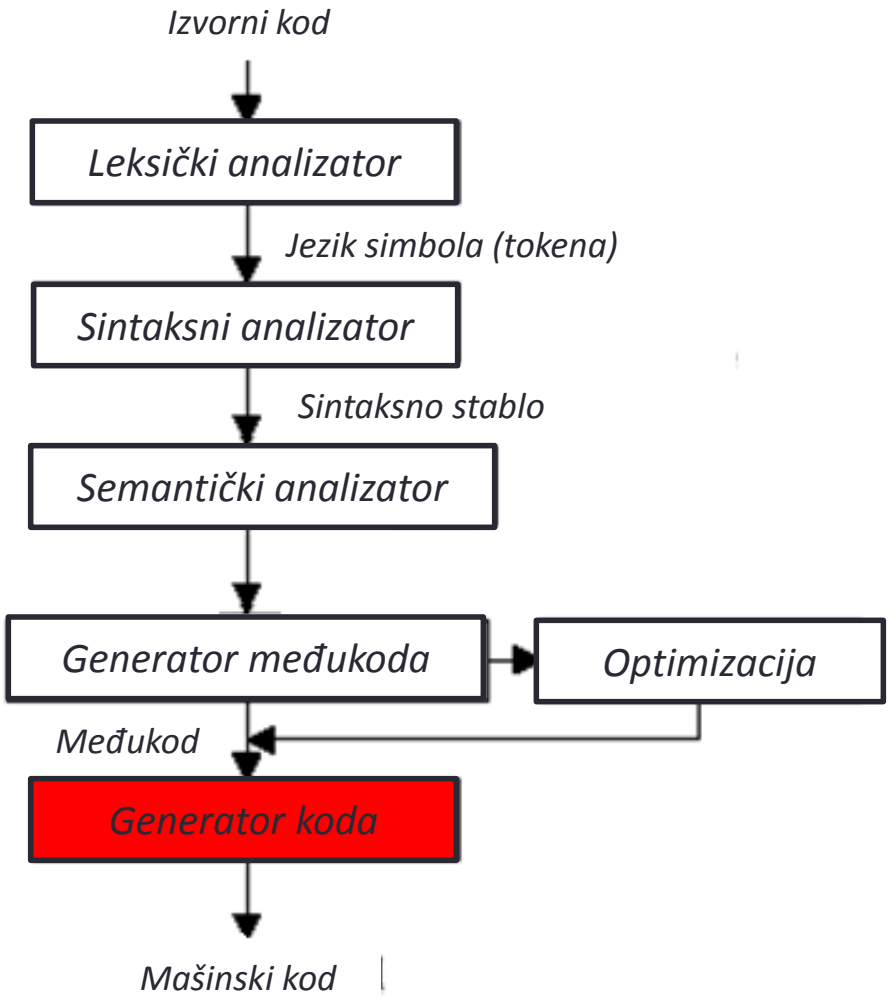
PROGRAMSI PREVODIOCI
- Generisanje izlaznog koda -

Struktura kompilatora



Preduslov za razvoj generatora koda

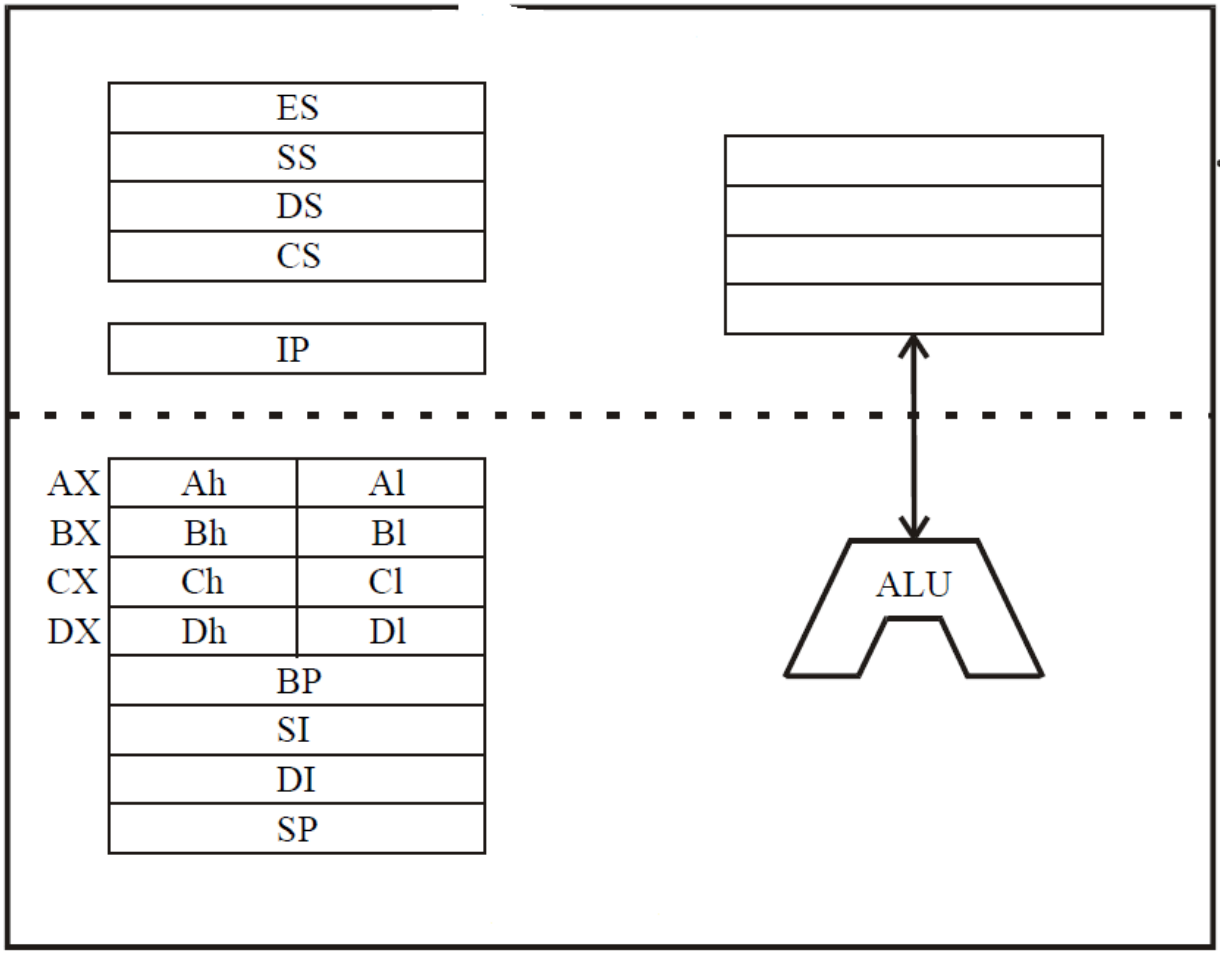
- Poznavanje odredišne mašine za koju se kod generiše
  - Asemblerski jezik,
  - Arhitektura procesora (skup registara i njihove namene)

Zadaci generatora koda

- **Izbor strategije za alokaciju memorije (runtime memory management).**
- Definisati način preslikavanja naredbi međukoda u naredbe asemblerskog jezika.

Odredišna mašina – procesori iz familije 8086

- Arhitektura procesora



Skup instrukcija procesora 8086

Instrukcija za prenos podataka	
MOV *dst*, *src*	[*dst*] ← [*src*]
Aritmetičke instrukcije	
ADD *dst*, *src*	[*dst*] ← [*dst*] + [*src*]
SUB *dst*, *src*	[*dst*] ← [*dst*] - [*src*]
INC *src*	[*dst*] ← [*dst*] + 1
DEC *src*	[*dst*] ← [*dst*] - 1
MUL *src*	[DX : AX] ← [AX] * [src]
DIV *src*	[AX] ← [DX : AX] / [src], [DX] ← [DX : AX] mod [src]

Skup instrukcija procesora 8086

Instrukcija za rad sa stekom	
PUSH *src*	[*stek*[SP]] ← [*src*], [SP] ← [SP] – 1
POP *dst*	[*dst*] ← [*stek*[SP]], [SP] ← [SP] – 1
Instrukcije za promenu toka izvršenja programa	
JMP *adr*	[IP] ← [*adr*]
J*c adr*	if *c* then [IP] ← [*adr*]
CALL *adr*	PUSH IP, [IP] ← [*adr*]
RET	POP IP

Načini adresiranja kod procesora 8086

Način adresiranja	Format
Neposredno	CONST
Direktno memorijsko	NAME, [CONST]
Direktno registarsko	*reg*
Indirektno registarsko	[*reg*]
Bazno	[*reg*+CONST]
Indeksno	NAME[*reg*], CONST[*reg*]

Upravljanje memorijom

- Zadaci:
  - Organizacija memorije u toku izvršenja programa:
    - odredjivanje gde će se u memoriji smestiti kod programa,
    - kako će se memorisati podaci različitih tipova,
    - gde će se u memoriji smestiti podaci različitih klasa memorije
  - Pristup podacima smeštenim u različitim zonama memorije
  - Promena sadržaja memorije u trenutku poziva potprograma i povratka na pozivajući modul
    - Prevođenje naredbe poziva i naredbe povratka

Organizacija memorije

- U toku izvršenja programa u memoriji se čuva:
  - Kod programa,
  - Podaci koji se u programu obradjuju
  - Podaci o pozivima potprogramima – aktivacioni slogovi potprograma

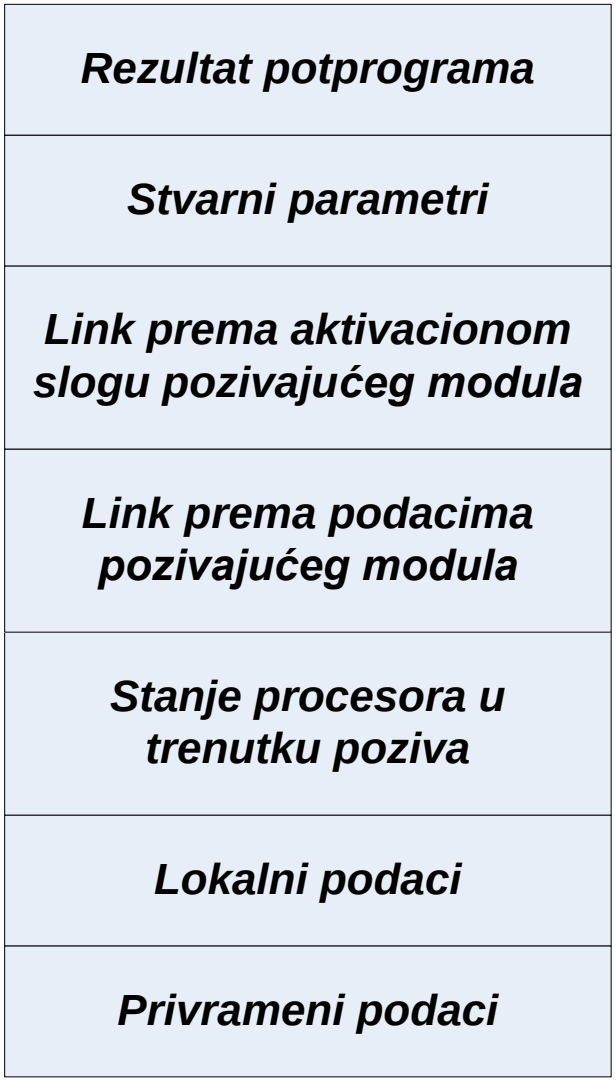
Strategije upravljanja memorijom

- **Staička alokacija memorije**
  - Sadržaj memorije se ne menja u toku izvršenja programa – i kod programa i svi podaci su poznati u fazi prevodjenja
    - Za svaki potprogram postoji samo jedan aktivacioni slog što znači da ova strategija ne podržava rekurzivne potprograme
- **Dinamička alokacija memorije**
  - Sadržaj memorije se stalno menja u toku izvršenja programa
    - mogu se kreirati novi podaci, brisati
    - mogu se kreirati novi delovi koda, brisati
    - mogu se kreirati novi aktivacioni slogovi, brisati
  - Povremeno se akriviraju posebni pomoćni programi (***garbage collectori***) koji vode računa o korišćenju memorijskog prostora
    - kad se uoče delovi memorije na koje ne ukazuje ni jedna referenca, oni se uklanjaju iz memorije, tj. taj deo memorije se oslobađa

Strategije upravljanja memorijom

- **Polu-statička ili stek alokacija memorije**
  - Operativna memorija se deli na 2 zone:
    - Statičku – čuva kod programa i globalne podatke
    - Dinamičku koja se deli na:
      - Stack – čuva aktivacione slogove potprograma
      - Heap – čuva podatke koji se po potrebi kreiraju u toku izvršenja programa i uklanjaju kada više nisu potrebni

Struktura aktivacionog sloga

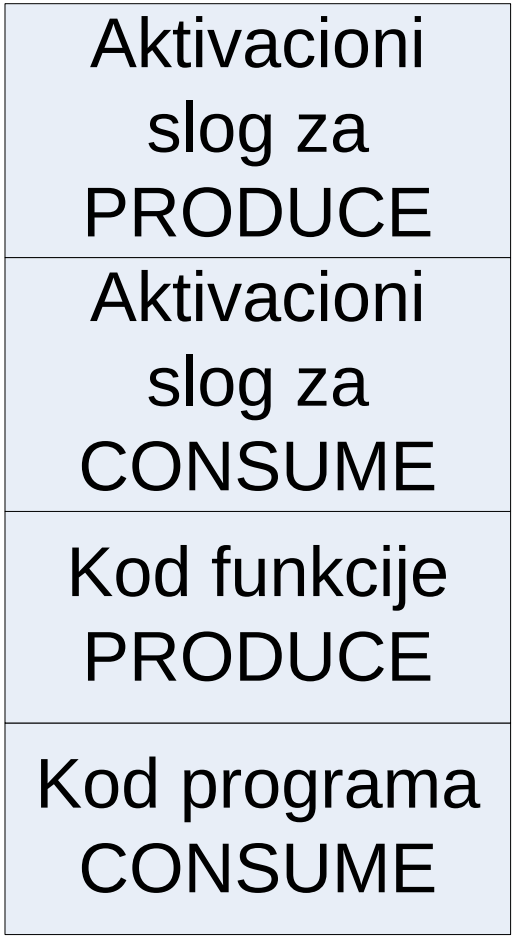


Primer statičke alokacije memorije

```
      PROGRAM CONSUME
            CHARACTER * 50 BAF
            INTEGER  NEXT
            CHARACTER C, PRODUCE
            DATA NEXT /1/, BUF /’   ‘/
6           C = PRODUCE ( )
            BUF(NEXT:NEXT) = C
            NEXT = NEXT + 1
            IF (C .NE. ‘ ‘) GO TO 6
            WRITE (*, ‘(A)’) BUF
            END
```

```
CHACTER FUNCTION PRODUCE ( )
CHARACTER * 80 BUFFER
INTEGER NEXT
SAVE BUFFER, NEXT
DATA NEXT /81/
IF (NEXT .GT. 80 ) THEN
READ (*, ‘(A)’) BUFFER
NEXT = 1
END IF
PRODUCE = BUFFER(NEXT:NEXT)
NEXT = NEXT + 1
END
```

Primer statičke alokacije memorije



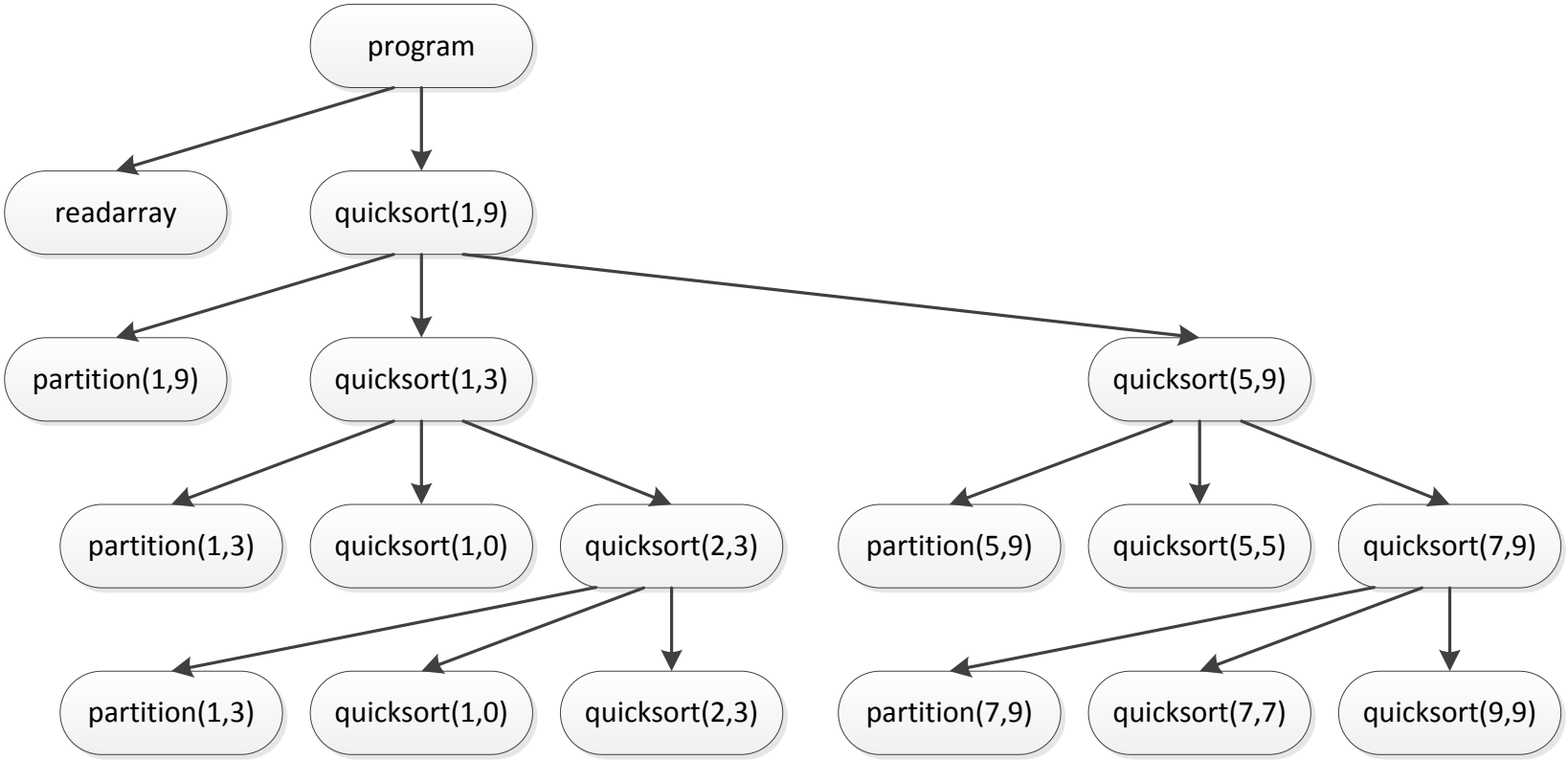
Primer za stek i dinamičku alokaciju memorije

```
int a[11];
void quiksort(int m, int n)
{ int i;
  if (n > m)
  {
    i = partition(m,n);
    quiksort(m, i - 1);
    quiksort(i + 1, n);
  }
}
```

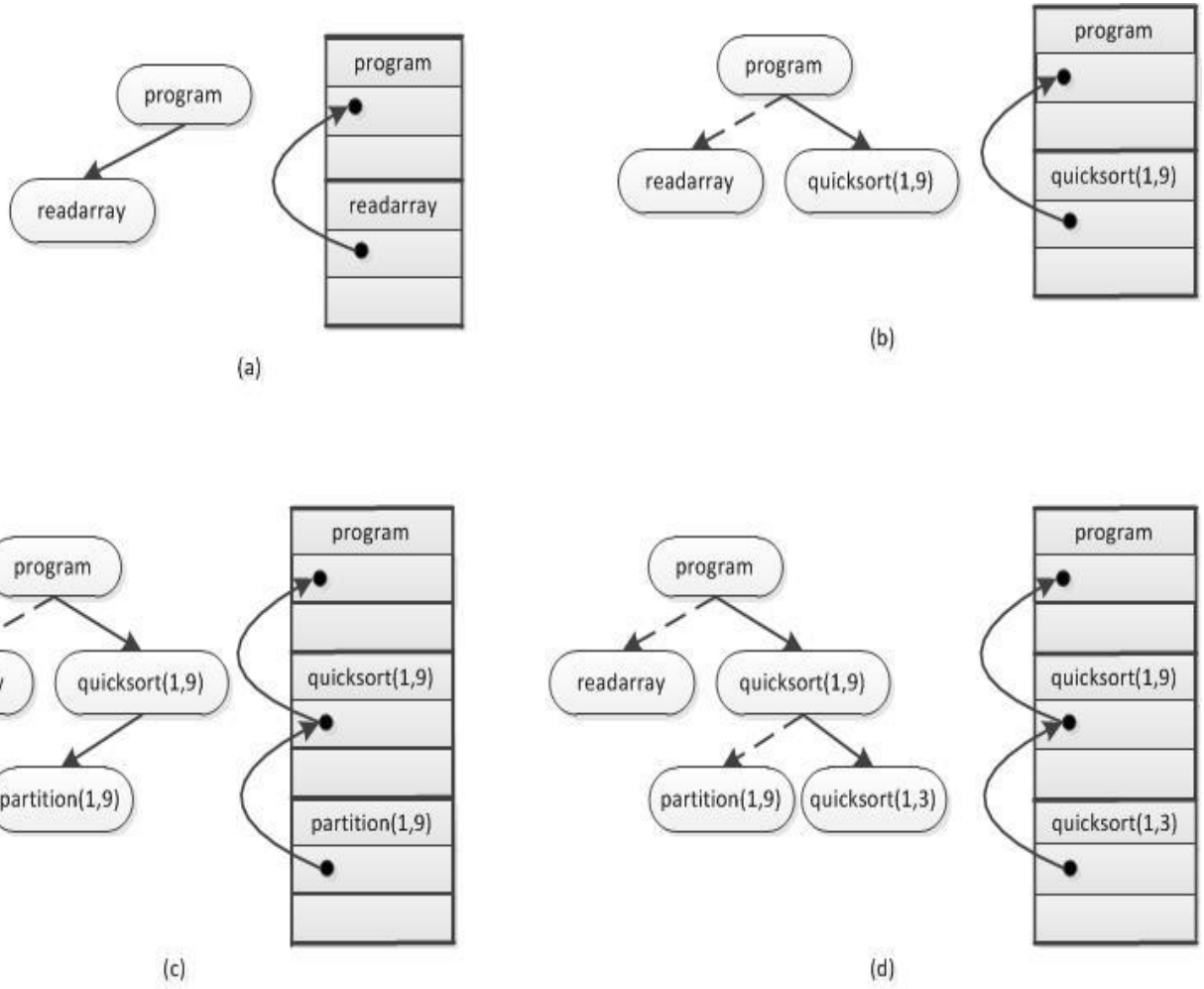
```
void main ()
{
  a[0] = -9999;
  a[10] = 9999;
  readarray();

  quiksort(1,9);
}
```

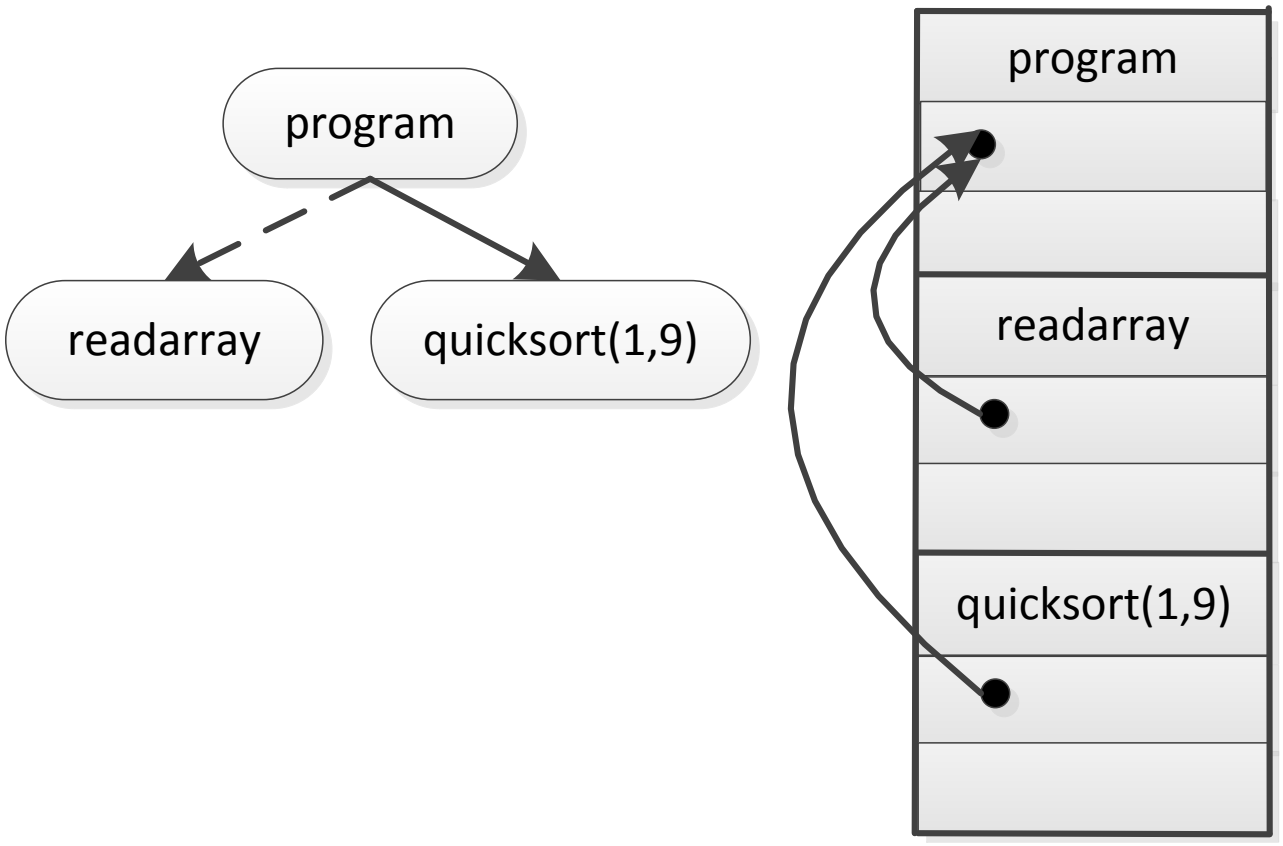
Aktivaciono stablo za program iz primera



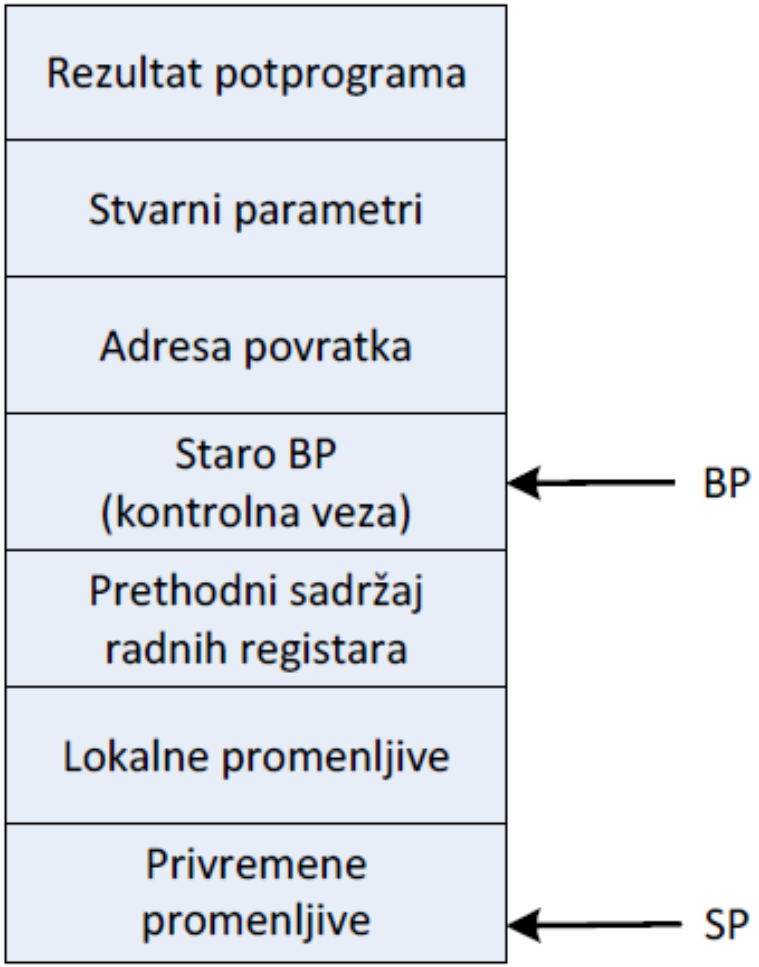
Promena sadržaja memorije kod stek alokacije



Promena sadržaja memorije kod dinamičke alokacije



Stek alokcija memorije kod procesora 8086



Slika 14.7 Aktivacioni slog koji se koristi u jeziku 8086.

Generisanje sekvence poziva

- Sekvenca poziva – skup instrukcija koje će se izvršiti u trenutku poziva potprograma
  - Deo instrukcija koje pripadaju pozivajućem modulu
    - Rezervacija prostora za rezultat potprograma

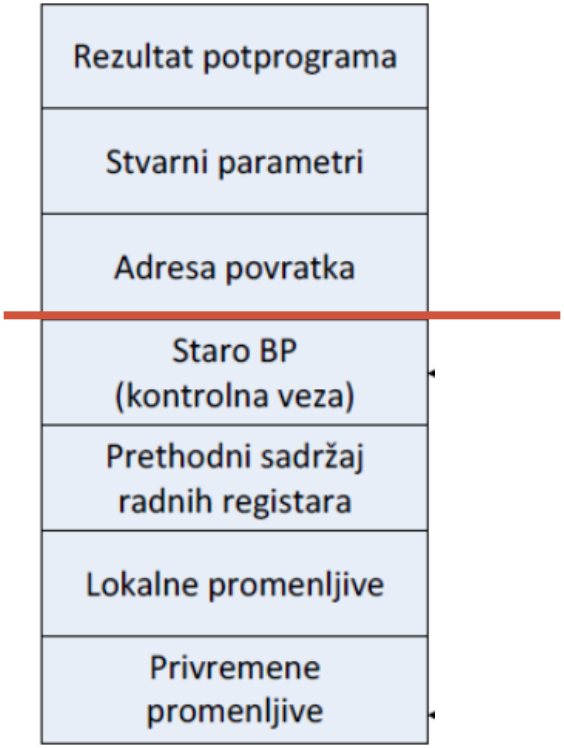
      ```
      SUB SP, size
      ```

    - Smeštanje u stek stvarnih parametara

      ```
      PUSH arg_n
      ...
      PUSH arg_1
      ```

    - Poziv potprograma

      ```
      CALL name
      ```



Generisanje sekvence poziva

- Deo instrukcija koje pripadaju pozvanom modulu
  - Smeštanje u stek prethodne vrednosti baznog pokazivača

    ```
    PUSH BP
    ```

  - Promena sadržaja baznog pokazivača

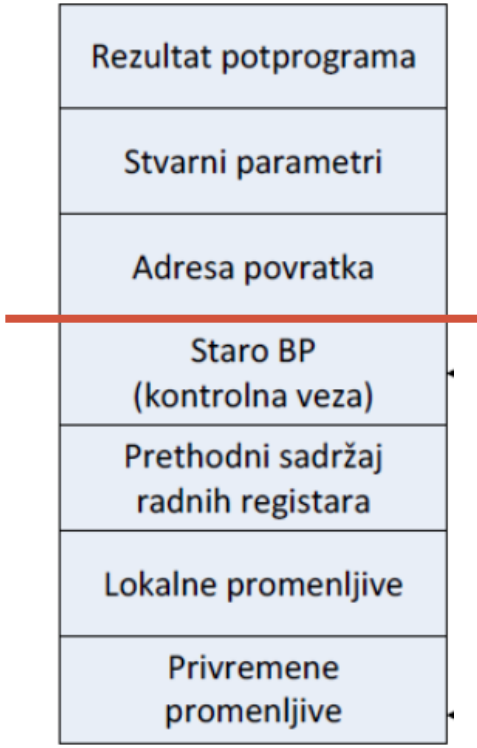
    ```
    MOV BP, SP
    ```

  - Smeštanje u stek konteksta procesora (sadržaja radnih registara)

    ```
    PUSH AX
    PUSH BX
    PUSH CX
    PUSH DX
    PUSH SI
    PUSH DI
    ```

  - Rezervacija prostora za lokalne promenljive

    ```
    SUB SP, size
    ```



Generisana sekvenca poziva za *quicksort(1,9)*

Pozivajući modul

```
MOVE AX,9
PUSH AX
MOVE AX,1
PUSH AX
CALL QUICKSORT
```

Pozvani podul

```
PUSH BP
MOVE BP, SP
PUSH AX
PUSH BX
PUSH CX
PUSH DX
PUSH SI
PUSH DI
SUB SP, 2
```

Generisanje sekvence povratka

- Sekvenca povratka – skup instrukcija koje će se izvršiti u trenutku povratka u pozivajući modul
  - Deo instrukcija koje pripadaju pozvanom modulu
    - Upis rezultata u predviđenu lokaciju

      ```
      MOV [BP+x], rez
      ```

    - Izbacivanje lokalnih promenljivih

      ```
      ADD SP, size
      ```

    - Vraćanje starog konteksta procesora

      ```
      POP DI
      POP SI
      POP DX
      POP CX
      POP BX
      POP AX
      ```

    - Preuzimanje stare vrednosti baznog pokazivača

      ```
      POP BP
      ```

    - Povratak u pozivajući modul

      ```
      RET
      ```

Generisanje sekvence povratka

- Deo instrukcija koje pripadaju pozivajućem modulu
  - Izbacivanje stvarnih parametara is steka

    ```
    ADD SP, size
    ```

  - Preuzivanje rezultata

    ```
    POP reg
    ```

Generisana sekvenca povratka za *quicksort(1,9)*

Pozvani podul

```
ADD SP, 2
POP DI
POP SI
POP DX
POP CX
POP BX
POP AX
POP BP
RET
```

Pozivajući podul

```
ADD SP, 4
```

Smeštanje podataka i pristup podacima

- Gde će podaci biti smešteni zavisi od toga gde su definisani i koju klasu memorije imaju.
- U programskom jeziku C:
  - Globalni podaci i lokalni podaci sa klasom memorije static
    - u statičkoj zoni memorije
    - pristup - direktno memorijsko adresiranje (NAME ili [CONST])
  - Parametri funkcija i lokalni podaci sa klasom memorije auto
    - u steku (u akrivacionom slogu)
    - pristup - bazno adresiranje ([BP+p])
  - Lokalni podaci sa klasom memorije register
    - u reistrima procesora
    - pristup - direktno registarsko adresiranje (DX)
  - Podaci u dinamičkoj zoni memorije
    - u heapu
    - pristup - indirektno registarsko adresiranje (adresa podatka u registru) ([BX])

Prevodjenje naredbi međukoda

Prevođenje naredbe oblika: x := y *op* z

Napomena: Za sve registre se uvode deskriptori koji kažu koji podatak oni trenutno sadrže

1. Odrediti lokaciju L gde će biti smešten rezultat x (Ako je L registar, postaviti odgovarajući deskriptor)
2. Ukoliko y nije u L generisati instrukciju. MOV L, y'
   - Ako je L registar, postaviti deskriptor tog registra na x, izbrisati x iz svih ostalih deskriptora
3. Generisati instrukciju: OP L, z'
4. Oslobađanje registara:
   - Ukolko su y i z u registrima i ne koriste se više u tekućem bloku, osloboditi registre (odgovarajuće deskriptore postaviti na „slobodan")

Određivanje lokacije rezultata (L)

1. Ako je y u registru, i y se više ne koristi u tekućem bloku, vratiti taj registar kao L.
2. Ako prva opcija nije moguća i postoji slobodan registar, vratiti slobodan registar kao L.
3. Ako operacija OP zahteva da prvi operand bude u nekom konkretnom registru, osloboditi taj registar i vratiti registar kao L.
4. Ako ni jedna od prethodnih opcija nije moguća. vratiti adresu promenljive X kao L.

Prevođenje naredbi dodele - primer

Naredba međukoda	Generisani kod	Deskriptor registra	Deskriptor adrese
		registri prazni	a, b, c i d u memoriji
$t_1 := a - b$	`MOV AX, a` `SUB AX, b'`	AX sadrži t_1	t_1 u AX a, b i c u memoriji
$t_2 := a - c$	`MOV BX, a` `SUB BX, c`	AX sadrži t_1 BX sadrži t_2	t_1 u AX a, b i c u memoriji t_2 u BX
$t_3 := t_1 + t_2$	`ADD AX, BX`	AX sadrži t_3 BX sadrži t_2	t_2 u BX a, b i c u memoriji t_3 u AX
$d := t_3 - t_2$	`ADD AX, BX`	AX sadrži d	d u AX a, b i c u memoriji
	`MOV d, AX`	registri prazni	a, b, c i d u memoriji

Generisanje koda za naredbe sa indeksiranim promenljivama

Naredba međukoda	Indeks i je u registru	Indeks i je u memoriji M_i	Indeks i je na steku sa pomerajem d_i u odnosu na BP

$a := b[i]$	`MOV AX, b[SI]`
	`MOV a, AX`
		`MOV SI, Mi`
		`MOV AX, b[SI]`
		`MOV a, AX`
			`MOV SI, [BP + di]`
			`MOV AX, b[SI]`
			`MOV a, AX`

$a[i] := b$	`MOV AX, b`
	`MOV a[DI], AX`
		`MOV AX, b`
		`MOV DI, Mi`
		`MOV a[DI], AX`
			`MOV AX, b`
			`MOV DI, [BP + di]`
			`MOV a[DI], AX`

Generisanje koda za naredbe sa pokazivačima

Naredba međukoda	Pokazivač p je u registru BX	Pokazivač p je u memoriji M_p	Pokazivač p je na steku sa pomerajem d_p u odnosu na BP

$a := *p$	`MOV AX, [BX]`
	`MOV a, AX`
		`MOV BX, Mp`
		`MOV AX, [BX]`
		`MOV a, AX`
			`MOV BX, [BP + dp]`
			`MOV AX, [BX]`
			`MOV a, AX`

$*p := a$	`MOV AX, a`
	`MOV [BX], AX`
		`MOV BX, Mp`
		`MOV AX, a`
		`MOV [BX], AX`
			`MOV BX, [BP + dp]`
			`MOV AX, a`
			`MOV [BX], AX`

Generisanje koda za naredbe uslovnog skoka

```
if x relop y goto z
```

Generisani kod:

```
CMP  'x, 'y
JNGE z
```