MP Assignment II

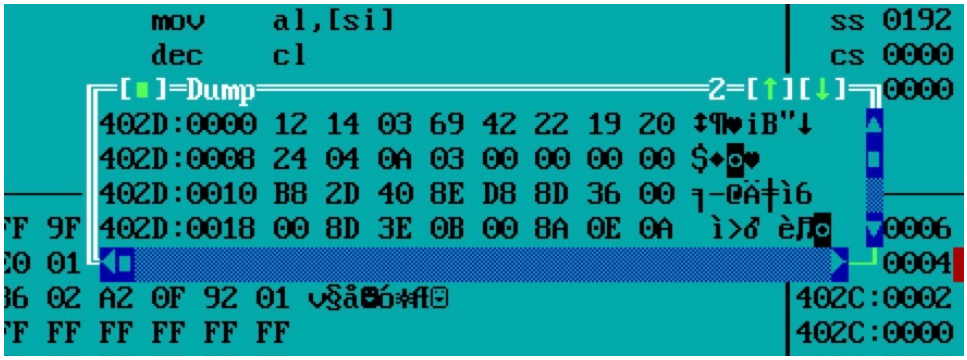
1A) What is the difference between JMP and CALL Instruction?

JMP	CALL
JMP simply 'jumps' to the label you provide it with.	CALL stores the location where it will return (below the CALL instruction) in the stack, then JMPs to the label, and then at the RET instruction, JMPs back to the location which was stored.
JMP changes the IP content.	CALL pushes the IP content onto the stack and updates the IP, which is reset after RET.

1B) Write an ALP to find smallest number in a given block of data using near procedure.

```
data segment
      num db 12h, 14h, 03h, 69h, 42h, 22h, 19h, 20h, 24h, 04h
      siz db 0ah
      sml db ?
data ends

code segment
assume ds: data, cs: code
start:  mov ax, data;
        mov ds, ax;

      lea si, num;      Store address of list
      lea di, sml;      Store address of destination
      mov cl, siz;      Store the count

      call smallest;    Call the goddamn procedure
      jmp finnish;      Finnish it!


      smallest proc near

            mov al, [si];           Load into al the first element
            up:
                  dec cl;           Decrement cl
                  jz fns;           If zero, finnish procedure
                  inc si;           Increment si
                  mov bl, [si];     Load [si] to bl
                  cmp al, bl;       Compare, change al if bl is less
                  jng up;

                  mov al, [si];
                  jmp up;
```

```
            fns:
                  mov [di], al;     Copy al to destination
                  ret;              Return!

      smallest endp

      finnish:
            mov ah, 4ch
            int 21h

code ends
end start
```



2A) Consider the following fragment of assembly code:

```
Data segment
array dw 7,6,5,4
count dw 4
Data ends
Code segment
______
xor ax,ax
stc
mov cx,count
mov si,offset array
label1: adc ax,word ptr [si]
add si,2
loop label1
label2:
_______
_______
```

What will be the value in AX when control reaches label2? Show the calculation for all iteration for the value in AX. Write the final answer in hexadecimal.

When the control reaches label2, AX will have the value '0017h'.

ITERATION	AX	CALCULATION
1	0008	ADC AX, WORD PTR[SI] AX ← AX + [SI] + 1; Carry is set before AX ← 0000 + 0007 + 1 AX ← 0008
2	000E	ADC AX, WORD PTR[SI] AX ← AX + [SI] + 0; Carry was reset AX ← 0008 + 0006 AX ← 000E
3	0013	ADC AX, WORD PTR[SI] AX ← AX + [SI] + 0; Carry was reset AX ← 000E + 0005 AX ← 0013
4	0017	ADC AX, WORD PTR[SI] AX ← AX + [SI] + 0; Carry was reset AX ← 0013 + 0004 AX ← 0017

2B) What is 'REP'? Discuss the various types of REP.

REP or Repeat, repeats a string instruction the number of times specified in the count register, CX, or until the condition of the ZF (Zero Flag) is no longer met.

Types:

REP - Repeat MOVSB, MOVSW, LODSB, LODSW, STOSB, STOSW instructions CX times.

REPE - Repeat following CMPSB, CMPSW, SCASB, SCASW instructions while ZF = 1 (result is Equal), maximum CX times.

REPNE - Repeat following CMPSB, CMPSW, SCASB, SCASW instructions while ZF = 0 (result is Not Equal), maximum CX times.

REPZ - Repeat following CMPSB, CMPSW, SCASB, SCASW instructions while ZF = 1 (result is Zero), maximum CX times.

REPNZ - Repeat following CMPSB, CMPSW, SCASB, SCASW instructions while ZF = 0 (result is Not Zero), maximum CX times.

3A) Write down the equivalent string instructions for the following two.

i)
```
MOV AL, [DI]
CMP AL, [SI]
DEC SI
DEC DI
```

```
STD;        We're going in reverse
CMPSB;      Yep, that should do it..
```

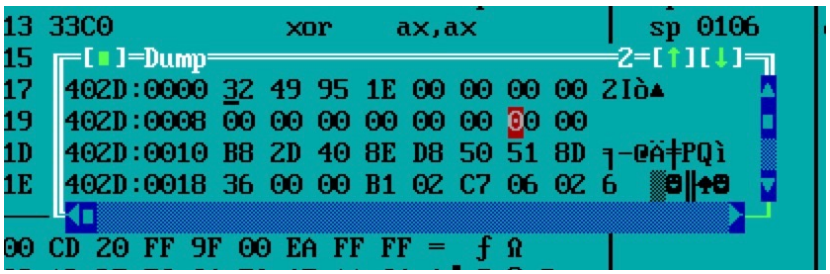
ii)
```
MOV AL, [SI]
MOV [DI], AL
INC SI
```

```
CLD;        We're going forward
MOVSB;      Again, should do it..
DEC DI;     Cause it's not incremented int the problem?
```

3B) Using MACRO, write ALP to solve $P = X^2+Y^2$ where X and Y are 8 bit numbers.

```
data segment
      xy db 32h, 49h
      p dw ?
data ends

code segment
assume ds: data, cs: code
start:      mov ax, data;
            mov ds, ax;

            ; Very frakking important, macros are defined before using them

            sqc macro src, cnt, dest            ; Define the macro

                  push ax;                      ; Push regs
                  push cx;

                  lea si, src;                  ; Load address of source
                  mov cl, cnt;                  ; Load the count

                  mov dest, 0000h;              ; Clear the dest

                  sqcalc:
                        xor ax, ax;             ; Clear AX
                        mov al, [si];           ; Load the numbers
                        mul al;                 ; Multiply by self
                        add dest, ax;           ; Add to destination
                        inc si;                 ; Increment and decrement
                        dec cl;
                        jnz sqcalc;

                  pop cx;                       ; Pop regs
                  pop ax;
```

```
            endm                                    ; end the macro

            sqc xy, 02h, p;                         ; Call the goddamn macro

            mov ah, 4ch
            int 21h

      code ends
end start
```



4A) You are stepping through the execution of an 8086 assembly language program. Shown are memory dump for vector table, a disassembled listing of the part of the program that is currently executing.

DUMP OF INTERRUPT VECTOR TABLE:

```
0000:0000 BB 08 0B 02 65 04 70 00-16 05 DA 09 65 04 70 00 ....e.p.....e.p.
0000:0010 65 04 70 00 D7 04 00 C0-85 98 00 F0 53 FF 00 F0 e.p.........S...
0000:0020 00 00 00 C9 28 00 DA 05-3A 00 DA 05 52 00 DA 05 ....(...:...R...
0000:0030 6A 00 DA 05 82 00 DA 05-9A 00 DA 05 65 04 70 00 j...........e.p.
```

LISTING OF THE PROGRAM CODE:

```
1266:0033 EB02 JMP 0037
1266:0035 46 INC SI
1266:0036 47 INC DI
1266:0037 803C00 CMP BYTE PTR [SI],00
1266:003A 7505 JNZ 0041
1266:003C 803D00 CMP BYTE PTR [DI],00
1266:003F 7412 JZ 0053
1266:0041 8A04 MOV AL,[SI]
→ 1266:0043 3A05 CMP AL,[DI]
1266:0045 74EE JZ 0035
1266:0047 7305 JNB 004E
1266:0049 B8FFFF MOV AX,FFFF
1266:004C EB07 JMP 0055
1266:004E B80100 MOV AX,0001
1266:0051 EB02 JMP 0055
1266:0053 33C0 XOR AX,AX
1266:0055 C3 RET
```

NMI INTERRUPT SERVICE ROUTINE:

```
NMIISR: PUSH AX
PUSH SI
CALL HANDLENMI ;Process the NMI, doesn't modify any
;registers or flags except AX and SI
POP SI
POP AX
IRET
```

The instruction shown in bold in the program listing is the current instruction being executed. While this instruction is executing, an NMI occurs. The NMI will be serviced before the next instruction begins executing. What is the address of the NMI interrupt service routine? Explain.

The Interrupt Vector Table is an array of DWORD entries (each entry is 4 bytes).

The NMI Interrupt uses vector 2. The offset of entry 2 in the Interrupt Vector Table is at: 2 * 4 = 8. This entry is made up of the bytes underlined above.

Each entry in the table is a SEG:OFF pair giving the CS and IP values for the entry point of the interrupt service routine.

DUMP OF INTERRUPT VECTOR TABLE:

```
0000:0000 BB 08 0B 02 65 04 70 00- 16 05 DA 09 65 04 70 00 ....e.p.....e.p.
```

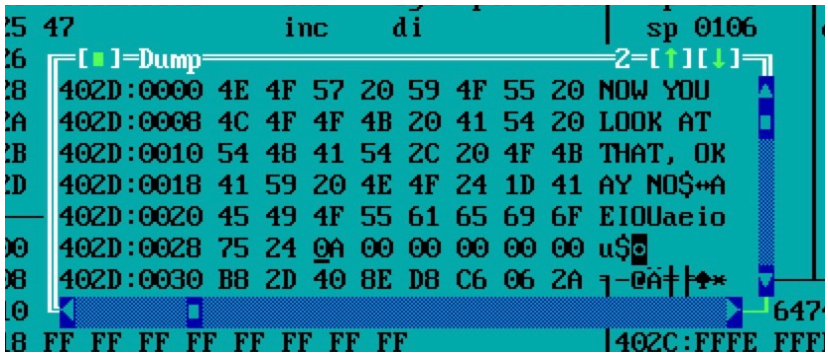
But since it's a little endian scheme, the actual address = `09DA:0516`

PS: The whole question is copied from
Washington State University Midterm Exam #1 Answer Key

4B) Write an assembly language program to count number of vowels in a given string.

```
data segment
      string db "NOW IF YOU LOOK AT THAT, OKAY NO$"
      len db 32
      vowel db "AEIOUaeiou$"
      count db ?
data ends

code segment
assume ds: data, cs: code
start:      mov ax, data;
            mov ds, ax;

            mov count, 00h;         Set count to zero

            mov cl, len;            Put length in cl
            dec cl;                 Acutal length = length - 1
```

```
            lea si, string;         Load string address
            oloop:

                  lea di, vowel;          Load vowel address
                  mov dl, 09h;            Put number of vowels to check in dl

                  iloop:
                        mov al, [si];     Move stuff to compare in
                        mov bl, [di];     al and bl, increment count if equal
                        cmp al, bl;
                        jne endil;
                        add count, 01h;
                        endil:
                              inc di;
                              dec dl;
                        jnz iloop;

                  inc si;
                  dec cl;
                  jnz oloop;

            mov ah, 4ch
            int 21h

      code ends
end start
```



5A) Identify and explain the type of call performed by each of the following instruction:

i) Call 1000h

Intra-segment direct call.

It'll take call the procedure located at address 1000h.

ii) Call word ptr [100h]

Intra-segment indirect call.

It'll dereference the word address location 100h and call it.

iii) Call dword ptr [BX+SI]

Inter-segment indirect call.

Effective calling address is taken as the content in BX and SI, then the procedure at the address is called.

(Thanks Jitesh for the answer)

5B) List the actions taken by 8086 when responding to an interrupt request. The interrupt vector table is always created in the first 1K area of the memory. Justify the statement.

When an interrupt occurs, 8086 does the following:

- Pushes the flag register onto the stack.
- Pushes a far return address `(segment : offset)` onto the stack
- Determines the cause of the interrupt and fetches a 4 byte interrupt vector from address `0 : vector * 4`
- Transfers the control to the routine specified by the interrupt vector table entry.

The 8086 can handle 256 types of INTR interrupts, each holding starting address of Interrupt Service Procedures (ISPs) taking 4 byte space each. The starting address are stored in the first 1 KB in the memory (Address 00000H to 003FFH).