Структурная и функциональная организация ЭВМ (Computer Organization and Design)

БГУИР
кафедра ЭВМ

Лекция 13
«Способы адресации»

2019

План лекции

1. Результаты промежуточного теста 03
2. Способы адресации операндов
3. Порядок байт в памяти при адресации

Количество адресов в команде

Время выполнения программы и адресность команд

Одноадресная команда быстрее, но для реализации трёхадресной команды нужно три одноадресных.

Определяющим при выборе является тип алгоритмов, на которые ориентирована ВМ:

- последовательные
- параллельные
- комбинированные

Количество адресов в команде

Время выполнения программы и адресность команд

В последовательных программах – результат предыдущей операции используется для последующей. Выгодна одноадресная команда.

В параллельных – результат пересылается в память. Трёхадресные команды будут эффективнее.

В комбинированных – лучше будут одноадресные и полутороадресные.

Способы адресации операндов

Один из центральных вопросов при проектировании ВМ

Исполнительный адрес операнда ($А_{ИСП}$) – двоичный код номера ячейки памяти, служащей источником/приёмником операнда (адрес на ША или номер регистра).

Адресный код команды ($А_К$) - двоичный код в адресном поле команды, из которого необходимо сформировать исполнительный адрес операнда.

В современных ВМ как правило $А_{ИСП}$ и $А_К$ не совпадают, требуется соответствующее преобразование.

Способ адресации – это способ формирования исполнительного адреса по адресному коду команды.

Способы адресации операндов

Один из центральных вопросов при проектировании ВМ

Исполнительный адрес операнда ($А_{ИСП}$) – двоичный код номера ячейки памяти, служащей источником/приёмником операнда (адрес на ША или номер регистра).

Адресный код команды ($А_К$) - двоичный код в адресном поле команды, из которого необходимо сформировать исполнительный адрес операнда.

В современных ВМ как правило $А_{ИСП}$ и $А_К$ не совпадают, требуется соответствующее преобразование.

Способ адресации – это способ формирования исполнительного адреса по адресному коду команды.

Способы адресации операндов

Способы адресации (по Цилькеру и Орлову):

- Непосредственная
- Прямая
- Косвенная
- Регистровая
- Косвенная регистровая
- Адресация со смещением
  - Относительная
  - Базовая регистровая
  - Индексная
- Страничная
- Блочная
- Стековая

Способы адресации операндов

Непосредственная (Immediate addressing) – в адресном поле вместо адреса сам операнд.

Код операции	СА	Непосредственный операнд

Операции сравнения, загрузки констант в регистры, арифм. операции.

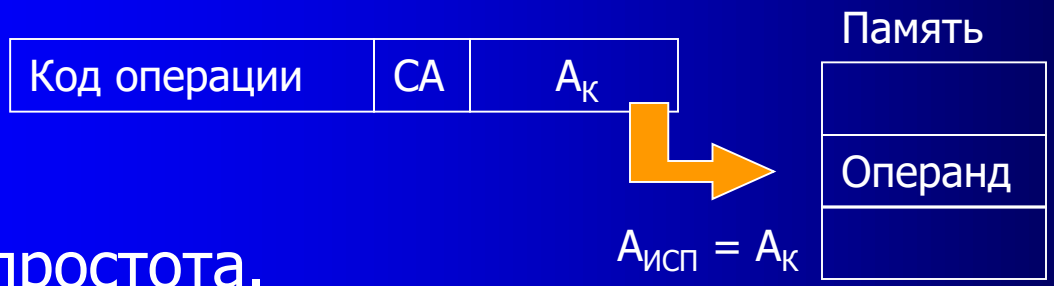
Операнд – число в (обычно) доп. коде.

Основная проблема – не каждый операнд можно передать непосредственно – адр. часть команды как правило меньше машинного слова. 50-60% - до 8 бит, 70-80% - до 16 бит.

Плюсы – малое время выполнения команды, экономия памяти.

Способы адресации операндов

Прямая (Direct addressing) – адресный код прямо указывает номер ячейки памяти операнда.



Плюсы – простота.

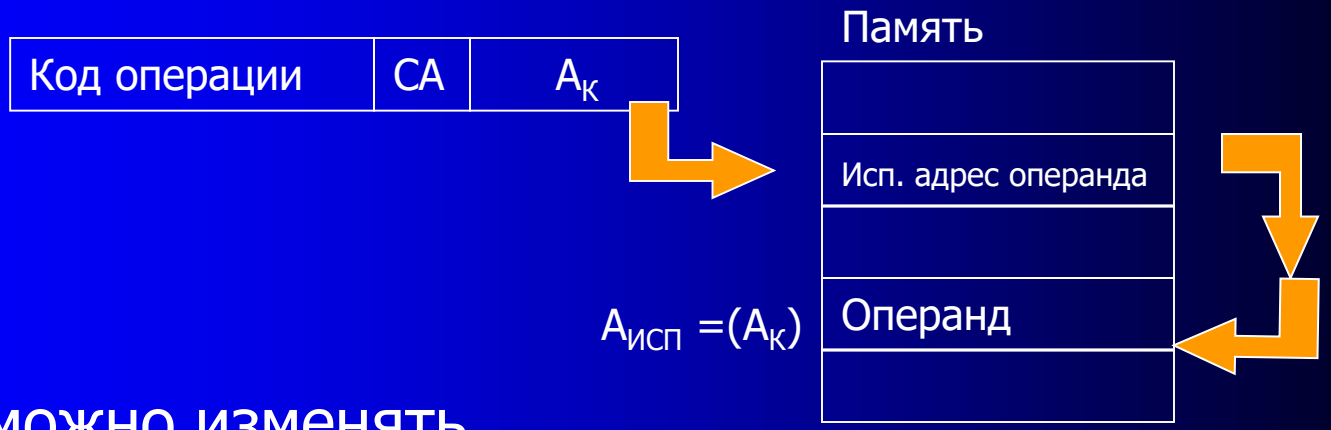
Существенный недостаток – ограниченный размер адресного пространства.

Главный недостаток?

Адрес в команде не может быть изменён в процессе вычислений -> ограничение на размещение программы в ОЗУ.

Способы адресации операндов

Косвенная (Indirect addressing) – ограниченное адресное поле указывает указывает адрес ячейки, содержащей полноразрядный адрес памяти операнда.



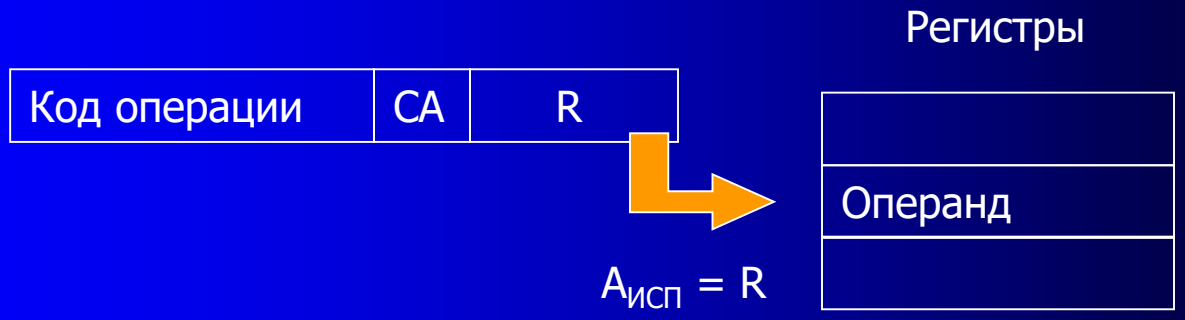
Плюсы – можно изменять адреса операндов в процессе вычислений.

Существенный недостаток – двухкратное обращение к памяти.

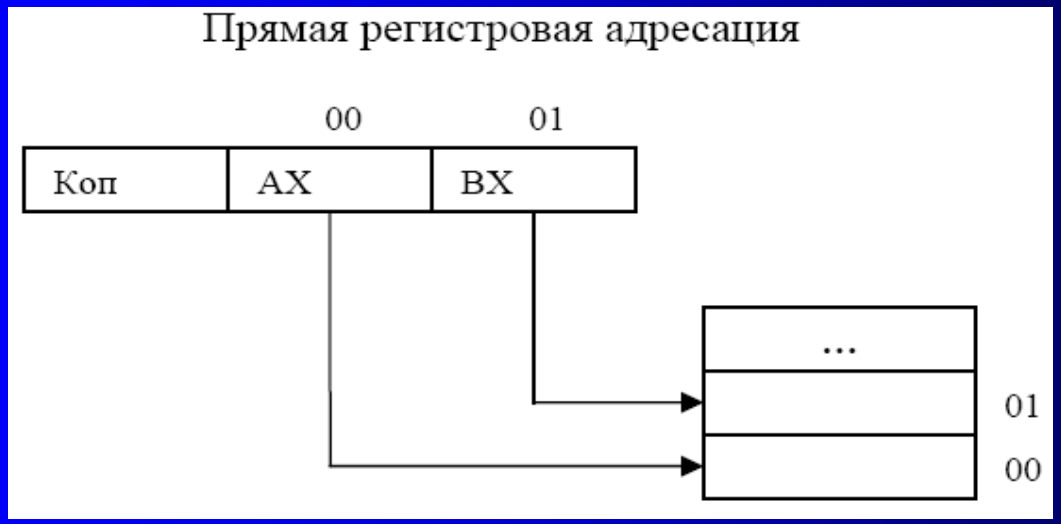
Многоуровневая или каскадная косвенная адресация – иногда удобна для обработки многомерных массивов.

Способы адресации операндов

Регистровая (Register) – адресное поле указывает на номер регистра, содержащего операнд.

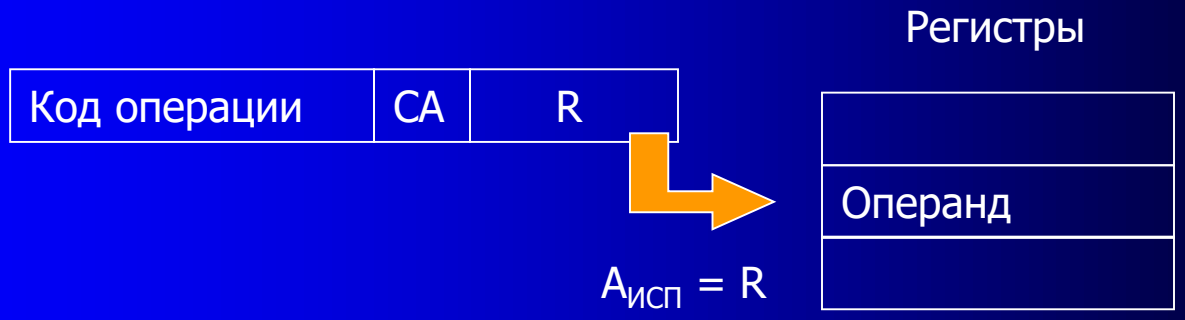


По Добулевичу: Прямая регистровая



Способы адресации операндов

Регистровая (Register) – адресное поле указывает на номер регистра, содержащего операнд.



Плюсы – размер адресного поля 3-4 бита для 8-16 РОН.
Исключение обращений к памяти.

Минусы – мало РОН.

Способы адресации операндов

Косвенная регистровая (Register indirect) – адресное поле указывает на номер регистра, содержащего операнд.

Плюсы аналогичны обычной косвенной адресации – можно изменять адреса операндов в процессе вычислений.

На одно обращение к памяти меньше.

Способы адресации операндов

Косвенная регистровая (Register indirect) – адресное поле указывает на номер регистра, содержащего операнд.

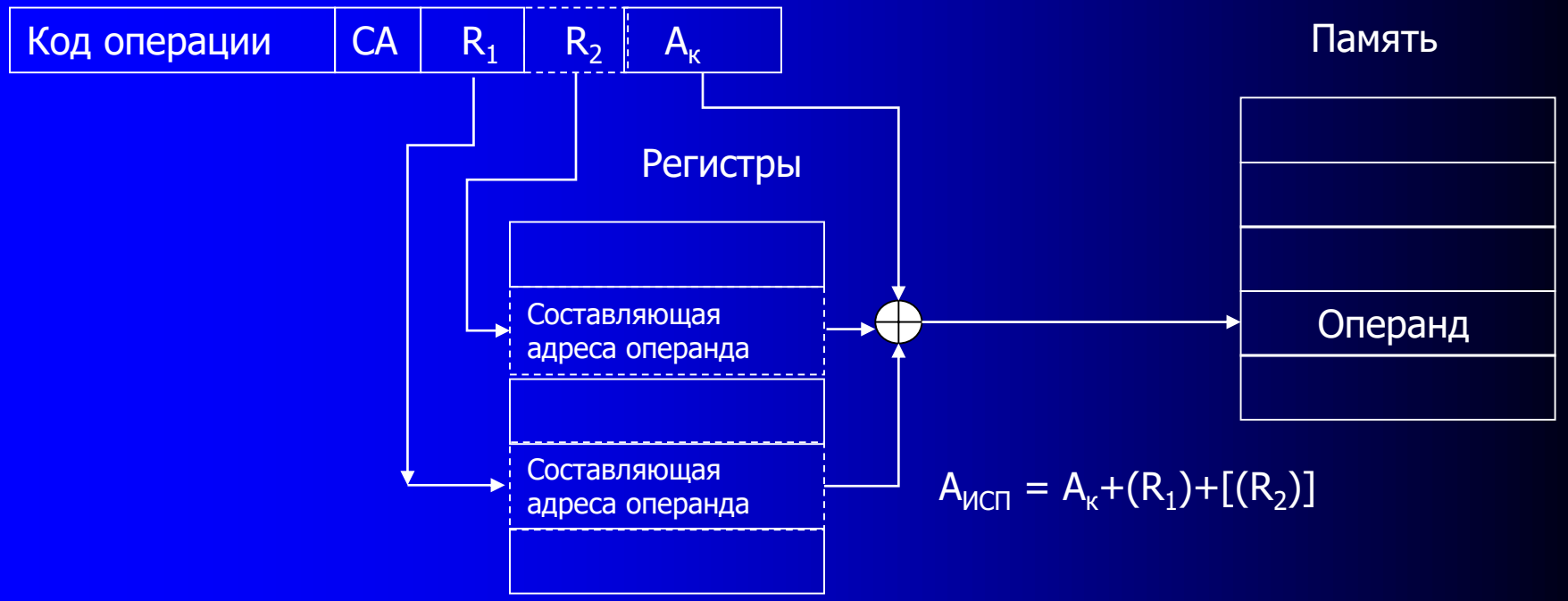
Вот фрагмент, суммирующий в регистре ax элементы массива слов:

```
        mov     si,  "начальный адрес массива"
        mov     cx,  "количество элементов массива"
ckl     add     ax, [si];  прибавляем следующий элемент
        add     si, 2;  модифицируем адрес в массиве
        loop    ckl;  организуем цикл
```

Выделенные команды эквивлентны Си-фрагменту sum += *(piMas++); .

Способы адресации операндов

Адресация со смещением (Displacement) – исп. адрес формируется суммированием содержимого адресного поля команды с содержимым одного или нескольких регистров ЦП.



Способы адресации операндов

Адресная часть включает как минимум одно поле $A_к$ и содержит константу. Смысл константы может меняться – базовый адрес, либо смещение.

Регистры могут быть как РОН, так и специализированные – базовый и (или) индексный.

Спец регистры могут указываться по умолчанию, тогда только одно поле $A_к$.

Если РОН – то дополнительно R_1 (R_2, R_3…).

Дополнительный R (обычно РОН) может появляться для указания масштабного коэффициента.

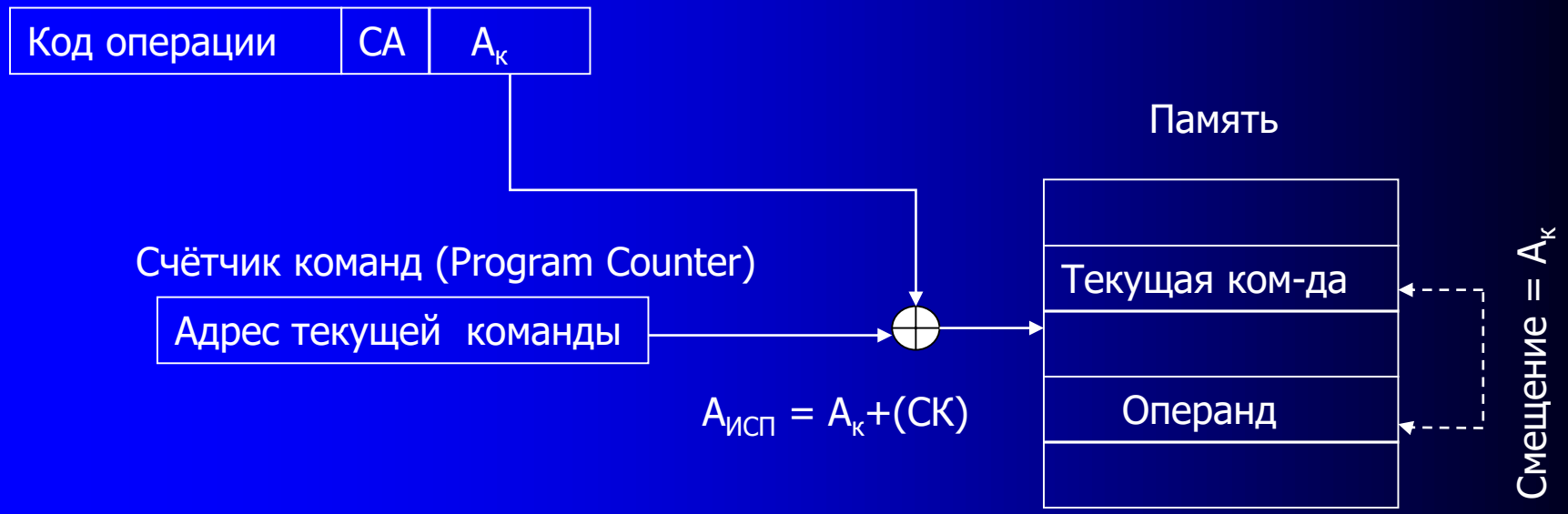
Наиболее общий вариант – два поля $A_к$ и R.

Способы адресации операндов

Один из вариантов – вместо суммирования – конкатенация.

Способы адресации операндов

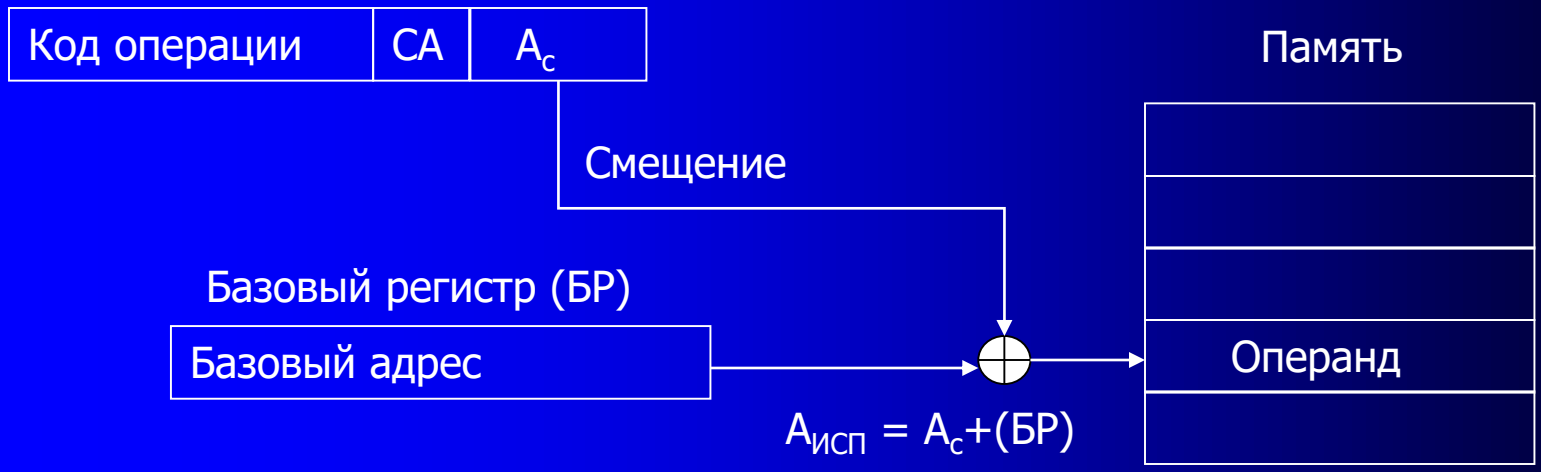
Адресная со смещением: относительная – для получения исполнительного адреса поле $A_к$ складывается с содержимым счётчика команд.



Базируется на свойстве локальности – обращение к ячейкам в непосредственной близости от текущей.
Плюсы – можно экономить на разрядности $A_к$, программа перемещаема в памяти.

Способы адресации операндов

Адресная со смещением: базовая регистровая адресация – регистр, называемый базовым, содержит полноразрядный адрес, поле A_c – смещение.

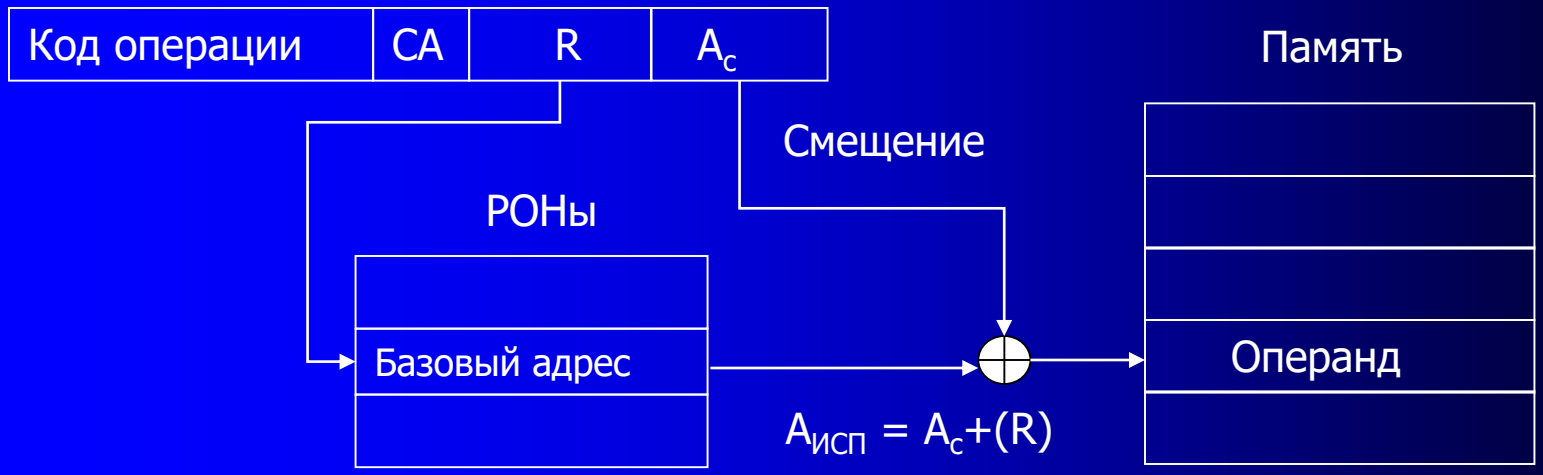


Базовый регистр – может быть задан не явно (если специальный регистр).

Способы адресации операндов

Адресная со смещением: базовая регистровая адресация

Базовый регистр – может быть задан явно (если РОН).



Способ используется для доступа к элементам массива.

Смещение – меньшая длина, чем полный адрес -> команда короткая.

Способы адресации операндов

Адресная со смещением: базовая регистровая адресация

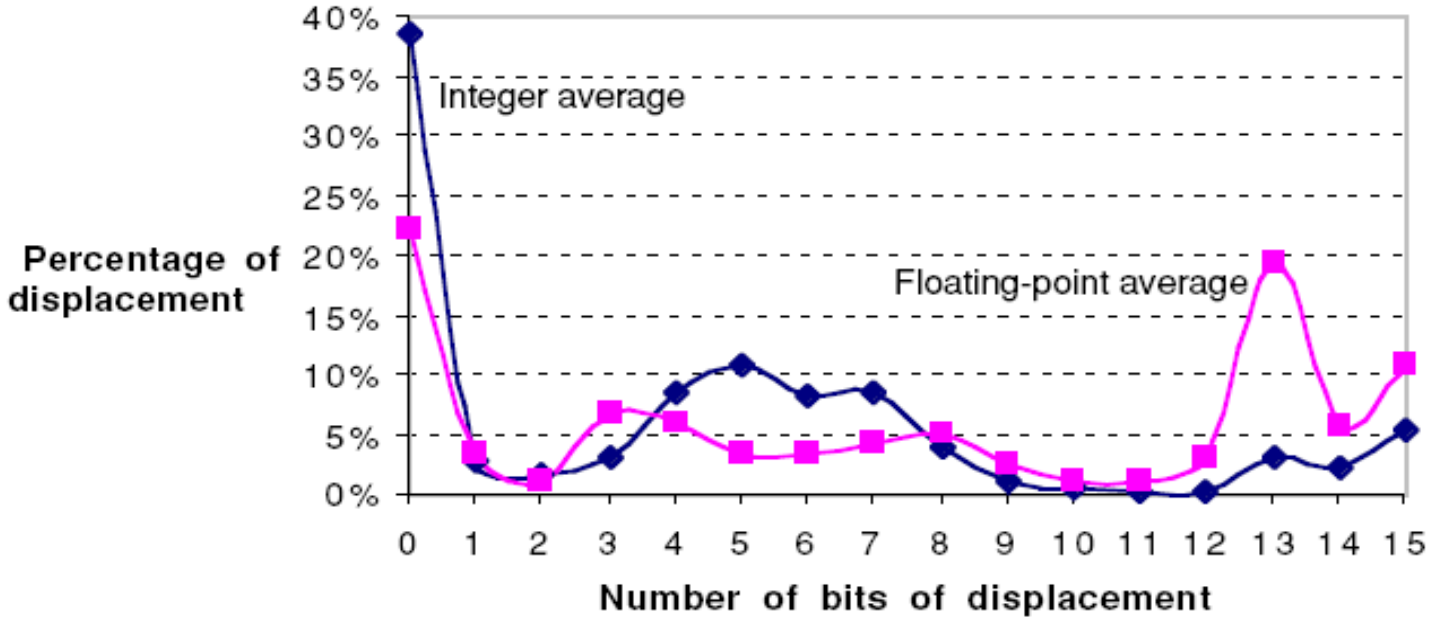
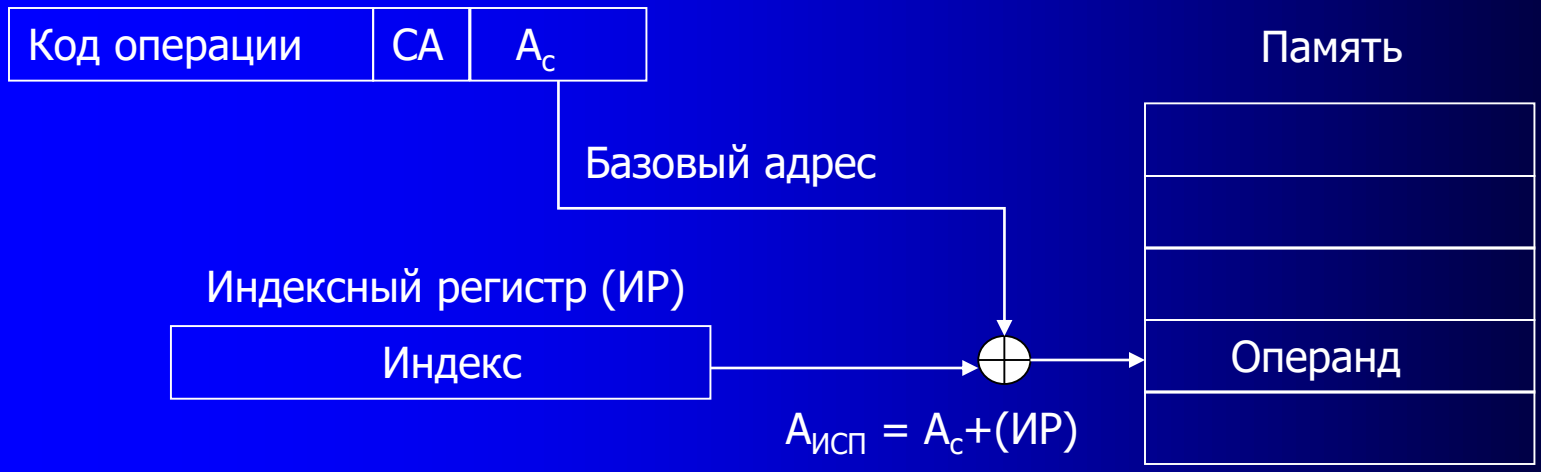


FIGURE 2.8 Displacement values are widely distributed. There are both a large number of small values and a fair number of large values. The wide distribution of displacement values is due to multiple storage areas for variables and different displacements to access them (see section 2.11) as well as the overall addressing scheme the compiler uses. The x axis is $\log_2$ of the displacement; that is, the size of a field needed to represent the magnitude of the displacement. Zero on the x axis shows the percentage of displacements of value 0. The graph does not include the sign bit, which is heavily affected by the storage layout. Most displacements are positive, but a majority of the largest displacements (14+ bits) is negative. Since this data was collected on a computer with 16-bit displacements, it cannot tell us about longer displacements. These data were taken on the Alpha architecture with full optimization (see section 2.11) for SPEC CPU2000, showing the average of integer programs (CINT2000) and the average of floating-point programs (CFP2000).

Способы адресации операндов

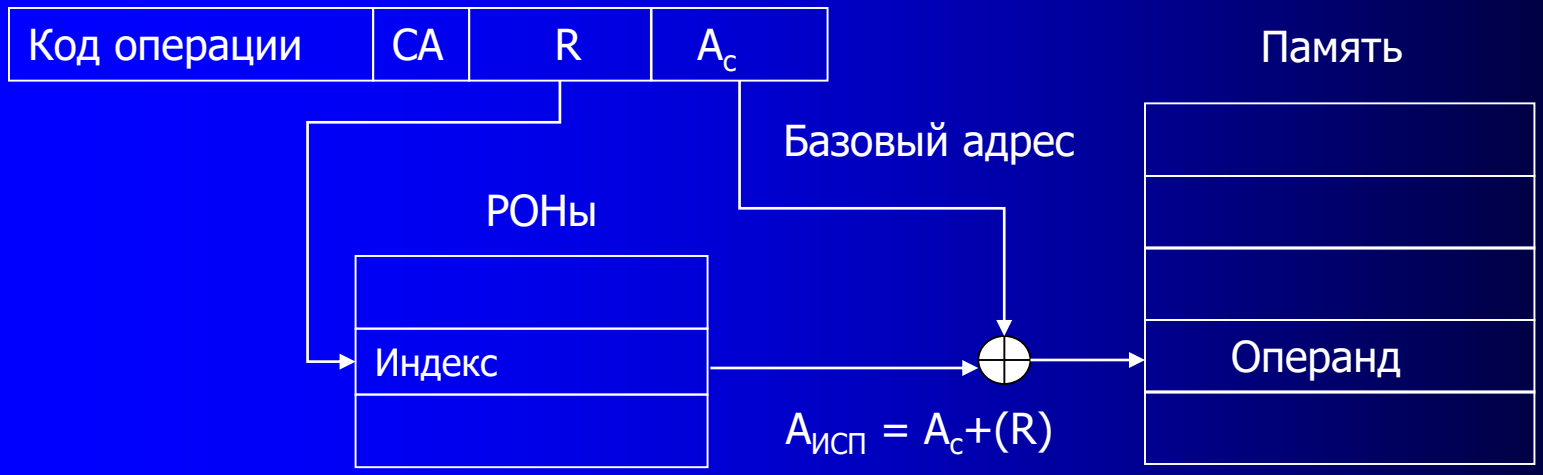
Адресная со смещением: индексная адресация – поле A_c содержит адрес памяти, регистр (явный/неявный) – смещение.



Разрядность A_c – больше чем в предыдущем варианте. Вычисления аналогичны.

Способы адресации операндов

Адресная со смещением: индексная адресация – поле A_c содержит адрес памяти, регистр (явный) – смещение.



Удобный механизм для организации итеративных вычислений.

Способы адресации операндов

Адресная со смещением: индексная адресация

Пример: Дан массив чисел (последовательно в памяти), начиная с адреса N. Необходимо увеличить на 1 все элементы массива.

Выборка всех элементов, инкрементация и возврат обратно в память. Адреса – N, N+1, N+2…

Значение N – из Ac, а индексный регистр – 0, +1,+2 …

Типичный случай – индексный регистр изменяется автоматически как часть машинного цикла (автоиндексирование).

Почему нельзя сделать то же самое с помощью базовой регистровой?

Способы адресации операндов

Адресная со смещением: индексная адресация

Если индексный регистр – то неявное автоиндексирование называется автоинкрементная адресация (autoincrement).

$A_{ИСП} = A_c+(R)$, R <- (R)+1

Постинкрементное автоиндексирование

R <- (R) +1 , $A_{ИСП} = A_c+(R)$,

Преинкрементное автоиндексирование

Автодекрементная адресация (autodecrement).

$A_{ИСП} = A_c+(R)$, R <- (R) -1

Постдекрементное автоиндексирование

R <- (R) -1 , $A_{ИСП} = A_c+(R)$,

Предекрементное автоиндексирование

Индексная адресация с масштабированием и смещением – содержимое инд. регистра умножается на масш. коэф. и суммируется с A_c. Коэф. = 1,2,4,8 (Intel).

Способы адресации операндов

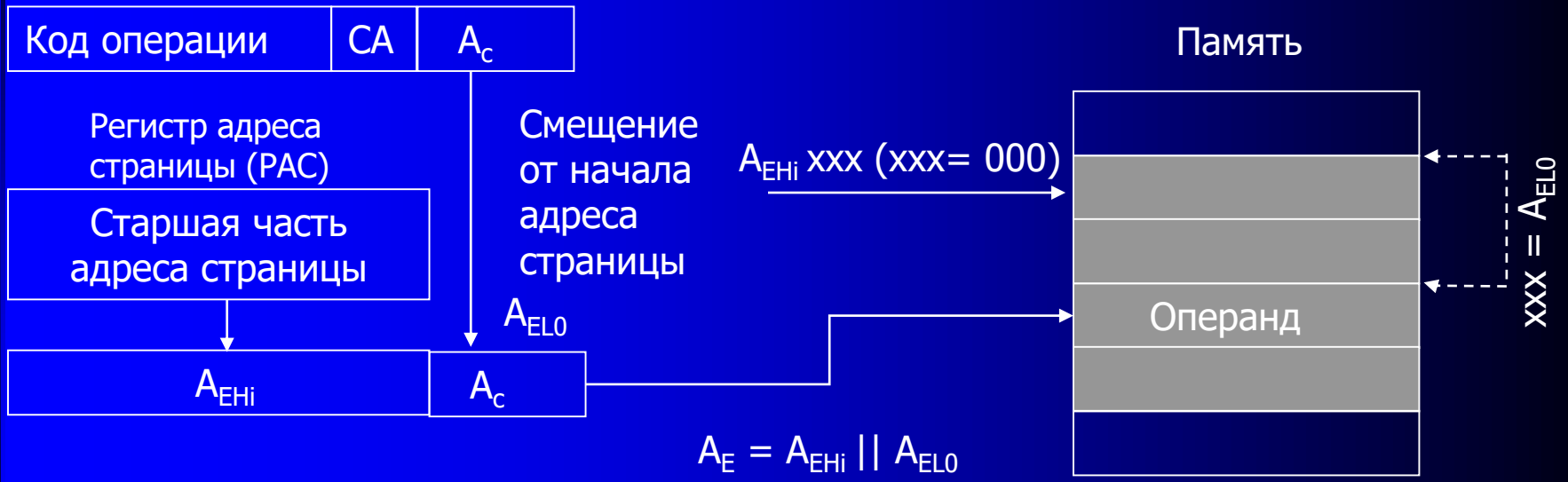
Страничная адресация

Разбиение адресного пространства на страницы.

Начальный адрес страницы – база.

Старшая часть адреса страницы – в спец. регистре (РАС)

Адресный код – смещение внутри страницы.



Способы адресации операндов

Блочная адресация

Используется в командах, обрабатывающих блок данных, расположенный в последовательных ячейках памяти.

Блок обычно описывается – адрес первой или последней ячейки памяти + количество элементов блока (в байтах либо ячейках).

Может быть так же использован специальный признак «конец блока», идущий за последним элементом блока.

Удобен для работы с внешними ЗУ, операциях с векторами.

Способы адресации операндов

Стековая адресация

Используется в ВМ со стековой архитектурой.

Стековый доступ к памяти происходит в большинстве процессоров в следующих ситуациях:

1) При выполнении команд стекового доступа: "поместить в стек" **push** или "извлечь из стека" **pop**.

2) При выполнении команды вызова подпрограммы и при возврате из нее.

3) При входе в прерывание и возврате из него.

Способы адресации операндов

Положение стека в адресуемой памяти определяется содержимым указателя стека.

Указатель стека программно доступен, программист может задать его значение обычной командой пересылки: **mov sp,#1000h -** задает положение стека, начиная с адреса 1000h.

Содержимое указателя стека используется как адрес операнда-приемника при записи или как адрес операнда- источника при считывании из стека.

При обращении к стеку автоматически модифицируется и содержимое указателя стека, чтобы обеспечить следующее обращение к очередной ячейке стека.

Обращение к стеку - косвенно-регистровая адресация через *указатель стека* с автоиндексацией.

Способы адресации операндов

Стек может быть организован по разному:

1. Направление роста стека. Если при записи в стек, содержимое указателя стека автоматически увеличивается (и, соответственно, при считывании автоматически уменьшается), то говорят, что стек растет в сторону увеличения адресов. В противоположном случае говорят, что стек растет в сторону уменьшения адресов.

2. Если модификация указателя стека выполняется до записи и соответственно после считывания, то указатель стека всегда указывает на последнюю занятую ячейку стека. Наоборот, если модификация производится после записи и до считывания, указатель стека всегда указывает на первую свободную ячейку стека.

Способы адресации операндов

Стек может быть организован по разному:

Запись в стек **push data**	Чтение из стека **pop data**	Стек растет в сторону	Указтель стека показывает на ячейку:
mov [sp]+,data	**mov data, -[sp]**	увеличения адресов	первую свободную
mov +[sp],data	**mov data, [sp]-**	увеличения адресов	последнюю занятую
mov [sp]-,data	**mov data, +[sp]**	уменьшения адресов	первую свободную
mov -[sp],data	**mov data, [sp]+**	уменьшения адресов	последнюю занятую

Способы адресации операндов

Addressing mode	Example instruction	Meaning	When used
Register	`Add R4,R3`	`Regs[R4]←Regs[R4] + Regs[R3]`	When a value is in a register.
Immediate	`Add R4,#3`	`Regs[R4]←Regs[R4]+3`	For constants.
Displacement	`Add R4,100(R1)`	`Regs[R4]←Regs[R4] + Mem[100+Regs[R1]]`	Accessing local variables (+ simulates register indirect, direct addressing modes)
Register indirect	`Add R4,(R1)`	`Regs[R4]←Regs[R4] + Mem[Regs[R1]]`	Accessing using a pointer or a computed address.
Indexed	`Add R3,(R1 + R2)`	`Regs[R3]←Regs[R3] +Mem[Regs[R1]+Regs[R2]]`	Sometimes useful in array addressing: R1 = base of array; R2 = index amount.
Direct or absolute	`Add R1,(1001)`	`Regs[R1]←Regs[R1] + Mem[1001]`	Sometimes useful for accessing static data; address constant may need to be large.
Memory indirect	`Add R1,@(R3)`	`Regs[R1]←Regs[R1] + Mem[Mem[Regs[R3]]]`	If R3 is the address of a pointer p, then mode yields $*p$.
Autoincrement	`Add R1,(R2)+`	`Regs[R1]←Regs[R1] + Mem[Regs[R2]]` `Regs[R2]←Regs[R2]+d`	Useful for stepping through arrays within a loop. R2 points to start of array; each reference increments R2 by size of an element, d.
Autodecrement	`Add R1,-(R2)`	`Regs[R2]←Regs[R2]-d` `Regs[R1]←Regs[R1] + Mem[Regs[R2]]`	Same use as autoincrement. Autodecrement/increment can also act as push/pop to implement a stack.
Scaled	`Add R1,100(R2)[R3]`	`Regs[R1]← Regs[R1]+ Mem[100+Regs[R2] + Regs[R3]*d]`	Used to index arrays. May be applied to any indexed addressing mode in some computers.

Способы адресации операндов

Addressing mode	Example instruction	Meaning	When used
Register **Регистровая**	Add R4,R3	Regs[R4]←Regs[R4] + Regs[R3]	When a value is in a register.
Immediate	Add R4,#3	Regs[R4]←Regs[R4]+3	For constants. **Непосредственая**
Displacement **Со смещением**	Add R4,100(R1)	Regs[R4]←Regs[R4] + Mem[100+Regs[R1]]	Accessing local variables (+ simulates register indirect, direct addressing modes)
Register indirect **Регистровая косвенная**	Add R4,(R1)	Regs[R4]←Regs[R4] + Mem[Regs[R1]]	Accessing using a pointer or a computed address.
Indexed **Индексная**	Add R3,(R1 + R2)	Regs[R3]←Regs[R3] +Mem[Regs[R1]+Regs[R2]]	Sometimes useful in array addressing: R1 = base of array; R2 = index amount.
Direct or absolute **Прямая**	Add R1,(1001)	Regs[R1]←Regs[R1] + Mem[1001]	Sometimes useful for accessing static data; address constant may need to be large.
Memory indirect **Ковсенная**	Add R1,@(R3)	Regs[R1]←Regs[R1] + Mem[Mem[Regs[R3]]]	If R3 is the address of a pointer p, then mode yields $*p$.
Autoincrement **Автоинкрементная**	Add R1,(R2)+	Regs[R1]←Regs[R1] + Mem[Regs[R2]] Regs[R2]←Regs[R2]+d	Useful for stepping through arrays within a loop. R2 points to start of array; each reference increments R2 by size of an element, d.
Autodecrement **Автодекрементная**	Add R1,-(R2)	Regs[R2]←Regs[R2]-d Regs[R1]←Regs[R1] + Mem[Regs[R2]]	Same use as autoincrement. Autodecrement/increment can also act as push/pop to implement a stack.
Scaled **Индексная со смещением и масштабированием**	Add R1,100(R2)[R3]	Regs[R1]← Regs[R1]+ Mem[100+Regs[R2] + Regs[R3]*d]	Used to index arrays. May be applied to any indexed addressing mode in some computers.

Способы адресации операндов

Метод адресации	Пример команды	Смысл команды метод использования
Регистровая	Add R4,R3	R4<-R4+R3 Требуемое значение в регистре
Непосредственная или литеральная	Add R4,#3	R4 <- R4+3 Для задания констант
Базовая со смещением	Add R4,100(R1)	R4 <- R4+M[100+R1] Для обращения к локальным переменным
Косвенная регистровая	Add R4,(R1)	R4 <- R4+M[R1] Для обращения по указателю или вычисленному адр.
Индексная	Add R3,(R1+R2)	R3 <- R3+M[R1+R2] Иногда полезна при работе с массивами: R1 - база, R3 - индекс
Прямая или абсолютная	Add R1,(1000)	R1 <- R1+M[1000] Иногда полезна для обращения к стат. данным
Косвенная	Add R1,@(R3)	R1 <- R1+M[M[R3]] Если R3-адрес указателя p, то выбирается значение по этому указателю
Автоинкрементная	Add R1,(R2)+	R1 <- R1+M[R2] R2 <- R2+d Полезна для прохода в цикле по массиву с шагом: R2 - начало массива В каждом цикле R2 получает приращение d
Автодекрементная	Add R1,-(R2)	R2 <- R2-d R1 <- R1+M[R2] Аналогична предыдущей Обе могут использоваться для реализации стека
Индексная со смещением и масштабированием	Add R1, 100(R2)[R3]	R1<-R1+M[100]+R2+R3*d Для индексации массивов

Способы адресации операндов

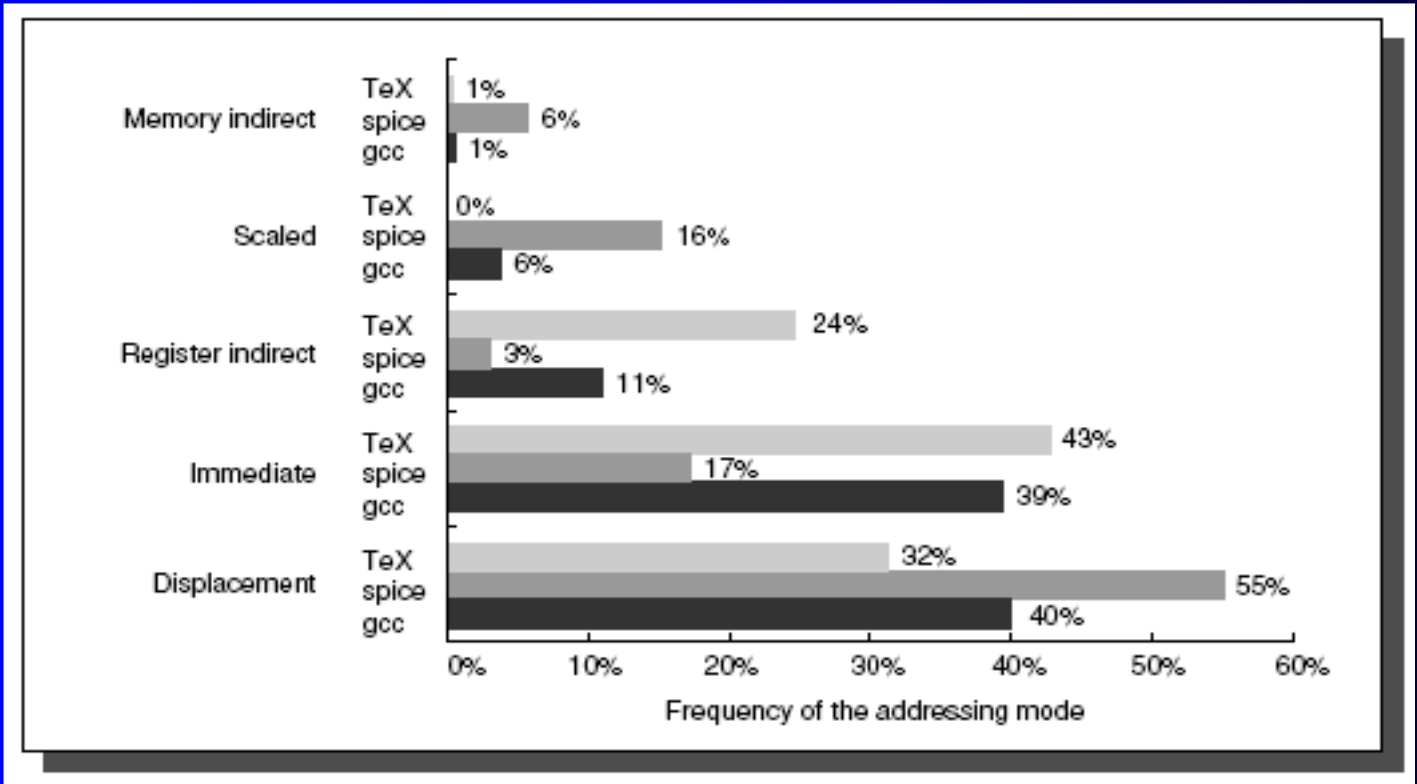
Autoincrement	Add R1, (R2)+	Regs [R1] ←Regs [R1] + Mem [Regs [R2]] Regs [R2] ←Regs [R2] +*d*	Useful for stepping through arrays within a loop. R2 points to start of array; each reference increments R2 by size of an element, *d*.
Autodecrement	Add R1, –(R2)	Regs [R2] ←Regs [R2] –*d* Regs [R1] ←Regs [R1] + Mem [Regs [R2]]	Same use as autoincrement. Autodecrement/increment can also act as push/pop to implement a stack.
Scaled	Add R1,100(R2) [R3]	Regs [R1] ← Regs [R1] + Mem [100+Regs [R2] + Regs [R3] *d]	Used to index arrays. May be applied to any indexed addressing mode in some computers.

d- варьируется от размера данных (1,2,4,8 байт), useful только при последовательном размещении в памяти.

RISC процессоры используют адресацию со смещением для эмуляции регистровой косвенной (0 в поле адреса) и эмулируют прямую адресацию (с 0 в базовом регистре).

Способы адресации операндов

Данные для DEC VAX:



RISC архитектура - преимущественный способ адресации: регистровая. Для аккумуляторной архитектуры главные способы адресации - прямая и непосредственная.

Способы адресации операндов

Данные для Intel процессоров:

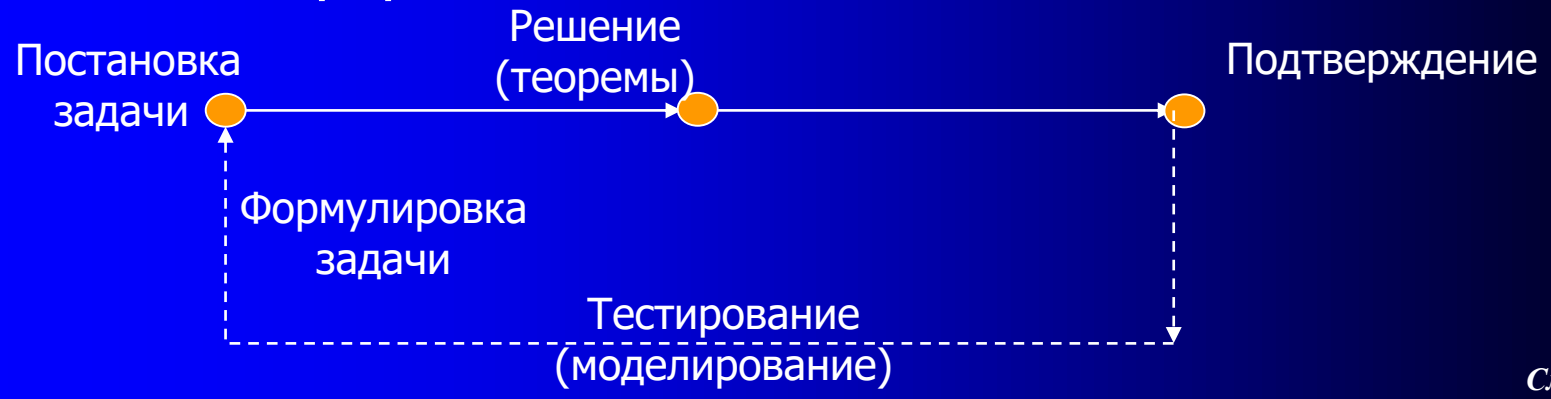
Тип адресации	GCC	Spice
Косвенно- регистровая [bx]	10%	3%
Индексная с масштабированием [2*ebx+5]	20%	33%
Прямая [123]	19%	51%
Базово-регистровая [bp+5]	48%	14%

Наиболее активно используется прямая и базово-регистровая адресации, хотя интенсивность использования различных способов тесно связана с решаемой задачей.

Способы адресации операндов

Система операций – список операций, выполняемый непосредственно техн. средствами ВМ.

Выбор системы операций:
- на основе сущ. Аналогов
- на основе статистики использования отдельных команд (с использованием тестов).
- с ориентацией на языки ВУ (Wang, HP, IBM, Tektronix).
- на основе формализации и систематизации:



Способы адресации операндов

Выбор системы операций:
- на базе метода группировки (все операции по группам)
- на основе восходящей совместимости (включение команд ВМ старого поколения).
- с учётом требований пользователя.
- с помощью метода условной интерпретации – учёт соотношения «стоимость/производительность» интерпретации отдельных команд либо их реализации аппаратно.

Big-endian vs Little endian

- Little endian - младший байт по младшему адресу (e.g., Base Address+0 Byte0, Base Address+1 Byte1, Base Address+2 Byte2, Base Address+3 Byte3)

Intel, DEC

- Big endian – старший байт по младшему адресу (E.g., Base Address+0 Byte3, Base Address+1 Byte2, Base Address+2 Byte1, Base Address+3 Byte0)

Motorola, IBM

- Что лучше?
  - Printing, Binary-to-decimal conversion, sign-testing, multiple precision math routines

- Различные форматы файлов используют различный порядок:
  - Big Endian – Photoshop, TCP/IP sockets, JPEG, etc.
  - Little Endian – BMP, GIF, RTF, etc.