

NAME: Arjun R Krishna COURSE: B-Tech BRANCH: ECE

SUBJECT NAME: MICROPROCESSOR AND MICROCONTROLLER

SIGNATURE:

DATE: 22/08/2022

# Part 1

**Q1. Identify the difference between Macro and subroutine.**

**Ans.** Macros can only be used in the program they are defined in and only after the definition is expanded at compilation/generation.

Subroutines (FORM) can be called from both the program they are defined in and other programs.

A MACRO is more or less an abbreviation for some lines of code that are used more than once or twice. A FORM is a local subroutine (which can be called external). A FUNCTION is (more or less) a subroutine that is called external.

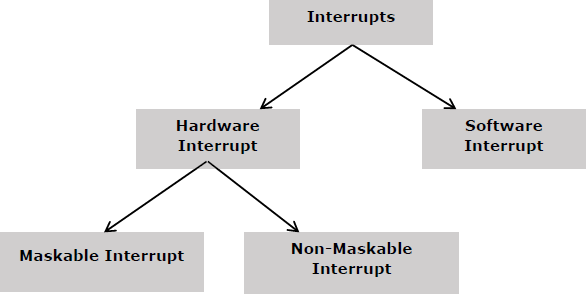
Since debugging a MACRO is not really possible, prevent the use of them (I’ve never used them, but seen them in action). If the subroutine is used only local (called internal) use a FORM. If the subroutine is called external (used by more than one program) use a FUNCTION.

1. When a task is to be done repeatedly then it is written as subroutine and this subroutine will be called each time to perform that task.
2. Subroutine program will be stored in some memory location and program control will be transferred to that location each time.
3. whereas in macro the number of instructions will be less than subroutine. Here each time u call a macro that set of instructions will be inserted in that location.
4. macro doesn't have a return statement while subroutine has.
5. memory requirement for macro is higher.
6. execution time of macro is lesser than subroutine.

# Q2. Name the different types of interrupts supported by 8086.

**Ans**. The interrupt is the method of creating a temporary halt during program execution and allows peripheral devices to access the microprocessor. The microprocessor responds to that interrupt with an ISR (Interrupt Service Routine), which is a short program to instruct the microprocessor on how to handle the interrupt.

The following image shows the types of interrupts we have in an 8086 microprocessor −



Interrupts

* + Hardware Interrupts

A hardware interrupt is caused by any peripheral device by sending a signal through a specified pin to the microprocessor.

The 8086 has two hardware interrupt pins, i.e. NMI and INTR. NMI is a non-maskable interrupt and INTR is a maskable interrupt having lower priority. One more interrupt pin associated is INTA called interrupt acknowledge.

* + NMI

It is a single non-maskable interrupt pin (NMI) having higher priority than the maskable interrupt request pin (INTR)and it is of type 2 interrupt.

When this interrupt is activated, these actions take place −

Completes the current instruction that is in progress. Pushes the Flag register values onto the stack.

Pushes the CS (code segment) value and IP (instruction pointer) value of the return address onto the stack. IP is loaded from the contents of the word location 00008H.

CS is loaded from the contents of the next word location 0000AH. The interrupt flag and trap flag are reset to 0.

* + INTR

The INTR is a maskable interrupt because the microprocessor will be interrupted only if interrupts are enabled using set interrupt flag instruction. It should not be enabled using clear interrupt Flag instruction.

The INTR interrupt is activated by an I/O port. If the interrupt is enabled and NMI is disabled, then the microprocessor first completes the current execution and sends ‘0’ on the INTA pin twice. The first ‘0’ means INTA informs the external device to get ready and during the second ‘0’ the microprocessor receives the 8-bit, say X, from the programmable interrupt controller.

These actions are taken by the microprocessor −

First completes the current instruction.

Activates INTA output and receives the interrupt type, say X.

Flag register value, CS value of the return address and IP value of the return address is pushed onto the stack.

IP value is loaded from the contents of word location X × 4 CS is loaded from the contents of the next word location.

The interrupt flag and trap flag is reset to 0

* + Software Interrupts

Some instructions are inserted at the desired position into the program to create interrupts. These interrupt instructions can be used to test the working of various interrupt handlers. It includes −

INT- Interrupt instruction with type number

It is 2-byte instruction. First byte provides the op-code and the second byte provides the interrupt type number. There are 256 interrupt types under this group.

Its execution includes the following steps −

The flag register value is pushed onto the stack.

CS value of the return address and IP value of the return address is pushed onto the stack. IP is loaded from the contents of the word location ‘type number’ × 4

CS is loaded from the contents of the next word location. Interrupt Flag and Trap Flag is reset to 0

The starting address for type0 interrupt is 000000H, for type1 interrupt is 00004H similarly for type2 is 00008H and ……so on. The first five pointers are dedicated interrupt pointers. i.e. −

TYPE 0 interrupt represents division by zero situation.

TYPE 1 interrupt represents single-step execution during the debugging of a program. TYPE 2 interrupt represents a non-maskable NMI interrupt.

TYPE 3 interrupt represents break-point interrupt. TYPE 4 interrupt represents overflow interrupt.

The interrupts from Type 5 to Type 31 are reserved for other advanced microprocessors, and interrupts from 32 to Type 255 are available for hardware and software interrupts.

INT 3-Break Point Interrupt Instruction

It is a 1-byte instruction having an op-code is CCH. These instructions are inserted into the program so that when the processor reaches there, then it stops the normal execution of a program and follows the break-point procedure.

Its execution includes the following steps −

The flag register value is pushed onto the stack.

CS value of the return address and IP value of the return address is pushed onto the stack. IP is loaded from the contents of the word location 3×4 = 0000CH

CS is loaded from the contents of the next word location. Interrupt Flag and Trap Flag is reset to 0

INTO - Interrupt on overflow instruction

It is a 1-byte instruction and their mnemonic INTO. The op-code for this instruction is CEH. As the name suggests it is a conditional interrupt instruction, i.e. it is active only when the overflow flag is set to 1 and branches to the interrupt handler whose interrupt type number is 4. If the overflow flag is reset then, the execution continues to the next instruction.

Its execution includes the following steps −

Flag register values are pushed onto the stack.

CS value of the return address and IP value of the return address is pushed onto the stack. IP is loaded from the contents of word location 4×4 = 00010H

CS is loaded from the contents of the next word location. Interrupt flag and Trap flag are reset to 0

# Q3. List the flag register in 8086.

Ans. The Flag register is a Special Purpose Register. Depending upon the value of the result after any arithmetic and logical operation, the flag bits become set (1) or reset (0).



Figure – Format of the flag register

There are a total of 9 flags in 8086 and the flag register is divided into two types:

1. Status Flags – There are 6 flag registers in the 8086 microprocessor which become set(1) or reset(0) depending upon condition after either 8-bit or 16-bit operation. These flags are conditional/status flags. 5 of these flags are the same as in the case of the 8085 microprocessor and their working is also the same as in the 8085 microprocessor. The sixth one is the overflow flag.

The 6 status flags are:

* + Sign Flag (S)
  + Zero Flag (Z)
  + Auxiliary Cary Flag (AC)
  + Parity Flag (P)
  + Carry Flag (CY)

1. Control Flags – The control flags enable or disable certain operations of the microprocessor. There are 3 control flags in the 8086 microprocessor and these are:
   * Directional Flag (D) – This flag is specifically used in string instructions.

If the directional flag is set (1), then access the string data from the higher memory location towards the lower memory location.

If the directional flag is reset (0), then access the string data from the lower memory location towards the higher memory location.

* + Interrupt Flag (I) – This flag is for interrupts.

If the interrupt flag is set (1), the microprocessor will recognize interrupt requests from the peripherals.

If the interrupt flag is reset (0), the microprocessor will not recognize any interrupt requests and will ignore them.

* + Trap Flag (T) – This flag is used for on-chip debugging. Setting the trap flag puts the microprocessor into a single-step mode for debugging. In single stepping, the microprocessor executes an instruction and enters into single step ISR.

If the trap flag is set (1), the CPU automatically generates an internal interrupt after each instruction, allowing a program to be inspected as it executes instruction by instruction.

If the trap flag is reset (0), no function is performed.

# Q4. What are the assembler directives there in 8086?

**Ans.** The 8086 instructions are categorized into the following main types. 1. Data Copy / Transfer Instructions 2. Arithmetic and Logical Instructions 3. Shift and Rotate Instructions 4. Loop Instructions 5. Branch Instructions 6. String Instructions 7. Flag Manipulation Instructions 8. Machine Control Instructions.

The 8086 instructions are categorized into the following main types.

1. Data Copy / Transfer Instructions
2. Arithmetic and Logical Instructions
3. Shift and Rotate Instructions
4. Loop Instructions
5. Branch Instructions
6. String Instructions
7. Flag Manipulation Instructions
8. Machine Control Instructions

# Q5. Outline the different types of addressing modes of 8086

**Ans.** The way of specifying data to be operated by an instruction is known as addressing modes. This specifies that the given data is immediate data or an address. It also specifies whether the given operand is register or register pair.

Types of addressing modes:

* + Register mode – In this type of addressing mode both the operands are registers.

Example:

MOV AX, BX XOR AX, DX ADD AL, BL

* + Immediate mode – In this type of addressing mode the source operand is a 8-bit or 16-bit data. The destination operand can never be immediate data.

Example:

MOV AX, 2000 MOV CL, 0A ADD AL, 45

AND AX, 0000

Note that to initialize the value of the segment register a register is required.

MOV AX, 2000 MOV CS, AX

* + Displacement or direct mode – In this type of addressing mode the effective address is directly

given in the instruction as displacement.

Example:

MOV AX, [DISP] MOV AX, [0500]

* + Register indirect mode – In this addressing mode the effective address is in SI, DI or BX. Example: Physical Address = Segment Address + Effective Address

MOV AX, [DI] ADD AL, [BX] MOV AX, [SI]

* + Based indexed mode – In this, the effective address is the sum of the base register and index register.

Base register: BX, BP Index register: SI, DI

The physical memory address is calculated according to the base register. Example:

MOV AL, [BP+SI] MOV AX, [BX+DI]

* + Indexed mode – In this type of addressing mode the effective address is the sum of the index register and displacement.

Example:

MOV AX, [SI+2000] MOV AL, [DI+3000]

* + Based mode – In this, the effective address is the sum of base register and displacement.

Example:

MOV AL, [BP+ 0100]

* + Based indexed displacement mode – In this type of addressing mode, the effective address is the

sum of the index register, base register and displacement.

Example:

MOV AL, [SI+BP+2000]

* + String mode – This addressing mode is related to string instructions. In this, the value of SI and DI are auto-incremented and decremented depending upon the value of the directional flag.

Example:

MOVS B MOVS W

* + Input/Output mode – This addressing mode is related to input-output operations.

Example:

IN A, 45

OUT A, 50

* + Relative mode –

In this, the effective address is calculated with reference to the instruction pointer. Example:

JNZ 8 bit address IP=IP+8 bit address

