V22.0201, Computer Systems Organization I Fall 1996

NYU ID: ______________ NAME: ______________

SEAT NO: ______________ PROFESSOR: ______________

Write directly on this exam question sheets. Except for True/False questions, you must justify all answers. For full credit, show all work.

1. **(8+8 POINTS)**

   Give one or two sentence explanations of these terms:

   (a) (i) TSR

   ANSWER: TSR refers to a terminate and stay resident program. It is kept in main memory upon after termination, ready to be invoked by some other program.

   (b) (ii) Bit Planes

   ANSWER: Bit Planes refers to a method of organizing the video buffer for EGA graphics. There are four parallel planes, each holding one bit per pixel that can be addressed in parallel.

2. **(5+5+5+5 POINTS)**

   Circle the correct answer (True or False). Brief reasoning can be accomodated. You receive up from 0 to -2 points for a wrong answer, and 0 points if you leave the answer blank.

   (i) **True or False:** In Two's Complement representation, even numbers are indicated by the fact that their least significant bit (lsb) is 0. NOTE: be sure to consider negative numbers.

   (ii) **True or False:** The overflow flag is useful for unsigned arithmetic operations.

   (iii) **True or False:** You IBM PC can generate a continuous tone while writing characters on the screen.

   (iv) **True or False:** The following situation will always behave properly: a procedure P is called in your main procedure and at the same time, P is also set up as an interrupt routine.

   ANSWER: (i) True (for signed and unsigned numbers),

   (ii) False (it is only useful for signed arithmetic),

   (iii) True (these are handled by different peripheral devices),

   (iv) False (you need to use IRET instead of RET in interrupt routines). We discussed this in class: if you answer "True" you get full credit if you add the following further conditions: (A)

P uses IRET instead of RET, and (B) when you call P from the main program, you first push the flags on to the stack (in addition to whatever else you need to do for calling P).

3. **(10 POINTS)**

   Give the 8-bit representations of the following two decimal numbers, writing your answer in hex. Show working for full credit.

   $$236, \quad -16$$

   ANSWER: 0ECh and 0Fh.

4. **(5 POINTS)**

   Modify this instruction so that it assembles without error:

   ```
   mov [BX], 300
   ```

   You must keep it as a single instruction. HINT: there may appear to be two choices, but the value '300' determine the correct choice.

   ANSWER: The assembler does not know whether to move a byte or a word into [BX]. But '300' means that you need a word. Hence the answer is

   ```
   mov [BX], WORD PTR 300
   ```

5. **(8+8 POINTS)**

   (i) Consider the registers BP and SP. Describe one similarity and one difference between these two registers in regards to their use in addressing modes.

   ANSWER: Similarity: both are used with the SS segment register. Difference: SP cannot be used in register indirect mode while BP can.

   (ii) Consider the instructions CMP and TEST. Describe one similarity and one difference.

   ANSWER: Both instructions are used for setting flags but otherwise have no effect on the memory or register contents. CMP is used for arithmetic comparisons (bigger, smaller or equal) while TEST is used for bitwise logical comparisons.

6. **(10 POINTS)**

   Student Quick noticed that he often saves the registers AX,BX,CX. He decides writes a procedure to simplify the task:

```
save_abc proc
  push ax
  push bx
  push cx
  ret
save_abc endp
```

Explain to Student Quick why this will not work. Also provide a correct alternative solution.

ANSWER: This approach will not work because the return from save_reg will be incorrect since the top of the stack when executing a ret statement is supposed to have the return address, not the pushed value of bx (not cx). The correct solution is to use macros:

```
save_abc macro
  push ax
  push bx
  push cx
  endm
```

and similarly for restore_abc.

7. **(15 POINTS)**

Write a macro called THREE_WAY to implement the following pseudo-Pascal code:

```
if (ax < N)
    then  cx := X
  else if (ax = N)
     then cx := Y
     else cx := Z
  end_if
end_if
```

We assume N is an unsigned number stored in a word, and X,Y,Z are also word-sized values. The macro THREE_WAY is to be invoked as

```
THREE_WAY N,X,Y,Z
```

ANSWER: You need to use local labels. Here is one solution

```
      THREE_WAY macro N,X,Y,Z
local Done
mov cx,X
```

```
cmp ax,N
jb Done
mov cx,Y
je Done
mov cx,Z
Done:
      THREE_WAY endm
```

8. **(20 POINTS)**

   Write a procedure called CONVERT which takes as input a string represented as a linked list, and outputs a copy of that string in ASCIIZ format. (An ASCIIZ string is just a series of ASCII characters followed by a numeric zero.) We want you to use the string instructions (MOVSB, etc).

   On input, assume that SI points to the linked list. Each node consists of two fields: a single byte, which is the ASCII character; and a word, which is the offset to the next node - somewhere in the DS. An offset of 0 is a Null pointer, indicating the end of the linked list. Also assume that on input DI points to a buffer of length 255 which can hold the output ASCIIZ string. You should detect an error when the linked list has more than 254 nodes.

   Thus, if before you called your procedure, the DS was somehow set up this way:

ADDRESS	CONTENTS	REMARKS
1000h	'H'	
1001h	10h	Note that the contents of a word are written
1002h	30h	as 2 bytes but in reverse order.
⋮		
2080h	'!'	
2081h	00h	
2082h	00h	
⋮		
3010h	'e'	
3011h	13h	
3012	30h	
3013	'l'	
3014	00h	
3015	40h	
3016	'o'	
3017	80h	
3018	20h	
⋮		
4000h	'l'	
4001h	16h	
4002h	30h	

And that your calling sequence looked like:

```
out_str db 256 dup (?)
 ⋮
mov si, 1000h
mov di, offset out_str
call convert
```

After this call, the first seven bytes of out_str contains the characters

'Hello!',0

ANSWER:

```
;  input:  SI points to the linked list of characters
;          DI points to an buffer of 255 characters
;  output:  an ASCIIZ string stored in [DI]
convert PROC
    save <di,ax,cx>          ; macro
;prepare for string operations
    cld                      ; set direction flag
    mov es,ds                ; assume output buffer in ds:di
    mov  cx, 255             ; max no. of chars
again:    cmp si, 0          ; null pointer?
   je  done                  ; if so, done!
   movsb                     ; move from [si] to [di] and inc. ptrs
   mov si, [si]              ; find next character in linked list
                             ; or, "stosw" followed by "mov si,ax"
   loop again
done:
   mov [di], byte ptr 0      ; move out zero to terminate string
   restore <cx,ax,di>        ; macro
   ret
convert ENDP
```

9. **(20+20 POINTS)**

Use video mode 3 (CGA text mode, 80x25 with 16 colors). We have a variable called CURR_POS storing a number from 1 to 78. This number is the position of a white vertical column. To the left of the white column, the screen is red, and the the right, it is blue. By moving the mouse, the user can move the white column to the left or right. Note that column 0 is always red and column 79 is always blue. We now ask you to implement procedures to maintain this screen setup.

You may assume an EXTRN procedure DRAW_COL that draws a column at position CURR_POS. The color of the column is given in register BL, which is set up as the attribute byte in INT 10h, function 9 (write a character with attribute). DO NOT WRITE THIS PROCEDURE. Some needed interrupts are provided below.

NOTE: in case you do not know the way to call an interrupt, you will get partial credit if you call an EXTRN procedure to do the job. Of course, you must describe any EXTRN procedure you use.

(i) Write a procedure UPDATE that checks the current mouse position. If the mouse is to the left of the white column, then UPDATE moves the white column to the left by one column. The old white column is repainted blue, of course. But if the CURR_POS is already 1, UPDATE does not do anything. Similarly, the actions of UPDATE when the current mouse position is to the right of the white column is to increment CURR_POS and move the white column to the right. Your procedure UPDATE must call DRAW_COL. I will emphasize again: you only change CURR_POS by +1 or -1 in this procedure! HINT: that the x-coordinates of the mouse is in pixel coordinates, so it must be divided by 8 in order to obtain the column number.

(ii) Write a program containing a main procedure that sets up the screen initially with the white column as column 40. Then it loops to update the position of the mouse. It exits when any mouse button is pressed. Your program should be complete in the sense that I can assemble it, link it with any external procedures and execute it.

ANSWER: (i)

```
 update proc
save_regs ax,bx,cx
mov ax,3
int 33h ; get mouse pos
mov ax,cx        ; cx is horizontal pixel pos
mov cx,3
shr ax,cx        ; divide by 8 to get column no.
mov cl,al        ; cl has column no.
cmp cl,curr_pos
je done         ; if cl=curr_pos, done!
jg right        ; if cl>curr_pos, move column to the right
 ; so mouse is to the left of curr_pos
cmp curr_pos, 1
jle done        ; do not move left of column 1, so done!
mov bl,11h      ; blue color for foreground and background
call draw_col
dec curr_pos
jmp done
 right:
```

```
     cmp curr_pos,78
     jge done        ; do not move right of col.78, so done!
     mov bl,44h      ; red color
     call draw_col
     inc curr_pos
      done:
     mov bl,77h      ; white color
     call draw_col   ; draw the white central line
     restore_regs cx,bx,ax
     ret
      update endp
```

(ii)

```
     title mouse line
     .model small
include ..\mac\basic.mac        ; use class macros
     .stack 100h
     extrn draw_col:near, update:near
     .data
     curr_pos db 40
     .code
     main proc
     initdata        ;macro
      ; get old video mode
     mov ah,0fh
     int 10h
     save <bx,ax>    ; macro
      ; set new video mode
     xor ah,ah
     mov al,3        ; mode 3
     int 10h
      ; mouse reset
     xor ax,ax
     int 33h
      ; init screen
     mov bl,44h      ; red
     mov cx,40
      red:
     dec curr_pos    ; initially curr_pos is 40
     call draw_col
     loop red
      ; next do blue half of screen
     mov bl,11h      ; blue
     mov cx,39
```

```
    mov curr_pos,79
     blue:
    call draw_col
    dec curr_pos
    loop blue
     ; now curr_pos is at 40
    mov bl,77h
    call draw_col   ; draw the white line
     mainloop:
    call update     ; do the update step
    mov ax,3
    int 33h
    test bx,0111b   ; is any mouse button pressed?
    jz mainloop
     ; clean up
    pop ax          ; restore display mode
    mov ah,0
    int 10h
    pop bx          ; restore active page
    mv al,bh
    mov ah,5
    int 10h
    exitdos         ; macro
   main endp
   ; ****************************************
   ;input:  the column number in curr_pos
   ;         the color (attribute) in the bl register
   draw_col proc
save <ax,bx,cx,dx>     ;macro
    mov   dl, curr_pos
     mov   dh, 0      ; start in row 0
     mov   cx, 1      ; one char at a time
     mov   al, ' '    ; write out spaces
   col_loop:
    ; set cursor position
            mov   bh,0
            mov   ah,2
            int   10h
    ; paint one character
            mov   bh,0          ; write the characters
            mov   ab,9
            int   10h
    ;
     inc   dh         ; next row
     cmp   dh, 24
```

```
     jbe   col_loop   ; end of screen??
restore <dx,cx,bx,ax>   ;macro
     ret
   draw_col endp
   ; ****************************************
    end main
```

RELEVANT INTERRUPTS (PROVIDED)

MS Mouse Driver (INT 33H) Calls

```
Function 0 -- Reset
Input
AX = 0
Output
AX = mouse status  -1 = installed
0 = not installed
BX = number of buttons


Function 3 -- Get Mouse Position & Button Status
Input
AX = 3
Output
BX = Button Status
   xxxx xxxx xxxx xMRL
M=middle (if present) R=right L=left
0= not pressed  1 = pressed
CX = Horizontal Mouse Cursor Position
DX = Vertical Mouse Cursor Position
(div positions by 2 for med res
 graphics; div by 8 for text mode)
```

Relevant Video Driver (INT 10H) Calls

```
Function 0 -- Select Display Mode
Input: AH=0
AL=mode number
Output: None

Function 2 -- Set Cursor Position
Input:  AH = 2
DH = row (0-24)
DL = col (0-79)
BH = 0 (the video page)
Output: the cursor is placed

Function 5 -- Select Active Display Page
Input: ah = 5
al = active display page
Output: none
```

```
Function 9 -- Write Character with Attribute
Input: AH = 9
AL = ASCII character
BL = attribute
BH = 0 (the video page)
CX = repeat count
Output: character on screen (cursor is NOT moved)

Function 0Fh -- Get Video Mode
Input: AH=0Fh
Output: AH = number of screen columns
AL = display mode
BH = active display page
```