Program Control & Processor control Instructions

08.11.12

Objectives

- Processor control instructions
- Program control instructions
- Relational assembly language statements

References:

- Chapter 6 /T2
- Class Notes

PROCESSOR CONTROL INSTRUCTIONS

Processor control Instructions

- Control of the flag bits
  - CLD: *Clear Direction Flag.*
  - STD: *Set Direction Flag*
  - CLI :*Clear Interrupt Flag.*
  - STI : *Set Interrupt Flag.*

Controlling the Carry Flag Bit

- The carry flag (C)
  - propagates the carry or borrow .
  - can indicate errors in assembly language procedures
- Three instructions control the contents of the carry flag:
  - STC (set carry)
  - CLC (clear carry)
  - CMC (complement carry)

WAIT

- Monitors the hardware *TEST* pin .
  - pin inputs a busy condition when at a logic 0 level
  - if *the* pin = 0 the microprocessor waits for the pin to return to a logic 1
  - If the WAIT instruction executes while the pin = 1, nothing happens and the next instruction executes.

HLT

- Often synchronizes external hardware interrupts with the software system.
- Stops the execution of software.
- There are three ways to exit a halt:
  - by interrupt; a hardware reset, or DMA operation

NOP

- When the microprocessor encounters a NOP, it takes a short time to execute.

- Applications:
  - In time delays to waste time.
  - To add instructions at some future point in the program.

ESC

- ESC is considered obsolete as an opcode.
- Six bits of the ESC instruction provide the opcode to the coprocessor and begin executing a coprocessor instruction.
- When an ESC executes, the microprocessor provides the memory address, if required, but otherwise performs a NOP.

LOCK Prefix

- *LOCK* pin of 8086 often disables external bus masters or other system components.
- The prefix Causes the LOCK pin to become a logic 0.
- The prefix Causes LOCK pin to activate for duration of the instruction.
- If more than one sequential instruction is locked, *LOCK* pin remains logic 0 for duration of the sequence of instructions.
- Ex:
  - LOCK:MOV AL,[SI]

Try Yourself

- Describe what happens to the status flags as the sequence of instructions that follows is executed.

  MOV AX, 1234H

  MOV BX, 0ABCDH

  CMP AX, BX

- **Solution:**

  $(AX) = 1234_{16} = 0001001000110100_2$

  $(BX) = ABCD_{16} = 1010101111001101_2$

  $(AX) – (BX) = 0001001000110100_2 – 1010101111001101_2 = 0110011001100111_2$

  Therefore, ZF = 0, SF = 0, OF = 0, PF = 0, CF = 1, AF = 1

PROGRAM CONTROL INSTRUCTIONS

JUMP

- Allows programmer to skip program sections and branch to any part of memory for the next instruction.
- A conditional jump instruction allows decisions based upon numerical tests.
  - results are held in the flag bits, then tested by conditional jump instructions
- LOOP and conditional LOOP are also forms of the jump instruction.

Unconditional Jump (JMP)

- Three types: short jump, near jump, far jump.
- Example:

```
0000     33 db            xor bx, bx
0002     b8 0001          start:  mov ax, 1
0005 03 c3                 add ax, bx
0007     Eb 17            jmp short next

<skipped memory locations>

0020 8b d8               next:   mov bx,ax
0022 eb de                       jmp start
```

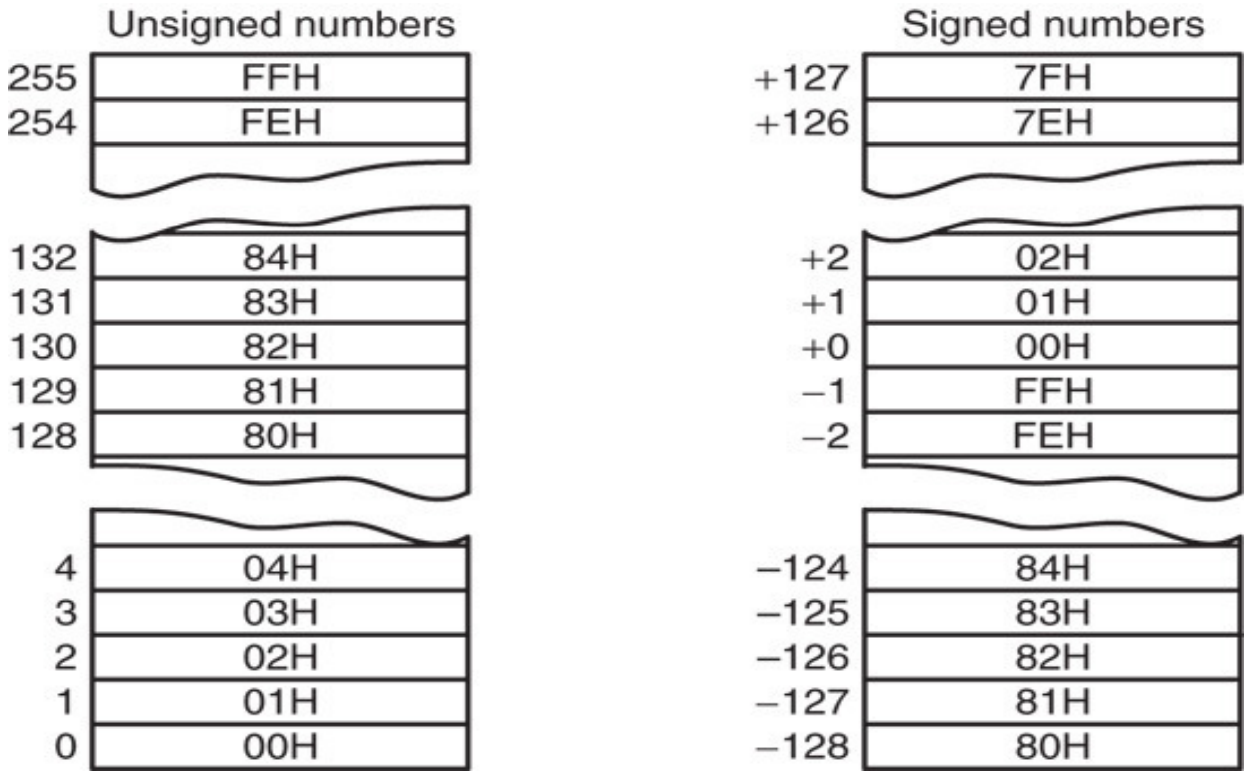
- Example 6.1,6.2,6.3,6.4,6.5

Conditional Jump (JXX)

- Conditional jump instructions test flag bits:
  - sign (S), zero (Z), carry (C)
  - parity (P), overflow (0)

- If the condition under test is true, a branch to the label associated with the jump instruction occurs.
  - if false, next sequential step in program executes

Figure 6–5 Signed and unsigned numbers follow different orders.



- Two sets of conditional jump instructions for magnitude comparisons.
  - When signed numbers are compared, use the JG, JL, JGE, JLE, JE, and JNE instructions.
  - terms *greater than* and *less than* refer to signed numbers
  - When unsigned numbers are compared, use the JA, JB, JAB, JBE, JE, and JNE instructions.
  - terms *above* and *below* refer to unsigned numbers
- Remaining conditional jumps test individual flag bits, such as overflow and parity.

Conditional Jump Table

Mnemonic	Meaning	Jump Condition
JA	Jump if Above	CF = 0 and ZF = 0
JAE	Jump if Above or Equal	CF = 0
JB	Jump if Below	CF = 1
JBE	Jump if Below or Equal	CF = 1 or ZF = 1
JC	Jump if Carry	CF = 1
JE	Jump if Equal	ZF = 1
JNC	Jump if Not Carry	CF = 0
JNE	Jump if Not Equal	ZF = 0
JNZ	Jump if Not Zero	ZF = 0
JPE	Jump if Parity Even	PF = 1
JPO	Jump if Parity Odd	PF = 0
JZ	Jump if Zero	ZF = 1

Also refer Table 6.1 from T2

Opcode	Instruction	Description
77 *cb*	JA *rel8*	Jump short if above (CF=0 and ZF=0)
73 *cb*	JAE *rel8*	Jump short if above or equal (CF=0)
72 *cb*	JB *rel8*	Jump short if below (CF=1)
76 *cb*	JBE *rel8*	Jump short if below or equal (CF=1 or ZF=1)
72 *cb*	JC *rel8*	Jump short if carry (CF=1)
E3 *cb*	JCXZ *rel8*	Jump short if CX register is 0
E3 *cb*	JECXZ *rel8*	Jump short if ECX register is 0
74 *cb*	JE *rel8*	Jump short if equal (ZF=1)
7F *cb*	JG *rel8*	Jump short if greater (ZF=0 and SF=OF)
7D *cb*	JGE *rel8*	Jump short if greater or equal (SF=OF)
7C *cb*	JL *rel8*	Jump short if less (SF<>OF)
7E *cb*	JLE *rel8*	Jump short if less or equal (ZF=1 or SF<>OF)
76 *cb*	JNA *rel8*	Jump short if not above (CF=1 or ZF=1)
72 *cb*	JNAE *rel8*	Jump short if not above or equal (CF=1)
73 *cb*	JNB *rel8*	Jump short if not below (CF=0)
77 *cb*	JNBE *rel8*	Jump short if not below or equal (CF=0 and ZF=0)
73 *cb*	JNC *rel8*	Jump short if not carry (CF=0)
75 *cb*	JNE *rel8*	Jump short if not equal (ZF=0)
7E *cb*	JNG *rel8*	Jump short if not greater (ZF=1 or SF<>OF)
7C *cb*	JNGE *rel8*	Jump short if not greater or equal (SF<>OF)
7D *cb*	JNL *rel8*	Jump short if not less (SF=OF)
7F *cb*	JNLE *rel8*	Jump short if not less or equal (ZF=0 and SF=OF)
71 *cb*	JNO *rel8*	Jump short if not overflow (OF=0)
7B *cb*	JNP *rel8*	Jump short if not parity (PF=0)
79 *cb*	JNS *rel8*	Jump short if not sign (SF=0)
75 *cb*	JNZ *rel8*	Jump short if not zero (ZF=0)
70 *cb*	JO *rel8*	Jump short if overflow (OF=1)
7A *cb*	JP *rel8*	Jump short if parity (PF=1)
7A *cb*	JPE *rel8*	Jump short if parity even (PF=1)
7B *cb*	JPO *rel8*	Jump short if parity odd (PF=0)
78 *cb*	JS *rel8*	Jump short if sign (SF=1)
74 *cb*	JZ *rel8*	Jump short if zero (ZF = 1)
0F 87 *cw/cd*	JA *rel16/32*	Jump near if above (CF=0 and ZF=0)
0F 83 *cw/cd*	JAE *rel16/32*	Jump near if above or equal (CF=0)
0F 82 *cw/cd*	JB *rel16/32*	Jump near if below (CF=1)
0F 86 *cw/cd*	JBE *rel16/32*	Jump near if below or equal (CF=1 or ZF=1)
0F 82 *cw/cd*	JC *rel16/32*	Jump near if carry (CF=1)
0F 84 *cw/cd*	JE *rel16/32*	Jump near if equal (ZF=1)
0F 84 *cw/cd*	JZ *rel16/32*	Jump near if 0 (ZF=1)
0F 8F *cw/cd*	JG *rel16/32*	Jump near if greater (ZF=0 and SF=OF)

LOOP

- A combination of a decrement CX and the JNZ conditional jump.

- LOOP decrements CX.
  - if CX != 0, it jumps to the address indicated by the label
  - If CX becomes 0, the next sequential instruction executes

- There is no direct move from segment register to segment register instruction.

- Study example 6-7

Conditional LOOPs

- LOOP instruction also has conditional forms: LOOPE and LOOPNE

- LOOPE (**loop while equal**)

instruction

  - jumps if CX != 0 while an equal condition exists.
  - exit loop if the condition is not equal or the CX register decrements to 0

Conditional LOOPs

- LOOPNE (**loop while not equal**)
  - jumps if CX != 0 while a not-equal condition exists.
  - exit loop if the condition is equal or the CX register decrements to 0
- Alternates exist for LOOPE and LOOPNE.
  - LOOPE same as LOOPZ
  - LOOPNE instruction is the same as LOOPNZ
- In most programs, only the LOOPE and LOOPNE apply.

Conditional Jump Statement.

- .IF
- .ELSE
- .ELSEIF
- .ENDIF
- Ex.6.9

```
0023 2C 57

002F 2C 37

0033 2C 30
```

```
.IF AL >= 'a' && AL <= 'f'
        SUB  AL,57H
.ELSEIF .IF AL >= 'A' && AL <= 'F'
        SUB  AL,37H
.ELSE
        SUB  AL,30H
.ENDIF
```

EXAMPLE 6–8(a)

```
.IF AL >= 'A'   &&   AL <= 'F'
        SUB   AL,7
.ENDIF
SUB   AL,30H
```

Operator	*Function*
==	Equal or the same as
!=	Not equal
>	Greater than
>=	Greater than or equal
<	Less than
<=	Less than or equal
&	Bit test
!	Logical inversion
&&	Logical AND
||	Logical OR
|	OR

WHILE Loops

- .WHILE-----.ENDW
- The .BREAK and .CONTINUE statements are available for use with the while loop.
  - .BREAK is often followed by .IF to select the break condition as in .BREAK .IF AL == 0DH
  - .CONTINUE can be used to allow a DO–.WHILE loop to continue if a certain condition is met

```
 .WHILE AL != 0DH
 jmp   @C0001
:
 MOV   AH,1
 INT   21H
 STOSB
 .ENDW
```

REPEAT-UNTIL Loops

- The .REPEAT statement defines the start of the loop.
  - end is defined with the .UNTIL statement, which contains a condition

```
.REPEAT

:
MOV   AH,1
INT   21H
STOSB

.UNTIL AL == 0DH
```

PROCEDURES

- A procedure is a group of instructions
  - CALL links to the procedure
  - CALL pushes the address of the instruction following the CALL (**return address**) on the stack.
  - RET (**return**) instruction returns from the procedure
  - RET instruction removes an address from the stack so the program returns to the instruction following the CALL.
- A procedure is a reusable section of the software stored in memory once, used as often as necessary.

PROCEDURES

- Disadvantage of procedure is time it takes the computer to link to, and return from it.
- A procedure begins with the PROC directive and ends with the ENDP directive.
  - each directive appears with the procedure name
- PROC is followed by the type of procedure:
  - NEAR or FAR

PROCEDURES

- Procedures that are to be used by all software (global) should be written as far procedures.
- Procedures that are used by a given task (local) are normally defined as near procedures.
- Most procedures are near procedures.
- Near RET pop a 16-bit number from the stack and save it into the IP.
- Far RET pop a 32-bit number from the stack and save it into the IP and CS.

CALL

- Transfers the flow of the program to the procedure.
- CALL instruction differs from the jump instruction because a CALL saves a return address on the stack.
- The return address returns control to the instruction that immediately follows the CALL in a program when a RET instruction executes.
- Ex.6.6, 6.7,6.8 &6.15