程序的机器级表示（3）

柴云鹏

2022.10

Data Move

Move Instructions

- Format
  - mov src, dest (-->)
  - src and dest can only be one of the following
    - Immediate
    - Register
    - Memory

Move Instructions

- Format
  - The only possible combinations of the (src, dest) are
    - (immediate, register)
    - (memory, register) load
    - (register, register)
    - (immediate, memory) store
    - (register, memory) store

Data Movement

Instruction	Effect	Description
movl S, D	D ← S	Move double word
movw S, D	D ← S	Move word
movb S, D	D ← S	Move byte
movsbl S, D	D ← SignedExtend(S)	Move sign-extended byte
movzbl S, D	D ← ZeroExtend(S)	Move zero-extended byte
pushl S	R[%esp] ← R[%esp]-4 M[R[%esp]] ← S	Push
popl D	D ← M[R[%esp]] R[%esp] ← R[%esp]+4	Pop

Data Movement Example

movl	$0x4050, %eax	immediate	register
movl	%ebp, %esp	register	register
movl	(%edx, %ecx), %eax	memory	register
movl	$-17, (%esp)	immediate	memory
movl	%eax, -12(%ebp)	register	memory

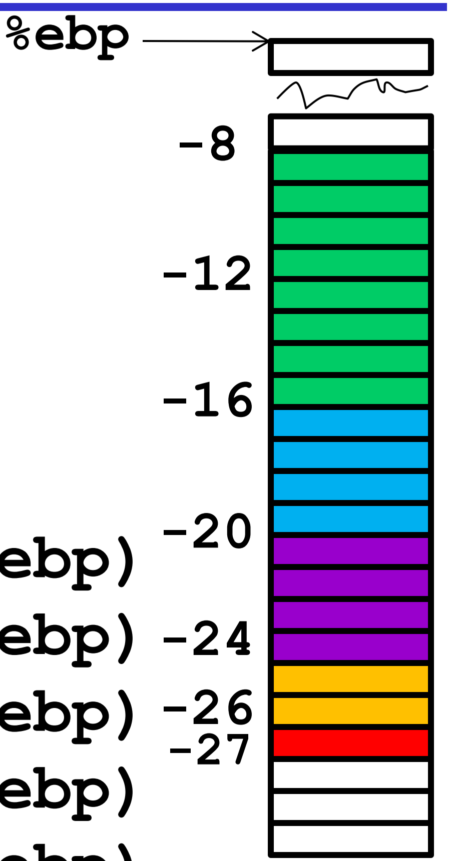
Data Formats

- Move data instruction
  - mov (general)
  - movb (move byte)
  - movw (move word)
  - movl (move double word)
  - movq (move quadruple word)

Access Objects with Different Sizes

```
int main(void){
   char c = 1;   short s = 2;
   int i = 4;    long l = 4L;
   long long ll = 8LL;
   return;
}
```

```
8048335:c6 movb   $0x1,0xffffffe5(%ebp)
8048339:66 movw   $0x2,0xffffffe6(%ebp)
804833f:c7 movl   $0x4,0xffffffe8(%ebp)
8048346:c7 movl   $0x4,0xffffffec(%ebp)
804834d:c7 movl   $0x8,0xfffffff0(%ebp)
8048354:c7 movl   $0x0,0xfffffff4(%ebp)
```



Array in Assembly

Persistent usage

- Store the base address

```
void f(void){
   int i, a[16];
   for(i=0; i<16; i++)
     a[i]=i;
}
movl %edx,-0x44(%ebp,%edx,4)

a:   -0x44(%ebp)
i:   %edx
```

Data Movement Example

```
Initial value %dh=8d           %eax =98765432

1   movb      %dh, %al         %eax=9876548d
2   movsbl    %dh, %eax        %eax=ffffff8d
3   movzbl    %dh, %eax        %eax=0000008d
```

- 1-byte registers
  - %al, %ah, %cl, %ch, %dl, %dh, %bl, %bh
- 2-byte registers
  - %ax, %cx, %dx, %bx, %si, %di, %sp, %bp

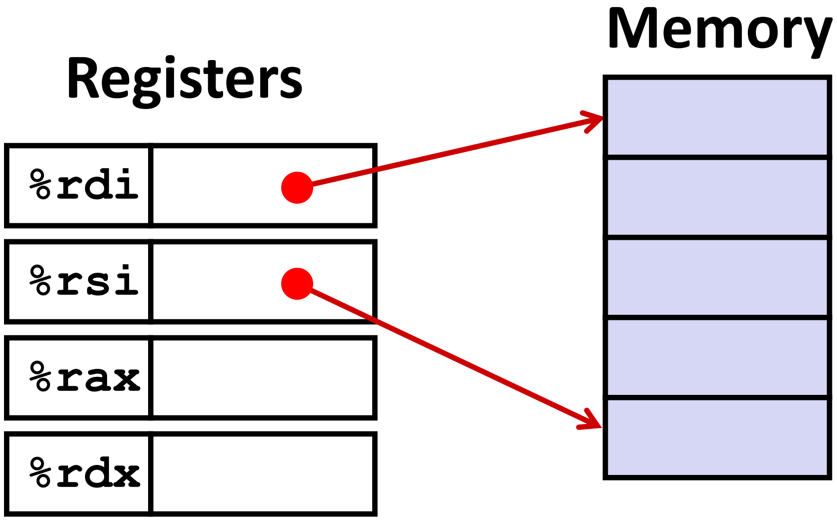
Example of Simple Addressing Modes

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

```
swap:
   movq     (%rdi), %rax
   movq     (%rsi), %rdx
   movq     %rdx, (%rdi)
   movq     %rax, (%rsi)
   ret
```

Understanding Swap()

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



Register	Value
%rdi	xp
%rsi	yp
%rax	t0
%rdx	t1

```
swap:
   movq    (%rdi), %rax   # t0 = *xp
   movq    (%rsi), %rdx   # t1 = *yp
   movq    %rdx, (%rdi)   # *xp = t1
   movq    %rax, (%rsi)   # *yp = t0
   ret
```

Understanding Swap()

Registers

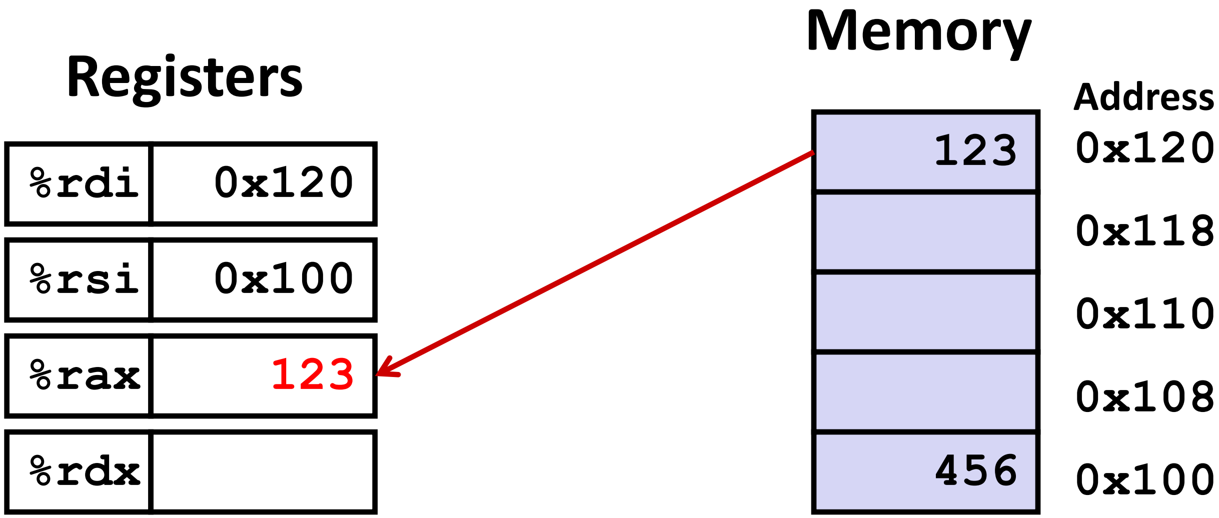
%rdi	0x120
%rsi	0x100
%rax	
%rdx	

Memory

	Address
123	0x120
	0x118
	0x110
	0x108
456	0x100

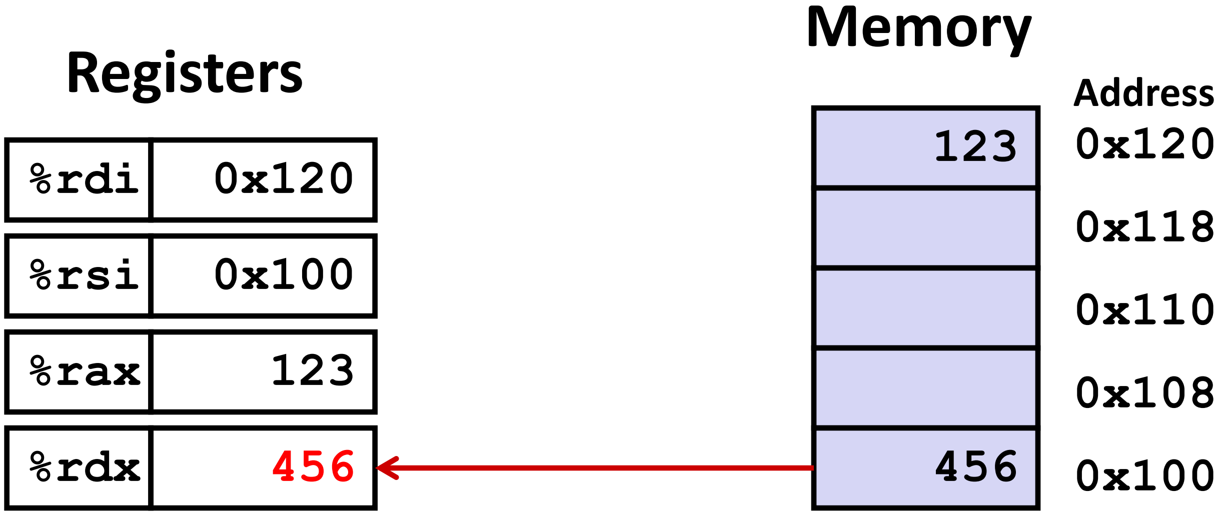
```
swap:
   movq    (%rdi), %rax   # t0 = *xp
   movq    (%rsi), %rdx   # t1 = *yp
   movq    %rdx, (%rdi)   # *xp = t1
   movq    %rax, (%rsi)   # *yp = t0
   ret
```

Understanding Swap()



```
swap:
   movq     (%rdi), %rax   # t0 = *xp
   movq     (%rsi), %rdx   # t1 = *yp
   movq     %rdx, (%rdi)   # *xp = t1
   movq     %rax, (%rsi)   # *yp = t0
   ret
```

Understanding Swap()



Registers

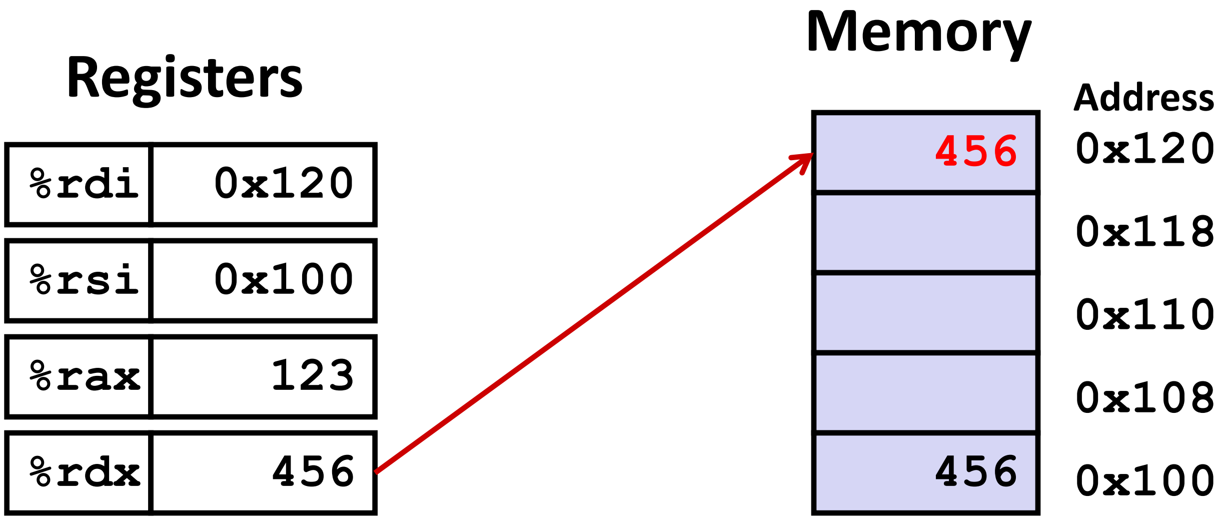
%rdi	0x120
%rsi	0x100
%rax	123
%rdx	456

Memory

	Address
123	0x120
	0x118
	0x110
	0x108
456	0x100

```
swap:
   movq     (%rdi), %rax   # t0 = *xp
   movq     (%rsi), %rdx   # t1 = *yp
   movq     %rdx, (%rdi)   # *xp = t1
   movq     %rax, (%rsi)   # *yp = t0
   ret
```

Understanding Swap()



Registers

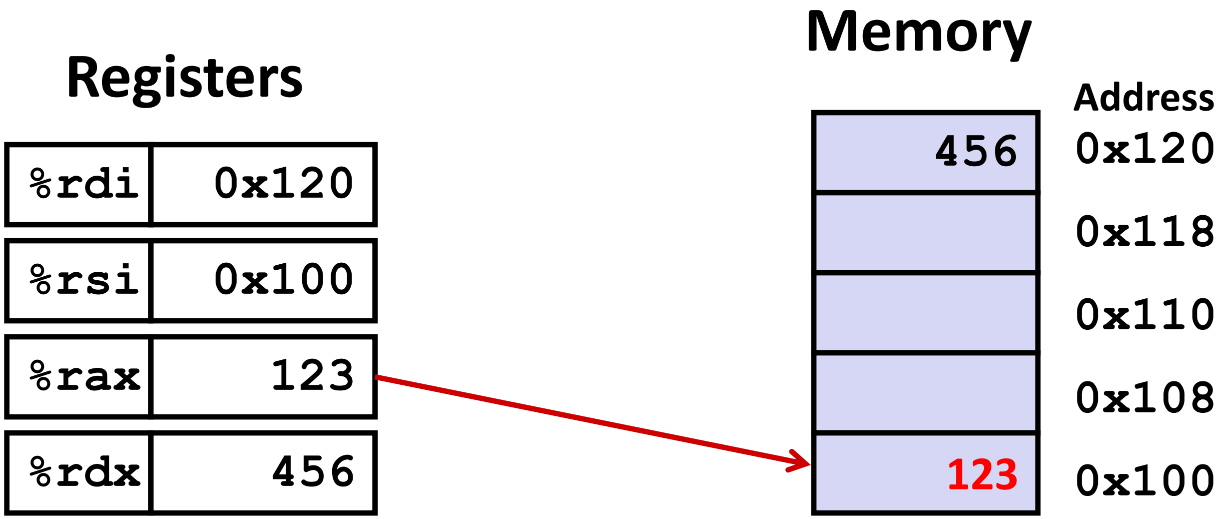
%rdi	0x120
%rsi	0x100
%rax	123
%rdx	456

Memory

	Address
456	0x120
	0x118
	0x110
	0x108
456	0x100

```
swap:
   movq     (%rdi), %rax   # t0 = *xp
   movq     (%rsi), %rdx   # t1 = *yp
   movq     %rdx, (%rdi)   # *xp = t1
   movq     %rax, (%rsi)   # *yp = t0
   ret
```

Understanding Swap()



```
swap:
   movq    (%rdi), %rax   # t0 = *xp
   movq    (%rsi), %rdx   # t1 = *yp
   movq    %rdx, (%rdi)   # *xp = t1
   movq    %rax, (%rsi)   # *yp = t0
   ret
```

MOV 指令遵循的规则

- 两个操作数的尺寸必须一致
- 两个操作数不能同时为内存操作数
- 目的操作数不能是CS，EIP和IP
- 立即数不能直接送至段寄存器
- 段寄存器之间不能直接传送数据
- 不能一次传送多个对象
- 标识寄存器（EFlags）和指令指针寄存器（IP）不能用MOV指令设置

课堂练习

- 下列汇编语句错在哪里？
  - movb $0xF, (%bl)
  - movl %ax, (%esp)
  - movw (%eax), 4(%esp)
  - movl %eax, %dx
  - movb %si, 8(%esp)
  - movb %ah, %sh
  - movl %eax, 0x123

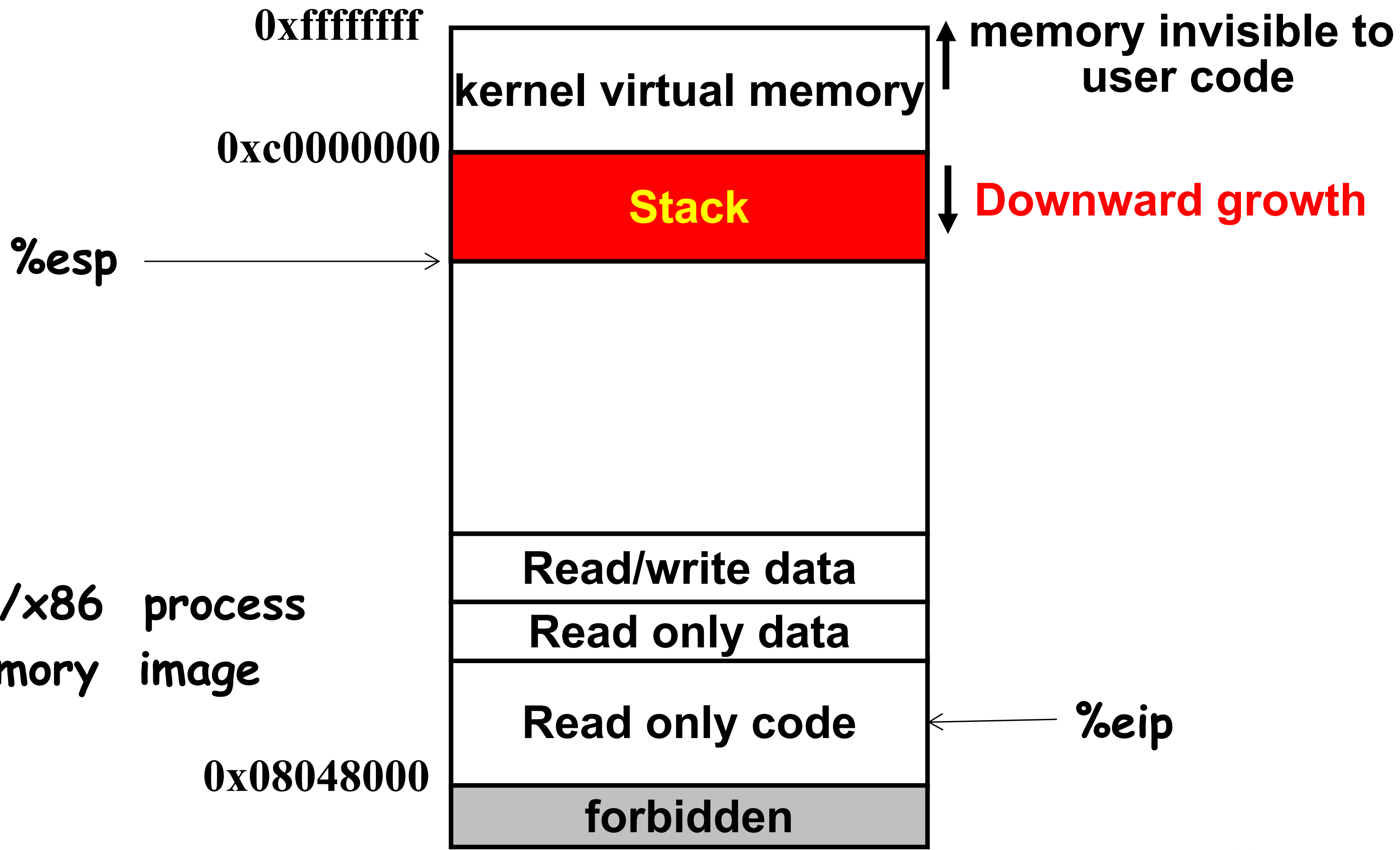
课堂练习

- 已知原型为
- Void decode1 (long *xp, long *yp, long *zp);
- 的函数编译成汇编，得到如下代码
- # xp in %rdi, yp in %rsi, zp in %rdx
- Decode1:
  - Movq (%rdi), %r8
  - Movq (%rsi), %rcx
  - Movq (%rdx), %rax
  - Movq %r8, (%rsi)
  - Movq %rcx, (%rdx)
  - Movq %rax, (%rdi)
- 请写出等效的C语言代码

Stack operation

- Stack is a special kind of data structure
  - It can store objects of the same type
- The top of the stack must be explicitly specified
  - It is denoted as top
- There are two operations on the stack
  - push and pop
- There is a hardware stack in x86
  - 速度快，通过push, pop等指令操作
  - its bottom has high address number
  - its top is indicated by %esp

Stack Layout



Linux/x86 process memory image

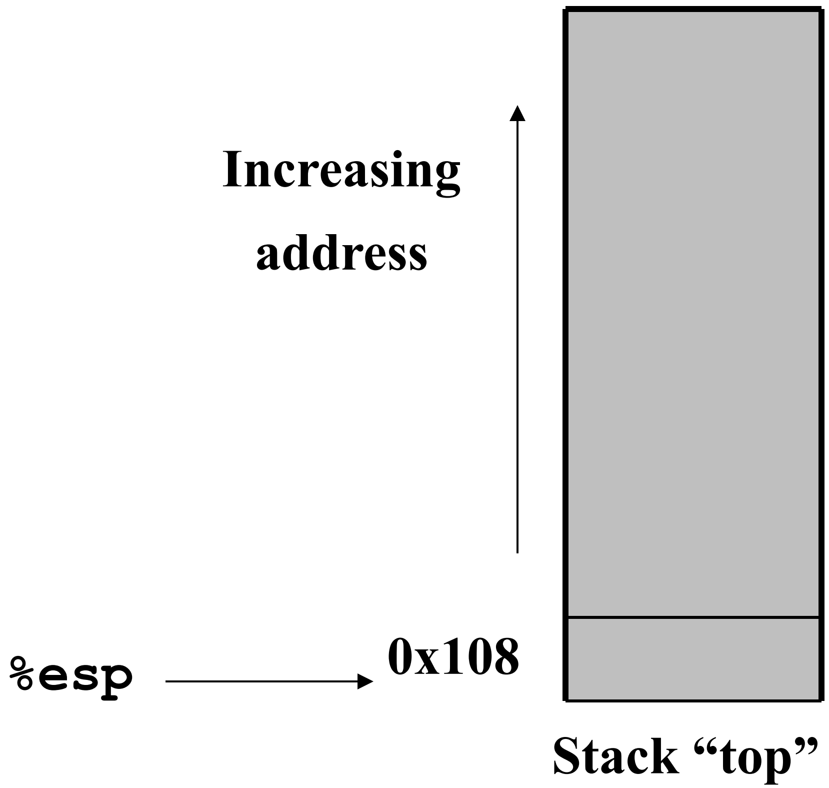
Stack operation

- There are two stack operation instructions
  - Push and Pop
- Push
  - decreases the %esp (enlarge the stack)
  - stores the value in a register into the stack
- Pop
  - stores the value in the top of the stack into a register
  - increases the %esp (shrink the stack)

Stack operations

%eax	0x123
%edx	0
%esp	0x108

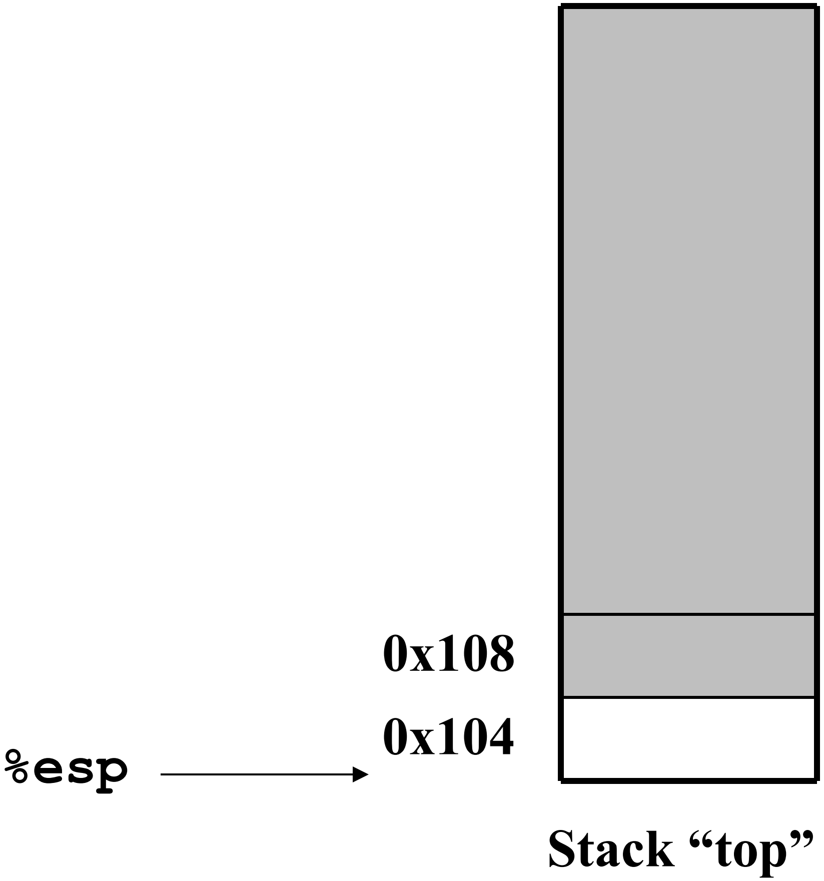
pushl %eax ?



Stack operations

%eax	0x123
%edx	0
%esp	0x104

pushl %eax

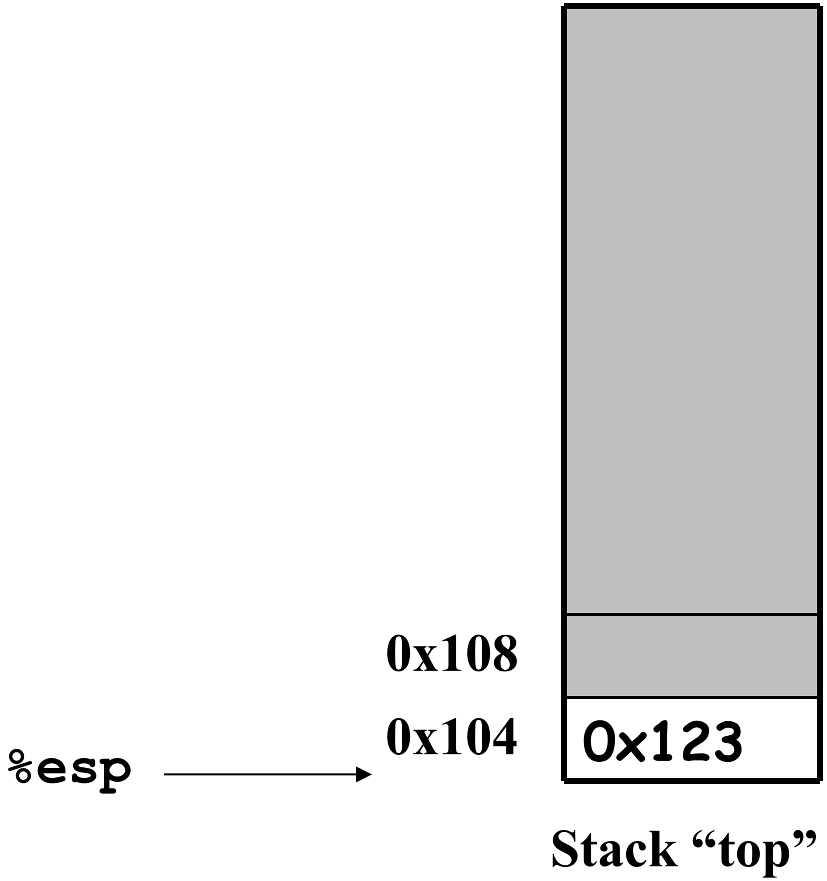


Stack operations

%eax	0x123
%edx	0
%esp	0x104

pushl %eax

popl %edx ?

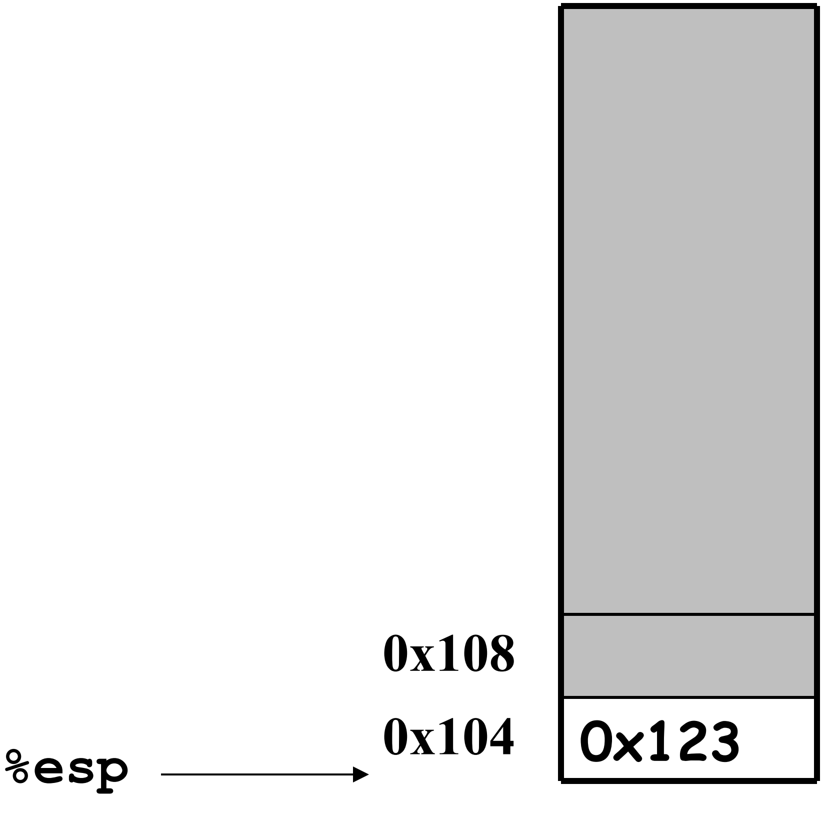


Stack operations

%eax	0x123
%edx	0x123
%esp	0x104

pushl %eax

popl %edx ?



0x108

0x104

0x123

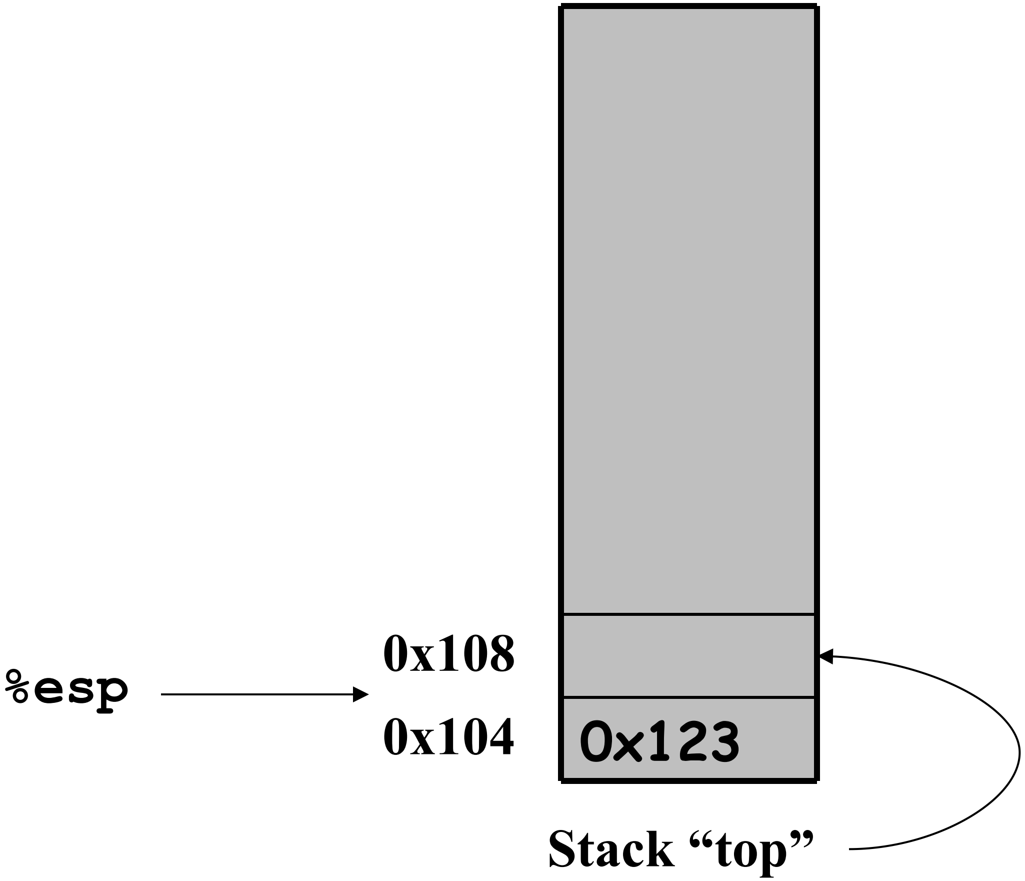
%esp

Stack “top”

Stack operations

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

popl %edx



%esp

0x108

0x104

0x123

Stack “top”

Stack Operation

Instruction	Effect	Description
pushl S	R[%esp] ← R[%esp]-4 M[R[%esp]] ← S	Push
popl D	D ← M[R[%esp]] R[%esp] ← R[%esp]+4	Pop

Pointers vs. Addresses

Outline

- Pointer in C
- Data Movement Example

Pointers in C

- In C, an object may be
  - an integer
  - a structure or
  - some other program unit
- Declaring Pointers

```
T *p;
```

Pointers in C

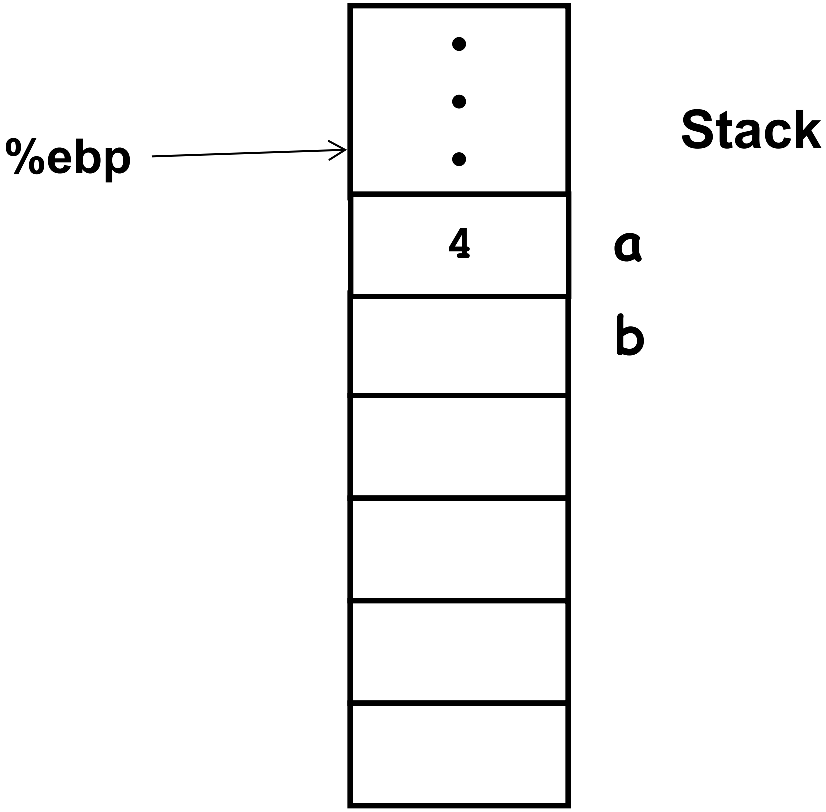
- In C, the value of a pointer in C is
  - the virtual address of the first byte of some block of storage
  - Type information is used to
    - identify the length of the object
    - interpret the object *p
- Generating Pointers

```
p = &obj
```

Example

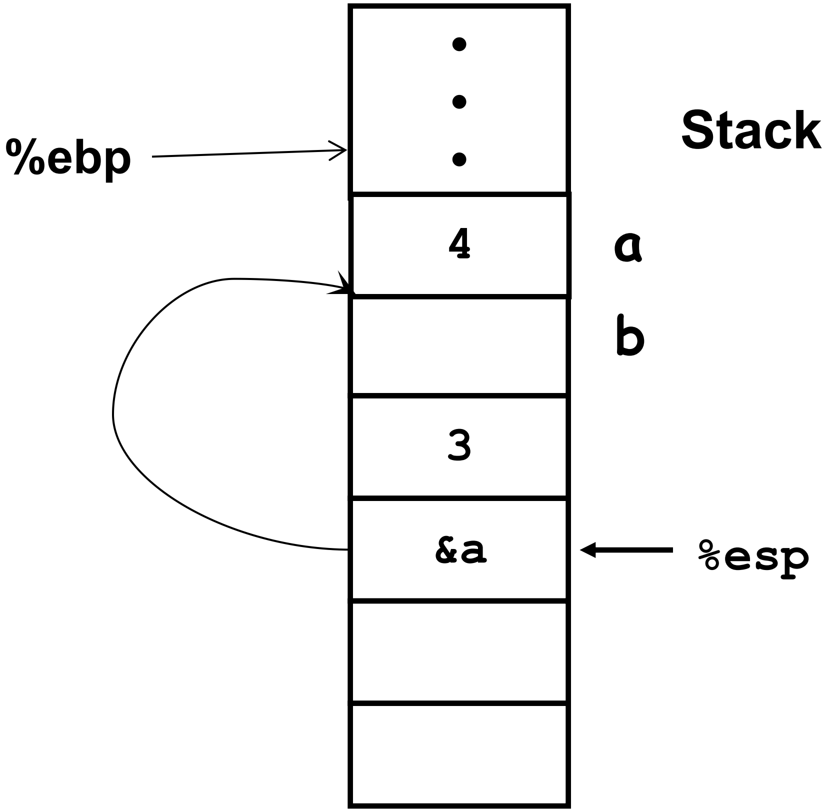
```
int main()
{
    int a = 4 ;
    /* “address of” operator creates a pointer */
    int b = exchange(&a, 3);
    printf(“a = %d, b = %d\n”, a, b);
}
```

Example



Can the variable a be in a register?

Example



Example

```
int exchange(int *xp, int y)

{
     /* operator * performs deferencing */
    int x = *xp ;

    *xp = y ;

    return x ;

}
```

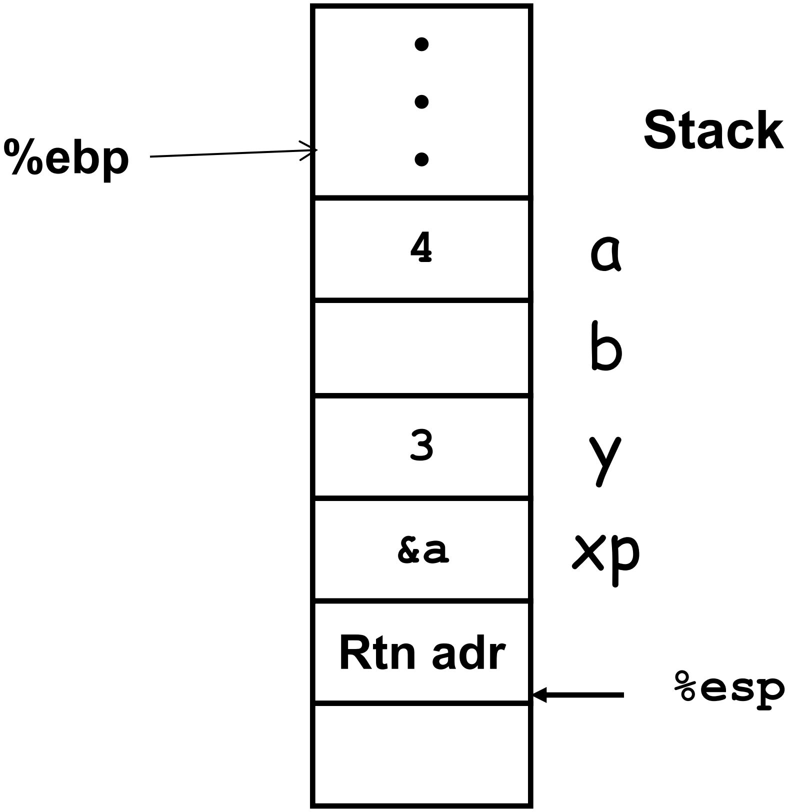
Example

```
int exchange(int *xp, int y)
{
    int x = *xp ;

    *xp = y ;
    return x ;
}
```

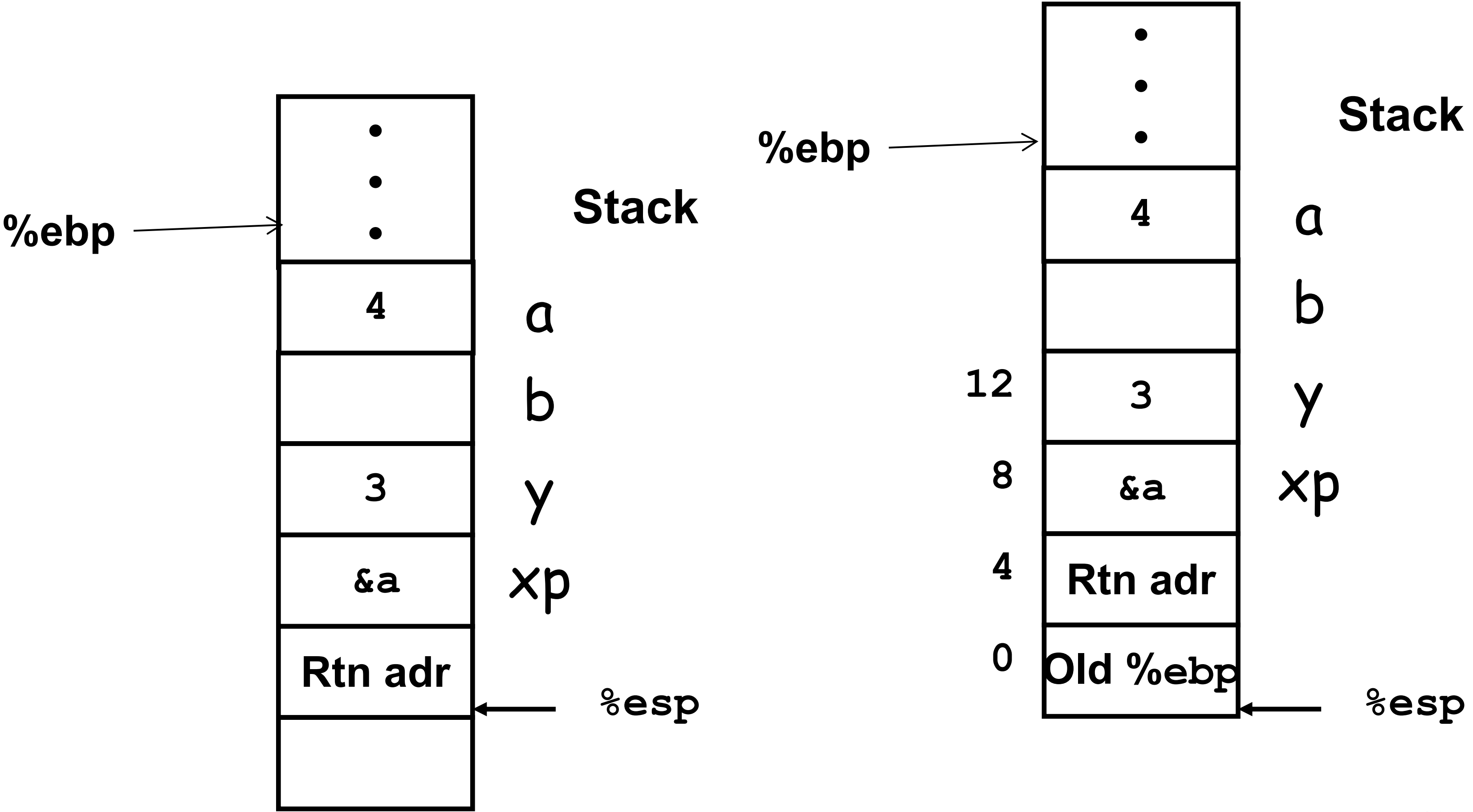
```
pushl  %ebp
movl   %esp, %ebp
movl   8(%ebp),  %eax #xp
movl   12(%ebp), %edx #y
movl   (%eax), %ecx # x
movl   %edx,    (%eax)
movl   %ecx,   %eax
movl   %ebp, %esp
popl   %ebp
```

Example



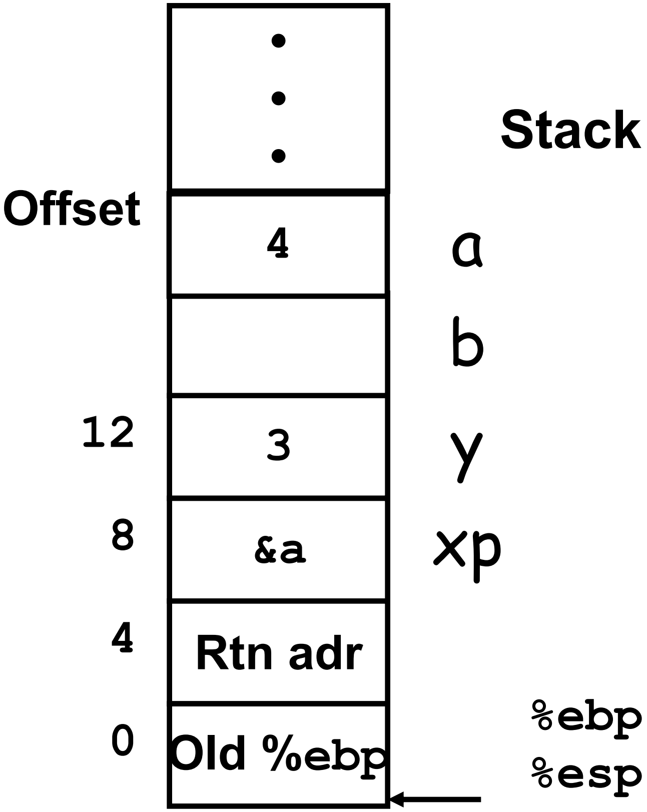
Example

1 pushl %ebp



Example

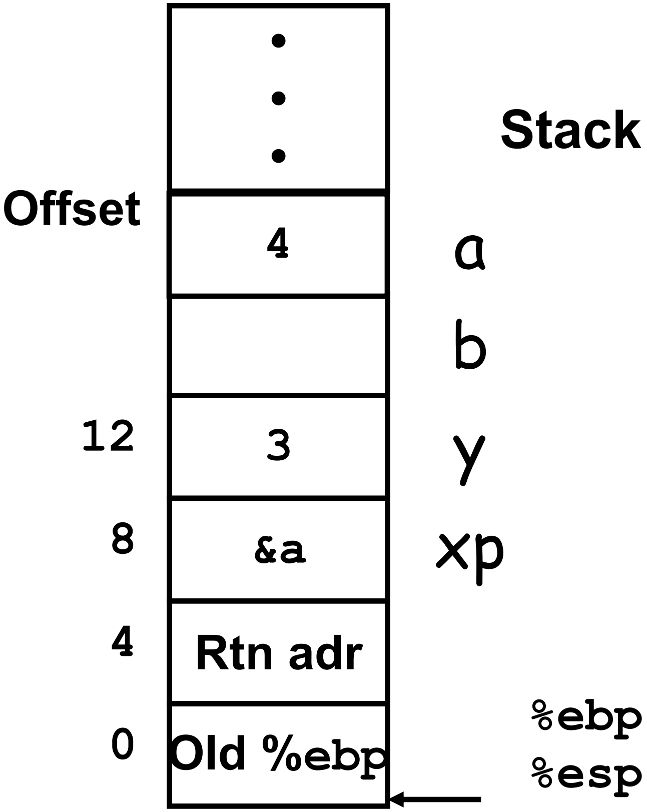
2 movl %esp, %ebp



Example

3 movl 8(%ebp), %eax

%eax: xp

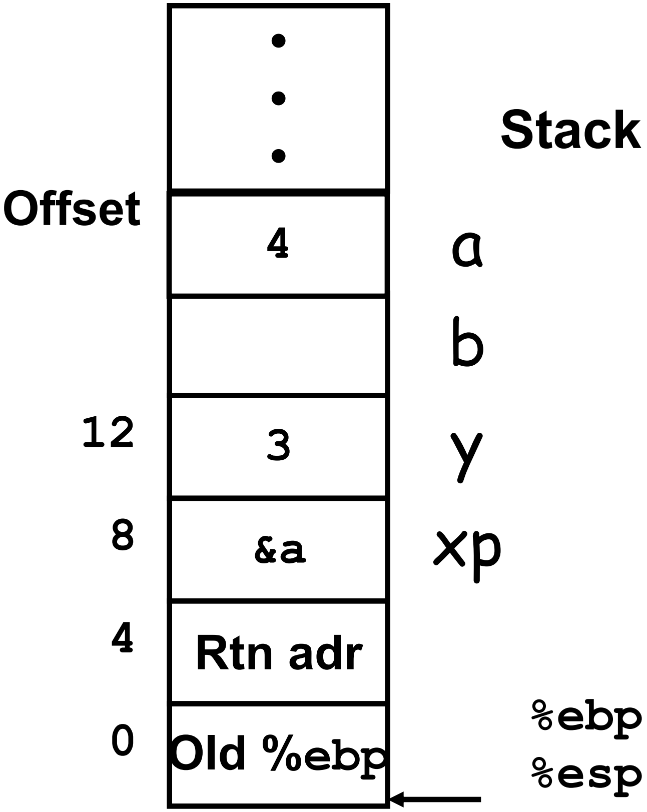


Example

```
3 movl    8(%ebp),  %eax
4 movl    12(%ebp), %edx
```

%eax: xp

%edx: 3



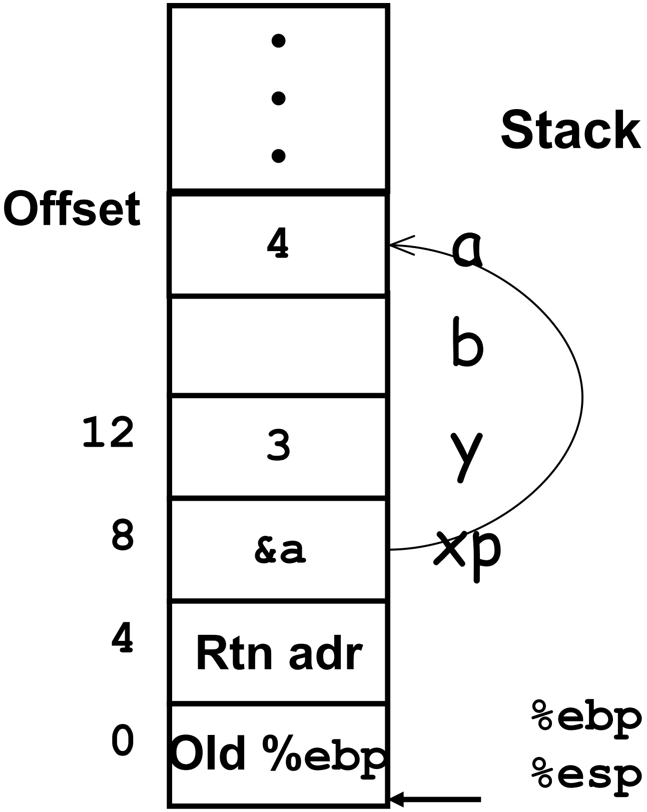
Example

```
3 movl    8(%ebp),  %eax
4 movl   12(%ebp), %edx
5 movl    (%eax),    %ecx
```

%eax: xp

%edx: 3

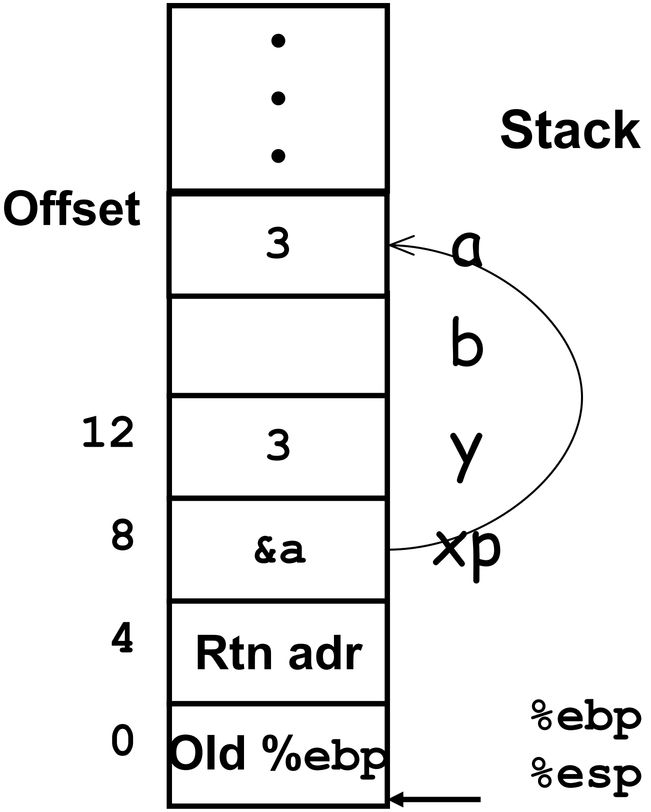
%ecx: 4



Example

```
3 movl   8(%ebp),  %eax
4 movl   12(%ebp), %edx
5 movl   (%eax),   %ecx
6 movl   %edx,     (%eax)
```

%eax: xp
%edx: 3
%ecx: 4
*xp(a): 3



Data Movement Example

```
3 movl    8(%ebp),  %eax
4 movl    12(%ebp), %edx
5 movl    (%eax),   %ecx
6 movl    %edx,     (%eax)
7 movl    %ecx,     %eax
```

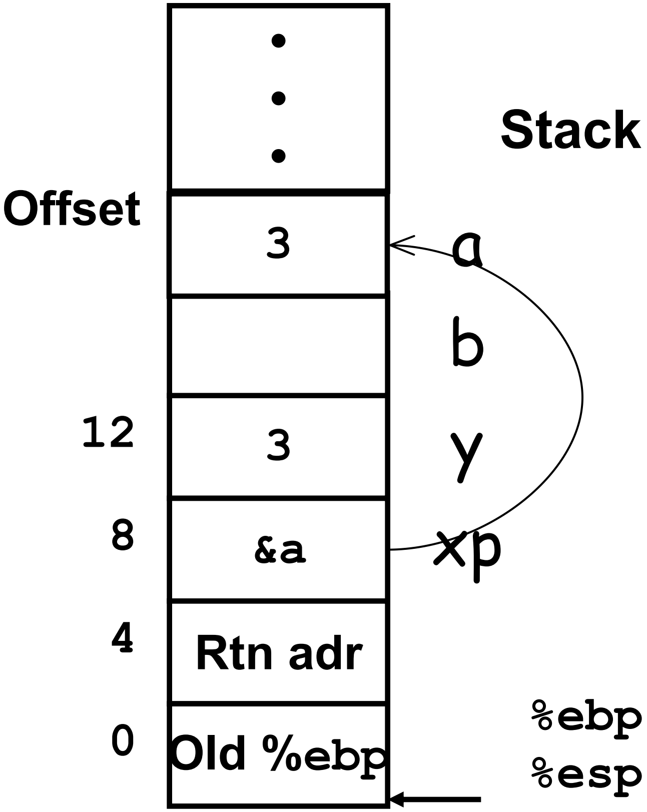
%eax: xp

%edx: y

%ecx: 4

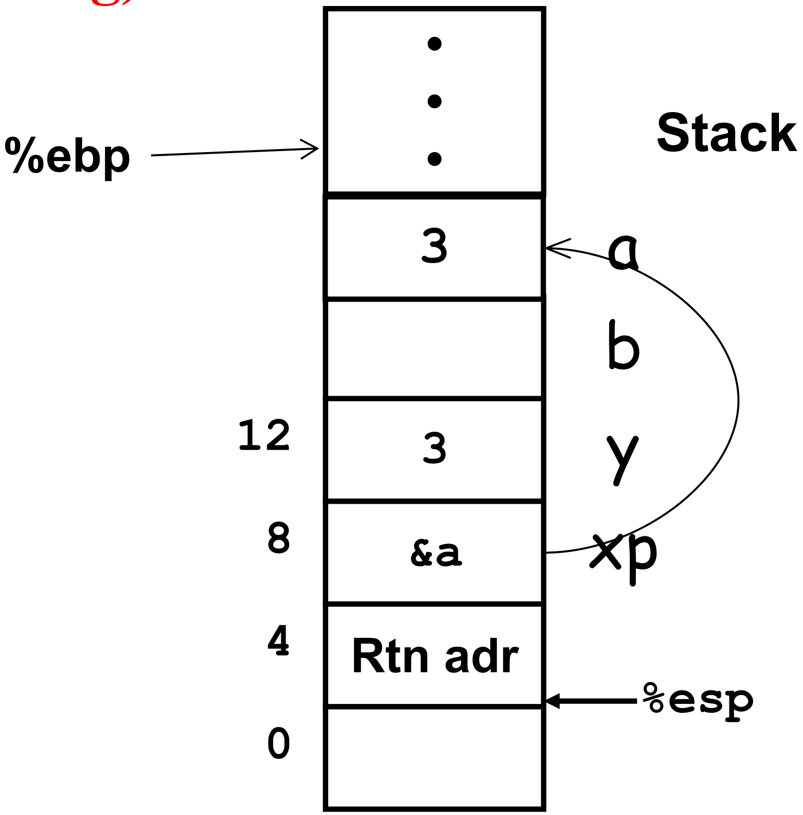
*xp(a): 3

%eax: 4 (old *xp) return value



Data Movement Example

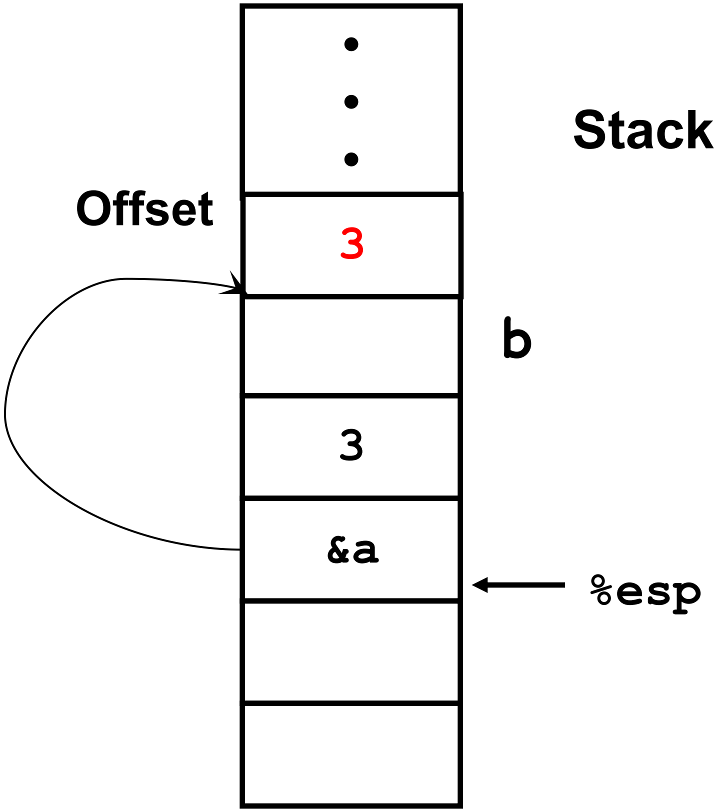
```
8 movl    %ebp, %esp (do nothing)
9 popl    %ebp
```



Example

```
int main()
{
    int a = 4 ;
    /* “address of” operator creates a local pointer
    variable */
    int b = exchange(&a, 3);
    printf(“a = %d, b = %d\n”, a, b);
}
```

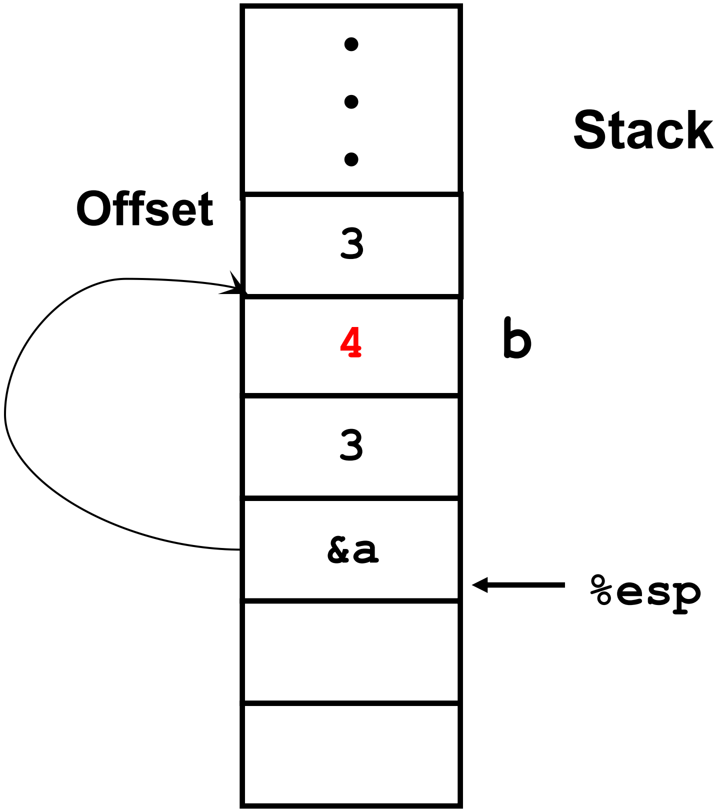
Example



%eax 4

Return value

Example



Data Manipulation

Arithmetic and Logical Operations

Instruction	Effect	Description
leal S, D	D ← &S	Load effective address
incl D	D ← D + 1	Increment
decl D	D ← D – 1	Decrement
negl D	D ← -D	Negate
notl D	D ← ~D	Complement
addl S, D	D ← D + S	Add
subl S, D	D ← D – S	Subtract
cmpl S, D	D – S	Subtract
imull S, D	D ← D * S	Multiply

Arithmetic and Logical Operations

Instruction	Effect	Description
xorl S, D	D ← D ^ S	Exclusive-or
orl S, D	D ← D | S	Or
andl S, D	D ← D & S	And
testl S, D	D & S	And
sall k, D	D ← D << k	Left shift, k为8 bit
shll k, D	D ← D << k	Left shift
sarl k, D	D ← D >> k	Arithmetic right shift
shrl k, D	D ← D >> k	Logical right shift

Arithmetic and Logical Operations

Address	Value
0x100	0xFF
0x104	0xAB
0x108	0x13
0x10C	0x11

Register	Value
%eax	0x100
%ecx	0x1
%edx	0x3

Instruction	Destination	Value
addl %ecx, (%eax)	0x100	0x100
subl %edx, 4(%eax)	0x104	0xA8
imull $16, (%eax, %edx, 4)	0x10C	0x110
incl 8(%eax)	0x108	0x14
decl %ecx	%ecx	0x0
subl %edx, %eax	%eax	0xFD

Examples for Lea Instruction

%eax holds x, %ecx holds y

Expression	Result
leal 6(%eax), %edx	6+x
leal (%eax, %ecx), %edx	x+y
leal (%eax, %ecx, 4), %edx	x+4*y
leal 7(%eax, %eax, 8), %edx	7+9*x
leal 0xA(, %ecx, 4), %edx	10+4*y
leal 9(%eax, %ecx, 2), %edx	9+x+2*y

Assembly Code for Arithmetic Expressions

```
int arith(int x, int y, int z)
{
  int t1 = x+y;
  int t2 = z*48;
  int t3 = t1&0xFFFF;
  int t4 = t2*t3;
  return t4;
}
```

```
movl 12(%ebp),%eax        Get y
movl 16(%ebp),%edx        Get z
addl 8(%ebp),%eax         Compute t1=x+y
leal (%edx,%edx,2),%edx   Compute 3*z
sall $4,%edx              Compute t2=48*z
andl $0xFFFF,%eax         Compute t3=t1&FFFF
imull %edx,%eax           Set t4 as return val
```

Special Arithmetic Operations

imull S	R[%edx]:R[%eax] ←S*R[%eax]	Signed full multiply
mull S	R[%edx]:R[%eax] ←S*R[%eax]	Unsigned full multiply
Cltd	R[%edx]:R[%eax] ← SignExtend(R[%eax])	Convert to quad word
idiv S	R[%edx] ← R[%edx]:R[%eax] mod S R[%eax] ← R[%edx]:R[%eax] ÷ S	Signed divide
divl S	R[%edx] ←R[%edx]:R[%eax] mod S R[%eax] ← R[%edx]:R[%eax] ÷ S	Unsigned divide

Examples

Initially x at %ebp+8, y at %ebp+12

```
movl 8(%ebp), %eax
imull 12(%ebp)
pushl %edx   #高位
pushl %eax   #低位
```

```
movl 8(%ebp), %eax
cltd
idivl 12(%ebp)
pushl %eax  #商
pushl %edx  #余数
```