**Microcontroller System Design (8051)**

**Lab 2**

**Arithmetic, Logical, and Branching Operations**

**Objectives**

1. To study the 8051 special function registers
2. To study the 8051 instruction types
3. To perform arithmetic, logical, and branching operations

**Equipment Required**

1. PC and windows XP, Vista Windows 7
2. Keil C51 Compiler

# Introduction

**1.1 8051 Special Function Registers**

P0

SP

DPL

DPH

PCON

TCON

TMOD

TL0

TL1

TH0

TH1

P1

SCON

SBUF

P2

IE

P3

IP

PSW

ACC

B

80H

88H

90H

98H

A0H

A8H

B0H

B8H

C0H

C8H

D0H

D8H

E0H

E8H

F0H

F8H

87H

8FH

97H

9FH

A7H

AFH

B7H

BFH

C7H

CFH

D7H

DFH

E7H

EFH

F7H

FFH

8 Bytes

Bit-Addressable

Table 1: Memory Map of Special Function Registers

A map of the on-chip memory area called SFR (special function registers) space is shown in Table 1. Note that within this space only 21 of the locations are defined as SFRs. The functions of the SFRs are outlined below.

***1.1.1 Accumulator (ACC)***

ACC is the Accumulator register, which is the most commonly used register in 8051 instructions.

***1.1.2 B Register (B)***

The B register is used during multiply and divide operations. For other instructions, it can be treated as a general purpose register.

***1.1.3 Program Status Word (PSW)***

The PSW register contains program status information. The bits within PSW are identified as PSW.0 to PSW.7. Figure 1 shows the format of the PSW together with the addresses of each bit.

**BIT SYMBOL FUNCTION**

PSW.7 CY Carry flag.

PSW.6 AC Auxiliary carry flag. (For BCD operations)

PSW.5 F0 Flag 0. (Available to the user for general purposes)

PSW.4 RS1 Register bank select control bit 1.

Set/Cleared by software to determine working register bank\*.

PSW.3 RS0 Register bank select control bit 0.

Set/Cleared by software to determine working register bank\*.

PSW.2 OV Overflow flag.

PSW.1 − User-defined flag.

PSW.0 P Parity flag.

Set/Cleared by hardware each instruction cycle to indicate an odd/even number of “one” bits in the Accumulator, i.e. even parity.

\*Note: The contents of (RS1, RS0) enable the working register banks as follows:

(0, 0) − Bank 0 (00H~07H)

(0, 1) − Bank 1 (08H~0FH)

(1, 0) − Bank 2 (10H~17H)

(1, 1) − Bank 3 (18H~1FH)

CY

AC

F0

RS1

RS2

OV

−

P

MSB

LSB

**PSW**

Figure 1: PSW (Program Status Word) Register

***1.1.4 Stack Pointer (SP)***

SP is used to point to the top of the stack. In the 8051 family, the stack resides in the on-chip RAM, and grows upwards. This means that the PUSH instruction first increments the Stack Pointer, then copies the required byte onto the stack. The default setting of the SP upon reset is 07H.

***1.1.5 Data Pointer (DPTR)***

The DPTR consists of a high byte (DPH) register and a low byte (DPL) register. Its intended function is to hold a 16-bit address for external code or data memory access.

***1.1.6 Port Registers (P0, PI, P2, P3)***

The 8051 has four I/O ports labeled Port 0, Port 1, Port 2, and Port 3. P0 toP3 are the

SFR latches of Port 0to Port 3, respectively.

***1.1.7 Timer Registers***

Register pairs (TH0, TL0) and (TH1, TL1) are 16-bit counting registers for Timers/Counters.

***1.1.8 Control Registers***

IP, IE, TMOD, TCON, SCON, and PCON contain control and status bits for the interrupt system, timer/counter operations, and the serial communication.

**1.2 8051 Instruction Types**

The 8051instructions are divided into five functional groups:

1. Arithmetic
2. Logical
3. Data transfer
4. Boolean variable
5. Program branching

A summary of each group is discussed below. For detailed descriptions, refer to the MCS®-51 instruction set in Appendix A*.*

***1.2.1 Arithmetic Instructions***

The 8051 microcontroller provides some basic arithmetic instructions for performing addition, subtraction, increment, decrement, multiplication, division, and decimal adjust for BCD number arithmetic, as listed below:

1. ADD - add
2. ADDC - add with carry
3. SUBB - subtract with borrow
4. INC - increment
5. DEC - decrement
6. MUL - multiply
7. DIV - divide
8. DA - decimal adjust

***1.2.2 Logical Instructions***

Logical instructions are performed on one or two bytes of data. Available instructions are listed below:

1. ANL - logical AND
2. ORL - logical OR
3. XRL - exclusive OR
4. CLR - clear
5. CPL - complement
6. RL - rotate left
7. RLC - rotate left through carry
8. RR - rotate right
9. RRC - rotate right through carry
10. SWAP - swap nibbles

***1.2.3 Data Transfer Instructions***

Data transfer instructions move data within or between various memory spaces internal or external to the microcontroller. Instructions available are:

1. MOV - move data memory
2. MOVC - move code memory
3. MOVX - move external RAM data
4. PUSH - push data onto stack
5. POP - pop data from stack
6. XCH - exchange data
7. XCHD - exchange low-order nibble

***1.2.4 Boolean Variables Instructions***

The Boolean instructions perform operations on single bits only. There are instructions to clear, set, or complement any bit-addressable location. The instructions with carry flag (C) as operand are coded with single byte only, others are coded using two bytes where the first byte is the opcode and the second byte is the direct bit address.

1. CLR - clear
2. SETB - set
3. CPL - complement
4. ANL - logical AND
5. ORL - logical OR
6. MOV - move bit memory

***1.2.5 Program Branching Instructions***

Program branching is a very important feature of programs running in microcontrollers and microprocessors. It allows execution of different flows of program according to different input situations. While arithmetic and logical instructions contribute to numerical computation capability of a CPU, program branching instructions play the major role in the “intelligent” capability of a CPU.

All the other types of instructions, when puttogether as a program, can only be executed sequentially one after another. Program branching instructions can break the program execution sequence, conditionally or unconditionally.

1. AJMP - absolute jump
2. LJMP - long jump
3. SJMP - short jump
4. JMP @A+DPTR - jump indirect relative to the DPTR
5. ACALL - absolute subroutine call
6. LCALL - long subroutine call
7. RET - return from subroutine
8. RETI - return from interrupt
9. JZ - jump if ACC is zero
10. JNZ - jump if ACC is not zero
11. JC - jump if carry is set
12. JNC - jump if carry is not set
13. JB - jump if direct bit is set
14. JNB - jump if direct bit is not set

# Arithmetic Operation

1. Create a new file (Exp2a.a51), and write the following sample program.
2. Compile the program, download to the simulator (debugger), and observe the program execution in single step mode.
3. Record the contents of register ACC, B, and PSW for every single step operation.

ORG 0H

MOV A, #32H

MOV B, #78H

ADD A, B

MOV A, #12H

MOV B, #29H

ADD A, B

DA A

MOV A, #123

MOV B, #5

MUL AB

DIV AB

SJMP $

END

1. Explain the DA (decimal adjust) operation.
2. Write a program to sum 1 + 2 + 3 + 4 + ... + 10 and store the result in register B.

# Logical Operation

1. Create a new file (Exp2b.a51), and write the following sample program.
2. Compile the program, download to the simulator (debugger), and record the contents of registers ACC and PSW for every single step operation.

ORG 0200H

MOV A, #1

RL A

RL A

RR A

ORL A, #11000101B

ANL A, #11110000B

XRL A, #10011010B

SWAP A

SJMP $

END

1. Explain the SWAP operation.
2. Write a program to invert bits 7 to 3,set bits 4 & 3, and reset bits 1 & 0 of ACC. The initial value of ACC is 49H. Find the value of ACC after the operation.
3. Give a short description of the function of the flag bits in PSW:

**Flag in PSW** **Symbol** **Function(s)**

Carry C

Auxiliary Carry AC

Register Bank Select RS0, RS1

Parity P

Overflow OV

Flag 0 F0

# Program Branching

1. Create a new file (Exp2c.asm), and write the following sample program.
2. Compile the program, download to the simulator (debugger), and observe the program execution in single step mode.

ORG 0200H

MOV P1, #0

MAIN:

INC P1

CALL DELAY

SJMP MAIN

DELAY:

MOV R1, #2

DLY0:

MOV R0, #5

DJNZ R0, $

DJNZ R1, DLY0

RET

END

1. Explain the DJNZ operation.
2. Find the total number of machine cycles required to execute the DELAY subroutine call.