Program Control Instructions

Contents:
• Jump Instructions (short/near/far JMP)
• Procedures (CALL, RET)
• Interrupts (INT, INT0, INT3,IRET)
• Miscellaneous Control Instruction (STC, CLC, CMC, HLT, NOP, WAIT, CLI, CLD)

Prepared By:
Mohammed Abdul kader
Lecturer, EEE, IIUC
Jump (JMP) instruction

Jump (JMP) instruction allows the programmer to skip sections of a program and branch to any part of the memory for the next instruction. Jump are two types:

- Unconditional Jump
- Conditional Jump

Unconditional Jump (JMP XXX)

It does not depend any condition or numerical tests. Three types:

- Short Jump
- Near Jump
- Far jump

✔ Short and near jump are often called **intrasegment** jump and far jumps are often called **intersegment** jump.
✔ Short jump and near jump follows a distance or displacement to jump where as far jump follows an address (segment + offset) to jump.
Short Jump (JMP 1byte-displacement)

• Short jump is a two-byte instruction.
• Instead of a jump address, it jumps by following a 8-bit (one byte) signed displacement.
• It allows jumps or branches to memory location within +127 and -128 bytes from the address following the jump.
• The displacement is sign-extended and added to the instruction pointer (IP) to generate the jump address within the current code segment.

Example:
JMP 04H
Jump (JMP) instruction (Continued)

Near Jump (JMP 2byte-displacement)

- Near jump is similar to short jump, except that the distance is farther.
- Near jump is a three-byte instruction.
- Displacement is 16-bit (2 byte) signed displacement.
- It allows jumps or branches to memory location within $\pm 32 \text{K bytes}$ of current code segment.

\[ 2^{16} = 65536 = 64 \text{ Kbyte} \left(\frac{65536}{1024}\right) = -32\text{Kbyte to } +32\text{Kbyte} \]

- The signed displacement added to the instruction pointer (IP) to generate the jump address.

Example:
JMP 0002H

Example:
JMP 0002H

CS=1000
IP=0003H
New IP= 0005

Near jump
Jump (JMP) instruction (Continued)

Far Jump (JMP 4byte-displacement)

- A far jump instruction obtains a new segment and offset address to accomplish the jump.
- It is a 5-byte instruction.
- Byte 2 and 3 contain the new offset address. Byte 4 and 5 contain the new segment address.
- It allows jumps or branches to any memory location of any memory segment. That’s why far jump is called intersegment jump.

Example:
JMP 0127: A300

Jump to CSX10+IP = A300X10+0127 = A3127
Conditional Jump

- A conditional jump instruction allows the programmer to make decision based upon numerical tests.
- The conditional jump instructions are always short jump in 8086.
- Conditional jump instructions test the following flag bits: sign (S), zero (O), carry (C), parity (P) and overflow (O).
- If the condition under test is true, a branch to the label associated with the jump instruction occurs. If the condition is false, the next sequential step in the program executes. For example, a JC will jump if the carry bit is set.

Example of some common conditional jump

<table>
<thead>
<tr>
<th>Assembly language</th>
<th>Tested Condition</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>JNE or JNZ</td>
<td>Z=0</td>
<td>Jump if not equal or jump if not zero</td>
</tr>
<tr>
<td>JE or JZ</td>
<td>Z=1</td>
<td>Jump if equal or jump if zero</td>
</tr>
<tr>
<td>JNO</td>
<td>O=0</td>
<td>Jump if no overflow</td>
</tr>
<tr>
<td>JNP or JPO</td>
<td>P=0</td>
<td>Jump if no parity of jump if parity odd</td>
</tr>
<tr>
<td>JP or JPE</td>
<td>P=1</td>
<td>Jump if parity or jump if parity even</td>
</tr>
</tbody>
</table>
Problems:
Using jump instruction write assembly language program to find out the sum of following series.
(a) 1+4+7+10+13+………………..+112
(b) 1+2+3+………………………..+100
(c) 5+6+8+11+15+…………………..+110
(d) 1+2+3+4+………………………………………………∞

(C) The series can be written as:
5+(5+1)+(5+1+2)+(5+1+2+3)+………
……..+(5+105).
n(n+1)/2=105, => n=14
No of terms, N= 14+1=15

(a) a+(n-1)d=112 => n=38
So, (n-1) or, 37 (25H) times addition is needed to find out the sum of this series.

MOV CX, 25H
MOV AX, 1H
MOV BX, 4H
XXX : ADC AX, BX
    ADD BX, 3H
    DEC CX
    JNZ XXX    // jump if result (value of CX) not zero

MOV CX, 0EH
MOV AX, 5
MOV BX, 6
MOV SI,1
XXX: ADC AX, BX
    INC SI
    ADD BX, SI
    DEC CX
    JNZ XXX
**LOOP**

- The loop instruction is a combination of a decrement CX and the JNZ conditional jump.
- In the 8086 through the 80286 processor, LOOP decrement CX, if CX!=0, it jumps to the address indicated by the label. If CX becomes 0, the next sequential instruction executes.

**Problem:**
Using LOOP instruction write assembly language program to find out the sum of following series.
1+2+3+..........................+100

**Solution:**
No of terms=100, No of addition needed=99 (63H)
MOV CX, 63H
MOV AX, 01H
MOV BX, 02H
SUM: ADC AX, BX
     ADD BX,01H
     LOOP SUM
PROCEDURES

A procedure is a group of instructions (subroutine or function) that usually perform a specific task.

Advantages:

(a) It is reusable section of the software that is stored in memory once, but used as often as necessary.
(b) It saves memory space.
(c) Makes easier to develop software.

Disadvantages:

It takes the compiler a small amount of time to link the procedure and return from it.
How procedure links with main program

- The CALL instruction links to the procedure and the RET (return) instruction return from the procedure.
- The CALL instruction pushes the address (return address) of the instruction following the CALL on the stack. The RET instruction removes an address from the stack so the program return to the instruction following the CALL.
- A procedure begins with the PROC directive and ends with the ENDP directive. The PROC directive is followed by the type of procedure: NEAR (intrasegment) or FAR (intersegment).

**Format of Procedure**

```
XXX    PROC    NEAR/FAR
.........................
.........................
.........................
      RET
XXX    ENDP
```

**Example:**

```
SUMS   PROC   NEAR
ADD    AX,BX
ADD    AX,CX
ADD    AX,DX
      RET
SUMS   ENDP
```

**N.B**

- XXX is the name of level (both level name should be same)
- To call a procedure in main program write: CALL XXX
CALL instruction

The CALL instruction transfer the flow of the program to the procedure. The CALL instruction differ from a jump instruction because a CALL saves a return address on the stack.

Whenever a CALL instruction executes it:

✔ Pushes the IP or, CS:IP on the stack.
✔ Changes the value of IP or, CS:IP.
✔ Jumps to the procedure by new IP or, CS:IP address.

Difference between JMP and CALL instruction

<table>
<thead>
<tr>
<th>JMP</th>
<th>CALL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Doesn’t use stack</td>
<td>Uses stack</td>
</tr>
<tr>
<td>Doesn’t return to the next instruction of JMP</td>
<td>Must return to the next instruction of CALL</td>
</tr>
</tbody>
</table>
Types of CALL

(a) Near CALL  (b) Far CALL

Difference between Near CALL and Far Call

<table>
<thead>
<tr>
<th>Near CALL</th>
<th>Far CALL</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1) Procedure located within the same code segment (±32KB)</td>
<td>(1) Procedure located in the entire memory (1 MB)</td>
</tr>
<tr>
<td>(2) 3-byte instruction</td>
<td>(2) 5-byte instruction</td>
</tr>
<tr>
<td>(3) Only IP content is replaced by (IP±displacement)</td>
<td>(3) Both CS and IP contents are replaced by new CS and IP address</td>
</tr>
<tr>
<td>(4) Stack stores only return IP address (2 byte)</td>
<td>(4) Stack stores the return CS and IP address. (4 byte)</td>
</tr>
</tbody>
</table>

RET instruction

- The return (RET) instruction removes a 16-bit number (near return) from the stack and places it into IP or removes a 32-bit number (far return) and places it into IP and CS.
- The near and far return instructions are both defined in the procedure’s PROC directive, which automatically selects the proper return instruction.
INTERRUPTS

Definition

✓ An interrupt is either a hardware-generated CALL (externally derived from a hardware signal) or a software-generated CALL (internally derived from the execution of an instruction or some other internal event) that allow normal program execution to be interrupted (stopped).

✓ In response to an interrupt, the microprocessor stops execution its current program and calls a procedure called interrupt service procedure (ISP).

✓ An IRET instruction at end of the interrupt-service procedure returns execution to the interrupted program.

Instruction: INT nn ; where nn indicates interrupt vector number (0 to 255)
Each INT instruction is 2-byte long. 1st byte contain opcode and 2nd byte contains vector type number. (exception: INTO and INT3 both are 1-byte instruction)

Types

The 8086 interrupts can be classified into three types. These are:

1. Predefined interrupts
2. User-defined software interrupts
3. User-defined hardware interrupts
Interrupt Vectors

✓ Interrupt vector is the 4 byte long (CS:IP) address of interrupt service procedure stored in the first 1024 bytes (out of 1Mbytes) of the memory (00000-003FFH). This memory location (1024 byte) is known as interrupt vector table.

✓ There are 256 different interrupt vectors, and each vector contains 4 byte address of ISP. The first two bytes contain the IP and last two byte contains the CS.

Instruction: INT nn;
where nn indicates interrupt vector number

Finding address of ISP
For the interrupt type nn(Instruction INT nn), the table address for IP=4×nn and the table address for CS=4×nn+2.

Assign interrupt types
- Types 0 to 4 are for the predefined interrupts.
- Types 5 to 31 are reserved by intel for future use.
- Types 32 to 255 are available for maskable interrupts.

8086 interrupt vector table /pointer table

<table>
<thead>
<tr>
<th>Interrupt type code</th>
<th>20-bit Memory Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>IP 00000H</td>
</tr>
<tr>
<td></td>
<td>CS 00002H</td>
</tr>
<tr>
<td>1</td>
<td>IP 00004H</td>
</tr>
<tr>
<td></td>
<td>CS 00006H</td>
</tr>
<tr>
<td>2</td>
<td>IP 00008H</td>
</tr>
<tr>
<td></td>
<td>CS 0000AH</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>255</td>
<td>IP 003FEH</td>
</tr>
<tr>
<td></td>
<td>CS 00400H</td>
</tr>
</tbody>
</table>
Problem
Find the physical address of interrupt service procedure for the following interrupt instructions:
(a) INT 01H
(b) INT FFH
Interrupt vector table is given.

Solution (a)
Address for IP = 4 × 1 = 00004H
Address for CS = 4 × 1 + 2 = 00006H
So, IP= AC5EH and CS= C83AH
Physical address = CS × 10+IP
= (C83AH × 10+AC5EH)
= D2FFEH

Solution (b)
Address for IP = 4 × FF = 003FCH
Address for CS = 4 × FF + 2 = 003FEH
So, IP= 5A99H and CS= 9800H
Physical address = CS × 10+IP
= (5A99H × 10+9800H)
= 64190H
The predefined interrupts (it is defined by the manufacturer) include

- DIVISION ZERO (type 0)
- SINGLE STEP (type 1)
- NONMASKABLE INTERRUPT pin (type 2)
- BREAKPOINT INTERRUPT (type 3) and
- INTERRUPT ON OVERFLOW (type 4).

**Type 0 (divided by zero):** The 8086 is automatically interrupted whenever a division by zero is attempted.

**Type 1 (Single step execution):** Once TF is set to one, the 8086 automatically generates a TYPE 1 interrupt after execution of each instruction.

**Type 2 (NMI pin):** The nonmaskable interrupt is initiated via the 8086 NMI pin. It is edge triggered (LOW to HIGH) and must be active for two clock cycles to guarantee recognition. It is normally used for catastrophic failures such as power failure.
INT 3 (Break point interrupt)-Type 3

✓ When a break point interrupt inserted (it is inserted by INT 3 instruction), the system executes the instruction up to break point.
✓ Unlike the single step feature which stops execution after each instruction, the break point features executes all the instruction up to the inserted breakpoint and then stop execution.
✓ It is a 1-byte instruction.

INTO (Interrupt on overflow)-Type 4

✓ Interrupt on overflow (INTO) is a conditional software interrupt that tests the overflow flag (O).
✓ If O=0, the INTO instruction performs no operation. And if O=1, INTO call procedure whose address is stored in interrupt vector with type number 4.
Consequences of Software interrupt instruction (INT instruction)

Whenever a software interrupt executes it:

✓ Pushes flags onto stack.
✓ Clears the T and I flag bits.
✓ Pushes CS onto stack.
✓ Fetches new CS from vector table.
✓ Pushes IP onto stack.
✓ Fetches new IP from vector table.
✓ Jumps to service procedure pointed by new CS:IP.

IRET instruction

The interrupt return instruction (IRET) is used only with software and hardware interrupt service procedure.
It is a special return instruction which perform following task-
• POP stack data back into the IP.
• POP stack data back into CS.
• POP stack data back into the flag register
Why we need to clear T and I flag in case of software interrupt?

✓ I flag controls the external hardware interrupt. During software interrupt I flag is cleared to prevent hardware interrupt, because microprocessor does not allow hardware and software interrupt simultaneously.

✓ T flag is cleared to stop debugging so that no debugging occurs during interrupt.
Miscellaneous Control Instruction

Controlling the carry flag bit

STC= Sets the carry flag.
CLC= Clears the carry flag.
CMC= Complements the carry flag.

HLT (Halt) instruction

HLT instruction stops the execution of the program.
There are three ways to exit of HLT state-
(1) By an interrupt.
(2) By a hardware reset.
(3) A DMA operation.

NOP

✓ It just takes time to execute NOP instruction but performs no operation.
✓ Used to insert time delay.
WAIT instruction

- Wait instruction monitors the $\overline{TEST}$ pin of 8086 microprocessor. If WAIT instruction executes while $\overline{TEST}$ pin is 1(high), nothing will happen.
- If WAIT instruction executes while $\overline{TEST}$ pin is low microprocessor waits until $\overline{TEST}$ pin becomes 1(high).

CLD (Clear direction flag)

- This instruction resets the direction flag to 0. No other flags are effected.
- If the direction flag is reset, SI and DI will automatically be incremented when one of the string instructions, such as MOVBS, CMPS or, SCAS executes

CLI (Clear interrupt flag)

- This instruction resets the interrupt flag without effecting other flag bits.
- If the interrupt flag is reset, the 8086 will not respond to an interrupt signal on its INTR input.