4. Assembly Language Concepts
Definitions

*instruction* - statement translated by the assembler into machine language.

*directive* - statement which give directions to the assembler during the assembly process but are not translated into machine instructions.

*mnemonics* - acronyms that represent machine language instructions; used to facilitate program development.

*identifiers* - acronyms that represent addresses and perhaps data.
## Data Definition

<table>
<thead>
<tr>
<th>Assembly Construct</th>
<th>Variable Defined</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB</td>
<td>Byte (8-bit) variable</td>
</tr>
<tr>
<td>DW</td>
<td>Word (16-bit) variables</td>
</tr>
<tr>
<td>DD</td>
<td>Double Word (32-bit) variable</td>
</tr>
</tbody>
</table>

## Numeric Constants

<table>
<thead>
<tr>
<th>Base</th>
<th>Suffix</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Binary</td>
<td>B</td>
<td>10101000b, 1B</td>
</tr>
<tr>
<td>Decimal*</td>
<td>D</td>
<td>10; 123d</td>
</tr>
<tr>
<td>Octal</td>
<td>O</td>
<td>5O; 0o</td>
</tr>
<tr>
<td>Hexadecimal**</td>
<td>H</td>
<td>5a5ah, 0xffffffffh</td>
</tr>
</tbody>
</table>

* Default
** Numbers staring with characters ‘A’ - ‘F’ should be preceded by a ‘0’.
String Constants

String constants should be enclosed in single quotes ('). Examples: 'J', '5', 'Hello'

Valid Identifiers

Identify program entities such as variables, labels and constants defined with an EQU. A legal identifier:

• Can contain letters (A-Z, a-z), digits (0-9) or special characters as: ?, @, _.

• Must begin with a letter or special character.

• Can be up to 255 characters (unique for the 1st 31).

Example:

```
magic_num equ 12345678H

.Data
name db 'John Doe',0
initials db 'J','D'
_serial_num dd magic_num
num_1285 dw 0505h
```
Assembler Directives: Defining a Program

The assembler directives serve for the purpose of defining a program and make it easier to read.

Directives (syntax) could vary between different Assemblers. Both Microsoft *masm* and Borland *tasm* assemblers support a simplified set of directives that include:

```
.MODEL specifies the memory model to be used during assembly

.STACK, .DATA & .CODE define the stack, data & code segments, respectively.

END   specifies the end of the assembly process. On the main module (or if a single module program), it contains a label identifier that indicates the execution start point.
```
Assembler Directives: Making a program easier to read

The following directives are used to make the code easier to read and maintain it:

*OFFSET* calculates the offset (effective address) and places it into the instruction.

*PTR* overwrites the type of definition or to give a type of operand.

*EQU* provides a mean of defining symbols for many constructs.
EXAMPLE (directives shown in bold):

CR  equ  0dh
LF  equ  0ah

.model small
.stack 200h
.data
msg      db   'Hello World',CR,LF,'$'
.code
start:   mov ax,@data
         mov ds,ax
         mov bx,offset msg
         mov byte ptr [bx],'h'
         mov dx,bx
         mov ah,9
         int 21h
exit:    mov ax,4c00h
         int 21h
end    start
Instruction Format

Label:   Mnemonic  Operands       ;Comments

• **Label** is an optional unique identifier that defines
  the memory location where the instruction can be
  found.

• **Mnemonic** is a valid CPU or FPU instruction.

• **Operands** are the arguments of the instruction.
  Depending on the instruction there could be 0, 1 or
  2 operands. If there are 2, the left-side Operand is
  the destination and right-side Operand is the source.

• **Comment** is an optional field used by the
  programmer to document the code.

• Example:

    read:   cli       ;label & no operand
            inc ax    ;single operand
            mov bx,ax ;2 operands: source & destination
The Development Process

1. Use ASCII Editor (e.g. Notepad++) to create or modify .ASM File.

2. Use Assembler (e.g. TASM) to generate .OBJ & .LST Files.

3. Verify .LST File to evaluate syntax errors.

4. Use Linker (e.g. TLINK) to generate .EXE File.

5. Run the .EXE File to test it and use Debugger (e.g. TD) to evaluate logical errors.