Question 1 [11 marks] Answer the following short questions. Do not exceed the allocated space.

a) [4 marks] A simple way of achieving synchronization between CPU and I/O devices is that the CPU waits a fixed amount of time for the I/O device to finish with the task assigned. Discuss in detail advantages and disadvantages of such approach.

b) [3 marks] External non-maskable interrupts are used to signal events that should serviced with tight timing constraints. Suggest at least two possible means of reducing the *latency* for servicing such interrupts in the HC12.

c) [2 marks] Explain the basic functional behavior of an AD converter.

Question 2 [11 marks]

Definition: A **program fragment** is a set of instructions that is part of a program that carries out a set of activities. It is NOT a subroutine. It DOES NOT require defining the variables to be used (unless explicitly stated), calling procedures or memory initialization information.

a) [6 marks] You have a string stored at $800-$807, the string is terminated with a NULL character (ASCII 0). Currently, there are only 4-bytes used (for instance, the string "lit" is stored). Write a program fragment to insert a character in the third byte of the string (for instance, if the character **s** is inserted in the previous example, the string "list" is created). Assume the new character to be inserted is already defined in the memory location $808.

b) [5 marks] Write a program fragment to calculate how many "A" characters (ASCII 41) are included in a string stored at memory locations starting from $800. Assume that the size of the string, N, is already defined for your use as an EQU constant and that it is no larger than 255.

Question 3 [16 marks]

Write an HC12 assembly language subroutine that calculates and returns the sum of the squares of an array of signed integers, i.e. for array A containing N integers, the subroutine calculates:

$$\sum_{i=0}^{N-1} A[i]^2$$

The sum is returned as a long_int (32-bit) value with register Y containing the high word, and register X containing the low word. The prototype for the subroutine is:

int **SumSQ** (int & **A**[ ] , int **N ,** int & **Success**)

A is the array of integers
N is the number of integers in A
Success **is returned true (non-zero) if the returned value is valid**
Success **is returned false (zero) if the returned value is not valid**

Your solution must call the SQUARE subroutine (below) to calculate the square of an integer value. Assume that SQUARE has been written previously, and follows the policies assumed

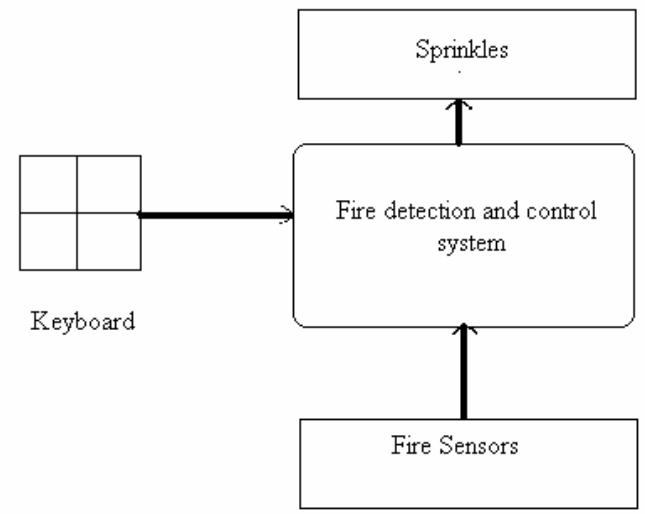
in this course for passing parameters on the stack and returning the result in a register – **DO NOT provide any code for the SQUARE subroutine**. The prototype for SQUARE is:

int **SQUARE** (int & **X**) // Returns X^2 as a long_int

Your subroutine must follow the policies assumed in this course, and must pass parameters on the stack, and pass the return type in registers.

Question 4 [14 marks]

We want to develop an application intended to detect and control fire situations in buildings. The application will use an HC12-based platform with a similar structure to the one available in the course labs. The system architecture can be described as follows:



Besides the HC12 microcontroller, the Fire Detection & Control system uses Port T of the HC12 to interface 4 digital sensors and 4 digital actuators (the sprinklers). The sensors are on the lower 4 bits of Port T where a bit is set if the corresponding sensor has detected a fire, and the bit is clear if no fire has been detected. The sprinklers are on the upper 4 bits of Port T and are activated by setting the appropriate bit.

By default, the application runs as an Automatic control system. The sensors are checked every 10 ms. A fire is identified when any of the sensors have been set for at least 30 ms. At this point, the sprinklers are turned on. The sprinklers remain on for a minimum of 5 seconds **and** until all sensors indicate no fire for 30 ms.

A keypad, similar to the one in the lab but with only 4 numeric keys (1 to 4) and an M key, letter is used for Manual control of the sprinklers. If a user presses the 'M' key, the Automatic control system is turned off instantaneously, and Manual control of the sprinkles is enabled. If the 'M' key is pressed again, Automatic control is resumed. While Manual control is enabled, the sensors are monitored but the sprinklers are not activated. The user can turn on/off the sprinklers by pressing the key corresponding to the number of sprinkler (i.e., sprinkler No. 2 is turned on if you press the key '2'). Pressing the key toggles the existing state of the sprinkler (for instance, pressing '2' when the sprinkler is off, will turn it on; pressing '2' again it will turn

it off). If Automatic control is activated, the keys pressed are ignored. Unlike the lab's keypad, this keypad used is able to generate interrupts (XIRQ, external non-masked interrupts).

The main program runs a loop displaying the present state of fire detection and sprinkler status on a LCD display. Every other part of the application is handled under control of interrupts. The initial state is no fire detected anywhere, all sprinklers off and automatic control.

(a) [10 marks] Write a sketch of the application in **pseudocode** that identifies the required state variables and describes the behavior of the main program and each of the ISRs comprising this system. Programming details - such as reading/writing ports, reading row and columns for the keypad, calling procedures and standard entry/exit code - are **not** important in this question. Continue on the back of this sheet, if needed.

Question 6 [7 marks]

Answer the following questions in relation to the execution cycle for the HC12 microcontroller:

a) [3 marks] How and when does the processor decides if an interrupt must be serviced ? Why do you think that the decision is taken in this moment?

b) [4 marks] The time sharing mechanism enables sharing a unique CPU between different processes. Explain briefly and in informal terms what is the base of this technique, and how the Process Model lets two different processes to the processor (do not include any kind of source code: focus in explaining the basic concepts and components of time sharing).

An instruction summary

Addressing modes for accessing *memory* :
op16a, oprx0_xysp,oprx5_xysp, oprx16_xysp, [D, xysp],[oprx16,xysp]

Transfer instructions TFR source, dest where source,dest = abcdxys

Compare instructions : CMPA, CMPB, CPD, CPX, CPY, CPS
Compare a register to *memory*

Simple Branches : BCC BCS BEQ BMI BNE BPL BVC BVS

Unsigned Branches : BHI BHS BLO BLS

Signed Branches : BGE BGT BLE BLT

Subroutine Branches: BSR *rel* JSR *#opr*

Bit Condition Branches : BRSET or BRCLR *memory, mask, relative*

Logical instructions : ANDA(or B) #opr ANDA(or B) *memory*
ORA(or B) #opr ORA(or B) *memory*
EORA(or B) #opr EORA(or B) *memory*
BCLR *memory* (bit clear)
BSET *memory* (bit set)

Arithmetic Instructions : ADDA #opr ADDA *memory* (similar for SUBA)
ADDB #opr ADDB *memory* (similar for SUBB)
ADDD #opr ADDD *memory* (similar for SUBB)
ADCA #opr ADCA *memory* (similar for SUB
ADCB #opr ADCA *memory* (similar for SUB
ABA (A=A+B) ABX (X=X+B) ABY (Y=Y+B)

EDIV *unsigned* (Y:D) / X => Y (remainder D)
EDIVS *signed* (Y:D) / X => Y (remainder D)
IDIV *unsigned* D / X => Y (remainder D)
IDIVS *signed* D / X => Y (remainder D)
EMUL *unsigned* D * X => Y:D
EMULS *signed* D * X => Y:D

Shift Instructions:

Arithmetic :	ASLA(or B)	ASRA(or B)	ASL *memory*	ASR *memory*
Logical	LSLA(or B)	LSRA(or B)	LSL *memory*	LSR *memory*

If you need any other instruction and you don't remember the precise syntax, make a valid assumption, describe the instruction and use it.