Module 3

Branching instructions

Jump & Call instructions:

Jump & Call instructions are decision making codes. These instructions change the sequence of the program (alter the flow of program) by changing the contents of PC

Jump instruction permanently changes the contents of the PC based on certain conditions or unconditionally.

Call instruction temporarily changes the PC to allow another part of the program to run

Classification of Jump instructions

* Unconditional Jump
* Conditional Jump
* Bit Jump
* Byte Jump
* Relative Jump
* Absolute Jump
* Long Jump

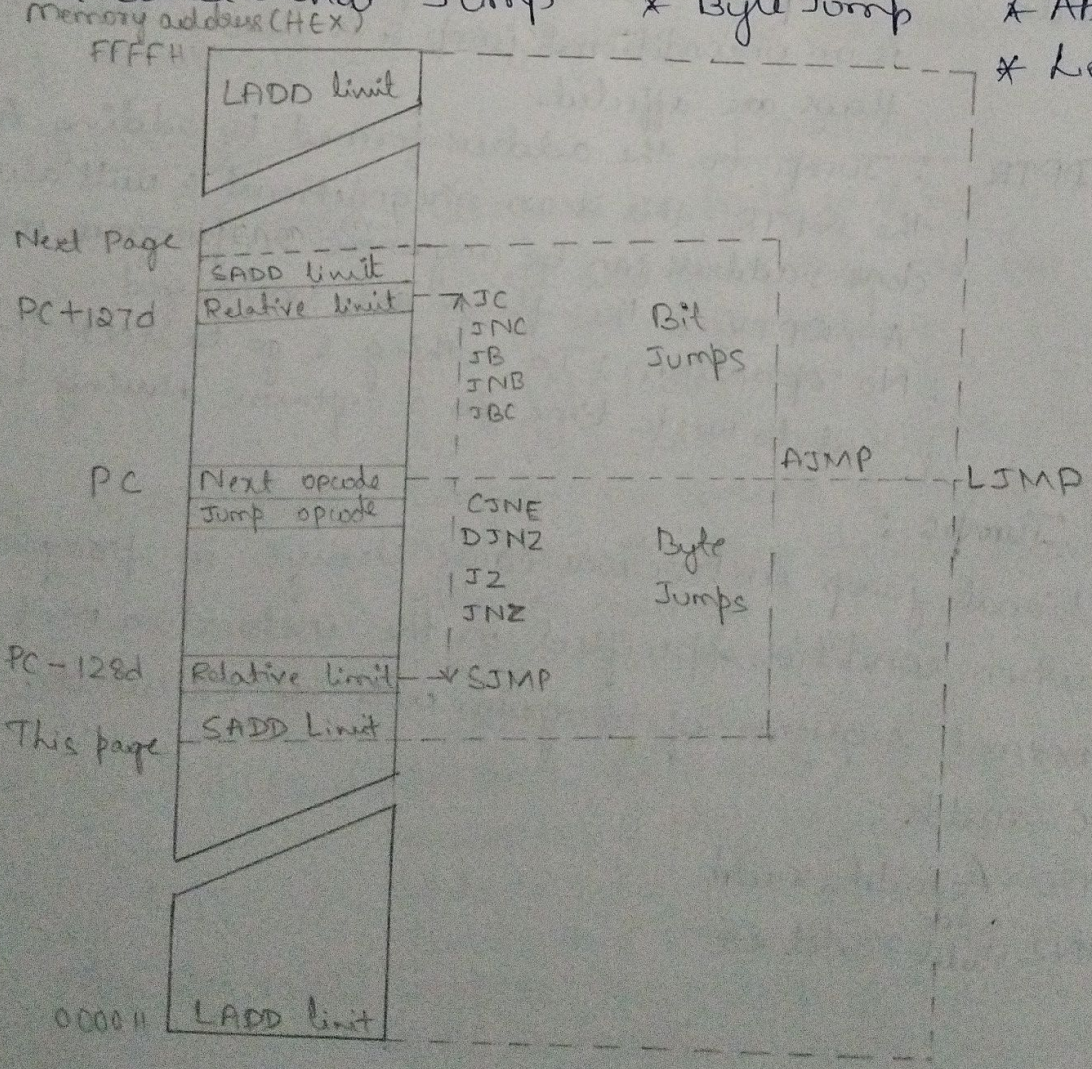


Fig. Jump Instruction Ranges

Unconditional Jumps :

Unconditional jump instructions, alter the flow of prog without any conditions. During the execution of these instructions PC is loaded with address specified in the instructions & sequence of program changes accordingly.

These Jumps do not test any bit (or) byte to determine whether the jump should be taken. No flags are affected by these instructions

Mnemonic	Operation
SJMP radd	; Jump to relative address radd ; this is an unconditional jump & its always taken ; no flags are affected.
AJMP sadd	; Jump to absolute short range address sadd ; this is an unconditional jump & is always taken. no flags are affected.
LJMP ladd	; Jump to absolute long range address ladd ; this is an unconditional jump & is always taken ; no flags are affected.
JMP @A+DPTR	; Jump to the address formed by adding A to the DPTR ; this is an unconditional & will always be done ; address can be anywhere in program memory ; A, DPTR & the flags are unchanged
NOP	; No operation ; Do nothing & go to next instruction ; Used to waste time in a software timing loop.

Conditional Jumps :

Conditional jump instructions cause change in program flow only when condition specified in the instruction met. Otherwise normal sequence of program will be executed.

Eg: JC radds ;
CJNE A, add, radd
DJNZ Rn, radd etc

Bit Jumps:

Bit jumps all operate according to the status of Carry flag in the PSW (or) the status of any bit-addressable location. All bit jumps are relative to program counter.

Ex: JC radd ; Jump relative if carry flag is set to 1

JNC radd ; Jump relative if the carry flag is reset to 0

JB b, radd ; Jump relative if addressable bit is set to 1

JNB b, radd ; Jump relative if addressable bit is reset to 0

JBC b, radd ; Jump relative if addressable bit is set & clear the addressable bit to 0.

Byte Jumps:

Byte jumps - jump instructions that test bytes of data - behave as bit jumps. If the condition that is tested is true the jump is taken; if condition is false, the instruction after jump is executed. All byte jumps are relative to PC.

Mnemonic	Operation
CJNE A, add, radd ;	Compare contents of A reg. with contents of direct address if they are not equal, then jump to relative address If [A] < [add] then [C] = 1 [A] ≥ [add] then [C] = 0
CJNE A, #n, radd ;	Compare the contents of A reg with immediate number n; if they are not equal then jump to relative address [A] < #n then [C] = 1 [A] ≥ #n then [C] = 0
CJNE Rr, #n, radd ;	Compare the contents of reg Rr with immediate number n; if they are not equal jump to relative address; Set the Carry flag to 1 if Rn is less than number; otherwise set carry flag to 0
CJNE @Rp, #n, radd ;	Compare the contents of the address contained in reg. Rn to number n; if they are not equal jump to relative address if [Rp] < #n ; C = 1 [Rp] ≥ #n ; C = 0

DJNZ Rn,radd ; Decrement the register Rn by 1 and jump to relative address if the result is not 0; no flags are affected.

DJNZ add,radd ; Decrement the direct address by 1 & jump to address if result is not 0; No flags are affected unless direct address is PSW.

JZ radd ; Jump to relative address if A is 0; no flags are changed

JNZ radd ; Jump to relative address if A is not 0; flags & A are not changed.

Jump & CALL Program Range

Jump (or) Call instruction can replace the contents of PC with a new program address number that causes program execution to begin at the code located at new address. The difference in bytes of this new address from the address in the program where jump (or) call is located is called the range of jump (or) call.

It may have one of three ranges:
* Relative range
* Absolute range
* Long range

(a) Relative Range Jumps

Jumps that replace the PC contents with a new address that is greater than the address of instruction following the jump by 127d (7FH) (or) less than address of instruction following the jump by 128d (FFH) are called relative jumps.

Advantages

* It is a 2 byte instruction, one byte is opcode & another byte is relative address (Data)
* Specifying only one byte address, saves program bytes & speeds-up Program execution.
* The program that is written using relative jumps can be used anywhere in the program space.

All jumps are just within +127d & -128d bytes. But most jumps form program loops over a short code ranges that are within relative address capability.

* Jumps are the only branch instructions that can use relative range.

(b) Short Absolute Jumps

Absolute range make use of concept of dividing memory into logical divisions called pages. There are total 32d pages. The upper 5 bits of program counter hold the page number & lower 11 bits holds the address within each page.

An absolute address is formed by taking page number of the instruction following the branch & attaching the absolute page range address of 11 bits to it to form the 16-bit address.

Advantage of absolute range is that fewer bytes are needed & the code is code is relocatable as long as the relocated code remains on the following page.

(c) Long Absolute Jumps:

Address that can access entire program space from 0000H - FFFFH, use long-range addressing. Long range address require more bytes of code to specify the range. Long range addressing has the advantage of using the entire program address space available to 8051

Eg: LJMP laddr

① Put a random number in R3 & increment it until it equals E1h

Label	Mnemonics	Comments
	ORG 0000H	
	MOV R3, #01H	
BACK:	INC R3	
	CJNE R3, #0E1h, BACK	; Compare till [R3] #E1h
HERE:	SJMP HERE	
	END	

② Put a random number in address 20H & increment until it equals a random number put in R5

Label	Mnemonics	Comments
	ORG 0000H	
	MOV 20H, #03H	
	MOV R5, # F0H	
	MOV A, R5	
BACK:	INC 20H	; Increment address content
	CJNE A, 20H, BACK	; Compare R5 contents till equ F0H
HERE:	SJMP HERE	
	END	

③ Put a random number in R3 & decrement it until it equals E1h

Label	Mnemonics	Comments
	ORG 0000H	
	MOV R3, # 0BAh	
BACK:	DEC R3	; Decrement contents of R3
	CJNE R3, #0E1H, BACK	; Compare R3 contents till equals E1H
HERE:	SJMP HERE	
	END	

④ Write a program to store data FFH into RAM memory locations 30H to 35H using indirect addressing mode

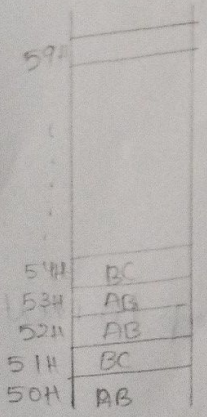
Label	Mnemonics	Comments
	ORG 0000H	
	MOV A, #0FFH	
	MOV R0, #30H	
	MOV R5, #06H	; Counts i.e., 30H - 35H
BACK:	MOV @R0, A	; Copy contents of A to 30H
	INC R0	; Increment memory locations @R0
	DJNZ R5, BACK	; Decrement the count till it equals 0 (R5)
	END	

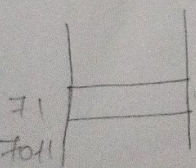
Write a program to clear ten RAM locations starting at address 1000H

Label	Mnemonics	Comments
	ORG 0000H	
	MOV DPTR, #1000H	
	CLR A	; Clear the contents in A
	MOV R4, #0AH	; Counter
BACK:	MOVX @DPTR, A	; Clear the contents of 1000H
	INC DPTR	
	DJNZ R4, BACK	; Decrement counts till it equals 0
HERE:	SJMP HERE	
	END	

⑥ Write a program to add ten, 8-bit hexadecimal no's stored in RAM locations starting from 50H. Store result at locations 70H & 71H

Label	Mnemonics	Comments
	ORG 0000H	
	MOV R0, #50H	
	MOV R1, #0AH	
	MOV R2, #00H	
	CLR A	
BACK:	ADD A, @R0	; Add contents of A with 50H
	MOV R3, A	; Store the result in R3
	CLR A	
	ADDC A, R2	; Get the Carry
	MOV R2, A	; Store the Carry in R2
	MOV A, R3	; Get back result to A
	INC R0	; Increment RAM address to 51H
	DJNZ R1, BACK	; Decrement R1 till it equals 00H
	MOV 70H, R3	; Store the sum in 70H
	MOV 71H, R2	; Store the Carry in 71H
HERE:	SJMP HERE	
	END	





⑦ Write a program to find largest number among an array of ten, 8-bit no's, stored at RAM address starting from 30H

Label	Mnemonics	Comments
	ORG 0000H	
	MOV R0, #30H	; starting address of RAM is 30H
	MOV R2, #09H	; Counter (N-1)
	MOV A, @R0	
BACK:	INC R0	
	MOV 03H, @R0	; Copy contents of RAM 31H to R3
	CJNE A, 03H, NEXT	; check whether A & R3 contents equal
NEXT:	JNC NEXT1	; (or) not
	MOV A, @R0	; If [CY]=1 put larg no in A
NEXT1:	DJNZ R2, BACK	
HERE:	SJMP HERE	
	END	

⑧ Write a program to find smallest number among the array of ten, 8-bit no's stored starting from RAM locations 30H

Label	Mnemonics	Comments
	ORG 0000H	
	MOV R0, #30H	
	MOV R3, #09H	
	MOV A, @R0	
BACK:	INC R0	
	MOV 02H, @R0	
	CJNE A, 02H, NEXT	
NEXT:	JC NEXT1	; check [CY]=1, if yes jump to Next
	MOV A, @R0	; Store small no in A.
NEXT1:	DJNZ R3, BACK	
HERE:	SJMP HERE	
	END	

Note: CJNE instruction affects [CY] flag: [A] ≥ [addr] ; [CY]=0

[A] < [addr] ; CY=1

Count the number of 1's in any number in register B and put the count in R5

Label	Mnemonics	Comments
	ORG 0000H	
	MOV B,#0FH	
	MOV A,B	
	MOV R0,#08H	; Counter to rotate
	MOV R5,#00H	; To coord number of 1's
BACK:	RRC A	
	JNC NEXT	
	INC R5	
NEXT:	DJNZ R0, BACK	
HERE:	SJMP HERE	
	END	

(10) Count the number of 0's in any number in Reg.B and put the count in R5

Same program as above but replace JNC with JC

(11) If the signed number placed in R7 is negative. Set the carry flag to 1, otherwise clear it.

Label	Mnemonics	Comments
	ORG 0000H	
	MOV A,R7	
	RLC A	
	JC NEXT	
	CLR C	
	SJMP STOP	
NEXT:	SETB	
STOP:	SJMP STOP	
	END	

(12) Increment DPTR from any initialized value to ...

Label	Mnemonics	Comments
	ORG 0000H	
	MOV DPTR, #0000H	
BACK:	INC DPTR	
	MOV R1, 82H	; Low byte of DPTR
	CJNE R1, #0CDH, BACK	; Compare contents Low byte with #0CDH
	MOV R2, 83H	; Store high byte of DPTR
	CJNE R2, #0ABH, BACK	; Compare high byte with #0ABH.
	END	

(13) If the lower nibble of any number placed in A is larger than upper nibble, Set the C flag to 1 otherwise clear it.

Label	Mnemonics	Comments
	ORG 0000H	
	MOV R0, A	
	ANL A, #0FH	
	MOV R1, A	; Low nibble in R1
	MOV A, R0	
	ANL A, #0F0H	
	RR A	
	RR A	
	RR A	
	RR A	; High nibble in A
	CJNE A#01H, NEXT	
NEXT:	JC NEXT1	
	CLR C	
	SJMP STOP	
NEXT1:	SETB C	
STOP:	SJMP STOP	
	END	

(14) Use R4 (LSB) & R5 (MSB) as a single 16-bit Counter & decrement the pair until they equal 0000H

Label	Mnemonics	Comments
	ORG 0000H	
	MOV R4, #0AH	; LSB
	MOV R5, #0BH	; MSB
BACK:	DJNZ R4, BACK	
	DJNZ R5, BACK	
	END	

put a random number in address 20h (LSB) & 21h (MSB) and decrement them as if they were a single 16-bit counter until they equal random number in R2 (LSB) & R3 (MSB)

Label	Mnemonics	Comments
	ORG 0000H	
	MOV 20H, #0ABH	; Load EFAB to 21&20H
	MOV 21H, #0EFH	
	MOV R2, #0AH	; Load 0B0A to R3&R2
	MOV R3, #0BH	
BACK:	DEC 20H	
	MOV A, R2	
	CJNE A, 20H, BACK1	
	DEC 21H	
	MOV A, R3	
	CJNE A, 21H, BACK1	; Decrement till 0B0A
HERE:	SJMP HERE	
	END	

(16) Decrement the DPTR from any initialized value to 0033H as a 16-bit register

Label	Mnemonics	Comments
	ORG 0000H	
	MOV DPTR, #0ABEFH	
BACK:	DEC 82H	; Address of DPL is 82H
	MOV A, 82H	
	CJNE A, #33H, BACK	
	DEC 83H	; Address of DPH is 83H
	MOV R1, 83H	
	CJNE R1, #00H, BACK	
HERE:	SJMP HERE	
	END	

(17) Increment the DPTR from any initialized value to ABCDH

Label	Mnemonics	Comments
	ORG 0000H	
	MOV DPTR, #000AH	
BACK:	INC DPTR	
	MOV A, 82h	; Copy contents of DPL to A
	CJNE A, #0CDH, BACK	
	MOV R1, 83H	; Copy contents of DPH to R1

```
        CJNE  R1,#0ABh,BACK
HERE:   SJMP  HERE
        END
```

(18) Write a program to move block of memory from 30-35H to 50-55H

Label	Mnemonics		Comments
	ORG	0000H	
	MOV	R0, #30H	
	MOV	R1, #50H	
	MOV	R2, #06H	
BACK:	MOV	A, @R0	; Copy contents of 30H to A
	MOV	@R1, A	; Copy contents of A to 50H
	INC	R0	
	INC	R1	
	DJNZ	R2, BACK	; Decrement R2 till zero
HERE:	SJMP	HERE	
	END		

(19) Write a program to exchange block of data from 30-39H with 50-59H

Label	Mnemonics		Comments
	ORG	0000H	
	MOV	R0, #30h	;
	MOV	R1 ; #50H	
	MOV	R2, #0AH	
BACK:	MOV	A , @R0	; Copy contents of 30H to A
	MOV	R3, A	; Copy contents of A to R3
	MOV	A, @R1	; Copy contents of 50H to A
	MOV	@R0, A	; Copy content of A to 30H
	MOV	A, R3	; Copy contents of R3 to A
	MOV	@R1, A	
	INC	R0	
	INC	R1	
	DJNZ	R2, BACK	
HERE:	SJMP	HERE	
	END		

Find whether the byte at RAM location 30H is palindrome or NOT. If it is Palindrome display FFH @ port P1 otherwise 00H

Label	Mnemonics	Comments
	ORG 0000H	
	MOV A, 30H	
	MOV R0, A	
	ANL A, #0FH	
	MOV R1, A	
	MOV A, R0	
	ANL A, #0F0H	
	SWAP A	
	CJNE A, 01, NEXT	
	MOV A, #FFH	
	MOV P1, A	
	SJMP HERE	
NEXT:	MOV A, #00H	
	MOV P1, A	
HERE:	SJMP HERE	
	END	

(21) Generate Arithmetic series upto FFH & display result in Port P1

Label	Mnemonics	Comments
	ORG 0000H	
	MOV 30H, #08H	
	MOV 31H, #05H	
	MOV A, 30H	
BACK:	ADD A, 31H	
	MOV P1, A	
	JNC BACK	
HERE:	SJMP HERE	
	END	

Let a=8, d=5

08, 0d, 12h, 17h, ...
... FFH

(22) Generate Geometric series upto FFH. Display result in P1

Label	Mnemonics	Comments
	ORG 0000H	
	MOV 30H, 04H	
	MOV 31H, 03H	
	MOV A, 30H	
BACK:	MOV B, 31H	

Let a = 04
r = 03

```
        MUL   AB
        MOV   P1, A
        MOV   R1, B
        MOV   R2, A
        MOV   A, B
        ANL   A, R1
        JZ    BACK
HERE:   SJMP  HERE
        END
```

(23) Generate Fibonacci's series upto FFH. Store result starting from 30H.

Label	Mnemonics	Comments
	ORG 0000H	
	MOV R2, #00H	00H, 01H, 01H, 02H, ...
	MOV R3, #01H	
	MOV R0, #30H	
BACK:	MOV A, R2	
	MOV @R0, A	
	INC R0	
	ADD A, R3	
	MOV R2, 03H	
	MOV R3, A	
	JNC BACK	
HERE:	SJMP HERE	
	END	

(24) Find the factorial of the number less than 06

Label	Mnemonics	Comments
	ORG 0000H	
	MOV R2, #05H	
	MOV R3, #00H	; Store factorial
	MOV A, R2	
BACK:	DEC R2	
	MOV B, R2	
	MUL AB	
	MOV R3, A	
	CJNE R2, #01H, BACK	
HERE:	SJMP HERE	
	END	

Arrange array of N bytes stored starting from 30h in ascending order.

Label	Mnemonics	Comments
	ORG 0000H	
	MOV R1, #09H	; Count N-1
BACK2:	MOV R2, #09H	; Count N-1
	MOV R0, #30H	; Pointer
BACK1:	MOV A, @R0	
	INC R0	
	MOV 03H, @R0	
	CJNE A, 03H, NEXT	
NEXT:	JC NEXT1	
	MOV 04h, @R0	; Save RAM Content in R4
	DEC R0	
	MOV @R0, 04h	
	INC R0	; Exchange nos
	MOV @R0, A	
NEXT1:	DJNZ R2, BACK1	
	DJNZ R1, BACK2	
HERE:	SJMP HERE	
	END	

(26) Arrange array of N bytes stored starting from 30H in the descending order

* Same program but instead of JC write JNC.

CALL instructions

CALL instructions are used to CALL Subroutines

* Subroutine is a piece of program written separately from the main program to perform a dedicated task

Advantages

* It makes a program more structural
* It improves the readibility of the program
* It saves the memory space

There are two unconditional CALL instructions in 8051

ACALL addr ; Call the subroutine located on the same page (within 2KB) & Push the address of the ins -tion immediately after the CALL on stack

LCALL laddr ; CALL the subroutine anywhere in the program space & push the address of the instruction immediat following the CALL on the stack.

RET ; Pop 2 bytes from the stack into PC.

Sequence of operations that occur during execution of CALL instructions :

1. Address of next instruction after the CALL instruction is pushed on to the stack. Address of next instruction is present in PC. LSB of address is stored on the stack
2. Stack Pointer is incremented by 2
3. Address of the subroutine given in the CALL instruction is placed in the PC
4. Subroutine is executed

Subroutine

1. Return instructions is used at the end of subroutine
2. Two bytes are poped back to PC from the stack
3. SP is decremented by 2
4. µC starts execution of instroction after CALL instruction.

① Get the contents of the PC to DPTR

```
        MOV 81H, #07H     ; Initialize SP at 07H
        MOV A, #0FFH
        MOV R1, #0AH
        ACALL Subr
HERE:   SJMP HERE
```

```
SUBR:   POP   83h      ; 83 address of DPH
        POP   82h      ; 82 address of DPL
        RET
```

(OR)

```
        MOV   81h,#07h
        ACALL SUBR
HERE:   SJMP  HERE

SUBR:   MOV   83h,09h  ; Copy high byte of PC to DPH
        MOV   82h,08h  ; Copy ~~high~~ low byte of PC to DPL
        RET
```

② Get the contents of the DPTR to the PC

```
        MOV   81h,#07h
        MOV   DPTR,#0000h
        ACALL SUBR
HERE:   SJMP  HERE

SUBR:   MOV   09h,83h
        MOV   08h,82h
        RET
```

③ Get ~~Any~~ any 2 bytes you wish to PC

```
        MOV   A,#02h
        MOV   R1,#00h
        ACALL SUBR
HERE:   SJMP  HERE

SUBR:   MOV   09h,R1
        MOV   08h,A
        RET
```

④ Write a program for hexadecimal upcounter 00-FFh. Use subroutine for time delay using two registers R1 & R2. Display count in the port P1

```
        MOV A,#00h
BACK:   ACALL DELAY
        MOV P1,A
        INC A
        CJNE A,#00h,BACK
HERE:   SJMP HERE
DELAY:  MOV R1,#0FFh
LOOP2:  MOV R2,#0FFh
LOOP1:  DJNZ R2,LOOP1
        DJNZ R1,LOOP2
```

⑤ Write a program for hexadecimal Downcounter FFH-00H. Use subroutine for time delay using two registers R1 & R2. Display count in the port P1

⑥ Write a program for decimal (8-bit) upcounter 00-99. Use subroutine for time delay using 2 registers R1&R2. Display count in the port P1

```
        MOV A,#00h
BACK:   ACALL DELAY
        MOV P1,A
        ADD A,#01h
        DA A
        CJNE A,#00h,BACK
HERE:   SJMP HERE
DELAY:  MOV R1,#0FFh
LOOP2:  MOV R2,#0FFh
LOOP1:  DJNZ R2,LOOP1
        DJNZ R1,LOOP2
        RET
```

⑦ Write a program for decimal (8-bit) down counter 99-00h

```
        MOV  A,#00h
BACK:   ADD  A,#99h
        DA   A
        MOV  P1,A
        ACALL DELAY
        CJNE  A,#00h,BACK
HERE:   SJMP  HERE
DELAY:  MOV  R1,#0FH
LOOP2:  MOV  R2,#0FFH
LOOP1:  DJNZ  R2,LOOP1
        DJNZ  R1,LOOP2
        RET
```

⑧ Write a program for 16 bit hexadecimal upcounter 0000-FFFFh Use subroutine for time delay using two registers R1 & R2 Display count in Ports P0 (LSB) & P1 (MSB)

```
        MOV R1,#00h
BACK2:  MOV R2,#00h
BACK1:  MOV A,R2
        ACALL DELAY
        MOV  P0,A
        MOV  A,R1
        MOV  P1,A
        INC  R2
        CJNE R2,#00h,BACK1
        INC  R1
        CJNE R1,#00h,BACK2
HERE:   SJMP HERE
DELAY:  PUSH 01h
        PUSH 02h
        MOV  R1,#0Fh
LOOP2:  MOV  R2,#0FFh
LOOP1:  DJNZ R2,LOOP1
        DJNZ R1,LOOP2
        POP  02h
        POP  01h
        RET
```

⑨ Write a program to convert an ASCII number into binary

```
        MOV  A, 30H
        CLR  C
        SUBB A, #30H
        CJNE A, #09H, NEXT
NEXT:   JC   NEXT1
        CLR  C
        SUBB A, #07h
NEXT1:  MOV  R2, A
HERE:   SJMP HERE
```

⑩ Write a program to convert a binary digit to ASCII number

```
        MOV  A, 30H
        ADD  A, #30H
        MOV  R1, 30H
        CJNE R1, #09, NEXT
NEXT:   JC   NEXT1
        ADD  A, #07H
NEXT1:  MOV  R2, A
HERE:   SJMP HERE
```