Programming the 8051 Microcontroller for the EdSim51 Simulator

Lesson 1

Introduction

The Intel 8051 microcontroller:
- See courses 8-9.

SDCC compiler:
- Command line tool.
- Download from https://sourceforge.net/projects/sdcc/files/ the version corresponding to your OS.
- Install it, following all recommendations from the installer.
- Compiling a source file:
  sdcc *file.c*
  successful compilation results in the creation of several files, one of which is *file.ihx*

EdSim51 simulator:
- Download the ZIP archive from https://www.edsim51.com/ and unpack it.
- To run it, simply double-click on the JAR file (requires a Java Virtual Machine to be installed).
- Programs must be inserted in the upper-middle window of the application.
- The lower window of the application shows the status of the external peripheral devices.
- To see the external peripheral devices and the way they are connected to the microcontroller, look at the large image at https://www.edsim51.com/examples.html.

Writing and running C programs for the simulator:
- Each program must start with "`#include <8051.h>`". On the other hand, the standard libraries, like `<stdio.h>`, which we are used to work with, are not available here.
- The core of function `main` is usually an infinite loop, which executes repeatedly the same operations. Various initializations can be performed before the infinite loop, if necessary.
- It is recommended not to use many variables (either local or global), as the microcontroller memory is not very large.
- The four general-purpose ports of the microcontroller can be accessed, either in reading or writing, by using the predefined names `P0`, `P1`, `P2`, `P3`.
- Individual bits of the ports can also be accessed; e.g., `P0_3` denotes bit 3 of port `P0`.
- After the source file has been saved, it is compiled with SDCC as shown above.
- To run the program in the EdSim51 simulator, push the *Load* button (on top of the program window) and open the file with the same name of the source file, but with extension *.ihx*; as a result, the compiled code (i.e., machine code) is loaded.
- In the beginning of the machine code there are two jump instructions, which usually look as below:

```
LJMP 0006H
LJMP 0062H
```

- Change the first of these two instructions to be identical to the second one:

```
LJMP 0062H
LJMP 0062H
```

- Only then should the *Run* button be pressed.
- When a program is long and slow, it could take time to run it and watch the results. It is possible to change the control *Update Frequency* (upper-left window of the simulator) to a value higher than 1 - either by picking a value from the predefined list of by just writing another value. This will make the simulator show the evolution of the system not after every instruction, but after every *n* instructions (where *n* is the update frequency).

All subsequent lessons will be about using the EdSim51 simulator and writing C programs for it.

Caution: sometimes the SDCC compiler does not generate the correct code, especially for multiplication and division operations, so the output may be wrong. Nevertheless, it is a useful tool and will be used during these lessons.

Problems

1. Write a C program that continuously alternates the LEDs which are turned on: the odd-indexed ones (i.e., 1, 3, 5, 7) and the even-indexed ones (i.e., 0, 2, 4, 6).

Indications:

The LEDs are connected to port `P1`. In order to turn on/off a certain LED, the corresponding bit of port P1 must be assigned value 0/1 (not the opposite).
It is possible to assign each bit of a port in a separate instruction (e.g., `P1_0=1;` etc.). However, in most cases it is easier to assign all bits of the port in a single instruction. We remind that the C language allows writing base 2 constants by using the `0b` prefix.
For example, the following instruction turns on LEDs 0, 2, 4, 6 and off LEDs 1, 3, 5, 7, all at once:

```
P1=0b10101010;
```

2. Write a C program that turns the LEDs on and off based on the values of switches SW0...SW7 (inputs from the user).
Indications:
The switches are connected to port `P2`: each switch corresponds to a bit in `P2`. By reading a certain bit of `P2`, we get value 0 if the corresponding switch is closed and 1 if the switch is open.

3. Write a C program that turns on LED 0, then LED 1, ..., then LED 7. At each moment, a single LED is turned on. When LED 7 is reached, the program must go back, by turning on LED 6, then LED 5, ..., and so on.

Lesson 2

Problems

1. Write a C program that controls the motor, by making it continuously change its spinning direction. That is, the motor spins clockwise for a time *T*, then counter clockwise for another time *T*, and so on.
Indications:
In order to control the motion of the motor, one must write the right values to bits 1 and 0 of port `P3`:
- If the bits have the same value (either 0 or 1), the motor stops.
- If the bit values are different, the motor spins; the spinning direction depends on which bit has value 0 and which has value 1.
- In conclusion, the values that must be written to port `P3` for making the motor spin are 1 and 2, respectively; they correspond to binary combinations 01 and 10, respectively, on bits 1 and 0.
- The time *T* is actually a delay and can be implemented as an empty loop. It is not important what is the value of the delay, but is recommended not to make it very long, because the program will become slow.

2. Write a C program that controls the spinning direction based on the value of switch SW0.

3. Write a C program that, in addition to the previous one, also decides whether the motor stops or spins, depending on the value of switch SW1:
- If SW1 has value 0, the motor stops.
- If SW1 has value 1, the motor spins and its spinning direction is given by the value of SW0.

Lesson 3

Problems

1. Write a C program that displays a 4-digit number (positive integer) on the 7-segment displays.
Indications:
The display control is more complicated:
- The configuration to be displayed is written on port `P1`.
- The display on which the configuration is shown is controlled by bits 3 and 4 of port `P3` and, additionally, by bit 7 of port `P0`.

While all displays get their values from port `P1`, at most one of them can be turned on at any moment, due to the decoder:
- If bit 7 of port `P0` has value 0, all displays are turned off (the decoder has no active output).
- Otherwise, 3 and 4 of port `P3` make a 2-bit number which indicates which decoder output is active - that is, which display is turned on. For example, if the 2-bit number is 11 (which means 3 in base 2), then display 3 is turned on.

Thus, in order to display multiple digits, one must cycle through those digits, always taking care that the decoder is controlled in such a way that each digit is sent to the right display.
For a given number, the base-10 digits are determined by repeatedly dividing the number by 10; the digits are the remainders of the division-by-10 operations.

2. Write a C program that displays a multi-digit number, as above, but ignores non-significant digits. For example, if the number is 698, display 3 is never turned on (in the previous case, is shows value 0).

Lesson 4

Problems

1. Write a C program that reads a key from the keypad and displays it on one of the 7-segment displays. The key on the keypad must have already been pressed when the program starts and the situation on the keypad is not changing during program execution.
Indications:
The keypad is connected to the microcontroller in a special manner, which leads to a more sophisticated scan procedure:
- Each row of the keypad is connected to one of bits 0..3 of port `P0`.
- Similarly, each column of the keypad is connected to one of bits 4..6 of port `P0`.

To detect that a key is pressed, one must first write value 0 on the corresponding keypad row (keeping the other rows on value 1) and subsequently read the value on the corresponding keypad column. For example, we can detect that key '6' has been pressed by writing 0 on the second-highest row (i.e., bit 2 on port `P0`) and then testing if value is 0 on the rightmost column (i.e., bit 4 on port `P0`).
As it is not known in advance which key is pressed (if any), the whole keypad must be scanned; that is, all rows and columns, sequentially.

2. Write a C program that collects 4 keys pressed by the user, one after the other; when all 4 keys have been read, they are sent to the 7-segment displays.
Indications:
In this case, the keypad scanning must be done as bellow:
- Wait for a first key to be pressed; when this happens, memorize it.
- Wait for the key to be released.
- Wait for a second key to be pressed and memorize it.
- ... And so on, until 4 keys have been memorized.

Lesson 5

Problems

1. Write a C program that turns on all LEDs individually, one LED at a time, then moves to the next LED and so on. Moving from one LED to the next is done at a constant pace.
Indications:
The most difficult problem here is measuring the time. As program loops do not provide an accurate and reliable control of the elapsed time, it is necessary to use the built-in timers (see the 8051 data sheet).
Timer control (examples below are for timer 0; the commands are similar for timer 1):
- Set the working mode of the timer by writing the proper value into register `TMOD`. For our purposes, the value to be written into `TMOD` is 1.
- Load the desired initial value of the timer (2 bytes) into registers `TH0` (upper byte) and `TL0` (lower byte).
- Start/stop the timer by assigning value 1/0 to bit `TR0`. When the timer is started, its value is incremented at a constant rate; when the timer reaches value 0 (overflow), bit `TF0` is set.

There are two ways of detecting timer overflow:
a) Test bit `TF0` repeatedly until it becomes 1.
b) Configure the timer to generate an interrupt on overflow.

In case a), inside the infinite loop in function `main`, we can use an inner loop in which we do nothing but wait for bit `TF0` to become 1. When this happens, bit `TF0` must be explicitly reset by software, thus preparing it for the next iteration of the main loop.

In case b), some operations are required in order to control the interrupt system:
- Bit `EA` can be used to disable all interrupts, resetting it. When `EA` is set, each interrupt source is controlled by its specific interrupt control bit.
- Bits `ET0` and `ET1` are used to enable/disable (1/0) interrupts generated by timers 0 and 1, respectively.
- When a timer is allowed to generate an interrupt, the corresponding interrupt service routine must be written. Such a routine is written like a function, but the syntax requires some additional information. For example:

```
void func(void) __interrupt (1) {...}
```

The header above tells that function `func` is automatically called whenever timer 0 generates an interrupt. For timer 1, the value following the keyword `__interrupt` must be 3. Keep in mind that such a function is never called explicitly in our code.
- Communication between the interrupt service routine and function `main` is often done through global variables. All variables that are accessed both in the interrupt service routine and in function `main` must be preceded, when declared, by the keyword `volatile`.

2. Write a C program that periodically scans the keypad for pressed keys. The periodic action is performed by using timer interrupts, as above.