#



UNIVERSITY OF SCIENCE &TECHNOLEGY

FACUALTY OF ENGNEERING

BIOMEDICAL DEPARTMENT

# Microcontroller (1) Lab Manual



 Prepared By:

Eng: Mohsen Ali AL-awami

Supervisered By:

Dr: Fadel AL-aqawa

2010-2011

LAB Expeirment (2)

* Main Topics:
* Jump ,Loop and Call instructions
* Assembely Arthimatics and Logic Operations
* Learning Objectives/Tasks:

 Upon Completion this experiment ,you will be able to :

* Code 8051 assembly language instruction using loops
* Code 8051 assembly language conditional jump instructions
* Explain condition that determine each conditional jump instruction
* Code 8051 subroutines
* Describe precautions in using the stack in subroutines
* Define the range of numbers possible in 8051 unsigned data
* Code addition and subtraction instructions for unsigned data
* Define the range of numbers possible in 8051 signed data
* Code addition and subtraction instructions for signed data
* Explain carry and overflow problems and their corrections
* Define the truth table for logic function AND,OR and XOR
* Code 8051 assembly language logic function instructions

**Section 1:** Loop and jump instructions

Repeating a sequence of instructions a certain number of times are called a **LOOP,**

The loop is one of the most widely used actions is performed by the instraction

[‘’ DJNZ reg, lable ‘’].

In this instruction, the register is decremented, if not zero, it jump to target address referred to by the label.

**Example 1:**

Write a program to

(a) Clear ACC

(b) Add 3 to accumulator ten times

Solution:

 Mov A,#0 ;a=0 clear ACC

###  Mov R2,#10 ; load counter r2=10

AGAIN: ADD A,#03 ; add 03 to acc

 Djnz R2, AGAIN ;repeat untel r2=0 (10 times )

###  Mov R5,A ; save A in R5

###

**Example 2:**

What is maximum number of times that the loop in last example can be repeated?

Solution:

Since the holds the count and R2 ia an 8-bites register, it can be hold maximum of FFH

(256 in decimal times)

###

**Loop inside a loop:**

As shown in **Example 2 the** maximum number of count is 256 , what happens if we want to repeat an action more times than 256?

To do that, we use loop in side loop which is called a nested loop

**Example 3:**

Write a program to

1. load the accumulator with the value 55H
2. complement the ACC 700 times

Solution:

; Since 700 is larger than 255 (the maximum capacity of any register), so we will use two registers to hold account.

 Mov A,#55h ; A=55

 Mov R3,#10 ; r3=10

 NEXT : Mov R2,#70 ; r2=70

 AGAIN: CPL A ;compelement A

 DJNZ R2 , AGAIN ;repeat it 70 times (inner loop)

 DJNZ R3 , NEXT

* **Other Conditional Jumps**

Conditional jumps for the 8051 are summerized in the next table:

Such as JZ (jump if A =0)

 JC (jump if carry =1 )

|  |  |
| --- | --- |
|  instraction |  Action |
| JZ |  Jump if a=0 |
| JNZ | Jump if a not= 0 |
| DJNZ | Decrement and Jump if a not=0 |
| CJNE A,BYTE | Jump if a not= byte |
| CJNE REG,#DATA | Jump if a not= #data |
| JC | Jump if carry=1 |
| JNC | Jump if carry=0 |
| JB | Jump if bit=1 |
| JNB | Jump if bit=0 |
| JBC | Jump if bit=1 and clear bit |

All conditional jumps are short jumps:

The address of the target must within -128 to +127 bytes of the contents of PC

**Example 4:**

Write a program to determine if R5 contains the value 0 .if so, put 55H in it.

Solution:

Mov A,R5 ;copy R5 to A

JNZ NEXT ;jump if A is not zero

Mov R5,#55H

NEXT: …………………………

* **JNC(jump if no carry, jumps if cy=0):**
* In executing ‘’JNC’’ ,the processor looks at the carry flag to see if it raised (cy=1).if it is not ,the CPU starts to fetch and execute instructions from the address of the label .if the carry =1 ,it will not it will execute the next instraction below **JNC.**

It need to be noted that there is also **‘’JC lable ’’** instruction .in the jc instruction, if cy=1 it jumps to the target address.

**Example 5:**

Find the sum of the values **79H,F5H** and **E2H**.put the sum in the registers R0(low byte)and R5(high byte).

Solution:

 Mov A,#0 ; clear A (A=0)

 Mov R5,A ; clear R5

 ADD A,#79H ; A=0+79H=79H

 JNC N1 ; if no carry ,add next number

 INC R5 ;if cy=1,increment R5

N1: ADD A,#0F5H ; A=79+F5=6E

 JNC N2 ; jump if cy=0

 INC R5 ; if cy=1 ,then incerement R5

N2 : ADD A,#0E2H ; A=6E+E2=50 and CY=1

 JNC over ; jump if cy=0

 INC R5 ; if cy=1 ,then incerement R5

OVER: mov R0,A ;Now R0=50h , and R5 =02 SS

* The unconditional jump is a jump in which control is transferred unconditionally to the target location
* **LJMP** (long jump)
* 3-byte instruction
* First byte is the opcode
* Second and third bytes represent the 16-bit

 target address

 – Any memory location from 0000 to FFFFH

* **SJMP** (short jump)
* 2-byte instruction
* First byte is the opcode
* Second byte is the relative target address

 – 00 to FFH (forward +127 and backward

 -128 bytes from the current PC).

Call instructions

* Call instruction is used to call subroutine
* Subroutines are often used to perform tasks

 that need to be performed frequently

* This makes a program more structured in

 addition to saving memory space

* **LCALL** (long call)
* 3-byte instruction
* First byte is the opcode
* Second and third bytes are used for address of target subroutine

– Subroutine is located anywhere within 64K byte address space

* **ACALL** (absolute call)
* 2-byte instruction
* 11 bits are used for address within 2K-byte range
* When a subroutine is called, control is transferred to that subroutine, the processor
* Saves on the stack the the address of the instruction immediately below the LCALL
* Begins to fetch instructions form the new location
* After finishing execution of the subroutine
* The instruction RET transfers control back to the calle
* Every subroutine needs RET as the last instruction

**Example 6:**

 ORG 0

BACK: MOV A,#55H ;load A with 55H

 MOV P1,A ;send 55H to port 1

LCALL DELAY ;time delay

 MOV A,#0AAH ;load A with AA (in hex)

 MOV P1,A ;send AAH to port 1

LCALL DELAY

SJMP BACK ;keep doing this indefinitely

;---------- this is delay subroutine ------------

 ORG 300H ;put DELAY at address 300H

DELAY:MOV R5,#0FFH ;R5=255 (FF in hex), counter

AGAIN:DJNZ R5,AGAIN ;stay here until R5 become 0

 RET

 END

001 0000 ORG 0

002 0000 7455 BACK: MOV A,#55H ;load A with 55H

003 0002 F590 MOV P1,A ;send 55H to p1

004 0004 120300 LCALL DELAY ;time delay

005 0007 74AA MOV A,#0AAH ;load A with AAH

006 0009 F590 MOV P1,A ;send AAH to p1

007 000B 120300 LCALL DELAY

008 000E 80F0 SJMP BACK ;keep doing this

009 0010

010 0010 ;-------this is the delay subroutine------

011 0300 ORG 300H

012 0300 DELAY:

013 0300 7DFF MOV R5,#0FFH ;R5=255

014 0302 DDFE AGAIN: DJNZ R5,AGAIN ;stay here

015 0304 22 RET ;return to caller

016 0305 END ;end of asm file

|  |  |
| --- | --- |
| 0A |  |
| 09 | 00 |
| 08 | 07 |

* Stack fram after the first LCALL

08

Low byte goes first then high byte

 SP (stack pointer) = 09

* The use of ACALL instead of LCALL

can save a number of bytes of program ROM space .

**ARITHMETIC & LOGIC**

**INSTRUCTIONS AND**

**PROGRAMS**

* Assembely Arthimatics Operations:
* Addition of unsigned numbers
* ADD A,source ;A = A + source
* The instruction ADD is used to add two operands
* Destination operand is always in register A
* Source operand can be a register, immediate data, or in memory
* Memory-to-memory arithmetic operations are never .

Example 1:

Show how the flag register is affected by the following instruction.

 MOV A,#0F5H ;A=F5 hex

 ADD A,#0BH ;A=F5+0B=00

**Solution:**

 F5H 1111 0101

 + 0BH + 0000 1011

 -------- -----------

 100H 0000 0000

* When adding two 16-bit data operands,the propagation of a carry from lower byte to higher byte is concerned.

Example 2:

 1

 3C E7

 + 3B 8D

 ---------

 78 74

Write a program to add two 16-bit numbers. Place the sum in R7 and

R6; R6 should have the lower byte.

**Solution:**

 CLR C ;make CY=0

 MOV A, #0E7H ;load the low byte now A=E7H

 ADD A, #8DH ;add the low byte

 MOV R6, A ;save the low byte sum in R6

 MOV A, #3CH ;load the high byte

 ADDC A, #3BH ;add with the carry

 MOV R7, A ;save the high byte sum

* + The binary representation of the digits 0 to 9 is called BCD (Binary Coded Decimal)
* Unpacked BCD

In unpacked BCD, the lower 4 bits of the number represent the BCD number, and the rest of the bits are 0 .

Ex. 00001001 and 00000101 are unpacked BCD for 9 and 5.

* Packed BCD

In packed BCD, a single byte has two BCD number in it, one in the lower 4 bits, and one in the upper 4 bits .

Ex. 0101 1001 is packed BCD for 59H.

* Adding two BCD numbers must give a BCD result.

Example 2:

 MOV A, #17H

 ADD A, #28H

The result above should have been 17 + 28 = 45 (0100 0101).

To correct this problem, the programmer must add 6 (0110) to the

low digit: 3F + 06 = 45H.

* DA A ;decimal adjust for addition
* The DA instruction is provided to correct the aforementioned problem associated with BCD addition
* The DA instruction will add 6 to the lowe nibble or higher nibble if need .

**Example 3 :**

 MOV A,#47H ;A=47H first BCD operand

 MOV B,#25H ;B=25H second BCD operand

 ADD A,B ;hex(binary) addition(A=6CH)

 DA A ;adjust for BCD addition

 (A=72H)

The “DA” instruction works only on A. In other word, while the source

can be an operand of any addressing mode, the destination must be in

register A in order for DA to work.

* Subtraction of unsigned numbers :
* In many microprocessor there are two different instructions for subtraction: SUB and SUBB (subtract with borrow)
* In the 8051 we have only SUBB
* The 8051 uses adder circuitry to perform the subtraction

 SUBB A,source ;A = A – source – CY

To make SUB out of SUBB, we have to make CY=0 prior to the execution of the instruction

* Notice that we use the CY flag for the borrow
* SUBB when CY = 0
* 1. Take the 2’s complement of the subtrahend (source operand)
* 2. Add it to the minuend (A)
* 3. Invert the carry

**Example 4:**

 CLR C

 MOV A,#4C ;load A with value 4CH

 SUBB A,#6EH ;subtract 6E from A

 JNC NEXT ;if CY=0 jump to NEXT

 CPL A ;if CY=1, take 1’s complement

 INC A ;and increment to get 2’s comp

NEXT: MOV R1,A ;save A in R1

**Solution:**

 4C 0100 1100 0100 1100

- 6E 0110 1110 1001 0010

 ---- ----- -----

 -22 01101 1110

* SUBB when CY = 1
* This instruction is used for multi-byte numbers and will take care of the borrow of the lower operand .

Example 5:

 CLR C

 MOV A,#62H ;A=62H

 SUBB A,#96H ;62H-96H=CCH with CY=1

 MOV R7,A ;save the result

 MOV A,#27H ;A=27H

 SUBB A,#12H ;27H-12H-1=14H

 MOV R6,A ;save the result

**Solution:**

We have 2762H - 1296H = 14CCH.

**SIGNED ARITHMETIC INSTRUCTIONS**

(Signed 8-bit Operands )

* D7 (MSB) is the sign and D0 to D6 are the magnitude of the number
* If D7=0, the operand is positive, and if D7=1, it is negative
* Positive numbers are 0 to +127
* Negative number representation (2’s complement)

1. Write the magnitude of the number in 8-bit binary (no sign)

2. Invert each bit

3. Add 1 to it.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |

 ----

 Sign Magnitude

Show how the 8051 would represent -34H

**Solution:**

1. 0011 0100 34H given in binary

2. 1100 1011 invert each bit

3. 1100 1100 add 1 (which is CC in hex)

Signed number representation of -34 in 2’s complement is CCH

**SIGNED ARITHMETIC INSTRUCTIONS**

 **(Overflow Problem)**

* If the result of an operation on signed numbers is too large for the register.
* An overflow has occurred and the programmer must be noticed.

Examine the following code and analyze the result.

 MOV A,#+96 ;A=0110 0000 (A=60H)

 MOV R1,#+70 ;R1=0100 0110(R1=46H)

 ADD A,R1 ;A=1010 0110

 ;A=A6H=-90,INVALID

**Solution:**

 +96 0110 0000

+ +70 0100 0110

 ----- -------------

+ 166 1010 0110 and OV =1

According to the CPU, the result is -90, which is wrong. The CPU

sets OV=1 to indicate the overflow

**SIGNED ARITHMETIC INSTRUCTIONS**

**(2's Complement)**

* To make the 2’s complement of a number

 CPL A ;1’s complement (invert)

 ADD A,#1 ;add 1 to make 2’s comp.

# LOGIC AND COMPARE INSTRUCTIONS

1. **AND LOGIC)**

ANL destination,source ;dest = dest AND source

* This instruction will perform a logic AND on the two operands and place the result in the destination
* The destination is normally the accumulator
* The source operand can be a register, in memory, or immediate

Example 1:

Show the results of the following.

 MOV A,#35H ;A = 35H

 ANL A,#0FH ;A = A AND 0FH

 35H 0 0 1 1 0 1 0 1

 0FH 0 0 0 0 1 1 1 1

= ------- -------- -----------

 05H 0 0 0 0 0 1 0 1

1. **OR LOGIC)**
* ORL destination,source ;dest = dest OR source

The destination and source operands are ORed and the result is placed in the destination .

* The destination is normally the accumulator
* The source operand can be a register, in memory, or immediate .

**Example 2:**

Show the results of the following.

 MOV A,#04H ;A = 04

 ORL A,#68H ;A = 6C

 04H 0 0 0 0 0 1 0 0

 68H 0 1 1 0 1 0 0 0

 =----- --------- ---------

 6CH 0 1 1 0 1 1 0 0

**(3- XOR)**

**XRL** destination, source ;dest = dest XOR source

* This instruction will perform XOR operation on the two operands and

place the result in the destination

* The destination is normally the accumulator
* The source operand can be a register, in memory, or immediate .

**Example 3:**

Show the results of the following.

 MOV A,#54H

 XRL A,#78H

 54H 0 1 0 1 0 1 0 0

 78H 0 1 1 1 1 0 0 0

= ------ -------- ----------

 2CH 0 0 1 0 1 1 0 0

**4- Compare Instruction**

CJNE destination,source,rel. addr.

* The actions of comparing and jumping are combined into a single instruction called CJNE (compare and jump if not equal)
* The CJNE instruction compares two operands, and jumps if they are not equal.
* The destination operand can be in the accumulator or in one of the Rn registers The source operand can be in a register, in memory, or immediate The operands themselves remain unchanged.
* It changes the CY flag to indicate if the destination operand is larger or smaller.

Example 4:

 CJNE R5,#80,NOT\_EQUAL ;check R5 for 80

 ... ;R5 = 80

NOT\_EQUAL:

 JNC NEXT ;jump if R5 > 80

 ... ;R5 < 80

NEXT: ...

* **Compare Carry Flag**
* destination ≥ source CY = 0
* destination < source CY = 1
* The compare instruction is really a Subtraction.

**Rotating Right and Left**

* **RR A** ;rotate right A
* In rotate right
* The 8 bits of the accumulator are rotated right one bit, and
* Bit D0 exits from the LSB and enters into MSB, D7

MSB LSB

 MOV A,#36H ;A = 0011 0110

 RR A ;A = 0001 1011

 RR A ;A = 1000 1101

 RR A ;A = 1100 0110

 RR A ;A = 0110 0011

t’)

* **RL A** ;rotate left A
* In rotate left
* The 8 bits of the accumulator are rotate left one bit, and
* Bit D7 exits from the MSB and enters into LSB, D0

MSB LSB

 MOV A,#72H ;A = 0111 0010

 RL A ;A = 1110 0100

 RL A ;A = 1100 1001

**Rotating through Carry**

 **RRC A** ;rotate right through carry

* In RRC A
* Bits are rotated from left to right
* They exit the LSB to the carry flag, and the carry flag enters the MSB.

 CY

MSB LSB

 CLR C ;make CY = 0

 MOV A,#26H ;A = 0010 0110

 RRC A ;A = 0001 0011 CY = 0

 RRC A ;A = 0000 1001 CY = 1

 RRC A ;A = 1000 0100 CY = 1

 **RLC** A ;rotate left through carry

* In RLC A
* Bits are shifted from right to left.
* They exit the MSB and enter the carry flag,

 and the carry flag enters the LSB.

MSB LSB

 CY

Write a program that finds the number of 1s in a given byte.

 MOV R1,#0

 MOV R7,#8 ;count=08

 MOV A,#97H

AGAIN: RLC A

 JNC NEXT ;check for CY

 INC R1 ;if CY=1 add to count

NEXT: DJNZ R7,AGAIN