Microprocessor systems EMISY

Principles of assembler language for 8051
Lecture 6

Semester 20L – Summer 2020

© Maciej Urbanski, MSc

email: M.Urbanski@elka.pw.edu.pl

WUT

Important remark

Low and high level programming languages

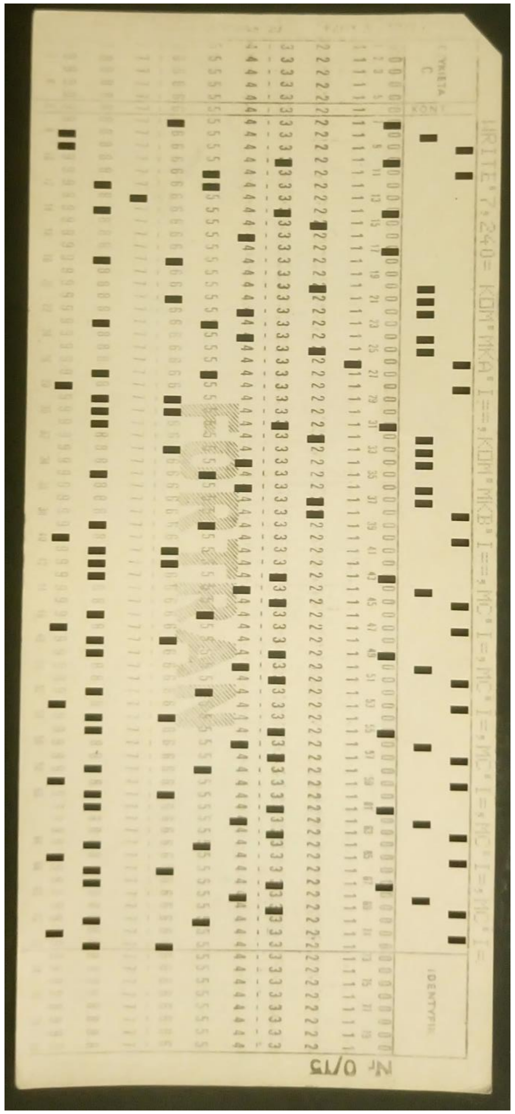
At first there was nothing...

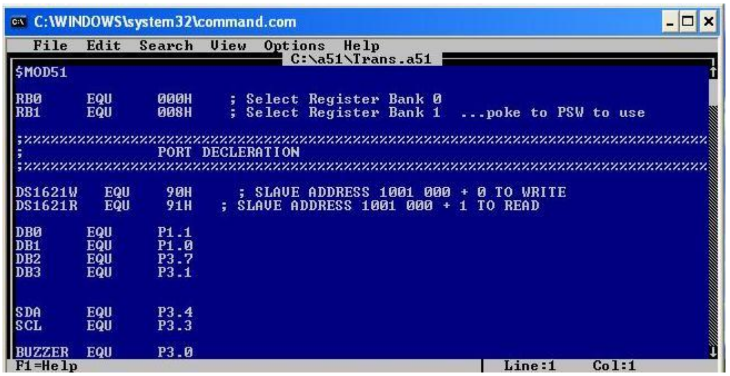
And then the 'analog programming' appeared. Programs were 'drilled' , bit after bit in paper punched cards. Each column represented one bit.
This type of memory is rather fragile – after several read operations card was destroyed.

The next step was to add user interfaces to computers – keyboards and displays.
This helped in development of first programming languages that allowed to create programs using numbers, but also words (aliases), thus enabling code returns, jumps, loops, etc.

Many microprocessor manufacturers started to design their own low-level languages.
That is how **assembly languages** and assemblers (linkers) were created. This was a huge step forward, but still requiring deep knowledge on specific microprocessor architecture.

There was a need to develop universal, simple programming languages, independent (or at least partially dependent) of hardware. The solution was to create a high-level programming language, which would be compiled – translated to machine instructions by compiler. Examples of such languages are BASIC, FORTRAN, C, C++, etc.





C:\WINDOWS\system32\command.com
File Edit Search View Options Help
C:\a51\Trans.a51

```
$MOD51

RB0      EQU     000H    ; Select Register Bank 0
RB1      EQU     008H    ; Select Register Bank 1  ...poke to PSW to use

;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;                PORT DECLERATION
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DS1621W    EQU    90H      ; SLAVE ADDRESS 1001 000 + 0 TO WRITE
DS1621R    EQU    91H    ; SLAVE ADDRESS 1001 000 + 1 TO READ

DB0      EQU     P1.1
DB1      EQU     P1.0
DB2      EQU     P3.7
DB3      EQU     P3.1


SDA      EQU     P3.4
SCL      EQU     P3.3

BUZZER   EQU     P3.0
F1=Help                                                  Line:1     Col:1
```

Image source : http://www.8051projects.info/threads/how-to-create-hex-file-with-8051-assembler.697/

Assembler and assembler (assembly) languages

- Assembler is a program that translates assembly language to machine code during assembly process.
- It is a kind of compiler for assembly language.

- First assembler was designed by Konrad Zuse in 1945 and was used to create machine code for computers calculating wing parameters in German planes, mostly fighters and bombers.

  His company existed until 1969, when it became part of Siemens AG
- First assembly language was created in 1947 by Kathleen Booth for ARC2 computer, in Princeton, USA

- Nowadays assembly languages are mostly used to create the most critical parts of firmware, especially when time dependencies and code execution control is required

Assembly language principles

- Assembler program elements are:
  - Instructions - direct commands for the microcontroller
  - Directives - direct commands for the assembler
  - Comments - direct help for designers and reviewers

- Assembler instruction consists of:
  - Mnemonics - symbolic labels for machine instructions
  - Arguments - can be numeric value, or equation made of constants, symbols and operators

- It is possible (and very useful) to use label to mark a part of the code – it may represent macroinstruction

- Assembler reads instructions one by one, in order given in program. It does not goes back to previously read commands.

- During instruction processing assembler reads the instruction – its mnemonic and arguments. Then it analyses the addressing mode for the instruction and using code tables it calculates command length, command code and memory address to store it.

Directives, comments and constants

- Directives are used to control the behaviour of the assembler.

- The most useful directives are:

• ORG	-	it defines the start address in microcontroller memory. (during laboratory classes student's programs **must** start at 8000H)
• EQU	-	used to define constants
• DEFB	-	used to store constant in memory (one cell)
• DEFW	-	used to store constant in memory (two cells)
• END	-	used to mark the end of code

- Comment is any sign that is preceded by semicolon sign, star sign,
  or double slash (like in C)

Assembly language program structure

- This is a simple assembler program written for AT89S4051
- It blinks LED connected to P3.0 pin

Example of immediate addressing – argument is a constant value

```
start:  mov     0xb0,       #0x00   //write 0 to P3 - all turned off
        mov     r1,         #0xff   //write 255 to R1
L1:     mov     r0,         #0xff   //write 255 to R0
L2:     djnz    r0,         L2      //decrement and jump L2 if R0 is not zero
        djnz    r1,         L1      //decrement and jump L1 if R1 is not zero
                                    //primitive way of introducing delay

        mov     0xb0,       #0x01   //write 1 to P3 - light up the LED
        mov     r1,         #0xff   //write 255 to R1
L3:     mov     r0,         #0xff   //write 255 to R0
L4:     djnz    r0,         L4      //decrement and jump L4 if R0 is not zero
        djnz    r1,         L3      //decrement and jump L3 if R1 is not zero
                                    //primitive way of introducing delay

        sjmp    start               //go back to start - infinite loop
```

Label

Example of „for” loop

Short jump to address given by start: label

Code source: https://www.quaxio.com/programming_an_at89s4051_with_an_arduino/

8051 Assembly language function description

- Command list for 8051 microcontroller consists of:
    - 111 instructions
        - 49 one byte instructions
        - 45 two byte instructions
        - 17 three byte instructions
- 8051 microcontrollers use following addressing modes:
    - Register addressing – instruction argument is stored in one of general purpose registers R0 to R7 from active register bank
    - Direct addressing – argument is a direct address of proper SFR register, lower internal RAM byte or bits available for direct addressing
    - Immediate addressing – argument is a constant
    - Indirect addressing (with/without offset) – argument address is stored in R0 or R1 registers. It is used for stack operations

- Command list includes instructions allowing data transfer, arithmetical and logic operations and code execution.

- 8051 microcontroller needs 12 clock cycles to execute instruction (machine cycle). There are some instructions that require 48 cycles (MUL and DIV)

- **Please refer to 8051 instruction list (available in course materials)!!**
- **For mid-term test the knowledge of 8051 instruction list will be required.**
- **NOT ALL of 8051 instructions are given in this lecture!!**

All the instruction descriptions are taken from Atmel 8051 Microcontroller Instruction Set.

8051 Assembly language function description

ACALL - Absolute call

ACALL unconditionally calls a subroutine located at the indicated address. This instruction increments the PC (Program Counter) twice to obtain the address of the following instruction, then pushes the 16-bit result onto the stack and increments the Stack Pointer twice.

Example: ACALL SUBRTN

LCALL - Long call

LCALL calls a subroutine located at the indicated address. The instruction adds three to the program counter to generate the address of the next instruction and then pushes the 16-bit result onto the stack, incrementing the Stack Pointer by two.

Example: LCALL SUBRTN

8051 Assembly language function description

ADD - addition

ADD adds the byte variable indicated to the accumulator, leaving the result in accumulator. The carry and auxiliary-carry flags are set, respectively if there is a carry-out from bit 7 or bit 3, and cleared otherwise. When adding unsigned integers, the carry flag indicates an overflow occured.

Example:	ADD A,R0	this causes that value stored in R0 is added to the accumulator
	ADD A,#10	this causes that decimal 10 is added to the accumulator
	ADD A,@R0	this causes that value stored at address given in R0 is added to the accumulator

ADDC - add with carry

ADDC simultaneously adds the byte variable indicated, the carry flag and the accumulator contents, leaving the result in accumulator.

DA - decimal-adjust accumulator for addition

DA A adjusts the eight-bit value in the accumulator resulting from the earlier addition of two variables (each in packed-BCD format), producing two four-bit digits. ADD or ADDC may be used to perform the addition.

8051 Assembly language function description

DEC - Decrement

DEC byte decrements the variable indicated by 1. 00H underflows to 0FFH. No flags are addected.

Important! When this instruction is used to modify an output port, the value used as the port data will be taken from the output data latch, not the input pins.

SUBB - Subtract with borrow

SUBB substracts the indicated variable and the carry flag together from the accumulator, leaving the result in the accumulator. SUBB sets the carry (borrow) flag if a borrow is needed for bit 7 and clears C otherwise.

Example: SUBB A,R2	This subtracts value in R2 from the accumulator, leaving result in accumulator.

INC - **Increment**

INC increments the indicated variable by 1. 0FFH overflows to 00H. No flags are affected. Three addressing modes are allowed: register, direct or indirect.

8051 Assembly language function description

MUL - Multiply

MUL AB multiplies the unsigned 8-bit integers in the accumulator and register B. The low-order byte of the 16-bit product is left in the accumulator and the high-order byte in B. If the product is greater than 0FFH the overflow flag is set, otherwise it is cleared. The carry flag is always cleared.

This instruction requires many resources and needs 48 clock cycles.

DIV - Divide

DIV AB divides the unsigned eight-bit integer in the accumulator by the unsigned eight-bit integer in register B. The accumulator receives the integer part of the quotient, register B receives the integer remainder. The carry flag and OV flags are cleared.

The overflow flag will be set when dividing by zero occurs.

This instruction requires many resources and needs 48 clock cycles.

8051 Assembly language function description

AJMP - Absolute jump

AJMP transfers program execution to the indicated address, which is formed at run-time by concatenating the high-order five bits of the PC (after incrementinc PC twice), opcode bits 7 through 5, and the second byte of the instruction.
<u>The destination must be within the same 2kB block of program memory (page) as the first byte of the instruction following AJMP.</u>

LJMP - Long jump

LJMP causes an unconditional branch to the indicated address, by loloading the high-order and low-order bytes of the PC (respectively) with the second and third instruction bytes. The destination may therefore be anywhere in the full 64K program memory address space. No flags are affected.

SJMP - Short jump

Program control branches unconditionally to the address indicated. The branch destination is computed by adding the signed displacement in the second instruction byte to the PC, after incrementing the PC twice. Therefore, the range of destinations allowed is from 128 bytes preceding this instruction 127 bytes following it.

8051 Assembly language function description

CJNE - Compare and jump if not equal

CJNE compares the magnitudes of the first two operands and branches if their values are not equal. The branch destination is computed by adding the signed relative-displacement in the last instruction byte to the PC, after incrementing the PC to the start of the next instruction. The carry flag is set if the unsigned integer value of <dest-byte> is less than the unsigned integer value of <src-byte>; otherwise, the carry is cleared. Neither operand is affected. The first two operands allow four addressing mode combinations: the Accumulator may be compared with any directly addressed byte or immediate data, and any indirect RAM location or working register can be compared with an immediate constant.ACALL unconditionally calls a subroutine located at the indicated address.

DJNZ - Decrement and jump if not zero

DJNZ decrements the location indicated by 1, and branches to the address indicated by the second operand if the resulting value is not zero. An original value of 00H underflows to 0FFH. No flags are affected. The branch destination is computed by adding the signed relative-displacement value in the last instruction byte to the PC, after incrementing the PC to the first byte of the following instruction. The location decremented may be a register or directly addressed byte.

8051 Assembly language function description

JB - Jump if bit is set

If the indicated bit is a one, JB jump to the address indicated; otherwise, it proceeds with the next instruction. The branch destination is computed by adding the signed relative-displacement in the third instruction byte to the PC, after incrementing the PC to the first byte of the next instruction. The bit tested is not modified. No flags are affected.
ACALL unconditionally calls a subroutine located at the indicated address.

JBC - Jump if bit is set and clear bit

If the indicated bit is one, JBC branches to the address indicated; otherwise, it proceeds with the next instruction. The bit will not be cleared if it is already a zero. The branch destination is computed by adding the signed relative-displacement in the third instruction byte to the PC, after incrementing the PC to the first byte of the next instruction. No flags are affected.

JC - Jump if Carry is set

If the carry flag is set, JC branches to the address indicated; otherwise, it proceeds with the next instruction. The branch destination is computed by adding the signed relative-displacement in the second instruction byte to the PC, after incrementing the PC twice.
No flags are affected.

8051 Assembly language function description

JNB - Jump if bit is not set

If the indicated bit is a 0, JNB branches to the indicated address; otherwise, it proceeds with the next instruction. The branch destination is computed by adding the signed relative-displacement in the third instruction byte to the PC, after incrementing the PC to the first byte of the next instruction. The bit tested is not modified. No flags are affected.

JNC - Jump if carry is not set

If the carry flag is a 0, JNC branches to the address indicated; otherwise, it proceeds with the next instruction. The branch destination is computed by adding the signal relative-displacement in the second instruction byte to the PC, after incrementing the PC twice to point to the next instruction. The carry flag is not modified.

JNZ - Jump if the accumulator is not zero

If any bit of the Accumulator is a one, JNZ branches to the indicated address; otherwise, it proceeds with the next instruction. The branch destination is computed by adding the signed relative-displacement in the second instruction byte to the PC, after incrementing the PC twice. The Accumulator is not modified. No flags are affected.

JZ - Jump if the accumulator is zero

If all bits of the Accumulator are 0, JZ branches to the address indicated; otherwise, it proceeds with the next instruction. The branch destination is computed by adding the signed relative-displacement in the second instruction byte to the PC, after incrementing the PC twice. The Accumulator is not modified. No flags are affected.

8051 Assembly language function description

CLR - Clear bit or accumulator

CLR A clears the Accumulator (all bits set to 0). No flags are affected.

OR

CLR bit clears the indicated bit (reset to 0). No other flags are affected. CLR can operate on the carry flag or any directly addressable bit.

CPL - Complement accumulator or bit

CPL logically complements each bit of the Accumulator (one’s complement). Bits which previously contained a 1 are changed to a 0 and vice-versa. No flags are affected.

OR

CPL bit complements the bit variable specified. A bit that had been a 1 is changed to 0 and vice-versa. No other flags are affected. CLR can operate on the carry or any directly addressable bit.

ANL

8051 Assembly language function description

MOV - Move byte variable (or bit data)

The byte variable indicated by the second operand is copied into the location specified by the first operand. This is by far the most flexible instruction. Fifteen combinations of source and destination addressing modes are allowed

Examples:

MOV R0, #30H	Move the value 30H to R0 (write 30H to R0)
MOV A, @R0	Move the value from address given in R0 to the accumulator
MOV R1, A	Move the value from the accumulator to R1
MOV @R1, A	Move the value from the accumulator to the value from address given in R0
MOV P1.3, C	Move the value of the Carry flag to pin 3 of port 1
MOV DPTR, #1234H	Move the value 1234H (16 bits) to DPTR register. This is the only instruction which moves 16 bits of data at once.

8051 Assembly language function description

MOVC - Move code byte

The MOVC instructions load the Accumulator with a code byte or constant from program memory. The address of the byte fetched is the sum of the original unsigned 8-bit Accumulator contents and the contents of a 16-bit base register, which may be either the Data Pointer or the PC. In the latter case, the PC is incremented to the address of the following instruction before being added with the Accumulator; otherwise the base register is not altered. Sixteen-bit addition is performed so a carry-out from the low-order eight bits may propagate through higher-order bits. No flags are affected.

MOVX - Move External

The MOVX instructions transfer data between the Accumulator and a byte of external data memory, which is why “X” is appended to MOV. There are two types of instructions, differing in whether they provide an 8-bit or 16-bit indirect address to the external data RAM.

8051 Assembly language function description

NOP - No operation

Execution continues at the following instruction. Other than the PC, no registers or flags are affected. This instruction is used to generate simple delays (not recommended)

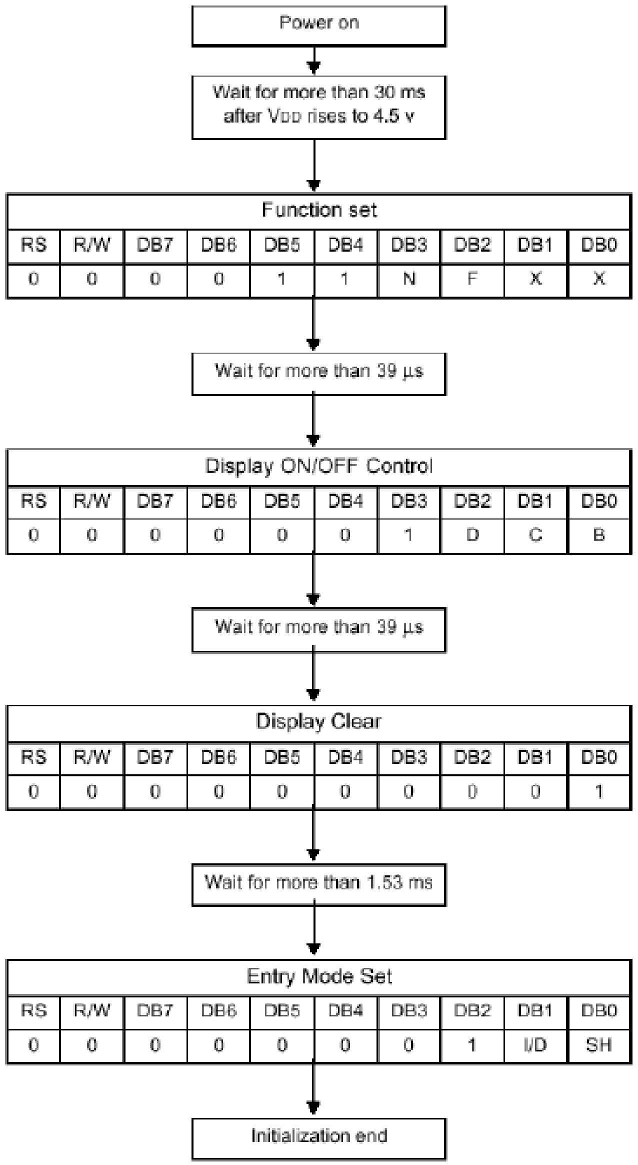
POP - Pop from stack

The contents of the internal RAM location addressed by the Stack Pointer is read, and the Stack Pointer is decremented by one. The value read is then transferred to the directly addressed byte indicated. No flags are affected.

PUSH - Push onto stack

The Stack Pointer is incremented by one. The contents of the indicated variable is then copied into the internal RAM location addressed by the Stack Pointer. Otherwise no flags are affected.

8051 assembly code example 1 – LCD display initialization and displaying data


Power on
Wait for more than 30 ms after VDD rises to 4.5 v

Function set

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	1	1	N	F	X	X

Display ON/OFF Control

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	0	0	1	D	C	B

Display Clear

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	0	0	0	0	0	1

Wait for more than 1.53 ms

Entry Mode Set

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	0	0	0	1	I/D	SH

N	0	1-line mode
	1	2-line mode

F	0	display off
	1	display on

D	0	display off
	1	display on

C	0	cursor off
	1	cursor on

B	0	blink off
	1	blink on

I/D	0	decrement mode
	1	increment mode

SH	0	entire shift off
	1	entire shift on

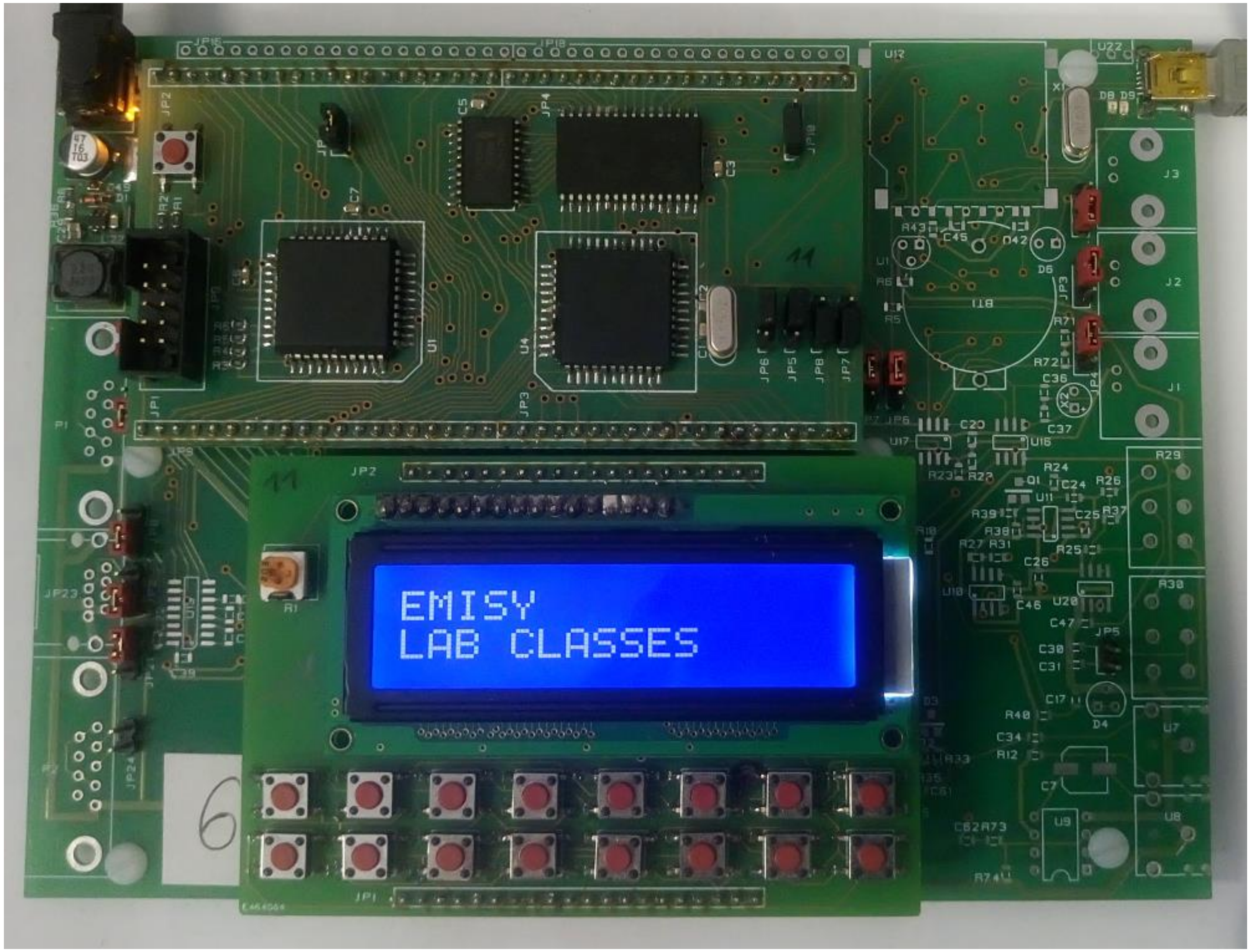
8051 assembly code example 1 – LCD display initialization and displaying data

```
//LCD display example. M. Urbanski 2019
//---------------------------------------
org 8000H            //start writing program code from 8000H, otherwise the system will fail
mov A,#00111000B     /*Function set
                       DL=1 -> 8 bit communication bus ,N=1 -> 2 line display, F=0-> 5x8 matrix
                       Command to be sent is stored in accumulator, to be later used by
                       send_command*/
lcall send_command   /*send data from the accumulator the the display,
                       using command transfer mode (RS=0)*/
lcall delay_us       //wait for more than 39us
mov A,#00001100B     /*Display ON/OFF control, D=1 -> display on, C=1 -> cursor on
                       B=0 -> cursor blinking off*/
lcall send_command   //send to display in command mode
lcall delay_us       //wait for more than 39 us
mov A,#00000001B     //Display clear
lcall send_command   //send to display in command mode
lcall delay_ms       //wait for more than 1.53 ms
mov A,#00000110B     /*Entry mode set I/D = 1 ->address incremented -> cursor moving right
                       S = 0 ->entire shift off*/
lcall send_command   //send to display in command mode
lcall delay_us       //wait for more than 39 us
mov A,#00000000B     //set position of the first sign
lcall send_command   //send to display in command mode
lcall delay_us       //wait for more than 39 us
mov A, #4DH          //write letter 'M' in the accumulator
lcall send_data      //send to display in data mode
lcall delay_us       //wait for more than 39 us
```

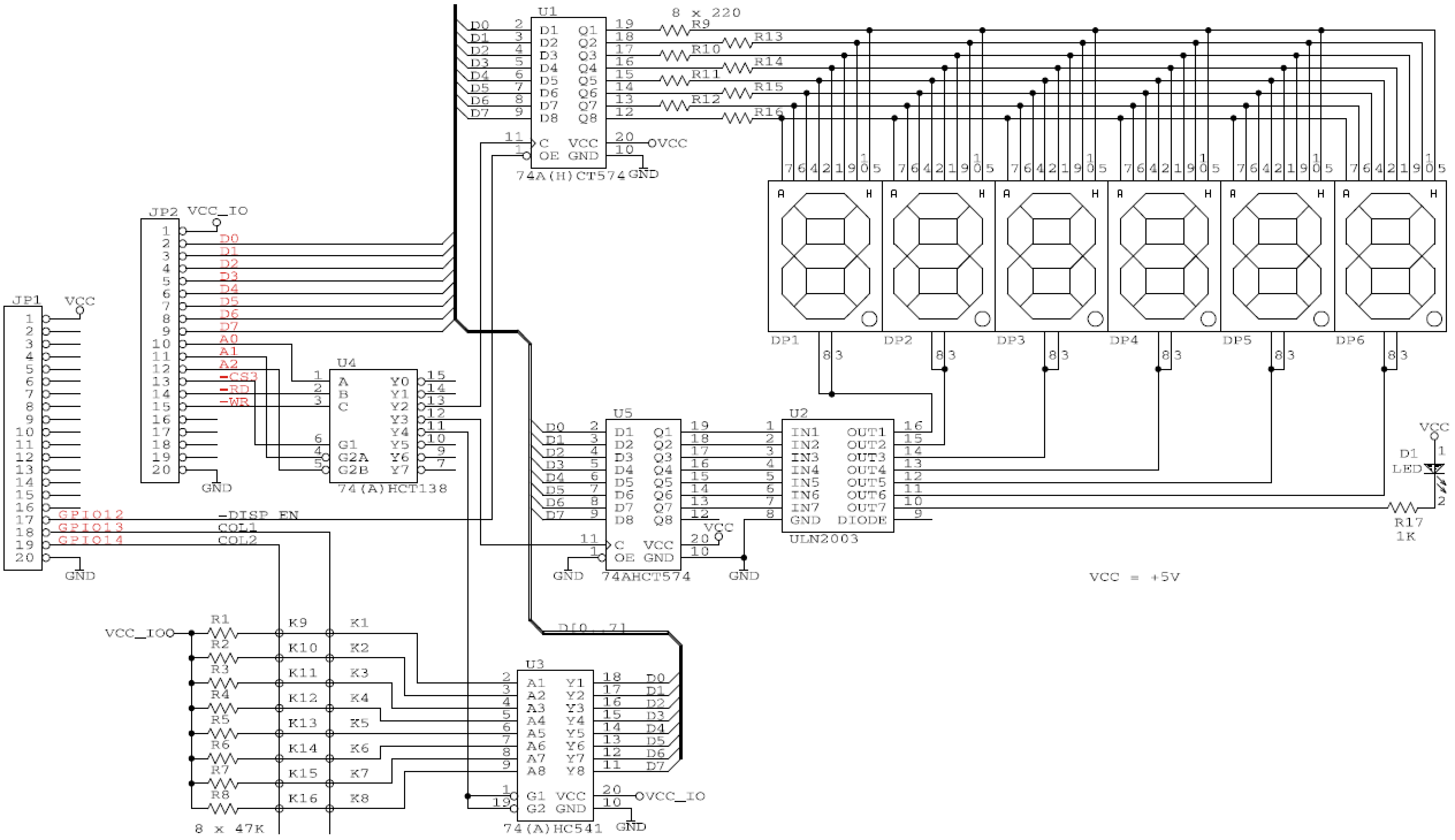
8051 assembly code example 1 – LCD display initialization and displaying data

```
delay_us:                     //~~39us (36 cycles)
          mov R0,#15D         //load register for decrementing
          djnz R0,$           //decrement and jump if R0 is not zero, jump to itself (loop)
          ret                 //return from subroutine
delay_ms:                     //~~1.53ms (approx. 1400 cycles)
          mov R1,#04D                   //load register for decrementing
          delay_jump:
                    mov R0,#176D        //load register for decrementing
                    djnz R0,$           //decrement and jump if R0 is not zero, loop
                    djnz R1,delay_jump  //decrement and jump if R1 is not zero, big loop
                    nop                 //waste some time
                    ret                 //return from subroutine
send_command:
          mov DPH,#0D0H       //0b11010000, send to CPLD, D7 to D0
                              //DPTR tells where to send data (to send it to the display)
          movx @DPTR,A        //command sent
          mov DPH,#0E0H       //0b11100000, sent to CPLD CS3MODE register - RS and E pins
          mov A,#010H         //00010000 - set RS 0 and E 1
          movx @DPTR,A        //command sent
          mov A,#00H          //0 - set RS 0 and E 0
          movx @DPTR,A        //command sent
          ret                 //return from subroutine
send_data:
          mov DPH,#0D0H       //0b11010000 - send to CPLD, D7 to D0
          movx @DPTR,A        //data sent
          mov DPH,#0E0H       //0b11100000 - set RS and E pins
          mov A,#030H         //0b00110000 - set RS 1 and E 1
          movx @DPTR,A        //data sent
          mov A,#020H         //0b00100000 - set RS 1 and E 0
          movx @DPTR,A        //data sent
          ret                 //return from subroutine
END                           //END DIRECTIVE. END OF CODE.
```

8051 assembly code example 1 – LCD display initialization and displaying data



8051 assembly code example 2 – LED display initialization and displaying data



8051 assembly code example 2 – LED display initialization and displaying data

```
;Displaying one letter with 7 segment LED display. M. Urbanski 2019
ORG 8000h               //start memory write at 8000H
JMP START               //go to initialization procedure
ORG 802BH               //start the memory write at 802BH (include JMP START command)
T2_HANDLER:             //handler for T2 interrupt
           MOV P1, #04H          //turn off displays anodes
           JMP DISPLAY1          //start DISPLAY1 subroutine
           CLR 0C8H+7            //clear TF2 interrupt flag - overload T2
           POP DPL               //read the DPL value from the stack and store it in DPTR
           POP DPH               //read the DPH value from the stack and store it in DPTR
           POP Acc               //read the ACC value from the stack and store it in ACC
           RETI                  //quit the subroutine and go back (return address in the stack)
DISPLAY1:                        //subroutine for displaying something in display 1 (first to the left)
           MOV DPTR, #0F001H     //store the data for the displays in 0F001H register (anodes).
                                 //This way the CS3 will be strobed automatically
           MOV A, #01H           //turn on display 1
           MOVX @DPTR, A         //copy to the address stored in DPTR value stored in ACC
                                 //write to 0F001H value 01H
           MOV DPTR, #0F000H     //send data to the cathodes register
           MOV A, #00011100B     //send letter code
           MOVX @DPTR, A         //send it to the register
           MOV P1, #00H          //turn on all the displays
           RETI                  //quit the subroutine
START:                           //initialization subroutine
           MOV P1, #04H          //turn off all the anodes
           MOV DPTR,#0E000H      //use the 0E000H register - CS3 config register
           MOV A,#08H  //prepare CS3 command - update CS3 with strobing RD/WR
                                 //according to lab instruction
           MOVX @DPTR, A         //send the command to the register
           MOV 0C8H, #0          //write zero to TF2 register of T2
           MOV 0CBH, #0FDH       //write value to RCAP2H - set the switching frequency
                                 //to around 1.8 kHz
           SETB 0A8H+5           //turn on T2 interrupts
           SETB EA               //turn on interrupts
           SETB 0C8H+2           //start T2 counter
LOOP:
           JMP LOOP              //infinite loop
END
```

8051 assembly code example 2 – LED display initialization and displaying data