

Roadmap

C:

```
car *c = malloc(sizeof(car));
c->miles = 100;
c->gals = 17;
float mpg = get_mpg(c);
free(c);
```

Java:

```
Car c = new Car();
c.setMiles(100);
c.setGals(17);
float mpg =
    c.getMPG();
```

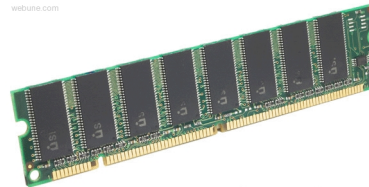
Assembly
language:

```
get_mpg:
    pushq    %rbp
    movq     %rsp, %rbp
    ...
    popq     %rbp
    ret
```

Machine
code:

```
0111010000011000
100011010000010000000010
1000100111000010
110000011111101000011111
```

Computer
system:



Memory & data
Integers & floats
Machine code & C
x86 assembly
Procedures & stacks
Arrays & structs
Memory & caches
Processes
Virtual memory
Memory allocation
Java vs. C

OS:



Section 4: x86 Assembly Programming

- Move instructions, registers, and operands
- Memory addressing modes ✓
- swap example: 32-bit vs. 64-bit
- Arithmetic operations ✓
- Condition codes ✓
- Conditional and unconditional branches)
- Loops ✓
- Switch statements)

Three Basic Kinds of Instructions

■ Transfer data between memory and register

- Load data from memory into register
 - $\%reg = \text{Mem}[\text{address}]$
- Store register data into memory
 - $\text{Mem}[\text{address}] = \%reg$

Remember:
memory is indexed
just like an array[]!

■ Perform arithmetic function on register or memory data

- $c = a + b;$

■ Transfer control

- Unconditional jumps to/from procedures
- Conditional branches

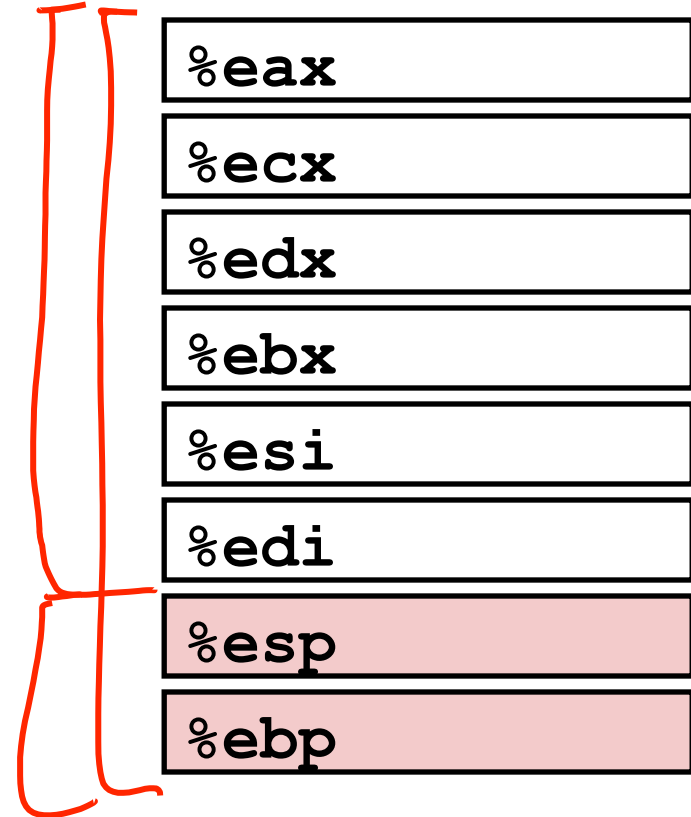
Moving Data: IA32

■ Moving Data

- movx Source, Dest
- x is one of {b, w, l}

- movl Source, Dest:
Move 4-byte “long word”
- movw Source, Dest:
Move 2-byte “word”
- movb Source, Dest:
Move 1-byte “byte”
?

■ Lots of these in typical code



Moving Data: IA32

■ Moving Data

`movl Source, Dest;`

■ Operand Types

- **Immediate:** Constant integer data
 - Example: `$0x400`, `$-533`
 - Like C constant, but prefixed with ``$'`
 - Encoded with 1, 2, or 4 bytes
- **Register:** One of 8 integer registers
 - Example: `%eax`, `%edx`
 - But `%esp` and `%ebp` reserved for special use
 - Others have special uses for particular instructions
- **Memory:** 4 consecutive bytes of memory at address given by register
 - Simplest example: `(%eax)`
 - Various other “address modes”

<code>%eax</code>
<code>%ecx</code>
<code>%edx</code>
<code>%ebx</code>
<code>%esi</code>
<code>%edi</code>
<code>%esp</code>
<code>%ebp</code>

movl Operand Combinations

	<u>Source</u>	Dest	Src, Dest	C Analog
<u>movl</u>	<u>Imm</u>	<u>Reg</u>	movl <u>\$0x4</u> , %eax	var_a = 0x4;
		<u>Mem</u>	movl <u>\$-147</u> , (<u>%eax</u>)	*p_a = -147;
	<u>Reg</u>	<u>Reg</u>	movl %eax, %edx	var_d = var_a;
		<u>Mem</u>	movl %eax, (<u>%edx</u>)	*p_d = var_a;
	<u>Mem</u>	<u>Reg</u>	movl (<u>%eax</u>), %edx	var_d = *p_a;

Handwritten annotations: Red circles around 'Imm', 'Reg', and 'Mem' in the Source column. Red circles around 'Reg' and 'Mem' in the Dest column. Red circles around '%eax', '%edx', and '(%eax)' in the Src, Dest column. Red circles around 'var_a', 'p_a', 'd', and '*p_a' in the C Analog column. Red arrows indicate the flow of data and the mapping between the instruction components and the C code.

Cannot do memory-memory transfer with a single instruction.

Memory Addressing Modes: Basic

■ Indirect

(R)

Mem[Reg[R]]

- Register R specifies the memory address

`movl (%ecx), %eax`

■ Displacement

D(R)

Mem[Reg[R]+D]

- Register R specifies a memory address
 - (e.g. the start of some memory region)
- Constant displacement D specifies the offset from that address

`movl 8 (%ebp), %edx`

$\%edx = \text{Mem}[\%ebp + 8]$

Using Basic Addressing Modes

```
void swap(int *xp, int *yp)
{
    int t0 = *xp;
    int t1 = *yp;
    *xp = t1;
    *yp = t0;
}
```

swap:

```
pushl %ebp
movl  %esp,%ebp
pushl %ebx
```

Set
Up

```
movl 12(%ebp),%ecx
movl 8(%ebp),%edx
movl (%ecx),%eax
movl (%edx),%ebx
movl %eax, (%edx)
movl %ebx, (%ecx)
```

Body

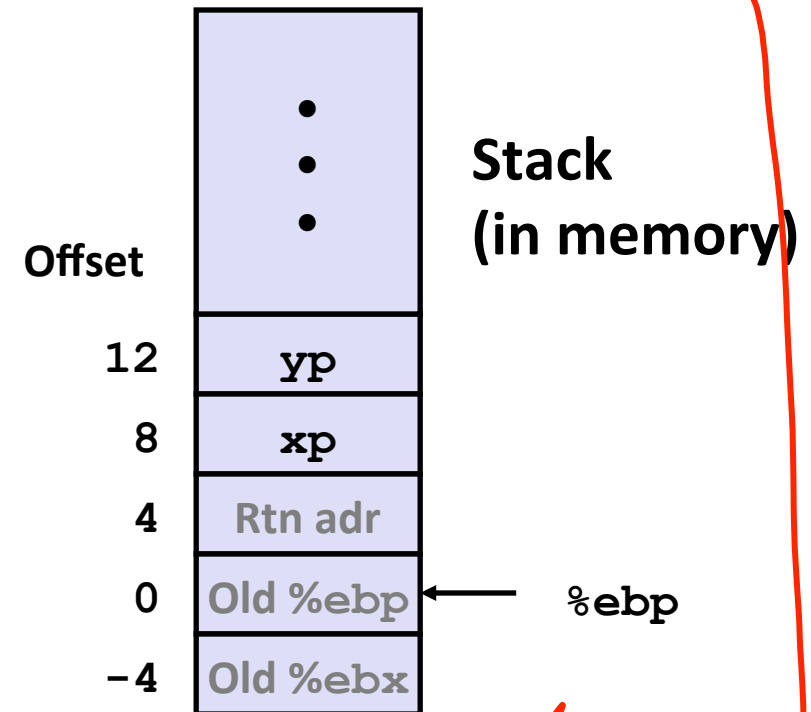
```
movl -4(%ebp),%ebx
movl %ebp,%esp
popl %ebp
ret
```

Finish

Understanding Swap

```
void swap(int *xp, int *yp)
{
    int t0 = *xp;
    int t1 = *yp;
    *xp = t1;
    *yp = t0;
}
```

Register	Value
<u>%ecx</u>	<u>yp</u>
<u>%edx</u>	<u>xp</u>
<u>%eax</u>	<u>t1</u>
<u>%ebx</u>	<u>t0</u>



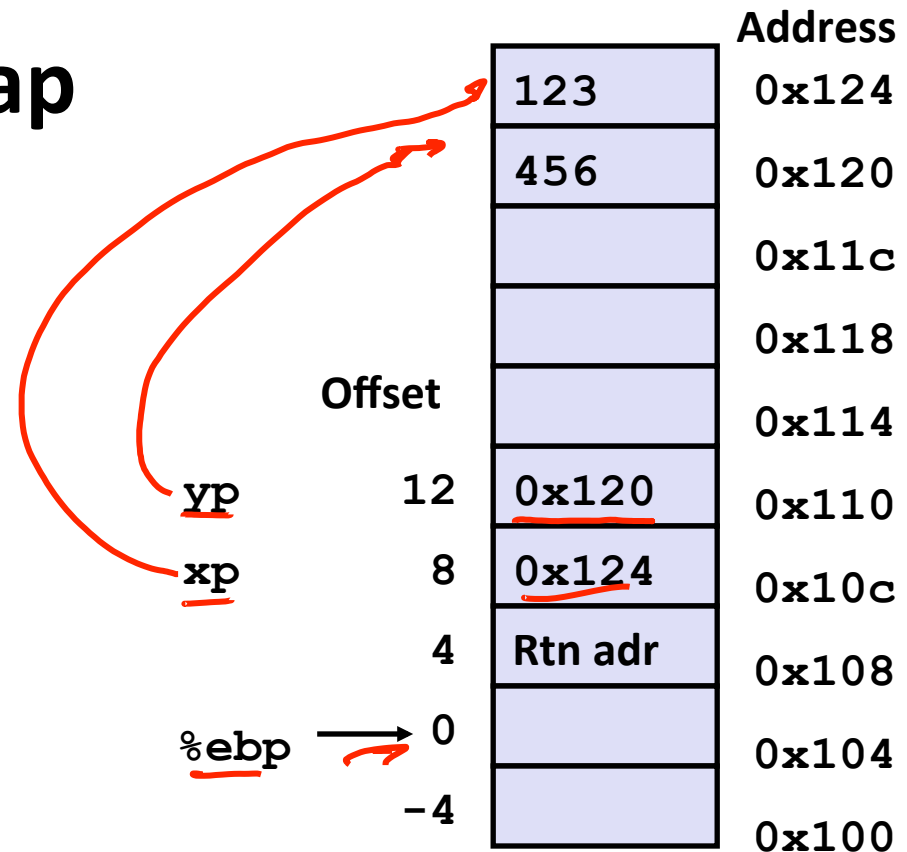
```

movl 12(%ebp), %ecx    # ecx = yp
movl 8(%ebp), %edx     # edx = xp
movl (%ecx), %eax      # eax = *yp (t1)
movl (%edx), %ebx      # ebx = *xp (t0)
movl %eax, (%edx)      # *xp = eax
movl %ebx, (%ecx)      # *yp = ebx

```

Understanding Swap

%eax	
%edx	
%ecx	
%ebx	
%esi	
%edi	
%esp	
%ebp	0x104



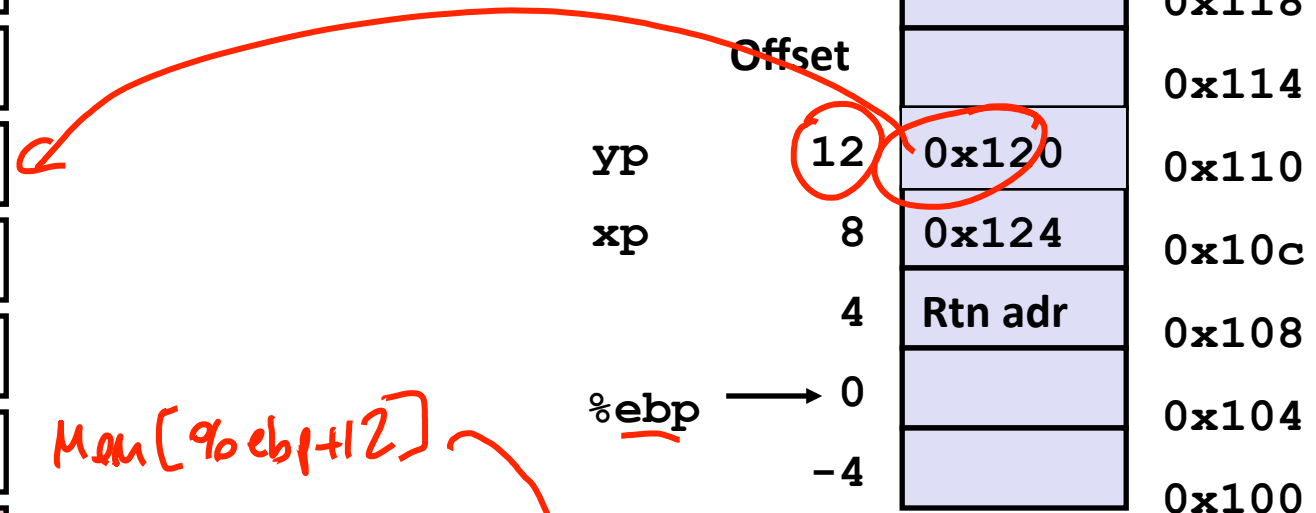
```

movl 12(%ebp), %ecx    # ecx = yp
movl 8(%ebp), %edx     # edx = xp
movl (%ecx), %eax      # eax = *yp (t1)
movl (%edx), %ebx      # ebx = *xp (t0)
movl %eax, (%edx)      # *xp = eax
movl %ebx, (%ecx)      # *yp = ebx

```

Understanding Swap

%eax	
%edx	
%ecx	0x120
%ebx	
%esi	
%edi	
%esp	
%ebp	0x104



→ `movl 12(%ebp), %ecx`

`movl 8(%ebp), %edx`

`movl (%ecx), %eax`

`movl (%edx), %ebx`

`movl %eax, (%edx)`

`movl %ebx, (%ecx)`

`ecx = yp`

`edx = xp`

`eax = *yp (t1)`

`ebx = *xp (t0)`

`*xp = eax`

`*yp = ebx`

Understanding Swap

%eax	
%edx	0x124
%ecx	0x120
%ebx	
%esi	
%edi	
%esp	
%ebp	0x104

	Offset		Address
		123	0x124
		456	0x120
			0x11c
			0x118
			0x114
yp	12	0x120	0x110
xp	8	0x124	0x10c
	4	Rtn adr	0x108
%ebp	0		0x104
	-4		0x100

```

movl 12(%ebp), %ecx      # ecx = yp
movl 8(%ebp), %edx      # edx = xp
movl (%ecx), %eax        # eax = *yp (t1)
movl (%edx), %ebx        # ebx = *xp (t0)
movl %eax, (%edx)        # *xp = eax
movl %ebx, (%ecx)        # *yp = ebx

```

Understanding Swap

%eax	456
%edx	0x124
%ecx	0x120
%ebx	
%esi	
%edi	
%esp	
%ebp	0x104

		Address
	123	0x124
	456	0x120
		0x11c
		0x118
		0x114
yp	12	0x120
xp	8	0x124
	4	Rtn adr
%ebp	0	0x108
		0x104
	-4	0x100

```

movl 12(%ebp), %ecx
movl 8(%ebp), %edx
movl (%ecx), %eax
movl (%edx), %ebx
movl %eax, (%edx)
movl %ebx, (%ecx)

```

```

# ecx = yp
# edx = xp
# eax = *yp (t1)
# ebx = *xp (t0)
# *xp = eax
# *yp = ebx

```

Understanding Swap

%eax	456
%edx	0x124
%ecx	0x120
%ebx	123
%esi	
%edi	
%esp	
%ebp	0x104

		Address
	123	0x124
	456	0x120
		0x11c
		0x118
		0x114
yp	12	0x120
xp	8	0x124
	4	Rtn adr
%ebp	0	0x108
		0x104
	-4	0x100

```

movl 12(%ebp), %ecx
movl 8(%ebp), %edx
movl (%ecx), %eax
movl (%edx), %ebx
movl %eax, (%edx)
movl %ebx, (%ecx)

```

```

# ecx = yp
# edx = xp
# eax = *yp (t1)
# ebx = *xp (t0)
# *xp = eax
# *yp = ebx

```

Understanding Swap

%eax	456
%edx	0x124
%ecx	0x120
%ebx	123
%esi	
%edi	
%esp	
%ebp	0x104

		Address
		0x124
		0x120
		0x11c
		0x118
		0x114
yp	12	0x120
xp	8	0x124
	4	Rtn adr
%ebp	0	0x108
		0x104
	-4	0x100

```

movl 12(%ebp), %ecx      # ecx = yp
movl 8(%ebp), %edx       # edx = xp
movl (%ecx), %eax        # eax = *yp (t1)
movl (%edx), %ebx        # ebx = *xp (t0)
movl %eax, (%edx)      # *xp = eax
movl %ebx, (%ecx)        # *yp = ebx

```

Understanding Swap

%eax	456
%edx	0x124
%ecx	0x120
%ebx	123
%esi	
%edi	
%esp	
%ebp	0x104

		Offset	Address
		456	0x124
		123	0x120
			0x11c
			0x118
			0x114
yp	12	0x120	0x110
xp	8	0x124	0x10c
	4	Rtn adr	0x108
%ebp	→ 0		0x104
	-4		0x100

```
movl 12(%ebp), %ecx      # ecx = yp
movl 8(%ebp), %edx       # edx = xp
movl (%ecx), %eax        # eax = *yp (t1)
movl (%edx), %ebx        # ebx = *xp (t0)
movl %eax, (%edx)        # *xp = eax
movl %ebx, (%ecx)      # *yp = ebx
```