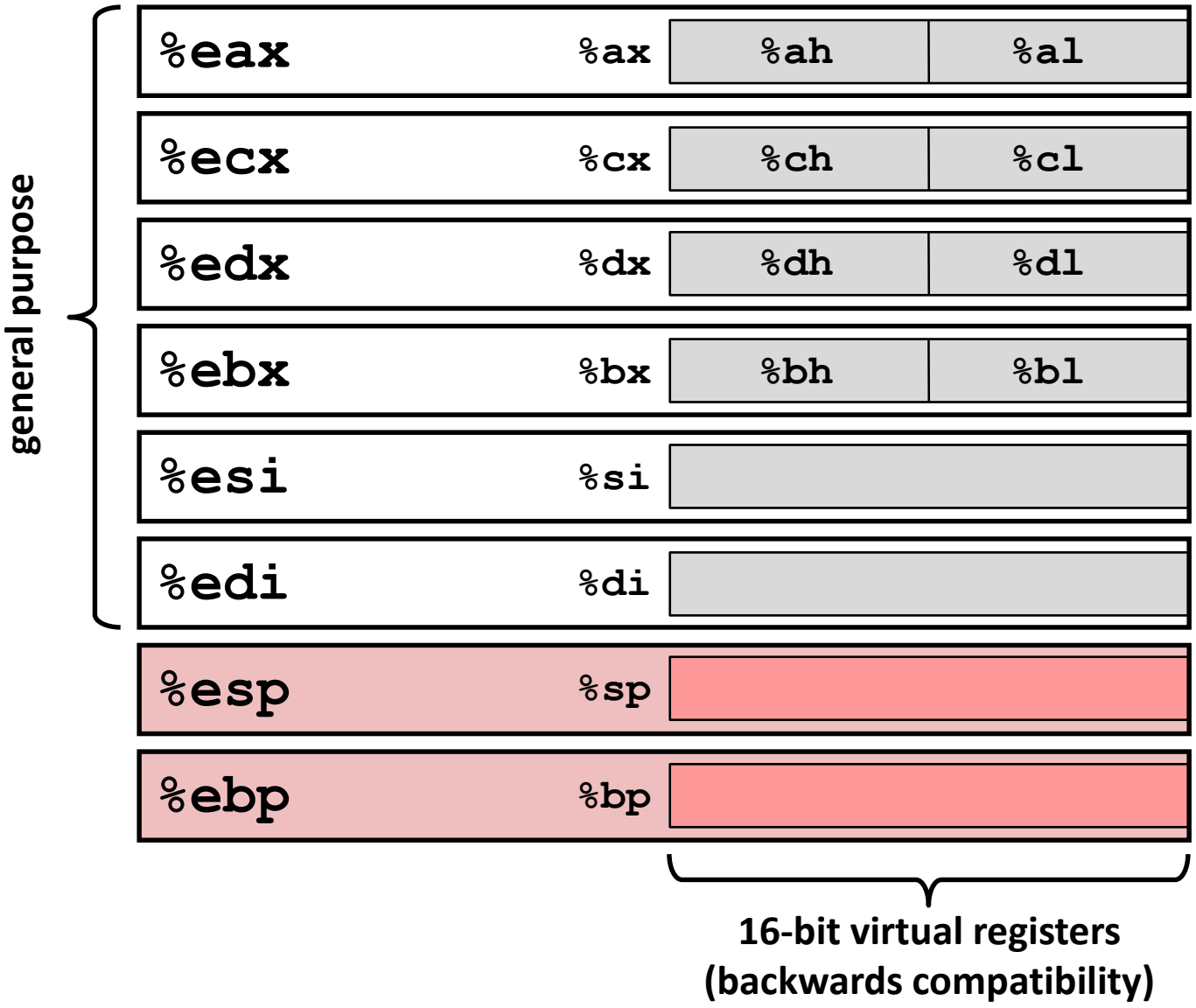
Week 07
Machine-Level Programming I: Basics

Today: Machine Programming I: Basics

- History of Intel processors and architectures
- C, assembly, machine code
- **Assembly Basics: Registers, operands, move**
- Intro to x86-64

Integer Registers (IA32)

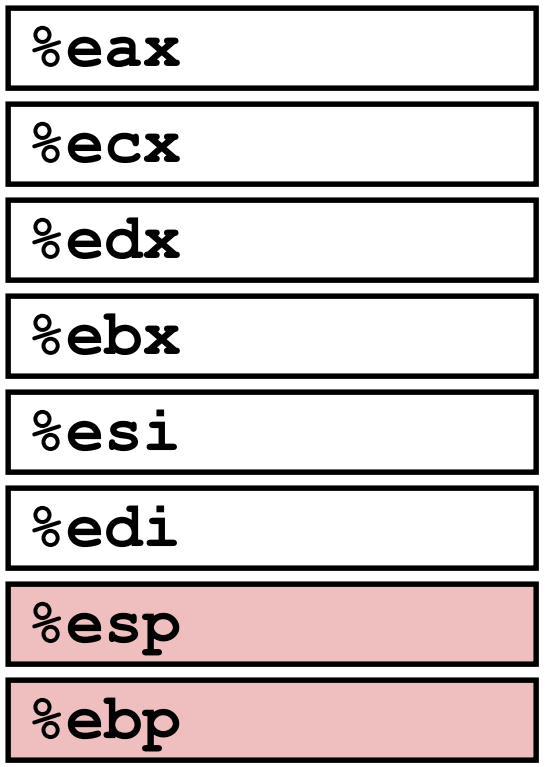


Moving Data: IA32

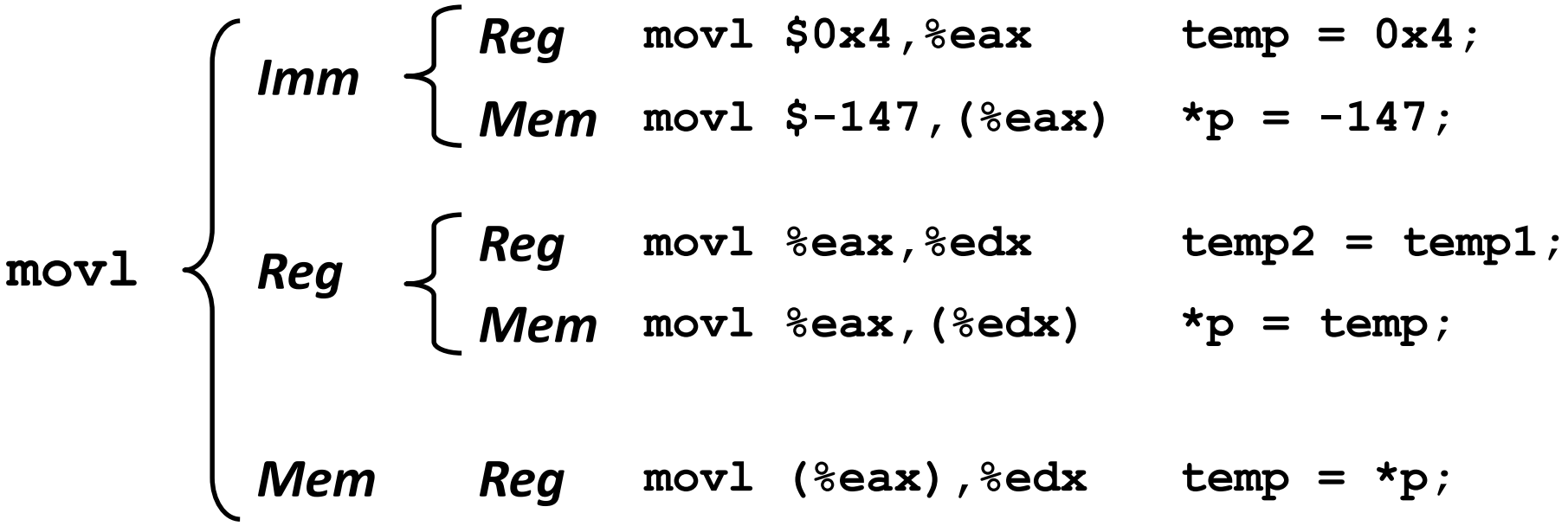
- **Moving Data**

  `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"



`movl` Operand Combinations



Cannot do memory-memory transfer with a single instruction

Simple Memory Addressing Modes

- **Normal (R) Mem[Reg[R]]**
  - Register R specifies memory address

```
movl (%ecx),%eax
```

- **Displacement D(R) Mem[Reg[R]+D]**
  - Register R specifies start of memory region
  - Constant displacement D specifies offset

```
movl 8(%ebp),%edx
```

Using Simple 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

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

popl    %ebx
popl    %ebp
ret
```

Set Up

Body

Finish

Using Simple 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

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

popl    %ebx
popl    %ebp
ret
```

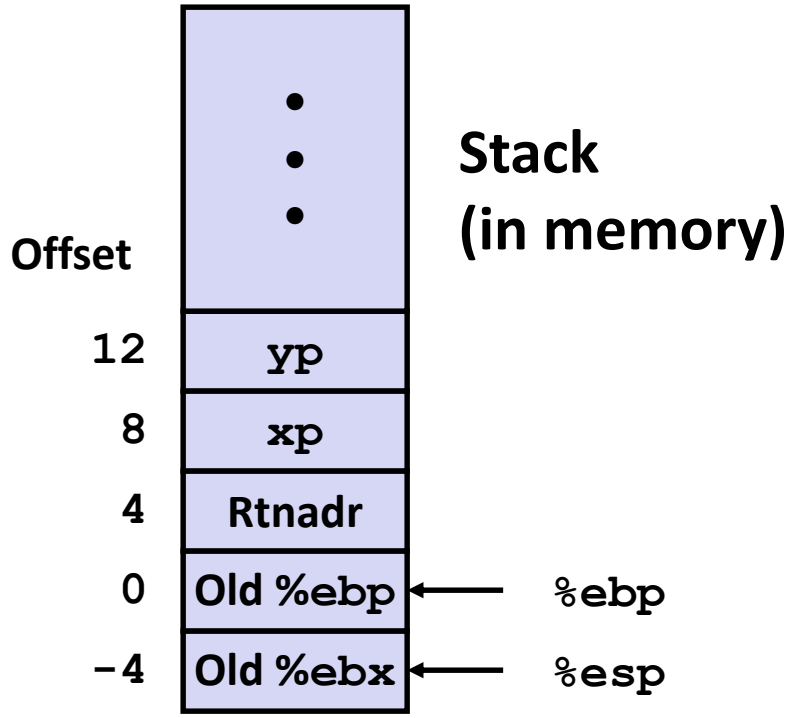
Set Up

Body

Finish

Understanding Swap

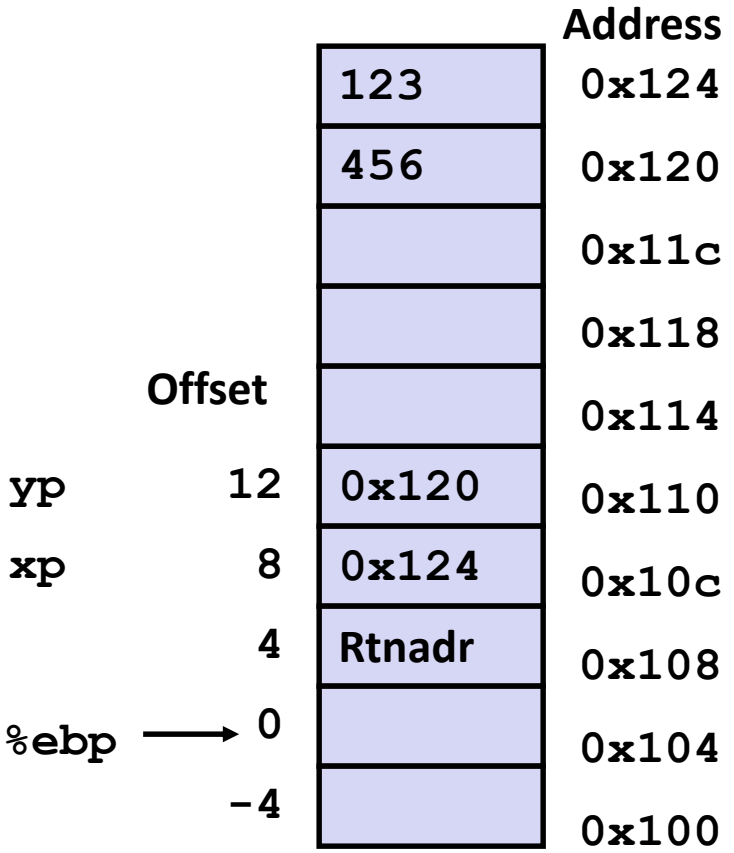
```
void swap(int *xp, int *yp)
{
  int t0 = *xp;
  int t1 = *yp;
  *xp = t1;
  *yp = t0;
}
```



Register	Value
%edx	xp
%ecx	yp
%ebx	t0
%eax	t1

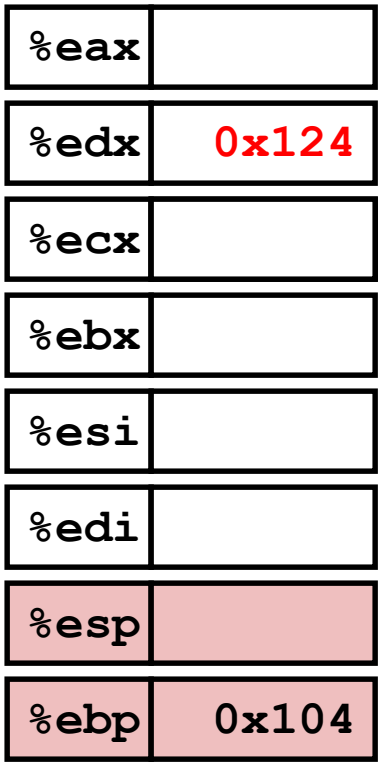
```
movl    8(%ebp), %edx    # edx = xp
movl    12(%ebp), %ecx   # ecx = yp
movl    (%edx), %ebx     # ebx = *xp (t0)
movl    (%ecx), %eax     # eax = *yp (t1)
movl    %eax, (%edx)     # *xp = t1
movl    %ebx, (%ecx)     # *yp = t0
```

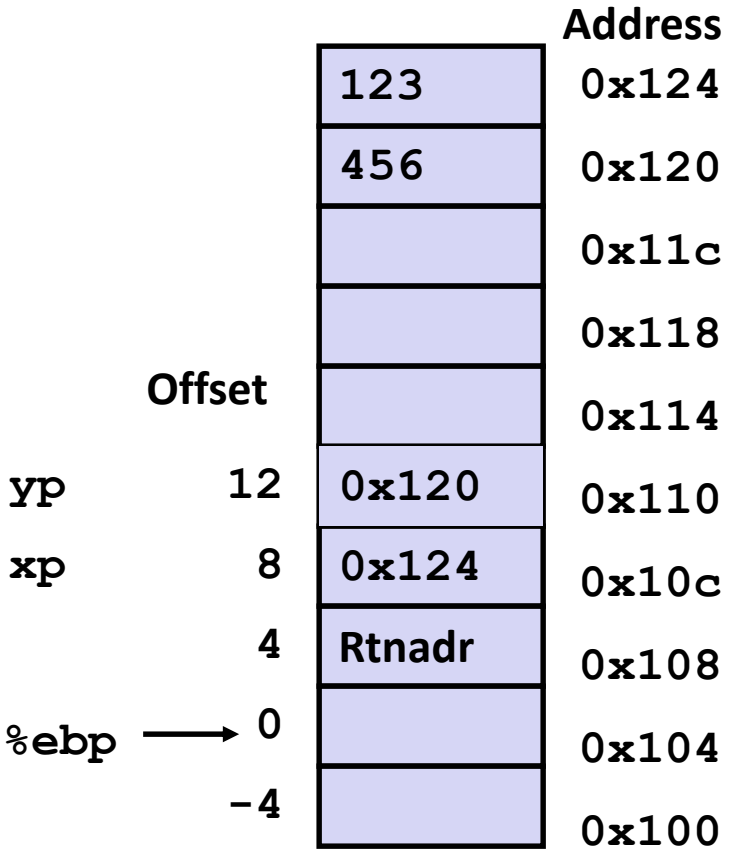
Understanding Swap




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

Understanding Swap

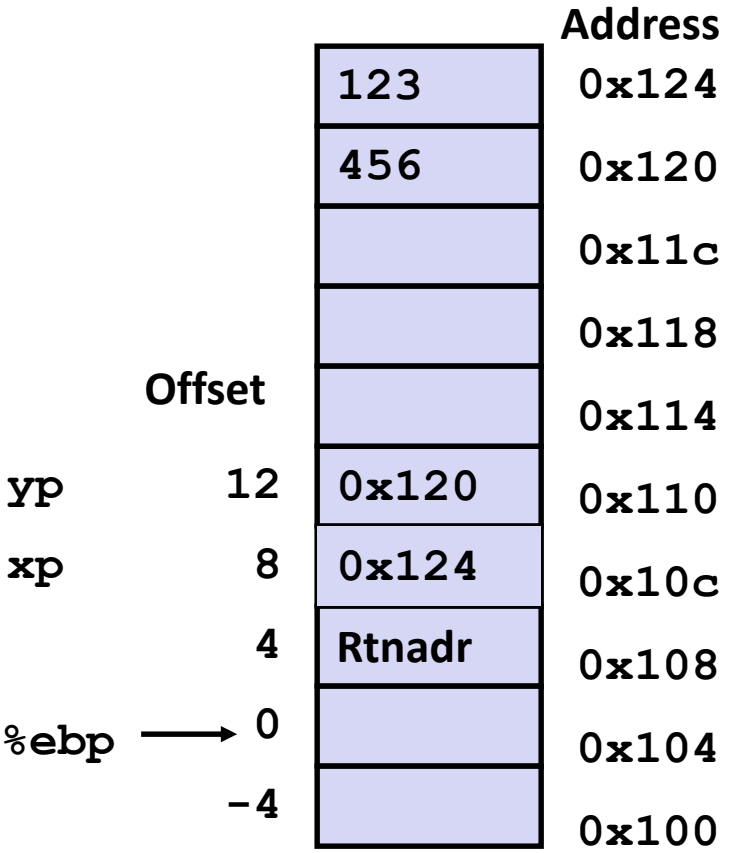





```
movl    8(%ebp), %edx    # edx = xp
movl    12(%ebp), %ecx   # ecx = yp
movl    %edx), %ebx      # ebx = *xp (t0)
movl    %ecx), %eax      # eax = *yp (t1)
movl    %eax, (%edx)     # *xp = t1
movl    %ebx, (%ecx)     # *yp = t0
```

Understanding Swap

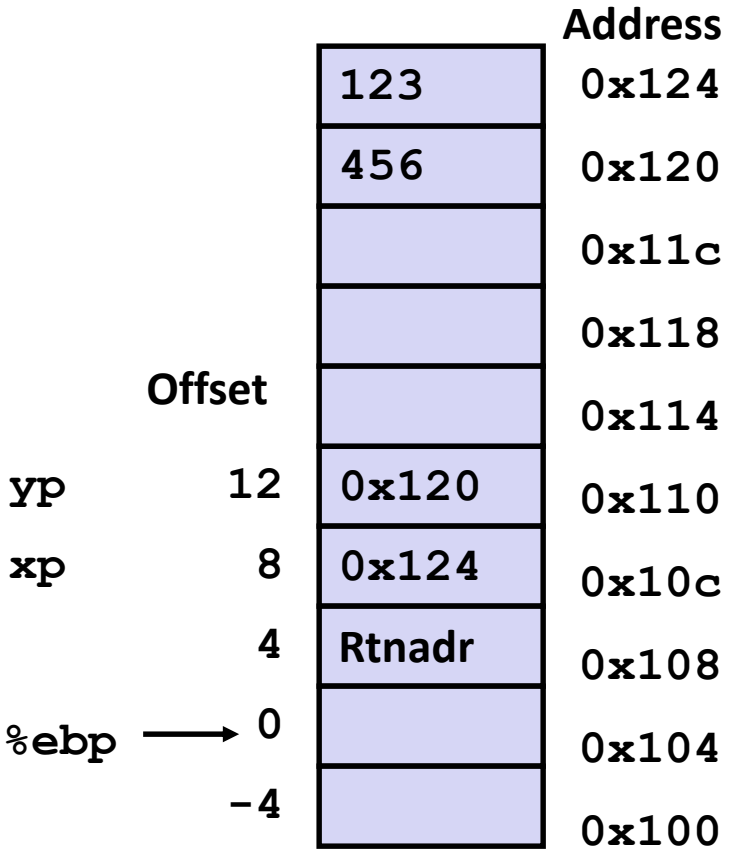
%eax	
%edx	0x124
%ecx	0x120
%ebx	
%esi	
%edi	
%esp	
%ebp	0x104




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

Understanding Swap

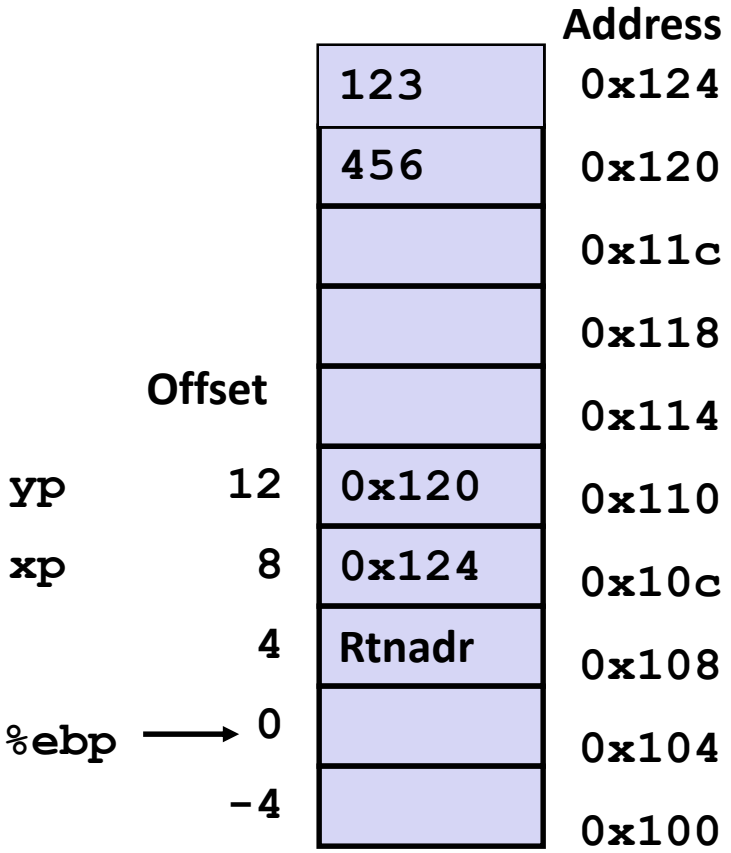
%eax	
%edx	0x124
%ecx	0x120
%ebx	123
%esi	
%edi	
%esp	
%ebp	0x104




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

Understanding Swap

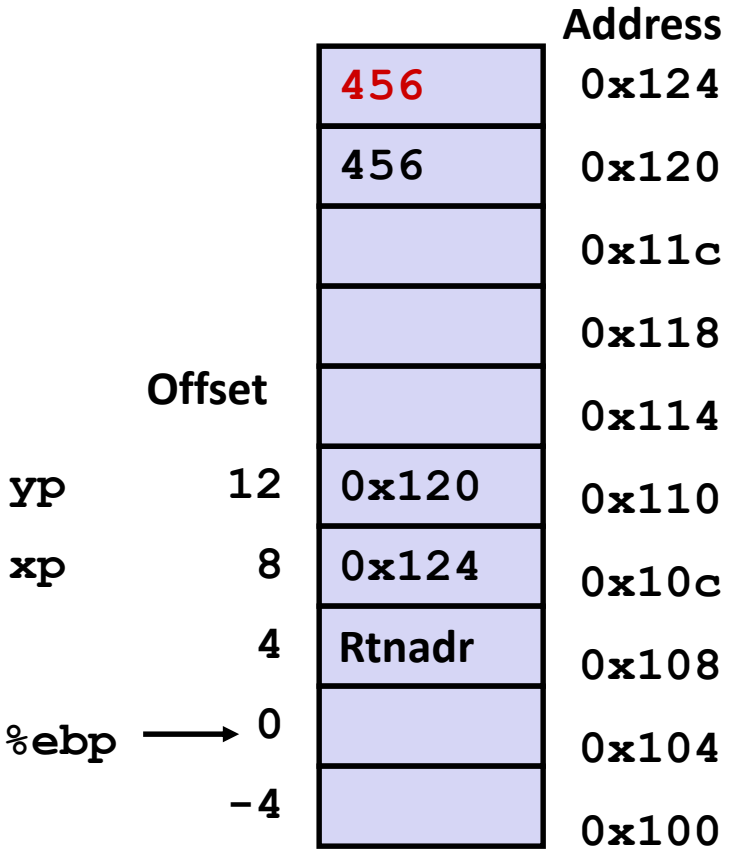
%eax	456
%edx	0x124
%ecx	0x120
%ebx	123
%esi	
%edi	
%esp	
%ebp	0x104




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

Understanding Swap

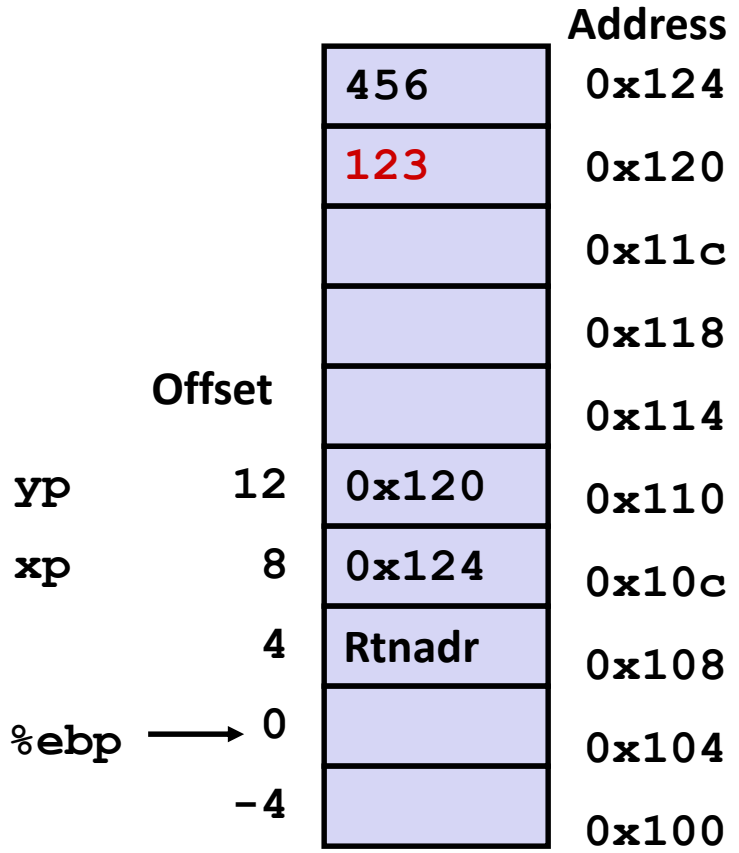
%eax	456
%edx	0x124
%ecx	0x120
%ebx	123
%esi	
%edi	
%esp	
%ebp	0x104




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

Understanding Swap

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




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

Complete Memory Addressing Modes

- **Most General Form**

  D(Rb,Ri,S) **Mem[Reg[Rb]+S*Reg[Ri]+ D]**

  - D: Constant "displacement" 1, 2, or 4 bytes
  - Rb: Base register: Any of 8 integer registers
  - Ri: Index register: Any, except for `%esp`
    - Unlikely you'd use `%ebp`, either
  - S: Scale: 1, 2, 4, or 8 (*why these numbers?*)

- **Special Cases**

  (Rb,Ri) **Mem[Reg[Rb] + Reg[Ri]]**

  D(Rb,Ri) **Mem[Reg[Rb] + Reg[Ri]+D]**

  (Rb,Ri,S) **Mem[Reg[Rb] + S*Reg[Ri]]**

Today: Machine Programming I: Basics

- History of Intel processors and architectures
- C, assembly, machine code
- Assembly Basics: Registers, operands, move
- **Intro to x86-64**

Data Representations: IA32 + x86-64

- **Sizes of C Objects (in Bytes)**

C Data Type	*Generic 32-bit*	*Intel IA32*	*x86-64*
unsigned	4	4	4
int	4	4	4
long int	4	4	8
char	1	1	1
short	2	2	2
float	4	4	4
double	8	8	8
long double	8	10/12	16
char *	4	4	8

  - *Or any other pointer*

x86-64 Integer Registers

%rax	%eax	%r8	%r8d
%rbx	%ebx	%r9	%r9d
%rcx	%ecx	%r10	%r10d
%rdx	%edx	%r11	%r11d
%rsi	%esi	%r12	%r12d
%rdi	%edi	%r13	%r13d
%rsp	%esp	%r14	%r14d
%rbp	%ebp	%r15	%r15d

- Extend existing registers. Add 8 new ones.
- Make **%ebp/%rbp** general purpose

Instructions

- **Long word `l` (4 Bytes) ↔ Quad word `q` (8 Bytes)**

- **New instructions:**
  - **`movl` → `movq`**
  - **`addl` → `addq`**
  - **`sall` → `salq`**
  - etc.

- **32-bit instructions that generate 32-bit results**
  - Set higher order bits of destination register to 0
  - Example: **`addl`**

32-bit code for swap

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

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

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

popl    %ebx
popl    %ebp
  ret
```

Set Up

Body

Finish

64-bit code for swap

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

```
swap:
                                    } Set Up
movl    (%rdi), %edx
movl    (%rsi), %eax
movl    %eax, (%rdi)                } Body
movl    %edx, (%rsi)
ret                                 } Finish
```

- **Operands passed in registers (why useful?)**
  - First (**xp**) in **%rdi**, second (**yp**) in **%rsi**
  - 64-bit pointers
- **No stack operations required**
- **32-bit data**
  - Data held in registers **%eax** and **%edx**
  - **movl** operation

64-bit code for long int swap

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

```
swap_l:

                                  } Set Up
movq    (%rdi), %rdx
movq    (%rsi), %rax              } Body
movq    %rax, (%rdi)
movq    %rdx, (%rsi)

ret                               } Finish
```

- **64-bit data**
  - Data held in registers **%rax** and **%rdx**
  - **movq** operation
    - "q" stands for quad-word

Machine Programming I: Summary

- **History of Intel processors and architectures**
  - Evolutionary design leads to many quirks and artifacts
- **C, assembly, machine code**
  - Compiler must transform statements, expressions, procedures into low-level instruction sequences
- **Assembly Basics: Registers, operands, move**
  - The x86 move instructions cover wide range of data movement forms
- **Intro to x86-64**
  - A major departure from the style of code seen in IA32

Any questions?

THANK YOU!