CSE2006
Microprocessor & Interfacing

Module – 2
Introduction to ALP

Dr. E. Konguvel
Assistant Professor (Sr. Gr. 1),
Dept. of Embedded Technology,
School of Electronics Engineering (SENSE),
konguvel.e@vit.ac.in
9597812810

VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

Module 2: Introduction to ALP

- Introduction
- ALP Development Tools
  - Editor
  - Assembler
  - Library Builder
  - Linker
  - Debugger
  - Simulator
  - Emulator
- **Assembler Directives**
- ALP Programs
  - Arithmetic Operations
  - Number System Conversions
  - Programs using Loops
  - If then else
  - For loop structures

Assembler Directives

Variables in Assembler Directives

- Symbols (or Terms) in ALP statements to represent the variable data and address.

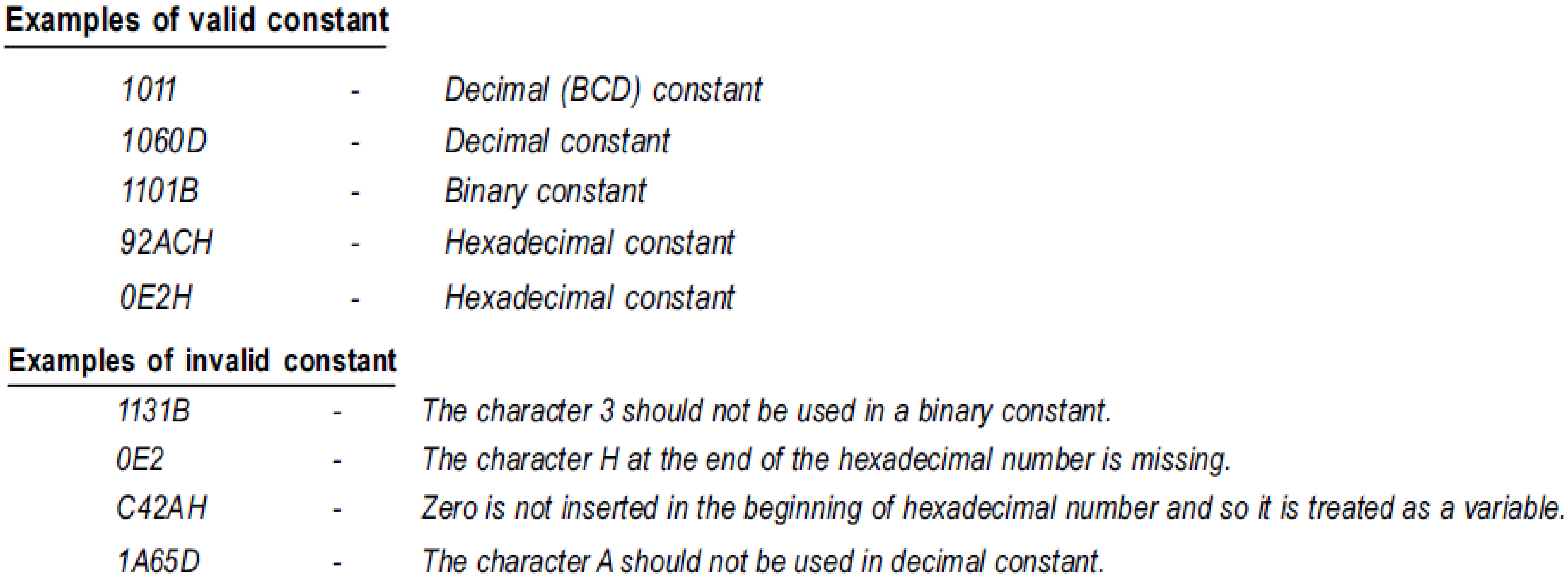
- A value has to be attached to each variable in the program, that can be varied while running the program.

- Rules:
  1. Can have the characters: A to Z, a to z, 0 to 9, @, _ (underscore).
  2. First character should be A - Z or a - z or _.
  3. Length depends on assembler and generally maximum length is 32 characters.
  4. Variables are case insensitive.

Assembler Directives

Constants in Assembler Directives

- Decimal, Binary or Hexadecimal numbers used to represent the data or address in ALP statements are called constants.
- Their values are fixed and cannot be changed while running a program.



Examples of valid constant

1011	-	*Decimal (BCD) constant*
1060D	-	*Decimal constant*
1101B	-	*Binary constant*
92ACH	-	*Hexadecimal constant*
0E2H	-	*Hexadecimal constant*

Examples of invalid constant

1131B	-	*The character 3 should not be used in a binary constant.*
0E2	-	*The character H at the end of the hexadecimal number is missing.*
C42AH	-	*Zero is not inserted in the beginning of hexadecimal number and so it is treated as a variable.*
1A65D	-	*The character A should not be used in decimal constant.*

Assembler Directives

Assembler Directives

- Instructions to the assembler regarding the program being assembled – Pseudo-instructions.

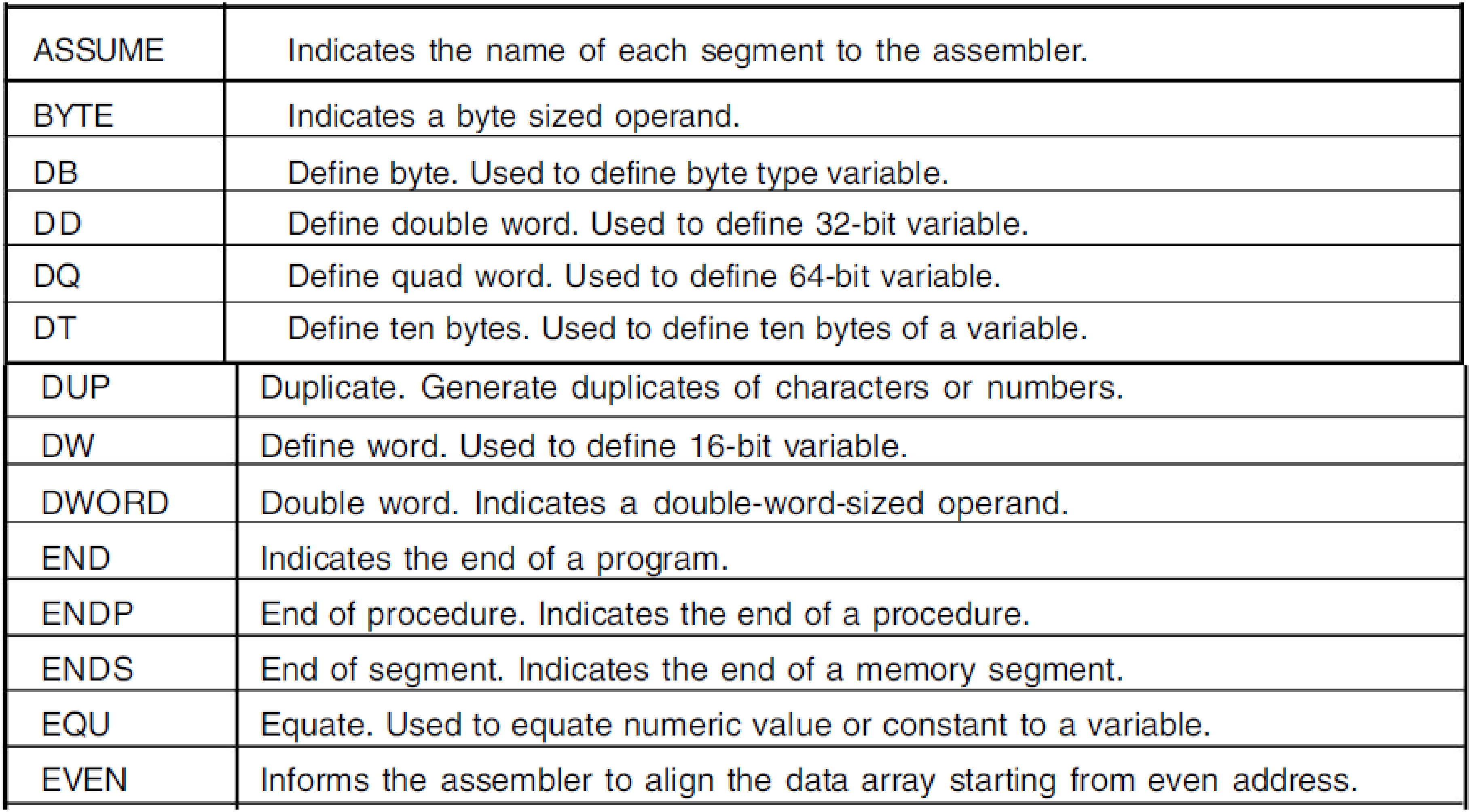
- Used to specify start and end of a program, attach value to variables, allocate storage locations to input/output data, to define start and end of segments, procedures, macros, etc.

- Control the generation of machine code and organization of the program.

- No machine codes are generated for assembler directives.

Assembler Directives

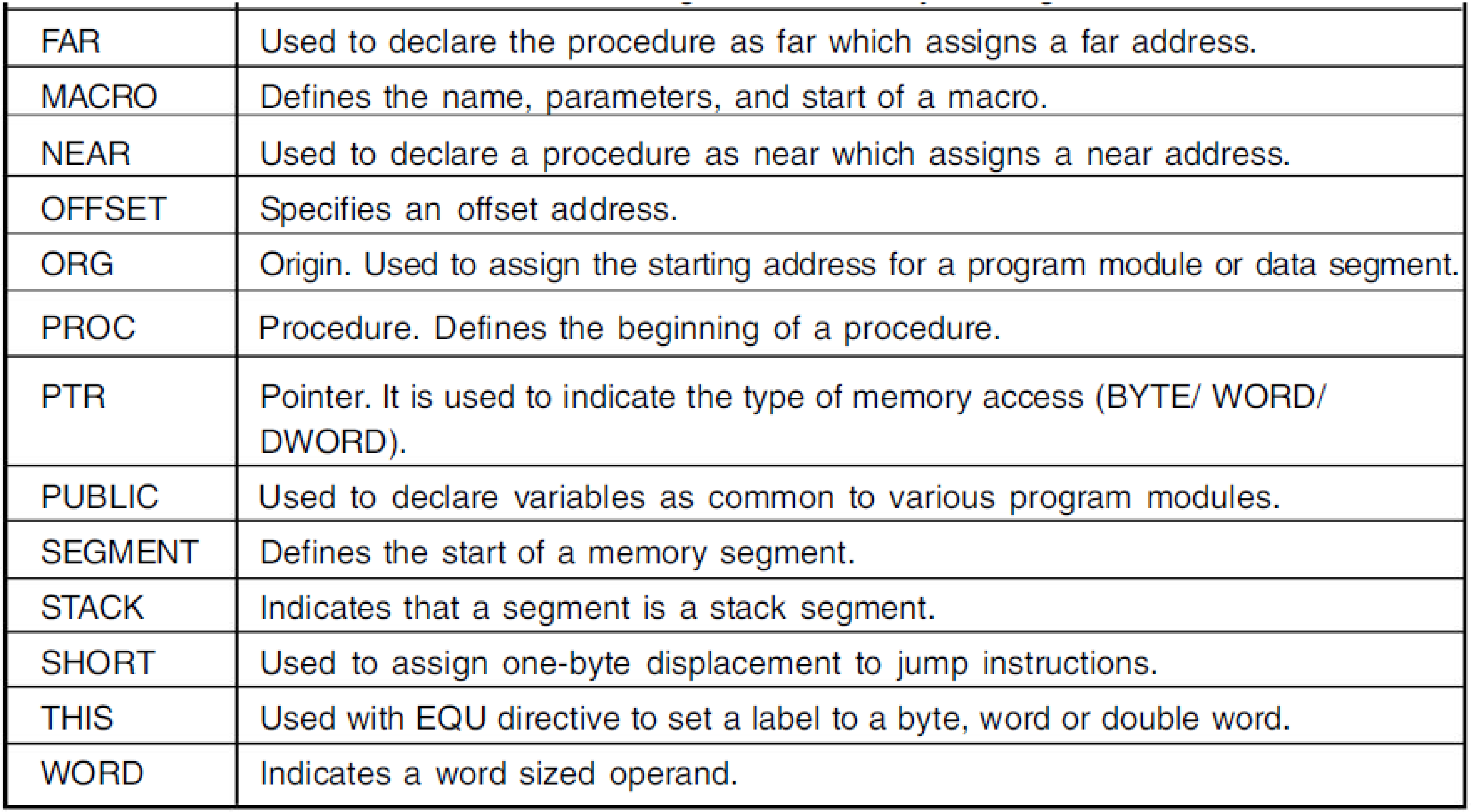
Assembler Directives (1/2)



ASSUME	Indicates the name of each segment to the assembler.
BYTE	Indicates a byte sized operand.
DB	Define byte. Used to define byte type variable.
DD	Define double word. Used to define 32-bit variable.
DQ	Define quad word. Used to define 64-bit variable.
DT	Define ten bytes. Used to define ten bytes of a variable.
DUP	Duplicate. Generate duplicates of characters or numbers.
DW	Define word. Used to define 16-bit variable.
DWORD	Double word. Indicates a double-word-sized operand.
END	Indicates the end of a program.
ENDP	End of procedure. Indicates the end of a procedure.
ENDS	End of segment. Indicates the end of a memory segment.
EQU	Equate. Used to equate numeric value or constant to a variable.
EVEN	Informs the assembler to align the data array starting from even address.

Assembler Directives

Assembler Directives (2/2)

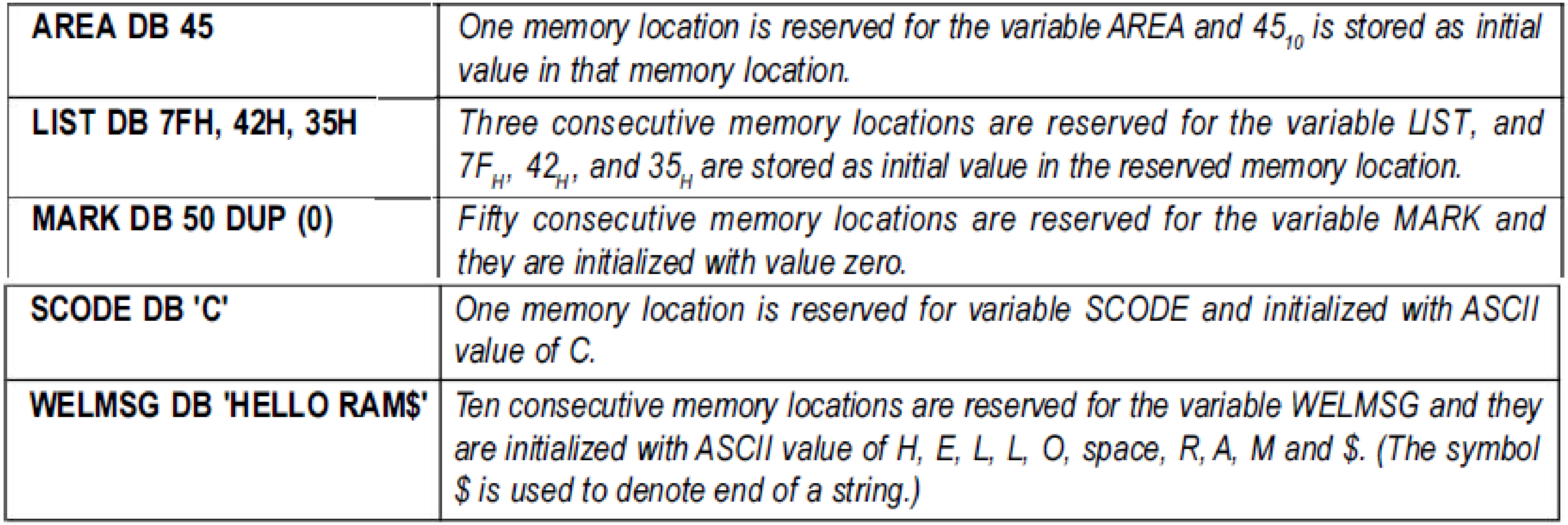


FAR	Used to declare the procedure as far which assigns a far address.
MACRO	Defines the name, parameters, and start of a macro.
NEAR	Used to declare a procedure as near which assigns a near address.
OFFSET	Specifies an offset address.
ORG	Origin. Used to assign the starting address for a program module or data segment.
PROC	Procedure. Defines the beginning of a procedure.
PTR	Pointer. It is used to indicate the type of memory access (BYTE/ WORD/ DWORD).
PUBLIC	Used to declare variables as common to various program modules.
SEGMENT	Defines the start of a memory segment.
STACK	Indicates that a segment is a stack segment.
SHORT	Used to assign one-byte displacement to jump instructions.
THIS	Used with EQU directive to set a label to a byte, word or double word.
WORD	Indicates a word sized operand.

Assembler Directives

DB (Define Byte)

- To define a byte type variable, reserves specific amount of memory to variables and stores the values specified in the statement as initial values in the allotted memory locations.
- Range: 0 to 255 (00_H to FF_H) for unsigned value, and -128 to 127 for signed value (00_H to $7F_H$ for positive values and 80_H to FF_H for negative values).

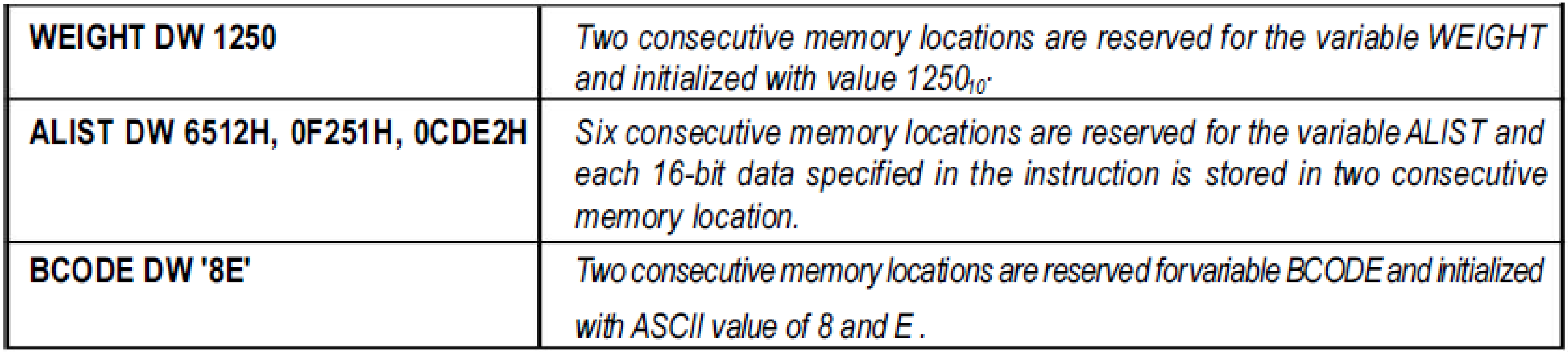


AREA DB 45	*One memory location is reserved for the variable AREA and 45_{10} is stored as initial value in that memory location.*
LIST DB 7FH, 42H, 35H	*Three consecutive memory locations are reserved for the variable LIST, and $7F_H$, 42_H, and 35_H are stored as initial value in the reserved memory location.*
MARK DB 50 DUP (0)	*Fifty consecutive memory locations are reserved for the variable MARK and they are initialized with value zero.*
SCODE DB 'C'	*One memory location is reserved for variable SCODE and initialized with ASCII value of C.*
WELMSG DB 'HELLO RAM$'	*Ten consecutive memory locations are reserved for the variable WELMSG and they are initialized with ASCII value of H, E, L, L, O, space, R, A, M and $. (The symbol $ is used to denote end of a string.)*

Assembler Directives

DW (Define Word)

- To define word type (16-bit) variable, reserves two consecutive memory locations to each variable and store the 16-bit values specified in the statement as initial value in the allotted memory locations.
- Range: 0 to 65,535 (0000_H to $FFFF_H$) for unsigned value, -32,768 to +32,767 for signed value (0000_H to $7FFF_H$ for positive value and 8000_H to $FFFF_H$ for negative value).

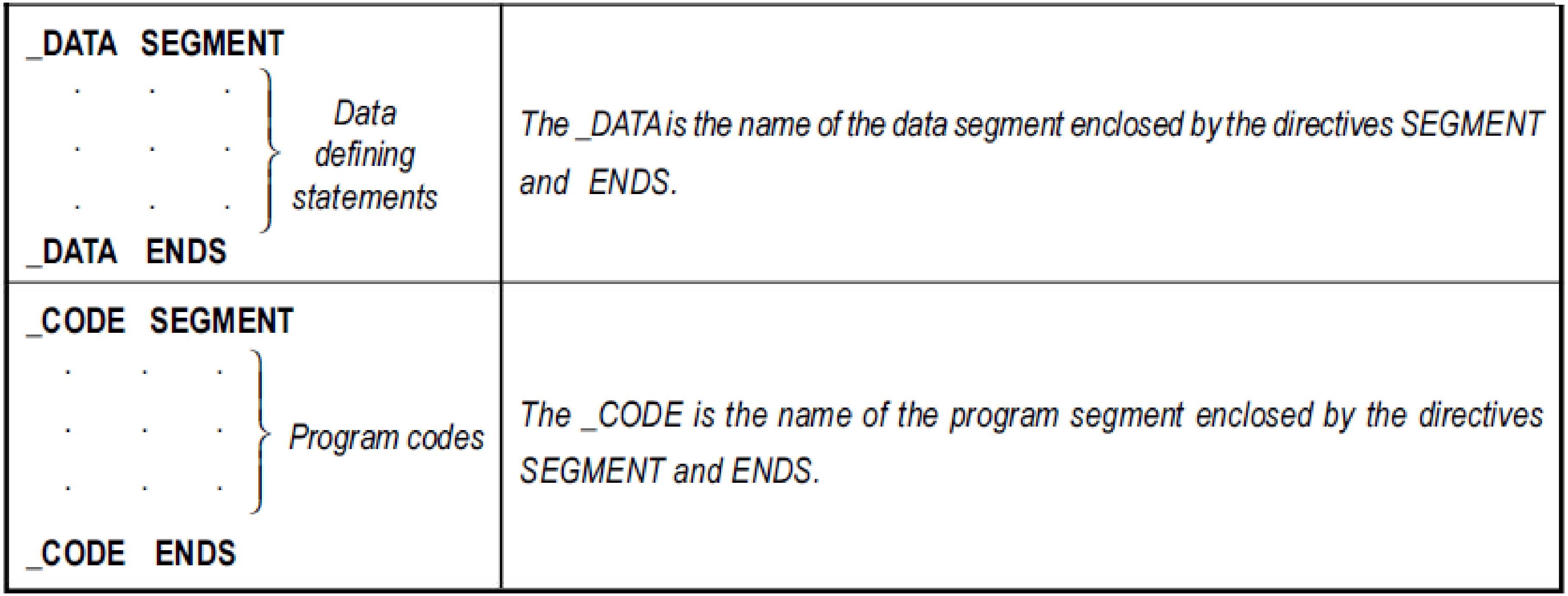


WEIGHT DW 1250	*Two consecutive memory locations are reserved for the variable WEIGHT and initialized with value 1250_{10}.*
ALIST DW 6512H, 0F251H, 0CDE2H	*Six consecutive memory locations are reserved for the variable ALIST and each 16-bit data specified in the instruction is stored in two consecutive memory location.*
BCODE DW '8E'	*Two consecutive memory locations are reserved forvariable BCODE and initialized with ASCII value of 8 and E .*

Assembler Directives

SEGMENT & ENDS (End of Segment)

- SEGMENT is used to indicate the beginning of a code/data/stack segment.
- ENDS is used to indicate the end of a code/data/stack segment.

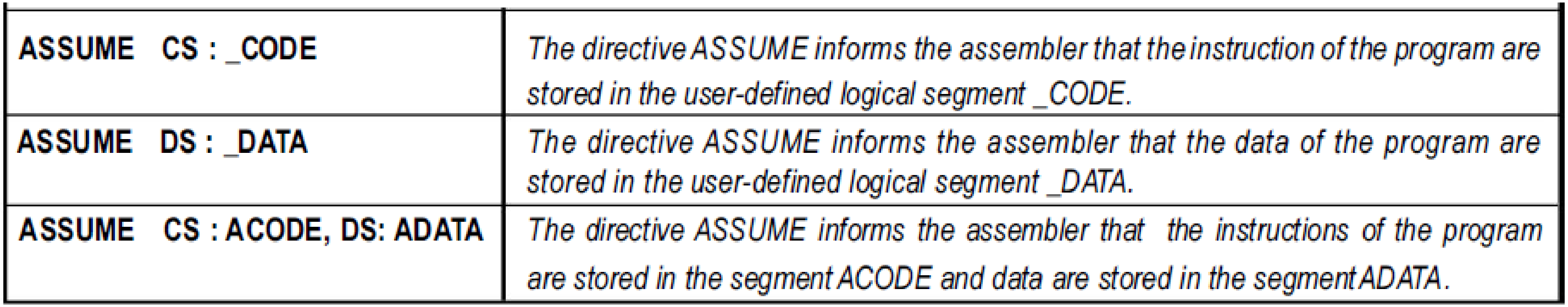


_DATA SEGMENT . . . . _DATA ENDS Data defining statements	The _DATA is the name of the data segment enclosed by the directives SEGMENT and ENDS.
_CODE SEGMENT . . . . _CODE ENDS Program codes	The _CODE is the name of the program segment enclosed by the directives SEGMENT and ENDS.

Assembler Directives

ASSUME

- ASSUME informs the assembler the name of the program/data segment that should be used for a specified segment.
- The segment register can be any of the CS, SS, DS and ES registers and segment name can be any valid assembler variable.



ASSUME CS : _CODE	*The directive ASSUME informs the assembler that the instruction of the program are stored in the user-defined logical segment _CODE.*
ASSUME DS : _DATA	*The directive ASSUME informs the assembler that the data of the program are stored in the user-defined logical segment _DATA.*
ASSUME CS : ACODE, DS: ADATA	*The directive ASSUME informs the assembler that the instructions of the program are stored in the segment ACODE and data are stored in the segment ADATA.*

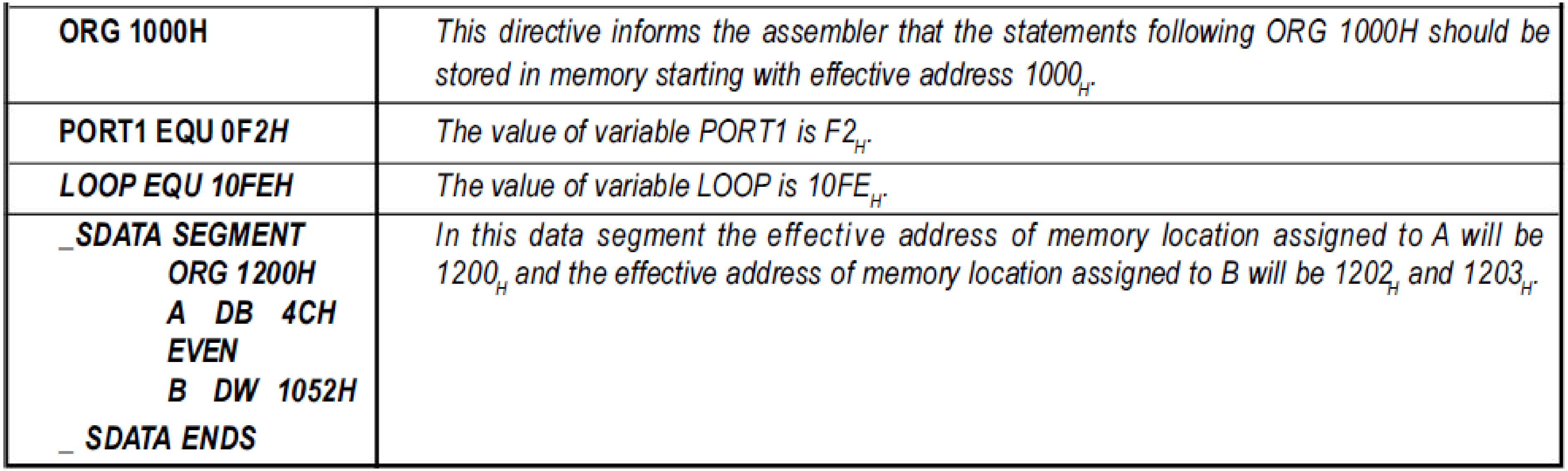
Assembler Directives

ORG, END, EVEN & EQU

- ORG (Origin) is used to assign the starting address (effective address) for a program/data segment.
- END is used to terminate a program. Statements after END will be ignored.
- EVEN will inform the assembler to store the program/data segment starting from an even address.
  - 8086 requires one bus cycle to access a word at even address and two bus cycles to access a word at odd address.
  - The even alignment with EVEN directive helps in accessing a series of consecutive memory words quickly.
- EQU (Equate) is used to attach a value to a variable.

Assembler Directives

ORG, END, EVEN & EQU

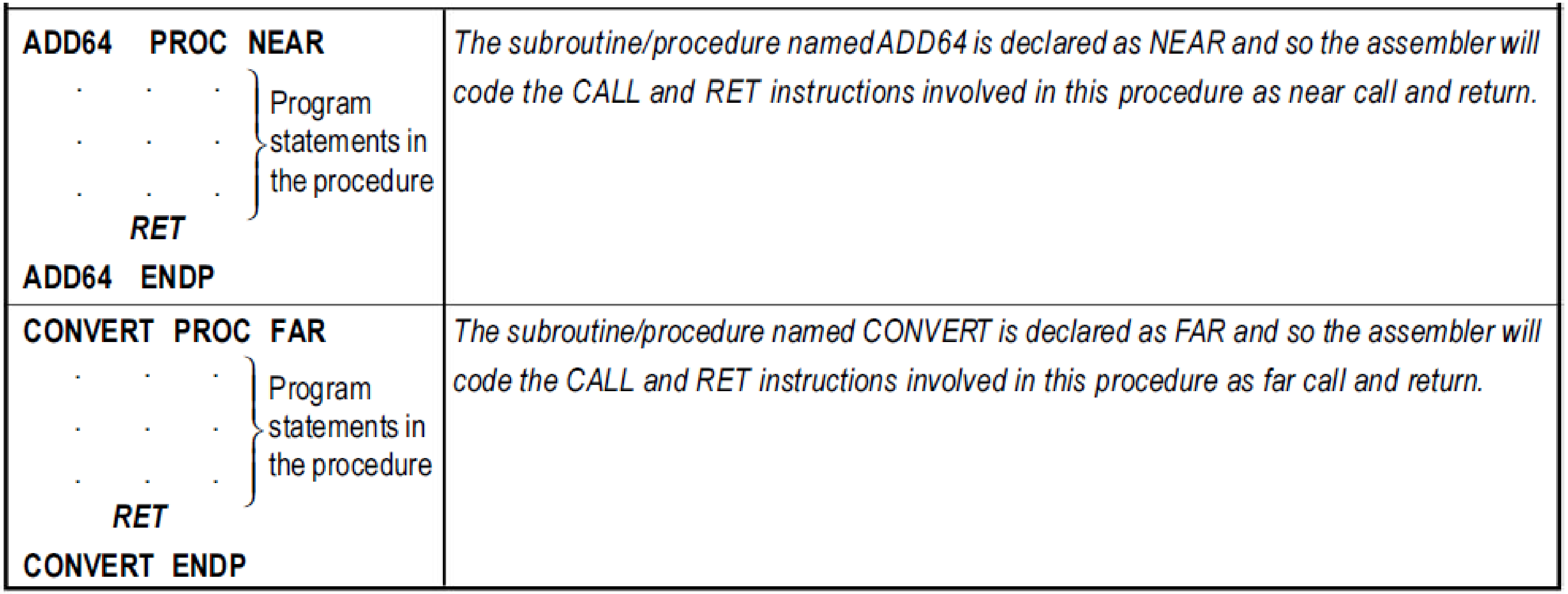


ORG 1000H	*This directive informs the assembler that the statements following ORG 1000H should be stored in memory starting with effective address 1000_H.*
PORT1 EQU 0F2H	*The value of variable PORT1 is $F2_H$.*
LOOP EQU 10FEH	*The value of variable LOOP is $10FE_H$.*
SDATA SEGMENT *ORG 1200H* *A DB 4CH* *EVEN* *B DW 1052H* * SDATA ENDS*	*In this data segment the effective address of memory location assigned to A will be 1200_H and the effective address of memory location assigned to B will be 1202_H and 1203_H.*

Assembler Directives

PROC, FAR, NEAR & ENDP

- PROC, FAR, NEAR and ENDP are used to define a procedure/subroutine.
- PROC & ENDP indicates beginning and end of a procedure.
- FAR or NEAR, are type specifiers (Optional - near), to differentiate intrasegment call and intersegment call.

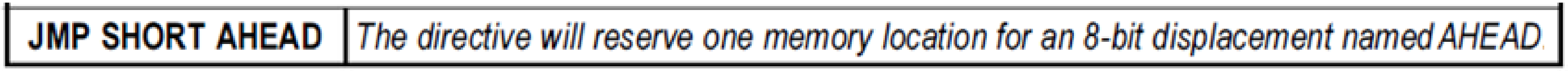


ADD64 PROC NEAR . . . } Program . . . } statements in . . . } the procedure ***RET*** **ADD64 ENDP**	*The subroutine/procedure namedADD64 is declared as NEAR and so the assembler will code the CALL and RET instructions involved in this procedure as near call and return.*
CONVERT PROC FAR . . . } Program . . . } statements in . . . } the procedure ***RET*** **CONVERT ENDP**	*The subroutine/procedure named CONVERT is declared as FAR and so the assembler will code the CALL and RET instructions involved in this procedure as far call and return.*

Assembler Directives

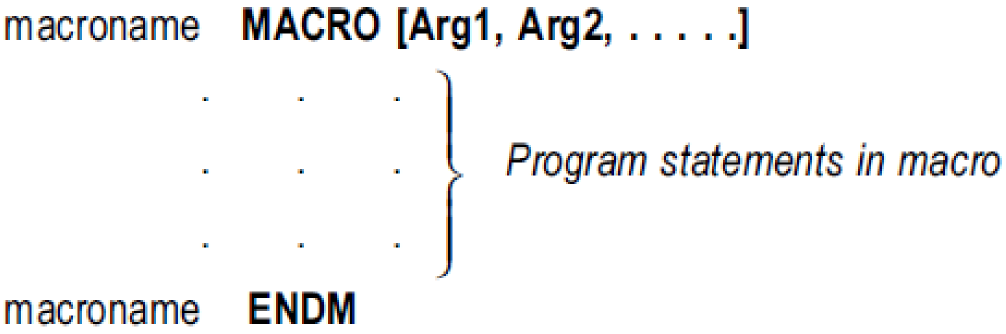
SHORT, MACRO & ENDM

- SHORT is used to reserve one memory location for 8-bit signed displacement in jump instructions.



JMP SHORT AHEAD	*The directive will reserve one memory location for an 8-bit displacement named AHEAD*

- MACRO and ENDM are used to indicate beginning and end of a macro, encloses the definitions, declarations and program statements which are to be substituted at the invocation of a macro.



```
macroname   MACRO [Arg1, Arg2, . . . . .]
              .    .    .
              .    .    .     Program statements in macro
              .    .    .
macroname   ENDM
```

Assembler Directives

Procedures & Macros

- When a group of instructions are to be used several times to perform a same function in a program called ***procedure or subroutine***.
- When a procedure is called in the main program, the program control is transferred to procedure and after executing the procedure the program control is transferred back to the main program.
- CALL is used to call a procedure in the main program and the instruction RET is used to return the control to the main program.
- Advantage: The machine codes for the group of instructions in the procedure has to be put in memory only once.
- Disadvantages: Need for a stack, and the overhead time required to call the procedure and return to the calling program.

Assembler Directives

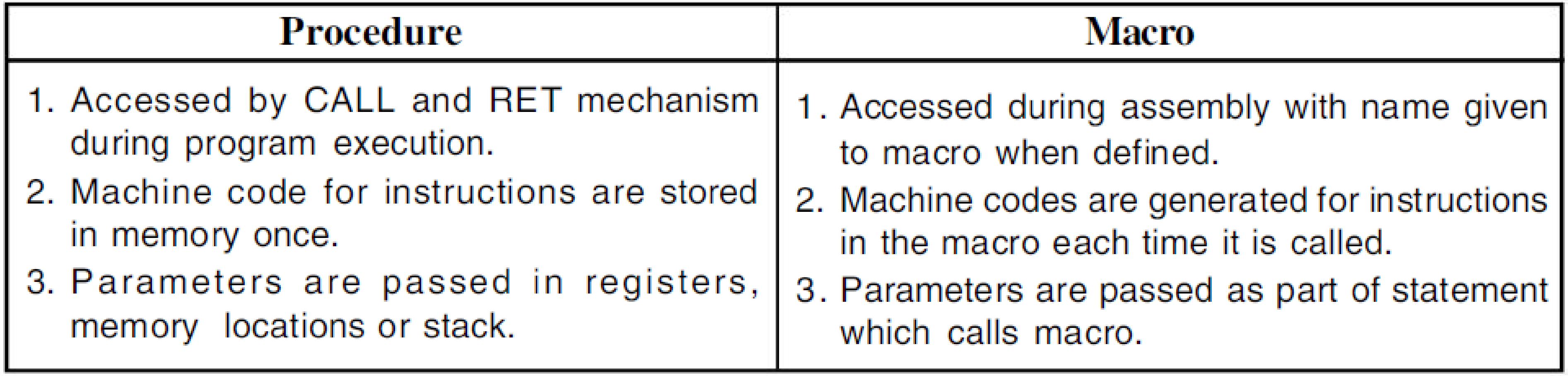
Procedures & Macros

- When a group of instructions are to be used several times to perform a same function in a program and they are too small to be written as a procedure, they are ***macros***.
- Open Subroutines: whenever a macro is called in a program, the assembler will insert the defined group of instructions in place of the call.
- Macros are identified by their name and usually defined at the start of a program.
- Process of replacing the macro with the instructions is called expanding the macro.
- Advantages: Avoiding the overhead time involved in calling and returning from a procedure.

Assembler Directives

Procedures & Macros

- Disadvantage: Program may take up more memory due to insertion of the machine codes in the program at the place of macros.
- Macros should be used only when its body has a few program statements.



Procedure	Macro
1. Accessed by CALL and RET mechanism during program execution.	1. Accessed during assembly with name given to macro when defined.
2. Machine code for instructions are stored in memory once.	2. Machine codes are generated for instructions in the macro each time it is called.
3. Parameters are passed in registers, memory locations or stack.	3. Parameters are passed as part of statement which calls macro.