Лекция №2. Структура процессора. Микропрограммное управление

Галимов Р.Р.

Оренбургский государственный университет

2013

2.1 Микропрограммный автомат

Рисунок 2.1 – Структурная схема процессора

Операционное устройство **(ОУ)** — устройство, в котором выполняются операции. Оно включает в качестве узлов регистры, сумматоры, каналы передачи информации, мультиплексоры для коммутации каналов, шифраторы, дешифраторы и т.д.

Управляющее устройство **(УУ)** координирует действия узлов операционного устройства; оно вырабатывает в некоторой временной последовательности управляющие сигналы, под действием которых в узлах операционного устройства выполняются требуемые действия.

2.1 Микропрограммный автомат

Процесс выполнения программы в ВМ представляет собой последовательность машинных циклов.

Детализируем основные целевые функции, реализуемые устройством управления в ходе типового машинного цикла.

Для простоты примем, что ВМ обеспечивает одноадресную систему команд.

При этом, в частности, полагается, что до начала выполнения двухоперандной арифметической команды второй операнд уже находится в процессоре.

2.1 Микропрограммный автомат

- Первым этапом в машинном цикле является **выборка команды** из памяти (этап ВК). За выборкой команды следует этап декодирования ее операционной части (кода операции). Для простоты пока будем рассматривать этот этап в качестве составной части этапа ВК.
- Вторая целевая функция - формирование адреса следующей команды. На это выделяется специальный такт работы — этап ФАСК, которому соответствует целевая функция ЦФ-ФАСК.
- Далее следует этап формирования исполнительного адреса операнда или адреса перехода (этап ФИА), на котором УУ реализует функцию ЦФ-ФИА. Функция имеет столько модификаций, сколько способов адресации (СА) предусмотрено в системе команд ВМ.
- На четвертом этапе реализуется целевая функция **выборки операнда (ЦФ-ВО)** по исполнительному адресу, сформированному на предыдущем этапе.
- Наконец, на последнем этапе машинного цикла действия задаются целевой функцией исполнения операции - ЦФ-ИО. Очевидно, что количество модификаций ЦФ-ИО равно количеству операций, имеющихся в системе команд ВМ.

2.1 Микропрограммный автомат

Каждое элементарное действие, выполняемое в одном из узлов ОУ в течение одного тактового периода, называется ***микрооперацией.***

В определенные тактовые периоды одновременно могут выполняться несколько микроопераций.

Такая совокупность одновременно выполняемых микроопераций называется ***микрокомандой***, а весь набор микрокоманд, предназначенный для решения определенной задачи, — ***микропрограммой***.

2.2 Модель устройства управления процессора

Для выполнения своих функций **УУ** должно иметь входы, позволяющие определить состояние управляемой системы, и выходы, через которые реализуется управление поведением системы.

Рисунок 2.2 – Модель устройства управления

2.2 Модель устройства управления процессора

В зависимости от способа формирования микрокоманд различают микропрограммные автоматы:

- с жесткой или аппаратной логикой;
- с программируемой логикой.

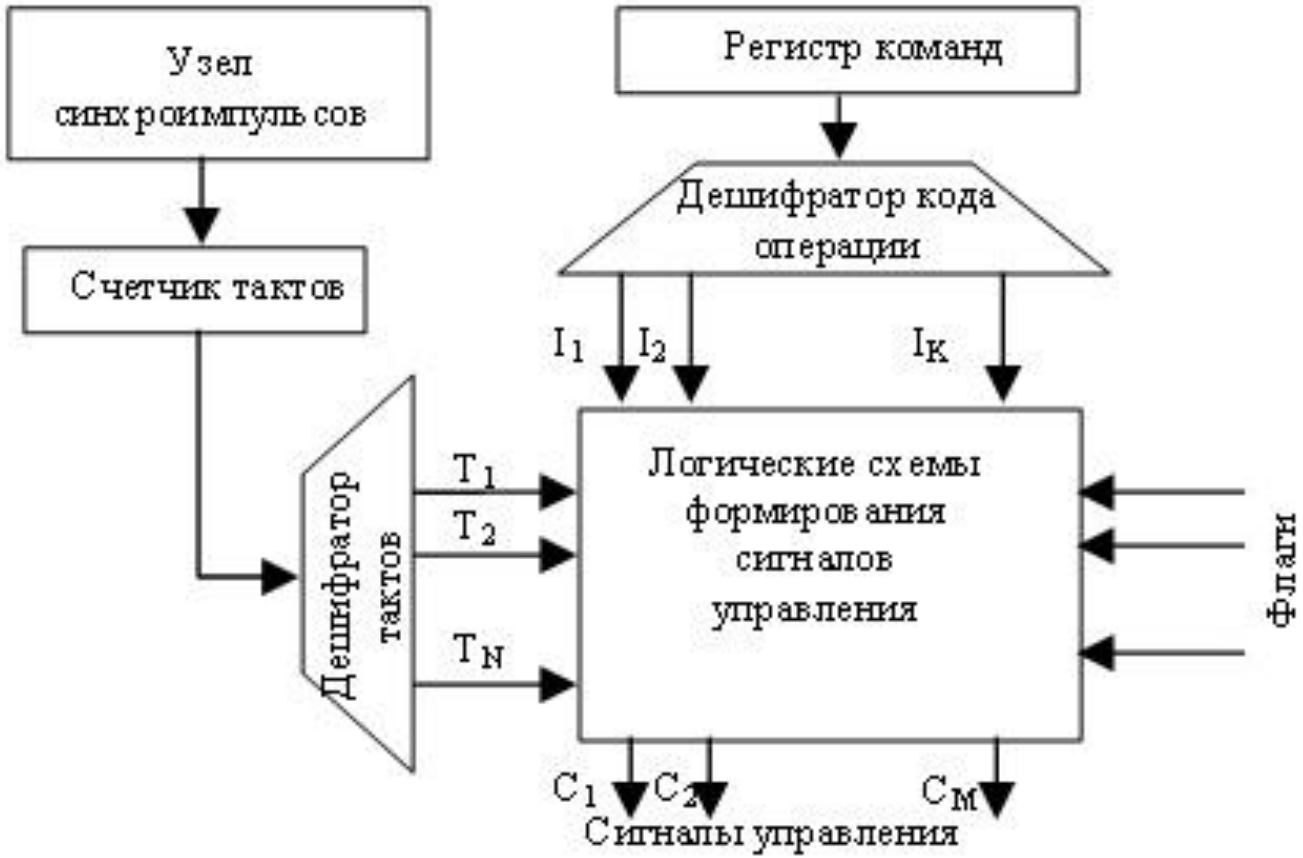
УУ с жесткой логикой управления имеет в своем составе МПА с жесткой (аппаратной) логикой.

При создании такого МПА выходные сигналы управления реализуются за счет однажды соединенных между собой логических схем.

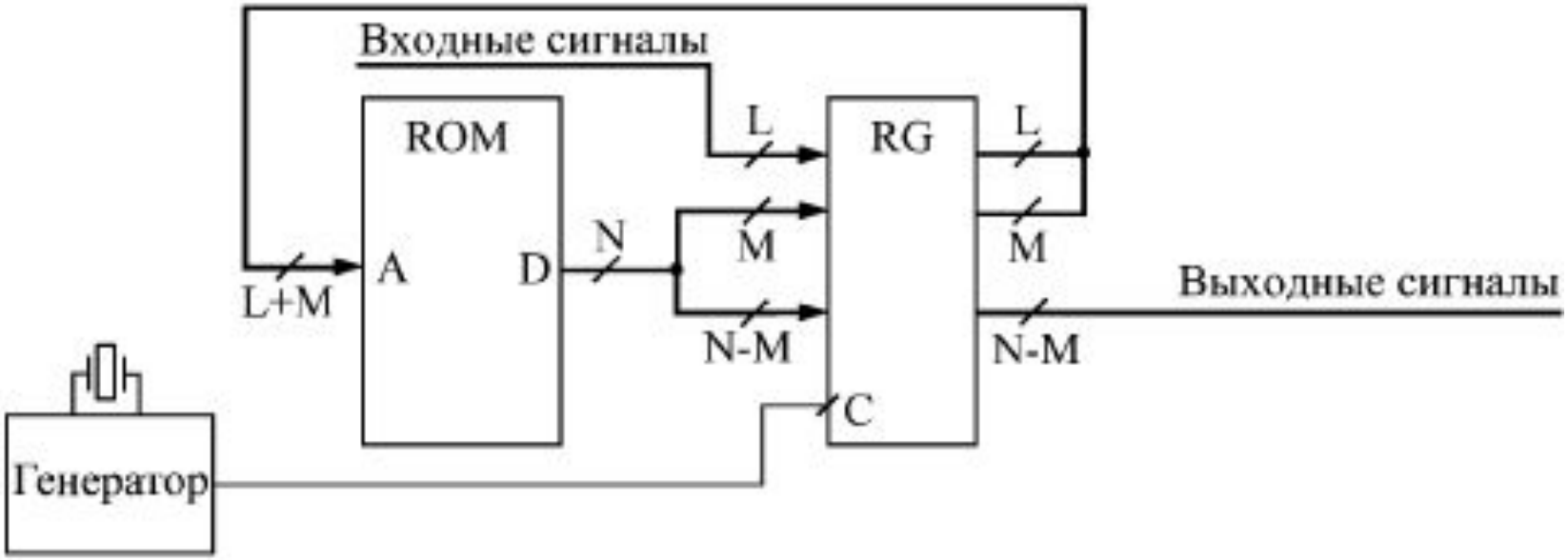
Отличительной особенностью **микропрограммного автомата с программируемой логикой** является хранение микрокоманд в виде кодов в специализированном запоминающем устройстве - памяти микропрограмм.

Каждой команде ВМ в этом ЗУ в явной форме соответствует микропрограмма, поэтому часто устройства управления, в состав которых входит микропрограммный автомат с программируемой логикой, называют микропрограммными.

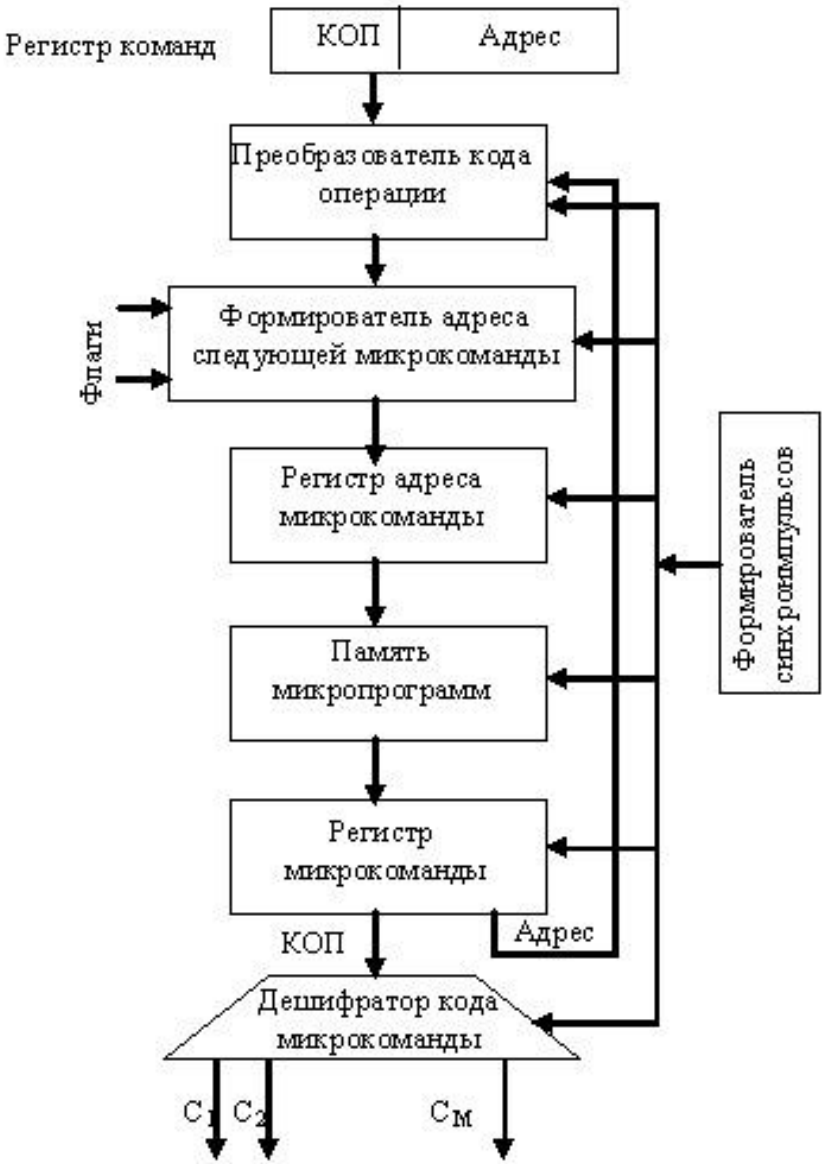
2.2 Модель устройства управления процессора



2.2 Модель устройства управления процессора



2.2 Модель устройства управления процессора



2.3 Программная модель процессора i8086

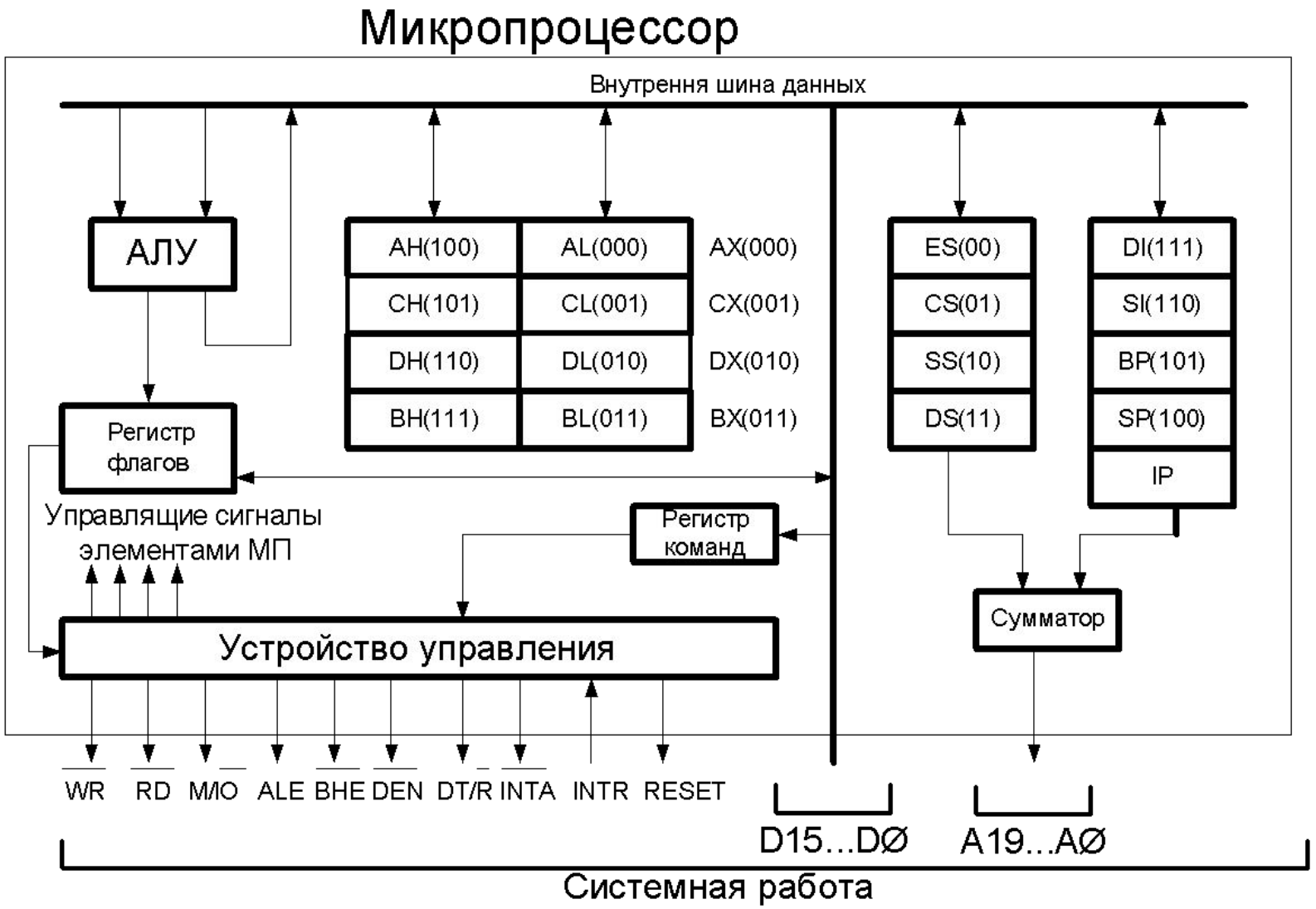


Рисунок 2.3 – Модель устройства управления

2.3 Программная модель процессора i8086

IP — регистр-счетчик текущего адреса программы (программный счетчик). После выборки из ЗУ очередного кода операции какой-либо команды его содержимое автоматически увеличивается на «1», подготавливая тем самым выборку следующего операнда из ОЗУ.

Регистры ***SI*** и ***DI*** предназначены, главным образом, для использования в специальных— цепочных (или строковых) командах, когда, например, необходимо «переместить» массив данных из одного места ОЗУ (ПЗУ) в другое место ОЗУ. В этих цепочных командах ***SI*** хранит адрес источника данных, а ***DI*** — адрес приемника данных. При этом после каждой пересылки одного операнда их содержимое автоматически увеличивается (или уменьшается— это как задано направление), подготавливая адреса транспортировки очередного операнда.

Группа регистров: AX, BX, CX и DX. Они могут функционировать как 16-разрядные регистры в количестве 4 штук или как 8-разрядные в количестве 8 штук. Размер регистра полностью определяется кодом операции команды. Эти регистры в командах используются самым различным образом. Поэтому они получили название: РОНы— регистры общего назначения.

2.3 Сегментная организация памяти

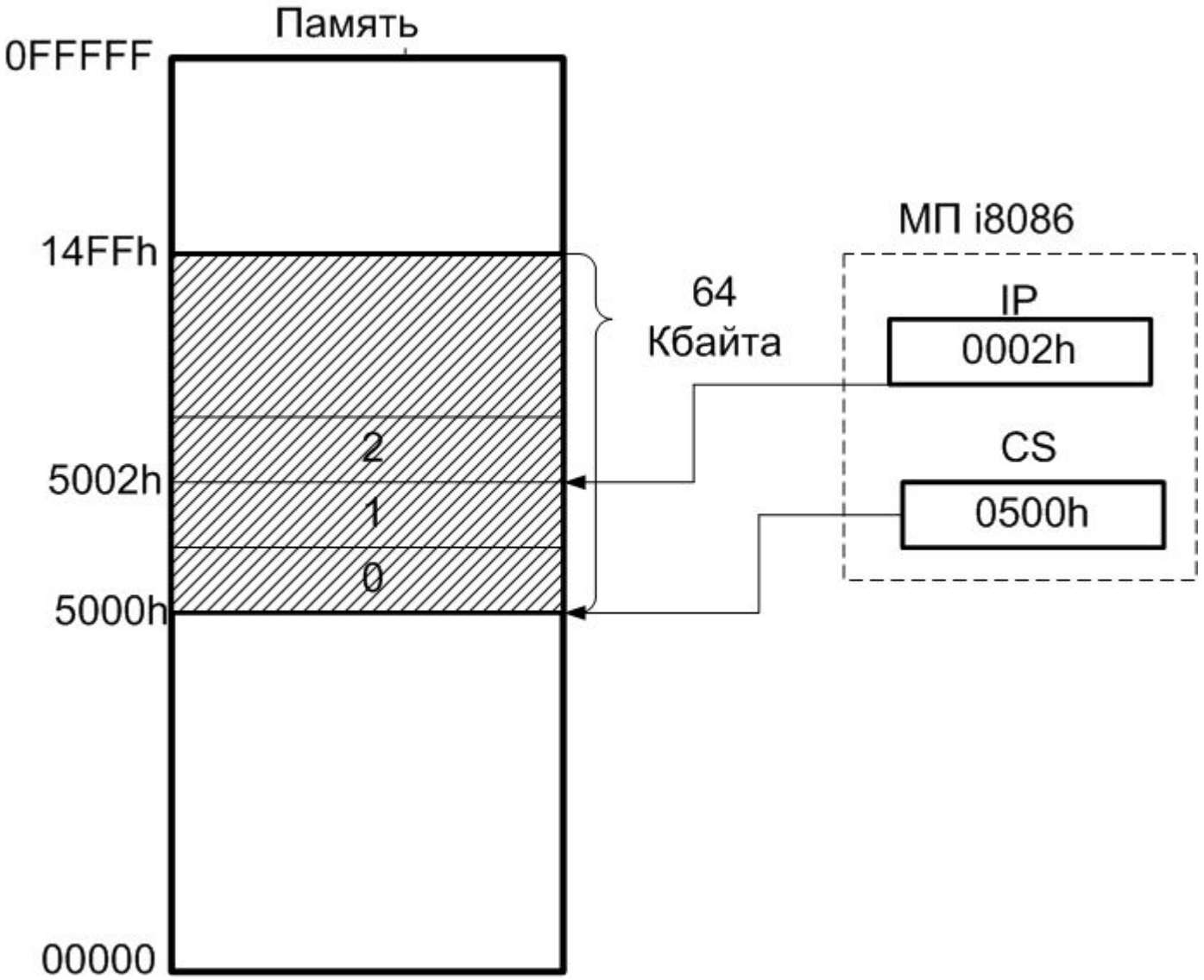


Рисунок 2.4 – Схема формирования физического адреса

Формула формирования физического адреса в МП i8086

$$
\begin{array}{ll}
RS: & A_{19}\ A_{18}\ A_{17}\ A_{16}\ A_{15}\ A_{14}\ A_{13}\ A_{12}\ A_{11}\ A_{10}\ A_9\ A_8\ A_7\ A_6\ A_5\ A_4\ 0\ 0\ 0\ 0 \\
+ & \\
EA: & 0\ 0\ 0\ 0\ A_{15}\ A_{14}\ A_{13}\ A_{12}\ A_{11}\ A_{10}\ A_9\ A_8\ A_7\ A_6\ A_5\ A_4\ A_3\ A_2\ A_1\ A_0 \\
\hline
ADDR: & A_{19}\ A_{18}\ A_{17}\ A_{16}\ A_{15}\ A_{14}\ A_{13}\ A_{12}\ A_{11}\ A_{10}\ A_9\ A_8\ A_7\ A_6\ A_5\ A_4\ A_3\ A_2\ A_1\ A_0
\end{array}
$$

где ***RS*** –сегментный регистр;
EA –эффективный адрес;
ADDR-физический адрес.

2.4 Стек

Стек является безадресной памятью. В большинстве современных процессоров реализован аппаратный стек, который представляет из себя специально организованное оперативное запоминающее устройство.

В МП 8086 под стек отводится область в **ОЗУ** и используется в основном для следующих целей:

- для хранения временных данных. Программист может разместить любые данные, не задумываясь, в какую ячейку памяти они будут размещены;

2.4 Стек

Стек относится к памяти типа ***LIFO*** (Last Input First Output, последним пришел - первым вышел), что означает, что последние загруженные данные будут выгружены в первую очередь. Здесь существует аналогия со стопкой тарелок: последнюю размещенную тарелку в стопке берем в первую очередь.

В МП 8086 каждый элемент стека занимает 2 байта, причем старший байт расположен в ОЗУ по старшему адресу, младший – по младшему.

Микропроцессор для обращения к данным в стеке использует два регистра: ***SS*** и ***SP***.

Сегментный регистр ***SS*** определяет начало блока памяти, отведенного под стек, а ***SP*** – смещение последний записи от начала сегмента.

Стек растет «вниз», т.е. при записи в стек данных значение регистра-счетчика ***SP*** автоматически уменьшается на 2, а при чтении данных – увеличивается на 2.

Для работы со стеком используются две основные команды: ***PUSH*** и ***POP***.

2.4 Стек

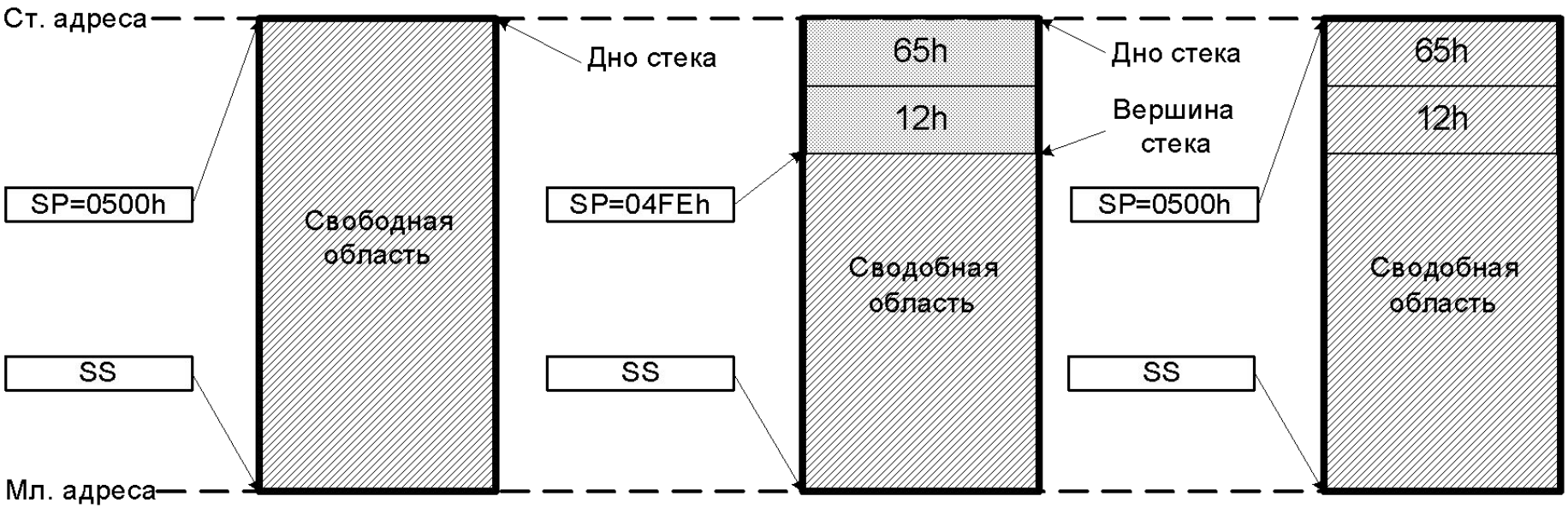


Рисунок 2.6 – Схема стека при выполнении команды ***PUSH*** и ***POP***
а) – до выполнения команд; б) – после выполнения команды ***PUSH***;
в) –после выполнения команды ***POP***

2.4 Стек

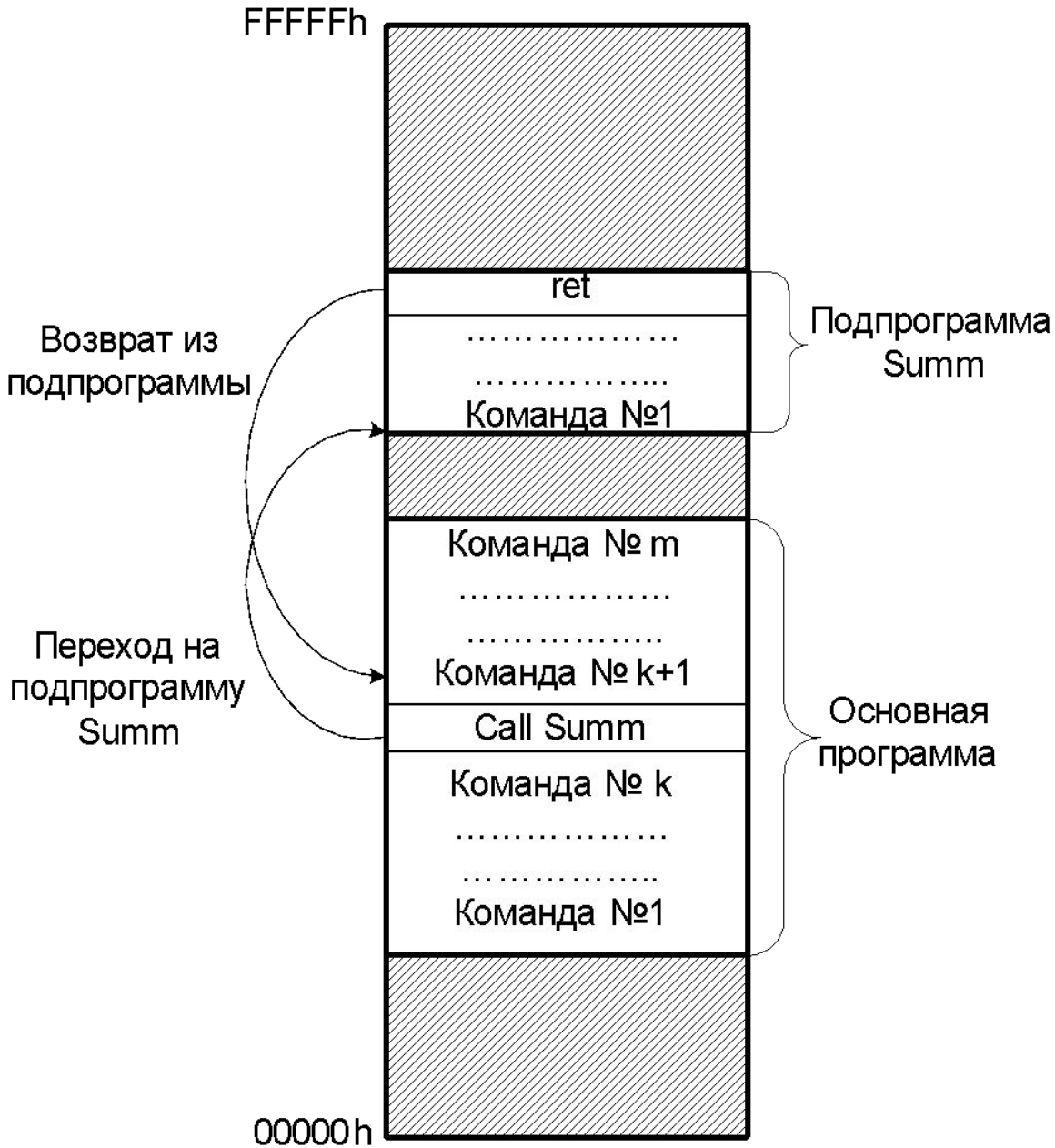
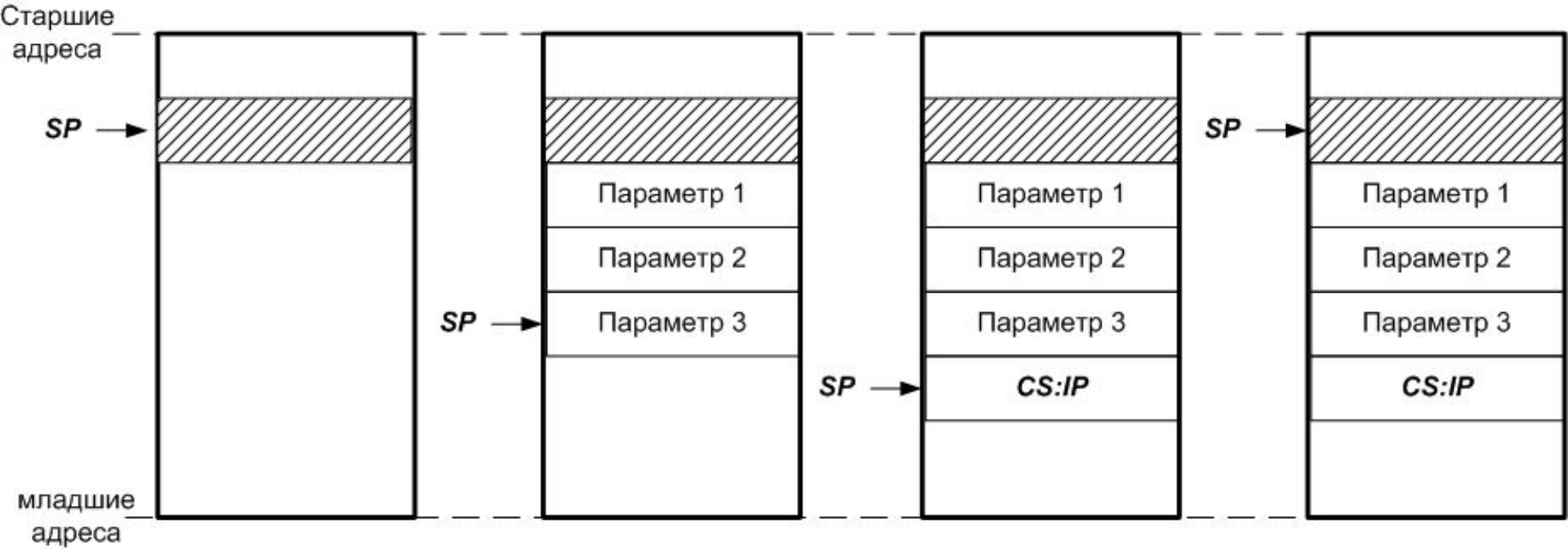


Рисунок 2.5 – Вызов подпрограммы


Старшие адреса
SP
Параметр 1
Параметр 2
Параметр 3
SP
Параметр 1
Параметр 2
Параметр 3
CS:IP
SP
Параметр 1
Параметр 2
Параметр 3
CS:IP
SP
младшие адреса

```
; add your code here
push 150            ; передача параметра 1
push 150            ; передача параметра 2
push 30             ; передача параметра 3
call myFunction   ; вызов подпрограммы MyFunction
HLT         ; halt!  ; остановка работы процессора
myFunction proc ;подпрограмма MyFunction
    push bp         ; сохраняем предыдущее значение
    mov bp,sp               ; запоминаем в регистре BP адрес последнего элемента стека
    xor ah,ah               ;обнуляем старший байт регистра AX (AH)
    mov al,[bp+4]  ; записываем в регистр AL значение  параметра 3
    add al,[bp+6]   ; складываем содержимое регистров AL и параметра2
; и сохраняем результат в AL
    jnc l1              ; если есть переполнение регистра AL,
    inc ah              ; то увеличиваем регистр AH на 1
l1:                     ;метка
    add al,[bp+8]   ; прибавляем к регистру AL содержимое параметра 1
    jnc l2:             ; если есть переполнение регистра AL,
    inc ah              ; то увеличиваем регистр AH на 1
                ; в итоге в регистре AX будет содержать сумму 3 параметров
l2:
    mov bl,3        ; помещаем в регистр BL значение 3
    div bl              ; делим содержимое регистра AX на BL
                ; в регистре AL - будет целая часть результата операции
                ;деления, в AH -остаток.
    pop bp          ; восстанавливаем содержимое регистра BP
    ret 6               ; вытаскиваем из стека адрес возврата и устанавливаем
;SP=SP+6
myFunction endp
```