inżynier przyszłości
Wzmocnienie potencjału dydaktycznego
Politechniki Poznańskiej

Adrian Kliks

INSTRUCTIONS FOR 8051 MICROCONTROLLER

Materiały dydaktyczne dzięki dofinansowaniu ze środków
Europejskiego Funduszu Społecznego dystrybuowane są bezpłatnie

KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY

Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego

Poznan University of Technology
Faculty of Electronics and Telecommunications

8051 MICROCONTROLLER

Simple arithmetic operations
&
Microcontroller stack

Poznań 2014

1. Simple arithmetic operations

In order to efficiently perform fundamental mathematical and logical operations, such as addition, multiplication, logical AND, exclusive-OR, the 8051 microcontroller is equipped with the Arithmetic-Logic Unit (ALU). This entity, applied in the 8051 microcontroller, is able to perform operations on one-byte numbers (operands). There exist several instructions (commands) that can be used for the realization of simple mathematical operations.

Command	Meaning		Example
ADD	Addition	A←A + □	ADD A, B ADD A, #1
ADDC	Addition with carry	A←A + □ + C	ADDC A, B
MUL	Multiplication	A*B	MUL A, B
SUBB	Subtraction	A←A – □ - C	SUBB A, #1
INC	Increment		INC A
DEC	Decrement		DEC A
DIV	Division	A/B	DIVA, B
SWAP	Swap the position of four higher bits with the position of four lower bits		SWAP A
XCH	Exchange the values of the operands		XCH A, B
DA	Decimal adjust for addition (or Decimal Adjust Accumulator)		DA A

Moreover, there are numerous subroutines that are used for obtaining data from the matrix keyboard and displaying information on the LCD screen. Exemplary functions are listed in the table below. Please note that the programs can modify some of the registers such as A (accumulator), B, DPTR, or PSW (Program Status Word) register; they also use R0 and rarely R2.

In order to invoke a subroutine, use the LCALL command, e.g., LCALL LCD_CLR.

Subroutine	Meaning
LCD_CLR	Clear the LCD screen
WRITE_HEX	Write in hexadecimal form the data stored in the accumulator on the LCD screen
WRITE_TEXT	Write the text indicated by the DPTR register on the LCD screen
WRITE_DATA	Write the data stored in the accumulator on the LCD screen in the form of a sign (i.e., taking into account, e.g., ASCII formatting)
WRITE_INSTR	Send the instructions from the accumulator to the LCD screen

WAIT_ENTER	Display on the screen the phrase „PRESS ENTER...” and wait for pressing the [Enter] button
WAIT_KEY	The executed program will wait until any key from the matrix keyboard is pressed by the user. The number of the pressed key is stored in the accumulator.
WAIT_ENT_ESC	Wait for pressing the [Enter] or [Esc] key and return C=0 when [Enter] was pressed, and C=1 for [Esc] key
GET_NUM	Get the number in the BCD (binary-coded decimal) form (4 digits) from the matrix keyboard and store it in the memory address @R0; the end of digit entering is recognized by pressing [Enter] (C=0), or after the 4th digit [Esc] (C=1)

In the DSM-51 platform, the keys [0] ... [9] in the matrix keyboard are numbered as 0-9, while the rest has the values from 10 to 15, i.e., 0AH ... 0FH in the hexadecimal form. The order of the keys is the following: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, ←, →, ↑, ↓, ESC, Enter.

2. Microcontroller stack

The utilization of the stack in any microcontroller belongs to the fundamental operations performed during program execution. Such an option also exists in the 8051 microcontroller, with the possibility of storing some data on the stack. The stack itself can be interpreted as the memory space used for storing some data in the form of a dedicated collection, similarly to the stack of books on the bookstand or desk. The new data can be only stored on the top of the stack, and the top stack entries have to be first removed in order to get the value from a specific stack position. Storing new data is realized by invoking the PUSH operation, whereas reading from the stack is done by the POP command. The pair PUSH-POP can also be interpreted as the Last-In-First-Out mechanism.

In order to guarantee the proper functioning of the stack, the developer has to allocate the contiguous fragment of the RAM memory which cannot be used for other purposes. The current top position of the stack is indicated by the Stack Pointer (SP) register. By default, after the RESET signal, the SP is set to 7. It can be modified by invoking the following operation: MOV SP, #60H

Please note that the stack can be used explicitly by the user, but it is also used implicitly every time when the subroutine is called. During this operation, the return address (i.e. the address of the next instruction to be executed by the processor after finishing all commands from the subroutine) and other registers are stored in the stack.

3. Exercise Flow

1. Write a program that will add two numbers provided by the user via the matrix keyboard (please use the WAIT_KEY subroutine). The signs + and =, as well as the result of the addition should be displayed on the LCD screen. For example, the execution of the program should be 3+5=8. Note: for displaying numbers and data, use appropriate functions from the list provided in the instructions.
2. Repeat the above exercise for subtraction. Please note that negative numbers have to be displayed in the form understandable by humans (e.g. 3-4=-1, 3-1=2).
3. Write a program that will convert a binary number (lower than 100) to the number expressed in BCD form. In order to display the result, store the converted number into the accumulator and use an appropriate function.
4. Write a code for the multiplication of two one-byte numbers read from the matrix keyboard. Display the result on the LCD screen.
5. Write a program that will add numbers provided one-by-one by the user. Each time the number is provided (i.e. the user presses the button) the result has to be updated. The numbers shall be presented in the BCD form. For example

   User pressed 3, Displayed: 3+
   User pressed 5 Displayed: 3+5=8
   User pressed 4 Displayed: 8+4 = 12
   User pressed A Displayed: 12 + 10 = 22

   and so on.
6. Write a program for the multiplication of numbers represented in the BCD.
7. Write a program that will use the stack and your own subroutines for displaying the numbers provide by the user and stored in accumulator first in decimal and then in hexadecimal form. Please note that a one-byte number in the accumulator can take values from 0 to 255. In your own subroutine, please perform the hexadecimal to BCD conversion.

   Note: in order to write your own subroutine, one has to create a label (e.g. CONV_BCD) after the end of the main program, followed by the code of that subroutine. Then, the body of the subroutine has to be provided, ending with the RET command. In order to invoke the subroutine, use the LCALL command, i.e. LCALL CONV_BCD. Of course, ACALL can be used as well.

   Example:

```
                main code
                main code
                main code
                LCALL CONV_BCD
                end of main code

        CONV_BCN:
                body of the subroutine
                body of the subroutine
                RET
```