#### Assembly Programing With NASK (and GAS)

DR. ARKA PROKASH MAZUMDAR

## NASM Data Directives

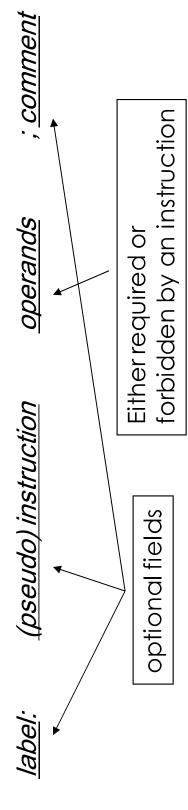
; byte	; word	; byte	; byte	; byte	; double word	; uninitialized byte	; ascii code = 'A'	; 4 bytes	; string		; 100 bytes of zero	; 100*2(word bytes)
0	1000	110101b	12h	170	1A92h	_	V	0,1,2,3	w','o','r','d',0	'word', 0	0	100
qp	Μp	qp	qp	qp	pp	resb	qp	qp	qp	qp	times 100 db	resw
				► L5							► L12	► L13

# Data Types in IA32 (AT&T format)

C declaration	Intel data type	Assembly code suffix	Size (by
char	Byte	q	1
short	Word	M	2
int	Double word	ı	4
long int	Double word	1	4
long long int	1	1	4
char *	Double word	1	4
float	Single precision	Ø	4
double	Double precision	7	8
long double	Extended precision	ţ	10/1

### Instruction Basics

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



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

### Example Program

```
Save frame
                      Create ne
                                 Retrieve
                                             Retrieve )
                                                         Add *xp to
                                                                     Store t at
                                                                                 Restore fi
                                                                                              Return
                                             12(%ebp), %eax
(%edx), %eax
                                8(%ebb), %edx
                                                                    %eax, (%edx)
                     %esb, %ebb
          %epb
          pushl
sımple:
                      movl
                                  movl
                                              movl
                                                                     mov1
                                                                                 pop1
                                                         addl
int simple(int *xp, int y)
                           int t = *xp + y;
                                                        return t;
                                          *xp = t;
```

### Operand Modes

### Source Destination

#### C Analog

Reg movl \$0x4, %eax

Mem movl \$-147, (%eax)

Reg Reg

movl

movl %eax, %edx

Mem movl %eax, (%edx)

Reg Mem

movl (%eax), %edx

temb = \*p;

temp2 = temp\*p = -147;

temp = 0x4;

\*p = temp;

#### Examples

### Data directives (different to MASM)

Mov al, [L1]

; copy byte at L1

Mov eax, L1

; copy ah into byte at L1

; eax = address of byte at L1

Mov [L1], ah

; copy double word

Mov eax, [L6]

; eax = eax + double word at L6

Add [16], eax Add eax, [16]

; double word at L6 += eax

Mov al, [L6]

; operation size is not specified

; copy first byte of double word at L6 into al

Mov [L6], 1

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 and locc
- 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]
- reads the double word at [ESP]
- ► And then adds 4 to ESP

### Instructions: Setup

- Call subprogram
- ► CALL
- ► Make an unconditional jump to a subprogram
- And pushes the address of the next instruction on the stack
- RET
- Pops off an address
- ► And jumps to that address.
- When using this inst.: It is very important that one manage stack correctly so that the right number is popped off by

#### Example

swap:

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

```
Finish
                                                                          Body
        Set
Up
                                            movl 12(%ebp), %ecx
                                                                                                                           movl -4 (%ebp), %ebx
                                                        movl 8 (%ebp), %edx
                                                                   (%ecx), %eax
                                                                              (%edx), %ebx
                                                                                         %eax, (%edx)
                                                                                                     %ebx, (%ecx)
           movl %esp, %ebp
                                                                                                                                       movl %ebp, %esp
                      pushl %ebx
pushl %ebp
                                                                                                                                                popl %ebp
                                                                    mov1
                                                                              mov1
                                                                                          mov1
                                                                                                      movl
```

```
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), %edx
      # edx = xp

      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
```

Old %eb

Rtn adr

χb

12

Offset

Old %eb

	•		10,40	בורי ביי	12	80	4	0 4
					ΥP	ď		«верь
			= *yp (t1)	(to)				
	YP	Хp	*YP	dx*	eax	ebx		
		II	II		П	II		
	ecx =	edx	eax	ebx =	dx*	*YP		
	#	#	#	#	#	#		
	movl 12 (%ebp), %ecx	movl 8 (%ebp), %edx	movl (%ecx), %eax	movl (%edx), %ebx	1 %eax, (%edx)	movl %ebx, (%ecx)		
	MOV.	mov.]	MOV.	MOV	movl	MOV.		
								0×104
S C C C C C C C C C C C C C C C C C C C	4000	%ecx	%ebx		%esi	%edi	%esp	%ebp

Rtn adr

0×120

456

123

0×124

123	456	al.		ָם ע	12 0×120	8 0×124	4 Rtn ad	0 1	-4
			7040	5	Ϋ́Ρ	ďx		%ebp	
			(t1)	(to)					
	9	d'x	*yp	dx*	eax	ebx			
	11		11	II	II	11			
	20	edx	eax	ebx	4xp	*YP			
	#	#	#	#	#	#			
	mov1 12 (%ebb) %ecx		1 (%ecx), %eax	1 (%edx), %ebx	movl %eax, (%edx)	movl %ebx, (%ecx)			
	MOM	MOV.	mov1 (	movl	MOV	MOV			
		0×120						0×104	-0440
%eax	%edx	%ecx	%ebx		%esi	%edi	dse%	%ehn	4
							erica.	The edit	

	123	456					0×12	0×12	Rtn a		
,					10,00	ב ב	12	8	4	î	-4
							Ϋ́Ρ	ф		%ebp	
					(t1)	(to)					
			= YP	ďx	*yp	dx*	eax	ebx			
		9	II	11	II	II		II			
				edx	eax	ebx	= dx*	= d\( \lambda_* \)			
		=	#	#	#	#	#	#			
		0 1 1 0/01	movi 12(%ebp), %ecx	8 (%ebp) , %edx	(%ecx), %eax	. (%edx), %ebx	movl %eax, (%edx)	movl %ebx, (%ecx)			
			TAOM	movl 8	mov1 (	movl	mov1	mov1			
		0×124	3,	0×1×0						0×104	-
	%eax	%edx	,	%ecx %ecx	%ebx		%esi	%edi	%esp	%ahp	1

_		_		_	_	_	-		-	
123	456					0×120	0×124	Rtn adr		
					Offset	12	80	4	1	-4
						Ϋ́Р	ф		%ebp -	
				(t1)	(t0)					
		χb	хb	* 47	dx*	eax	ebx			
			II	11	II	II	11			
		ecx.	edx	eax	ebx	*xp	*yp			
		# =	#	#	#	#	#			
		movl 12(%ebp),%ecx	movl 8 (%ebp), %edx	(%ecx), %eax			movl %ebx, (%ecx)			
	•	TAOM	TAOM	movl	movl	movl	movl			
456	0×124	120	0×1×0						0×104	
%eax	%edx		<b>₹</b>	Sohe	4000	%esi	%edi	%esp	%ehn	1

123	456				0x120	0×124	Rtn adr		
				Offset	12	ω	4	î	-4
					Ϋ́	ď		%ebp	
			(£1)	(to)					
		dy (	* 40			ebx			
		11 1			11	II			
		ecx	Y X		*xp	*YP			
		# #		#	#	#			
		movl 12(%ebp),%ecx	movi (%ecx), %eax		OF	movl %ebx, (%ecx)			
		TAOM	mov1	mov1	movl	movl			
456	0×124	0×120	,	123				0×104	
%eax	%edx	× C		%ebx	%esi	%edi	%esp	%ehn	1
									_

ecx = yp	II		$\frac{1}{2}$ $\frac{1}$	$^*$ $yp = ebx$ $xp$	- dqə%
#	#	# #	<b>#</b>	#	
movl 12 (%ebp), %ecx	movl 8 (%ebp), %edx	movl (%ecx), %eax	movl %eax, (%edx)	movl %ebx, (%ecx)	
mov1 12	movl	movl	O O	MO	
0x124 movl 12	0x120 mov1	123 mov	OH OH	ОШ	0×104

Rtn adr

0×120

456

456

456

%eax

0×124

123					0×120	0×124	Rtn adr		
7				Offiser	12	8	4	î	-4
					ΥP	ď		%ebp —	
			(t1)	(to)					
	ΥP	хb	*yp	dx*	eax	ebx			
	II	II	11	11	11	H			
	ecx =	edx	eax	ebx	4xp	*YP			
	#	#	#	#	#	#			
	mov1 12 (%ebp), %ecx	mov1 8 (%ebp), %edx	movl (%ecx), %eax	movl (%edx), %ebx	movl %eax, (%edx)	movl %ebx, (%ecx)			
0×124		0 <b>x</b> 120	123					0×104	- 2
%edx		%ecx	%ehx		%esi	%edi	%esp	aqe*	1

456

456

%eax

## Some Integer Instructions

Format

Computation

**Two-Operand Instructions** 

add1 Src, Dest + Src

Dest = Dest - Src

subl Src, Dest

imull Src, Dest = Dest \* Src

Dest = Dest << k

sall k,Dest

sarl k,Dest

Also called sh11

Arithmetic

Logical

Dest = Dest >> k shrl k,Dest

k is an immediate value or contents of %cl

## Some Integer Instructions

Format

Computation

**Two-Operand Instructions** 

xorl Src, Dest - Dest - Src

Dest = Dest & Src

andl Src,Dest

orl Src, Dest | Src

## Some Integer Instructions

Format

Computation

One-Operand Instructions

incl Dest Dest = Dest + 1

Dest = Dest - 1

Dest = -Dest

negl Dest

notl Desf

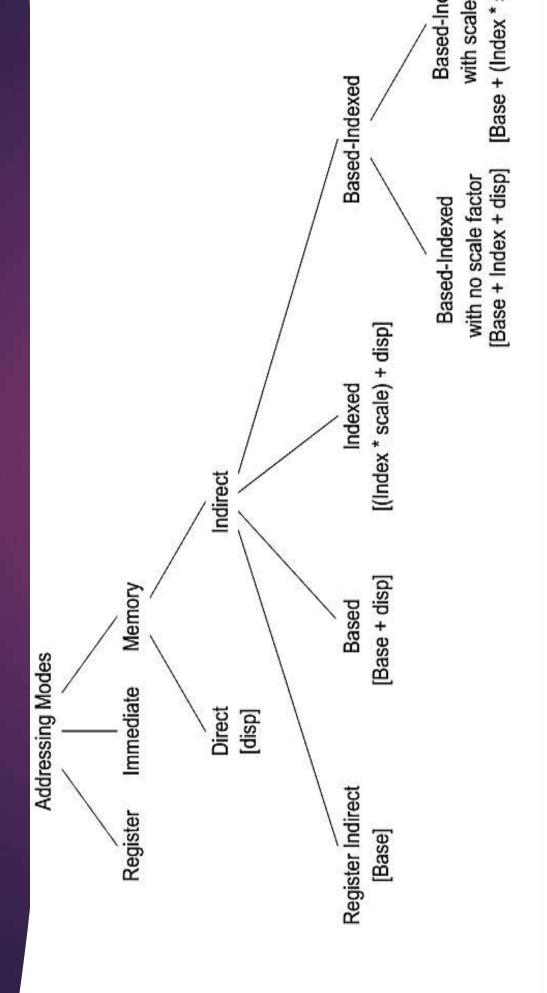
decl Dest

Dest = ~Dest

### Operands in X86

- ► 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

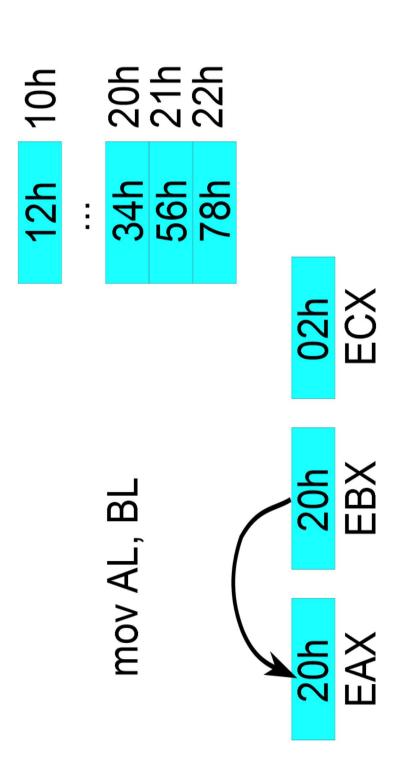
## Addressing modes (32-bits)



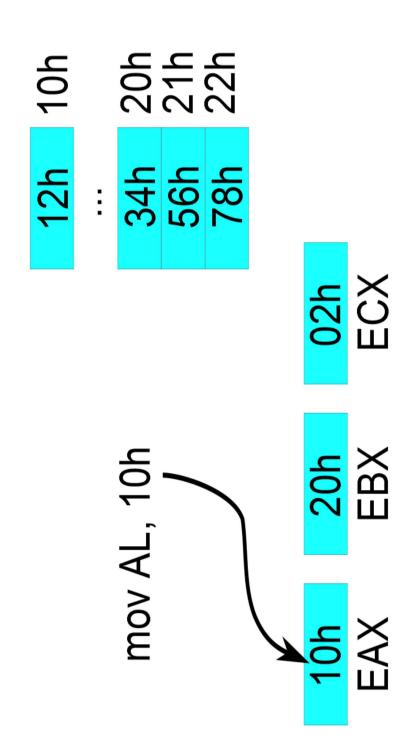
### Addressing modes

- ▶ Direct: MOV EAX, [10h]
- Copy value located at address 10h
- ► Indirect: MOV EAX, [EBX]
- Copy value pointed to by register BX
- ► Indexed: MOV AL, [EBX + ECX \* 4 + 10h]
- ► Copy value from array (BX[4 \* CX + 0x10])
- Pointers can be associated to type
- ► MOV AL, byte ptr [BX]

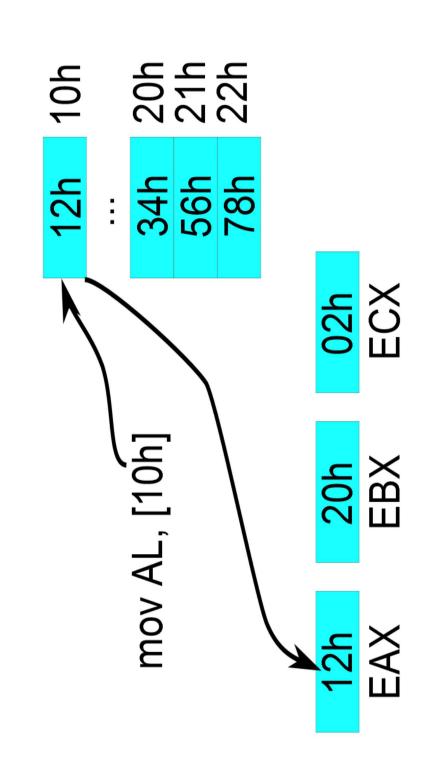
### Operands and addressing modes: Register



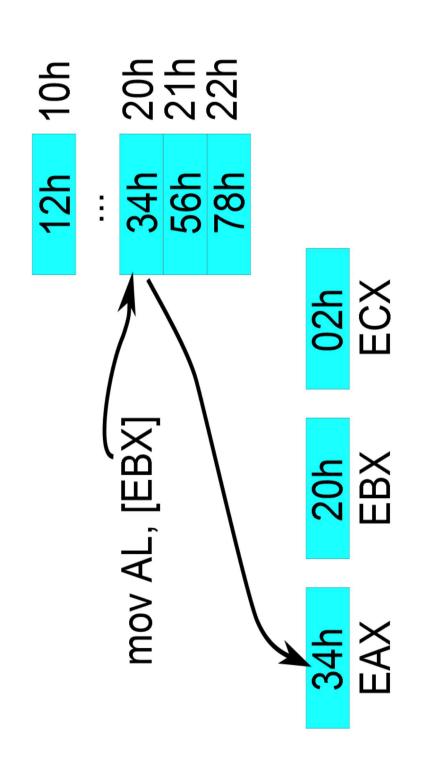
### Operands and addressing modes: Immediate



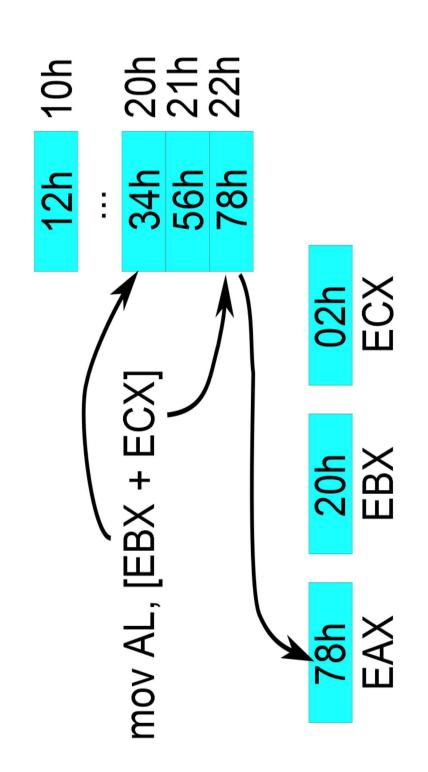
### Operands and addressing modes: Direct



### Operands and addressing modes: Indirect



### Operands and addressing modes: Indexed



# Generic 32bit Addressing Mode

Segment + Base + (Index \* Scale) + displacement

EAX EAX

EBX

no displacement

8-bit displacement

32-bit displacement

EBX ECX EDX ESI EDI EDI

ECX EDX ESI EDI

EBP EBP

## Addressing Mode GNU-AS

►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

# GAS - Indexed Addressing Modes

- 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] ]

### Solve the Followings:

#### Expression

0x8(%edx)

(%edx,%ecx)

(%edx,%ecx,4)

0x80(,%edx,2)

0x£000	0×100
%edx	%ecx

► Given:

# Another Programing Example

#### Objective:

- Take an integer through Command-line
- Print the value
- Add 1 to that
- Print the result

### Another Program (1)

```
SECTION .text
global main
main:
; set-up phase
push ebp
mov ebp, esp
mov esp, ebp
pop ebp
ret
```

```
get the second argument data
                                                                                                                                                                                                                                 ; get argv starting address
Another Program (2)
                                                                                                                                                                                                                                mov ebx, DWORD [esp + 12]
                                                                                                                                                                                                              ; get the command-line data
                                                                                                                                                                                                                                                  mov ebx, [ebx + 4]
                                                                                                                                                                         mov ebp, esp
                                                                                                                                                                                                                                                                                                        mov esp, ebp
                                                                                                   global main
                                                                                                                                                          dqe ysnd
                                                                                                                                                                                                                                                                                                                         pop ebp
ret
                                                                                                                                        ; set-up phase
                                                                                                                                                                                                                                                                                      ; finish phase
                                                                                   SECTION . text
```

### Another Program (3)

```
; print argv[1] data
                  "You Entered - %s", 10, 0
SECTION . data
                   <del>අ</del>
                     msg:
```

```
SECTION .text
extern printf
global main
```

```
main:
    set-up phase
    push ebp
    mov ebp, esp
```

### Another Program (3)

```
; get the second argument data
                                                                                              ; put data on stack for call
              ; get argv starting address
                                                                                                                                                                                               ; esp back to start
                                                                                                                  print the value
                    mov ebx, DWORD [esp + 12]
; get the command-line data
                                    mov ebx, [ebx + 4]
                                                                                                                                                                                                                   mov esp, ebp
                                                                                                                                       call printf
                                                                            ; print the value
                                                                                                                                                                                               add esp, 8
                                                                                               push ebx
                                                                                                                    bsw ysnd
                                                                                                                                                                           ; finish phase
                                                                                                                                                                                                                                      dqe dod
                                                                                                                                                                                                                                                          ret
```

### Another Program (4)

```
"You Entered - %s", 10, 0 ; print argv[1] data "This is int - %d", 10, 0 ; print INT equivalent
                                                                                                    extern printf
                                                                                                                         extern atoi
                                                                                                                                           global main
                                                                                                                                                                                                                         dqa ysnd
                                                                                                                                                                                                      ; set-up phase
                                                                               SECTION . text
SECTION .data
                      <del>අ</del>
                                       msg2: db
                                                                                                                                                                                    main:
                       msg:
```

mov ebp, esp

### Another Program (4)

```
; get the second argument data
                                                                                                             ; put data on stack for call
                 ; get argv starting address
                                                                                                                                   print the value
                       mov ebx, DWORD [esp + 12]
; get the command-line data
                                           mov ebx, [ebx + 4]
                                                                                                                                                          call printf
                                                                                       ; print the value
                                                                                                             bush ebx
                                                                                                                                      bsm ysnd
```

; call atoi - return in EAX?

; stack points to entry edx

; convert to integer

add esp, 4

call atoi

### Another Program (4)

```
; increase eax for testing
                   ; esp points to start
                                                                                                   ; push arg for print
; get return value and add 1
                                                                               ; print the result
                    add esp, 4
                                                                                                                        push msg2
                                                                                                   push eax
                                        inc eax
```

```
; push print message
                                                         ; esp back to start
                                                                        mov esp, ebp
              call printf
                                                          add esp, 8
                                         ; finish phase
                                                                                       dqe dod
                                                                                                      ret
```

## Another Program (final)

```
"You Entered - %s", 10, 0 ; print argv[1] data "This is int - %d", 10, 0 ; print INT equivalent
                                                                                                        extern printf
                                                                                                                              extern atoi
                                                                                                                                                 global main
                                                                                                                                                                                                                ; set-up phase
                                                                                  SECTION . text
SECTION .data
                      <del>අ</del>
                                       msg2: db
                                                                                                                                                                                             main:
                       msg:
```

mov ebp, esp

dqa qsnd

## Another Program (final)

```
; get the second argument data
                                                                                                              ; put data on stack for call
                 ; get argv starting address
                                                                                                                                     print the value
                       mov ebx, DWORD [esp + 12]
; get the command-line data
                                           mov ebx, [ebx + 4]
                                                                                                                                                            call printf
                                                                                         ; print the value
                                                                                                               push ebx
                                                                                                                                       bsm ysnd
```

; call atoi - return in EAX?

; stack points to entry edx

; convert to integer

add esp, 4

call atoi

## Another Program (final)

```
; get return value and add 1
add esp, 4
inc eax
```

```
print the result

push eax

push msg2

call printf
```

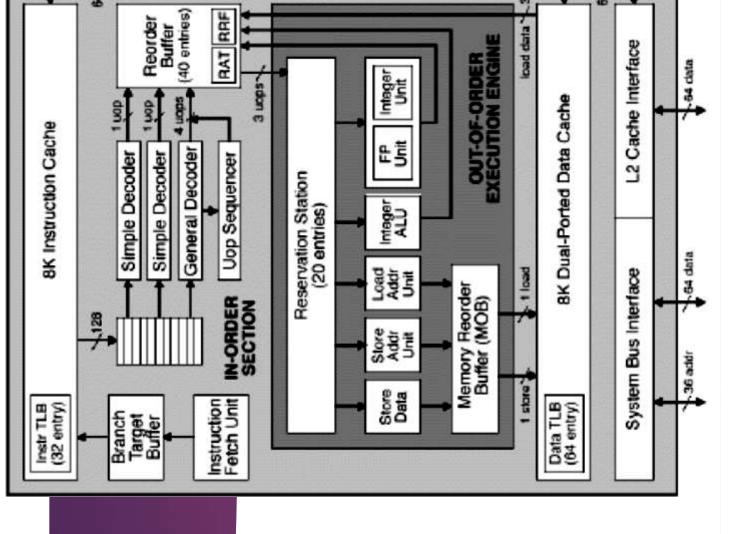
```
; finish phase
add esp, 8
mov esp, ebp
pop ebp
ret
```

```
; esp points to start
; increase eax for testing
```

```
; push arg for print
; push print message
```

```
; esp back to start
```

#### PentiumPro Block Diagram



#### Operations

Translates instructions dynamically into "Uops" / "µops"

▶ 128 bits wide

► Holds

► Operation 3

► Two sources, and +32 +32

▶ Destination +32

= 128bits

Executes Uops with "Out of Order" engine

Uop executed when

1. Operands are available

2. Functional unit available

#### Operations

- ► Execution controlled by "Reservation Stations"
- ► Keeps track of data **dependencies** between uops
- ► Allocates resources
- Consequences
- ▶ Indirect relationship between
- ► IA32 code &
- ► What actually gets executed
- Tricky to predict / optimize performance at assembly level

#### Machine View



- ► Address of next instruction
- ► Register File
- ► Heavily used program data

Instructions

Condition

Codes

Data

Addresses

Registers

- Condition Codes
- ▶ Store status information about most recent arithmetic operation
- ► Used for conditional branching

#### ► Memory

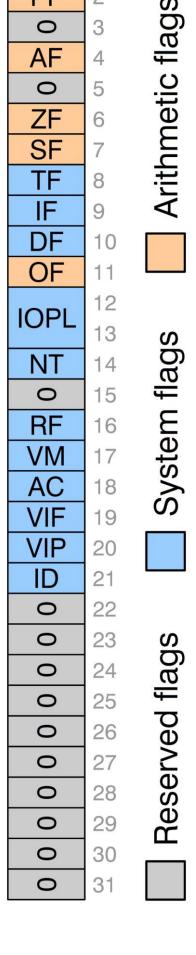
- ▶ Byte addressable array
- ► Code, user data, (some) OS data
- Includes stack used to support procedures

## Flow Control Instructions

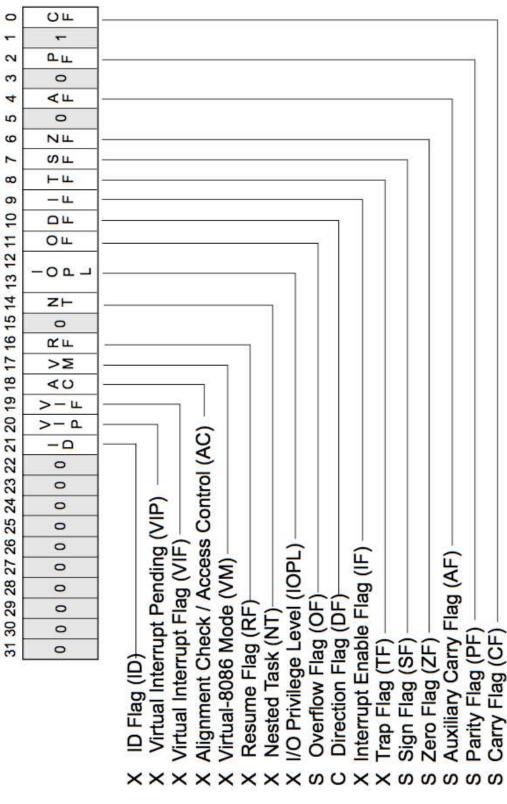
- ▲ JMP
- ► Jcc ► CALL
- ▼ RET

#### Revisit EFLAGS

#### eflags register



#### EFLAGS Revisit



S Indicates a Status Flag C Indicates a Control Flag X Indicates a System Flag

Indicates a Control Flag

Indicates a System Flag

Reserved bit positions. DO NOT USE. Always set to values previously read.

#### JMP: jump

► Syntax:

JMP dest

► Operation (absolute jump):

EIP  $\leftarrow$  dest

► Operation (relative jump):

EIP  $\leftarrow$  EIP + dest

ı	pf
ı	af
ı	Zf
ı	Sf
ı	df
ı	of

### Unconditional Jumps

- ▶ Jmp statement label
- ► We have two types of jumps,
- ► Intersegment
- ► Intrasegment
- ► Address can be in a register, variable or label.

### Unconditional Jumps

Example:

Start: MOV AX, 0

INC AX,

JMP Start

## Jcc: short jump conditional

#### ► Syntax:

Jcc dest

#### ▶ Operation:

if 
$$(cc)$$
  
EIP  $\leftarrow$  EIP +  $dest$   
endif

**Notes:** cc is any of the condition codes. dest must be within a signed 8-bit range (-128 to 127).



#### Condition Codes

► Implicitly Set By Arithmetic Operations

add1 Src, Dest C analog: t = a + b

► CF set if carry out from most significant bit

► Used to detect unsigned overflow

► ZF set if t == 0

► SF set if t < 0

► OF set if two's complement overflow

(a>0 && b>0 && t<0) || (a<0 && b<0 && t>=0)

►Not Set by leal instruction

#### Condition Codes

Sufix	Meaning	Flags
0	Overflow	OF=1
ON	No Overflow	0=40
S	Carry	
Ω	Below	CF=1
NAE	NAE   Not Above nor Equal	
NC	No Carry	
NB	Not Below	CF=0
AE	Above or Equal	

Sufix	Meaning	Flags
Z	Zero	$\Sigma F=1$
Ш	E Equal	
ZN	Not Zero	ZF=0
Ш	Not Equal	
BE	Below or Equal	CF=1 OR ZF=1
۲	Not Above	
Y	Above	CF=0 AND ZF=0
NBE	NBE Not Below nor Equal	

Sufix	Meaning	Flags
S	Sign	SF=1
SN	Not Sign	SF=0
<b>a</b>	Parity	PF=1
PE	Parity Even	
NP	Not Parity	PF=0
PO	Parity Odd	

Sufix	Meaning	Flags
7	7 Ssez	SF<>OF
NGE Not	Not Greater nor Equal	
<b>3</b> 9	Greater or Equal	SF=OF
٦	Not Less	
37	Less or Equal	ZF=1 OR SF<>OF
Ŋ	Not Greater	
9	Greater	ZF=0 AND SF=OF
NLE	Not Less nor Equal	

- ► Above and Below are used for unsigned integer comparisons.
- ► Greater and Less are used for signed integer comparisons.

### Conditional Jumps

- ► Dependent on condition codes.
- Example:

JZ  $\rightarrow$  jump if zero flag is set.

### Conditional Codes

Code the following C routine using assembly language ... (body for negative condition) ... (body for positive condition) ... (body for zero condition) Add a value X to eax; instructions. Else if x = 0If x < 0 Then Else

### Conditional Codes

```
Solution
```

;add a value to eax Add eax, X

;jump if eax is not negative JNS elselfZero ; code for negative condition

JMP endCheck

#### elseifZero:

; jump if x is not zero JNZ elsePos

; code for zero condition

JMP endCheck

; code for positive balance

elsePos: ...

endCheck:

### Conditional Codes

Add eax, X ;add a value to eax

JNS elselfZero

•

JMP endCheck

**SeifZero** 

JNZ elsePos

; code for zero condition

; jump if x is not zero

1

JMP endCheck

elsePos:

; code for positive balance

endCheck:

Add a value X to eax

**If** × < 0

Then

; code for negative condition

;jump if eax is not negative

... (body for negative co

Else if x = 0

... (body for zero condit

Else

... (body for positive cor

End if