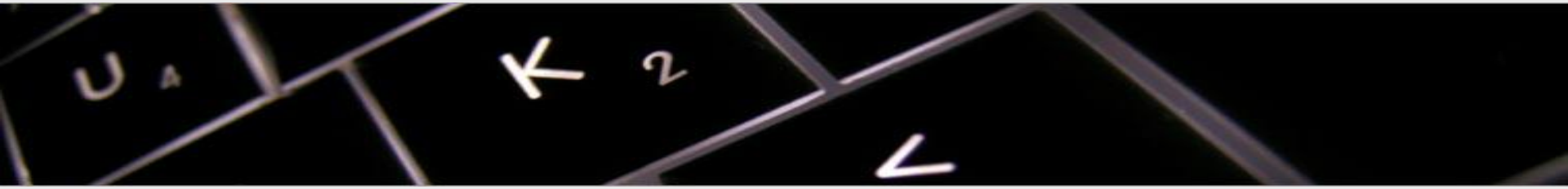
Катедра за рачунарску технику и информатику

Програмирање 1

Прва година студијских програма

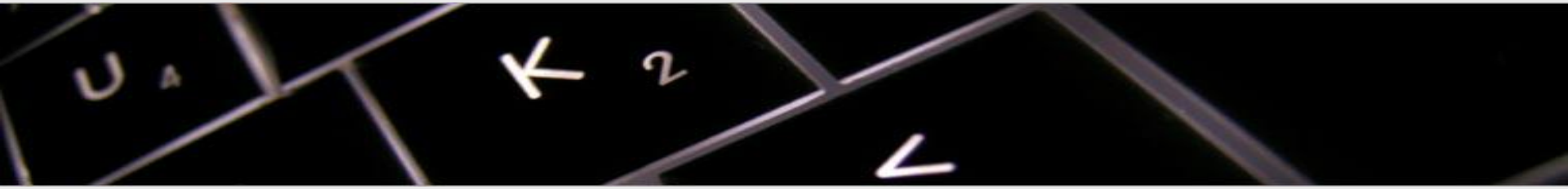
- **Електротехника и рачунарство**
- **Софтверско инжењерство**

Програмирање 1



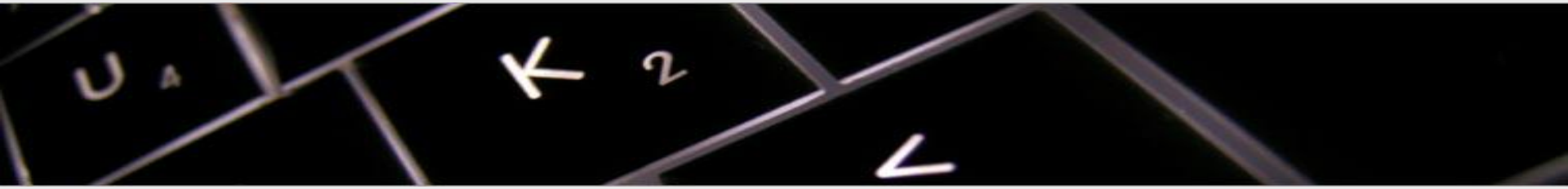
- **У школској 2020/21 предмет се изводи по новој акредитацији из 2019. године**
- **Сајт предмета: https://rti.etf.bg.ac.rs/rti/ir1p1/index_si.html**
- **Наставници:**
  - **Мило Томашевић**
  - **Јелица Протић**
  - **Марко Мишић**
- **Сарадници:**
  - **Владимир Јоцовић**
  - **Јован Ђукић**
  - **Алекса Србљановић**

Практикум из ПРОГРАМИРАЊА 1



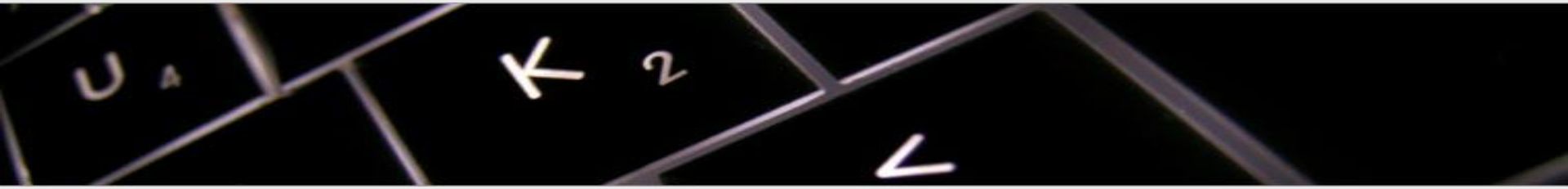
- **Прати предмет Програмирање 1**
- **Такође се изводи даљински**
- **Неће кренути пре 4. недеље наставе**
- **Сајт предмета: http://rti.etf.bg.ac.rs/rti/ir1pp1/index.html**
- **Циљ је практичан рад кроз домаће задатке**
- **Неки задаци ће се бранити уживо у лабу**
- **Практикум је обавезан предмет за Софтверско инжењерство а изборни за Електротехнику и рачунарство**

САДРЖАЈ



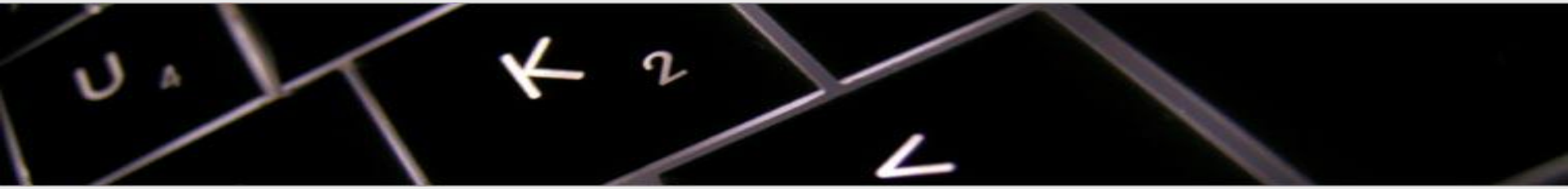
- **О КУРСУ**
- **ИСТОРИЈАТ**
- **ПРЕГЛЕД ЈЕЗИКА**
- **УВОД У ОРГАНИЗАЦИЈУ РАЧУНАРА**
- **ЖИВОТНИ ЦИКЛУС СОФТВЕРА**

Садржај курса



- **Увод**
- ***p*C**
- **Синтаксне нотације**
- **Високи ПЈ (на примеру Python-а)**
- **Основне структуре и алгоритми**

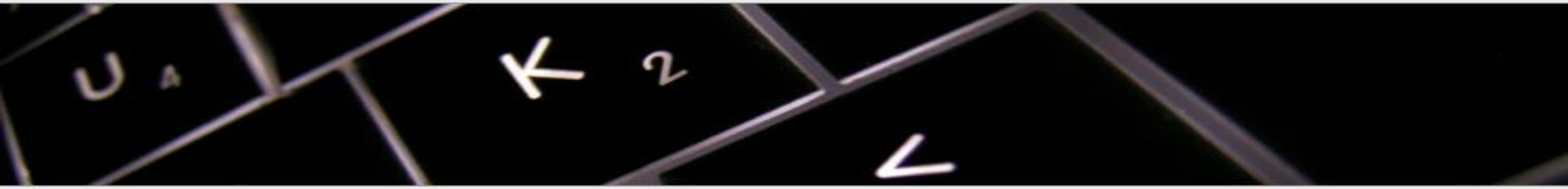
Литература



Почетни теоријски део:

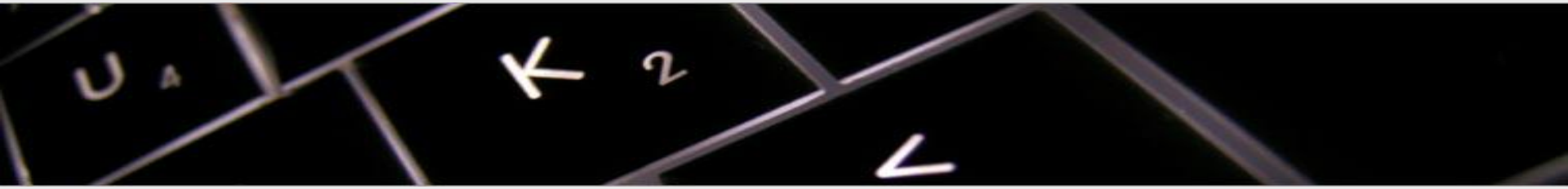
- **Програмски језици и методе програмирања – Проф. др Јозо Дујмовић**
- **Збирка задатака из програмских језика и метода програмирања - Мр Ласло Краус**
- **Python: нема званичног уџбеника, могу да се користе разне књиге**

Литература - Python



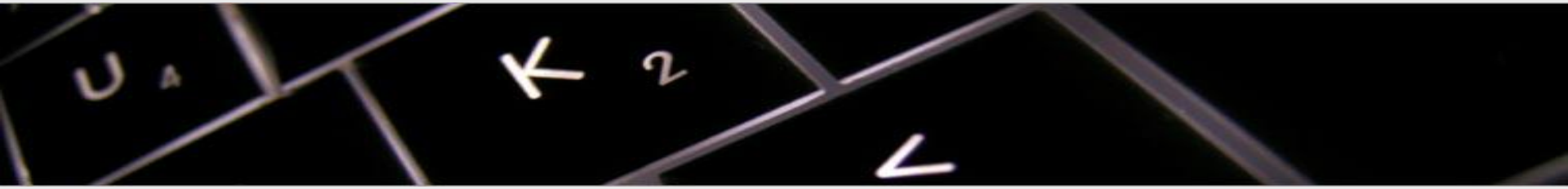
- **М. Ковачевић, Основе програмирања у Пајтону, Академска мисао, Београд 2017**
- **M. Lutz, Learning Python: Powerful object-oriented programming, 5th edition, O'Reilly Media, Inc., 2013.**
- **J. Zelle, Python Programming: An Introduction to Computer Science, 3rd Ed., Franklin, Beedle & Associates, 2016.**
- **D. Beazley, B. K. Jones, Python Cookbook, 3rd edition, O'Reilly Media, 2013.**
- **A. Downey, J. Elkner, C. Meyers, How To Think Like A Computer Scientist: Learning With Python, free e-book**

Литература – Python (*online* извори)



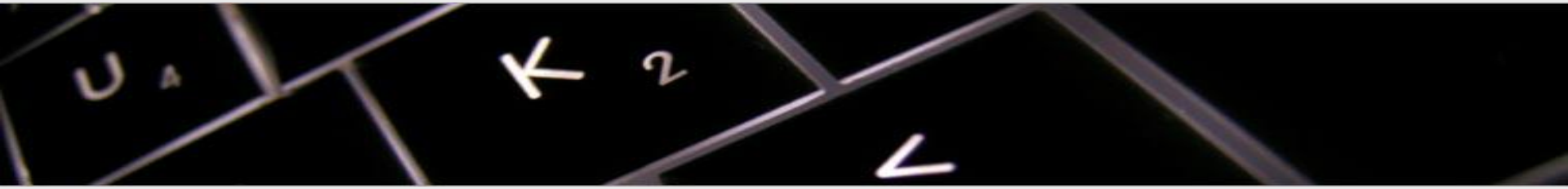
- **На курсу се ради верзија 3.x**
- **Python 3.8.0 documentation, https://docs.python.org/3/index.html**
- **Colin Morris, 7-day Python course, https://www.kaggle.com/learn/python**
- **Learn Python, Basic tutorial, https://www.learnpython.org/**
- **TutorialsPoint, Python tutorial https://www.tutorialspoint.com/python/index.htm**

Увод



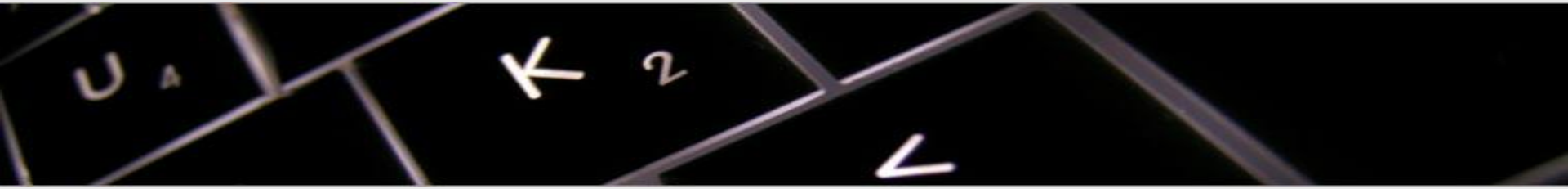
- Рачунари и програми – Инжењерски производи
- Хијерархијска декомпозиција проблема
- Стандарди у развоју инжењерског производа
- Документација

Увод



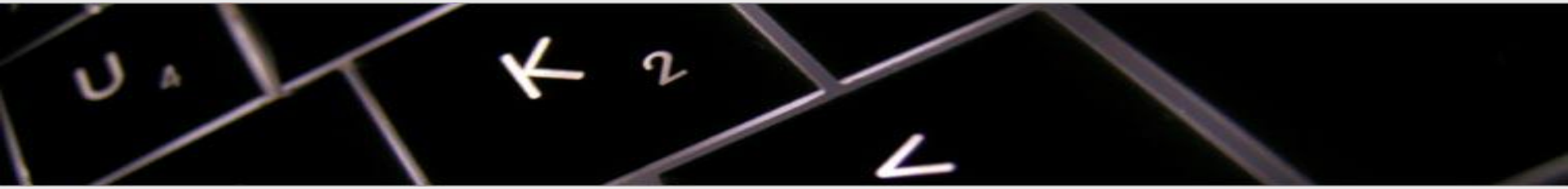
- Програме често развијају полупрофесионалци и аматери
- За квалитет је потребна инжењерска култура
- Мора се радити анализа трошкова и перформанси решења

Увод



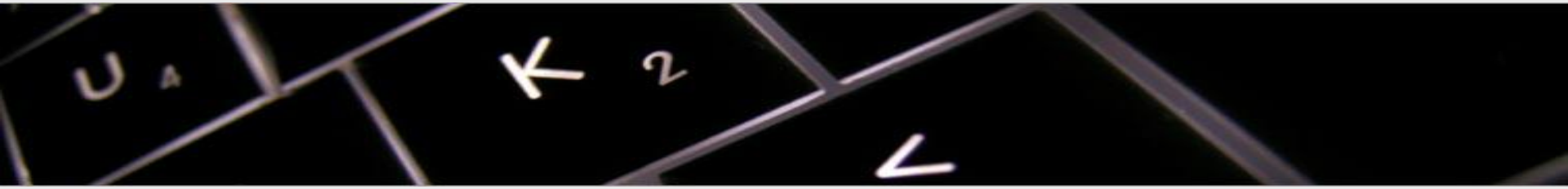
- Професионални морал (вируси)
- Технички и етички стандарди
- Веза између теоријских знања и практичних потреба

Историјат



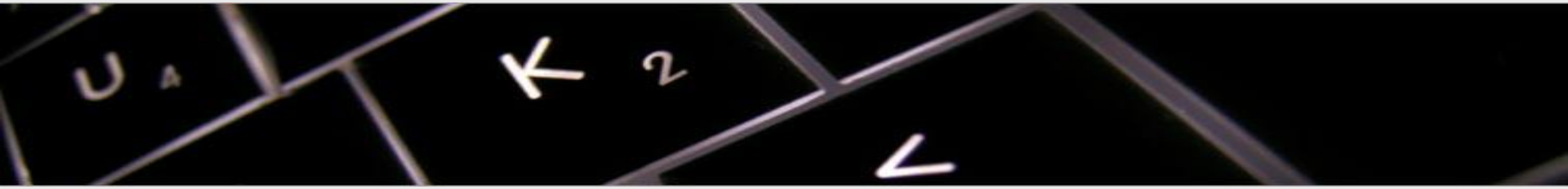
- Стара Грчка – Еуклидов алгоритам за највећи заједнички делилац
- Кина – Абакус и алгоритми за дељење и множење
- 1600 - Лењир са покретном скалом

Историјат



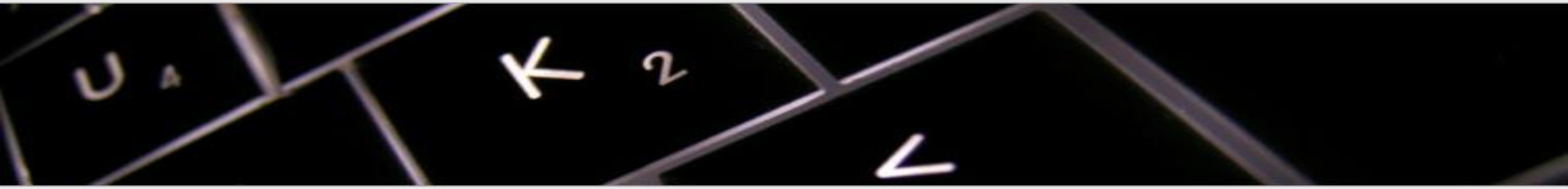
- 1642 – Паскал (Blaise Pascal), механичко сабирање бројева
- ~1800 – Бебиџ (Charles Babbage), механичко рачунање наутичких таблица и формирање машине за аналитичко рачунање
- ~1800 – Кинг (Ada King), први програмер

Историјат



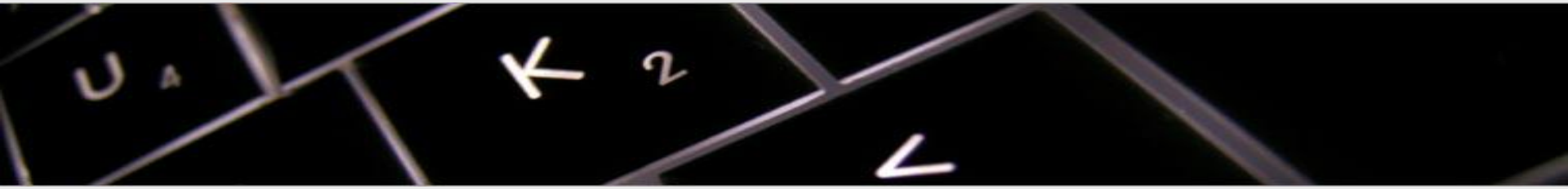
- 1854 – Бул (George Boole), алгебра над коначним скупом
- 1890 – Холерит (Hollertih), аутомат за рад са бушеним картицама
- 1930 – Тјуринг (Alan Turing), Zuse - електромеханички системи

Историјат



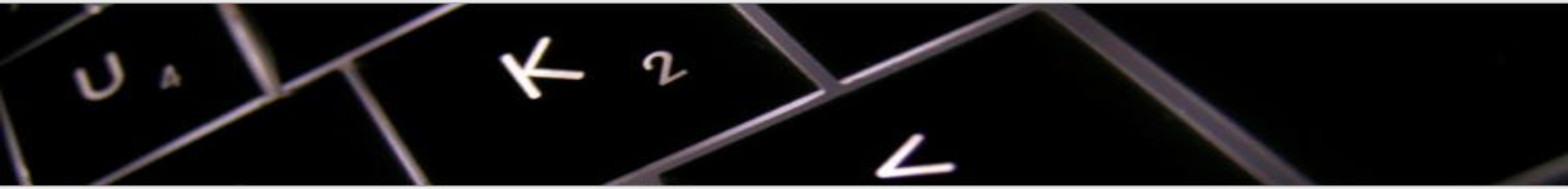
- 1930 – Екерт(Eckert), Мокли (Mauchly) и фон Нојман (von Neumann), идеје о савременим рачунарима
- 1940 – Први електронски рачунари
- ~1940 – Тјуринг (Alan Turing), развија теорију о рачунарима (Тјурингова машина)

Историјат



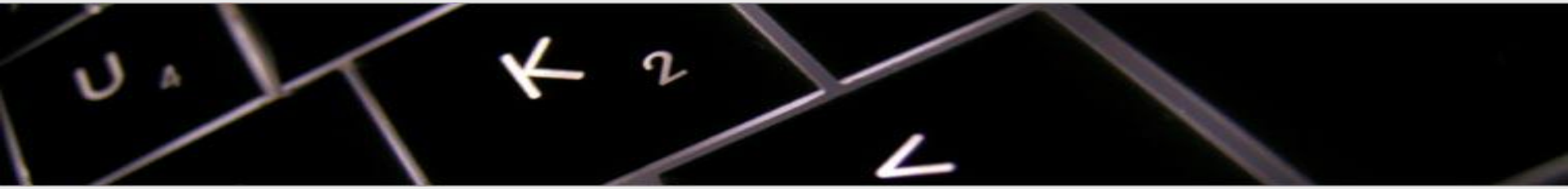
- ~1950 – Појава транзистора
- 1957 – Развијен FORTRAN
- 1960-те – Могућност смештања ~1000 транзистора на исти чип
- 1964 – Трећа генерација рачунара (IBM 360)

Историјат



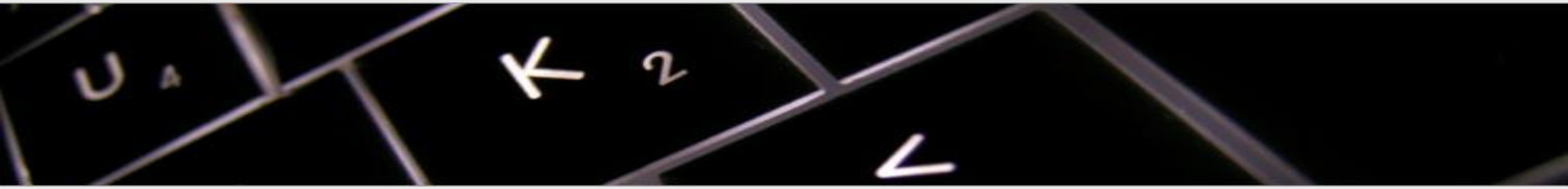
- 1969 – Појава прве мреже ARPAnet
- 1971 – Појава микропроцесора, флопи дискова, развој PASCAL-а
- 1972 – Четврта генерација рачунара LSI (Large Scale Integration)
- 1974 – Intel креира 8086 процесор

Историјат



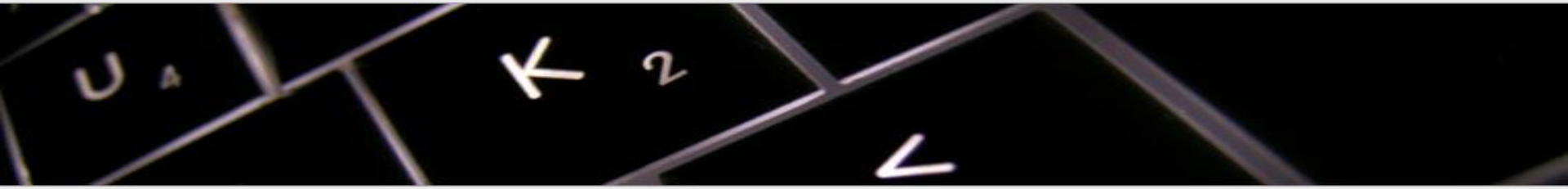
- 1974 – Бил Гејтс (Bill Gates) и Пол Алан (Paul Allen) оснивају Мајкрософт (Microsoft)
- 1981 – IBM прелази на PC тржиште
- 1982 – Times проглашава рачунар за “Човека године”
- 1984 – Mac представља графичко окружење

Историјат



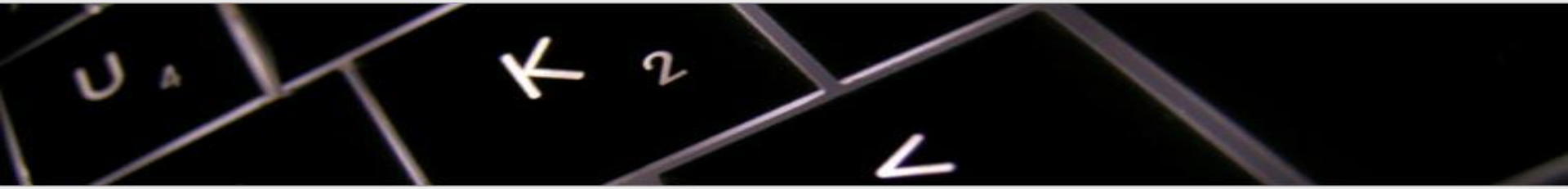
- 1985 – Мајкрософт (Microsoft) представља своје графичко окружење - Windows
- 1989 - World Wide Web
- 1991 – Linux
- 1995 - Windows '95
- 1997 - Intel представља Pentium MMX

Историјат



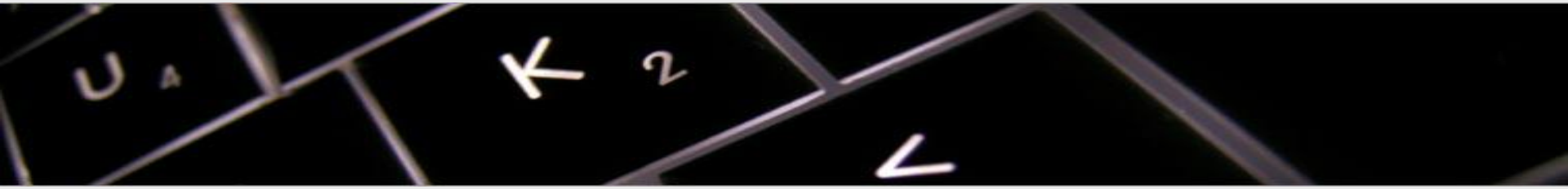
- 1999 - AMD представља Athlon процесор
- 2000 - Мајкрософт Windows 2000
- 2002 – Мајкрософт .NET
- 2005 – Први 64-битни процесори AMD Athlon 64 X2 и Intel Pentium D
- 2007 – Apple представља први iPhone

Историјат



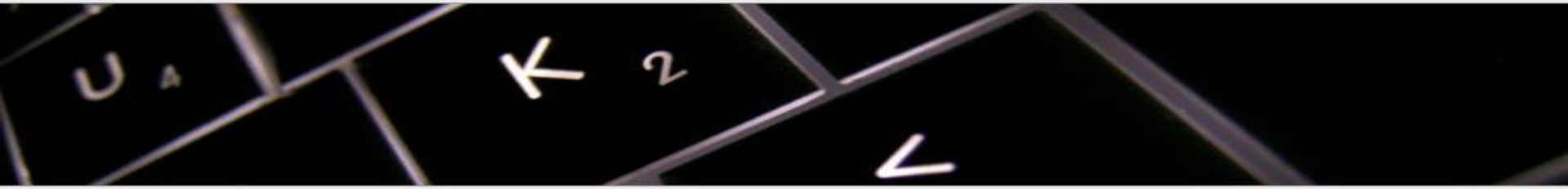
- 2008 - Google представља прву верзију оперативног система Android
- 2009 - Мајкрософт Windows 7
- 2009 – Дигитална валута Bitcoin
- 2010 – Apple представља iPad
- 2012 – Представљен Raspberry Pi
- 2015 – Мајкрософт Windows 10

Машински језик



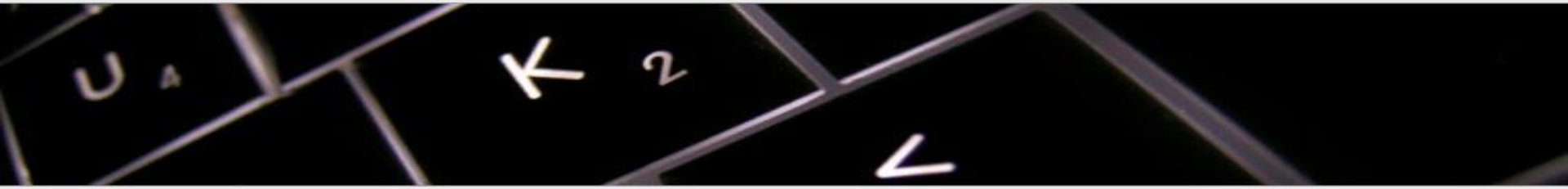
- Низ нула и јединица
- Неразумљивост
- Директно управљање меморијом
- Непостојање библиотека

Асемблерски језик



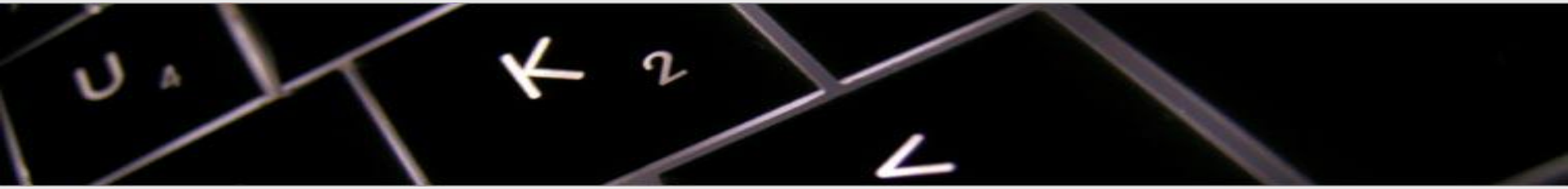
- Симболичко кодирање машинских инструкција
- Могућност употребе лабеле
- Већи степен разумљивости

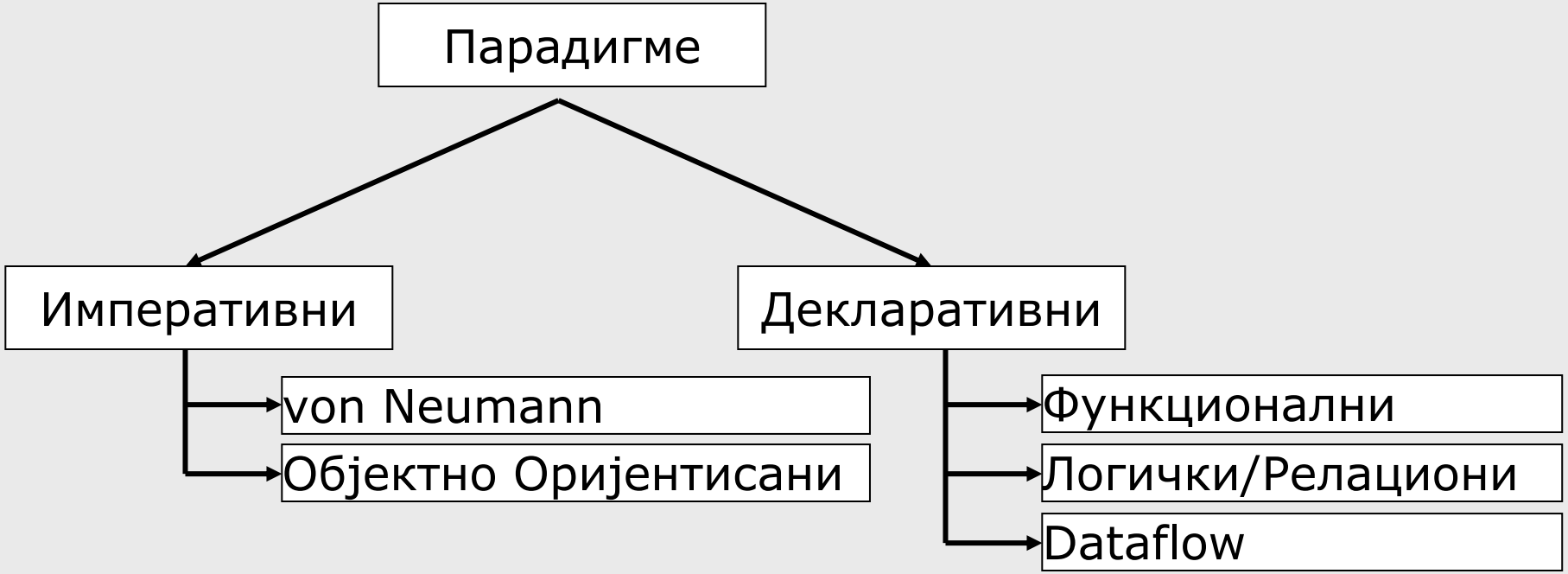
Виши програмски језици



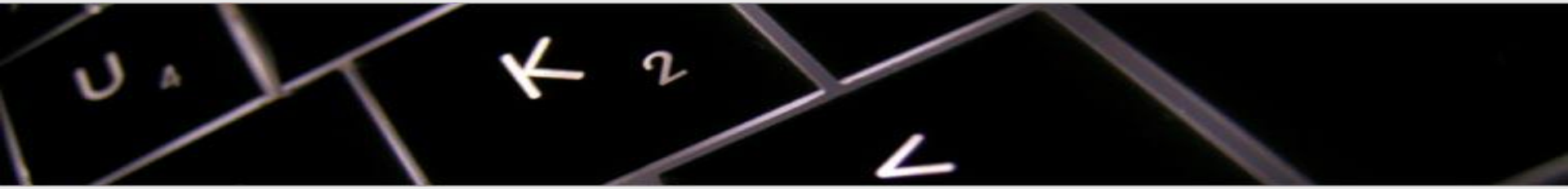
- Декларација типова података
- Комплексни алгебарски изрази
- Провера типова података
- Лако управљање ресурсима
- Употреба библиотека

Програмске парадигме



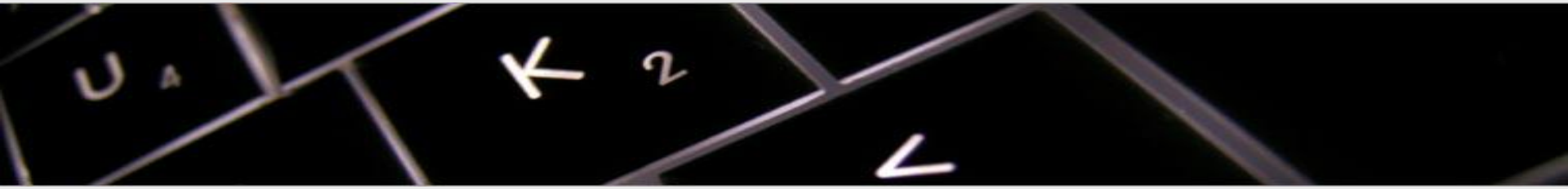


Програмске парадигме

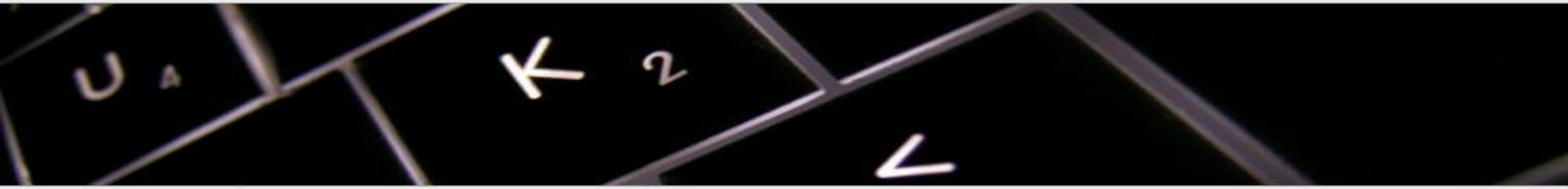


- Императивна – заснована на von Neumann-овој архитектури
- КАКО решити проблем
  - Редослед корака алгоритма
  - Груписање у процедуре
  - C, Pascal, Basic, Fortran, PL…
- Објектна
  - Објекти интегришу податке и процедуре
  - Интеракција објеката путем порука
  - Хијерархија класа; наслеђивање
  - SmallTalk, Simula 67, Java, C++, C#...

Програмске парадигме



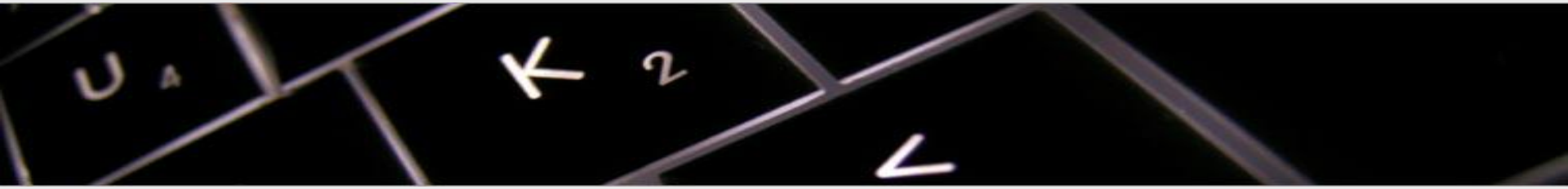
- Функционална
  - Израчунавања су резултат математичких функција (дефинише се ШТА а не КАКО урадити)
  - Имају математичку основу у ламбда рачуну
  - Lisp, Scheme, Haskell, ML…
- Логичка
  - Заснива се на аксиомама, правилима извођења и упитима
  - Основа је математичка логика тј. предикатски рачун
  - Prolog, ASP, Datalog, CLP, Solver…



Dataflow

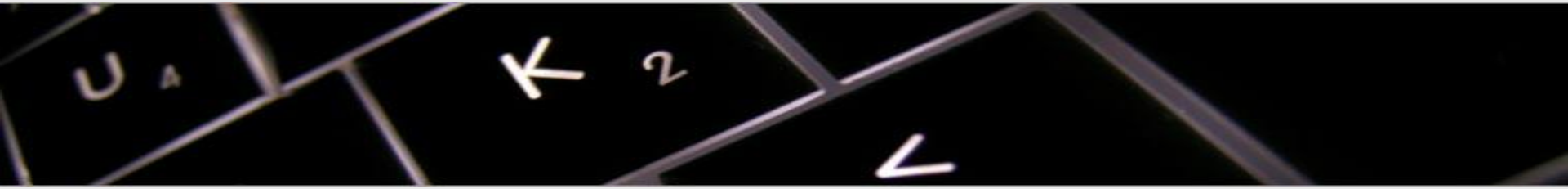
- Не познаје појам меморије за податке и код
- Постоји ток података са улаза од инструкције до инструкције која их процесира
- У суштини подржавају паралелизам
- Језици за опис хардвера (VHDL, Verilog…) спредшит апликације, неки елементи језика Oz, Groovy…

Развој програмских језика



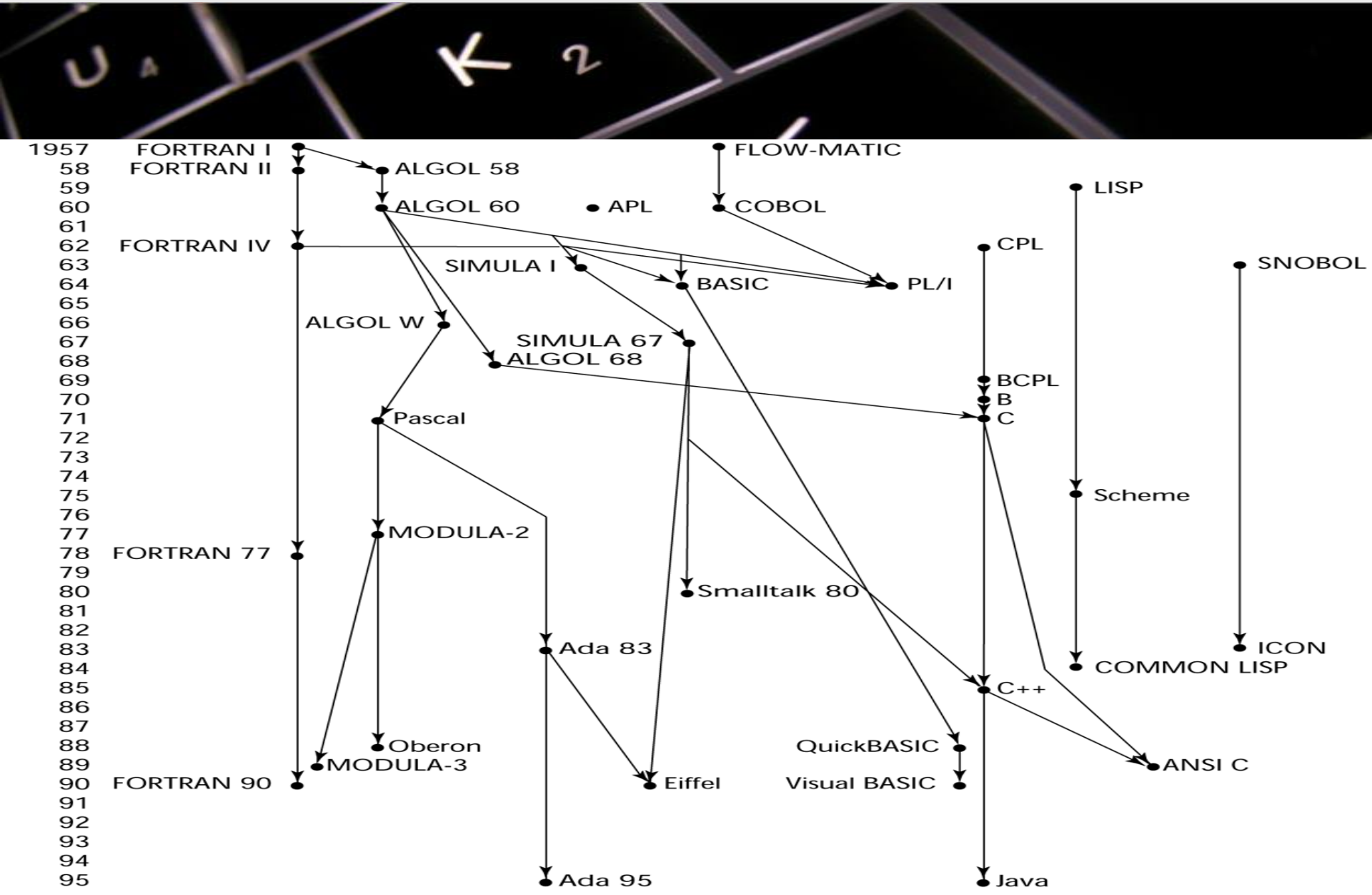
- **1940**-те: први рачунари; машински језик → асемблерски језик
- **1950-**те: нумеричка рачунања доминантна, појава **FORTRAN**-а првог правог вишег програмског језика
  Настају: **Cobol, Algol 58, Lisp**
- **1970**-те: структурирано програмирање – увођење реда **Pascal, Prolog, C, Ada**

Развој програмских језика

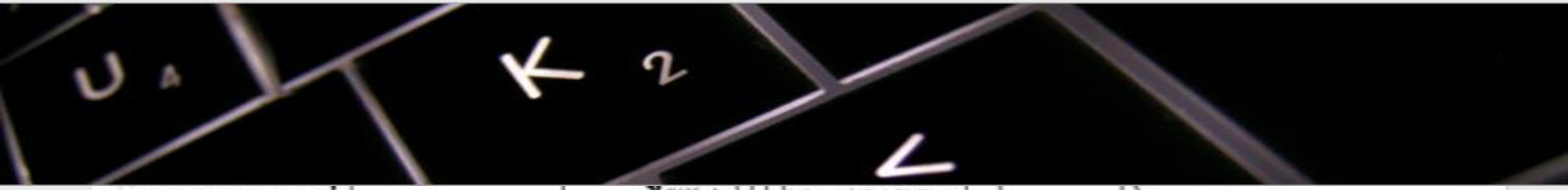


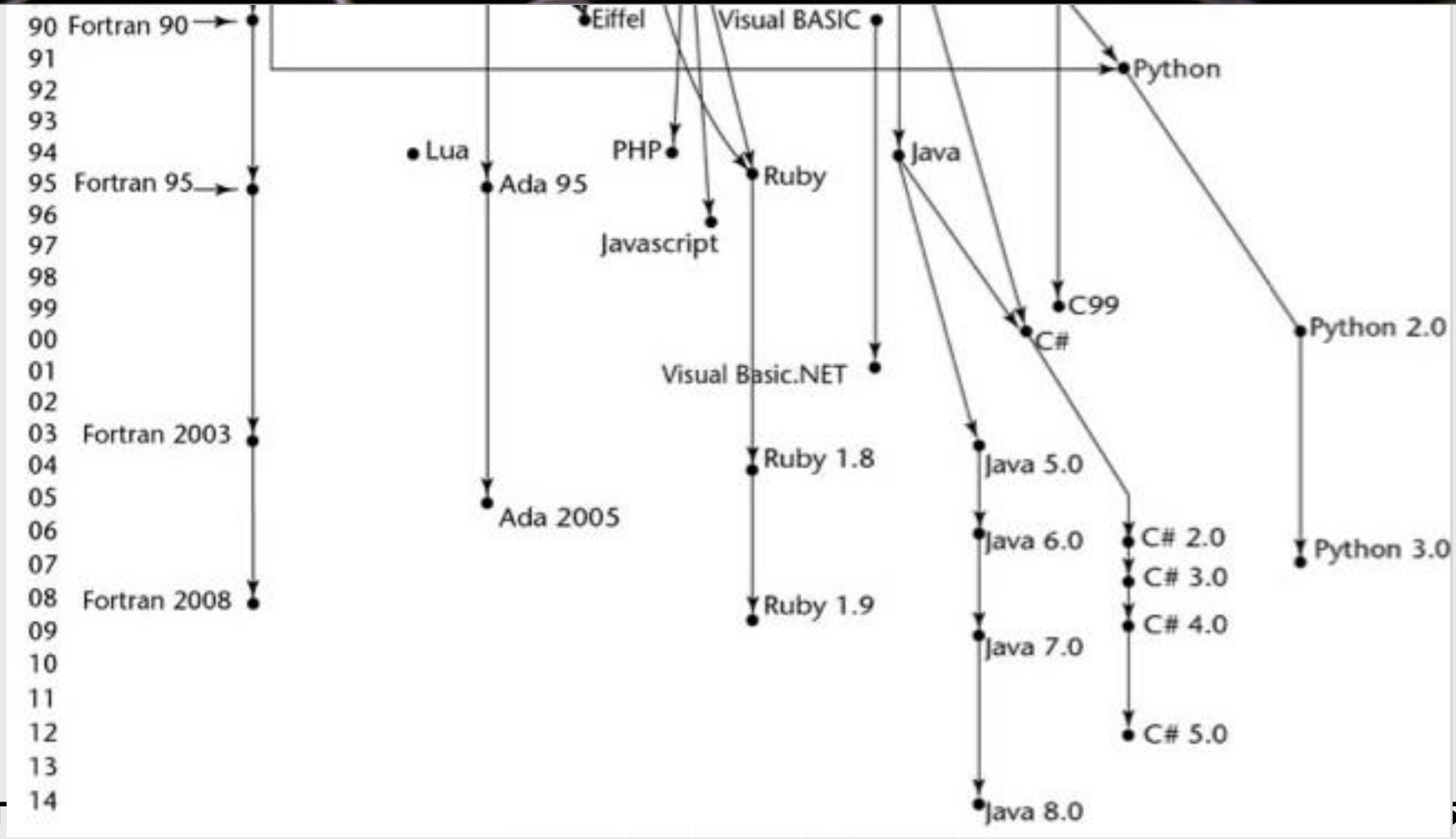
- **1980-**те: Објектно оријентисано програмирање **C++**
- **1990-**те: Развој скрипт језика и употреба виртуелне машине: **HTML, Java, PERL**
- **2000**-те: Широка употреба виртуелних машина и web сервиса: **.NET, C#**

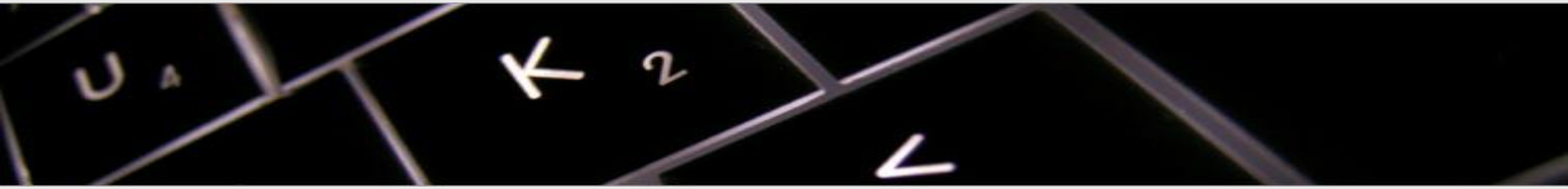
Развој програмских језика



...и наставак од деведесетих

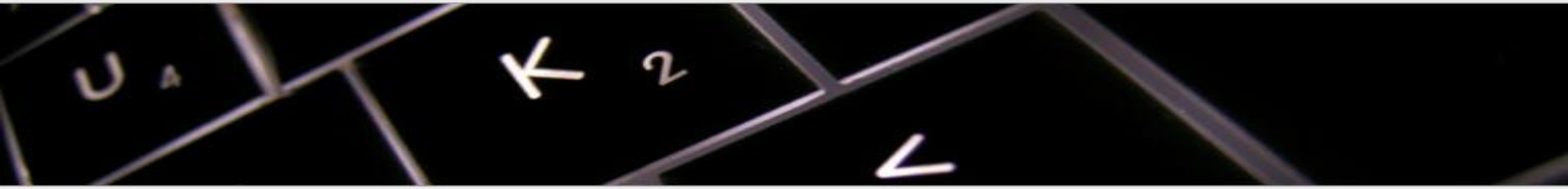






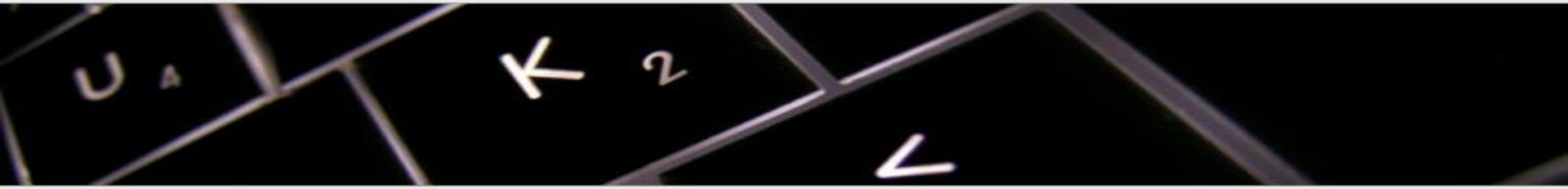
Algol 60

- Блоковски оријентисан
- Прва употреба Bachus-Naurove форме (BNF)
- База императивног програмирања
- Наследник Algol 68
- Утицао на Pascal, Adu и C
- Једноставнија варијанта AlgolW



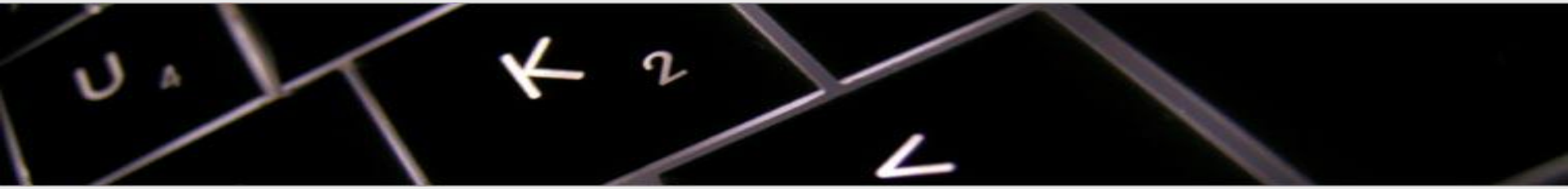
Cobol

- Настао на прелазу 50-тих и 60-тих
- Служио за пословне апликације
- Оригинално покренут од U.S. Department of Defense
- Поседује форматирање и рад са децималним бројевима



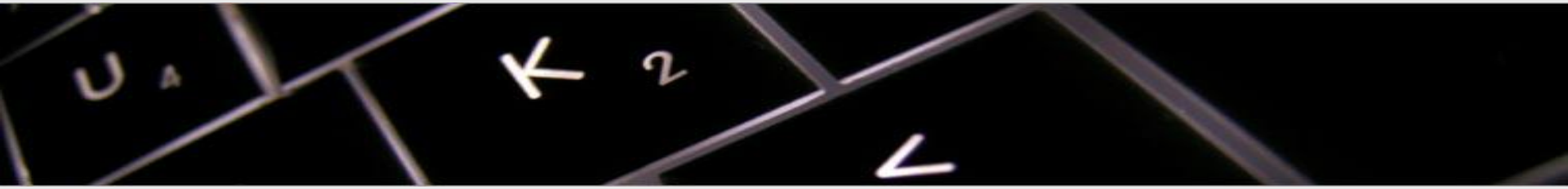
Basic

- Циљ језика је да се лако учи
- Настао раних 60-тих
- Наследник Visual Basic из '91.-е



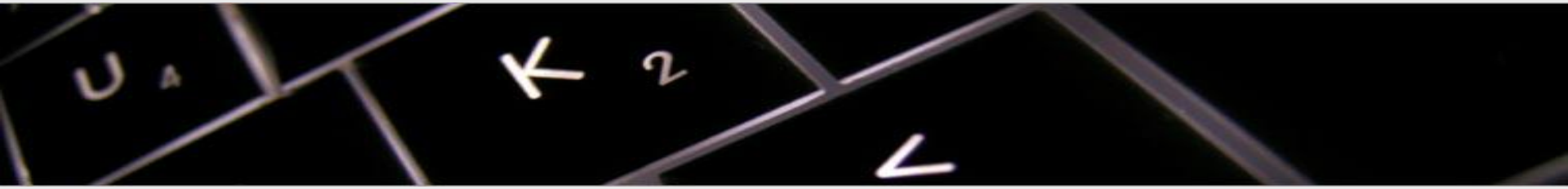
PL/I

- Направио и форсирао IBM
- Први увео поинтере и обраду изузетака
- Лош дизајн
- Гломазан језик



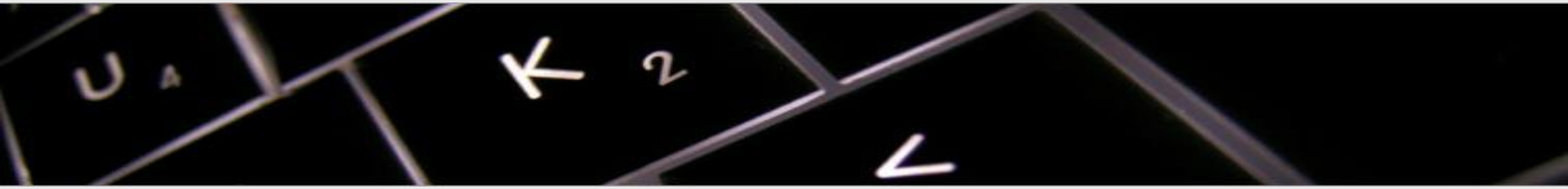
Pascal

- Мали, једноставан
- Концептуално чист
- Дефинисао Wirth
- Јако коришћен 70.-тих и 80.-тих
- Стандарди ISO и ANSI 1990



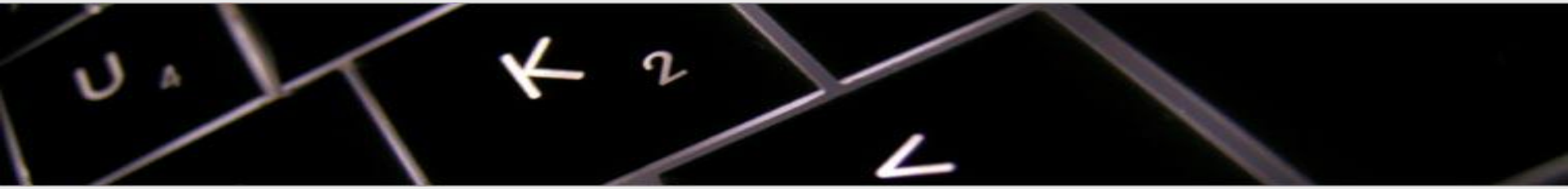
Ada

- Покренуо Department of Defense
- Велики језик
- Ada95 – подршка за објектно оријентисано програмирање



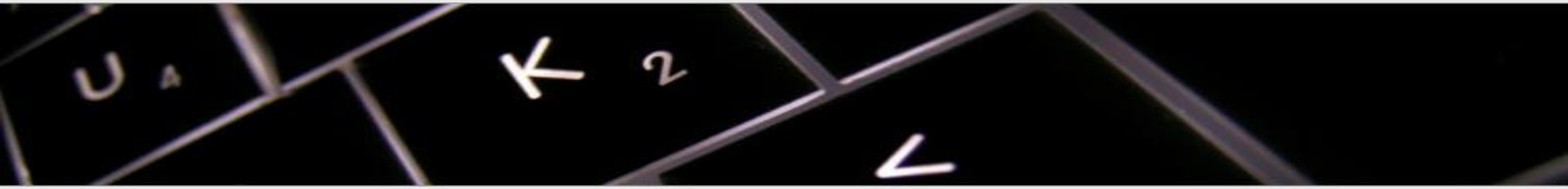
Prolog

- Базиран на формалној логици
- Логичко програмирање



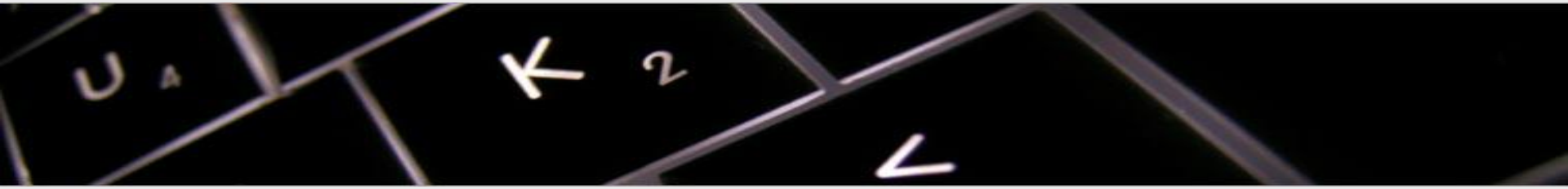
Java

- Настао 1995
- Производ корпорације SUN
- Објектно оријентисан
- Платформски независан
- Интерпретира га JVM - Java виртуелна машина



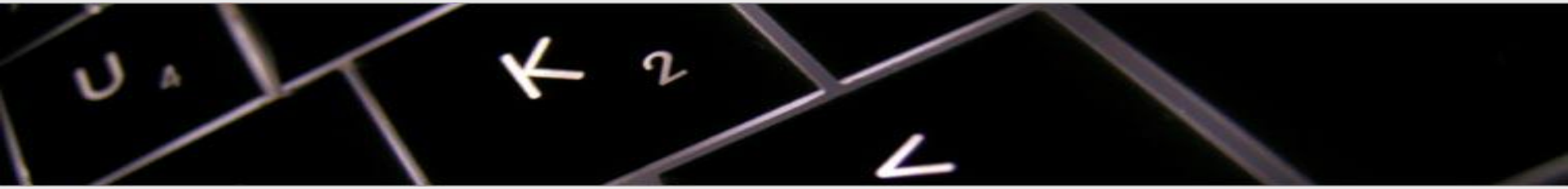
.NET

- Microsoft .NET
- Могућност рада са више језика (C++, C#, Visual Basic, COBOL, Fortran, Eiffel)
- Коришћење виртуалне машине
- Рад са web сервисима



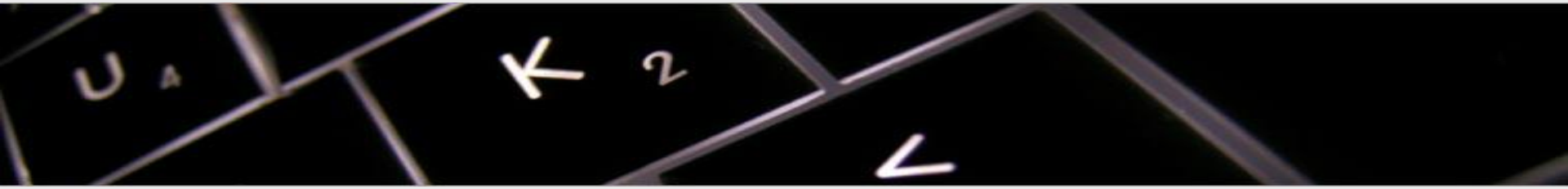
Java Script

- Компактан, објектно заснован скриптни језик, намењен развоју клијентске стране веб апликација
- Код интегрисан у HTML интерпретира се у browser-у, који подржава Java Script
- Подржава динамичке типове
- Постоји могућност и за серверске, па и за мобилне апликације
- Постоје бројне библиотеке (Jquery, Angular JS framework…)



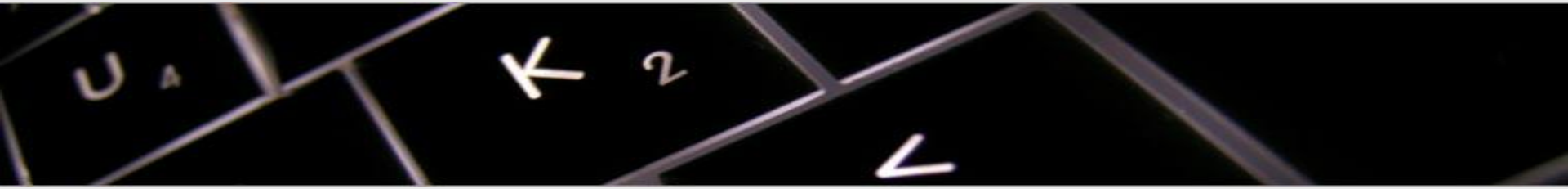
PHP

- Најпопуларнији језик за израду Web апликација (на серверској страни)
- Био је процедурални скриптинг језик, а касније је додата подршка за објектно-оријентисано програмирање
- Компаније за Web hosting нуде сервере који подржавају PHP; јефтинији је од других платформи
- Web апликације Word Press, Joomla... писане у њему омогућавају лакшу израду сајтова



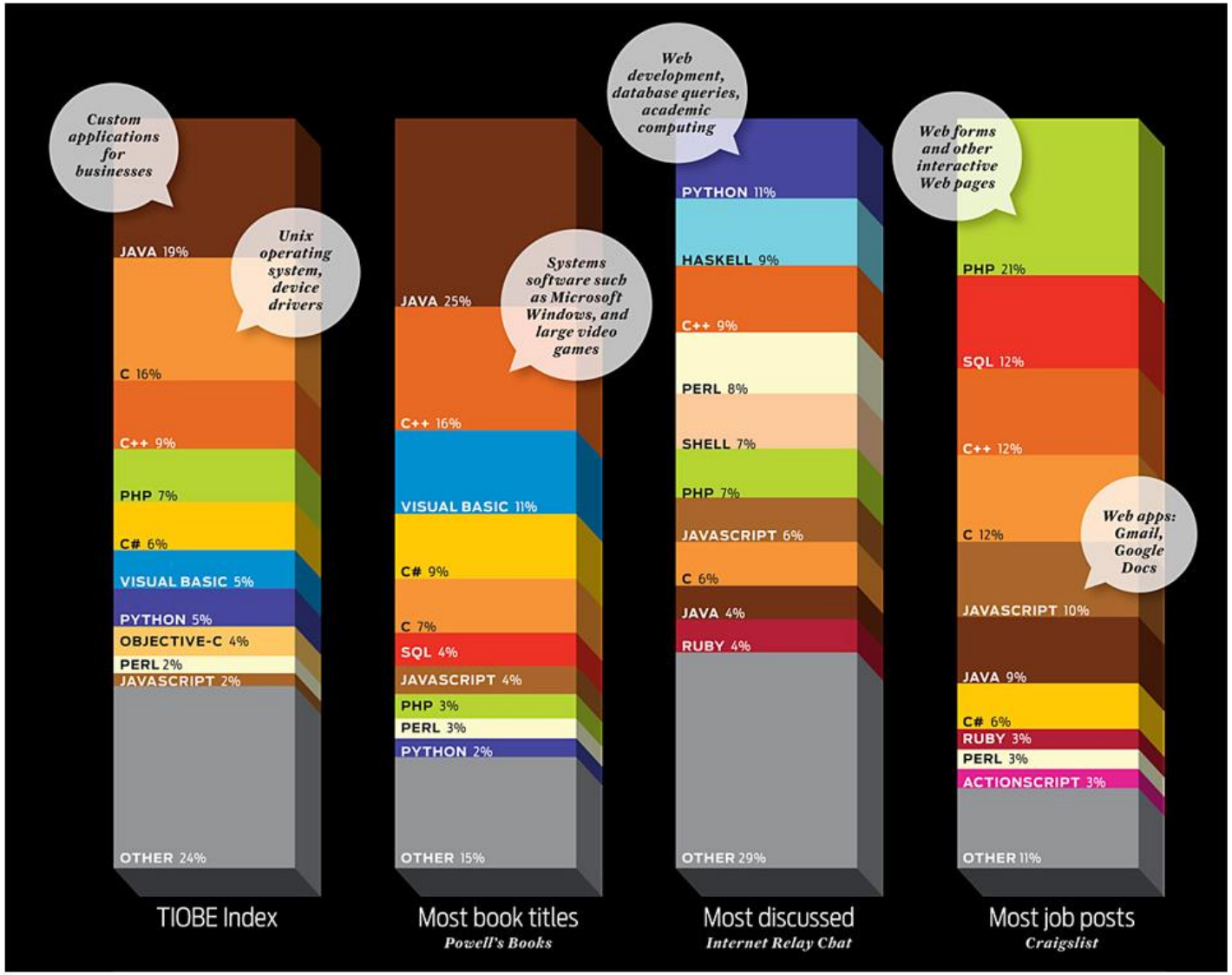
C#

- Anders Hejlsberg који је радио на развоју Turbo Pascal-а покреће развој језика 1999. године у оквиру .NET иницијативе Microsoft-а
- C# је сличан језику Java, али уз неке особине језика C++ по коме је добио име (# треба да асоцира на двоструки ++)
- Концепт генерика, затим делегата, као безбедни показивач на функцију
- Подржава све три парадигме



Python

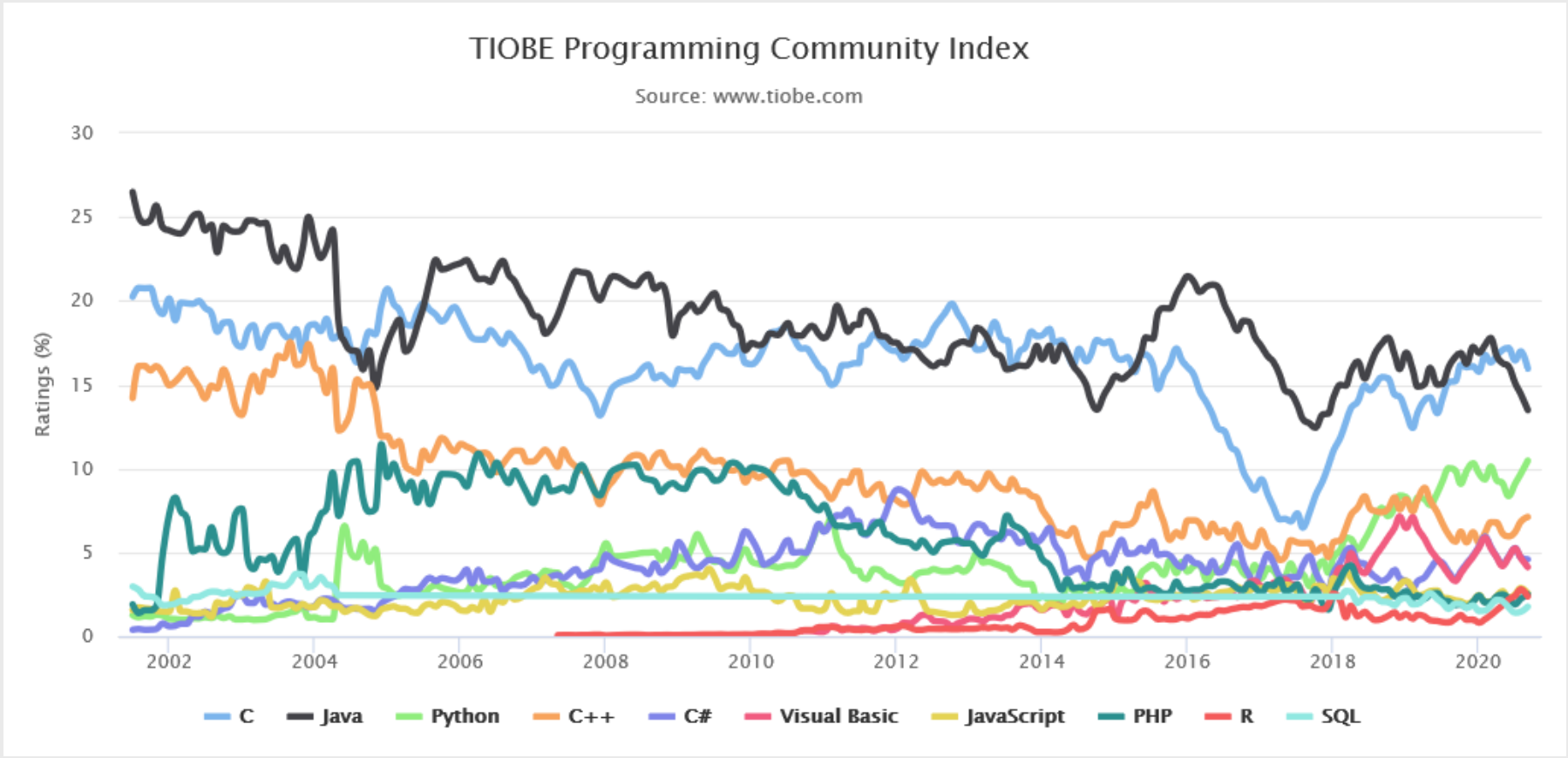
- Језик за писање једноставних скрипти, али и великих, сложених апликација (Desktop, Web...)
- Подржава све три парадигме
- Преводи се у byte-code који се касније интерпретира
- Минималистички дизајн
- Код се јако лако пише и чита, јер подсећа на псеудо-код, па се програми писани у њему лакше одржавају



SOURCES: TIOBE, LangPop.com. Because of rounding, categories may not total 100 percent.

Sep 2020	Sep 2019	Change	Programming Language	Ratings	Change
1	2	↑	C	15.95%	+0.74%
2	1	↓	Java	13.48%	-3.18%
3	3		Python	10.47%	+0.59%
4	4		C++	7.11%	+1.48%
5	5		C#	4.58%	+1.18%
6	6		Visual Basic	4.12%	+0.83%
7	7		JavaScript	2.54%	+0.41%
8	9	↑	PHP	2.49%	+0.62%
9	19	⇈	R	2.37%	+1.33%
10	8	↓	SQL	1.76%	-0.19%
11	14	↑	Go	1.46%	+0.24%
12	16	⇈	Swift	1.38%	+0.28%
13	20	⇈	Perl	1.30%	+0.26%
14	12	↓	Assembly language	1.30%	-0.08%
15	15		Ruby	1.24%	+0.03%
16	18	↑	MATLAB	1.10%	+0.04%
17	11	⇊	Groovy	0.99%	-0.52%
18	33	⇈	Rust	0.92%	+0.55%
19	10	⇊	Objective-C	0.85%	-0.99%
20	24	⇈	Dart	0.77%	+0.13%

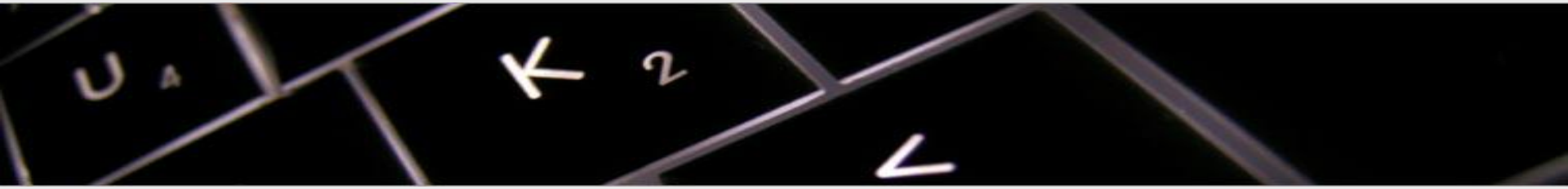
TIOBE index за септембар 2020



Трендови популарности језика 2002 - 2020

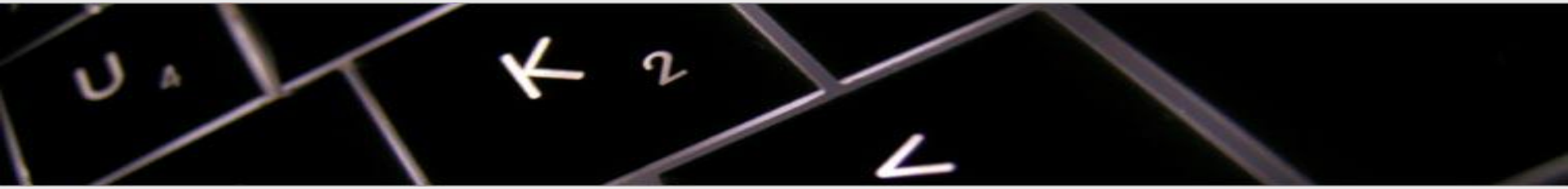
Трендови популарности језика 1965 – 2019
https://www.youtube.com/watch?v=Og847HVwRSI

Особине ДОБРОГ језика



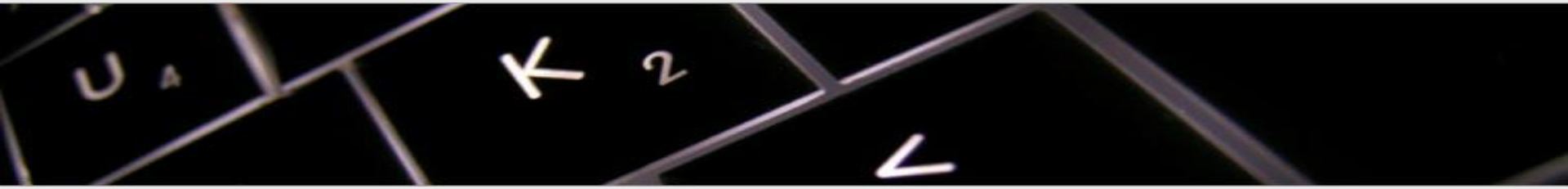
- Лак за читање/разумевање
- Лак за писање
- Експресиван
- Такав да отежава прављење грешака
- Брзо превођење
- Ефикасан код
- Преносивост

Успешан језик



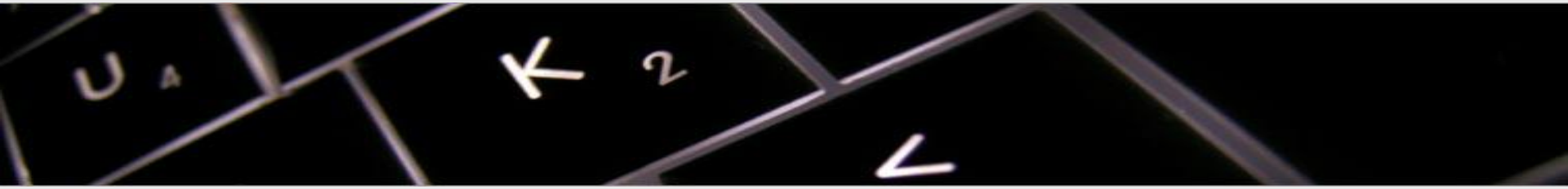
- Једини језик погодан за дати проблем
- Питање укуса
- Добро развојно окружење
- Брза компилација
- Подршка од владе и великих фирми

Успешан језик



- Сви га користе
- Шеф је рекао да га користим
- Већ смо пуно уложили у њега - економија и инерција
- Уморан сам да бих учио нови - лењост

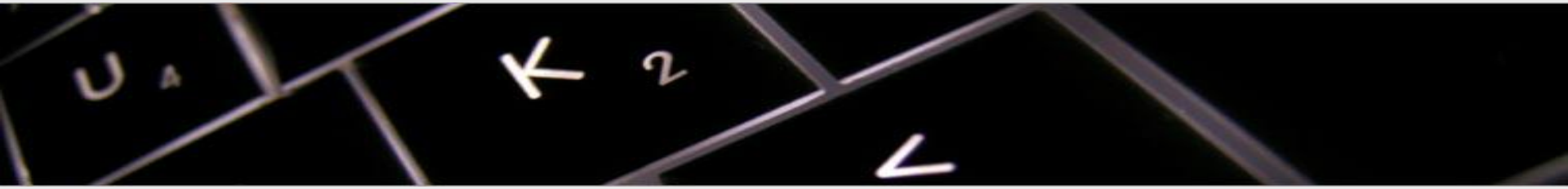
Подела програмских система



По намени:

- Системски софтвер
- Апликативни софтвер

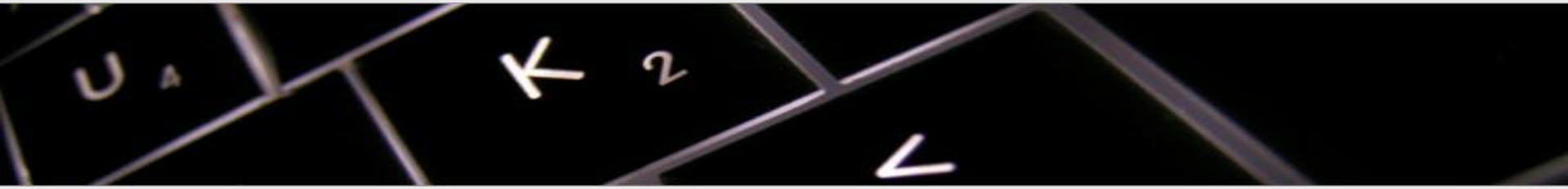
Подела програмских система



По обиму: линије кода као класична мера (веома груба)

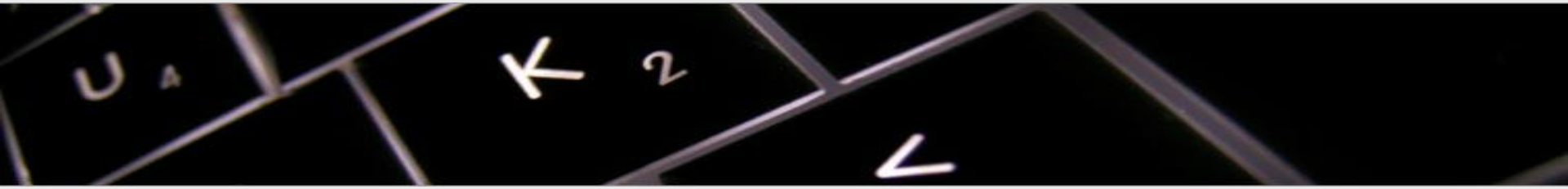
Софтверски систем	Линије кода
мали	до 2000
средњи	од 2000 до 100.000
велики	од 100.000 до 1.000.000
веома велики	више од 1.000.000

Решавање проблема



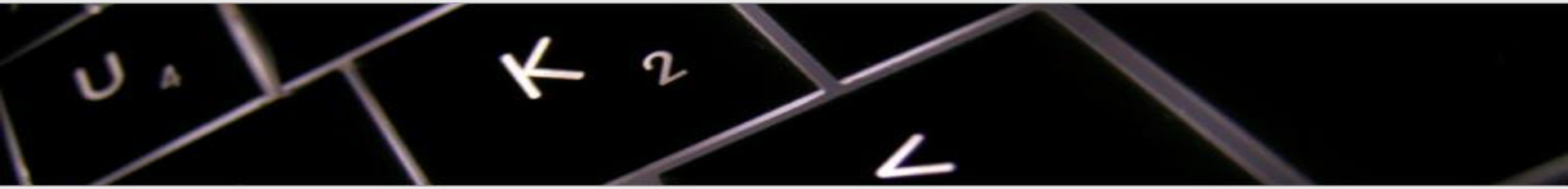
- Развој плана решавања проблема (алгоритма)
- Развој решења на конкретном програмском језику

Развој плана решавања проблема



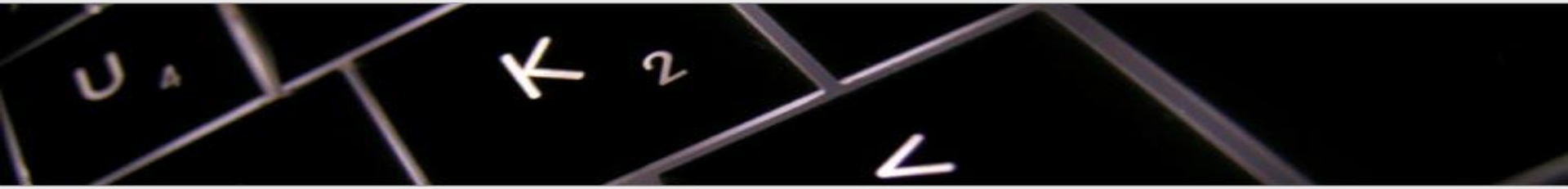
- Разумевање проблема
  са аспекта корисника
- Избор технике /средства решавања и
  потребне структуре података
- Прецизно дефинисати
  све фазе решавања
  на логичан и коректан начин

Развој решења на програмском језику



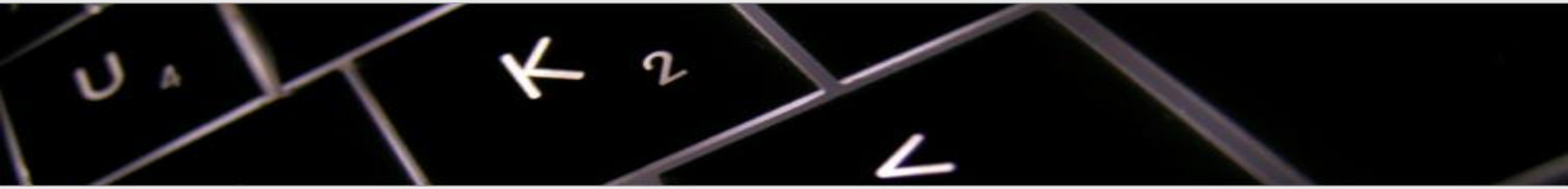
- Изабрати програмски језик
  који је расположив за циљни систем
- Језик треба да одговара проблему
- Програмер треба да има искуства са
  тим језиком

Животни циклус софтвера



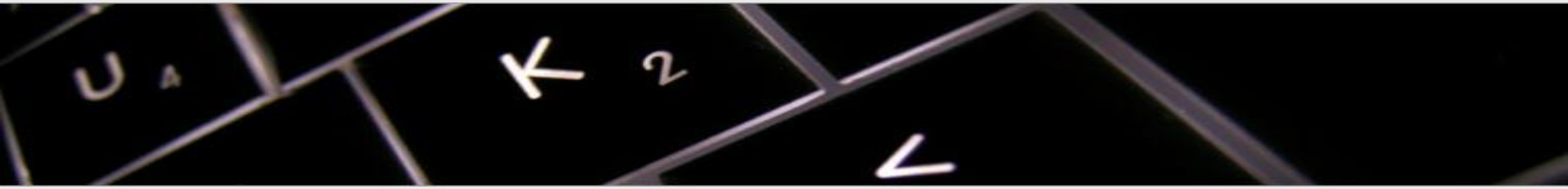
- Анализа
- Дизајн
- Кодирање
- Тестирање и верификација
- Одржавање
- Напуштање

Анализа



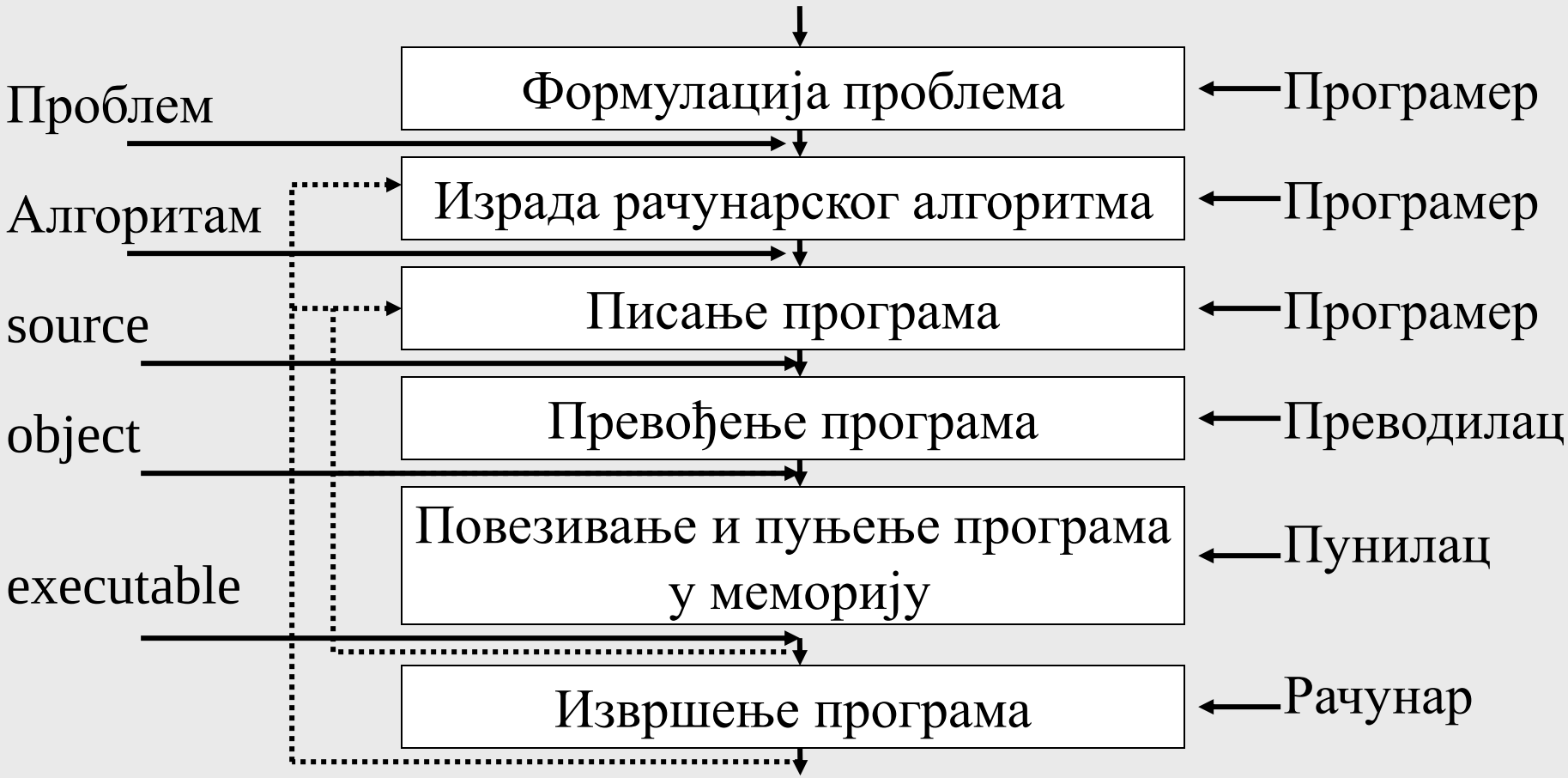
- Шта су улази?
- Шта треба да су излази?
- Које су информације потребне?

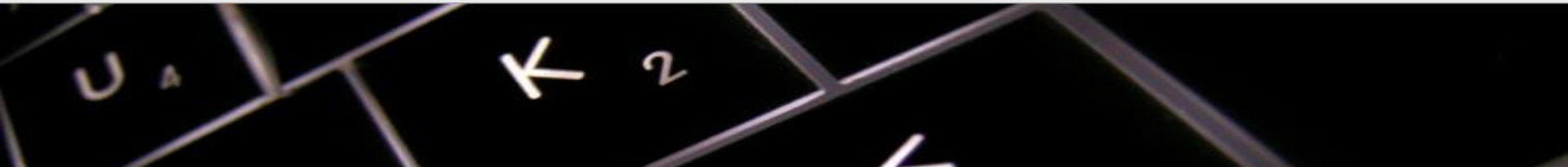
Временска подела

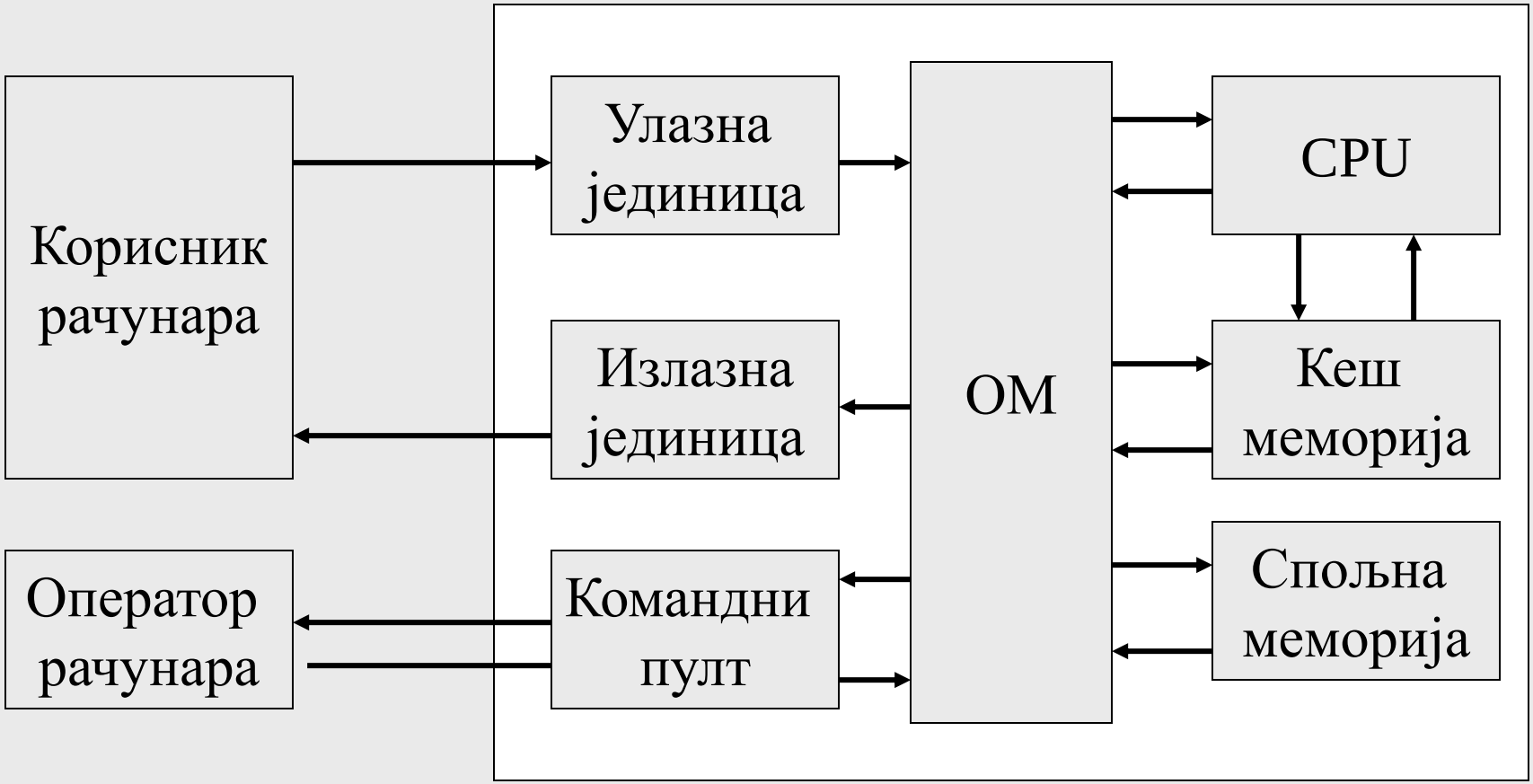


- 10-15% - анализа проблема, планирање пројекта
- 40-50 % - дизајн и имплементација
- 25-30% - тестирање, исправке
- 10-15% - документовање

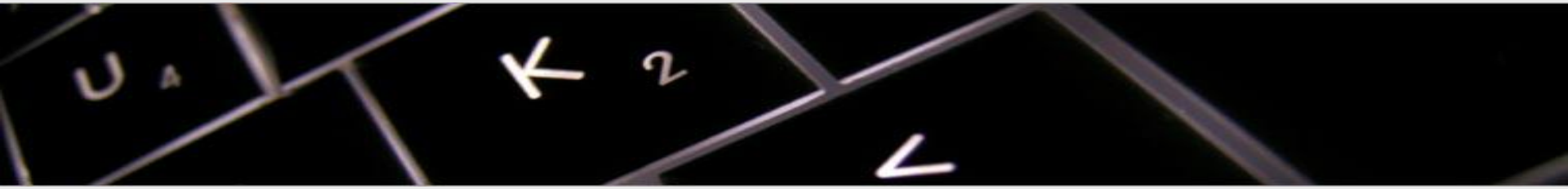
Израда програма





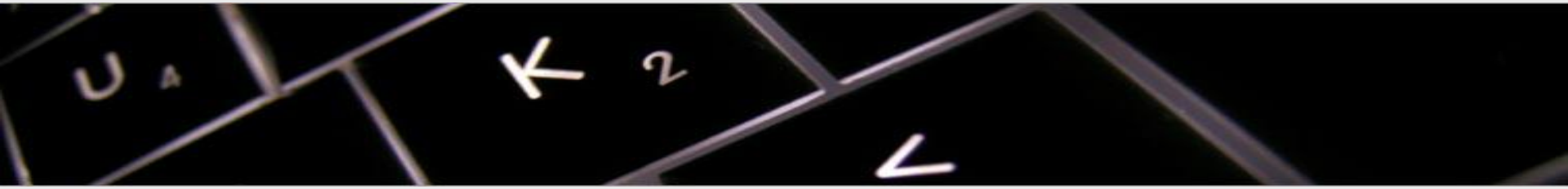

Корисник рачунара
Оператор рачунара
Улазна јединица
Излазна јединица
Командни пулт
ОМ
CPU
Кеш меморија
Спољна меморија

von Neumann-ова архитектура

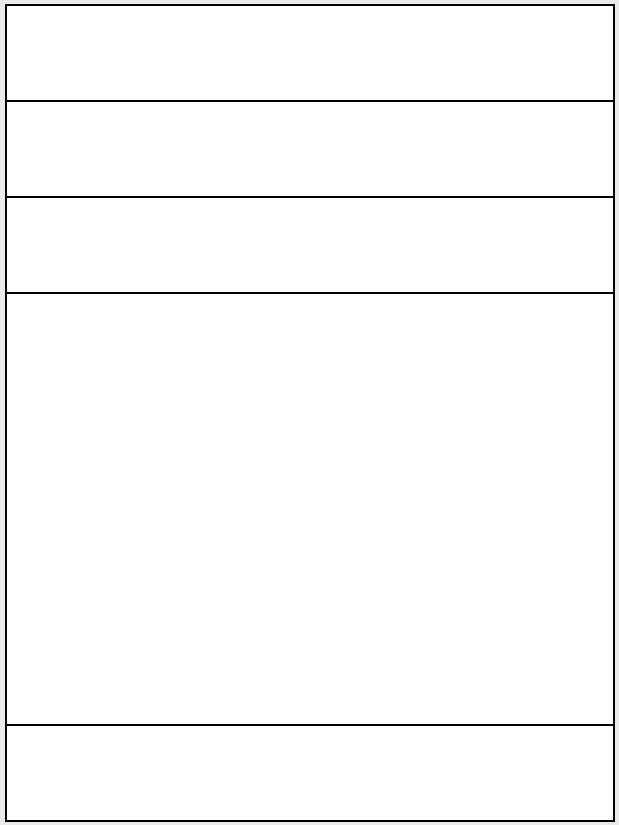


- Користи принцип програмског управљања (меморија садржи податке и програм, који су бинарно кодирани)
- Рачунар обавља инструкције, редоследом којим су наведене у програму
- Изменом програма постиже се измена функционалности

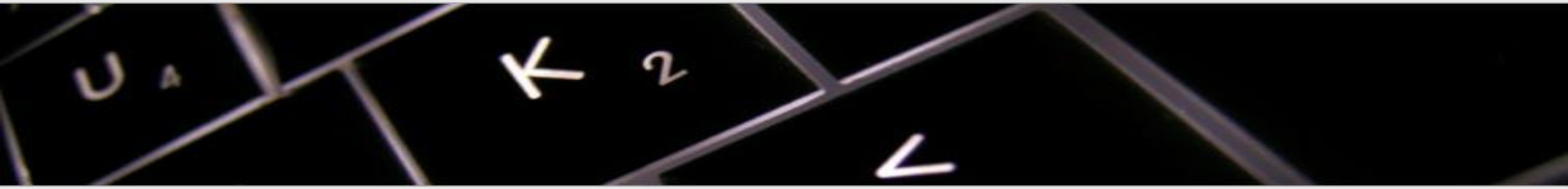
Оперативна меморија



- Низ меморијских ћелија (адресибилних јединица) са адресама 0..m-1
- Адресибилна јединица ширине n бита
- Капацитет меморије n*m



Кеш меморија

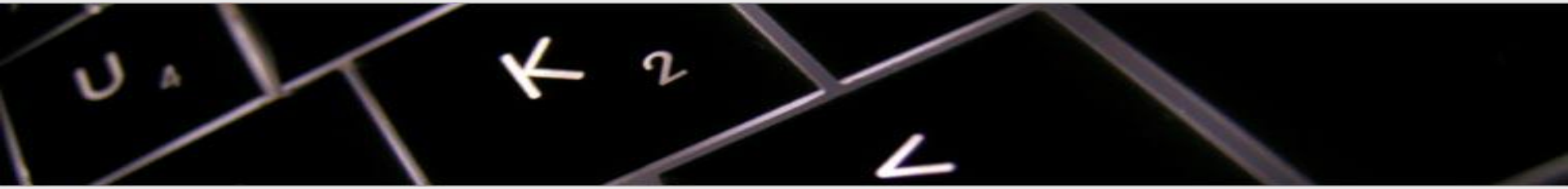


Мала меморија која служи за убрзавање приступа подацима/програму

Управљање радом је хардверско

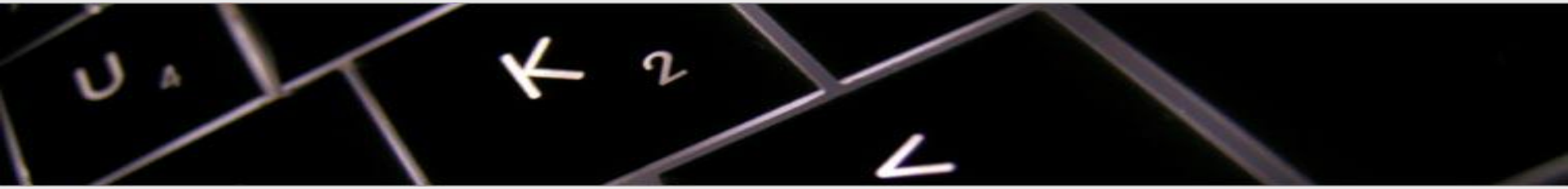
Може бити на истом чипу са процесором (“on chip”, примарни кеш) или ван њега (“off chip”,секундарни кеш)

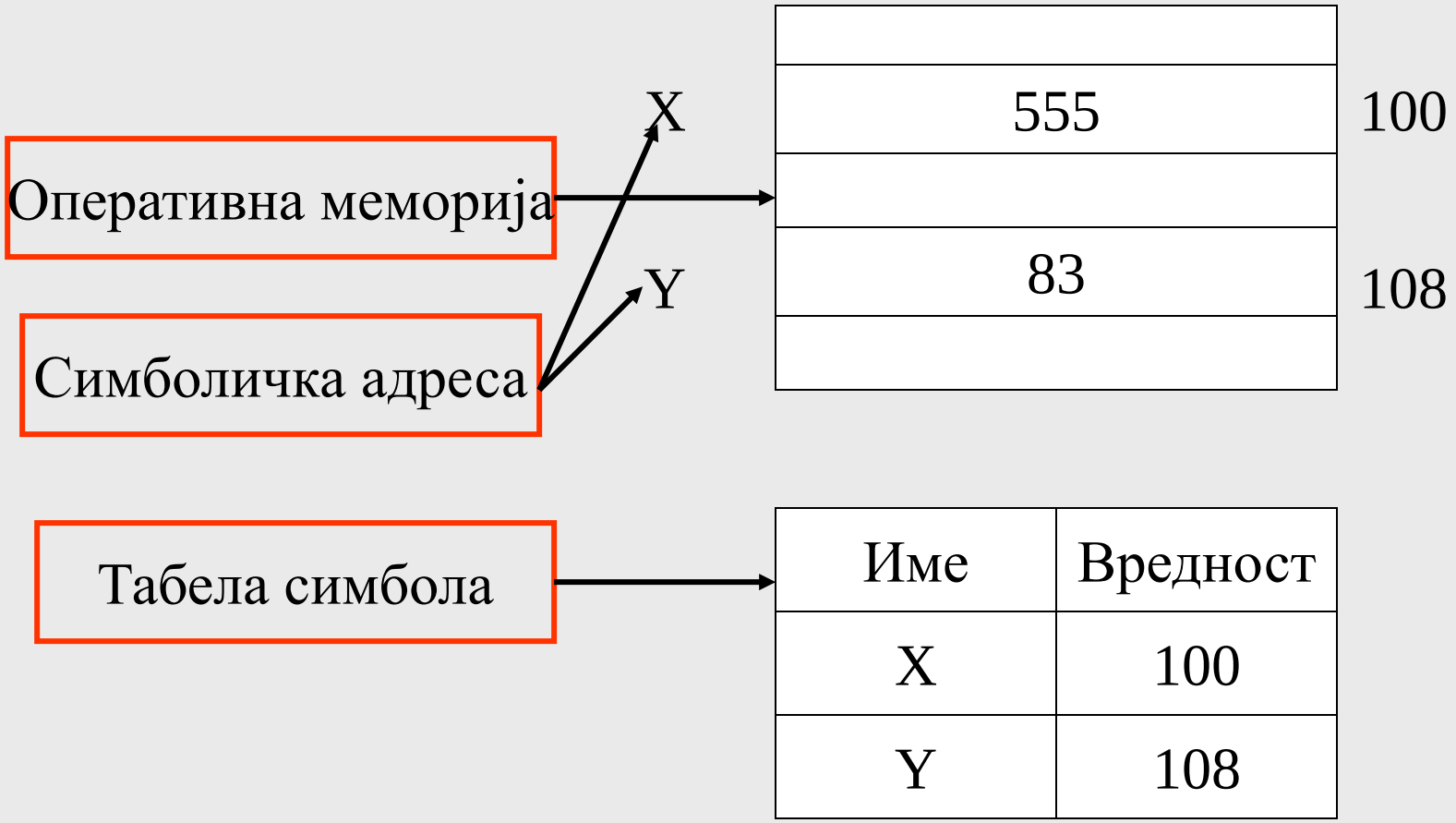
Садржај меморије



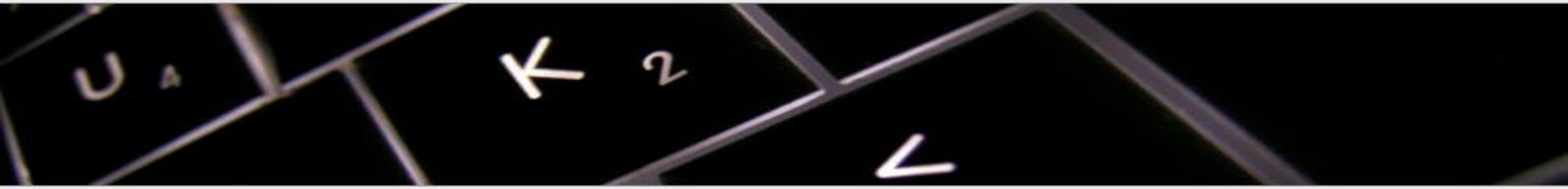
- Инструкције
- Адресе
- Цели бројеве (различите дужине)
- Бројеви у покретном зарезу
- Знакови
- Бинарно-кодирани децимални бројеви
- Структуре података

Симболичко адресирање





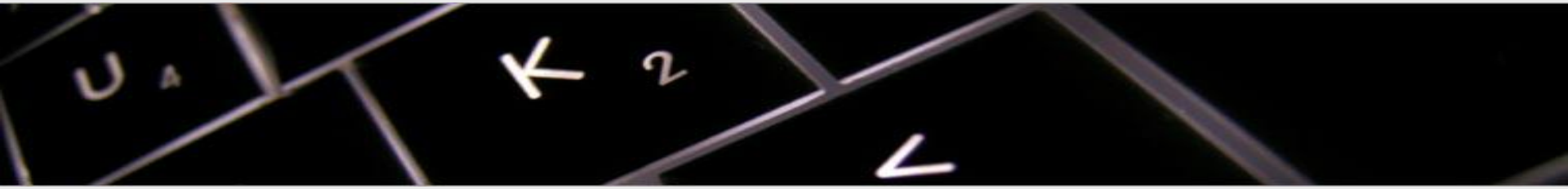
Симболичко адресирање



X може означавати било какав апстрактни објекат у некој области меморије

За карактеризацију симбола погодно је увести функције adr и val (адреса и вредност)

Симболичко адресирање



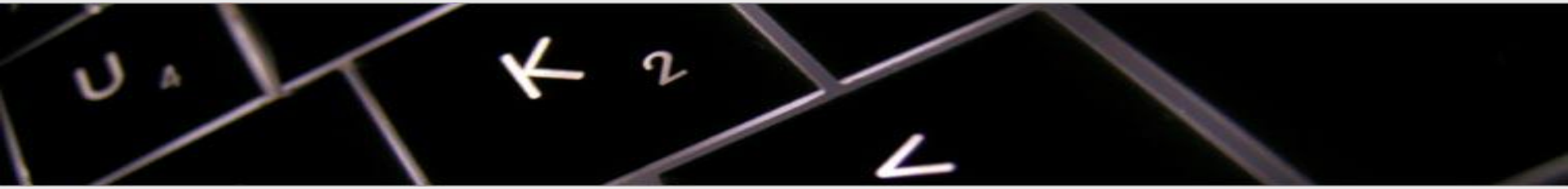
adr(X) = 100

val(X) = 555

X	555	100
Y	83	108

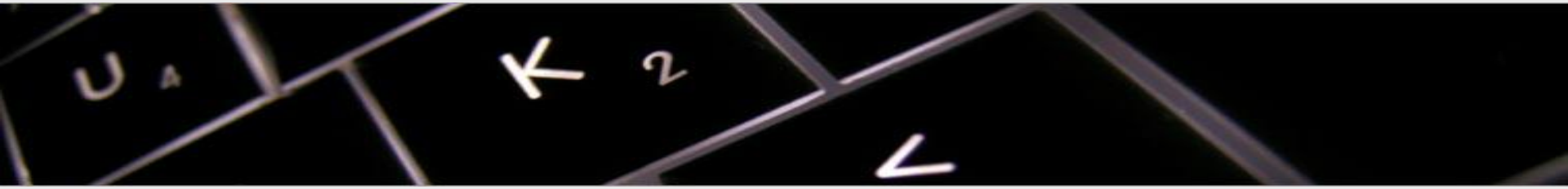
Име	Вредност
X	100
Y	108

Симболичко адресирање



- Вредност променљиве
- Вредност симбола

Симболичко адресирање

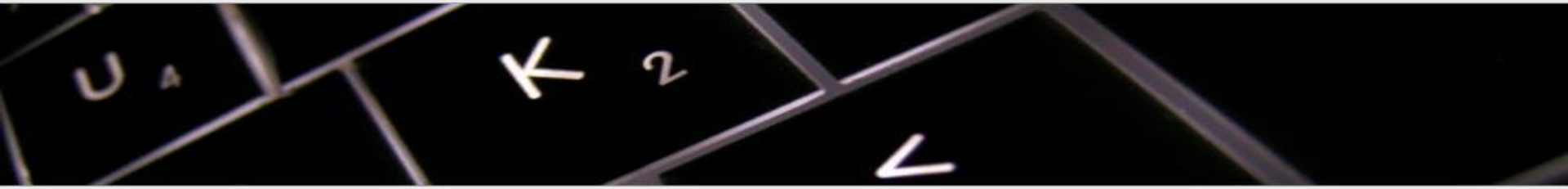


На овоме курсу ће се користити следеће означавање:

val(X) одговара X

adr(X) одговара #X

Додела вредности

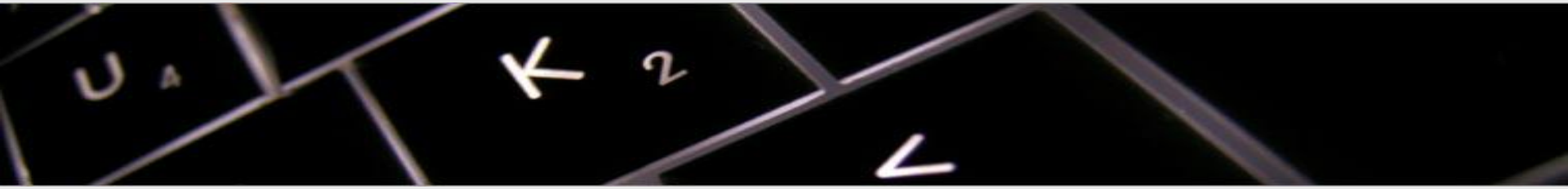


Користи се симбол доделе вредности :=

Као леви аргумент, има само један симбол са леве стране

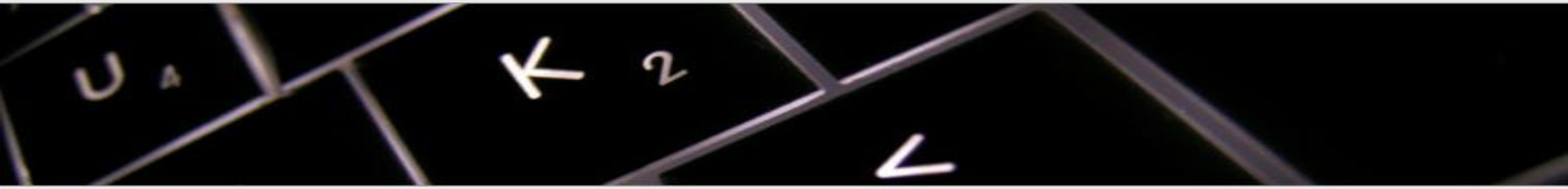
	Додела вредности	Релациони оператор
Pascal	:=	=
C	=	==
Python	= (али од верзије 3.8 и := assignment expression)	==
FORTRAN	=	.EQ.

Додела вредности



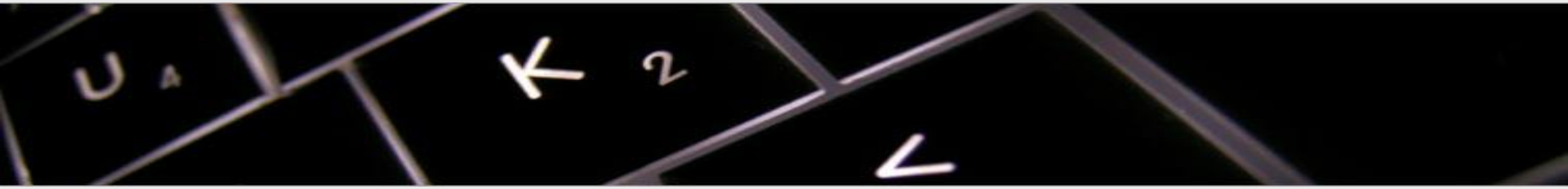
Y := X значи “преписати садржај променљиве X у променљиву Y”, ово је недеструктивна операција јер X чува своју вредност

Додела вредности



X := 123 значи “уписати вредност 123 у променљиву X”;
у овом случају 123 је константа

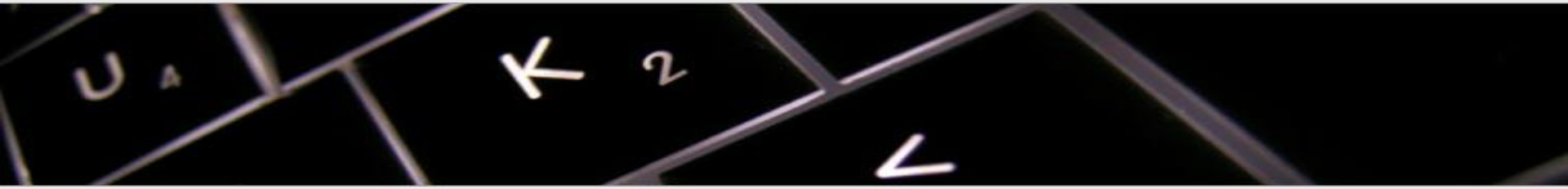
Додела вредности



123 := X – Грешка: не може се променити вредност константе

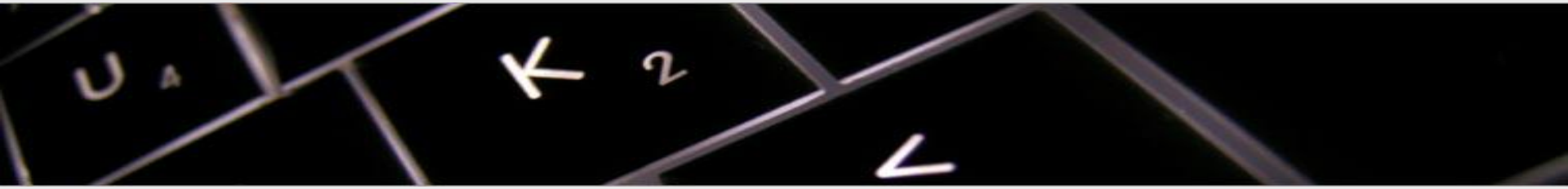
123 није “лева вредност” *(left value)*

Додела вредности



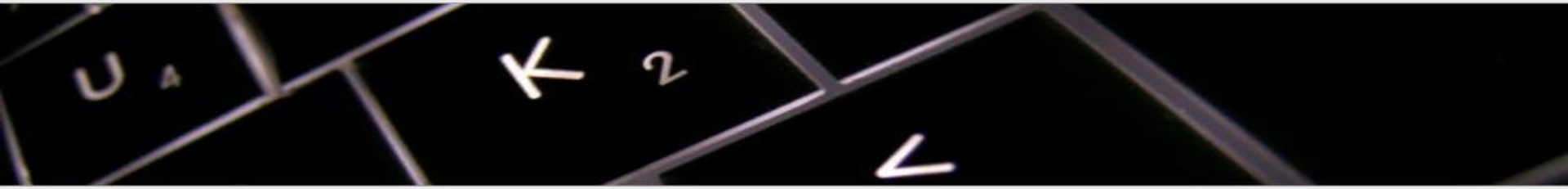
X := X+Y значи “сабрати вредности променљивих X и Y и резултат уписати у променљиву X”

Додела вредности



X+Y :=X - Грешка (X+Y је само
међурезултат сабирања;
не може му се доделити вредност)

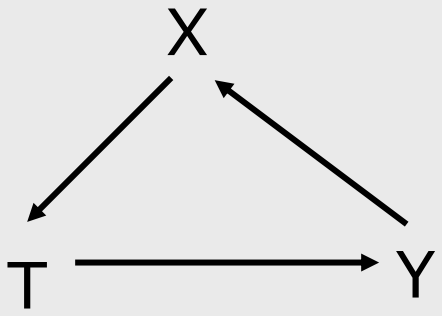
Додела вредности



Међусобна замена вредности
две променљиве:

T := X; X := Y; Y := T

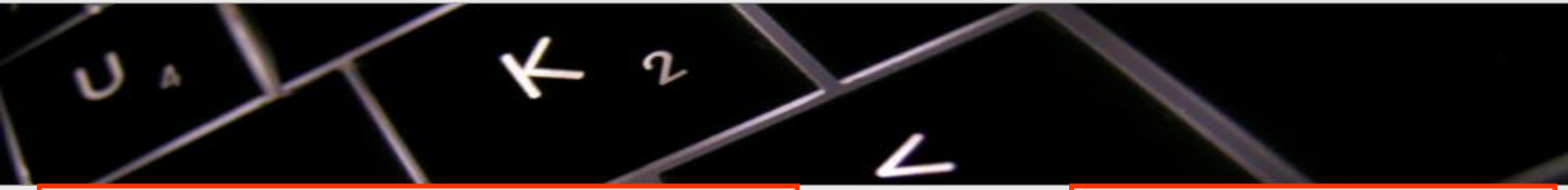
; - “раздвајач”
тј. сепаратор наредби

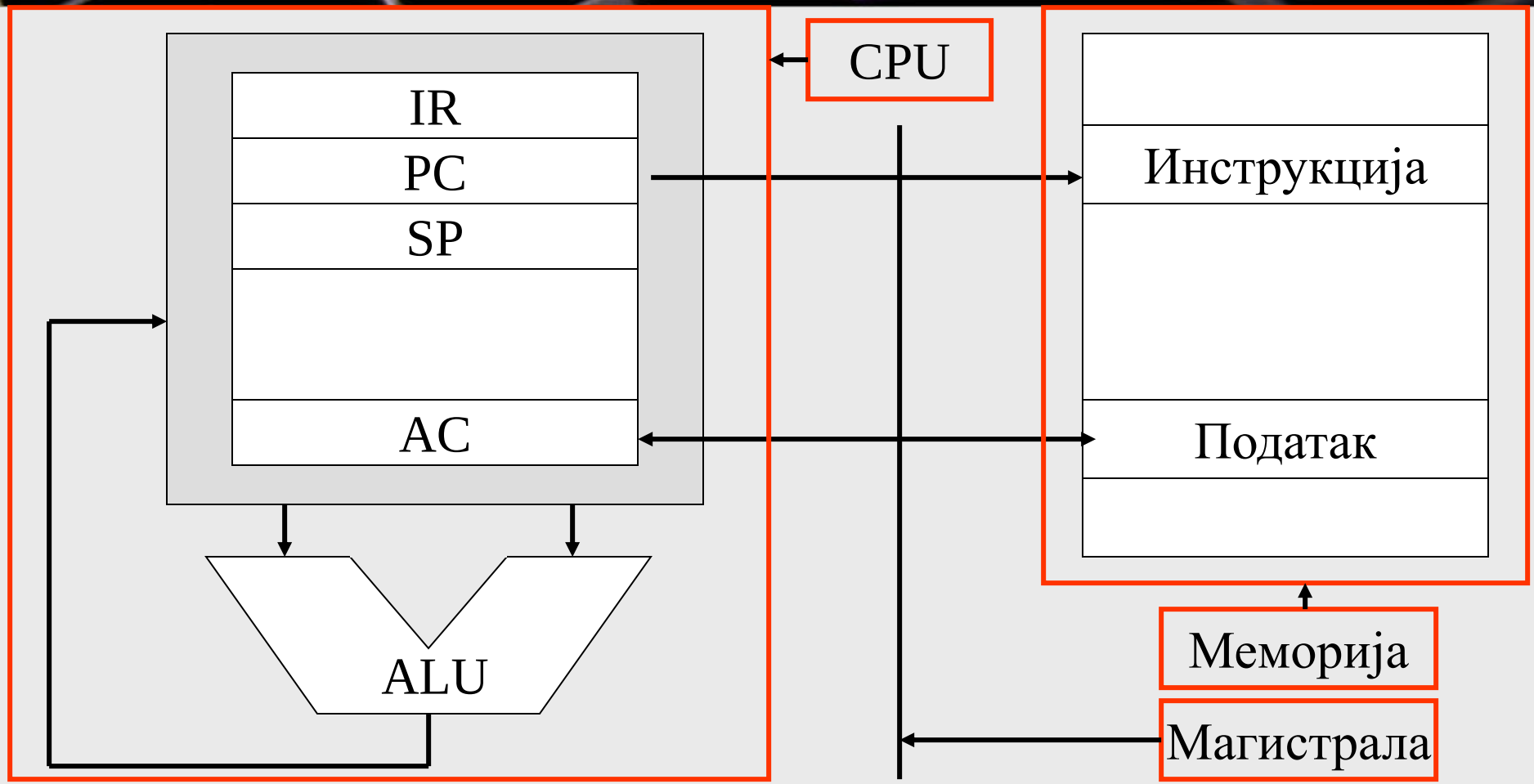


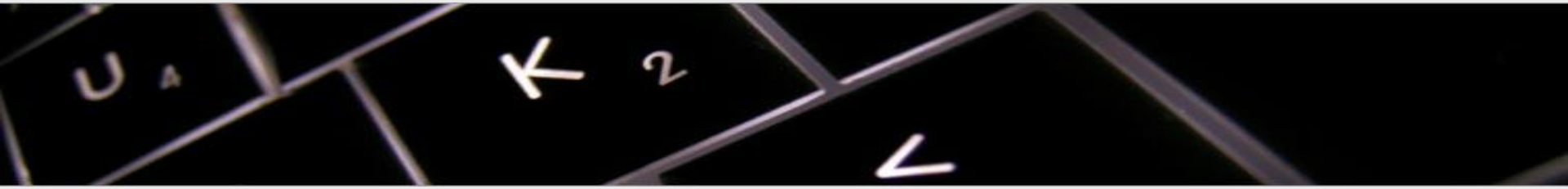
Постоји и логички (XOR) и аритметички SWAP

X := X XOR Y; Y := Y XOR X; X := X XOR Y (морају различите променљиве)

X := X + Y; Y := X - Y; X := X - Y

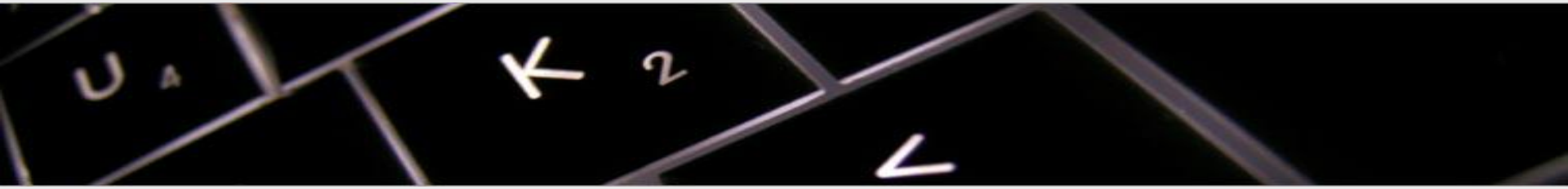



CPU
IR
PC
SP
AC
ALU
Инструкција
Податак
Меморија
Магистрала



- ALU служи за извршавање аритметичко/логичких операција
- Регистри служе за чување података са најбржим временом приступа- адресибилним приступа асемблерски програмер,интерним не!
- Контролна логика служи за координисање и управљање свим акцијама процесора

Регистри процесора



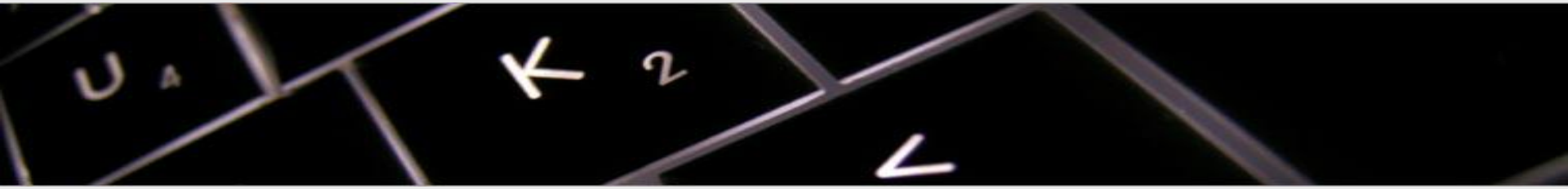
PC – програмски бројач,садржи адресу следеће инструкције

DP –“data pointer” – садржи апсолутну адресу неког податка

XR – индекс регистар,садржи релативну адресу,учествује у израчунавању апсолутне

SP – Stack Pointer – садржи адресу врха стека

Регистри процесора



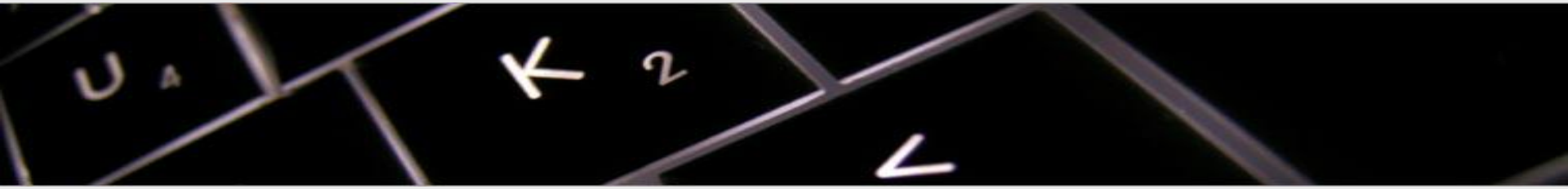
SR – Status Register – садржи податке о стању процеса нпр. флагове - прекорачење и сл.

AC – Accumulator – акумулатор

DC – Data counter – аутоматско инкрементирање

GP – General purpose register – регистар опште намене, има их више (R0,R1,...)

Регистри процесора



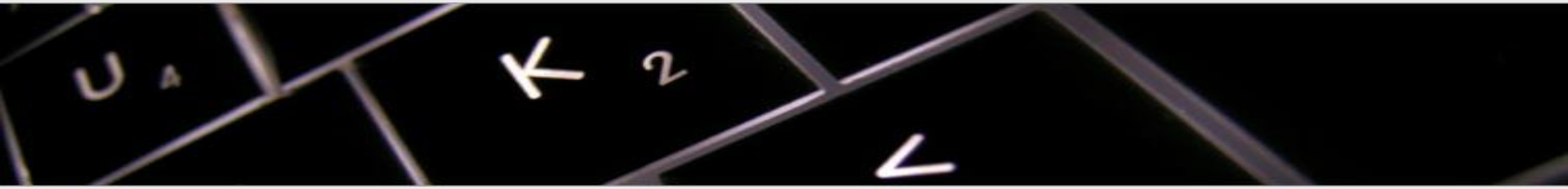
MA – Memory address register - адресни регистар меморије

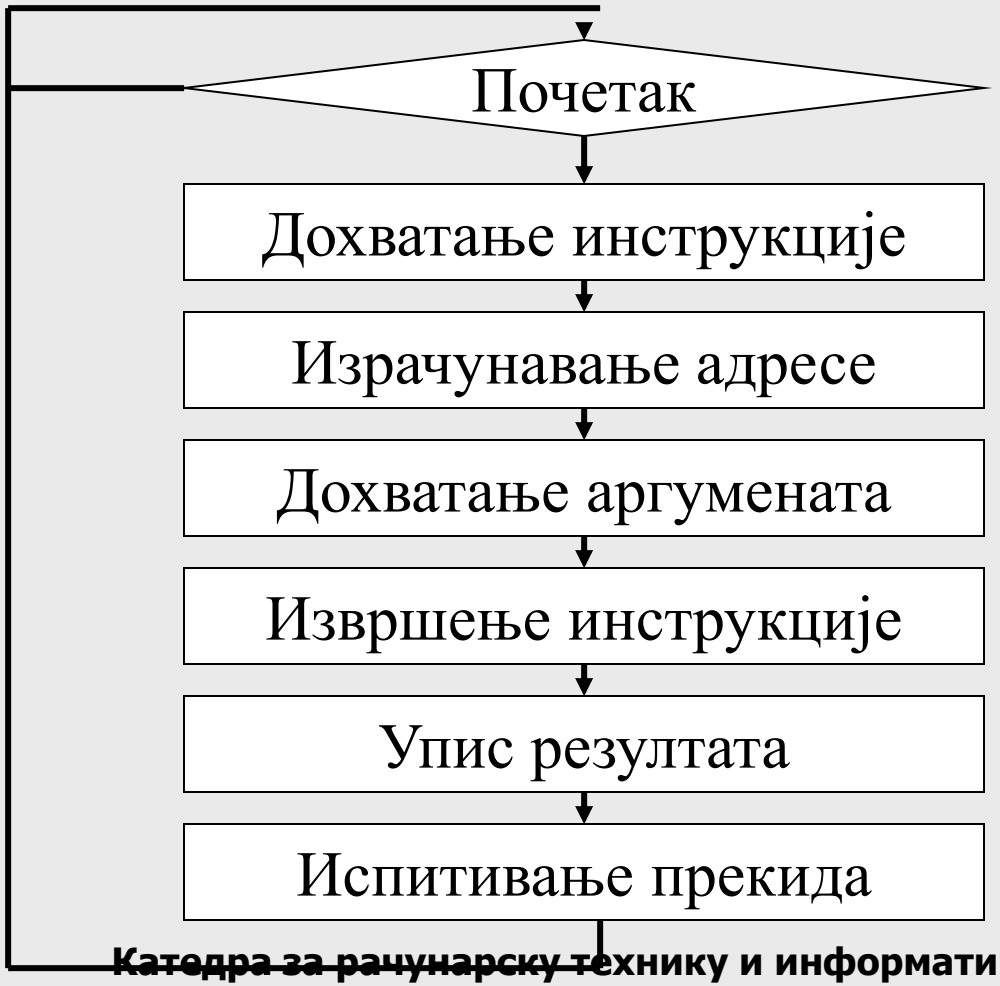
MD – Memory data register - меморијски бафер регистар или прихватни регистар меморије

IR – Instruction register - Инструкцијски регистар

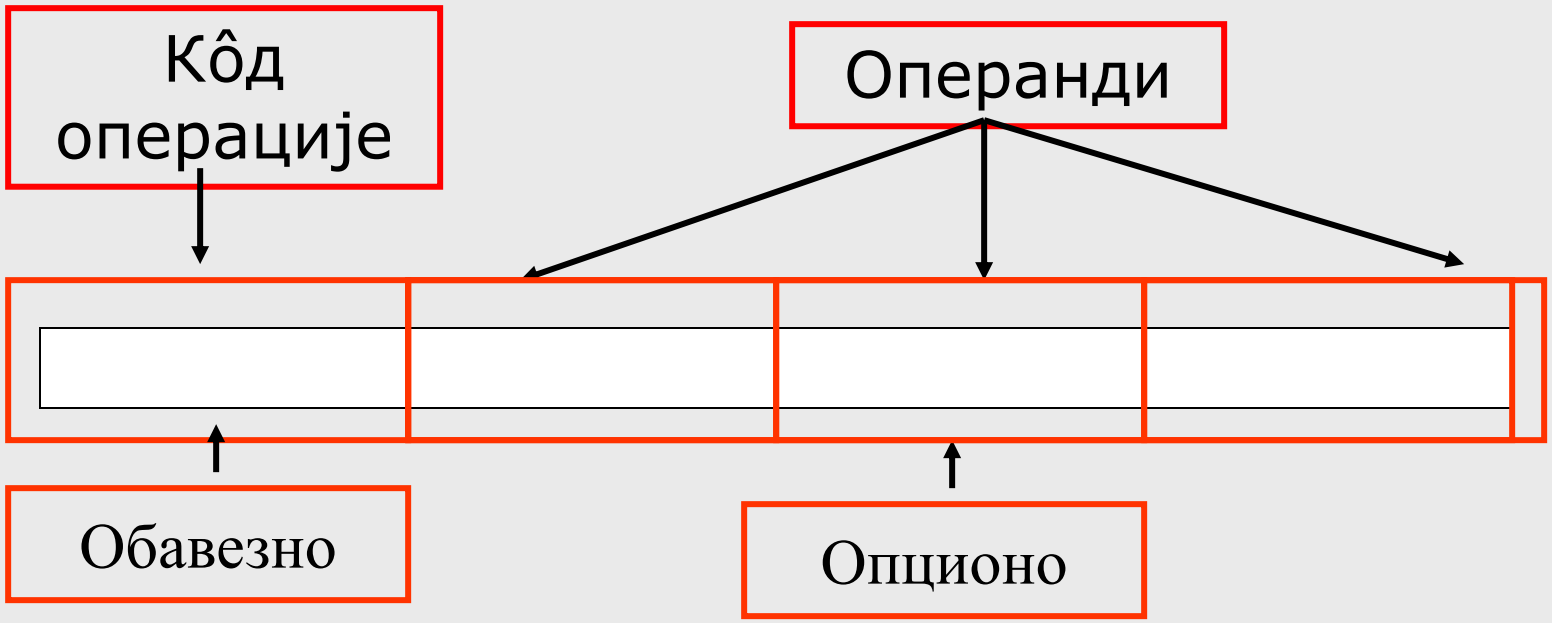
IB – Интерни бафер – помоћни регистар, нпр. смештај другог операнда

Фазе у извршењу инструкција

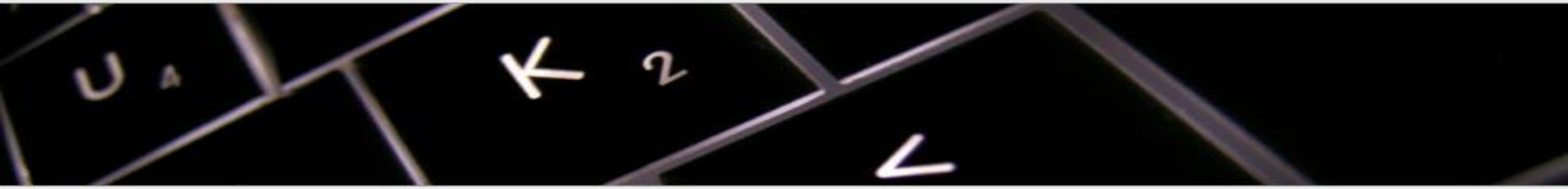




Машинске инструкције

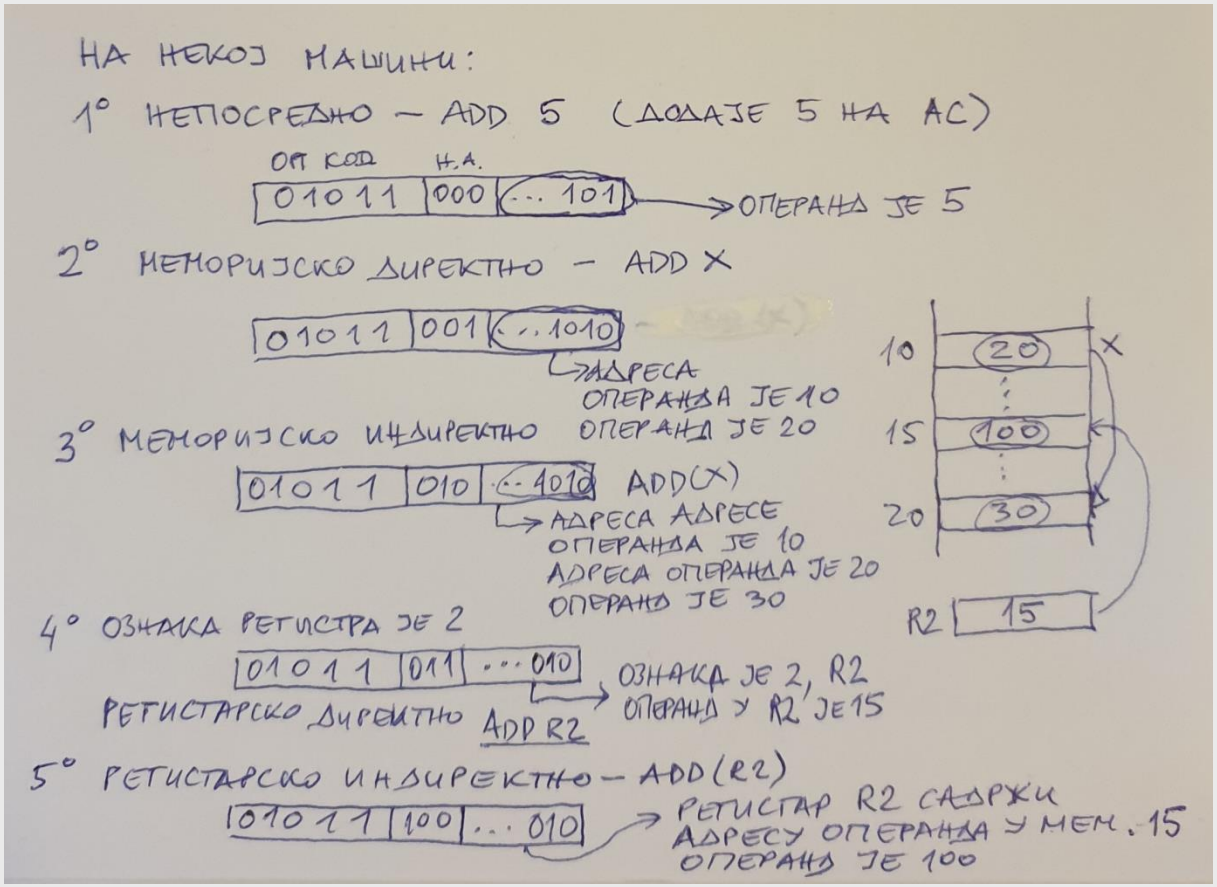


Начини адресирања

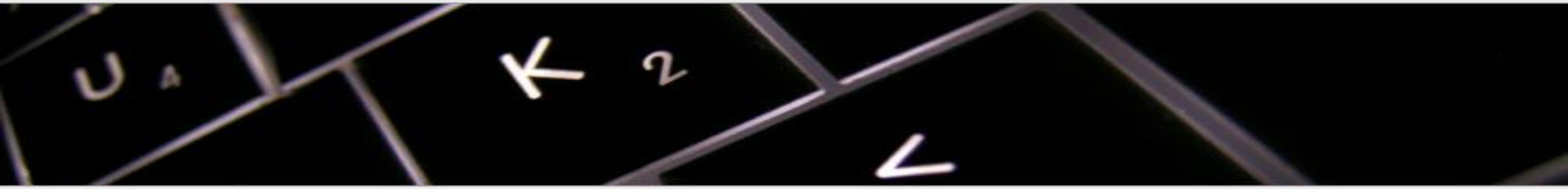


- Непосредно: у пољу је сама вредност
- Мем. директно: у пољу је апсолутна адреса податка у меморији
- Мем. индиректно: у пољу је адреса адресе податка у меморији
- Регистарско директно: у пољу адресе је “име” регистра у коме је податак
- Регистарско индиректно: у пољу адресе је “име” регистра у коме је адреса податка

Начини адресирања

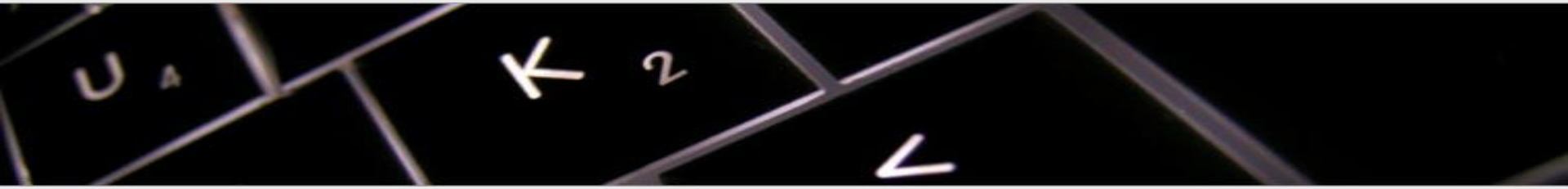


Машинске инструкције



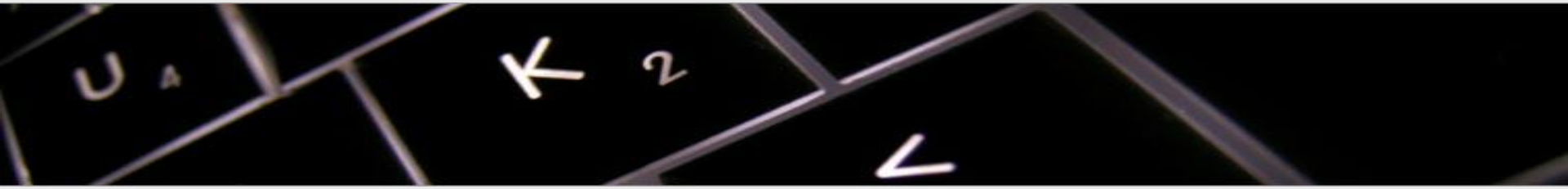
- Једноадресне
- Двоадресне
- Троадресне
- Безадресне

Једноадресне инструкције



- Један аргумент је у АКУМУЛАТОРУ
- Резултат остаје у АКУМУЛАТОРУ

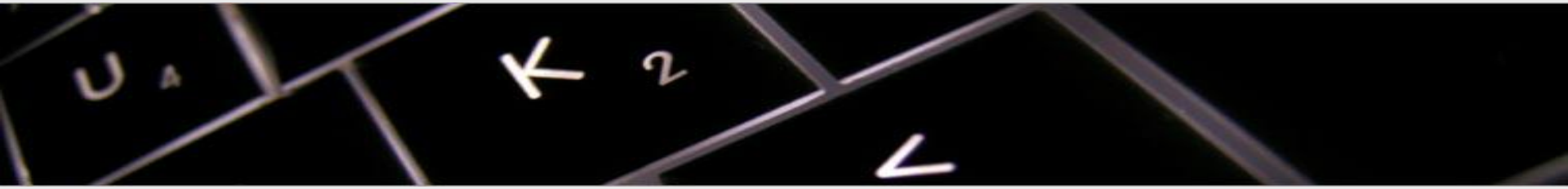
Једноадресне инструкције



Пример: Z := X + Y

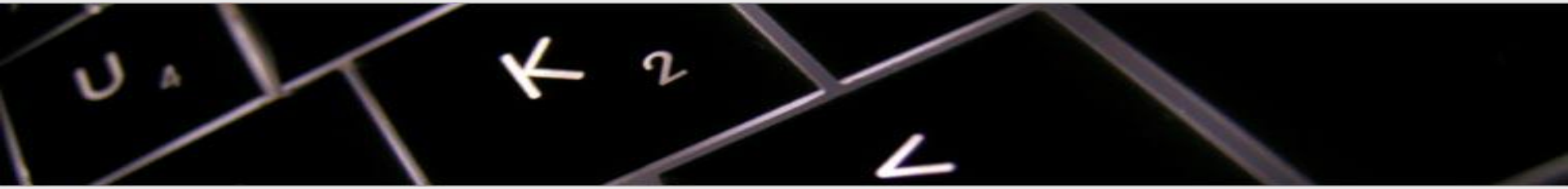
Инструкција	**Значење**
Load X	AC := X
Add Y	AC := AC + Y
Store Z	Z := AC

Двоадресне инструкције



- Оба операнда се наводе у инструкцији
- Резултат остаје на месту једног од њих

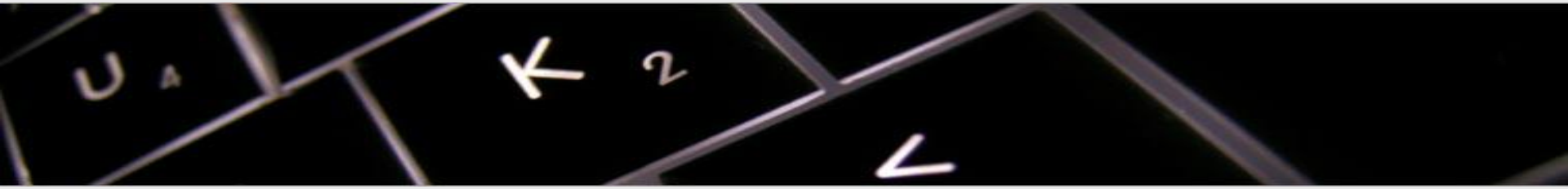
Двоадресне инструкције



Пример: Z := X + Y

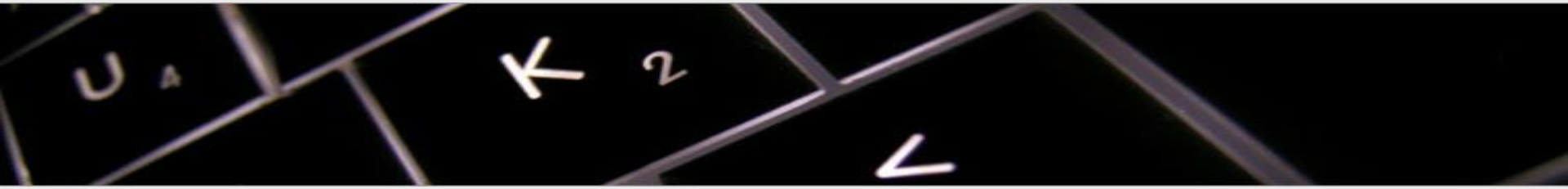
Инструкција	**Значење**
Mov Z, X	Z := X
Add Z, Y	Z := Z + Y

Троадресне инструкције



- Оба операнда се наводе у наредби
- Резултат се наводи у наредби

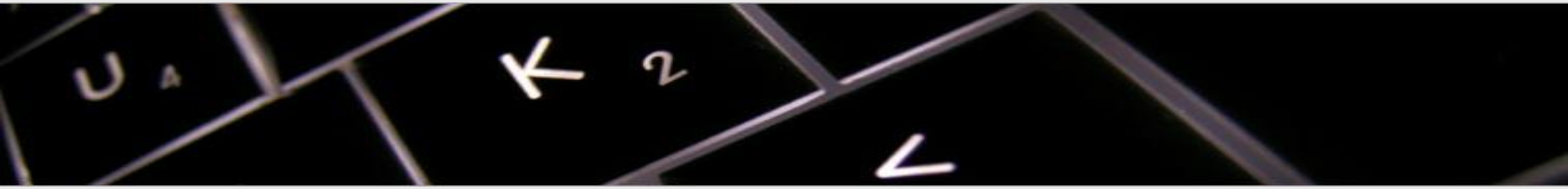
Троадресне инструкције



Пример: Z := X + Y

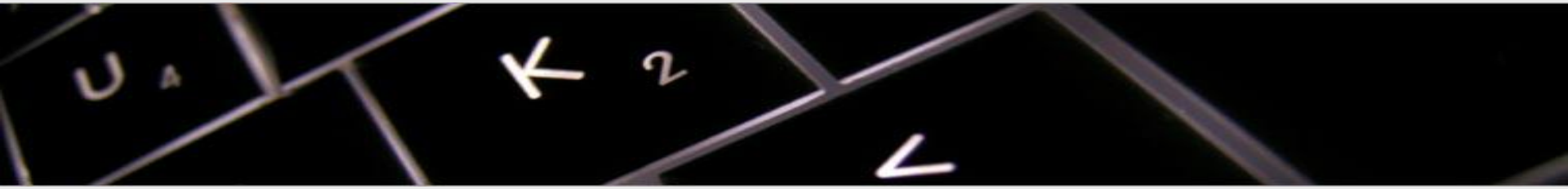
Инструкција	**Значење**
Add Z, X, Y	Z := X + Y

Безадресне инструкције



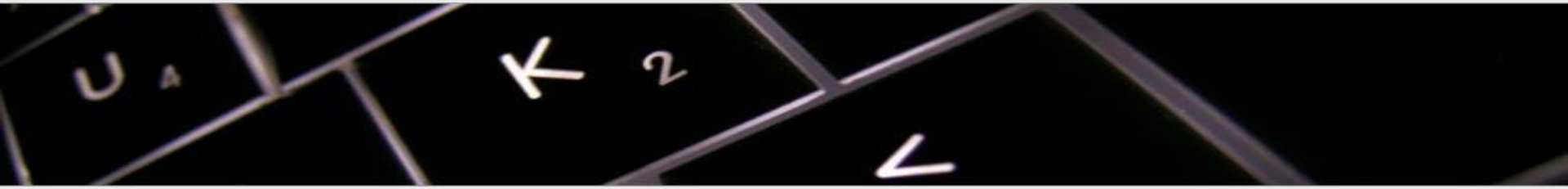
- Оба операнда на врху стека
- Резултат се смешта на врх стека
- На стеку остаје само резултат

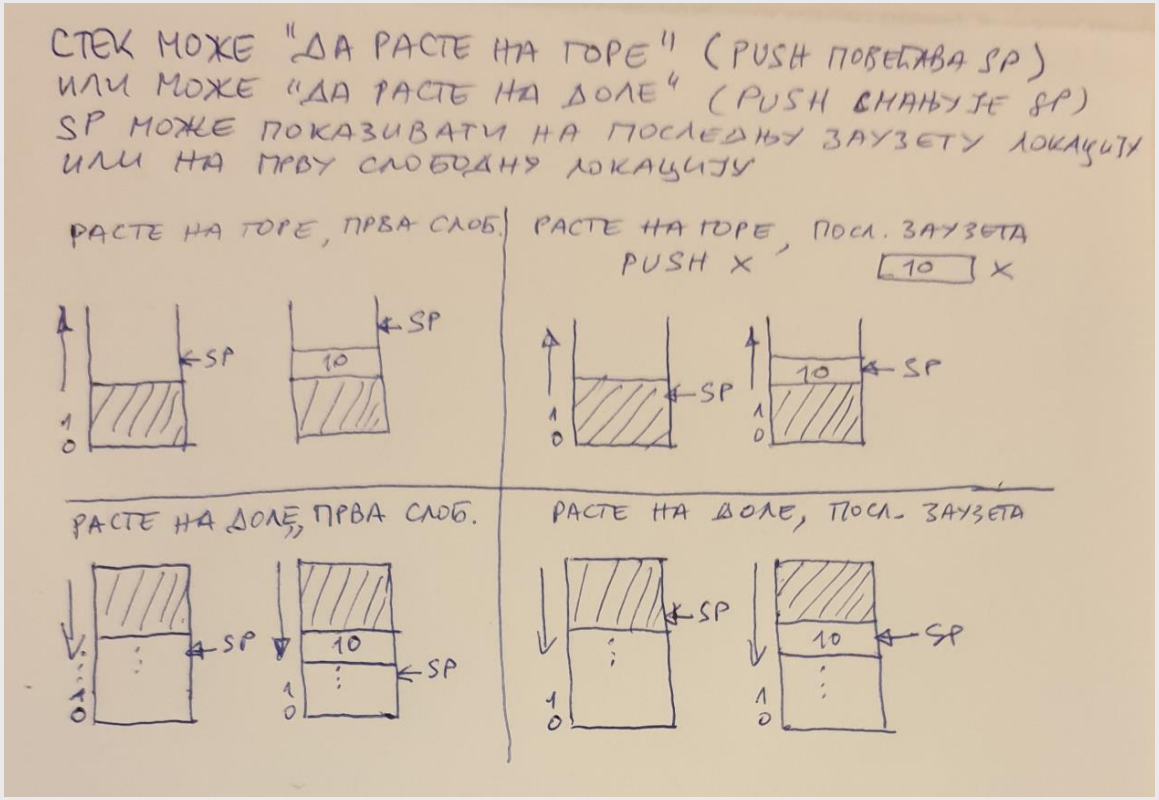
Безадресне инструкције



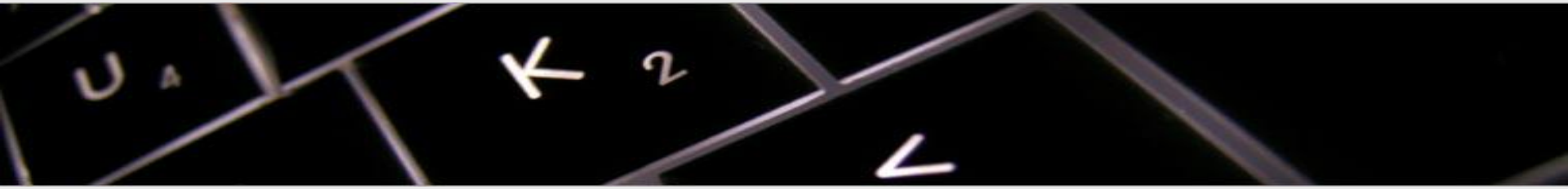
- SP показује на последњу заузету локацију на стеку
- Стек расте ка вишим адресама
- На стек се смешта/скида реч по реч

Стек



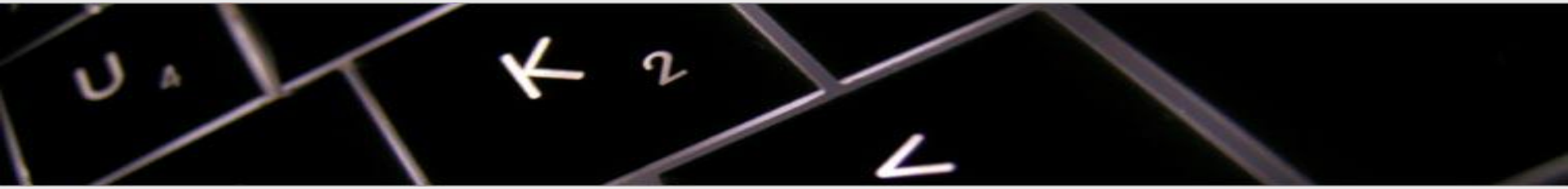

СТЕК МОЖЕ "ДА РАСТЕ НА ГОРЕ" (PUSH ПОВЕЋАВА SP)
ИЛИ МОЖЕ "ДА РАСТЕ НА ДОЛЕ" (PUSH СМАЊУЈЕ SP)
SP МОЖЕ ПОКАЗИВАТИ НА ПОСЛЕДЊУ ЗАУЗЕТУ ЛОКАЦИЈУ
ИЛИ НА ПРВУ СЛОБОДНУ ЛОКАЦИЈУ
РАСТЕ НА ГОРЕ, ПРВА СЛОБ.
SP
10
РАСТЕ НА ГОРЕ, ПОСЛ. ЗАУЗЕТА
PUSH X
10
X
РАСТЕ НА ДОЛЕ, ПРВА СЛОБ.
РАСТЕ НА ДОЛЕ, ПОСЛ. ЗАУЗЕТА

Безадресне инструкције



Инструкција	**Значење**
PUSH X	SP:=SP+1, MEM[SP]:=X
POP X	X:=MEM[SP], SP:=SP-1

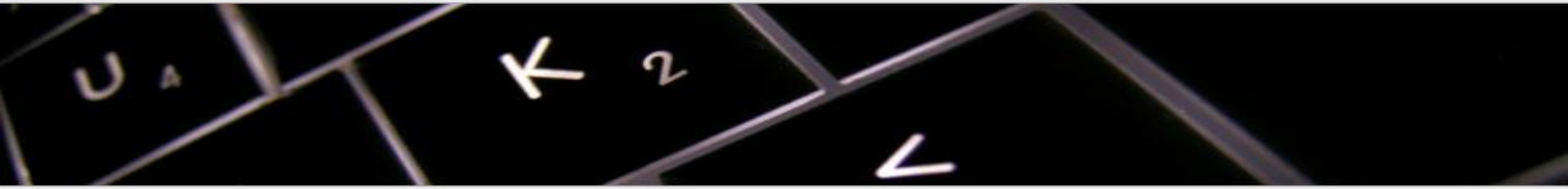
Безадресне инструкције

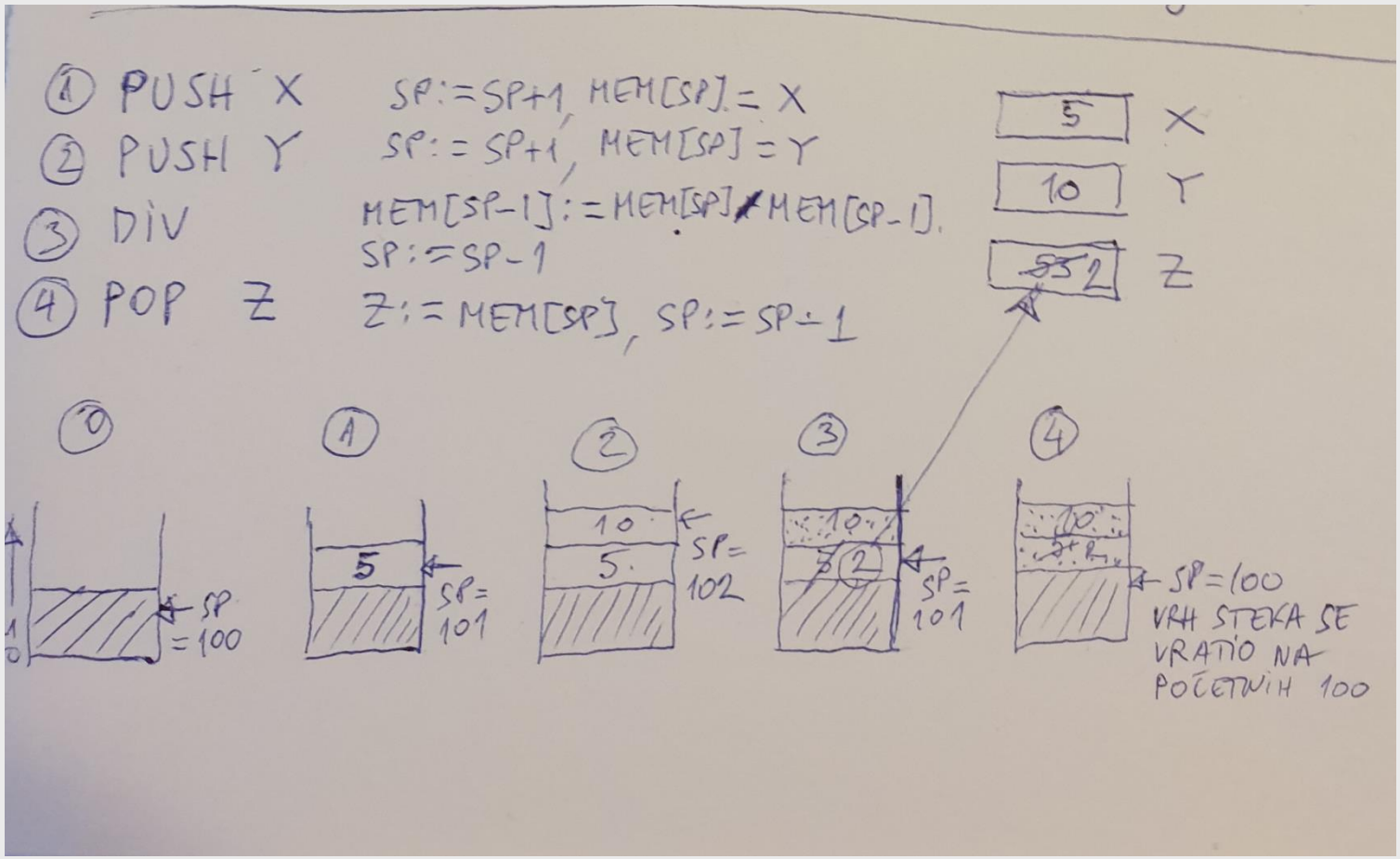


Пример: Z := X + Y

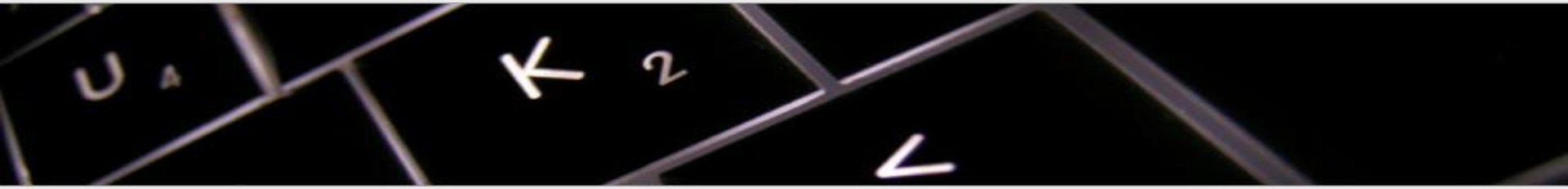
Инструкција	**Значење**
PUSH X	SP:=SP+1, MEM[SP]:=X
PUSH Y	SP:=SP+1, MEM[SP]:=Y
ADD	**MEM[SP-1]:=MEM[SP]+MEM[SP-1], SP:=SP-1**
POP Z	Z:=MEM[SP], SP:=SP-1

Дељење




1 PUSH X SP:=SP+1, MEM[SP]:=X
2 PUSH Y SP:=SP+1, MEM[SP]:=Y
3 DIV MEM[SP-1]:=MEM[SP]/MEM[SP-1], SP:=SP-1
4 POP Z Z:=MEM[SP], SP:=SP-1
5 X
10 Y
2 Z
0
1
2
3
4
SP=100
SP=101
SP=102
SP=101
SP=100
VRH STEKA SE VRATIO NA POČETNIH 100

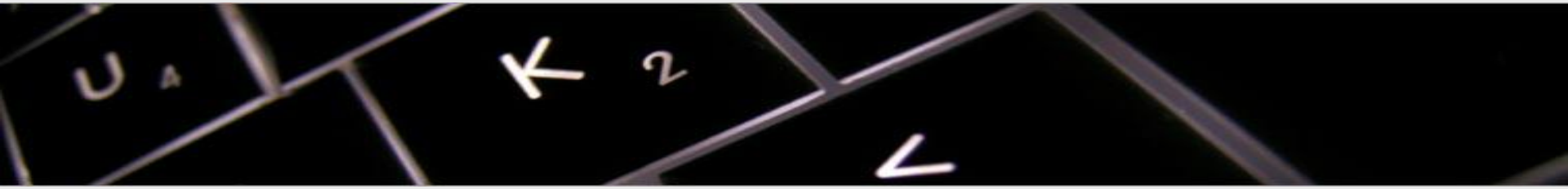
Поређење начина адресирања



Пример: Z := X + Y

Бр. адреса	3	2	1	0
Бр. инструкција	1	2	3	4

Пример 2016



Pitanje

Ako su promenljive A, B i C smeštene u memorijskim lokacijama, koliko je više mašinskih instrukcija potrebno da bi se sledeći izraz izvršio na nula-adresoj mašini nego na jednoadresnoj mašini: D := B * (A + B) + (C * A) / B. Pri izračunavanju izraza promenljive A, B i C treba da imaju svoju početnu vrednost. Broj dobijenih mašinskih instrukcija u oba slučaja treba da bude minimalan.

(®) 3 ®) 5 ®) 4

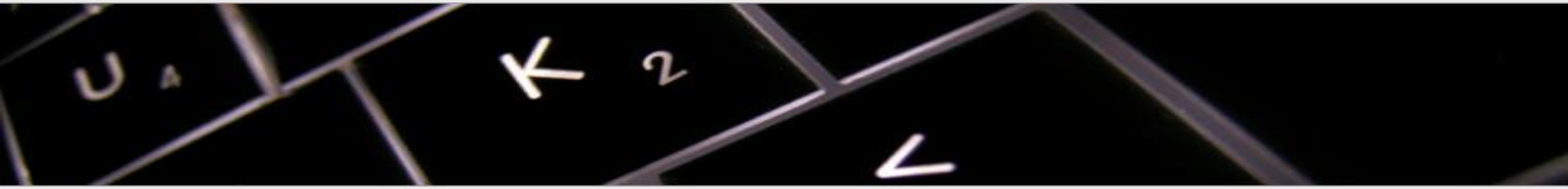
```
LOAD A
ADD B
MUL B
STORE D
LOAD C
MUL A
DIV B
ADD D
STORE D
```

```
PUSH A
PUSH B
ADD
PUSH B
MUL
PUSH B
```

```
PUSH C
PUSH A
MUL
DIV
ADD
POP D
```

12-9=3

Пример 2017



1)Ako su promenljive A, B i C smeštene u memorijskim lokacijama, koliko više mašinskih instrukcija treba da se izvrši na nula-adresnoj mašini nego na jedno-adresnoj mašini da bi se izračunao izraz D:=(A * A + B * (C - A)) / C * B. Pri izračunavanju izraza promenljive A, B i C treba da imaju svoju početnu vrednost.

A)5 **(в)** 4 C)6

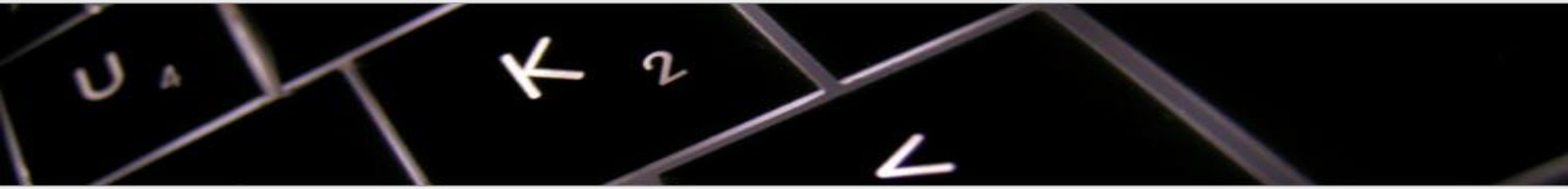
```
PUSH C
PUSH B
MUL
PUSH A
PUSH C
SUB
PUSH B
MUL
```

```
PUSH A
PUSH A
MUL
ADD
DIV
POP D
```

14-10=4

```
LOAD C
SUB A
MUL B
STORE D
LOAD A
MUL A
ADD D
DIV C
MUL B
STORE D
```

Пример 2018



1)Ako su promenljive A, B, C i D smeštene u memorijskim lokacijama, koliko više mašinskih instrukcija treba da se izvrši na jednoadresnoj (1A) mašini nego na dvoadresnoj (2A) mašini da bi se izračunao izraz E := (A * B - C) * (B - D). Pri izračunavanju izraza promenljive A, B, C i D treba da zadrže svoju početnu vrednost. Dozvoljeno je koristiti jednu pomoćnu promenljivu.

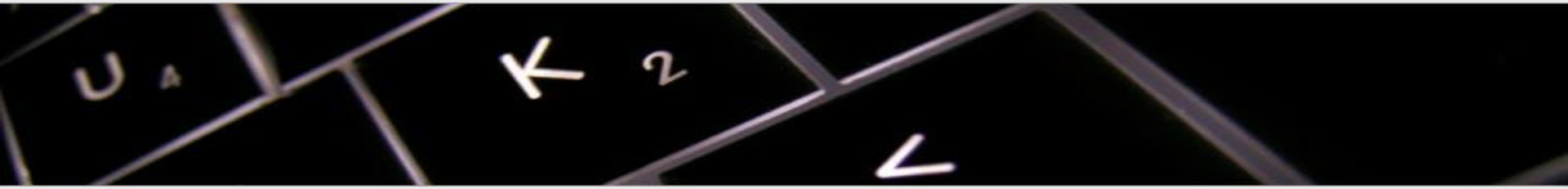
A)3 **(B)** 2 C)4

```
LOAD B
SUB D
STORE E
LOAD A
MUL B
SUB C
MUL E
STORE E
```

```
MOV E, A
MUL E, B
SUB E, C
MOV X, B
SUB X, D
MUL E, X
```

8-6=2

Пример 2019



1)Neka se posmatra kod u prilogu napisan na dvoadresnoj (2A) mašini. Koliko mašinskih instrukcija treba da se izvrši na troadresnoj (3A) mašini da bi se izračunao isti izraz? Pri izračunavanju izraza promenljive A, B, C i D treba da zadrže svoju početnu vrednost. Dozvoljeno je koristiti jednu pomoćnu promenljivu.

`MOV T, B`
`SUB T, D`
`MOV E, A`
	`DIV E, B`
	`SUB E, C`
	`DIV E, T`

A)3 **(B)** 4 C)5

E=(A/B-C)/(B-D)

SUB X,B,D
DIV E,A,B
SUB E,E,C
DIV E,E,X