ASSEMBLY

MONITORIA INFRAESTRUTURA DE SOFTWARE

2017.2

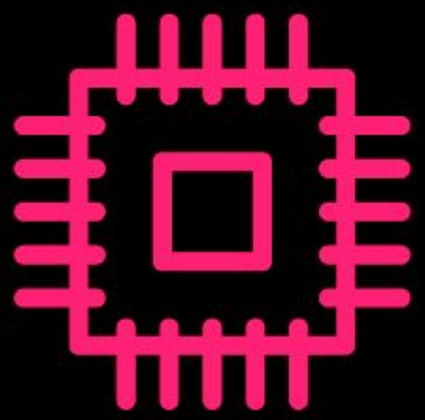
SUMÁRIO

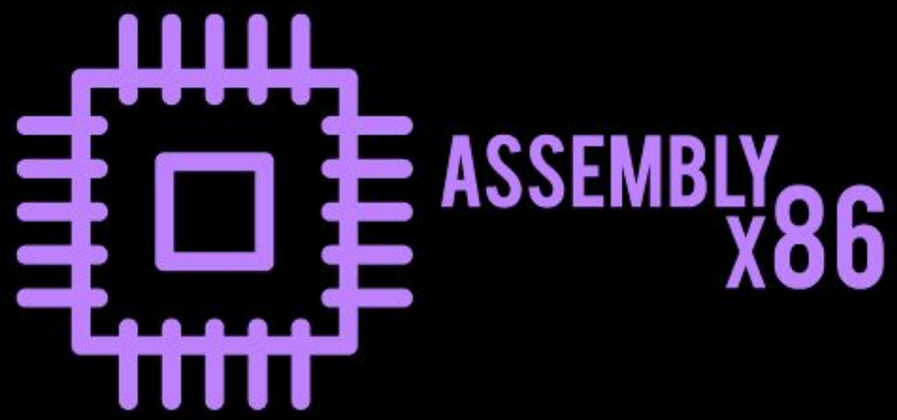
MONITORIA INFRAESTRUTURA DE SOFTWARE
2017.2

O QUE É ASSEMBLY?

Assembly, ou linguagem de montagem, é uma abstração da linguagem de máquina legível para os humanos.

Cada arquitetura de processador tem sua linguagem de montagem.





MONITORIA INFRAESTRUTURA DE SOFTWARE

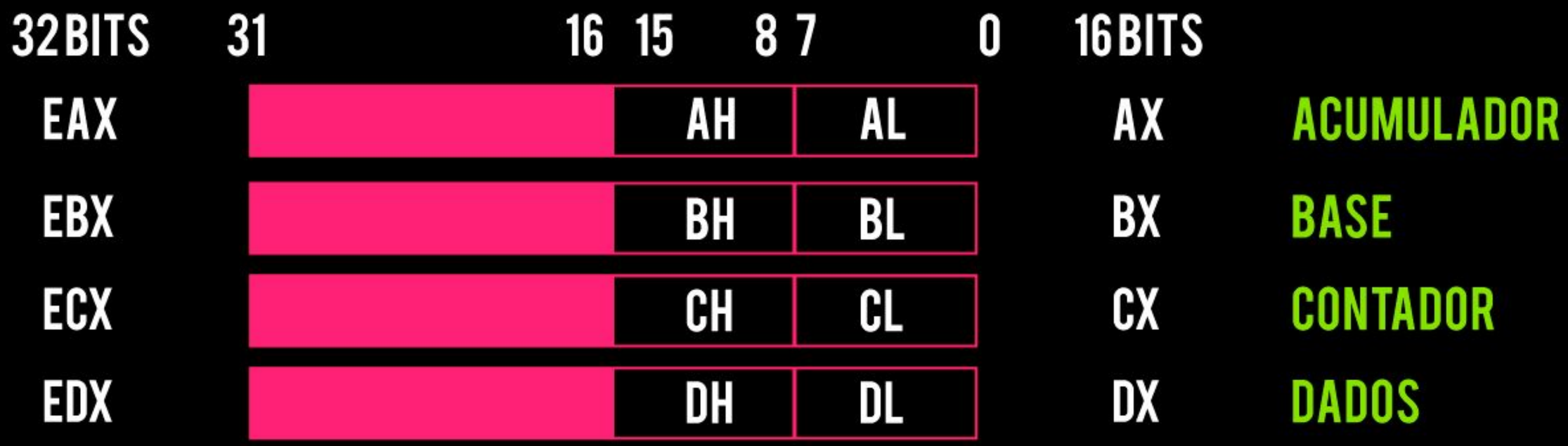
2017.2

O QUE É ASSEMBLY?

Assembly é uma linguagem formada por todas as instruções suportadas por uma arquitetura. Quando um código é “compilado”, o montador transforma cada instrução em seu respectivo valor em linguagem de máquina (bits).

MONITORIA INFRAESTRUTURA DE SOFTWARE
2017.2

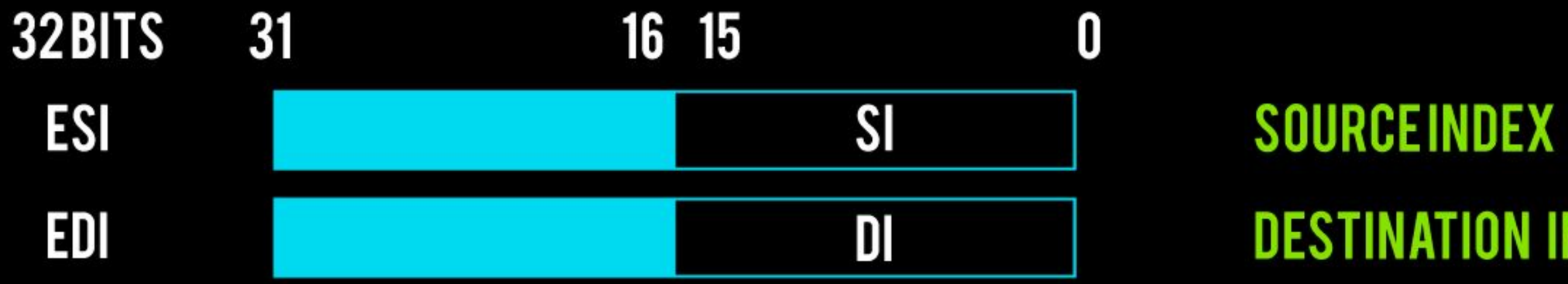
REGISTRADORES



Embora sejam de propósito geral, em algumas instruções os diferentes registradores tem um uso específico

- EAX, EDX – Multiplicação, divisão, operações de E/S
- ECX – Instruções de laço, rotação, deslocamento de bits

REGISTRADORES



Registradores de Índice devem ser utilizados em instruções que manipulam strings, pois estes tipos de instruções geralmente incrementam/decrementam implicitamente estes registradores.

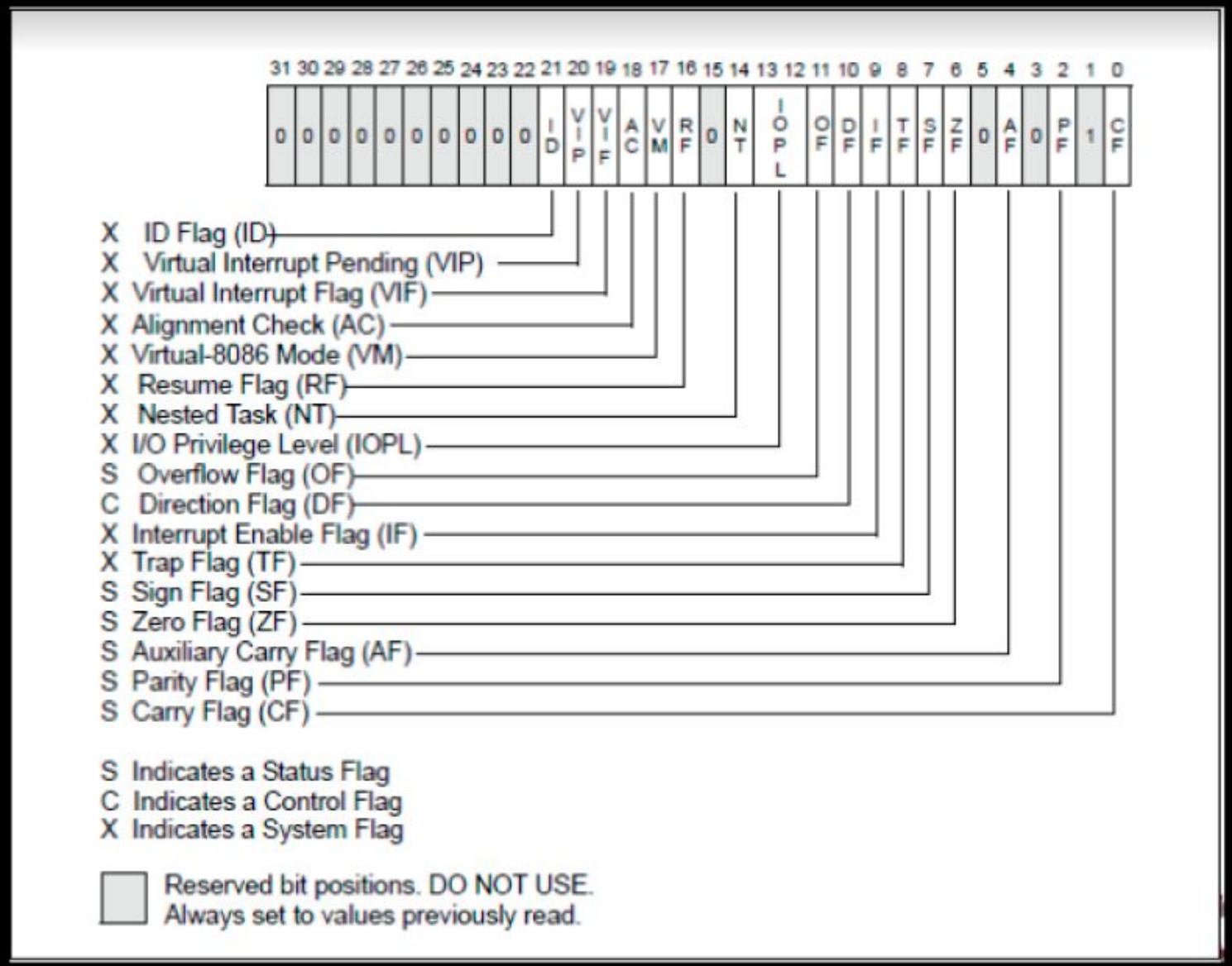
REGISTRADORES

DE SEGMENTO

Servem para controlar e apontar para porções da memória que devem ser acessados

- CS – Busca de instruções
- SS – Operações na pilha, como chamada e retorno de subrotinas
- DS – Ler e escrever dados do programa
- ES, FS, GS – Não tem função específica

FLAGS


31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 ID VIP VIF AC VM RF 0 NT IOPL OF DF IF TF SF ZF 0 AF 0 PF 1 CF
X ID Flag (ID)
X Virtual Interrupt Pending (VIP)
X Virtual Interrupt Flag (VIF)
X Alignment Check (AC)
X Virtual-8086 Mode (VM)
X Resume Flag (RF)
X Nested Task (NT)
X I/O Privilege Level (IOPL)
S Overflow Flag (OF)
C Direction Flag (DF)
X Interrupt Enable Flag (IF)
X Trap Flag (TF)
S Sign Flag (SF)
S Zero Flag (ZF)
S Auxiliary Carry Flag (AF)
S Parity Flag (PF)
S Carry Flag (CF)
S Indicates a Status Flag
C Indicates a Control Flag
X Indicates a System Flag
Reserved bit positions. DO NOT USE.
Always set to values previously read.

MODO REAL

Primeiro modo de operação a ser criado.

Acesso direto às rotinas da BIOS.

Não há mecanismos de proteção aos segmentos de memória, o programador deve ter cuidado para que um segmento não invada o outro.

Sintaxe

Monitoria Infraestrutura de Software
2017.2

Atribuir valor

MOV destino, origem

Copia o conteúdo de origem em destino.
Modos:

destino, origem	exemplo
registrador, registrador	mov ax, cx
registrador, imediato	mov dx, 0xe
registrador, memória	mov cx, [0xe]
memória, registrador	mov [label], bx
memória, imediato	mov byte[si], 2

Equivalente em **C**:
destino = origem;

Obs.: Não pode ser usado de
memória -> memória

Comparação

CMP destino, origem

Equivalente em **C**:
if(destino==origem)

Subtrai o operador origem do destino (destino - origem). Não armazena a operação, apenas seta o estado das flags do registrador de flags.

destino, origem	exemplo
registrador, registrador	cmp ax, dx
registrador, imediato	cmp dx, 0xa
registrador, memória	cmp cx, [0x1e]
memória, registrador	cmp [bx], dx
memória, imediato	cmp byte[si], 23h

Instruções de desvio

JXX endereço

Formatos:
JMP, JE, JNE, JG, JGE, JL, JLE

Funcionamento:
Checa o estado do registrador de FLAGS e põe endereço no registrador IP se a condição for satisfeita.

Bastante usado para desvios condicionais (equivalente a if-else em C)

Exemplo:

```
mov ax, 10
mov cx, 5
cmp ax, cx
jne diferente
add ax, 9
diferente:
add cx, 5
```

Pilha

O registrador ESP/SP é o ponteiro da pilha e seu valor é atualizado a cada operação de inserção ou remoção de valores na pilha.

O ponteiro SP referência o endereço mais elevado (pilha cresce de baixo para cima).

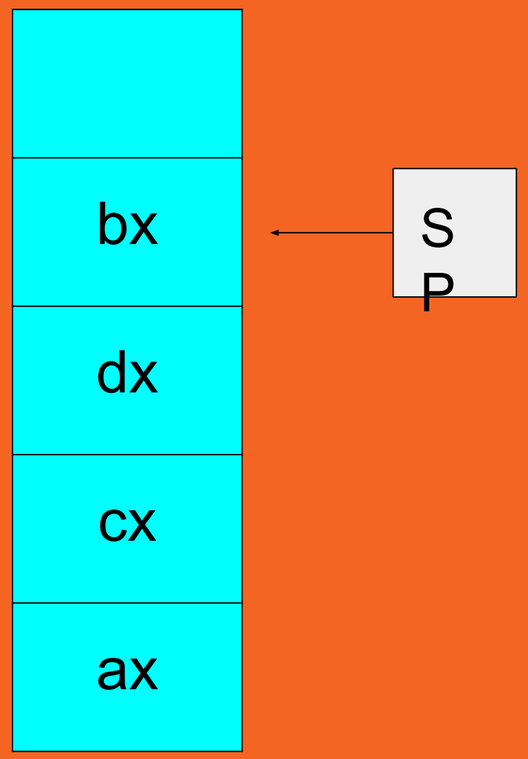
Operações da pilha:
PUSH e POP (e suas variações)

PUSH operando
Coloca o operando no topo da pilha. Incrementa SP.

POP operando
Retira operando do topo da pilha. Decrementa SP.

operando pode ser registrador ou memória.

Exemplo:
push ax
push cx
push dx
push bx



Instrução ‘CALL’

```
CALL
endereço
```

A instrução causa um desvio no código, guardando o endereço da próxima instrução (IP) na pilha e chamando a instrução localizada no endereço passado.

É o equivalente a chamada de uma função (subrotina) em C.

Exemplo:

```
Inicio:
mov ax, 10
mov cx, 5
call Soma
sub ax, 3
jmp Fim


Soma:
add ax, cx
ret

Fim:
```

Instrução ‘RET’

RET

Esta instrução retira o endereço que está no topo da pilha e atualiza EIP/IP com este.

Geralmente é utilizada ao final de uma subrotina chamada por uma instrução “CALL".

OBS.: perceba que o topo da pilha precisa ser o endereço de retorno, portanto, tudo que for empilhado em uma subrotina, deve ser desempilhado antes de usar a instrução.

Exemplo:

```
Inicio:
mov ax, 10
mov cx, 5
call Soma
sub ax, 3
jmp Fim


Soma:
add ax, cx
ret

Fim:
```

Interrupções

Interrupções

Interrompe o procedimento normal de um programa para executar uma outra rotina (rotina de interrupção).

Quando uma interrupção acontece, ela guarda tudo ou uma parte do estado de execução do programa na pilha e pula para executar a rotina de interrupção.

Vídeo - int 10h

Selecionar o modo de vídeo:

```
mov AH, 0     ;Número da chamada
mov AL, 13h ;Modo de vídeo. (VGA 320x200)
int 10h
```

Vídeo - int 10h

Alterar cor de fundo da tela:

```
mov AH, 0xb  ;Número da chamada
mov BH, 0    ;ID da paleta de cores
mov BL, 4    ;Cor desejada (vermelho)
int 10h
```

Atributos de cores da BIOS

QEMU

Pt 6:58 PM

FundoDaTela.asm

```
1  org 0x7c00      ;carrega o programa no bootsector area
2  jmp 0x0000:start
3
4  start:
5      mov ax, 0
6      mov ds, ax
7
8      mov ah, 0     ;Número da chamada
9      mov bh, 13h ;Modo de vídeo. (VGA)
10     int 10h
11
12     mov ah, 0xb  ;Número da chamada
13     mov bh, 0      ;ID da paleta de cores
14     mov bl, 4      ;Cor desejada (vermelho)
15     int 10h
16
17     jmp $
18
19 times 510-($-$$) db 0   ; assinatura de boot,
20 dw 0xaa55
```

QEMU

Line 20, Column 10 | Tab Size: 4 | Assembly x86 (NASM)

Vídeo - int 10h

Imprimir um caractere na tela:

```
mov AH, 0xe ;Número da chamada
mov AL, “a” ;Caractere em ASCII a se escrever
mov BH, 0 ;Número da página.
mov BL, 0xf ;Cor da letra, Branco, apenas em modos gráficos
int 10h
```

QEMU

FundoDaTela.asm × kernel.asm ×

```
3
4   _start:
5
6     mov ax, 0
7     mov ds, ax
8
9     mov ah, 0   ; escolhe o modo de video
10    mov al, 12h ; modo VGA
11    int 10h
12
13    mov ah, 0xb ; escolhe a cor da tela
14    mov bh, 0
15    mov bl, 1h  ; cor da tela (azul escuro)
16    int 10h
17
18
19    mov ah, 0xe ;Número da chamada
20    mov al, "a" ;Caractere em ASCII a se escrev
21    mov bh, 0 ;Número da página.
22    mov bl, 0xf ;branco
23    int 10h
24
25  fim:
26      jmp $
27
28  times 510-($-$$) db 0   ; assinatura de boot,
29  dw 0xaa55
```

QEMU

a

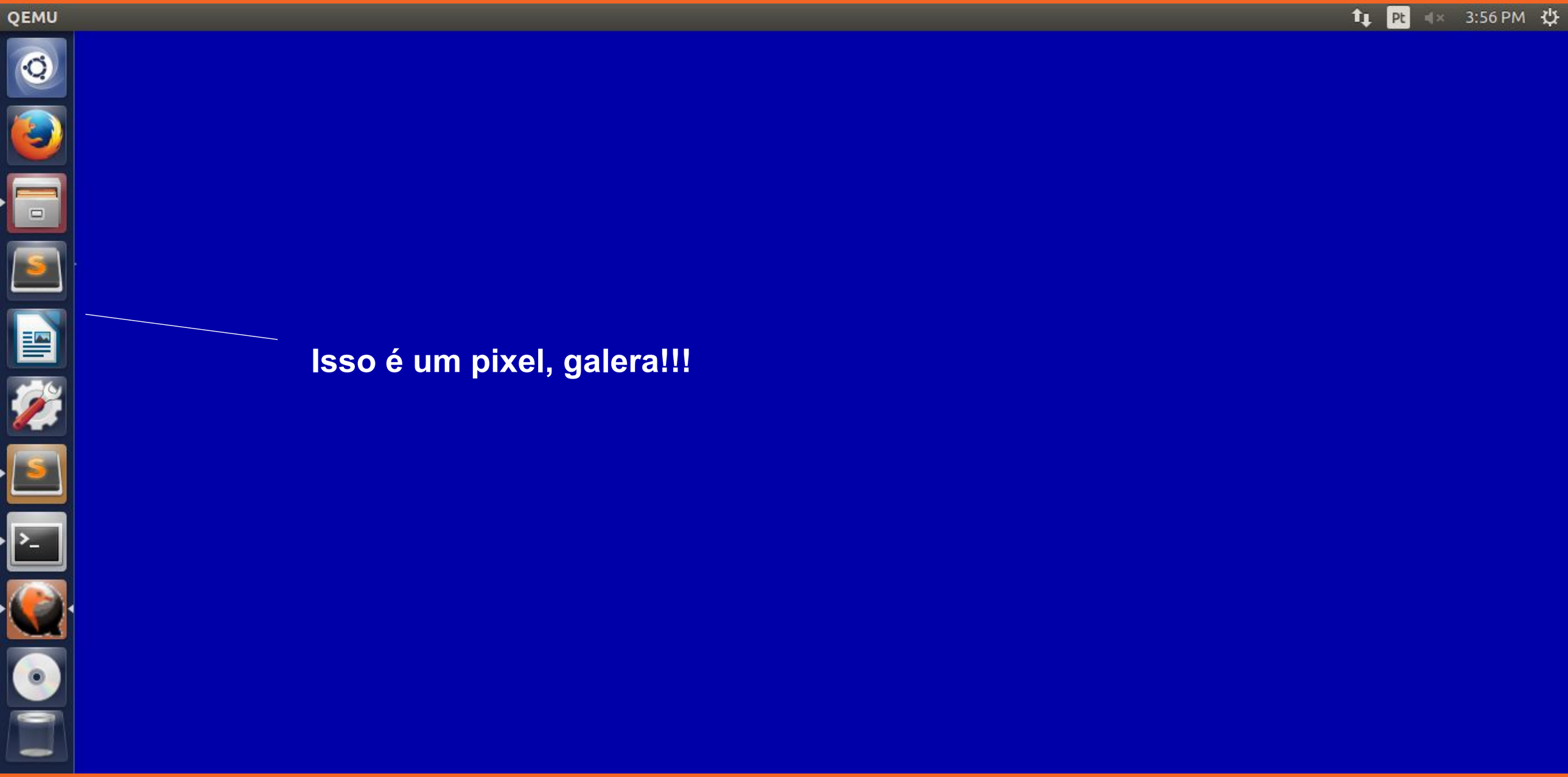
.* Aa “” strcpy Find Find Prev Find All

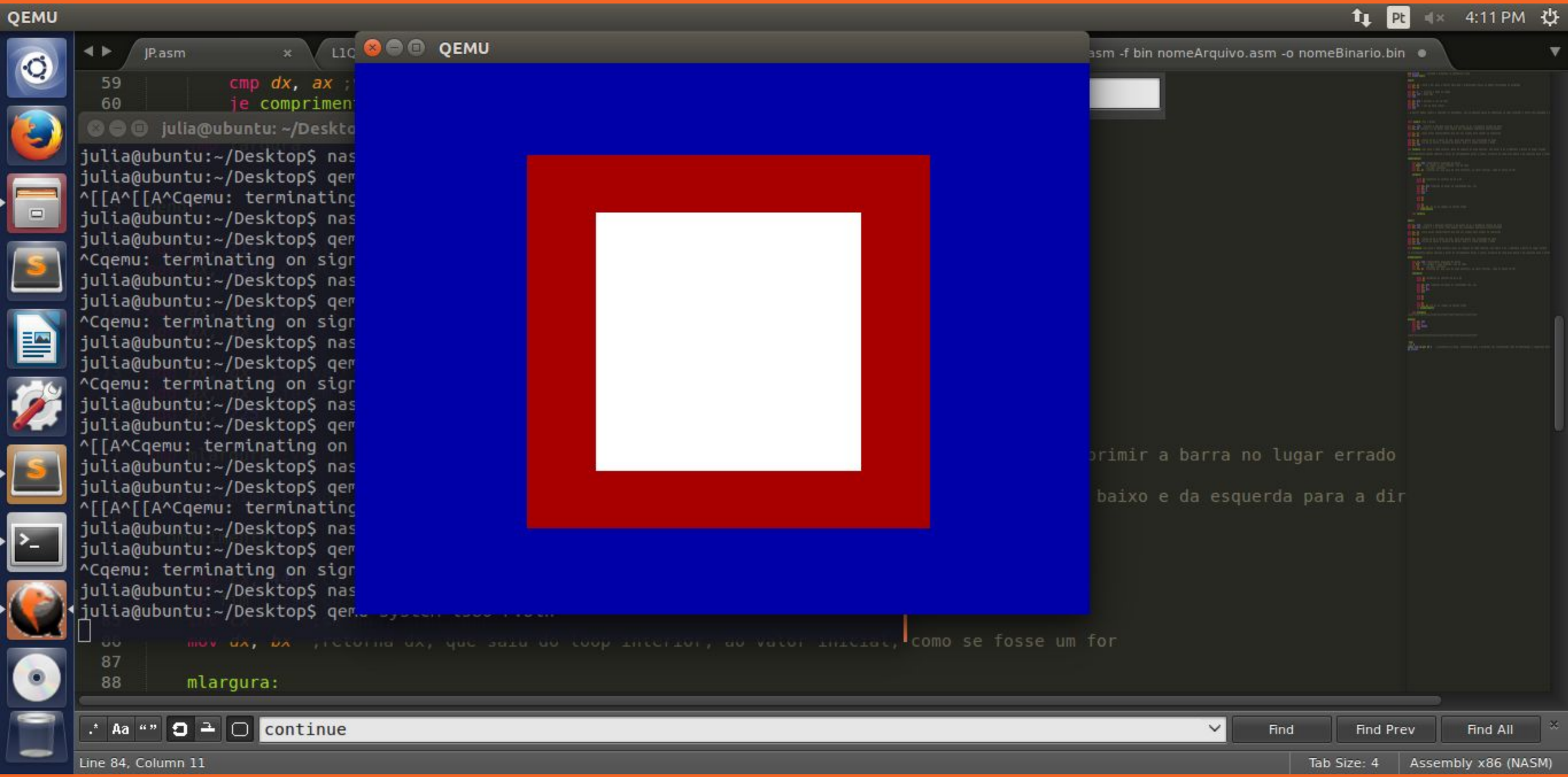
Line 22, Column 22 Tab Size: 4 Assembly x86 (NASM)

Vídeo - int 10h

Imprimir um pixel na tela:

```
mov ah, 0ch ;pixel na coordenada [dx, cx]
mov bh, 0
mov al, 0ah ;cor do pixel (verde claro)
int 10h
```


QEMU
Pt
3:56 PM
Isso é um pixel, galera!!!


QEMU
4:11 PM
JP.asm
59 cmp dx, ax ;
60 je comprimen
julia@ubuntu: ~/Deskto
julia@ubuntu:~/Desktop$ nas
julia@ubuntu:~/Desktop$ qem
^[[A^[[A^Cqemu: terminating
julia@ubuntu:~/Desktop$ nas
julia@ubuntu:~/Desktop$ qem
^Cqemu: terminating on sign
julia@ubuntu:~/Desktop$ nas
julia@ubuntu:~/Desktop$ qem
^Cqemu: terminating on sign
julia@ubuntu:~/Desktop$ nas
julia@ubuntu:~/Desktop$ qem
^Cqemu: terminating on sign
julia@ubuntu:~/Desktop$ nas
julia@ubuntu:~/Desktop$ qem
^[[A^Cqemu: terminating on
julia@ubuntu:~/Desktop$ nas
julia@ubuntu:~/Desktop$ qem
julia@ubuntu:~/Desktop$ nas
julia@ubuntu:~/Desktop$ qem
julia@ubuntu:~/Desktop$ nas
julia@ubuntu:~/Desktop$ qem
QEMU
asm -f bin nomeArquivo.asm -o nomeBinario.bin
primir a barra no lugar errado
baixo e da esquerda para a dir
como se fosse um for
87
88 mlargura:
continue
Find
Find Prev
Find All
Line 84, Column 11
Tab Size: 4
Assembly x86 (NASM)

Teclado - int 16h

Ler um caractere do teclado:
mov AH, 0 ;Número da chamada.
int 16h
Após a execução dessa interrupção o caractere lido estará armazenado em AL.

Quando você lê um caractere do teclado, ele não aparece automaticamente na tela! Você deve fazer isso manualmente :)

Sobreposição de Dados

...Após a execução dessa interrupção o caractere lido estará armazenado em AL.

Ou seja, o dado anterior em AL é perdido. É bom lembrar que AL é a parte baixa de AX, os 8 bits menos significativos. AH tem os 8 bits restantes.

Quando você move algo para AX, os dados em AL e AH são perdidos.

Links para tabela de interrupção

http://webpages.charter.net/danrollins/techhelp/0027.HTM

http://www.ctyme.com/intr/int.htm

Instruções aritméticas

,

Utilizam o primeiro campo como destino e operando. Os outros são apenas operandos.

Exemplos de instruções por número de operandos:
1 operando: INC, DEC, DIV, IDIV, MUL, IMUL
2 operandos: ADD, SUB, IMUL, AND, XOR
3 operandos: IMUL

mul op
ax = al*op if ah = 0

div op
al = ax/op ah =
resto

Exemplo:

```
mov ax, 2
mov cx, 4
mov bl, 2
add ax, 5
sub cx, 3
inc ax
dec cx
div bl
xor ax, ax
```

Diretivas

times

- **Faz com que uma instrução seja executada n vezes.**

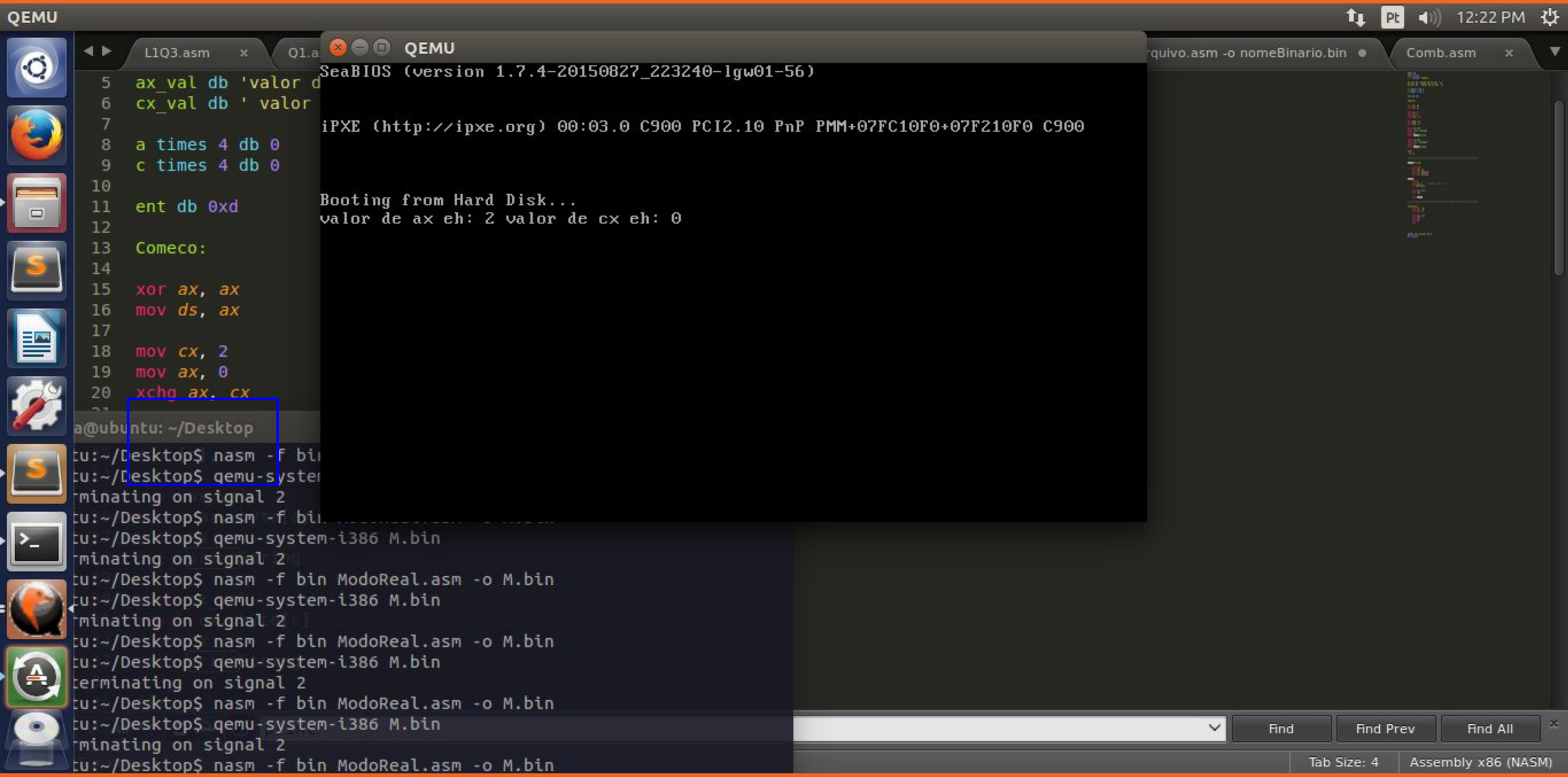
```
nome times 16 db 0 ;declara 16 bytes com valor 0
```

Instruções Importantes

xchg

xchg origem, destino
Troca o conteúdo de origem com destino. Os parâmetros devem ser do mesmo tamanho. Pelo menos um deles deve ser um registrador.

xchg ax, cx
xchg ax, [memoria]


QEMU
12:22 PM
L1Q3.asm
Comb.asm
QEMU
SeaBIOS (version 1.7.4-20150827_223240-lgw01-56)
iPXE (http://ipxe.org) 00:03.0 C900 PCI2.10 PnP PMM+07FC10F0+07F210F0 C900
Booting from Hard Disk...
valor de ax eh: 2 valor de cx eh: 0
5 ax_val db 'valor d
6 cx_val db ' valor
7
8 a times 4 db 0
9 c times 4 db 0
10
11 ent db 0xd
12
13 Comeco:
14
15 xor ax, ax
16 mov ds, ax
17
18 mov cx, 2
19 mov ax, 0
20 xchg ax, cx
a@ubuntu: ~/Desktop
tu:~/Desktop$ nasm -f bi
tu:~/Desktop$ qemu-syste
rminating on signal 2
tu:~/Desktop$ nasm -f bi
tu:~/Desktop$ qemu-system-i386 M.bin
rminating on signal 2
tu:~/Desktop$ nasm -f bin ModoReal.asm -o M.bin
tu:~/Desktop$ qemu-system-i386 M.bin
rminating on signal 2
tu:~/Desktop$ nasm -f bin ModoReal.asm -o M.bin
tu:~/Desktop$ qemu-system-i386 M.bin
terminating on signal 2
tu:~/Desktop$ nasm -f bin ModoReal.asm -o M.bin
tu:~/Desktop$ qemu-system-i386 M.bin
rminating on signal 2
tu:~/Desktop$ nasm -f bin ModoReal.asm -o M.bin
quivo.asm -o nomeBinario.bin
Find
Find Prev
Find All
Tab Size: 4
Assembly x86 (NASM)

loop

loop label

Desvia para label enquando cx != 0

```
mov cx, 10
contagem:
   inc ax
loop contagem
```

lodsb

Carrega em AL byte apontado por DS:SI e após o carregamento SI é automaticamente incrementado ou decrementado.

Existem também as formas lodsw, lodsd, que são load word e load double word, respectivamente.

stosb

Oposto da instrução lodsb, armazena um byte de AL em uma posição de memória apontada por ES:DI. Após o carregamento DI, é automaticamente incrementado ou decrementado.

Existem também as formas stosw, stosd, que são store word e store double word, respectivamente.

Montando e executando programas em modo real

```
nasm -f bin nomeArquivo.asm -o nomeBinario.bin
qemu-system-i386 nomeBinario.bin
```

(em outras versões do qemu)

```
nasm -f bin nomeArquivo.asm -o nomeBinario.bin
qemu-i386 nomeBinario.bin
```

Estrutura básica

untitled • - Sublime Text (UNREGISTERED)

untitled

```
1  org 0x7c00          ;endereço de memória em que o programa será carregado
2  jmp 0x0000:start    ;far jump - seta cs para 0
3
4  start:
5      xor ax, ax      ;zera ax, xor é mais rápido que mov
6      mov ds, ax      ;zera ds (não pode ser zerado diretamente)
7      mov es, ax      ;zera es
8
9      ;código
10
11     jmp done
12
13 done:
14     jmp $           ;$ = linha atual
15
16 times 510 - ($ - $$) db 0
17 dw 0xaa55           ;assinatura de boot
```

Line 10, Column 1 | Tab Size: 4 | Assembly x86 (NASM)

HelloWorld.asm

~/Downloads/exemplo.asm - Sublime Text (UNREGISTERED)

exemplo.asm

```
1  org 0x7c00        ;endereço de memória em que o programa será carregado
2  jmp 0x0000:start  ;far jump - seta cs para 0
3
4  hello db 'Hello, World!', 13, 10, 0 ;reserva espaço na memória para a string
5
6  start:
7      xor ax, ax  ;zera ax, xor é mais rápido que mov
8      mov ds, ax  ;zera ds (não pode ser zerado diretamente)
9      mov es, ax  ;zera es
10
11     mov si, hello ;faz si apontar para início de 'hello'
12     call print_string
13
14     jmp done
15
16 print_string:
17     lodsb       ;carrega uma letra de si em al e passa para o próximo caractere
18     cmp al, 0   ;chegou no final? (equivalente a um \0)
19     je .done
20
21     mov ah, 0eh ;código da instrução para imprimir um caractere que está em al
22     int 10h     ;interrupção de vídeo.
23
24     .done:
25         ret     ;retorna para o start
26
27 done:
28     jmp $       ;$ = linha atual
29
30 times 510 - ($ - $$) db 0
31 dw 0xaa55        ;assinatura de boot
```

Line 31, Column 36 | Tab Size: 4 | Assembly x86 (NASM)

Link:

http://bit.ly/2weMr8s

Dúvidas?

Links importantes

https://en.wikipedia.org/wiki/INT_10H

http://www.jegerlehner.ch/intel/IntelCodeTable.pdf

http://wiki.osdev.org/Real_mode_assembly_I