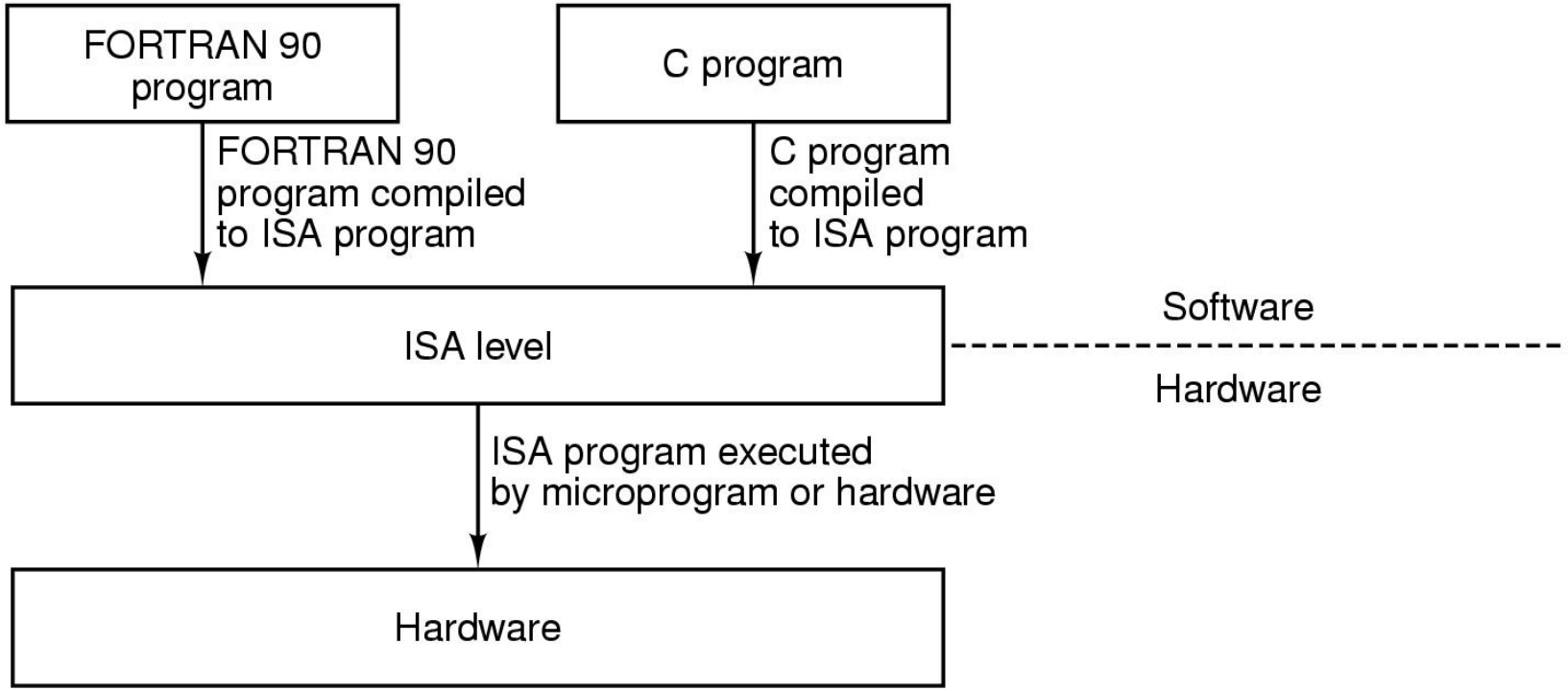
5 Ниво на архитектурата на инструкциското множество (L2)

- Интерфејс (посредник) помеѓу софтверот и хардверот
  - Програмите напишани во различни јазици од високо ниво се преведуваат до една заедничка форма – **ниво на архитектурата на инструкциското множество (ISA)** – хардверот, потоа, непосредно ги извршува
  - „архитектура на машината"; „асемблерски јазик" (некоректно!)
- При појавата на нова машина, најчесто поставувани прашања од страна на потенцијалните корисници се:
  - „Дали е компатибилна со нејзиниот претходник?"
  - „Дали на неа можат да се извршуваат истите оперативни системи?"
  - „Дали на неа можат да се извршуваат сите постојни апликативни програми, без дополнителни измени?"

5 Ниво на архитектурата на инструкциското множество (L2)

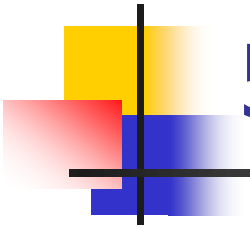
- Токму поради тоа, архитектите настојуваат архитектурата на инструкциското множество да биде **наназад компатибилна (backward compatible)**
  - Преминот од дизајн со микропрограма кон директно хардверско извршување, или додавањето на нов проток (pipeline), суперскаларни карактеристики (или, што и да е друго), не претставува проблем се додека архитектурата на инструкциското множество е **компатибилна** со претходната

5 Ниво на архитектурата на инструкциското множество (L2)



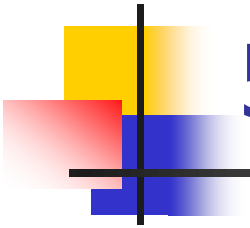
5.1.1 Карактеристики на нивото на архитектурата на инструкциското множество

- За да може да произведе кòд на нивото на инструкциското множество, оној што пишува преведувач (compiler) мора да знае:
  - Каков е **меморискиот модел**?
  - Колку и кои **регистри** се на располагање?
  - Кои **типови на податоци** се на располагање?
  - Кои **инструкции** се на располагање?
- Сите овие информации, заедно, го дефинираат (определуваат) **нивото на архитектурата на инструкциското множество**
- Кај повеќето машини, се разликуваат најмалку два режими (начини) на работа:
  - **Kernel mode** – наменет за извршување на оперативниот систем и овозможува извршување на **сите** инструкции
  - **User mode** – наменет за извршување на апликативните програми и не дозволува извршување на одредени „чувствителни" инструкции (на пример, инструкции кои директно манипулираат со кеш меморијата)



5.1.3 Регистри

- Категории на регистри
  - Регистри со **посебна** намена (special-purpose registers)
  - Регистри со **општа** намена (general-purpose registers) – за чување на вредности на локални променливи и меѓурезултати од пресметувањата, заради брз пристап без обраќање до меморијата
    - Вообичаено, RISC машините имаат најмалку 32 регистри со општа намена



5.1.4 Инструкции

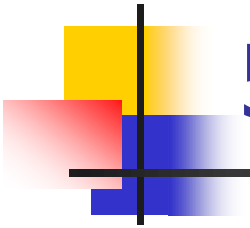
- Множество машински инструкции – главна карактеристика на нивото на архитектурата на инструкциското множество
  - LOAD / STORE инструкции (во една или во друга форма) – за пренесување на податоци помеѓу меморијата и регистрите
  - MOVE инструкции – за копирање на податоци помеѓу регистрите
  - Аритметички инструкции
  - Логички (Boolean) инструкции
  - Инструкции за споредување на податоци и разгранување

5.1.5 Кус осврт кон Java Virtual Machine (JVM)

- Мемориски модел – исто како и кај IJVM, дополнет со уште едно подрачје
  - Рамка на локални променливи
  - Stack за операнди
  - Подрачје на методи
  - Подрачје на константи
  - **Heap („Куп“)** – за чување на динамички или мошне големи објекти
    - Пример (Java):
      - int a[] = new int [4096]
      - Се алоцира парче од heap-от и се враќа покажувач кон него
      - ЗАБЕЛЕШКА: сите други мемориски обраќања се остваруваат со релативно поместување (offset) во однос на еден од регистрите LV, SP, PC или CPP (без посредство на покажувачи)

5.1.5 Кус осврт кон Java Virtual Machine (JVM)

- **Garbage collector („собирач на отпадоци“)** – парче софтвер кое бара објекти на heap-от кои повеќе не се во употреба, со цел да ослободи дополнителен мемориски простор (при појава на преполнување на heap-от)
- JVM **нема** регистри со општа намена – чисто stack-организирана машина, со голем број мемориски обраќања, но со едноставна и елегантна архитектура на инструкциското множество

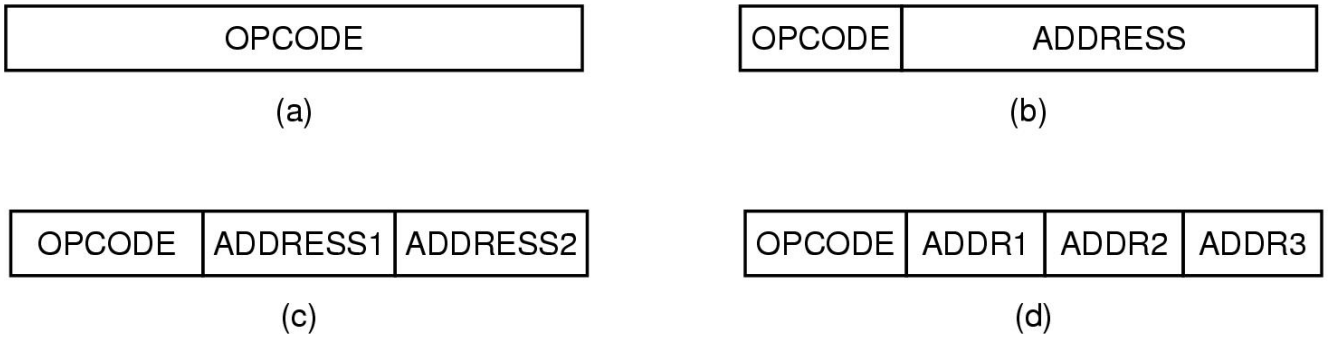


5.2 Типови на податоци

- Нумерички типови на податоци
  - Цели броеви (integer) – 8, 16, 32 или 64 бита
    - 32-битен цел број без знак (unsigned): 0 до 2^{32}-1
    - 32-битен цел број со знак: -2^{31} до 2^{31}-1
  - Броеви со подвижна запирка (floating-point) – 32, 64 или 128 бита
  - Бинарно кодирани декадни броеви (binary coded decimals; BCD)
- Ненумерички типови на податоци
  - Знаци (character)
  - Низи од знаци (string)
  - Логички вредности (Boolean)
  - Покажувачи (pointer) – машински адреси

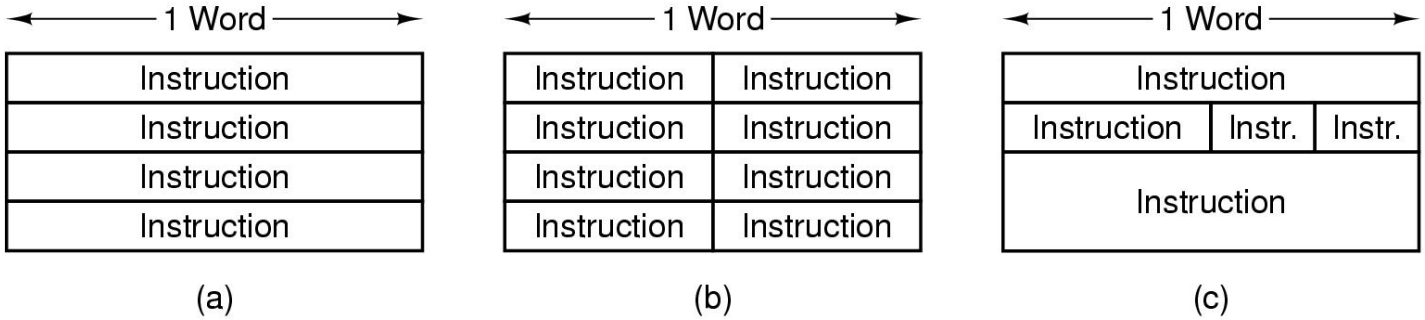
5.3 Формати на инструкции

- Една инструкција се состои од:
  - код на операцијата (opcode)
  - дополнителни информации (адреси)
    - од каде доаѓаат операндите
    - каде треба да се запишат резултатите
- Можни формати на инструкции:
  - Нула-адресни
  - Едно-адресни
  - Дво-адресни
  - Три-адресни



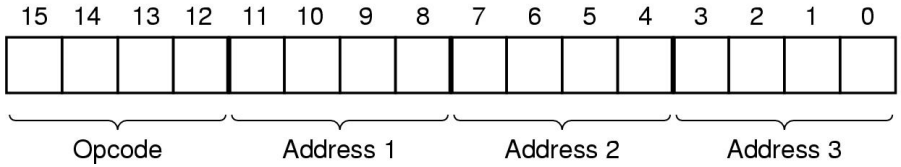
5.3 Формати на инструкции

- Должина на инструкциите
  - Инструкции со различна должина
  - Инструкции со иста должина – едноставно декодирање, но загуба на мемориски простор (сите инструкции треба да бидат долги колку најдолгата)



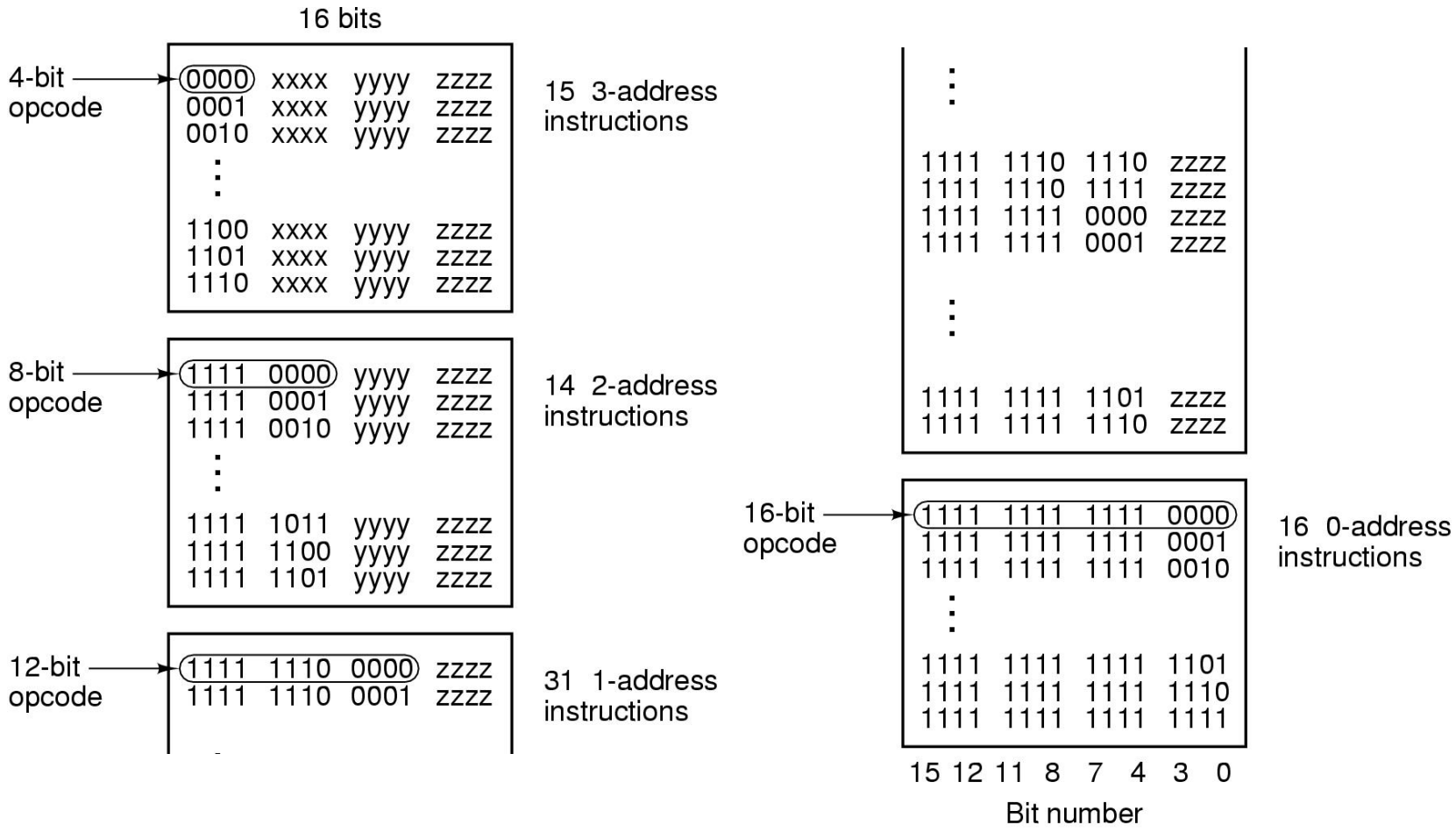
5.3.2 Кодови на операциите

- Пософистицирана шема – **кодови кои се прошируваат (expanding opcode)**
- ПРИМЕР 1:
  - Должина на инструкциите = 16 бита
  - Должина на адресите = 4 бита



  - Инструкции
    - 15 три-адресни
    - 14 дво-адресни
    - 31 едно-адресни
    - 16 нула-адресни

5.3.2 Кодови на операциите



5.3.2 Кодови на операциите

- ПРИМЕР 2:
  - Должина на инструкциите = 8 бита
  - Должина на адресите = 2 бита
  - Инструкции
    - 2 три-адресни
    - 6 дво-адресни
    - 7 едно-адресни
    - 4 нула-адресни

00 xx yy zz	**10 00** yy zz	**11 10 00** zz	**11 11 11 00**
01 xx yy zz	**10 01** yy zz	**11 10 01** zz	**11 11 11 01**
	10 10 yy zz	**11 10 10** zz	**11 11 11 10**
	10 11 yy zz	**11 10 11** zz	**11 11 11 11**
	11 00 yy zz	**11 11 00** zz	
	11 01 yy zz	**11 11 01** zz	
		11 11 10 zz	