Question 2 [21 marks] Array Programming and Subroutines

The address B800:0000h in the main memory of an 8086-based PC, is used as the starting address of an array with a specific purpose. We will call this the *display array*.

(1) The display array is directly mapped to the display screen. This means that any write to an element in this array results in an update to your screen.

(2) Each element in the array is a byte-pair that can be described as follows:

```
struct   {  char  ASCII;
            unsigned byte colour; } DISPLAY_CHAR;
```

where colour = {07h(white), 08h(grey), 09h(blue), 0A(green) …

(3) The array is 2-dimensional, and it represents a rectangular screen of 60 rows (screen height) and 80 characters per row (screen width).

DISPLAY_CHAR displayArray[60][80];

Example : Write 'A' to the upper-left corner

```
displayArray[0][0].ASCII = 'A';
```

Example : Change the rightmost character on the 2nd row to blue.

```
displayArray[1][79].colour= 09h;
```

a) [1 mark] How big is this array (total size in bytes) ?

b) [3 marks] Write an equation that calculates the offset of the character in row `r` and column `c`.

c) [5 marks] Write an ASM code fragment that fills the entire screen with green character 'A's.

d) [2 marks] In class, we've said that the Intel 8086 supports isolated I/O and have studied several examples (timer, PIC, keyboard, switches, LEDs). This display array however is a form of memory-mapped I/O because writing to this array (at memory address B800:0000) causes a character to be displayed on the screen (an output device). Suggest one reason why memory-mapped I/O would be more convenient for displays (Hint : How would the code you wrote for the previous question change?)

e) [10 marks] Write the complete ASM ***implementation*** of the following subroutine according to the provided pseudo-code. Course policies must be used.

```
boolean putString(char[] &msg, unsigned byte size, unsigned byte row )
// Print a string of "size" length on the "row", starting at column=0.
// Return false if row is too big or msg is too long to fit on 1 line.
// if (size > 80) return false;
// boolean error = false;
// for (int i=0; i<size && error != true; i++)
//    {   error = putChar( msg[i], row, i ) ;   }
// return error;
```

You are ***provided*** with the implementation of putChar() to use in your subroutine.

```
boolean putChar ( char ASCII,  unsigned byte row, unsigned byte column )
```

```
// Display a character at [row,column] and returns true;
// return false if row or column are beyond the screen size.
```

Question 3 [8 marks] Assembly Development

a) [1 mark] IP = address of the next instruction to be fetched (not the instruction currently being executed). Give one example where this definition is useful.

b) [4 marks] Below are a few possible errors messages from the assembler. For each one, identify during which pass of the assembler the message would be printed.

Unresolved Reference _____

Invalid instruction operands _____

Target for conditional jump is out-of-range _____

Incompatible operands _____

c) [3 marks] Three separate (i.e. they don't do anything useful) ASM statements follow. For each one, state whether the calculation is static (done during program build) or dynamic (done during program execution).

EQU INT_TYPE **9*4** _____

MOV ES:[**BX+2**], CS _____

MOV **msg+2**, 'A' _____

Question 4 [20 marks] Interrupt Processing

a) [3 marks] Name three functions of the Intel 8259 Programmable Interrupt Controller.

b) [3 marks] The NMI is a hardware interrupt. Write an ASM code fragment to show how you can artificially create an NMI in software. Give one reason why you may want to do this.

Question 5 [20 marks] Designing a complete application

One of the uses of robots in factories is for repetitive movements where, time after time, a sequence of precise movements is required. Suppose we have a simple robot that works as follows:

a) Whenever stationary, the robot reads the contents of its 8-bit write-only data port (at I/O address 0389h) that contains a position value written by the software of the robot controller.
b) The robot then moves to the next given position, if different.
c) The robot stops.

d) Repeat the previous steps forever

Suppose that the robot controller is built using an 8086-based system.

a) [3 marks] Assume that the software for the robot controller has a global variable (one byte) called *nextPosition*. Write an assembly code fragment that writes the contents of *nextPosition* to the robot's data port.

b) [15 marks] Our simple robot is to be used in a chocolate factory environment. Its job will be to transfer a row of chocolate bars from their molds traveling on one conveyor belt to their packaging on another conveyor belt. The system will also have a small keyboard with four buttons: (1) RESET to start the robot moving, beginning at its initial position (2) PAUSE to temporarily stop the movement (3) RESUME to continue moving from its current position (4) QUIT to power down. Inappropriate keystrokes (e.g. hit RESUME when already running) will be ignored.

Write the ***pseudo-code solution*** for a program that will cause our robot to move through a repeated sequence of positions. The required sequence of movements is encoded in an array of position values declared as follows (assume it is initialized to proper values) :

```
positions   db 10 dup (?)     ; positions to be used by the robot
```

The program must simply iterate through the array at a regular interval of 5 seconds, at the same time handling any activity on the keyboard.

Assume that the timer is programmed to interrupt every 1 second. *The solution must be completely interrupt-driven with main() simply waiting to quit. Show only the logic of the solution and the use of data variables.* No marks will be given for assembly code, for ISR installation, for programming the timer, for initializing the `positions` array, or for STI/CLI/EOI.

The PC I/O Map

LED Data port	378h	Switch Data port	379h
PIC Command Register	20h	PIC Mask Register	21h
PIC Timer 0 Data Register	40h	PIT Control Register	43h
Keyboard Data Port	60h	Keyboard Control Port	61h

An instruction summary

Transfer instructions	MOV dest, src	PUSH reg/mem	POP reg/mem	
	IN ALorAX, imm8	IN ALorAX, DX		
	OUT DX, ALorAX	OUT imm8, ALorAX		
Arithmetic Instructions :	ADD dest, src	SUB dest, src	CMP dest, src	
	MUL reg	DIV reg		

INC operand DEC operand

Logical instructions :	AND dest, src	TEST dest,src	
	OR dest, src	XOR dest,src	NOT operand

Shift Instructions:

Arithmetic :	SAR operand, 1	SAR operand, CL
	SAL operand, 1	SAL operand, CL
Logical :	SHR operand, 1	SHR operand, CL
	SHL operand, 1	SHL operand, CL

Control Flow

Unconditional :	JMP target			
Simple Branches :	JC JS JO JZ		LOOP target	
Unsigned Branches :	JA JB JAE JBE			
Signed Branches :	JG JGE JL JLE			
Subroutines:	CALL target	RET		
ISRs :	INT type	IRET	INTO	
	CLI Disable		STI	Enable