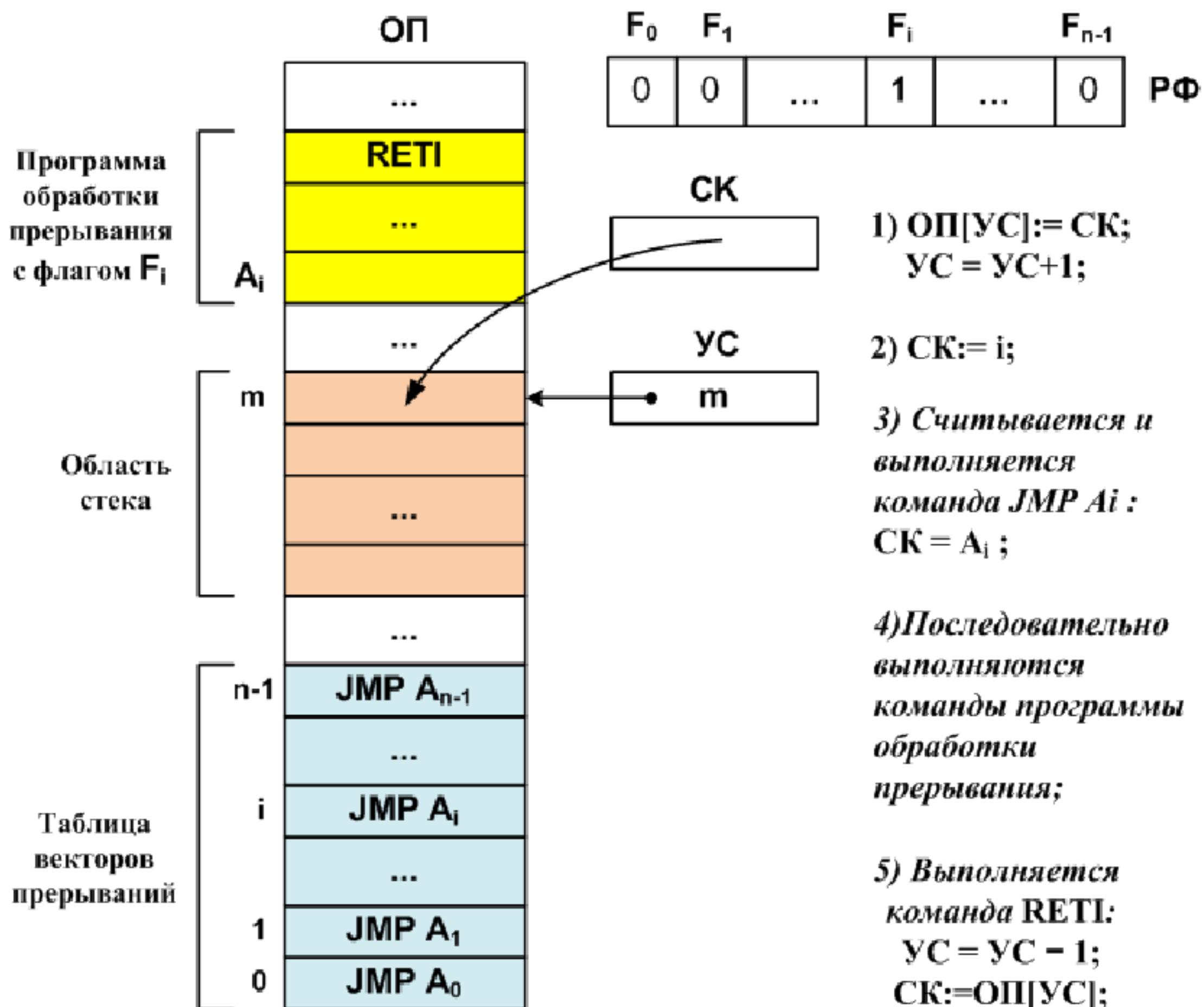


# Переключения с программы на программу. Обработка прерываний

*Для поддержки системы прерываний в состав процессора введены регистр флагов (РФ) и регистр-указатель стека (УС).*



Во время выполнения программы может возникнуть прерывание, вызванное, например, исключительной ситуацией, произошедшей в процессе выполнения очередной команды (деление на ноль, переполнение с фиксированной точкой, исчезновение порядка, переполнение порядка, потеря значимости и.т.п.). Подобные прерывания называются исключениями и, как правило, не могут быть замаскированы (маскирование отменяет обработку прерывания).

Другой тип прерываний – прерывания, поступившие от внешних устройств (поступил запрос на ввод или вывод данных от внешнего устройства, пользователь нажал кнопку <Num Lock> на клавиатуре). Все наблюдали последнюю ситуацию: во время работы программы, например, текстового процессора WORD, при нажатии клавиши <Num Lock> загорается лампочка, т.е. прерывание от клавиатуры обрабатывается. Это значит, что во время работы программы Word осуществляется переход на программу обработки прерывания, которая и зажигает лампочку (или тушит , если клавиша <Num Lock> была нажата повторно). После выполнения программы обработки прерывания процессор возвращается к выполнению прерванной программы (Word).

Прерывания от внешних устройств могут быть замаскированы, т.е. их обработка может быть отменена.

Современные компьютеры работают в мультипрограммном режиме. Каждой из программ выделяется квант времени  $\tau$  (например,  $\tau=1/30$  с). По истечении этого такта возникает прерывание от таймера, и процессор переключается на выполнение другой программы. Поскольку выделяемый квант мал, у пользователя создается иллюзия непрерывной работы его программы. Возврат осуществляется именно в то место программы, в котором она была прервана.

### Аппаратная поддержка системы прерываний

Для поддержки системы прерываний процессор имеет два регистра специальных функций: регистр флагов (РФ) прерываний и регистр масок (РМ) прерываний. Число битов в этих регистрах соответствует числу источников (причин) прерываний,  $i$ -ый бит соответствует  $i$ -ой причине прерывания. Если в РМ процессора установить в единицу  $i$ -ый бит (это можно сделать специальной командой, которая есть в системе команд процессора), то соответствующее прерывание обрабатываться не будет. Как уже говорилось выше, не все прерывания могут быть замаскированы.

В ОП (если гарвардская архитектура, то в памяти программ) первые  $n$  слов отводятся под таблицу векторов прерываний (ТВВ). Программист не должен их использовать при низкоуровневом программировании (на ассемблере или в машинных кодах). Трансляторы с языков высокого уровня их тоже не используют.

Каждому биту  $F_i$  в регистре флагов РФ, как уже говорилось, соответствует определенная причина прерывания и слово в ТВВ.

В этом ( $i$ -ом) слове ТВВ записана (в двоичном виде) команда безусловного перехода на адрес  $A_i$  первой команды программы обработки соответствующего прерывания, которая также хранится в ОП.

Для осуществления возможности возврата в определенное место прерванной программы (то, в котором она была прервана) в ОП выделяется определенная область под стек, а в структуру процессора вводится еще один специальный регистр – **указатель стека (УС)**.

В УС всегда находится адрес слова, которое является «верхушкой» стека (верхнего свободного элемента стека).

**Стек – структура данных, реализующая принцип LIFO (last in, first out) – последний вошел, первый вышел.** Если информация записывается в стек, то она записывается в «верхушку стека», т.е. в слово, адрес которого записан в УС. После этого адрес в УС изменяется (в нашей иллюстрации увеличивается) так, чтобы он вновь указывал на свободное слово (верхушку стека). При извлечении информации из стека сначала изменяется значение в УС (уменьшается в нашей иллюстрации) для того, чтобы из стека было извлечено значение, записанное последним.

После выполнения очередной команды процессор анализирует РФ. При обнаружении единицы в  $i$ -ом бите РФ ( $F_i = 1$ ), производятся следующие действия:

1) В слово ОП, на которое указывает УС, копируется содержимое СК (счетчик команд указывает на следующую, еще не выполненную команду). После этого УС увеличивается на 1, чтобы указывать на свободное слово.

$$\begin{aligned} \text{ОП} [\text{УС}] &= \text{СК} ; \\ \text{УС} &= \text{УС} + 1; \end{aligned}$$

2) В СК загружается адрес слова ТВВ, соответствующего биту  $F_i$  (причине прерывания).

$$\text{СК} = i ;$$

3) Процессор выполняет команду, на которую указывает СК. В  $i$ -ом слове ОП находится команда безусловного перехода на адрес  $A_i$  первой команды программы обработки соответствующего прерывания.

$$\text{JMP } A_i ;$$

Результатом выполнения этой команды процессором является появление в СК адреса  $A_i$ :  $\text{СК} = A_i$

4) Процессор последовательно выполняет команды программы обработки прерывания.

5) Последняя команда в программе обработки прерываний – команда возврата из прерывания RETI (Return Interrupt).

Она выполняется следующим образом: из стека извлекается запомненное значение адреса возврата и загружается в СК, работа процессора продолжается (дополняется прерванная программа).

$$\begin{aligned} \text{УС} &= \text{УС} - 1; \\ \text{СК} &= \text{ОП} [\text{УС}]; \end{aligned}$$

Если после выполнения очередной команды в регистре флагов обнаружено несколько единиц, то выполняется та или иная процедура опроса прерываний с учетом или без учета их приоритетов , и все возникшие незамаскированные прерывания так или иначе обрабатываются (этот материал рассматривается в курсе «Операционные системы»).

Недостаток систем, основанных на векторах прерываний состоит в том, что запоминается только адрес возврата. Если для нормального дovskyполнения прерванной программы необходимо запомнить содержимое некоторых регистров специальных функций (РСФ) и регистров общего назначения (РОН) (они могут быть изменены программой обработки прерывания), то эта задача возлагается на программиста, который пишет программу обработки прерывания (при входе в программу обработки прерывания он сохраняет значения нужных регистров (например, регистра признака результата РПР) в ОП или в РОН, а перед командой RETI восстанавливает содержимое этих регистров, если оно было запорчено программой обработки прерываний).

Существуют системы , которые при возникновении прерывания сохраняют, а после обработки прерывания восстанавливают не только адрес возврата, но и значения некоторых регистров специальных функций, например, РПР, регистра масок прерываний и других специальных регистров, объединенных общим названием «Слово состояния программы» (CCP или PSW).

# **Методы адресации операндов в машинных командах**

**(способы сокращения длины  
команд)**

Команда – это приказ компьютеру на выполнение какой-либо операции, 10 например, операции сложения двух чисел (операндов), которые хранятся в оперативной памяти.

В рассмотренном выше примере использовалась прямая адресация обоих операндов. Такой вариант был возможен, т.к. число битов шины адреса было невелико (объем ОП – 256 байт, число битов шины адреса – 8). С увеличением объема ОП длина команды при использовании прямой адресации операндов может сильно увеличиться, из-за этого возрастет и размер программы в целом, а значит, для ее хранения потребуется значительная область памяти..

Адрес первого операнда в ОП	111...011	010...001	001...011
Код операции			Адрес второго операнда в ОП

Для сокращения длины команды адрес области памяти, куда нужно поместить результат не указывается. Результат, как правило, помещается в память на место первого операнда.

# Методы адресации операндов в машинной команде

**Цель применения различных способов адресации  
операндов – уменьшение длины команды.**

## Задачи:

- 1) ознакомиться с типами машинных команд;
- 2) научиться применять различные способы  
адресации операндов при разработке форматов  
машинных команд.

Как уже говорилось, в рассмотренном выше примере выполнения арифметической команды использовалась прямая адресация operandов. Размер ОП был равен 256 байтам, соответственно прямой адрес каждого операнда в команде занимал

$$\log_2 256 = 8 \text{ (бит)} = 1 \text{ байт.}$$

**Если ОП имеет достаточно большой объем, указание прямых адресов operandов в программе может вызвать значительное увеличение длины команды, а, следовательно, и программы в целом.**

Для того, чтобы этого избежать, используются другие способы адресации operandов.

В процессор добавляется блок регистров общего назначения (РОН), состоящий, например, из 8, 16 или 32 четырехбайтных регистров, которые используются для хранения operandов (считанных из ОП), а, иногда, их адресов.

**Пусть**

$$E_{оп} = 256M(\text{байт}) = 2^8 * 2^{20}(\text{байт}) = 2^{28}(\text{байт}).$$

Тогда при указании в команде прямых адресов операндов, длина команды будет равна  $8(\text{КОП}) + 28(\text{A1}) + 28(\text{A2}) = 64$  бита = 8 байт.

Команд с такой длиной вы не найдете. Для уменьшения длины команды используются различные способы адресации.

Покажем их на примерах. Напомним, что в структуру процессора добавлена **регистровая память**. Пусть она состоит из шестнадцати **32-разрядных РОН**.

**Пример 1.** Команда арифметическая (например, сложение). Первый операнд находится в РОН, второй – в ОП. Способ адресации первого операнда – регистровый, второго – относительный. Результат помещается на место первого операнда.

Формат команды:

КОП	R <sub>1</sub>	B <sub>2</sub>	D <sub>2</sub>
0	7 8	11 12	15 16 31

Длина команды – 4 байта (вместо 8).

КОП	$R_1$	$B_2$	$D_2$
0	7 8	11 12	15 16

$R_1$  – номер РОН, в котором находится первый операнд (регистровая память содержит 16 регистров, значит для указания номера регистра нужно 4 бита.

$$\log_2 16 = 4$$

$B_2$  – номер РОН, в котором находится начальный адрес блока ОП, в котором находится второй операнд.

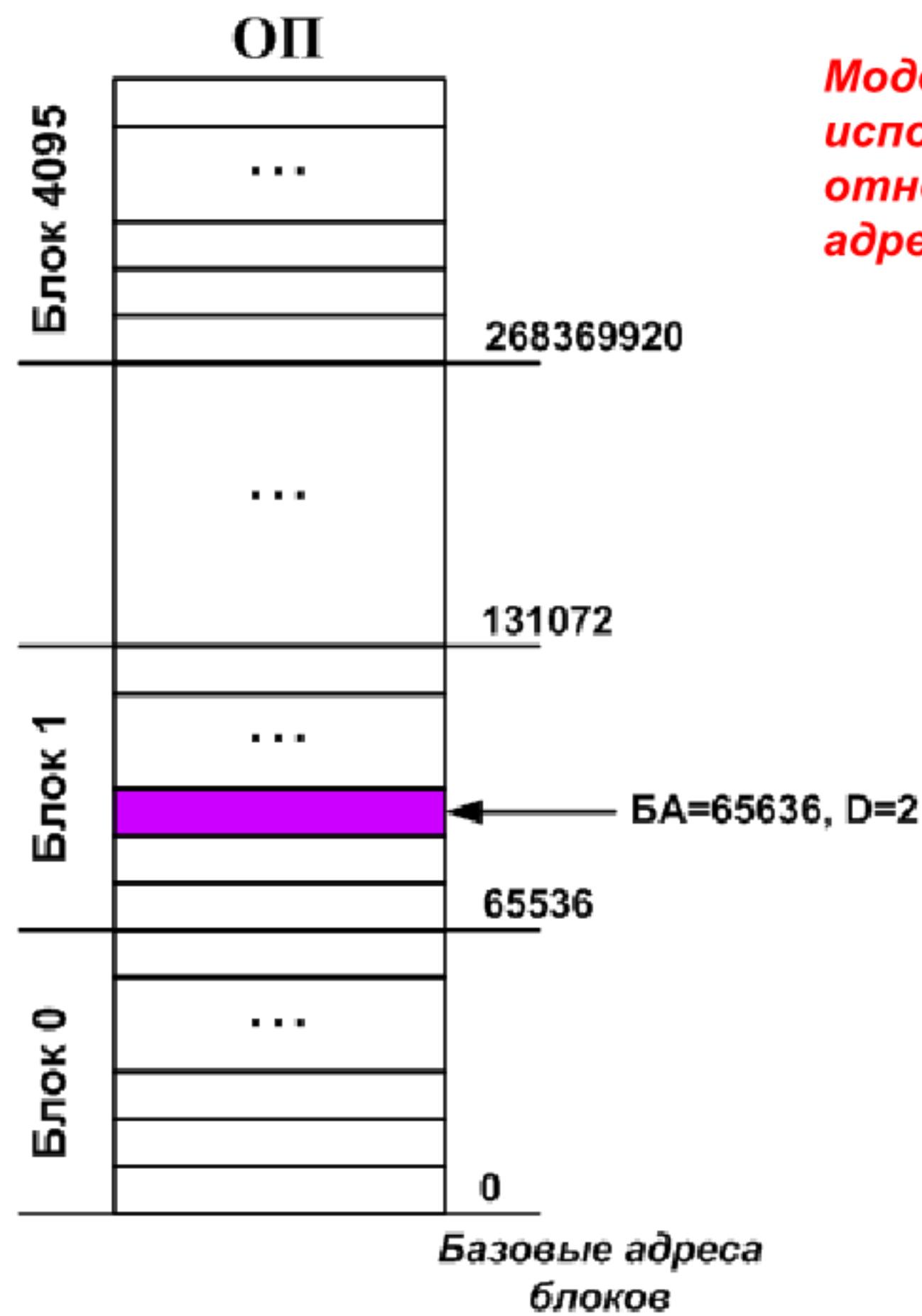
$D_2$  – смещение в блоке (адрес второго операнда относительно начала блока, в котором он находится).

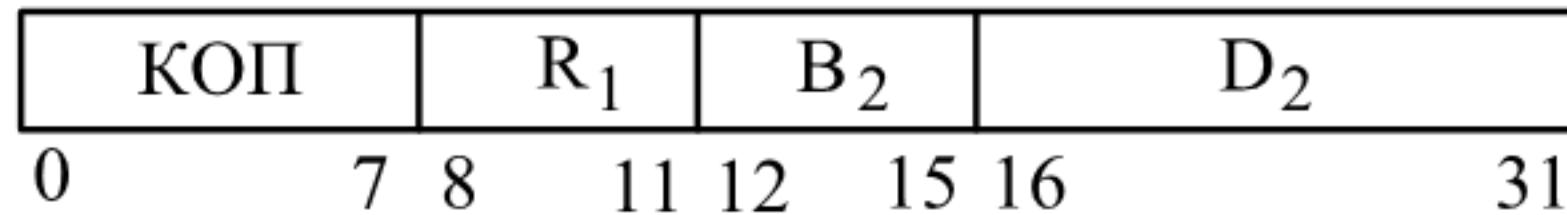
Вся ОП считается разделенной на блоки. Размер блока определяет длину поля  $D_2$ . В нашем случае размер блока равен

$$2^{16} \text{ байт} = 65636 \text{ байт} = 64 \text{ К байт}$$

В 32-разрядный РОН с номером  $B_2$  можно записать целое положительное беззнаковое (unsigned) число (адрес начального байта блока) в диапазоне от 0 до  $2^{32}-1$ . Это даже больше, чем нам нужно (в нашей памяти  $2^{28}$  байтов).

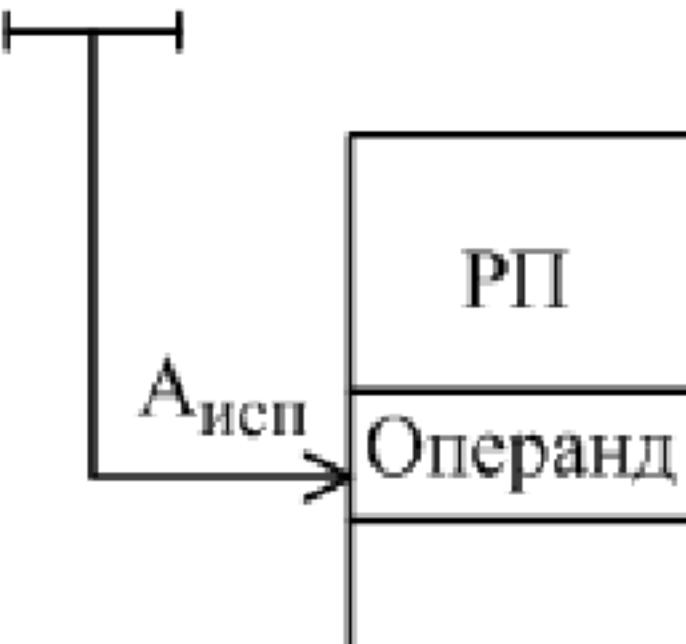
Количество блоков памяти:  $2^{28} : 2^{16} = 2^{12} = 4096$





**Первый операнд считывается из регистровой памяти (РП):**

$R_1$  – номер регистра РП



**Регистровая  
адресация:**

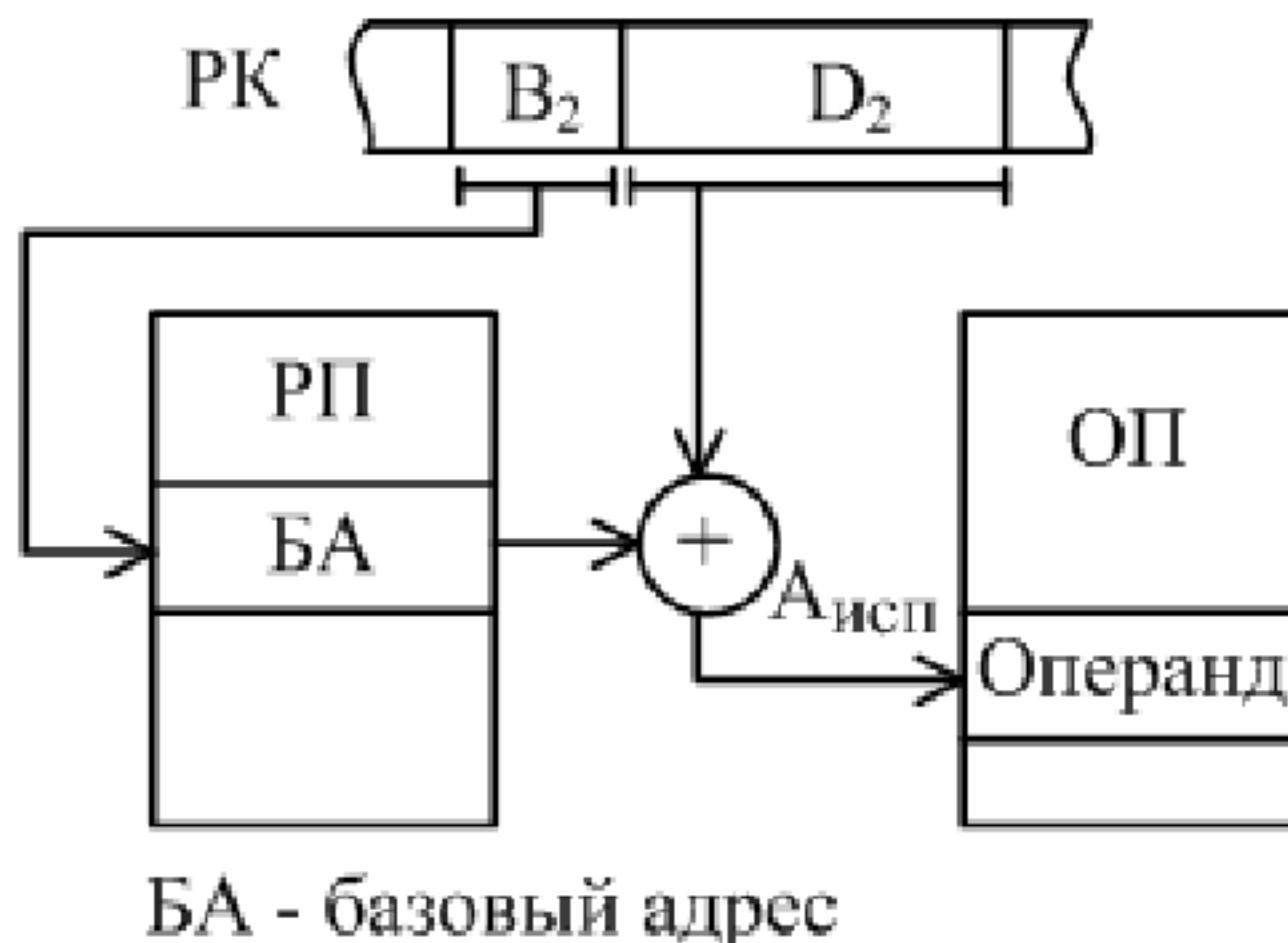
**адрес операнда  
равен номеру  
регистра  
 $A_{исп} = R_1$**

**Достоинства:** 1) малоразрядное поле для указания номера РОН;  
2) быстрое считывание операнда (регистровая память находится в процессоре и работает на частоте процессора).

## Второй operand считывается из оперативной памяти:

$B_2$  – номер регистра

$D_2$  – смещение



БА - базовый адрес

### Относительная адресация:

$$A_{исп} = BA + D_2$$

(БА считывается из регистра с номером  $B_2$ )

Пример 2. Команда логическая (например, дизъюнкция). Длина операндов – 1 байт. Первый operand находится в ОП, его **адрес** находится в регистре с номером  $R_1$ , второй operand задан непосредственно в команде в поле  $Im_2$  ( от immediate – непосредственный).

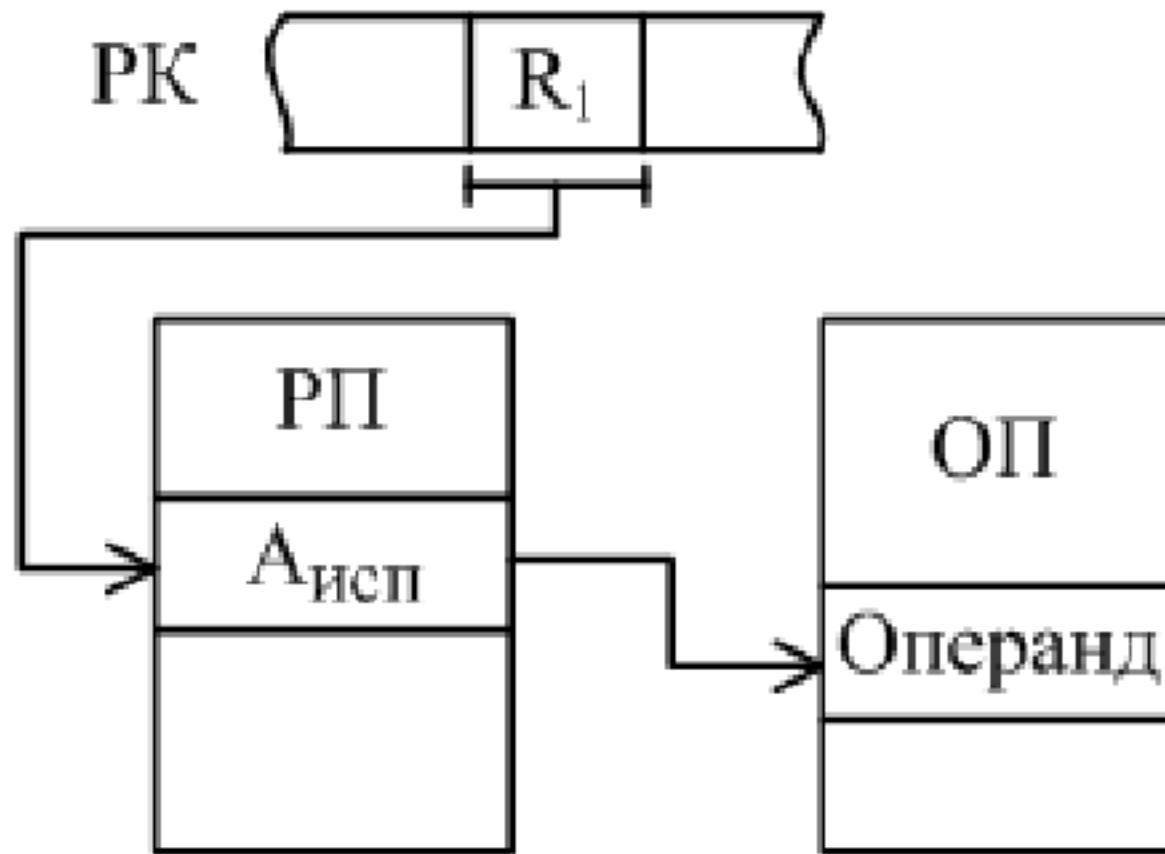
Способ адресации первого операнда – косвенно-регистровый, способ адресации второго операнда – непосредственный. Результат помещается в ОП на место первого операнда.

Формат команды:

КОП	$R_1(k)$		$Im_2$
0	7 8	11 12	15 16 23

Длина команды – 3 байта (вместо 8). Закрашенное поле не используется.

## Выборка первого операнда:

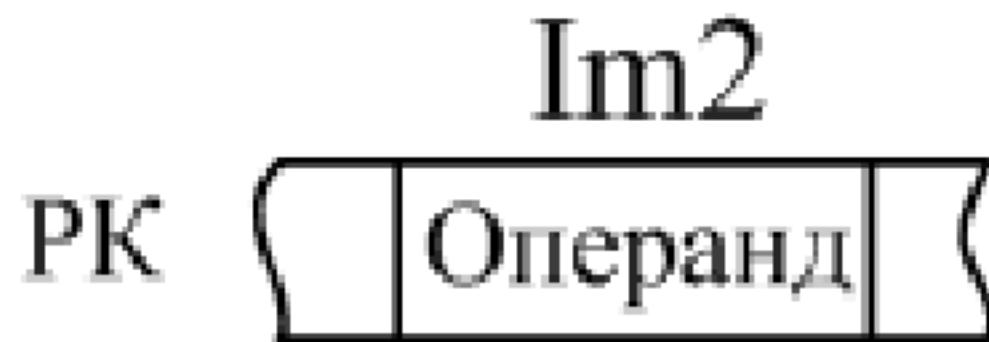


### Косвенно-регистровая адресация (косвенная через регистр):

Сначала из регистра с номером  $R_1$  считывается адрес операнда ( $A_{исп}$ ), затем по этому адресу из ОП считывается сам операнд.

В 32-разрядный РОН с номером  $R_1$  можно записать число (адрес байта памяти) в диапазоне от 0 до  $2^{32}-1$ . Это даже больше, чем нам нужно (в нашей памяти  $2^{28}$  байтов).

## Выборка второго операнда:



### Непосредственная адресация:

операнд выбирается **прямо из регистра команд (РК)**, где находится команда (из поля, соответствующего  $Im2$ ).

Достоинство метода: не нужно выполнять сравнительно долгую операцию обращения к ОП для выборки операнда, т.к. он уже выбран (на этапе выборки команды) и присутствует в РК процессора.

Недостаток: непосредственно в команде можно задавать только короткие операнды, например, константы сдвига.

### Пример 3. Команда передачи управления – «Переход по счетчику» 21 (используется для организации циклов).

К значению РОН (счетчика), номер которого указан в команде, прибавляется 1. Если результат этой операции не равен нулю, осуществляется переход (адрес перехода, указанный в программе, загружается в СК) в противном случае СК продвигается на длину команды перехода.

При указании прямого адреса перехода в команде, длина команды будет равна 8 (КОП) + 4 (R) + 28(A) = 40 бит = 5 байт.

Применение других способов адресации позволяет уменьшить длину команды.

КОП	R <sub>1</sub>	B <sub>2</sub>	D <sub>2</sub>
0	7 8	11 12	15 16

31

Длина команды – 4 байта (вместо 5).

Поле R<sub>1</sub> задает номер РОН, который используется в качестве счетчика. Адрес перехода вычисляется по правилам, принятым для относительной адресации (к содержимому РОН с номером B<sub>2</sub> (базовому адресу) прибавляется смещение D<sub>2</sub>. Вычисленный Аисп используется не для обращения к памяти, а для загрузки в СК (при ненулевом значении счетчика)

Можно уменьшить длину команды до 2 байтов, если адрес перехода задать в РОН с номером  $R_2$ .

КОП	$R_1$	$R_2$
0	7 8	11 12 15

#### Пример 4. Команда пересылки информации. Данное из ОП

пересыпается в РОН процессора (загрузка регистра). Длину данного (число байтов) определяет КОП. Команды, работающие с данными разной длины, имеют разные КОП.

Адрес данного в ОП задан косвенно полем  $A_2(Y)$ . Номер регистра, в который пересыпается информация задан полем  $R_1$  (регистровая адресация).

Формат команды:

КОП	$R_1$		$A_2(Y)$
0	7 8	11 12	15 16 23

Длина команды – 3 байта. Для команды пересылки данного из РОН в ОП (запись в память) можно использовать тот же формат.

Укороченный адрес  $A_2(Y)$  – адрес 4-байтного слова, находящийся в области младших адресов ОП (от 0 до 255, т.к. поле адреса занимает 8 бит). В это четырехбайтное (32-битное) слово предварительно (одной из предыдущих команд программы) записан адрес операнда  $A_{исп}$ .

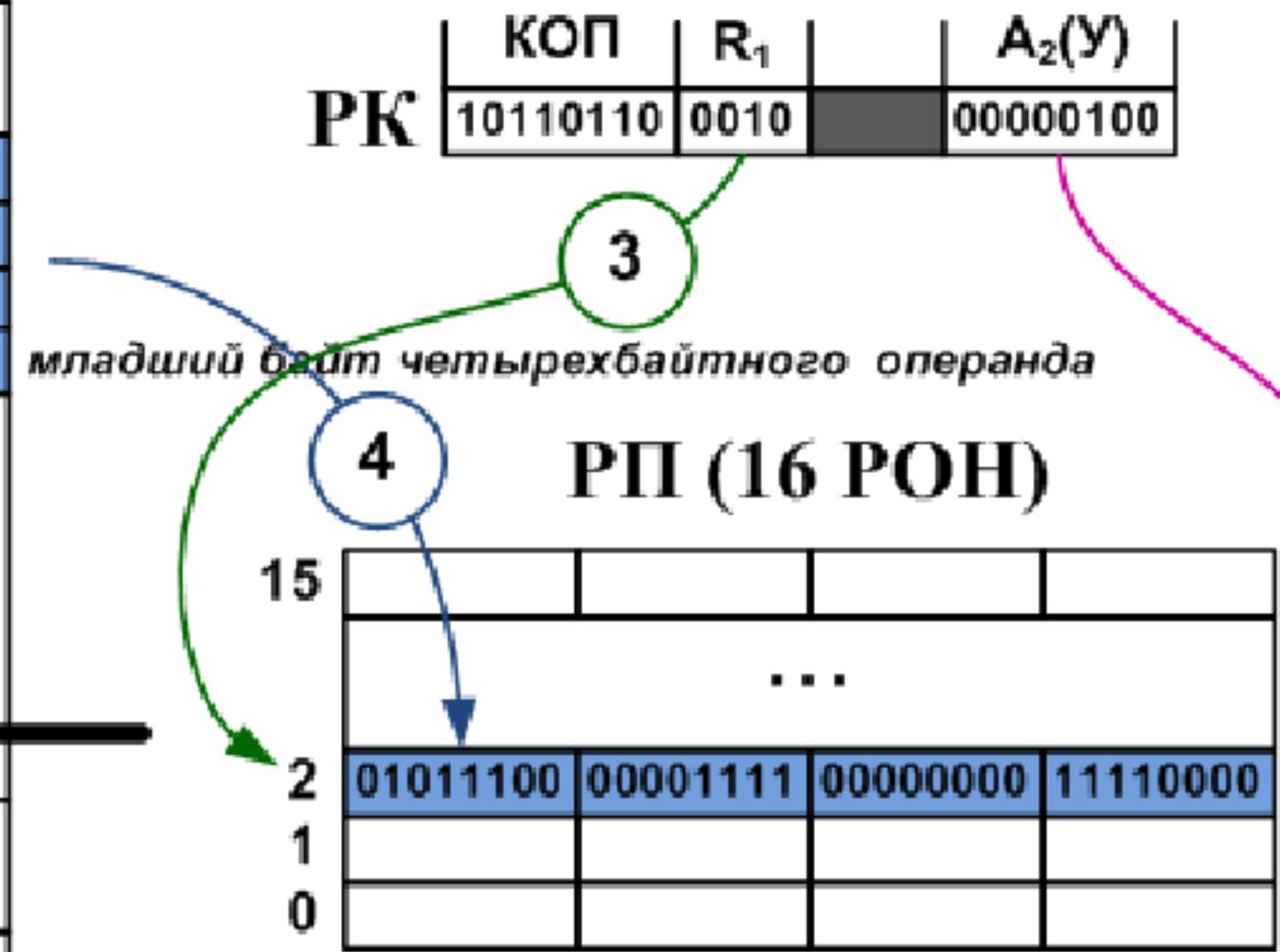
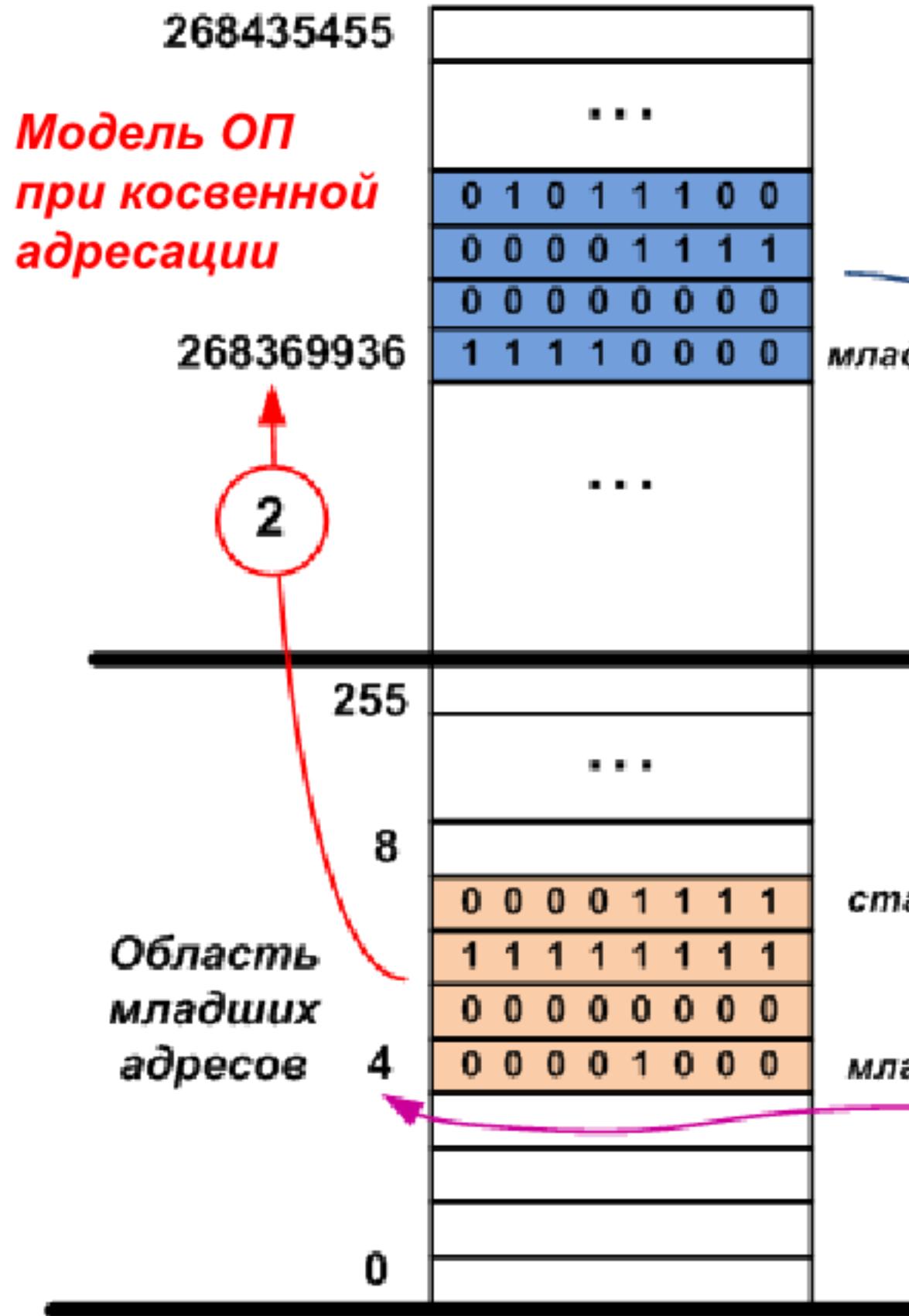
В 32-разрядное слово можно записать число (адрес байта памяти) в диапазоне от 0 до  $2^{32}-1$ . Это даже больше, чем нам нужно (в нашей памяти  $2^{28}$  байтов).

## Выполнение команды

*Следующая иллюстрация сделана при таких предположениях:*

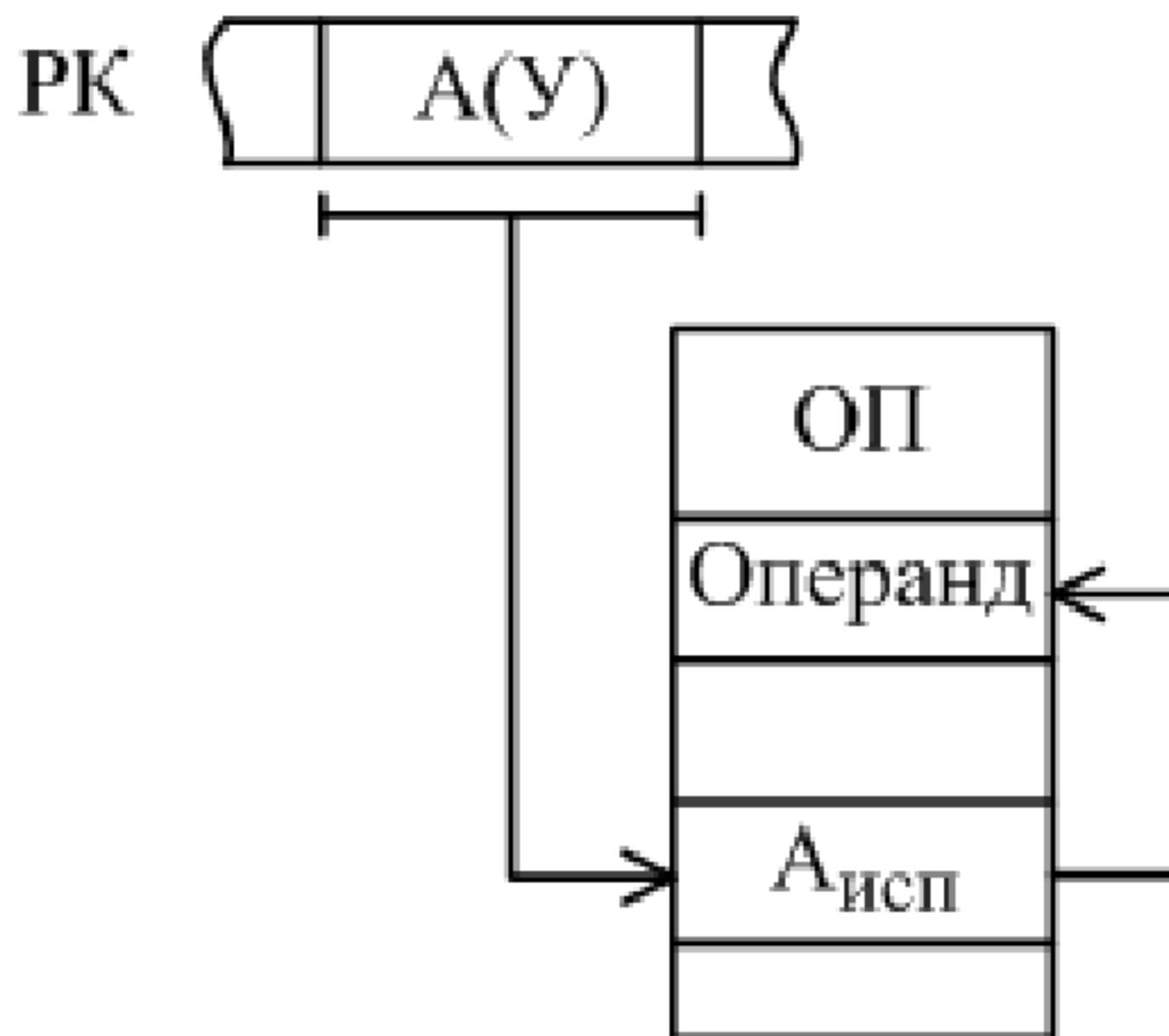
- 1) для многобайтных числовых данных в меньших адресах хранятся младшие байты чисел;**
- 2) КОП подразумевает данные длиной 4 байта.**

## Взаимодействие регистров процессора и ОП



## Схема выборки операнда при косвенной адресации

$A(Y)$  – укороченный адрес ОП



## Неявная адресация

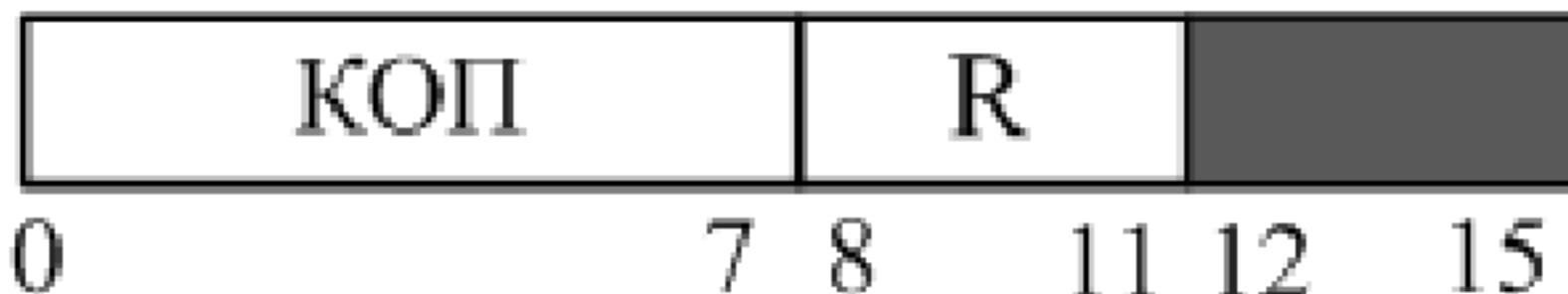
Пример 5. Команда сложения аккумулятора с РОН.

Аккумулятор – это регистр АЛУ, в который помещается операнд, а после выполнения операции – результат.

Номер РОН, в котором находится второй операнд, задан полем R. КОП подразумевает использование аккумулятора в качестве первого операнда.

Результат операции остается в аккумуляторе, и может быть использован как операнд следующей арифметической операции того же формата.

Формат команды:



Длина команды – 2 байта.

**Пример 5.а. Тот же пример для системы команд конкретного (восьмибитного) микроконтроллера КМ1816ВЕ51. Как видно из таблицы, команда сложения содержимого аккумулятора и регистра занимает всего 1 байт, который отводится под КОП.**

Группа команд арифметических операций

Название команды	Мнемокод		КОП	Операция
Сложение аккумулятора с регистром ( $n = 0 \div 7$ )	ADD	A, Rn	00101 <del>пп</del>	(A) $\leftarrow$ (A) + (Rn)
Сложение аккумулятора с прямоадресуемым байтом	ADD	A, ad	00100101	(A) $\leftarrow$ (A) + (ad)
Сложение аккумулятора с байтом из РПД ( $i = 0,1$ )	ADD	A, @Ri	0010011i	(A) $\leftarrow$ (A) + ((Ri))
Сложение аккумулятора с константой	ADD	A, #d	00100100	(A) $\leftarrow$ (A) + # d
Сложение аккумулятора с регистром и переносом	ADDC	A, Rn	00111 <del>пп</del>	(A) $\leftarrow$ (A) + (Rn) + (C)
Сложение аккумулятора с прямоадресуемым байтом и переносом	ADDC	A, ad	00110101	(A) $\leftarrow$ (A) + (ad) + (C)
Сложение аккумулятора с байтом из РПД и переносом	ADDC	A, @Ri	0011011i	(A) $\leftarrow$ (A) + ((Ri)) + (C)
Сложение аккумулятора с константой и переносом	ADDC	A, #d	00110100	(A) $\leftarrow$ (A) + # d + (C)

**Как разработчикам удалось уменьшить размер команды до 1 байта?**

Аккумулятор подразумевается кодом операции неявно, под номер регистра общего назначения (РОН) в КОП отводится 3 бита. Т.е. в команде есть возможность адресоваться к восьми РОН (с номерами от 000 до 111).

Но в микроконтроллере не 8, а 32 регистра! А чтобы адресоваться к 32 регистрам нужно 5 разрядов адреса! 3 разряда в КОП. Где взять еще два?

# А вот они – в регистре специальных функций – PSW (CCP)

**32 регистра  
разделены на 4  
банка. В каждом  
банке по 8  
регистров. Перед  
использованием  
рассматриваемой  
команды  
сложения  
программист  
должен  
установить  
номер банка  
регистров, к  
которому будет  
обращение.**

Таблица 3.2. Формат слова состояния программы (CCP)

Символ	Позиция	Имя и назначение	
C	PSW.7	Флаг переноса. Устанавливается и сбрасывается аппаратурными средствами или программой при выполнении арифметических и логических операций	
AC	PSW.6	Флаг вспомогательного переноса. Устанавливается и сбрасывается только аппаратурными средствами при выполнении команд сложения и вычитания и сигнализирует о переносе или заеме в бите 3	
F0	PSW.5	Флаг 0. Может быть установлен, сброшен или проверен программой как флаг, специфицируемый пользователем	
RS1	PSW.4	Выбор банка регистров. Устанавливается и сбрасывается программой для выбора рабочего банка регистров (см. примечание)	
RS0	PSW.3		
OV	PSW.2	Флаг переполнения. Устанавливается и сбрасывается аппаратурно при выполнении арифметических операций	
–	PSW.1	Не используется	
R	PSW.0	Флаг паритета. Устанавливается и сбрасывается аппаратурно в каждом цикле команды и фиксирует нечетное/четное число единичных бит в аккумуляторе, т.е. выполняет контроль четности	
Примечание. Выбор рабочего банка регистров			
RS1	RS0	Банк	Границы адресов
0	0	0	00H-07H
0	1	1	08H-0FH
1	0	2	10H-17H
1	1	3	18H-1FH

Название команды	Мнемокод	КОП	Т	В	Ц	Операция
Сброс переноса	CLR C	11000011	1	1	1	(C) $\leftarrow 0$
Сброс бита	CLR bit	11000010	4	2	1	(b) $\leftarrow 0$
Установка переноса	SETB C	11010011	1	1	1	(C) $\leftarrow 1$
Установка бита	SETB bit	11010010	4	2	1	(b) $\leftarrow 1$
Инверсия переноса	CPL C	10110011	1	1	1	(C) $\leftarrow \bar{C}$
Инверсия бита	CPL bit	10110010	4	2	1	(b) $\leftarrow \bar{b}$
Логическое И бита и переноса	ANL C, bit	10000010	4	2	2	(C) $\leftarrow (C) \wedge (b)$
Логическое И инверсии бита и переноса	ANL C, /bit	10110000	4	2	2	(C) $\leftarrow (C) \wedge \bar{b}$
Логическое ИЛИ бита и переноса	ORL C, bit	01110010	4	2	2	(C) $\leftarrow (C) \vee (b)$
Логическое ИЛИ инверсии бита и переноса	ORL C, /bit	10100000	4	2	2	(C) $\leftarrow (C) \vee \bar{b}$
Пересыпка бита в перенос	MOV C, bit	10100010	4	2	1	(C) $\leftarrow b$
Пересыпка переноса в бит	MOV bit, C	10010010	4	2	2	(b) $\leftarrow (C)$

Пусть в сложении участвует регистр с номером 18. Это второй регистр (010) во втором банке (10). Полный адрес: 10010 (как раз 18). Чтобы установить требуемый номер банка надо бит PSW.4 установить, а бит PSW.3 сбросить.

В приведенной таблице bit – это прямой адрес бита.

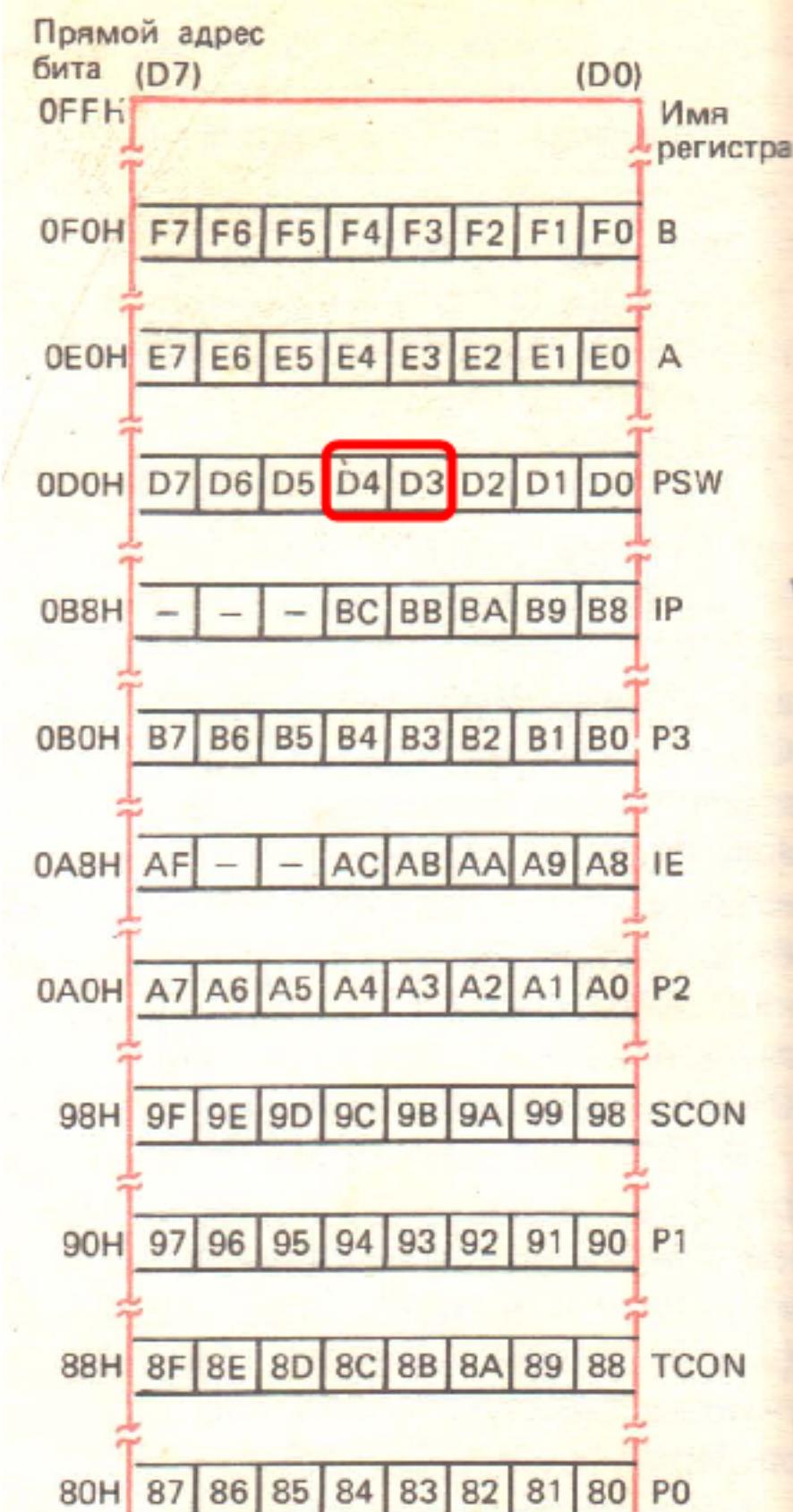


Рис. 3.21. Карта адресуемых бит в блоке регистров специальных функций

**Фрагмент программы, записывающий содержимое аккумулятора в регистр с номером 18 (второй регистр во втором банке).**

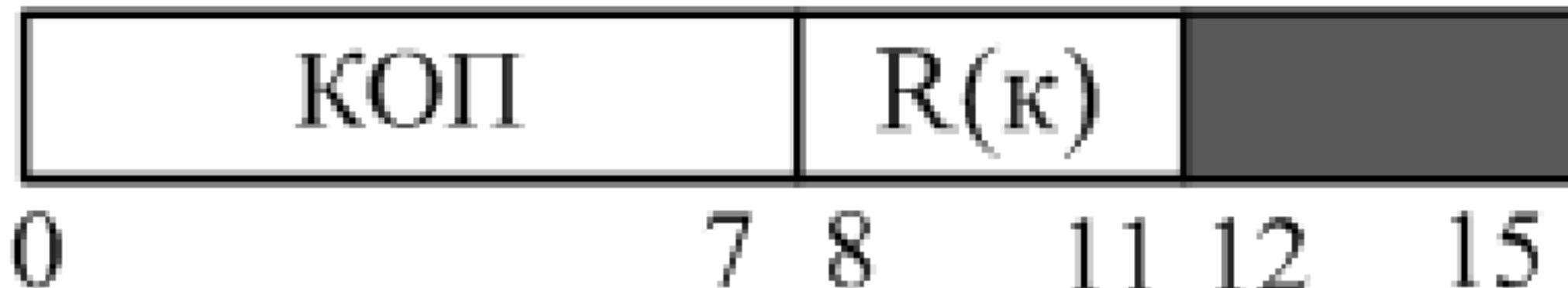
Ассемблер	Машинные коды	
	КОП	Второй байт
SETB D4	11010010	11010100
CLR D3	11000010	11010011
ADD A, R2	00101010	

**В двухбайтовых командах микроконтроллера КМ1816ВЕ51 второй байт используется для задания прямого адреса операнда (как в нашем примере) или константы (непосредственного операнда).**

Пример 6. Команда пересылки данного (результата предыдущей арифметической операции) из аккумулятора в ОП.

КОП подразумевает использование аккумулятора (неявная адресация) в качестве регистра, информация из которого пересыпается в ОП. Адрес ОП берется из регистра с номером R (косвенно-регистровая адресация).

Формат команды:

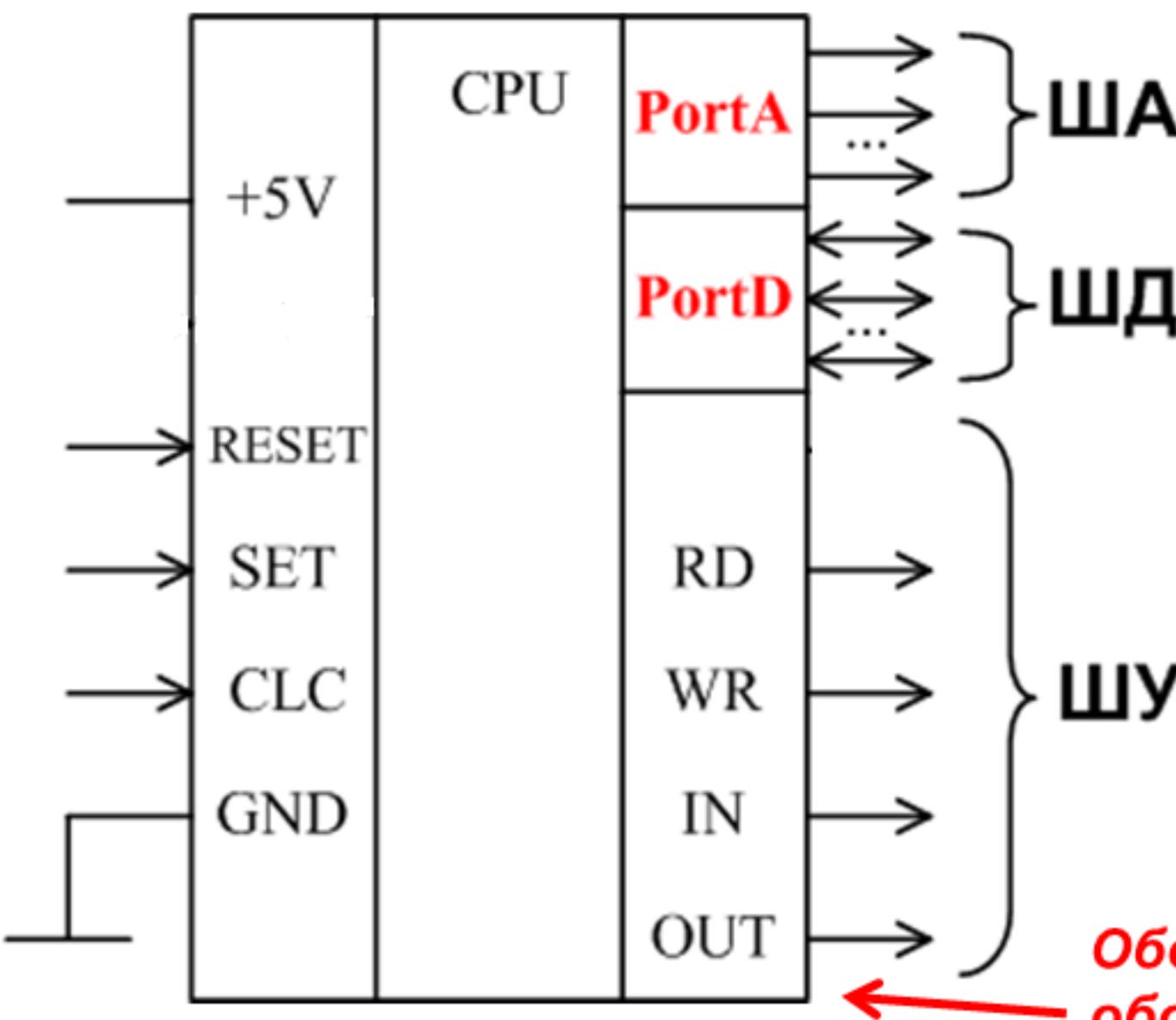


Длина команды – 2 байта.

Процессор помимо всей вышеперечисленной аппаратуры (СК, РК, РОН, РгПР, АЛУ и.т.д.) имеет специальные регистры: **порт адреса** (ПортА) и **порт данных** (ПортД).

**ПортA** подключен к внешним выводам процессора, которые, в свою очередь, подключены к шине адреса.

**ПортD** подключен к внешним выводам процессора, которые, в свою очередь, подключены к шине данных.



Взаимодействие процессора с ОП и внешними устройствами осуществляется через ПортA(ША), ПортD(ШД) «под руководством» **устройства управления и синхронизации** (УУиС) процессора.

**УУиС** выставляет на шину управления в нужные моменты времени активные уровни сигналов RD (WR) для чтения (записи) ОП или IN (OUT) для чтения (записи) внешних устройств.

**Обобщенное условно-графическое обозначение процессорного блока.**

Внешние устройства обмениваются с процессором, как правило, байтами.

Нужный байт должен быть предварительно записан в младший байт порта данных (ПортД) процессора (в системе команд процессора существуют команды пересылки информации в порты из аккумулятора или РОН).

Впорт адреса процессор из соответствующего поля РК выставляет номер (адрес) устройства ввода вывода (НУВВ). Число ВУ в системе, как правило, меньше 256, значит возможна прямая адресация (указание полного адреса устройства).

Формат команды:

КОП	НУВВ
0	7 8 15

НУВВ – номер  
устройства  
ввода-вывода

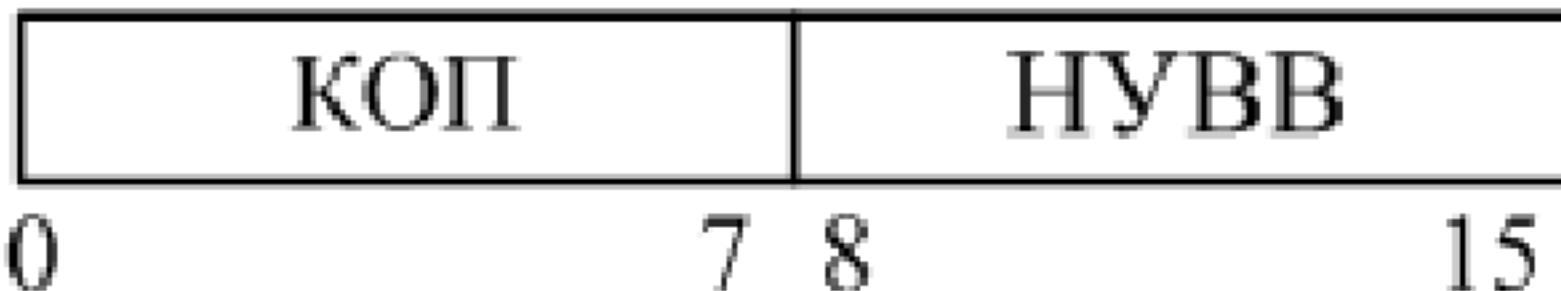
Длина команды – 2 байта.

Порт данных подразумевается кодом операции (неявная адресация).

Если КОП подразумевает команду вывода из процессора во внешнее устройство, то в процессе ее выполнения УУиС выдаст на шину управления активный уровень сигнала OUT.

## 6.2. Формат команды ввода байта из внешнего устройства в процессор.

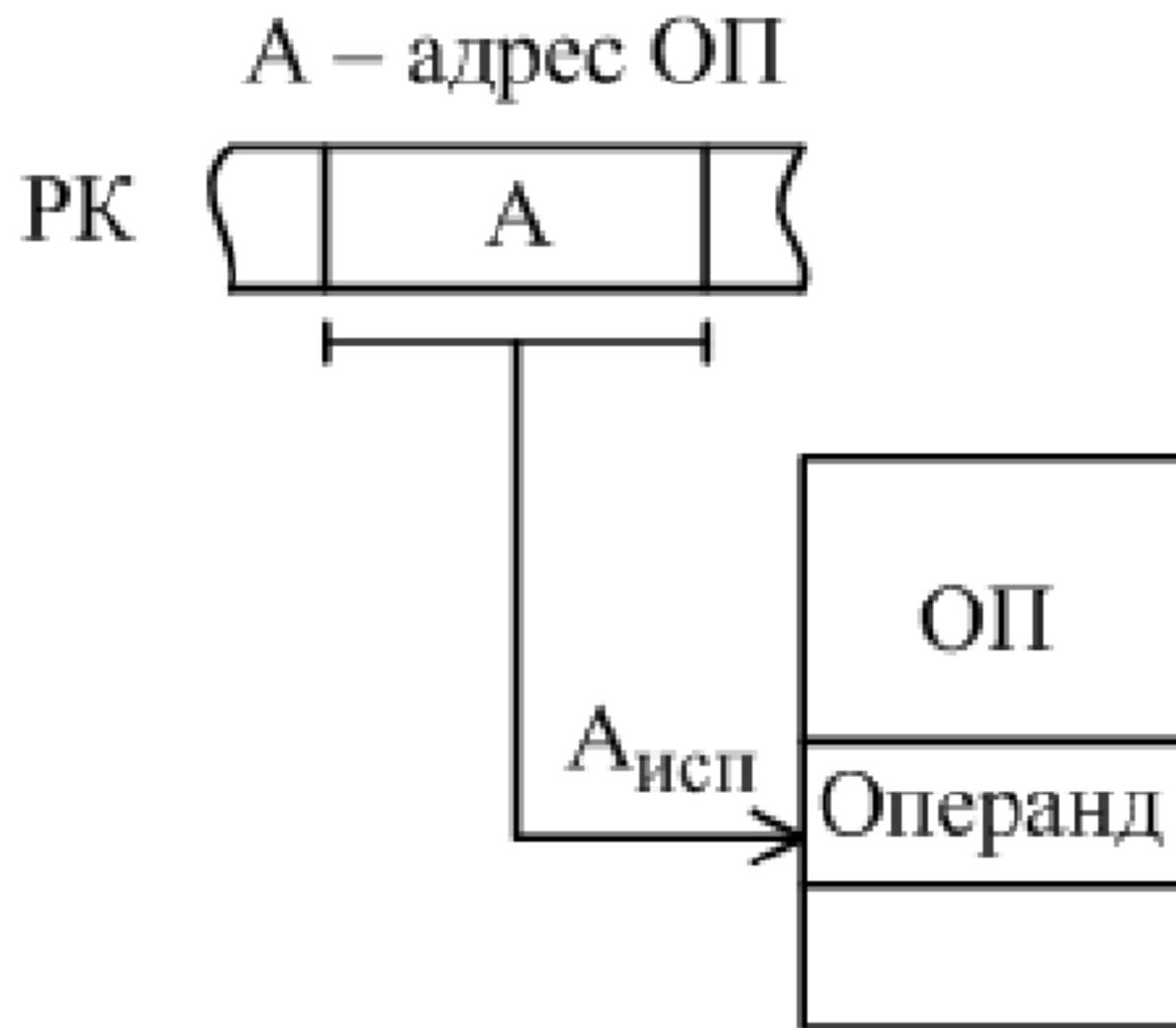
Формат команды ввода аналогичен формату команды вывода. КОП команды ввода отличается от КОП команды вывода.



При выполнении команды ввода байта из внешнего устройства процессор копирует в ПортА значение НУВВ, заданное в соответствующем поле РК, затем УУиС процессора выдает на шину управления активный уровень сигнала IN. В результате этого байт, выставленный на шину данных внешним устройством, попадает в младший байт ПортД.

Далее эта информация может быть переписана из ПортД в РОН или аккумулятор (в системе команд процессора, как правило, есть соответствующие команды пересылки информации).

## Прямая адресация

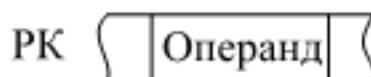


В команде задан полный адрес операнда. При большом размере адресного пространства не используется. В противном случае команды, а, следовательно, и программа в целом, имели бы слишком большую длину.  
Для сокращения длины команды и служат все остальные способы адресации.

## Способы адресации:

- а) непосредственный (Н),
- б) прямой (П),
- в) косвенный (К),
- г) регистровый (Р),
- д) косвенный через регистр (KP),
- е) относительный (О),

ж) неявный (не изображен), когда используемый в команде элемент процессора (регистр, аккумулятор, порт данных) подразумевается неявно в КОП.

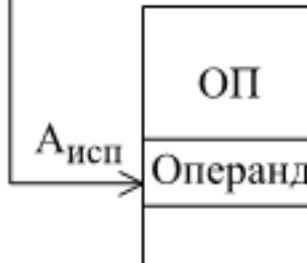


а)

А – адрес ОП

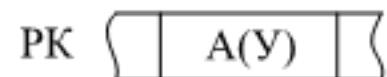


↓



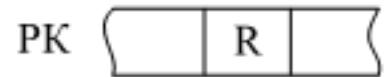
б)

А (У) – укороченный адрес ОП



в)

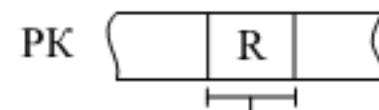
R – номер регистра РП



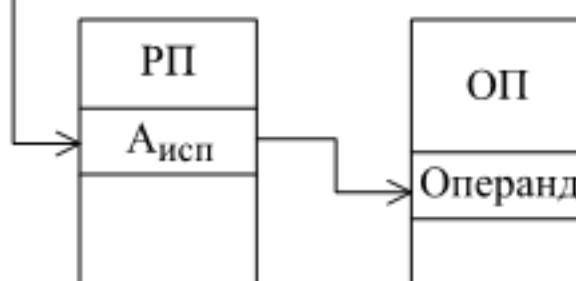
↓



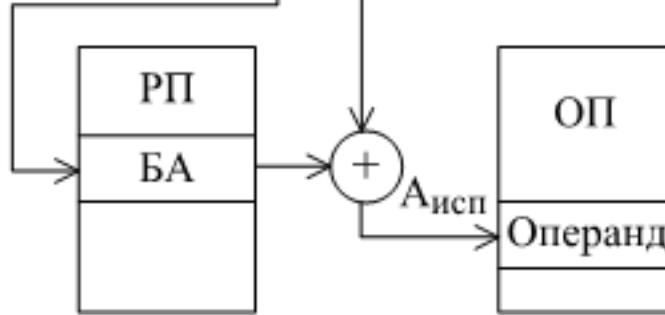
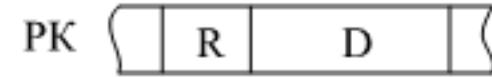
г)



д)



D - смещение



е)

### Нужно понимать:

- 1) ОП работает более медленно по сравнению с процессором (считывание из ОП и запись в ОП может занимать несколько тактов работы процессора). Регистровая память, находящаяся внутри процессора работает на частоте процессора. Поэтому команды, операнды которых находятся в РОН, выполняются гораздо быстрее команд, операнды которых находятся в ОП (даже если используется их прямая адресация).
- 2) Использование косвенно-регистровой адресации увеличивает время выполнения команды на время выборки адреса операнда из РОН.
- 3) Использование относительной адресации увеличивает время выполнения команды на время выборки базового адреса операнда из РОН и время его сложения со смещением;
- 4) Дольше всего выполняется выборка операнда при косвенной адресации, т.к. используется двойное обращение к ОП.

### Устройства, входящие в состав процессора

- 1) УУиС (управляет работой всех других устройств);
- 2) СК (длина определяется количеством битов адреса в системе, зависящим от объема оперативной памяти в байтах);
- 3) РК (длина определяется максимальной длиной команды);
- 4) АЛУ, имеющее два регистра для операндов, один из которых является аккумулятором (длина регистров определяется максимальной длиной операнда);

#### *Регистры специальных функций (РСФ)*

- 5) РгПР (двухразрядный);
- 6) РФ и РМ (разрядность определяется количеством возможных прерываний в системе);
- 7) УС (длина определяется количеством битов адреса в системе, зависящим от объема оперативной памяти в байтах);
- 8) ПортА (длина равна количеству битов (линий) шины адреса).
- 9) ПортД (длина равна количеству битов (линий) шины данных).

#### *Регистры общего назначения (РОН)*

- 10) Регистровая память, состоящая из ограниченного количества РОН;  
*и другие устройства*

## Виды команд, обрабатываемых процессором

- 1) Команды преобразования информации (выполняются АЛУ):
  - 1.1) арифметические команды;
  - 1.2) логические команды (поразрядные логические операции и операции сдвига);
- 2) Команды пересылки информации (между двумя РОН, между РОН и аккумулятором, РОН и ОП, РОН и ПортД, РОН и ПортА, и т.п.)
- 3) Команды перехода (передачи управления): безусловный переход, условный переход (по маске), переход по счетчику и др.;
- 4) Команды ввода-вывода (запись байта из ПортД во внешнее устройство, считывание данных из внешнего устройства в ПортД);
- 5) Команды управления режимами работы процессора, например, команды, позволяющие маскировать прерывания.

## Способы адресации operandов, используемые в командах:

- а) прямой,
- б непосредственный,
- в) регистровый,
- г) косвенный через регистр,
- д) относительный,
- е) косвенный,
- ж) неявный

## Форматы данных, обрабатываемых процессором

- 1) Целые двоичные числа со знаком (форматы с фиксированной точкой);
- 2) Целые двоичные числа без знака – положительные (форматы с фиксированной точкой);
- 3) Двоичные вектора (форматы с фиксированной точкой);
- 4) Вещественные числа со знаком (форматы с плавающей точкой);
- 5) Коды символов (однобайтные или двухбайтные);
- 6) Логические (булевские) данные (`false` представляется байтом, равным нулю, `true` – байтом, не равным нулю).

Каждый процессор исполняет определенный набор команд, который называется системой команд процессора.

Данные о системе команд конкретного процессора можно получить из соответствующих справочников.

В качестве примера приведем фрагмент описания системы команд восьмиразрядного микропроцессора (микроконтроллера) КМ1816ВЕ51.

Для каждой команды приводится ее название, мнемокод (запись на языке ассемблера), машинный код (КОП), Т – тип команды (определяет ее формат, т.е. способы адресации операндов), Б – длина команды в байтах, Ц – длительность команды (число машинных циклов, требуемое для исполнения команды), описание выполняемой операции (команда – это приказ на исполнение некоторой операции).

Таблица А.12 Группа команд передачи данных

Название команды	Мнемокод	КОИ	Т	В	Д	Операция
Пересылка в аккумулятор из регистра ( $i = 0 \div 7$ )	MOV A, R $n$	11101 $rr$	1	1	1	(A) $\leftarrow$ (R $n$ )
Пересылка в аккумулятор прямоадресуемого байта	MOV A, ad	11100101	3	2	1	(A) $\leftarrow$ (ad)
Пересылка в аккумулятор байта из РПД ( $i = 0,1$ )	MOV A, @R $i$	1110011 $i$	1	1	1	(A) $\leftarrow$ ((R $i$ ))
Загрузка в аккумулятор константы	MOV A, # d	01110100	2	2	1	(A) $\leftarrow$ # d
Пересылка в регистр из аккумулятора	MOV R $n$ , A	11111 $rr$	1	1	1	(R $n$ ) $\leftarrow$ (A)
Пересылка в регистр прямоадресуемого байта	MOV R $n$ , ad	10101 $rr$	3	2	2	(R $n$ ) $\leftarrow$ (ad)
Загрузка в регистр константы	MOV R $n$ , # d	01111 $rr$	2	2	1	(R $n$ ) $\leftarrow$ # d
Пересылка по прямому адресу аккумулятора	MOV ad, A	11110101	3	2	1	(ad) $\leftarrow$ (A)
Пересылка по прямому адресу регистра	MOV ad, R $n$	10001 $rr$	3	2	2	(ad) $\leftarrow$ (R $n$ )
Пересылка прямоадресуемого байта по прямому адресу	MOV add, ads	10000101	9	3	2	(add) $\leftarrow$ (ads)
Пересылка байта из РПД по прямому адресу	MOV ad, @R $i$	1000011 $i$	3	2	2	(ad) $\leftarrow$ ((R $i$ ))
Пересылка по прямому адресу константы	MOV ad, # d	01110101	7	3	2	(ad) $\leftarrow$ # d
Пересылка в РПД из аккумулятора	MOV @R $i$ , A	1111011 $i$	1	1	1	((R $i$ )) $\leftarrow$ (A)
Пересылка в РПД прямоадресуемого байта	MOV @R $i$ , ad	0110011 $i$	3	2	2	((R $i$ )) $\leftarrow$ (ad)
Пересылка в РПД константы	MOV @R $i$ , # d	0111011 $i$	2	2	1	((R $i$ )) $\leftarrow$ # d
Загрузка указателя данных	MOV DPTR, # d16	10010000	13	3	2	(DPTR) $\leftarrow$ # d16
Пересылка в аккумулятор байта из ИП	MOV C A, @A + + DPTR	10010011	1	1	2	(A) $\leftarrow$ ((A) + + (DPTR))
	MOV C A, @A + PC	10000011	1	1	2	(PC) $\leftarrow$ (PC) + 1 (A) $\leftarrow$ ((A) + (PC))
Пересылка в аккумулятор байта из ВПД	MOVX A, @R $i$	11100011	1	1	2	(A) $\leftarrow$ ((R $i$ ))
Пересылка в аккумулятор байта из расширенной ВПД	MOVX A, @DPTR	11100000	1	1	2	(A) $\leftarrow$ ((DPTR))
Пересылка в ВПД из аккумулятора	MOVX @R $i$ , A	11110011	1	1	2	((R $i$ )) $\leftarrow$ (A)
Пересылка в расширенную ВПД из аккумулятора	MOVX @DPTR, A	11110000	1	1	2	((DPTR)) $\leftarrow$ (A)
Загрузка в стек	PUSH ad	11000000	3	2	2	(SP) $\leftarrow$ (SP) + 1 (SP) $\leftarrow$ (ad)
Извлечение из стека	POP ad	11010000	3	2	2	(ad) $\leftarrow$ (SP) (SP) $\leftarrow$ (SP) - 1

Как видно из предыдущей иллюстрации, каждая машинная команда имеет буквенный аналог – мнемонику (запись команды на языке ассемблера).

Язык ассемблера – язык программирования низкого уровня, жестко привязанный к конкретному процессору. Чтобы программировать на языке ассемблера нужно хорошо представлять себе структуру конкретного процессора и вычислительной системы в целом. Такое программирование – достаточно сложный, долгий и дорогостоящий процесс.

Человеку удобно записывать программу на языке, близком к естественному. Подобные языки называются языками высокого уровня. Они во многом абстрагируются от структуры конкретной вычислительной системы:

Pascal, C(«Си»), C++, C#, Java

Программа на языке программирования переводится в двоичные команды (процессора) специальной очень сложной программой (транслятором).

Если переводится сразу вся программа, то программа-транслятор называется компилятором. Как правило, при компиляции создается файл с расширением .exe, который можно загрузить в ОП и запустить на выполнение.



Если перевод делается постепенно, по отдельным «фразам», и переведенное тут же исполняется, программа-транслятор называется интерпретатором.

Программа-транслятор должна соответствовать операционной системе.

При разработке языка Java, авторы делали особый упор на мультиплатформенность (независимость от вычислительной платформы).

Вычислительная платформа – это совокупность аппаратных средств компьютера и операционной системы, под управлением которой он работает.

Авторы языка Java выбрали для трансляции комбинацию:

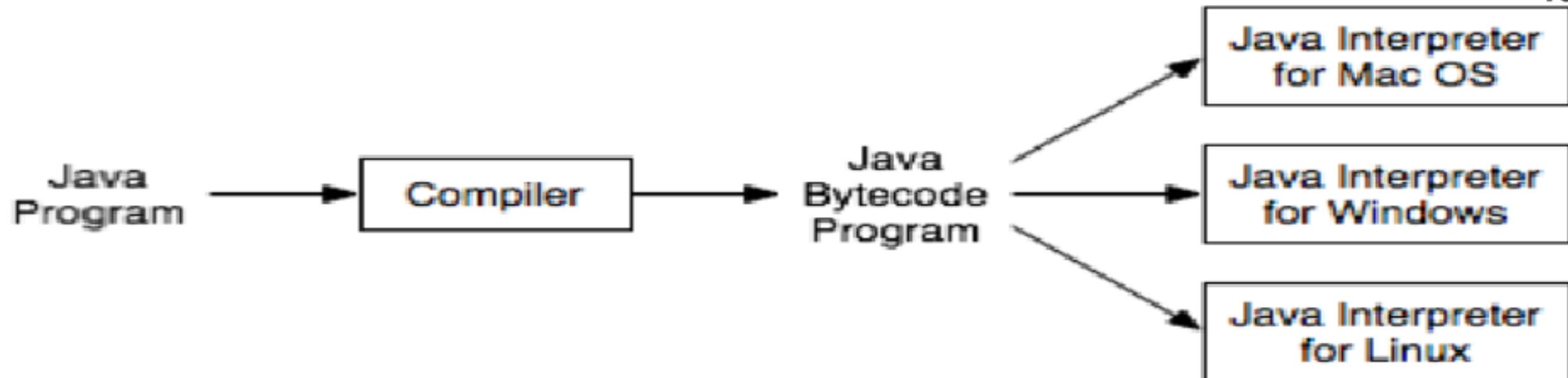
**компилятор + интерпретатор**

Компилятор переводит программу в байт-код.  
Интерпретатор выполняет команды байт-кода.

По сути, интерпретатор является Java-машиной (выполняет команды).

Поскольку машины на самом деле нет, а есть специальная программа, моделирующая ее работу, машину назвали «виртуальная», т.е. возможная.

Для каждого типа компьютеров разработана своя Java-машина, а компилятор необходим только один.



**Компилятор гораздо сложнее интерпретатора, поэтому сделать несколько разных, но одинаково хороших компиляторов намного сложнее, чем сделать несколько интерпретаторов.**

**Вопросы безопасности:** загружать чужую программу, непосредственно управляющую компьютером, опасно. Безопаснее загрузить «полуфабрикат» – (байт-код), который «пройдет» через Java-машину. В процессе обработки интерпретатор решает также вопросы, связанные с безопасностью.

### Достоинства технологии:

- Байт- код ( значит, и исходная программа) не зависит от платформы.
- Байт- код более безопасен, чем программа а командах процессора.
- Java– современный объектно-ориентированный язык высокого уровня, воплотивший все достижения «программистской мысли»

# Характеристики ЭВМ

1. Емкость (измеряется в гигабайтах) и архитектура ОП (принстонская или гарвардская).
2. Ширина выборки из ОП (разрядность шины данных). Чем больше ширина выборки, тем выше быстродействие компьютера (за одно обращение к ОП в 64-разрядных компьютерах выбирается сразу 8 байтов).
3. Операционные ресурсы (система команд процессора).
4. Типы обрабатываемых данных.

## 5. Производительность.

Из двух компьютеров, имеющих процессорные системы с одинаковой тактовой частотой, созданных на одинаковой элементной базе, один может быть более производительным, чем другой. На производительность компьютера влияют архитектурные особенности компьютера (размер шины данных, стратегии кэширования ОП, параметры видеокарты и др).

Смесь Гибсона – это тест, состоящий из специально подобранных команд и данных по которому интегрально определяется производительность компьютера.

Из двух компьютеров более производительным считается тот, на котором тест выполняется быстрее.

## Данные из Википедии

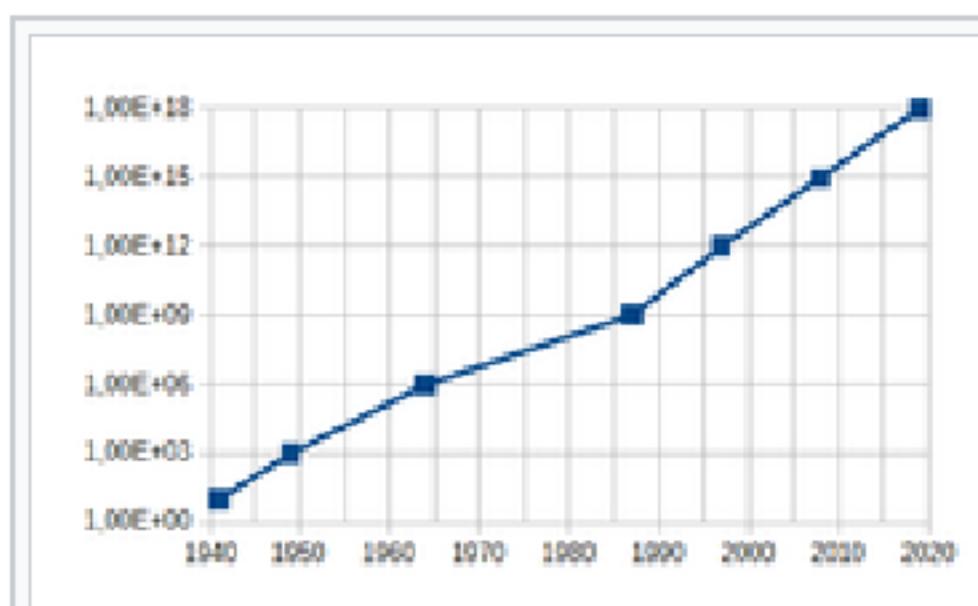
Вычислительная мощность компьютера (производительность компьютера) — это количественная характеристика скорости выполнения определённых [операций](#) на [компьютере](#). Чаще всего вычислительная мощность измеряется во [флопсах](#), от [англ.](#) *FLoating-point Operations Per Second* (количество операций с [плавающей запятой](#) в [секунду](#)), а также производными от неё. На данный момент принято причислять к [суперкомпьютерам](#) системы с вычислительной мощностью более 10 [teraфлопсов](#) ( $10 \times 10^{12}$  или десять триллионов флопсов; для сравнения среднестатистический современный [настольный компьютер](#) имеет производительность порядка 0.1 терафлопса). Одна из наиболее мощных на тесте [HPL](#) компьютерных систем — китайский [Sunway TaihuLight](#) — имеет производительность, превышающую несколько десятков петафлопсов.

## Производительность суперкомпьютеров

Название	год	флопсы
флопс	1941	$10^0$
килофлопс	1949	$10^3$
милафлопс	1964	$10^6$
гигафлопс	1987	$10^9$
терафлопс	1997	$10^{12}$
петафлопс	2008	$10^{15}$
эксафлопс	2019 или позже [1][2]	$10^{18}$
зеттафлопс	не ранее 2030 [1]	$10^{21}$

<https://ru.wikipedia.org/wiki/FLOPS>

**1 флопс =  $10^0$  = 1 оп/с**



Рост производительности  
суперкомпьютеров во флопсах

**6. Математическое (программное обеспечение) расширяет операционные ресурсы компьютера).**

- 1) системное ПО;
- 2) средства разработки;
- 3) прикладные программы

1) **Системное программное обеспечение** – это операционные системы, а также различные программы-утилиты для диагностики ресурсов компьютера (например, тестирования оперативной памяти), предоставления пользователю удобного способа взаимодействия с компьютером (например, команная строка), а также обслуживания ресурсов компьютера (например, разметка диска).

**Операционная система**, сокр. ОС (англ. operating system, OS) — комплекс взаимосвязанных программ, предназначенных для управления ресурсами вычислительного устройства и организации взаимодействия с пользователем (графический или текстовый интерфейс пользователя).

В логической структуре типичной вычислительной системы операционная система занимает положение между устройствами с их микроархитектурой, машинным языком и, возможно, собственными (встроеннымми) микропрограммами (драйверами) — с одной стороны — и прикладными программами с другой.

Разработчикам программного обеспечения операционная система позволяет абстрагироваться от деталей реализации и функционирования устройств, предоставляя минимально необходимый набор функций (интерфейс программирования приложений).

Понятие интерфейса вообще можно описать как набор методов для организации взаимодействия двух и более объектов. Интерфейс может быть между пользователем и программой, между программами, а также между программой и аппаратным обеспечением.

2) К средствам программирования относятся множество языков программирования, средства для автоматизации процесса создания программ, компиляторы и интерпретаторы.

Языки и системы программирования являются по своему назначению инструментами для создания действительно полезного ПО. С их помощью создается как прикладное так и системное программное обеспечение, а также новые средства разработки.

3) Огромную долю в ПО занимают прикладные программы, которые в свою очередь делят на универсальные и специализированные. Однако это деление в какой-то степени условно.

## КЛАССИФИКАЦИЯ ЭВМ (ПО НАЗНАЧЕНИЮ)

1. Общего назначения.
2. Проблемно-ориентированные
3. Специализированные

**1. ЭВМ общего назначения.** Предназначены для решения широкого класса задач, имеют универсальную систему команд (CISC-архитектура), обрабатывают большинство типов данных, характеризуются достаточно высокой производительностью, способностью работать в мультипрограммном режиме, используются в больших вычислительных центрах коллективного пользования. К этой же категории относятся и персональные компьютеры.

Двумя основными архитектурами набора команд, используемыми компьютерной промышленностью на современном этапе развития вычислительной техники являются архитектуры CISC и RISC. Основоположником CISC-архитектуры можно считать компанию IBM с ее базовой архитектурой IBM/360, ядро которой используется с 1964 года и дошло до наших дней, например, в таких современных мейнфреймах, как IBM ES/9000. Лидером в разработке микропроцессоров с полным набором команд (CISC – Complete Instruction Set Computer) считается компания Intel со своей серией x86 и Pentium. Эта архитектура является практическим стандартом для рынка микрокомпьютеров. Для CISC-процессоров характерно: сравнительно небольшое число регистров общего назначения; большое количество машинных команд, некоторые из которых нагружены семантически аналогично операторам высокого уровня языков программирования и выполняются за много тактов; большое количество методов адресации; большое количество форматов команд различной разрядности; преобладание двухадресного формата команд; наличие команд обработки типа регистр-память. Источник::

[http://www.erudition.ru/referat/printref/id.35668\\_1.html](http://www.erudition.ru/referat/printref/id.35668_1.html)

(микроконтроллеры), предназначенные для встраивания в качестве элемента управления в различные системы (системы управления технологическими процессами, бортовые системы управления и т.п.).

Если тактовая частота процессоров, используемых в универсальных ЭВМ, в основном, составляет **1,0 – 4 ГГц**, то частота современных микроконтроллеров, например, ATtiny2313/V фирмы Atmel, составляет всего **20 МГц** (быстродействие – 20 млн операций в секунду).

Микроконтроллеры, как правило имеют RISC-архитектуру (характеризуется урезанной системой команд, в частности, отсутствием операций с плавающей точкой). Система команд упрощается с целью увеличения быстродействия. Среди других особенностей RISC-архитектур следует отметить наличие достаточно большого регистрового файла (в типовых RISC-процессорах реализуются 32 или большее число регистров по сравнению с 8 – 16 регистрами в CISC-архитектурах), что позволяет большему объему данных храниться в регистрах на процессорном кристалле большее время и упрощает работу компилятора по распределению регистров под переменные.

Разрядность РОН, регистров АЛУ и портов ввода-вывода невелика (как правило, 8-разрядные). Микроконтроллеры имеют встроенные таймеры, могут иметь встроенные цифро-анalogовые (ЦАП) и аналого-цифровые (АЦП) преобразователи.

**Быстродействия этих контроллеров достаточно, чтобы осуществлять управление тем или иным объектом в режиме реального времени.**

### 3. Специализированные ЭВМ.

Применяются для супербыстрого решения задач определенного класса. Специализация применяется с целью увеличения быстродействия.

Классическая архитектура (фон-неймановская) не может дать требуемого быстродействия.

Используются специализированные архитектуры ЭВМ: матричные архитектуры, систолические процессоры, ассоциативные процессоры, нейронные сети (изучаются в отдельном курсе).