Mašinske instrukcije i programi

Vladimir Filipović
vladaf@matf.bg.ac.yu

Brojevi, aritmetičke operacije i znaci

- **Kao što smo videli, logička kola u računaru predstavljaju informacije preko električnih impulsa sa dve vrednosti (dva nivoa napona).**
- **Stoga, najprirodniji način za predstavljanje brojeva u računaru je predstavljanje kao niske binarnih cifara, tj. niske bitova.**
- **Znakovi se takođe predstavljaju preko niza bitova koji se naziva kod znaka.**

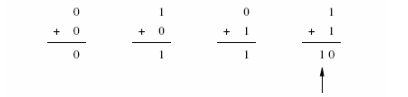
Predstavljanje brojeva

- **Neoznačeni celi brojevi se predstavljaju u "klasičnom" binarnom obliku.**
- **Za predstavljane označenih celih brojeva koristi se jedna od sledećih notacija:**
  - **Znak i apsolutna vrednost**
  - **Nepotpuni komplement (1's complement)**
  - **Potpuni komplement (2's complement)**

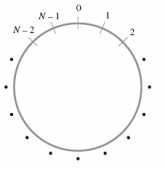
Predstavljanje brojeva

B	Values represented		
$b_3b_2b_1b_0$	Sign and magnitude	1's complement	2's complement
0 1 1 1	+7	+7	+7
0 1 1 0	+6	+6	+6
0 1 0 1	+5	+5	+5
0 1 0 0	+4	+4	+4
0 0 1 1	+3	+3	+3
0 0 1 0	+2	+2	+2
0 0 0 1	+1	+1	+1
0 0 0 0	+0	+0	+0
1 0 0 0	−0	−7	−8
1 0 0 1	−1	−6	−7
1 0 1 0	−2	−5	−6
1 0 1 1	−3	−4	−5
1 1 0 0	−4	−3	−4
1 1 0 1	−5	−2	−3
1 1 1 0	−6	−1	−2
1 1 1 1	−7	−0	−1

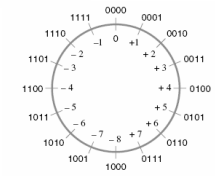
Operacije sa brojevima



sabiranje cifara



kružna reprezentacija
za mod N



kružna reprezentacija
Za 4-bitni puni komplement

Operacije sa brojevima u potpunom komplementu

(a)	0 0 1 0	(+2)	(b)	0 1 0 0	(+4)
	+ 0 0 1 1	(+3)		+ 1 0 1 0	(−6)
	0 1 0 1	(+5)		1 1 1 0	(−2)
(c)	1 0 1 1	(−5)	(d)	0 1 1 1	(+7)
	+ 1 1 1 0	(−2)		+ 1 1 0 1	(−3)
	1 0 0 1	(−7)		0 1 0 0	(+4)
(e)	1 1 0 1	(−3)	⇒	1 1 0 1	
	− 1 0 0 1	(−7)		+ 0 1 1 1	
				0 1 0 0	(+4)
(f)	0 0 1 0	(+2)	⇒	0 0 1 0	
	− 0 1 0 0	(+4)		+ 1 1 0 0	
				1 1 1 0	(−2)

Operacije sa brojevima u potpunom komplementu

(g)	0 1 1 0	(+6)	⇒	0 1 1 0	
	− 0 0 1 1	(+3)		+ 1 1 0 1	
				0 0 1 1	(+3)
(h)	1 0 0 1	(−7)	⇒	1 0 0 1	
	− 1 0 1 1	(−5)		+ 0 1 0 1	
				1 1 1 0	(−2)
(i)	1 0 0 1	(−7)	⇒	1 0 0 1	
	− 0 0 0 1	(+1)		+ 1 1 1 1	
				1 0 0 0	(−8)
(j)	0 0 1 0	(+2)	⇒	0 0 1 0	
	− 1 1 0 1	(−3)		+ 0 0 1 1	
				0 1 0 1	(+5)

Prekoračenje kod celobrojne aritmetike

•Aritmetičko prekoračenje nastupa onda kada se rezultat aritmetičke operacije ne nalazi u opsegu brojeva koji se mogu predstaviti u registru računara.

- •Prekoračenje može nastupiti pri sabiranju dva broja broja istog znaka
- •Signal prenosa sa mesta najveće težine nije ni potreban ni dovoljan indikator da je došlo do prekoračenja.

Memorijske lokacije i adrese

•Memorija se sastoji od više miliona memorijskih ćelija, koje čuvaju jedan bit informacije.

•Obično se ne radi sa pojedinačnim bitovima u memoriji, već se pri radu bitovi grupišu u grupe fiksne veličine.

•Svaka grupa od n bitova se označava kao reč, a broj n predstavlja dužinu reči.

•Dužine reči kod današnjih modernih računara su između 16 i 64 bita.

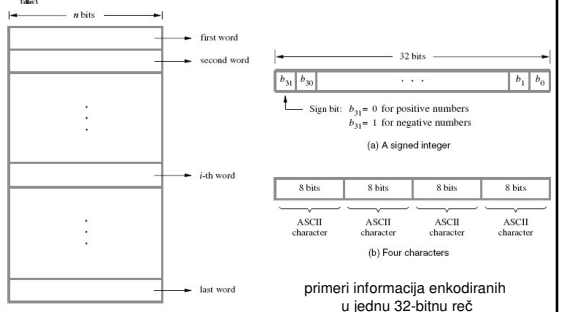
•Mašinske instrukcije mogu zahtevati jednu ili više reči za svoju reprezentaciju.

•Pristup memoriji zahteva postojanje različitih adresa za svaku lokaciju.

•Sve (teorijski) moguće adrese oformljuju adresni prostor računara.

•Bajt-adresibilna memorija je memorija u kojoj uzastopne adrese referišu na uzastopne bajtove u memoriji – takav je slučaj kod skoro svih modernih računara.

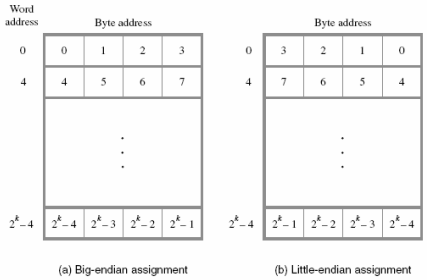
Memorijske lokacije i adrese



shematski prikaz memorije

primeri informacija enkodiranih u jednu 32-bitnu reč

Adresiranje bajtova i reči



Poravnavanje reči

•Za 32-bitne reči, prirodno je da memorijske lokacije bajtova od kojih te reči počinju budu na adresama 0,4,8,12,16,20,24,32,...

•U takvom slučaju, kaže se da reči imaju poravnatu adresu.

•Nema fundamentalnog razloga zbog kog reči ne bi mogle počinjati od bilo koje adrese. Za reči u memoriji kod kojih je takav slučaj, kaže se da imaju neporavnatu adresu.

Pristup znacima i znakovnim niskama

Memorijske operacije

- **Dve osnovne memorijske operacije:**
  - **Čitanje iz memorije (load, read, fetch)**
  - **Upis u memoriju (store, write)**
- **Informacija koja se čita ili piše je jedna reč ili jedan bajt koji se u jednoj operaciji prenosi između memorije i procesora (preciznije, nekog od registara u procesoru).**
- **Registri procesora predstavljaju izvor ili odredište kod operacije prenosa u ili iz memorije.**
- **Pri prenosu čitanju ili upisu bajta iz memorije, obično se koristi bajt manje težine registra.**

Instrukcije i sekvence instrukcija

- **Računar mora sadržavati instrukcije sposobne da podrže četiri tipa operacija:**
  - **Prenos podataka između memorije i registara procesora**
  - **Aritmetičke i logičke operacije nad podacima**
  - **Programske sekvence i kontrolu**
  - **Prenos sa ulaza/izlaza**

Instrukcije i sekvence instrukcija

- **Notacija sa registarskim prenosom**
- **Notacija asemblerskog jezika**

Osnovni tipovi instrukcija

- **Troadresne instrukcije**

Operation Source1,Source2,Destination

```
Add  A,B,C
```

- **Dvoadresne instrukcije**

Operation Source,Destination

```
Move  B,C
Add   A,C
```

- **Jednoadresne instrukcije**

Implicitno se koristi akumulator

```
Load  A
Add   B
Store C
```


Osnovni tipovi instrukcija

- Vrlo često, procesori su dizajnirani tako da se aritmetičke instrukcije izvršavaju samo nad podacima koji su već smešteni u registar:

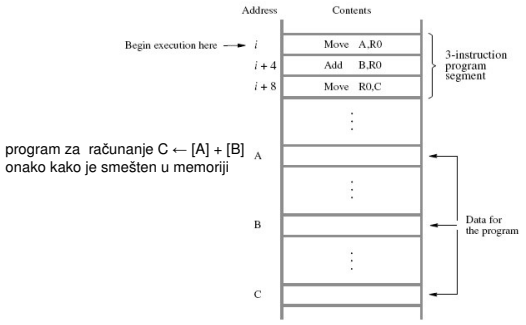
```
Move  A,Ri
Move  B,Rj
Add   Ri,Rj
Move  Rj,C
```

- Ako procesor podržava instrukcije sabiranja sa memorijskim operandom, tada se sabiranje može realizovati i u manjem broju instrukcija:

```
Move  A,Ri
Add   B,Ri
Move  Ri,C
```

- Moguće je koristiti i instrukcije kod kojih se svi operandi definišu implicitno, u strukturi **potisni stek**. Takve instrukcije su nula adresne.

Sekvencijalni program



program za računanje C ← [A] + [B] onako kako je smešten u memoriji

Sekvencijalni program

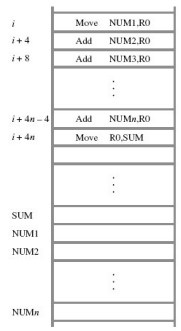
Izvršavanje ovakvog programa se naziva linearno sekvenciranje: registar PC redom uzima vrednosti i, i+4, i+8, ...

Izvršavanje konkretne instrukcije je procedura koja se odvija u dve faze:
•Dohvatanje instrukcije: u registar IR se smešta instrukcija koja je dohvaćena sa memorijske lokacije čiju adresu sadrži registar PC.
•Izvršenje instrukcije, gde se ispituje sadržaj IR da bi se odredilo koja će se instrukcija izvršavati. Potom se ta instrukcija izvrši od strane procesora, što često uključuje dohvatanje operanda iz memorije ili iz registara, izvršavanje aritmetičke ili logičke operacije i smeštanje rezultata u odredište.

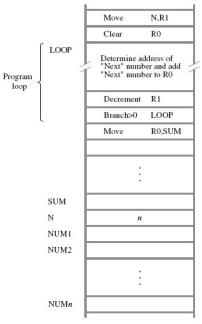
U nekom trenutku rada ove dvofazne procedure, sadržaj registra PC se uveća, kako bi on pokazivao na sledeću instrukciju. Tako, kad se završi faza izvršenja, PC sadrži adresu nove instrukcije i ponovo počinje faza dohvatanja instrukcije za tu novu instrukciju.

Kod najvećeg broja procesora, faza izvršavanja je podeljena u podfaze koje odgovaraju dohvatanju operanada, izvršenju operacija i smeštanju rezultata.

Program sa grananjem



Sekvencijalni program za sabiranje n brojeva



Petlja za sabiranje n brojeva

Program sa grananjem

Instrukcija grananja (ili instrukcija skoka) smešta novu vrednost u registar PC. Kao rezultat toga, procesor dohvata i izvršava instrukciju sa te nove adrese, umesto da izvrši instrukciju koja neposredno sledi iza instrukcije skoka. Uslovno grananje (odnosno uslovni skok) dovodi do skoka samo ako je zadovoljen neki konkretan uslov. Ako uslov nije zadovoljen, sadržaj registra PC se inkrementira na uobičajen način, pa se dohvata i izvršava sledeća instrukcija u sekvencijalnom rasporedu.

U prošlom primeru, instrukcija

Branch>0 LOOP

Predstavlja instrukciju uslovnog skoka, gde će skok na poziciju LOOP biti realizovan, ako je pri izvršenju operacija koja neposredno prethodi uslovnom skoku dobijeni rezultat bio veći od 0.

Mogućnost testiranja uslova i izbora alternativa za nastavak izračunavanja se veoma široko primenjuje. Ona postoji kod skupa instrukcija svih procesora i fundamentalna je za sva netrivijalna izračunavanja.

Kodovi za uslove

Procesor čuva informacije o rezultatima različitih operacija.
flegovi (elementi statusne reči):

N (negative)	Set to 1 if the result is negative; otherwise, cleared to 0
Z (zero)	Set to 1 if the result is 0; otherwise, cleared to 0
V (overflow)	Set to 1 if arithmetic overflow occurs; otherwise, cleared to 0
C (carry)	Set to 1 if a carry-out results from the operation; otherwise, cleared to 0

Generisanje memorijske adrese

Adresni modovi:

Name	Assembler syntax	Addressing function
Immediate	#Value	Operand = Value
Register	R*i*	EA = R*i*
Absolute (Direct)	LOC	EA = LOC
Indirect	(R*i*) (LOC)	EA = [R*i*] EA = [LOC]
Index	X(R*i*)	EA = [R*i*] + X
Base with index	(R*i*,R*j*)	EA = [R*i*] + [R*j*]
Base with index and offset	X(R*i*,R*j*)	EA = [R*i*] + [R*j*] + X
Relative	X(PC)	EA = [PC] + X
Autoincrement	(R*i*)+	EA = [R*i*]; Increment R*i*
Autodecrement	–(R*i*)	Decrement R*i*; EA = [R*i*]

EA = effective address
Value = a signed number

Implementacija promenljivih i konstanti

• Registarski mod – operand je sadržaj registra. Instrukcija sadrži naziv registra za specifikaciju operanda

• Apsolutni mod – operand je memorijska lokacija. Adresa lokacije je data eksplicitno instrukcijom. Apsolutni mod se kod nekih procesora naziva direktni mod.

• Neposredni mod – operator je direktno enkodiran u instrukciju.

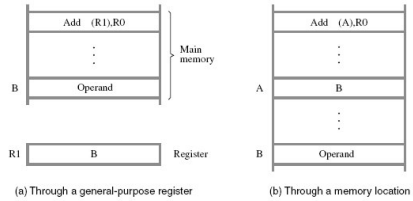
Apsolutni mod može da služi za predstavljanje globalnih promenljivih u programu.

A = B + 6

```
Move  B,R1
Add   #6,R1
Move  R1,A
```

Indirekcija i pokazivači

Indirektno adresiranje – efektivna adresa je sadržaj registra ili memorijske lokacije čija se adresa pojavljuje u instrukciji.



Indirektno adresiranje

Indirekcija i pokazivači

Address	Contents		
	Move	N,R1	Initialization
	Move	#NUM1,R2	
	Clear	R0	
LOOP	Add	(R2),R0	
	Add	#4,R2	
	Decrement	R1	
	Branch>0	LOOP	
	Move	R0,SUM	

Korišćenje indirektnog adresiranja u prethodnom primeru

Registar ili memorijska adresa koja sadrži adresu operanda se naziva **pokazivač**.

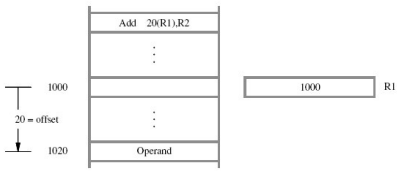
$A = *B;$

```
Move  B,R1
Move  (R1),A
```

```
Move  (B),A
```
(retko)

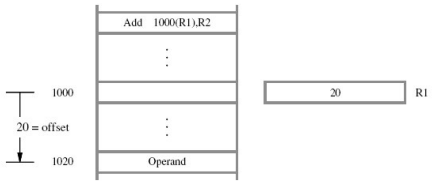
Indeksiranje i nizovi

Indeksno adresiranje – efektivna adresa se generiše dodavanjem konstantne vrednosti na vrednost registra.



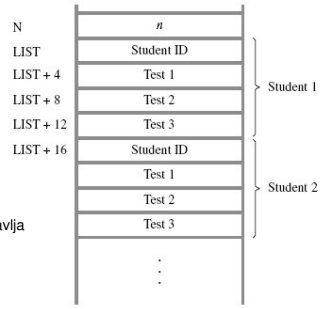
Korišćenje indeksnog adresiranja gde je ofset niza dat kao konstanta

Indeksiranje i nizovi



Korišćenje indeksnog adresiranja gde je ofset niza dat kao indeksni registar

Indeksiranje i nizovi



Struktura podataka koja predstavlja Spisak ocena za studenta

Indeksiranje i nizovi

	Move	#LIST,R0
	Clear	R1
	Clear	R2
	Clear	R3
	Move	N,R4
LOOP	Add	4(R0),R1
	Add	8(R0),R2
	Add	12(R0),R3
	Add	#16,R0
	Decrement	R4
	Branch>0	LOOP
	Move	R1,SUM1
	Move	R2,SUM2
	Move	R3,SUM3

Indeksirano adresiranje koje se koristi za pristup ocenama studenta koje se nalaze u prethodno opisanoj strukturi podataka

Relativno adresiranje

Kod relativnog adresiranja se suštinski radi o indeksnom adresiranju gde je umesto opšteg registra korišćen registar brojač naredbi PC.

U prethodnom slučaju, na primer, odredište skoka kod

Branch>0 LOOP

iznosi

-16(PC)

Dodatni adresni modovi

Autoinkrement mod – efektivna adresa operanda je sadržaj registra koji je specificiran tom funkcijom. Po pristupu operandu, sadržaj registra se automatski inkrementira tako da pokazuje na sledeći elemenat u listi.

Uobičajeni zapis autoinkrement moda je

(Ri)+

Autodekrement mod – sadržaj registra se automatski dekrementira tako da pokazuje na prethodni elemenat u listi, a efektivna adresa operanda je sadržaj tako promenjenog registra.

Uobičajeni zapis autodekrement moda je

-(Ri)

Dodatni adresni modovi

	Move	N,R1	Initialization
	Move	#NUM1,R2	
	Clear	R0	
LOOP	Add	(R2)+,R0	
	Decrement	R1	
	Branch>0	LOOP	
	Move	R0,SUM	

Program za računanje sume brojeva koji koristi autoinkrement mod

Asemblerski jezik

- Mnemonik
- Asemblerski jezik
- Sintaksa asemblerskog jezika
- Asembler, izvorni program, objekt program
- OP kod
- Asemblerske instrukcije

MOVE R0,SUM

ADD #5,R3
ADDI 5,R3

MOVE #5,(R2)
MOVEI 5,(R2)

Asemblerske direktive

Sadržaj memorije za prethodno opisani program

	100	Move	N,R1
	104	Move	#NUM1,R2
	108	Clear	R0
LOOP	112	Add	(R2),R0
	116	Add	#4,R2
	120	Decrement	R1
	124	Branch>0	LOOP
	128	Move	R0,SUM
	132		
		⋮	
SUM	200		
N	204	100	
NUM1	208		
NUM2	212		
		⋮	
NUM*n*	604		

Asemblerske direktive

Reprezentacija prethodnog programa u asemblerskom jeziku

	Memory address label	Operation	Addressing or data information
Assembler directives	SUM	EQU	200
		ORIGIN	204
	N	DATAWORD	100
	NUM1	RESERVE	400
		ORIGIN	100
Statements that generate machine instructions	START	MOVE	N,R1
		MOVE	#NUM1,R2
		CLR	R0
	LOOP	ADD	(R2),R0
		ADD	#4,R2
		DEC	R1
		BGTZ	LOOP
		MOVE	R0,SUM
Assembler directives		RETURN	
		END	START

Sintaksa asemblerske instrukcije: Label Operation Operand(s) Comment

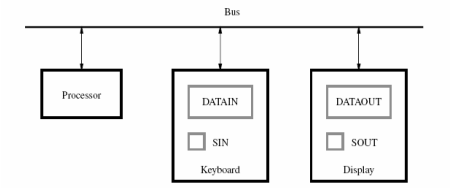
Asemblirani program i izvršavanje

- Način rada asemblera
- Dvoprolazni asembleri
- Punilac
- Dibager

Zapis brojeva

```
ADD   #93,R1        ADD   #%01011101,R1        ADD   #$5D,R1
```


Osnovne U/I operacije



Magistrala za procesor, tastaturu i monitor

Osnovne U/I operacije

```
READWAIT   Branch to READWAIT if SIN = 0
           Input from DATAIN to R1

WRITEWAIT  Branch to WRITEWAIT if SOUT = 0
           Output from R1 to DATAOUT
```

Sekvence operacija za prenos podataka sa ulaza i na izlaz

```
READWAIT   Testbit     #3,INSTATUS
           Branch=0    READWAIT
           MoveByte    DATAIN,R1

WRITEWAIT  Testbit     #3,OUTSTATUS
           Branch=0    WRITEWAIT
           MoveByte    R1,DATAOUT
```

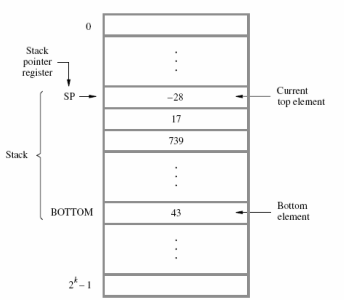
Sekvence operacija za prenos podataka sa ulaza i na izlaz
(kod memorijski mapiranih uređaja)

Osnovne U/I operacije

```
          Move       #LOC,R0          Initialize pointer register R0 to point to the
                                        address of the first location in memory
                                        where the characters are to be stored.
READ      TestBit    #3,INSTATUS      Wait for a character to be entered
          Branch=0   READ             in the keyboard buffer DATAIN.
          MoveByte   DATAIN,(R0)      Transfer the character from DATAIN into
                                        the memory (this clears SIN to 0).
ECHO      TestBit    #3,OUTSTATUS     Wait for the display to become ready.
          Branch=0   ECHO
          MoveByte   (R0),DATAOUT     Move the character just read to the display
                                        buffer register (this clears SOUT to 0).
          Compare    #CR,(R0)+        Check if the character just read is CR
                                        (carriage return). If it is not CR, then
          Branch≠0   READ             branch back and read another character.
                                        Also, increment the pointer to store the
                                        next character.
```

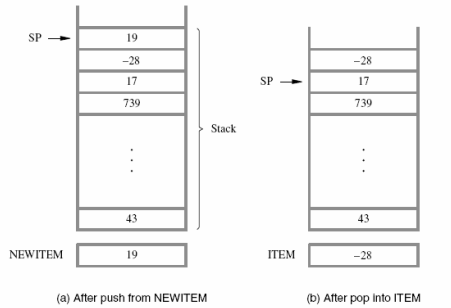
Program koji čita liniju sa ulaza i prikazuje je na izlaz

Stekovi i redovi



Stek reči u memoriji

Stekovi i redovi



Efekat operacija sa stekom

Stekovi i redovi

```
Subtract  #4,SP          Move  (SP),ITEM
Move      NEWITEM,(SP)   Add   #4,SP
```

push i pop operacije

```
Move  NEWITEM,−(SP)      Move  (SP)+,ITEM
```

Ovako realizovane operacije ne proveravaju da li je prilikom guranja na stek isti prepunjen, odnosno da li prilikom skidanja elemenata sa steka stek već prazan

Stekovi i redovi

```
SAFEPOP   Compare   #2000,SP      Check to see if the stack pointer contains
          Branch>0  EMPTYERROR    an address value greater than 2000. If it
                                  does, the stack is empty. Branch to the
                                  routine EMPTYERROR for appropriate
                                  action.
          Move      (SP)+,ITEM    Otherwise, pop the top of the stack into
                                  memory location ITEM.
```

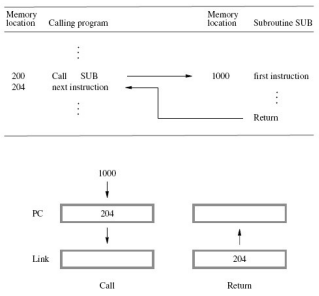
(a) Routine for a safe pop operation

```
SAFEPUSH  Compare   #1500,SP      Check to see if the stack pointer
          Branch≤0  FULLERROR     contains an address value equal
                                  to or less than 1500. If it does, the
                                  stack is full. Branch to the routine
                                  FULLERROR for appropriate action.
          Move      NEWITEM,−(SP) Otherwise, push the element in memory
                                  location NEWITEM onto the stack.
```

(b) Routine for a safe push operation

push i pop operacije sa proverom

Potprogrami



prelazak i povratak preko link registra

Potprogrami

Prenos parametara po vrednosti
Prenos parametara po referenci

Calling program

```
          Move      N,R1          R1 serves as a counter.
          Move      #NUM1,R2      R2 points to the list.
          Call      LISTADD       Call subroutine.
          Move      R0,SUM        Save result.
          ⋮
```

Subroutine

```
LISTADD   Clear     R0            Initialize sum to 0.
LOOP      Add       (R2)+,R0      Add entry from list.
          Decrement R1
          Branch>0  LOOP
          Return                  Return to calling program.
```

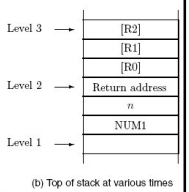
Prethodni potprogram ispisan u asemblerskom jeziku
Parametri su preneseni preko registra

Potprogrami

Assume top of stack is at level 1 below.

```
          Move      #NUM1,−(SP)   Push parameters onto stack.
          Move      N,−(SP)
          Call      LISTADD       Call subroutine
                                  (top of stack at level 2).
          Move      4(SP),SUM     Save result.
          Add       #8,SP         Restore top of stack
                                  (top of stack at level 1).
          ⋮
LISTADD   MoveMultiple R0−R2,−(SP) Save registers
                                  (top of stack at level 3).
          Move      16(SP),R1     Initialize counter to n.
          Move      20(SP),R2     Initialize pointer to the list.
          Clear     R0            Initialize sum to 0.
LOOP      Add       (R2)+,R0      Add entry from list.
          Decrement R1
          Branch>0  LOOP
          Move      R0,20(SP)     Put result on the stack.
          MoveMultiple (SP)+,R0−R2 Restore registers.
          Return                  Return to calling program.
```

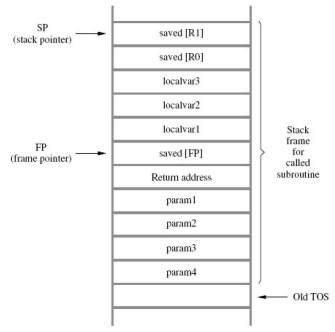
(a) Calling program and subroutine



(b) Top of stack at various times

Prethodni potprogram ispisan u asemblerskom jeziku
Parametri su preneseni preko steka

Potprogrami



Primer stek-okvira
(aktivacijskog sloga)

Memory location		Instructions	Comments
Main program			
	⋮		
2000	Move	PARAM2,−(SP)	Place parameters on stack.
2004	Move	PARAM1,−(SP)	
2008	Call	SUB1	
2012	Move	(SP),RESULT	Store result.
2016	Add	#8,SP	Restore stack level.
2020	next instruction		
	⋮		

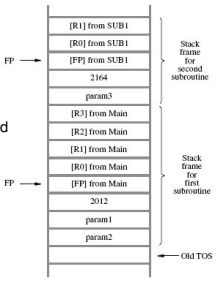
First subroutine

2100	SUB1	Move	FP,−(SP)	Save frame pointer register.
2104		Move	SP,FP	Load the frame pointer.
2108		MoveMultiple	R0−R3,−(SP)	Save registers.
2112		Move	8(FP),R0	Get first parameter.
		Move	12(FP),R1	Get second parameter.
		⋮		
		Move	PARAM3,−(SP)	Place a parameter on stack.
2160		Call	SUB2	
2164		Move	(SP)+,R2	Pop SUB2 result into R2.
		⋮		
		Move	R3,8(FP)	Place answer on stack.
		MoveMultiple	(SP)+,R0−R3	Restore registers.
		Move	(SP)+,FP	Restore frame pointer register.
		Return		Return to Main program.

Second subroutine

3000	SUB2	Move	FP,−(SP)	Save frame pointer register.
		Move	SP,FP	Load the frame pointer.
		MoveMultiple	R0−R1,−(SP)	Save registers R0 and R1.
		Move	8(FP),R0	Get the parameter.
		⋮		
		Move	R1,8(FP)	Place SUB2 result on stack.
		MoveMultiple	(SP)+,R0−R1	Restore registers R0 and R1.
		Move	(SP)+,FP	Restore frame pointer register.
		Return		Return to Subroutine 1.

Stek okvir za prethodni kod



Dodatne instrukcije

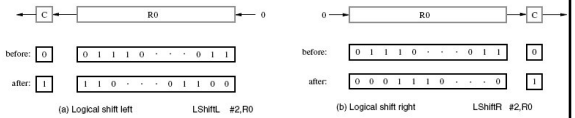
I kod dosad pređenih instrukcija: Move, Load, Store, Clear, Add, Subtract, Increment, Testbit, Compare, Call i Return ima redundansi, ali to je uobičajeno za praktične skupove instrukcija procesora – ista funkcionalnost se može postići na više različitih načina.

Logičke instrukcije

Logičke instrukcije su naročito korisne za izvršavanje logičkih operacija u softveru.

```
Not   dst          Not   R0          Negate  R0        And      #$FF000000,R0
                   Add   #1,R0                           Compare  #$5A000000,R0
                                                         Branch=0 YES
```


Instrukcije pomeranja i rotacije



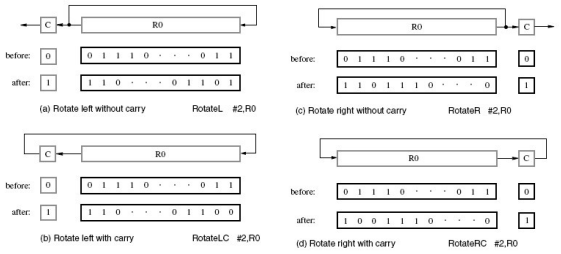
Aritmetičko pomeranje
Logičko pomeranje

Instrukcije pomeranja i rotacije

Move	#LOC,R0	R0 points to data.
MoveByte	(R0)+,R1	Load first byte into R1.
LShiftL	#4,R1	Shift left by 4 bit positions.
MoveByte	(R0),R2	Load second byte into R2.
And	#$F,R2	Eliminate high-order bits.
Or	R1,R2	Concatenate the BCD digits.
MoveByte	R2,PACKED	Store the result.

Figure 2.31 A routine that packs two BCD digits.

Instrukcije pomeranja i rotacije



Instrukcije množenja i deljenja

Ovde se, naravno, radi o celobrojnim operacijama. Format je isti kao kod instrukcija sabiranja:

Multiply Ri,Rj Divide Ri,Rj

$Rj \leftarrow [Ri] \times [Rj]$ $Rj \leftarrow [Rj]/[Ri]$

Proizvod dva broja dužine n bitova ne mora nužno da stane u n bitova. Stoga, mnogi procesori za smeštaj rezultata pri izvršenju instrukcije množenja koriste dva (obično susedna ili na neki drugi način predefinisana registra) – u jednom smeste nižu reč, a u drugom višu reč rezultata.
Veoma često, uporedo sa računanjem količnika dve vrednosti, procesor računa i ostatak pri deljenju - koji smešta u susedni ili nekeko drugačije predefinisani) registar.

Primeri programa

$$\text{Dot Product} = \sum_{i=0}^{n-1} A(i) \times B(i)$$

	Move	#AVEC,R1	R1 points to vector A.
	Move	#BVEC,R2	R2 points to vector B.
	Move	N,R3	R3 serves as a counter.
	Clear	R0	R0 accumulates the dot product.
LOOP	Move	(R1)+,R4	Compute the product of
	Multiply	(R2)+,R4	next components.
	Add	R4,R0	Add to previous sum.
	Decrement	R3	Decrement the counter.
	Branch>0	LOOP	Loop again if not done.
	Move	R0,DOTPROD	Store dot product in memory.

Primeri programa

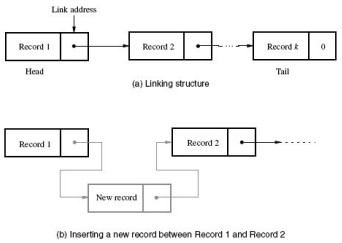
```
for  (j = n−1; j > 0; j = j − 1)
     { for ( k = j−1; k >= 0; k = k − 1 )
          { if   (LIST[k] > LIST[j])
               {  TEMP = LIST[k];
                  LIST[k] = LIST[j];
                  LIST[j] = TEMP;
               }
          }
     }
```

(a) C-language program for sorting

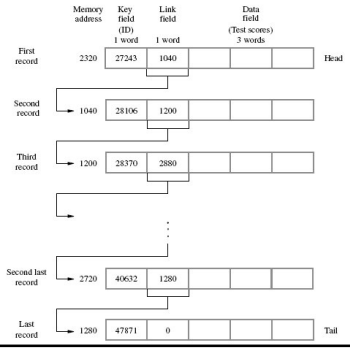
	Move	#LIST,R0	Load LIST into base register R0.
	Move	N,R1	Initialize outer loop index
	Subtract	#1,R1	register R1 to $j = n - 1$.
OUTER	Move	R1,R2	Initialize inner loop index
	Subtract	#1,R1	register R2 to $k = j - 1$.
	MoveByte	(R0,R1),R3	Load LIST(j) into R3, which holds current maximum in sublist.
INNER	CompareByte	R3,(R0,R2)	If LIST(k) $\leq$ [R3],
	Branch≤0	NEXT	do not exchange.
	MoveByte	(R0,R2),R4	Otherwise, exchange LIST(k)
	MoveByte	R3,(R0,R2)	with LIST(j) and load
	MoveByte	R4,(R0,R1)	new maximum into R3.
	MoveByte	R4,R3	Register R4 serves as TEMP.
NEXT	Decrement	R2	Decrement index registers R2 and
	Branch≥0	INNER	R1, which also serve as
	Decrement	R1	as loop counters, and branch
	Branch>0	OUTER	back if loops not finished.

(b) Assembly language program for sorting

Primeri programa



Primeri programa



Primeri programa

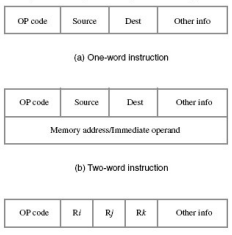
INSERTION	Compare	#0, RHEAD	
	Branch>0	HEAD	
not empty	Move	RNEWREC, RHEAD	new record becomes a one-entry list
	Return		
HEAD	Compare	(RHEAD), (RNEWREC)	
	Branch>0	SEARCH	
insert new record somewhere after current head	Move	RHEAD, 4(RNEWREC)	new record becomes new head
	Move	RNEWREC, RHEAD	
	Return		
SEARCH	Move	RHEAD, RCURRENT	
LOOP	Move	4(RCURRENT), RNEXT	
	Compare	#0, RNEXT	
	Branch=0	TAIL	
new record becomes new tail	Compare	(RNEXT), (RNEWREC)	
	Branch<0	INSERT	
insert new record in an interior position	Move	RNEXT, RCURRENT	
	Branch	LOOP	
INSERT	Move	RNEXT, 4(RNEWREC)	
TAIL	Move	RNEWREC, 4(RCURRENT)	
	Return		

Primeri programa

DELETION	Compare	(RHEAD), RIDNUM
	Branch>0	SEARCH
not the head record	Move	4(RHEAD), RHEAD
	Return	
SEARCH	Move	RHEAD, RCURRENT
LOOP	Move	4(RCURRENT), RNEXT
	Compare	(RNEXT), RIDNUM
	Branch=0	DELETE
	Move	RNEXT, RCURRENT
	Branch	LOOP
DELETE	Move	4(RNEXT), RTEMP
	Move	RTEMP, 4(RCURRENT)
	Return	

Enkodiranje mašinskih instrukcija

Instrukcije enkodirane prema kompaktnom binarnom obrascu nazivaćemo mašinskim instrukcijama, dok će instrukcije zapisane u simboličkom jeziku i korišćenejm skraćenica biti nazivane asemblerskim instrukcijama



Enkodiranje mašinskih instrukcija

Add R1, R2

Move 24(R0), R5 Mov R2, MEMLOC Add R1, R2, R3

LshiftR #2, R0 And #$FF0000000, R2

Move #$3A, R1

Branch >0 LOOP

• Jedan pristup dovodi do instrukcija promenljive dužine, koja zavisi od broja operanada i tipova adresnih modova. Korišćenjem većeg broja reči implementiraju se veoma kompleksne instrukcije, koje se približavaju operracijama viših programskih jezika. Procesori sa skupom instrukcija te vrste su CISC procesori.
• Alternativni pristup zahteva da sve instrukcije budu iste dužine, što dovodi do visoko funkcionalnog skupa instrukcija koji efikasno koristi procesorske resurse. Pored ograničenja dužine reči, ovaj pristup sadrži i druga ograničenja, na primer da se sva maipulacija nad podacima mora izvršiti nad operandima koji su u registrima. Procesori sa takvim skupom instrukcija su RISC procesori.

Zadaci

•
•
•