VTŠ: Osnovi računarske tehnike

Centralna upravljačka jedinica

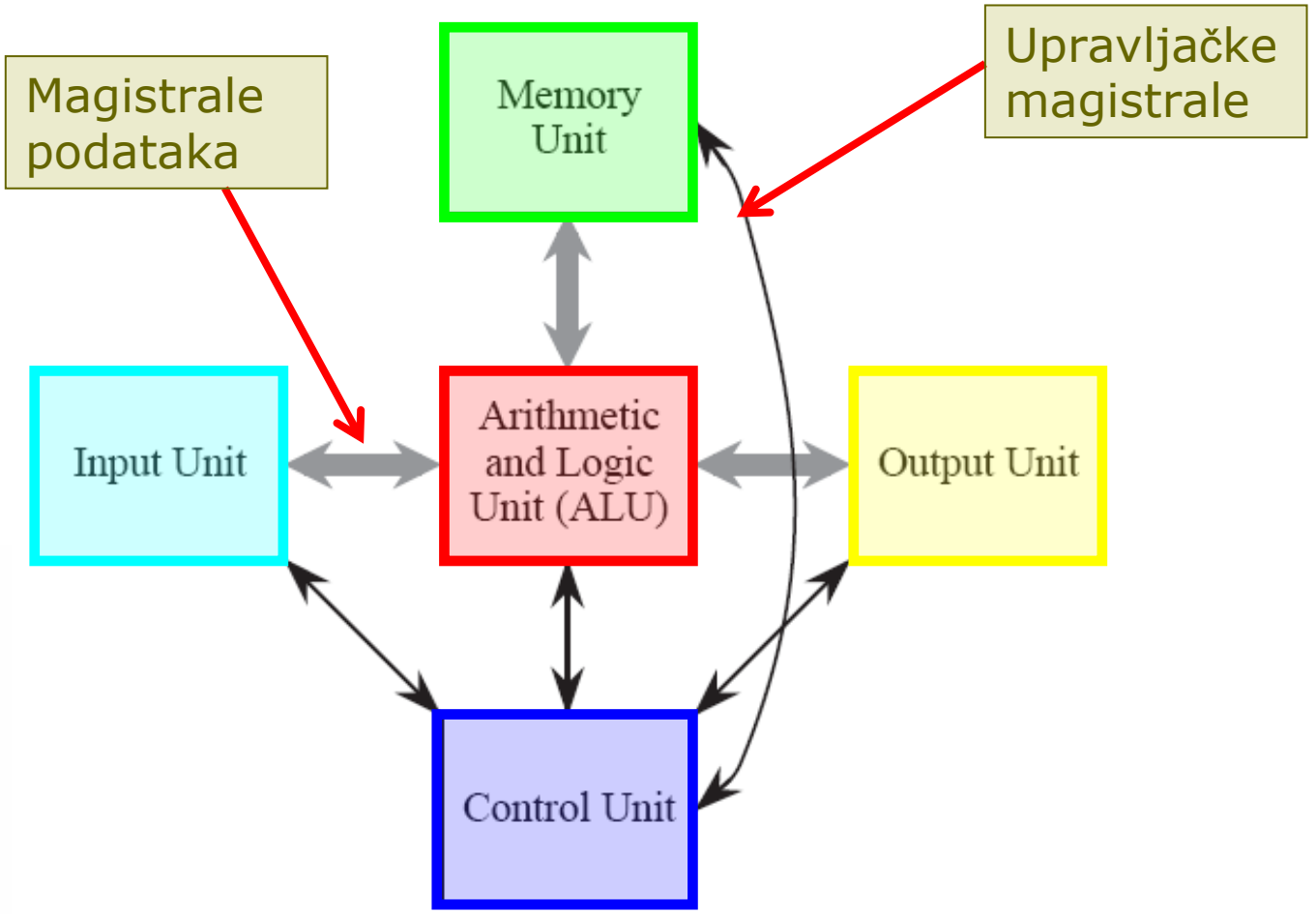
CPU

mr. Veličković Zoran, dipl.inž

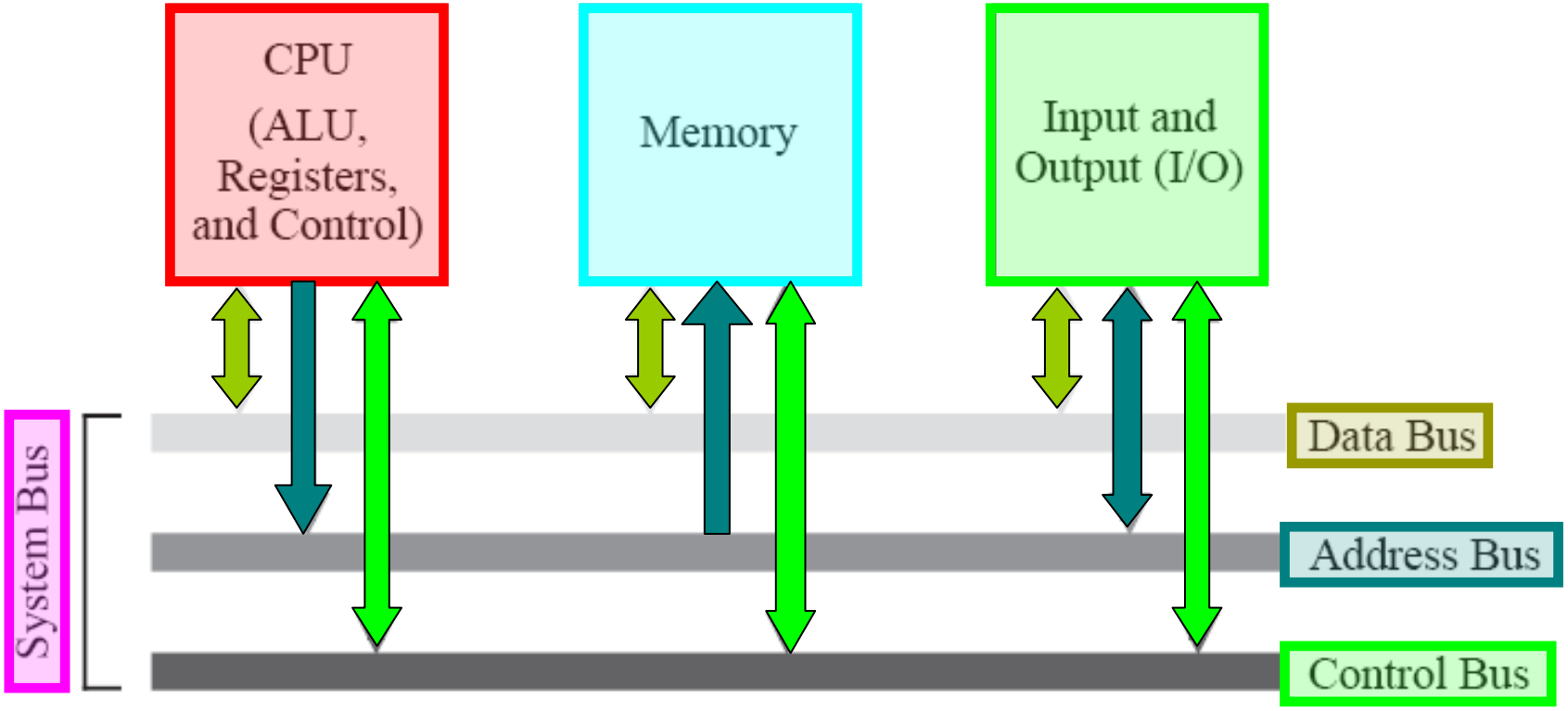
Maj, 2010.

Von Neumann-ov model

- Von Neumann-ov model digitalnog računara

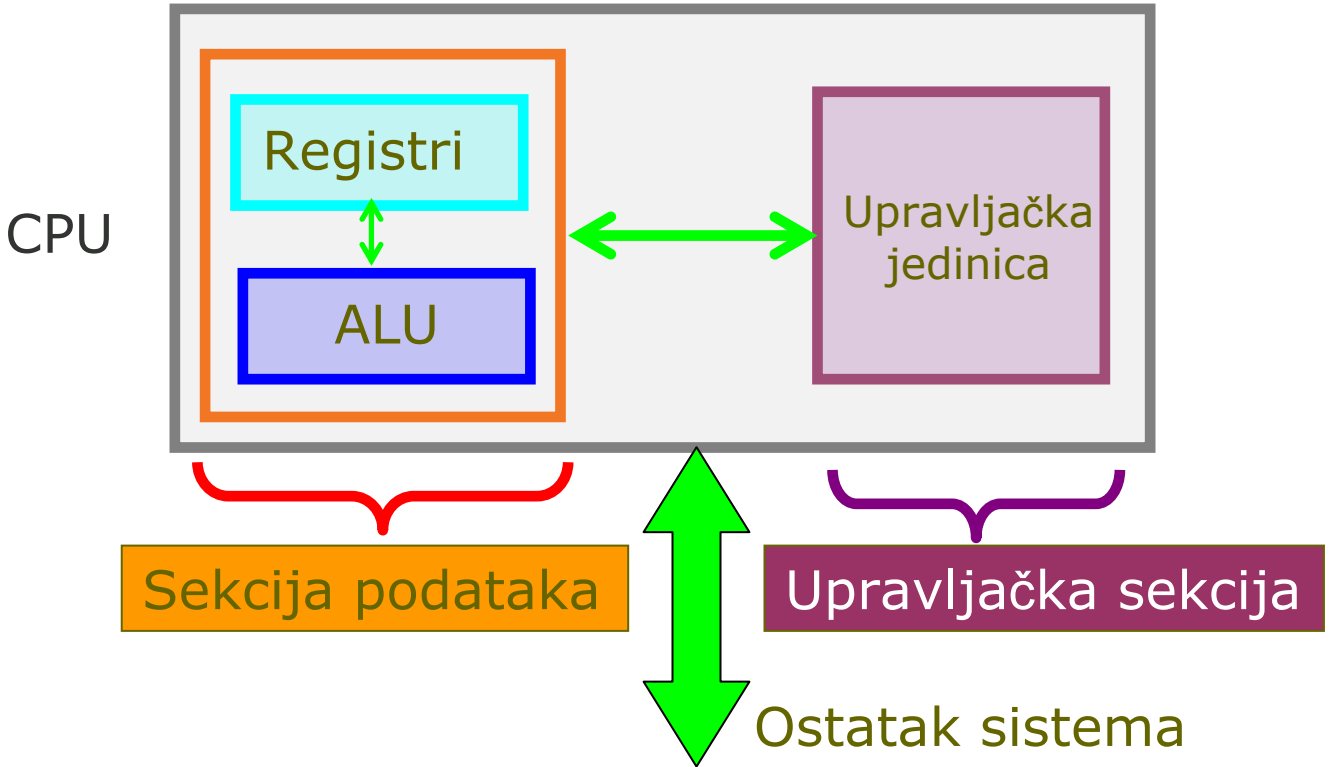


System Bus Model (1)



CPU (1)

- Minimalno se CPU sastoji od sekcije podataka (data section) koja se sastoji od *registara* i *ALU*, i upravljačke sekcije (control section), koja interpretira instrukcije i definiše transfere podataka iz/u registara.



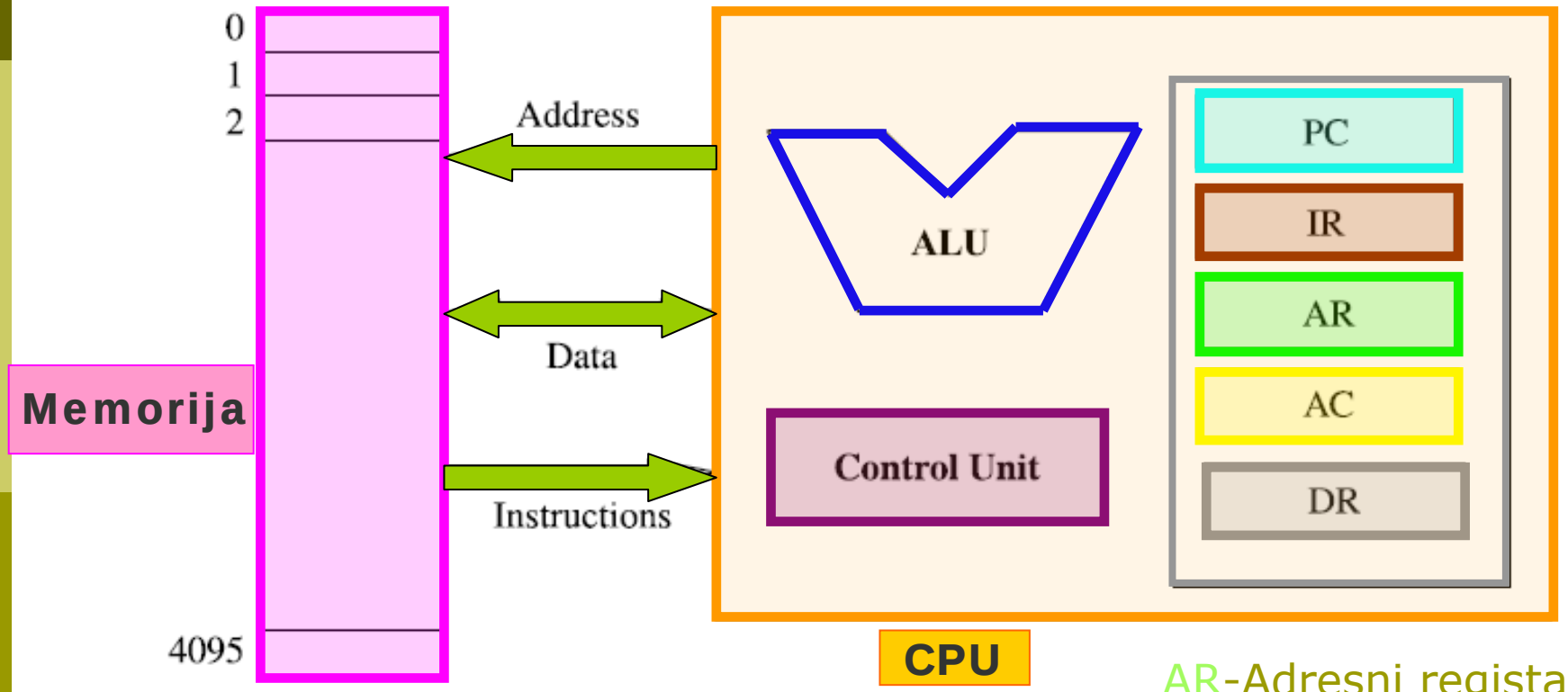
Algoritmi i CPU

- Niz koraka primenjenih za rešavanje nekog problema naziva se **ALGORITAM**.
- Pogledajmo algoritam za sabiranje dva broja A=B+C:
  - Smesti prvi argument zbira (A) u registar r1 koji čuva prvu ulazni parametar u ALU: **r1←B**
  - Smesti drugi argument zbira (B) u registra r2 koji čuva drugi ulazni parametar u ALU: **r2←C**
  - Dovedi odgovarajući kod operacije sabiranja na ALU: **A=B+C**
  - Smesti rezultat (C) u odgovarajući registar r3: **r3←C**
- Posle ovih koraka rezultat operacije sabiranja se nalazi u **odgovarajućem** registru.
- Algoritmi se pamte u memoriji a svaki korak se **kodira programskim instrukcujama** nekog programskog jezika.

CPU registri (1)

- Upravljačka jedinica računara je **odgovorna** za izvršenje programskih instrukcija koje se nalaze u glavnoj memoriji.
- Pretpostavljeno je da se (mašinski) **kod** interpretira pomoću upravljačke jedinice - jedna po jedna instrukcija, iako savremeni procesori mogu procesirati nekoliko instrukcija istovremeno!
- Dva registra koja formiraju interfejs između upravljačke jedinice i jedinice podataka su ***Programski Brojač* - PC** (program counter) i ***Instrukcijski Registar* - IR** (instruction register).
- *PC registar sadrži* **memorijsku adresu** *instrukcije koja će biti izvršena.*
- Instrukcija na koju pokazuje **PC** se **puni** (fetch) iz memorije i smešta u **IR** gde se **interpretira**.

CPU registri (2)



AR-Adresni registar

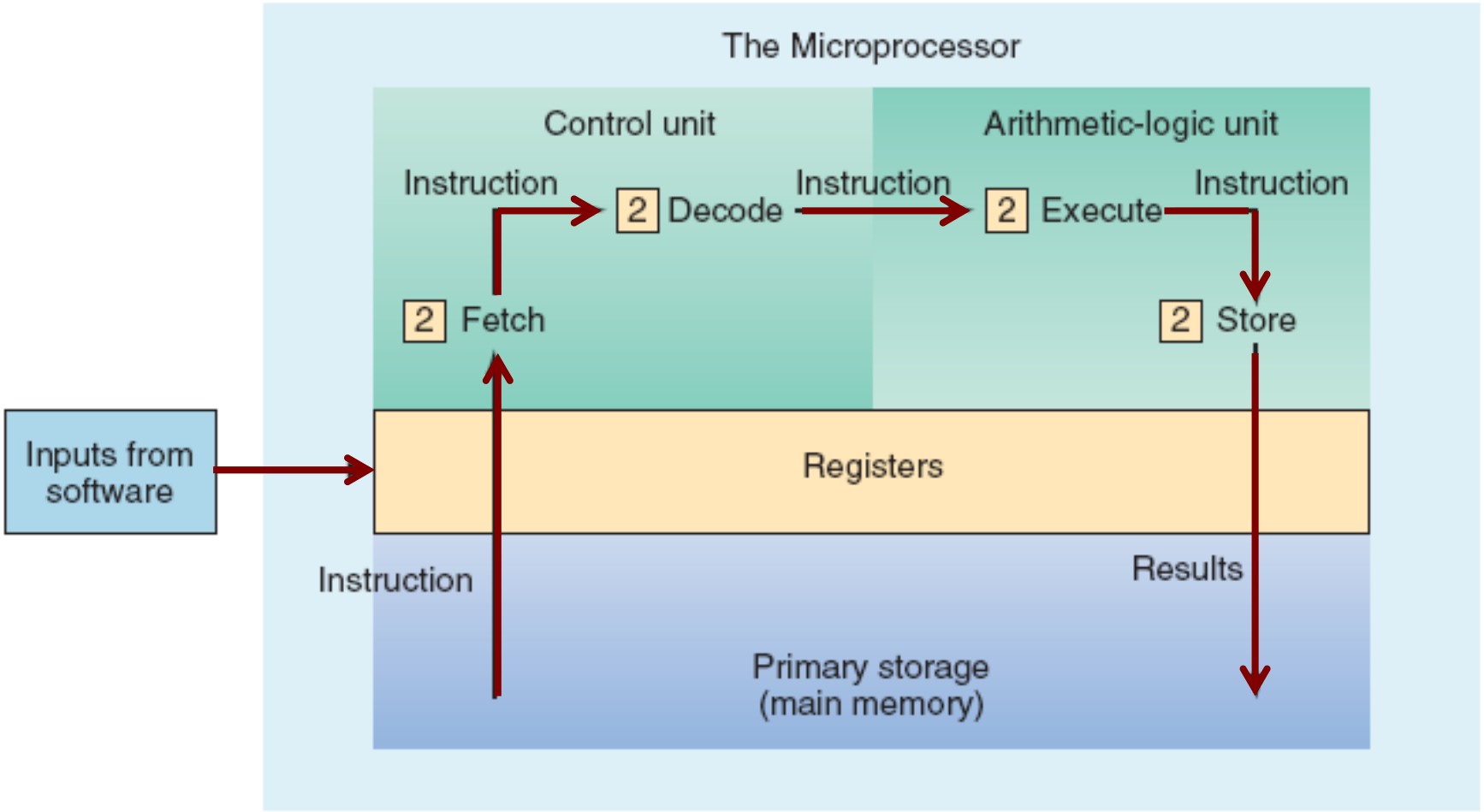
AC-Akumulator

DR-Data registar

CPU fetch-execute ciklus

- Koraci koje treba da izvši upravljačka jedinica pri izvršenju instrukcije su:
  1. Puni sledeću instrukciju koja će biti izvršena iz memorije,
  2. Dekodira kod operacije (op kod),
  3. Čita operande iz glavne memorije (ako ih ima),
  4. Izvrši instrukciju i smesti rezultat,
  5. Skoči na korak 1.
- Niz ovih koraka naziva se **fetch-execute** ciklus.
- Upravljačka jedinica je odgovorna za **koordinaciju** ovih koraka pri izvršavanju računarskog programa.
- "Računar unutar računara" je dobra asocijacija za ovaj proces.

Grafički prikaz: fetch-execute ciklus



CPU magistrale i registri

- CPU poseduje **sopstvenu** (unutrašnju) magistralu podataka koja povezuje skup registara (*register file*) i *arihmetičko-logičku jedinicu (ALU).*
- Registar fajl se može smatrati malom, brzom memorijom odvojenom od sistemske memorije, koja služi za **privremeno** smeštanje podataka u vreme izvršavanja instrukcija.
- Tipična veličina registarskog fajla se kreće od nekoliko registara pa do nekoliko hiljada registara!
- Kao i sistemska memorija, svaki registar u registar fajlu je označen svojom **adresom** koje stratuju od nule.

CPU registri i memorija

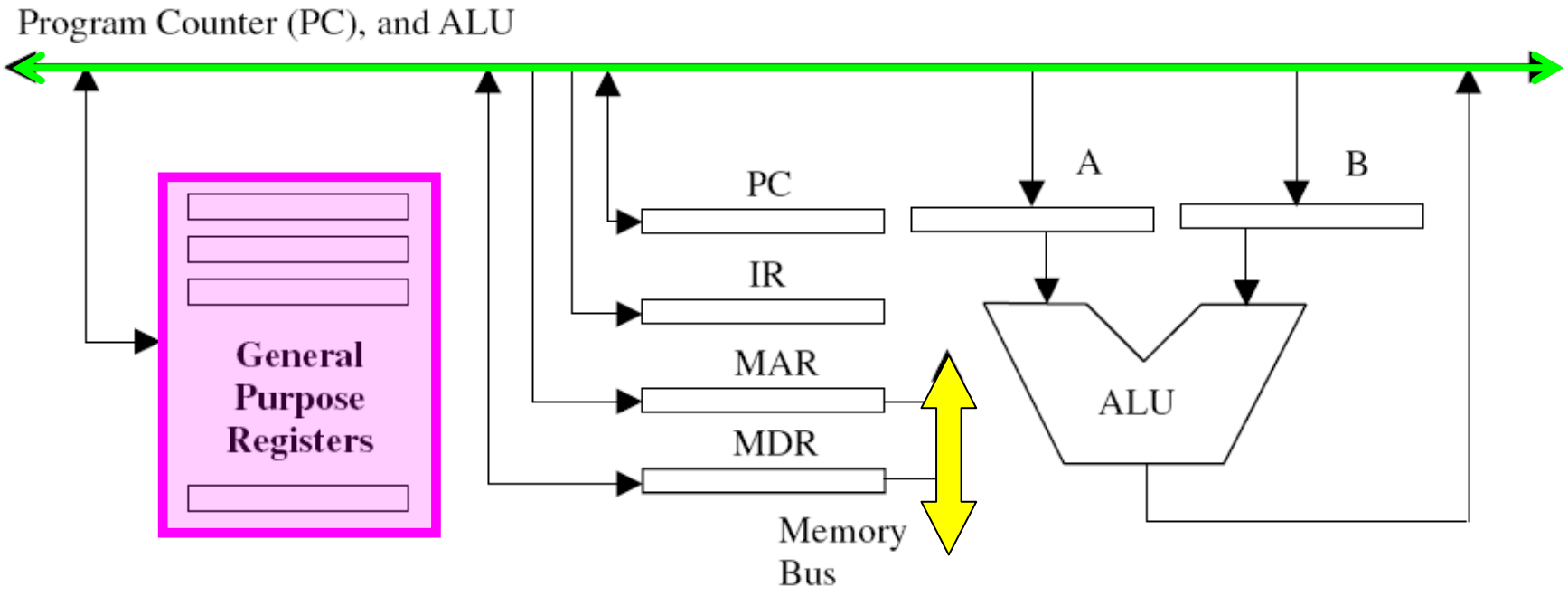
- Registarske adrese su značajno **manje** u odnosu na adrese glavne memorije.
- Register fajl koji se sastojati od **32 registra** - poseduju samo **5-bitnu** addresu.
- Glavna razlika između **registarskog fajla** i **sistemske memorije** je da se registar fajl nalazi **unutar** CPU-a.
- Ova činjenica registarski fajl čini **veoma brzim**.
- Instrukcije koje se obavljaju nad podacima iz registarskog fajla su **deset puta brže** nego iste instrukcije koje se izvršavaju sa podacima iz glavne memorije.

Dodatne magistrale u CPU

- Ovo je razlog zašto su programi koji koriste registarski fajl **znatno brži** nego ekvivalentni programi zasnovani na memorijskim instrukcijama (iako su često realizaovani sa većim brojem instrukcija).
- Unutar samog CPU-a postoji **nekoliko** magistrala unutar same magistrale podataka.
- Ove magistrale povezuju puteve podataka unutara CPU-a sa sistemskom magistralom.
- Ovo omogućava transfer podataka **u(i)** glavnu memoriju **i(u)** registarski fajl.
- **Tri** dodatne magistrale povezuju registar fajl sa ALU.
- Na ovaj način se **istovremeno** pune operandi iz registarskog fajla, izvršava istrukcija sa operandom u ALU a rezultat smešta u registar fajl.

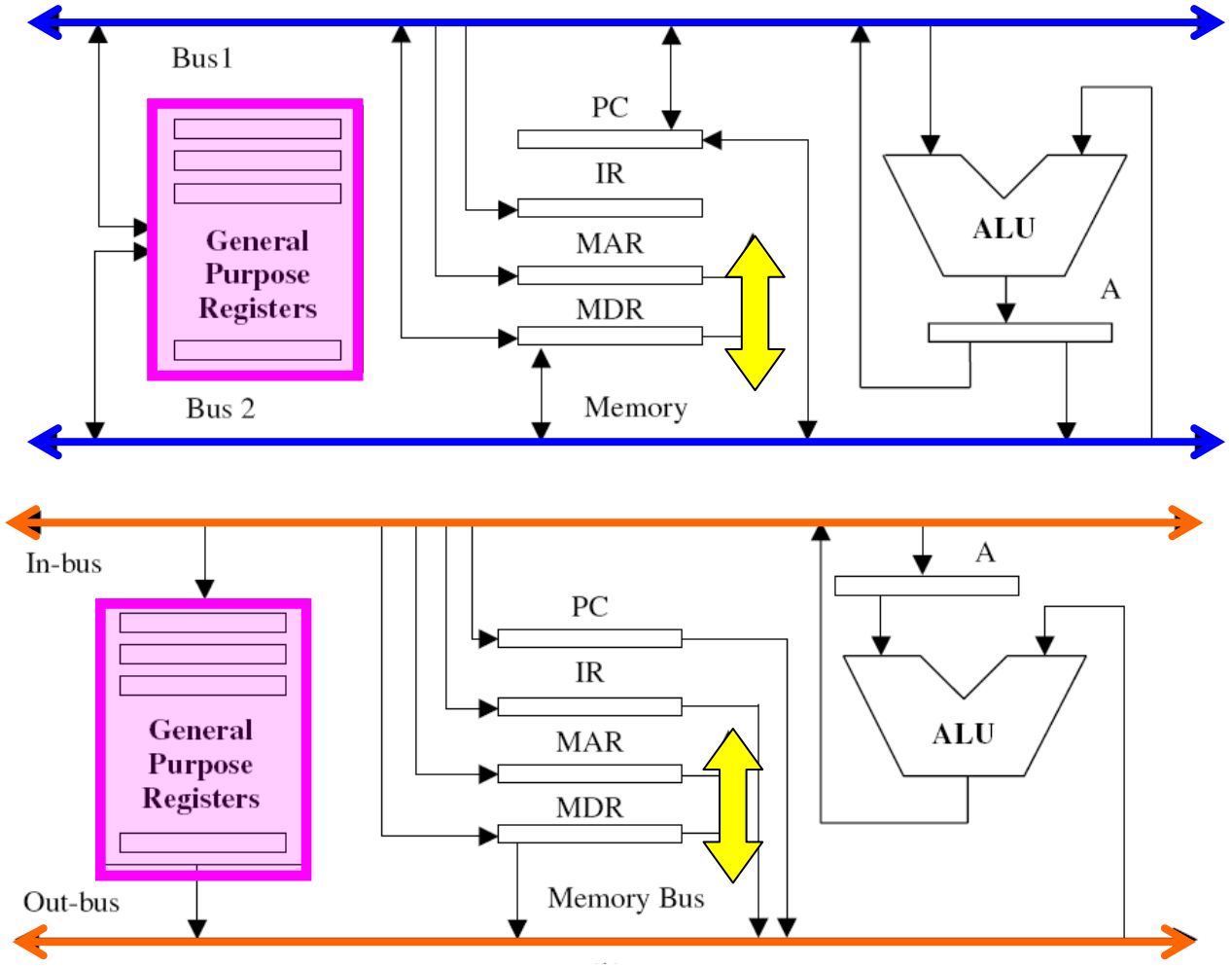
CPU BUS organizacija (1)

- Organizacija oko jedne magistrale



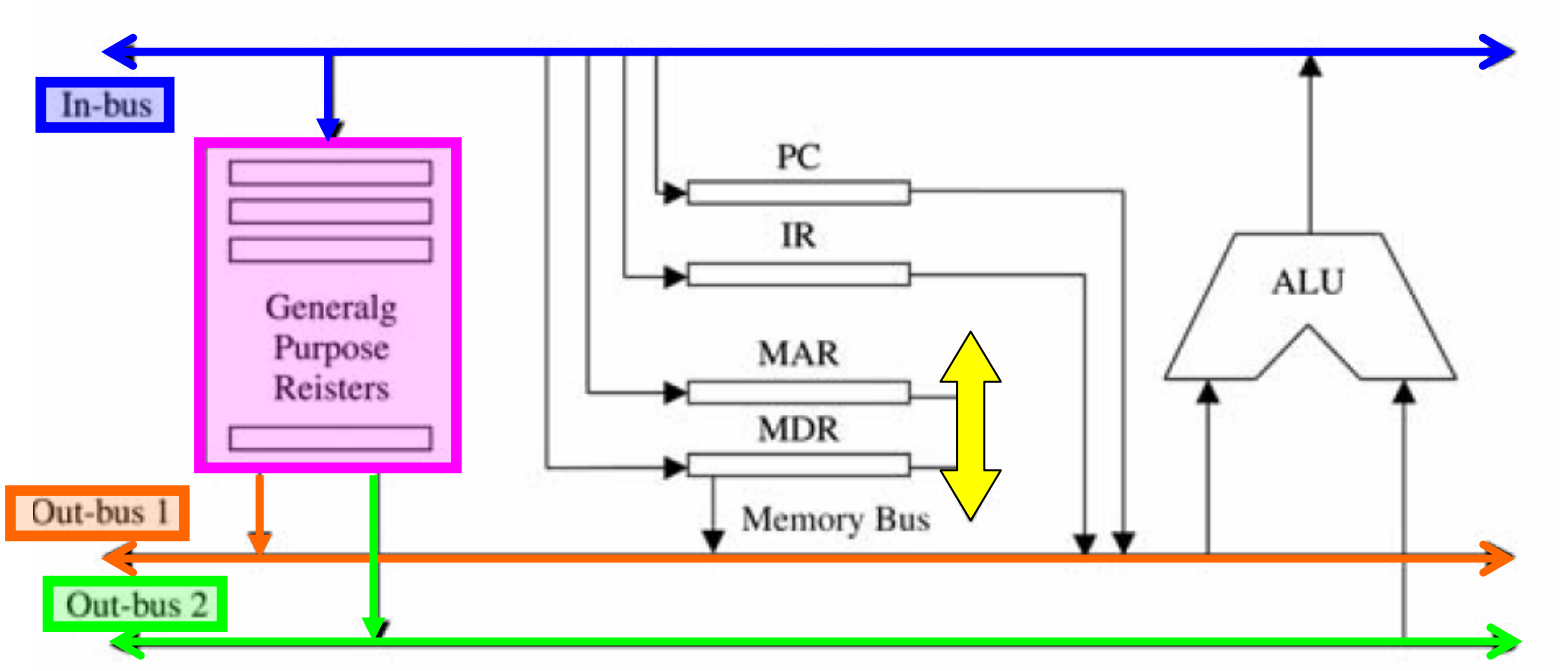
CPU BUS organizacija (2)

- Organizacija oko dve magistrale (dve verzije)



CPU BUS organizacija (3)

- Organizacija oko tri magistrale



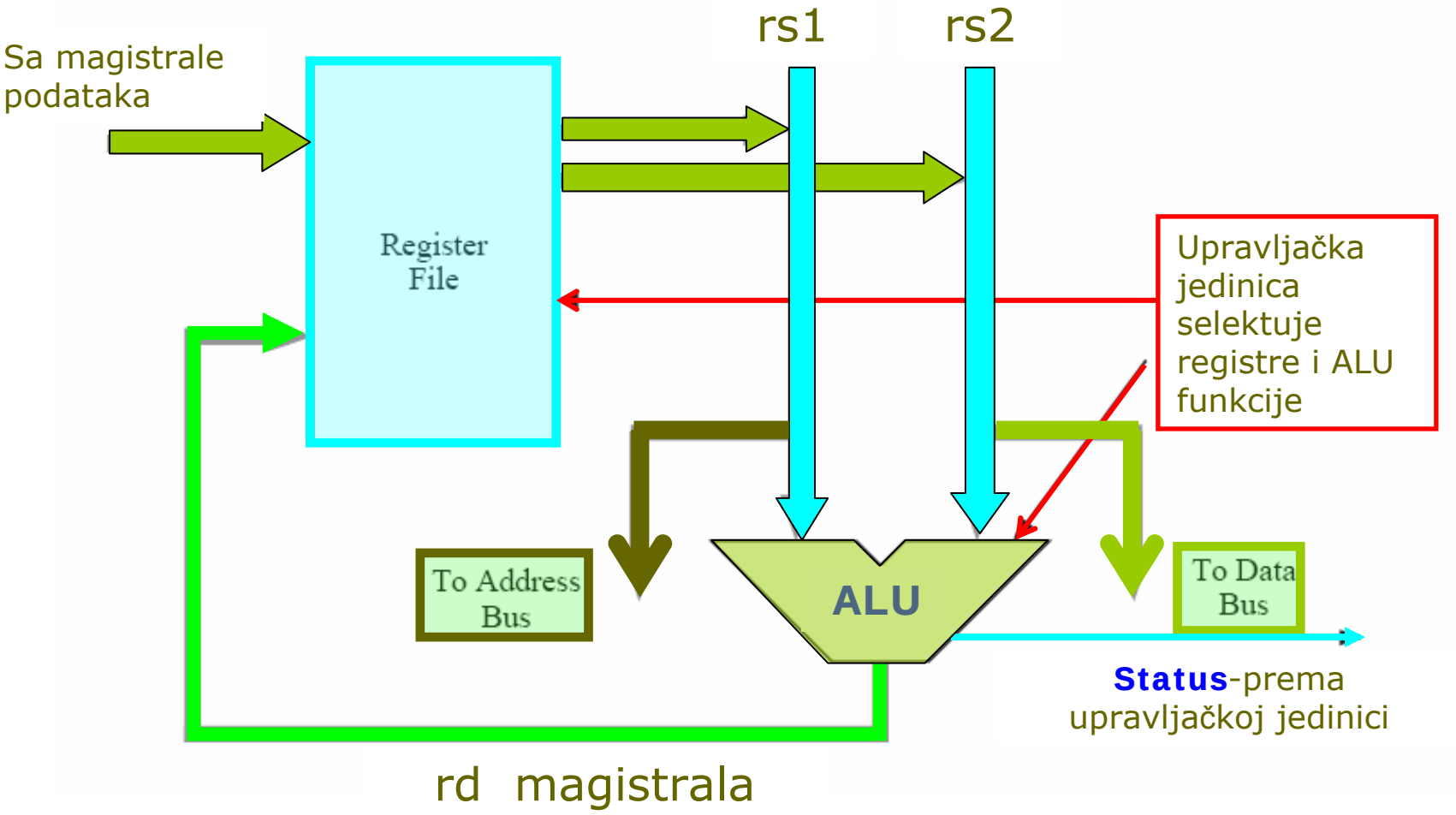
ALU operacije (1)

- ALU implementira *binarne* i *unarne* operacije.
- Primeri ovih operacija su: sabiranje (**add**-bin), invertovanje (**not**-un), ili (**or**-bin) i množenje (**multiplay**-bin)
- Operacije i operandi koji se koriste u izračunavanju se **selektuju** upravljačkom jedinicom.
- Dva operanda (podataka) se pune iz registarskog fajla magistralom "Register Source 1 - **rs1**" i "Register Source 2 - **rs2**".
- Izlaz iz ALU se postavlja na magistralu označenu "Register Destination - **rd**," i rezultat se vraća nazad u registarski fajl.

ALU operacije (2)

- U mnogim sistemima ove veze obuhvataju veze prema sistemskoj magistrali tako da memorija i periferije mogu biti adresirane.
- To je označeno “From Data Bus”, “To Data Bus”, i “To Address Bus.”

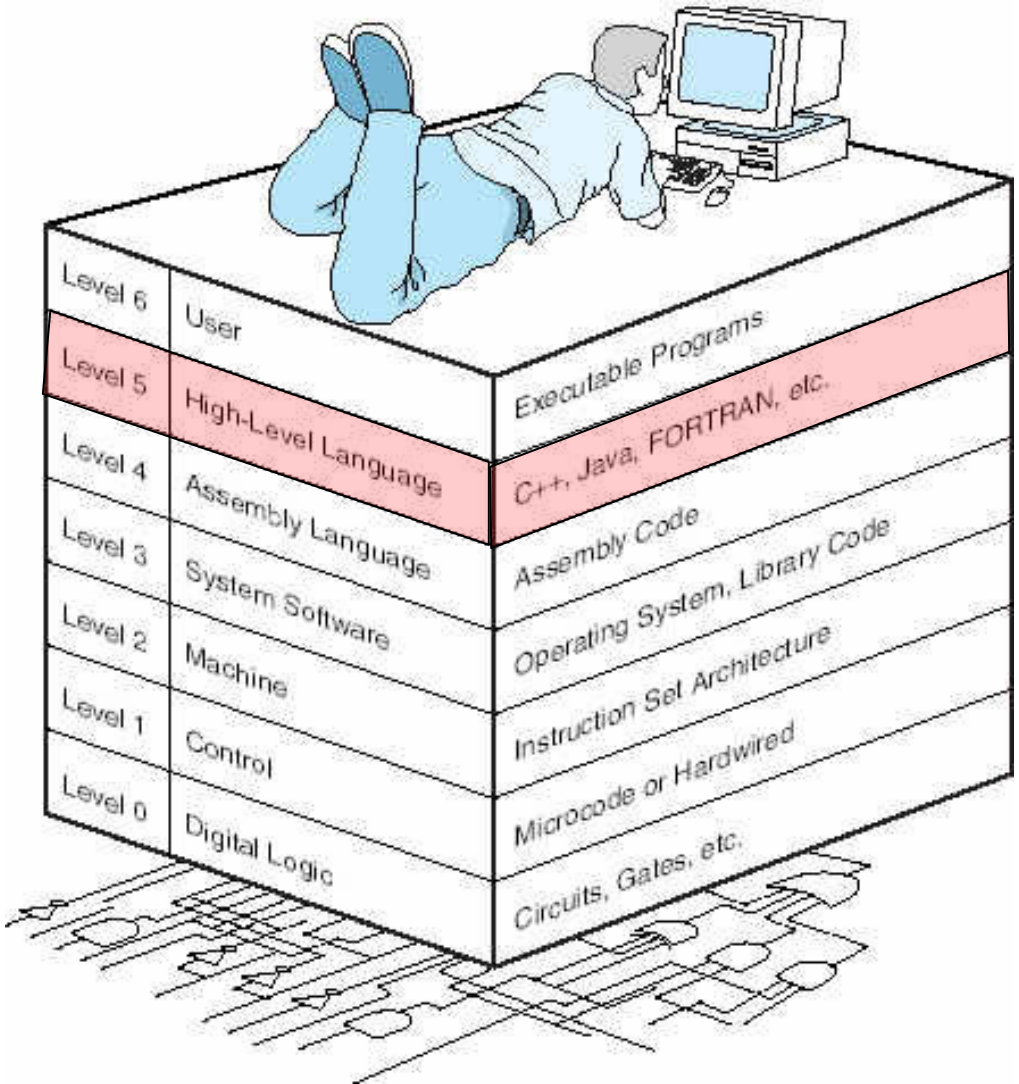
CPU magistrale podataka



CPU i instrukcijski set

- **Skup instrukcija** (instruction set) predstavlja kolekciju instrukcija koje procesor može izvršiti i on (skup instrukcija) zapravo određuje procesor.
- **Skup instrukcija** se veoma razlikuje od procesora do procesora.
- Razlika se odnosi na **veličinu instrukcija**, **tipove operanda** i **tipove izlaznih podatka**.
- Nekompatibilnost instruction set-a je u **potpunoj suprotnosti** sa kompatibilnišću viših programskih jezika kao što su C, Pascal, Ada, C#, Java!?
- Programi napisani u višem programskom jeziku mogu se izvršavati skoro na svim procesorima ali se moraju prethodno **prekompilovati** za odredišni procesor.

CPU i viši programski jezici



Izvorni kod p.j. JAVA

Rad sa virtuelnim promenljivama!

```
abstract class A {
  abstract void callme();
  // Konkretne metode su dozvoljene u apstraktnim klasama
  void callmetoo() {
    System.out.println("This is a concrete method.");
  }
}
class B extends A {
  void callme() {
    System.out.println("B's implementation of callme.");
  } }
class AbstractDemo {
  public static void main(String args[]) {
    B b = new B();    b.callme();    b.callmetoo();
  } }
```

Izvorni kod p.j. ASEMBLER

Direktan rad sa CPU registrima!

```
        LD X            \ AC ← X
        MOVAC           \ DR ← AC
        LD Y            \ AC ← Y
        ADD             \ AC ← AC + DR
        ST Z            \ Z ← AC
        STOP
X       W       350     \ reserve a word initialized to 350
Y       W       96      \ reserve a word initialized to 96
Z       W       0       \ result stored here
```

Primer seta ASM instrukcija CPU-a

	Mnemonic	Meaning
Memory	ld	Load a register from memory
	st	Store a register into memory
Logic	sethi	Load the 22 most significant bits of a register
	andcc	Bitwise logical AND
	orcc	Bitwise logical OR
	orncc	Bitwise logical NOR
	srl	Shift right (logical)
Arithmetic	addcc	Add
Control	call	Call subroutine
	jmpl	Jump and link (return from subroutine call)
	be	Branch if equal
	bneg	Branch if negative
	bcs	Branch on carry
	bvs	Branch on overflow
	ba	Branch always

Format ASM instrukcija (1)

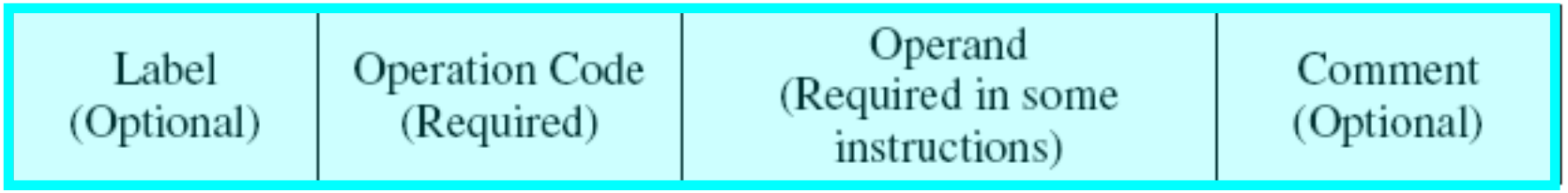
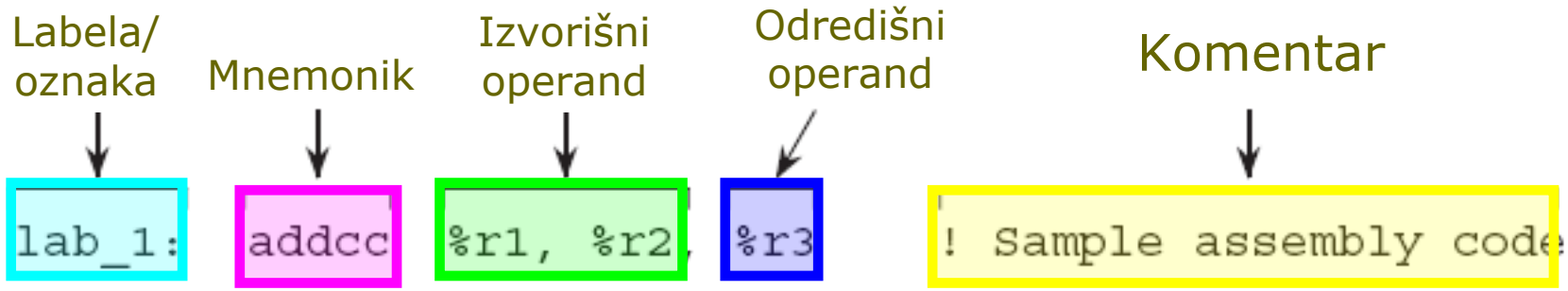


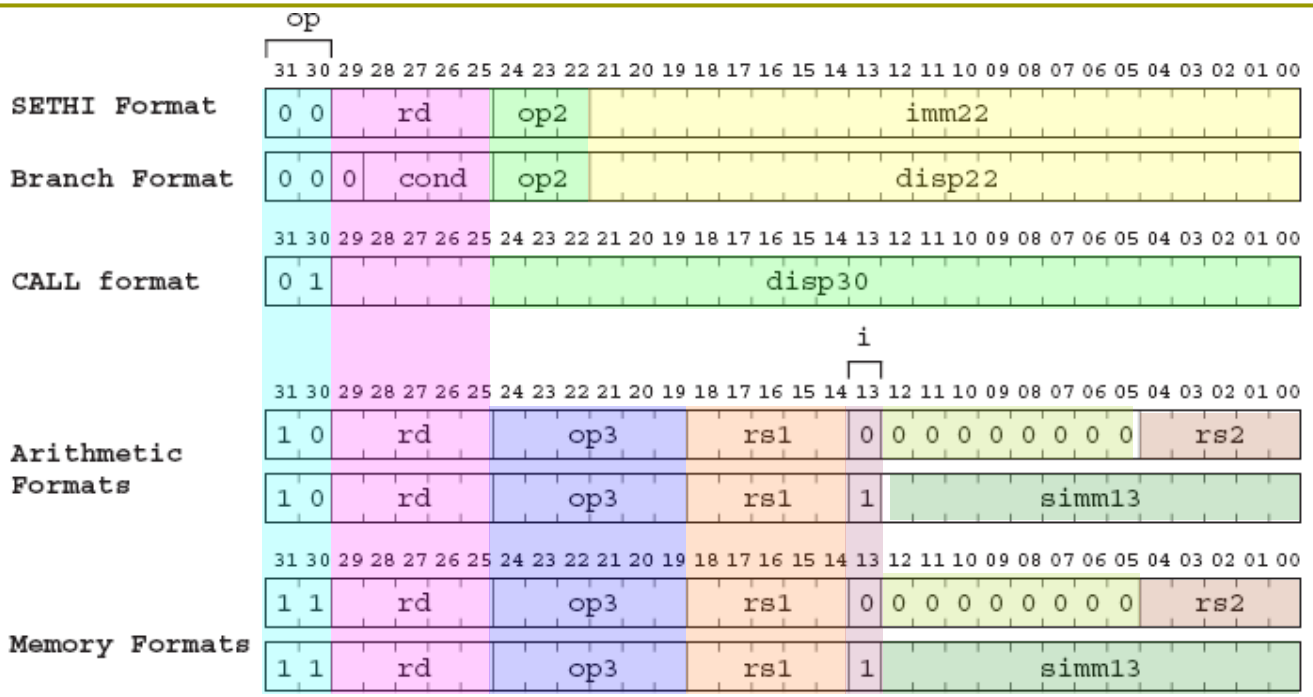
Figure 3.2 Assembly instruction format




```
START    LD X     \ copy the contents of location X into AC

BRA START     \ go to the statement with label START
```

Format ASM instrukcija (2)



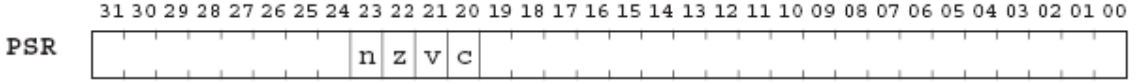
op	Format
00	SETHI/Branch
01	CALL
10	Arithmetic
11	Memory

op2	Inst.
010	branch
100	sethi

op3 (op=10)	
010000	addcc
010001	andcc
010010	orcc
010110	orncc
100110	srl
111000	jmpl

op3 (op=11)	
000000	ld
000100	st

cond	branch
0001	be
0101	bcs
0110	bneg
0111	bvs
1000	ba

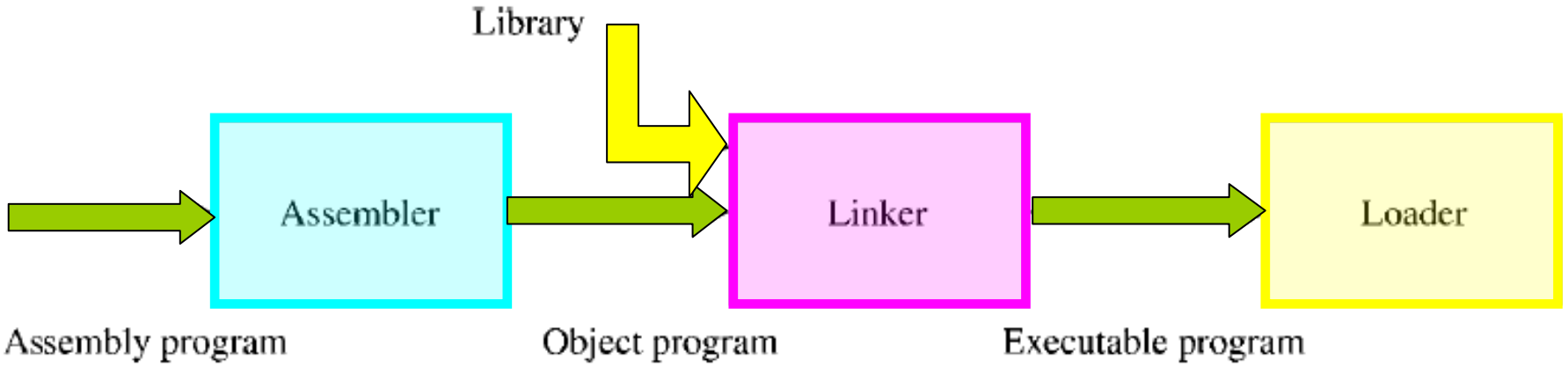


ASEMBLER (1)

TABLE 3.4 Assembly Language for the Simple Processor

Mnemonic	Operand	Meaning of instruction
STOP		Stop execution
LD	*x*	Load operand from memory (location x) into AC
ST	*x*	Store contents of AC in memory (location x)
MOVAC		Copy the contents AC to DR
MOV		Copy the contents of DR to AC
ADD		Add DR to AC
SUB		Subtract DR from AC
AND		And bitwise DR to AC
NOT		Complement contents of AC
BRA	*adr*	Jump to instruction with address *adr*
BZ	*adr*	Jump to instruction *adr* if AC = 0

Proces formiranja izvršnog koda



Mašinski jezik (1)

TABLE 3.2 Simple Machine Language Program in Binary (Example 1)

Memory location (bytes)	Binary instruction	Description
0000 0000 0000	0001 0000 0000 1100	Load the contents of location 12 in AC
0000 0000 0010	0011 0000 0000 0000	Move contents of AC to DR
0000 0000 0100	0001 0000 0000 1110	Load the contents of location 14 into AC
0000 0000 0110	0101 0000 0000 0000	Add DR to AC
0000 0000 1000	0010 0000 0001 0000	Store contents of AC in location 16
0000 0000 1010	0000 0000 0000 0000	Stop
0000 0000 1100	0000 0001 0101 1110	Data value 350
0000 0000 1110	0000 0000 0110 0000	Data value is 96
0000 0001 0000	0000 0000 0000 0000	Data value is 0

Mašinski jezik (2)

TABLE 3.3 Simple Machine Language Program in Hexadecimal (Example 1)

Memory location (bytes)	Hex instruction
000	100C
002	3000
004	100E
006	5000
008	2010
00A	0000
00C	015E
00E	0060
010	0000

x86 Familija (1)

TABLE 3.7 Main Features of the Intel *X86* Microprocessor Family

Feature	8086	286	386	486	Pentium
Date introduced	1978	1982	1985	1991	1993
Data bus	8 bits	16 bits	32 bits	32 bits	64 bits
Address bus	20 bits	24 bits	32 bits	32 bits	32 bits
Operating speed	5,8,10 MHz	6,8,10, 12.5, 16, 20 MHz	16, 20,25, 33, 40, 50 MHz	25, 33, 50 MHz	50, 60, 66, 100 MHz
Instruction cache size	NA	NA	16 bytes	32 bytes	8 Kbytes
Data cache size	NA	NA	256 bytes	8 Kbytes	8 Kbytes
Physical memory	1 Mbytes	16 Mbytes	4 Gbytes	4 Gbytes	4 Gbytes
Data word size	16 bits	16 bits	16 bits	32 bits	32 bits

Registri x86 Familije (2)

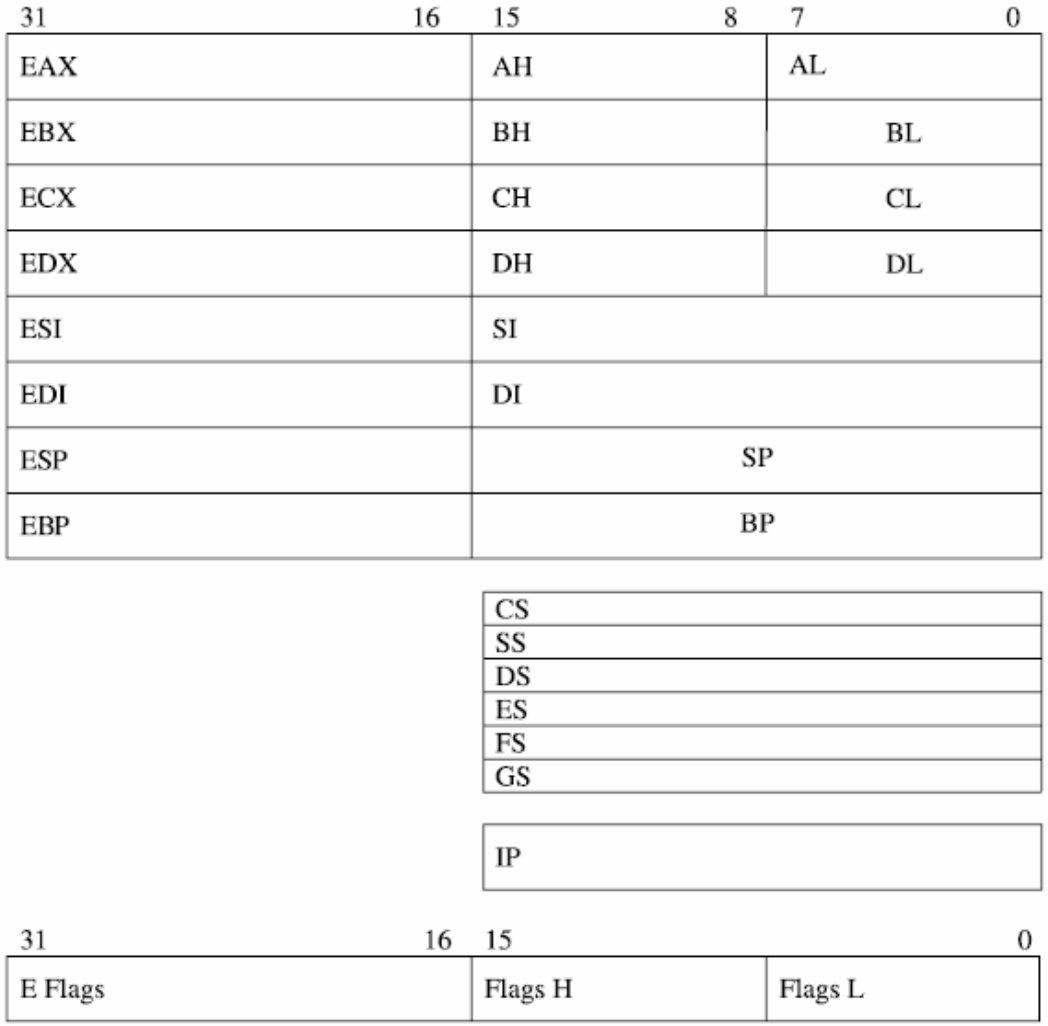


Figure 3.6 The base register sets of the *X86* programming model

CPU Intel P4 procesor