МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Юго-Западный государственный университет»
(ЮЗГУ)

Кафедра программной инженерии

УТВЕРЖДАЮ
Проректор по учебной работе
_____ О.Г. Локтионова
«15» 12 2017

ПРОГРАММИРОВАНИЕ РАСШИРЕННОЙ ПАМЯТИ НА ЯЗЫКЕ ASSEMBLER

Методические указания для проведения лабораторных занятий и выполнения самостоятельной внеаудиторной работы по дисциплине «Системное программное обеспечение» для студентов направления подготовки 09.03.04 «Программная инженерия»

Курск 2017

УДК 681.3

Составитель: А.В. Малышев

Рецензент
Кандидат технических наук,
начальник отдела информатизации
ГУ КРО ФСС РФ *А.Ф. Рубанов*

Программирование расширенной памяти на языке Assembler : методические указания для проведения лабораторных занятий и выполнения самостоятельной внеаудиторной работы по дисциплине «Системное программное обеспечение» / Юго-Зап. гос. ун-т; сост. А.В. Малышев. Курск, 2017. 15 с.: ил. 2. Библиогр.: с. 15

Содержат сведения, предназначенные для изучения студентами основных функций расширенной памяти, с последующим приобретением ими практических навыков по её программированию и использованию.

Предназначены для студентов направления подготовки 09.03.04.

Текст печатается в авторской редакции

Подписано в печать . Формат 60x84 1/16
Усл. печ. л. . Уч.-изд. л. . Тираж экз. Заказ. Бесплатно.
Юго-Западный государственный университет.
305040, г. Курск, ул. 50 лет Октября, 94.

1.1 Цель лабораторной работы

Изучение основных функций драйвера расширенной памяти HIMEM.SYS; приобретение практических навыков по программированию XMB, HMA, UMB для хранения данных; изучение формата управляющей структуры копирования блоков памяти

2. Основные понятия

Работа прикладной программы (ПП) с XMS памятью выполняется при наличии HIMEM.SYS. Все функции драйвера HIMEM.SYS разделены на 5 групп:

1) Функции получения информации о драйвере (Oh)
2) Функции управления областью HMA (lh - 2h)
3) Функции управления линией A20 (3h – 7h)
4) Функции управления расширенной памятью (8h - Fh)
5) Функции управления UMB (10h-11h)

Ограничение. Корректная работа ПП с функциями драйвера HIMEM требует наличие сегмента стека размером не менее 256 байт (рекомендуется 512 байт). Перед использованием функций HIMEM необходимо выполнить 2 этапа: этап проверки наличия HIMEM.SYS и этап получения адреса управляющей структуры.

Эта проверки выполняется 43 функцией 2Fh прерывания. Получение адреса – 10 подфункция 43 функция прерывания 2Fh.

```
        mov ax, 4300h
        int 2fh
        cmp a1, 80h
        je  HMM_OK
        jmp ERROR
HMM_OK: mov ax, 4310h
        int 2fh         ; ES:BX  лог. адрес управляющей программы
        mov word ptr DS:[HMM], bx
        mov word ptr DS:[HMM+2], es
```

В дальнейшем сохраненный в переменной HMM адрес используется для выполнения всех функций по обслуживанию расширенной памяти. Использование управляющей программы сводится к настройке входных регистров и косвенному вызову подпрограммы с адресом HMM.

2.1 Функция получения информации о драйвере

Получить версию XMS
Регистры на входе: AH =00h.
Регистры на выходе:
BX= номер внутренней модификации драйвера;

DX = 000lh - если существует область HMA;
0000h - если область HMA не существует.

Функция возвращает номера версии и модификации XMS в двоично-десятичном (BCD) формате. Например, если AX = 0250h. это означает, что драйвер соответствует спецификации XMS версии 2.50. Дополнительно функция позволяет проверить наличие в системе области HMA.

2.2 Функция управления областью HMA

Запросить область HMA
Регистры на входе: *AH = 01h;*
DX = размер памяти в байтах в области HMA.
Регистры на выходе:
AX=000lh - если функция выполнена успешно;
0000h - если произошла ошибка.

Область HMA распределяется запросившей программе только в том случае, если запрошенный в регистре DX размер больше или равен указанному в параметре /HMAMIN. Такой механизм позволяет ограничить использование области HMA только тем программами, которые максимально используют ее размер.

Освободить область HMA
Регистры на входе *AH = 02h.*
Регистры на выходе *AX=000lh - если функция выполнена успешно, 0000h - если произошла ошибка.*

Программы, которые запрашивали область HMA, должны освобождать ее после использования при помощи этой функции. При этом данные, которые находились в этой области, будут потеряны, а область HMA становится доступной другим программам.

2.3 Функция управления линией A20

Глобальное открывание линии A20
Регистры на входе: *AH= 03h.*
Регистры на выходе: *AX=000lh - если функция выполнена успешно, 0000h - если произошла ошибка.*

Эта функция предназначена для ПП, которые будут использовать область HMA. Она разрешает работу заблокированной по умолчанию 21 адресной линии процессора. Перед возвратом управления системе программа должна закрыть линию A20 с помощью функции 04h.

Глобальное закрывание линии A20
Регистры на входе: *AH = 04h.*
Регистры на выходе: *AX=000lh - если функция выполнена успешно, 0000h - если произошла ошибка.*
Локальное открывание линии A20
Регистры на входе: AH = 05h
Регистры на выходе: AX=000lh - если функция выполнена успешно,

0000h - если произошла ошибка.

Эта функция предназначена для ПП, которые непосредственно управляют расширенной памятью. Перед завершением работы программа должна закрыть линию A20 при помощи функции 06h.

Локальное закрывание линии A20

Регистры на входе: *AH=06h.*

Регистры на выходе: *AX=000lh - если функция выполнена успешно, 0000h - если ошибка.*

Определение состояния линии A20

Регистры на входе: *AH = 07h.*

Регистры на выходе: *AX=000lh - если линия A20 открыта, 0000h - если линия A20 закрыта.*

Функция выполняет попытку адресоваться за границу 1 Мбайт памяти и проверяет, не происходит ли при этом обращение к началу памяти (то есть «свертка памяти»).

2.4 Функция управления расширенной памятью

Определение размера свободной расширенной памяти

Регистры на входе: *AH = 08h.*

Регистры на выходе: *AX=размер свободного блока расширенной памяти с максимальным размером (K);*
DX=общий размер свободной расширенной памяти (K).

При определении размера свободной расширенной памяти в возвращаемое значение не включается 64 Кбайт области HMA, даже если эта область не используется программами.

Получить блок XMB

Регистры на входе: *AH=09h;*
DX - размер требуемого блока (K).

Регистры на выходе: *AX=000lh - если функция выполнена успешно, 0000h - если произошла ошибка;*
DX = 16-разрядный идентификатор (handle) полученного блока XMB.

Количество блоков XMB, которое может быть заказано, определяется в командной строке драйвера HIMEM.SYS параметром NUMHANDLES=. Значение по умолчанию-32, максимальное значение - 128.

Освободить блок XMB

Регистры на входе: AH=OAh;
DX=идентификатор освобождаемого блока XMB.

Регистры на выходе: AX=000lh - если функция выполнена успешно, 0000h - если произошла ошибка.

Функция освобождает блок XMB, заказанный предыдущей функцией. При этом все данные, находившиеся в блоке, будут потеряны.

Копирование блоков XMB

Эта функция выполняет основную операцию с блоками XMB - копирование данных. Данные могут пересылаться по одному из трех направлений:

1. между обычной памятью и блоком XMB;
2. между 2-мя различными блоками XMB;
3. между 2-мя блоками обычной памяти.

Регистры на входе: *AH=Obh;*
DS:SI указатель на управляющую структуру, определяющую, откуда, куда как и сколько будет выполняться копирование.

Регистры на выходе: *AX=000lh - если функция выполнена успешно, 0000h - если произошла ошибка.*

Формат управляющей структуры:

```
XMS_Struct struc
Length          dd ?   ; количество пересылаемых байт
SourceHandle    dw ?   ; идентификатор исходного блока
SourceOffset    dd ?   ; смещение в исходном блоке
DestHandle      dw ?   ; идентификатор блока-назначения
DestOffset      dd ?   ; смещение в блоке-назначении
EMS_Struct ends
```

Поле Length управляющей структуры указывает количество пересылаемых байт данных, которое должно быть четным.

Поля SourceHandle и DestHandle указывают соответственно идентификаторы исходного блока и блока назначения. Если в качестве идентификатора задано значение 0000h, то в качестве источника или приемника данных используется обычная память.

Поля SourceOffset и DestOffset указывают 32-разрядное смещение в блоке XMB или логический адрес в обычной памяти в формате *сегмент:смещение.*

Функция копирования сама управляет линией A20, восстанавливая ее состояние после выполнения копирования. Поэтому программе не требуется управлять линией A20. Во время выполнения копирования разрешены прерывания.

Блокирование XMB

Регистры на входе: *AH=OCh;*
DX = идентификатор блока XMB.

Регистры на выходе: *AX=000lh - если функция выполнена успешно, 0000h - если произошла ошибка;*
DX:BX=32-разрядный линейный адрес заблокированного XMB.

Функция блокирует XMB и возвращает его базовый адрес как линейный

32-разрядный адрес. Для заблокированного XMB невозможно выполнить операцию копирования. Полученный линейный адрес действителен только для заблокированного XMB.

Разблокирование XMB

Регистры на входе: AH=ODh;
DX =идентификатор блока XMB.

Регистры на выходе: AX=000lh - если функция выполнена успешно, 0000h - если произошла ошибка.

Функция разблокирует XMB, заблокированный при вызове предыдущей функции. Полученный от нее линейный адрес становится недействительным.

Получить информацию об идентификаторе блока XMB

Регистры на входе: AH=OEh;
DX=идентификатор блока XMB.

Регистры на выходе: AX=0001h - если функция выполнена успешно, 0000h - если произошла ошибка;
BH=содержимое счетчика блокировок XMB;
BL=количество свободных иденти-фикаторов блоков XMB в системе;
DX = размер блока, Кбайт.

Изменить размер XMB

Регистры на входе: AH=OFh;
DX = идентификатор блока XMB;
BX = новый размер XMB, (K).

Регистры на выходе: AX=000lh - если функция выполнена успешно, 0000h - если произошла ошибка.

Функция изменяет размер незаблокированного XMB. Если размер блока уменьшается, данные в старших адресах блока будут потеряны.

2.5 Функции управления UMB

Запросить область UMB

Регистры на входе: AH=10h;
DX=размер запрашиваемого блока UMB в параграфах.

Регистры на выходе: AX=000lh - если функция выполнена успешно, 0000h - если произошла ошибка;
BX = сегмент полученного UMB;
DX = размер полученного блока или размер свободного максимального блока UMB (если невозможно выделить блок требуемого размера).

Эта функция позволяет программе получить доступ к блокам UMB, лежащим в пределах первого мегабайта адресного пространства. Для использования этих блоков не требуется управлять линией A20.

Если вам надо определить размер доступной области UMB, задайте при

вызове этой функции DX=0FFFFh.

Освободить область UMB

Регистры на входе: *AH = 11h;*
DX = сегмент освобождаемого UMB.

Регистры на выходе: *AX=000lh - если функция выполнена успешно, 0000h - если произошла ошибка.*

После освобождения UMB данные, которые там находились, будут потеряны.

Задача. Выполнить копирование второй половины массива на место первой половины с использованием XMB. Размер массива 16 байт.

Решение. На рис.1 дана графическая постановка задачи, символическая запись которой имеет следующий вид

$$CONV \rightarrow XMB \rightarrow CONV$$

Таким образом, решение задачи достигается через использование функции копирования между блоками памяти.

Для копирования создадим управляющую структуру memory, а на ее основе – 2 переменные: var1, var2.

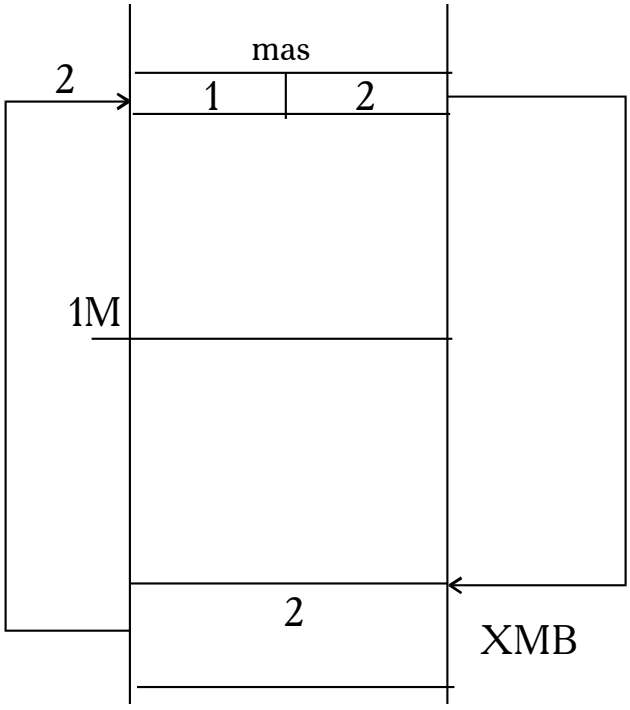


Рис.1. Графическая постановка задачи копирования массива

Граф-схема решения задачи имеет вид, представленный на рис. 2.

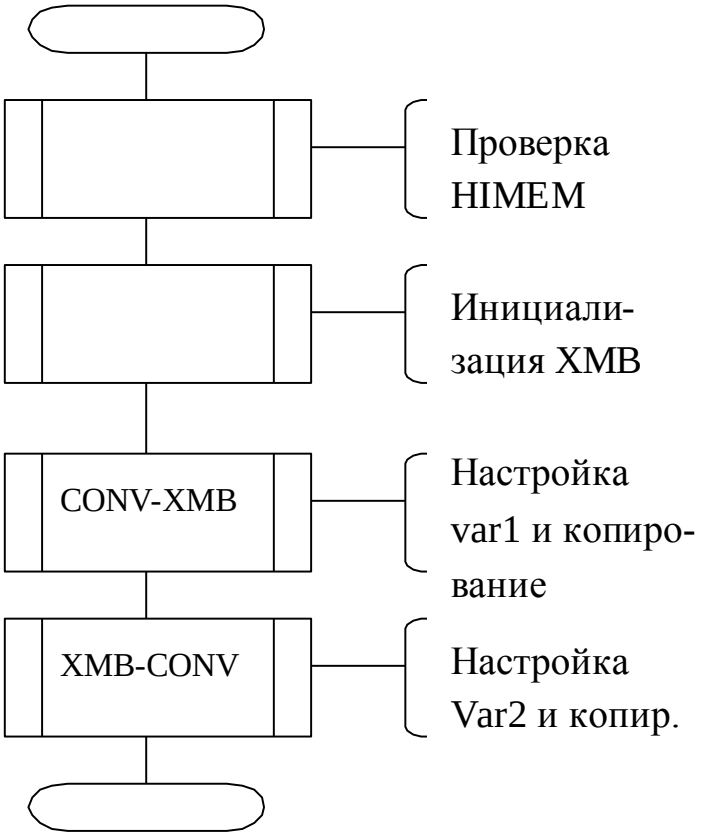


Рис.2. Граф-схема задачи копирования массива памяти

Сущность этапа инициализации XMB заключается в получении идентификатора XMB с требуемым размером. В случае, если не существует XMB с запрашиваемым размером, дальнейшая работа невозможна.

Основной этап выполняется двумя операциями копирования. При этом возникает задача корректного заполнения полей переменных var1, var2 с типом memory. Заполнение полей выполняется в сегментах данных и кода. В CODE при заполнении 32-битового поля **offset** возникает коллизия формата: для XMB данное поле является единым и хранит 32-битовое смещение; тогда как для обычной памяти **offset** хранит логический адрес, который, как известно, состоит из 2-х частей. Поэтому для обычной памяти в поле **offset** заполняется командами:

```
MOV word ptr var.offset[0], offset mas
MOV word ptr var.offset[2], seg mas
```

Директивы offset, seg определяют 16-битовые значения смещения и сегмента, в котором определена переменная mas. Поскольку поле **offset** размером 32 бита, необходимо использовать префикс замены формата данных - word ptr.

Текст программы

```
model small
```

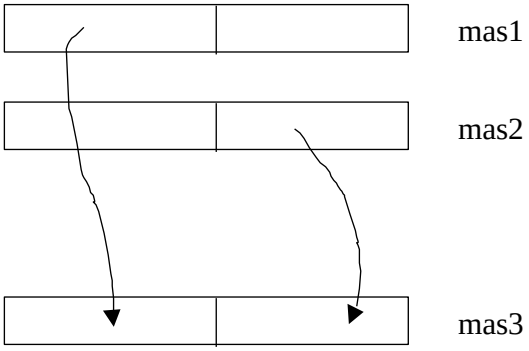
```
        .stack 512
        memory struc
        lengh  dd      ?
        handle1        dw      ?
        offset1        dd      ?
        handle2        dw      ?
        offset2        dd      ?
        memory         ends
        data segment
        HMM dd      ?
        errors db      'Произошла ошибка$',13,10
        HMM_ok     db      'Успешное копирование!$',13,10
        mas    db 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16
        len=$-mas
        var1   memory              <,0,,,0>
        var2   memory              <,,0,0,,>
        data ends
        code segment para public 'code'
        assume ds:data, cs:code
        .486
        begin:
        mov ax,data ; инициализация
        mov ds,ax   ; реального режима
; проверка установки драйвера HIMEM
; и получение адреса управляющей программы HIMEM
        mov ax,4300h
        int 2fh
        cmp al,80h
        je HiMEM
        jmp error
HiMEM:     mov ax,4310h
        int 2fh
        mov word ptr ds:[HMM],bx
        mov word ptr ds:[HMM]+2,es
; определение max размера XMB
        mov ax,0800h
        call [HMM] ; max размер XMB в dx в Кбайтах,
        mov cx,len     ; в cx размер mas в байтах
        shl edx,10     ; в dx размер XMB в байтах
        cmp edx,ecx; edx => ecx
        jb error       ; если меньше, ошибка
        shr cx,10
        inc cx         ; в cx размер mas в Кбайтах с запасом
; получить идентификатор XMB
```

```
        mov dx,cx    ; в dx требуемый размер XMB
        mov ax,0900h       ;
        call [HMM] ; в dx идентификатор XMB
        cmp ax,0       ; в ax код ошибки
        jz error
; настройка полей управляющей структуры для CONV-XMB
        mov bx,len
        shr bx,1               ; длина половины массива
        mov word ptr var1.lengh,bx      ; кол-во пересылаемых байт
        mov var1.handle2,dx       ; идентификатор-приемник
        mov word ptr var1.offset1[0],offset mas+8;
        mov word ptr var1.offset1[2],seg mas
; сохранение значений длины и
; идентификатора-приемника
        push bx
        push dx
; копирование mas/2 в EMB
        mov ax,0b00h
        lea si,var1    ; ds:si - указатель на var1
        call [HMM] ; копирование
        cmp ax,0      ; в ax код ошибки
        jz error
; настройка полей управляющей структуры для XMB-CONV
        pop dx
        pop bx
        mov word ptr var2.lengh,bx      ; кол-во пересылаемых байт
        mov var2.handle1,dx       ; идентификатор-источник
        mov word ptr var2.offset2[0],offset mas
        mov word ptr var2.offset2[2],seg mas
; копирование из EMB в mas/2
        mov ax,0b00h
        lea si,var2
        call [HMM]
        cmp ax,0
        jnz ALL_OK
        error:  lea dx,errors
        mov ah,09h
        int 21h
        jmp q
ALL_OK:    lea dx,HMM_ok
        mov ah,09h
        int 21h
q:      mov ax,4c00h
        int 21h
```

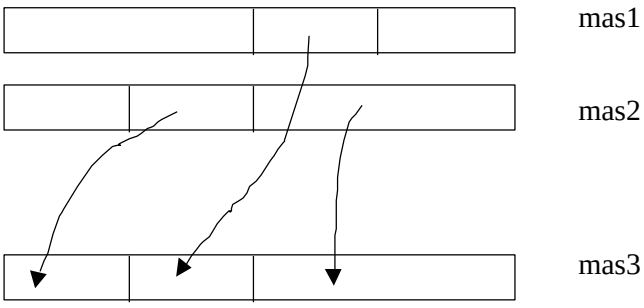
```
code ends
end begin
```

3. Варианты заданий

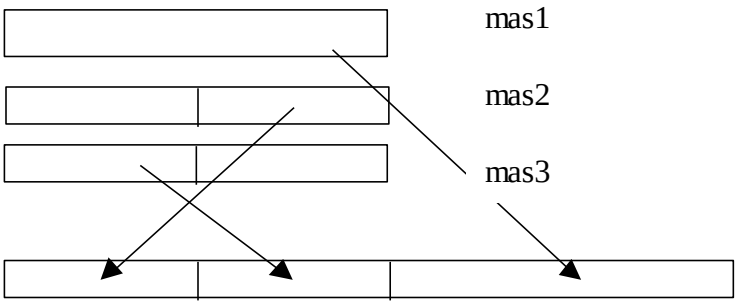
1. На основе двух исходных массивов mas1 и mas2 размером 32 байта выполнить заполнение mas3 с использованием XMB по следующей схеме



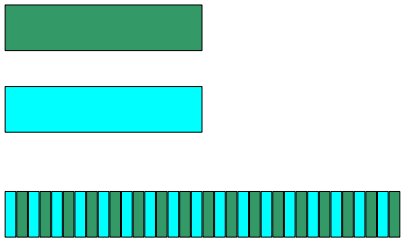
2. Выполнить копирование массива mas1 размером 64 слова в mas2 по следующей схеме: CONV→XMB_i→XMB_j→CONV. Получить линейные адреса XMB_i, XMB_j
3. Выполнить инверсное копирование массива mas1 размером 32 байта в mas2 по следующей схеме: CONV→XMB→CONV. Получить линейный адрес XMB.
4. .На основе двух исходных массивов mas1 и mas2 размером 64 байта выполнить заполнение mas3 с использованием XMB по следующей схеме



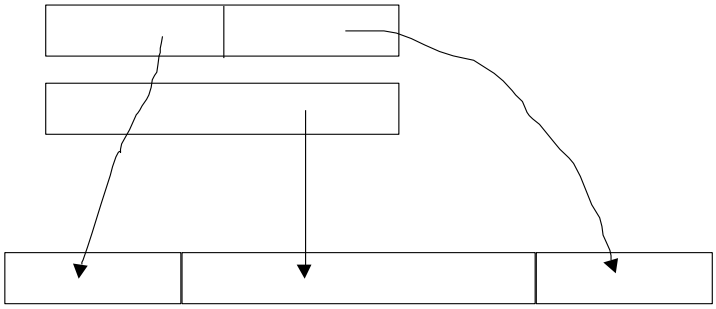
5. Выполнить копирование массива mas1 размером 2048 байт в mas2 по следующей схеме: CONV→XMB_i→ XMB_j →CONV. Получить линейные адреса XMB_i, XMB_j.
6. На основе трех исходных массивов mas1, mas2 и mas3 размером 32 слова выполнить заполнение mas4 с использованием XMB по следующей схеме



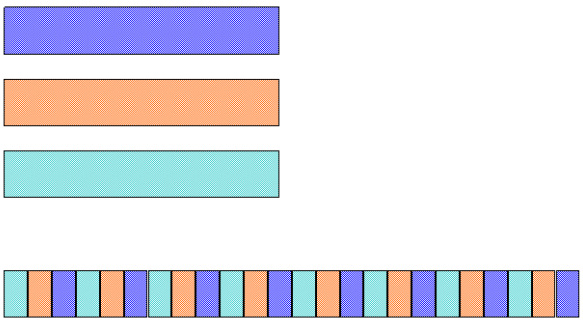
7. Выполнить копирование массива mas1 размером 4096 байт в mas2 по следующей схеме: XMA→CONV→XMB→CONV. Логический адрес mas1 в XMA 0FFFFh:0010h.
8. Выполнить копирование массива mas1 размером 128 слов в mas2 по следующим схемам: CONV→UMB, CONV→XMA, UMB→XMA. Получить логический адрес UMB. Логические адреса XMA для 2-ой и 3-ей схем 0FFFFh:0010h, 0FFFFh:0F000h соответственно.
9. На основе двух массивов mas1 и mas2 размером 256 слов каждый заполнить mas3 конкатенацией mas2,mas1.
10. Выполнить копирование массива mas1 размером 1024 слова в mas2 по следующей схеме: XMA→UMB→CONV. Логический адрес mas1 в XMA 0FFFFh:0F000h. Получить логический адрес UMB.
11. Выполнить копирование массива mas1 размером 2048 байт в mas2 по следующей схеме: UMB→CONV→XMB→CONV. Получить линейный адрес XMB.
12. Выполнить копирование массива mas1 размером 16К байт в mas2 по следующей схеме: XMB_i→CONV→XMB_j→CONV. Получить линейный адрес XMB_j
13. На основе двух массивов mas1 и mas2 размером 128 байт каждый заполнить mas3 по следующей схеме



14. На основе трех массивов mas1, mas2 и mas3 размером 256 слов каждый заполнить mas4 конкатенацией mas2,mas1,mas3.
15. На основе двух массивов mas1 и mas2 размером 512 байт каждый заполнить mas3 по следующей схеме



16. Выполнить копирование массива mas1 размером 1024 байта в mas2 по следующей схеме: $XMB_i \rightarrow CONV \rightarrow XMB_j \rightarrow CONV$. Получить линейные адреса XMB_i, XMB_j.
17. Выполнить копирование массива mas1 размером 1024 байта в mas2 по следующей схеме: $XMB_i \rightarrow CONV \rightarrow UMB \rightarrow CONV$. Получить логический адрес UMB
18. На основе 3-х массивов mas1, mas2 и mas3 размером 64 слова каждый заполнить mas4 по следующей схеме



4. Контрольные вопросы

1. Назначение, размер, распределение стандартной памяти.
2. Назначение, размер, распределение верхней памяти.
3. Назначение, размер, распределение старшей памяти.
4. Назначение, размер, распределение дополнительной памяти.
5. Назначение, размер, распределение расширенной памяти.
6. Механизм окна для доступа к дополнительной памяти. Менеджер памяти EMM386/EXE.
7. Функции HIMEM.SYS для управления расширенной памятью.
8. Функции HIMEM.SYS для управления UMB.
9. Функции HIMEM.SYS для управления HMA.
10. Состав управляющей структуры и копирование блоков памятей.
11. Отображаемая память (ПЗУ BIOS, видеопамять).

Библиографический список

1. Калуцкий, Игорь Владимирович. Системное программное обеспечение [Текст] : учебное пособие / И.В. Калуцкий, Е.А. Титенко. - Курск: ЮЗГУ, 2014. - 231 с.
2. Кенин А. Самоучитель системного администратора [Текст] : самоучитель / А. Кенин. - СПб: БХВ-Петербург, 2012. - 512 с.