Primena programskog jezika C u sistemima za rad u realnom vremenu

Advanced C Programming Course

C for Real-Time Developers

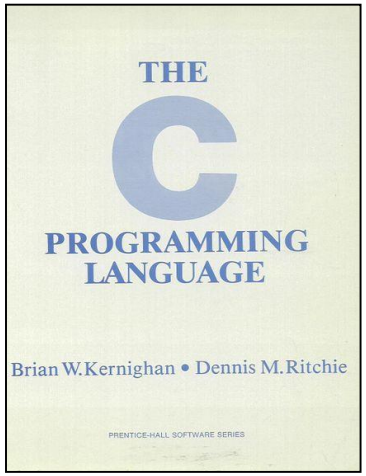
Osnovni cilj

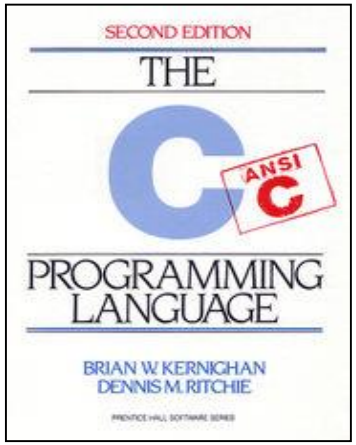
- Upoznavanje sa detaljima implementacije C jezika
  - “C ispod haube”
- Kurs “lepog” programiranja
  - pisanje pouzdane i razumljive programske podrške (PP), za
  - efikasan rad u realnom vremenu
- Primeri, vežbe i zadaci

Plan rada

- UVOD I TEORIJSKE OSNOVE
- TIPOVI C PODATAKA
  - Zavisnost od HW platforme (x86, MIPS), Big & Little Endian
- PROCESORSKI MODEL C PROGRAMA
  - Start-up kod (sprega sa OS, početna podešavanja)
- PROGRAMSKI MODULI i BIBLIOTEKE
- PREPORUKE ZA ORGANIZACIJU I PISANJE SIGURNIJEG KODA
  - Značaj dobre organizacije programa (struktuiranja podataka i koda)
  - Primena OOP principa u pisanju C koda
- MEMORIJSKI MODEL C PROGRAMA
  - Segmenti: kod, neinicijalizovani podaci, inicijalizovani podaci, stek, heap
  - Rad sa stekom i sa heap-om
- GENERISANJE KODA I POVEZIVANJE SA ASEMBLEROM

Programski jezik C





- Tip: proceduralni jezik
- Nivo: blizak asembleru, low-level pristup
- Namena: sistemska podrška, RT aplikacije
- Cilj: programiranje nezavisno od arhitekture
- Razvijen: 1972; Denis Riči, Bell Labs
- Standardi (**`__STDC_VERSION__`**)
  - K&R, 1978
  - **ANSI C89** - ANSI X3.159:1989
  - C99 - ISO/IEC 9899:1999
  - C11 - ISO/IEC 9899:2011
- Okruženja: GCC, MSVC, Borland, Watcom, ...
- HW platforme: brojne – svaki µProc
- Portabilnost:
  - teorijski neograničena
  - praktično – zavisi kako je napisan kod

Tematika kursa

- Nije sintaksa C jezika, kao i većina onog što se može naći u referentnim priručnicima
- Pokušaćemo da obradimo one elemente koji su najčešći izvor nerazumevanja i/ili grešaka
- Ukazaćemo na opšte principe razvoja programske podrške,
- Kao i na konkretna pravila koja pomažu u pisanju lepog C koda
- Šta je lepota koda:
  - Adekvatna organizacija funkcija, modula i biblioteka
  - Razumljivost koda, i na toj osnovi
  - Mogućnost kasnijeg održavanja razvijene PP
- Šta je nagrada za programera i ceo tim:
  - Mogućnost višestruke upotrebe (reusabilty)
  - Prenosivost na druge platforme (portability)
  - Uspeh u radu (bolja plata?)

Uzroci grešaka pri pisanju C programa

- Veličina ukupnog projekta i učešće većeg broja programera
- Loša metodologija rada + nedoslednost u njenoj primeni
- Preobimnost i nekonzistentnost korišćenih tipova podataka i globalnih promenljivih
- Uklapanje brojnih .h datoteka
- Neadekvatno korišćenje preprocesora (macro iskazi)
- Otklanjanje simptoma, a ne uzroka problema
- Nerazumljivost koda
- Nedostatak detaljnog testiranja, od nižih ka višim programskim nivoima

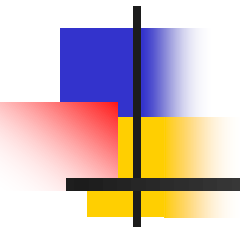
Način za prevazilaženje problema

- Usvajanje zajedničke metodologije i stila programiranja
- Upoznavanje ili obnavljanje znanja o principima arhitekture i organizacije računarskih sistema
- Dobro poznavanja konkretne platforme, HW i SW
- To je pogotovo važno za *real-time* sisteme
- Detaljno testiranje, pogotovo PP nižeg nivoa

- **I NAJVAŽNIJE**
- Primena jednostavnih rešenja
- Učenje na svojim i (ako može) tuđim greškama
- Uopštavanje stečenog programerskog iskustva

Literatura uz kurs (*additional readings*)

- C Predavanja
  - Materijali sa ovog kursa
- C Priručnici
  - K&R, Esential C, Tenouk kurs
- Lepo kodiranje
  - Elements of Style
  - BugFreeC, Practice of Programming K&P
- MIPS i Asembler
  - See MIPS Run, IA-32 Asembler
- Software
  - Primeri sa kursa



UVOD U C PROGRAMIRANJE

Struktura C programa

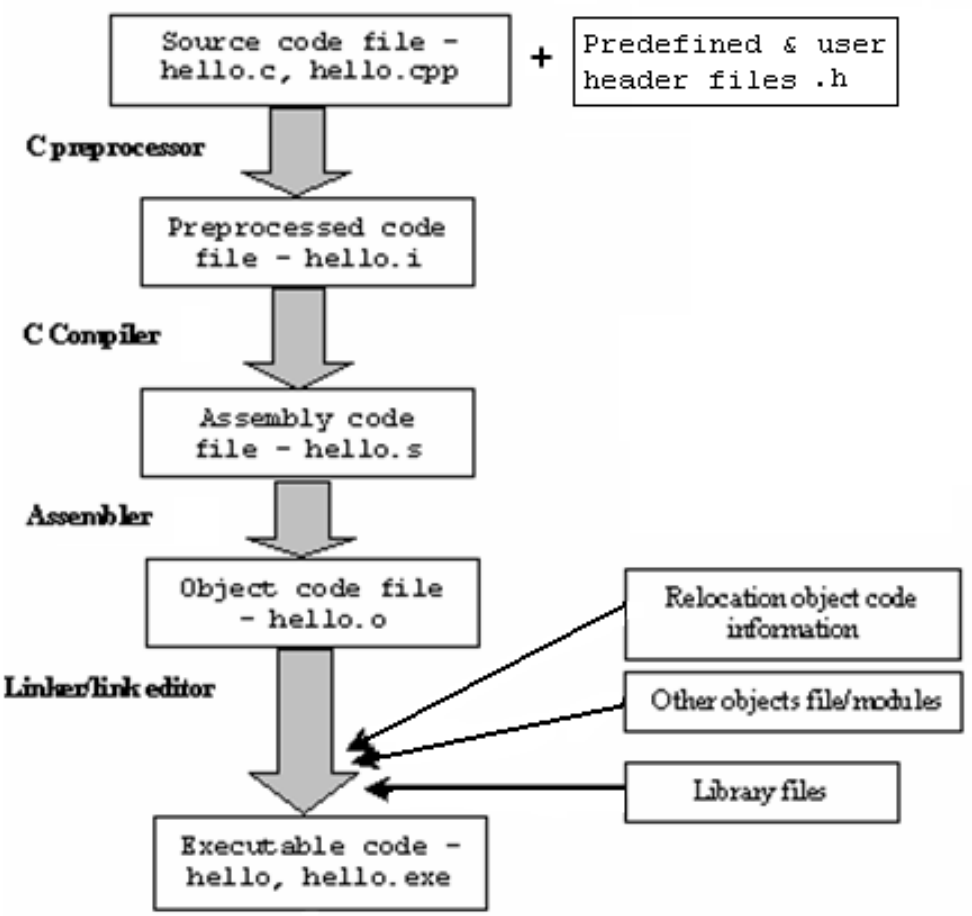
Struktura C programa

- konstante i tipovi podataka
- promenljive (lokalne, globalne, eksterne)
- izrazi (if, while, for...)
- operatori (+, -,*,/,~, ..)
- funkcije sa argumentima
  - pass by value
  - pass by address (pnt)
  - pass by reference (C++)
- makro izrazi
- preprocesorske direktive
- biblioteke
- komentari

```
#include <stdio.h>
#include "mydef.h"

#define ....          /* lokalne definicije */
int gvInt;            // globalna promenljiva
extern ...            /* referenca na gvar */
int func1(...);       /* redeklaracija */

void main( void )
{
    int i;            /* lokalna promenljiva */
    izraz 1;          /* komentar */
    func1(i, &i);     /* poziv funkcije */
    izraz 2;
    .....
}
int func( int k, int *pk )
{
    izraz3;           /* telo funkcije */
    return 0;
}
```

Prevođenje C programa

- file_name.c
  - C izvorni kod
- file_name.h
  - C header file
- file_name.i
  - Preprocesirani kod
- file_name.s, .asm, .S
  - Asemblerski kod
  - Asemblerski kod koji se mora preprocesirati
- file_name.o, .obj
  - Objektni kod
- file_name.exe
  - Izvršni kod



Osnovni elementi C jezika

- Imena (simboli)
  - konstante, promenljive (memorijske), funkcije
  - rezervisana i korisnička
  - do 31 znak (mađarska i CamelCase notacija)
- Operatori
  - aritmetički, logički, za rukovanje bitima
  - a++, ++a
  - prioriteti postoje, ali preporučuje se upotreba zagrada

Operators	Relative precedence	Rank
++, --	1	Highest ↓ Lowest
*, /, %	2	
+, -	3	
Highest → Lowest		

Kontrola toka izvršenja

- Grananja
  - if, if–else, if-else-if
  - switch-case-break
- Petlje
  - for
  - while
  - do...while
  - Ugnježdene petlje (nested loops)
- Druge kontrole poput
  - goto, continue, return
- Funkcije za prekid izvršenja programa
  - exit(), atexit(), abort()

Direktive C Preprocesora

- Uključivanje drugih datoteka u fajl za prevođenje
  - #include , <std_lib.h>, “user.h”
  - koristiti / (ne \) i originalna slova (npr. MyDef.h)
- Definicija konstanti, tipova podataka i makroa

```
#define PI 3.14
#define AREA(a,b) ((a)*(b))
```

- Kontrola prevođenja (conditional compilation)

```
#if !defined(NULL)
    #define NULL 0
#endif

#ifdef DEBUG
    printf("Var x = %d\n", x);
#endif
```

C Preprocesor

- Uslovno izvršenje preprocesorskih direktiva
  - #error – compile-time error poruka
  - #pragma – instrukcije za kompajlera (/Zp)
    - compiler - specific

```
#if !defined(__cplusplus)
#error C++ compiler required.
#endif

#pragma pack([n])    // pakovanje na n-byte adresu
#pragma pack(1)      // na svaki byte
```

Operatori # i

- Raspoloživi u ANSI C
- # operator se zamenjuje stringom
  - #define HELLO(x) printf("Hi, " #x "\n");
  - HELLO(Bata) → printf("Hi, " “Bata" "\n"); → printf("Hi, Bata\n");
- ## operator spaja dva simbola (tokena) u jedan
  - #define CAT(p, q) p ## q
  - CAT(O,K) → OK

Predefinisani makroi

Simbolička konstanta	Opis
__DATE__	Datum prevođenja
__LINE__	Broj linije u .c datoteci
__FILE__	Ime datoteke izvornog koda
__TIME__	Vreme prevođenja
__STDC__	Označava ANSI C kompatibilnost

Potvrđivanje (Assertion)

- Makro assert() je definisan u assert.h

```
#define assert(exp) \
(void)( (exp) || (_assert(#exp, __FILE__, __LINE__), 0) )
```

- Koristi se najčešće u fazi ispitivanja koda (Debug)
  - Proverava logičku vrednosti izraza
  - Ako je izraz FALSE, assert ispisuje grešku i poziva abort()
- Primer:

```
assert(q <= 100);
...
assert(string != NULL);
```

Assertion failed: string != NULL, file assert.c, line 24
abnormal program termination

Osnovni tipovi podataka

Tip	Size (bits)	HW zavisnost	Komentar
unsigned char	8	ne	
char	8	ne	
short int	16	ne	
unsigned int	16, 32, 64?	da	
int	16, 32, 64?	da	
__int*n*	8, 16, 32, 64	ne	Windows podržava
enum	=int	da	Uređen skup vrednosti
unsigned long	32, 64	?	Long je veći ili jednak int-u
long	32, 64	?	
float	32	ne	7-digit precision
double	64	ne	15-digit precision
long double	80	ne	18-digit precision

Međusobna konverzija podataka

- Pri izvođenju aritmetičkih operacija
- Implicitno pretvaranje iz jednog u neki drugi tip podataka
  - sabiranje float, int, char brojeva
  - prenos argumenata u drugačuje definisane funkcije
  - **samo kod mešanih tipova podataka**
- Generalno, pravila konverzije nastoje da
  - "manji" format se prevodi u "veći" bez problema, uz potencijalni problem sa znakom
  - obrnuto je moguće, ali uz manji ili veći gubitak informacije
    - float -> int gubi samo mantisu
    - int –> char gubi (možda) sve značajno

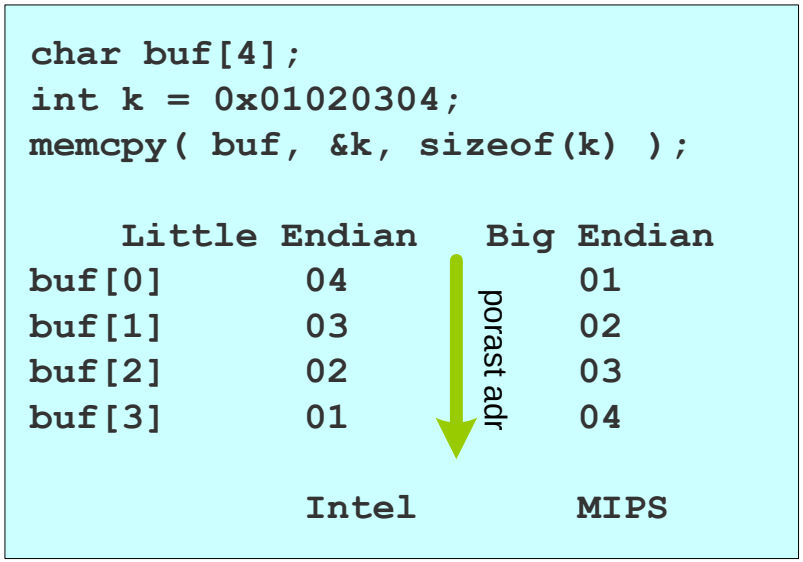
Kastovanje promenljivih (cast)

- Eksplicitno pretvaranje jednog tipa podataka u neki drugi tip
- Neophodno je zbog različite deklaracije argumenata
- Programer definiše način konverzije podataka
- Cast je unarni operator, koji se piše
  - (tip) izraz

```
unsigned long int next = 1;
/* rand: vraća slučajni celi broj iz [0,32767] */
int rand(void)
{
    next = next * 1103515245 + 12345;
    return (unsigned int)(next / 65536) % 32768;
}
...
int n;
sqrt((double)n)
```

Big i Little Endian

```
char buf[4];
int k = 0x01020304;
memcpy( buf, &k, sizeof(k) );
```



- Redosled slaganja okteta (byte-ova) u memoriji
- BE: najviši zadnji
- LE: najniži zadnji
- Bi-Endian: obe mogućnosti
- Zavisno od HW platforme
  - Intel je LE
  - MIPS je BE (ili LE - opciono)
- Isti problem je prisutan i kod
  - čitanja binarnih datoteka
  - komunikacione razmene podataka
- Ne postoji sistemsko rešenje, tj. mora se uprogramirati konverzija podataka
- Ograničava prenosivost koda

IEEE 754 format brojeva u pokretnom zarezu

- fp_value = S x 1.MANTISA x 2 **(Exp + Bias)**
- Eksponent je pomeren za konstantnu vrednost (bias)
- Mantisa je normalizovana, vodeća jedinica se čuva samo u real*10
  - -2 = -1.0 x 2**1 = 1100 0000 0000 0000 ... 0000 0000 = 0xC000 0000
  - 6 = +1.5 x 2**2 = 0100 1100 0000 0000 ... 0000 0000 = 0x4C00 0000
  - 0 = 1.0 X 2**-128

IEEE	C	Format	Bias
real*4	float	S EEEEEEEE FFFFFFFFFFFFFFFFFFFFFFF 0 1 8 9 31 sign bit, 8-bit exponent, 23-bit mantissa	127
real*8	double	sign bit, 11-bit exponent, 52-bit mantissa	1023
real*10	-	sign bit, 15-bit exponent, 64-bit mantissa	16383

C interpretacija

```
#if BYTE_ORDER == BIG_ENDIAN

struct ieee754dp_konst {
    unsigned sign:1;
    unsigned bexp:11;
    unsigned manthi:20;
    unsigned mantlo:32;
};

/* cannot get 52 bits into  */
/*     a regular C bitfield */

struct ieee754sp_konst {
    unsigned sign:1;
    unsigned bexp:8;
    unsigned mant:23;
};
```

```
#else /* little-endian */

struct ieee754dp_konst {
    unsigned mantlo:32;
    unsigned manthi:20;
    unsigned bexp:11;
    unsigned sign:1;
};

struct ieee754sp_konst {
    unsigned mant:23;
    unsigned bexp:8;
    unsigned sign:1;
};

#endif
```

Izvedeni tipovi podataka (korisnički)

- Direktive: struct, typedef, enum
- Sve korisničke tipove treba označiti typedef-om, i sufiksom _t
- Prvi član enum-a uvek mora biti inicijalizovan (CRules)
- Strukture su zgodan način za grupisanje i prenos podataka između funkcija
  - sizeof(struct) = Σ_i sizeof($član_i$), za pack(1)
  - pristup članu: ImeStruct.član, PntStruct->član

```
typedef struct {
   uint16_t      x;
   uint16_t      y;
} gdiPoint_t;

typedef enum { ILL = -1, GOOD, ...} boxStatus_t;

typedef struct listNode_t {
   struct listNode_t  *next;
   int                data;
} listNode_t;
```

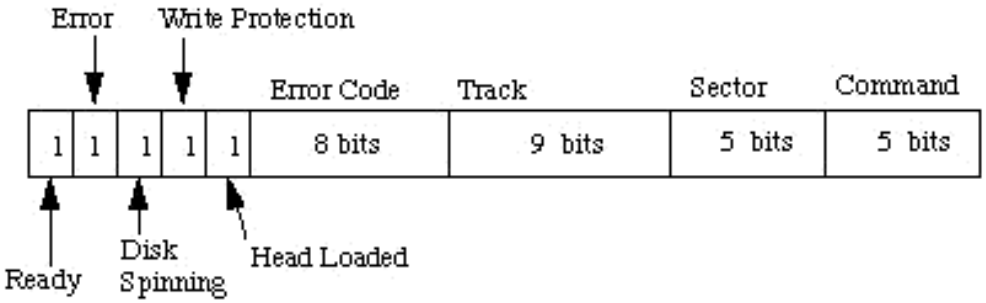
Izvedeni tipovi podataka

- union direktiva definiše tip podataka gde svu članovi dele isti prostor za smeštanje (storage space)
- sizeof(union) = sizeof(najduži član)

```
union sample
{
int p;
float q;
};
…
union sample content = {234};
  ili
union sample content = {24.67};

  pristip clanovima unije

content.p = 37;
content.q = 1.2765;
```

Bit Fields

- Pristup bitima (bitskim poljima) u okviru reči unutar struct
  - unsigned int – vezan za HW platformu (16, 32, ..)
  - broj bita < dužine reči, nema pointera na bit-field promenljivu
- Problem sa portabilnošću
  - signed / unsigned ?
  - pakovanje sa leve ili sa desne strane?
  - ako je reč manja od ukupne dužine definisanih polja (Σ DuzPolja)
    - neki kompajleri dodaju sledeću reč, a neki samo resetuju bitski ofset
- Alternativa: maskiranje i pomeranje bita




```
struct DISK_REGISTER  {
      unsigned ready:1;
      unsigned error_occured:1;
      unsigned disk_spinning:1;
      unsigned write_protect:1;
      unsigned head_loaded:1;
      unsigned error_code:8;
      unsigned track:9;
      unsigned sector:5;
      unsigned command:5;
};
```

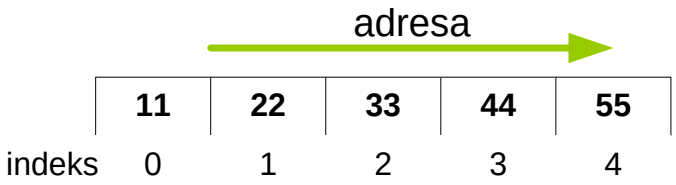
Promenljive

- Deo memorije koji čuva određeni tip podataka
  - tip_pod ImeVar;
  - sizeof(ImeVar) = sizeof(tip_pod)
- Tipovi promenljivih
  - lokalne - automatske (vidljive unutar jedne funkcije)
    - smeštene na stack-u
  - globalne (definisane izvan tela funkcije)
    - inicijalizovane ili neinicijalizovane
  - extern - spoljne (drugi modul)
  - static (dostupne samo funkcijama u jednom modulu)
  - const (konstantne – read only)
  - register – čuvaju se u registru
  - volatile – podložna promeni iz prekida npr.

Polja (nizovi)

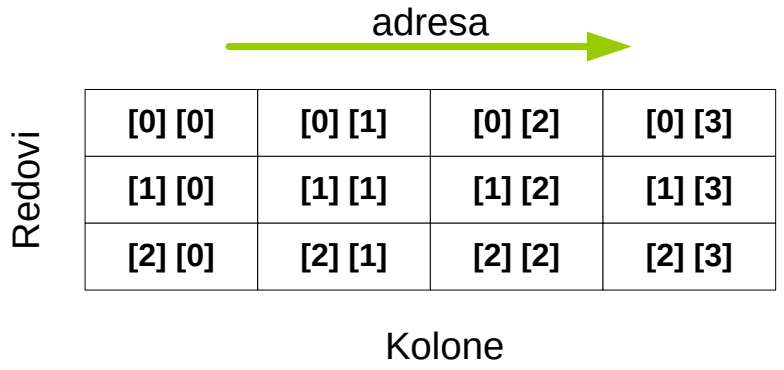
- Uređen skup podataka istog tipa, označenih jednim imenom
- Ime = adresa prvog elementa
- Pristup svakom od članova polja se ostvaruje pomoću jednog ili više indeksa, zavisno od dimenzije polja
  - d = 1, vektor d = 2, matrica

```
int k[5] = {11,22,33,44,55};
```



k - ime označava početak niza (adresu)
k[i] - vrednost niza na odstojanju i

```
int a[3][4] =
{1,2,3,4,5,6,7,8,9,10,11,12};
...
{{1,2,3,4},{5,6,7,8},{9,10,11,12}};
```



Mali test …

- Da li je korektna sledeća programska sekvenca?
- Ako jeste, šta će ispisati?

```
int i = 0, num[2] = { 16, 020 };
printf("   Nums = %d %d \n\n", num[i++], i[num]);
```

```
   Nums = 16 16

Press any key to continue . . .
```

Nizovi sa promenljivom dužinom

- *Variable-length array* (C99)
- Alokacija memorije na heap-u
- Visual Studio ne podržava !

```
float read_and_process(int n)
{
    float vals[n];
    for (int i = 0; i < n; i++)
        vals[i] = read_val();
    return process(vals, n);
}
```

Ukazivači - pointeri

- Ukazivač je promenljiva čija je vrednost adresa podatka smeštenog u memoriji, izražena u byte-ima
- Pointeri mogu adresirati
  - promenljive
  - funkcije
- Deklaracija pointera
  - tip_pod *pnt;
- tip_pod deklariše format i veličinu podatka na toj adresi
- uvećanje (++) i smanjenje (--) pointera menja njegovu vrednost za sizeof(tip_pod)
- isto važi i za sabiranje i oduzimanje pointera sa celobrojnom vrednošću
  - [pnt ± n] = [pnt] ± n x sizeof(tip_pod)

Ukazivači na promenljive

- Postavljanje pointera
  - char *bp = &chr;
  - int *ip = NULL; (0L)
- * - referenca (sadržaj promenljive tip_pod na adresi iz pointera)
  - char t = *bp;
- void * deklariše neoznačen pointer
  - samo adresa – odgovara unsigned char *
  - bez warning-a se može puniti ukazivačem proizvoljnog tipa
- Pointeri se mogu eksplicitno cast-ovati kao i promenljive
  - unsigned j = *(unsigned*) pnt
- Ukazivač na ukazivač
  - int **pnt; char **bp -> char *bp[];
- Polje ukazivača
  - int *pnt [20];

Ukazivači na promenljive

- Primer je dat za int, ali je adekvatan i bilo kom drugom tipu podataka

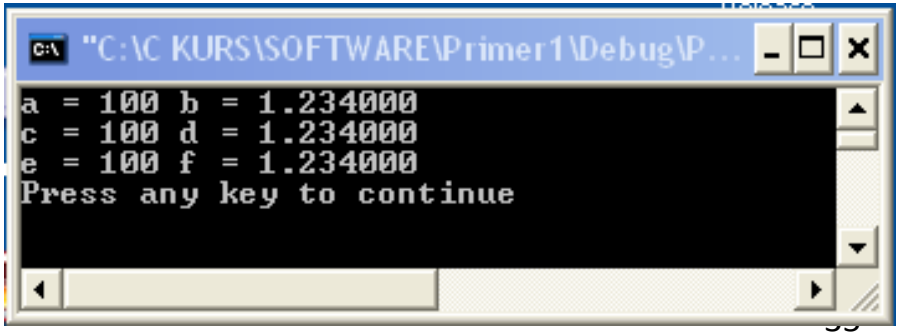
```
int i = 2;
int k[5]= {11,22,33,44,55};
int *pnt;
...
pnt = &i;
pnt = k; = &k[0] = k + 0;
pnt++;
pnt += i;

i = *pnt;    // i=44
```

Ukazivači na funkcije

- Adresa funkcije, suštinski je isto, ali imamo i deklaraciju argumenata

```c
#include <stdio.h>

/* functions' prototypes */
int fun1(int, double);
int fun2(int, double);
int fun3(int, double);

/* an array of a function pointers */
int (*p[3]) (int, double);

int main()
{
   int i;

   /* assigning address of functions */
   p[0] = fun1;
   p[1] = fun2;
   p[2] = fun3;

   /* calling with arguments */
   for(i = 0; i <= 2; i++)
   {
      (*p[i]) (100, 1.234);
   }
   return 0;
}
```

```c
/* functions' definition */
int fun1(int a, double b)
{
   printf("a = %d b = %f", a, b);
   return 0;
}

int fun2(int c, double d)
{
   printf("\nc = %d d = %f", c, d);
   return 0;
}

int fun3(int e, double f)
{
   printf("\ne = %d f = %f\n", e, f);
   return 0;
}
```

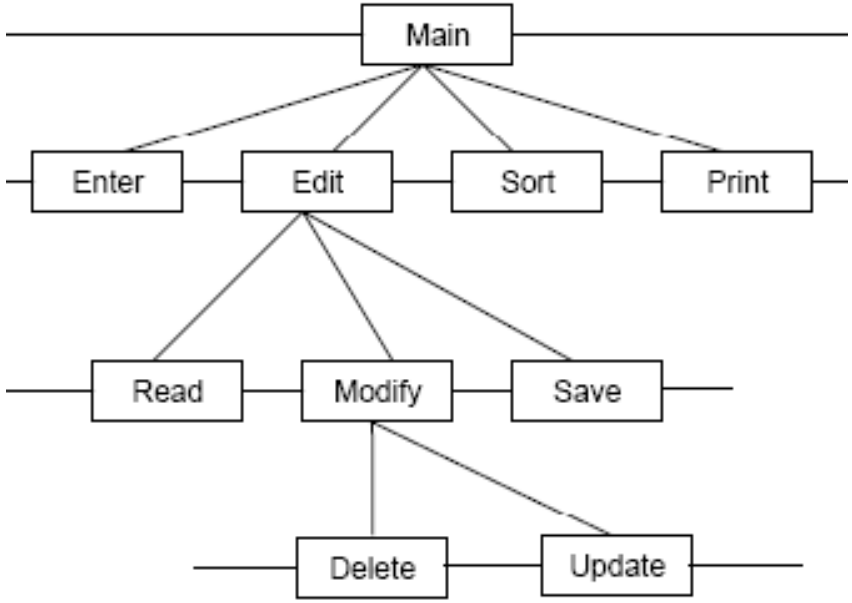


Funkcije

- Osnovna programska jedinica (procedura) koja izvršava određenu obradu (funkciju)
  - označena imenom preko kog se poziva
  - prihvata ulazne argumente i vraća rezultate
  - može koristiti globalne promenljive
  - izvršava se nezavisno od ostatka programa
- Višestruko korišćenje (*reuse*)
  - predefinisane funkcije (biblioteke - .lib, .h)
  - korisničke funkcije

Struktuirano programiranje

- Ono u kome se pojedini programski zadaci odvijaju u nezavisnim sekcijama programskog koda (funkcijama)
- Podela koda na manje celine koje se lakše
  - pišu, testiraju, održavaju i ponovo koriste
  - bez dupliranja istih segmenata koda
- *Top-down* pristup
- Vodi ka slojevitosti PP
- Izbegavanje *goto* izraza
- Upotreba *struct*, *while*, *for*



Definicija i poziv funkcije

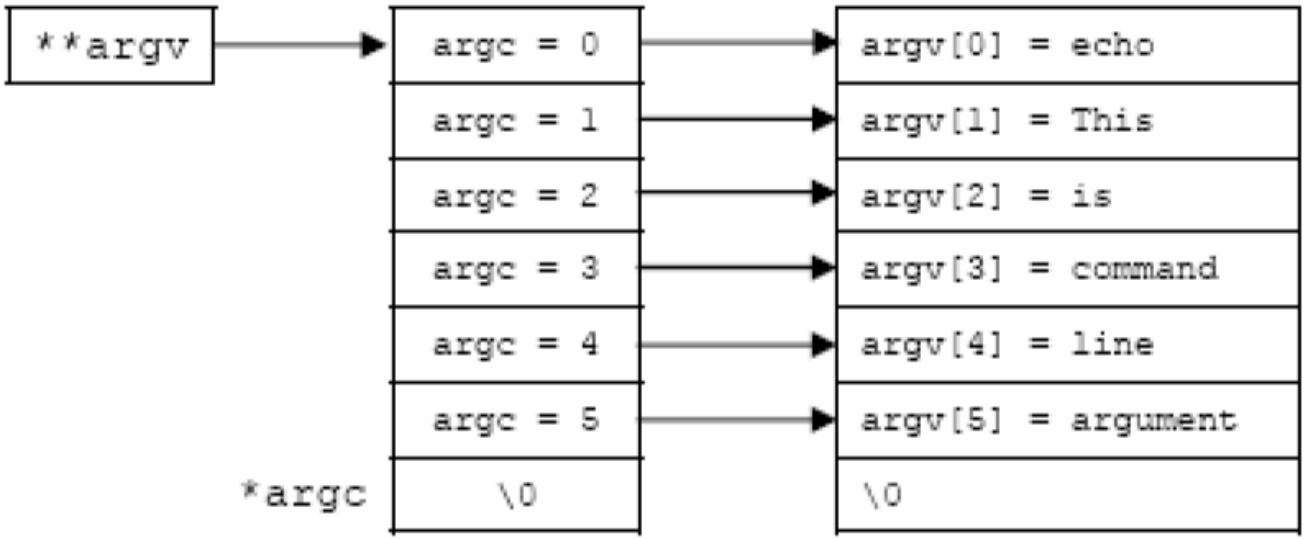
- Prototip (redeklaracija) – neophodna kompajleru
- void funkcija ne vraća povratni kod
- inline funkcija – kompajler ubacuje kod funkcije umesto poziva
- Prenos argumenata /rezultata preko
  - vrednosti
  - adrese

```
int fun3( int e, double *f);
...
int fun3( int e, double *f)
{
    int ret = 0;
    printf("e = %d f = %f\n", e, *f);
    *f = 3.25;
    return ret;
}
```

Main funkcija

- Ulazna tačka u program
- int main(int argc, char **argv)
- Primer: program echo

```
C:\>echo This is command line argument
This is command line argument
```



Rekurzivne funkcije

- One koje same sebe pozivaju
- Uslov: korišćenje **samo** argumenata i lokalnih promenljivih
  - korišćenje stack-a

```
// Rekurzivno racunanje faktorijala !n
long factor(long n)
{
   if( n <= 1 )
     return 1;
   else
     return( n * factor(n-1) );
}
```

Funkcije sa promenljivom listom argumenata

- varargs funkcije (*variadic functions*)
- koriste makroe definisane u stdarg.h

```
int sum_up( int count, ... )
{
   va_list ap;
   int i, sum = 0;

   va_start (ap, count);           /* Initialize arg_list */
   for( i = 0; i < count; i++ )
     sum += va_arg (ap, int);      /* Get next arg */
   va_end( ap );                   /* Clean up */
   return sum;
}
```

Standardne biblioteke

- Kolekcija predefinisanih funkcija, kao pomoć programeru
  - .lib sadrži objektni kod
  - .h sadrži prototipe i prateće definicije konstanti, makroa i tipova promenljivih
- ANSI C standardne biblioteke

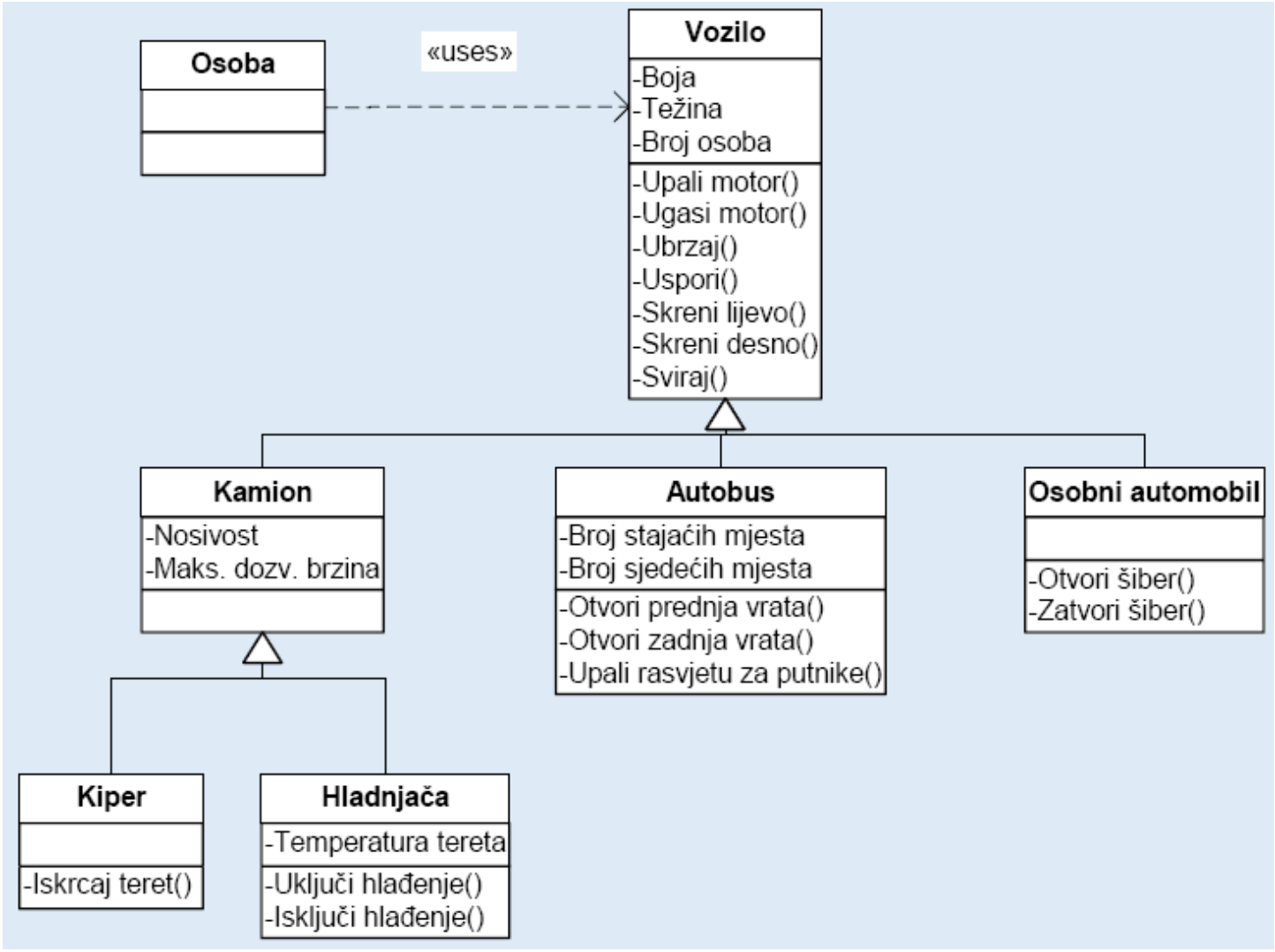
ANSI C Standard Library headers (C99)			
assert.h	inttypes.h	signal.h	stdlib.h
complex.h	iso646.h	stdarg.h	string.h
ctype.h	limits.h	stdbool.h	tgmath.h
errno.h	locale.h	stddef.h	time.h
fenv.h	math.h	stdint.h	wchar.h
float.h	setjmp.h	stdio.h	wctype.h

- POSIX (*Portable Operating System Interface*) API sprega
  - Unix, Linux, Nucleus, parcijalno Windows

Principi objektno orijentisanog programiranja

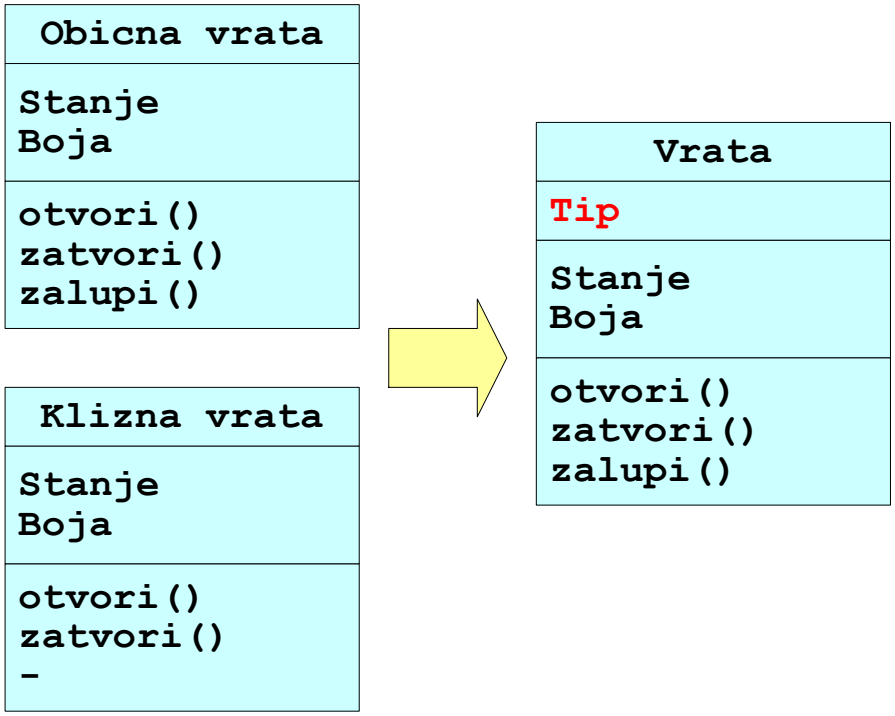
- Program = skup objekata u međusobnoj interakciji
- Tip objekta (klasa)
  - metodi (ponašanje) i svojstva (atributi)
  - mogu biti privatni, zaštićeni ili javni
- Objekat je instanca klase (izvršna kopija)
  - oponaša karakteristike objekata “stvarnog sveta”
  - pnt na baznu klasu + atributi
- Razmena podataka, preko odgovarajuće sprege
- Nasleđivanje – specijalizacija osnovne klase
- Enkapsulacija (učauravanje)
  - promene stanja objekta mogu se raditi samo posredstvom definisanih metoda, koji imaju ugrađene kontrolne mehanizme
- Apstrakcija – redukcija detalja, sa fokusom na deo od interesa
- Polimorfizam – definicija više metoda/operatora istog imena, koji funkcionišu nad objektima različitih tipova

Primer: VOZILO



Primena OOP u C programiranju

- C nije OOP programski jezik, ipak
- Deo principa OOP može se primeniti u razvoju C programa
- Razmotrimo primer Vrata



Definicija tipova podataka i osnovnih primitiva

- Definicija “klase” sa privatnim metodama

```
typedef struct vrata
  {
     int tip;              // obicna, klizna
     int stanje;           // zatv, otv
     char boja[20];
     void (*otvori)( struct vrata *vp );
     void (*zatvori)( struct vrata *vp );
     void (*zalupi)( struct vrata *vp );
  } VRATA;

  typedef enum
  {
    OBICNA = 0,
    KLIZNA
  };

  typedef enum
  {
    ZATVORENA = 0,
    OTVORENA
  };
```

```
void otvori( VRATA *vp )
  {
     vp->stanje = OTVORENA;
     // otvori rotaciono
     printf( "Otvori %s vrata\n", vp->boja );
  }

  void zatvori( VRATA *vp )
  {
     vp->stanje = ZATVORENA;
     // zatvori rotaciono
     printf( "Zatvori %s vrata\n", vp->boja );
  }

  void otvori_klizna( VRATA *vp )
  {
     vp->stanje = OTVORENA;
     // otvori klizno
     printf( "Otvori %s vrata\n", vp->boja );
  }

  void zatvori_klizna( VRATA *vp )
  {
     vp->stanje = ZATVORENA;
     // zatvori klizno
     printf( "Zatvori %s vrata\n", vp->boja );
  }
```

Definicija objekata i javnih metoda

```
VRATA vrata[2] =
{
   { OBICNA, ZATVORENA, "Bela", otvori, zatvori, zatvori },
   { KLIZNA, OTVORENA,  "Zuta", otvori_klizna, zatvori_klizna, NULL }
};
```

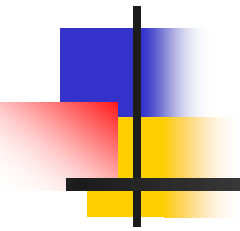
```
/*

#define Otvori(vp) vp->otvori(vp)
#define Zatvori(vp) vp->zatvori(vp)
#define Zalupi(vp) vp->zalupi(vp)

*/
```

```
void Otvori( VRATA *vp )
{
   if( vp && vp->otvori )
      vp->otvori( vp );
}

inline void Zatvori( VRATA *vp )
{
   if( vp && vp->zatvori )
      vp->zatvori( vp );
}

inline void Zalupi( VRATA *vp )
{
   if( vp && vp->zalupi )
      vp->zalupi( vp );
}
```

Glavni program

```
int main(int argc, char* argv[])
{
   register VRATA *vp = vrata;

   Otvori( vp );
   Zalupi( vp );

   //vp++;
   vp = &1[vrata];
   Zatvori( vp );
   Zalupi( vp );

   return 0;
}
```

- U C programu se veći deo OOP principa suštinski može ispoštovati, sem
  - Nasleđivanja i polimorfizma
  - Enkapsulacija se ostvaruje kroz dostupnost .h



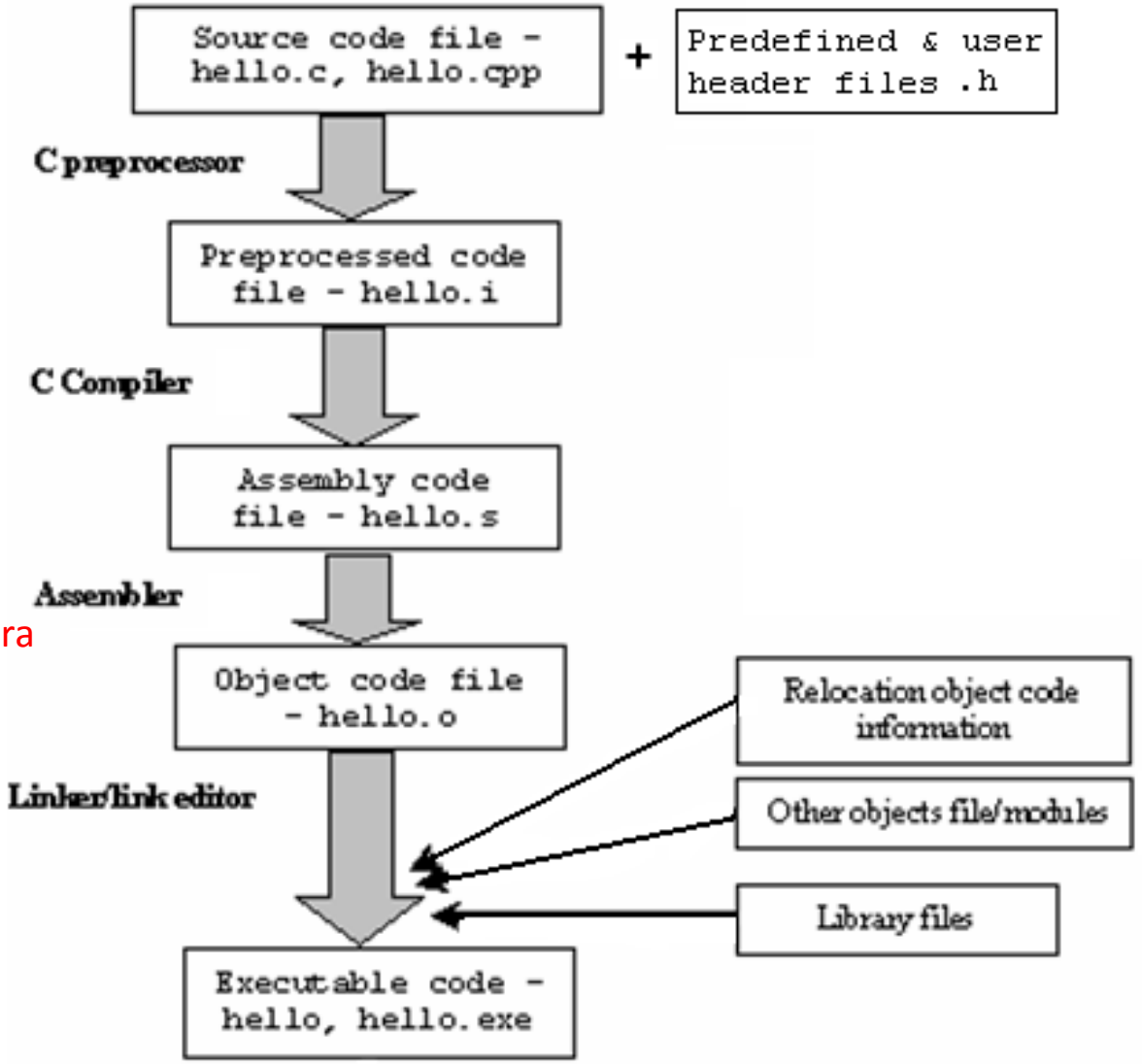
PREVOĐENJE i IZVRŠENJE C PROGRAMA

Proces prevođenja, run-time model

Kompajler, asembler, linker, loader

Prevođenje C programa

- file_name.c
  - C izvorni kod
- file_name.i
  - Preprocesirani kod
- file_name.h
  - C header file
- file_name.s, .asm
  - Asemblerski kod
- file_name.S
  - Asemblerski kod koji se mora preprocesirati
- file_name.o, .obj
  - Objektni kod
- file_name.exe
  - Izvršni kod



Faze prevođenja C programa

Izvorni kod → Asemblerski kod → Objektni kod → Izvršni kod

- *Preprocesor* – prvi prolaz prevođenja C koda
- *Kompajler* – drugi prolaz, generiše asemblerski kod
- *Asembler* – generiše objektni kod i asemblerski listing
- *Linker* – finalna faza, gde se
  - više .obj i .lib modula kombinuju u jedan izvršni (.exe)
  - rešavaju reference na spoljne simbole
  - vrši konačno dodeljivanje adresa funkcijama/promenljivim (relokacija)
- U IDE kompajlerima ove procedure su integrisane
- Razmotrićemo neke bitne elemente ovog procesa

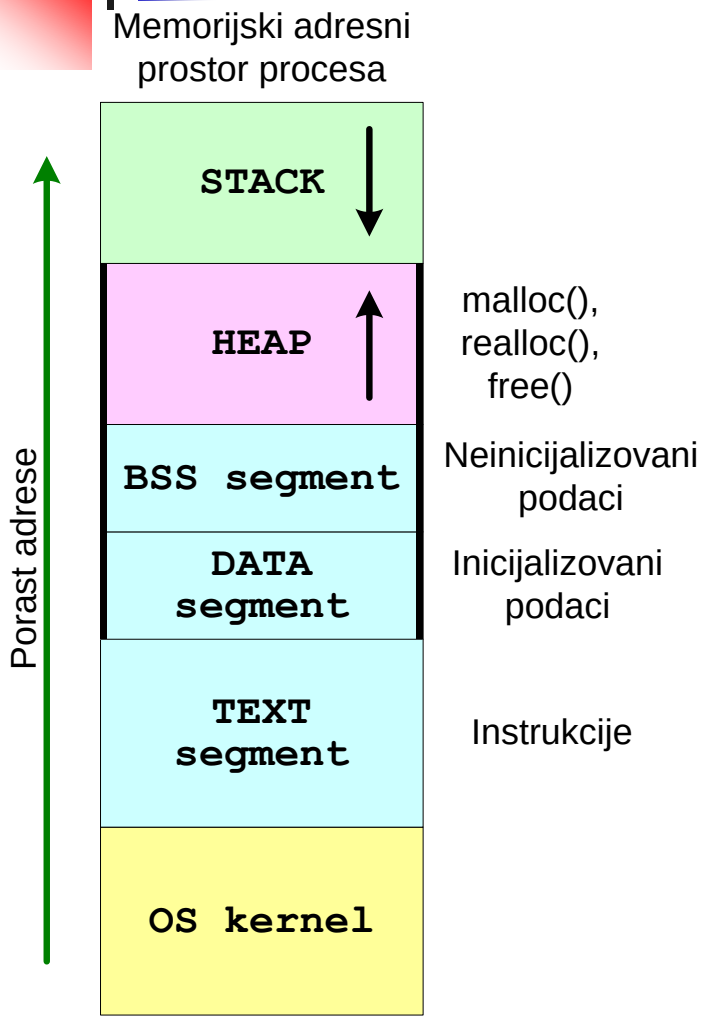
Programski model

- U krajnjoj posledici programski model je
  - Organizacija promenljivih i funkcija, i načina međusobnog pozivanja, u okviru izvršnog programa
  - Iako zavisan od HW platforme, sličan u većini implementacija
- Postaje značajan u krajnim fazama C prevođenja, zato pod programskim modelom podrazumevamo
- **Konvencije generisanja asemblerskog koda**
  - memorijska alokacija promenljivih
  - registarske konvencije i korišćenje “stack”-a
- Garancija međusobne kompatibilnosti
  - povezivanje programa različitih kompajlera
  - aplikativnih programa i operativnih sistema

Alokacija promenljivih (*storage class*)

- Segmenti memorije koji sadrže promenljive različitog tipa
  - Automatske – lokalne promenljive na stack-u
    - `auto int i;` `// def unutar funkcije`
  - Registarske – čuvaju se u registru radi brzine
    - `register int var;` `// optimizacija?`
  - Globalne – definisane izvan funkcije
    - inicijalizovane (dobra praksa) i neinicijalizovane
    - `extern, static, const`
    - static moze biti i lokalna (auto) promenljiva, ali samo u pogledu vidljivosti – alokacija je trajna
    - const rezultira alokacijom samo u slučaju adresne reference
  - Dinamički alocirane (*run-time*)

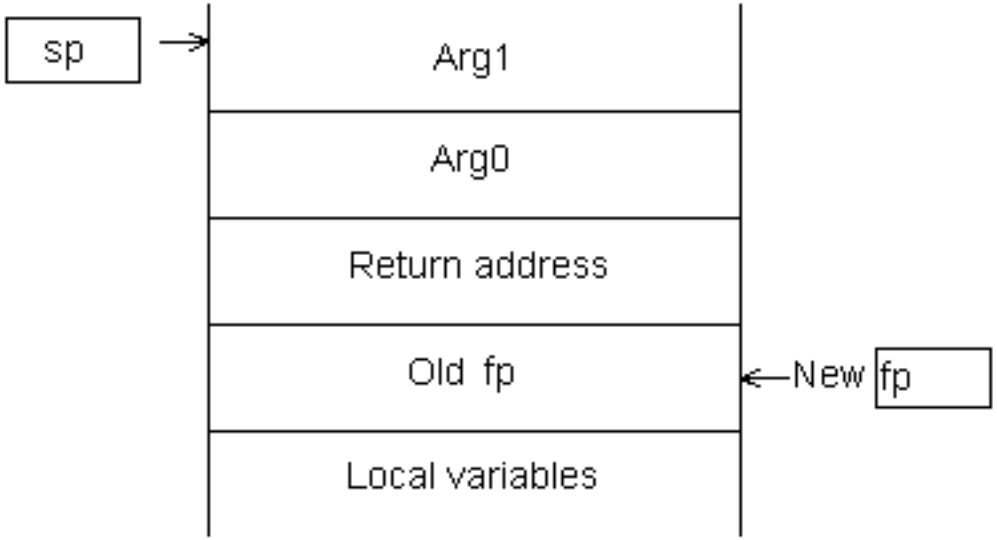
Memorijski segmenti programa



- Svaki program dobije svoj prostor pri punjenju
- TEXT sekcija može biti deljena
  - reentrant code - programs
- DATA – non-zero init global i static
  - .rdata – read only (const)
- BSS (*Block Started by Symbol*) - u izvršenju DATA, u .exe ga nema
- HEAP – dinamička memorija
  - pristup samo preko pointera
  - kontrola opsega (kompajler, run-time)
- STACK – lokalne promenljive, poziv funkcija i prenos argumenata
  - stack frame, STACKSIZE

Korišćenje stack-a pri pozivu funkcije

- Stack frame – osnovna struktura
  - formira se i briše pre/po svakom pozivu funkcije na tekućoj lokaciji stack-a
  - sadrži argumente, povratnu adresu i lokalne promenljive
  - koristi se poseban registar – frame pointer

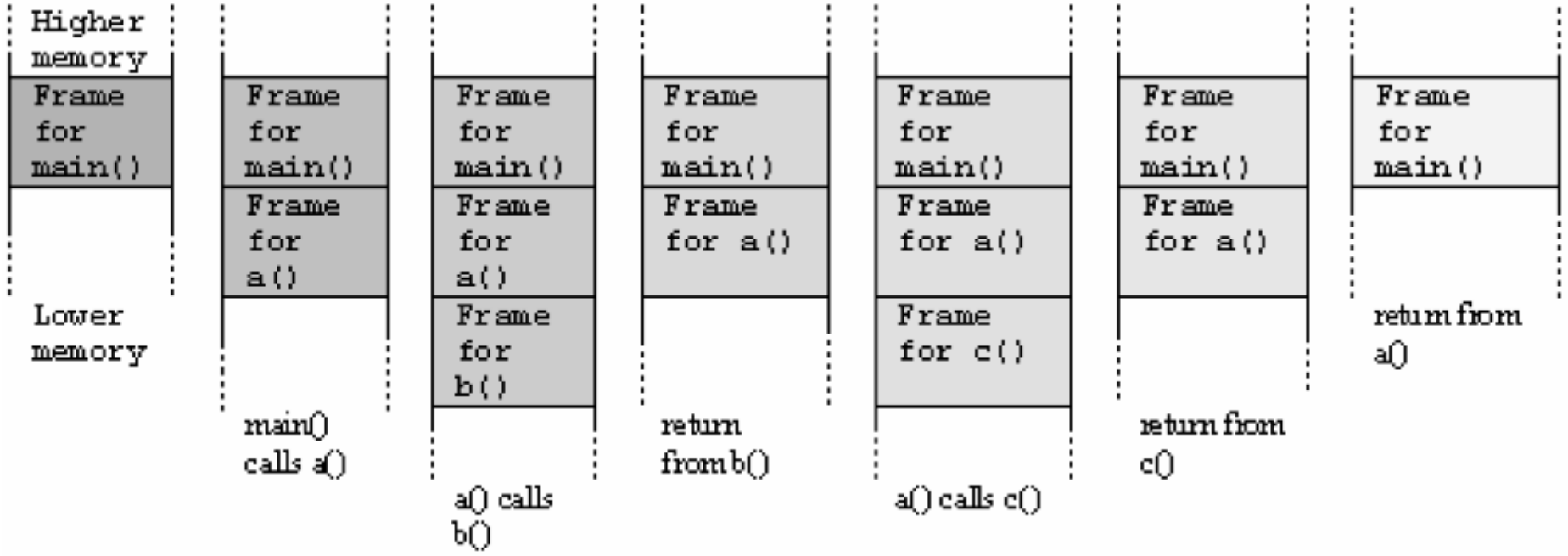


Primer “disanja” stacka

```
int main()
{
    a();
    return 0;
}
```

```
int a()
{
    b();
    c();
    return 0;
}
```

```
int b()
{ return 0; }

int c()
{ return 0; }
```



Konvencije poziva funkcije

- Neusaglašeno između proizvođača
  - redosled argumenata
  - ko ih uklanja (cleanup)
  - imenovanje/označavanje funkcija i opcije poziva

```
void __cdecl TestFunc(float a, char b, char c);   //Borland and Microsoft
void  TestFunc(float a, char b, char c)  __attribute__((cdecl));  //GNU GCC
```

Startup Code

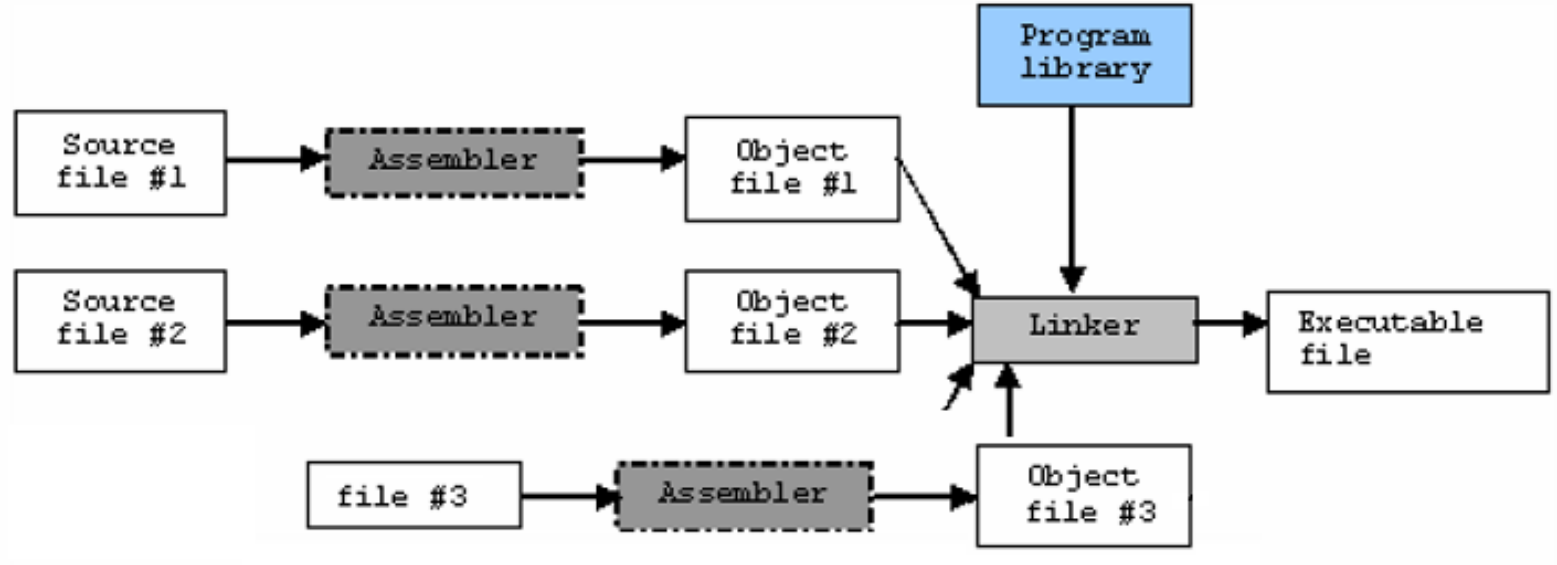
- Uobičajeno je da razvojni alati ubacuju deo asemblerskog koda koji priprema izvršenje programa pisanog u višem programskom jeziku
- Startup code obezbeđuje standardno run-time okruženje na desktop računaru sa OS
  - alokacija stack-a, prenos argumenata, env. promenljiva, itd.
- Kod embedded sistema, funkcionalnost ovog koda je značajno šira
  - *startup.asm, crt0.s* (C runtime), ili slično
  - Zabrani sve prekide
  - Kopiraj inicijalizovane podatke iz ROM u RAM memoriju
  - Upiši 0 (zero) u neinicijalizovane podatke
  - Alociraj prostor i inicijalizuj stack i SP
  - Kreiraj i inicijalizuj heap
  - Dozvoli prekide i pozovi main()
  - Kod iza main (ukoliko ipak izađe), treba da zaustavi procesor (halt) ili već ...

Objektni kod

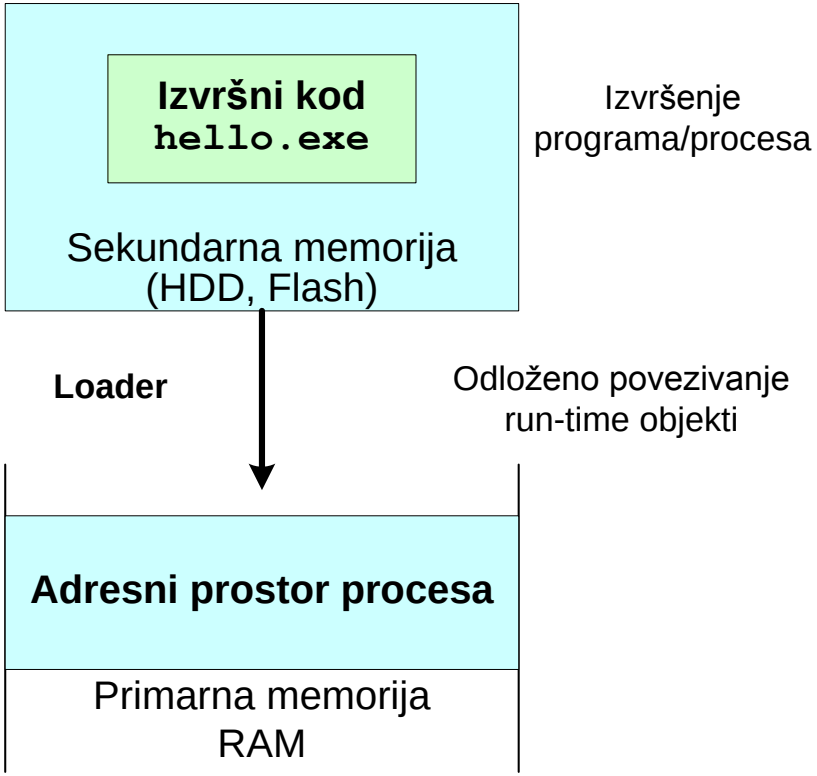
- Krajnji produkt kompajlera, koji u binarnom formatu sadrži sledeće informacije
  - Kod i inicijalizovane podatke (po segmentima)
  - Tabelu simbola (ime + adresa:data, text, extern)
  - Relokacioni slogovi (informacije za linkera)
  - Debug informacije (region asm.koda i broj C linije)
- Adrese su relokatibilne u odnosu na početak modula (malo uprošćeno)
- Različiti formati
  - ELF (Linux), COFF (UNIX), PE (Windows), ...
- Biblioteka (.lib) je kolekcija objektnih modula

Linker

- Povezivanje u izvršni kod, rešavanjem međusobnih adresnih referenci
  - Statičko – sve potrebno u izvršni fajl
  - Dinamičko – deljeni izvršni moduli (.dll)
- Adrese su i dalje relativne, sada u odnosu na početak .exe

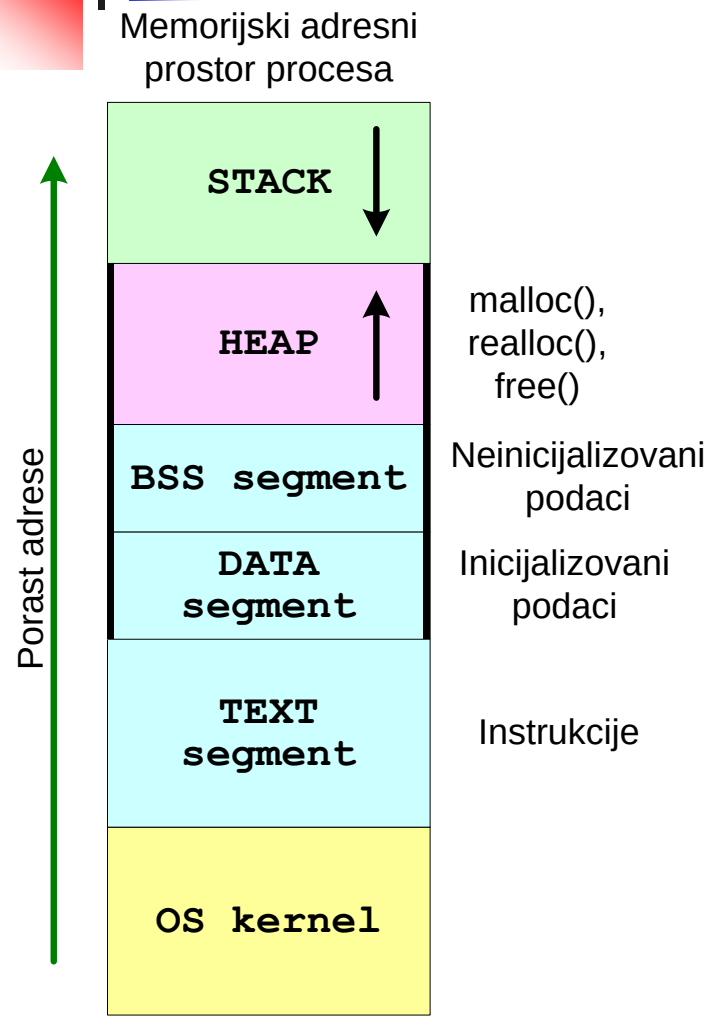


Punjač (*Loader*)

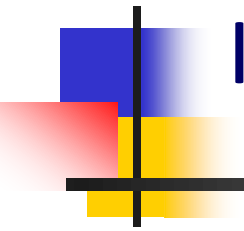


- Pre izvršenja, program se puni u operativnu memoriju
  - Funkcija OS
  - Sa diska
    - sve se kopira
  - Iz Flash-a
    - sve se kopira (max.varijanta)
    - samo inicijalizovane promenljive (min.varijanta)
- Dinamičko povezivanje
  - deffered linking
  - run-time moduli/biblioteke

Proces punjenja



- Verifikacija izvršnog programa i računanje memorijskih zahteva
- Verifikacija raspoložive memorije i prava pristupa
- Alokacija potrebne memorije i prenos sa sekundarne memorije
- Formiranje sekcije podataka (DATA+BSS+Heap sekcija)
- Inicijalizacija stack-a i argumenata za main()
- Pozivanje main() funkcije



IA-32 specifičnosti

x86 Arhitektura

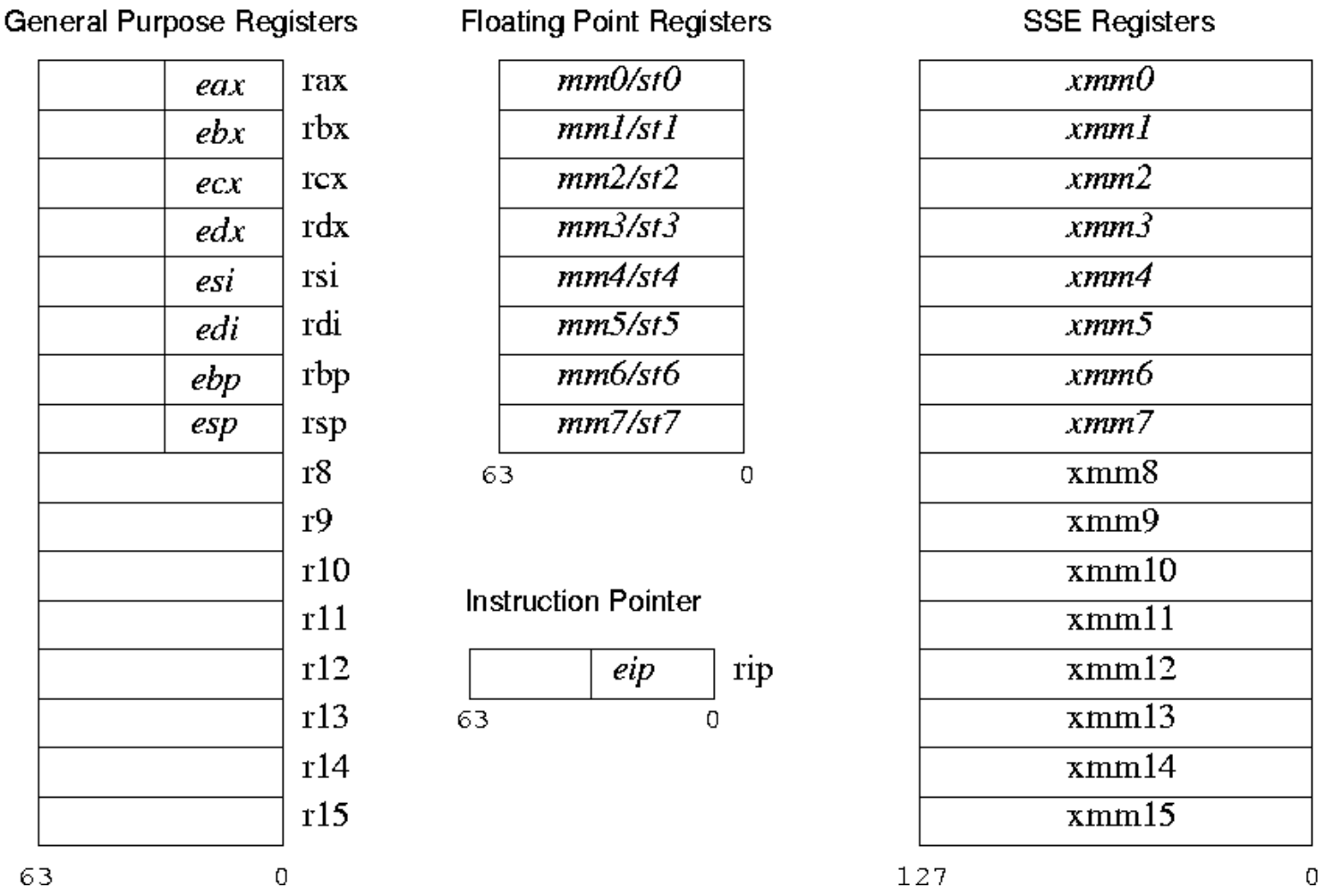
Visual C/C++ opcije

Oznaka	Brisanje stack-a	Prenos parametara
__cdecl	caller	sa desna na levo, default za variadic C funkcije pravi veći kod od *stdcall* jer svaki poziv uključuje "*cleanup*" kod
__stdcall	callee	tzv. __pascal, sa desna na levo, funkcije moraju imati prototip
__fastcall	callee	deo parametara se čuva u registrima, ako je moguće

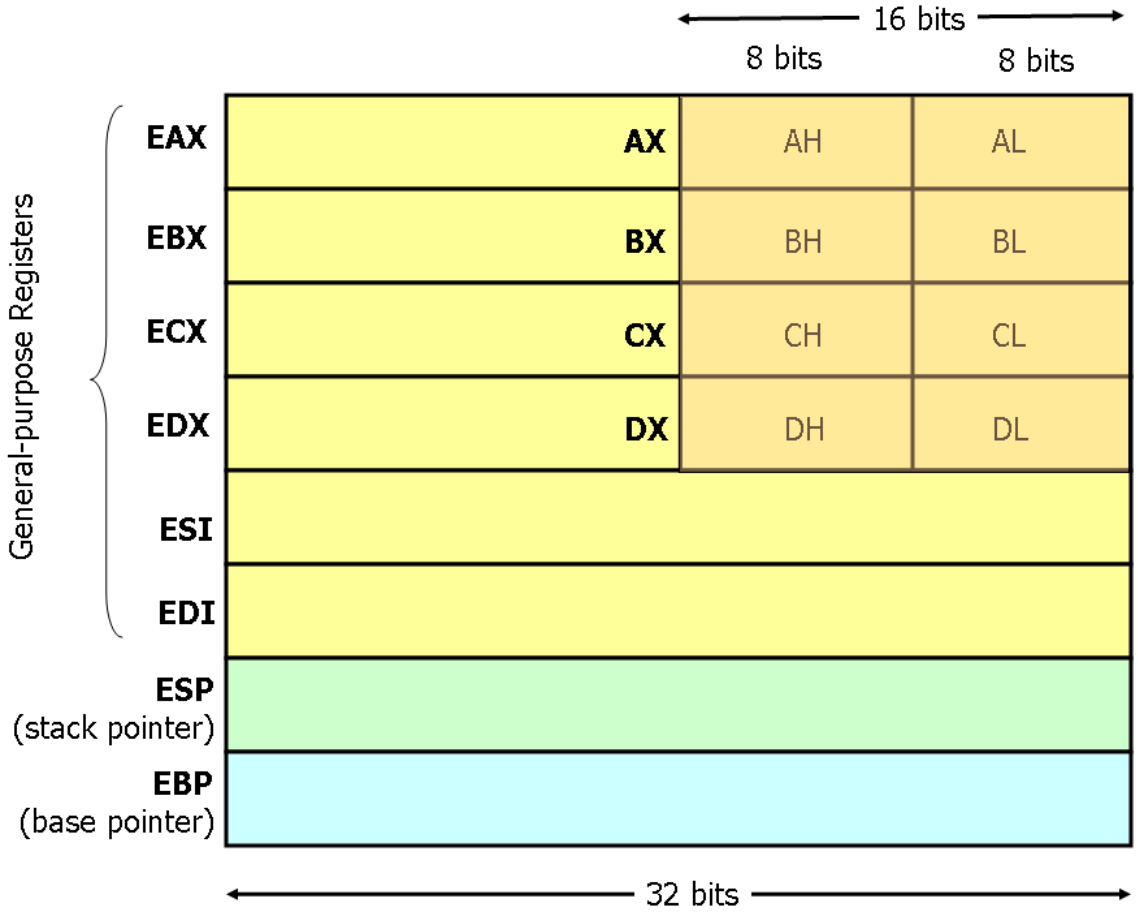
```
/*example of __cdecl*/
push arg1
push arg2
call function
add ebp, 12  ;stack cleanup
```

```
/*example of __stdcall*/
push arg2
push arg1
call function
;stack cleanup done by callee
```

IA-64 Registri



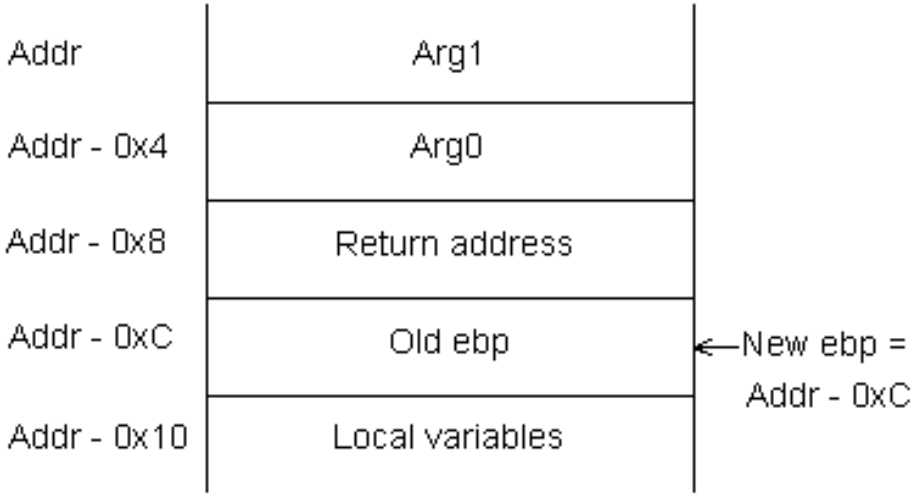
IA-32 Registri



IA-32 registarske konvencije

Type	Name	Purpose
General	eax	integer return value
	edx	dividend register (for divide operations)
	ecx	count register (shift and string operations)
	ebx	local register variable
	ebp	stack frame pointer
	esi	local register variable
	edi	local register variable
	esp	stack pointer
Floating-point	st(0)	floating-point stack top, return value
	st(1)	floating-point next to stack top
	st(...)	
	st(7)	floating-point stack bottom

IA-32 rukovanje stack-om



- Stack je opadajući

*(ebp) - frm_pnt caller funkcije

*(ebp + 4) – povratna adresa

*(ebp + 4 + 4*i) – Arg[i] (32bitni)

*(ebp - offset) - lokalne varijable

- **Prolog funkcije:**

```
push  ebp                ; Save ebp
mov  ebp, esp            ; Set stack fp
sub   esp, localbytes  ; Space for locals
push    <registers>    ; Save registers
```

- **Epilog funkcije:**

```
pop    <registers>  ; Restore regs
mov    esp, ebp       ; Restore sp
pop    ebp            ; Restore ebp
ret                   ; Return
```

Primer 1 (segmenti)

- _TEXT SEGMENT PARA USE32 PUBLIC 'CODE‘
  - sadrži instrukcije
- _DATA SEGMENT DWORD USE32 PUBLIC 'DATA‘
  - globalni inicijalizovani podaci
- CONST SEGMENT DWORD USE32 PUBLIC 'CONST'
  - read-only segment
- _BSS SEGMENT PARA USE32 PUBLIC 'BSS‘
  - globali neinicijalizovani podaci
- _TLS SEGMENT DWORD USE32 PUBLIC 'TLS'
  - Thread Local Storage (Windows specific)

- MMU prava su adekvatna nameni segmenta

Primer 2

- Ilustracija veze C i asemblerskog koda
- Prikazan je kod bez optimizacije
- Uporedite ga sa optimizovanim
- Primer sa FP brojevima bio bi manje razumljiv

```
int main( void )
{
    int a, b, c;

    c = adder( 2, 3 );
     return 0;
}
```

```
int adder( int a, int b )
{
    int z;
    z = a + b;
    return z;
}
```

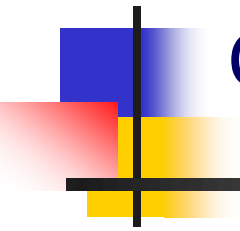
```
_main    PROC NEAR
    push ebp
    mov  ebp, esp
    sub  esp, 12
; 16: int a, b, c;
; 17:
; 18: c = adder( 2, 3 );
    push 3
    push 2
    call adder
    add  esp, 8
    mov  [ebp-12], eax
; 19: return 0;
    xor  eax, eax
; 20: }
    mov  esp, ebp
    pop  ebp
    ret
```

```
adder PROC NEAR
; 7: {
    push ebp
    mov  ebp, esp
    push ecx
; 8:     int z;
; 9:     z = a + b;
    mov  eax, [ebp + 8]
    add  eax, [ebp + 12]
    mov  [ebp - 4], eax
; 10:    return z;
    mov  eax, [ebp - 4]
; 11: }
    mov  esp, ebp
    pop  ebp
    ret
```

adder funkcija sa i bez optimizacije

```
_TEXT    SEGMENT

_a$ = 8
_b$ = 12
adder PROC NEAR
; 7     : {
; 8     :     int z;
; 9     :     z = a + b;
    mov   eax, DWORD PTR _b$[esp-4]
    mov   ecx, DWORD PTR _a$[esp-4]
    add   eax, ecx
; 10    :     return z;
; 11    : }

    ret   0
adder ENDP

_TEXT    ENDS
```

```
_TEXT    SEGMENT

_a$ = 8
_b$ = 12
_z$ = -4
adder PROC NEAR
; 7     : {
    push  ebp
    mov   ebp, esp
    push  ecx
; 8     :     int z;
; 9     :     z = a + b;
    mov   eax, DWORD PTR _a$[ebp]
    add   eax, DWORD PTR _b$[ebp]
    mov   DWORD PTR _z$[ebp], eax
; 10    :     return z;
    mov   eax, DWORD PTR _z$[ebp]
; 11    : }
    mov   esp, ebp
    pop   ebp
    ret   0
adder ENDP
```



C PROGRAMIRANJE NA MIPS PLATFORMI

MIPS C programming

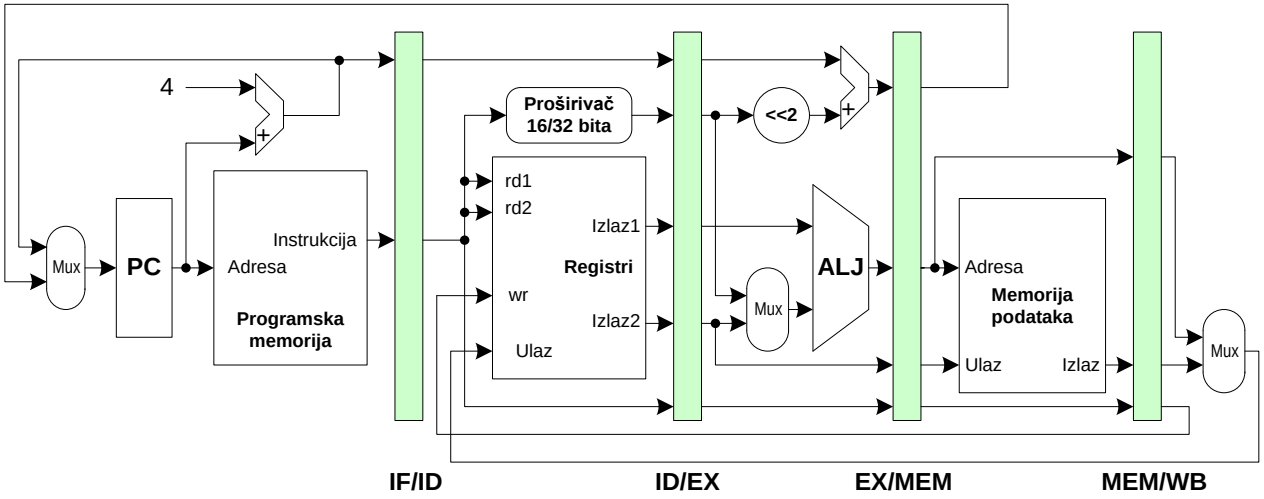
Specifičnosti C programiranja na *embedded* platformi

- Mešanje C i asemblerskog koda
- Neophodan je prisniji odnos sa HW osnovom
  - Instrukcije, adresiranja, memorijski prostor
  - Periferije – format registara i UI adrese
  - Zahtevi RISC protočne organizacije
- Poznavanje strukture podataka na stack-u nije opšte obrazovanje, nego živa potreba
- Poznavanje nestandardnih biblioteka
- Pristup UI uređajima iz C-a
- Rukovanje prekidima i DMA transferima
- Opcije kompajlera - pogotovo uticaj optimizacije
- U našem slučaju, interesantna je MIPS platforma

MIPS procesor

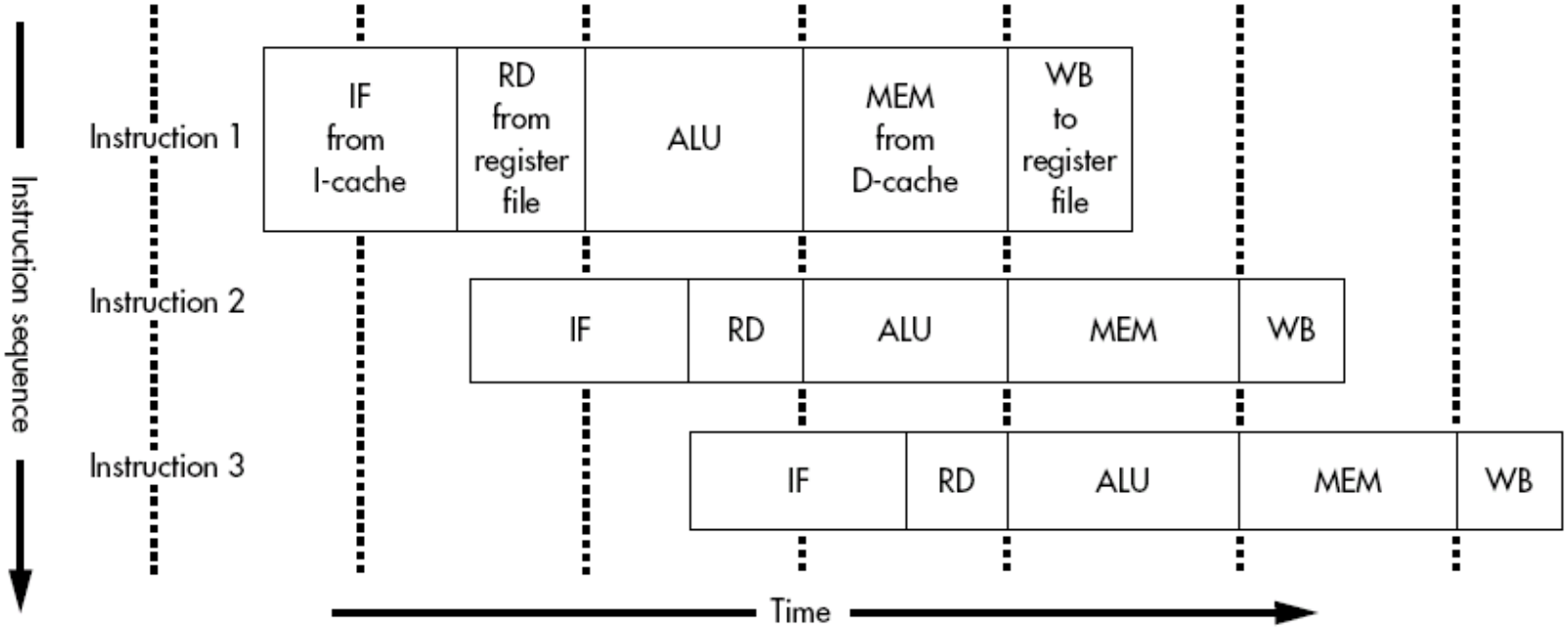
Microprocessor Without Interlocked Pipeline Stages

- Najelegantniji i najpopularniji RISC procesor
- 32 bita, 1982. godine, Univerzitet Stanford, SAD
- Danas raspoloživ u brojnim varijantama, uključujući i 64 bitnu
  - R2000, **R3000**, R4000, R6000, ...
- Trećina svih proizvedenih RISC procesora imaju MIPS jezgro
- Odlikuje se jasnim i konzistentnim skupom instrukcija
- Protočna organizacija (5-to stepena)



Razlika realnog od školskog MIPS-a

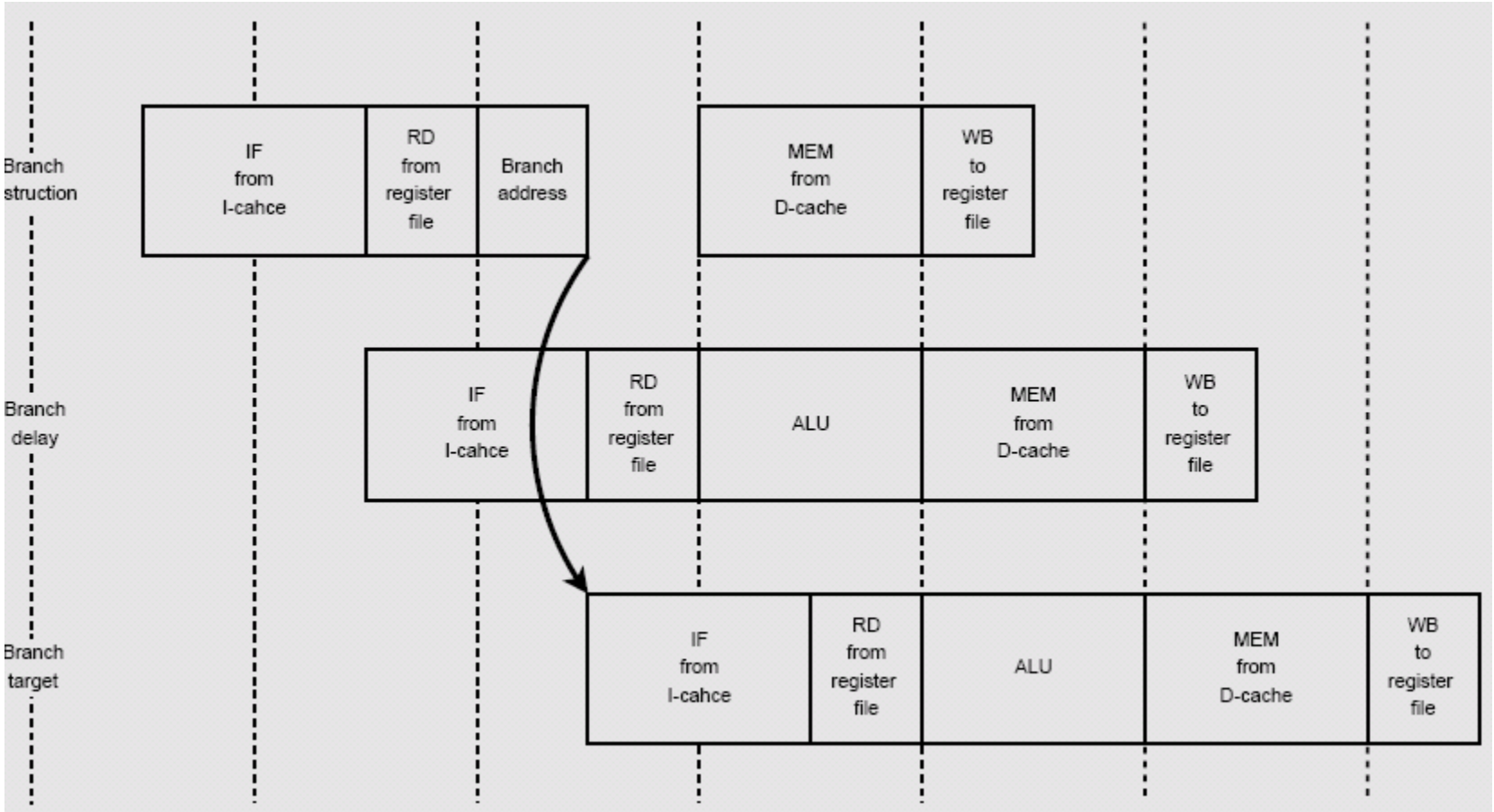
- Korišćenje cache memorija za instrukcije i podatke (I & D cache)
  - jedini način da se pipeline iskoristi
- Izvršenje 5 segmenata u 4 takta (ne 5)!



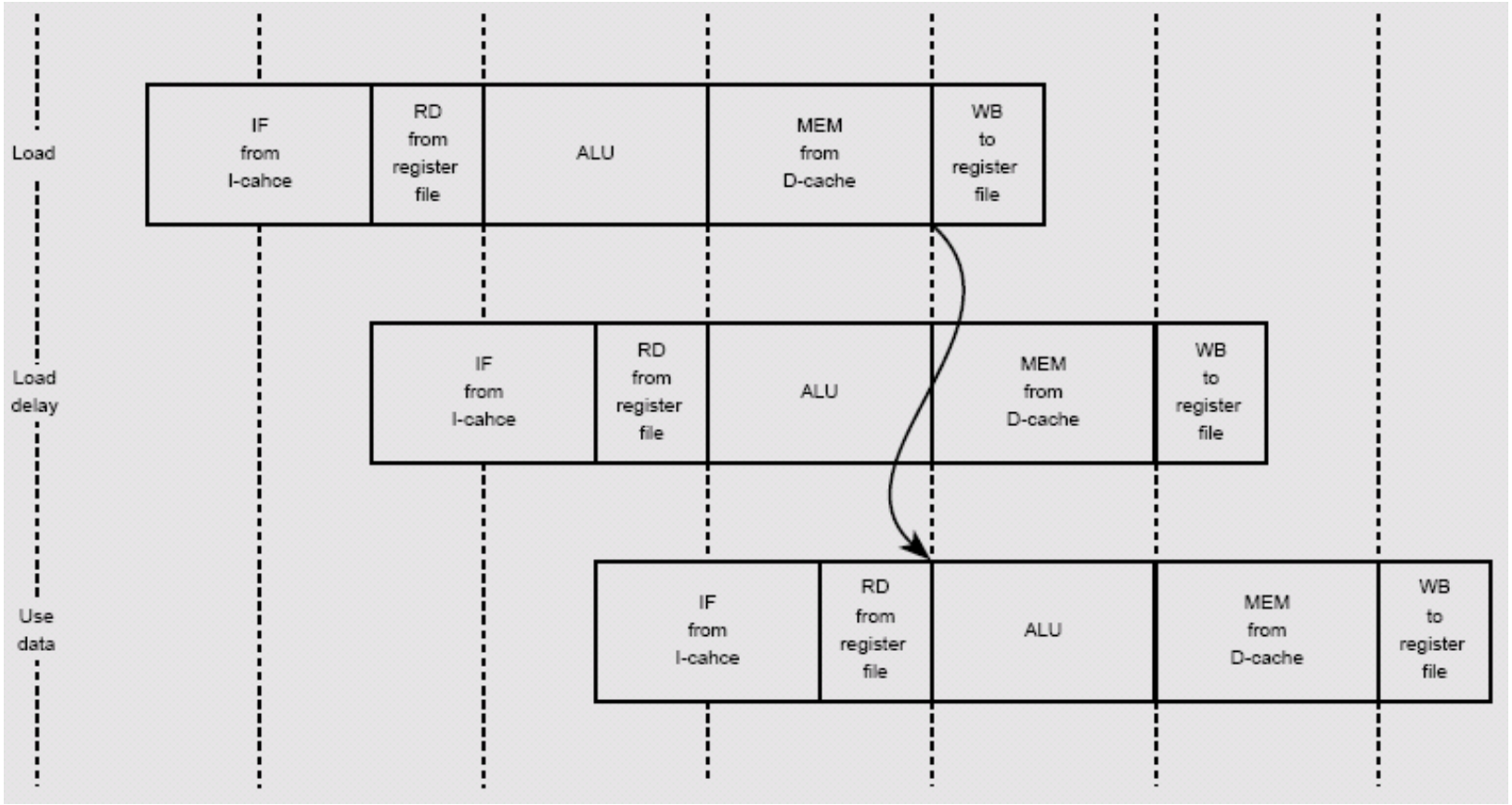
RISC ograničenja MIPS instrukcija

- Sve instrukcije su duge 32 bita
- Moraju se završiti pod ograničenjima protočne strukture
- Imaju 3 operanda i koriste 32 registra
- $0 – zero registar za efikasnije izvršenje
- Nedostatak indikatora za uslovna grananja (status bita)
- Memoriji pristupaju isključivo load/store, word (4-byte) aligned
- Samo jedno adresiranje podataka: bazni registar + 16 bit ofset
- Byte-adresiranje: pristup u memoriji može, ali nema u registrima
- Jump instrukcije: limitirane pseudo-adresom
- Slaba podrška za potprograme i nikakva za stack
- Slaba podrška za rukovanje prekidima
- Delayed branches i Load delay slot (MIPS I)

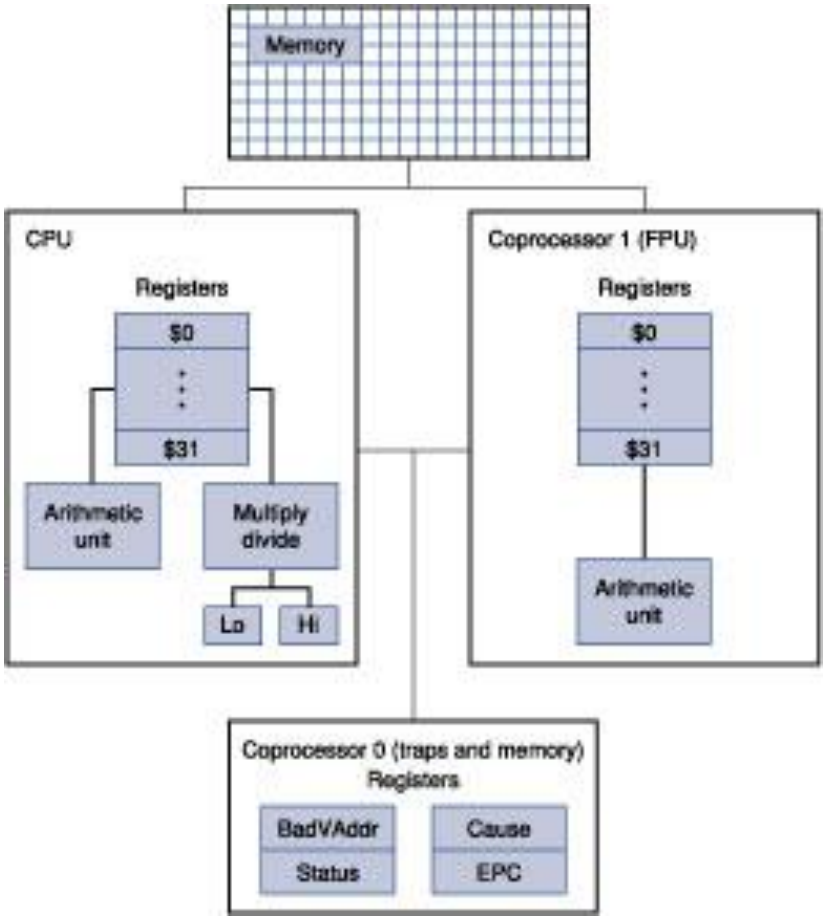
Zadržano grananje



Kašnjenje load instrukcije



Organizacija MIPS procesora



- CPU modul
  - Kernel i User mode
- Koprocesor 0
  - Kontrola procesora
  - mfc0, mtc0
- Koprocesor 1
  - Tekući zarez
- Memorijski prostor
  - 2^{30} reči
  - 2^{32} Byte

Skup registara

- $broj, $ime
- Postoje i dodatni registri, nevidljivi za programera
  - hi, lo

Broj	Ime	Opis
0	zero	Vrednost 0, nepromenljiva
1	$at	Rezervisano za asembler (*assembler temporary*)
2-3	$v0 - $v1	Smeštanje rezultata proračuna i funkcijskih poziva (*values*)
4-7	$a0 - $a3	Prenos argumenata ka potprogramima (*arguments*)
8-15	$t0 - $t7	Privremene vrednosti (*temporaries*)
16-23	$s0 - $s7	Sačuvane vrednosti (*saved values*)
24-25	$t8 - $t9	Privremene vrednosti (*temporaries*), nastavak prvih osam
26-27	$k0 - $k1	Rezervisano za jezgro OS (*kernel scratch*)
28	$gp	Pokazivač na 64KB segment statičkih podataka (*global pointer*)
29	$sp	Pokazivač na poslednju lokaciju "stack-a" (*stack pointer*)
30	$s8/$fp	$s8, ili pokazivač na okvir steka (*frame pointer*)
31	$a	Povratna adresa (*return address*)

FP registri i njihova namena

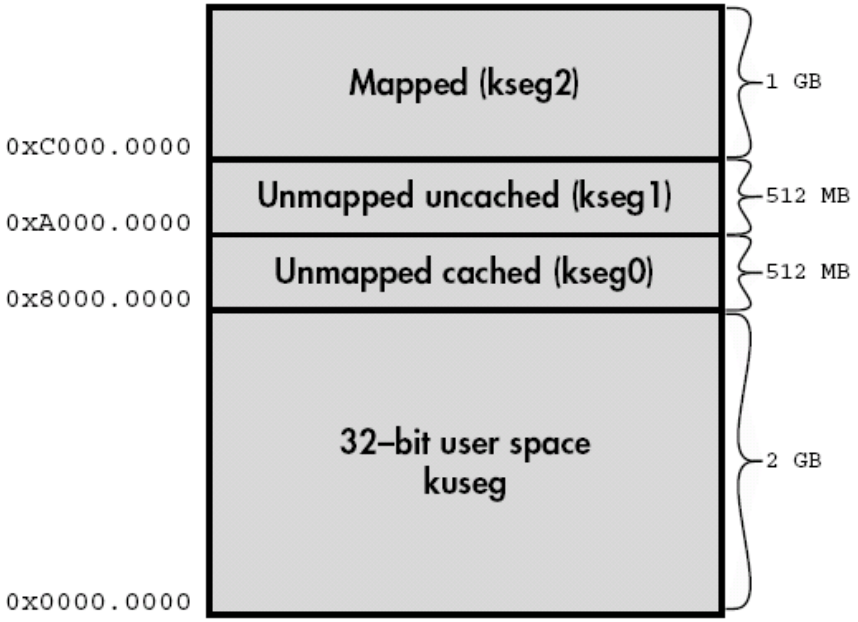
`$f0,$f2`	`fv0–fv1`	Value returned by function. `fv1` is used only for "complex" data type; it is not available in C.
`$f4,$f6, $f8,$f10`	`ft0–ft3`	Temporaries—subroutines can use without saving.
`$f12,$f14`	`fa0–fa1`	Function arguments.
`$f16,$f18`	`ft4–ft5`	Temporaries.
`$f20,$f22, $f24,$f26, $f28,$f30`	`fs0–fs5`	Register variables: A function that will write one of these must save the old value and restore it before it exits. The calling routine can rely on the value being preserved.

Memorijski prostor

- MIPS koristi virtuelne adrese (straničenje) koje MMU preslikava u realne
- Reči se adresiraju po oktetima, tj. adrese dve uzastopne reči razlikuju se za četiri
- Podela po segmentima, po pravu pristupa i korišćenju cache
  - user i kernel memorijski segmenti

Virtual address range	Physical address range	Name	Description
0xc000.0000 - 0xffff.ffff	0x0000.0000 - 0x7fff.ffff	kseg2	1024 MBytes mapped cached kernel segment
0xa000.0000 - 0xbfff.ffff 0xbfc0.0000	0x0000.0000 - 0x1fff.ffff 0x1fc0.0000	kseg1	512 MBytes unmapped uncached kernel segment. default reset address
0x8000.0000 - 0x9fff.ffff	0x0000.0000 - 0x1fff.ffff	kseg0	512 MBytes unmapped cached kernel segment.
0x0000.0000 - 0x7fff.ffff	0x4000.0000 - 0xbfff.ffff	kuseg	2048 MBytes user space, mapped and cached.

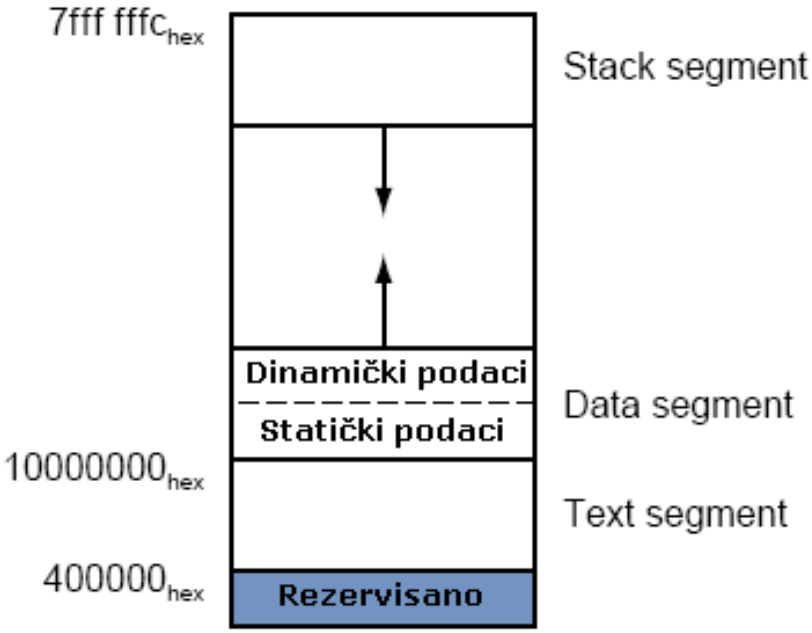
Memorijski prostor 2



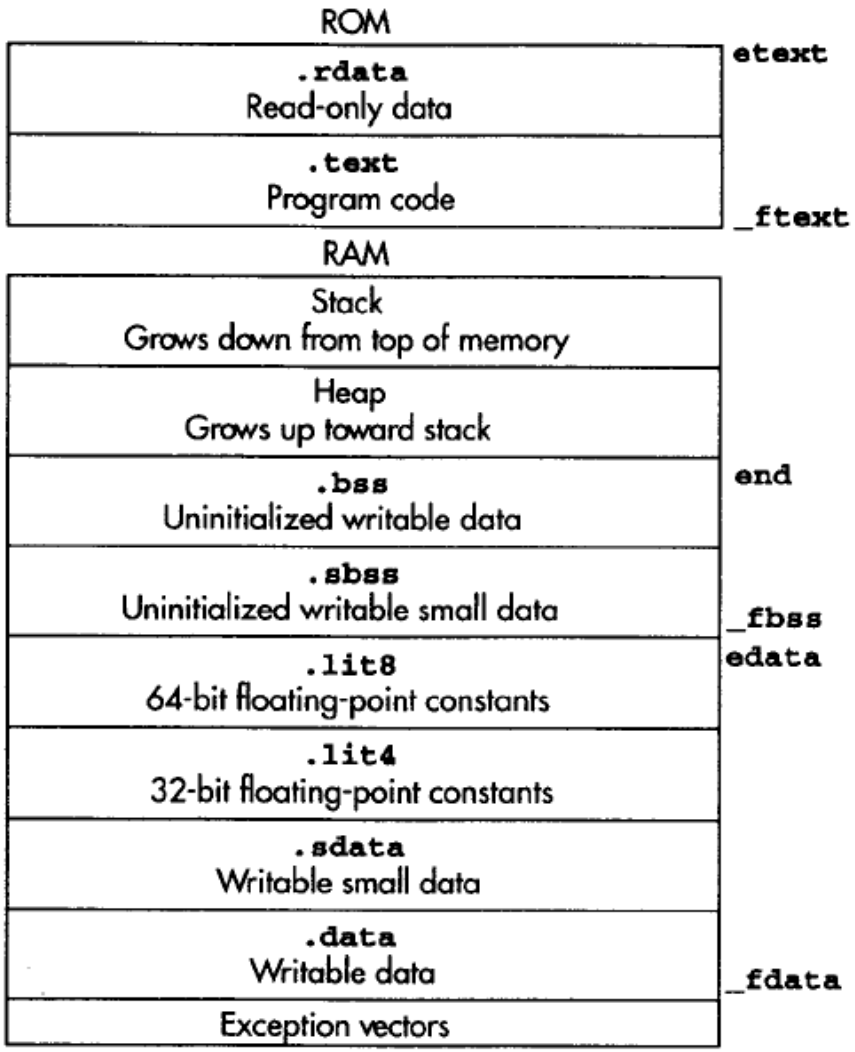
- *kuseg* – segment za korisničke programe
- *kseg0* – mapira se skidanjem MSB adrese u prvih 512M fizičke memorije. Koristi cache, koristi ga kernel.
- *kseg1* – skidanjem 3 MSB mapira se opet u donjih 512M. Ne koristi cache, zato su tu reset vektor+boot ROM, kao i IO registri.
- *kseg2* – segment u kom se izvršava kernel po inicijalizaciji MMU.
- Skromni ES sa manje od 512K koriste uglavnom kseg0 and kseg1, ostajući u kernel modu trajno.
- Stranice se preslikavaju preko HW tabele - TLB (*Translation Lookaside Buffer*)

Memorijski model korisničkog programa

- MIPS asembler prilikom prevođenja programa deli memoriju na tri segmenta
  - text - instrukcije programa
  - data - statički i dinamički podaci
  - stack – opadajući, bez instrukcija push i pop
- Segment “Rezervisano” koristi OS i nedostupan je korisniku.



Programski segmenti



- Standardni segmenti
  - Postoje varijacije kod raznih razvojnih alata
  - Ne moraju biti svi prisutni
  - Niti alocirani kao na slici (ROM)
- Eksplicitna definicija
  - .text, .data , .rdata
- Implicitna definicija
  - .bss
  - .lit4, .lit8 - FP konstante
  - .sdata, .sbss gp pristup

Primer 1: C kod

- C sekvenca

```
int a[10] = { 36, 20, 27, 15, 1, 72, 41, 12, 34, 17 };
int n = 10;
int max;
char txt[] = "Maksimalna vrednost niza je: “;

void main( void )
{
    for( i = 0; i < n; i++ )
    {
        if( a[i] <= max )
            max = a[i];
    }
}
```

Primer 1: Asemblerski kod

```
#-------------------------------------------------------------------------
#-- Odredjivanje maks.vrednosti niza upotrebom registara t0 - t9. Rezultat je 72
#-------------------------------------------------------------------------
.data                       # pocetak sekcije za podatke
a:      .word 36, 20, 27, 15, 1, 72, 41, 12, 34, 17
n:      .word 10
max:    .word 0
txt:    .asciiz  "Maksimalna vrednost niza je: "

.text                       # pocetak sekcije za kod
main:
        li   $t0, 0         # i (brojac) postavi u $t0, pocetna vrednost je nula
        li   $t1, 0         # max vrednost postavi u $t1, pocetna vrednost je nula
        lw   $t2, n         # n (broj elemenata niza) postavi u $t2
m1:
        bge  $t0, $t2, m3   # ako je i >= n skoci na m3 - kraj petlje
        mul  $t3, $t0, 4    # skaliraj i, pripremi za pristup memoriji
        lw   $t4, a($t3)    # citaj a[i] i postavi u $t4
        ble  $t4, $t1, m2   # ako je a[i] <= max skoci na m2
        move $t1, $t4       # u suprotnom postavi max na vrednost a[i]
m2:
        addi $t0, $t0, 1    # i++, uvacaj brojac za jedan
        b m1                # vrati se na pocetak petlje
m3:
        sw max, $t1         # napuni max sa izracunatom vrednoscu
        li   $v0, 10        # postavi kod sistemskog poziva, 10 - exit
        syscall             # instrukcija sistemskog poziva
```

Pozivi potprograma - konvencije

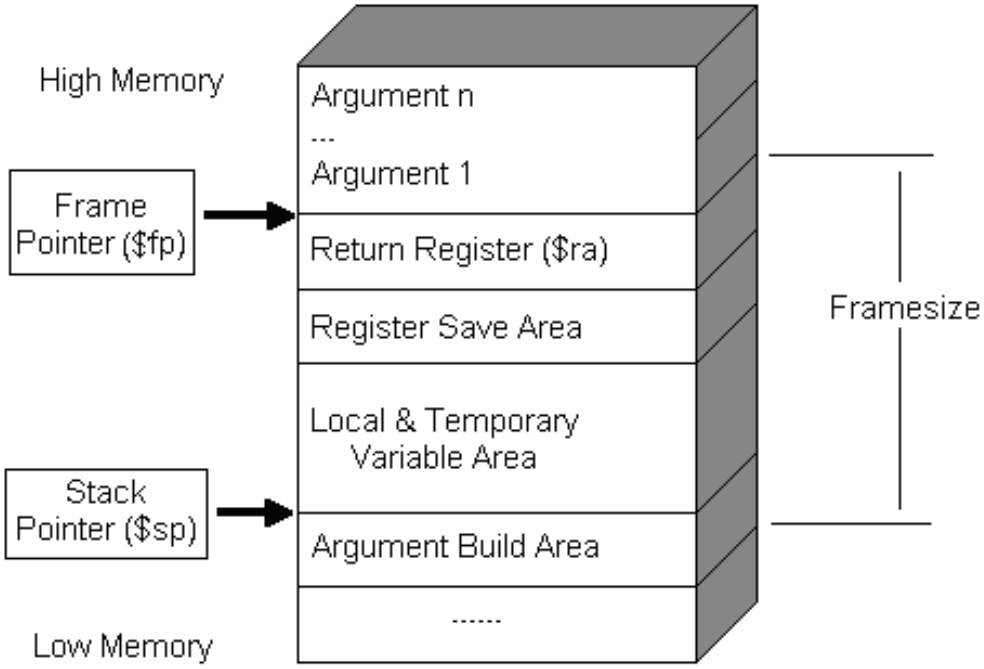
- `jal labela` `# poziv`
- `jr ra ($31)` `# povratak`

Registri	Namena
$at ($1), $k0 ($26), $k1 ($27)	Registri rezervisani za OS i asembler
$a0 - $a3 ($4 - $7)	Prenos prva 4 argumenta potprograma, ostali se prenose preko stack-a
$v0 i $v1 ($2 i $3)	Povratna vrednost potprograma
$t0 - $t9 ($8 - $15, $24 i $25)	Privremene vrednosti koje ne moraju biti sačuvane u okviru poziva
$s0 - $s7 ($16-$23)	Vrednosti koje se moraju čuvati tokom poziva
$gp ($28)	Globalni pokazivač -> sredinu bloka od 64K memorije
$sp ($29)	Pokazivač na stack
$fp ($30)	Pokazivač na okvir (engl. frame)
$ra ($31)	Povratna adresa pri izvrsenju ***jal*** instrukcije

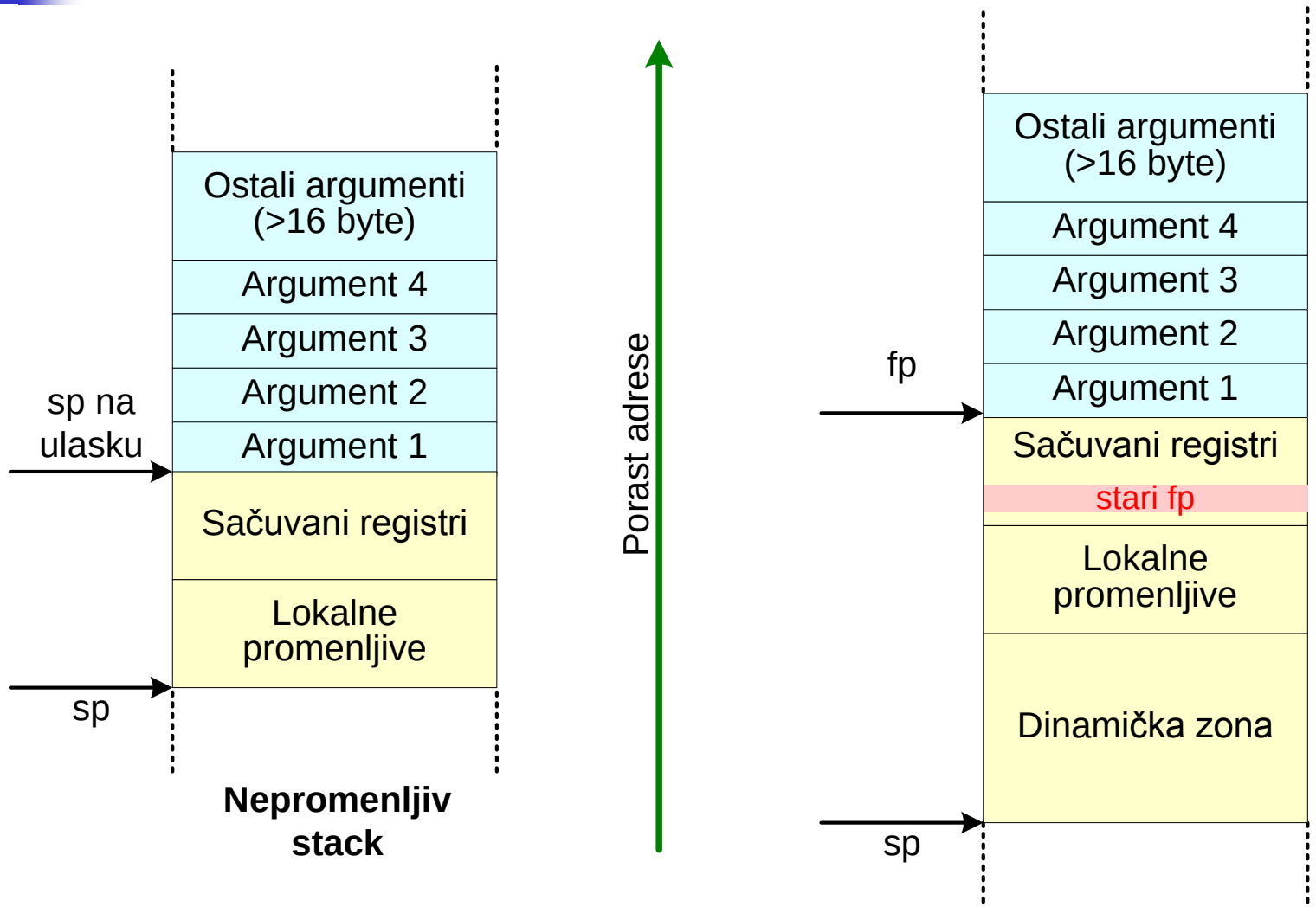
Struktura argumenata na stack-u

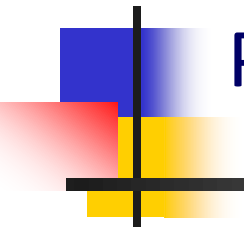
- Prva 4 argumenta se prenose preko a-registara
  - Ipak se rezerviše prostor na stacku, radi čuvanje ako zatreba pozvanoj f-i
- Dodatni argumenti se smeštaju samo na stack (arg5, 6, ...)

- LEAF funkcije
  - ne pozivaju nove funkcije
  - mogu se direktno vratiti sa $ra, i koristiti t, a i v registre
- NONLEAF funkcije
  - pozivaju druge funkcije
  - pun *stack-frame* format sa fp registrom



Stack-frame leaf i non-leaf funkcija

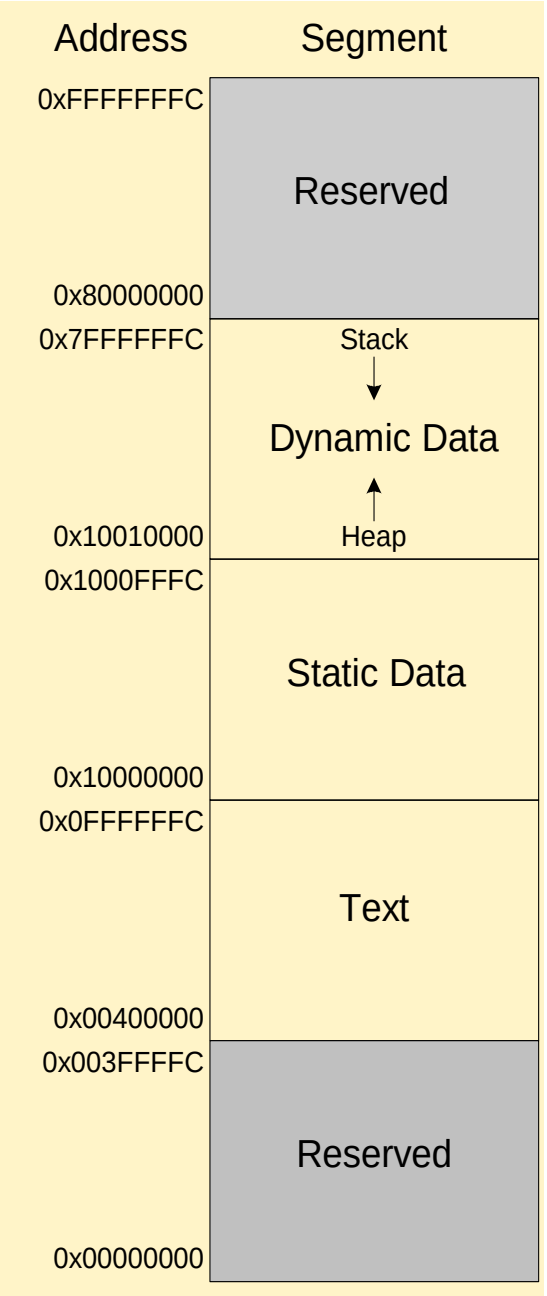




Primer: smeštanje C programa u memoriju

MIPS Memorijska Mapa

- Text segment:
  - sadrži instrukcije (skoro 256MB)
- Static and global data segment
  - prostor za globalne promenljive - 64KB
- Dynamic data segment čuva stack i heap
  - Dinamička alokacija
- Reserved segment
  - koristi operativni sistem



Example Program

```
int f, g, y;  // global variables


int main(void)
{
  f = 2;    //
  g = 3;    //

  y = sum(f, g);
  return y;
}

int sum(int a, int b) {
  return (a + b);
}
```


```
.data
f:
g:
y:

.text
main:
  addi $sp, $sp, -4    # stack frame
  sw   $ra, 0($sp)     # store $ra
  addi $a0, $0, 2      # $a0 = 2
  sw   $a0, f          # f = 2
  addi $a1, $0, 3      # $a1 = 3
  sw   $a1, g          # g = 3
  jal  sum             # call sum
  sw   $v0, y          # y = sum()
  lw   $ra, 0($sp)     # restore $ra
  addi $sp, $sp, 4     # restore $sp
  jr   $ra             # return to OS
sum:
  add  $v0, $a0, $a1   # $v0 = a + b
  jr   $ra             # return
```

Symbol Table

- tabela simbola za program iz primera

Symbol	Address
f	0x10000000
g	0x10000004
y	0x10000008
main	0x00400000
sum	0x0040002C

Izvršni kod programa iz primera

Executable file header	Text Size	Data Size
	0x34 (52 bytes)	0xC (12 bytes)
Text segment	**Address**	**Instruction**
	0x00400000	0x23BDFFFC
	0x00400004	0xAFBF0000
	0x00400008	0x20040002
	0x0040000C	0xAF848000
	0x00400010	0x20050003
	0x00400014	0xAF858004
	0x00400018	0x0C10000B
	0x0040001C	0xAF828008
	0x00400020	0x8FBF0000
	0x00400024	0x23BD0004
	0x00400028	0x03E00008
	0x0040002C	0x00851020
	0x00400030	0x03E0008
Data segment	**Address**	**Data**
	0x10000000	f
	0x10000004	g
	0x10000008	y

```
addi $sp, $sp, -4
sw   $ra, 0 ($sp)
addi $a0, $0, 2
sw   $a0, 0x8000 ($gp)
addi $a1, $0, 3
sw   $a1, 0x8004 ($gp)
jal  0x0040002C
sw   $v0, 0x8008 ($gp)
lw   $ra, 0 ($sp)
addi $sp, $sp, -4
jr   $ra
add  $v0, $a0, $a1
jr   $ra
```

Example Program: In Memory