Assembly Programing with NASM (and GAS)

DR. ARKA PROKASH MAZUMDAR

NASM Data Directives

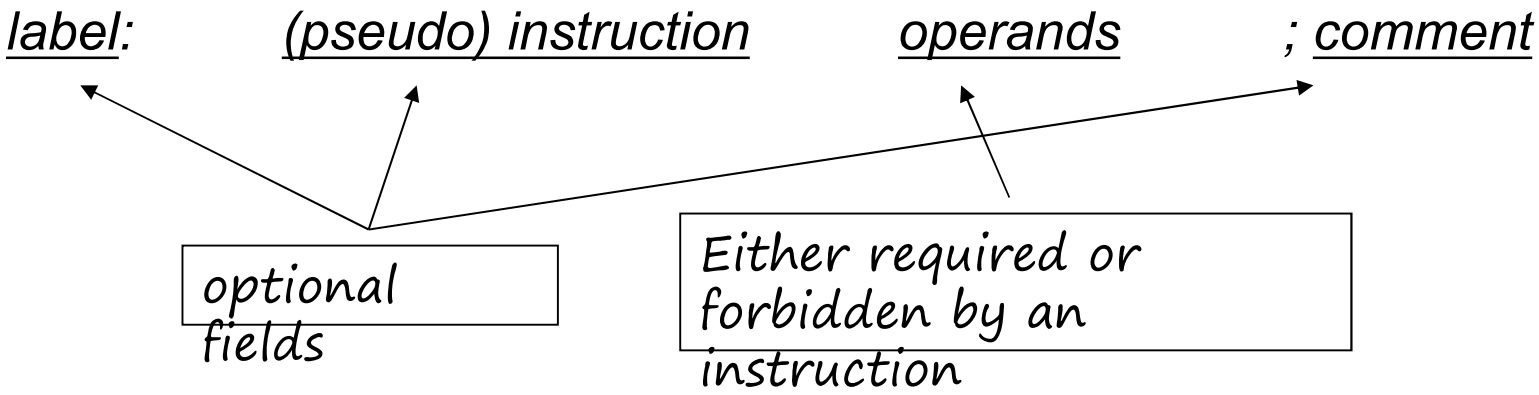
```
L1      db              0               ; byte
L2      dw              1000            ; word
L3      db              110101b         ; byte
L4      db              12h                 ; byte
L5      db              17o                 ; byte
L6      dd              1A92h           ; double word
L7      resb            1               ; uninitialized byte
L8      db              'A'             ; ascii code = 'A'
L9      db              0,1,2,3             ; 4 bytes
L10     db              'w','o','r','d',0 ; string
L11     db              'word', 0
L12     times 100 db    0               ; 100 bytes of zero
L13     resw            100                 ; 100*2(word bytes)
```

Data Types in IA32 (AT&T format)

C declaration	Intel data type	Assembly code suffix	Size (by
`char`	Byte	`b`	1
`short`	Word	`w`	2
`int`	Double word	`l`	4
`long int`	Double word	`l`	4
`long long int`	—	—	4
`char *`	Double word	`l`	4
`float`	Single precision	`s`	4
`double`	Double precision	`l`	8
`long double`	Extended precision	`t`	10/1

Instruction Basics

Each NASM standard source line contains a combination of the 4 fields:



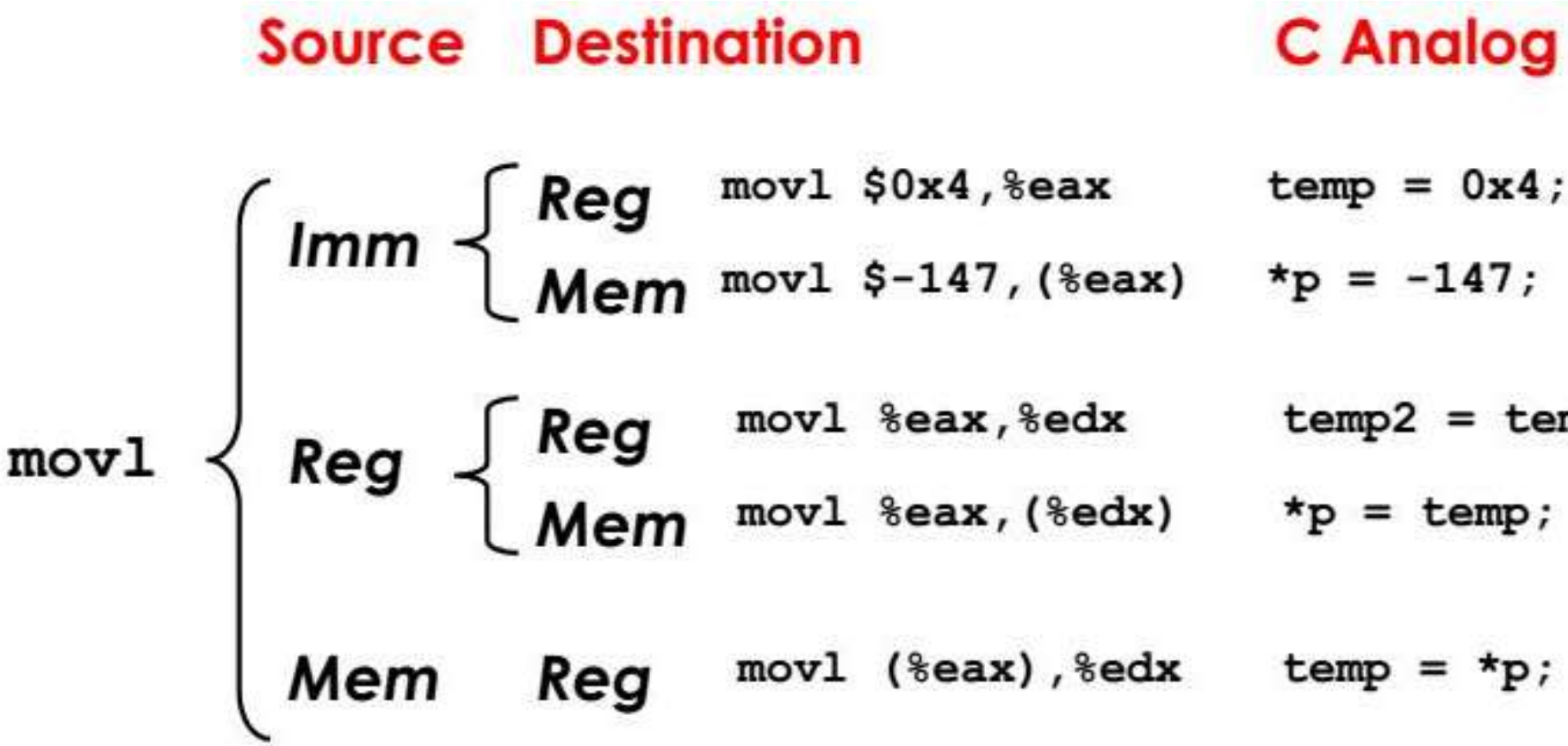
1. **Backslash** (\) uses as the line continuation character: if a line ends with backslash, the next line is considered to be a part of the backslash-ended line.
2. **No restrictions on white space** within a line.
3. A colon after a label is optional.

Example Program

```
1   int simple(int *xp, int y)
2   {
3     int t = *xp + y;
4     *xp = t;
5     return t;
6   }
```

```
1   simple:
2     pushl   %ebp              Save frame
3     movl    %esp, %ebp        Create new
4     movl    8(%ebp), %edx     Retrieve x
5     movl    12(%ebp), %eax    Retrieve y
6     addl    (%edx), %eax      Add *xp to
7     movl    %eax, (%edx)      Store t at
8     popl    %ebp              Restore fr
9     ret                       Return
```

Operand Modes



NASM Data Directives

```
L1      db              0               ; byte
L2      dw              1000            ; word
L3      db              110101b         ; byte
L4      db              12h                 ; byte
L5      db              17o                 ; byte
L6      dd              1A92h           ; double word
L7      resb            1               ; uninitialized byte
L8      db              'A'             ; ascii code = 'A'
L9      db              0,1,2,3             ; 4 bytes
L10     db              'w','o','r','d',0 ; string
L11     db              'word', 0
L12     times 100 db    0               ; 100 bytes of zero
L13     resw            100                 ; 100*2(word bytes)
```

Examples

Data directives (different to MASM)

- `Mov al, [L1]` ; copy byte at L1
- `Mov eax, L1` ; eax = address of byte at L1
- `Mov [L1], ah` ; copy ah into byte at L1
- `Mov eax, [L6]` ; copy double word
- `Add eax, [L6]` ; eax = eax + double word at L6
- `Add [L6], eax` ; double word at L6 += eax
- `Mov al, [L6]` ; copy first byte of double word at L6 into al
- `Mov [L6], 1` ; operation size is not specified
- `Mov dword [L6], 1` ; store a 1 at L6

Instructions: Setup

- STACK
  - can be used as a convenient place to store data temporarily
  - Also used for making subprogram calls, passing parameters a variables.
- Data can only be added in double word units
- PUSH
  - inserts a double word on the stack by **subtracting 4 from ESP**
  - And then stores the double word at [ESP]

Instructions: Setup

- POP
  - reads the double word at [ESP]
  - And then adds 4 to ESP

- CALL
  - Call subprogram
  - Make an unconditional jump to a subprogram
  - And pushes the address of the next instruction on the s

Instructions: Setup

- RET
  - Pops off an address
  - And jumps to that address.

- When using this inst.: It is very important that one m
  the stack correctly so that **the right number is popped**
  the RET.

Example

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

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

Set Up

Body

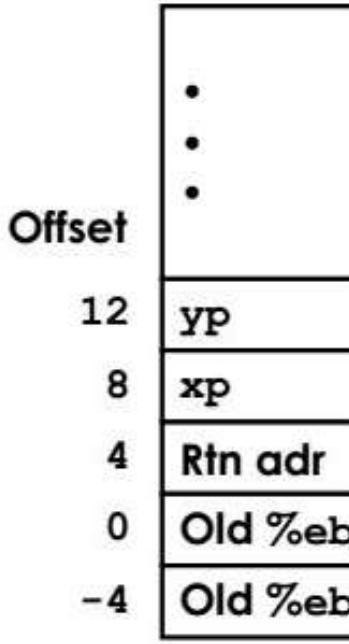
Finish

Example (contd.)

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

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

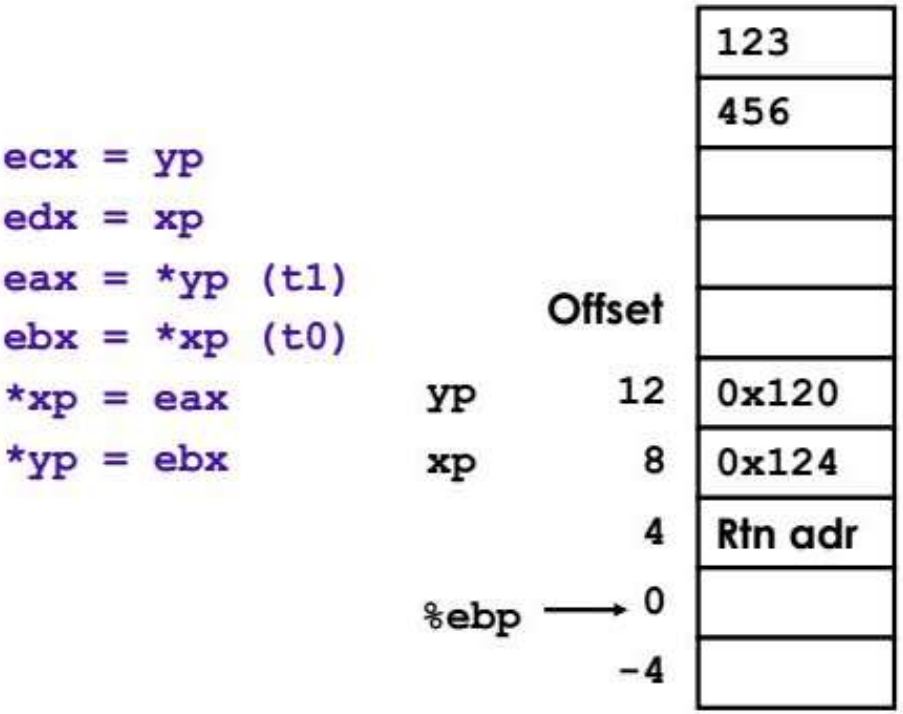
```
movl 12(%ebp),%ecx   # ecx = yp
movl 8(%ebp),%edx    # edx = xp
movl (%ecx),%eax     # eax = *yp (t1)
movl (%edx),%ebx     # ebx = *xp (t0)
movl %eax,(%edx)     # *xp = eax
movl %ebx,(%ecx)     # *yp = ebx
```



Example (contd.)

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

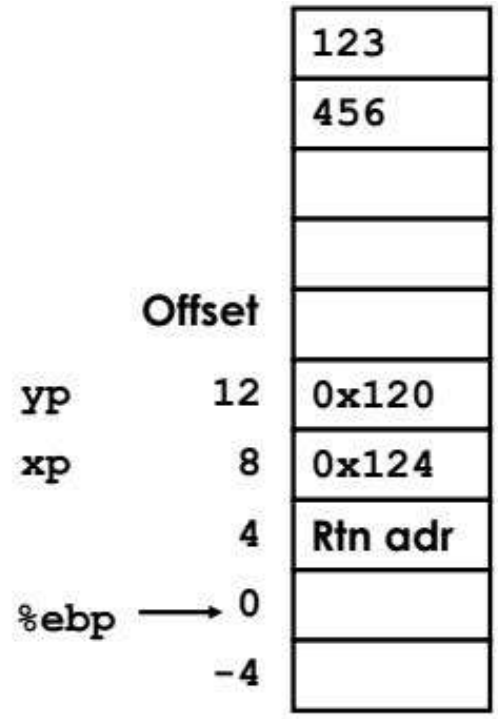
```
movl 12(%ebp),%ecx   # ecx = yp
movl 8(%ebp),%edx    # edx = xp
movl (%ecx),%eax     # eax = *yp (t1)
movl (%edx),%ebx     # ebx = *xp (t0)
movl %eax,(%edx)     # *xp = eax
movl %ebx,(%ecx)     # *yp = ebx
```



Example (contd.)

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

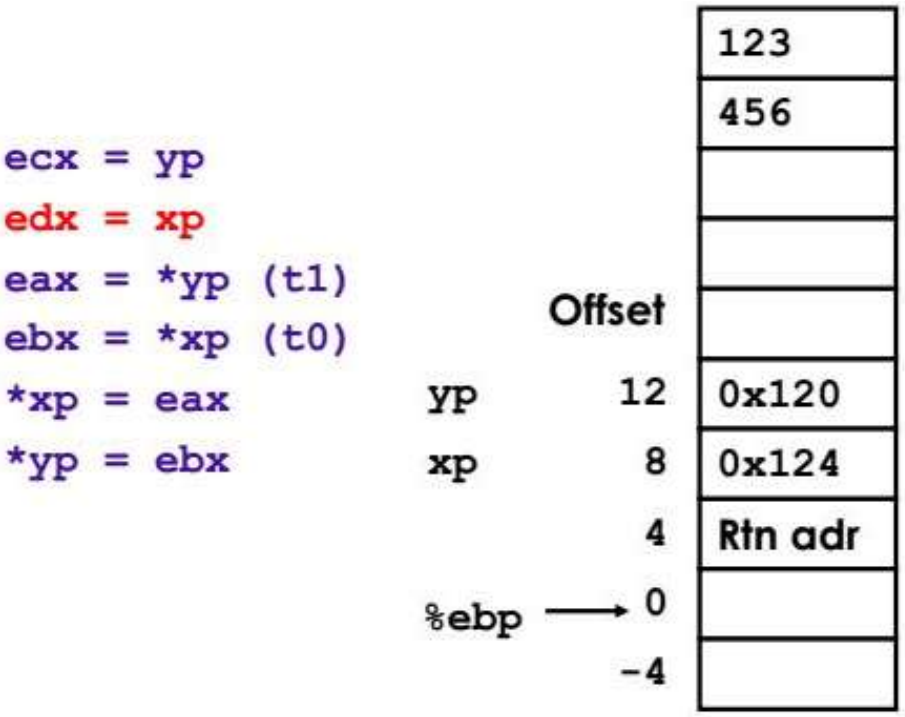
```
movl 12(%ebp),%ecx   # ecx = yp
movl 8(%ebp),%edx    # edx = xp
movl (%ecx),%eax     # eax = *yp (t1)
movl (%edx),%ebx     # ebx = *xp (t0)
movl %eax,(%edx)     # *xp = eax
movl %ebx,(%ecx)     # *yp = ebx
```



Example (contd.)

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

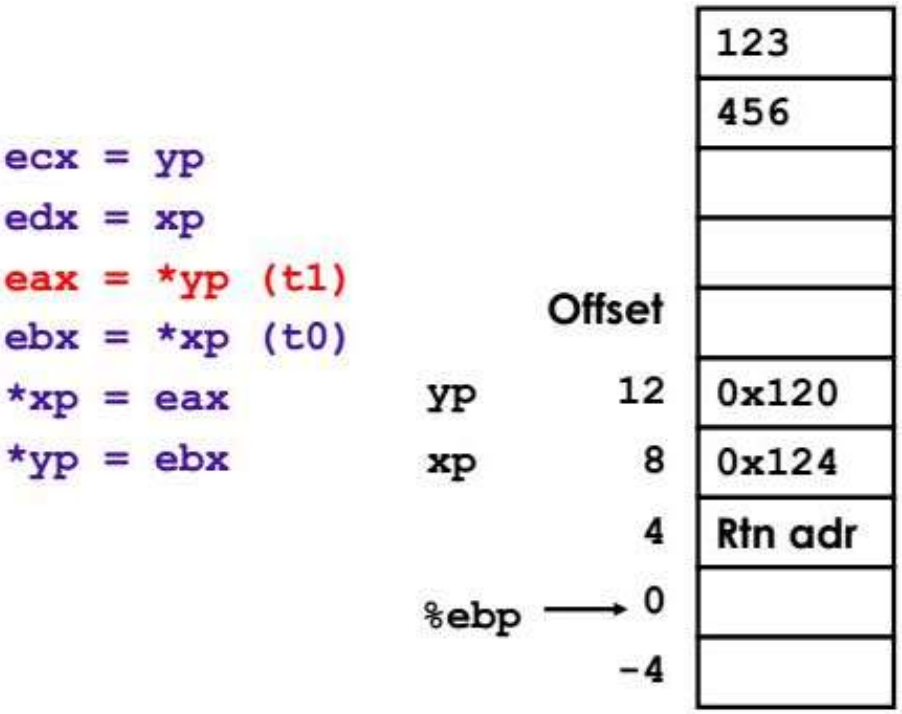
```
movl 12(%ebp),%ecx   # ecx = yp
movl 8(%ebp),%edx    # edx = xp
movl (%ecx),%eax     # eax = *yp (t1)
movl (%edx),%ebx     # ebx = *xp (t0)
movl %eax,(%edx)     # *xp = eax
movl %ebx,(%ecx)     # *yp = ebx
```



Example (contd.)

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

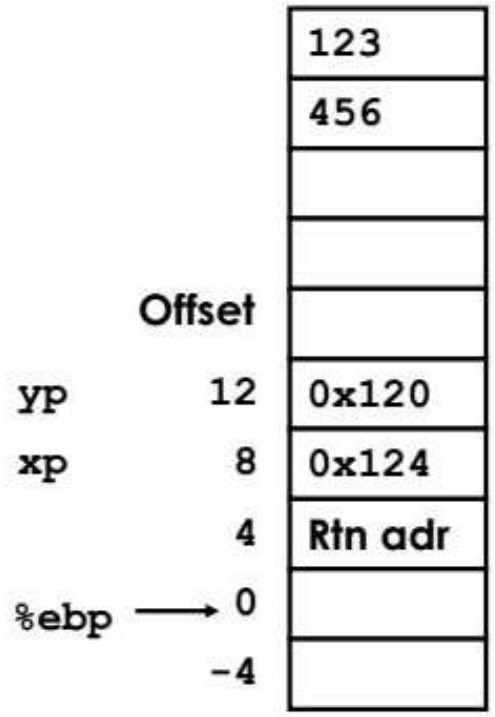
```
movl 12(%ebp),%ecx   # ecx = yp
movl 8(%ebp),%edx    # edx = xp
movl (%ecx),%eax     # eax = *yp (t1)
movl (%edx),%ebx     # ebx = *xp (t0)
movl %eax,(%edx)     # *xp = eax
movl %ebx,(%ecx)     # *yp = ebx
```



Example (contd.)

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

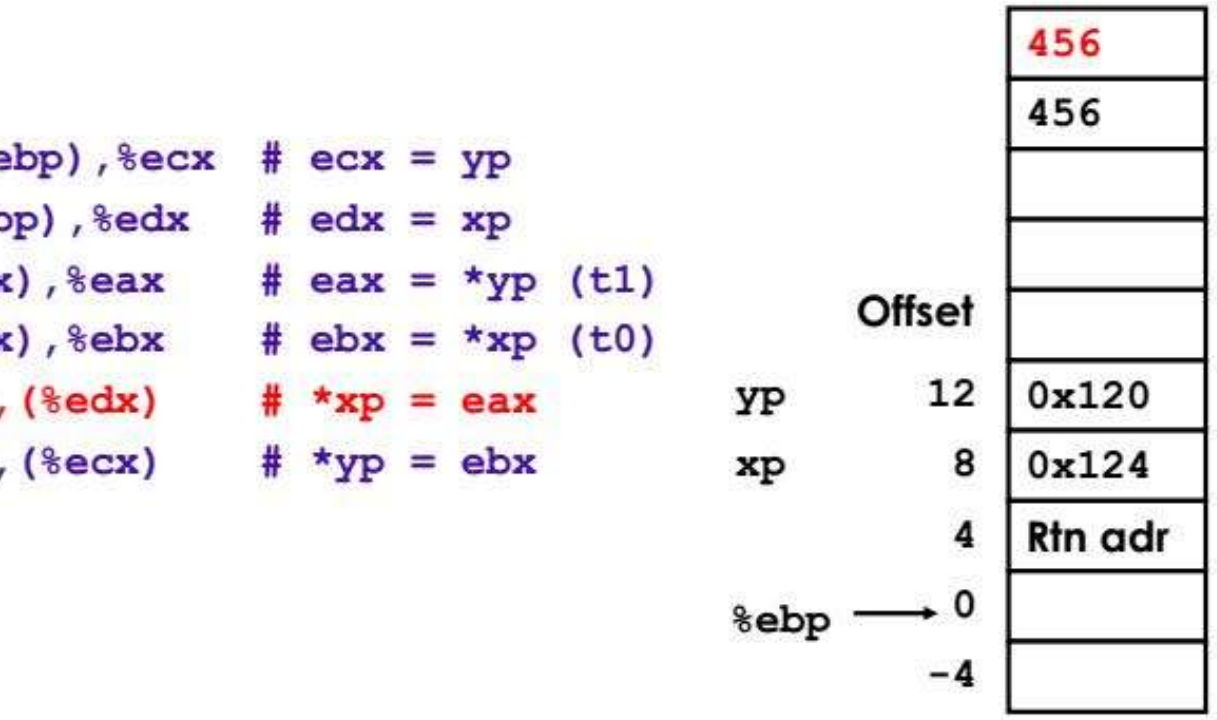
```
movl 12(%ebp),%ecx   # ecx = yp
movl 8(%ebp),%edx    # edx = xp
movl (%ecx),%eax     # eax = *yp (t1)
movl (%edx),%ebx     # ebx = *xp (t0)
movl %eax,(%edx)     # *xp = eax
movl %ebx,(%ecx)     # *yp = ebx
```



Example (contd.)

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

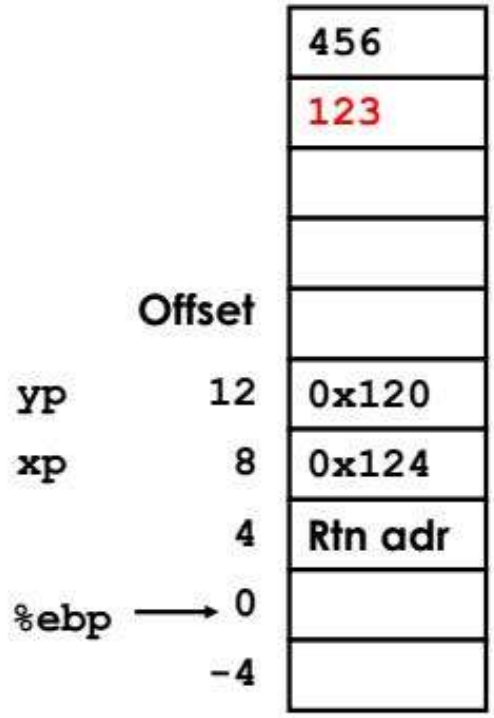
```
movl 12(%ebp),%ecx   # ecx = yp
movl 8(%ebp),%edx    # edx = xp
movl (%ecx),%eax     # eax = *yp (t1)
movl (%edx),%ebx     # ebx = *xp (t0)
movl %eax,(%edx)     # *xp = eax
movl %ebx,(%ecx)     # *yp = ebx
```



Example (contd.)

%eax	456
%edx	0x124
%ecx	0x120
%ebx	123
%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
```



Some Integer Instructions

Format	Computation	
Two-Operand Instructions		
`addl` *Src,Dest*	*Dest* = *Dest* + *Src*	
`subl` *Src,Dest*	*Dest* = *Dest* – *Src*	
`imull` *Src,Dest*	*Dest* = *Dest* * *Src*	
`sall` *k,Dest*	*Dest* = *Dest* << *k*	Also called `shll`
`sarl` *k,Dest*	*Dest* = *Dest* >> *k*	Arithmetic
`shrl` *k,Dest*	*Dest* = *Dest* >> *k*	Logical

k is an immediate value or contents of %cl

Some Integer Instructions

Format	Computation
Two-Operand Instructions	
`xorl` *Src,Dest*	*Dest* = *Dest* ^ *Src*
`andl` *Src,Dest*	*Dest* = *Dest* & *Src*
`orl` *Src,Dest*	*Dest* = *Dest* | *Src*

Some Integer Instructions

Format	Computation
One-Operand Instructions	
`incl` *Dest*	*Dest* = *Dest* + 1
`decl` *Dest*	*Dest* = *Dest* - 1
`negl` *Dest*	*Dest* = -*Dest*
`notl` *Dest*	*Dest* = ~*Dest*

Operands in X86 (NASM)

- Register: **MOV EAX, EBX**
  - *Copy* content from one register to another
- Immediate: **MOV EAX, 10h**
  - *Copy* constant to register
- Memory: different addressing modes
  - Typically at most one memory operand
  - Complex address computation supported