Kingdom of Saudi Arabia
Royal Commission at Yanbu
University College – Yanbu
Department of ACS & AIT
Yanbu Al-Sinaiyah

المملكة العربية السعودية
الهيئة الملكية ينبع
الكلية الجامعية-ينبع
ينبع الصناعية

2ND Semester 2007-08

CS-203 HANDOUT 4

ASSEMBLY LANGUAGE SYNTAX AND PROGRAM STRUCTURE

4.1 Introduction

A processor can directly execute a machine language program. Though it is possible to program directly in machine language, assembly language uses mnemonics to make programming easier.

An assembly language program uses mnemonics to represent:

1. Symbolic instructions and
2. The raw data that represent variables and constants.

A machine language program consists of:

1. A list of numbers representing the bytes of machine instructions to be executed
2. Data constants to be used by the program.

4.2 Assembly language syntax

An assembly language program consists of statements. The syntax of an assembly language program statement obeys the following rules:

- Only one statement is written per line
- Each statement is either an instruction or an assembler directive
- Each instruction has an op-code and possibly one or more operands
- An op-code is known as a mnemonic
- Each mnemonic represents a single machine instruction
- Operands provide the data to work with.

4.3 Program Statement:

The general format for an assembly language program statement is as follows:

```
Name mnemonic operand (destination), operand (source)  ; comment
```

4.3.1 Name Field:

This field is used for:

- Instruction label: if present, a label must be followed by a colon (**:**)
- Procedure names
- Variable names.

Examples of Name Fields

```
here:    MOV AX,0867H    ; Statement line with a label field
         JNC here        ; Statement with opcode and one operand
         INT 03H
SUM PROC NEAR            ; Procedure definition
```

4.3.2 Naming Conventions

Assembly language imposes some rules on how names are assigned to labels, variables, procedures and macros. It is the assembler's function to translate those names into memory addresses.
Note that naming conventions used in this course are related to the MASM 6.15 assembler used in the lab.

Here are the general rules on the use of names:

- A name is between 1 and 31 characters in length
- A name may include letters, numbers and special characters, such as **? . @ _ $ %**
- A name should not begin with a digit
- A name may begin with a letter or a special character
- If a period (**.**) is used, it must be the first character
- Names are not case sensitive.

Below are examples of legal and illegal names.

Examples of legal names

- COUNTER1
- @character
- SUM_TOTAL
- $1000
- Done?
- .TEST

Examples of illegal names

- TWO WORDS
- 2abc
- A45.28
- You&Me

- A+B

4.4 Operation Field:

- This field consists of a symbolic operation code, known as op-code
- The opcode describes the operation
- Symbolic op-codes are translated into machine language opcode.

4.4.1 Operand Field:

This field specifies data to be acted on. It may have one, two or no operands at all.

Examples of instructions with different operand fields

```
NOP       ; Instruction with no operand field
INC AX    ; Instruction with one operand field
ADD AX, 2 ; Instruction with two operand field
```

4.4.2 8086 Instruction Types:

There exist around 150 instructions for the 8086 processor. These instructions may be classified into different classes. The following table summarizes the different classes of instructions, together with examples of each class.

Instruction Type	Definiton	Examples
Data transfer instructions	Transfer information between registers and memory locations or I/O ports	MOV, XCHG, LEA,PUSH,POP
Arithmetic instructions	Perform arithmetic operations on binary or binary-coded-decimal (BCD) numbers	ADD, SUB, INC, DEC
Bit manipulation instructions	Perform shift, rotate, and logical operations on memory locations and registers	SHL, SHR, SAR, ROL
Control transfer instructions	Control sequence of program execution. Include jumps and procedure transfers	JL, JE, JNE, JGE
String handling instructions	Move, compare, and scan strings of data	OVSB, MOVSW, CMPS, CMPSB
Processor control instructions	Set and clear status flags, and change the processor execution state	STC, STD, STI, CLC
Interrupt instructions	Interrupt processor to service a specific condition	INT, INTO, IRET
Miscellaneous instructions		NOP, WAIT

Table 2: 8086 Instructions

4.5 Comment Field:

- A semicolon marks the beginning of a comment
- A semicolon in the beginning of a line makes it all a comment line
- Good programming practice dictates the use of a comment on almost every line.

4.5.1 Key rules for the use of comments:

- Do not say something that is obvious
- Put instruction in context of program

Examples of good and bad Comments

```
        MOV CX, 0  ; Move 0 to CX. (This is not a good
comment.)
        MOV CX, 0  ; CX counts terms, initially set to 0.
(This is a good comment.)
```

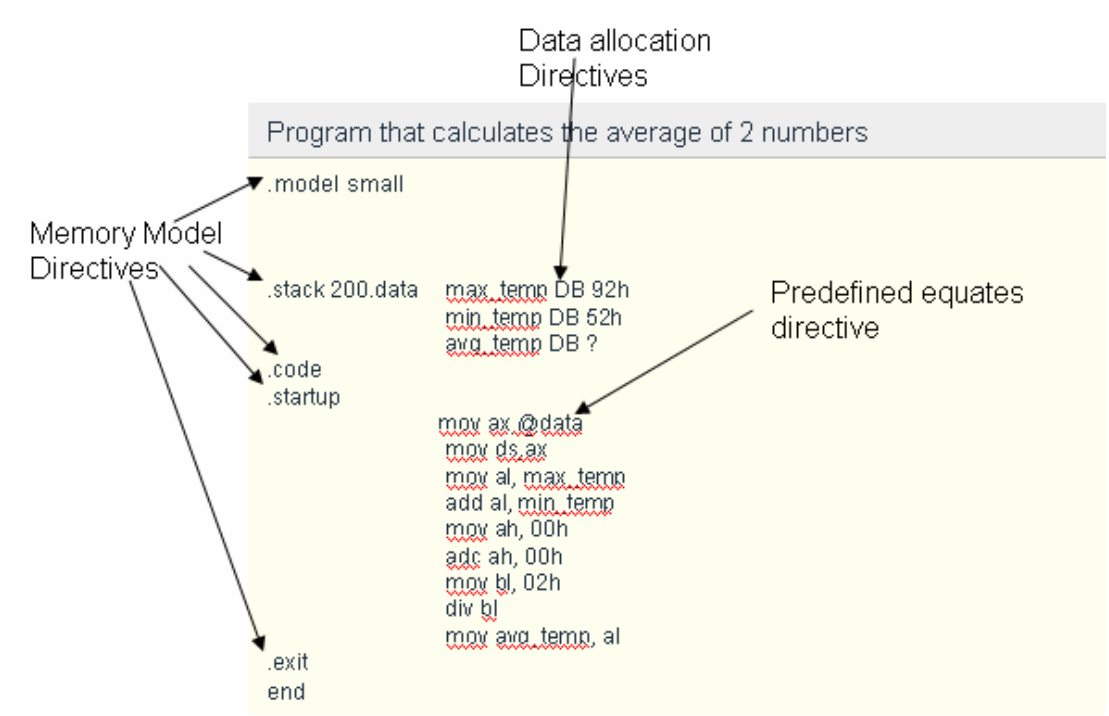
4.6 Assembler Directives:

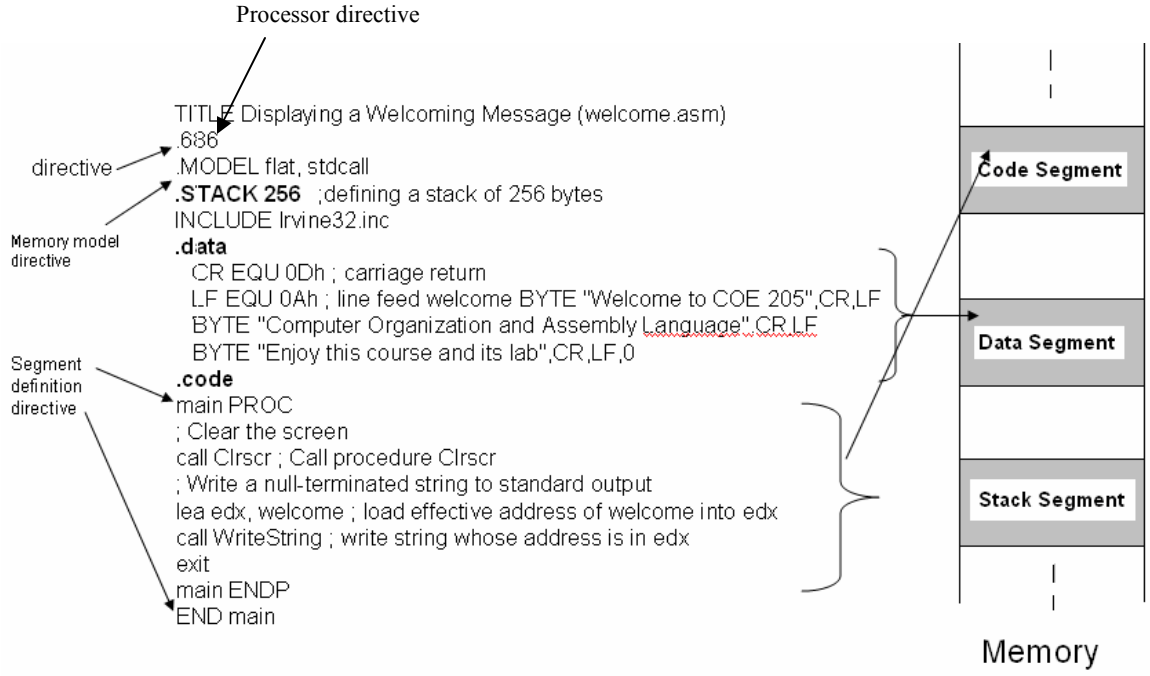
Assembler directives are instructions that are directed to the assembler. Assembler directives affect the generated machine code, but are not translated directly into machine code. Directives can be used to declare variables, constants, segments, macros, and procedures.

In general, a directive:

- tells the assembler to do a specific thing, and
- is not translated into machine code.

Memory Model	Segment Definition	Data Allocation	Predefined equates	Radix Specifiers	Macro Definition
.MODEL	name PROC [NEAR|FAR]	[name] DB init[,init...]	@code	.RADIX expr	name MACRO [parm[,parm...]]
.CODE	name ENDP	[name] DW init[,init...]	@codesize	.RADIX B	ENDM
.DATA	PUBLIC name[,name...]	[name] DD init[,init...]	@datasize	.RADIX Q	LOCAL name[,name...]
.STACK [size]	END [name]	[name] DF init[,init...]	@data	.RADIX O	EXITM
		[name] DQ init[,init...]	@stack	.RADIX D	REPT expr
				.RADIX H	





4.6.1 Title directive:

The title directive is optional and specifies the title of the program. Like a comment, it has no effect on the program. It is just used to make the program easier to understand.

4.6.2 The Model directive:

The model directive specifies the total amount of memory the program would take. In other words, it gives information on how much memory the assembler would allocate for the program. This depends on the size of the data and the size of the program or code.

4.6.3 Segment directives:

Segments are declared using directives. The following directives are used to specify the following segments:

- stack
- data
- code

4.6.3.1 Stack Segment:

- Used to set aside storage for the stack
- Stack addresses are computed as offsets into this segment
- Use: .stack followed by a value that indicates the size of the stack

4.6.3.2 Data Segments:

- Used to set aside storage for variables.
- Constants are defined within this segment in the program source.
- Variable addresses are computed as offsets from the start of this segment
- Use: .data followed by declarations of variables or definitions of constants.

4.6.3.3 Code Segment:

The code segment contains executable instructions macros and calls to procedures.
Use: .code followed by a sequence of program statements.

Example:

Here is how the "hello world" program would look like in assembly:

The hello world program in Assembly

```
.model small
.stack 200

.data
        greeting db 'hello world !',13,10,'$'

.code
mov ax,@data
mov ds,ax
mov ah,9
mov dx,offset greeting
int 21h
mov ah,4ch
```

```
int 21h
end
```

4.6.4 Memory Models

The memory model specifies the memory size assigned to each of the different parts or segments of a program. There exist different memory models for the 8086 processor The .MODEL Directive

The memory model directive specifies the size of the memory the program needs. Based on this directive, the assembler assigns the required amount of memory to data and code.

Each one of the segments (stack, data and code), in a program, is called a *logical segment* . Depending on the model used, segments may be in one or in different physical segments.

In MASM 6.X, segments are declared using the .MODEL directive. This directive is placed at the very beginning of the program, or after the optional title directive.

MODEL directive

.MODEL memory_model
Where memory_model can be:

- TINY
- SMALL
- COMPACT
- MEDIUM
- LARGE or
- HUGE.

Memory Model	Size of Code	Size of Data	Segments
TINY	Code + Data <= 64KB		Code & data combined
SMALL	Less <= 64KB	Less <= 64KB	1 code seg , 1 data seg
MEDIUM	Any size	Less <= 64 KB	Many code segs, 1 data seg
COMPACT	Less <= 64KB	Any size	1 code seg, many data segs
LARGE*	Any size	Any size	Many code and data segs
HUGE**	Any size	Any size	Many code and data segs

Table 1: Memory Models

(*) For the LARGE model, the largest arrays size can not exceed 64 KB.(br) (**)For the HUGE model, an array may have a size greater than 64 KB and hence can span more than one physical segment.

4.6.5 Use of memory models:

The amount of data that has to be manipulated and code that needs to be written are the major factors in determining the choice of an appropriate model. The following are guidelines to help chose the right model for a program.

For a small fast program that operates on small quantities of data, the SMALL or TINY models are the most suitable ones. These models allow up to 64K of memory (i.e. one single physical segment), but the executable code is fast. The only difference between these two models is that the TINY model generates a .COM module in which far references cannot be used, whereas the SMALL model generates a .exe module.

For very long programs that require more than one code segment and operate on large amounts of data which would require more than one data segment, the LARGE and HUGE models are most appropriate.

Structure of Assembly Language Program

4.7 Program Structure:

A program has always the following general structure:

Structure of an Assembly Language Program

```
.model small            ;Select a memory model

.stack stack size       ;Define the stack size

.data

                  ; Variable and array  declarations
                  ; Declare variables at this level
.code
main proc

          ; Write the program main code at this level

main endp


;Other Procedures

                        ; Always organize your program
                        ; into procedures


end main       ; To mark the end of the source file
```

4.8 Instruction Semantics:

The following rules have to be strictly followed in order to write correct code.

1 - Both operands have to be of the same size:

Instruction	Correct	Reason
MOV AX, BL	No	Operands of different sizes
MOV AL, BL	Yes	Operands of same sizes
MOV AH, BL	Yes	Operands of same sizes
MOV BL, CX	No	Operands of different sizes

Table 3: Register-Register Transfer Instructions

2 - Both operands cannot be memory operands simultaneously:

Instruction	Correct	Reason
MOV i , j	No	Both operands are memory variables
MOV AL, i	Yes	Move memory variable to register
MOV j, CL	Yes	Move register to memory variable

Table 4: Memory-Register Transfer Instructions

3 - First operand, or destination, cannot be an immediate value:

Instruction	Correct	Reason
ADD 2, AX	No	Move register to constant
ADD AX, 2	yes	Move constant to register

Table 5: Constant to Register Transfer Instructions

Example Test Questions

Fill in the blanks

1. TINY Model generates _____ file
2. DW directive specifies _____ bytes in the memory.
3. Each instruction has an op-code and possibly one or more _____.

Specify True or False

1. Multiple statements can be written in one line in assembly language (True/False)
2. Op-code is also known as mnemonic (True/False)
3. 6absdfc is a correct and legal label name (True/False)

Choose the best answer.

The data segment holds the following

a. All coding
b. Data variables and stack variables
c. Variables and constants
d. None of above

The .MODEL represents

a. Type of program
b. Memory model of program
c. Data model of program
d. Code model of program

Question: What is the difference between the LARGE and HUGE memory model.