ECE 33: PRINCIPLES OF COMPUTER ENGINEERING

Meghanad D. Wagh

8085 Processor architecture

Intel 8085 microprocessor is a general purpose microprocessor that has many interesting features. It can address up to 2^{16} distinct locations of an external memory using 16 bit addresses. The 16 lines that carry the address from the processor to the memory are together known as the *Address Bus.* Each memory location contains 8 bits of binary data. By activating either the $\overline{RD}$ or the $\overline{WR}$ signals to the memory, the processor can read from or write to the memory location addressed. The eight line that carry the data between the processor and the memory are together known as the *Data Bus.*

Internally, 8085 has seven data registers: A (also called the accumulator), B, C, D, E, H and L. Each of these registers can hold a binary string of length 8 bits (8 bits = 1 byte). Accumulator is by far the most important register of this processor since for every binary arithmetic operation, one of the operands is taken from this register and the result of the operation always goes to this register. The rest of the six registers are grouped into three pairs: B-C, D-E and H-L. A register pair is referred to by the name of the first register in the pair. Thus register pair "H" refers to the pair H-L. Clearly a register pair holds a 16 bit data string.

A register is provided to record the logical conditions (*flags*) generated by the arithmetic or logical operations such as additions or subtractions. The specific conditions recorded are the following:

- **The Zero Flag.** Records if the arithmetic/logical operation produced a zero result.
- **The Carry Flag.** Records if the arithmetic/logical operation produced a carry (i.e., an extra bit beyond the standard 8 bits).
- **The Sign Flag.** Records the sign bit of the result indicating if the result was negative (sign bit 1) or non-negative (sign bit 0).
- **The Parity Flag.** Records whether the number of 1's in the result is even or odd.
- **The Auxiliary Carry Flag.** Records if the operation caused a carry after the first four bits.

Note that a flag remains unchanged until one executes another instruction affecting it. Not all flags are equally important. For example, the Aux. Carry flag is not even accessible to a user program. (It is used internally by the processor for executing certain instructions.) The parity flag is used mostly in communication related programs to detect communication errors. The other three flags are, however, very useful for changing the flow of the program.

In addition to the seven 8-bit data registers, 8085 also has two 16 bit registers. Register PC (*Program Counter*) holds the address of the next program byte that is to be executed and SP (*Stack Pointer*) contains the address of the last byte stored on the stack.

8085 executes a program by repeating the following set of actions.

- **Instruction Fetch.** The processor sends the 16 bits address in register PC to the external memory and by activating the $\overline{RD}$ line, reads the operation code of the instruction.
- **Instruction Decode.** The instruction byte is decoded to determine the exact series of actions required to carry out that instruction.
- **Instruction Execution.** The processor carries out the specific set of actions to complete the execution of the instruction.

Assembly Language

The language understood by the processor is called the machine language. It only uses 1's and 0's and is therefore is very cryptic. However, it is also very powerful. It allows one to manipulate data *inside the processor.* The assembly language has the same powers as the machine language, but it uses mnemonics in place of strange codes for instructions, it allows one to name constants and variables and to use labels to identify addresses. An assembly program needs to be translated by an *Assembler* into the machine code understood by the processor.

An assembly program has two kinds of statements: assembly directives, which are supposed to guide the assembler during its translation process and the assembly instructions which are actually the statements that are translated into the machine language.

A typical Assembly program line has the following components:

label: op-code operand1 [,operand2] ; comment

The label is made up of up to six characters (letters, digits, $ and underscore) with the first character being a letter. Note that the label must be followed by a colon. The op-code is a numonic identifying the instruction. It is followed by one or more operands separated by commas. Finally, the line may also have a comment which is preceded by a semicolon. One or more components may be absent in a line. In a typical program, most lines will not have a label, and only a few will carry comments. Many op-codes do not require any operands and some lines may begin with semicons,i.e., they will contain only comments.

8085 Processor Instructions[1]

Notation

r, r_1, r_2 refer to one of the data registers A, B, C, D, E, H or L.
rr refers to a register pair.
(Pairs B-C, D-E and H-L are refered to as B, D and H respectively)
dd refers to 8 bit constant
dddd refers to a 16 bit constant.

Assembler allows one to write constants using any number system (Decimal, Hex, Octal or Binary). By default the constants are decimal. Hex constants must begin with a digit and should be followed by an H. Octal constants are followed by letter O and Binary by letter B. (Decimal constants may be followed by letter D, but this is superfluous). One may even use ASCII constants by enclosing the corresponding character in quotes. Constants with negative signs are converted by the assembler into 2's complement notation. The constants

may also be expressed as formulae which are evaluated in 16 bit integer arithmetic while the Assembler translates the program to machine language. One may use +, −, * and / to specify the four basic arithmetic operations in these formulae.

Data movement instructions

- **MOV r_1, r_2** Copies the contents of register r_2 to register r_1. (r_2 is not changed)
  1 byte code. Does not affect any flags.
  Example: If Reg L has 35H, then after executing MOV C, L, both registers C and L will have 35H.

- **MVI r, dd** Puts 8 bit constant dd in register r.
  2 byte code. Does not affect any flags.
  Example: After executing MVI D, 35H, old contents of register D will be replaced by 35H.

- **LDA dddd** Copy the contents of memory location dddd to accumulator.
  3 byte code. Does not affect any flags.
  Example: If memory location 3A35H contains 78H, then execution of LDA 3A35H replaces the contents of accumulator with 78H.

- **STA dddd** Write the contents of accumulator to memory location dddd.
  3 byte code. Does not affect any flags.
  Example: If register A has 78H, then after executing STA 3A35H, location 3A35H in memory will have 78H also.

- **MOV r, M** Copy the contents of to memory location specified in register pair H-L to register r.
  1 byte code. Does not affect any flags.

[1]Not all instructions are listed here.

Example: If register H contains 12H, L, 34H and memory location 1234H, 78H, then after executing MOV C, M, register C contents will become 78H.

- **MOV M, r** Write the contents of register r to memory location specified in register pair H-L.
  1 byte code. Does not affect any flags.
  Example: If register C contains 78H, H, 12H and L, 34H, then after executing MOV M, C, location 1234H in the memory will contain 78H.

- **LXI rr, dddd** Write the 16 bit constant dddd to register pair rrrr.
  3 byte code. Does not affect any flags.
  Example: On executing LXI B, 1234H, register B will have 12H and C will have 34H. Note that one may also load SP with a 16 bit address by using LXI SP, dddd.

- **PUSH rr** Push the register pair rr to the stack.
  Processor executes this instruction by decrementing register SP by 1, storing the first register of the pair at memory address in SP, decrementing SP again and then storing the second register at memory address in SP. Note that this does not destroy the contents of the registers.
  PUSH PSW pushes the Program Status Word (A register and the flags) to the memory.
  1 byte code, affects no flags.
  Example: If the B and C registers have 12H and 34H respectively and register SP has 2000H, then executing PUSH B will change the contents of memory address 1FFFH to 12H and of address 1FFEH to 34H. Register SP will change to 1FFEH. Registers B and C will not change.

- **POP rr** Pop the stack to register pair rr.
  Processor executes this instruction by loading the second register of the pair with the contents of memory address in SP, incrementing register SP by 1, loading the first register with the contents of memory address in SP and incrementing SP again.
  POP PSW retrives the Program Status Word (A register and the flags) from the stack.
  1 byte code, affects no flags.
  Example: If register SP has 1000H, and memory locations 1000H and 1001H contain 56H and 78H respectively, then executing POP B will change the contents registers B and C to 78H and 56H respectively. Register SP will change to 1002H.

Arihmetic and Logical instructions

- **ADD r** Add the contents of register r to accumulator and replace the accumulator contents with the result.
  ADD M adds the contents of the memory location (whose address is in the H-L pair) to the accumulator.
  1 byte code. Affects all flags.
  Example: If registers A and B contain 9DH and 7CH respectively, then executing ADD B will change register A contents to 19H and the flags will be set as follows: Zero flag

reset (result nonzero), Carry flag set, Sign flag positive, Parity flag odd and Aux carry set.

- **ADC r** Add the contents of register r and the carry flag to accumulator and replace the accumulator contents with the result.
  1 byte code. Affects all flags.
  Example: If registers A and B contain 58H and 37H respectively and if the carry flag is set, then executing ADD B will change register A contents to 90H and the flags will be set as follows: Zero flag reset (result nonzero), Carry reset (no carry), Sign negative, Parity even and Aux carry set.

- **ADI dd** Add constant dd to accumulator and replace the accumulator contents with the result.
  2 byte code. Affects all flags.
  Example: If accumulator contains 9DH then executing ADI 53H will change accumulator contents to 50H and the flags will be set as follows: Zero flag reset (result nonzero), Carry reset (no carry), Sign positive, Parity even and Aux carry set.

- **ACI dd** Add constant dd and the carry flag to accumulator and replace the accumulator contents with the result.
  2 byte code. Affects all flags.
  Example: If the accumulator contains 68H and carry flag is set, then executing ACI 0A2H will change accumulator contents to 0BH and the flags will be set as follows: Zero flag reset (result nonzero), Carry set (carry), Sign positive, Parity odd and Aux carry reset.

- **SUB r** Subtract the contents of register r from the accumulator and replace the accumulator contents with the result.
  SUB M subtracts the contents of the memory location (whose address is in the H-L pair) from the accumulator.
  1 byte code. Affects all flags.
  Example: If registers A and B contain 9DH and 7CH respectively, then executing SUB B will change register A contents to 21H and the flags will be set as follows: Zero flag reset (result nonzero), Carry reset (no borrow), Sign positive, Parity even and Aux carry reset (no borrow after 4 bits).

- **SBB r** Subtract the contents of register r and the carry flag from the accumulator and replace the accumulator contents with the result.
  1 byte code. Affects all flags.
  Example: If registers A and B contain 9DH and 7CH respectively and if the carry flag is set, then executing SUB B will change register A contents to 20H and the flags will be set as follows: Zero flag reset (result nonzero), Carry reset (no borrow), Sign positive, Parity even and Aux carry reset (no borrow after 4 bits).

- **SUI dd** Subtract constant dd from accumulator and replace the accumulator contents with the result.
  2 byte code. Affects all flags.
  Example: If accumulator contains 70H then executing SUI 9BH will change it to D8H and the flags will be set as follows: Zero flag reset (result nonzero), Carry set (borrow), Sign negative, Parity even and Aux carry set (borrow after 4 bits).

- **SBI dd** Subtract constant dd and the carry flag from accumulator and replace the accumulator contents with the result.
  2 byte code. Affects all flags.
  Example: If accumulator has 3CH and if the carry flag is set, then executing SBI 42H will change it to F9H and the flags will be set as follows: Zero flag reset (result nonzero), Carry set (borrow), Sign negative, Parity even and Aux carry reset (no borrow after 4 bits).

- **INR r** Increment data register r by 1. (If the register contains FFH, on incrementing, it becomes 0.)
  INR M increments the contents of the memory location whose address is in the H-L pair.
  1 byte code. Affects all flags except carry.
  Example: If register C contains F4H, then executing INR C would change it to F5H, and the flags would be as follows: Zero flag reset, Sign negative, parity even and Aux carry reset.

- **DCR r** Decrement data register r by 1. (If the register contains 0, on decrementing, it becomes FFH.)
  DCR M decrements the contents of the memory location whose address is in the H-L pair.
  1 byte code. Affects all flags except carry.
  Example: If register C contains F4H, then executing DCR C would change it to F3H, and the flags would be as follows: Zero flag reset, Sign negative, parity even and Aux carry reset.

- **INX rr** Increment register pair rr by 1. (If the register pair contains FFFFH, on incrementing, it becomes 0.)
  1 byte code. Affects no flags.
  Example: If registers B and C contains 04H and FFH respectively, then executing INX B would change them to 05H and 00H.

- **DCX rr** Decrement register pair rr by 1. (If the register pair contains 0, on decrementing, it becomes FFFFH.)
  1 byte code. Affects no flags.
  Example: If registers B and C contains 04H and FFH respectively, then executing DCX B would change them to 04H and FEH.

- **DAD rr** Add the contents of register pair rr to register pair H-L and put the result in H-L.
  1 byte code. Affects only the carry flag.
  Example: If registers B, C, H and L contain 40H, B5H, E3H and 8CH respectively, then executing DAD B will change registers H and L to 24H and 41H respectively and the carry flag would be set.

- **CMA** Complement all the bits of the accumulator.
  1 byte code. Affects no flags.
  Example: If accumulator contains 36H, then executiong CMA would change it to C9H.

- **CMC** Complement the carry flag.
  1 byte code. Affects only carry flag.
  Example: If carry flag is 1, then executiong CMA would change it to 0.

- **CMP r** Compare the accumulator with register r and set the flags according to the result of <A> − ¡r¿. Accumulator contents are not changed.
  Accumulator can be compared with the contents of memory location whose address is in the H-L pair by invoking CMP M.
  1 byte code. Affects all flags.
  Example: If registers A and B contain 46H and 75H respectively, then executing CMP B sets the flags as follows: Zero flag reset, Carry set, Parity even, Sign negative and Aux Carry reset.

- **CPI dd** Compare accumulator with constant dd and set the flags according to the result of <A> − dd. Accumulator contents are not changed.
  1 byte code. Affects all flags.
  Example: If registers A and B contain 46H and 75H respectively, then executing CMP B sets the flags as follows: Zero flag reset, Carry set, Parity even, Sign negative and Auxarry flag reset.

- **RAL** Rotate accumulator left through carry. (The carry flag goes in the rightmost bit of the accumulator and the leftmost bit of accumulator goes in the carry flag.)
  1 byte code. Affects only the carry flag.
  Example: If the carry flag is set and the A register contains 35H, then executing RAL changes A register to 6BH and resets the carry flag.

- **RAR** Rotate accumulator right through carry. (The carry flag goes in the leftmost bit of the accumulator and the rightmost bit of accumulator goes in the carry flag.)
  1 byte code. Affects only the carry flag.
  Example: If the carry flag is not set and the A register contains B1H, then executing RAR changes A register to 58H and sets the carry flag.

- **RLC** Rotate accumulator left. The leftmost bit of accumulator goes in the carry flag as well as in the rightmost position in the accumulator.)

1 byte code. Affects only carry flag.
Example: If the carry flag is set and the A register contains 35H, then executing RLC changes A register to 6AH and resets the carry flag.

- **RRC** Rotate accumulator right. The rightmost bit of accumulator goes in the carry flag as well as in the leftmost position in the accumulator.)
  1 byte code. Affects only carry flag.
  Example: If the carry flag is set and the A register contains 83H, then executing RRC changes A register to C1H and sets the carry flag.

- **ANA r** Perform logical AND between every bit of accumulator and the corresponding bit of the register r and put the result in the accumulator.
  ANA M logically ANDs the contents of the memory location (whose address is in the H-L pair) with the accumulator.
  1 byte code. Affects all flags.
  Example: If registers A and B contain A3H and 96H respectively, then executing ANA B changes register A to 82H and the flags are set as follows: Zero, carry and AuxCarry flags reset, sign negative and parity even.

- **ANI dd** Perform logical AND between every bit of accumulator and the corresponding bit of the constant dd and put the result in the accumulator.
  2 byte code. Affects all flags.
  Example: If register A contains 74H then executing ANI 23H changes register A to 20H and the flags are set as follows: Zero, carry and AuxCarry flags reset, sign positive and parity odd.

- **ORA r** Perform logical OR between every bit of accumulator and the corresponding bit of the register r and put the result in the accumulator.
  ORA M logically ORs the contents of the memory location (whose address is in the H-L pair) with the accumulator.
  1 byte code. Affects all flags.
  Example: If registers A and B contain A3H and 96H respectively, then executing ORA B changes register A to B7H and the flags are set as follows: Zero, carry and AuxCarry flags reset, sign negative and parity even.

- **ORI dd** Perform logical OR between every bit of accumulator and the corresponding bit of the constant dd and put the result in the accumulator.
  2 byte code. Affects all flags.
  Example: If register A contains 74H then executing ORI 23H changes register A to 77H and the flags are set as follows: Zero, carry and AuxCarry flags reset, sign positive and parity even.

- **XRA r** Perform logical ExOR between every bit of accumulator and the corresponding bit of the register r and put the result in the accumulator.
  XRA M logically ExORs the contents of the memory location (whose address is in the

H-L pair) with the accumulator.
1 byte code. Affects all flags.
Example: If registers A and B contain A3H and 96H respectively, then executing XRA B changes register A to 35H and the flags are set as follows: Zero, carry and AuxCarry flags reset, sign positive and parity even.

- **XRI dd** Perform logical OR between every bit of accumulator and the corresponding bit of the constant dd and put the result in the accumulator.
  2 byte code. Affects all flags.
  Example: If register A contains 74H then executing XRI 23H changes register A to 57H and the flags are set as follows: Zero, carry and AuxCarry flags reset, sign flag positive and parity flag odd.

Program Flow Control instructions

- **JMP dddd** Jump to instruction at address dddd.
  3 byte code, Affects no flags.
  Example: JMP 1000H causes the instruction at 1000H to be executed next.

- **JZ dddd** Jump to instruction at address dddd if the zero flag is set; otherwise do the next instruction.
  3 byte code, Affects no flags.
  Example: JZ 1000H causes the instruction at 1000H to be executed next if the zero flag is set.

- **JNZ dddd** Jump to instruction at address dddd if the zero flag is not set; otherwise do the next instruction.
  3 byte code, Affects no flags.
  Example: JNZ 1000H causes the instruction at 1000H to be executed next if the zero flag is not set.

- **JC dddd** Jump to instruction at address dddd if the carry flag is set; otherwise do the next instruction.
  3 byte code, Affects no flags.
  Example: JC 1000H causes the instruction at 1000H to be executed next if the carry flag is set.

- **JNC dddd** Jump to instruction at address dddd if the carry flag is not set; otherwise do the next instruction.
  3 byte code, Affects no flags.
  Example: JNC 1000H causes the instruction at 1000H to be executed next if the carry flag is not set.

- **JP dddd** Jump to instruction at address dddd if the sign flag is positive; otherwise do the next instruction.

3 byte code, Affects no flags.
Example: JP 1000H causes the instruction at 1000H to be executed next if the sign flag is positive.

- **JM dddd** Jump to instruction at address dddd if the sign flag is negative; otherwise do the next instruction.
  3 byte code, Affects no flags.
  Example: JM 1000H causes the instruction at 1000H to be executed next if the sign flag is negative.

- **JPE dddd** Jump to instruction at address dddd if the parity flag shows even parity; otherwise do the next instruction.
  3 byte code, Affects no flags.
  Example: JPE 1000H causes the instruction at 1000H to be executed next if the parity flag is even.

- **JPO dddd** Jump to instruction at address dddd if the parity flag shows odd parity; otherwise do the next instruction.
  3 byte code, Affects no flags.
  Example: JPO 1000H causes the instruction at 1000H to be executed next if the parity flag is odd.

- **CALL dddd** Call the subroutine at address dddd.
  Calling a subrotine causes the processor to push the return address (address of the next instruction) to the stack and then load register PC with the subroutine starting address.
  3 byte code, Affects no flags.
  Example: CALL 1000H pushes the return address to stack and jumps to the subroutine at 1000H.

- **CZ dddd** Call the subroutine at address dddd if the zero flag is set; otherwise do the next instruction.
  3 byte code, Affects no flags.
  Example: CZ 1000H pushes the return address to stack and jumps to the subroutine at 1000H if the zero flag is set.

- **CNZ dddd** Call the subroutine at address dddd if the zero flag is not set; otherwise do the next instruction.
  3 byte code, Affects no flags.
  Example: CNZ 1000H pushes the return address to stack and jumps to the subroutine at 1000H if the zero flag is not set.

- **CC dddd** Call the subroutine at address dddd if the carry flag is set; otherwise do the next instruction.
  3 byte code, Affects no flags.
  Example: CC 1000H pushes the return address to stack and jumps to the subroutine at 1000H if the carry flag is set.

- **CNC dddd** Call the subroutine at address dddd if the carry flag is not set; otherwise do the next instruction.
  3 byte code, Affects no flags.
  Example: CNC 1000H pushes the return address to stack and jumps to the subroutine at 1000H if the carry flag is not set.

- **CP dddd** Call the subroutine at address dddd if the sign flag is positive; otherwise do the next instruction.
  3 byte code, Affects no flags.
  Example: CP 1000H pushes the return address to stack and jumps to the subroutine at 1000H if the sign flag is positive.

- **CM dddd** Call the subroutine at address dddd if the sign flag is negative; otherwise do the next instruction.
  3 byte code, Affects no flags.
  Example: CM 1000H pushes the return address to stack and jumps to the subroutine at 1000H if the sign flag is negative.

- **CPE dddd** Call the subroutine at address dddd if the parity flag shows even parity; otherwise do the next instruction.
  3 byte code, Affects no flags.
  Example: CPE 1000H pushes the return address to stack and jumps to the subroutine at 1000H if the parity flag is even.

- **CPO dddd** Call the subroutine at address dddd if the parity flag shows odd parity; otherwise do the next instruction.
  3 byte code, Affects no flags.
  Example: CPO 1000H pushes the return address to stack and jumps to the subroutine at 1000H if the parity flag is odd.

- **RET** Return from the subroutine.
  Processor returns from a subroutine by popping the stack to register PC. (Note that the top of the stack should contain the address of the instruction following the CALL instruction.)
  1 byte code, Affects no flags.
  Example: RET returns to the main program.

- **RZ** Return from the subroutine if zero flag is set.
  1 byte code, Affects no flags.
  Example: RZ returns to the main program if zero flag is set; otherwise the next instruction is executed.

- **RNZ** Return from the subroutine if zero flag is not set.
  1 byte code, Affects no flags.
  Example: RNZ returns to the main program if zero flag is not set; otherwise the next instruction is executed.

- **RC** Return from the subroutine if carry flag is set.
  1 byte code, Affects no flags.
  Example: RC returns to the main program if carry flag is set; otherwise the next instruction is executed.

- **RNC** Return from the subroutine if carry flag is not set.
  1 byte code, Affects no flags.
  Example: RNC returns to the main program if carry flag is not set; otherwise the next instruction is executed.

- **RP** Return from the subroutine if sign flag is positive.
  1 byte code, Affects no flags.
  Example: RP returns to the main program if sign flag is positive. otherwise the next instruction is executed.

- **RM** Return from the subroutine if sign flag is negative.
  1 byte code, Affects no flags.
  Example: RM returns to the main program if sign flag is negative. otherwise the next instruction is executed.

- **RPE** Return from the subroutine if parity flag is even.
  1 byte code, Affects no flags.
  Example: RPE returns to the main program if sign flag is even. otherwise the next instruction is executed.

- **RPO** Return from the subroutine if parity flag is odd.
  1 byte code, Affects no flags.
  Example: RPO returns to the main program if sign flag is odd. otherwise the next instruction is executed.

Processor Control instructions

- **HLT** Halt the processor.

Assembler directives

- **ORG dddd** Start assembly at address dddd.
- **name EQU dddd** Replace *every* occurance of "name" in the assembly by the constant dddd.
- **name SET dddd** Replace *every* occurance of "name" in the assembly by the constant dddd till "name" is set to another value. Thus, unlike the EQU directive, the set directive allows one to redefine the variable any number of times in the same program. A particular substitution is in effect till it is redefined. (Generally, the EQU directive is prefered over the SET directive.)
- **DB $\mathbf{dd_1}$, $\mathbf{dd_2}$, ...** Define bytes.
  The assembler writes the bytes dd_1, dd_2, etc. in consecutive memory locations. A string delimited by quotes is written in memory by separating its chaaracter.
  Example: The assembly of statements **ORG 100H** followed by **DB 30, 18H, 'A'+3, 'ABC'** will force the following memory contents: at address 100H: 1EH, at 101H: 18H, at 102H: 44H, at 103H: 41H, at 104H: 42H, at 105H: 43H.
- **DW $\mathbf{dddd_1}$, $\mathbf{dddd_2}$, ...** Define words.
  The assembler writes the (16 bits or 2 byte) words $dddd_1$, $dddd_2$, etc. in consecutive memory locations. The lower byte of each word is written in the first free location and is then followed by the higher byte.
  Example: The assembly of statements **ORG 100H** followed by **DW 1234H, 30+5, 'A'+3** will force the following memory contents: at address 100H: 34H, at 101H: 12H, at 102H: 23H, at 103H: 00H, at 104H: 44H, at 105H: 00H.
- **DS dddd** Define space.
  The assembler does not write any code in the next dddd memory locations.
  Example: The assembly of statements **ORG 100H**, **DW 3759H**, **DS 30H** and **DB 'ab'** will result in the following memory contents: at address 100H: 59H, at 101H: 37H, at 132H: 61H, at 133H: 62H. Contents of locations 102H to 131H are unspecified.
- **\$** Address pointer.
  Assembler assigns the value of the address pointer at the begining of the current assembly statement to the symbol \$.
  Example: The assembly statements **ORG 100H** and **DW \$, \$** will result in the following memory contents: at address 100H: 00H, at 101H: 01H, at 132H: 00H, at 133H: 01H.
- **END** End of the Assembly program.
  The assembler stops reading the program at this statement. Programmers use the remaing space in the file to enumerate the development history, authors names, contacts, etc.