ASSEMBLY AND SYNTAX FUNDAMENTALS

Chapter 3

Objectives

- Distinguish differences in Assembly syntaxes
- Identify sections of Assembly code and explain the use of each
- Construct semantically correct data definitions
- Create working Assembly programs

Web Resources

- https://sourceware.org/binutils/docs/as/ (GAS Reference)
- https://msdn.microsoft.com/en-us/library/afzk3475.aspx (MASM Reference)
- http://www.nasm.us/doc/nasmdoc0.html (NASM Reference)

Syntax Conventions

Table 3.1 Syntax conventions

Assembler	Syntax	Development environment
GAS	AT&T	Apple Xcode
MASM	Intel	Microsoft Visual Studio
NASM	Intel	Command-line in Linux

Reserved Words

Table 3.2 Sample reserved words

GAS	MASM	NASM

`.data`
`sum: .long 0`
 	
`.text`
`.globl _main`
`_main:`
`movl $25, %eax`
`movl $50, %ebx`
`addl %ebx, %eax`
`movl %eax, sum`
 	
`pushl $0`
`subl $4, %esp`
`movl $1, %eax`
`int $0x80`
`.end`
	`.386`
	`.MODEL FLAT, stdcall`
	`.STACK 4096`
	`ExitProcess PROTO,`
	`dwExitCode:DWORD`
	
	`.data`
	`sum DWORD 0`
	
	`.code`
	`_main PROC`
	`mov eax, 25`
	`mov ebx, 50`
	`add eax, ebx`
	`mov sum, eax`
	
	`INVOKE ExitProcess, 0`
	`_main ENDP`
	`END`
		`SECTION .data`
		`sum: DD 0`
		
		`SECTION .text`
		`global _main`
		`_main:`
		`mov eax, 25`
		`mov ebx, 50`
		`add eax, ebx`
		`mov DWORD [sum], eax`
		
		`mov eax, 1`
		`mov ebx, 0`
		`int 80h`

Identifiers

Table 3.3 Sample identifiers

GAS	MASM	NASM

.data
sum: .long 0
 	
.text
.globl **_main**
_main:
movl $25, %eax
movl $50, %ebx
addl %ebx, %eax
movl %eax, **sum**
 	
pushl $0
subl $4, %esp
movl $1, %eax
int $0x80
.end
	.386
	.MODEL FLAT, stdcall
	.STACK 4096
	ExitProcess PROTO,
	dwExitCode:DWORD
	
	.data
	sum DWORD 0
	
	.code
	_main PROC
	mov eax, 25
	mov ebx, 50
	add eax, ebx
	mov **sum**, eax
	
	INVOKE **ExitProcess**, 0
	_main ENDP
	END
		SECTION .data
		sum: DD 0
		
		SECTION .text
		global **_main**
		_main:
		mov eax, 25
		mov ebx, 50
		add eax, ebx
		mov DWORD [**sum**], eax
		
		mov eax, 1
		mov ebx, 0
		int 80h

- **Letters (A-Z, a-z)**
- **Underscore (_)**
- **Question Mark (?)**
- **At-symbol (@)**
- **Dollar Sign ($)**

- **Recommend against ?, @, and $**

Directives

Table 3.4 Sample directives

GAS	MASM	NASM

`.data`
`sum: .long 0`
 	
`.text`
`.globl _main`
`_main:`
`movl $25, %eax`
`movl $50, %ebx`
`addl %ebx, %eax`
`movl %eax, sum`
 	
`pushl $0`
`subl $4, %esp`
`movl $1, %eax`
`int $0x80`
`.end`
	`.386`
	`.MODEL FLAT, stdcall`
	`.STACK 4096`
	`ExitProcess PROTO,`
	`dwExitCode:DWORD`
	
	`.data`
	`sum DWORD 0`
	
	`.code`
	`_main PROC`
	`mov eax, 25`
	`mov ebx, 50`
	`add eax, ebx`
	`mov sum, eax`
	
	`INVOKE ExitProcess, 0`
	`_main ENDP`
	`END`
		`SECTION .data`
		`sum: DD 0`
		
		`SECTION .text`
		`global _main`
		`_main:`
		`mov eax, 25`
		`mov ebx, 50`
		`add eax, ebx`
		`mov DWORD [sum], eax`
		
		`mov eax, 1`
		`mov ebx, 0`
		`int 80h`

MASM-specific Directives

Table 3.5 32-bit MASM-specific directives

Directive	Description
.386	Enables the 80386 processor instructions and disables newer instructions. Other valid settings to enable additional instructions are .486, .586, .686, .MMX, and .XMM, among others.
.MODEL	Sets the memory model. The only valid parameter for 32-bit programs is FLAT (protected mode). The .MODEL directive also takes a second parameter to set the function-calling convention, which is discussed in CHAPTER 6.
.STACK	Sets the size of the stack memory segment for the program. The directive cannot be used without the .MODEL directive. While the default value is 1024, we recommend using 4096 to make stack the same size as a memory page in 32-bit Windows.

Program Sections

Table 3.7 Sample program sections

GAS	MASM	NASM

`.data`
`sum: .long 0`
 	
`.text`
`.globl _main`
`_main:`
`movl $25, %eax`
`movl $50, %ebx`
`addl %ebx, %eax`
`movl %eax, sum`
 	
`pushl $0`
`subl $4, %esp`
`movl $1, %eax`
`int $0x80`
`.end`
	`.386`
	`.MODEL FLAT, stdcall`
	`.STACK 4096`
	`ExitProcess PROTO,`
	`dwExitCode:DWORD`
	
	`.data`
	`sum DWORD 0`
	
	`.code`
	`_main PROC`
	`mov eax, 25`
	`mov ebx, 50`
	`add eax, ebx`
	`mov sum, eax`
	
	`INVOKE ExitProcess, 0`
	`_main ENDP`
	`END`
		`SECTION .data`
		`sum: DD 0`
		
		`SECTION .text`
		`global _main`
		`_main:`
		`mov eax, 25`
		`mov ebx, 50`
		`add eax, ebx`
		`mov DWORD [sum], eax`
		
		`mov eax, 1`
		`mov ebx, 0`
		`int 80h`

Table 3.6 Assembler-specific program sections

Directive			Description
GAS	MASM	NASM	
`.bss`	`.data`	`SECTION .bss`	Uninitialized variables
`.data`		`SECTION .data`	Initialized variables
`.text`	`.code`	`SECTION .text`	Executable code/instructions

Instructions

Table 3.8 Sample instructions

GAS	MASM	NASM

`.data`
`sum: .long 0`
 	
`.text`
`.globl _main`
`_main:`
`movl $25, %eax`
`movl $50, %ebx`
`addl %ebx, %eax`
`movl %eax, sum`
 	
`pushl $0`
`subl $4, %esp`
`movl $1, %eax`
`int $0x80`
`.end`
	`.386`
	`.MODEL FLAT, stdcall`
	`.STACK 4096`
	`ExitProcess PROTO,`
	`dwExitCode:DWORD`
	
	`.data`
	`sum DWORD 0`
	
	`.code`
	`_main PROC`
	`mov eax, 25`
	`mov ebx, 50`
	`add eax, ebx`
	`mov sum, eax`
	
	`INVOKE ExitProcess, 0`
	`_main ENDP`
	`END`
		`SECTION .data`
		`sum: DD 0`
		
		`SECTION .text`
		`global _main`
		`_main:`
		`mov eax, 25`
		`mov ebx, 50`
		`add eax, ebx`
		`mov DWORD [sum], eax`
		
		`mov eax, 1`
		`mov ebx, 0`
		`int 80h`

```
stc                 ; no operands, sets the carry flag
inc eax             ; 1 operand, increments eax by 1
mov eax, 5          ; 2 operands, moves literal value 5 into eax
imul eax, ebx, 5    ; 3 operands, multiplies literal value 5 and the
                    ; value in ebx and stores the result in eax
```

Literals (MASM/NASM)

Table 3.9 MASM/NASM integer radix characters

Radix	Base
b	Binary (base-2)
d	Decimal (base-10)
h	Hexadecimal (base-16)
q, o	Octal (base-8)

MASM/NASM

```
"A"
'A'
```

MASM/NASM

```
00011111b   ; b is the radix character for binary
31          ; decimal values do not need radix characters
31d         ; but you can specify d for decimal
1Fh         ; h is the radix character for hexadecimal
37o         ; o is the radix character for octal
```

MASM/NASM

```
0FFFF0342h   ; the actual value is FFFF0342 in hexadecimal
```

MASM/NASM

```
"I don't understand contractions."              ; strings that have one
'"Good job," said the father to his son.'       ; type of quotes on the
                                                ; outside and a different
                                                ; type on the inside
```

Literals (GAS)

GAS

```
.data
0b00011111
31
0x1F
037
```

```
.text
$0b00011111
$31
$0x1F
$037
```

GAS

```
.data
'A'
```

```
.text
$'A'
```

GAS

```
"This string \"contains\" double quotes!"
```

Table 3.10 GAS prefix and radix characters

.data Prefix	*.text* Prefix	Base
0b	$0b	Binary (base-2)
n/a	$	Decimal (base-10)
0x	$0x	Hexadecimal (base-16)
0	$0	Octal (base-8)

String Storage

String Characters	D	a	i	s	y	,		d	a	i	s	y
ASCII Decimal Values	68	97	105	115	121	44	32	100	97	105	115	121

Labels


```
userLoop:
            inc counter

otherLoop: inc counter2
```

Comments

GAS

```
# Moves the counter value into eax
movl counter, %eax
```

MASM/NASM

```
mov eax, counter ; Moves the counter value into eax
```

GAS

```
mov $10, %eax        /* This is a multi-line comment that is
                     partially on the same line as
                     an instruction and partially on
                     separate lines */
```

MASM

```
COMMENT !
     This is the section of the code where
     employee salaries are calculated. Note
     how the exclamation point is not in the
     text of the comment.
!
```

Data Types

Table 3.11 Default data type directives

Directive			Description
GAS	MASM	NASM	
.byte, .ascii	DB, BYTE	DB	1 byte (8-bit) integer
	SBYTE		1 byte (8-bit) signed integer
.word	DW, WORD	DW	2 byte (16-bit) integer
	SWORD		2 byte (16-bit) signed integer
.long	DD, DWORD	DD	4 byte (32-bit) integer
	SDWORD		4 byte (32-bit) signed integer
.quad	DQ, QWORD	DQ	8 byte (64-bit) integer
	DT, TBYTE	DT	10 byte (80-bit) integer
.octa			16 byte (128-bit) integer

Data Definition

GAS

```
val1:       .byte 17
valArray:   .long 0xFFFFFF, 0xFFFFFE, 0xFFFFD
```

MASM

```
charInput   BYTE 'A'
myArray     DWORD 41h, 75, 0C4h, 01010101b
```

NASM

```
counter:    DB 0
wageArray:  DD 75, 100, 125
```

Uninitialized Variables (MASM)

MASM

```
.data
num DWORD  6                  ; defines an initialized identifier
sum SDWORD ?                  ; defines an uninitialized identifier
myArray   BYTE 10 DUP (1)     ; defines an array of initialized bytes
myUArray  BYTE 10 DUP (?)     ; defines an array of uninitialized bytes
```

Uninitialized Variables (GAS/NASM)

Table 3.12 Uninitialized data type directives

Directive		Description
GAS	NASM	
.lcomm	RESB	Reserve a byte (8 bits)
	RESW	Reserve a word (16 bits)
	RESD	Reserve a double word (32 bits)
	RESQ	Reserve a quad word (64 bits)
	REST	Reserve a ten-byte (80 bits)

GAS

```
.bss
.lcomm sum, 2          # Reserve 2 bytes
.lcomm answer, 4       # Reserves 4 bytes
```

NASM

```
SECTION .bss
memAddr: RESD 1        ; Reserves 1 DWORD (4 bytes)
buffer:  RESB 64       ; Reserves 64 bytes
```

Strings

GAS

```
# The \n in the middle of the string adds a newline
motd: .ascii "The only way to win\nis not to play\0"
motd: .asciz "The only way to win\nis not to play"
```

MASM

```
; motd contains a single-line string
motd  BYTE "Welcome to Earth...",0

; motd2 contains a multi-line string with a newline at the end
motd2 BYTE "Thank you for using our system.",0Dh,0Ah
      BYTE "All of your activity will be monitored"
      BYTE "by our system administrators",0Dh,0Ah,0
```

NASM

```
; The 0Ah in the middle of the string adds a newline
motd: DB "How about a nice",0Ah,"game of chess",0
```

EQU

GAS

```
.equ identifier, expression     # numeric operations
.equ identifier, value          # a single value (constant)
.equ identifier, "symbol"       # another existing symbol
```

MASM

```
identifier EQU expression       ; numeric operations
identifier EQU symbol           ; another existing symbol
identifier EQU <text>           ; useful for non-integer data (float, string)
```

NASM

```
identifier: EQU expression      ; numeric operations
identifier: EQU value           ; a single value (constant)
identifier: EQU "text"          ; text can be four characters max (32-bit)
```

EQU (Array Length)

GAS

```
motd: .byte "Are you in the matrix?\0"
.equ len, (. - motd)
```

MASM

```
motd  BYTE "Are you in the matrix?",0
len = ($ - motd)
```

NASM

```
motd: DB "I'm in the matrix?",0
len:  EQU ($ - motd)
```

TEXTEQU (MASM)

MASM

```
identifier TEXTEQU %(expression)
identifier TEXTEQU textmacro
identifier TEXTEQU <text>
```

MASM

```
MULTIPLIER = 2
freq    TEXTEQU %(400 * MULTIPLIER)
movFreq TEXTEQU <mov eax, freq>
```

MASM

```
.code
movFreq
```

Key Terms

- array
  character literals
- comments
  current location counter
- directives
  DUP
  EQU
  identifers
- immediate
- instructions
  integer literals
  labels
  mnemonic