12.9.2 No Operation

Flags It does not affect any flag.

Mnemonic NOP

NOP : No operation

Algorithm Do nothing

Addr. Mode Implied addressing mode

Operation The execution of this instruction causes the CPU to do nothing.

- This instruction causes the CPU to do nothing. This instruction uses three clock cycles and increments the instruction pointer to point to the next instruction.
- It can be used to increase the delay of a delay loop.

12.9.3 External Synchronisation

External Synchronisation

- HLT (Halt until interrupt or reset)
- WAIT (Wait for TEST pin active)
- ESC (Escape to external processor)
- LOCK (Lock bus during next instruction)

1. HLT – Halt until interrupt or reset

Mnemonic Halt processing **Flags** No flags are affected.

Operation

- The HLT instruction will cause the 8086 to stop fetching and executing instructions. The 8086 enters into a halt state. To come out of the halt state, there are 3 ways given below.
  (i) Interrupt signal on INTR pin (ii) Interrupt signal on NMI pin
  (iii) Reset signal on reset pin.
- It may be used as an alternative to an endless software loop in situations where a program must wait for an interrupt.

2. WAIT – Wait for TEST pin active

Mnemonic WAIT

Flags No flags are affected.

Operation When this instruction executes, the 8086 enters on idle condition in which it is doing no processing.

- The 8086 will stay in this idle state until 8086 $\overline{\text{TEST}}$ input pin is made low or on interrupt signal is received on the INTR or NMI interrupt pins.

- If a valid interrupt occurs while the 8086 is in the idle state, the 8086 will return to idle state after the interrupt service procedure executes.
- It is used to synchronize the 8086 with external hardware. Such as 8087 math processor.

3. ESC – Escape to external processor

Mnemonic ESC external – opcode, source.

Operation This instruction is used to pass instruction to a coprocessor, such as 8087 math co-processor which shares the address and data bus with on 8086.

- The instruction for the Co-processor are represented by a 6 bit code embedded in the escape instruction.
- When the 8086 fetches on ESC instruction, the coprocessor decodes the instruction and carries out the action specified by the 6 bit code specified in the instruction.
- In most cases 8086 treats the ESC instruction as a NOP in some cases 8086 will access a data item in memory for co-processor.

4. LOCK – Lock bus during next instruction

Mnemonic LOCK

Operation Many multiprocessor systems contain several microprocessors. Each microprocessor has its own local buses and memory. The individual microprocessors are connected together by a shared system bus so that each can access system resources such as disk drives or memory.

- Each microprocessor takes control of the system bus. Only when it needs to access some resource.
- Lock prefix allows a microprocessor to make sure that another processor does not take control of the system bus.
- While it is in the middle of a critical instruction which uses the system bus when an instruction with lock prefix executes the 8086 will assert its bus lock signal output. This signal is connected to an external bus controller, which then prevents any other processor from taking over the system bus.

Example LOCK XCHG SEMAPHORE, AL : The XCHG instruction requires two bus accesses. The lock prefix prevents another processor from taking control of system bus between two accesses.

12.10 Program Transfer Group

8086/8088 provides you :

1) Unconditional CALL

2) JMP $\rightarrow$ Conditional / Unconditional

3) INT (Software interrupt) and many more.

- These instructions are also referred to as Branch instructions.
  We have four subgroups under this :

- Unconditional transfers.
- Conditional transfers.
- Iteration control.
- Interrupt relate instructions.

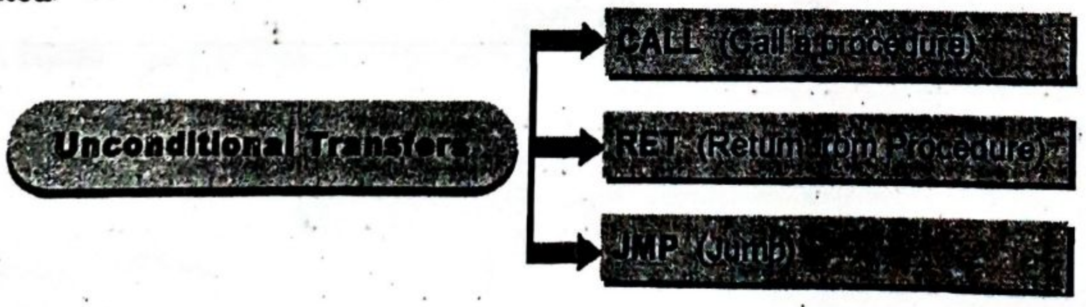
Instructions under these subgroups are listed as follows in Table 12.10.1.

Table 12.10.1 : Program transfer instructions

UNCONDITIONAL TRANSFERS		ITERATION CONTROLS	
CALL	Call procedure	LOOP	Loop
RET	Return from procedure	LOOPE / LOOPZ	Loop if equal/zero
JMP	Jump	LOOPNE/LOOPNZ	Loop if not equal/not zero
CONDITIONAL TRANSFERS		**INTERRUPTS**	
JA/JNBE	Jump if above / not below or equal	INT	Interrupt
JAE/JNB	Jump if above or equal / not below	INTO	Interrupt if overflow
JB/JNAE	Jump if below / not above nor equal	IRET	Interrupt return
JBE/JNA	Jump if below or equal / not above		
JC	Jump if carry		
JE/JZ	Jump if equal / zero		
JG/JNLE	Jump if greater / not less nor equal		
JGE/JNL	Jump if greater or equal / not less		
JL/JNGE	Jump if less / not greater nor equal		
JLE/JNG	Jump if less or equal / not greater		
JNC	Jump if not carry		
JNE/JNZ	Jump if not equal / not zero		
JNO	Jump if not overflow		
JNP/JPO	Jump if not parity / parity odd		
JNS	Jump if not sign		
JO	Jump if overflow		
JP/JPE	Jump if parity / parity even		
JS	Jump if sign		

12.10.1 Unconditional Transfers

The unconditional transfer instructions may transfer control to a target instruction within the current code segment (intrasegment transfer) or to a different code segment (intersegment transfer). The transfer is made unconditionally any time the instruction is executed.



1. CALL – Call a procedure

Q. Describe execution of CALL instruction.

Mnemonic : CALL procedure

Operation : This instruction is used to transfer program control to a subroutine or a procedure. There are two basic types of CALLS : NEAR Call and FAR Call.

- **Near Call :** A near call is a call to a procedure which is in the same segment, which has the CALL instruction. It is also called as Intra segment call.

  (i) If the call to the subroutine or procedure with a 16-bit signed displacement the 8086 will decrement the SP by 2 and push the IP contents onto the stack. Then adds the signed 16 bit value of DISP of IP. The contents of CS are unchanged. Such a call is an intra segment direct call.

  e.g. DISP PROC NEAR : It indicates that DISP is the name of procedure which is in the same code segment. DISP is 16 bit signed displacement.

  Addressing mode : Relative addressing mode

  (ii) If the CALL is to a subroutine is in the same segment and is addressed by the contents of a 16-bit general register/memory. Then the 8086 decrements the SP by 2 and pushes the contents of IP onto the stack and then the contents of specified 16 bit register/memory location.

  The registers can be BX, SI or DI to provide the new value of IP.

  The memory location is addressed by contents of 16 bit register such as BX, SI, DI into IP.

  The contents of CS remain unchanged. Such a call is called is an intra segment indirect call.

  e.g. : CALL Reg 16.

- **Far Call :** A far is a call to a procedure which is in a different segment from that which contains the CALL instruction. Far calls are also called as intersegment calls.

  (i) If the far call to a subroutine or procedure is with a signed displacement, the 8086 decrements SP by 2 and pushes the contents of CS onto the stack and moves the lower 16 bit value of the number like DISP in CALL DISP into CS.

  The SP is again decremented by 2. The contents of IP are pushed onto the stack. The IP is then loaded with the higher 16 bits i.e. MSB value of DISP.

  Thus, as this instruction CALLS a subroutine in another code segment it is an intersegment direct call.

  (ii) If the CALL to a subroutine in another segment is addressed by the contents of a 16 bit register then the 8086 decrements the SP by 2 and pushes CS contents onto the stack.

  The CS is then loaded with the contents of memory locations addressed by [reg 16 + 2] and [reg 16 + 3] in DS.

  The SP is then again decremented by 2, IP is pushed onto the stack. The IP is then loaded with contents of memory locations addressed by [reg 16] and [reg 16 + 1] in DS.

The registers used as reg 16 are BX, SI, DI.

Such a CALL is called as intersegment indirect call.

e.g. : CALL DWORD, PTR [reg16]

2. RET – Return from Procedure

Mnemonic RET optional – pop – value

Algorithm

- Return from near procedure

  POP from stack : IP

  If immediate operand is present :

  SP = SP + operand

- Return from far procedure

  POP from stack

  IP

  CS

  If immediate operand is present

  SP = SP + operand

Operation

- The RET instruction will return execution from a procedure to the next instruction after CALL instruction.
- If the procedure is a near procedure (i.e. in same code segment as CALL instruction), then the return will be done by replacing the IP with a word from the top of stack. This word from the top of stack is the offset of the next instruction after CALL. The SP will be incremented by 2. After return address is popped off the stack.
- If the procedure is a far procedure (in a different code segment), the instruction pointer will be replaced by the word at the top of stack. The SP will be incremented by 2. The CS is then replaced with a word from new top of stack. After CS word is popped the SP will again incremented by two.
- A RET instruction can be followed by a number.

Example RET 2.

- In this case the SP will be incremented by additional 2 addresses after the IP or IP and CS are popped off the stack. This form is used to increment the stack pointer UP over parameters passed to the procedure on the stack.

69

3. JMP – Jump

(Unconditional jump to specified destination)

Mnemonic	JMP	**Flags**	No flags are affected.

Algorithm Always jump.

Operation

- This instruction will cause the 8086 to fetch its next instruction from the location specified in the instruction rather than from next location after JMP instruction.
- There are two basic types of JMPs, near and far.
- Near JMP is a jump where destination location is in the same code segment only IP is changed to get destination location. It is known as intrasegment JMP.
- If the destination is in a segment with a different name from the segment containing the JMP instruction, then both the IP and CS contents will be changed to get the destination location. Such a JMP is far JMP. A far JMP is an Inter segment JMP.
- The near and far JMPs are further described as either direct or indirect. If the destination address is specified within instruction. It is a direct JMP, if the destination address is contained in register or memory location, the JMP is indirect, because 8086 has to access the specified register or memory to get the destination address.

Example JMP WORD PTR [BX]

- This instruction will replace IP with a word from memory location pointed by BX in DS.
- This is an indirect near JMP.

12.10.2 Conditional Transfers

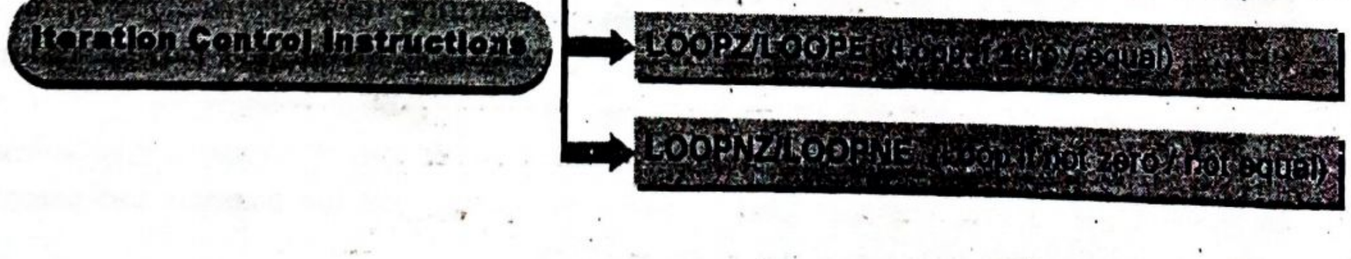
Conditional transfer instructions are also referred as conditional jump instructions.

There are total 18 instructions. Refer Table, each test a different combinations of flags, for a condition. If condition is true, then control is transferred to the target specified in the instruction. If the condition is ***false***, then control passes to the instruction that follows the conditional jump. Very important point regarding these instructions is, all conditional jumps are ***SHORT***, that is, the target must be within the current code segment and within – 128 to + 127 bytes of the first byte of the next instruction. Since the jump is made by adding the relative displacement of the target to the instruction pointer, all conditional jumps are self relative and are appropriate for position independent routines. These instructions does not affect any flag.

	Common Operations	
JC	JUMP if carry	CF = 1
JNC	JUMP if not carry	CF = 0
JZ/JE	JUMP if zero (or equal)	ZF = 1
JNZ/JNE	JUMP if not zero (or not equal)	ZF = 1
JP/JPE	JUMP if parity (or even parity)	PF = 1
JNP/JPO	JUMP if not parity (or odd parity)	PF = 0
JCXZ	JUMP if CX is zero	CX = 0000H
	Signed Operations	
JO	JUMP if overflow	OF = 1
JNO	JUMP if not overflow	OF = 0
JS	JUMP if sign (– ve)	S = 1
JNS	JUMP if not sign (+ ve)	S = 0
JL/JNGE	JUMP if less (i.e. either greater nor equal)	SF ⊕ OF = 1
JNL/JGE	JUMP if not less (i.e. either greater or equal)	SF ⊕ OF = 0
JLE/JNG	JUMP if less or equal (i.e. not greater)	(SF ⊕ OF) + ZF = 1
JNLE/JG	JUMP if neither less nor equal (i.e. greater)	(SF ⊕ OF) + ZF = 0
	Unsigned Operations	
JB/JNAE	JUMP if below (i.e. neither above nor equal)	CF = 1
JNB/JAE	JUMP if not below (i.e. either above or equal)	CF = 0
JBE/JNA	JUMP if below or equal (i.e. not above)	CF ⊕ ZF = 1
JNBE/JA	JUMP if neither below nor equal (i.e. above)	CF ⊕ ZF = 0

12.10.3 Iteration Control Instructions

The Instructions in this group are used to regulate the repetition of software loops. Like conditional transfers the iteration control instructions are self relative and may only transfer to targets that are within –128 to +127 bytes of themselves, i.e. they are short transfers.


Iteration Control Instructions
LOOP
LOOPZ/LOOPE (Loop if zero / equal)
LOOPNZ/LOOPNE (Loop if not zero / not equal)

1. LOOP

Mnemonic LOOP short_label

LOOP : Jump to specified lable if $CX \neq 0$ after auto decrement

Flags No flags are affected.

Algorithm
```
CX = CX – 1
If CX < > 0 then jump
else no jump, continue
```

Operation This instruction is used to repeat a series of instructions some number of times. The number of times the instruction sequence is to be repeated is loaded into CX. Each time loop executes CX is decremented by 1.

- If $CX \neq 0$ execution will jump to destination specified by label.
- If $CX = 0$ execution will go to the next instruction after loop.

2. LOOPZ / LOOPE – Loop if zero / equal

Mnemonic Loop while $CX \neq 0$ and $ZF = 1$

LOOPE short-label/LOOPZ short-label.

Flags No flags are affected

Algorithm
```
CX = CX – 1
If (CX < > 0) and ZF = 1 then jump
else
no jump, continue.
```

Operation
- This instruction is used to repeat a group of instructions some number of times or until zero flag becomes zero.
- Number of times of repetition is loaded in CX.

3. LOOPNZ / LOOPNE – Loop if not zero / not equal

Mnemonic Loop while $CX \neq 0$ and $ZF = 0$

LOOPNZ short-label or

LOOPNE short-label

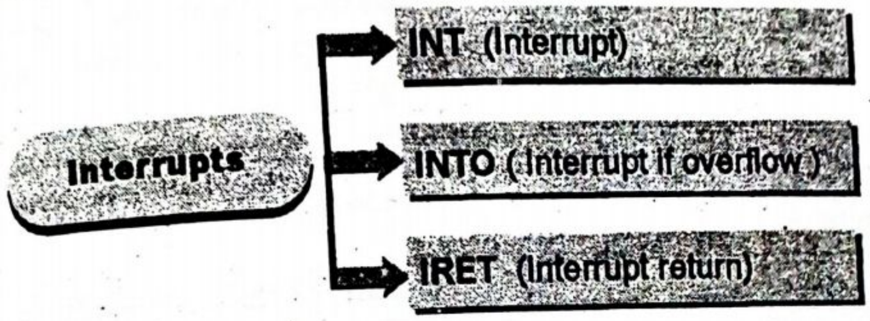
Flags No flags are affected.

Algorithm
```
• CX = CX – 1
• if (CX < > 0) and
        ZF = 0 then jump
  else
        no jump, continue
```

Operation
- This instruction is used to repeat a group of instructions some number of times or until zero flag becomes 1.
- Number of times of repetition is loaded in CX

2.10.4 Interrupts

The interrupt instructions allow interrupt service routines to be activated by rograms as well as by external hardware device. The effect of software interrupts is imilar to hardware-initiated interrupts.



1. INT – Interrupt

Mnemonic INT interrupt – type

Flags IF = 0 and TF = 0
No other flags are affected

Algorithm
- Push to stack
- flag register
- CS
- IP
- IF = 0, TF = 0
- Transfer control to interrupt procedure.

Operation This instruction causes 8086 to call a far procedure. The term 'type' refers number between 0 to 255 which identifies the interrupt.
When an 8086 executes an INT instruction, it will
(i) Decrement SP by 2 and push flag register on stack.
(ii) Decrement SP by 2 and push CS contents on stack.
(iii) Decrement SP by 2 and push the IP after INT on stack.
(iv) Get a new value for IP from a memory address of 4 times the type specified in instruction.
e.g. For INT 8, the new IP will be read from 00020H.
(v) Get new CS from memory address of 4 times the type specified in instruction plus 2. e.g. For INT 8, new value of CS will be read from 00022H.
e.g. INT 35 : New IP from 008C H,
New CS from : 008EH
(vi) Reset IF and TF

2. INTO – Interrupt if overflow

Mnemonic INTO

Flags IF = 0 and TF = 0
No other flag is affected.

Algorithm If OF =1 then INT