

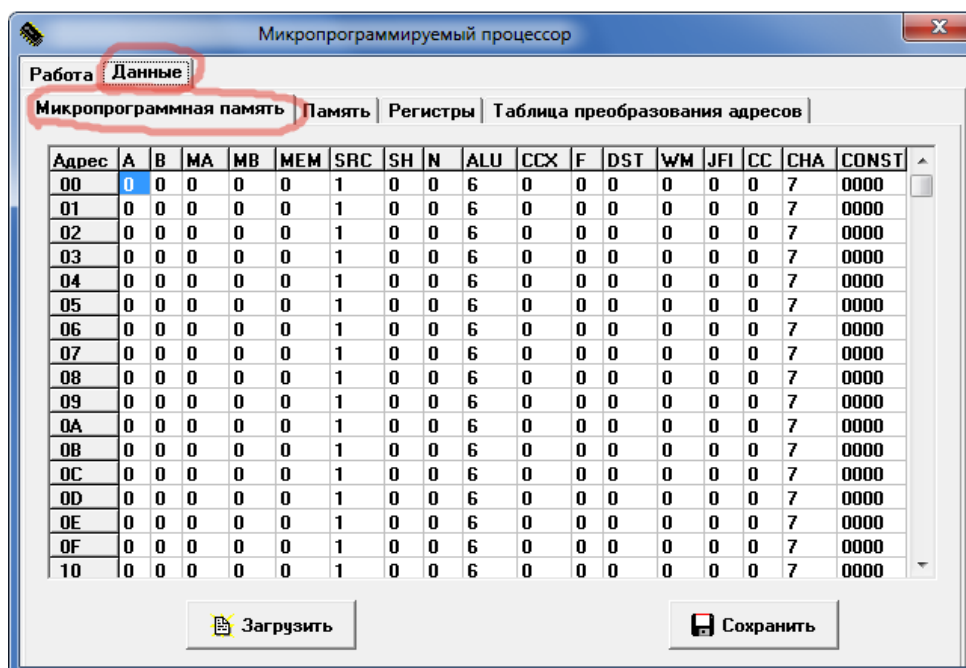
Программа «micro» состоит из 5 частей:

I. «Микропрограммная память» - то есть, то место куда будет записываться программа.

Каждая строка – это одна микрокоманда.

Пример:

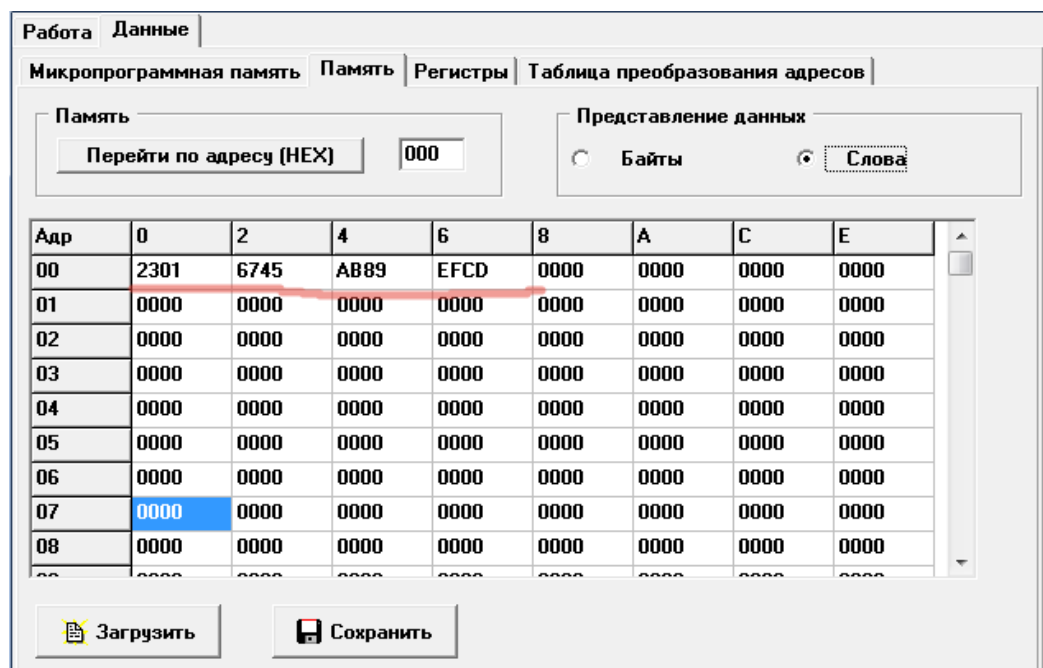
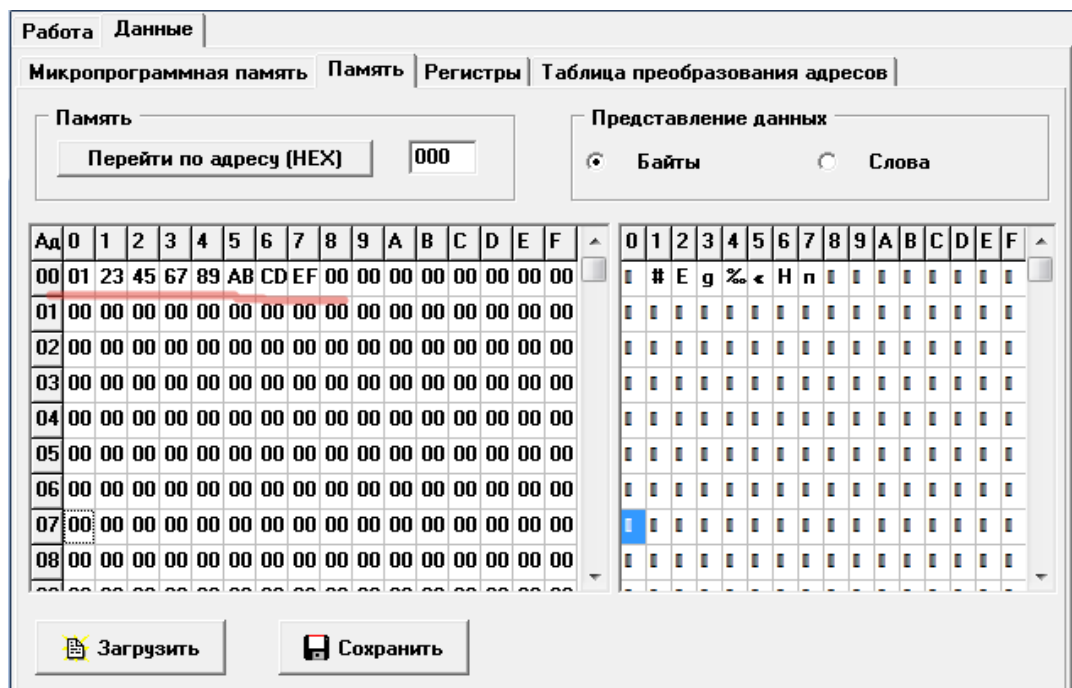
$AX=AX+BX$



Адрес	A	B	MA	MB	MEM	SRC	SH	N	ALU	CCX	F	DST	WM	JFI	CC	CHA	CONST
00	0	0	0	0	0	1	0	0	6	0	0	0	0	0	0	7	0000
01	0	0	0	0	0	1	0	0	6	0	0	0	0	0	0	7	0000
02	0	0	0	0	0	1	0	0	6	0	0	0	0	0	0	7	0000
03	0	0	0	0	0	1	0	0	6	0	0	0	0	0	0	7	0000
04	0	0	0	0	0	1	0	0	6	0	0	0	0	0	0	7	0000
05	0	0	0	0	0	1	0	0	6	0	0	0	0	0	0	7	0000
06	0	0	0	0	0	1	0	0	6	0	0	0	0	0	0	7	0000
07	0	0	0	0	0	1	0	0	6	0	0	0	0	0	0	7	0000
08	0	0	0	0	0	1	0	0	6	0	0	0	0	0	0	7	0000
09	0	0	0	0	0	1	0	0	6	0	0	0	0	0	0	7	0000
0A	0	0	0	0	0	1	0	0	6	0	0	0	0	0	0	7	0000
0B	0	0	0	0	0	1	0	0	6	0	0	0	0	0	0	7	0000
0C	0	0	0	0	0	1	0	0	6	0	0	0	0	0	0	7	0000
0D	0	0	0	0	0	1	0	0	6	0	0	0	0	0	0	7	0000
0E	0	0	0	0	0	1	0	0	6	0	0	0	0	0	0	7	0000
0F	0	0	0	0	0	1	0	0	6	0	0	0	0	0	0	7	0000
10	0	0	0	0	0	1	0	0	6	0	0	0	0	0	0	7	0000

II. Оперативная память

Обратите внимание на то, в каком порядке располагаются биты, при переходе в режим слова.



III. Регистры

Их можно разделить на 2 типа:

- 1) Те, в которых мы можем хранить любые данные (РОН-ы или же РЗУ) (обведены красным)

РОН – регистр общего назначения

РЗУ – регистровое запоминающее устройство

- 2) Те, которые исполняют строго определённую роль в работе вашей программы.

Лишь в половину из них мы сами записываем значения. В остальные значения записываются самостоятельно по ходу выполнения программы.

После настройки регистров нужно нажать «ОК» для сохранения.

Работа Данные

Микропрограммная память Память **Регистры** Таблица преобразования адресов

Р0Ны

AX	0000	SP	0000	CS	0000	IP	0000
BX	0000	BP	0000	SS	0000	PSW	0000
CX	0000	SI	0000	DS	0000	RGK	3453
DX	0000	DI	0000	ES	0000	RW	0000

Регистры флагов

RFI	04
RFD	00

Рабочий регистр

RGQ	0000
-----	------

Регистры блока МПУ

CMK	0005	ERROR	0000	Режим	MK
MUAD	0005	RACT	0000		
STP	0000				

Регистры ОП

ARAM	0000
RGW	0000
RGR	0000

Загрузить Сохранить OK Отмена

IV. «Работа» (само окно выполнения программы)

Здесь мы можем проследить как значения перемещаются по регистрам.

Для начала работы нужно нажать кнопку «Рестарт», и затем каждая строка микропрограммной памяти будет выполняться последовательно при нажатии кнопки «продолжить».

Чтобы не кликать постоянно кнопку «продолжить» можно поставить режим авто («Авто МК»). Режимы выбираются в поле «Регистры».

Кнопка «стоп» останавливает выполнение если вы в режиме авто.

«Настройка» позволяет выбрать задержку между выполнением команд в режиме авто.

память | Память | Регистры | Таблица преобразования

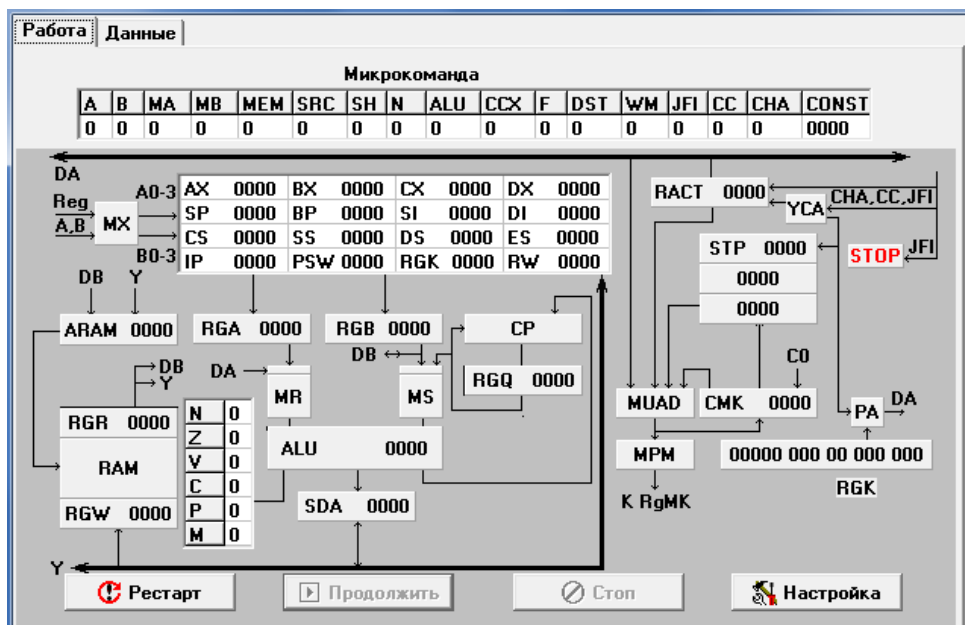
P	0000	CS	0000	IP	0000
P	0000	SS	0000	PSW	0000
I	0000	DS	0000	RGK	0000
I	0000	ES	0000	RW	0000

ИУ

ERROR	0000
RACT	0000

Режим MK

Сохранить OK



V. «Таблица преобразования адресов»

Она понадобится только в 4-ой и 5-ой лабораторных.

Каждую микрокоманду можно разделить на 2 блока:

1. Арифметическая или логическая операция

(над 1-им, 2-умя или в определённом случае 3-емя регистрами)

2. Операция перехода (к другой команде)

Адрес	A	B	MA	MB	MEM	SRC	SH	N	ALU	CCX	F	DST	WM	JFI	CC	CHA	CONST
00	0	0	0	0	0	1	0	0	6	0	0	0	0	0	0	7	0000

Также есть независимое поле **CONST** которое используется и в первом, и во втором блоке.

Зачастую, именно из-за него не получается разместить в одной команде и **АЛУ** и переход.

В **АЛУ** по умолчанию стоит операция считывания значения регистра **АХ** и ничего не делая с ним.

В блоке переходов по умолчанию стоит переход к следующей по номеру микрокоманде.

Арифметическую или логическую операцию можно представить в общем виде таким образом:

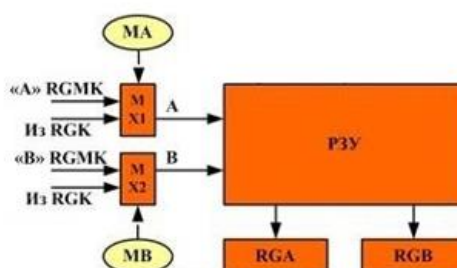
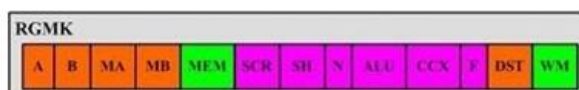
1. Выбираются регистры, с которыми мы будем работать.
2. Над значениями в этих регистрах выполняется логическая, арифметическая или какая-либо ещё операция.
3. Значение, полученное в предыдущей операции, можно подвергнуть сдвигу.
4. Выполняется запись полученного значения в какой-либо регистр

Адрес	A	B	MA	MB	MEM	SRC	SH	N	ALU	CCX	F	DST	WM	JFI	CC	CHA	CONST
00	0	0	0	0	0	1	0	0	6	0	0	0	0	0	0	7	0000

Неотмеченные поля понадобятся нам позже.

- I. Первые две ячейки **A** и **B** отвечают за выбор двух регистров из **РЗУ** (регистрового запоминающего устройства), с которыми мы будем работать. Значения этих регистров затем будут записаны в регистры **RGA** и **RGB**.

Поле	<u>A</u>	<u>B</u>
Бит	4	4
Def	0	0



Поле A/B	Значение	Поле A/B	Значение
0	AX	8	CS
1	CX	9	SS
2	DX	A	DS
3	BX	B	ES
4	SP	C	IP
5	BP	D	PSW
6	SI	E	RGK
7	DI	F	RW

- II. Далее за дело берётся поле микрокоманды **SRC**, которое определяет откуда будут взяты 2 операнда для выполнения дальнейших операций.

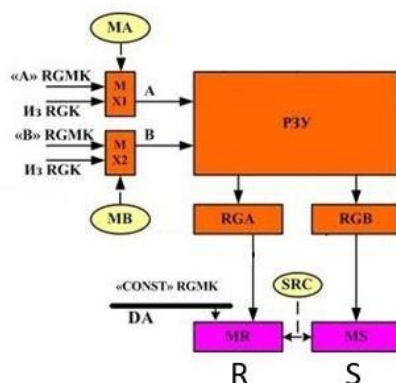
К примеру, при значении этого поля равным 1, берём значения из указанных ранее регистров, то есть значения регистров **RGA** и **RGB**. А при значении 5, вместо 1-ого регистра **RGA** берётся значение из поля **CONST** микрокоманды.

MR и **MS** – это просто мультиплексоры. То есть, они просто указывают на регистр, из которого будет браться значение.

Поле	A	B
Бит	4	4
Def	0	0

SRC
3
1

CONST
16
0



Поле SRC	0	1	2	3	4	5	6	7
Операнд R	0000	RGA	RGA	RGA	RGA*2	CONST	CONST	CONST
Операнд S	0000	RGB	RGQ	RGR	RGB	RGB	RGR	RGQ

III. Затем эти операнды отправляются в регистр **ALU** для дальнейшего выполнения арифметической, логической операции или просто переноса в **ALU** только одного из операндов. За это отвечает поле **ALU**. Как вы могли заметить, в таблице у некоторых операций идёт прибавление **C0**.

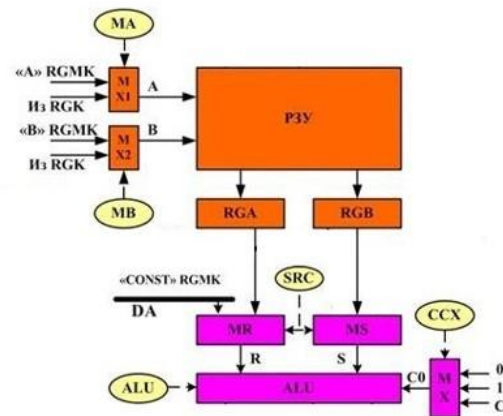
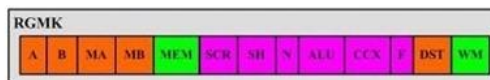
C0 – это число, которое может быть только 0 или 1, а зависит оно от поля **CCX** микрокоманды. При **CCX=0 C0=0**, при **CCX=1 C0=1**, а при значении **CCX 2** или **3** значение определяется флажком **C**.

Поле	A	B
Бит	4	4
Def	0	0

SRC
3
1

ALU	CCX
4	2
6	0

CONST
16
0



Поле ALU	Операция АЛУ	Поле ALU	Операция АЛУ
0	На всех выходах «0»	8	Умножение на 2 бита
1	$S - R - 1 + C0$	9	$R \& S$
2	$R - S - 1 + C0$	A	$R \& \overline{S}$
3	$R + S + C0$	B	$\overline{R} \& S$
4	$S + C0$	C	$R \vee S$
5	$\overline{S} + C0$	D	$\overline{R} \vee S$
6	$R + C0$	E	$R \oplus S$
7	$\overline{R} + C0$	F	$\overline{R} \oplus S$

R – первый операнд

S – второй операнд

IV. Затем значение из **ALU** переносится в регистр **SDA**, откуда его можно перенести назад в **P3Y**.

Для этого в поле **DST** напишем **4**.

Запись произойдёт в регистр, указанный в поле **B**.

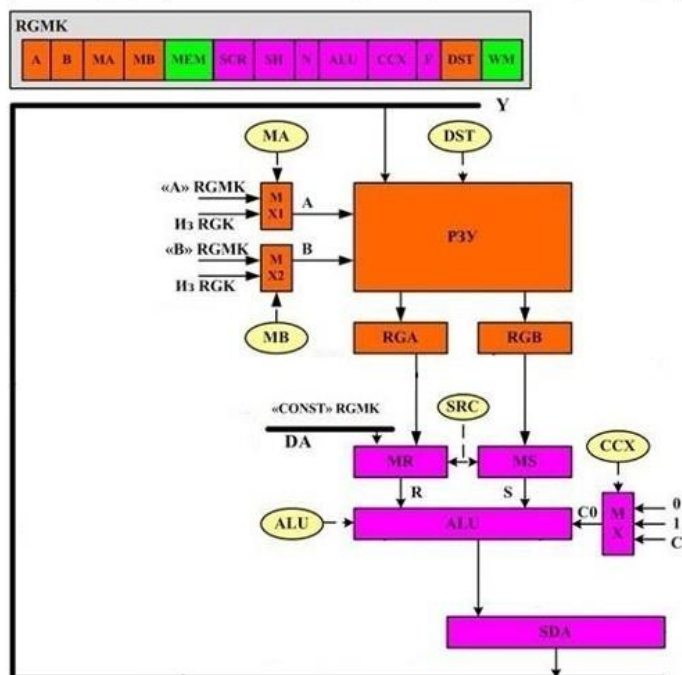
Поле	A	B
Бит	4	4
Def	0	0

SRC
3
1

ALU	CCX
4	2
6	0

DST
3
0

CONST
16
0



V. Теперь перейдём к рассмотрению сдвигов.

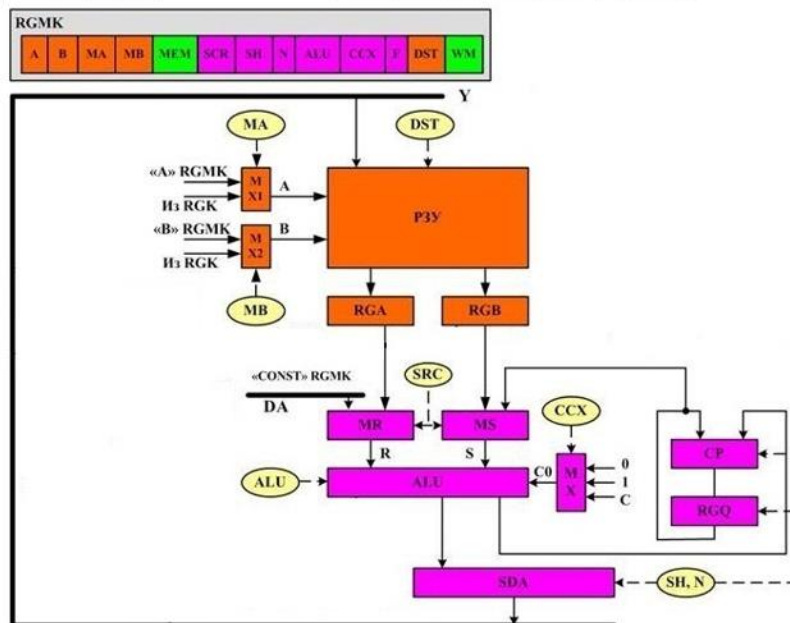
Перед тем, как отправить операнд из регистра **ALU** в **SDA**, его можно подвергнуть сдвигу (сдвиг выполняется после операции из поля **ALU**). За сдвиг отвечают 2 поля микрокоманды **SH** и **N**. Поле **SH** определяет вид сдвига, а поле **N** количество бит, на которых нужно выполнить сдвиг.

Поле	A	B
Бит	4	4
Def	0	0

SRC	SH	N	ALU	CCX
3	4	4	4	2
1	0	0	6	0

DST
3
0

CONST
16
0

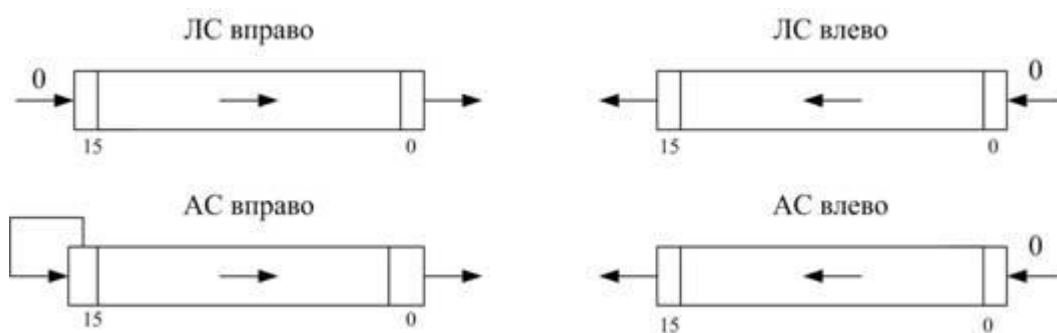


Поле SH	Операция
0	Без сдвига
1	АС АЛУ вправо
2	ЛС АЛУ вправо
3	АС АЛУ, RGQ вправо
4	ЛС АЛУ, RGQ вправо
5	ЛС RGQ вправо
6	RGQ ÷ ALU
8	ЛС АЛУ влево
A	ЛС АЛУ, RGQ влево
E	Расширение знака

Различают 2 вида сдвига – Логический (ЛС) и Арифметический (АС).

При арифметическом сдвиге вправо, все новые (появившиеся слева) биты будут равны самому левому (старшему) биту до сдвига.

А при логическом сдвиге все новые биты всегда равны нулю.

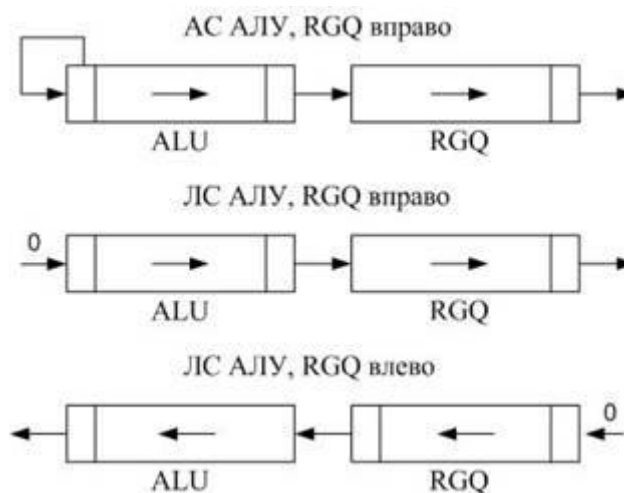


Арифметические сдвиги, к примеру, используются в тех случаях, когда записанный код является числом со знаком.

Также сдвиг может происходить не только в регистре **ALU**, но и в регистре **RGQ** или в них обоих.

При совместном (двойном) сдвиге эти два регистра как бы объединяются в один 32-ух разрядный.

Пример совместного сдвига:



В дальнейшем значение регистра **RGQ** можно отправить, как второй операнд с помощью поля **SCR**.

Так же в поле **SH** есть ещё 2 команды:

- 1) **RGQ v ALU** – копирование значения из **ALU** в **RGQ**
- 2) «расширение знака»

В ходе этой операции левые 8 бит регистра **ALU** станут равными 8-ому биту справа этого же регистра.

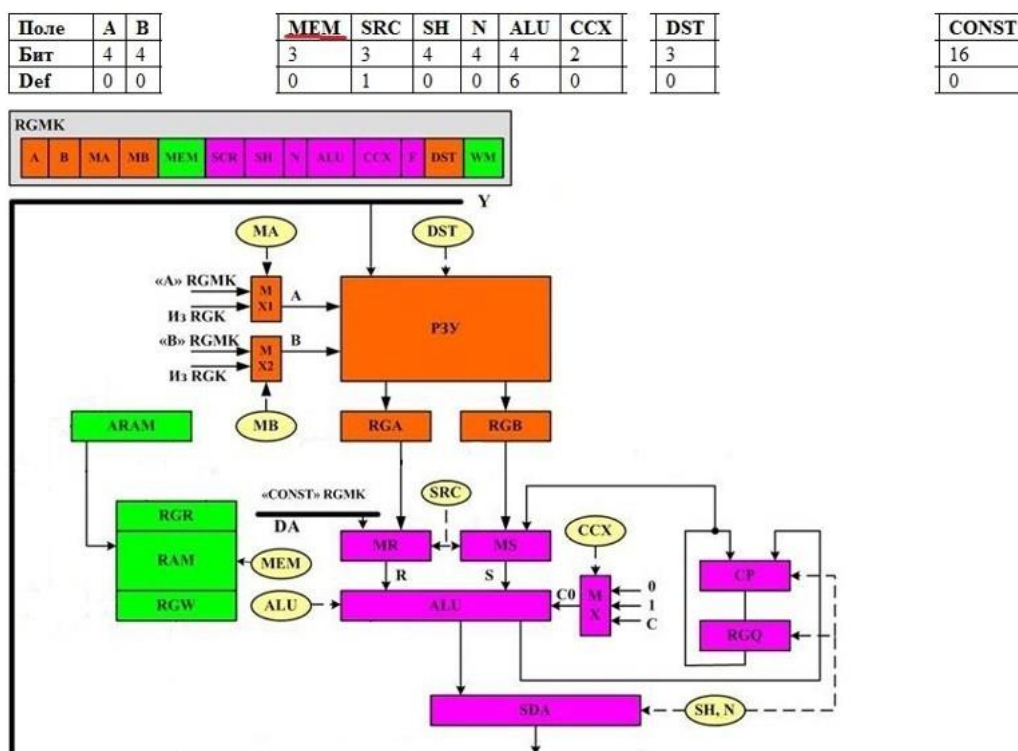
VI. Далее перейдём к изучению работы с памятью. (всё что к ней относится обозначено зелёным)

К памяти относятся регистры **RGR**, **RGW**, **ARAM** и сама оперативная память **RAM**.

RGW – это регистр из которого происходит запись в оперативную память.

RGR – соответственно регистр в который происходит считывание из оперативной памяти.

А в регистре **ARAM** хранится номер ячейки памяти, с которой будут проходить все эти операции.



Поле MEM	Значение
0-3	NOP
4	Чтение байта
5	Чтение слова
6	Запись байта
7	Запись слова

За запись и чтение оперативки отвечает поле **MEM**, оно состоит из 3 бит.

Первый бит – признак обращения к памяти (если он равен 0, то с памятью ничего не происходит)

Второй – определяет режим чтение/запись (0 – чтение, 1 – запись)

Третий – размер данных (0 – байт, 1 – слово)

VIII. За одну команду не получится и считать значение из **RGR**, и отправить её как второй операнд (**S**).

Теперь перейдём к циклам, переходам и подпрограммам.

За циклы, переходы к подпрограмме и т.д. всецело отвечает поле **СНА**, остальные же 3 поля отвечают лишь за проверку условия перехода.

1) **CJS** - переход к подпрограмме. Адрес следующей МК запоминается в стеке и происходит переход по адресу подпрограммы, заданному в поле **CONST**.

3) **CJP** - переход по адресу из поля **CONST**.

4) **RPCT** - повторение цикла. Если **RACT** не равен нулю, из него вычитается 1 и выполняется переход по адресу, расположенному в поле **CONST**. Если **RACT=0**, следующая микрокоманда выбирается в естественном порядке.

5) **CRTN** - условный возврат из подпрограммы. Переход по адресу из стека и декремент **STP**. (это регистр который хранит число значений в стеке)

6) **LDCT** - загрузка счетчика. Из поля **CONST** в **RACT** загружаются адрес или число циклов.

7) **CONT** - продолжить выборку микрокоманд в естественном порядке. Этот код устанавливается по умолчанию.

Далее рассмотрим, как осуществляется проверка перехода к выполнению поля **СНА**.

Поле **JFI** состоит из 3 бит.

Первый бит – это бит безусловного перехода. Если он равен нулю, то сразу происходит выполнение поля **СНА**.

В противном случае за дело берётся поле **СС**. Оно определяет, выполнить ли действие в **СНА** относительно указанного регистра флажков.

Существует 2 регистра флажков:

RFI – это регистр, значение которого обновляется каждую микрокоманду. Оно зависит от значения расположенного в регистре **ALU** (до сдвига).

RFD – это регистр, значение которого обновляется только если в поле микрокоманды **F** записана 1.

Для того чтобы указать, какой из двух регистров использовать нужен второй бит поля **JFI**. Если он равен 0, то **RFI**, если 1, то **RFD**.

Третий бит поля **JFI**, если он равен 1, инвертирует конечный результат (выполнить поле **СНА** или нет)

Команда 101 поля **JFI**, означает **STOP**.

В поле **СС** указываем флажок, по которому будет проверяться, выполнится ли поле **СНА** или нет.

1- да

0-нет

Так же можно указать не один флажок, а какую-нибудь их комбинацию.

Флажки:

N - знак минус (старший (самый левый) бит)

Z - признак нуля

V - признак

C - перенос из старшего бита (если в ходе арифметической операции самый левый бит не поместился)

P – признак паритета (нечетное число единиц в результате)

Поле СС	Условие перехода
0	$P=1$
1	$Z=1$
2	$N=1$
3	$V=1$
4	$C=1$
5	$N \oplus V=1$
6	$Z \vee (N \oplus V)=1$
7	$C \vee Z=1$