MIKROKONTROLER 8051

ASEMBLERSKE INSTRUKCIJE

Asemblerske instrukcije za 8051 – razmjena podataka

MOV

MOV instrukcija razmjenjuje podatke između dva specificirana bajta. Bajt koji je specificiran kao drugi operand kopira se na lokaciju specificiranu prvim operandom.

MOV @Ri, #25 => *(Ri)=25*	**MOV @Ri, A** => *(Ri)=A*
MOV @Ri, 32 => *(Ri)=(32)*	**MOV A, #FFh** => *A=FFh*
MOV A,@Ri => *A=(Ri)*	**MOV A,45** => *A=(45)*
MOV A,Rn => *A=Rn*	**MOV 17,33** => *(17)=(33)*
MOV 22,#25h => *(22)=25h*	**MOV 26,#58h** => *(26)=58h*
MOV 57,@Ri => *(57)=(Ri)*	**MOV 45,A** => *(45)= A*
MOV 34,Rn => *(34)=Rn*	**MOV Rn, #75** => *Rn=75*
MOV Rn,A => *Rn=A*	**MOV Rn,47** => *Rn=(47)*

i=0,1; n=0,1,2,...7

Upis adrese u registar DPTR (2 bajta)

MOV DPTR,#1234h => *DPTR=1234h*

Upis vrijednosti bita u Carry flag ili vrijednosti Carry flag-a u bit

MOV C,5 => *C=(5bit)* *(5bit) je na RAM adresi 20h.5*

MOV 3,C => *(3bit)=C*

MOV P2.1,C => *P2.1=C*

Asemblerske instrukcije za 8051 – razmjena podataka

MOVC

MOVC instrukcija smješta podatke iz ROM memorije u akumulator

MOVC A,@A+DPTR => *A=(A+DPTR)*

MOVC A,@A+PC => *A=(A+PC)*

MOVX instrukcija razmjenjuje podatke između akumulatora i externe RAM memorije. Eksterna memorija može biti adresirana 16-bitno u DPTR registru ili 8-bitno u R0 ili R1 registrima. Kada se koristi 8-bitno adresiranje, u port 2 mora biti upisan viši bajt adrese

MOVX @R1,A => *(R1)=A*

MOVX A, @R0 => *A=(R0)*

MOVX @DPTR,A => *(DPTR)=A*

MOVX A, @DPTR => *A=(DPTR)*

Asemblerske instrukcije za 8051 – razmjena podataka

SWAP

SWAP instrukcija razmjenjuje gornji i donji nibl podataka u akumulatoru.

SWAP A => $A[3\text{-}0] \leftrightarrow A[7\text{-}4]$

XCH instrukcija upisuje podatke iz specificiranog operanda u akumulator, a istovremeno upisuje prethodni sadržaj akumulatora u operand

XCH A,47 => $A \leftrightarrow (47)$

XCH A,Rn => $A \leftrightarrow Rn$ $n=0,1,...,7$

XCH A, @Ri => $A \leftrightarrow (Ri)$ $i=0,1$

XCHD instrukcija razmjenjuje sadržaj nižeg nibla akumulatora sa nižim niblom memorijske lokacije specificirane u internom RAM-u. Internom RAM-u se pristupa indirektno preko R0 i R1. Viši nibl oba operanda se ne mjenja.

XCHD A, @Ri => $A[3\text{-}0] \leftrightarrow (Ri)[3\text{-}0]$ $i=0,1$

Asemblerske instrukcije za 8051 – matematičke operacije

ADD

ADD instrukcija dodaje vrijednost bajta akumulatoru i rezultat smješta nazad u akumulator. Operacija utiče i na sadržaj nekih od flagova u PSW

ADD A,#35 => $A=A+35$

ADD A,57 => $A=A+(57)$

ADD A,Rn => $A=A+Rn$ $n=0,1,...,7$

ADD A, @Ri => $A=A+(Ri)$ $i=0,1$

ADDC instrukcija dodaje vrijednost bajta i carry flaga akumulatoru, a rezultat smješta u akumulator.

ADDC A,#35 => $A=A+C+35$

ADDC A,57 => $A=A+C+(57)$

ADDC A,Rn => $A=A+C+Rn$ $n=0,1,...,7$

ADDC A, @Ri => $A=A+C+(Ri)$ $i=0,1$

Asemblerske instrukcije za 8051 – matematičke operacije

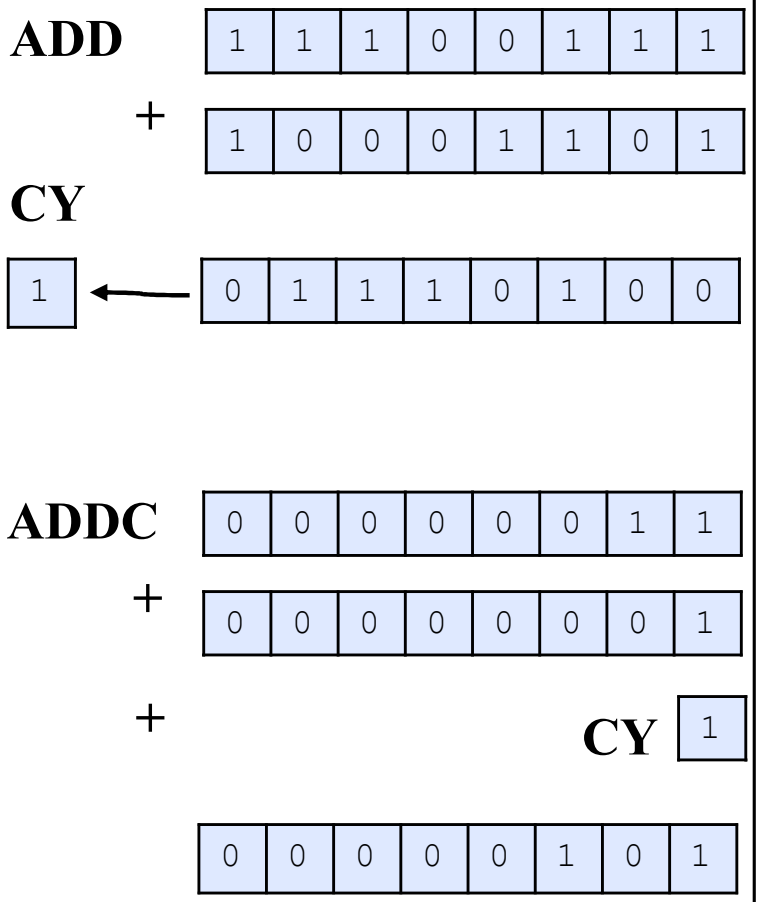
Primer: Sabiranje dva 16-bitna broja

999 => 3E7h

+397 => 18Dh

1396 574 h

```
MOV A,#E7h    ;Donji bajt prvog operanda
ADD A, #8Dh   ;Saberi donje bajtove
MOV R6,A      ;Sačuvaj donji bajt rezultata
MOV A, #3h    ;Gornji bajt prvog operanda
ADDC A, #1h   ;Saberi gornje bajtove
MOV  R7,A     ;Sačuvaj gornji bajt rezultata
```



Asemblerske instrukcije za 8051 – matematičke operacije

SUBB

SUBB instrukcija oduzima definisani bajt i carry flag od akumulatora. Rezultat se smješta u akumulator. Ako je rezultat oduzimanja zahtjeva borrow bit, setuje se carry flag.

SUBB A, #21 => *A=A-C-21*

SUBB A,44 => *A=A-C-(44)*

SUBB A,Rn => *A=A-C-Rn* *n=0,1,...,7*

SUBB A, @Ri => *A=A-C-(Ri)* *i=0,1*

Asemblerske instrukcije za 8051 – matematičke operacije

Primer: Oduzimanje dva 16-bitna broja

```
  1890  =>  762h
- 662   =>  296h
 1228        4CCh
```

```
CLR C          ;CY=0
MOV A,#62h     ;Donji bajt prvog operanda
SUBB A, #96h   ;Oduzmi donje bajtove
MOV R6,A       ;Sačuvaj donji bajt rezultata
MOV A, #7h     ;Gornji bajt prvog operanda
SUBB A, #2h    ;Oduzmi gornje bajtove
MOV  R7,A      ;Sačuvaj gornji bajt rezultata
```

Asemblerske instrukcije za 8051 – matematičke operacije

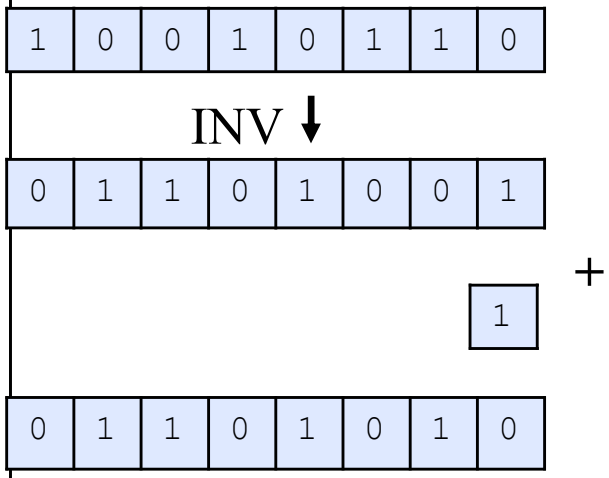
Oduzimanje dva 16-bitna broja

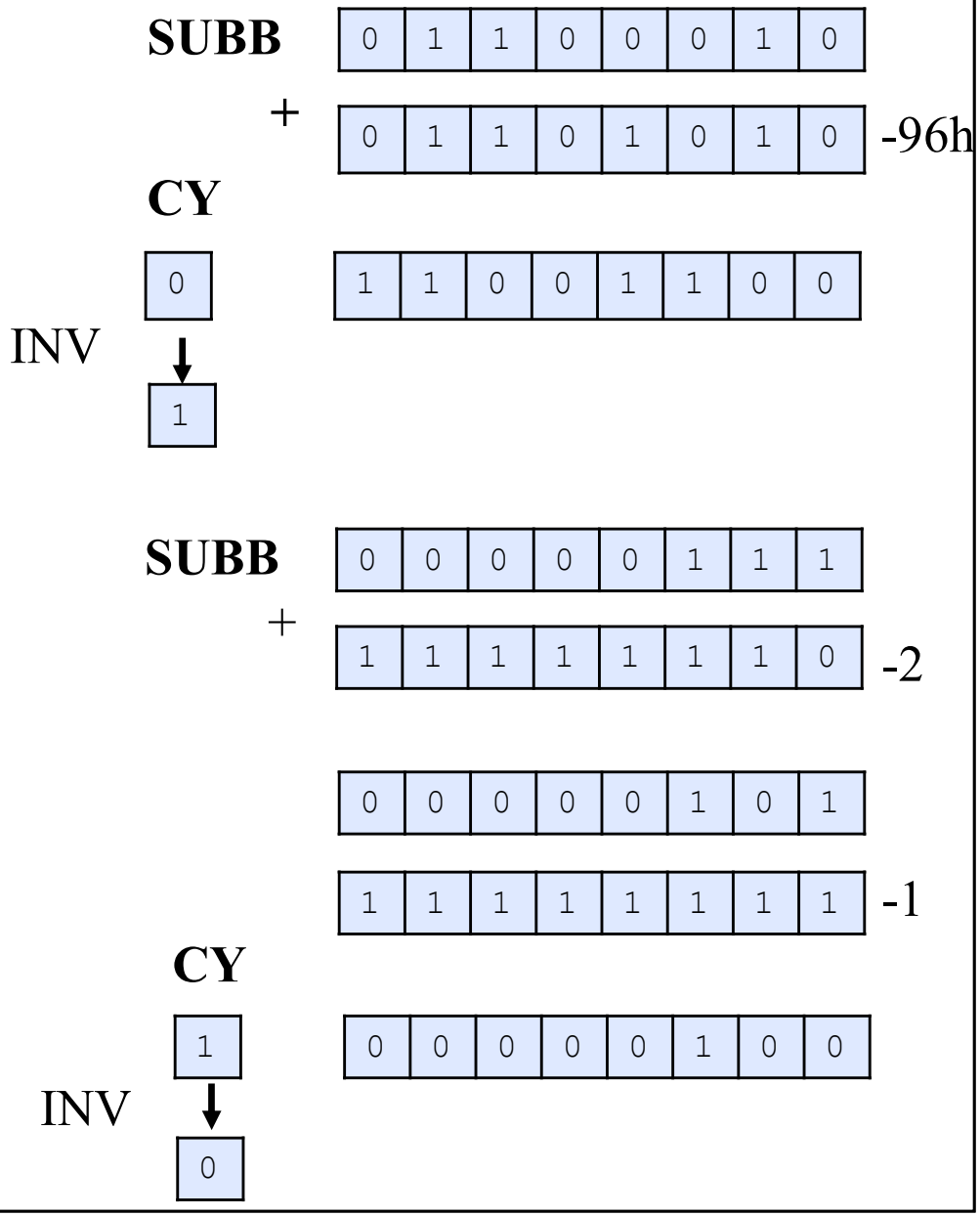
1890 => 762h

\- 662 => 296h

1228 4CCh

-96h





Asemblerske instrukcije za 8051 – matematičke operacije

INC instrukcija inkrementira specificirani operand za 1. Inkrementiranjem 0FFh dobija se 00h. Za DPTR, inkrementiranjem 0FFFFh dobija se 00h. Instrukcija ne utiče na flagove u PSW. Kada se instrukcija koristi da bi se modifikovala vrijednosti izlaznog porta, vrijednost koja se uzima sa porta čita se iz izlaznog latch-a, a ne sa ulaznih pinova porta.

INC 65 => *(65)=(65)+1* **INC A** => *A=A+1*

INC Rn => *Rn=Rn+1* *n=0,1,...,7*

INC @Ri => *(Ri)=(Ri)+1* *i=0,1*

INC DPTR => *DPTR=DPTR+1*

DEC instrukcija dekrementira specificirani operand za 1. Dekrementiranjem 00h dobija se 0FFh. Instrukcija ne utiče na flagove u PSW. Kada se instrukcija koristi da bi se modifikovala vrijednosti izlaznog porta, vrijednost koja se uzima sa porta čita se iz izlaznog latch-a, a ne sa ulaznih pinova porta.

DEC 72 => *(72)=(72)-1*

DEC A => *A=A-1*

DEC Rn => *Rn=Rn-1* *n=0,1,...,7*

DEC @Ri => *(Ri)=(Ri)-1* *i=0,1*

Asemblerske instrukcije za 8051 – matematičke operacije

MUL instrukcija vrši množenje 8-bitnog neoznačenog cijelog broja u akumulatoru i 8-bitnog neoznačenog cijelog broja u B registru. Rezultat je 16-bitni proizvod. Niži bajt proizvoda smješta se u akumulator, a viši u registar B. OV flag se setuje ukoliko je proizvod veći od 255, a carry flag se uvijek resetuje.

MUL AB => $BA=A*B$

DIV instrukcija dijeli neoznačeni cijeli 8-bitni broj u akumulatoru sa neoznačenim cijelim 8-bitnim brojem u registru B. Nakon dijeljenja, rezultat dijeljenja se smješta u akumulator, a ostatak u registar B. Carry flag i OV flag se resetuju.
Ukoliko registar B ima vrijednost 00h, operacija dijeljenja je nedefinisana, vrijednosti akumulatora i registra B nakon dijeljenja su nedefinisane, a OV flag se setuje kako bi se prijavila greška dijeljenja sa nulom.

DIV AB => $AB=A/B$

Asemblerske instrukcije za 8051 – operacije sa bit promj.

SETB instrukcija postavlja operand na vrijednost 1. Može da se primjeni samo na bit promjenljive. Instrukcija može da se primjeni na carry flag ili bilo koji drugi direktno adresibilan bit.

SETB C => *C=1*

SETB P1.3 => *P1.3=1*

SETB 12 => *(12bit)=1* *(12bit) je na RAM adresi 21h.5*

CLR instrukcija postavlja operand na vrijednost 0. Primjenjuje se na bit promjenljive, ali može da se primjeni i na čitav akumulator.

CLR C => *C=0*

CLR P2.1 => *P2.1=0*

CLR 16 => *(16bit)=0* *(16bit) je na RAM adresi 22h.0*

CLR A => *A=0*

Asemblerske instrukcije za 8051 – logičke bit operacije

ANL instrukcija obavlja operaciju logičko „I“ između odgovarajućih bita dva byte operanda ili između dva bit operanda. Rezultat se smješta u prvi operand.

Kada se ova instrukcija koristi kako bi se modifikovao izlazni port, vrijednost koja se uzima sa porta čita se iz njegovog latch-a, a ne ulaznih pinova porta.

ANL A,#25h => *A=A and 25h*

ANL A,@Ri => *A=A and (Ri); i=0,1*

ANL A,40h => *A=A and (40h)*

ANL A,Rn => *A=A and Rn; n=0,...,7*

ANL 50h,A => *(50h)=(50h) and A*

ANL 45h,#33h => *(45h)=(45h) and 33*

ANL C, /11 => *C=C and not (11bit);*

ANL C, 11 => *C=C and (11bit);* *(11bit) je na RAM adresi 21h.4*

Asemblerske instrukcije za 8051 – logičke bit operacije

ORL instrukcija obavlja operaciju logičko „ILI“ između odgovarajućih bita dva byte operanda ili između dva bit operanda. Rezultat se smješta u prvi operand.

Kada se ova instrukcija koristi kako bi se modifikovao izlazni port, vrijednost koja se uzima sa porta čita se iz njegovog latch-a, a ne ulaznih pinova porta.

ORL A,#25h => *A=A or 25h*

ORL A,@Ri => *A=A or (Ri); i=0,1*

ORL A,40h => *A=A or (40h)*

ORL A,Rn => *A=A or Rn; n=0,...,7*

ORL 50h,A => *(50h)=(50h) or A*

ORL 45h,#33h => *(45h)=(45h) or 33*

ORL C, /11 => *C=C or not (11bit);*

ORL C, 11 => *C=C or (11bit);* *(11bit) je na RAM adresi 21h.4*

Asemblerske instrukcije za 8051 – logičke bit operacije

XRL instrukcija obavlja operaciju logičko „EKSKLUZIVNO ILI“ između odgovarajućih bita dva byte operanda. Rezultat se smješta u prvi operand.

Kada se ova instrukcija koristi kako bi se modifikovao izlazni port, vrijednost koja se uzima sa porta čita se iz njegovog latch-a, a ne ulaznih pinova porta.

XRL A,#25h => *A=A xor 25h*

XRL A,@Ri => *A=A xor (Ri); i=0,1*

XRL A,40h => *A=A xor (40h)*

XRL A,Rn => *A=A xor Rn; n=0,...,7*

XRL 50h,A => *(50h)=(50h) xor A*

XRL 45h,#33h => *(45h)=(45h) xor 33*

CPL instrukcija vrši logički komplement bit operanda ili akumulatora

CPL C => *C=not C*

CPL 15 => *(15bit)=not (15bit); (15bit) je na RAM adresi 21h.7*

CPL A => *A=not A*

Asemblerske instrukcije za 8051 – rotiranje

RL instrukcija rotira 8 bita u akumulatoru lijevo za jednu poziciju (na poziciju koja je viša za jedan bit).

RL A => $A[n+1]=A[n],\ n=0,\ldots,6$

$A[0]=A[7]$

RLC instrukcija rotira 8 bita u akumulatoru i carry flag bit lijevo za jednu poziciju (na poziciju koja je viša za jedan bit).

RLC A => $A[n+1]=A[n],\ n=0,\ldots,6$

$A[0]=C,\ C=A[7]$

RR instrukcija rotira 8 bita u akumulatoru desno za jednu poziciju (na poziciju koja je niža za jedan bit).

RR A => $A[n]=A[n+1],\ n=0,\ldots,6$

$A[7]=A[0]$

RRC instrukcija rotira 8 bita u akumulatoru i carry flag bit desno za jednu poziciju (na poziciju koja je niža za jedan bit).

RRC A => $A[n]=A[n+1],\ n=0,\ldots,6$

$A[7]=C,\ C=A[0]$

Asemblerske instrukcije za 8051 – bezuslovni skok

SJMP instrukcija prebacuje izvršavanje programa na specificiranu adresu. Nova adresa može da bude najviše 128 bajtova prije adrese sljedeće instrukcije, odnosno 127 bajtova nakon adrese sljedece instrukcije. Zauzima 2 bajta u memoriji i traje 2 mašinska ciklusa.

SJMP label => *PC=PC+2*
PC=PC+offset

10000000
offset

AJMP instrukcija prebacuje izvršavanje programa na specificiranu adresu. Nova adresa mora da bude unutar istog bloka od 2kB memorije kao i sljedeca instrukcija. Zauzima 2 bajta u memoriji i traje 2 mašinska ciklusa.

AJMP label => *PC=PC+2*
PC[10-0]=adresa(label)

$A_{10}A_9A_800001$
$A_7A_6A_5A_4A_3A_2A_1A_0$

LJMP instrukcija prebacuje izvršavanje programa na specificiranu adresu

Zauzima 3 bajta u memoriji i traje 2 mašinska ciklusa.

LJMP label => *PC=adresa(label)*

00000010
$A_{15}A_{14}A_{13}A_{12}A_{11}A_{10}A_9A_8$
$A_7A_6A_5A_4A_3A_2A_1A_0$

JMP instrukcija prebacuje izvršavanje programa na adresu koja se dobija sabiranje vrijednosti u akumulatoru i DPTR registru

JMP @A+DPTR => *PC=A+DPTR*

Asemblerske instrukcije za 8051 – uslovni skok (bit uslov)

JB instrukcija prebacuje izvršavanje programa na adresu specificiranu drugim operandom, ako je vrijednost bita navedenog kao prvi operand 1. Bit koji se testira se ne modifikuje.

JB P2.3, labela => *PC=PC+3*

if P2.3== 1

PC=adresa(labela)

JBC instrukcija prebacuje izvršavanje programa na adresu specificiranu drugim operandom, ako je vrijednost bita navedenog kao prvi operand 1. Ukoliko bit koji se testira ima vrijednost 1, postavlja se na 0.

Ako se u instrukciji koristi bit sa nekog od portova, njegova vrijednost se čita iz latch-a.

JBC 28, labela => *PC=PC+3*

if (28bit)== 1

(28bit)=0

PC=adresa(labela)

Asemblerske instrukcije za 8051 – uslovni skok (bit uslov)

JNB instrukcija prebacuje izvršavanje programa na adresu specificiranu drugim operandom, ako je vrijednost bita navedenog kao prvi operand 0.

JNB 12, labela => *PC=PC+3*

if (12bit)== 0

PC=adresa(labela)

JC instrukcija prebacuje izvršavanje programa na specificiranu adresu ako je carry flag na 1.

JC labela => *PC=PC+2*

if C== 1

PC=adresa(labela)

JNC instrukcija prebacuje izvršavanje programa na specificiranu adresu ako je carry flag na 0.

JNC labela => *PC=PC+2*

if C== 0

PC=adresa(labela)

Asemblerske instrukcije za 8051 – uslovni skok (bajt uslov)

JNZ instrukcija prebacuje izvršavanje programa na specificiranu adresu ako vrijednost u akumulatoru nije 0. Ako je vrijednost u akumulatoru 0, prelazi se na sljedeću instrukciju. Instrukcija ne modifikuje akumulator niti bilo koji od flagova

JNZ labela => *PC=PC+2*

if A<> 0

PC=adresa(labela)

JZ instrukcija prebacuje izvršavanje programa na specificiranu adresu ako je vrijednost u akumulatoru 0. Ako je vrijednost u akumulatoru različita od 0, prelazi se na sljedeću instrukciju. Instrukcija ne modifikuje akumulator niti bilo koji od flagova

JZ labela => *PC=PC+2*

if A==0

PC=adresa(labela)

Asemblerske instrukcije za 8051 – uslovni skok (bajt uslov)

CJNE instrukcija upoređuje prva dva operanda i prebacuje izvršavanje programa na specificiranu adresu ako njihove vrijednosti nisu iste. Ako prvi operand ima vrijednost manju od drugog, u carry flag se upisuje 1. U suprotnom u carry flag se upisuje 0

CJNE @R0,#35,labela => *PC=PC+3*
if (R0) <> 35
PC=adresa(labela)
if (R0) < 35
C=1
else
C=0

CJNE A,#47,labela => *PC=PC+3*
if A <> 47
PC=adresa(labela)
if A < 47
C=1
else
C=0

Asemblerske instrukcije za 8051 – uslovni skok (bajt uslov)

CJNE A,63h,labela =>

PC=PC+3
if A <> (63h)
PC=adresa(labela)
if A < (63h)
C=1
else
C=0

CJNE R5,#21,labela =>

PC=PC+3
if R5 <> 21
PC=adresa(labela)
if R5 < 21
C=1
else
C=0

Asemblerske instrukcije za 8051 – uslovni skok (bajt uslov)

DJNZ instrukcija dekrementira bajt koji je definisan prvim operandom i, ako rezultujuća vrijednost nije 0, prebacuje izvršavanje programa na adresu koja je specificirana kao drugi operand.

DJNZ 57,labela => *PC=PC+3*
(57)=(57)-1
if (57) <> 0
PC=adresa(labela)

DJNZ R6,labela => *PC=PC+2*
R6=R6-1
if R6 <> 0
PC=adresa(labela)

Asemblerske instrukcije za 8051 – uslovni skok (bajt uslov)

Primer: Za mikrokontroler 8051 koji radi sa oscilatorom 24MHz napisati program koji na 1s invertuje P0.0

24MHz => Mašinski ciklus=1/2000000s=0.5μs

DJNZ - *izvršavanje traje dva mašinska ciklusa*

```
main: CPL P0.0
      MOV R2, #200
kas1: MOV R3,#100
kas2: MOV R4, #50
kas3: DJNZ R4, kas3
      DJNZ R3, kas2
      DJNZ R2, kas1
      LJMP  main
```

Asemblerske instrukcije za 8051 – poziv podrutine

LCALL instrukcija poziva podrutinu koja se nalazi na specificiranoj adresi. Instrukcija prvo u stack stavlja adresu sljedeće instrukcije, a nakon toga prebacuje izvršavanje programa na adresu podrutine.

LCALL labela => *PC=PC+3*
SP=SP+1
(SP)=PC[7-0]
SP=SP+1
(SP)=PC[15-8]
PC=adresa(labela)

00010010
$A_{15}A_{14}A_{13}A_{12}A_{11}A_{10}A_9A_8$
$A_7A_6A_5A_4A_3A_2A_1A_0$

ACALL instrukcija poziva podrutinu koja se nalazi na specificiranoj adresi. Instrukcija prvo u stack stavlja adresu sljedeće instrukcije, a nakon toga prebacuje izvršavanje programa na adresu podrutine. Adresa podrutine mora da bude u istom bloku od 2kB memorije kao i sljedeća instrukcija pošto se pri skoku na adresu podrutine koristi 5 najviših bita koji su već upisani u PC.

ACALL labela => *PC=PC+2*
SP=SP+1
(SP)=PC[7-0]
SP=SP+1
(SP)=PC[15-8]
PC[10-0]=adresa(labela)

$A_{10}A_9A_8$10001
$A_7A_6A_5A_4A_3A_2A_1A_0$

Asemblerske instrukcije za 8051 – povratak iz podrutine

RET instrukcija služi za povratak programa iz podrutine. Instrukcija iz stack-a u PC upisuje adresu na koju treba da se vrati program. To bi trebalo da je adresa ispod ACALL ili LCALL instrukcije.

RET => *PC[15-8]=(SP)*
SP=SP-1
PC[7-0]=(SP)
SP=SP-1

RETI instrukcija se koristi za povratak iz procedure za obradu prekida. Instrukcija iz stack-a u PC upisuje adresu na koju treba da se vrati program i vraća logiku za obradu prekida u stanje koje omogućava prihvatanje novih prekida. Izvršavanje programa se vraća na instrukciju neposredno nakon tačke u kojoj je detektovan prekid.
Ako pri izvršavanju instrukcije RETI već postoji prekid koji čeka na obradu, obavlja se jedna instrukcija na adresi na koju se program vratio prije nego što se obradi prekid koji čeka.

RETI => *PC[15-8]=(SP)*
SP=SP-1
PC[7-0]=(SP)
SP=SP-1

Asemblerske instrukcije za 8051 – stack

PUSH instrukcija inkrementira stack pointer i smješta vrijednost koja je specificirana operandom na RAM adresu koja je upisana u stack pointer.
PUSH 67 => *SP=SP+1*
(SP)=(67)

POP instrukcija čita bajt sa lokacije u RAM memoriji koja je upisana u stack pointer i smješta je na lokaciju koja je specificirana operandom. Nakon toga stack pointer se dekrementira.
POP 65 => *(65) = (SP)*
SP= SP-1

Asemblerske instrukcije za 8051

NOP instrukcija ne proizvodi nikakvu akciju. Izvršavanje se nastavlja sa sljedećom instrukcijom. Obično se koristi kako bi se generisalo kašnjenje pri izvršavanju. Instrukcija traje jedan mašinski ciklus.

NOP => *PC=PC+1*

DA instrukcija podešava osmobitnu vrijednost u akumulatoru koja je dobijena prethodnim sabiranjem dvije promjenljive u BCD formatu, kako bi se dobile dvije četverobitne cifre.

DA => *PC=PC+1*

if (A[3-0]>9) or (AC==1)

A=A+6

if (A[7-4]>9) or (C==1)

A=A+60h