EE 309 - Assignment 2

Prasann Viswanathan, 190070047

Q-1)

The instruction LEA means load effective address. LEA loads a pointer to the item you are addressing.

The MOV instruction is used to copy the data in the source address into the destination register.

Example:

MOV SI, 42H

---> MOV AX, 20H[SI];

---> LEA AX, 20H[SI];

MOV AX, 20H[BX]; stores the *value* at memory location 20H+SI in AX.

LEA AX, 20H[BX]; stores the *address* 20H+SI in AX.

Q-2)

Suppose we are considering numbers X and Y and performing X-Y for the below cases:

i) JL (SF ≠ OF) and JNL (SF = OF)

- X, Y > 0. If X>=Y then SF and OF=0. Therefore SF=OF and a jump occurs by JNL. If X<Y, SF=1 and OF=0 implying a jump via JL.
- X > 0, Y < 0. If the result stays below 127, both SF=OF=0 and if it crosses, both SF=OF=1. We will always have a JNL jump
- X < 0, Y > 0. If the result is greater than -128 then (SF=0) ≠ (OF=1) and if less than -128 again (SF=1) ≠ (OF=0). We will always have a JL jump.
- X, Y < 0. Same as X, Y > 0 with signs flipped. Replace X with -A and Y with -B to get B-A effectively.

ii) JG (SF = OF and Z = 0) and JNG (SF ≠ OF or Z = 1)

- X, Y > 0. If X>Y, SF=0 and OF=0, also X ≠ Y implies Z = 0. JG occurs. And if X<=Y we either have X=Y setting Z=1 or X<Y causing SF=1 and OF=0. Either way we have JNG jump.
- X > 0, Y < 0. Notice that Z always 0 for this case. Then we have the same situation as (i) part 2 and JG occurs.
- X < 0, Y > 0. Again, Z is 0 again, however from (i) part 3 JNG occurs.
- X, Y < 0. Again, same as X, Y > 0 with signs flipped. Replace X with -A and Y with -B to get B-A effectively.

In the cases when X ≠ 0 and Y = 0,

- X>0 implies SF=0 and OF=0. So we always have JNL or JG jumps
- X<0 implies SF=1 and OF=0. So we always have JL or JNG jumps

In the cases when X = 0 and Y ≠ 0,

- Y>0 implies SF=0 and OF=1. So we always have JL or JNG jumps
- Y<0 implies SF=1 and OF=1. So we always have JNL or JG jumps

And when X=Y=0 we trivially have JNL (SF=OF=0) and JNG(Z=1).

Q-3)

```
MOV AX, 7000H
MOV DS, AX
MOV SI, 00H
MOV DS:SI, 47H          ; Roll no. 47
MOV SI, 01H
MOV DS:SI, 79H          ; subtracting 79
MOV SI, 00H
MOV AL, DS:00H
SBB AL, DS:01H          ; Actual subtraction
DAS                     ; Decimal adjust after

MOV AH, AL
AND AL, 0FH             ; Standard logic to extract BCD digits
AND AH, 0F0H
ROR AH, 4
MOV DS:02H, AH
MOV DS:03H, AL
MOV BH, AH
MOV BL, AL
MUL BH                  ;Perform multiplication
MOV DL, AL
AAM                     ; Decimal adjust of multiplication
```

```
MOV DS:04H, AH
MOV DS:05H, AL

MOV BH, 7               ; In order to divide by 7
MOV AL, DL
MOV AH, 00H
DIV BH                  ; Finally divide by 7

MOV DS:04H, AH
MOV DS:05H, AL          ; Store value in same location
RET
```

--- Reporting values of Flags ---

a) 79 is subtracted from 47 decimal.
   68 is stored after DAS.
   Flags are - CF=1, SF=1, AF=1, ZF=0, OF=0, PF=0 (3 set bits).

b) AAM helps store 48 (6X8)
   Flags are - CF=0, SF = 0, AF=1, ZF=0, OF=1, PF=1 (2 set bits)

c) 48 is divided by 7 to give Q=6 and R=6.
   Flags are - CF=0, SF = 0, AF=0, ZF=0, OF=0, PF=1 (2 set bits)

Q-4)

```
MOV [20H], AH;     Move value in AH to memory
MOV AH, 00H;       Make AH 0
MOV SI, AX;        Store index of function
CALL TABLE[SI];    Call function from array
```