

Министерство образования Республики Беларусь
Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ
КАФЕДРА ИНФОРМАТИКИ

Отчёт по лабораторной работе №2
По дисциплине «Архитектура вычислительных систем»
По теме «Многозадачность в защищённом режиме»

Выполнил:
студент гр. 953501
Кореневский С. А.

Проверил:
старший преподаватель
Шиманский В.В.

Минск 2021

Содержание

1. Цель работы	3
2. Постановка задачи.....	4
3. Теоретические сведения	5
4. Программная реализация	10
5. Выводы	12
Список литературы	13
Приложение 1. Текст программы.....	14

1. Цель работы

Изучить принципы и средства реализации мультизадачности в защищенном режиме процессора. Получить практические навыки по программированию и использованию этих средств. А также замерим выполнение задач в последовательном и в мультизадачном режиме.

2. Постановка задачи

Написать программу, реализующую мультизадачность в защищенном режиме. Программа должна переключить процессор в защищенный режим, а затем запустить на выполнение 2-3 задачи, которые должны выполняться параллельно. Каждая задача выводит на экран свое сообщение. Задача выводит на экран часть сообщения, затем происходит переключение на другую задачу и т.д. Когда все задачи отработают программа должна вернуть процессор в реальный режим.

3. Теоретические сведения

Под мультизадачностью подразумевают способность компьютера выполнять несколько задач одновременно. На самом деле процессор некоторое время выполняет один командный поток, затем быстро переключается на второй и выполняет его, переключается на третий и т.д. При этом при каждом переключении сохраняется контекст прерываемого потока, так что потом процессор сможет "безболезненно" продолжить выполнение прерванного потока команд. Благодаря высокому быстродействию создается иллюзия того, что все задачи выполняются одновременно (параллельно).

Для управления мультизадачностью нет специальных команд. Задачи переключаются командами FAR CALL, FAR JMP, INT, IRET. Однако при этом участвуют специальные дескрипторы: дескриптор сегмента состояния задачи (Task State Segment) и дескриптор шлюза задачи. Когда управление передается на один из таких дескрипторов, происходит переключение задачи. При переключении задачи процессор сохраняет (восстанавливает) свой контекст в сегменте состояния задачи (TSS). Селектор TSS выполняемой задачи хранится в регистре задачи (Task Register). При переключении задачи процессор может сменить LDT, что позволяет назначить каждой задаче свое адресное пространство, недоступное для других задач. Можно также перегрузить CR3, что позволяет применить для изолирования задач механизм страничного преобразования.

Дескриптор TSS относится к системным дескрипторам. Поле Type дескриптора TSS может содержать код 1001, если это доступный TSS, или 1011, если это занятый TSS, т.е. если задача активна в настоящий момент.

На рис. 1 представлен формат сегмента TSS для процессора i80386. Из рисунка видно, что в TSS предусмотрены поля для хранения сегментных регистров GS, FS, DS, SS, CS, ES. Имеется поле для хранения содержимого регистра LDTR, указывающего на локальную таблицу дескрипторов, распределённую данной задаче. Для хранения содержимого 32-разрядных регистров используются поля TSS, обозначенные на рисунке как EDI, ESI, EBP, ESP, EBX, EDX, ECX, EAX, EFLAGS, EIP.

Поле CR3 хранит содержимое системного регистра CR3. Этот регистр является указателем на каталог таблиц страниц. Таким образом, каждая задача может иметь свой собственный каталог таблиц страниц, что позволяет выполнить изоляцию задач не только на уровне сегментов, но и на уровне страниц.

Битовая карта ввода/вывода (БКВВ)		
Дополнительная информация ОС		
Относительный адрес БКВВ	0	T 64
0	LDTR	60
0	GS	5C
0	FS	58
0	DS	54
0	SS	50
0	CS	4C
0	ES	48
EDI		44
ESI		40
EBP		3C
ESP		38
EBX		34
EDX		30
ECX		2C
EAX		28
EFLAGS		24
EIP		20
CR3		1C
0	SS2	18
ESP2		14
0	SS1	10
ESP1		C
0	SS0	8
ESP0		4
0	LINK	0

Рис. 1 – Сегмент TSS процессора i80386

TSS процессора i80386 содержит указатели на стеки для второго, первого и нулевого приоритетных колец. Это поля SS2:ESP2, SS1:ESP1, SS0:ESP0.

Поле *LINK* используется для ссылки на TSS, вызвавшей задачи при вложенном вызове задач, аналогично тому, как это было в процессоре i80286.

Бит *T* используется для отладки. Если он установлен в 1, при переключении на задачу возникает отладочное исключение, которое может быть использовано системным отладчиком.

Для обеспечения безопасной работы системы необходимо ограничить доступ программам пользователя ко всем или по крайней мере к некоторым портам ввода/вывода. Злонамеренная программа, имеющая доступ к портам контроллера прямого доступа к памяти, может выполнить с помощью этого контроллера чтение или запись информации по любым физическим адресам. Процессор i80286 хранит в регистре флагов уровень привилегий IOPR, на котором разрешено выполнять команды ввода/вывода. С помощью этого механизма можно запретить непривилегированным программам выполнять команды ввода/вывода.

Однако такой способ защиты не слишком удобен. Некоторые порты ввода/вывода не только безопасны для использования, но и весьма полезны для обычных программ (например, порт системного динамика или принтера).

Битовая карта ввода/вывода процессора i80386 позволяет для каждой задачи определить порты, которые эта задача может использовать. То есть операционная система имеет возможность санкционировать любую задачу для использования любого набора адресов портов ввода/вывода. Если задача попытается обратиться к несанкционированному порту ввода/вывода, произойдет исключение.

Сегмент TSS содержит поле, обозначенное на рис. 1 как база карты ввода/вывода. Оно служит для указания расположения битовой карты ввода/вывода задачи, использующей данный TSS. Поле базы карты ввода/вывода указывает 16-разрядное смещение начала битовой карты ввода/вывода относительно TSS. Предел TSS должен определяться с учетом карты. Каждый бит в карте ввода/вывода соответствует адресу байта порта ввода/вывода (карта состоит из 64 Кбит для описания доступа к 65536 портам). После битовой карты должен располагаться байт 0FFh.

При выполнении 16- или 32-разрядных операций ввода/вывода процессор проверяет все биты (2 или 4 бита), соответствующие адресу порта. Если проверяемый бит установлен в 1, происходит исключение.

Для привилегированных программ, если уровень привилегий меньше или равен уровню IOPR, процессор не выполняет проверку битовой карты ввода/вывода. Чтобы полностью запретить задаче обращаться к портам ввода/вывода, достаточно установить базу карты ввода/вывода большей или равной пределу TSS. В этом случае любая команда ввода/вывода приведет к генерации исключения.

Лишь значение первых 68h байт сегмента состояния задачи строго определены. Именно это число является минимальным размером TSS. Операционная система может по своему усмотрению устанавливать размер

TSS и включать в сегмент TSS дополнительную информацию, необходимую для работы задачи и зависящую от конкретной операционной системы (например, контекст сопроцессора, указатели открытых файлов или указатели на именованные конвейеры сетевого обмена). Включенная в этот сегмент информация автоматически заменяется процессором при выполнении команды CALL или JMP, селектор которой указывает на дескриптор сегмента TSS в таблице GDT (дескрипторы этого типа могут быть расположены только в этой таблице).

При переключении задачи с помощью прерывания или особого случая происходит автоматический возврат к прерванной задаче. Однако, организовав вложение задач, необходимо помнить, что, в отличие от процедур, при переключении задачи в стек ничего не включается. Дескриптор TSS задачи, выполняемой в данный момент, помечается как занятый. При переключении на другую задачу с вложением (по INT или FAR CALL) дескриптор TSS остается помеченным. Переключиться на занятую задачу нельзя (возникает нарушение общей защиты - исключение №13).

Для переключения задач также действуют правила привилегий. По команде JMP или CALL можно переключиться на задачу, TSS которой менее привилегирован:

$$DPL_{TSS} \geq \max(CPL, RPL).$$

Для особых случаев и прерываний это правило не действует. Если обработчик прерывания выполнен в виде отдельной задачи, то он может быть вызван независимо от значения CPL.

Не совсем удобно адресовать именно TSS для переключения задачи, т.к., во-первых, TSS могут быть размещены только в GDT (а в IDT или LDT – нет), а во-вторых, если пользоваться только TSS, то каждую задачу мы "намертво" привязываем к определенному уровню привилегий (DPL_{TSS}), с которого она доступна для переключения. Этих недостатков лишены шлюзы задачи. Формат дескриптора шлюза задачи приведен на рис. 2.

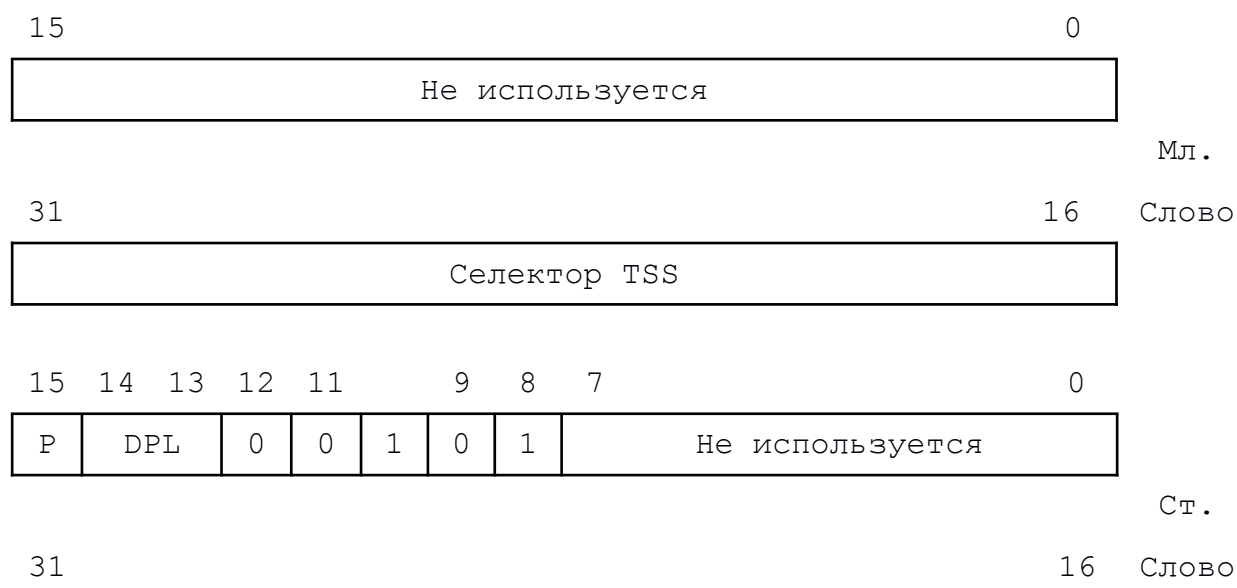


Рис. 2 – Дескриптор шлюза задачи для процессора i80386

Шлюз задачи содержит селектор TSS. Шлюзы задач можно размещать и в IDT, что позволяет выполнять обработчики прерываний в виде отдельных задач, и в LDT, что позволяет более гибко управлять переключением задач: для второй задачи первая может быть видна с одного уровня привилегий, а для третьей – с другого. Последняя возможность обеспечивается особым правилом привилегий: при переключении задачи через шлюз учитывается только $DPL_{\text{шлюза}}$, а DPL_{TSS} не играет роли, поэтому одной задаче может соответствовать множество шлюзов с различными DPL.

При переключении задач процессор выполняет следующие действия:

1. Выполняется команда CALL, селектор которой указывает на дескриптор сегмента типа TSS.
2. В TSS текущей задачи сохраняются значения регистров процессора. На текущий сегмент TSS указывает регистр процессора TR, содержащий селектор сегмента.
3. В TR загружается селектор сегмента TSS задачи, на которую переключается процессор.
4. Из нового TSS в регистр LDTR переносится значение селектора таблицы LDT в таблице GDT задачи.
5. Восстанавливаются значения регистров процессора (из соответствующих полей нового сегмента TSS).
6. В поле селектора возврата заносится селектор сегмента TSS снимаемой с выполнения задачи для организации возврата к прерванной задаче в будущем.

Вызов задачи через шлюз происходит аналогично, добавляется только этап поиска дескриптора сегмента TSS по значению селектора дескриптора шлюза вызова.

4. Программная реализация

В качестве средств для написания программы используются язык Assembler с компилятором TASM и средой исполнения DOSBox 0.74-3.

На рисунках представлен результат работы программы. Демонстрируется параллельный вывод строк в защищённом режиме и переход обратно в реальный режим.

Каждая из задач выполняется со своей скоростью, время выполнения отображено после задачи.

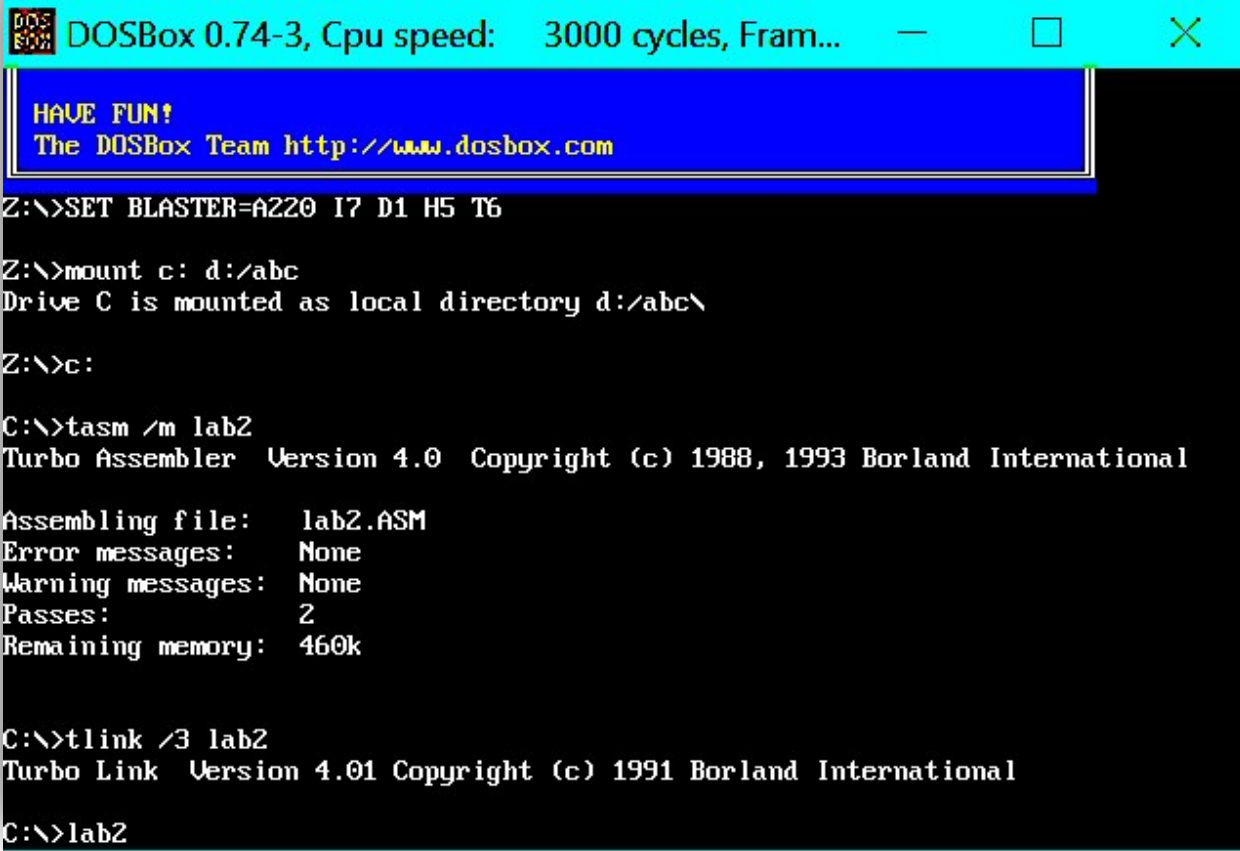
A screenshot of a DOSBox 0.74-3 window. The title bar is blue and contains the text "DOSBox 0.74-3, Cpu speed: 3000 cycles, Fram...". The main window has a black background with white text. At the top, there is a blue banner with yellow text that says "HAVE FUN!" and "The DOSBox Team http://www.dosbox.com". Below the banner, the command prompt shows the following sequence of commands and their outputs:
Z:\>SET BLASTER=A220 I7 D1 H5 T6
Z:\>mount c: d:/abc
Drive C is mounted as local directory d:/abc\
Z:\>c:
C:\>tasm /m lab2
Turbo Assembler Version 4.0 Copyright (c) 1988, 1993 Borland International
Assembling file: lab2.ASM
Error messages: None
Warning messages: None
Passes: 2
Remaining memory: 460k
C:\>tlink /3 lab2
Turbo Link Version 4.01 Copyright (c) 1991 Borland International
C:\>lab2

Рис.1 Подготовка программы



Рис.2 Последовательный режим



Рис.3 Мультизадачный режим

5. Выводы

В ходе выполнения лабораторной работы были изучены принципы и средства реализации мультизадачности в защищенном режиме процессора.

Получены практические навыки по программированию и использованию этих средств: написана программа, переключающая процессор в защищенный режим, затем запускающая на выполнение три параллельных задачи, и возвращающая процессор в реальный режим, когда все задачи отработают.

Список литературы

1. Волорова Н. А. Лабораторный практикум по курсу «Архитектура вычислительных систем» для студентов специальности «Информатика» / 93-444-487-2- Мн.: БГУИР, 2003. – 32с.:ил.
2. Калашников О.А. Учимся программировать. –СПб.: БХВ-Петербург 2011. – 336с. + Ил. CD-ROM
3. В.И. Юров Assembler. Учебник для вузов.2-е изд. / – СПб.: Питер, 2003. –637 с.: ил.

Приложение 1. Текст программы

```
; Программа, выполняющая переключение задач.

.386p
RM_seg segment para public "CODE" use16
    assume cs:RM_seg,ds:PM_seg,ss:stack_seg
start:
; подготовить сегментные регистры
    push    PM_seg
    pop     ds
; очистить экран
    mov     ax,3
    int     10h
; вычислить базы для всех дескрипторов сегментов данных
    xor     eax,eax
    mov     ax,RM_seg
    shl     eax,4
    mov     word ptr GDT_16bitCS+2,ax
    shr     eax,16
    mov     byte ptr GDT_16bitCS+4,al
    mov     ax,PM_seg
    shl     eax,4
    mov     word ptr GDT_32bitCS+2,ax
    mov     word ptr GDT_32bitSS+2,ax
    shr     eax,16
    mov     byte ptr GDT_32bitCS+4,al
    mov     byte ptr GDT_32bitSS+4,al
; вычислить линейный адрес GDT
    xor     eax,eax
    mov     ax,PM_seg
    shl     eax,4
    push    eax
    add     eax,offset GDT
    mov     dword ptr gdtr+2,eax
; загрузить GDT
    lgdt    fword ptr gdtr
; вычислить линейные адреса сегментов TSS наших двух задач
    pop     eax
    push    eax
    push    eax
    push    eax
    push    eax

    pop     eax
    add     eax,offset TSS_0
    mov     word ptr GDT_TSS0+2,ax
    shr     eax,16
    mov     byte ptr GDT_TSS0+4,al

    pop     eax
    add     eax,offset TSS_1
    mov     word ptr GDT_TSS1+2,ax
    shr     eax,16
    mov     byte ptr GDT_TSS1+4,al

    pop     eax
    add     eax,offset TSS_2
    mov     word ptr GDT_TSS2+2,ax
```

```

        shr        eax,16
        mov        byte ptr GDT_TSS2+4,al

        pop        eax
        add        eax,offset TSS_3
        mov        word ptr GDT_TSS3+2,ax
        shr        eax,16
        mov        byte ptr GDT_TSS3+4,al

; открыть A20
        mov        al,2
        out        92h,al
; запретить прерывания
        cli
; запретить NMI
        in         al,70h
        or         al,80h
        out        70h,al
; переключиться в PM
        mov        eax,cr0
        or         al,1
        mov        cr0,eax
; загрузить CS
        db         66h
        db         0EAh
        dd         offset PM_entry
        dw         SEL_32bitCS

RM_return:
; переключиться в реальный режим RM
        mov        eax,cr0
        and        al,0FEh
        mov        cr0,eax
; сбросить очередь предвыборки и загрузить CS
        db         0EAh
        dw         $+4
        dw         RM_seg
; настроить сегментные регистры для реального режима
        mov        ax,PM_seg
        mov        ds,ax
        mov        es,ax
        mov        ax,stack_seg
        mov        bx,stack_pointer_start
        mov        ss,ax
        mov        sp,bx
; разрешить NMI
        in         al,70h
        and        al,07FH
        out        70h,al
; разрешить прерывания
        sti
; завершить программу
        mov        ah,4Ch
        int        21h
RM_seg ends

PM_seg segment para public "CODE" use32
        assume cs:PM_seg

; таблица глобальных дескрипторов

```

```

GDT      label      byte
          db          8 dup(0)
GDT_flatDS      db      0FFh,0FFh,0,0,0,10010010b,11001111b,0
GDT_16bitCS      db      0FFh,0FFh,0,0,0,10011010b,0,0
GDT_32bitCS      db      0FFh,0FFh,0,0,0,10011010b,11001111b,0
GDT_32bitSS      db      0FFh,0FFh,0,0,0,10010010b,11001111b,0
; сегменты TSS задач (32-битный свободный TSS)
GDT_TSS0      db      067h,0,0,0,0,10001001b,01000000b,0
GDT_TSS1      db      067h,0,0,0,0,10001001b,01000000b,0
GDT_TSS2      db      067h,0,0,0,0,10001001b,01000000b,0
GDT_TSS3      db      067h,0,0,0,0,10001001b,01000000b,0
gdt_size = $ - GDT
gdtr      dw      gdt_size-1      ; размер GDT
          dd      ?      ; адрес GDT

; используемые селекторы
SEL_flatDS      equ      001000b
SEL_16bitCS      equ      010000b
SEL_32bitCS      equ      011000b
SEL_32bitSS      equ      100000b
SEL_TSS0      equ      101000b
SEL_TSS1      equ      110000b
SEL_TSS2      equ      111000b
SEL_TSS3      equ      100000b

; сегмент TSS_0 будет инициализирован, как только мы выполним переключение
; из нашей основной задачи.
TSS_0      db      100h dup(0)

; Сегменты TSS_123. В них будет выполняться переключение, так что надо
; инициализировать все, что может потребоваться:

TSS_1      dd      0,0,0,0,0,0,0,0      ; связь, стеки,
CR3      dd      offset task_1      ; EIP
; регистры общего назначения
          dd      0,0,0,0,0,stack_pointer_1,0,0,0B8140h ; (ESP и
EDI)
; сегментные регистры
          dd      SEL_flatDS,SEL_32bitCS,SEL_32bitSS,SEL_flatDS,0,0
          dd      0      ; LDTR
          dd      0      ; адрес таблицы
ввода-вывода

TSS_2      dd      0,0,0,0,0,0,0,0      ; связь, стеки,
CR3      dd      offset task_2      ; EIP
; регистры общего назначения
          dd      0,0,0,0,0,stack_pointer_2,0,0,0B8140h ; (ESP и
EDI)
; сегментные регистры
          dd      SEL_flatDS,SEL_32bitCS,SEL_32bitSS,SEL_flatDS,0,0
          dd      0      ; LDTR
          dd      0      ; адрес таблицы
ввода-вывода

TSS_3      dd      0,0,0,0,0,0,0,0      ; связь, стеки,
CR3      dd      offset task_3      ; EIP
; регистры общего назначения

```



```

                                dd      0,0,0,0,0,stack_pointer_3,0,0,0B8140h ; (ESP и
EDI)
; сегментные регистры
                                dd      SEL_flatDS,SEL_32bitCS,SEL_32bitSS,SEL_flatDS,0,0
                                dd      0 ; LDTR
                                dd      0 ; адрес таблицы
ввода-вывода

; точка входа в 32-битный защищенный режим
PM_entry:
; подготовить регистры
    xor     eax,eax
    mov     ax,SEL_flatDS
    mov     ds,ax
    mov     es,ax
    mov     ax,SEL_32bitSS
    mov     ebx,stack_pointer_start
    mov     ss,ax
    mov     esp,ebx
; загрузить TSS задачи 0 в регистр TR
    mov     ax,SEL_TSS0
    ltr     ax
; только теперь наша программа выполнила все требования к переходу
; в защищенный режим
    xor     eax,eax
    mov     edi,0B8000h ; DS:EDI - адрес начала экрана

; настройка менеджера задач
    mov     cx, 1000
    jmp     task_main

; процедура небольшой задержки
wait_proc proc
    push cx
    mov     cx, 32000
wait_loop:
    nop
    loop    wait_loop
    pop     cx
    ret
wait_proc endp

task_main:
; дальний переход на TSS иных задач с задержкой для наглядности

    db      0EAh
    dd      0
    dw      SEL_TSS1

    db      0EAh
    dd      0
    dw      SEL_TSS2

    db      0EAh
    dd      0
    dw      SEL_TSS3

    push     cx
    mov     cx,42
long_wait_loop:

```

```

        call wait_proc
    loop    long_wait_loop
    pop     cx

    loop    task_main

; дальний переход на процедуру выхода в реальный режим
    db      0EAh
    dd      offset RM_return
    dw      SEL_16bitCS

; задача 1
task_1_message    db      " Hello, Assembler! "
task_1_message_length = $ - task_1_message

; начало подпрограммы
task_1:
    add     edi,160                ; смещаем на несколько строк
    mov     esi,offset task_1_message
    mov     cx,task_1_message_length
task_1_loop:
    mov     al,cs:[esi]            ; записать символ в регистр
    inc     esi                    ; перейти на следующий символ
    mov     byte ptr ds:[edi],al ; вывести символ на экран
    add     edi,2                  ; увеличить адрес вывода

; пусть эта задача работает реже остальных
    mov     ax,cx
    mov     cx,42
    multiple_wait:
; переключиться на задачу 0
    db      0EAh
    dd      0
    dw      SEL_TSS0
; сюда будет возвращаться управление из задачи 0
    loop    multiple_wait
    mov     cx,ax

    loop    task_1_loop
; по завершении задачи просто сразу возвращаем управление
task_1_finish:
    db      0EAh
    dd      0
    dw      SEL_TSS0
    jmp     task_1_finish

; задача 2
task_2_message    db      " alles uber den Krieg,
"
                  db      " alles uber die angst,
"
                  db      " Ob der Sieg dir was bringt,
"
                  db      " Wenn du gar nichts mehr kannst.
"
                  db      " Wenn du gar nichts mehr hast -
"

```

```

        db      " Nicht Mal Luft oder Licht,
"

task_2_message_length = $ - task_2_message

; начало подпрограммы
task_2:
    add        edi,1440                ; смещаем на несколько строк
    mov        esi,offset task_2_message
    mov        cx,task_2_message_length
task_2_loop:
    inc        cx
    sub        cx,5
    push       cx
    mov        cx,5
multiple_print:
    mov        al,cs:[esi]             ; записать символ в регистр
    inc        esi                    ; перейти на следующий символ
    mov        byte ptr ds:[edi],al    ; вывести символ на экран
    add        edi,2                   ; увеличить адрес вывода
    loop       multiple_print
    pop        cx
; переключиться на задачу 0
    db         0EAh
    dd         0
    dw         SEL_TSS0
; сюда будет возвращаться управление из задачи 0
    loop       task_2_loop
; по завершении задачи просто сразу возвращаем управление
task_2_finish:
    db         0EAh
    dd         0
    dw         SEL_TSS0
    jmp        task_2_finish

; задача 3
task_3_message db      " Ist das ein grund jetzt, dass du mich hasst,
"
                db      " Sag, oder ist das ein grund fur mich?
"
                db      " Ist das ein grund jetzt, dass du mich hasst,
"
                db      " Sag, oder ist das ein grund fur mich?
"
                db      " Ist das ein grund jetzt, dass du mich hasst,
"
                db      " Sag, oder ist das ein grund fur mich?
"
                db      " Ist das ein grund jetzt, dass du mich hasst,
"

task_3_message_length = $ - task_3_message

; начало подпрограммы
task_3:
    add        edi,2720                ; смещаем на несколько строк
    mov        esi,offset task_3_message
    mov        cx,task_3_message_length
task_3_loop:
    mov        al,cs:[esi]             ; записать символ в регистр

```

```

        inc     esi                ; перейти на следующий символ
        mov     byte ptr ds:[edi],al ; вывести символ на экран
        add     edi,2              ; увеличить адрес вывода
; переключиться на задачу 0
        db      0EAh
        dd      0
        dw      SEL_TSS0
; сюда будет возвращаться управление из задачи 0
        loop    task_3_loop
; по завершении задачи просто сразу возвращаем управление
task_3_finish:
        db      0EAh
        dd      0
        dw      SEL_TSS0
        jmp     task_3_finish

```

```
PM_seg ends
```

```

stack_seg segment para stack "STACK" ; стеки задач
stack_start      db      100h dup(?)
stack_pointer_start = $ - stack_start
stack_task1      db      100h dup(?)
stack_pointer_1 = $ - stack_start
stack_task2      db      100h dup(?)
stack_pointer_2 = $ - stack_start
stack_task3      db      100h dup(?)
stack_pointer_3 = $ - stack_start
stack_seg ends

```

```
end start
```