```
;                /--------------------\
;                |   Assignment - 2   |
;                \--------------------/
; ------------------------------------------------
; ------------------------------------------------
; by Subhajit Sahu
;
;
;
;
; Memory Structure and MOV Instruction
; ------------------------------------
;
; Q1> Write an 8086 ALP to find the factorial of the given byte of data
; using a recursive algorithm. The result is to be stored from the address
; 7000H: 1000H.
;

; A1>
mov     ax, 7000h
mov     ds, ax              ; DS = 7000h
mov     al, [1002h]         ; AL = 7000h:[1002h] = Input N (byte)
cbw
push    ax                  ;
call    factorial           ; call factorial(N);
mov     [1000h], ax         ; AX = factrorial(N), store at 7000h:[1000h]
hlt                         ; Halt (End)

factorial:                  ; function factorial(Word x)
mov     bp, sp              ; Assuming NEAR call
mov     bx, [bp+0002h]      ; BX = x;
cmp     bx, 0001h           ; if(BX == 1)
je      factorial_1         ; {goto factorial_1;}
mov     ax, bx
dec     ax                  ; AX = x-1;
push    ax                  ;
call    factorial           ; factorial(x-1)
mul     bx                  ; AX = x * factorial(x-1)
ret     02h                 ; return(AX);
factorial_1:
mov     ax, 0001h
ret     02h




;
; Q2> Is it possible to exchange the content of two memory locations
; or the content of two segment registers using the XCHG instruction?
; Why?
;

; A2>
; Not possible directly. XCHG instruction cannot operate on Mem <-> Mem
```

```
; or Seg <-> Seg operands. However, the contents can be exchanged
; indirectly by using an intermendiate register as follows:
; 1. Mem <-> Mem
; mov   Reg, Mem1
; xchg  Reg, Mem2
; mov   Mem1, Reg
; 2. Seg <-> Seg
; mov   Reg16, Seg1
; xchg  Reg16, Seg2
; mov   Seg1, Reg16



;
; Q3> Let the content of different registers in the 8086 be as follows:
; DS= 1000H, SS=2000H, ES=3000H, BX=4000H, SI=5000H,DI=6000H AND
; BP=7000H. Find the memory address / addresses from where the 8086
; accesses the data while executing the following instructions:
; MOV AX, [BX]          MOV AL, [DI]        MOV AX, [BX+DI]
; MOV CX, DS:[BP+4]     MOV BX, [SI]        MOV BH, SS: [SI]
; MOV AX, [BP+DI+5]     MOV BX, [SI-5]      MOV CX, [BP]
; MOV CX, ES: [DI]      MOV AH, [BX+10H]    MOV AX, [BX+10]
;

; A3>
; MOV AX, [BX]          =>  AX <- [14000h]
; MOV AL, [DI]          =>  AL <- [16000h]
; MOV AX, [BX+DI]       =>  AX <- [1A000h]
; MOV CX, DS:[BP+4]     =>  CX <- [17004h]
; MOV BX, [SI]          =>  BX <- [15000h]
; MOV BH, SS: [SI]      =>  BH <- [25000h]
; MOV AX, [BP+DI+5]     =>  AX <- [2D005h]
; MOV BX, [SI-5]        =>  BX <- [14FFBh]
; MOV CX, [BP]          =>  CX <- [27000h]
; MOV CX, ES: [DI]      =>  CX <- [36000h]
; MOV AH, [BX+10H]      =>  AH <- [14010h]
; MOV AX, [BX+10]       =>  AX <- [1400Ah]



;
; Q4> Is it possible to use two memory operands in the ADD and SUB
; instructions?
;

; A4>
; Not possible. ADD, SUB instructions cannot operate on Mem <-> Mem
; or Seg <-> Seg operands. However, the addition / subtraction can
; be done indirectly by using an intermendiate register.



;
```

```
; Q5> What is the difference between SUB and CMP instructions?
;

; A5>
; SUB   dest, src
; SUB instruction finds 'dest - src' and updates various flags as per
; the result obtained after subtraction, and then stores the result in
; 'dest'. Hence, it is used for subtracting two numbers.
; CMP   dest, src
; On the other hand, CMP instruction 'dest - src' and updates various
; flags as per the result obtained after subtraction, but does not store
; the result anywhere. Hence, it is used for just comparing two numbers.




;
; Q6> Consider the following pair of partial programs :
; (i)   MOV AX, 4000H       (ii)    MOV AX, 4000H
;       ADD AX, AX                  ADD AX, AX
;       ADC AX, AX                  RCL AX, 1
;       JC DOWN                     JC DOWN
; For each case what is the data in AX after the execution of third
; instruction and from where does the processor fetch the next
; instruction after execution of the fourth instruction?
;

; A6>
; (i)   AX = 4000h          (ii)    AX = 4000h
;       AX = 8000h                  AX = 8000h
;       AX = C000h, CF = 0          AX = 0000h, CF = 1
;       DontGoto DOWN               Goto DOWN
; The data in AX after execution of third instruction in each case is:
; (i)   AX = C000h          (ii)    AX = 0000h
; After execution of fourth instruction, the processor fetches
; instruction from:
; (i) The next instruction      (ii) Address DOWN (DOWN is a label to
;                                                  a memory address)




;
; Q7> Determine which of the following instructions are illegal, and
; state why:
; MOV AL, CX        MOV 1234H, AX           MOV DX, AL
; MOV CS, 1234H     MOV [1234H],[5678h]     MOV [CX], AX
; MOV AX, [BP+BX]   MOV [SI+DI], CX         MOV 1234H,[BX]
; MOV CS, [SI]
;

; A7>
; MOV AL, CX            =>  Illegal, sizeof(AL) != sizeof(CX).
; MOV 1234H, AX         =>  Illegal, immediate value cant be a
;                           destination operand.
```

```
; MOV DX, AL              =>  Illegal, sizeof(DX) != sizeof(AL).
; MOV CS, 1234H           =>  Illegal, immediate value cant be directly
;                             stored in a segment register.
; MOV [1234H],[5678h]     =>  Illegal, direct memory to memory transfer
;                             is not possible. Also size of memory is
;                             unspecified.
; MOV [CX], AX            =>  Illegal, register CX cant be used for
;                             indirect memory addressing.
; MOV AX, [BP+BX]         =>  Illegal, register BP cannot be used with
;                             register BX in memory addressing. Both are
;                             base registers.
; MOV [SI+DI], CX         =>  Illegal, register SI cannot be used with
;                             register DI in memory addressing. Both are
;                             index registers.
; MOV 1234H,[BX]          =>  Illegal, immediate value cant be a
;                             destination operand.
; MOV CS, [SI]            =>  Legal



;
; Q8> Store the number 5678h in memory location DS:2000H using
; indirect addressing only.
;

; A8>
mov     si, 2000h
mov     ax, 5678h
mov     [si], ax



;
; Q9> Store the number 1234h in absolute memory address 60000h.
;

; A9>
mov     ax, 6000h
mov     ds, ax
mov     ax, 1234h
mov     [0000h], ax



;
; Q10> Move the number at absolute memory address 60000h to DX.
;

; A10>
mov     dx, 6000h
mov     ds, dx
mov     dx, [0000h]
```

```
;
; Q11> Convert an 8B07H from machine language to assembly language.
;

; A11>
mov     ax, [bx]



;
; Q12> Convert an 8B1E004CH from machine language to assembly language.
;

; A12>
mov     bx, [4C00h]



;
; Q13> If a MOV SI, [BX+2] instruction appears in a program, what is
; its machine language equivalent?
;

; A13>
; Machine language equivalent = 8B7702h



;
; Q14> What is wrong with a MOV CS, AX instruction?
;

; A14>
; The value of CS cannot be modified through a register. Special
; instructions, like FAR Jump / FAR Call exist for that purpose.



;
; Q15> Form a short sequence of instructions that load the data
; segment register with a 1000H.
;

; A15>
mov     Reg16, 1000h
mov     Seg, Reg16



;
; Q16> Describe the operation of the following instructions:
```

```
; (i)  PUSH AX      (ii) POP SI         (iii) PUSH [BX]
; (iv) PUSHF        (v)  POP DS         (vi)  PUSH 4
;

; A16>
; (i)   PUSH AX     = Store the value of AX at the top of the stack.
; (iI)  POP SI      = Load the value of SI from the top of the stack.
; (iii) PUSH [BX]   = Store the value of memory address in DS pointed
;                     to by BX at the top of the stack.
; (iv)  PUSHF       = Store the value of Flag register at the top of
;                     the stack.
; (V)   POP DS      = Load the value of DS (Data Segment) from the top
;                     of the stack.
; (vi)  PUSH 4      = Store the value 04h at the top of the stack.



;
; Q17> What values appear in SP and SS if the stack is addressed at
; memory location 02200H?
;

; A17>
; One possible case:
; SS = 0000h
; SP = 2200h
; There can be many other possible cases.



;
; Q18> Compare the operation of a MOV DI, NUMB instruction with an
; LEA DI, NUMB instruction.
;

; A18>
lea     di, [NUMB]
; is same as
mov     di, NUMB
; if NUMB is an immediate value. However LEA is used in cases where
; small calculations are required to be done in one instruction like:
lea     si, [bx + di + 01h]
; which is same as
mov     si, bx
add     si, di
add     si, 0001h



;
; Q19> Develop a sequence of instructions that move the contents of
; data segment memory locations NUMB and NUMB+1 into BX, DX, and SI.
;
```

```
; A19>
mov     bx, [NUMB]
mov     dx, bx
mov     si, bx



;
; Q20> What is the purpose of the direction flag?
;

; A20>
; The direction flag is used for data transfer type instructions.
; It tells whether the index registers should be incremented or
; decremented upon data transfer of each BYTE / WORD.



;
; Q21> Explain the operation of the LODSB instruction.
;

; A21>
; The LODSB instruction loads a byte of data from memory location
; pointed by DS:[SI] into AL and then increments / decrements the
; the value of SI to the next byte in memory as per the direction
; flag. This instruction is mainly used for string processing where
; programs need to load each byte of a string into the accumulator
; and perform some operations on it.



;
; Q22> Explain the operation of the STOSW instruction.
;

; A22>
; The STOSW instruction stores a word of data to memory location
; pointed by ES:[DI] from AX and then increments / decrements the
; the value of DI (by 2) to the next word in memory as per the direction
; flag. This instruction is mainly used for string processing where
; programs need to load each word of a string into the accumulator
; and perform some operations on it and store it back to another memory
; location. It can also be used with array of 'short' / WORD numbers.



;
; Q23> Explain the operation of the OUTSB instruction.
;

; A23>
```

```
; The OUTSB instruction sends a byte of data to an I/O port determined
; by the port number stored in DX, from AL. This instruction is mainly
; used for sending a large chunk of data to a byte port. Programs
; requiring to send huge volume of data (like to a hard disk) often
; use this instruction.



;
; Q24> Write an 8086 ALP to find the sum of 100 words present in an
; array stored from the address 3000H: 1000H in the data segment and
; store the result from the address 3000H: 2000H.
;

; A24>
mov     ax, 3000h
mov     ds, ax                  ; DS = 3000h
mov     ax, 0000h               ; Sum = 0
mov     cx, 100                 ; i = 100 ( = Array.length)
mov     si, 1000h               ; ptr = 1000h
add_loop:                       ; do {
add     ax, [si]                ; Sum += *ptr;
add     si, 2                   ; ptr += 2;
dec     cx                      ; i--;
jnz     add_loop                ; } while(i>0);
mov     [2000h], ax             ; 3000h:[2000h] = Sum
hlt                             ; Stop



;
; Q25> Write an 8086 ALP to find the prime numbers in a list of 100
; bytes of data in an array stored from the address 4000H: 1000H in
; the data segment and store the result from the address 4000H: 3000H.
;

; A25>
mov     ax, 4000h
mov     ds, ax                  ; DS = 4000h
mov     es, ax                  ; ES = 4000h
mov     si, 1000h               ; src_ptr = 1000h
mov     di, 3000h               ; dst_ptr = 3000h
mov     cx, 100                 ; i = 100;
cld                             ; (auto ptr++)
src_arr_loop:                   ; do {
lodsb                           ; N (AL) = *src_ptr, src_ptr++;
mov     bl, al                  ;
mov     bh, 02h                 ; j = 2;
is_prime_loop:                  ; do {
mov     al, bl
div     bh                      ; dx = N % j;
or      ah, ah
jz      not_prime               ; if( N % j == 0 ) goto not_prime;
```

```
inc     bh                  ; j++;
cmp     bh, bl
jb      is_prime_loop       ; } while( j < N );
mov     al, bl
stosb                       ; *dst_ptr = N, dst_ptr++;
not_prime:
dec     cx                  ; i--;
jnz     src_arr_loop        ; } while( i > 0 );
mov     al, 00h             ; Dest. Array terminates with a zero
stosb
hlt



;
; Q26> Write an 8086 ALP to find the number of occurrences of the
; character ‘A’ among 50 characters of a string type data stored from
; the address 5000H:1000H in the data segment and store the result in
; the address 2000H:5000H.
;

; A26>
mov     ax, 5000h
mov     ds, ax              ; DS = 5000h
mov     si, 1000h           ; ptr = 1000h
mov     ax, 0               ; count = 0;
mov     cx, 50              ; i = 50;
src_arr_loop:               ; do {
cmp     BYTE PTR [si], 'A'
jne     not_A               ; if(*ptr != 'A') goto not_A;
inc     ax                  ; count++;
not_A:
inc     si                  ; ptr++;
dec     cx                  ; i--;
jnz     src_arr_loop        ; } while( i > 0 );
mov     cx, 2000h
mov     ds, cx              ; DS = 2000h
mov     [5000h], ax         ; 2000h:[5000h] = count;
hlt



;
; Q27> Write an 8086 ALP to add two matrices having word-type data in
; each element of the matrix. Assume that each element of the result
; after addition of the corresponding elements of the matrix is also
; word-type data. The data for one matrix is present in an array stored
; from the address 8000H: 1000H in the data segment and the corresponding
; data for another matrix is present in an array stored from the address
; 8000H: 2000H in the data segment. The result is to be stored from the
; address 7000H: 1000H.
;
```

```
; A27>
; Assuming that matrix is stored as follows:
; [W - NumRows][W - NumCols]...
mov     ax, 8000h
mov     ds, ax                  ; DS = 8000h
mov     ax, 7000h
mov     es, ax                  ; ES = 7000h
mov     bx, 1000h               ; ptrA = 1000h
mov     si, 2000h               ; ptrB = 2000h
mov     di, 1000h               ; ptrC = 1000h
mov     ax, [bx]
cmp     ax, [si]
jne     mat_size_mismatch       ; if(A.rows != B.rows)mat_size_mismatch;
mov     ax, [bx + 2]
cmp     ax, [si + 2]
jne     mat_size_mismatch       ; if(A.cols != B.cols)mat_size_mismatch;
mul     [bx]
or      dx, dx
jnz     mat_size_overflow       ; if(A.rows * A.cols > 65535)overflow;
mov     cx, ax                  ; i = A.rows * A.cols;
mov     ax, [bx]
mov     es:[di], ax             ; C.rows = A.rows;
mov     ax, [bx + 2]
mov     es:[di + 2], ax         ; C.cols = A.cols;
add     bx, 4                   ; ptrA += 4;
add     si, 4                   ; ptrB += 4;
add     di, 4                   ; ptrC += 4;
mat_add_loop:                   ; do {
mov     ax, [bx]
add     ax, [si]
mov     es:[di], ax             ; *ptrC = *ptrA + *ptrB;
add     bx, 2                   ; ptrA += 2;
add     si, 2                   ; ptrB += 2;
add     di, 2                   ; ptrC += 2;
dec     cx                      ; i--;
jnz     mat_add_loop            ; while(i>0);
hlt                             ; Stop
mat_size_mismatch:
mat_size_overflow:
mov     ax, 0
mov     es:[di], ax             ; C.rows = 0;
mov     es:[di + 2], ax         ; C.cols = 0;
hlt                             ; Stop




;
; Q28> Write an 8086 ALP to multiply two square matrices having word-type
; data in each element of the matrix. Assume that each element of the
; resultant matrix is of double word type. The data for one matrix is
; present in an array stored from the address 8000H:1000H in the data
; segment and the corresponding data for another matrix is present in an
```

```
; array stored from the address 8000H:1000H in the data segment. The
; result is to be stored from the address 7000H: 1000H.
;

; A28>
; Assuming that matrix is stored as follows:
; [W - NumRows][W - NumCols]...
mov     ax, 8000h
mov     ds, ax                  ; DS = 8000h
mov     ax, 7000h
mov     es, ax                  ; ES = 7000h
mov     bx, 1000h               ; ptrA = 1000h
mov     si, 2000h               ; ptrB = 2000h
mov     di, 1000h               ; ptrC = 1000h
push    [bx]            ; [bp + 16] = A.rows
push    [bx + 2]        ; [bp + 14] = A.cols
push    [si]            ; [bp + 12] = B.rows
push    [si + 2]        ; [bp + 10] = B.cols
mov     ax, [bx + 2]
cmp     ax, [si]
jne     mat_size_mismatch       ; if(A.cols != B.rows)mat_size_mismatch;
mov     ax, [bx]
mul     [si + 2]
or      dx, dx
jnz     mat_size_overflow       ; if(A.rows * B.cols > 65535)overflow;
push    ax              ; [bp + 8]  = Row*Col count
push    bx              ; [bp + 6]  = ptrA
push    si              ; [bp + 4]  = ptrB
push    ax              ; [bp + 2]  = Row count
push    ax              ; [bp]      = Col count
mov     bp, sp
mov     ax, [bx]
mov     es:[di], ax         ; C.rows = A.rows;
mov     ax, [si + 2]
mov     es:[di + 2], ax     ; C.cols = B.cols;
add     bx, 4
mov     [bp + 6], bx        ; ptrA += 4;
add     si, 4
mov     [bp + 4], si        ; ptrB += 4;
add     di, 4               ; ptrC += 4;


mov     ax, 0
mov     [bp + 2], ax
mov     [bp], ax

mov     cx, [bp + 2]
mul_loop_j:
mov     ax, [bx]
imul    WORD PTR [si]
add     [bp - 8], ax
add     bx, 2
mov     ax, [bp]
```

```
shl     ax, 1
add     si, ax
dec     cx
jnz     mul_loop_j
mov     ax, [bp - 8]
mov     es:[di], ax
add     di, 2


mat_add_loop:
mov     ax, [bx]
add     ax, [si]
mov     es:[di], ax
add     bx, 2
add     si, 2
add     di, 2
dec     cx
jnz     mat_add_loop
hlt
mat_size_mismatch:
mat_size_overflow:
mov     ax, 0
mov     es:[di], ax
mov     es:[di + 2], ax
hlt




;
; Q29> Select an OR instruction that will :
; i.    OR BL with AH and save the result in AH.
; ii.   OR 88H with ECX.
; iii.  OR DX with SI and save the result in SI
; iv.   OR 1122H with BP
; v.    OR the data addressed by BX with CX and save the result in
;       memory.
; vi.   OR the data stored 40 bytes after the location addressed by BP
;       with AL and save the result in AL
; vii.  OR AH with memory location WHEN and save the result in WHEN
;

; A29>
or      ah, bl          ; (i)
or      ecx, 88h        ; (ii)
or      si, dx          ; (iii)
or      bp, 1122h       ; (iv)
or      [bx], cx        ; (v)
or      al, [bp + 40]   ; (vi)
or      [WHEN], ah      ; (vii)
```

```
;
; Q30> Select the XOR instruction that will:
; i.    XOR BH with AH and save the result in AH.
; ii.   XOR 99H with CL.
; iii.  XOR DX with DI and save the result in DX.
; iv.   XOR the data stored 30 words after the location addressed by
;       BP with DI and save the result in DI.
; v.    XOR DI with memory location WELL and save the result in DI.
;

; A30>
xor     ah, bh          ; (i)
xor     cl, 99h         ; (ii)
xor     dx, di          ; (iii)
xor     di, [bp + 60]   ; (iv)
xor     di, [WELL]      ; (v)




;
; Q31> Select an add instruction that will:
; i.    add BX to AX
; ii.   add 12H to AL
; iii.  add 22H to CX
; iv.   add the data addressed by SI to AL
; v.    add CX to the data stored at memory location FROG
;

; A31>
add     ax, bx          ; (i)
add     al, 12h         ; (ii)
add     cx, 22h         ; (iii)
add     al, [si]        ; (iv)
add     [FROG], cx      ; (v)



;
; Q32> If AX=100H and DX=20FFH, list the sum and contents of flag of
; each flag register bit(C,A,S,Z and O) after the add AX,DX instruction
; executes.
;

; A32>
; Result: AX = 21FFh
; C = 0, A = 0, S = 0, Z = 0, O = 0



;
; Q33> Develop a short sequence of that adds AL,BL,CL,DL,and AH.Save the
; sum in the DH register.
```

```
;

; A33>
mov     dh, 00h
add     dh, al
add     dh, bl
add     dh, cl
add     dh, dl
add     dh, ah




;
; Q34> Select a SUB instruction that will:
; i.     subtract BX from CX
; ii.    subtract 0EEH from DH
; iii.   subtract DI from SI
; iv.    subtract 3322H from EBP
; v.     subtract the data address by SI from CH
; vi.    subtract the data stored 10 words after the location addressed
;        by SI from DX
; vii.   subtract AL from memory location FROG
;

; A34>
sub     cx, bx          ; (i)
sub     dh, 0EEh        ; (ii)
sub     si, di          ; (iii)
sub     ebp, 0EEh       ; (iv)
sub     ch, [si]        ; (v)
sub     dx, [si + 20]   ; (vi)
sub     [FROG], al      ; (vii)




;
; Q35> Develop a sequence of instruction that converts the unsigned
; number in AX (values of 0-65535) into a 5-digit BCD number stored in
; memory, beginning at location addressed by the BX register in the data
; segment. Note that the most-significant character is stored first and
; no attempt is made to blank leading zeros.
;

; A35>
mov     di, bx
add     di, 4           ; ptr = BX + 4;
mov     bx, 10          ; rem = 10;
mov     cx, 5           ; i = 5;
loop_rem:               ; do {
cwd
div     bx              ; AX = AX / 10, DL = AX % 10;
add     dl, '0'         ; DL += '0';
mov     [di], dl        ; *ptr = DL
```

```
dec     di              ; ptr--;
dec     cx              ; i--;
jnz     loop_rem        ; while( i > 0 );
hlt                     ; Stop



;
; Q36> Select an AND instruction that will:
; i.    AND BX with DX and save the result in BX
; ii.   AND 0EAH with DH
; iii.  AND DI with BP and save the result in DI
; iv.   AND the data addressed by BP with CX and save the result in
;       memory
; v.    AND the data stored in four words before the location addressed
;       by SI with DX and save the result in DX
; vi.   AND AL with memory location WHAT and save the result at location
;       WHAT
;

; A36>
and     bx, dx          ; (i)
and     dh, 0EAh        ; (ii)
and     di, bp          ; (iii)
and     [bp], cx        ; (iv)
and     dx, [si - 8]    ; (v)
and     [WHAT], al      ; (vi)
```