Результаты опытов на реальной ЭВМ «Электроника Д3-28»

Уважаемая Наталия **xlat** – участница форума «Полигон Призраков», – предложила помощь в изучении ЭВМ «Электроника Д3-28», и в период с 26-10-2019 по 29-11-2019 выполнила более сорока опытов на восстановленной ею реальной машине Д3-28. В опытах выявляются подробности ряда команд этой ЭВМ.

Условия экспериментов Наталия предложила формулировать следующим образом: «достаточно листинга (желательно со значением КП) и инструкций в случае нестандартного запуска (если требуется предварительная инициализация регистров или других ячеек ОЗУ, запуск не с ВР, результат не в X/Y и подобное)».

Соответственно, описание опытов строилось в форме листинга по стандарту программы распечатки «Д8». Т. е. в строках описания экспериментальной программки указывались: пятизначный десятичный номер шага, записанные через пробел тетрады кода команды, мнемоника. Для менее подготовленного читателя и для себя я (Синус :-) добавил в описание опытов пояснение их цели и результирующие выводы.

Опыт 1:

Это попытка узнать, различает ли машина числа –0 и +0 в команде условного перехода 11 03 B2 A2 (BSAZ R_i .+e) по условию "содержимое R-регистра равно нулю". На шаге 0 мы заносим в регистр X число 1. На шагах 1–4 заносим в регистр R8 (состоящий из однобайтовых регистров S0 и S1) двухбайтовое число $(08\ 00\ \ 00\ 00)_{16}$ с двоичным кодом $(10000000\ \ 00000000)_2$. Если машина интерпретирует это число как "число со знаком", то единица в его старшем бите означает знак "минус", и тогда это число есть "–0".

На шагах 5–6 исполняется команда условного перехода "если R8 равно нулю, то переходим на B2+1=3 шага вперёд", т. е. к шагу 5+3=8; там в X заносится число 2, и затем идёт команда END. В этом случае после останова по команде END на X-табло отобразится число 2. Если же машина решит, что число "–0" не равно нулю, то она выполнит шаг 7 с командой STOP, и тогда на X-табло отобразится число 1.

```
00000 07 01          DIG 1
00001 13 00  08 00   MOV #08 00,S00
00003 13 01  00 00   MOV #00 00,S01
00005 11 03  02 08   BSAZ R08 00008
00007 05 15           STOP
00008 07 02           DIG 2
00009 05 12           END
```

КП: 96

Результат опыта: X = 1.

Вывод: в команде 11 03 B2 A2 (т. е. BSAZ R_i .+e) число "–0" не считается равным нулю.

Опыт 2:

Посмотрим, пересылает ли машина знак минус в регистр X из R-регистра, когда там "–0". Начало здесь такое же, как в "Опыте 1". На шаге 5 тетрады регистра R8 = "–0" пересылаются в X "как цифры", без перевода из шестнадцатеричной в десятичную систему счисления. Надо посмотреть, что покажет табло X после останова: –0 или просто 0?

```
00000 07 01          DIG 1
00001 13 00  08 00  MOV #08 00,S00
00003 13 01  00 00  MOV #00 00,S01
00005 04 13  03 08  MOV R08,X
00007 05 12          END
```

КП: 71

Результат: X = –0.

Опыт 3:

Почти такой же опыт, как предыдущий, но только на шаге 5 здесь применена команда пересылки из R8 в X с переводом в десятичную систему. Надо посмотреть, что покажет табло X после останова: –0 или просто 0?

```
00000 07 01          DIG 1
00001 13 00  08 00  MOV #08 00,S00
00003 13 01  00 00  MOV #00 00,S01
00005 04 13  04 08  MOVD R08,X
00007 05 12          END
```

КП: 72

Результат: X = –0.

Вывод: бит знака R-регистра учитывается при пересылках числа "–0" в X-регистр командами MOV R_i, X и MOVD R_i, X, т. е. нули "–0" и "+0" в этих операциях различаются.

Опыт 4:

Это попытка разобраться, что означают слова из *Справочника программиста* (стр. 53): "содержимое R-регистров рассматривается как код" в командах прибавления и вычитания константы. Здесь шаги 1–4 такие же, как в предыдущем опыте; в регистр R8 заносим "–0". А на шаге 5 "прибавляем к содержимому R8 единицу". Что должно получиться? Если содержимое R8 интерпретируется просто как беззнаковый двухбайтовый код, то получится код, в двоичном виде равный 10000000 00000001, который при пересылке в X станет числом со знаком минус: –1. Если же машина прибавляет единицу к R8 как "к числу –0 по правилам алгебры", то можно ожидать ответ –0+1=1. Надо посмотреть, что покажет табло X после останова: –1 или 1? Желательно и прямо посмотреть R8.

```
00000 07 01          DIG 1
00001 13 00  08 00  MOV #08 00,S00
00003 13 01  00 00  MOV #00 00,S01
00005 10 00  01 08  ADD #01,R08
00007 04 13  03 08  MOV R08,X
00009 05 12          END
```

КП: 90

Результат: X = –1, R8 = 08 00 00 01 (в 16-ричной системе в тетрадно-десятичной форме).

Вывод: в командах ADD #e, R_i / SUB #e, R_i машина прибавляет / вычитает константу не "по правилам алгебры чисел со знаком", а работает с содержимым R_i как с беззнаковым двухбайтным числом.

Опыт 5:

Проверяем, возможен ли вывод на индикацию и ввод отрицательного номера шага. На шагах 1–4 в регистр R9 загружаем число с десятичным значением 32. На шаге 5 оно пересылается в регистр BP, а на следующем шаге выполняется STOP, при этом регистр PC будет содержать адрес следующей команды, т. е. число 8. Номер шага по определению равен разности PC – BP; в данном примере эта разность равна –24. Вопросы:

а) после останова этой программки надо перейти в режим "B" и посмотреть, будет ли на X-табло виден отрицательный номер шага –24?

б) к какому номеру шага теперь приводит нажатие кнопки "C"?

в) возможно ли вводить вручную (после нажатия кнопки "НШ") отрицательный номер шага?

```
00000 07 01          DIG 1
00001 13 02  00 00   MOV #00 00,S02
00003 13 03  02 00   MOV #02 00,S03
00005 04 13  06 09   MOV R09,BP
00007 05 15          STOP
00008 05 12          END
```

КП: 93

Результаты (ответы на вопросы *а*, *б*, *в*):

а) номер шага: –24,
б) шаг = 0, PC = BP = 00 00 02 00 (в 16-ричной системе в тетрадно-десятичной форме),
в) пояснения уважаемой Наталии **xlat** о вводе номера шага в режиме "B":

При вводе номера шага по НШ фактически в качестве цифры вводится младшая тетрада кода нажатой клавиши. Двоично-десятичная коррекция при этом не выполняется – тетрада просто домножается на вес текущей десятичной позиции и складывается с накопленным результатом. Ввод тетрады 15 возможен только в последней позиции (в других местах нажатие просто игнорируется).

Например, набрав "0", "0", "e^x", "lg x", "5" получим шаг 1505 = (14·100 + 10·10 + 5). Набрав "0", "0", "0", "1", "1/x" получим шаг 25 = (1·10 + 15).

Тетрады с 10 по 15 декодируются блоком индикации следующим образом:

10 → "8"
11 → "9"
12 → "8"
13 → "9"
14 → "8"
15 → "_" (пустой разряд)

Таким образом, ввод отрицательного шага с помощью НШ невозможен в принципе.

Опыт 6:

В этом и в следующем опытах проверяется вывод о том, что слова «как код» применительно к содержимому R-регистра означают «как беззнаковое двухбайтовое число». Занесём в R8 два байта $(15\ 15\ \ 15\ 15)_{16}$, так что все 16 двоичных разрядов регистра R8 будут равны 1; и прибавим 1. Если машина работает с R8 как с беззнаковым числом, то во всех 16 двоичных разрядах образуются нули. А если машина думает, что в R8 было отрицательное число $(–07\ 15\ 15\ 15)_{16}$, то по правилам алгебры чисел со знаком должно получиться на 1 большее число, т. е. –07 15 15 14; в десятичном виде это есть –32766. Надо посмотреть, что покажет X-табло после останова: 0 или –32766?

```
00000 07 06          DIG 6
00001 13 00  15 15  MOV #15 15,S00
00003 13 01  15 15  MOV #15 15,S01
00005 10 00  01 08  ADD #01,R08
00007 04 13  04 08  MOVD R08,X
00009 05 12          END
```

КП: 148

Результат: X = 0.

Опыт 7:

Заносим в R8 число 1, вычитаем 2, и смотрим (через регистр X) результат. Если машина действует здесь с R8 "как с числом со знаком", то должно получиться 1–2 = –1. А если "как с беззнаковым кодом", то в R8 запишутся байты 15 15 15 15, в десятичном виде означающие число $–32767_{10}$.

```
00000 07 01          DIG 1
00001 04 13  12 08  MOVH X,R08
00003 10 01  02 08  SUB #02,R08
00005 04 13  04 08  MOVD R08,X
00007 05 12          END
```

КП: 95

Результат: X = –32767.

Вывод: подтверждено, что в командах ADD #e, R_i / SUB #e, R_i машина прибавляет / вычитает константу не "по правилам алгебры чисел со знаком", а работает с содержимым R_i как с беззнаковым двухбайтным числом.

В следующей серии опытов разбираемся с условными переходами 14 08 B2 A2 (SOBZ R_j .+e) и 14 09 B2 A2 (ABGE R_i, R_j). В этих командах сначала изменяется на единицу содержимое R_j, а затем в зависимости от результата может выполняться переход. В *Справочнике программиста* не сказано, по каким правилам здесь выполняется вычитание или прибавление единицы в R-регистре. Как к беззнаковому двухбайтовому коду? Или как к беззнаковому адресу в пределах 0 ... 32767_{10}? Или как к числу со знаком? В *Справочнике программиста* сказано, что в этих командах включается

индикация ОП ("Ошибка Программы") при отрицательном содержимом R-регистров, но не сказано – в случае с ОП остаётся в R-регистре исходное содержимое или изменённое?

Опыт 8:

Изучаем команду SOBZ R_j .+e. Инициализируем R8 нулём; тогда при вычитании из него единицы условие перехода (R8 = 0) не выполнится (но на всякий случай заносим в Y единицу для индикации перехода). Из-за отрицательности R8 может появиться сигнал ОП. В этом случае ОП гасится командой 0510, и выполняется переход от шага 6 к шагу 9; при этом будет Y = 0, а иначе получим Y = 1. Итоговое значение R8 смотрим на X-табло: если машина здесь действует с R8 как с "числом со знаком", то получим X = –1, а если как с "беззнаковым двухбайтным кодом", то будет X = –32767.

Для начального сброса X и Y в ноль надо перед запуском программок нажимать "C"; запуск: "S".

```
00000 05 14          GO
00001 05 14          GO
00002 04 13  10 08   CLR R08
00004 14 08  02 08   SOBZ R08 00007
00006 05 10          BPER 00009
00007 07 01          DIG 1
00008 06 04          MOV X,Y
00009 04 13  04 08   MOVD R08,X
00011 05 12          END
```

КП: 167

Результат: X = –32767, Y = 0.

Опыт 9:

Это почти такой же опыт, как предыдущий, но только засылаем в R8 число "–0", а не 0; тогда при вычитании из него 1 "как из беззнакового кода" получим $(0715\ \ 1515)_{16} = 32767_{10}$. Условие перехода (R8 = 0) здесь в любом случае не должно выполняться. Если исходный минус в R8 не вызывает ОП, то на шаге 6 команда BPER не совершит перехода к шагу 9, и тогда получим Y = 1.

```
00000 13 00  08 00   MOV #08 00,S00
00002 13 01  00 00   MOV #00 00,S01
00004 14 08  02 08   SOBZ R08 00007
00006 05 10          BPER 00009
00007 07 01          DIG 1
00008 06 04          MOV X,Y
00009 04 13  04 08   MOVD R08,X
00011 05 12          END
```

КП: 129

Результат: X = 32767, Y = 1

Опыт 10:

Здесь почти всё то же самое, но только инициализируем R8 числом "–1". Тогда при вычитании из него 1 "как из беззнакового кода" получится "–0", перехода к шагу 7 не будет. Отрицательность результата в R8 вызовет ОП с переходом от шага 6 к шагу 9, так что Y останется равным нулю.

```
00000 13 00  08 00  MOV #08 00,S00
00002 13 01  00 01  MOV #00 01,S01
00004 14 08  02 08  SOBZ R08 00007
00006 05 10          BPER 00009
00007 07 01          DIG 1
00008 06 04          MOV X,Y
00009 04 13  04 08  MOVD R08,X
00011 05 12          END
```

КП: 130

Результат: X = –0, Y = 0.

Вывод: по команде 14 08 B2 A2 (т. е. SOBZ R_j .+e) декрементацию $R_j \leftarrow R_j - 1$ машина выполняет "как с кодом", т. е. как с беззнаковым 2-байтовым числом (и результат сохраняется, несмотря на ОП). В *Справочнике программиста* этот факт пропущен, причём допущена неточность: опыт 9 показал, что в команде SOBZ R_j .+e проверяется "на отрицательность" только результат, а не исходное содержимое R-регистра.

Опыт 10-штрих, тривиальный:

Это контрольный опыт, с заранее известным результатом: заменим в "Опыте 10" байт 08 00 на шаге 1 байтом 00 00; тем самым задаётся начальное значение R8 = 1, и тогда при любой интерпретации содержимого R8 (как "числа со знаком" или как "беззнакового кода") после останова должно получиться X = 0, Y = 1.

```
00000 13 00  00 00  MOV #08 00,S00
00002 13 01  00 01  MOV #00 01,S01
00004 14 08  02 08  SOBZ R08 00007
00006 05 10          BPER 00009
00007 07 01          DIG 1
00008 06 04          MOV X,Y
00009 04 13  04 08  MOVD R08,X
00011 05 12          END
```

КП: 122

Результат: X = 0, Y = 1.

Теперь изучим команду условного перехода 14 09 B2 A2, т. е. ABGE R_i, R_j. Здесь сначала к содержимому R_j прибавляется единица ($R_j \leftarrow R_j + 1$), а затем проверяется выполнение условия перехода (на 4 шага вперёд) по содержимому R-регистров: "$R_i \geq R_j$". В *Справочнике программиста* сказано, что "включается индикатор ОП, если содержимое анализируемых регистров отрицательное".

Можно было бы задать 4 вопроса. По какому правилу прибавляется единица $R_j \leftarrow R_j + 1$: как для чисел со знаком или как для беззнаковых кодов? По какому правилу проверяется условие "$R_i \geq R_j$"? Препятствует ли "отрицательность" переходу? В случае с "отрицательностью" сохраняется ли в R_j изменённое содержимое? Ниже из результатов опытов мы увидим, что "отрицательность" R-регистров препятствует переходу. Значит, проверка условия $R_i \geq R_j$ актуальна только для неотрицательных чисел в R-регистрах; при этом исчезает различие правил "для чисел со знаком" и "для беззнаковых кодов", так что вопрос о правиле проверки $R_i \geq R_j$ отпадает.

Ниже в программках инициализируются два R-регистра: R9 и R8. Гашение ОП (командой 05 10) здесь для простоты не применяем. Поэтому после останова с ОП табло X будет мигать; это мигание не должно помешать увидеть результат на табло X и Y.

Опыт 11:

В опыте 11 начальные значения R-регистров выбраны так: R9 = R8 = $(10000000\ 00000001)_2$. Если прибавление единицы к R8 происходит "как для беззнакового кода", то получаем R8 $\leftarrow$ $(10000000\ 00000010)_2$, то есть: R9 = "–1", R8 = "–2". В случае прибавления единицы по правилам алгебры "чисел со знаками" можно ожидать результат R8 $\leftarrow$ –1+1 = 0. Для различения этих случаев содержимое R8 на шаге 12 пересылаем в X (с переводом в десятичную систему).

Если исследуемая команда (на шаге 8) вызывает переход к шагу 12, то сохранится начальное значение Y = 0; если же перехода нет, то на шагах 10–11 в Y занесётся 1.

Надо посмотреть, что покажут после останова X-табло и Y-табло.

```
00000 13 00  08 00  MOV #08 00,S00
00002 13 01  00 01  MOV #00 01,S01
00004 13 02  08 00  MOV #08 00,S02
00006 13 03  00 01  MOV #00 01,S03
00008 14 09  09 08  ABGE R09,R08 00012
00010 07 01          DIG 1
00011 06 04          MOV X,Y
00012 04 13  04 08  MOVD R08,X
00014 05 12          END
```

КП: 163

Результат: X = –2, Y = 1. Включен индикатор ОП, табло X мигает.

Вывод: в команде ABGE R_i, R_j прибавление единицы $R_j \leftarrow R_j + 1$ происходит "как для беззнакового кода", изменённое значение R_j сохраняется и в случае с сигналом ОП.

Опыт 12:

В этом опыте начальные значения R-регистров выбраны так: R9 = (00000000 00000000)$_2$ = "+0" , R8 = (10000000 00000000)$_2$ = "–0" . Рассуждения аналогичны предыдущим. Надо посмотреть, что покажут после останова X-табло и Y-табло.

```
00000 13 00  08 00  MOV #08 00,S00
00002 13 01  00 00  MOV #00 00,S01
00004 13 02  00 00  MOV #00 00,S02
00006 13 03  00 00  MOV #00 00,S03
00008 14 09  09 08  ABGE R09,R08 00012
00010 07 01          DIG 1
00011 06 04          MOV X,Y
00012 04 13  04 08  MOVD R08,X
00014 05 12          END
```

КП: 153.

Результат: X = –1, Y = 1. Включен индикатор ОП, табло X мигает

Опыт 13:

В этом опыте начальные значения R-регистров выбраны так: R9 = (00000000 00000000)$_2$ = "+0" , R8 = (07 15 15 15)$_{16}$ = (01111111 11111111)$_2$ = 32767$_{10}$. Рассуждения аналогичны предыдущим. Надо посмотреть, что покажут после останова X-табло и Y-табло.

```
00000 13 00  07 15  MOV #07 15,S00
00002 13 01  15 15  MOV #15 15,S01
00004 13 02  00 00  MOV #00 00,S02
00006 13 03  00 00  MOV #00 00,S03
00008 14 09  09 08  ABGE R09,R08 00012
00010 07 01          DIG 1
00011 06 04          MOV X,Y
00012 04 13  04 08  MOVD R08,X
00014 05 12          END
```

КП: 197

Результат: X = –0, Y = 1. Включен индикатор ОП, табло X мигает.

Таким образом, подтверждён вывод о том, что в команде ABGE R_i , R_j прибавление единицы $R_j \leftarrow R_j + 1$ происходит "как для беззнакового кода", изменённое значение R_j сохраняется, несмотря на сигнал ОП.

Опыт 14, контрольный:

Это опыт с заранее известным результатом. При начальных значениях R9 = "1", R8 = "0", после "прибавления единицы" к R8 получится в любом случае R8 = R9 = "1", так что условие перехода от шага 8 к шагу 8+4=12 будет выполнено, и поэтому в результате мы получим X = 1, Y = 0:

```
00000 13 00  00 00  MOV #00 00,S00
00002 13 01  00 00  MOV #00 00,S01
00004 13 02  00 00  MOV #00 00,S02
00006 13 03  00 01  MOV #00 01,S03
00008 14 09  09 08  ABGE R09,R08 00012
00010 07 01          DIG 1
00011 06 04          MOV X,Y
00012 04 13  04 08  MOVD R08,X
00014 05 12          END
```

КП: 146

Результат: X = 1, Y = 0.

Теперь выясним, не препятствует ли "отрицательность" R8 и / или R9 переходу по выполненному условию в команде ABGE R_i, R_j. В примерах ниже условие перехода (R9 ≥ R8) выполнено (в том или ином смысле), и при этом есть "отрицательность", так что может возникнуть сигнал ОП (и табло X будет мигать). Если машина выполняет переход, то получим Y = 0; а если машина не выполняет перехода, получим Y = 1.

Опыт 15:

В этом опыте начальные значения R-регистров выбраны так: R9 = $(00000000\ 00000000)_2$ = "+0", R8 = $(15\ 15\ 15\ 15)_{16} = (11111111\ 11111111)_2 = -32767_{10}$. После "прибавления единицы" к R8 получится R9 = R8 = "+0", так что условие перехода (R9 ≥ R8) выполнится в форме равенства.

```
00000 13 00  15 15  MOV #15 15,S00
00002 13 01  15 15  MOV #15 15,S01
00004 13 02  00 00  MOV #00 00,S02
00006 13 03  00 00  MOV #00 00,S03
00008 14 09  09 08  ABGE R09,R08 00012
00010 07 01          DIG 1
00011 06 04          MOV X,Y
00012 04 13  04 08  MOVD R08,X
00014 05 12          END
```

КП: 205

Результат: X = 0 (мигает), Y = 1. Включен индикатор ОП.

Опыт 16:

В этом опыте после "прибавления единицы" к R8 получается R9 = R8 = "–0", условие перехода (R9 ≥ R8) выполняется как равенство (верное и для "беззнаковых кодов" и для "чисел со знаком").

```
00000 13 00  07 15  MOV #07 15,S00
00002 13 01  15 15  MOV #15 15,S01
00004 13 02  08 00  MOV #08 00,S02
00006 13 03  00 00  MOV #00 00,S03
00008 14 09  09 08  ABGE R09,R08 00012
```

```
00010 07 01          DIG 1
00011 06 04          MOV X,Y
00012 04 13  04 08   MOVD R08,X
00014 05 12          END
```

КП: 205

Результат: X = –0 (мигает), Y = 1. Включен индикатор ОП.

Опыт 17:

В этом опыте отрицательным является R9 = "–0". Инкрементируемый регистр R8 содержит неотрицательные числа: "+0" до и "1" после прибавления единицы. Условие перехода (R9 ≥ R8) здесь выполнено только в смысле "беззнаковых кодов", для "чисел со знаком" оно не выполнено.

```
00000 13 00  00 00  MOV #00 00,S00
00002 13 01  00 00  MOV #00 00,S01
00004 13 02  08 00  MOV #08 00,S02
00006 13 03  00 00  MOV #00 00,S03
00008 14 09  09 08  ABGE R09,R08 00012
00010 07 01          DIG 1
00011 06 04          MOV X,Y
00012 04 13  04 08   MOVD R08,X
00014 05 12          END
```

КП: 153.

Результат: X = 1, Y = 1.

Вывод: здесь "отрицательность" константы (в регистре R9) не вызвала сигнала ОП. Причиной отсутствия перехода (по условию R9 ≥ R8) можно считать указанную "отрицательность" или невыполнение условия перехода "для чисел со знаком".

Опыт 18:

Здесь все числа отрицательные: R9 = "–2", регистр R8 содержит "–0" до и "–1" после прибавления единицы. Условие перехода выполнено для "беззнаковых кодов", а для "для чисел со знаком" – нет.

```
00000 13 00  08 00  MOV #08 00,S00
00002 13 01  00 00  MOV #00 00,S01
00004 13 02  08 00  MOV #08 00,S02
00006 13 03  00 02  MOV #00 02,S03
00008 14 09  09 08  ABGE R09,R08 00012
00010 07 01          DIG 1
00011 06 04          MOV X,Y
00012 04 13  04 08   MOVD R08,X
00014 05 12          END
```

КП: 163

Результат: X = –1 (мигает), Y = 1.

Из результатов этой серии опытов можно сделать вывод, что "отрицательность" чисел в любом из двух R-регистров препятствует переходу по условию $R_i \geq R_j$ при исполнении команды 14 09 i j (ABGE R_i, R_j).

Ниже – серия опытов, в которой мы выясняем правило проверки условия перехода (на 4 шага вперёд) " $R_i \geq R_j$ " в команде условного перехода 14 10 i j (BGE R_i, R_j). В отличие от предыдущей команды условного перехода здесь содержимое R-регистров не изменяется, сигнал ОП не включается.

Если машина выполняет переход, то программка сохраняет начальное значение X = Y = 0. Если переход не выполняется, то на шагах 10–11 устанавливается значение X = Y = 1.

Опыт 19:

В этом опыте R8 = "+0", R9 = "–0". Условие перехода R9 ≥ R8 выполнено для "беззнаковых кодов" и не выполнено для "чисел со знаком".

```
00000 13 00  00 00  MOV #00 00,S00
00002 13 01  00 00  MOV #00 00,S01
00004 13 02  08 00  MOV #08 00,S02
00006 13 03  00 00  MOV #00 00,S03
00008 14 10  09 08  BGE R09,R08 00012
00010 07 01          DIG 1
00011 06 04          MOV X,Y
00012 05 12          END
```

КП: 125

Результат: X = 1, Y = 1, т. е. переход не выполнен.

Опыт 20:

Почти такой же опыт, только для контроля поменялись местами регистры R8 и R9 на шаге 8. Условие перехода R8 ≥ R9 не выполнено для "беззнаковых кодов" и выполнено для "чисел со знаком".

```
00000 13 00  00 00  MOV #00 00,S00
00002 13 01  00 00  MOV #00 00,S01
00004 13 02  08 00  MOV #08 00,S02
00006 13 03  00 00  MOV #00 00,S03
00008 14 10  08 09  BGE R08,R09 00012
00010 07 01          DIG 1
00011 06 04          MOV X,Y
00012 05 12          END
```

КП: 125

Результат: X = 0, Y = 0, т. е. переход выполнен.

Пара аналогичных опытов с ненулевыми значениями в R-регистрах:

Опыт 21:

В этом опыте R8 = "–32766", R9 = "–32767". Условие перехода R8 ≥ R9 не выполнено для "беззнаковых кодов" и выполнено для "чисел со знаком".

```
00000 13 00  15 15  MOV #15 15,S00
00002 13 01  15 14  MOV #15 14,S01
00004 13 02  15 15  MOV #15 15,S02
00006 13 03  15 15  MOV #15 15,S03
00008 14 10  08 09  BGE R08,R09 00012
00010 07 01          DIG 1
00011 06 04          MOV X,Y
00012 05 12          END
```

КП: 236

Результат: X = 0, Y = 0, т. е. переход выполнен.

Опыт 22:

В этом опыте R8 = "–2", R9 = "1". Условие перехода R8 ≥ R9 выполнено для "беззнаковых кодов" и не выполнено для "чисел со знаком".

```
00000 13 00  08 00  MOV #08 00,S00
00002 13 01  00 02  MOV #00 02,S01
00004 13 02  00 00  MOV #00 00,S02
00006 13 03  00 01  MOV #00 01,S03
00008 14 10  08 09  BGE R08,R09 00012
00010 07 01          DIG 1
00011 06 04          MOV X,Y
00012 05 12          END
```

КП: 128

Результат: X = 1, Y = 1, т. е. переход не выполнен.

Вывод: в ходе проверки условия $R_i \geq R_j$ по команде 14 10 i j (т. е. BGE R_i, R_j) содержимое R-регистров машина интерпретирует как "числа со знаком".

Далее – серия опытов, в которых проверяется предположение о сохранности (неизменности) бита знака в R-регистрах в тех командах, в которых изменяемое содержимое R-регистра несёт смысл адреса ячеек ОЗУ. В таких командах бит знака, по идее, не должен играть роли, так что в случае его сохранности программисты могут использовать этот бит (старший бит R-регистра) отдельно от адресных битов для каких-либо своих целей.

Опыт 23:

Здесь на шагах 0–3 в регистр R1 загружается отрицательное число "–1", т. е. старший бит регистра R1 устанавливается равным 1, а остальные 15 бит могут интерпретироваться как равный 1 адрес ячейки ОЗУ. На шаге 4 команда NSN 07 11 начиная с указанного в R1 адреса ищет первый не равный 07 11 код, записывает его в регистр S3, а в R1 записывает адрес кода, следующего за найденным, – в нашем примере этот адрес равен 3.

Таким образом, если бит знака сохраняется, то в данном примере получим отрицательное число R1 = "–3", а если значением 3 обновляются все 16 бит регистра R1, то получим положительное число R1 = "3". На шаге 6 число из R1 пересылается в X. Надо посмотреть, отрицательное или положительное число отобразится на X-табло.

```
00000 07 01          DIG 1
00001 07 11          NEG X
00002 04 13  12 01   MOVH X,R01
00004 10 04  07 11   NSN 07 11
00006 04 13  04 01   MOVD R01,X
00008 05 12          END
```

КП: 110

Результат: X = –3. Вывод: команда 10 04 B2 A2 (т. е. NSN d) не изменяет бит знака в R1.

Опыт 24:

Это аналогичный предыдущему опыт с командой NSS d. Команда NSS d аналогична команде NSN d, но ищет код, равный заданному d. В нашем примере в R1 запишется адрес "2"; получим R1 = "–2", если бит знака сохраняется, либо получим R1 = "2", если бит знака обновляется. Надо посмотреть, отрицательное или положительное число отобразится на X-табло.

```
00000 07 01          DIG 1
00001 07 11          NEG X
00002 04 13  12 01   MOVH X,R01
00004 10 05  07 11   NSS 07 11
00006 04 13  04 01   MOVD R01,X
00008 05 12          END
```

КП: 111

Результат: X = –2. Вывод: команда 10 05 B2 A2 (т. е. NSS d) не изменяет бит знака в R1.

Опыт 25:

Это аналогичный опыт с командой ATOI d. Команда ATOI 05 15 (см. *Справочник программиста*, стр. 58) в нашем примере должна установить R1 на адрес "11".

```
00000 07 08          DIG 8
00001 07 11          NEG X
00002 04 13  12 01   MOVH X,R01
00004 10 03  05 15   ATOI 05 15
```

```
00006 04 13  04 01  MOVD R01,X
00008 05 15          STOP
00009 03 00          это код цифры 0
00010 05 14          GO
00011 05 12          END
```

КП: 160

Результат: X = –11. Вывод: команда 10 03 B2 A2 (т. е. ATOI d) не изменяет бит знака в R1.

Пара опытов для проверки предположения о сохранности бита знака в R-регистре, указывающем вершину стека:

Опыт 26:

Для начального сброса X и Y в ноль надо перед запуском программок нажимать "C"; запуск: "S". При нажатии "C" в указатель стека R13 записывается адрес 07 13 00 00 = 32000_{10}.

На шаге 0 изменяется знак числа в R13. На шаге 2 вызывается подпрограмма, при этом адрес в R13 уменьшается на 2. Таким образом, если бит знака в R13 сохраняется, то получим R13 = "–31998", а иначе R13 = "31998". В подпрограмме число из R13 пересылается в X и в Y. После возврата из подпрограммы число из R13 пересылается в X и выполняется останов (с индикацией X и Y на табло). Надо посмотреть: какие числа отобразятся на табло?

```
00000 04 13  09 13  NEG R13
00002 00 01          JSM 00 01
00003 04 13  04 13  MOVD R13,X
00005 05 15          STOP
00006 04 08  00 01  MARK 00 01
00008 04 13  04 13  MOVD R13,X
00010 06 04          MOV X,Y
00011 05 11          RTS
00012 05 12          END
```

КП: 167

Результат: X = –32000, Y = –31998.

Вывод: при вызове подпрограммы в ходе обновления указателя стека его бит знака не изменяется.

Опыт 27:

Здесь на шаге 6 в «самодельный» стек с указателем в R8 засылается R9; на шаге 11 содержимое регистра R9 восстанавливается из этого стека. Начальное значение R8 = "–32" по ходу этих действий должно обновляться до "–30" (сохраняем его в Y) и затем снова до "–32" (сохраняем его в X), если бит знака в R8 не изменяется. Если же бит знака в R8 сбрасывается в ноль, то получим Y = 30, X =32.

```
00000 13 00  08 00  MOV #08 00,S00
00002 13 01  02 00  MOV #02 00,S01
00004 04 13  10 09  CLR R09
00006 10 12  09 08  MOV R09,-(R08)
```

```
00008 04 13  04 08  MOVD R08,X
00010 06 04          MOV X,Y
00011 10 15  09 08  MOV (R08)+,R09
00013 04 13  04 08  MOVD R08,X
00015 05 12          END
```

КП: 222

Результат: X = –32, Y = –30.

Вывод: бит знака любого указателя стека («стандартного» R13 или «самодельного») не изменяется в операциях записи в стек (MOV R_i ,–(R_j)) и чтения из стека (MOV (R_j)+, R_i).

Пара опытов для проверки возможности устанавливать бит в разряде знака регистров BD и BP:

Опыт 28:

Здесь на шаге 4 отрицательное число R8 = "–32" пересылается в регистр BD; на шаге 6 число из BD пересылается в R8 и затем в X. Если указанные пересылки не изменяют бит знака в BD, то получим X = –32.

```
00000 13 00  08 00  MOV #08 00,S00
00002 13 01  02 00  MOV #02 00,S01
00004 04 13  05 08  MOV R08,BD
00006 04 13  13 08  MOV BD,R08
00008 04 13  04 08  MOVD R08,X
00010 05 15          STOP
00011 05 12          END
```

КП: 154

Результат: X = –32.

Вывод: допускается устанавливать равным 1 бит знака индексного регистра BD.

Опыт 29:

Это аналогичный предыдущему опыт с регистром BP.

```
00000 13 00  08 00  MOV #08 00,S00
00002 13 01  02 00  MOV #02 00,S01
00004 04 13  06 08  MOV R08,BP
00006 04 13  14 08  MOV BP,R08
00008 04 13  04 08  MOVD R08,X
00010 05 15          STOP
00011 05 12          END
```

КП: 156

Результат: X = –32.

Вывод: допускается устанавливать равным 1 бит знака индексного регистра BP.

Опыт 30:

В этом опыте проверяется отсутствие нормализации в ходе пересылок содержимого 8-байтового регистра RR (т. е. R4, … , R7) в X, а также – X в 8-байтовую ячейку ОЗУ и из неё в Y. Наряду с этим смотрим, как ведёт себя машина в арифметических операциях (на примере ADD X,Y, т. е. Y ← Y + X) при ненормализованных операндах, и как работает нормализация X при "недесятичном" содержимом X, – когда в тетрадах регистра X могут присутствовать числа, превышающие 9. Для примера на шагах 19–21 значения тетрад 04, 03, 02, 01 заменяются на 12, 03, 02, 01.

Перед запуском программы (клавишей "S") следует нажать клавишу "C", чтобы сбросились в ноль X и Y. В программе семь остановов; при этом надо смотреть содержимое разрядов X- и Y-табло. После каждого из первых шести остановов следует нажимать "S" для продолжения работы программки; нажимать при этом "C" не надо.

```
00000 12 06          MOV X,RR
00001 13 00  04 03   MOV #04 03,S00
00003 13 01  02 01   MOV #02 01,S01
00005 11 04  08 05   MOV R08,R05
00007 12 07          MOV RR,X
00008 05 15          STOP              это 1-й останов
00009 06 04          MOV X,Y
00010 04 12  02 08   MOV X,(R08)
00012 05 15          STOP              это 2-й останов
00013 06 00          ADD X,Y
00014 05 15          STOP              это 3-й останов
00015 06 00          ADD X,Y
00016 05 15          STOP              это 4-й останов
00017 04 12  01 08   MOV (R08),Y
00019 04 13  09 05   NEG R05
00021 12 07          MOV RR,X
00022 05 15          STOP              это 5-й останов
00023 12 09          NORM
00024 06 00          ADD X,Y
00025 05 15          STOP              это 6-й останов
00026 13 02  13 12   MOV #13 12,S02
00028 13 03  11 10   MOV #11 10,S03
00030 11 04  09 05   MOV R09,R05
00032 12 07          MOV RR,X
00033 06 04          MOV X,Y
00034 12 09          NORM
00035 05 12          END               это 7-й останов
```

КП: 528

Результаты (указан номер останова и содержимое разрядов табло):

1)
Y = .000000000000
X = .000043210000

2)
Y = .000043210000
X = .000043210000
Вывод: нормализация при пересылках содержимого X не производится.

3)
Y = .000043210000
X = .000043210000

4)
Y = .000043210000
X = .000043210000
Вывод: при равных нулю первых разрядах мантисс X и Y сложение не производится.

5)
Y = .000043210000
X = .000083210000
Вывод: недесятичное значение 12 пятой тетрады отобразилась на X-табло цифрой "8".

6)
Y = .832100000000–04
X = .832100000000–04
Вывод: несмотря на недесятичность первого разряда мантиссы X (в данном примере там должно быть число 12), операция NORM выполнена. Операция сложения Y ← Y + X свелась к пересылке Y ← X, так как ненормализованное число в Y с равным нулю первым разрядом мантиссы, по-видимому, машина считает равным нулю.

7)
Y = .000098980000
X = .989800000000–04
Вывод: недесятичные значения тетрад 13, 12, 11, 10 в регистрах X и Y отображаются на табло цифрами "9", "8", "9", "8" (см. также пояснение (*в*) к результатам опыта 5). Операция NORM выполнена, несмотря на недесятичность всех не равных нулю тетрад.

Дополнительные вопросы, сформулированные как "опыты":

Опыт 31:

В *Справочнике программиста* на стр. 66 написано, что по команде END в PC заносится BD. Это опечатка? (Более логичным представляется занесение в PC значения BP.) Для ответа надо после останова командой END в какой-либо программке перейти в режим "Ввод" клавишей "B" и посмотреть на табло машины значения PC и номер шага.

Ответ (дословно): «PC = BP, иначе не работал бы быстрый перезапуск клавишей S после останова по END.»

Опыт 32:

Надо посмотреть в режиме "Ввод", какой "мусор" содержится в неиспользованных ячейках ОЗУ. Может быть, после включения машины все байты ОЗУ (кроме инициализированных специально) равны 15 15 ?

Ответ: «Неинициализированные области ОЗУ в Д3-28 содержат типичный для динамического ОЗУ мусор (такой же, например, видно при просмотре памяти "Радио-86РК"). Обычно, он представляет собой частично повторяющийся случайный паттерн из небольшого множества байт. Например, одна из возможных последовательностей:

15 05
15 15
15 00
05 00
05 00
05 15
05 05
05 10
15 02
15 05
15 15
15 08
05 00
… »

В продолжение опыта 30 вопросы 8), 9) и 10):

Здесь в тетрады мантиссы (с пятой по седьмую) регистра X записываются "недесятичные" данные: 15, 14, 13; в восьмую тетраду записываем допустимое (т. е. меньшее 9) число: 7.

```
00000 07 00          DIG 0
00001 12 06          MOV X,RR
00002 13 00  15 14   MOV #15 14,S00
00004 13 01  13 07   MOV #13 07,S01
00006 11 04  08 05   MOV R08,R05
00008 12 07          MOV RR,X
00009 06 04          MOV X,Y
00010 12 09          NORM
00011 05 15          STOP          останов 8)
00012 12 06          MOV X,RR
00013 04 13  04 04   MOVD R04,X
00015 05 15          STOP          останов 9)
00016 13 00  08 00   MOV #08 00,S00
00018 13 01  15 01   MOV #15 01,S01
00020 04 13  03 08   MOV R08,X
00022 05 12          END           останов 10)
```

КП: 341

Результаты (указан номер останова и содержимое разрядов табло):

8)
Y = .0000_8970000
X = ._89700000000–04
Вывод: недесятичное значение 15 в разряде мантиссы (в данном примере в первом не равном нулю) X и Y отображается на табло как погашенный разряд. Операция NORM оставляет его погашенным.

9)
Y = .0000_8970000
X = –32471.0000000
Вывод: наличие в X значений тетрад 15, 14, 13, 7 при том, что они отображались как _, 8, 9, 7, подтверждено пересылкой их в регистр R4 и выводом R4 = $(15\ 14\ 13\ 07)_{16} = -32471_{10}$ в регистр X.

10)
Y = .0000_8970000
X = –_1.0000000000
Вывод: в ходе пересылки в X командой MOV R08,X тетрад из R8 "как цифр", где R8 = 08 00 15 01, машина интерпретировала (как и ожидалось) тетраду 08 как "минус", следующую тетраду 00 машина "отнормализовала" как незначащий нуль, и затем отобразила тетрады 15 и 01 как погашенный разряд и цифру 1.

Дополнительные вопросы 11), 12) и 13):

11) Сопровождается ли нормализацией ввод чисел вручную с пульта машины ("в режиме калькулятора")? Например, что получится, если набрать число .02, переслать его клавишей "стрелка вверх" в Y и нажать клавишу "+"?

Ответ:

Поведение машины при вводе числа .02:

Нажатие "."

X = ._____

Нажатие "0"

X = ._____–01

Нажатие "2"

X = .2_____–01

Нажатие "стрелка вверх"

Y = .200000000000–01
X = .200000000000–01

Нажатие "+"

Y = .400000000000–01
X = .200000000000–01

До получения этого ответа Синус предполагал, что ввод в X числа .02 с пульта машины не сопровождается нормализацией. И думал, что тогда после пересылки X в Y команда "+" не произведёт сложения Y ← Y + X, так как для команд десятичной арифметики "+" и "–" операнды с нулевым первым разрядом мантисс машина считает равными нулю. Вопросы 12 и 13 были порождены подобными предположениями:

12) Что получится, если набрать число .02, переслать его в Y, набрать .2 и нажать клавишу "–"? (Т. е. произойдёт ли пересылка Y ← –X = –.2 в ходе операции Y ← Y–X, если нет нормализации?)

13) Производится ли нормализация при умножении? Например, что получится, если вручную набрать в X число .02, переслать его в Y, и нажать клавишу "×" ?

Ответы:

12)
Y = –.180000000000
X = .200000000000

13)
Y = .400000000000–03
X = .200000000000–01

Вывод: надо было Синусу лучше учить доки; в *Инструкции по эксплуатации* на стр. 36 написано – «если нуль вводится после запятой, перед которой не было значащих цифр, то каждый введённый нуль уменьшает порядок числа на единицу»; в *Справочнике программиста* на стр. 35 написано – «по командам, следующим за последовательностью команд цифрового ввода DIG A1, E, NEG X, POINT, CLR X содержимое X приводится к машинному виду: погашенные разряды мантиссы заполняются нулями, в разряды порядка, если они погашены, вписывается порядок введённого числа».

Производит ли машина нормализацию десятичных данных (не после их ввода вручную, а после пересылок при работе по программе) в ходе проверки условий условных переходов с участием регистров X, Y ?

Опыт 33:

Здесь, как и в опыте 30, на шагах 0–7 в регистр X через регистр RR засылается ненормализованная мантисса 000043210000 при равном нулю порядке. На шаге 9 выполняется команда BEQZ X, т. е. переход (на 4 шага вперёд), если машина решит на основе анализа первого разряда мантиссы, что "X = 0"; при этом в Y сохранится нулевое значение. В отсутствие перехода в Y записываем X.

```
00000 12 06          MOV X,RR
00001 13 00  04 03   MOV #04 03,S00
00003 13 01  02 01   MOV #02 01,S01
00005 11 04  08 05   MOV R08,R05
00007 12 07          MOV RR,X
00008 05 14          GO
00009 04 12  06 11   BEQZ X  00013
00011 06 04          MOV X,Y
00012 05 15          STOP
00013 05 12          END
```

КП: 184

Результат: Y = .000000000000, X = .000043210000

Вывод: в ходе исполнения команды BEQZ X нормализация X не производится.

Опыт 34:

Аналогичный предыдущему опыт; здесь к ненормализованному X применяется команда NEG X и затем команда условного перехода BMI X: переход, если знак числа X есть "минус".

```
00000 12 06          MOV X,RR
00001 13 00  04 03   MOV #04 03,S00
00003 13 01  02 01   MOV #02 01,S01
00005 11 04  08 05   MOV R08,R05
00007 12 07          MOV RR,X
00008 07 11          NEG X
00009 04 12  07 10   BMI X  00013
00011 06 04          MOV X,Y
00012 05 15          STOP
00013 05 12          END
```

КП: 183

Результат: Y = .000000000000, X = –.000043210000

Вывод: в ходе исполнения команд NEG X и BMI X нормализация X не производится.

Опыт 35:

Аналогичный опыт с командой BNEZ Y. Перед её исполнением в регистр Y засылается ненормализованное число из X. Если машина решит на основе анализа первого разряда мантиссы Y, что "Y не равно нулю", то произойдёт переход к шагу 9 + 4 = 13 с неизменным X; иначе на шаге 11 знак X изменится.

```
00000 12 06          MOV X,RR
00001 13 00  04 03   MOV #04 03,S00
00003 13 01  02 01   MOV #02 01,S01
00005 11 04  08 05   MOV R08,R05
00007 12 07          MOV RR,X
00008 06 04          MOV X,Y
00009 04 12  05 11   BNEZ Y  00013
00011 07 11          NEG X
00012 05 15          STOP
00013 05 12          END
```

КП: 182

Результат: Y = .000000000000, X = –.000043210000

Вывод: в ходе исполнения команды BNEZ Y нормализация Y не производится.

Опыт 36:

Аналогичный опыт с командой BEQ Y, X (переход, если "Y = X"; при этом машина проверяет только равенство нулю первого разряда разности Y – X).

```
00000 12 06          MOV X,RR
00001 13 00  04 03   MOV #04 03,S00
00003 13 01  02 01   MOV #02 01,S01
00005 11 04  08 05   MOV R08,R05
00007 12 07          MOV RR,X
00008 06 04          MOV X,Y
00009 05 14          GO
00010 05 09          BEQ Y,X 00013
00011 07 11          NEG X
00012 05 15          STOP
00013 05 12          END
```

КП: 183

Результат: Y = .000043210000, X = .000043210000

Вывод: в ходе исполнения команды BEQ Y, X нормализация X и Y не производится.

Опыт 37:

Аналогичный опыт с командой BSA Y, X (переход, если "Y = X" в смысле поразрядного совпадения X и Y).

```
00000 12 06          MOV X,RR
00001 13 00  04 03   MOV #04 03,S00
00003 13 01  02 01   MOV #02 01,S01
00005 11 04  08 05   MOV R08,R05
00007 12 07          MOV RR,X
00008 06 04          MOV X,Y
00009 05 14          GO
00010 12 04          BSA Y,X 00013
00011 07 11          NEG X
00012 05 15          STOP
00013 05 12          END
```

КП: 185

Результат: Y = .000043210000, X = .000043210000

Вывод: в ходе исполнения команды BSA Y, X нормализация X и Y не производится.

Наконец, в последних пяти опытах мы пытаемся понять, как выполняются пересылки блоков командами MOVBR и MOVBL. (В *Справочнике программиста* на стр. 114 говорится о "сдвижке " и "раздвижке" при адресации наложенных блоков, но не объясняется, что это такое.)

Адрес блока-приёмника задаётся в регистре R9, адрес блока-источника задаётся в регистре R10, количество пересылаемых байт задаётся в регистре R12. В нашем примере блок-источник выбран в виде следующих трёх байт:

```
01 00
02 00
03 00
```

с десятичными адресами 40, 41 и 42. Блок-приемник будем выбирать с адресами на две единицы меньше или больше, так что у приемника и источника крайняя ячейка ОЗУ будет одной и той же. Задача в опытах 38–41: после завершения каждой программы найти новое положение указанных байт (01 00, 02 00, 03 00), просматривая в режиме "Ввод" шаг за шагом ОЗУ в окрестности адресов 40–41.

Опыт 38:

Здесь на шагах 0–6 очищаются нулями 8 байт, начиная с адреса R10 = 32. На шагах 7–10 устанавливается R12 = 3. На шагах 11–20 формируется блок-источник с начальным адресом R10 = 32 + 8 = 40. На шагах 21–24 задаётся начальный адрес приёмника: R9 = R10 – 2 = 38, так что однобайтовая ячейка ОЗУ с адресом 40 будет общей для приёмника и источника. По команде MOVBR три байта блока-источника пересылаются в приёмник, начиная с младших адресов; в данном примере – с адреса 40 на адрес 38, затем с адреса 41 на адрес 39, и затем с адреса 42 на адрес 40, так что никаких проблем с общей для приёмника и источника ячейкой ОЗУ здесь не возникает.

```
00000 07 00          DIG 0
00001 13 04  00 00   MOV #00 00,S04
00003 13 05  02 00   MOV #02 00,S05
00005 04 12  02 10   MOV X,(R10)
00007 13 08  00 00   MOV #00 00,S08
00009 13 09  00 03   MOV #00 03,S09
00011 10 00  08 10   ADD #08,R10
00013 07 01          DIG 1
00014 07 00          DIG 0
00015 07 02          DIG 2
00016 07 00          DIG 0
00017 07 03          DIG 3
00018 07 10          E
00019 04 12  02 10   MOV X,(R10)
00021 11 04  10 09   MOV R10,R09
00023 10 01  02 09   SUB #02,R09
00025 05 14          GO
00026 05 14          GO
00027 04 12  15 12   MOVBR
00029 05 12          END
```

КП: 369

Результат:

Y = .000000000000, X = .102030000000 (и так же будет в трёх следующих опытах),

```
00038 01 00
00039 02 00
00040 03 00
00041 02 00
00042 03 00
```

Опыт 39:

Отличие от предыдущей программки – на шаге 23. Теперь начальный адрес приёмника равен 42; ячейка с этим адресом является и последней ячейкой источника, поэтому её содержимое (03 00) будет затёрто содержимым первой ячейки источника (01 00) в ходе пересылки с адреса 40 на адрес 42. Решает ли машина как-нибудь эту проблему?

```
00000 07 00          DIG 0
00001 13 04  00 00   MOV #00 00,S04
00003 13 05  02 00   MOV #02 00,S05
00005 04 12  02 10   MOV X,(R10)
00007 13 08  00 00   MOV #00 00,S08
00009 13 09  00 03   MOV #00 03,S09
00011 10 00  08 10   ADD #08,R10
00013 07 01          DIG 1
00014 07 00          DIG 0
00015 07 02          DIG 2
00016 07 00          DIG 0
00017 07 03          DIG 3
00018 07 10          E
00019 04 12  02 10   MOV X,(R10)
00021 11 04  10 09   MOV R10,R09
00023 10 00  02 09   ADD #02,R09
00025 05 14          GO
00026 05 14          GO
00027 04 12  15 12   MOVBR
00029 05 12          END
```

КП: 368

Результат:

```
00038 00 00
00039 00 00
00040 01 00
00041 02 00
00042 01 00
00043 02 00
00044 01 00
```

Вывод: машина не обращает внимания на проблему: пересылает уже изменённое содержимое.

Опыт 40:

Отличие от предыдущей программки – на шагах 24, 28. По команде MOVBL три байта блока-источника пересылаются в приёмник, начиная со старших адресов; в данном примере – с адреса 42 на адрес 40 (при этом имеем проблему: исходное содержимое источника 01 00 затирается байтом 03 00), затем пересылка идёт с адреса 41 на адрес 39, и затем с адреса 40 на адрес 38.

```
00000 07 00          DIG 0
00001 13 04  00 00   MOV #00 00,S04
00003 13 05  02 00   MOV #02 00,S05
00005 04 12  02 10   MOV X,(R10)
00007 13 08  00 00   MOV #00 00,S08
00009 13 09  00 03   MOV #00 03,S09
00011 10 00  08 10   ADD #08,R10
00013 07 01          DIG 1
00014 07 00          DIG 0
00015 07 02          DIG 2
00016 07 00          DIG 0
00017 07 03          DIG 3
00018 07 10          E
00019 04 12  02 10   MOV X,(R10)
00021 11 04  10 09   MOV R10,R09
00023 10 00  02 10   ADD #02,R10
00025 05 14          GO
00026 05 14          GO
00027 04 12  15 04   MOVBL
00029 05 12          END
```

КП: 361

Результат:

```
00038 03 00
00039 02 00
00040 03 00
00041 02 00
00042 03 00
00043 00 00
00044 00 00
```

Вывод: машина не обращает внимания на проблему: пересылает уже изменённое содержимое.

Опыт 41:

Отличие от предыдущей программки – на шагах 25, 26. Здесь также по команде MOVBL три байта блока-источника пересылаются в приёмник, начиная со старших адресов. Но теперь пересылка идёт с адреса 42 на адрес 44, затем с 41 на 43, и затем с 40 на 42, так что данные источника не теряются.

```
00000 07 00          DIG 0
00001 13 04  00 00   MOV #00 00,S04
00003 13 05  02 00   MOV #02 00,S05
00005 04 12  02 10   MOV X,(R10)
00007 13 08  00 00   MOV #00 00,S08
00009 13 09  00 03   MOV #00 03,S09
00011 10 00  08 10   ADD #08,R10
00013 07 01          DIG 1
00014 07 00          DIG 0
00015 07 02          DIG 2
00016 07 00          DIG 0
00017 07 03          DIG 3
00018 07 10          E
00019 04 12  02 10   MOV X,(R10)
00021 11 04  10 09   MOV R10,R09
00023 10 00  02 10   ADD #02,R10
00025 10 00  04 09   ADD #04,R09
00027 04 12  15 04   MOVBL
00029 05 12          END
```

КП: 346

Результат:

```
00038 00 00
00039 00 00
00040 01 00
00041 02 00
00042 01 00
00043 02 00
00044 03 00
00045 00 00
```

Вывод: машина выполняет пересылку блоков последовательно со старших адресов в старшие по команде MOVBL, и с младших адресов в младшие по команде MOVBR, несмотря на возможную потерю данных источника в случае перекрывающихся адресов источника и приемника.

Опыт 42:

Отличие от предыдущей программки – на шагах 25, 26, 29-31. В этом опыте надо после останова посмотреть Х-табло, чтобы узнать, изменяется ли в команде MOVBL содержимое R12, если оно задано равным 0. (В *Справочнике программиста* сказано, что при R12 = 0 пересылка блока не выполняется, и что по завершению команды будет R12 = 15 15 15 15, т. е. $R12 = -32767_{10}$. Возникает вопрос: так ли это, если пересылка не выполняется?) В программке перед остановом по END число из регистра R12 выводится в десятичной форме на Х-табло.

```
00000 07 00          DIG 0
00001 13 04  00 00   MOV #00 00,S04
00003 13 05  02 00   MOV #02 00,S05
```

```
00005 04 12  02 10  MOV X,(R10)
00007 13 08  00 00  MOV #00 00,S08
00009 13 09  00 03  MOV #00 03,S09
00011 10 00  08 10  ADD #08,R10
00013 07 01          DIG 1
00014 07 00          DIG 0
00015 07 02          DIG 2
00016 07 00          DIG 0
00017 07 03          DIG 3
00018 07 10          E
00019 04 12  02 10  MOV X,(R10)
00021 11 04  10 09  MOV R10,R09
00023 10 00  02 10  ADD #02,R10
00025 04 13  10 12  CLR R12
00027 04 12  15 04  MOVBL
00029 04 13  04 12  MOVD R12,X
00031 05 12          END
```

КП = 395

Результат:

```
00038 00 00
00039 00 00
00040 01 00
00041 02 00
00042 03 00
00043 00 00
00044 00 00
00045 00 00
```

X = –32767.0000000

Вывод: операция пересылки блока всегда завершается установкой R12 = 15 15 15 15, даже когда количество пересылаемых блоков задано равным нулю.

Спасибо, Наталия **xlat**!

29.12.2019. Sinus.