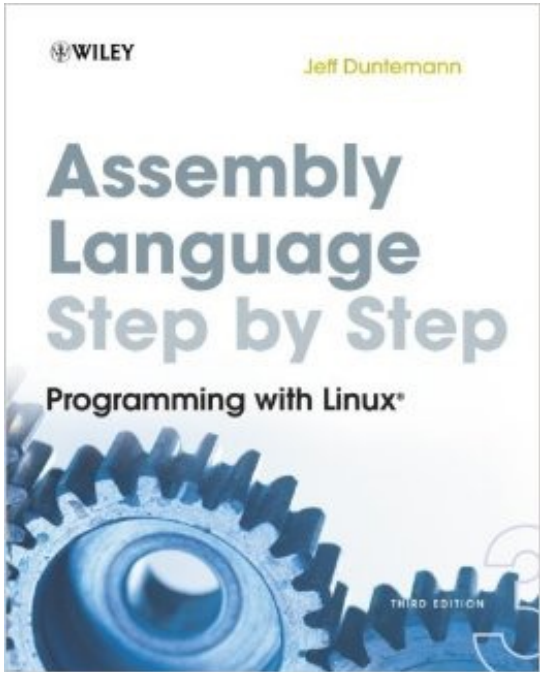
# Software Security

*Module 2 - x86 Assembly Wrap-Up*

CMPE279
Software Security Technologies
San Jose State University




Examples Based on Jeff Duntemann's Book

# Assembly Program

# Basic Structure

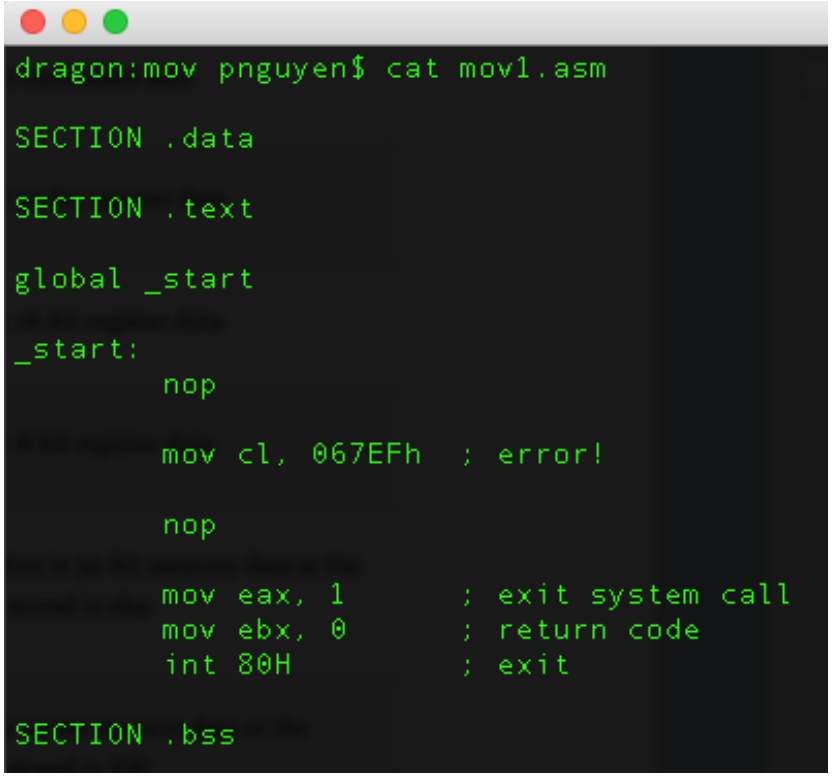
```
section .data
section .text

        global _start

_start:
        nop
; Put your experiments between the two nops...


; Put your experiments between the two nops...
        nop

section .bss
```

| | |
|---|---|
| .data | initialized variables |
| .bss | un-initialized variables |
| .text | instructions (aka code segment) |
| _start | entry point (label) required by Linux |

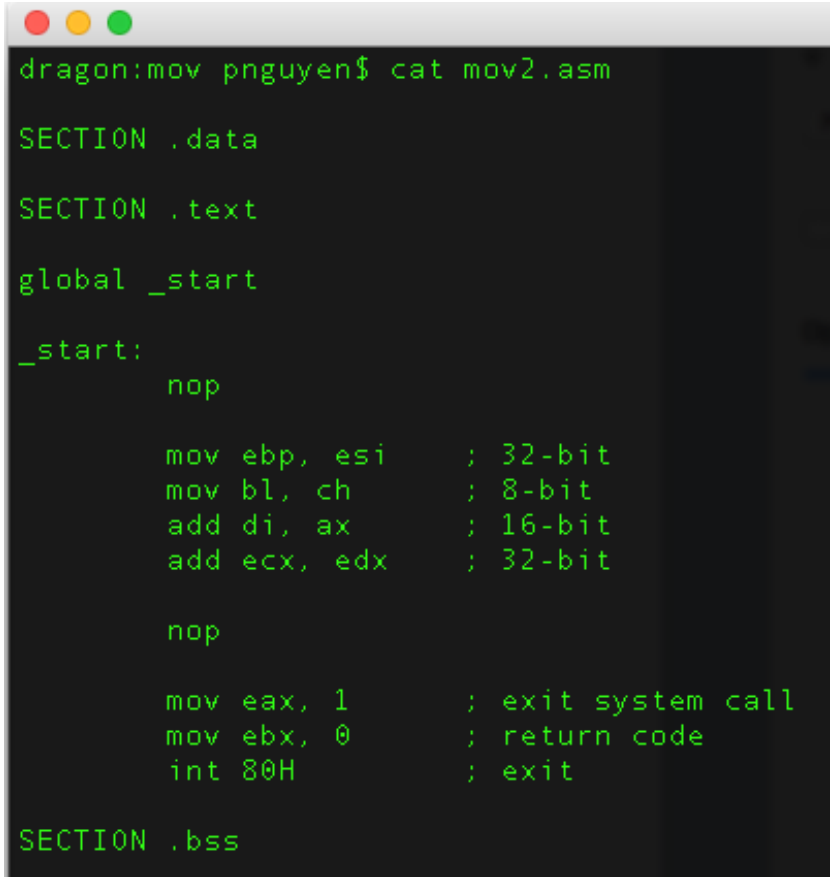
MOV

# Registers, Immediate & Memory

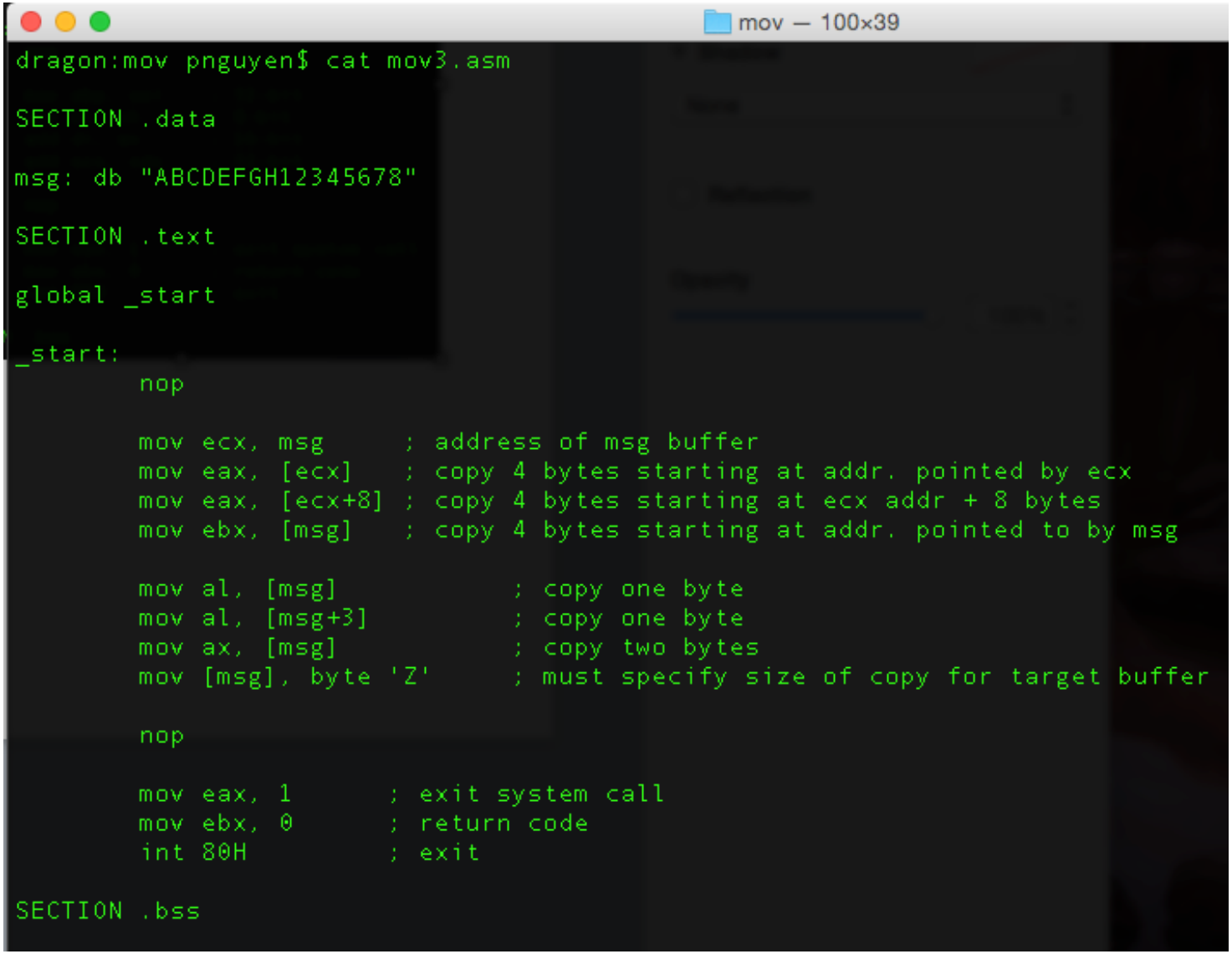
| Machine Instruction | Destination Operand | Source Operand | Operand Notes |
|---|---|---|---|
| MOV | EAX, | 42h | Source is immediate data |
| MOV | EBX, | EDI | Both are 32-bit register data |
| MOV | BX, | CX | Both are 16-bit register data |
| MOV | DL, | BH | Both are 8-bit register data |
| MOV | [EBP], | EDI | Destination is 32-bit memory data at the address stored in ebp |
| MOV | EDX, | [ESI] | Source is 32-bit memory data at the address stored in ESI |

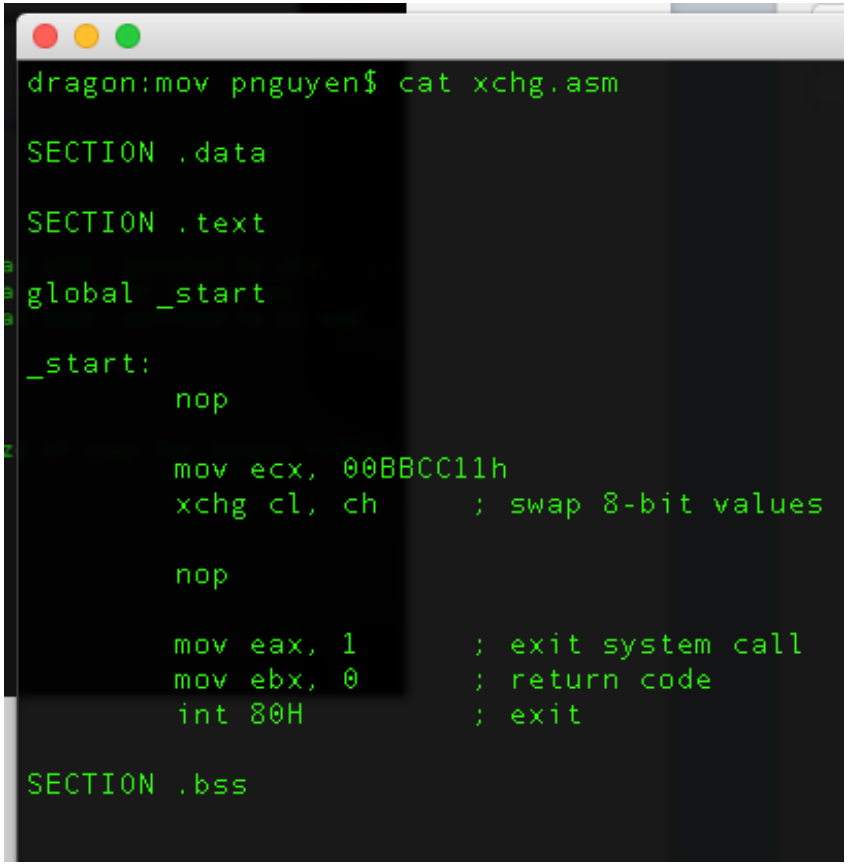


```
dragon:mov pnguyen$ cat mov1.asm

SECTION .data

SECTION .text

global _start

_start:
        nop

        mov cl, 067EFh  ; error!

        nop

        mov eax, 1      ; exit system call
        mov ebx, 0      ; return code
        int 80H         ; exit

SECTION .bss
```



```
dragon:mov pnguyen$ cat mov2.asm

SECTION .data

SECTION .text

global _start

_start:
        nop

        mov ebp, esi    ; 32-bit
        mov bl, ch      ; 8-bit
        add di, ax      ; 16-bit
        add ecx, edx    ; 32-bit

        nop

        mov eax, 1      ; exit system call
        mov ebx, 0      ; return code
        int 80H         ; exit

SECTION .bss
```

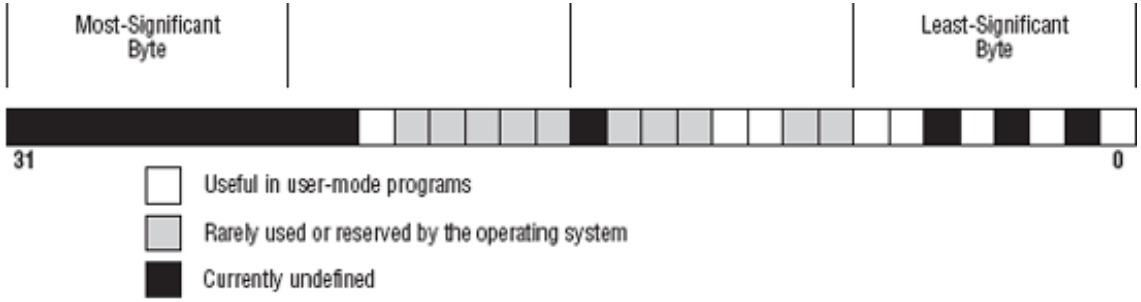

mov — 100×39
dragon:mov pnguyen$ cat mov3.asm
SECTION .data
msg: db "ABCDEFGH12345678"
SECTION .text
global _start
_start:
nop
mov ecx, msg ; address of msg buffer
mov eax, [ecx] ; copy 4 bytes starting at addr. pointed by ecx
mov eax, [ecx+8] ; copy 4 bytes starting at ecx addr + 8 bytes
mov ebx, [msg] ; copy 4 bytes starting at addr. pointed to by msg
mov al, [msg] ; copy one byte
mov al, [msg+3] ; copy one byte
mov ax, [msg] ; copy two bytes
mov [msg], byte 'Z' ; must specify size of copy for target buffer
nop
mov eax, 1 ; exit system call
mov ebx, 0 ; return code
int 80H ; exit
SECTION .bss


dragon:mov pnguyen$ cat xchg.asm
SECTION .data
SECTION .text
global _start
_start:
nop
mov ecx, 00BBCC11h
xchg cl, ch ; swap 8-bit values
nop
mov eax, 1 ; exit system call
mov ebx, 0 ; return code
int 80H ; exit
SECTION .bss

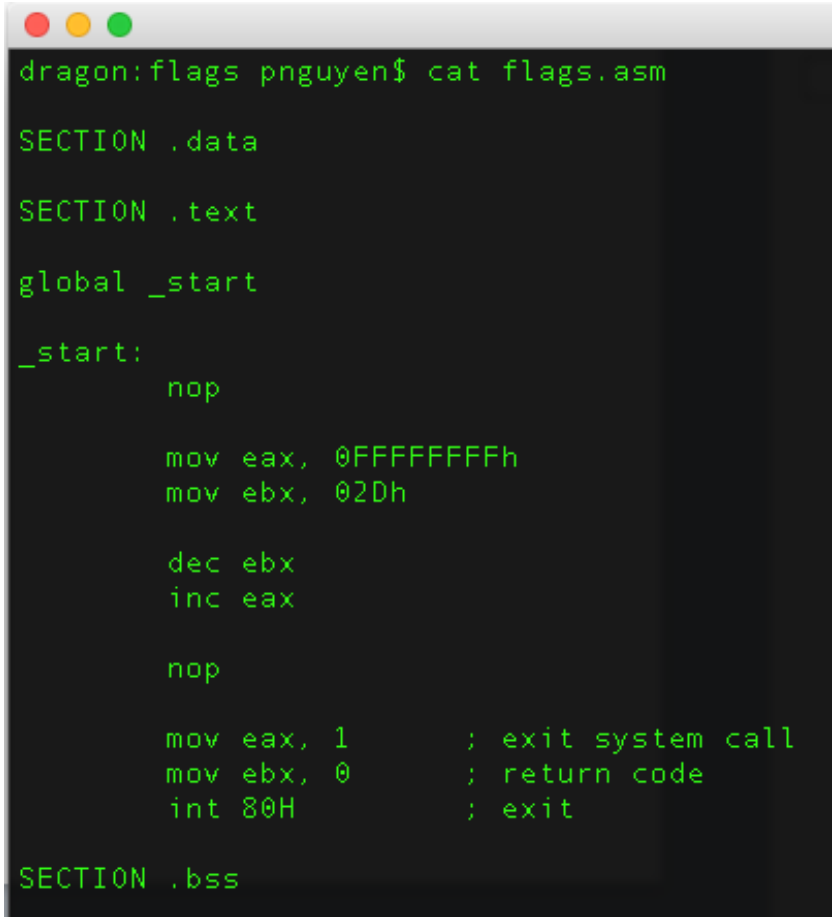
# JUMPS

| Mnemonics | | Synonyms | |
|---|---|---|---|
| JA | Jump if Above | JNBE | Jump if Not Below or Equal |
| JAE | Jump if Above or Equal | JNB | Jump if Not Below |
| JB | Jump if Below | JNAE | Jump if Not Above or Equal |
| JBE | Jump if Below or Equal | JNA | Jump if Not Above |
| JE | Jump if Equal | JZ | Jump if result is Zero |
| JNE | Jump if Not Equal | JNZ | Jump if result is Not Zero |
| JG | Jump if Greater | JNLE | Jump if Not Less than or Equal |
| JGE | Jump if Greater or Equal | JNL | Jump if Not Less |
| JL | Jump if Less | JNGE | Jump if Not Greater or Equal |
| JLE | Jump if Less or Equal | JNG | Jump if Not Greater |

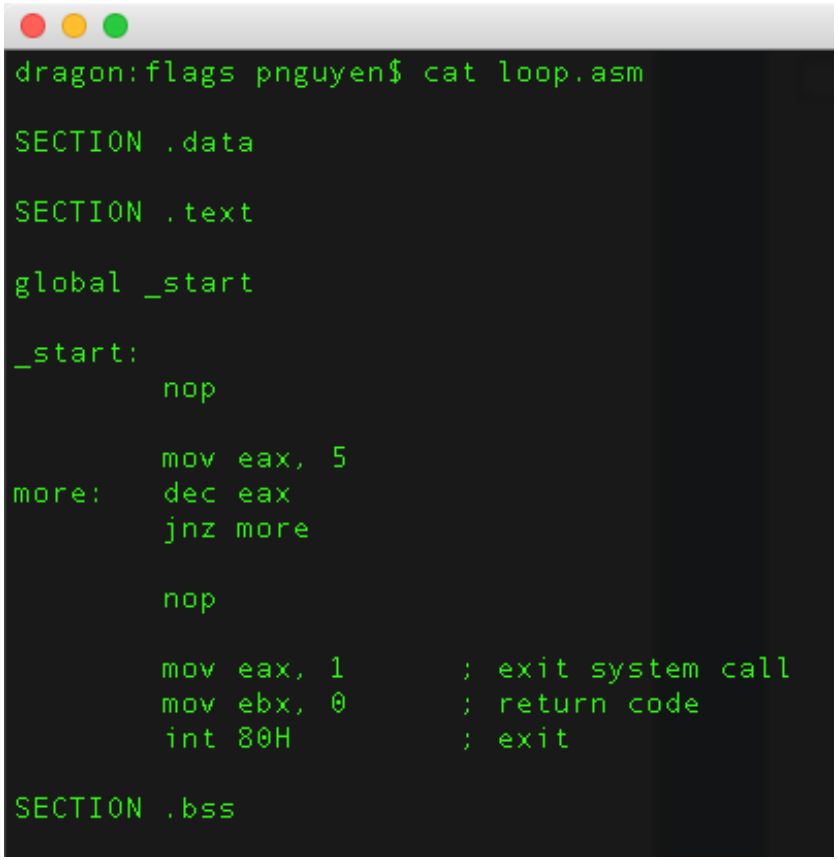
# EFLAGS



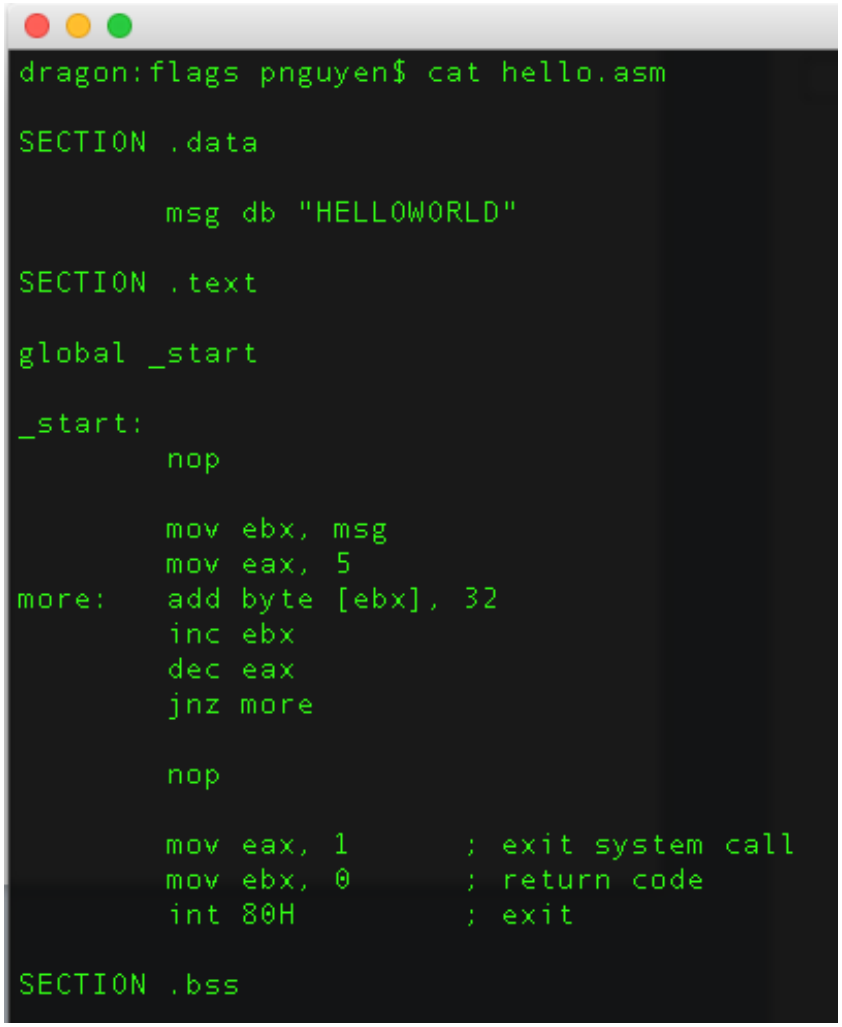

| | | | |
|---|---|---|---|
| 0 | CF | Carry Flag | 0 = No carry in operation; 1 = carry |
| 1 | - | (Undefined) | |
| 2 | PF | Parity Flag | 0 = # of 1-bits in byte is odd; 1 = # of 1-bits in byte is even |
| 3 | - | (Undefined) | |
| 4 | AF | Auxiliary Carry Flag | 0 = No carry in BCD operation; 1 = BCD carry |
| 5 | - | (Undefined) | |
| 6 | ZF | Zero Flag | 0 = Operand became nonzero; 1 = operand became 0 |
| 7 | SF | Sign Flag | 0 = Operand did not become negative; 1= operand became negative |
| 8 | TF | Trap Flag | Facilitates single stepping |
| 9 | IF | Interrupt Enable Flag | Reserved by operating system in protected mode |
| 10 | DF | Direction Flag | 0 = Autoincrement is up-memory; 1 = Autoincrement is down-memory |
| 11 | OF | Overflow Flag | 0 = No overflow in signed operation; 1 = overflow in signed operation |
| 12 | IOPL | I/O Privilege Level 0 | Reserved by operating system in protected mode |
| 13 | IOPL | I/O Privilege Level 1 | Reserved by operating system in protected mode |
| 14 | NT | Nested Task Flag | Reserved by operating system in protected mode |
| 15 | - | (Undefined) | |
| 16 | RF | Resume Flag | Facilitates single-stepping |
| 17 | VM | Virtual-86 Mode Flag | Reserved by operating system in protected mode |
| 18 | AC | Alignment Check Flag | Reserved by operating system in protected mode |
| 19 | VIF | Virtual Interrupt Flag | Reserved by operating system in protected mode |
| 20 | VIP | Virtual Interrupt Pending | Reserved by operating system in protected mode |
| 21 | ID | CPU ID | If this bit can be changed by user space programs, CPUID is available |
| 22 | - | (Undefined) | |
| . . . | | | |
| 31 | - | (Undefined) | |



```
dragon:flags pnguyen$ cat flags.asm

SECTION .data

SECTION .text

global _start

_start:
        nop

        mov eax, 0FFFFFFFFh
        mov ebx, 02Dh

        dec ebx
        inc eax

        nop

        mov eax, 1      ; exit system call
        mov ebx, 0      ; return code
        int 80H         ; exit

SECTION .bss
```



```
dragon:flags pnguyen$ cat loop.asm

SECTION .data

SECTION .text

global _start

_start:
        nop

        mov eax, 5
more:   dec eax
        jnz more

        nop

        mov eax, 1      ; exit system call
        mov ebx, 0      ; return code
        int 80H         ; exit

SECTION .bss
```


dragon:flags pnguyen$ cat hello.asm
SECTION .data
msg db "HELLOWORLD"
SECTION .text
global _start
_start:
nop
mov ebx, msg
mov eax, 5
more: add byte [ebx], 32
inc ebx
dec eax
jnz more
nop
mov eax, 1 ; exit system call
mov ebx, 0 ; return code
int 80H ; exit
SECTION .bss

# MATH

# http://en.wikipedia.org/wiki/Two's_complement

**8-bit two's-complement integers**

| Bits | Unsigned value | 2's complement value |
|---|---|---|
| 0111 1111 | 127 | 127 |
| 0111 1110 | 126 | 126 |
| 0000 0010 | 2 | 2 |
| 0000 0001 | 1 | 1 |
| 0000 0000 | 0 | 0 |
| 1111 1111 | 255 | –1 |
| 1111 1110 | 254 | –2 |
| 1000 0010 | 130 | –126 |
| 1000 0001 | 129 | –127 |
| 1000 0000 | 128 | –128 |

| Decimal | 7-bit notation | 8-bit notation |
|---|---|---|
| –42 | 1010110 | 1101 0110 |
| 42 | 0101010 | 0010 1010 |

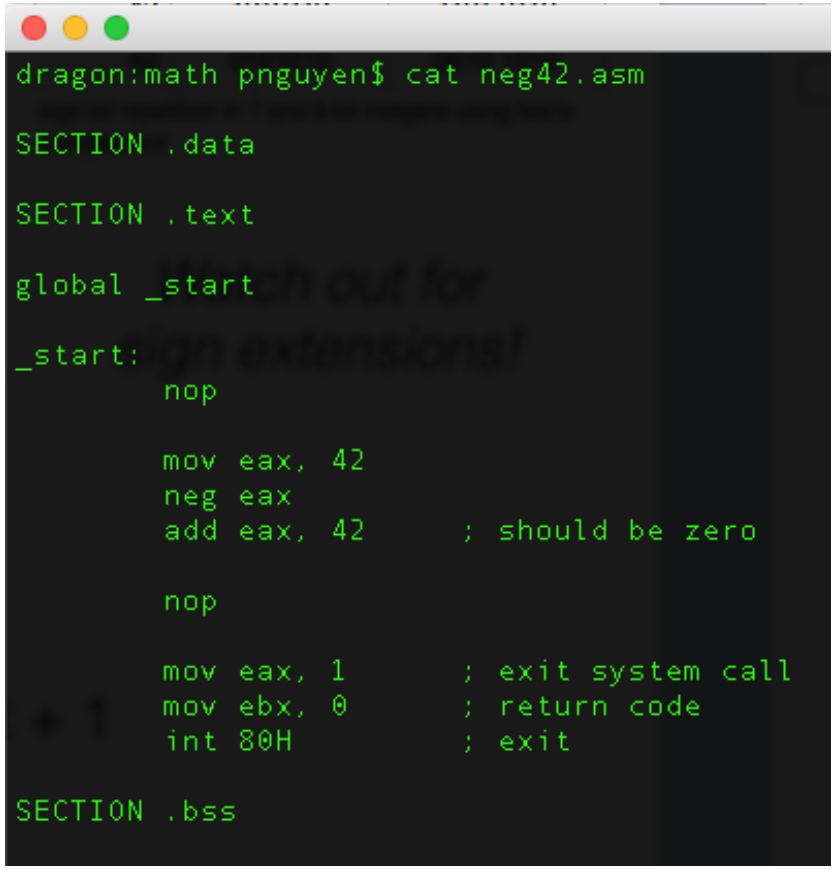
sign-bit repetition in 7 and 8-bit integers using two's-complement

*Watch out for sign extensions!*

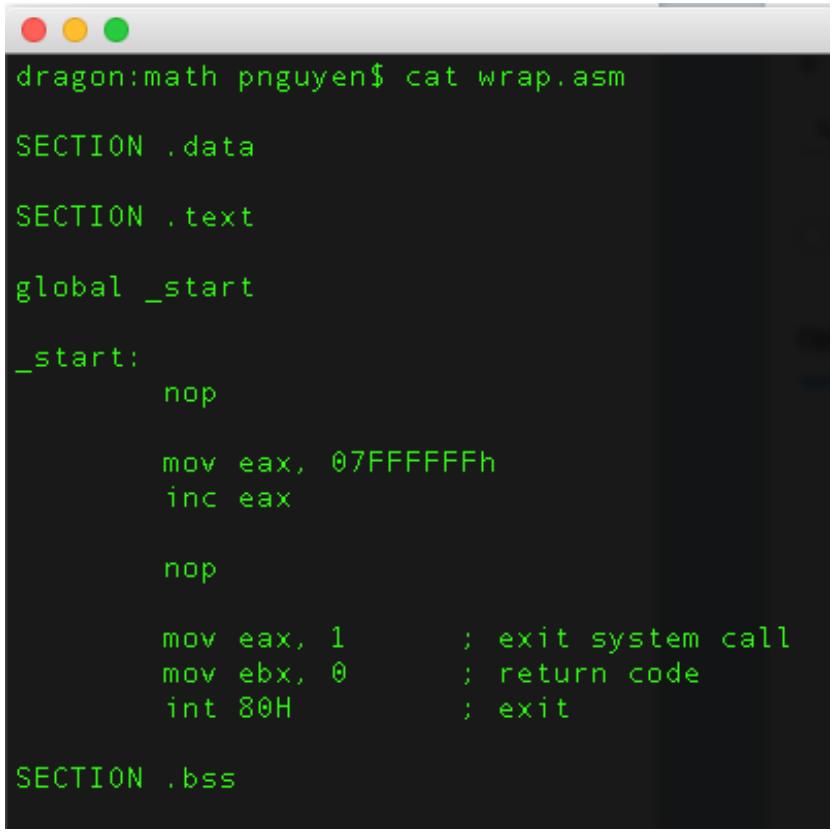
2's Complement = 1's Complement + 1

# Signed Integer Ranges

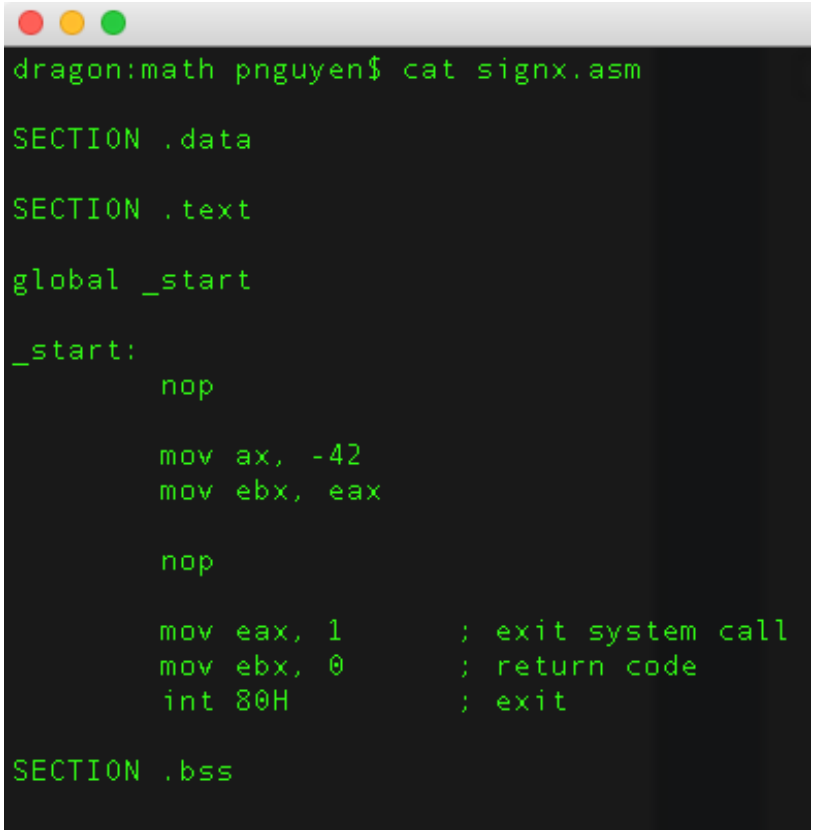
| | Greatest Negative Value | | Greatest Positive Value | |
|---|---|---|---|---|
| **Value Size** | **Decimal** | **Hex** | **Decimal** | **Hex** |
| Eight Bits | –128 | 80h | 127 | 7Fh |
| Sixteen Bits | –32768 | 8000h | 32767 | 7FFFh |
| Thirty-Two Bits | –2147483648 | 80000000h | 2147483647 | 7FFFFFFFh |



```
dragon:math pnguyen$ cat neg42.asm

SECTION .data

SECTION .text

global _start

_start:
        nop

        mov eax, 42
        neg eax
        add eax, 42     ; should be zero

        nop

        mov eax, 1      ; exit system call
        mov ebx, 0      ; return code
        int 80H         ; exit

SECTION .bss
```



```
dragon:math pnguyen$ cat wrap.asm

SECTION .data

SECTION .text

global _start

_start:
        nop

        mov eax, 07FFFFFFh
        inc eax

        nop

        mov eax, 1      ; exit system call
        mov ebx, 0      ; return code
        int 80H         ; exit

SECTION .bss
```



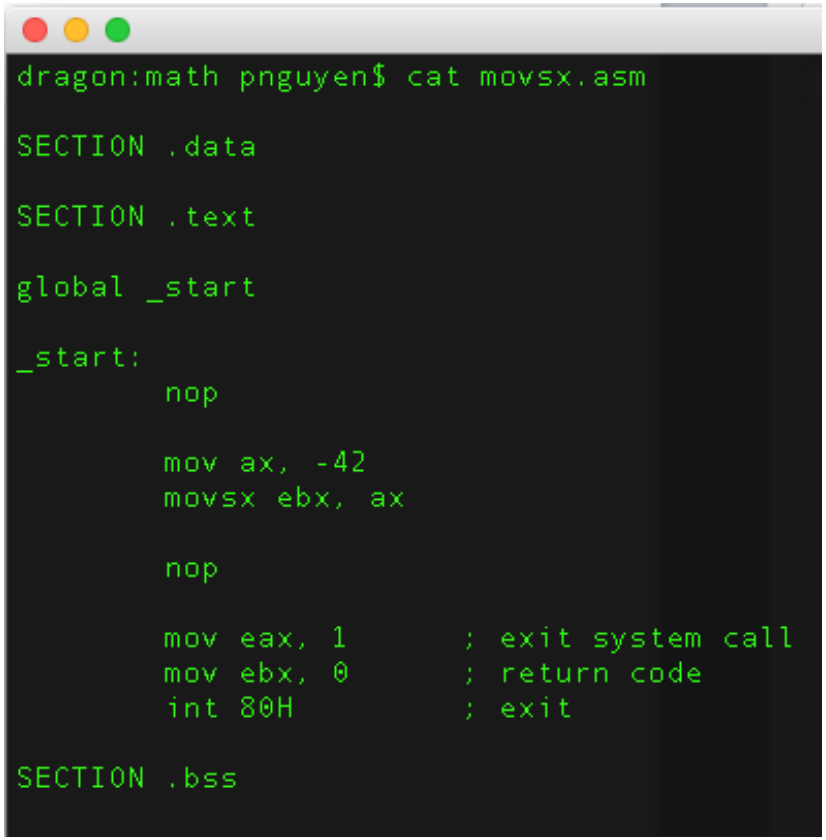
```
dragon:math pnguyen$ cat signx.asm

SECTION .data

SECTION .text

global _start

_start:
        nop

        mov ax, -42
        mov ebx, eax

        nop

        mov eax, 1      ; exit system call
        mov ebx, 0      ; return code
        int 80H         ; exit

SECTION .bss
```

# MOVSX / SRC & DEST DIFFERENT SIZES

| Machine Instruction | Destination Operand | Source Operand | Operand Notes |
|---|---|---|---|
| MOVSX | r16 | r/m8 | 8-bit signed to 16-bit signed |
| MOVSX | r32 | r/m8 | 8-bit signed to 32-bit signed |
| MOVSX | r32 | r/m16 | 16-bit signed to 32-bit signed |

## Notes:

***r16*** *= any 16-bit register*
***r/m*** *= register or memory*
***r/m16*** *= any 16-bit register or memory location*



```
dragon:math pnguyen$ cat movsx.asm

SECTION .data

SECTION .text

global _start

_start:
        nop

        mov ax, -42
        movsx ebx, ax

        nop

        mov eax, 1      ; exit system call
        mov ebx, 0      ; return code
        int 80H         ; exit

SECTION .bss
```

# MUL / IMPLICIT OPERANDS

| | |
|---|---|
| MUL | Unsigned Multiplication |
| DIV | Unsigned Division |
| IMUL | Signed Multiplication |
| IDIV | Signed Division |

***Problem**: If we multiply two 16-bit values…*
*The Product will be > 16-bits!*

# MUL

| Machine Instruction | Explicit Operand (Factor 1) | Implicit Operand (Factor 2) | Implicit Operand (Product) |
|---|---|---|---|
| mul | r/m8 | AL | AX |
| mul | r/m16 | AX | DX and AX |
| mul | r/m32 | EAX | EDX and EAX |

**Notes:**

- *1st Factor = Value, Register or Memory*
- *2nd Factor always "A" General Purpose Register*
- *CF (Carry Flag) Set of Product Overflow into Extra "D" Register*

# DIV

| Machine Instruction | Explicit Operand (Divisor) | Implicit Operand (Quotient) | Implicit Operand (Remainder) |
|---|---|---|---|
| DIV | r/m8 | AL | AH |
| DIV | r/m16 | AX | DX |
| DIV | r/m32 | EAX | EDX |

**Notes:**

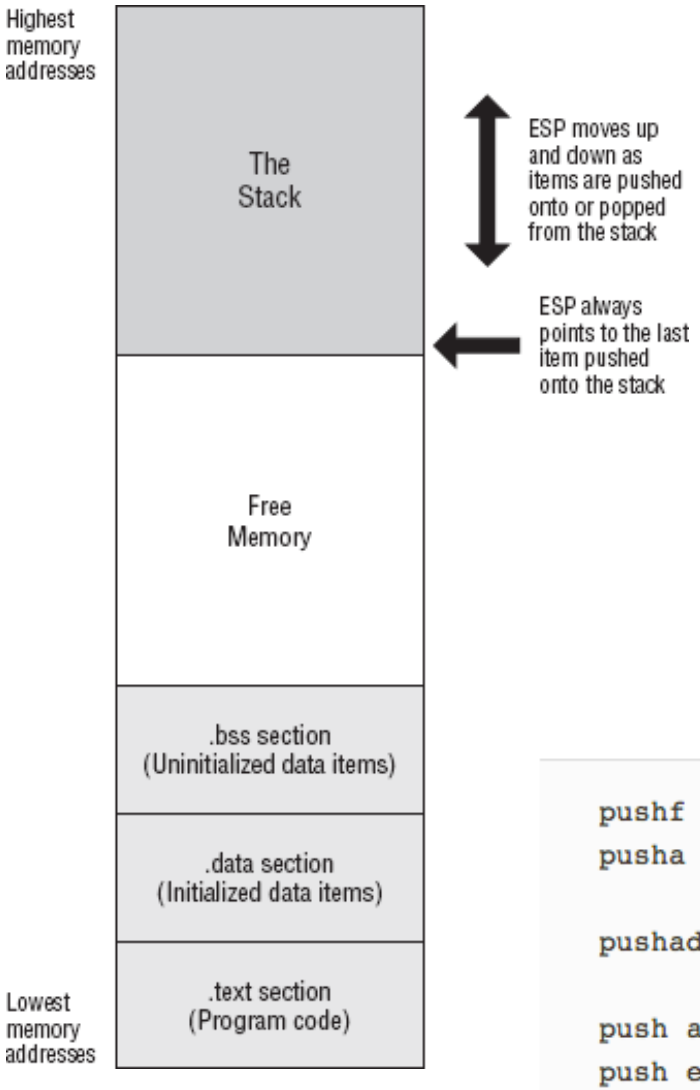
- *DIV doesn't affect any flags*
- *Divisor = Zero is undefined*

```
dragon:math pnguyen$ cat mul.asm

SECTION .data

SECTION .text

global _start

_start:
        nop

        mov eax, 447
        mov ebx, 1739
        mul ebx

        mov eax, 0FFFFFFFFh
        mov ebx, 03B72h
        mul ebx

        nop

        mov eax, 1      ; exit system call
        mov ebx, 0      ; return code
        int 80H         ; exit

SECTION .bss
```

```
dragon:math pnguyen$ cat div.asm

SECTION .data

SECTION .text

global _start

_start:
        nop

        mov ecx, 00000002h
        mov edx, 00000000h
        mov eax, 00000008h
        div ecx
        inc ecx
        div ecx

        nop

        mov eax, 1      ; exit system call
        mov ebx, 0      ; return code
        int 80H         ; exit

SECTION .bss
```

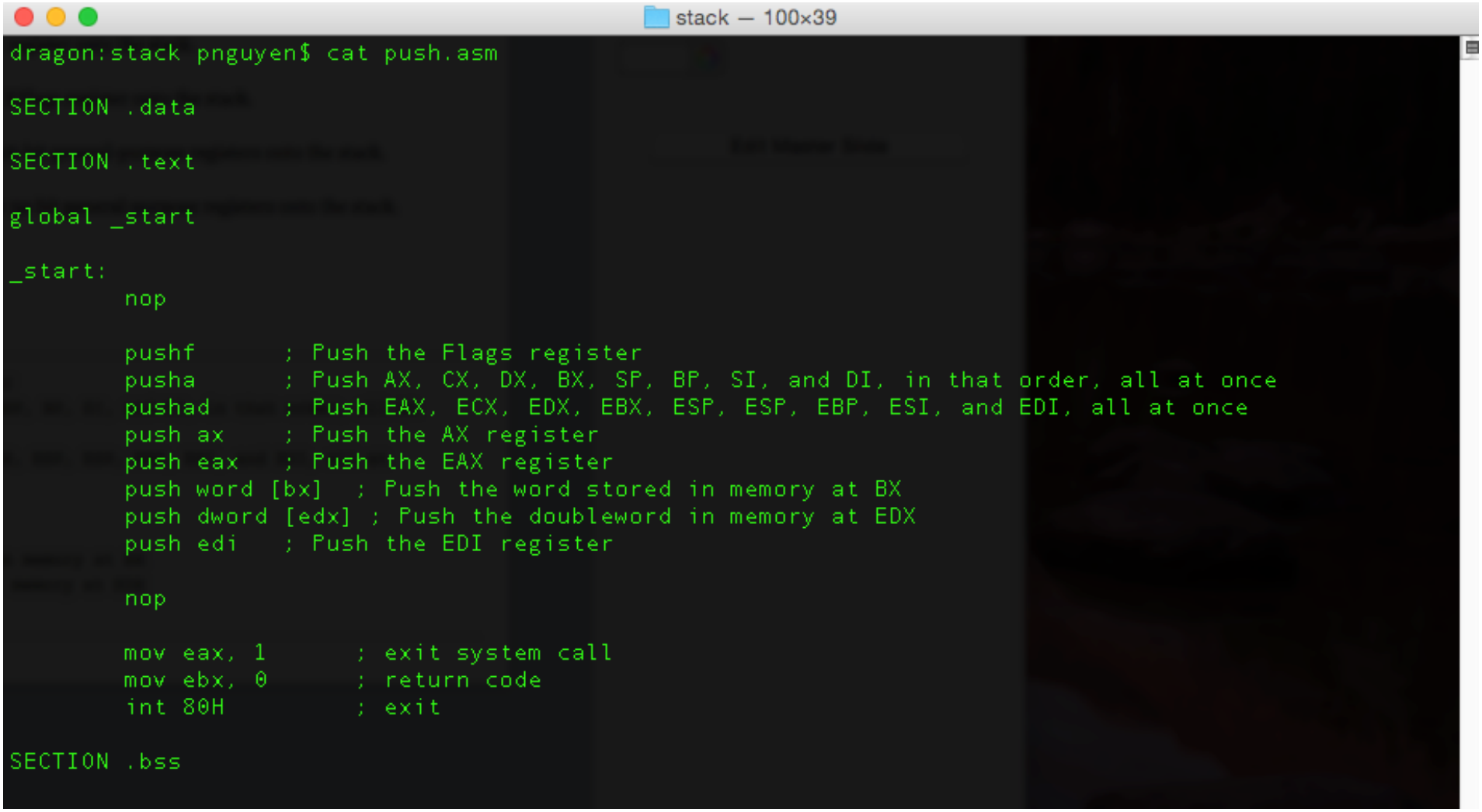
# STACK

# PUSH

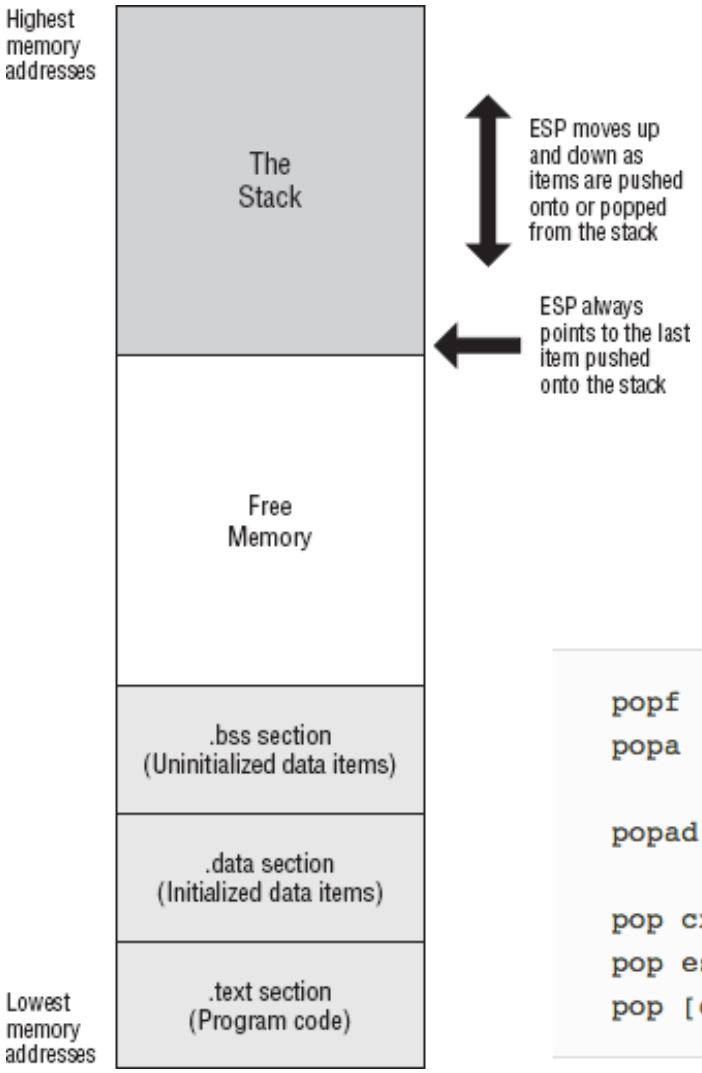



- `PUSH` pushes a 16-bit or 32-bit register or memory value that is specified by you in your source code.
- `PUSHF` pushes the 16-bit Flags register onto the stack.
- `PUSHFD` pushes the full 32-bit EFlags register onto the stack.
- `PUSHA` pushes all eight of the 16-bit general-purpose registers onto the stack.
- `PUSHAD` pushes all eight of the 32-bit general-purpose registers onto the stack.

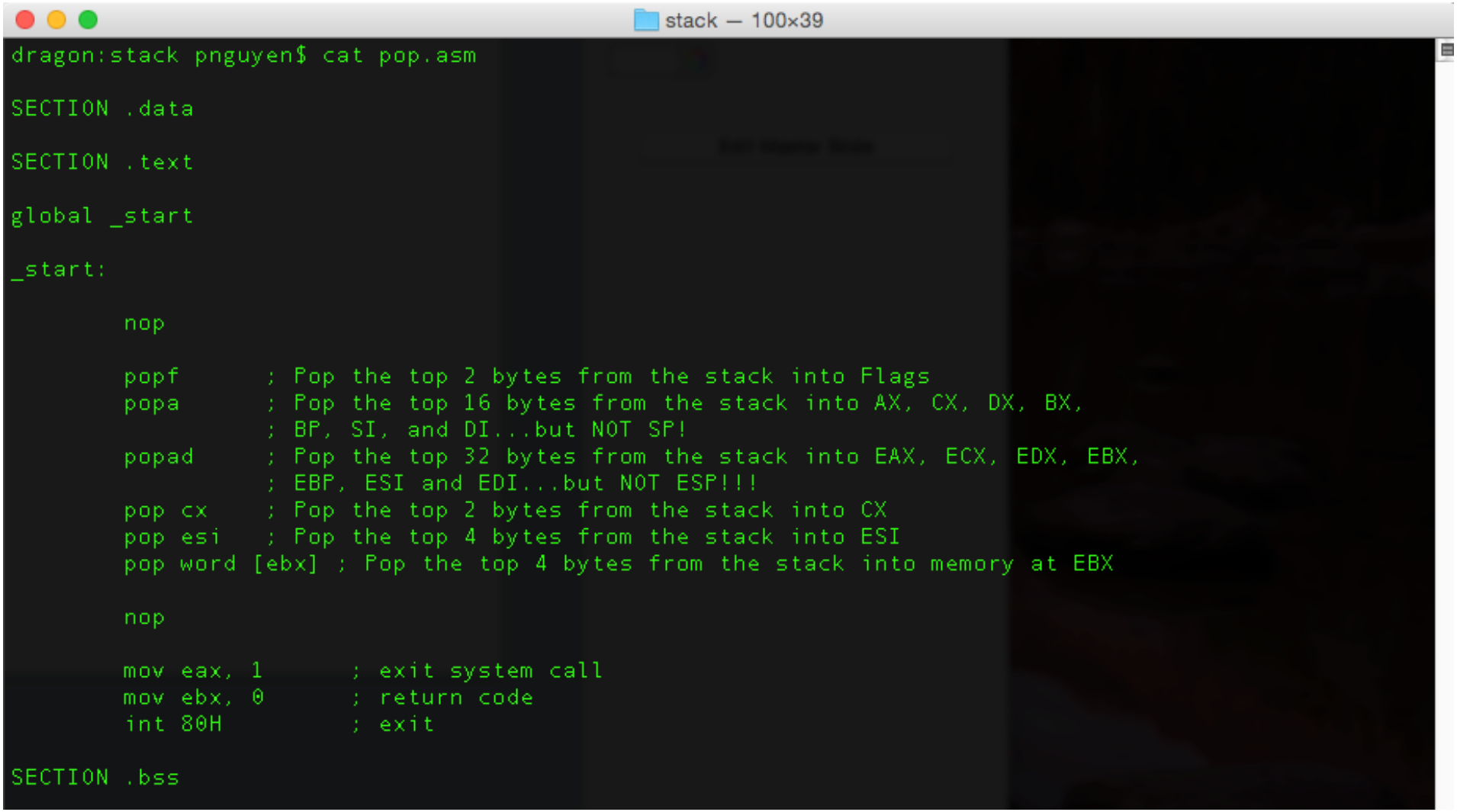
```
pushf      ; Push the Flags register
pusha       ; Push AX, CX, DX, BX, SP, BP, SI, and DI, in that order, all at
           ; once
pushad     ; Push EAX, ECX, EDX, EBX, ESP, ESP, EBP, ESI, and EDI, all at
           ; once
push ax    ; Push the AX register
push eax   ; Push the EAX register
push [bx]  ; Push the word stored in memory at BX
push [edx] ; Push the doubleword in memory at EDX
push edi   ; Push the EDI register
```


stack — 100×39
dragon:stack pnguyen$ cat push.asm
SECTION .data
SECTION .text
global _start
_start:
nop
pushf ; Push the Flags register
pusha ; Push AX, CX, DX, BX, SP, BP, SI, and DI, in that order, all at once
pushad ; Push EAX, ECX, EDX, EBX, ESP, ESP, EBP, ESI, and EDI, all at once
push ax ; Push the AX register
push eax ; Push the EAX register
push word [bx] ; Push the word stored in memory at BX
push dword [edx] ; Push the doubleword in memory at EDX
push edi ; Push the EDI register
nop
mov eax, 1 ; exit system call
mov ebx, 0 ; return code
int 80H ; exit
SECTION .bss

# POP

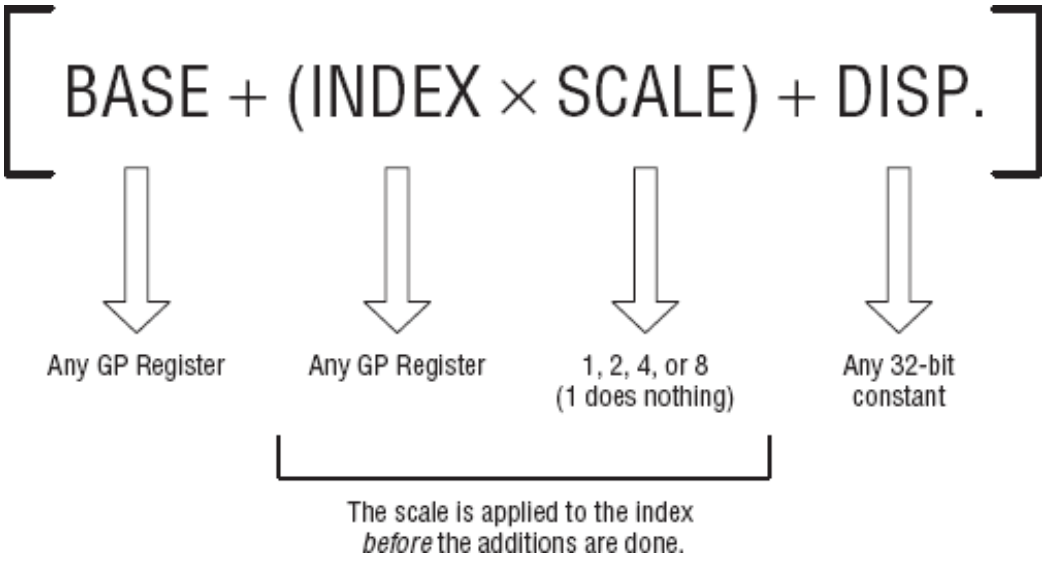



```
popf      ; Pop the top 2 bytes from the stack into Flags
popa      ; Pop the top 16 bytes from the stack into AX, CX, DX, BX,
          ; BP, SI, and DI...but NOT SP!
popad     ; Pop the top 32 bytes from the stack into EAX, ECX, EDX, EBX,
          ; EBP, ESI and EDI...but NOT ESP!!!
pop cx    ; Pop the top 2 bytes from the stack into CX
pop esi   ; Pop the top 4 bytes from the stack into ESI
pop [ebx] ; Pop the top 4 bytes from the stack into memory at EBX
```


stack — 100×39
dragon:stack pnguyen$ cat pop.asm
SECTION .data
SECTION .text
global _start
_start:
nop
popf ; Pop the top 2 bytes from the stack into Flags
popa ; Pop the top 16 bytes from the stack into AX, CX, DX, BX,
; BP, SI, and DI...but NOT SP!
popad ; Pop the top 32 bytes from the stack into EAX, ECX, EDX, EBX,
; EBP, ESI and EDI...but NOT ESP!!!
pop cx ; Pop the top 2 bytes from the stack into CX
pop esi ; Pop the top 4 bytes from the stack into ESI
pop word [ebx] ; Pop the top 4 bytes from the stack into memory at EBX
nop
mov eax, 1 ; exit system call
mov ebx, 0 ; return code
int 80H ; exit
SECTION .bss

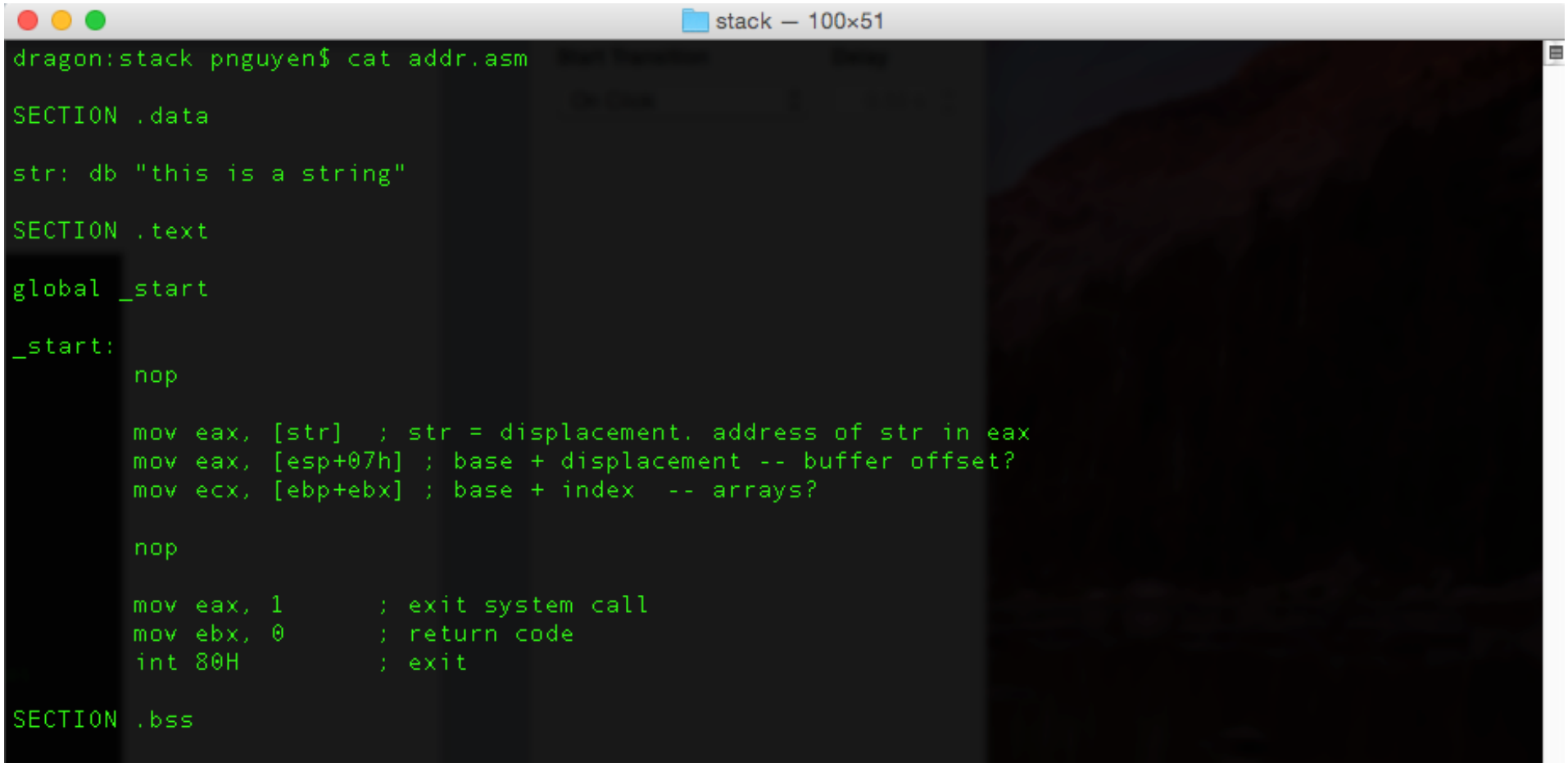
# ADDRESSING

Protected Mode Memory Addressing



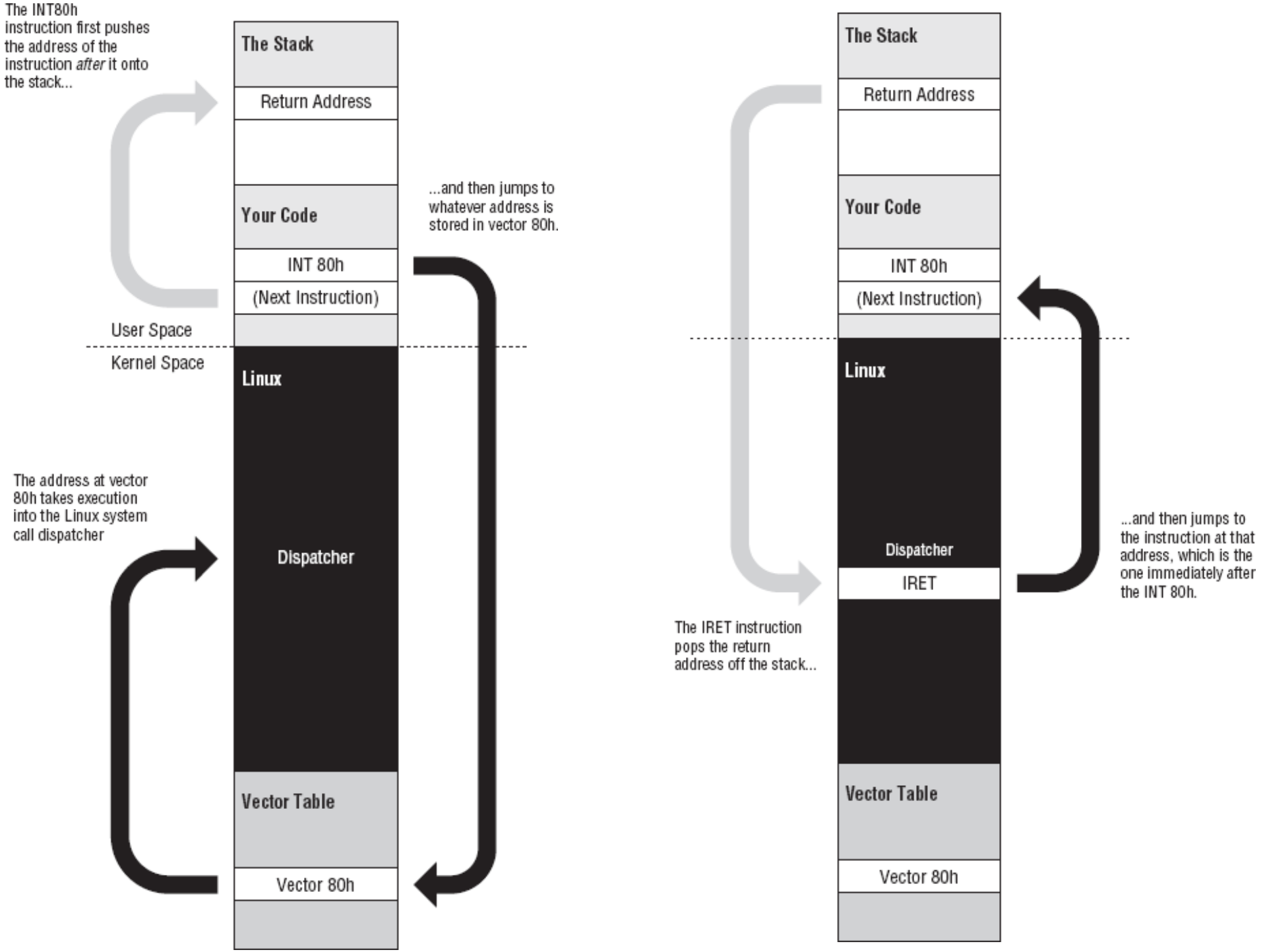

- The displacement may be any 32-bit constant. Obviously, 0, while legal, isn't useful.
- The scale must be one of the values 1, 2, 4, or 8. That's it! The value 1 is legal but doesn't do anything useful, so it's never used.
- The index register is multiplied by the scale before the additions are done. In other words, it's not (base + index) × scale. Only the index register is multiplied by the scale.
- All of the elements are optional and may be used in almost any combination.
- 16-bit and 8-bit registers may *not* be used in memory addressing.

| Scheme | Example | Description |
| --- | --- | --- |
| [BASE] | [edx] | Base only |
| [DISPLACEMENT] | [0F3h] or [<variable>] | Displacement, either literal constant or symbolic address |
| [BASE + DISPLACEMENT] | [ecx + 033h] | Base plus displacement |
| [BASE + INDEX] | [eax + ecx] | Base plus index |
| [INDEX × SCALE] | [ebx * 4] | Index times scale |
| [INDEX × SCALE + DISPLACEMENT] | [eax * 8 + 65] | Index times scale plus displacement |
| [BASE + INDEX × SCALE] | [esp + edi * 2] | Base plus index times scale |
| [BASE + INDEX × SCALE + DISPLACEMENT] | [esi + ebp * 4 + 9] | Base plus index times scale plus displacement |


stack — 100×51
dragon:stack pnguyen$ cat addr.asm
SECTION .data
str: db "this is a string"
SECTION .text
global _start
_start:
nop
mov eax, [str] ; str = displacement. address of str in eax
mov eax, [esp+07h] ; base + displacement -- buffer offset?
mov ecx, [ebp+ebx] ; base + index -- arrays?
nop
mov eax, 1 ; exit system call
mov ebx, 0 ; return code
int 80H ; exit
SECTION .bss

# SYSCALL

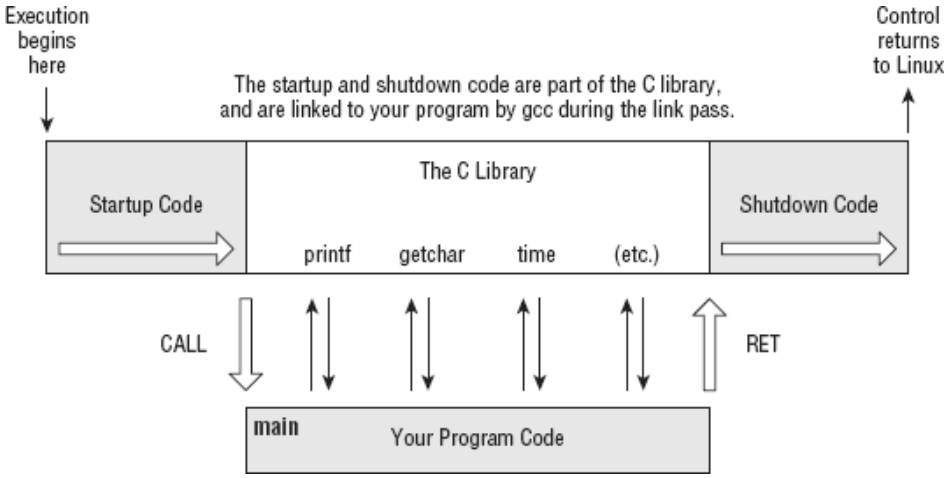
INT 80H

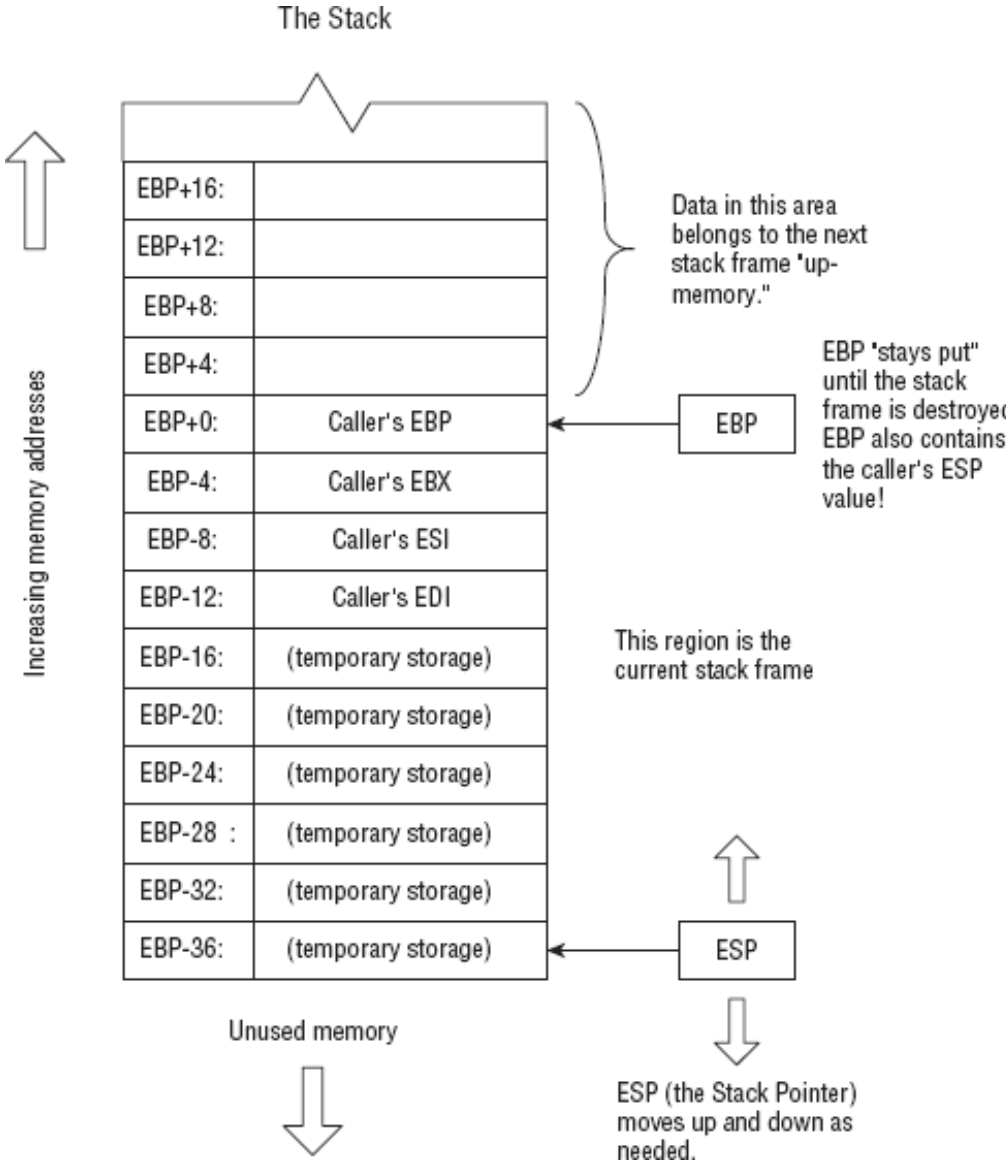

The INT80h instruction first pushes the address of the instruction *after* it onto the stack...
The Stack
Return Address
Your Code
INT 80h
(Next Instruction)
...and then jumps to whatever address is stored in vector 80h.
User Space
Kernel Space
Linux
The address at vector 80h takes execution into the Linux system call dispatcher
Dispatcher
Vector Table
Vector 80h
The Stack
Return Address
Your Code
INT 80h
(Next Instruction)
Linux
Dispatcher
IRET
...and then jumps to the instruction at that address, which is the one immediately after the INT 80h.
The IRET instruction pops the return address off the stack...
Vector Table
Vector 80h

# hello example

```
; hello.asm
;
; build:
; nasm -f elf -g -F stabs hello.asm
; ld -o hello hello.o

SECTION .data

msg:    db      "Hello Assembly World!", 22
len:    equ     $-msg

SECTION .bss

SECTION .text

global _start

_start:
        nop
        mov eax, 4      ; sys_write system call
        mov ebx, 1      ; std output
        mov ecx, msg    ; offset of msg
        mov edx, len    ; length of msg
        int 80H         ; invoke syscall
        nop
        mov eax, 1      ; exit system call
        mov ebx, 0      ; return code
        int 80H         ; exit
```

# CALLING C LIBRARY




## C Calling Conventions (callee = procedure)

---

Callee must save: EBX, ESP, EBP, ESI & EDI

---

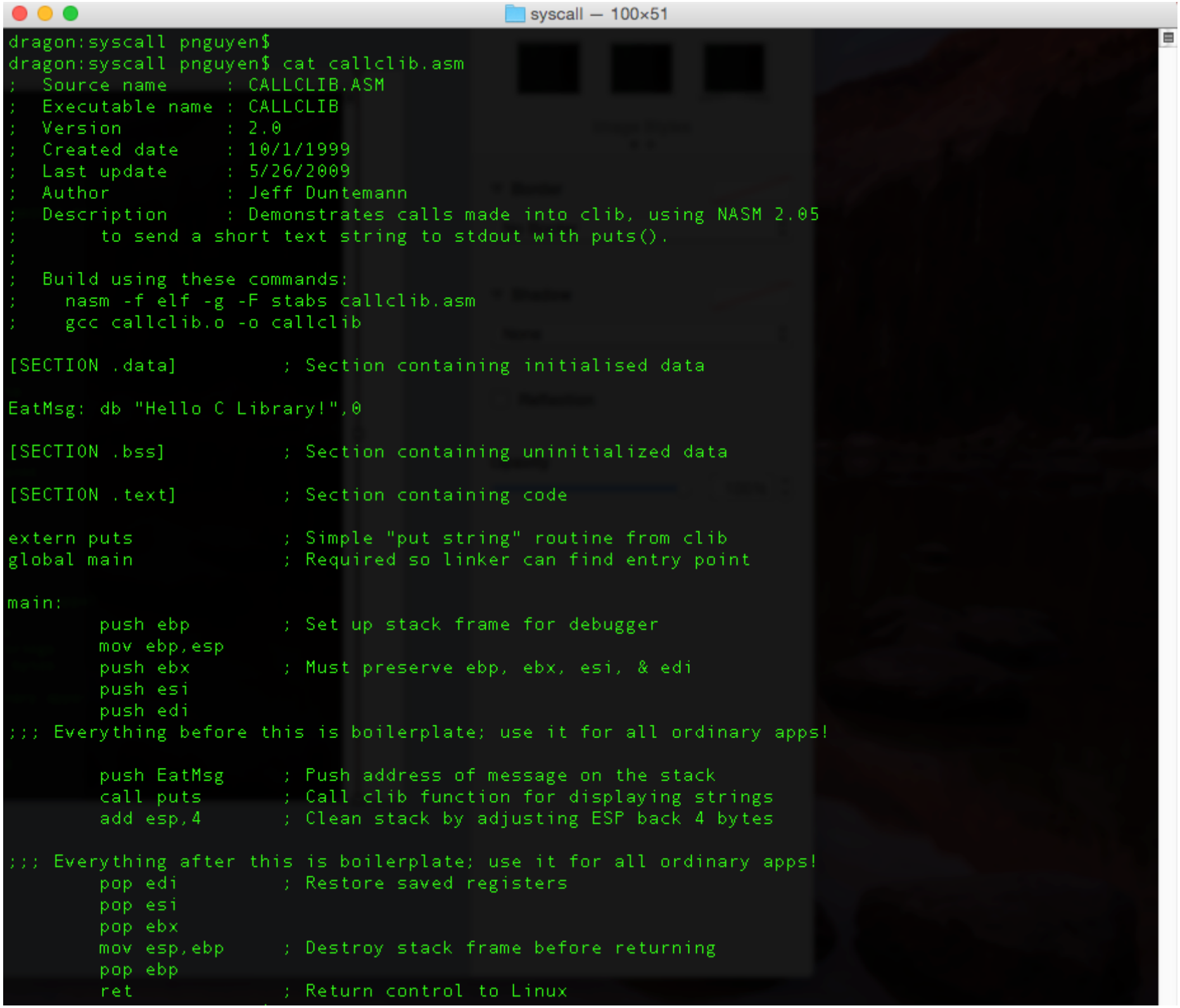
Callee's return value will be in EAX (if <= 32-bits).
If >32-bits, high bits in EDX.

---

Caller pass parameters on stack from right to left.
Example: Func( A, B, C). Push C, then B, and A.

---

Callee do not pop from Stack! Callers will pop or change SP offset upon return.


syscall — 100×51
dragon:syscall pnguyen$
dragon:syscall pnguyen$ cat callclib.asm
;  Source name      : CALLCLIB.ASM
;  Executable name : CALLCLIB
;  Version          : 2.0
;  Created date     : 10/1/1999
;  Last update      : 5/26/2009
;  Author           : Jeff Duntemann
;  Description      : Demonstrates calls made into clib, using NASM 2.05
;       to send a short text string to stdout with puts().
;
;  Build using these commands:
;    nasm -f elf -g -F stabs callclib.asm
;    gcc callclib.o -o callclib
[SECTION .data]          ; Section containing initialised data
EatMsg: db "Hello C Library!",0
[SECTION .bss]           ; Section containing uninitialized data
[SECTION .text]          ; Section containing code
extern puts              ; Simple "put string" routine from clib
global main              ; Required so linker can find entry point
main:
        push ebp         ; Set up stack frame for debugger
        mov ebp,esp
        push ebx         ; Must preserve ebp, ebx, esi, & edi
        push esi
        push edi
;;; Everything before this is boilerplate; use it for all ordinary apps!
        push EatMsg      ; Push address of message on the stack
        call puts        ; Call clib function for displaying strings
        add esp,4        ; Clean stack by adjusting ESP back 4 bytes
;;; Everything after this is boilerplate; use it for all ordinary apps!
        pop edi          ; Restore saved registers
        pop esi
        pop ebx
        mov esp,ebp      ; Destroy stack frame before returning
        pop ebp
        ret              ; Return control to Linux

**Callee must save: EBX, ESP, EBP, ESI & EDI**

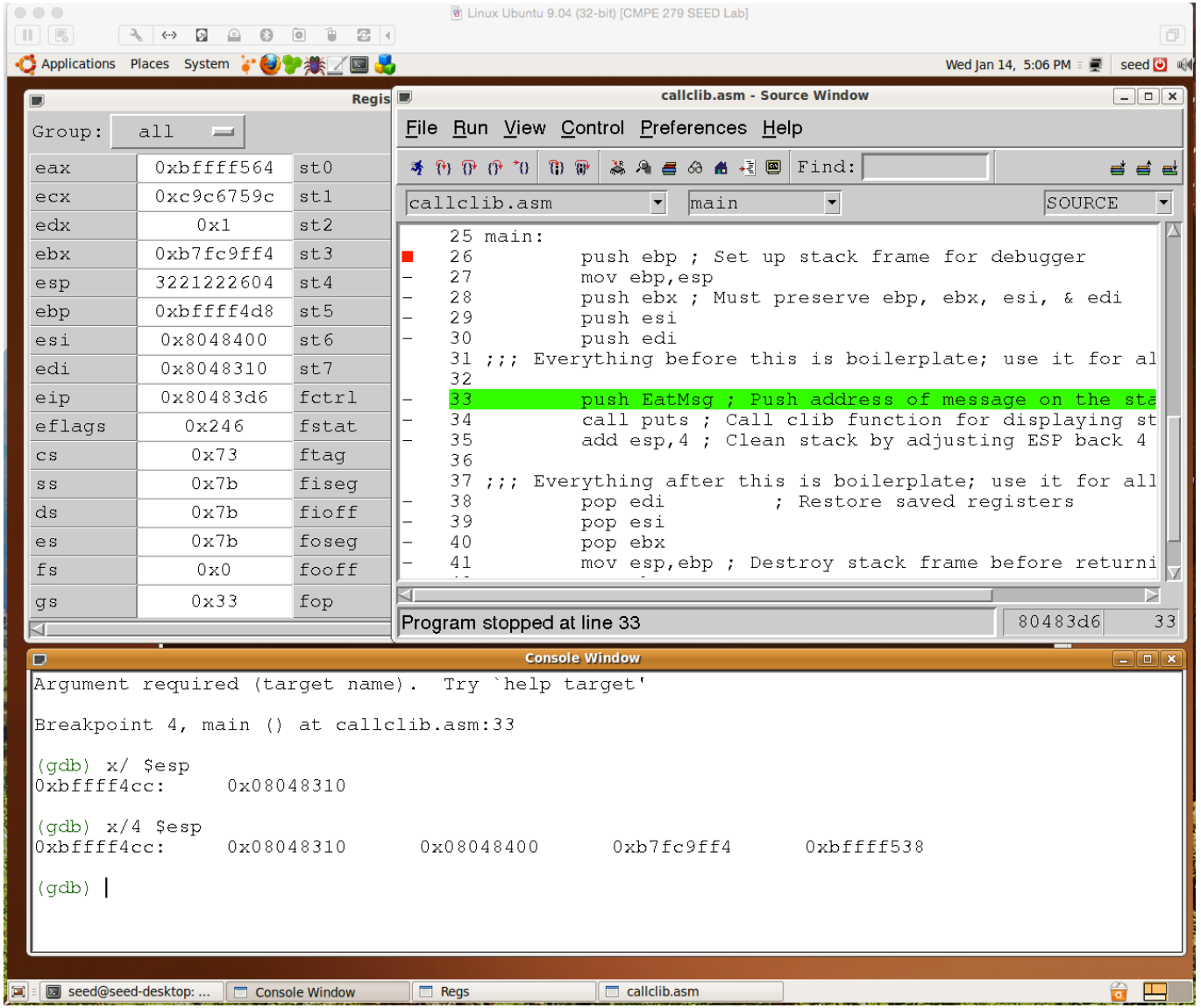
---

Caller pass parameters on stack from right to left.

---

Call “puts”

---



Callee must save:
EBX, ESP, EBP,
ESI & EDI

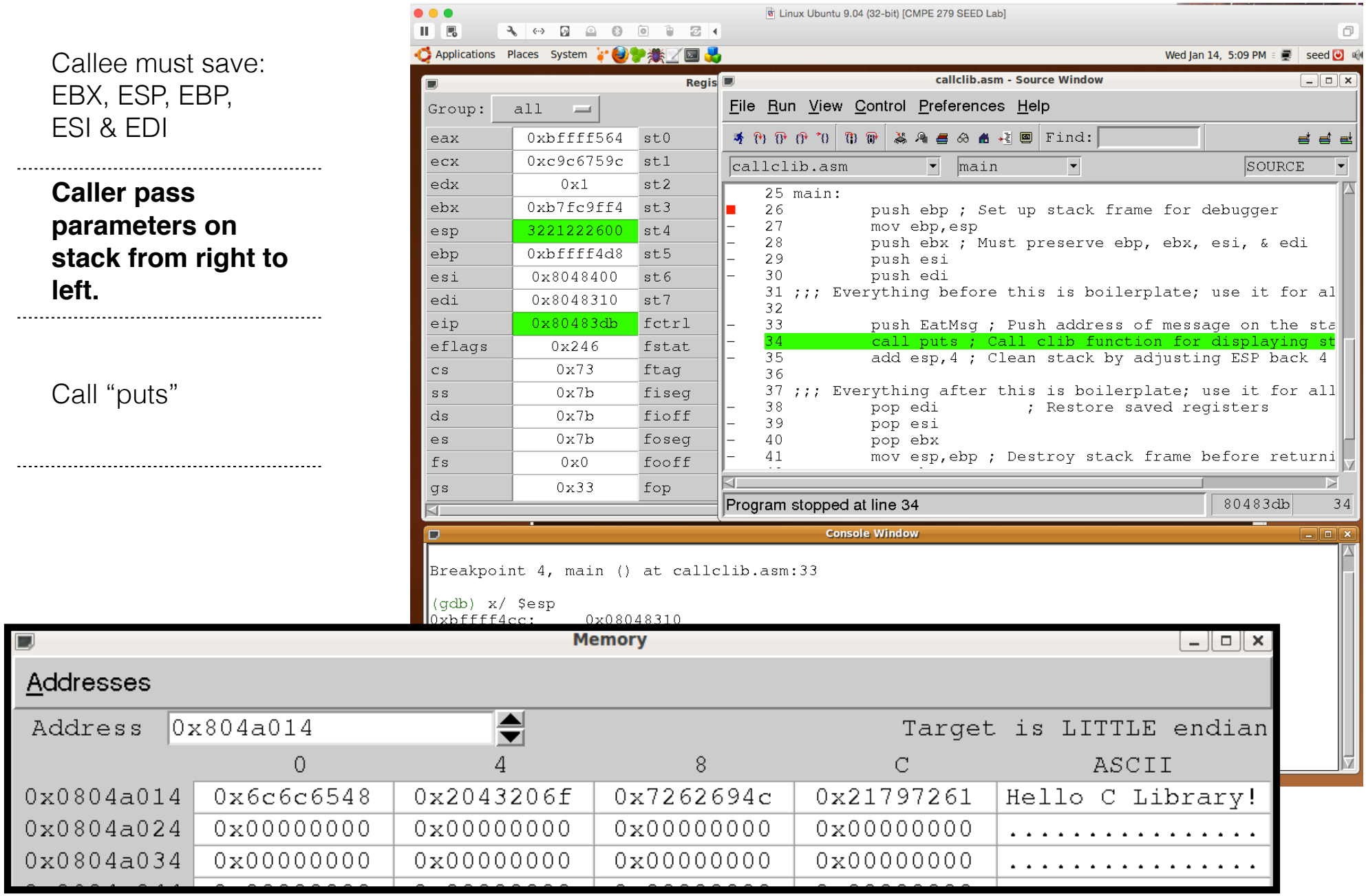
---

**Caller pass parameters on stack from right to left.**

---

Call “puts”

---



Callee must save:
EBX, ESP, EBP,
ESI & EDI

---

Caller pass parameters on stack from right to left.

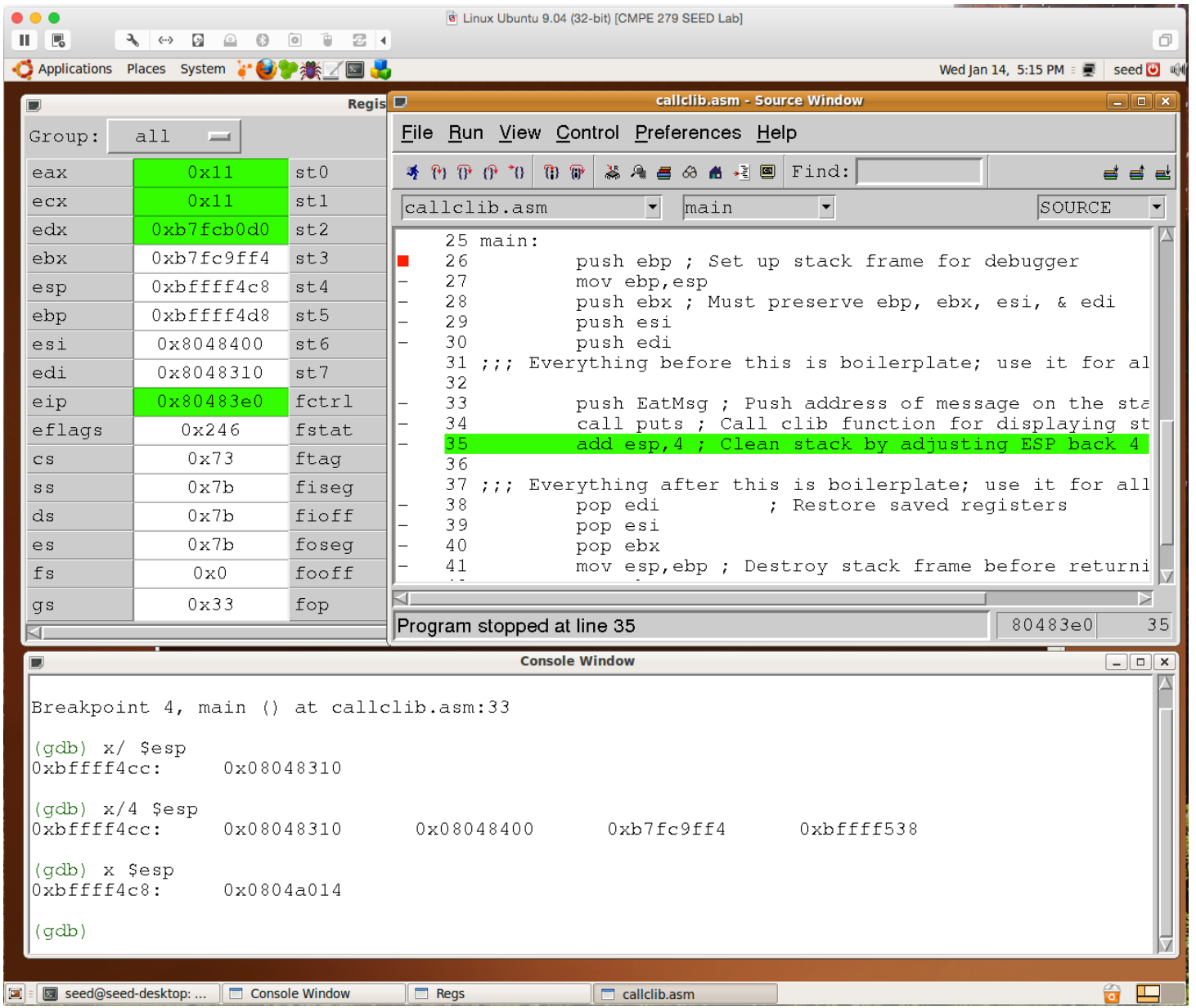
---

**Call “puts”**

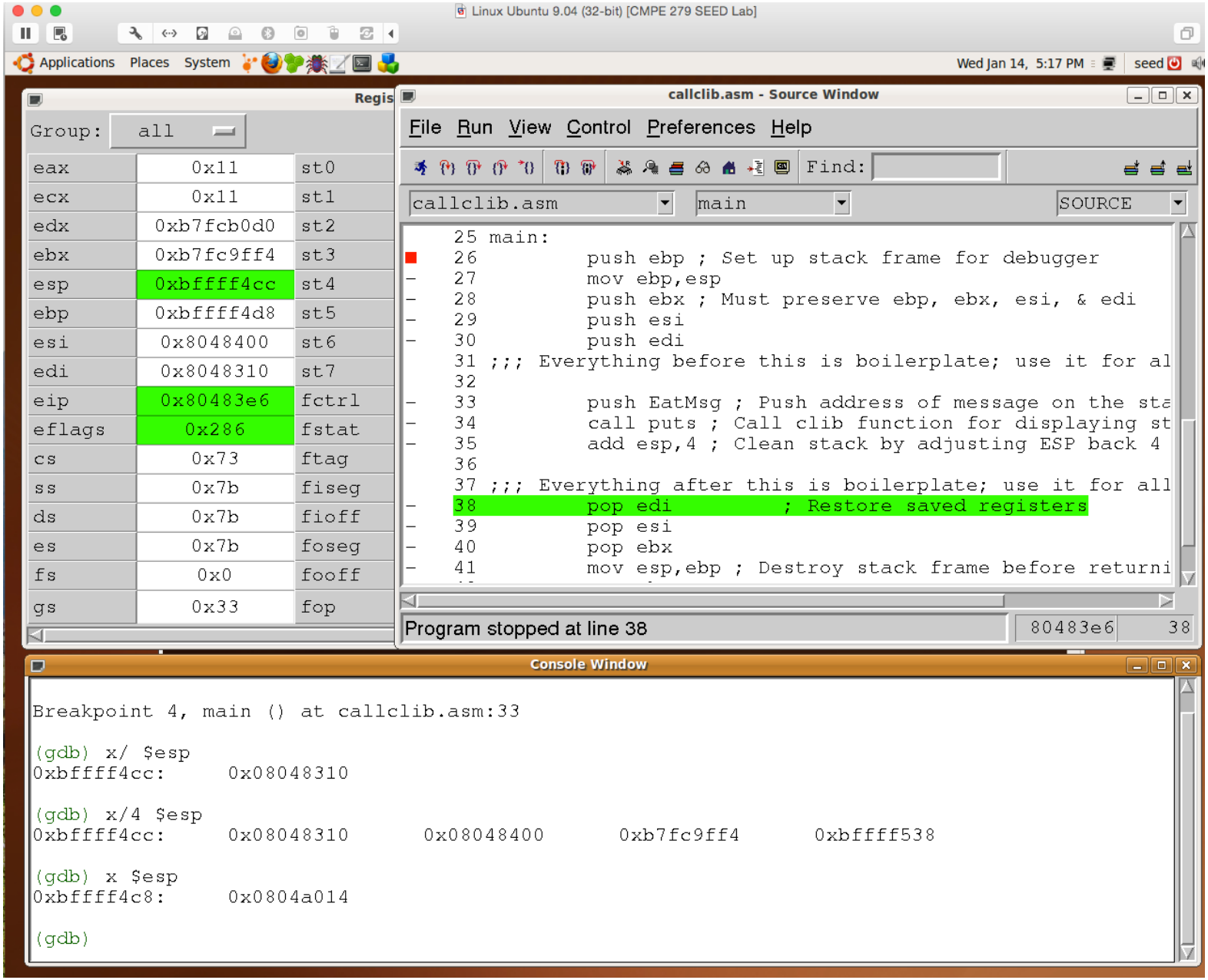
---

***Callee’s return value will be in EAX (if <= 32-bits).***

***If >32-bits, high bits in EDX.***



***Callee do not pop from Stack! Callers will pop or change SP offset upon return.***



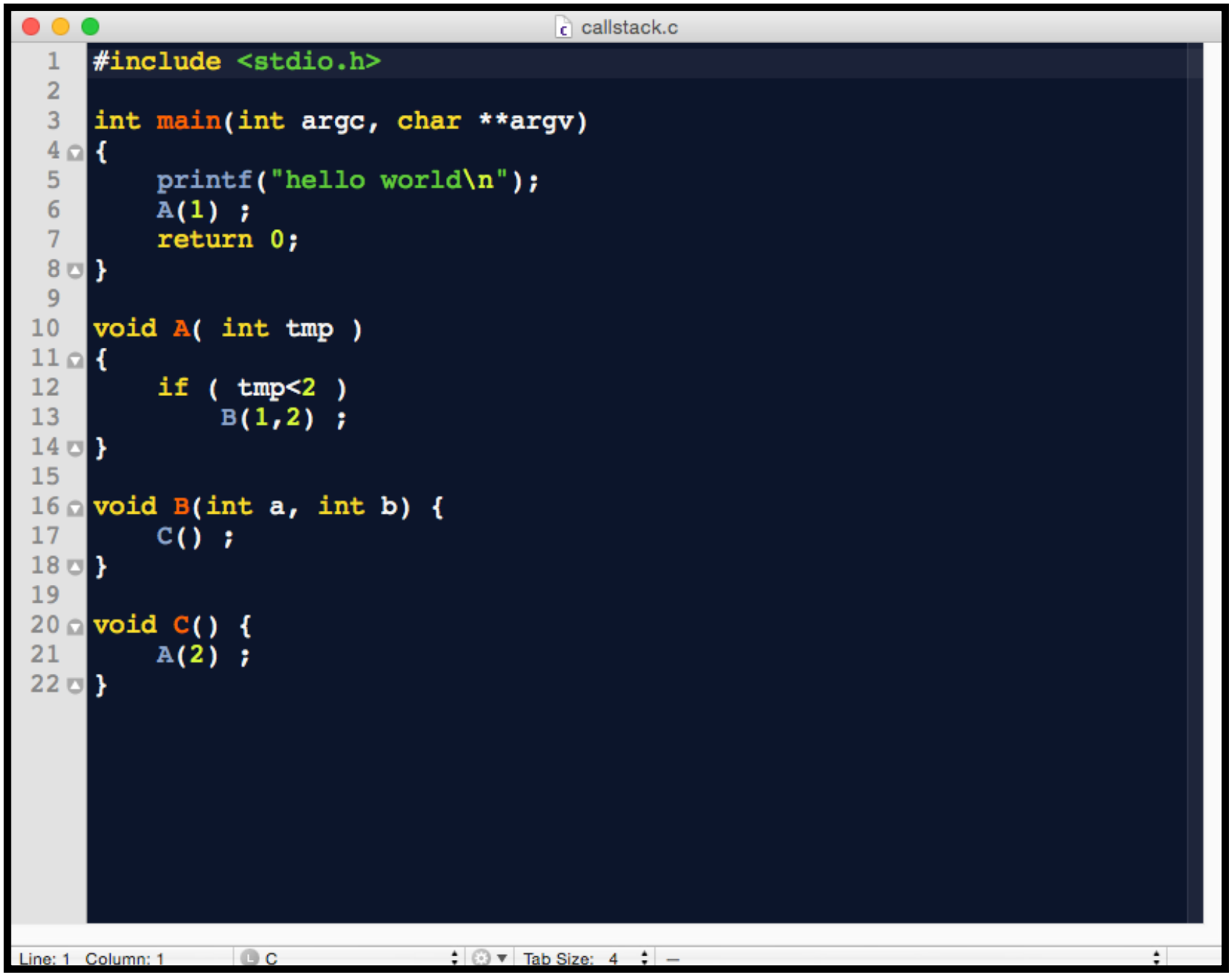
# MEMORY LAYOUT

layout.c

```c

#include<stdio.h>
#include<malloc.h>

int glb_uninit;          /* Part of BSS Segment -- global uninitialized variable, at runtime i
int glb_init = 10;       /* Part of DATA Segment -- global initialized variable */

void  foo(void)
{
    static int num = 0;         /* stack frame count */
    int autovar;         /* automatic variable/Local variable */
    int *ptr_foo = (int*)malloc(sizeof(int));
    if (++num == 4)             /* Creating four stack frames */
        return;
    printf("Stack frame number %d: address of autovar: %p\n", num, & autovar);
    printf("Address of heap allocated inside foo() %p\n",ptr_foo);
    foo();                      /* function call */
}

int main()
{
    char *p, *b, *nb;
    int *ptr_main = (int*)malloc(sizeof(int));
    printf("Text Segment:\n");
    printf("Address of main: %p\n", main);
    printf("Address of func foo: %p\n",foo);
    printf("Stack Locations:\n");
    foo();
    printf("Data Segment:\n");
    printf("Address of glb_init: %p\n", & glb_init);
    printf("BSS Segment:\n");
    printf("Address of glb_uninit: %p\n", & glb_uninit);
    printf("Heap Segment:\n");
    printf("Address of heap allocated inside main() %p\n",ptr_main);

    return 0;
}

```

Line: 1  Column: 1  C  Tab Size: 4

# C CALL STACK


callstack.c
#include <stdio.h>
int main(int argc, char **argv)
{
printf("hello world\n");
A(1) ;
return 0;
}
void A( int tmp )
{
if ( tmp<2 )
B(1,2) ;
}
void B(int a, int b) {
C() ;
}
void C() {
A(2) ;
}
Line: 1 Column: 1
C
Tab Size: 4