al & print it

.repeat_next_char:
    lodsb             ; get character from string
    cmp al, 0             ; cmp al with end of string
    je .done_print            ; if char is zero, end of string
    int 0x10              ; otherwise, print it
    jmp .repeat_next_char       ; jmp to .repeat_next_char if not 0

.done_print:
    ret                   ;return

;****** print string with different colors

_print_DiffColor_String:
        mov bl,1            ;color value
    mov ah, 0x0E

.repeat_next_char:
    lodsb
    cmp al, 0
    je .done_print
    add bl,6            ;increase color value by 6
    int 0x10
    jmp .repeat_next_char

.done_print:
    ret

;****** print string with green color

_print_GreenColor_String:
    mov bl,10
    mov ah, 0x0E

.repeat_next_char:
    lodsb
    cmp al, 0
    je .done_print
    int 0x10
    jmp .repeat_next_char

.done_print:
    ret

;****** print string with white color

_print_WhiteColor_String:
    mov bl,15
    mov ah, 0x0E
```

```
    int 0x10
    jmp .repeat_next_char

.done_print:
    ret

;****** print string with yellow color

_print_YellowColor_String:
    mov bl,14
    mov ah, 0x0E

.repeat_next_char:
    lodsb
    cmp al, 0
    je .done_print
    int 0x10
    jmp .repeat_next_char

.done_print:
    ret


    ;//////////////////////////////////////////
    ; boot loader magic number
    times ((0x200 - 2) - ($ - $$)) db 0x00     ;set 512 bytes for boot sector which are necessary
    dw 0xAA55                                  ; boot signature 0xAA & 0x55


;/////////////////////////////////////////////////////////////////////////////////////////

_OS_Stage_2 :

    mov al,2                ; set font to normal mode
    mov ah,0                ; clear the screen
    int 0x10                ; call video interrupt

    mov cx,0                ; initialize counter(cx) to get input

    ;***** print login_label on screen
    ;set cursor to specific position on screen
    mov ah,0x02
    mov bh,0x00
    mov dh,0x00
    mov dl,0x00
    int 0x10

    mov si,login_label          ; point si to login_username
    call print_string           ; display it on screen

    ;****** read username

    ;set cursor to specific position on screen
    mov ah,0x02
    mov bh,0x00
    mov dh,0x02
    mov dl,0x00
    int 0x10

    mov si,login_username       ; point si to login_username
    call print_string           ; display it on screen

_getUsernameinput:

    mov ax,0x00         ; get keyboard input
    int 0x16            ; hold for input
```

```
    mov ah,0x0E           ;display input char
    int 0x10

    inc cx                ; increase counter
    cmp cx,5              ; compare counter reached to 5
    jbe _getUsernameinput  ; yes jump to _getUsernameinput
    jmp .inputdone        ; else jump to inputdone

.inputdone:
    mov cx,0              ; set counter to 0
    jmp _getUsernameinput  ; jump to _getUsernameinput
    ret                   ; return

.exitinput:
    hlt


    ;****** read password

    ;set x y position to text
    mov ah,0x02
    mov bh,0x00
    mov dh,0x03
    mov dl,0x00
    int 0x10

    mov si,login_password             ; point si to login_username
    call print_string                 ; display it on screen

_getPasswordinput:

    mov ax,0x00
    int 0x16

    cmp ah,0x1C
    je .exitinput

    cmp ah,0x01
    je _skipLogin

    inc cx

    cmp cx,5
    jbe _getPasswordinput

    jmp .inputdone

.inputdone:
    mov cx,0
    jmp _getPasswordinput
    ret
.exitinput:
    hlt

;****** display display_text on screen

    ;set x y position to text
    mov ah,0x02
    mov bh,0x00
    mov dh,0x08
    mov dl,0x12
    int 0x10

    mov si, display_text      ;display display_text on screen
    call print_string
```

```
    int 0x10

    mov si, os_info     ;display os_info on screen
    call print_string

    ;set x y position to text
    mov ah,0x02
    mov bh,0x00
    mov dh,0x11
    mov dl,0x11
    int 0x10

    mov si, press_key_2     ;display press_key_2 on screen
    call print_string

    mov ah,0x00
    int 0x16


;////////////////////////////////////////////////////////////////

_skipLogin:

    ;////////////////////////////////////////////////////////
    ; load third sector into memory

    mov ah, 0x03                ; load third stage to memory
    mov al, 1
    mov dl, 0x80
    mov ch, 0
    mov dh, 0
    mov cl, 3                   ; sector number 3
    mov bx, _OS_Stage_3
    int 0x13

    jmp _OS_Stage_3

;////////////////////////////////////////////////////////////////////////////////////////

_OS_Stage_3:

    mov ax,0x13             ; clears the screen
    int 0x10

;////////////////////////////////////////////////////////
; drawing window with lines

    push 0x0A000              ; video memory graphics segment
    pop es                  ; pop any extar segments from stack
    xor di,di               ; set destination index to 0
    xor ax,ax               ; set color register to zero

    ;////////////////////////////////////////////
    ;******drawing top line of our window
    mov ax,0x02               ; set color to green

    mov dx,0                ; initialize counter(dx) to 0

    add di,320              ; add di to 320(next line)
    imul di,10              ;multiply by 10 to di to set y cordinate from where we need to start drawing

    add di,10               ;set x cordinate of line from where to be drawn

_topLine_perPixel_Loop:

    mov [es:di],ax          ; move value ax to memory location es:di
```

```
    hlt                     ; halt process after drawing

    ;/////////////////////////////////////////////
    ;******drawing bottm line of our window
    xor dx,dx
    xor di,di
    add di,320
    imul di,190         ; set y cordinate for line to be drawn
    add di,10          ;set x cordinate of line to be drawn

    mov ax,0x01         ; blue color

_bottmLine_perPixel_Loop:

    mov [es:di],ax

    inc di
    inc dx
    cmp dx,300
    jbe _bottmLine_perPixel_Loop
    hlt

    ;/////////////////////////////////////////////
    ;******drawing left line of our window
    xor dx,dx
    xor di,di
    add di,320
    imul di,10          ; set y cordinate for line to be drawn

    add di,10           ; set x cordinate for line to be drawn

    mov ax,0x03         ; cyan color

_leftLine_perPixel_Loop:

    mov [es:di],ax

    inc dx
    add di,320
    cmp dx,180
    jbe _leftLine_perPixel_Loop

    hlt

    ;/////////////////////////////////////////////
    ;******drawing right line of our window
    xor dx,dx
    xor di,di
    add di,320
    imul di,10          ; set y cordinate for line to be drawn

    add di,310          ; set x cordinate for line to be drawn

    mov ax,0x06         ; orange color

_rightLine_perPixel_Loop:

    mov [es:di],ax

    inc dx
    add di,320
    cmp dx,180
    jbe _rightLine_perPixel_Loop

    hlt

    ;/////////////////////////////////////////////
```

```
    imul di,27          ; set y cordinate for line to be drawn

    add di,11           ; set x cordinate for line to be drawn

    mov ax,0x05         ; pink color

_belowLineTopLine_perPixel_Loop:

    mov [es:di],ax

    inc di
    inc dx
    cmp dx,298
    jbe _belowLineTopLine_perPixel_Loop

    hlt

    ;***** print window_text & X char

    ;set cursor to specific position
    mov ah,0x02
    mov bh,0x00
    mov dh,0x01         ; y cordinate
    mov dl,0x02         ; x cordinate
    int 0x10

    mov si,window_text              ; point si to window_text
    call _print_YellowColor_String

    hlt

    ;set cursor to specific position
    mov ah,0x02
    mov bh,0x00
    mov dh,0x02          ; y cordinate
    mov dl,0x25          ; x cordinate
    int 0x10

    mov ah,0x0E
    mov al,0x58          ; 0x58=X
    mov bh,0x00
    mov bl,4             ; red color
    int 0x10

    hlt

    ;set cursor to specific position
    mov ah,0x02
    mov bh,0x00
    mov dh,0x02          ; y cordinate
    mov dl,0x23          ; x cordinate
    int 0x10

    mov ah,0x0E
    mov al,0x5F          ; 0x58=X
    mov bh,0x00
    mov bl,9             ; red color
    int 0x10

    hlt

    ;set cursor to specific position
    mov ah,0x02
    mov bh,0x00
    mov dh,0x05   ; y cordinate
    mov dl,0x09   ; x cordinate
    int 0x10
```

```
hlt

;set cursor to specific position
mov ah,0x02
mov bh,0x00
mov dh,0x12   ; y cordinate
mov dl,0x03  ; x cordinate
int 0x10

mov si,display_text
call _print_WhiteColor_String

hlt


; add how much memory we need
times (1024 - ($-$$)) db 0x00
```

Posted 21st February 2017 by PritamZope

26 View comments