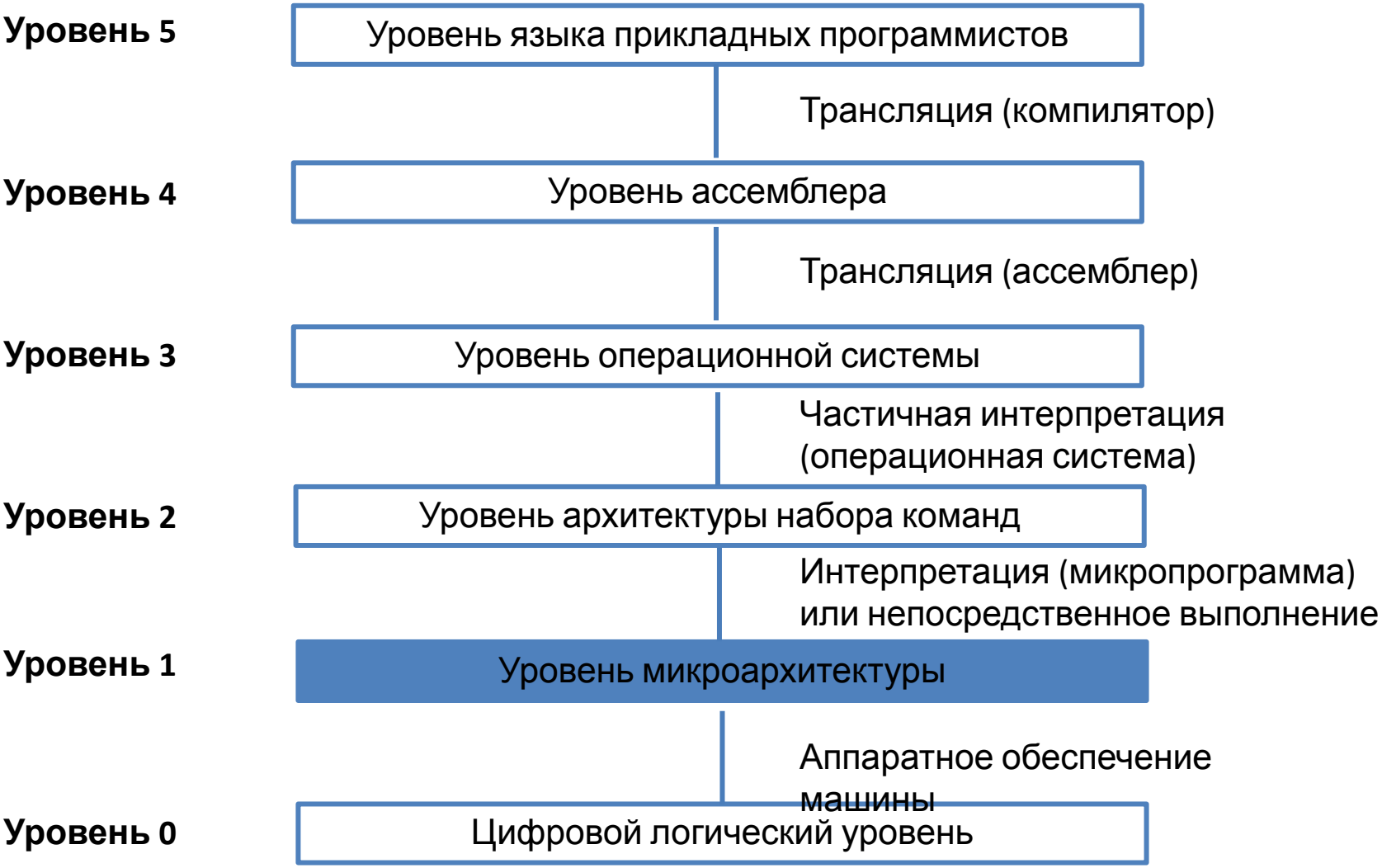
Уровень микроархитектуры

Современные многоуровневые машины



Шестиуровневый компьютер.

Уровень микроархитектуры

Задача – интерпретация команд уровня архитектуры команд.

Строение уровня микроархитектуры зависит от того каков уровень архитектуры команд, а также от стоимости и назначения компьютера:

- *RISC машины:* на уровне архитектуры команд обычно находятся простые команды которые выполняются за один цикл
- *Core i7:* на этом уровне имеются более сложные команды; выполнение одной такой команды занимает несколько циклов

  [Чтобы выполнить команду, нужно найти операнды в памяти, считать их и записать полученные результаты обратно в память]

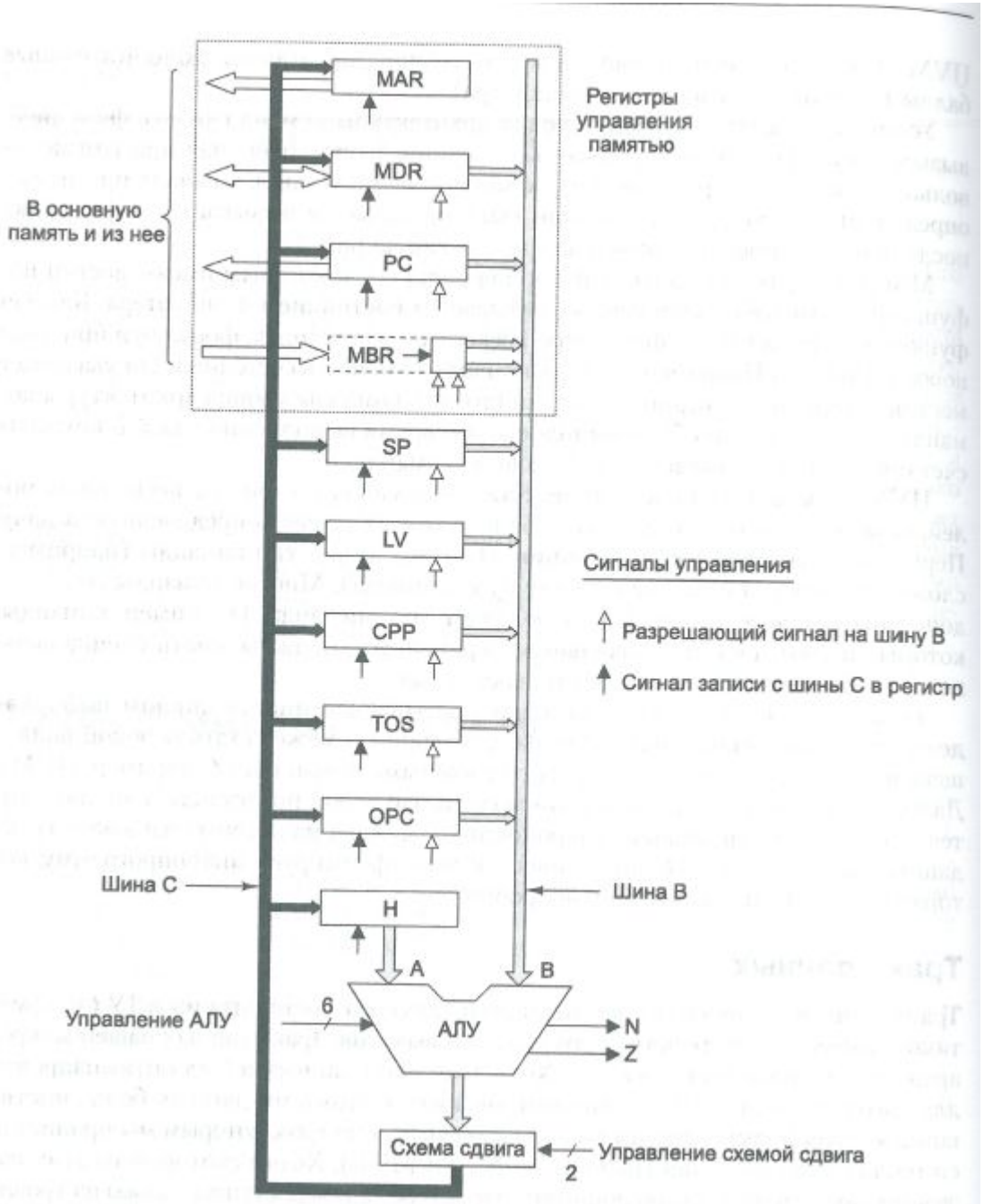
Пример микроархитектуры

Общих принципов разработки уровня микроархитектуры не существует!!!

Пример: рассмотрим подмножество виртуальной машины Java (содержит только целочисленные команды), которое назовем **IJVM** (Integer Java Virtual Machine - виртуальная машина Java для целых):

- МА содержит микропрограмму (МП) (в ПЗУ), которая д. вызывать, декодировать и выполнять команды IJVM.
- МП содержит набор переменных, к которым имеют доступ все функции (команды уровня АК). Этот набор переменных называется *состоянием компьютера*.
- Каждая команда IJVMсостоит из нескольких полей:1-ое поле – код операции, 2-ое (не обязательное) – определяет тип операнда.

Тракт данных



Тракт данных - это часть ЦП, состоящая из АЛУ, его входов и выходов

- 32-разрядных регистров
- шина B
- АЛУ: 6 линий управления–F_0, F_1, ENA, ENB, INVA, INC
- схема сдвига
- шина C
- 2 линии управления выходом АЛУ: SLL8,SRA1

Тракт данных

Некоторые комбинации сигналов АЛУ и соответствующие им функции

F_0	F_1	ENA	ENB	INVA	INC	Функция
0	1	1	0	0	0	A
0	1	0	1	0	0	B
0	1	1	0	1	0	$\overline{A}$
1	0	1	1	0	0	$\overline{B}$
1	1	1	1	0	0	A + B
1	1	1	1	0	1	A + B + 1
1	1	1	0	0	1	A + 1
1	1	0	1	0	1	B + 1
1	1	1	1	1	1	B – A
1	1	0	1	1	0	B – 1
1	1	1	0	1	1	–A
0	0	1	1	0	0	A И B
0	1	1	1	0	0	A ИЛИ B
0	1	0	0	0	0	0
0	1	0	0	0	1	1
0	1	0	0	1	0	–1

знак плюс (+) означает арифметический плюс,
знак минус (-) – арифметический минус,
-A означает дополнение A

Тракт данных

- Содержание большинства регистров передается на шину B.
- Выходной сигнал АЛУ управляет схемой сдвига и далее шиной C.
- Значение с шины C может записываться в один или несколько регистров одновременно.
- Шину A мы введем позже, а пока представим, что ее нет.

Функционирование АЛУ зависит от линий управления.

[Перечеркнутая стрелочка с цифрой 6 сверху указывает на наличие шести линии управления АЛУ:

- F_0 и F_1 служат для задания операции;
- ENA и ENB – для разрешения входных сигналов A и B соответственно;
- INVA – для инверсии левого входа ;
- INC – для переноса бита в младший разряд ≡ прибавление 1 к результату.]

Тракт данных

АЛУ содержит два входа для данных:

- левый вход (A) [с левым входом связан регистр временного хранения H]
- правый вход (B) [с правым входом связана шина B, на которую могут поступать значения с одного из девяти источников (серые стрелочки)].

В регистр H может поступать функция АЛУ, которая проходит через правый вход (из шины B) к выходу АЛУ. Одна из таких функций — сложение входных сигналов АЛУ: сигнал ENA отрицателен и левый вход получает значение 0. Если к значению шины B прибавить 0, это значение не изменится. Затем результат проходит через схему сдвига (также без изменений) и сохраняется в регистре H.

Линии управления SLL8 и SRA1

- используются независимо от остальных.
- служат для управления выходом АЛУ.

Линия SLL8 (*Shift Left Logical* — логический сдвиг влево) сдвигает число влево на 1 байт, заполняя 8 самых младших двоичных разрядов нулями.
Линия SRA1 (*Shift Right Arithmetic* — арифметический сдвиг вправо) сдвигает число вправо на 1 бит, оставляя самый старший двоичный без изменений

Тракт данных

Операции чтения и записи регистра могут выполняться за один цикл:

Пример: *значение SP поместить на шину B, закрыть левый вход АЛУ, установить сигнал INC и сохранить полученный результат в регистре SP увеличив т.о. его значение на 1*

Процессы чтения и записи происходят в разных частях цикла:

1. Когда в качестве правого входа АЛУ выбирается один из регистров, его значение помещается на шину B в начале цикла и хранится там на протяжении всего цикла.
2. Затем АЛУ выполняет свою работу результат которой через схему сдвига поступает на шину C.
3. Незадолго до конца цикла, когда значения выходных сигналов АЛУ и схемы сдвига стабилизируются, содержание шины C передается в один или несколько регистров. [Одним из этих регистров вполне может быть тот, с которого поступил сигнал на шину B].

Синхронизация тракта данных

Операции чтения и записи регистра могут выполняться за один цикл:

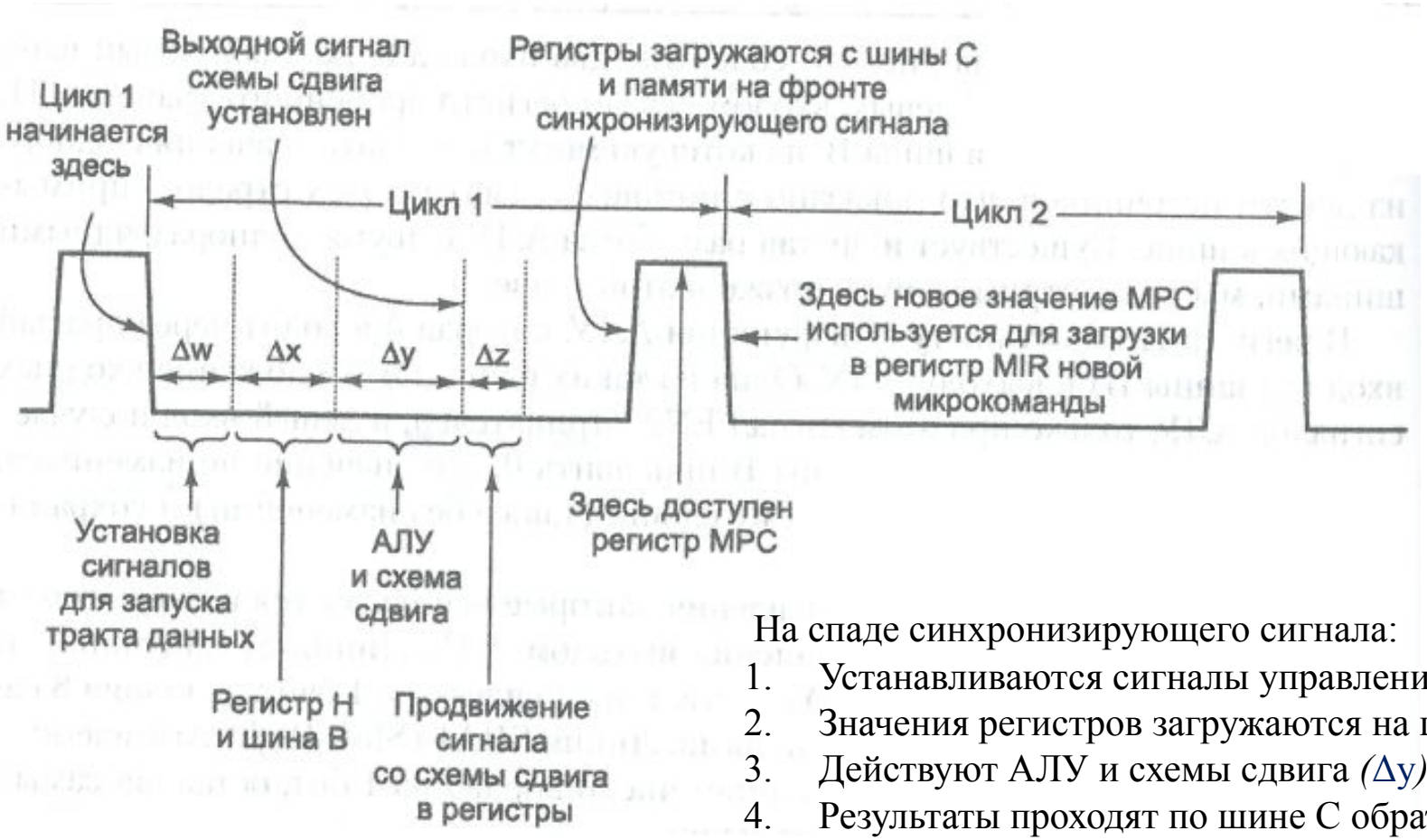
Пример: *значение SP поместить на шину B, закрыть левый вход АЛУ, установить сигнал INC и сохранить полученный результат в регистре SP увеличив т.о. его значение на 1*

Процессы чтения и записи происходят в разных частях цикла:

1. Когда в качестве правого входа АЛУ выбирается один из регистров, его значение помещается на шину B в начале цикла и хранится там на протяжении всего цикла.
2. Затем АЛУ выполняет свою работу результат которой через схему сдвига поступает на шину C.
3. Незадолго до конца цикла, когда значения выходных сигналов АЛУ и схемы сдвига стабилизируются, содержание шины C передается в один или несколько регистров. [Одним из этих регистров вполне может быть тот, с которого поступил сигнал на шину B].

Синхронизация тракта данных

Временная диаграмма цикла тракта данных



На спаде синхронизирующего сигнала:

1. Устанавливаются сигналы управления *(Δw)*.
2. Значения регистров загружаются на шину B *(Δx)*.
3. Действуют АЛУ и схемы сдвига *(Δy)*.
4. Результаты проходят по шине C обратно к регистрам (Δz).

На фронте следующего цикла результаты сохраняются в регистрах

Синхронизация тракта данных

- В начале каждого цикла генерируется короткий импульс
- на спаде импульса устанавливаются биты, которые будут запускать все вентили [Δw]
- выбирается регистр, и его значение передается на шину B [Δx]
- АЛУ и схема сдвига начинают оперировать поступившими к ним данными. После промежутка Δy выходные сигналы АЛУ и схемы сдвига стабилизируются [Δy]
- результаты проходят по шине C к регистрам, куда они загружаются на фронте следующего импульса [Δz]

  [Загрузка должна запускаться фронтом сигнала и осуществляться мгновенно, так что даже в случае изменений каких-либо входных регистров изменения в шине C будут происходить только после полной загрузки регистров. На фронте импульса регистр, запускающий шину B, приостанавливает свою работу и ждет следующего цикла.]

Функционирование памяти

Взаимодействие с памятью:

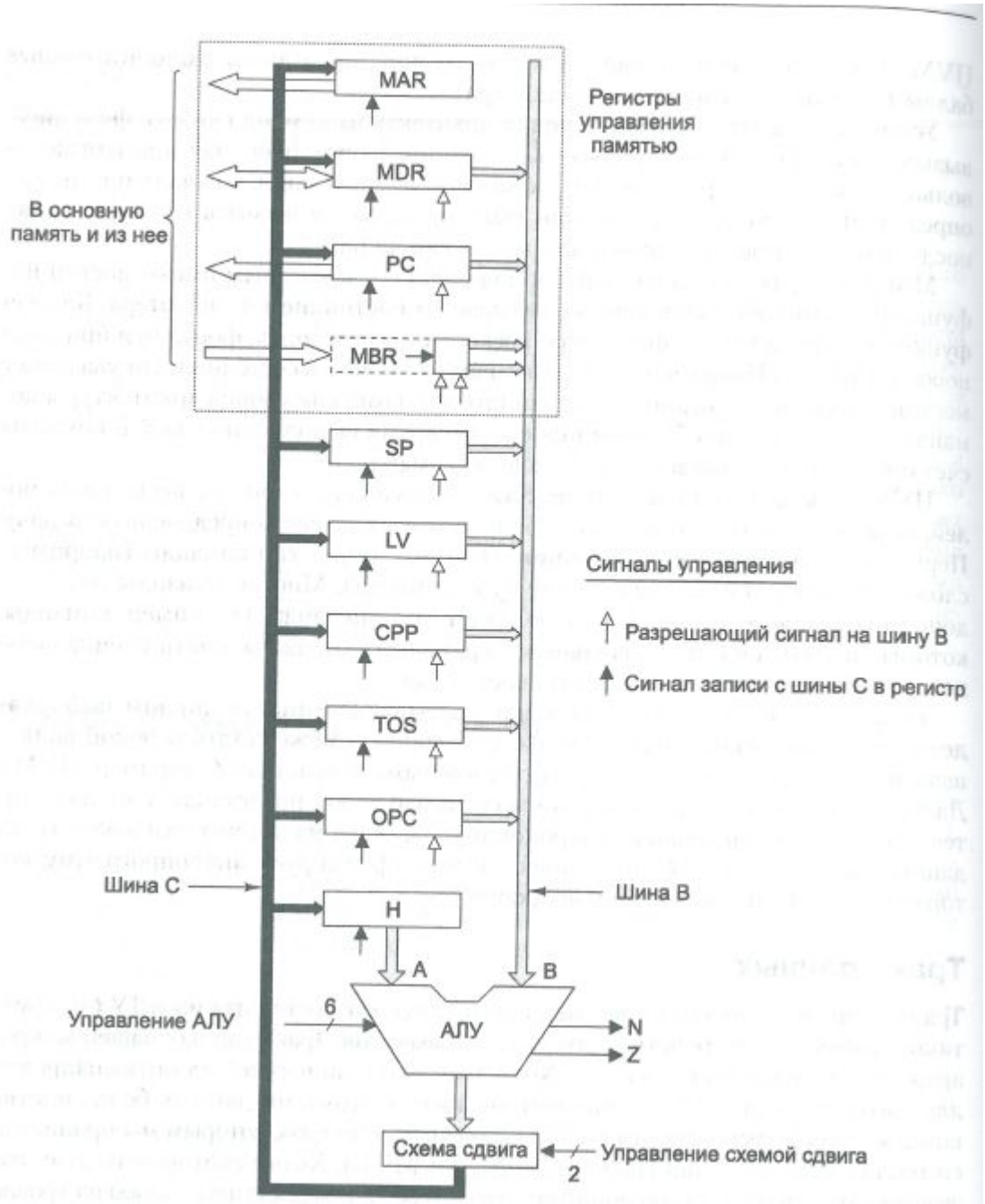
- через *порт с пословной адресацией* (32-разрядный)
- через *порт с побайтовой адресацией* (8-разрядный).

Порт с пословной адресацией управляется двумя регистрами:

✔ **MAR** (Memory Address Register - **адресный регистр памяти)**

✔ **MDR** (Memory Data Register - **информационный регистр памяти**)

Порт с побайтовой адресацией управляется регистром PC, который записывает 1 байт в 8 младших битов регистра **MBR** (Memory Buffer Register — **буферный регистр памяти)**



Регистры запускаются одним из **сигналов управления.**

Белая стрелка – СУ, который разрешает передавать выходной сигнал регистра на шину B. [регистр MAR не связан с шиной B; регистр H единственно возможный левый вход АЛУ и поэтому всегда разрешен].

Черная стрелка – СУ, который записывает регистр с шины C. [регистр MBR не может загружаться с шины C, у него нет записывающего сигнала управления]

Чтобы инициировать процесс считывания из памяти или записи в память, нужно загрузить соответствующие регистры памяти, а затем передать памяти сигнал чтения или записи

Функционирование памяти

Регистр MAR содержит адреса *слов,* таким образом, значения 0, 1, 2 и т. д. указывают на последовательные слова.

Регистр PC содержит адреса *байтов,* таким образом, значения 0, 1, 2 и т. д. указывают на последовательные байты.

[Если значение 2 поместить в регистр PC и начать процесс чтения, то из памяти считается байт 2, который затем будет записан в 8 младших битов регистра MBR. Если значение 2 поместить в регистр MAR и начать процесс чтения, то из памяти считаются байты с 8 по 11 (то есть слово 2), которые затем будут записаны в регистр MDR]

- Регистры MAR и MDR используются для чтения и записи слов данных на уровне архитектуры команд.
- Регистры PC и MBR — для считывания программы уровня архитектуры команд, которая состоит из потока байтов.
- Во всех остальных регистрах, содержащих адреса, применяется принцип пословной адресации, как и в MAR.

Функционирование памяти

Когда значение регистра MAR помещается на адресную шину, 32 бита этого значения не попадают точно на 32 адресные линии (с 0 по 31):

- бит 0 соединяется с адресной линией 2, бит один — с адресной линией 3 и т. д.
- два старших бита не учитываются, поскольку они нужны только для адресов свыше 2^{32}, (такие адреса недопустимы в нашей машине на 4 Гбайт)
- когда значение MAR равно 1, на шину помещается адрес 4; когда значение MAR равно 2, на шину помещается адрес 8 и т. д.

Функционирование памяти

Данные, считанные из памяти через 8-разрядный порт, сохраняются в 8-разрядном регистре MBR. Этот регистр может быть скопирован на шину B двумя способами: со знаком и без знака.

Когда требуется значение без знака, 32-разрядное слово, помещаемое на шину B, содержит значение MBR в младших 8-ми битах и нули в остальных 24-х битах. Значения без знака нужны для индексирования таблиц или получения целого 16-разрядного числа из двух последовательных байтов (без знака) в потоке команд.

Функционирование памяти

- Другой способ превращения 8-разрядного регистра MBR в 32-разрядное слово - считать его значением со знаком от -128 до +127 включительно и использовать это значение для порождения 32-разрядного слова с тем же самым численным значением. Это преобразование делается путем дублирования знакового (самого левого) бита регистра MBR в верхние 24 битовые позиции шины B. Такой процесс называется *расширением по знаку*, или знаковым расширением.
- Если выбран данный параметр, то либо все старшие 24 бита примут значение 0, либо все они примут значение 1 в зависимости от того, каков самый левый бит регистра MBR: 0 или 1.
- В какое именно 32-разрядное значение (со знаком или без знака) превратится 8-разрядное значение регистра MBR, определяется тем, какой из двух сигналов управления (две белые стрелки под регистром MBR) установлен. Пунктирный прямоугольник обозначает способность 8-разрядного регистра MBR действовать в качестве источника 32-разрядных слов для шины B.

Функционирование памяти

- Другой способ превращения 8-разрядного регистра MBR в 32-разрядное слово - считать его значением со знаком от -128 до +127 включительно и использовать это значение для порождения 32-разрядного слова с тем же самым численным значением. Это преобразование делается путем дублирования знакового (самого левого) бита регистра MBR в верхние 24 битовые позиции шины B. Такой процесс называется *расширением по знаку*, или знаковым расширением.
- Если выбран данный параметр, то либо все старшие 24 бита примут значение 0, либо все они примут значение 1 в зависимости от того, каков самый левый бит регистра MBR: 0 или 1.
- В какое именно 32-разрядное значение (со знаком или без знака) превратится 8-разрядное значение регистра MBR, определяется тем, какой из двух сигналов управления (две белые стрелки под регистром MBR) установлен. Пунктирный прямоугольник обозначает способность 8-разрядного регистра MBR действовать в качестве источника 32-разрядных слов для шины B.

Микрокоманды

Для управления ТД необходимо 29 сигналов

- 9 сигналов для записи данных с шины C в регистры;
- 9 сигналов для разрешения передачи регистров на шину B и в АЛУ;
- 8 сигналов для управления АЛУ и схемой сдвига;
- 2 сигнала, которые указывают, что нужно осуществить чтение или запись через регистры MAR/MDR;
- 1 сигнал, который указывает, что нужно осуществить вызов из памяти через регистры PC/MBR.

Микрокоманды

Значения этих 29 сигналов управления определяют операции для одного цикла ТД.

Цикл состоит из передачи значений регистров на шину B прохождения этих сигналов через АЛУ и схему сдвига, передачи полученных результатов на шину C и записи их в нужный регистр(ы).

Если установлен сигнал считывания данных, то в конце цикла после загрузки регистра MAR начинает работать память. Данные из памяти помещаются в MBR или MDR в конце следующего цикла, а использоваться эти данные могут в цикле который идет после него.

[если считывание из памяти через любой из портов начинается в конце цикла k, то полученные данные не смогут использоваться в цикле k + 1 (только в цикле k + 2 и позже)].

Микрокоманды

формат микрокоманды

6 групп, содержащие 36 сигналов:

- Addr – адрес следующей потенциальной микрокоманды;
- JAM – определение того, как выбирается следующая микрокоманда;
- ALU – функции АЛУ и схемы сдвига;
- C – выбор регистров, которые записываются с шины C;
- Mem – функции памяти.

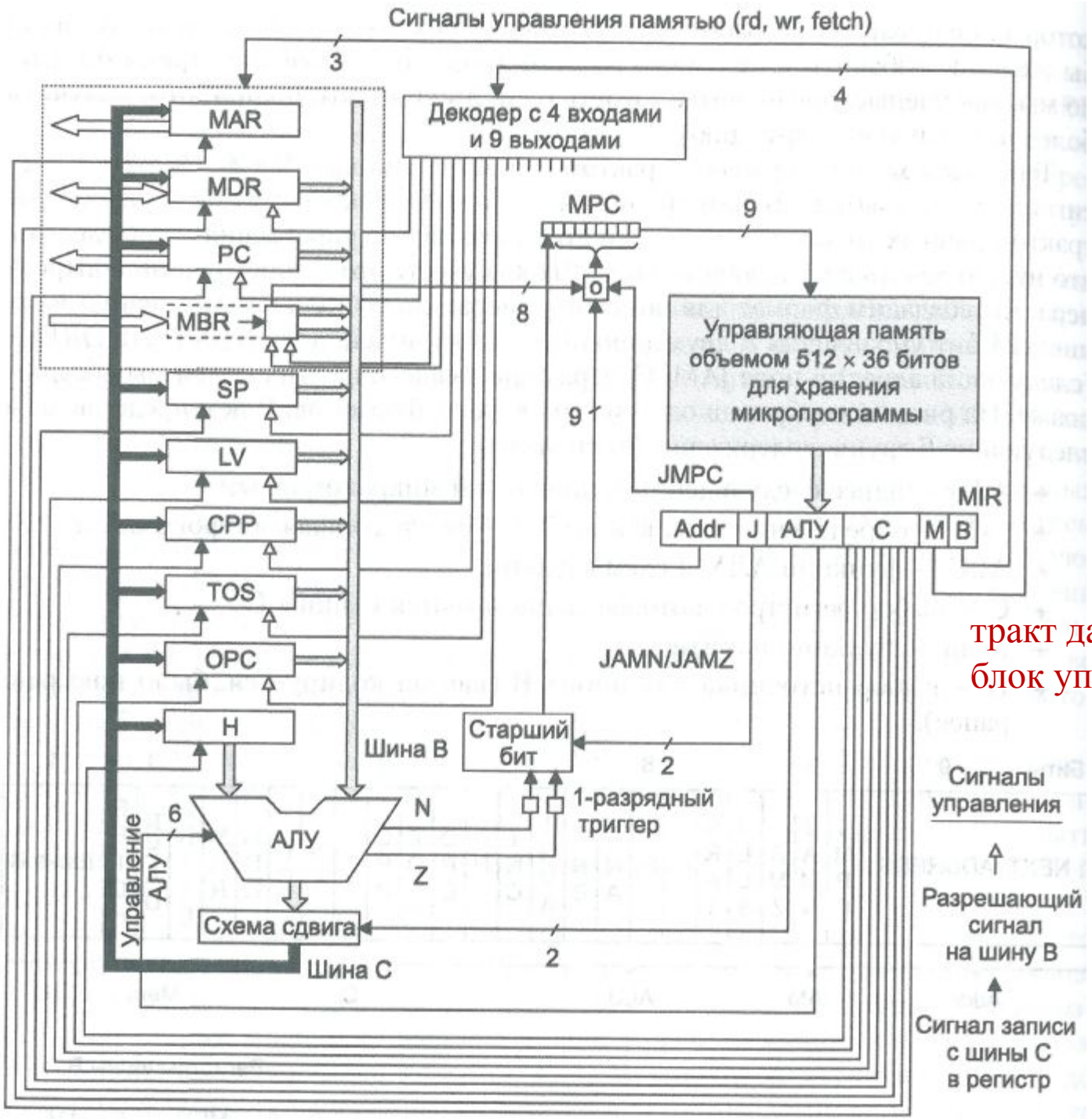
Управление микрокомандами

Контроллер последовательности отвечает за последовательность операций, необходимых для выполнения одной команды.

В каждом цикле должен выдавать следующую информацию:

- состояние каждого сигнала управления в системе;
- адрес микрокоманды, которая будет выполняться следующей.

Полная диаграмма микроархитектуры Mic-1



тракт данных *(слева)*
блок управления *(справа)*

Управление микрокомандами

Блок управления

Самой большой и самой важной частью блока управления является *управляющая память*

Будем рассматривать ее как память, в которой хранится вся микропрограмма, хотя иногда микропрограмма реализуется в виде набора логических вентилей

[не путать с основной памятью, доступ к которой осуществляется через регистры MBR и MDR].

УП представляет собой память, в которой вместо обычных команд хранятся микрокоманды.

[В нашем примере она содержит 512 слов; каждое слово состоит из одной 36-разрядной микрокоманды, формат которой показан на слайде 22]

Отличие УП от основной памяти: микропрограммы не упорядочены в управляющей памяти [команды, хранящиеся в основной памяти, всегда выполняются в порядке адресов (за исключением переходов)]

Управление микрокомандами

Блок управления

Управляющая память функционально представляет собой ПЗУ

- нужны собственные адресный регистр и регистр данных.
- не требуются сигналы чтения и записи, поскольку процесс считывания происходит постоянно.

Адресный регистр управляющей памяти назовем **MPC** (Microprogram Counter — **счетчик микропрограмм).**

Регистр данных назовем **MIR** (Microinstruction Register — **регистр микрокоманд)** – содержит текущую микрокоманду, биты которой запускают сигналы управления, влияющие на ра боту тракта данных.

Управление микрокомандами

Блок управления

Управляющая память функционально представляет собой ПЗУ

- нужны собственные адресный регистр и регистр данных.
- не требуются сигналы чтения и записи, поскольку процесс считывания происходит постоянно.

Адресный регистр управляющей памяти назовем **MPC** (Microprogram Counter — **счетчик микропрограмм).**

Регистр данных назовем **MIR** (Microinstruction Register — **регистр микрокоманд)** – содержит текущую микрокоманду, биты которой запускают сигналы управления, влияющие на ра боту тракта данных.

Управление микрокомандами

Блок управления

Регистр MIR, содержит те же шесть групп сигналов, которые показаны на слайде 22.

- Группы Addr и J (то же, что JAM) контролируют выбор следующей микрокоманды.
- Группа ALU содержит 8 бит, которые позволяют выбрать функцию АЛУ и запустить схему сдвига. Биты C загружают отдельные регистры с шины C.
- Сигналы M управляют работой памяти.
- последние 4 бита запускают декодер, который определяет, значение какого регистра будет передано на шину B.

Управление микрокомандами

Схема работы

Δw – в начале каждого цикла (фронт синхр-го сигнала) в регистр MIR загружается слово из управляющей памяти MPC

Δx – значение определенного регистра помещается на шину B, а АЛУ узнает, какую операцию нужно выполнять

После периода Δw + Δx входные сигналы АЛУ стабилизируются.

После периода Δy стабилизируются сигналы N и Z АЛУ, а также выходной сигнал схемы сдвига. Затем значения N и Z сохраняются в двух 1-разрядных триггерах. [Эти биты, как и биты всех регистров, которые загружаются с шины C и памяти, сохраняются на фронте синхронизирующего сигнала, ближе к концу цикла тракта данных.]

Δy– работа АЛУ и схемы сдвига [выходной сигнал АЛУ не сохраняется, а просто передается в схему сдвига].

Управление микрокомандами

Схема работы

После следующего интервала, Δz, выходной сигнал схемы сдвига, пройдя через шину С, достигает регистров. Регистры загружаются в конце цикла на фронте синхронизирующего сигнала

Δz – происходит загрузка регистров и триггеров N и Z.
Подцикл завершается сразу после окончания фронта, когда все значения сохранены, результаты предыдущих операций памяти доступны, регистр МРС загружен.

Управление микрокомандами

Вычисление адреса следующей команды

Вычисление адреса следующей микрокоманды начинается после загрузки регистра MIR. Сначала в регистр MPC копируется 9-разрядное поле NEXTADDRESS. Пока происходит копирование, проверяется поле JAM (см. слайд 24 блок «Старший бит»)

Если поле JAM содержит значение 000, то ничего больше делать не нужно, и когда копирование поля NEXT ADDRESS завершится, регистр MPC укажет на следующую микрокоманду.

F = ((JAMZ И Z) ИЛИ (JAMN И N)) ИЛИ NEXTADDRESS [8]

Если один или несколько битов в поле JAM равны 1

Если бит JAMZ равен 1, то триггер Z соединяется через схему ИЛИ со старшим битом регистра MPC. Если оба бита равны 1, они оба соединяются через схему ИЛИ с тем же битом.

Управление микрокомандами

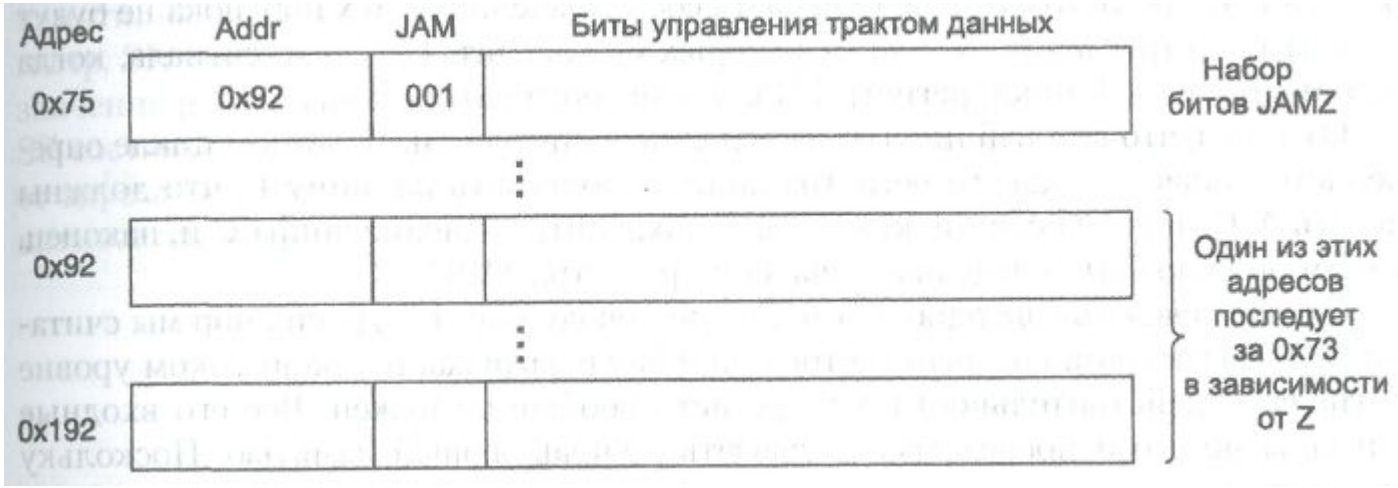
Вычисление адреса следующей команды

Назначение триггеров N и Z:

после фронта сигнала (и вплоть до спада) шина B больше не запускается, поэтому выходные сигналы АЛУ уже не могут считаться правильными. Сохранение флагов состояния АЛУ в регистрах N и Z делает правильные значения установившимися и доступными для вычисления регистра MPC, независимо от того, что происходит вокруг АЛУ.

Управление микрокомандами

Вычисление адреса следующей команды



Если все биты JAM равны 0, то адрес следующей команды – 9-разрядное число в поле NEXT ADDRESS.

Если бит JAMN или TAMZ равен 1, то существует два потенциально возможных адреса следующей микрокоманды: адрес NEXT_ADDRESS и адрес NEXT_ADDRESS, соединенный операцией ИЛИ со значением 0x100 (предполагается, что NEXT ADDRESS < 0xFF)

Текущая микрокоманда с адресом 0x73 содержит ноле NEXT ADDRESS - 0x92, бит JAMZ = 1 Следующий адрес микрокоман ды зависит от значения бита Z, сохраненного при предыдущей операции АЛУ. Если бит Z равен 0, то следующая микрокоманда имеет адрес 0x92. Если бит Z равен 1, то следующая микрокоманда имеет адрес 0x192.

Управление микрокомандами

Вычисление адреса следующей команды

Третий бит в поле JAM – JMPC. Если он установлен, то 8 бит регистра MBR поразрядно связываются операцией ИЛИ с 8 младшими битами поля NEXT_ ADDRESS текущей микрокоманды. Результат отправляется в регистр MPC.

На слайде 24 меткой «O» обозначена схема, которая выполняет операцию ИЛИ над MBR и NEXT_ADDRESS, если бит JMPC равен 1, и просто отправляет NEXT_ ADDRESS в регистр MPC, если бит JMPC равен 0.

Если бит JMPC равен 1, то младшие 8 бит поля NEXT ADDRESS равны 0. Старший бит может быть 0 или 1, поэтому значение поля NEXT_ADDRESS обычно 0x000 или 0x100.

Управление микрокомандами

Вычисление адреса следующей команды

Возможность выполнения операции ИЛИ над MBR и NEXT_ADDRESS и сохранения результата в регистре MPC позволяет реализовывать межуровневые переходы.

Биты, находящиеся в регистре MBR, позволяют задать любой адрес из 256 возможных. Регистр MBR содержит код операции, поэтому использование бита JMPC приведет к единственно возможному выбору следующей микрокоманды =>

Этот метод позволяет осуществлять быстрый переход к функции, соответствующей вызванному коду операции.

Управление микрокомандами

Во время подцикла 1, который инициируется спадом сигнала, адрес, находящийся в регистре MPC, загружается в регистр MIR.
Во время подцикла 2 регистр MIR устанавливает сигналы, и на шину B загружается выбранный регистр.
Во время подцикла 3 работают АЛУ и схема сдвига.
Во время подцикла 4 стабилизируются значения шины C, шин памяти и АЛУ.
На фронте сигнала загружаются регистры из шины C, загружаются триггеры N и Z, а регистры MBR и MDR получают результаты из памяти, начавшей функционировать в конце предыдущего цикла (если эти результаты вообще имеются).
Как только регистр MBR получает свое значение, загружается регистр MPC. Это происходит где-то в середине отрезка между фронтом и спадом, но уже после загрузки регистров MBR и MDR.
Регистр MPC может загружаться либо уровнем (но не фронтом) сигнала, либо через фиксированный отрезок времени после фронта. Все это означает, что регистр MPC не получает своего значения до тех пор, пока не будут готовы регистры MBR, N и Z, от которых он зависит.
На спаде сигнала, когда начинается новый цикл, регистр MPC может обращаться к памяти.

Пример архитектуры набора команд — IJVM

Уровень архитектуры набора команд (ISA), которые должна интерпретировать микропрограмма машины IJVM.

Стек

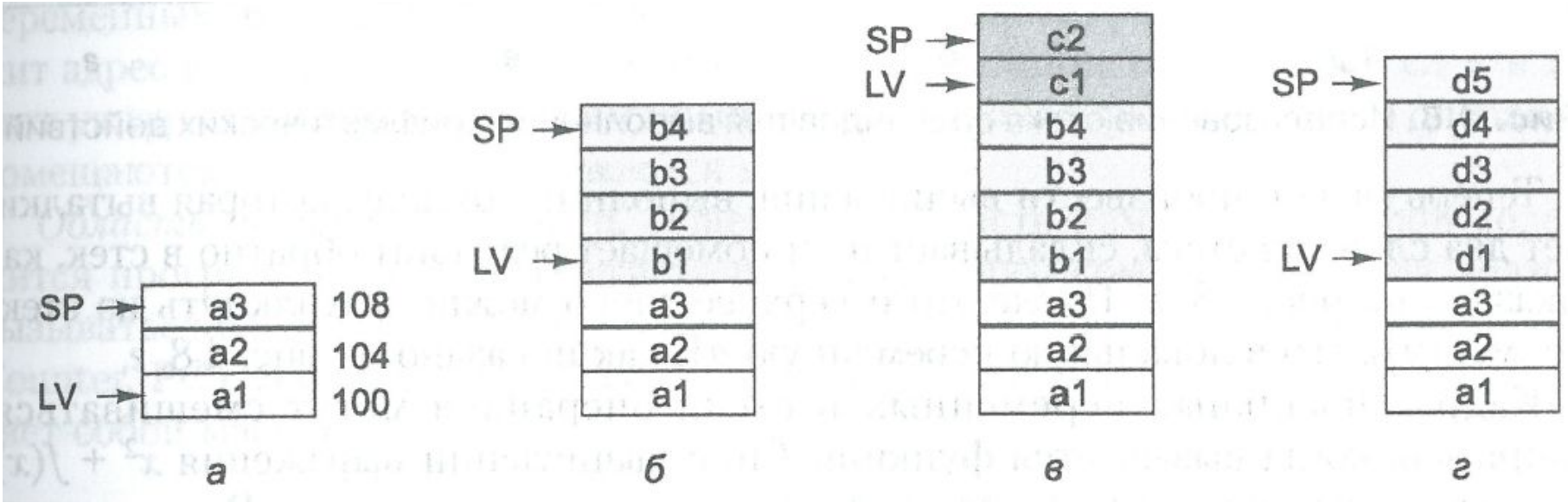
Где должны храниться локальные переменные?

Нельзя связать каждую переменную с абсолютным адресом в памяти, например для рекурсивных процедур

Для переменных резервируется особая область памяти, которая называется **стеком** и в которой отдельные переменные не получают абсолютных адресов.

LV указывает на базовый адрес кадра локальных переменных текущей процедуры,
SP – на верхнее слово этого кадра.

Стек

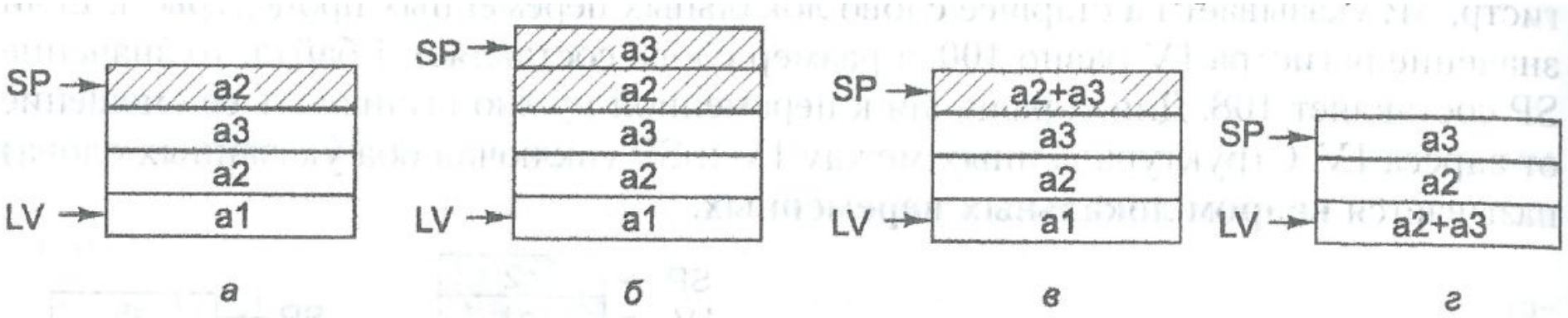


Стек для хранения локальных переменных:

- во время вызова процедуры A (а);
- после того как процедура A вызывает процедуру B (б);
- после того как процедура B вызывает процедуру C (в);
- после того как процедуры C и B завершаются, а процедура A вызывает процедуру D (г)

Структура данных между LV и SP (включая оба указанных слова) называется **кадром локальных переменных.**

Стек операндов



Использование стека операндов для выполнения арифметических действий

✔ Кадры локальных переменных и стеки операндов могут смешиваться.

[Например, когда вызывается функция *f* при вычислении выражения $x^2 + f(x)$, часть этого выражения (x^2) может находиться в стеке операндов.]

✔ Все машины используют стек для хранения локальных переменных, но не все используют его для хранения операндов.

Модель памяти IJVM

- Память рассматриваем либо как массив из 4 294 967 296 байт (4 Гбайт), либо как массив из 1 073 741 824 слов, каждое из которых содержит 4 байта.

- Виртуальная машина Java не выполняет обращений к памяти, видимых на уровне команд, но имеет несколько неявных адресов, которые составляют основу указателя. IJVM-команды могут обращаться к памяти только через эти указатели.

Модель памяти IJVM

- Определены следующие области памяти:

Набор констант

- недоступна для записи из IJVM-программы, состоит из констант, строк и указателей на другие области памяти, на которые можно делать ссылку
- загружается в момент загрузки программы в память и после этого не меняется
- имеется скрытый регистр CPP (Constant Pool Pointer — указатель набора констант), который содержит адрес первого слова набора констант

Модель памяти IJVM

- Определены следующие области памяти:

Кадр локальных переменных

- предназначена для хранения переменных во время выполнения процедуры.
- в начале располагаются параметры (или аргументы) вызванной процедуры.
- не включает в себя стек операндов, который размещается отдельно [стек операндов расположили над кадром локальных переменных]
- существует скрытый регистр, который содержит адрес первой переменной кадра, назовем этот регистр LV (Local Variable - локальная переменная).

Модель памяти IJVM

- Определены следующие области памяти:

Стек операндов.

- не должен быть больше определенного размера, который заранее вычисляется компилятором Java.
- пространство стека операндов располагается прямо над кадром локальных переменных
- существует виртуальный регистр, который содержит адрес верхнего слова стека. Этот регистр меняется во время выполнения процедуры, поскольку операнды помещаются в стек и выталкиваются из него.

Модель памяти IJVM

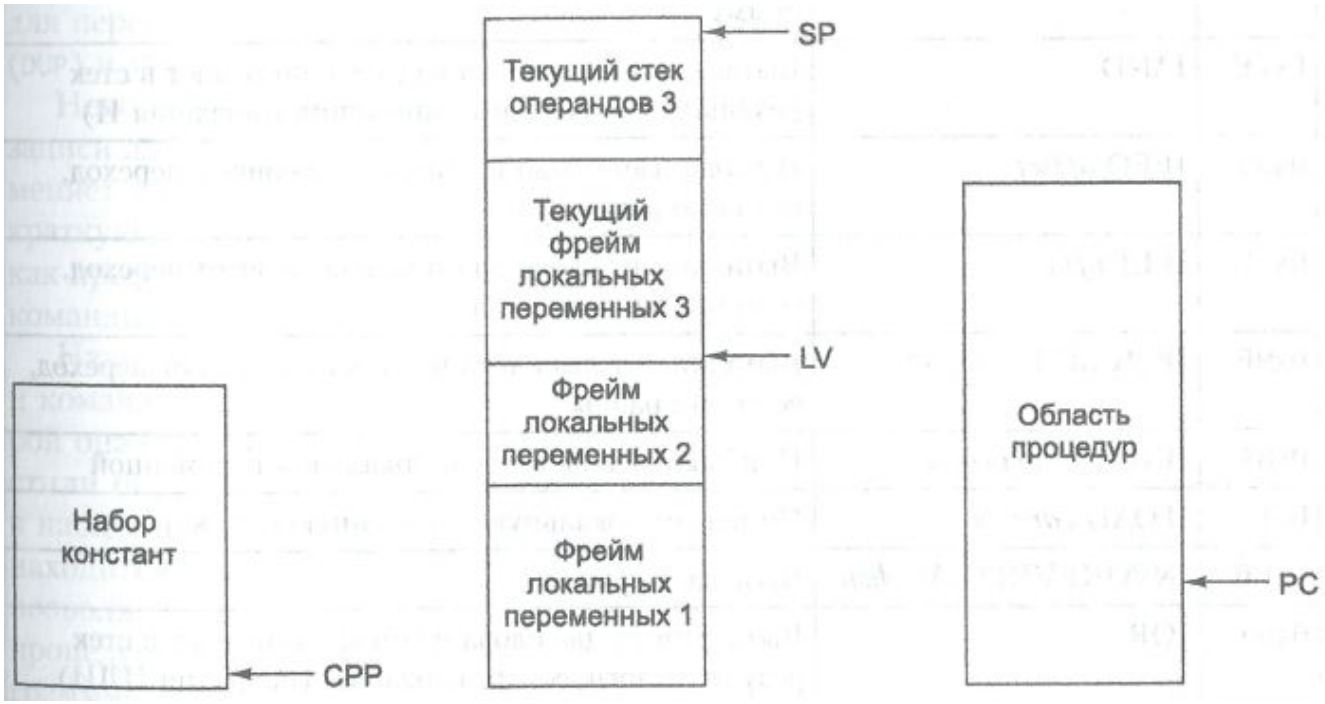
- Определены следующие области памяти:

Область процедур

- область памяти, в которой содержится программа.
- скрытый регистр содержит адрес команды, которая должна вызываться следующей. Этот указатель называется *счетчиком команд* (Program Counter, PC).
- область процедур представляет собой массив байтов.

Модель памяти IJVM

Области памяти IJVM



- ✔ Регистры CPP, LV и SP указывают на *слова,* а не на *байты,* и смещения происходят на определенное число слов. [Например, значения LV, LV + 1 и LV + 2 указывают на первые три слова кадра локальных переменных, а LV, LV + 4 и LV + 8 - на слова, расположенные на расстоянии четырех слов (16 байт) друг от друга.]
- ✔ Регистр PC содержит адреса байтов, и его изменение означает увеличение на определенное количество байтов, а не слов.

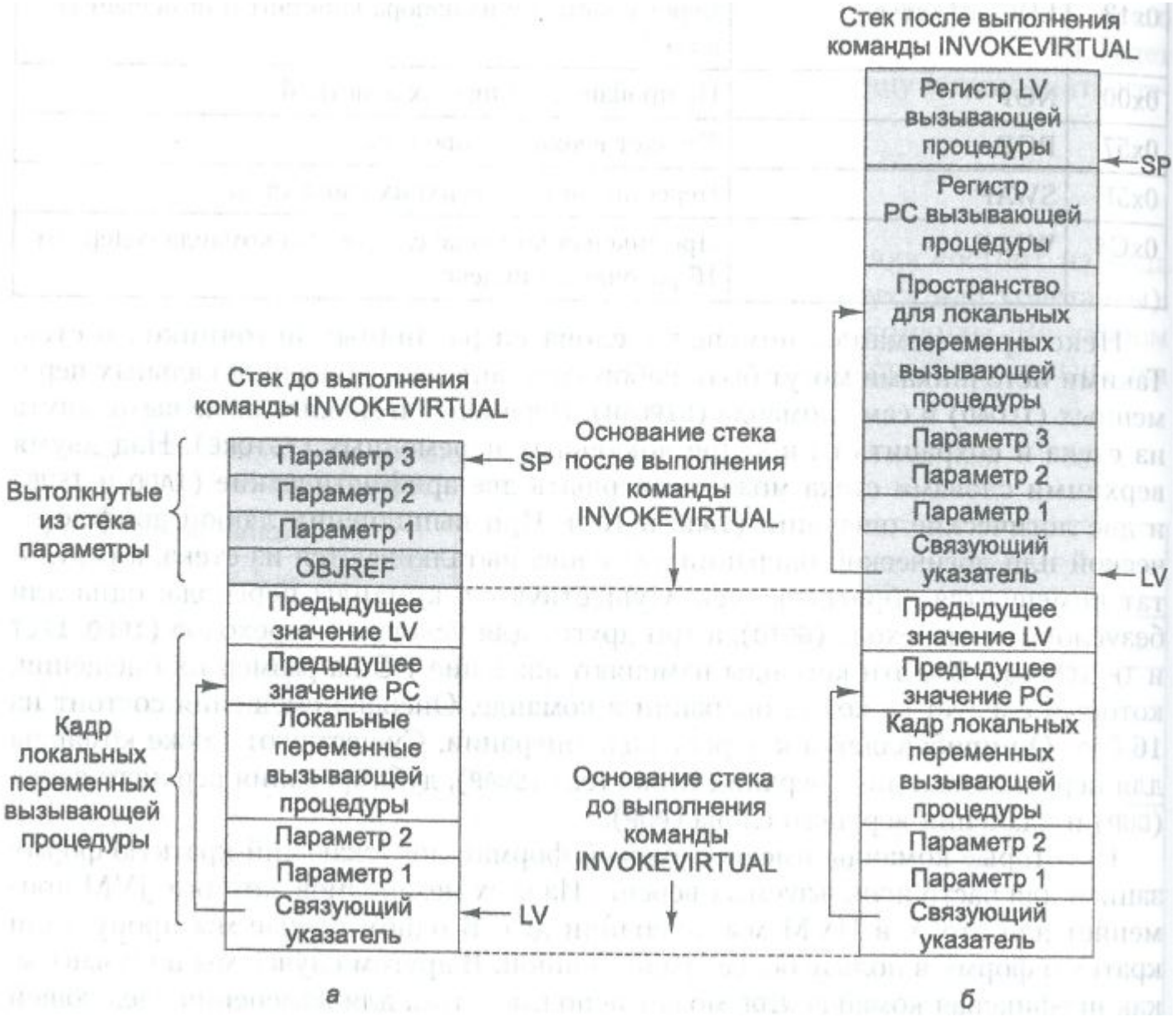
Набор IJVM-команд

Шестнадца-теричный код команды	Мнемоника на языке ассемблера	Назначение команды
0x10	BIPUSH byte	Помещает байт в стек
0x59	DUP	Копирует верхнее слово стека и помещает его в стек
0xA7	GOTO offset	Безусловный переход
0x60	IADD	Выталкивает два слова из стека; помещает в стек их сумму
0x7E	IAND	Выталкивает два слова из стека; помещает в стек результат логического умножения (операция И)
0x99	IFEQ offset	Выталкивает слово из стека и совершает переход, если оно равно нулю
0x9B	IFLT offset	Выталкивает слово из стека и совершает переход, если оно меньше нуля
0x9F	IFJCMPEQ offset	Выталкивает два слова из стека; совершает переход, если они равны
0x84	IINC varnum const	Прибавляет константу к локальной переменной
0x15	ILOAD varnum	Помещает локальную переменную в стек

Набор IJVM-команд

Шестнадца-теричный код команды	Мнемоника на языке ассемблера	Назначение команды
0xB6	INVOKEVIRTUAL *disp*	Вызывает процедуру
0x80	IOR	Выталкивает два слова из стека; помещает в стек результат логического сложения (операция ИЛИ)
0xAC	IRETURN	Выдает результат выполнения процедуры (целое число)
0x36	ISTORE *varnum*	Выталкивает слово из стека и запоминает его в кадре локальных переменных
0x64	ISUB	Выталкивает два слова из стека; помещает в стек их разность
0x13	LDC_W *index*	Берет константу из набора констант и помещает ее в стек
0x00	NOP	Не производит никаких действий
0x57	POP	Удаляет верхнее слово стека
0x5F	SWAP	Переставляет два верхних слова стека
0xC4	WIDE	Префиксная команда; следующая команда содержит 16-

Механизм вызова процедуры



Механизм вызова процедуры

Вызывающая программа помещает в стек сначала указатель на вызываемый объект (OBJREF), затем параметры процедуры (Параметр 1, Параметр 2 и Параметр 3)

Выполняется команда INVOKEVIRTUAL

INVOKEVIRTUAL включает в себя смещение, которое определяет позицию в наборе констант. В этой позиции находится начальный адрес вызываемой процедуры, которая хранится в области процедур. Первые 4 байта в области процедур содержат специальные данные. Первые 2 байта представляют собой целое 16-разрядное число, указывающее на количество параметров данной процедуры (сами параметры были ранее помещены в стек). В данном случае указатель OBJREF считается параметром - параметром 0. Это 16-разрядное целое число вместе со значением SP дает адрес OBJREF. Отметим, что регистр LV указывает на OBJREF, а не на первый реальный параметр. Выбор того, на что указывает LV, в какой-то степени произволен.

Механизм вызова процедуры

Следующие 2 байта в области процедур представляют еще одно 16-разрядное целое число, задающее размер области локальных переменных для вызываемой процедуры. Для данной процедуры предоставляется новый стек, который размещается прямо над кадром локальных переменных, для этого и нужно это число. Наконец, пятый байт в области процедур содержит код первой операции, которую нужно выполнить

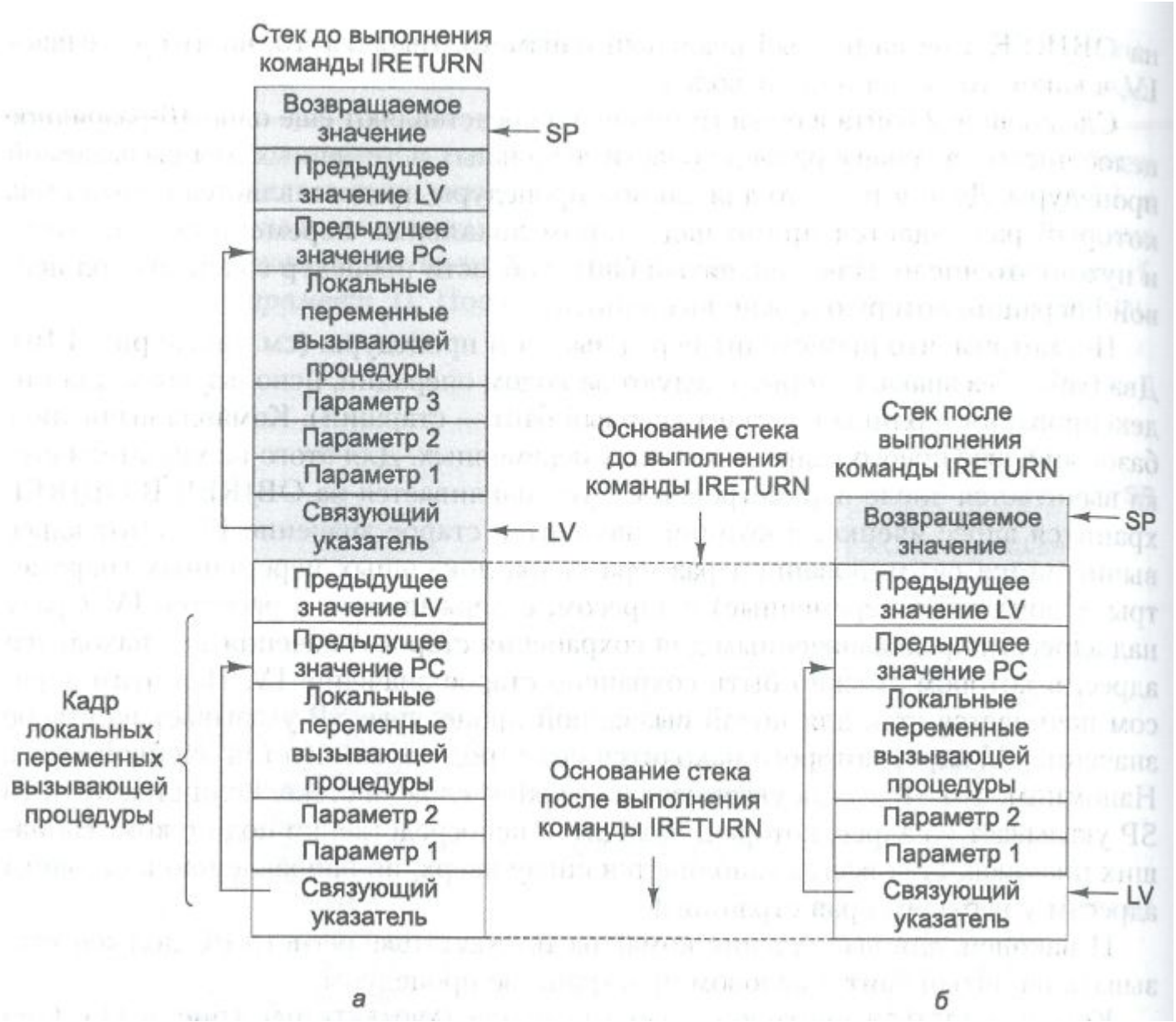
Механизм вызова процедуры

Перед вызовом процедуры

Два байта без знака, которые следуют за кодом операции, используются для индексирования таблицы констант (первый байт – старший). Команда вычисляет базовый адрес нового кадра локальных переменных. Для этого из указателя стека вычитается число параметров, а LV устанавливается на OBJREF. В OBJREF хранится адрес ячейки, в которой находится старое значение PC. Этот адрес вычисляется суммированием размера кадра локальных переменных (параметры + локальные переменные) с адресом, содержащимся в регистре LV. Сразу над адресом, предназначенным для сохранения старого значения PC, находится адрес, в котором должно быть сохранено старое значение LV. Над этим адресом начинается стек для новой вызванной процедуры. SP указывает на старое значение LV, адрес которого находится сразу под первой пустой ячейкой стека. Напомним, что SP всегда указывает на верхнее слово в стеке. Если стек пуст, то SP указывает на адрес, который находится непосредственно под стеком. На наших рисунках стек всегда заполняется снизу вверх, по направлению к старшим адресам у верхнего края страницы.

И наконец, для выполнения команды INVOKEVIRTUAL регистр PC должен указывать на пятый байт в кодовом пространстве процедуры.

Механизм вызова процедуры



Механизм вызова процедуры

Команда IRETURN противоположна команде INVOKEVIRTUAL. Она освобождает память, используемую процедурой, а также возвращает стек в предыдущее состояние, за исключением того, что, во-первых, OBJREF и все параметры удаляются из стека; во-вторых, возвращенное значение помещается в стек, туда, где раньше находился параметр OBJREF. Чтобы восстановить прежнее состояние, команда IRETURN должна вернуть прежние значения указателей PC и LV. Для этого она обращается к связующему указателю (это слово, определяемое текущим значением LV). В этом месте, где изначально находился параметр OBJREF, команда INVOKEVIRTUAL сохранила адрес, содержащий старое значение PC. Это слово, а также слово над ним извлекаются, чтобы восстановить старые значения PC и LV соответственно. Возвращенное значение, которое хранится на самой вершине стека завершающейся процедуры, копируется туда, где изначально находился параметр OBJREF, после чего SP начинает указывать на этот адрес. И тогда управление передается команде, которая следует сразу за INVOKEVIRTUAL,

Разработка уровня микроархитектуры
Быстродействие и стоимость

Существуют три основных подхода, которые позволяют увеличить скорость выполнения операций:

1. Сократить количество циклов, необходимых для выполнения команды.
2. Упростить организацию машины таким образом, чтобы можно было сделать цикл короче.
3. Сделать так, чтобы несколько операций выполнялось одновременно.

Число циклов, необходимых для выполнения набора операций, называется *длиной пути*

Сокращение длины пути

В процессоре Mic-1 используется минимум аппаратного обеспечения:

- 10 регистров
- простое АЛУ, продублированное 32 раза
- декодер
- схема сдвига
- управляющая память
- некоторые связующие элементы.

Для построения всей системы требуется менее 5000 транзисторов, плюс управляющая память (ПЗУ), плюс основная память (ОЗУ).

Какими способами можно снизить количество микрокоманд в одной команде (т.е. каким образом можно сократить длину пути)?

Слияние цикла интерпретатора с микропрограммой

1. Помещение основного цикла в конце каждой последовательности микрокоманд.

В микроархитектуре Mic-1 основной цикл состоит из микрокоманды, которая должна выполняться в начале каждой IJVM-команды. В некоторых случаях до пустимо ее перекрытие предыдущей командой.

Концепцию перекрытия начала команды можно развивать и дальше. В неко торых случаях основной цикл можно свести к нулю. Это происходит следующим образом. Рассмотрим каждую последовательность микрокоманд, которая завершается переходом к Main1. Каждый раз основной цикл может добавляться в конце этой последовательности (а не в начале следующей), при этом межуровневый переход дублируется много раз (но всегда с одним и тем же набором целевых объектов). В некоторых случаях микрокоманда микроархитектуры Mic-1 может сливаться с предыдущими микрокомандами, поскольку эти команды используются не всегда полностью.

Слияние цикла интерпретатора с микропрограммой

Новая микропрограмма для выполнения команды POP

Микро команда	Операции	Комментарий
popl	MAR = SP = SP — 1; rd	Считывание второго сверху слова в стеке
pop2		Ожидание, пока из памяти считается новое значение TOS
pop3	TOS = MDR; goto Mainl	Копирование нового слова в регистр TOS
Mainl	PC = PC + 1; fetch; goto(MBR)	Регистр MBR содержит код операции; вызов следующего байта; переход

Усовершенствованная микропрограмма для выполнения команды POP

Микро команда	Операции	Комментарий
popl	MAR - SP - SP - 1; rd	Считывание второго сверху слова в стеке
Mainl. pop	PC = PC + 1; fetch	Регистр MBR содержит код операции; вызов следующего байта
pop3	TOS = MDR; goto(MBR)	Копирование нового слова в регистр TOS; пере ход к коду операции

Слияние цикла интерпретатора с микропрограммой

2. Переход от 2-шинной к 3-шинной архитектуре

Можно подвести к АЛУ две полные входные шины, A и B; таким образом, всего получится три шины. Все (или по крайней мере большинство регистров) должны иметь доступ к обеим входным шинам. Преимущество такой системы состоит в возможности складывать любой регистр с любым другим регистром за один цикл

Микропрограмма для выполнения команды ILOAD

Микро команда	Операции	Комментарий
iload 1	H = LV	MBR содержит индекс; копирование LV в H
iload2	MAR = MBRU + H; rd	MAR = адрес локальной переменной, которую нужно поместить в стек
iload3	MAR = SP = SP + 1	Регистр SP указывает на новую вершину стека; подготовка к записи
iload4	PC = PC + 1; fetch; wr	Увеличение PC на 1; вызов следующего кода операции; запись вершины стека
iload5	TOS = MDR; goto Mainl	Обновление TOS
Mainl	PC = PC + 1; fetch; goto(MBR)	Регистр MBR содержит код операции; вызов следующего байта; переход

Слияние цикла интерпретатора с микропрограммой

Микропрограмма для выполнения команды ILOAD в 3-шинной архитектуре

Микро команда	Операции	Комментарий
Iloadl	MAR - MBRU + LV; rd	MAR = адрес локальной переменной, которую нужно поместить в стек
Iload2	MAR = SP - SP + 1	Регистр SP указывает на новую вершину стекаГ подготовка к записи
Iload3	PC = PC + 1; fetch; wr	Увеличение PC на 1; вызов следующего кода операции; запись вершины стека
Iload4	TOS - MDR	Обновление TOS
IloadS	PC - PC + 1; fetch; goto(MBR)	Регистр MBR уже содержит код операции; вы зов индексного байта

Блок выборки команд

В команде могут происходить следующие операции:

- значение PC пропускается через АЛУ и увеличивается на 1;
- PC используется для вызова следующего байта в потоке команд;
- операнды считываются из памяти;
- операнды записываются в память;
- АЛУ выполняет вычисление, и результаты сохраняются в памяти

Если команда содержит дополнительные поля (для операндов), каждое поле должно вызываться явно, по одному байту за раз. Поле можно использовать только после того, как эти байты будут объединены. При выборке и компонов ке поля АЛУ должно для каждого байта увеличивать PC на единицу, а затем объединять получившийся индекс или смещение. Когда, помимо выполнения основной работы команды, приходится вызывать и объединять ноля этой команды, АЛУ используется практически в каждом цикле

Блок выборки команд

В микроархитектуре Mic-1 с АЛУ можно снять большую часть нагрузки, если создать независимый блок для вызова и обработки команд.

Этот блок, который называется **блоком выборки команд** (Instruction Fetch Unit, **IFU),** может независимо от АЛУ увеличивать значение PC на единицу и вызывать байты из потока байтов до того, как они понадобятся.

Блок IFU содержит схему инкремента, которая по строению гораздо проще, чем полный сумматор

Блок выборки команд

Блок выборки команд может также объединять 8-разрядные и 16-разрядные операнды, чтобы они могли использоваться сразу, как только потребуются.

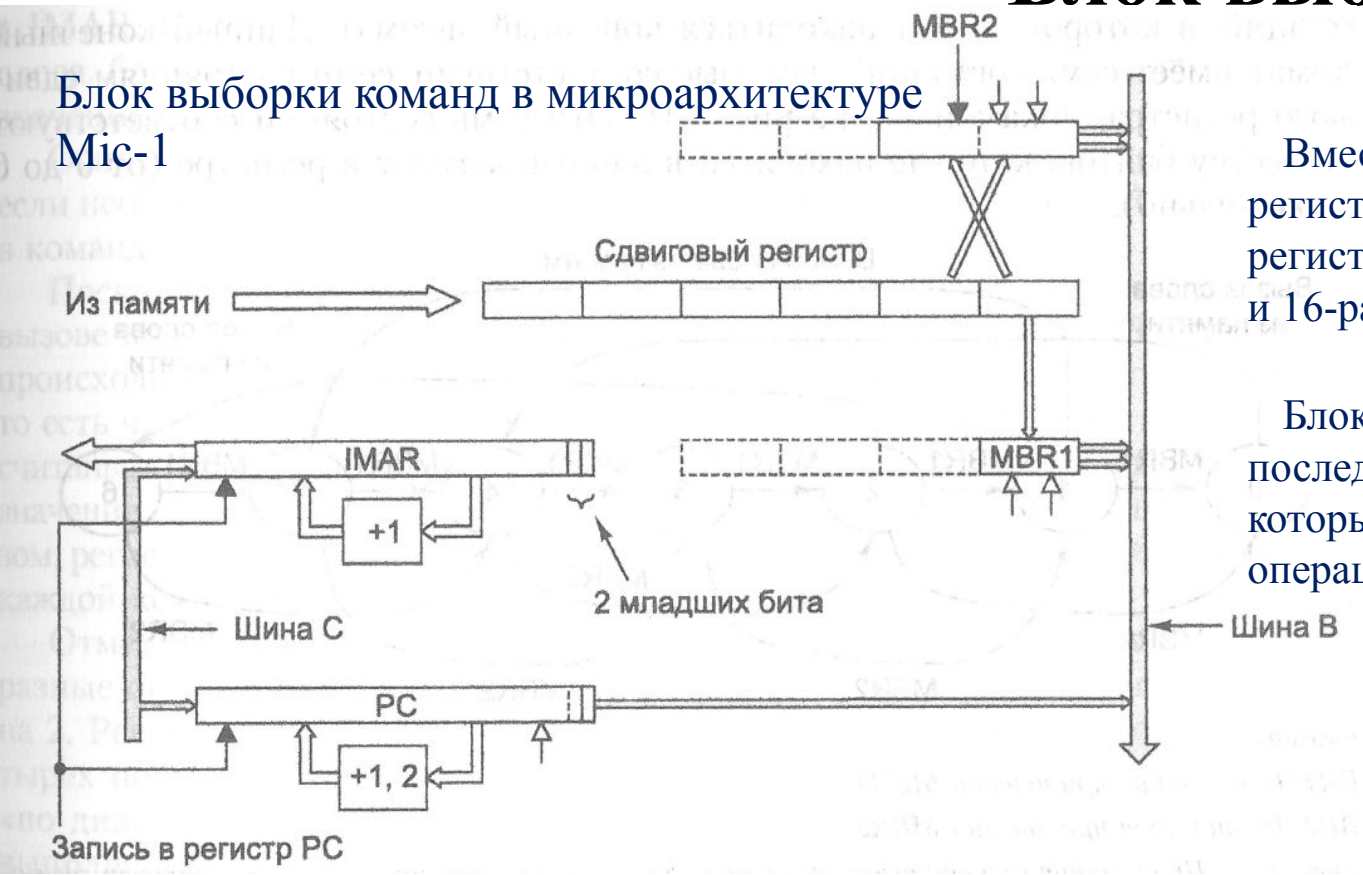
Это можно осуществить по крайней мере двумя способами:

- Блок IFU может интерпретировать каждый код операции, определять, сколько дополнительных полей нужно вызвать, и собирать их в регистр, который будет использоваться основным операционным блоком.

- Блок IFU может постоянно предоставлять следующие 8- или 16-разрядные фрагменты данных независимо от того, имеет это смысл или нет. Тогда основной операционный блок может запрашивать любые данные, которые ему требуются.

Блок выборки команд

Блок выборки команд в микроархитектуре Mic-1



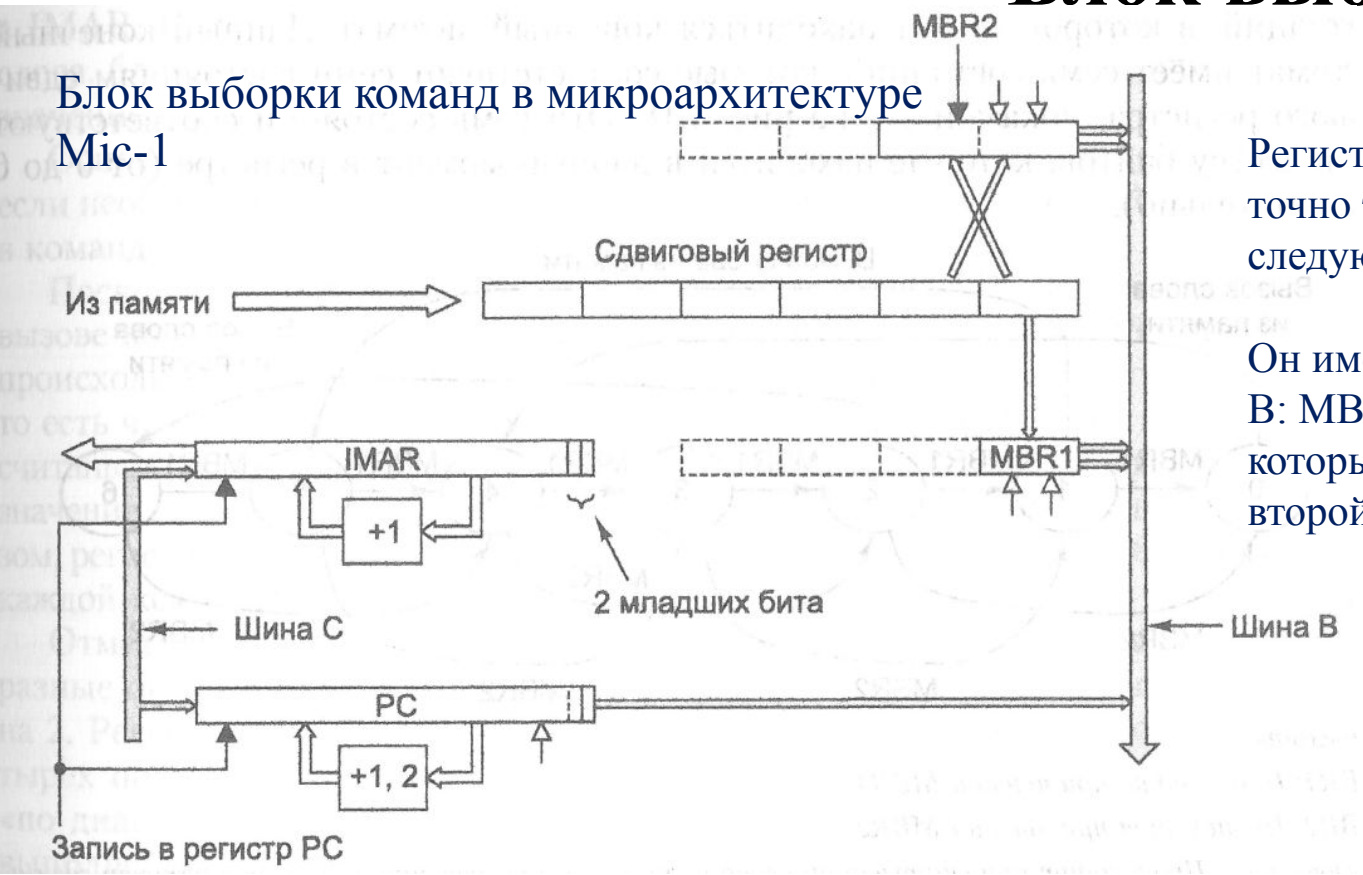
Вместо одного 8-разрядного регистра MBR присутствуют два регистра MBR: 8-разряд-пый MBR1 и 16-разрядный MBR2.

Блок IFU следит за самым последним байтом или байтами, которые поступили в основной операционный блок.

Блок IFU также передает следующий байт в регистр MBR, как и в архитектуре Mic-1, только в данном случае он автоматически определяет, когда значение регистра считано, вызывает следующий байт и сразу загружает его в регистр MBR1. Как и в ми кроархитектуре Mic-1, он имеет два интерфейса с шиной B: MBR1 и MBR1U. Первый получает знаковое расширение до 32 битов, второй дополнен нулями.

Блок выборки команд

Блок выборки команд в микроархитектуре Mic-1



Регистр MBR2 функционирует точно так же, но содержит следующие 2 байта.

Он имеет два интерфейса с шиной B: MBR2 и MBR2U, первый из которых расширен по знаку, а второй дополнен до 32 бит нулями.

Блок выборки команд отвечает за выборку потока байтов. Для этого он ис пользует стандартный 4-байтный порт памяти, вызывая полные 4-байтные слова заранее и загружая следующие байты в сдвиговый регистр, который выдает их по одному или по два за раз в том порядке, в котором они вызываются из памяти.

Блок выборки команд

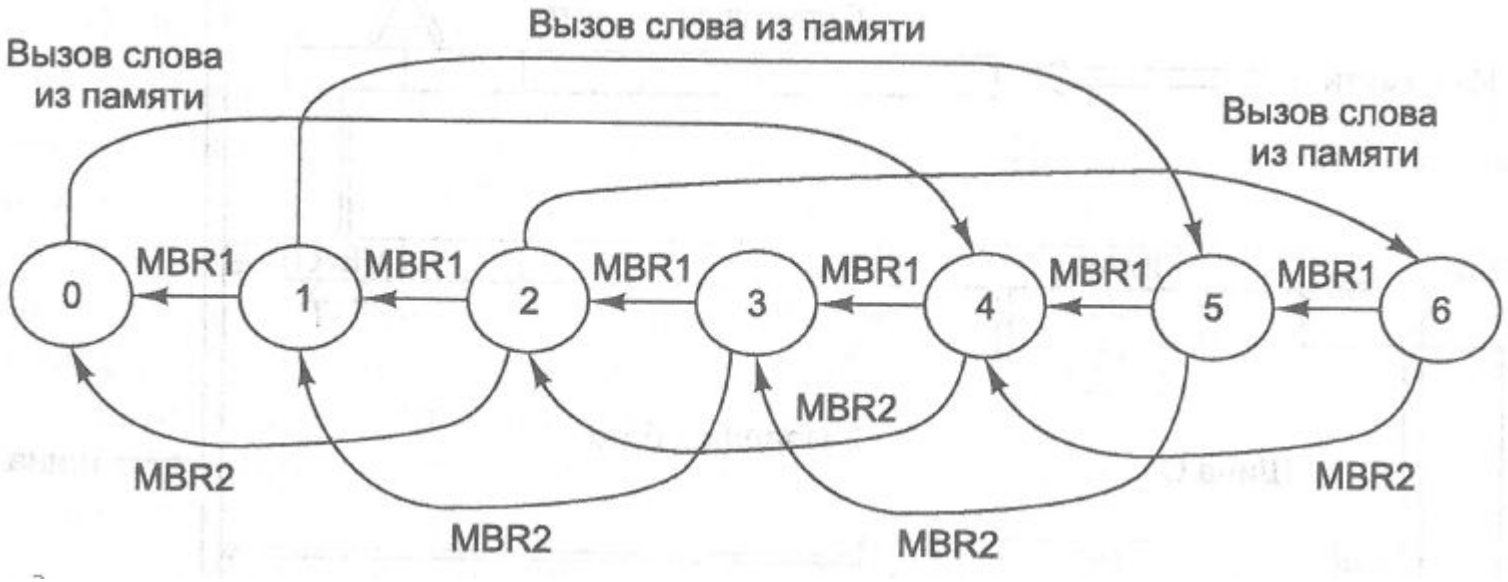
Задача сдвигового регистра — сохранить последовательность поступающих байтов для загрузки в регистры MBR1 и MBR2.

MBR1 содержит самый старший байт сдвигового регистра (при считывании значение сдвигового регистра сдвигается вправо на 1 байт)

MBR2 содержит 2 старших байта (старшим является левый байт), которые формируют 16-разрядное целое число (при считывании значение сдвигового регистра сдвигается вправо на 2 байта) [Два байта в регистре MBR2 могут быть получены из различных слов памяти, поскольку IJVM-команды никак не связаны с границами слов]

Конечный автомат (Finite State Machine, FSM) для реализации блока выборки команд

Блок выборки команд



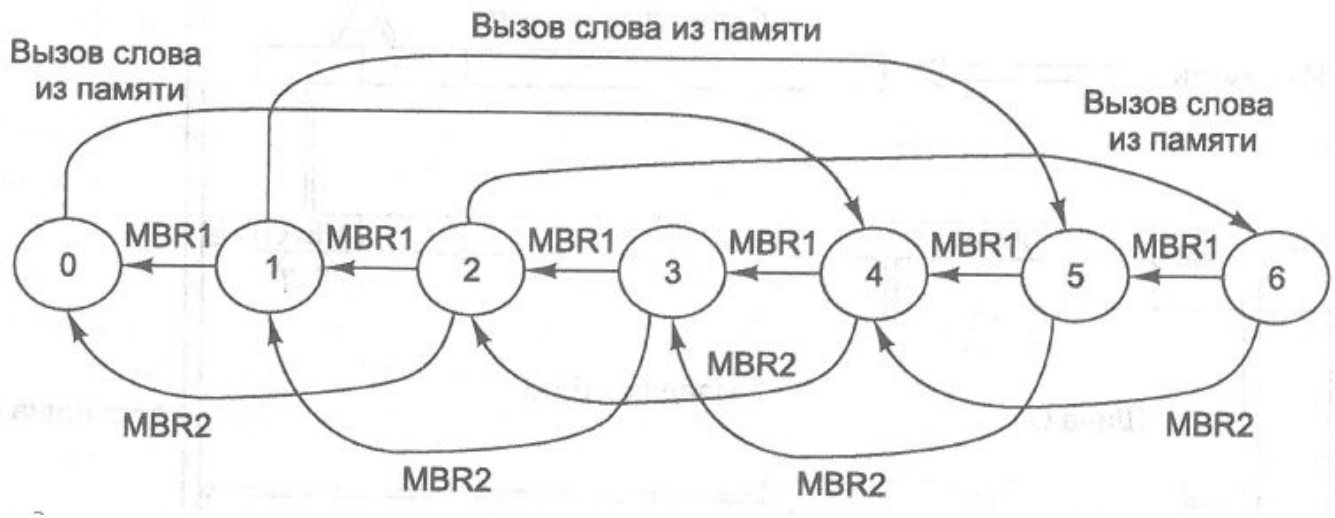
КА характеризуется **состояниями** (на рисунке это кружочки) и **переходами** (это дуги от одного состояния к другому).

Каждое состояние – это одна из возможных ситуаций, в которой может находиться конечный автомат.

Каждая дуга отражает возможное событие.

Конечный автомат (Finite State Machine, FSM) для реализации блока выборки команд

Блок выборки команд

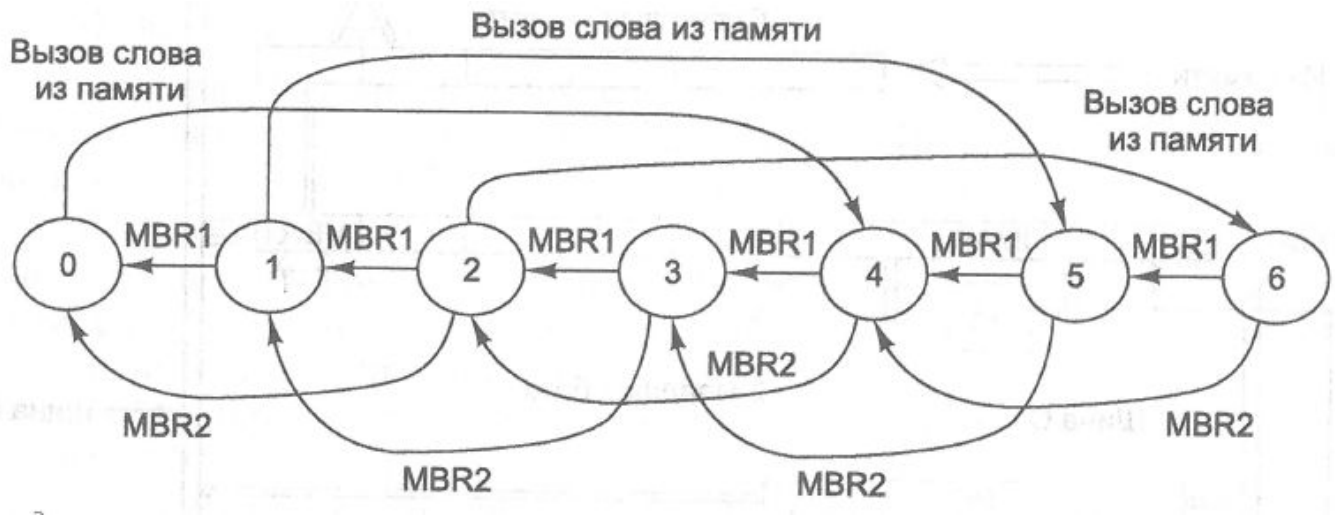


Данный КА:
имеет семь состояний, которые соответствуют семи состояниям сдвигового регистра. Эти семь состояний соответствуют количеству байтов, которые находятся в данный момент в регистре (от 0 до 6 включительно)

В нашем КА возможны два различных события.
Первое – чтение одного байта из регистра MBR1 [Активизирует сдвиговый регистр, самый правый байт в нем исчезает, и осуществляется переход в другое состояние (меньшее на 1)]
Второе событие – чтение 2 байт из регистра MBR2. [Осуществляется переход в состояние, меньшее на 2 (например, из состояния 2 в состояние 0 или из состояния 5 в состояние 3). Оба этих перехода вызывают перезагрузку регистров MBR1 и MBR2. Когда КА переходит в состояния 0, 1 или 2, инициируется обращение к памяти, чтобы вызвать новое слово. При поступлении слова номер состояния увеличивается на 4]

Конечный автомат (Finite State Machine, FSM) для реализации блока выборки команд

Блок выборки команд



Данный КА:
имеет семь состояний, которые соответствуют семи состояниям сдвигового регистра. Эти семь состояний соответствуют количеству байтов, которые находятся в данный момент в регистре (от 0 до 6 включительно)

В нашем КА возможны два различных события.
Первое – чтение одного байта из регистра MBR1 [Активизирует сдвиговый регистр, самый правый байт в нем исчезает, и осуществляется переход в другое состояние (меньшее на 1)]
Второе событие – чтение 2 байт из регистра MBR2. [Осуществляется переход в состояние, меньшее на 2 (например, из состояния 2 в состояние 0 или из состояния 5 в состояние 3). Оба этих перехода вызывают перезагрузку регистров MBR1 и MBR2. Когда КА переходит в состояния 0, 1 или 2, инициируется обращение к памяти, чтобы вызвать новое слово. При поступлении слова номер состояния увеличивается на 4]

Блок выборки команд

Для правильного функционирования схемы выборки команд (СВК)

- СВК должна блокироваться в том случае, если от нее требуют произвести какие-то действия, которые она выполнить не может (например, передать значение в MBR2, когда в сдвиговом регистре находится только 1 байт, а память все еще занята вызовом нового слова)
- БВК не может выполнять несколько операций одновременно, поэтому все входящие события должны передаваться последовательно.
- при каждом изменении РС БВК должен обновляться.

Блок выборки команд

Для правильного функционирования схемы выборки команд (СВК)

- СВК должна блокироваться в том случае, если от нее требуют произвести какие-то действия, которые она выполнить не может (например, передать значение в MBR2, когда в сдвиговом регистре находится только 1 байт, а память все еще занята вызовом нового слова)
- БВК не может выполнять несколько операций одновременно, поэтому все входящие события должны передаваться последовательно.
- при каждом изменении РС БВК должен обновляться.

Блок выборки команд

БВК имеет собственный регистр адреса ячейки памяти, называемый IMAR и используемый для обращения к памяти, когда нужно вызвать новое слово.

У IMAR есть специальная схема инкремента, поэтому основному АЛУ не требуется прибавлять единицу к значению PC для вызова следующего слова.

БВК должен контролировать шину C, чтобы каждый раз при загрузке регистра PC новое значение PC также копировалось в IMAR.

Основной операционный блок записывает значение в PC только в том случае, если необходимо изменить характер последовательности байтов. Это происходит в команде перехода, а также в командах INVOKEVIRTUAL и IRETURN.

Блок выборки команд

Т.к. микропрограмма больше не увеличивает PC явным образом при вызове кода операции, блок выборки команд должен обновлять PC сам.

Блок IFU способен распознать, что байт из потока команд получен, то есть что значения регистров MBR1 и MBR2 (или их вариантов без знака) уже считаны. С регистром PC связана отдельная схема инкремента, которая увеличивает значение на 1 или на 2 в зависимости от того, сколько байтов получено. То, регистр PC всегда содержит адрес первого еще не полученного байта. В начале каждой команды в регистре MBR находится адрес кода операции этой команды.

Блок выборки команд

Существует две разных схемы инкремента, которые выполняют разные функции

- Регистр PC считает *байты* и увеличивает значение на 1 или на 2.
- Регистр IMAR считает *слова* и увеличивает значение только на 1 (для четырех новых байтов).

Как и MAR, регистр IMAR соединен с адресной шиной «по диагонали»: бит 0 регистра IMAR связан с адресной линией 2 и т. д. для выполнения неявного перехода от адреса слова к адресу байта.

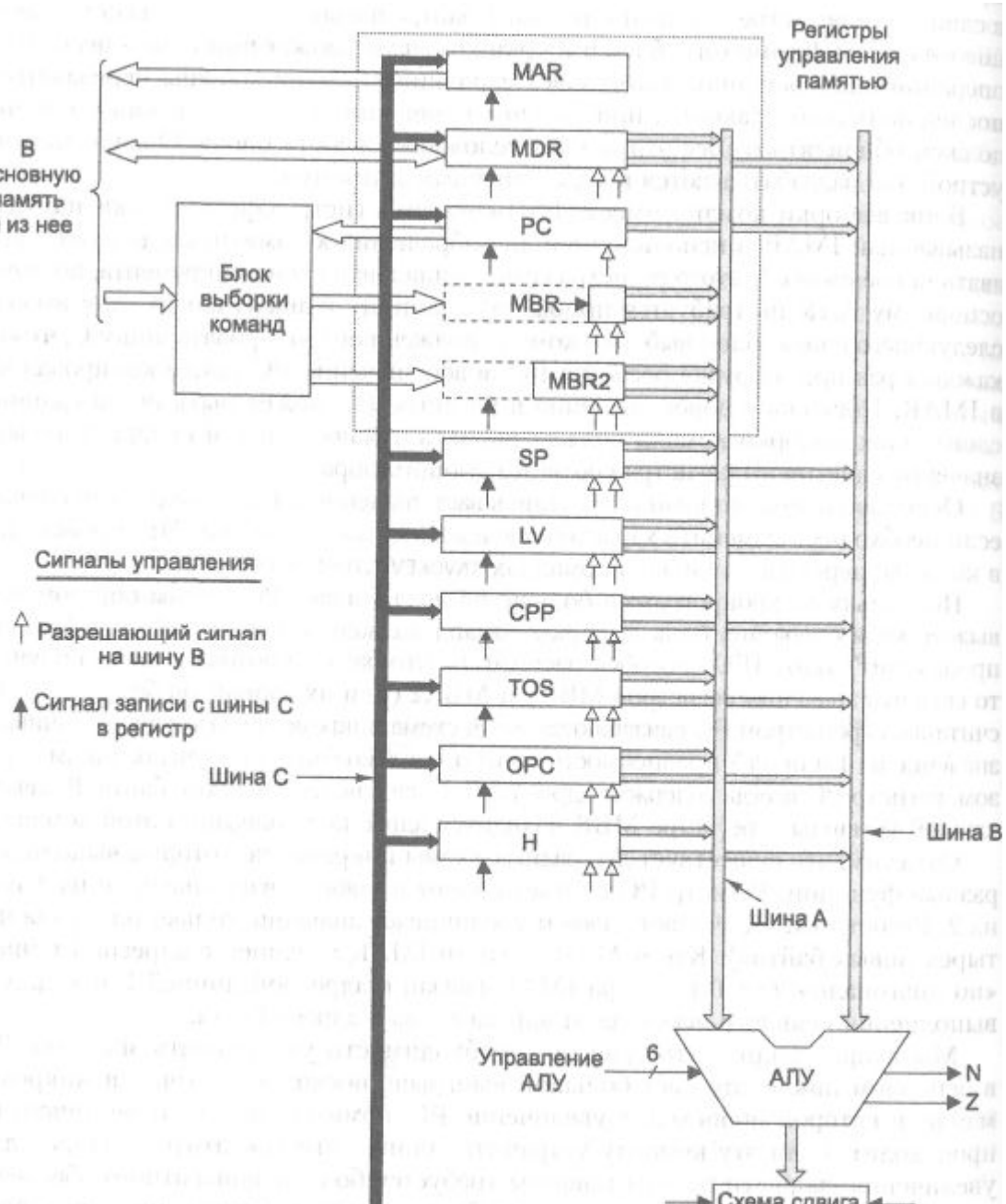
Упреждающая выборка команд из памяти

3. Выборка команд из памяти осуществляется специализированным функциональным блоком.

БВК может значительно сократить длину пути для средней команды:

- ✔ полностью устраняет основной цикл, поскольку в конце каждой команды просто стразу осуществляется переход к следующей.
- ✔ АЛУ не нужно увеличивать значение PC.
- ✔ блок IFU сокращает длину пути всякий раз, когда вычисляется 16-разрядный индекс или смещение, поскольку объединяет 16-разрядное значение и сразу передает его в АЛУ в виде 32-разрядного значения без необходимости производить объединение в регистре

Микроархитектура Mic-2 – усовершенствованная версия Mic-1, к которой добавлен блок выборки команд



Работает быстрее и требует меньше управляющей памяти

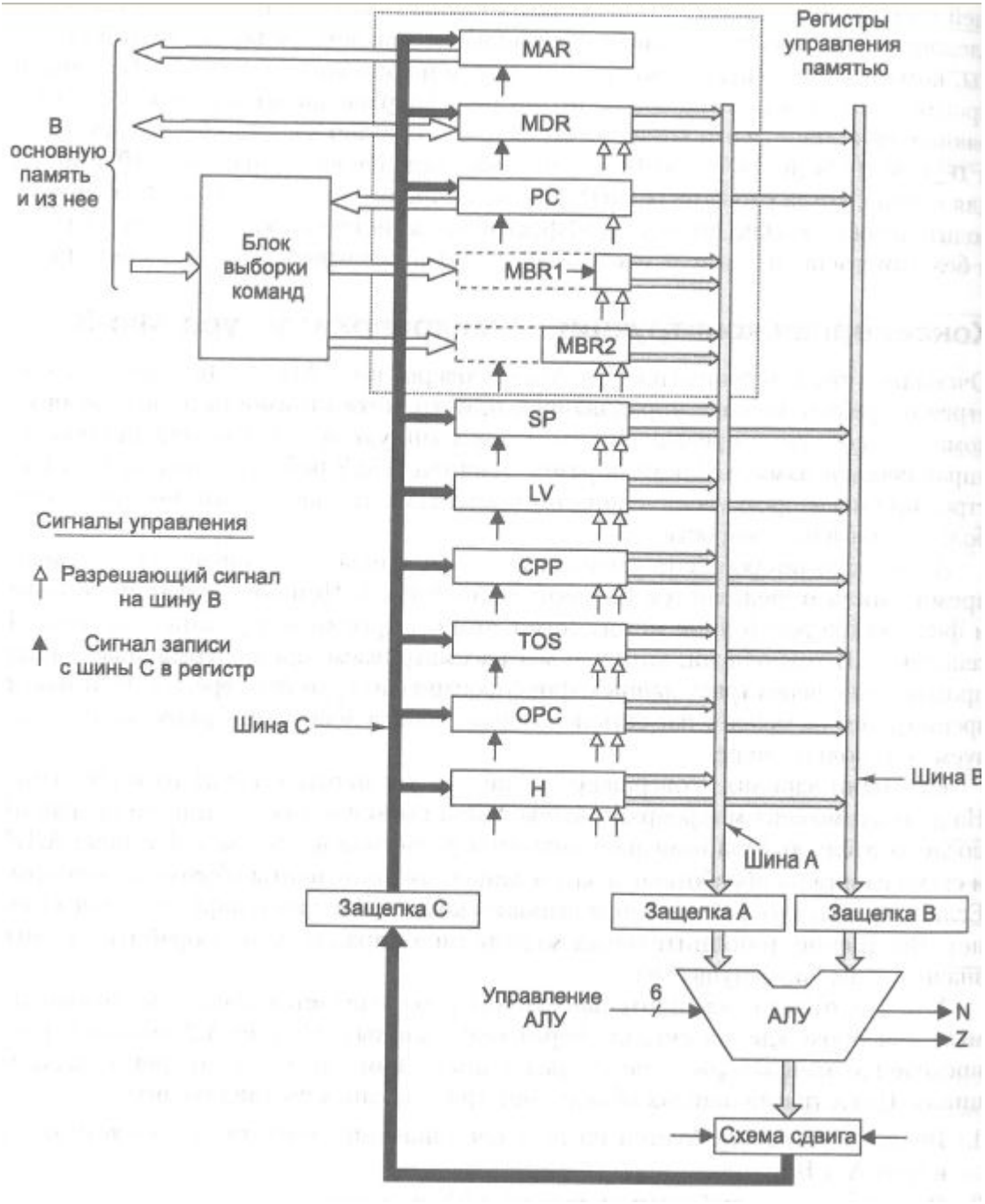
Mic-2 выполняет большинство операций последовательно. Она помещает значения регистров на шины, ждет, пока АЛУ и схема сдвига их обработают, а затем записывает результаты обратно в регистры. Если не учитывать работу блока выборки команд, никакого параллелизма здесь нет.

Конвейерная конструкция

Еще один вариант усовершенствования — увеличить степень параллелизма

Цикл тракта данных объединяет три основных составляющих:

- ✔ Время, которое требуется на передачу значений выбранных регистров на шины A и B.
- ✔ Время, которое требуется на работу АЛУ и схемы сдвига.
- ✔ Время, которое требуется на передачу полученных значений обратно в регистры и сохранение этих значений.



3-шинная архитектура с блоком выборки команд и тремя дополнительными защелками (регистрами), каждая из которых расположена в середине каждой шины.

Эти регистры записываются в каждом цикле. Они делят тракт данных на отдельные части, которые могут функционировать независимо друг от друга.

Мы будем называть такую архитектуру **конвейерной моделью**

Конвейерная конструкция

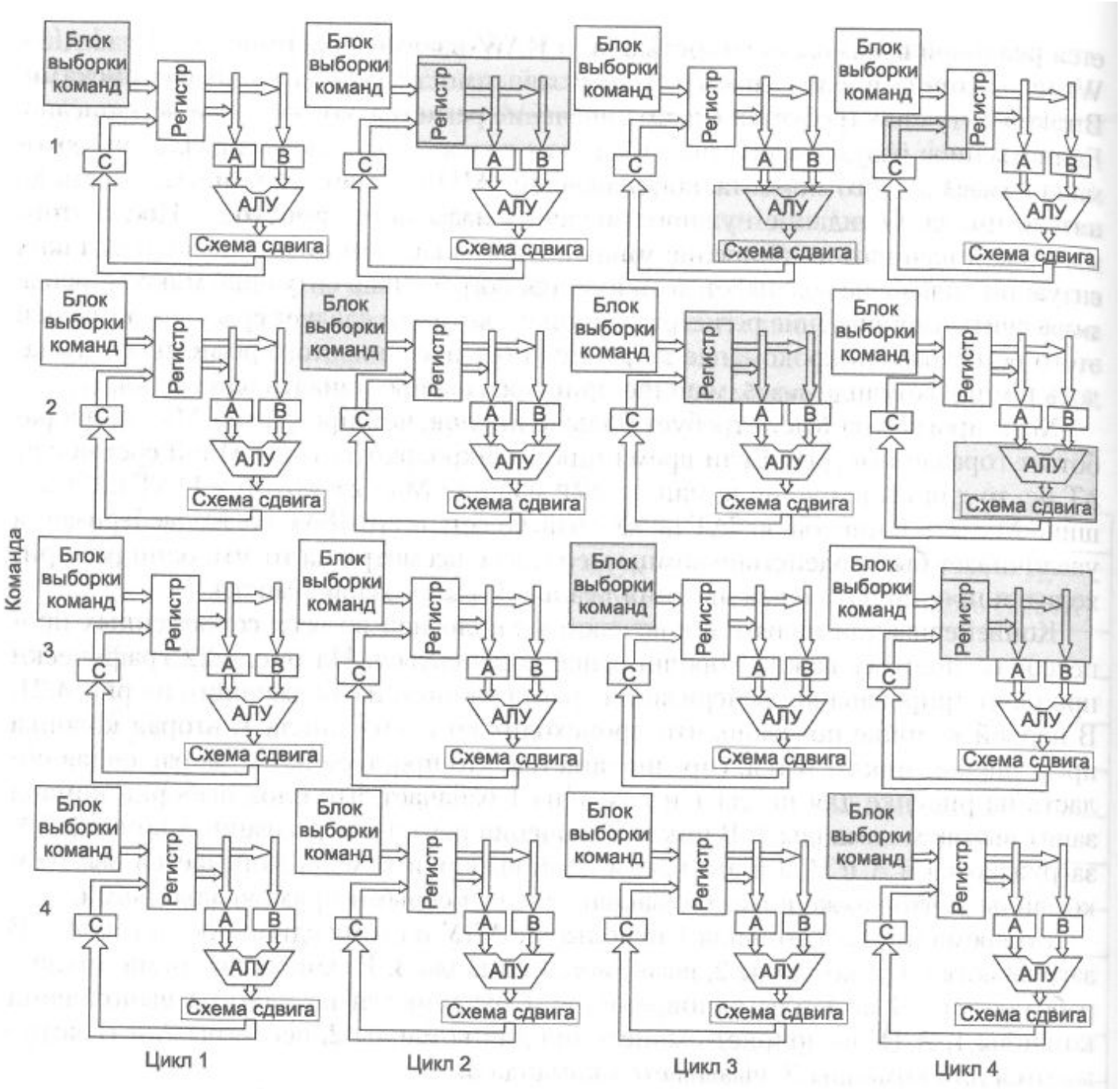
Теперь для прохождения сигнала через тракт данных требуются 3 цикла: один для загрузки регистров A и B, второй для запуска АЛУ и схемы сдвига, а также загрузки регистра C, третий для сохранения значения регистра-защелки C обратно в нужных регистрах.

Зато:

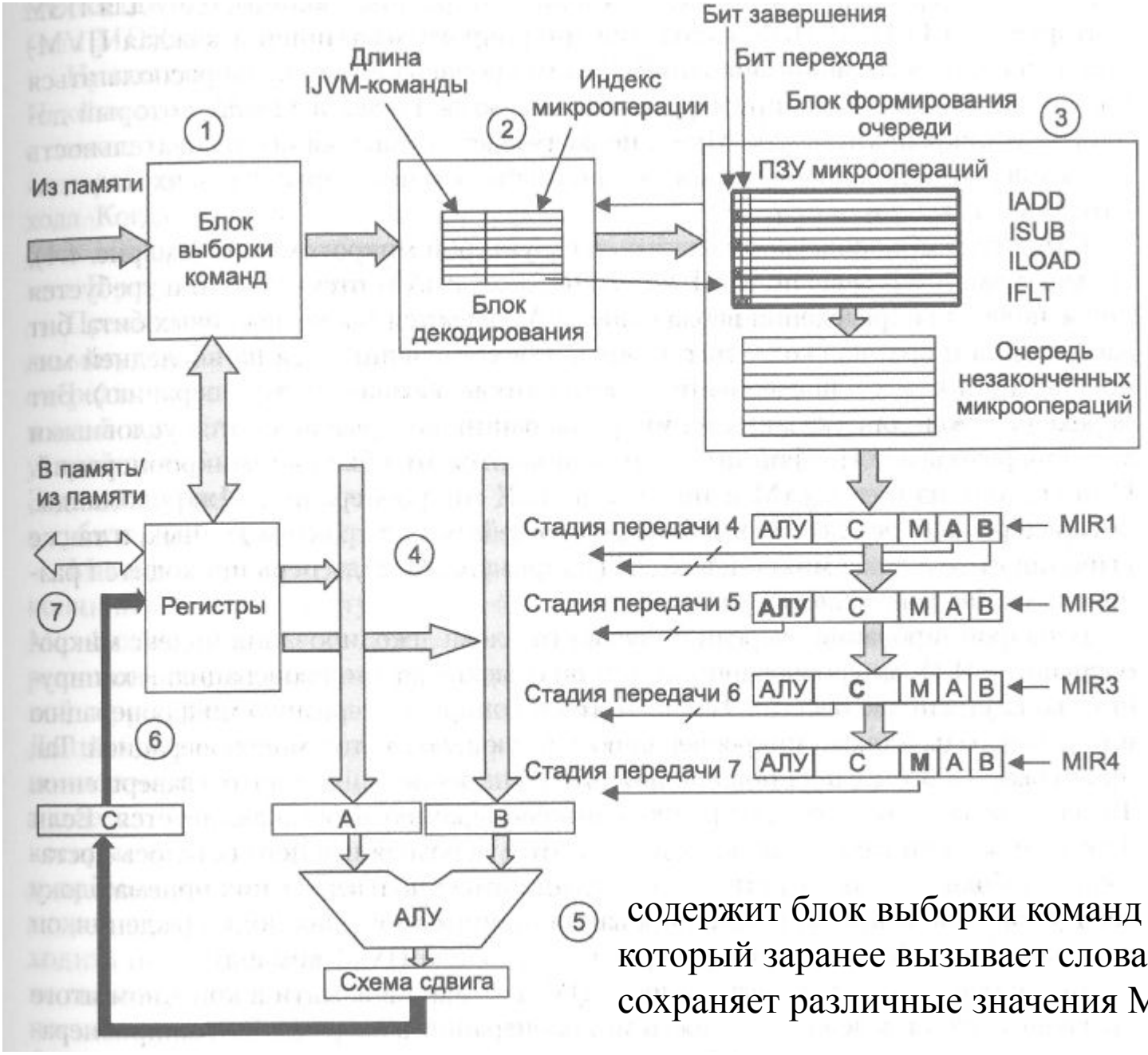
1. Мы можем повысить тактовую частоту, поскольку максимальная задержка теперь стала меньше.
2. Во время каждого цикла мы можем использовать все части тракта данных.

После разбиения тракта данных на три части максимальная задержка прохождения сигнала уменьшается, в результате тактовая частота может повышаться. [Будем считать, что если разбить цикл тракта данных на три примерно равных интервала, тактовая частота увеличится втрое]

Графическая иллюстрация работы конвейера



Семиступенчатый конвейер



содержит блок выборки команд (IFU), который заранее вызывает слова из памяти и сохраняет различные значения MBR

Семиступенчатый конвейер

Блок выборки команд передает входящий поток байтов в новый компонент – **блок декодирования**

БД содержит внутреннее ПЗУ, которое индексируется кодом IJVM-операции. Каждый элемент (ряд) блока состоит из двух частей: поля длины IJVM-команды и индекса в другом ПЗУ – ПЗУ микроопераций.

Длина IJVM-команды нужна для того, чтобы блок декодирования мог разделить входящий поток байтов и установить, какие байты являются кодами операций, подсчитывает, сколько раз этот регистр используется выполняющимися командами в качестве источника. Если одновременно может выполняться максимум 15 команд, будет достаточно 4-разрядного счетчика. Когда запускается команда, элементы счетчика обращений, соответствующие регистрам операндов, увеличиваются на 1. Когда выполнение команды завершено, соответствующие элементы счетчика уменьшаются на 1.