CS354: Machine Organization and Programming

Lecture 9
Wednesday the September 23th 2015

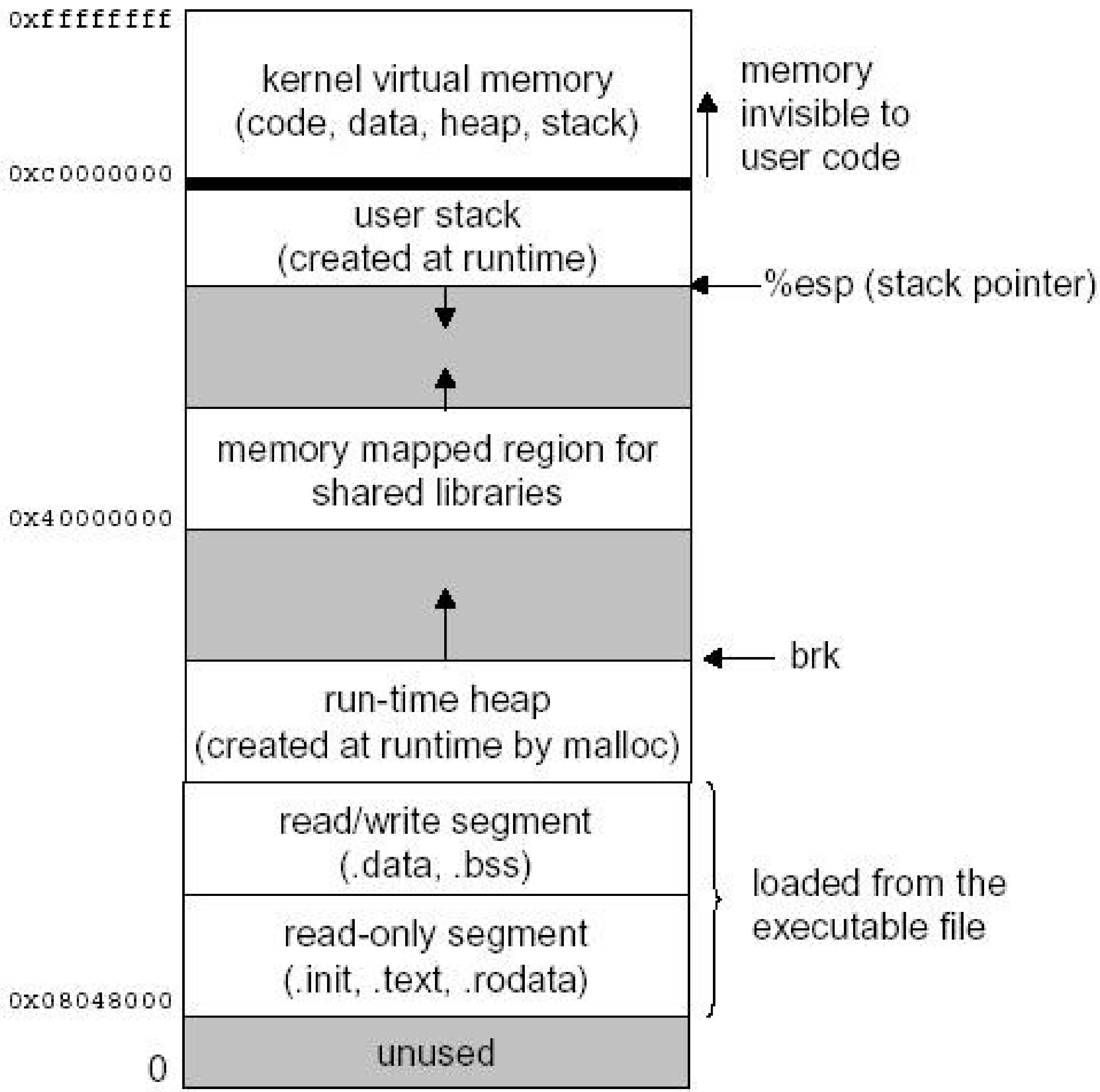
Section 2
Instructor: Leo Arulraj

Class Announcements

1. Take backups of your C files periodically. Saves lot of work in case bad things happen.
2. Brief session on C Programming aspects relevant to Assignment 1 in later part of next lecture. (Turns out I cannot go into details because that is part of the assignment).

Lecture Overview

- Stack related Data Movement operations
- Data Movement example
- Arithmetic instructions


0xffffffff
kernel virtual memory
(code, data, heap, stack)
memory
invisible to
user code
0xc0000000
user stack
(created at runtime)
%esp (stack pointer)
memory mapped region for
shared libraries
0x40000000
brk
run-time heap
(created at runtime by malloc)
read/write segment
(.data, .bss)
loaded from the
executable file
read-only segment
(.init, .text, .rodata)
0x08048000
unused
0

Stack Example: pushl, popl

Initially

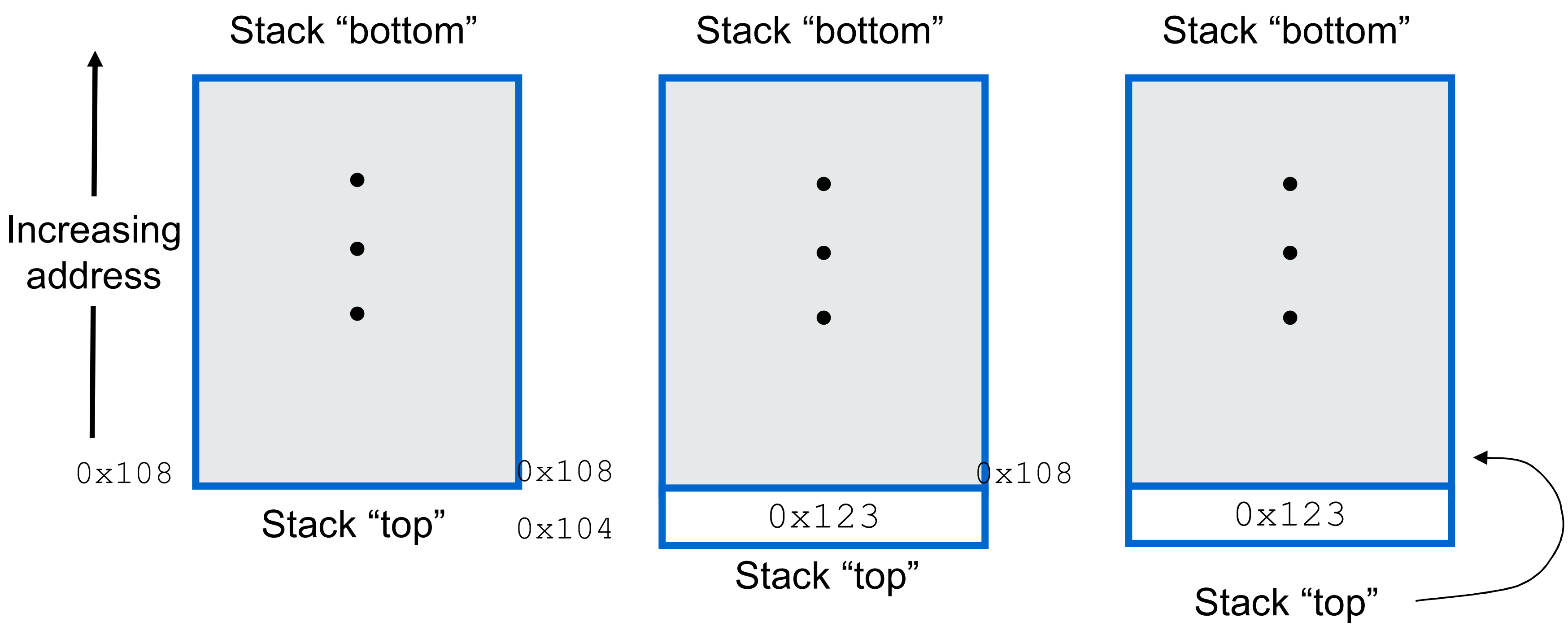
%eax	0x123
%edx	0
%esp	0x108

`pushl %eax`

%eax	0x123
%edx	0
%esp	0x104

`popl %edx`

%eax	0x123
%edx	0x123
%esp	0x108



pushl and popl

- pushl %ebp is equivalent to:

```
subl $4, %esp
movl %ebp, (%esp)
```

- popl %eax is equivalent to:

```
movl (%esp), %eax
addl $4, %esp
```

Data Movement Example (Trace through during lecture)

```
.data
value:
        .long 52713
heapvar:
        .long 0x5000
.text
.globl main
main:
        movl $103, %eax
        movl %eax, %esi
        movl value, %ebx
```

Continued from left column:

```
movl %esp, %ecx
movl %eax, (%ecx)
movl heapvar, %eax
movl 8(%eax), %edx
push %edx
push $207
pop %edi
movl $3,%ecx
movl (%eax, %ecx, 4), %edx
ret
```

Arithmetic Instructions

`leal`	S, D	(load effective address) D gets the address defined by S
`inc`	D	D gets D + 1 (two's complement)
`dec`	D	D gets D - 1 (two's complement)
`neg`	D	D gets -D (two's complement additive inverse)
`add`	S, D	D gets D + S (two's complement)
`sub`	S, D	D gets D - S (two's complement)
`imul`	S, D	D gets D * S (two's complement integer multiplication)

More Arithmetic Instructions, with 64 bits of results

`imull`	S	%edx‖%eax gets 64-bit *two's complement* product of S * %eax
`mull`	S	%edx‖%eax gets 64-bit *unsigned* product of S * %eax
`idivl`	S	*two's complement* division of %edx‖%eax / S; %edx gets remainder, and %eax gets quotient
`divl`	S	*unsigned* division of %edx‖%eax / S; %edx gets remainder, and %eax gets quotient

Notice *implied* use of %eax and %edx.

`leal` is commonly used to calculate addresses. Examples:

```
leal 8(%eax), %edx
```

- 8 + contents of eax goes into edx
- used for pointer arithmetic in C
- very convenient for acquiring the address of an array element

```
leal (%eax, %ecx, 4), %edx
```

- contents of eax + 4 * contents of ecx goes into edx
- *even more convenient* for addresses of array elements, where eax has base address, ecx has the index, and each element is 4 bytes

Examples

Assume %eax is x and %ecx is y
and %edx=10, address 10 has value 100

1. leal 6(%eax), %edx :: ?
2. leal 9(%eax,%ecx,2), %edx :: ?
3. addl %ecx, (%edx) :: ?
4. decl %ecx :: ?

Examples

Assume %eax is x and %ecx is y
and %edx=10, address 10 has value 100

1. leal 6(%eax), %edx :: 6+x
2. leal 9(%eax,%ecx,2), %edx :: 9 + x + 2y
3. addl %ecx, (%edx) :: (y +100) stored @ address 10
4. decl %ecx :: (y-1) stored in %ecx

Examples

Assume x at %ebp+8, y at %ebp+12, z at %ebp+16

1 movl 16(%ebp), %eax	*z*
2 leal (%eax,%eax,2), %eax	*z*3*
3 sall $4, %eax	*t2 = z*48*
4 movl 12(%ebp), %edx	*y*
5 addl 8(%ebp), %edx	*t1 = x+y*
6 andl $65535, %edx	*t3 = t1&0xFFFF*
7 imull %edx, %eax	*t4 = t2*t3*