



## 5.4 Адресирање

---

- Како се интерпретираат битовите од адресните полиња за да се најде операндот?
  - Адресното поле може да ја содржи мемориската адреса на операндот, определена уште во процесот на преведување
  - Но, постојат и други можности кои овозможуваат пократки спецификации и динамичко определување на адресите(!)
- Начини на адресирање (адресни режими):
  - Непосредно адресирање
  - Директно адресирање
  - Регистерско адресирање
  - Регистерско индиректно адресирање
  - Индексирано адресирање
  - Базно-индексирано адресирање
  - Stack адресирање

## 5.4.1 Непосредно адресирање (Immediate Addressing)

- **Наместо адреса, инструкцијата веќе го содржи операндот (операндот не треба да се бара ниту во меморија, ниту во регистер)!**
  - Адресниот дел од инструкцијата, всушност, го содржи самиот операнд, наместо адреса или некоја друга информација која опишува каде се наоѓа операндот (наједноставен начин за специфицирање на операнди!)
- **Непосреден операнд** (immediate operand) – автоматски се презема од меморијата со преземањето на самата инструкција
- ПРИМЕР: Инструкција која запишува константна вредност 4 во регистерот R1

MOV	R1	4
-----	----	---



## 5.4.2 Директно адресирање (Direct Addressing)

---

- **Операндот се наоѓа во меморија, а мемориската адреса е директно наведена!**
  - За операнд кој се наоѓа во меморијата, може **директно** да се наведе неговата мемориска адреса
- Инструкцијата **секогаш** ќе пристапува на истата мемориска локација (вредноста содржана на таа локација може да се менува, но не и локацијата)
  - Може да се користи само за пристап до глобални променливи чии адреси се познати уште во процесот на преведувањето



## 5.4.3 Регистерско адресирање (Register Addressing)

- **Операндот се наоѓа во регистер, а регистерот е директно наведен!**
  - Во основа, исто како и директното адресирање – но, наместо мемориска локација, се специфицира **регистер**
- Кај **load/store** архитектурите, речиси сите инструкции го користат **исклучиво** овој начин на адресирање
  - Не се користи единствено при пренесување на операнд од меморија во регистер (LOAD инструкция), или од регистер во меморија (STORE инструкция). Но, дури и тогаш, еден од операндите е регистер!



## 5.4.4 Регистерско индиректно адресирање (Register Indirect Addressing)

---

- **Операндот се наоѓа во меморија, но мемориската адреса треба да се прочита од некој регистер (регистерот е покажувач)!**
  - Операндот кој се специфицира доаѓа од меморијата (или се запишува во меморијата), но неговата адреса не е директно наведена во инструкцијата, туку е содржана во регистер
- Кога адресата се користи на овој начин, се нарекува **покажувач (pointer)**
  - Можно е обраќање до различни мемориски локации при повторните извршувања на истата инструкција(!)



## 5.4.4 Регистерско индиректно адресирање (Register Indirect Addressing)

---

- ПРИМЕР: пресметување на збирот на елементите на едно-димензионално поле (низа) од 1024 цели броеви!

	MOV R1, #0	; R1 го содржи збирот, иницијално 0 (непосреден операнд)
	MOV R2, #A	; R2 = адреса на полето (A) (непосреден операнд)
	MOV R3, #A+4096	; R3 = адреса на првиот збор после полето (непосреден операнд)
LOOP:	ADD R1, (R2)	; регистерско индиректно адресирање на операндот (преку R2)
	ADD R2, #4	; инкрементирај го R2 за еден збор (4 бајти)
	CMP R2, R3	; дали сме дошле до крајот? ("compare")
	BLT LOOP	; ако R2<R3, не е крај, продолжи! ("branch if less than")

## 5.4.5 Индексирано адресирање (Indexed Addressing)

- **Операндот се наоѓа во меморија, но адресата треба да се добие со собирање на две вредности (регистер + константа)!**
  - Мемориските локации се адресираат со наведување на регистер (експлицитно или имплицитно) и константно релативно поместување (offset)
- **ПРИМЕР1:** пристапот до локалните променливи кај IJVM се остварува со посредство на покажувач кон меморијата (содржан во регистерот LV) и релативно поместување содржано во самата инструкција)
- **ПРИМЕР2: (обратно!)** покажувачот кон меморијата може да биде содржан во самата инструкција ( $A=124300$ ), а релативното поместување ( $i$ ) во регистер ( $R2$ )

**MOV R4, A(R2) ; R4 = A[i]**

MOV	R4	R2	124300
-----	----	----	--------

## 5.4.6 Базно-индексирано адресирање (Based-Indexed Addressing)

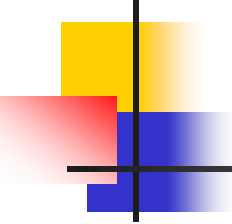
- **Операндот се наоѓа во меморија, но адресата треба да се добие со собирање на две или три вредности (регистер + регистер + константа)!**
- Мемориските адреси се пресметуваат со собирање на два регистри (база + индекс) и (незадолжително) дополнително релативно поместување (offset)

MOV	R4	R2	R5
-----	----	----	----

**MOV R4, (R2+R5)**

; ако R5 ја содржи адресата A, тогаш  $R4 = A[i]$





## 5.4.7 Stack адресирање (Stack Addressing)

---

- Крајна граница во настојувањето да се редуцираат должините на адресите е неупотребата на адреси, воопшто — **нула-адресни инструкции**
- ПРИМЕР: IADD во спрега со stack (кај JVM машината)

## 5.4.7.1 Инверзна Полска нотација (Reverse Polish Notation)

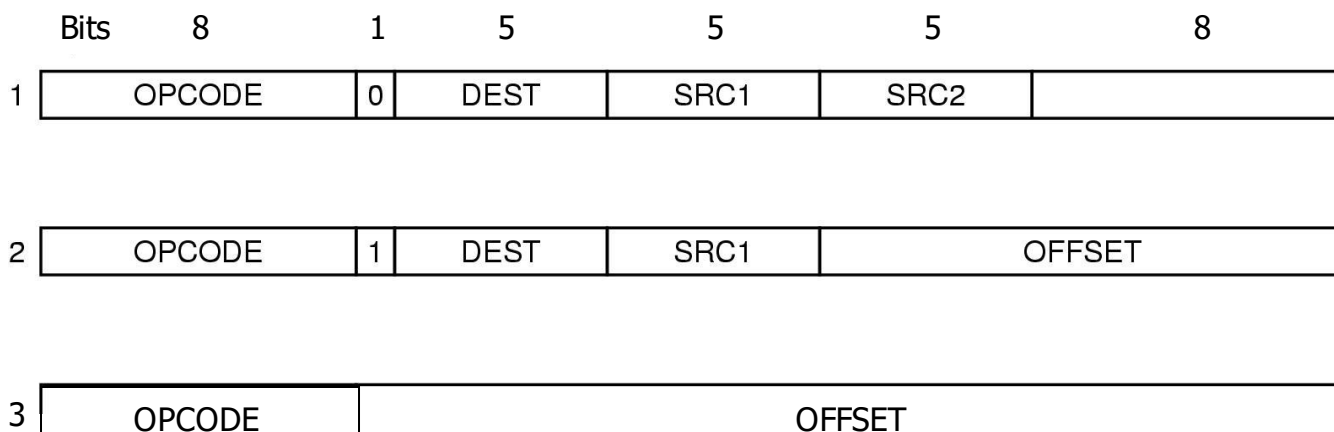
- Запишување на алгебарски формули – наместо операторот да се става помеѓу операндите (infix; inorder), тој се запишува на крајот (postfix; postorder; RPN – J.Lukasiewicz (1958))

Infix	RPN
$A+BxC$	$ABCx+$
$AxB+C$	$ABxC+$
$AxB+CxD$	$ABxC Dx+$
$(A+B)/(C-D)$	$AB+CD-/$
$AxB/C$	$ABxC/$
$((A+B)xC+D)/(E+F+G)$	$AB+ CxD+EF+G+ /$

- Како се интерпретира?
  - Штом се забележи **операнд**, истиот се запишува на врвот од stack-от (PUSH)
  - Штом се забележи **оператор**, се извршува соодветната инструкция

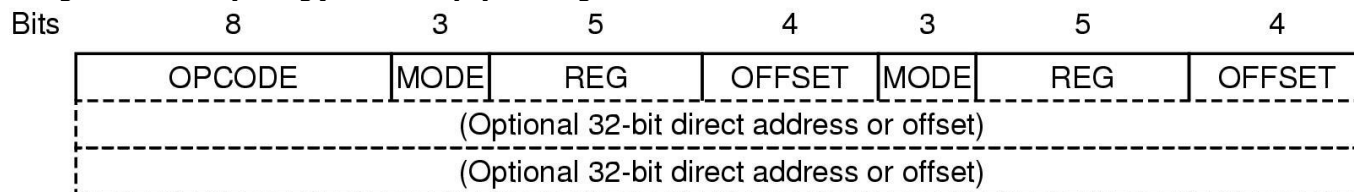
## 5.4.8 Однос помеѓу кодовите на операции и начините на адресирање

- ПРИМЕР1: едноставен дизајн на формати на инструкции кај три-адресна машина
  1. Сите аритметички и логички инструкции (три регистри)
  2. LOAD и STORE инструкции – пристап до меморија со индексан начин на адресирање (регистер SRC1 + 13-битна непосредна константа – OFFSET)
  3. Инструкции за условно разгранување (24-битен OFFSET)



## 5.4.8 Однос помеѓу кодовите на операции и начините на адресирање

- ПРИМЕР2: едноставен дизајн на формати на инструкции кај дво-адресна машина, чии операнди можат да се читаат и од меморија
  - MODE – начин на адресирање
    - 000 – непосредно
    - 001 – директно
    - 010 – регистерско
    - 011 – регистерско-индиректно
    - 100 – индексирано
    - 101 – stack адресирање
    - 110 – ...
    - 111 – ...
  - За секој директно адресиран операнд, или за индексирано адресирање со 32-битен offset, бидејќи НЕМА доволно битови, може да се користи дополнителен збор, со што (во најлош случај) инструкцијата би била долга 96 бита





## 5.5 Типови на инструкции

---

- Инструкции за копирање на податоци
- Дијадни операции
- Монадни операции
- Споредувања и условни разгранувања
- Инструкции за повикување на процедури
- Влезно/излезни инструкции



## 5.5.1 Инструкции за копирање на податоци

---

- Доделување вредности на променливи
  - $A=B$  – вредноста што се наоѓа на мемориска адреса B се копира на локација A
- 4 можни видови на копирање на податоци
  - Податокот може да доаѓа од меморија или од регистер, а може да се запишува во меморија или во регистер
- Некои компјутери имаат 4 различни инструкции за 4-те различни случаи
- Најчесто:
  - Податоците се читаат од меморија во регистер со LOAD инструкция
  - Податоците од регистер во меморија се запишуваат со STORE инструкция
  - Со MOVE инструкция податоците се пренесуваат од регистер во регистер
  - Не постои инструкция за копирање од меморија во меморија

## 5.5.2 Дијадни операции

- **Дијадни операции (Dyadic Operations)** – операции кои комбинираат два операнди за да продуцираат резултат
  - Собирање, одземање, множење и делење на цели броеви
  - Булови инструкции – кај повеќето машини, обично секогаш се присутни AND и OR, а понекогаш и XOR, NOR и NAND
  - Инструкции за работа со броеви со подвижна запирка
- ПРИМЕР1: примена на AND за **издвојување** на битови **од** збор (издвојување на вториот бајт од 4-бајтен збор)
  - 10110111 10111100 11011011 10001011                      A
  - 00000000 **11111111** 00000000 00000000                      B (маска)
  - ---
  - 00000000 **10111100** 00000000 00000000                      A AND B
  - Резултатот, потоа, може да се помести надесно за 16 бит-позиции



## 5.5.2 Дијадни операции

- ПРИМЕР2: примена на OR за **запишување** на битови **во** збор (измена на четвртиот бајт во 4-бајтен збор)
  - 10110111 10111100 11011011 10001011 A
  - 11111111 11111111 11111111 00000000 B (маска)
  - ---
  - 10110111 10111100 11011011 00000000 A AND B
  - 00000000 00000000 00000000 01010111 C
  - ---
  - 10110111 10111100 11011011 01010111 (A AND B) OR C





## 5.5.3 Монадни операции

- **Монадни операции (Monadic Operations)** – операции кои имаат еден операнд и продуцираат еден резултат
- Операции за поместување (shift) или ротирање (rotate) на содржината на еден збор или бајт
  - 00000000 00000000 00000000 **01110011**      А
  - 00000000 00000000 00000000 00**011100**      А поместено 2 бита надесно
  - **11**000000 00000000 00000000 00**011100**      А ротирано 2 бита надесно
- Поместувањето надесно, често пати, се прави со задржување на знакот (sign extension)
  - 11111111 11111111 11111111 **11110000**      А
  - **00**111111 11111111 11111111 11**111100**      **без** задржување на знакот
  - **11**111111 11111111 11111111 11**111100**      **со** задржување на знакот



## 5.5.3 Монадни операции

---

- Поместувањето може да се користи за реализација на множење и делење со степени на бројот 2
  - Ако позитивен цел број се помести налево за  $k$  битови, резултатот (без преполнување) е оригиналниот број помножен со  $2^k$
  - Ако позитивен цел број се помести надесно за  $k$  битови, резултатот е оригиналниот број поделен со  $2^k$
- Поместувањето може да се користи за забрзување на одредени аритметички операции
  - ПРИМЕР:  $18 * n = 16 * n + 2 * n$ , може да се реализира со
    - Поместување на копија на  $n$  налево за 4 бита
    - Поместување на  $n$  налево за 1 бит
    - Собирање
    - ВКУПНО: 1 копирање + 2 поместувања + 1 собирање, што често пати е побрзо од 1 множење



## 5.5.3 Монадни операции

---

- Други често користени монадни операции
  - CLR – поставување на „нула“ (со еден операнд)
  - INC – монадна форма на ADD – додавање на 1
  - NEG – монадна форма на  $(0 - X)$
- ЗАБЕЛЕШКА: дијадните и монадните операции, многу почесто се групираат според функцијата што ја извршуваат, а не според бројот на операнди
  - Аритметички операции (вклучувајќи и негација)
  - Логички операции (вклучувајќи и поместување)
  - ...

## 5.5.4 Споредувања и условни разгранувања

- Тестирање на податоци и промена на секвенцата на инструкции што треба да се извршат
  - ПРИМЕР:  $\text{sqrt}(x)$  – ако  $x$  е негативно, се појавува порака за грешка; инаку, се пресметува квадратниот корен на  $x$
- Инструкции за условно разгранување – проверуваат одреден услов и предизвикуваат разгранување (скок) на соодветна мемориска адреса ако условот е исполнет
  - Проверка дали одреден бит во машината е 0 или не (на пр. битот за знак на некој број – sign bit)
  - Проверка на битови за условни кодови (на пр. бит за преполнување – overflow bit, бит за пренос – carry bit, Z-бит – zero bit,...)
  - Споредување на два збора или два знака, со цел да се утврди дали се еднакви или не
    - Триадресна инструкция – две адреси за податоците, и една адреса на која треба да продолжи извршувањето ако условот е исполнет
    - Двоадресна инструкция – инструкция која прави споредба и поставува еден или повеќе битови за условни кодови, кои потоа ги проверува следната инструкция (Pentium II, UltraSPARC II, ...)

## 5.5.5 Инструкции за повикување на процедури

- **Процедура** – група од инструкции која извршува одредена задача и може да биде повикана од повеќе различни места во програмата (**procedure, subroutine, method**)
- Штом процедурата ќе ја изврши задачата, мора да се врати на наредбата која следува непосредно по повикот (повратна адреса – return address)
- Повратната адреса може да се зачува во:
  - Меморија
  - Регистер
  - Stack – штом ќе заврши процедурата, повратната адреса се зема од врвот на stack-от и се запишува во програмскиот бројач (се избегнуваат проблемите кои се појавуваат кога една процедура повикува друга, или кога **процедурата се повикува самата себе (рекурзија – recursion)**)

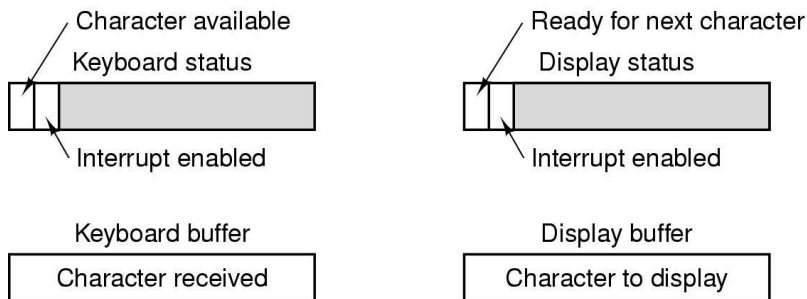
## 5.5.6 Влезно/излезни инструкции

- Најмногу се разликуваат од машина до машина
- Влезно/излезни шема кои се применуваат кај персоналните компјутери:
  - **Програмиран** влез/излез
  - Влез/излез реализиран со **прекини** (interrupts)
  - Влез/излез реализиран со **директен пристап до меморијата** (Direct Memory Access – DMA)

## 5.5.6.1 Програмиран влез/излез

- Обично, постои една влезна инструкција и една излезна инструкција
- Секоја од нив може да селектира еден од влезно/излезните уреди
- Еден единствен знак (character) се пренесува помеѓу одреден регистер во процесорот и селектираниот влезно/излезен уред
- Процесорот мора да изврши секвенца од инструкции за секој еден знак што се чита или запишува
- НЕДОСТАТОК: Процесорот поминува најголем дел од времето во повеќекратно извршување на блок од наредби, чекајќи додека влезно/излезниот уред биде спремен за читање или запишување на знак – **busy waiting (зафатено чекање)**

## 5.5.6.1 Програмиран влез/излез



```
void output_buffer (char buf[], int count)
{
    int status, i, ready;
    for (i=0; i<count; i++)
    {
        do
        {
            status = in (display_status_reg);
            ready = (status >> 7) & 0x01;
        }
        while (ready != 1);
        out (display_buffer_reg, buf[i]);
    }
}
```



## 5.5.6.2 Влез/излез реализиран со прекини

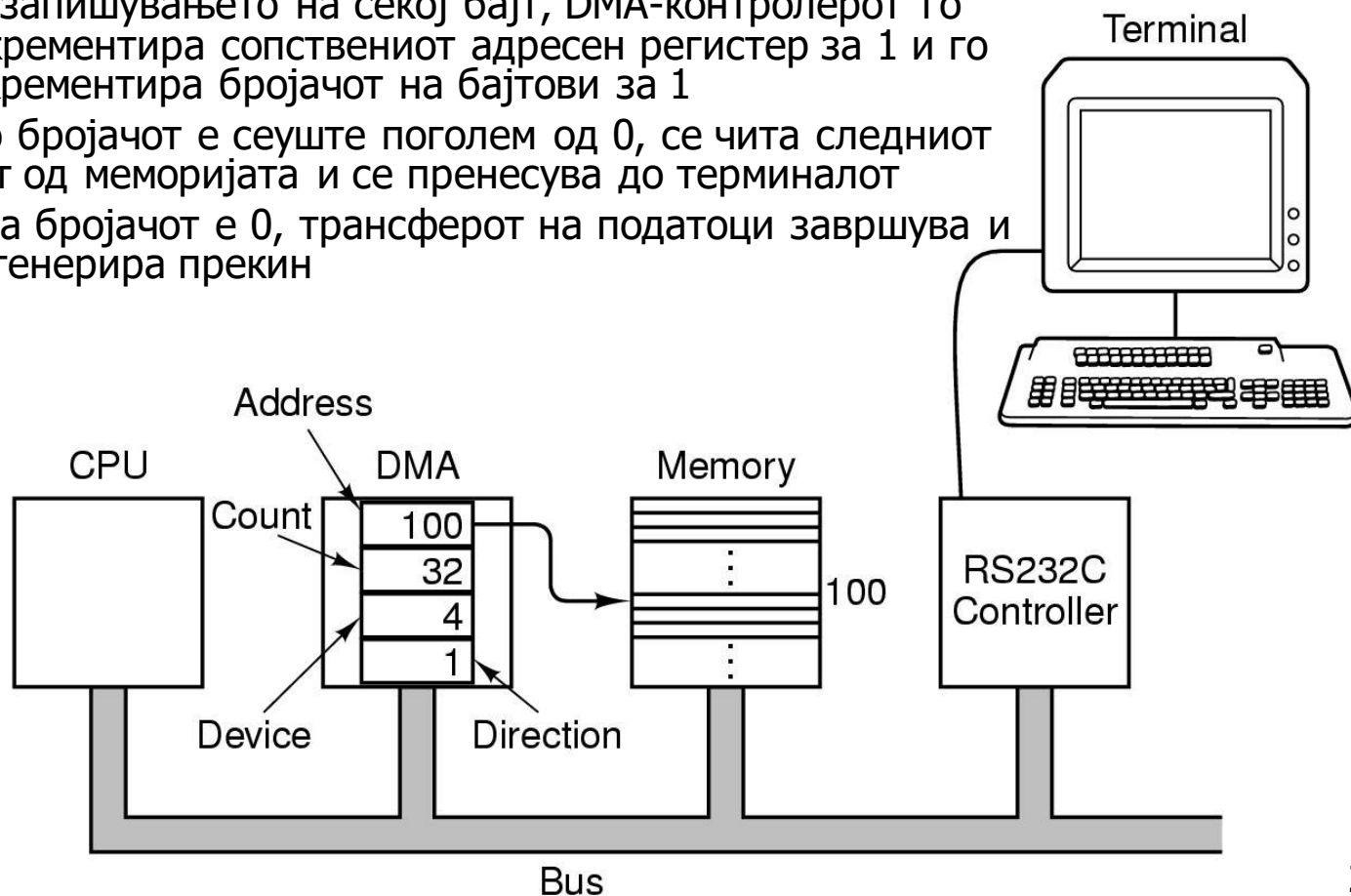
- Наместо да чека, процесорот може да иницира одреден влезно/излезен уред и да побара од него да генерира **прекин (interrupt)** кога влезно/излезната операција ќе биде комплетирана (со поставување на INTERRUPT ENABLE битот во статусниот регистер на уредот)
- НЕДОСТАТОК: За секој пренесен знак се генерира прекин, а опслужувањето на прекините не е едноставно!

## 5.5.6.3 Влез/излез реализиран со директен пристап до меморијата

- DMA-контролер – чип со директен пристап до магистралата кој има (најмалку) 4 регистри чија содржина може да се менува софтверски
  - Мемориска адреса од која се чита или запишува
  - Број на бајтови (или зборови) што треба да се пренесат
  - Идентификатор на влезно/излезниот уред
  - Вид на операцијата: читање (на пр. 0) или запишување (на пр. 1)
- ЗАБЕЛЕШКА: во споредба со процесорот, DMA секогаш има повисок приоритет за пристап до магистралата, бидејќи влезно/излезните уреди најчесто не можат да толерираат доцнења
  - **крадење на циклуси (cycle stealing)** – DMA-контролерот „краде“ циклуси на магистралата (процесорот ќе мора да чека, но барем не мора да опслужува прекини)

## 5.5.6.3 Влез/излез реализиран со директен пристап до меморијата

- ПРИМЕР: запишување на блок од 32 бајти **од** мемориска адреса 100 **на** монитор (на пр. уред број 4)
  - По запишувањето на секој бајт, DMA-контролерот го инкрементира сопствениот адресен регистар за 1 и го декрементира бројачот на бајтови за 1
  - Ако бројачот е сеуште поголем од 0, се чита следниот бајт од меморијата и се пренесува до терминалот
  - Кога бројачот е 0, трансферот на податоци завршува и се генерира прекин



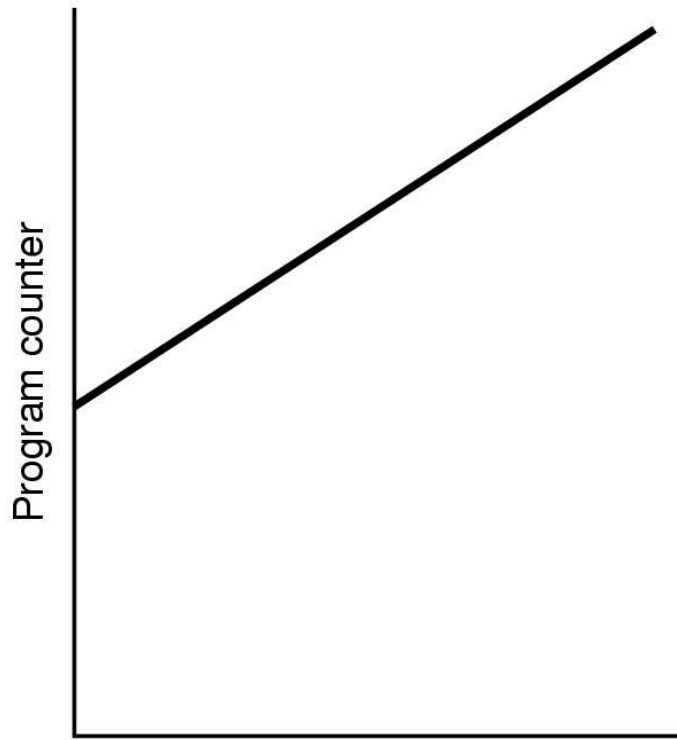


## 5.6 Тек на контролата

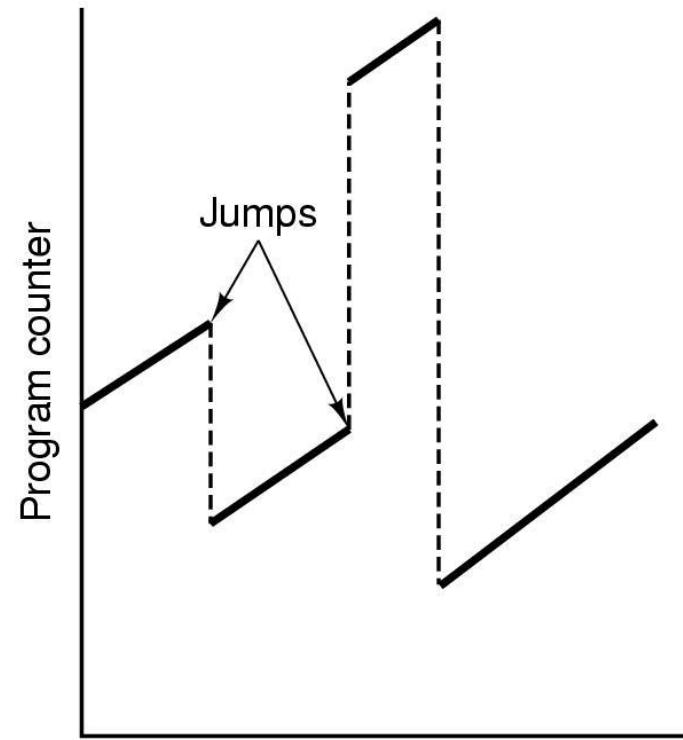
---

- **Текот на контролата (Flow of Control)** се однесува на **редоследот** по кој **динамички** се извршуваат инструкциите (за време на извршувањето на програмата)
  - Во отсуство на разгранувања и повикувања на процедури, инструкциите се извршуваат **сукцесивно** и се преземаат од **последователни** мемориски адреси
- **Повикувањата на процедурите** предизвикуваат промена на текот на контролата – извршувањето на тековната процедура запира, а започнува извршувањето на повиканата процедура
- Слични промени во текот на контролата предизвикуваат т.н. **корутини (coroutines)** – при повторно повикување на корутината, извршувањето започнува не од почеток, туку од онаа наредба каде што се застанало претходниот пат (погодно за симулирање на паралелни процеси)
- При појава на посебни услови, промена на текот на контролата предизвикуваат и т.н. **стапици (traps)** и **прекини (interrupts)**

## 5.6.1 Разгранувања

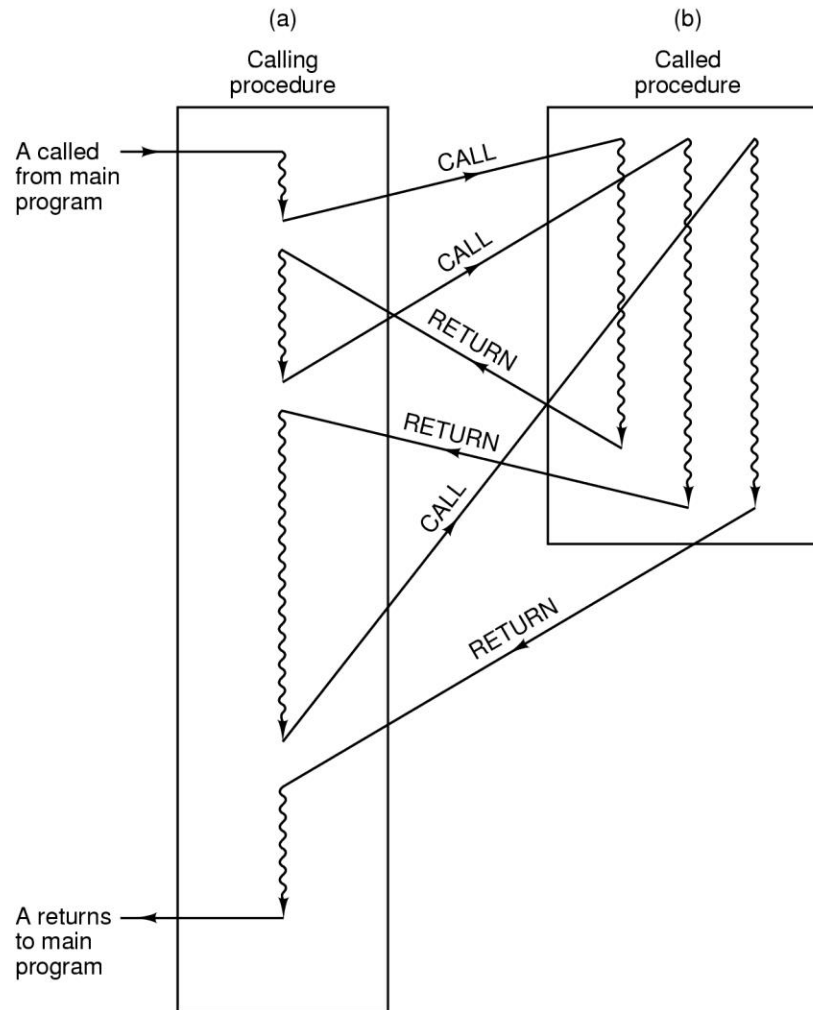


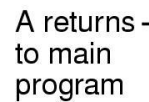
(a)



(b)

## 5.6.2 Процедури







## 5.6.4 Стапици и прекини

- **Стапица (trap)** – еден вид автоматско повикување на одредена процедура (**trap handler**), иницирано од некоја состојба предизвикана од програмата, а детектирана од хардверот или микропрограмата
  - ПРИМЕРИ: преполнување, недефиниран код на операција, иницирање на непознат влезно/излезен уред, делење со нула, ...
- **Прекин (interrupt)** – промена на текот на контролата предизвикана не од програмата која се извршува, туку од други причини (најчесто поврзани со влезно/излезните операции)
  - Прекилот предизвикува
    - **1)** запирање на програмата која се извршува,
    - **2)** запишување на програмскиот бројач и PSW (Program Status Word) на stack и
    - **3)** пренесување на контролата на т.н. **interrupt handler (опслужувач на прекилот)** кој
      - **4)** ја запишува содржината на регистрите на stack,
      - **5)** извршува одредена акција, а потоа
      - **6)** ја враќа контролата на прекинатата програма





## 5.6.4 Стапици и прекини

---

- РАЗЛИКА:

- стапиците се **синхрони** со програмата (предизвикани директно од неа) – при исти влезни податоци, секогаш се појавуваат на исто место во програмата, при секое повторно извршување
- прекините се **асинхрони** – нивното појавување варира и зависи од моментот кога ќе се иницира влезно/излезната операција (на пример, **кога точно** операторот на тастатурата ќе притисне Enter)