

Материалы к занятию 20 мая 2020

Тема: функциональная отладка программ на ассемблере.

Проблемы отладки программ с помощью отладчика мы уже рассмотрели ранее. Это удобный вариант, имеющий, однако, один существенный недостаток. Для этого необходимо достаточно хорошо владеть этим отладчиком, что как следует из пособия, приведенного на прошлом занятии, нетривиально. Мало того, однажды хорошо изучив отладчик, спустя некоторое время (при отсутствии практики работы с ним) все забывается (у меня это произошло примерно через полтора года). Приходится опять заново все изучать. При написании серьезных управляющих программ без отладчика, конечно не обойтись. Однако, если программы не очень сложные, можно обойтись «подручными» средствами.

Эти средства достаточно просты:

- служебная (отладочная) печать,
- точки останова (вставление ожидания ввода с клавиатуры),
- точки принудительного выхода из программы и/или из процедуры.

Рассматривать эти средства мы будем на примере сочетания двух программ:

Резидентной **tsr2.asm**, которая умеет складывать и вычитать два десятизначных десятичных числа, и возвращать вызывающей программе результат, а также признаки отрицательности результата и признак переполнения при сложении, если оно возникло,

Программы для вызова этого резидента **tsr2\_c.asm**, которая должна формировать запрос резиденту, а затем, получив результат его работы, вывести на печать результат. Если в результате сложения возникло переполнение, в результате должен отображаться лишний (одиннадцатый разряд), который по понятным причинам может содержать только единицу. Если результат получился отрицательным, результат должен отображаться со знаком минус (при положительной результате знак плюс не отображается).

Для простоты будем полагать, что при вычитании вычитается второе число из первого.

Сначала рассмотрим программы. Начнем с резидента **tsr2.asm**.

Эта программа должна принять пакет данных от вызывающей программы, в котором указано, что надо сделать (складывать или вычитать, отображается символом '+' или '-'), а также два десятизначных неупакованных десятичных числа. В пакете данных должны быть предусмотрены поля для возврата результата сложения или вычитания, а также однобайтовые поля для признаков отрицательности результата и переполнения. Структура такого пакета приведена в программе **tsr2.asm** в строках с 7 по 13. В пакете 6 полей, назначение и формат которых ясны из комментариев.

Начало резидентной программы (строки 14-17) стандартное и в пояснениях не нуждается.

Далее идет провозглашение дальней процедуры **resprog** (строка 18) и переход на процедуру инициализации (строка 19), которые также

стандартны, и в пояснениях также не нуждаются. Отметим лишь, что для вызова резидента используется прерывание **int 65h**, которое входит в диапазон прерываний пользователя и поэтому в последующем восстановлении не нуждается. Кроме того, данный резидент не защищен от повторной загрузки, и не может быть выгружен, так как основной познавательный акцент здесь другой.

Главное, что нас здесь интересует, это новая процедура обработки прерывания **int 65h**, в которой и выполняются все необходимые действия.

Мы договоримся с вызывающей программой, что адрес пакета данных будет передаваться в регистрах **ES:BX** (что является стандартом для **DOS**).

Итак, в переданной структуре в сторону резидента передается три поля:

- **cmd** – символьный признак действия над числами ('+'или '-''),
- **num1** – первое число для действия (при вычитании – уменьшаемое),
- **num2** – второе число для действия (при вычитании – вычитаемое).

От резидента вызывающей программе возвращаются также три поля:

- **over** – признак переполнения (при переполнении =1),
- **nneg** – признак отрицательности результата (+1), то есть признак того, что второе число при вычитании по модулю больше первого,
- **res** – собственно результат (в случае переполнения при сложении к результату относится также признак переполнения).

Итак, процедура обработки прерывания **int 65h** начинается с команды сохранения всех регистров общего назначения (**POH**) – строка 22, **pusha**. Цель этого действия мы уже объясняли ранее.

Затем (**важно!!**) инициализируем признаки переполнения и отрицательности (флаги), которые мы должны возвращать (строки 24 и 25 – команды **mov es:[bx].over,0** и **mov es:[bx].nneg,0**, соответственно). Обозначение полей структуры Вам уже знакомо: **es:[bx]** означает адрес начала структуры, а поле после точкой **.nneg** – поле этой структуры.

Затем идет инициализация индекса – номера цифры (строка 26 – **mov si,0**) и счетчика цифр (строка 27 – **mov cx,10**).

Затем мы выясняем, что надо делать (строка 28 – **cmp byte ptr es:[bx].cmd**).

В строке 29 используется результат этой проверки (**jz minus**) – если был минус, выполняется переход на вычитание (метка **minus**), если был плюс (точнее, не минус), команда условного перехода пропускается, и происходит естественный переход в сложению.

Далее в строке 31 выполняется сброс флага переноса (**clc**), чтобы первое сложение не учло нечаянно случайный перенос.

Затем идет цикл сложения (строки 32-37). Подробно:

в строке 32 (**ad1: mov al,es:[bx+si].num1**) в **AL** загружается младшая цифра первого числа (вспомним, что младшая цифра находится по младшему адресу); адресация идет по двум регистрам: **BX** – начало структуры, **SI** – номер цифры (начиная от младшей), **.num1** – соответствующее поле

структуры; **byte ptr** здесь писать не надо, так как формат операнда определяет AL; метка **ad1**: нужна для замыкания цикла,

в строке 33 (**adc al,es:[bx+si].num2**) с содержимым **AL** складывается (с учетом разряда переноса, который для младшей цифры сброшен) младшая цифра второго числа,

в строке 34 (**aaa**) выполняется десятичная коррекция результата,

в строке 35 (**mov es:[bx+si].res,al**) получившаяся младшая цифра результата записывается на младшее же место соответствующего поля (при этом флаг переноса содержит возможный перенос),

в строке 36 (**inc si**) выполняется переход к следующей (старшей) цифре,

в строке 37 (**loop ad1**) замыкается цикл (который повторяется 10 раз – содержимое **CX**).

В строке 37 (**jnc ad2**) выясняется не было ли переноса из старшего (последнего) разряда. Если переполнения не было выполняется переход на метку **ad2**. Если перенос есть (то есть, имеет место переполнение) выполняется следующая команда (строка 39 – **mov es:[bx].over,1**) запись признака переполнения в пакете данных.

Затем в обоих случаях (строка 40 – **ad2: popa**) восстанавливаются все **РОН**, и выполняется выход из процедуры (строка 41 – **iret**).

Далее идет последовательность команд, реализующих вычитание чисел (куда мы приходим из строки 29).

Эта последовательность команд в точности соответствует последовательности команд сложения (разумеется, с заменой команды сложения на команду вычитания и, в случае возникновения переноса (заема) из старшего разряда, единица записывается не в поле переполнения, а в поле отрицательности результата – строка 51 **mov es:[bx].nneg,1**).

Если после вычитания старших цифр флаг переноса сброшен (не было заема), то выполняется переход на выход из процедуры (строка 50 – **jnc mi3**), где опять восстанавливаются все **РОН** и выполняется выход из процедуры (строки 63 и 64 – **mi3: popa** и **iret**).

Если флаг переноса не был сброшен (был заем), значит результат отрицательный (второе число больше первого). В это случае в поле **nneg** пакета данных записывается 1, и вычитание выполняется заново, но уже из второго числа вычитается первое.

Завершается все это также командами в строках 63 и 64.

### Программа **tsr2.asm**

```
;2020 Одновходовая TSR
;int 65h - сложение и вычитание
;десятизначных десятичных чисел,
;es:bx - адрес буфера данных (bx=0)
;Структура буфера
buf struc          ; Структура буфера
cmd  db  ?  ; Символ "+" или "-"
over db  ?  ; 1 - признак переполнения (d)
nneg db  ?  ; 1 - отрицательный результат
num1 db  10 dup(?) ; Первое 10-значное число
```

```

num2 db    10 dup(?) ; Второе число
res   db    10 dup(?) ; Результат
buf   ends      ; Конец структуры
Assume CS: Code, DS: Code
Code SEGMENT
    org 100h
    .286

resprog proc    far
    jmp init
; Новый обработчик прерывания 65h
new_65 proc    far
    pusha      ; Сохранение POH
; Инициализация флагов
    mov es:[bx].over,0 ; Переполнение = 0
    mov es:[bx].nneg,0 ; Отр. рез. = 0
    mov si,0 ; Индекс
    mov cx,10 ; Счетчик цифр
    cmp byte ptr es:[bx].cmd,'-' ; Минус ?
    jz minus; Да, на вычитание
; Сложение
    cld ; Сброс флага переноса для ADC
ad1: mov al,es:[bx+si].num1 ; Загрузка мл.цифры 1 числа (DS:SI)
    adc al,es:[bx+si].num2 ; Сложение с цифрой 2 числа
    aaa ; Десятичная коррекция сложения
    mov es:[bx+si].res,al ; Полученная цифра в результат
    inc si
    loop ad1
    jnc ad2 ; Переполнения нет
    mov es:[bx].over,1 ; Переполнение
ad2: popa ; Восстановление POH
    iret ; Выход при сложении
; Вычитание
minus: cld ; Сброс флага переноса для ADC
mi1: mov al,es:[bx+si].num1 ; Загрузка мл.цифры 1 числа
    sbb al,es:[bx+si].num2 ; Сложение с цифрой 2 числа
    aas ; Десятичная коррекция вычитания
    mov es:[bx+si].res,al ; Полученная цифра в результат
    inc si ; К следующей цифре
    loop mi1
    jnc mi3 ; Переполнения нет
    mov es:[bx].nneg,1; Отрицательный результат (переполнение)
;Здесь второе число больше первого
;Вычитаем из второго первое
    mov si,0 ; Индекс
    mov cx,10 ; Счетчик цифр
    cld ; Сброс флага переноса для ADC
mi2: mov al,es:[bx+si].num2 ; Загрузка мл.цифры 2 числа
    sbb al,es:[bx+si].num1 ; Сложение с цифрой 1 числа
    aas ; Десятичная коррекция вычитания
    mov es:[bx+si].res,al ; Полученная цифра в результат
    inc si ; К следующей цифре
    loop mi2
mi3: popa ; Восстановление POH

```

```

        iredt ; Выход при сложении
new_65   endp
resprog  endp
ressize  equ  $-resprog ; Размер в байтах резидентной части
init proc near
        mov  ax,2565h ; Функция 25h, вектор 65h
        lea  dx,new_65
        int  21h      ; Запись нового вектора 65h
; Завершение программы инициализации с оставлением
; резидентной части в памяти
        mov  dx,(ressize+10fh)/16
        mov  ax,3100h
        int  21h
init endp
Code ENDS
END resprog

```

Далее идет вызывающая программа **tsr2\_c.asm**.

Задача этой программы – подготовить пакет данных для резидента (разумеется, структура пакета та же, что и в предыдущей программе), передать его резиденту, принять от него результат, и вывести на экран результат. При этом надо иметь в виду, что:

- при выводе результата на экран начинать вывод надо со старшей цифры (соответственно, и старшего адреса),
- при наличии переполнения в случае сложения надо выводить дополнительный старший разряд (который, как уже говорилось, может быть только единицей),
- при получении отрицательно результата перед значением результата надо вывести знак минус.

Начало программы **tsr2\_c.asm** такое же, как у резидента (включая структуру пакета данных), поэтому останавливаться на нем не будем.

После первой команды программы в строке 18 (**start: jmp start1**), которая «перепрыгивает» через переменные и имеет метку для точки входа идут сами переменные.

Переменные здесь представлены двумя буферами **bufa** и **bufb**, предназначенными для формирования пакетов данных для резидента.

Первый буфер (**bufa**) показан лишь для того, чтобы показать, как можно отвести место для пакета с известной структурой.

Второй буфер представляет собственно пакет данных, которые мы хотим передать резиденту. Мы не занимаемся здесь вводом данных для пакета с клавиатуры, чтобы не усложнять программу. При экспериментах с программой поля **cmd** (строка 21), **num1** (строка 24) и **num2** (строка 25) надо менять в тексте программы (при этом надо помнить, что числа здесь записываются «задом наперед» – сначала младшие цифры, затем старшие. Затем надо транслировать программу и затем запускать (все как обычно).

В этой программе к основному тексту добавлены следующие служебные процедуры вывода на печать:

- стандартные процедуры печати **prax**, **pral**, **prsn**, **print**, **wk**, **space**, которые уже неоднократно использовались в разных программах (описывать не будем),
- процедуры печати регистров **CS**, **DS**, **ES**, **BX** по отдельности и всех вместе, есть еще процедура печати регистра **SI** (в моем тексте не используется, но можно использовать по аналогии),
- процедуры печати признака отрицательности и признака переполнения,
- процедура уведомления о вызове резидента (**tsr**),
- процедура натуральной (так, как цифры расположены в буфере) печати текущего буфера (из **ES:BX**),
- процедура «правильной» (начиная со старшей цифры) печати текущего буфера (из **ES:BX**).

К служебным процедурам следует также отнести процедуру «натуральной» печати результата **spr10** (которая не в режиме отладки не нужна). Пояснения мы напишем для пары процедур, остальные действуют аналогично.

Сначала разберем процедуру печати регистра **CS** (строки 128-134). Все процедуры выполнены без формального обозначения процедуры.

Строка 128 (**mes cs: lea dx,m\_cs**) помещает в **DX** адрес сообщения «**CS=**» (строка 194). В строке 129 (**call msg**) вызывается процедура вывода на экран строки (функция 9 системного прерывания). В строке 130 (**call space**) вызывается процедура вывода на экран пробела (строка 272). В строке 131 (**mov ax,cs**) в **AX** помещается значение **CS**, а в строке 132 (**call prax**) вызывается процедура печати четырехзначного шестнадцатеричного значения из **AX** (строка 206). В строке 133 (**call wk**) вызывается процедура вывода на экран символов перевода строки и возврата каретки (строка 264). Наконец, в строке 134 (**ret**) выполняется возврат из процедуры.

Процедура «натуральной» (то есть, в порядке расположения в буфере) печати текущего буфера (в регистрах **ES:BX** должен находиться адрес этого буфера). Процедура располагается в строках 90-100.

Строка 90 (**mes\_buf:**) входная метка процедуры. В строке 91 (**lea si,[bx].num1**) в регистр **SI** помещается адрес начала (младшей цифры) поля первого числа. В строке 92 (**call spr10**) вызывается процедура «натуральной» (служебной) печати 10-значного числа (строки 76-81), которая достаточно ясна без пояснений. Затем в строке 93 (**mov al,[bx].cmd**) в **AL** из пакета данных помещается символ действия (символ плюса или минуса), и в строке 94 (**call print**) вызывается процедура печати одного символа (строка 255).

В строке 95 (**lea si,[bx].num2**) в **SI** помещается адрес начала (младшей цифры) поля второго числа. В строке 96 (**call spr10**) опять вызывается процедура «натуральной» печати 10-значного числа.

В строках 97 и 98 так же, как в строках 93,94 печатается символ знака равенства.

В строках 99 и 100 печатается результат из пакета данных (аналогично печати чисел). Здесь в строке 100 процедура **spr10** не вызывается командой **call**, а происходит безусловный переход на нее. Так как это последний вызов в рассматриваемой процедуре, возврат из нее будет выполнен командой **ret** из процедуры **spr10**.

Следующая процедура «правильной» печати буфера (строки 102-120) выполнена аналогичным образом. Отличие в том, что «натуральная» печать выполняется в одну строку, а «правильная» – в столбик. Для этого здесь после каждой печати вызывается процедура печати **БК+ПС** (строка 264).

Рассмотрим еще процедуру «правильной» печати десятизначного числа **pr10** (строки 68-74).

Строка 68 (**pr10:mov cx,10**) – инициализация счетчика печати (10 цифр). Строка 69 (**add si,9**) смещение с младшей цифры на старшую. Добавляется 9, так как смещение считается от 0. (Напомню, что в регистре **SI** здесь находится смещение того поля, которое должно печататься (**num1**, **num2** или **res**)).

Строка 71 (**call prsn**) вызов процедуры печати одной цифры из **AL** (строка 248).

Строка 72 (**dec si**) УМЕНЬШЕНИЕ **SI** на 1, так как мы идем от старшей цифры к младшей (от старшего адреса к младшему).

И, наконец, замыкание цикла (строка 73 – **loop pr101**) и выход из процедуры (строка 74 – **ret**).

Остальные процедуры либо давно знакомы (как **kbin**), либо аналогичны уже описанным.

Перейдем к рассмотрению основного тела программы (начало метка **start1**, расположенная сразу после переменных).

Первая и главная команда в строке 29 (**lea bx,bufb**) – помещение в регистр **BX** смещения буфера, содержащего передаваемую в резидент структуру данных. (Адрес пакета должен находиться в регистрах **ES:BX**, но о регистре **ES** позаботилась **DOS** при запуске программы, поместив туда то же, что и в **CS**.) Эту пару регистров (**ES:BX**) мы в программе портить не будем, так что она всегда будет указывать на пакет данных. Внутри вызывающей программы регистр **ES** можно не указывать (что мы и делаем), так как в регистрах **CS** и **DS** находятся те же значения.

Далее в программе сразу идет служебная печать:

Строка 31 (**call mes\_buf**) «натурально» распечатывает содержимое подготовленного пакета данных. Помним, что там содержательны только поля **cmd**, **num1** и **num2**, которые заполнили мы сами. Здесь можно убедиться в том, что содержимое полей буфера соответствует тому, что мы хотели.

Строка 32 (**call wk**) выполняет переход на следующую строку.

Строка 33 (**call mes\_all**) выводит на экран все интересующие нас регистры (**CS**, **DS**, **ES**, **BX**). При желании можно туда добавить регистр **SI**, так как процедура для него есть. Первый фрагмент служебной печати на этом завершился.

Затем на строке 35 (**call mes\_tsr**) выводится сообщение о том, что вызывается резидент (собственно, эта печать также является служебной).

В строке 36 (**int 65h**) выполняется собственно вызов резидента.

После этого идет новый фрагмент служебной печати.

В строке 38 (**call mes\_sign**) выводится признак отрицательности результата.

В строке 39 (**call mes\_over**) – признак переполнения.

В строке 40 (**call mes\_res**) выводится содержимое буфера в «правильном» виде (со старшей цифры) и столбик. Здесь можно увидеть, что вернул резидент и, с учетом м признаков отрицательности и переполнения понять, насколько правильно все сделано.

Роль вызова в строке 41 (**call wk**) понятна без пояснений.

На этом второй фрагмент служебной печати завершился.

Далее идет штатная печать результата (строки 47-61) с печатью ведущего минуса, если результат отрицательный, и с дополнительным старшим, если при сложении произошло переполнение.

Далее в строках 63 и 64 идут ожидание нажатия клавиши и выход из программы.

На строке 62 есть закомментированная служебная печать, которая в данный момент не нужна. Таким образом можно комментировать (или добавлять) служебную печать в любом месте программы, если это необходимо.

Хочу отметить, что я действительно пользовался такой служебной печатью, когда отлаживал резидент (пока не отладил алгоритм вычитания).

**ВАЖНО.** Использовать такую служебную печать в резидентных программах и драйверах нельзя, так как системное прерывание **int 21h** нереентерабельно (не обладает свойством «повторной входимости»). Поэтому использование его в резидентах приводит к зависанию системы.

Отладку резидентов можно проводить используя точки выхода. Например, сначала в резиденте можно все закомментировать, кроме команды **iret**. Если возврат из резидента выполняется успешно, значит вызываем его мы правильно.

Затем надо потихоньку раскомментировать команды. При этом, разумеется надо следить за тем, чтобы раскомментированные команды образовывали правильный код (например, если Вы раскомментировали команду **pusha**, но не раскомментировали парную ей **popa**, очевидно, система зависнет.

Точки останова и выхода можно использовать и в вызывающей программе. Например, когда у меня резидент вызывал зависание системы, я поставил в строке 35 (**jmp ex**) переход на выход из программы. Запустив программу, я получил распечатку буфера и всех интересующих меня регистров, чтобы убедиться в том, что все в порядке.

Такую точку выхода можно ставить везде, где необходимо до наступления зависания проверить состояние буфера и регистров.



## Текст программы tsr2\_c.asm

```
;Программа вызова TSR int 65h
;сложения и вычитания
;десятизначных десятичных чисел,
;es:bx - адрес буфера данных (bx=0)
;Структура буфера
buf struc      ; Структура буфера
cmd           db  ?      ; Символ "+" или "-"
over          db  ?      ; 1 - признак переполнения
nneg          db  ?      ; 1 - отрицательный результат
num1          db  10 dup(?) ; Первое 10-значное число
num2          db  10 dup(?) ; Второе число
res           db  10 dup(?) ; Результат
buf           ends      ; Конец структуры
Assume CS: Code, DS: Code
Code          SEGMENT
               .286
               org      100h
start:        jmp      start1
bufa db       size buf dup(?); Отведение места для буфера
bufb:
            db  '-'      ; Символ "+" или "-"
            db  ?        ; 1 - признак переполнения
            db  ?        ; 1 - отрицательный результат
            db  0,0,0,0,0,0,0,0,0,1 ; Первое 10-значное число
            db  0,0,0,0,0,0,0,0,0,3 ; Второе число
            db  10 dup(?) ; Результат
            db  '$'      ;Терминатор для возможности печати буфера
start1:
            lea  bx,bufb ;Смещение буфера          *
;*****Служебная печать*****
            call mes_buf                                ; *
            call wk      ;ВК+ПС                      *
            call mes_all ;Печать нужных регистров *
;*****
;            jmp      ex ; Выход для отладки
            call      mes_tsr ;Вызов резидента
            int       65h ;Собственно вызов резидента
;*****Служебная печать*****
            call      mes_sign;Признак минуса          *
            call      mes_over;Признак переполнения    *
            call      mes_res ;                        *
            call      wk      ;                        *
;            call      mes_buf ;                        *
;            call      wk      ;                        *
;            call      mes_all ;Печать нужных регистров *
;*****
;Печать результата
            lea      si,[bx].res;Смещение результата
;Печать минуса, если отрицательно
            cmp      [bx].nneg,0
            jz       st1 ;Нет минуса
            mov      al,'-'
```

```

        call    print    ;Печать минуса
        jmp     st2      ;Пропуск перенесенной 1 при вычитании
st1:cmp    [bx].over,0
        jz      st2      ;Нет переполнения
;Здесь при сложении был перенос, появилась лишняя
;единица
        mov     al,'1'
        call    print    ;Печать перенесенной единицы
st2:call    pr10
        call    wk
;        call    mes_all;Служебная печать
ex: call    kbin
        int     20h
;Правильная печать (начиная со старшей цифры)
pr10:mov    cx,10
        add     si,9;Печать начинается со старшей цифры
pr101:mov    al,[si];Текущая цифра
        call    prsn;Печать одной цифры из AL
        dec     si    ;Переход к младшей цифре
        loop    pr101
        ret
;Служебная печать (натуральный порядок, с младшей)
spr10: mov    cx,10
spr101: mov    al,[si];Текущая цифра
        call    prsn;Печать одной цифры из AL
        inc     si    ;Переход к младшей цифре
        loop    spr101
        ret
;Процедура ввода с клавиатуры
kbin     proc near;Ввод с клавиатуры
        mov     ah,0;Функция 0
        int     16h ;Клавиатурного прерывания
        ret
kbin     endp
;*****Служебные процедуры*****
;Печать текущего буфера [BX] (натуральная)
mes_buf:
        lea     si,[bx].num1
        call    spr10
        mov     al,[bx].cmd
        call    print
        lea     si,[bx].num2
        call    spr10
        mov     al,'='
        call    print
        lea     si,[bx].res
        jmp     spr10
;Печать текущего буфера правильная (со старшей цифры)
mes_res:
        lea     dx,m_num1
        call    msg
        lea     si,[bx].num1
        call    pr10

```

```

    call wk
    mov  al,[bx].cmd
    call print
    call wk
    lea  dx,m_num2
    call msg
    lea  si,[bx].num2
    call pr10
    call wk
    lea  dx,m_res
    call msg
    lea  si,[bx].res
    call pr10
    jmp  wk
;Печать регистров CS,DS,ES,BX
mes_all: call  mes_cs
        call    mes_ds
        call    mes_es
        call    mes_bx
        jmp     mes_si
;Печать регистра CS
mes_cs:  lea  dx,m_cs
        call    msg
        call    space
        mov     ax,cs
        call    prax
        call    wk
        ret
;Печать регистра DS
mes_ds:  lea  dx,m_ds
        call    msg
        call    space
        mov     ax,ds
        call    prax
        call    wk
        ret
;Печать регистра ES
mes_es:  lea  dx,m_es
        call    msg
        call    space
        mov     ax,es
        call    prax
        call    wk
        ret
;Печать регистра BX
mes_bx:  lea  dx,m_bx
        call    msg
        call    space
        mov     ax,bx
        call    prax
        call    wk
        ret
;Печать регистра SI

```

```

mes_si:    lea    dx,m_si
           call   msg
           call   space
           mov     ax,si
           call   prax
           call   wk
           ret

;Печать знака
mes_sign:lea    dx,m_sign
           call   msg
           call   space
           mov     al,[bx].nneg
           call   prsn
           call   wk
           ret

;Печать переполнения
mes_over:lea    dx,m_over
           call   msg
           call   space
           mov     al,[bx].over
           call   prsn
           call   wk
           ret

;Печать вызова tsr
mes_tsr:lea     dx,m_tsr
           call   msg
           call   wk
           ret

;Процедура печати сообщения
msg        proc near;Вывод сообщения
           mov     ah,9
           int     21h
           ret
msg        endp
m_cs db    'CS=$'
m_ds db    'DS=$'
m_es db    'ES=$'
m_bx db    'BX=$'
m_si db    'SI=$'
m_sign db   'Знак=$'
m_over db   'Переполнение=$'
m_tsr db    'Вызов tsr$'
m_num1 db   'NUM1=$'
m_num2 db   'NUM2=$'
m_res db    'RES = $'
;*****Вспомогательные процедуры печати
prax proc near ; Печать четырехзначного шестнадцатеричного числа
из AX
           push    bx
           push    cx
           push    ax
           and     ax,0f000h
           mov     cl,12

```

```

        shr ax,cl
        call prsn
        pop ax
        push ax
        and ax,0f00h
        mov cl,8
        shr ax,cl
        call prsn
        pop ax
        push ax
        and ax,0f0h
        mov cl,4
        shr ax,cl
        call prsn
        pop ax
        push ax
        and ax,0fh
        call prsn
        pop ax
        pop cx
        pop bx
        ret
prax endp
; Печать двухзначного шестнадцатеричного числа из AL
pral proc near
    pusha
    push ax
    mov cl,4
    shr al,cl
    call prsn
    pop ax
    call prsn
    popa
    ret
pral endp
; Печать одной цифры из al
prsn proc near
    and al,0fh
    add al,30h
    cmp al,39h
    jle print
    add al,7
; Печать 1 ASCII символа
print proc near
    pusha
    mov ah,0eh
    int 10h
    popa
    ret
print endp
prsn endp
; Печать ВК и ПС
wk: pusha

```

```

        mov  al,0ah
        call print
        mov  al,0dh
        call print
        popa
        ret
; Печать пробела
space    proc near
        mov  al,32 ; Код пробела
        jmp  print ; Печать 1 ASCII символа
space    endp
Code ENDS
        END start

```

## Эксперименты с программами

### Эксперименты с резидентом

1. Поставить после строки 22 (tsr2.asm) команду `jmp ad2` или `jmp mi3` (это эквивалентно). Убедиться, что резидент ничего не делает. При этом надо не забывать перетранслировать резидент, а также не забывать его запускать. Если запустить программу `tsr2_c` без запущенного резидента, система зависнет по понятной причине (кстати, почему?).

2. Поставить после строки 43 (по исходному, не измененному тексту) команду `jmp mi3`. Убедиться, что резидент выполняет сложение, но не выполняет вычитание.

### Эксперименты с вызывающей программой

1. Попробовать сложение с различными исходными числами. Проверить, как выполняется перенос из разряда в разряд (надеюсь, с арифметикой у Вас проблем нет). Убедиться в том, что при возникновении переноса из старшего разряда в окончательном результате печатается дополнительный (одиннадцатый) разряд.

2. Попробовать вычитание с различными исходными числами (когда первое число больше второго, и наоборот).

3. Убедиться в том, что правильно работают признаки отрицательного результата и переполнения.

4. Попробуйте добавлять или убирать (можно закомментировав) служебную печать.

Все программы в общем архиве, в папке 05\_22N. В архиве же лежит образ этой папки для виртуальной машины. При переписывании папки в виртуальной машине не забывайте с файлов, которые должны изменяться, атрибут «только чтение», иначе будут ошибки трансляции и другие.

Если возникнут вопросы, пишите. Желаю успехов.