Architetture dei Sistemi Di Elaborazione	Delivery date: 21st December 2023
Laboratory 9	Expected delivery of lab_09.zip must include: - zipped project folder of Exercise 1 and Exercise 2

Exercise 1) – Read and convert a MORSE message.
Samuel Finley Breese Morse invented the Morse code in 1836. It is based on long and short signals (dashes and points) representing numbers and alphabet letters.

A	·−	J	·−−−	S	···	2	··−−−
B	−···	K	−·−	T	−	3	···−−
C	−·−·	L	·−··	U	··−	4	····−
D	−··	M	−−	V	···−	5	·····
E	·	N	−·	W	·−−	6	−····
F	··−·	O	−−−	X	−··−	7	−−···
G	−−·	P	·−−·	Y	−·−−	8	−−−··
H	····	Q	−−·−	Z	−−··	9	−−−−·
I	··	R	·−·	1	·−−−−	0	−−−−−

Use the LANDTIGER board and the System-On-Chip LPC1768 for implementing a system that deciphers a message in Morse code and returns:

- Its ASCII conversion.
- The total number of translated Morse symbols.

The system must work as follow:

- The message to translate is in a vector of 8-bit unsigned integers (vett_input) with a maximum length of 100 elements. The vector can only contain 0,1,2,3,4 as values.
- **The 0 is used for the dot symbol, 1 for the dash symbol, 2 indicates the character end, 3 for the space, and 4 for the end of the sentence.**

For example, the vector can be initialized as shown below:

H	O	L	A	M	U	N	D	O	1	2
00002	1111	2	0100	2	01	3	11	2	00	1	2	10	2	10	0	2	1	1	1	3	01	1	1	1	2	00	1	1	1	4

The sentence *HOLA MUNDO 12* can be represented above, and it contains 13 symbols. The space has to be considered as a symbol.

1. By pressing **KEY1**, it starts the **message reading** (in C).
   - All LEDs are switched off.
   - At the end of the message (after reading the digit 4 in vett_input), all LEDs are switched on for 3s. During this period, INT0 and KEY2 are disabled. Use TIMER0 to set up this period.

2. By pressing **KEY2**, the program starts the **message conversion in ASCII** (in Assembly).
   - During the conversion phase, the buttons **INT0** and **KEY1** are disabled.
   - The assembly function for translating the message is invoked.

The assembly function has the following prototype:

```
int translate_morse(char* vett_input, int vet_input_lenght, char* vett_output, int vet_output_lenght, char change_symbol, char space, char sentence_end);
```

The function arguments are:

- char* vett_input, Morse code vector to convert.
- int vet_input_lenght, length of the vector to convert.
- char* vett_output, ASCII output vector.
- int vet_output_lenght, length of the output vector.
- char change_symbol, the value representing the change of symbol in the input vector.
- char space, the value representing the space in the input vector.
- char sentence_end, the value representing the end of the sentence in the input vector.

The function returns the total number of converted Morse symbols into an integer variable called RES. Space must be considered as a converted symbol.
In *vett_output* there is the translation in ASCII of the input vector.

Upon returning from the ASM function:

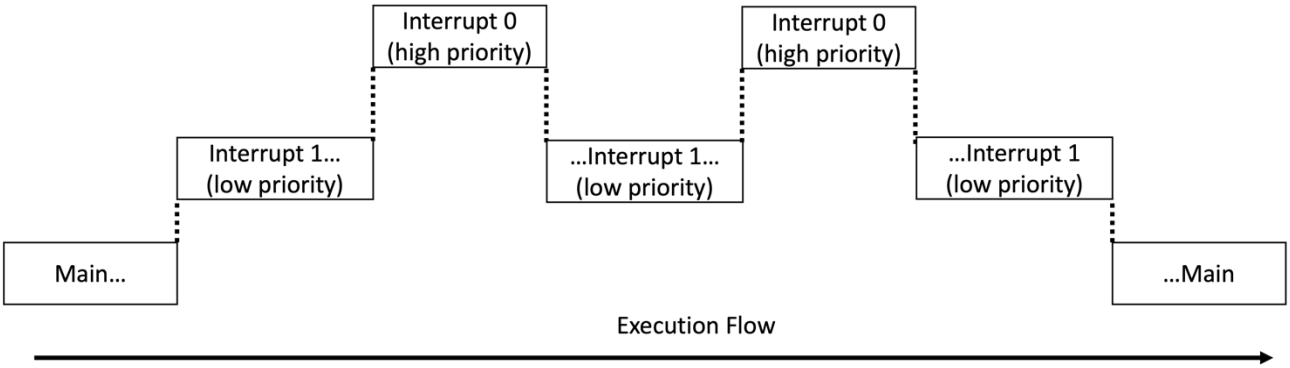
- All LEDs show the total number of converted symbols (letters, numbers, and spaces), blinking at a frequency of 2 Hz.
- Buttons **INT0** and **KEY1 must be enabled,** by pressing **INT0** the process must restart from point 1).

Hint (Morse code):

A	01	**J**	0111	**S**	000	**2**	00111
B	1000	**K**	101	**T**	1	**3**	00011
C	1010	**L**	0100	**U**	001	**4**	00001
D	100	**M**	11	**V**	0001	**5**	00000
E	0	**N**	10	**W**	011	**6**	10000
F	0010	**O**	111	**X**	1001	**7**	11000
G	110	**P**	0110	**Y**	1011	**8**	11100
H	0000	**Q**	1101	**Z**	1100	**9**	11110
I	00	**R**	010	**1**	01111	**0**	11111

Exercise 2) Interrupt an interrupt handler.

Interrupt handlers are usually kept short. They interrupt the normal flow of execution of the main program, briefly execute a specific task and then continue the main program.
In this exercise, we are going to badly program an interrupt handler to be slow and inefficient. The goal of this exercise is to have some higher level interrupts in the middle of the execution of the slow and inefficient interrupt handler and to count the number of such interruptions.



To achieve this goal, you are going to use a global variable and two timers, **Timer 2 (high priority)** and **Timer 3 (low priority)**:

- The global variable ("IntCount") contains the number of times the Timer 3 interrupt handler has been interrupted during its last execution. It is reset every time at the end of Timer 3 handler. It is incremented by 1 every time Timer 2 interrupt handler is executed as detailed later.

- **Timer 2** will just increase the value of IntCount by one unit.

- **Timer 3** will have a slow and inefficient interrupt handler. **In this timer you must waste time.** You can achieve this result you can create a "for" loop with a huge number of repetitions. Let's call X the number of loops of your "*for cycle"*. You can play with X to obtain slower execution. Be prepared to use very large numbers. It doesn't matter what you place inside the loop (even a NOP or nothing is fine) if you waste time.
  At the beginning of its execution, Timer 3 handler should reset the variable "IntCount" to 0.

  At the end of the execution of its handler, Timer 3 should display on the LEDs the value of the variable "IntCount" and then reset it (without turning off the led).
  - If the value of "IntCount" cannot be represented on the 8 LEDs as a signed integer, all LEDs are ON.

Timer 2 will have a higher priority than Timer 3. **Timer 3 handler should be triggered every 2 seconds.** Play with the period of Timer 2 and with the number of loops (X) to modify the number of times that Timer 3 gets interrupted and report the results in the following table (hint: use breakpoint to debug and stop the execution of the program just before showing the value of "IntCount" on the LEDs or simply take notes on the value displayed on the led.)

Number of loops (X)	Timer 2 period	IntCount value
10000000	1sec 0x01D7840	0x000000CD=>15
1000	1sec	0x00000019=>2
1000	0.001sec	0x0000007D0 =>2000
10000000	0.1sec	0X00000096=>150