#### Programme with Assembly (and GAS) NASM

DR. ARKA PROKASH MAZUMDAR

# NASM Data Directives

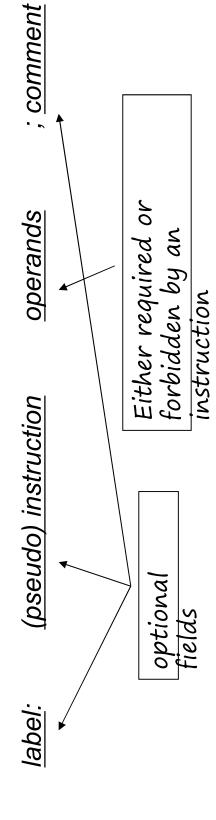
```
; 100 bytes of zero
; 100*2(word bytes)
              ; byte
; byte
; double word
; uninitialized byte
; ascii code = 'A'
; 4 bytes
                                             ; string
    ; word
; byte
         ; byte
                                        0,1,2,3
'w','o','r','d',0
         1101011
                                                  word', o
                    170
1A92h
                                                       times 100 db 0
```

# Data Types in IA32 (AT&T format)

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

### Instruction Basics

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



- ends with backslash, the next line is considered to Backslash (N) uses as the line continuation character: if a line be a part 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 new
                                 Retrieve
                                             Retrieve ;
                                                         Add *xp to
                                                                     Store t at
                                                                                 Restore fi
                                                                                              Return
                                            12(%ebp), %eax
                                8(%ebp), %edx
                                                       (%edx), %eax
                                                                    %eax, (%edx)
                     %esb, %epb
          %epb
          pushl
sımple:
                      movl
                                  movl
                                             movl
                                                         addl
                                                                     mov1
                                                                                pop1
int simple(int *xp, int y)
                           int t = *xp + y;
                                                        return t;
                                           *xp = t;
```

### Operand Modes

### Source Destination

C Analog

Imm | Reg

Reg movl \$0x4, %eax

temp = 0x4;

\*p = -147;

Mem mov1 \$-147, (%eax)

Reg Reg

mov1

movl %eax, %edx

Mem movl %eax, (%edx)

Mem Reg

movl (%eax), %edx

temb = \*p;

temp2 = temp

\*p = temp;

# NASM Data Directives

```
; 100 bytes of zero
; 100*2(word bytes)
              ; byte
; byte
; double word
; uninitialized byte
; ascii code = 'A'
; 4 bytes
                                             ; string
    ; word
; byte
         ; byte
                                        0,1,2,3
'w','o','r','d',0
         1101011
                                                  word', o
                    170
1A92h
                                                       times 100 db 0
```

#### Examples

# Data directives (different to MASM)

```
u Mov al, [L1]
```

; eax = address of byte at L1

2

2

### Instructions: Setup

#### U STACK

- v can be used as a convenient place to store data temporarily
- Also used for making subprogram calls, passing parameters au
- Data can only be added in double word units

#### PUSH

- v inserts a double word on the stack by **subtracting 4 from ESF**
- And then stores the double word at [ESP]

### Instructions: Setup

- POP
- v reads the double word at [ESP]
- o And then adds 4 to ESP
- CALL
- v Call subprogram
- v Make an unconditional jump to a subprogram
- v And pushes the address of the next instruction on the

### Instructions: Setup

- v RET
- v Pops off an address
- o And jumps to that address.
- when using this inst.: It is very important that one make stack correctly so that the right number is popped the RET.

#### Example

:dews

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

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

```
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

Ϋ́

12

Offset

Old %eb

		1000	Oilsei	12	00	4	1
				ΥP	άx		%ebp
		(t1)	(to)				
ΔX	М	= *yp	dx*	eax	= ebx		
II	II	II	II	II	Ш		
ecx	edx	eax	ebx	dx*	*yp		
#	#	#	#	#	#		
movl 12 (%ebp), %ecx	movl 8 (%ebp), %edx	movl (%ecx), %eax	movl (%edx), %ebx	movl %eax, (%edx)	movl %ebx, (%ecx)		
lvon	mov1	movl	movl	novl	lov1		
H	7.555		100	H	н		
		Г		н	н		0×104

Rtn adr

0×120

456

123

0×124

0x120       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	V-	%ebp → 0	4 (	ω	12	Olisa	Office			
0x120       movl 12 (%ebp), %ecx # ecx = yp         movl 8 (%ebp), %edx # edx = xp         movl (%ecx), %eax # eax = *yp         movl (%edx), %ebx # ebx = *xp         movl %eax, (%edx) # *xp = eax         movl %ebx, (%ecx) # *yp = ebx		%ebp		ď	Ϋ́					
mov1 12 (%ebp), %ecx  # ecx = mov1 8 (%ebp), %edx  # edx = mov1 (%ecx), %eax  # eax = mov1 (%edx), %ebx  # ebx = mov1 %eax, (%edx)  # *xp = mov1 %ebx, (%ecx)  # *yp = mov1 %ebx, (%e										
mov1 12 (%ebp), %ecx  # ecx = mov1 8 (%ebp), %edx  # edx = mov1 (%ecx), %eax  # eax = mov1 (%edx), %ebx  # ebx = mov1 %eax, (%edx)  # *xp = mov1 %ebx, (%ecx)  # *yp = mov1 %ebx, (%e				ebx	eax	*xp	*yp	dx	YP	
mov1 12 (%ebp), %ecx # ecx     mov1 8 (%ebp), %edx # edx     mov1 (%ecx), %eax # eax     mov1 (%edx), %ebx # ebx     mov1 %eax, (%edx) # *xp     mov1 %ebx, (%ecx) # *yp     mov1 %ebx, (%ebx) # *yp					11	H	11	11		
movl 12 (%ebp), %ecx # movl 8 (%ebp), %edx # movl (%ecx), %eax # movl (%edx), %ebx # movl %edx)				*47		ebx	eax		ecx	
0x120 0x120				#	#	#	#	#	=#=	
0x120				sebx, (%ecx)	seax, (sedx)		(%ecx),%eax	8 (%ebp) , %ed	. 12(%ebp),%e	
				mov]	mov1	mov]	mov1	mov]	[vom	
X X X X I I I I I I I I I	-	0×104						0×120		
%edx %ebx %ebx %edi	4	%ebp	%esp	%edi	%es1		%ebx	%ecx	%edx	%eax

Rtn adr

0×124

0×120

456

123

123	456				0×12	0×12	Rtn a		
			) ffe o		12	8	4	î	The Control of
					ΥP	ď		%ebp	
			(t1)	(to)					
	8	d'x	*YP	*xp	eax	ebx			
	11	II	11	II	II	П			
	A C	edx	eax	ebx	dx*	*YP			
	#	#	#	#	#	#			
	mov1 12 (%ebb) .%ecx	8 (%ebp), %edx	(%ecx), %eax	(%edx), %ebx	movl %eax, (%edx)	movl %ebx, (%ecx)			
	Twom1	movl	movl	movl	movl	movl			
	0×124	0×120						0×104	
%eax	%edx	%ecx	%ebx		%esi	%edi	dse%	%ebp	4

			Office		12	00	4	<b>↓</b> 4
					Ϋ́	ďx		%ebp
			(t1)	(to)				
	Q.A	c d'x	*YP	dx*	eax	ebx		
	H	11	11	II	11	II		
	ecx		eax	ebx	dx*	*yp		
	#	#	#	#	#	#		
	ebp), %ec	movl 8 (%ebp), %edx	movl (%ecx), %eax	(%edx), %ebx	movl %eax, (%edx)	movl %ebx, (%ecx)		
	12 (	œ						
	movl 12 (%ebp), %ecx	mov1 8	movl	movl	movl	movl		
456	0x124 mov1 12(	0x120 mov1 8	movl	movl	movl	movl	2	0×104

Rtn adr

0×124

0×120

456

123

			1090		12	<b>®</b>	4	0 4
					ΥP	ď		%ebp
			(t1)	(to)				
	2	, d	*yP	dx*	eax	ebx		
	П	11	11	H	11	II		
	ecx =	edx	eax	ebx	*xp	*YP		
	#	#	#	#	#	#		
	movl 12 (%ebp), %ecx	movl 8 (%ebp), %edx	l (%ecx), %eax	1 (%edx), %ebx	1 %eax, (%edx)	movl %ebx, (%ecx)		
	MOV	MOV	movl	movl	mov1	MOV.		
456	0x124	0x120	123					0x104
%eax	%edx	%ecx	%ebx		%esi	%edi	%esp	%epb

Rtn adr

0×124

0×120

456

123

	456		je set	12 0×120	8 0×124	4 Rtn ad	0.4	-4
			Offset	ΥΡ	ф		%ebp	
			(t1)					
	8	x dx	* YP	eax x	ebx			
	II		11 1		II			
	×		eax		*yp			
	#	= #	# #	= #	#			
	mov1 12 (%ebb) %ecx	mov1 8 (%ebp), %edx	(%ecx), %eax	O.	movl %ebx, (%ecx)			
	LACE	mov1	mov1	movl	movl			
2	0×124	0×120	123				0×104	->==
4000	%edx	%ecx	%ebx	%esi	%edi	%esp	%ehn	1

	123					0×120	0×124	Rtn adı		
				1000		12	00	4	%ebp → 0	-4
						ΥP	ďx		%ebp	
				(t1)	(£0)					
		YP	фx	*yP	dx*	eax	ebx			
		II	11	11	II	11	H			
		ecx	edx	eax	ebx	dx*	*YP			
		#	#	#	#	#	#			
		movl 12 (%ebp), %ecx	mov1 8 (%ebp), %edx	(%ecx), %eax	(%edx), %ebx	movl %eax, (%edx)	movl %ebx, (%ecx)			
		movl	movl	mov1	movl	movl	movl			
	0×124		0x120	123			-		0×104	
	%edx		%ecx	%ebx		%esi	%edi	%esp	%ehp	1
•										

456

456

%eax

# Some Integer Instructions

Format

Computation

Two-Operand Instructions

add1 Src, Dest = Dest + Src

Dest = Dest - Src

subl Src, Dest

imull Src, Dest \* Src

Dest = Dest << k

sall k,Dest

sarl k,Dest

Also called sh11

Arithmetic

Logical

Dest = Dest >> k

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

Dest = Dest + 1incl Dest Dest = Dest - 1

Dest = -Dest

negl Dest

notl Dest

decl Dest

Dest = ~Dest

# Operands in X86 (NASM)

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