МЕТОДЫ ТЕСТИРОВАНИЯ ОПЕРАТИВНОЙ ПАМЯТИ ПЕРСОНАЛЬНОГО КОМПЬЮТЕРА

Маргулис М.Б., Синельников А.Б.
Кафедра ЭВМ ДГТУ

Abstract

Margulis M.B., Sinelnikov A.B. RAM testing methods. *The RAM testing is basic element of diagnostic personal computers. This article contents RAM diagnostic and testing methods as C++ programms. The programm modules used certain testing sequence.*

Введение

Тестирование аппаратных ресурсов компьютера включает в себя несколько основных этапов. Важным и интересным направлением этой задачи является тестирование оперативной памяти персонального компьютера. В этой статье исследованы методы тестирования памяти, которые реализованы в виде программ на языке С (компилятор Borland C++ 3.1).

При тестировании оперативной памяти используются определенные тестовые последовательности, которые ориентированы на выявление различных видов отказов аппаратуры. Вследствие большой емкости современных запоминающих устройств, тесты их контроля отличаются большой длительностью и сложностью. Разные требования к времени прохождения теста и ориентация на преимущественное выявление определенных, наиболее характерных дефектов, породили большое разнообразие используемых тестов.

1. Определение участков памяти, подлежащих тестированию.

Рассматриваются шесть методов тестирования оперативной памяти, которые реализованы в виде отдельных функций. Эти функции вызываются из главной программы. Для получения стабильных результатов тестирования главная программа должна работать в однопрограммном режиме (MS DOS). В этом случае тестируемые участки памяти разбиваются на три части. Во-первых, это Convertional memory; во-вторых, Upper Memory Blocks и в-третьих, Extended Memory [2].

Для первых двух участков, алгоритм просмотра памяти основан на использовании метода определения свободной памяти с помощью блоков MCB (Memory Control Block) [2]. Каждый из этих двух участков имеет последовательность блоков памяти, которая завершается последним блоком MCB, имеющим тип 'Z'. Свободный блок памяти не имеет владельца, т.е. в поле владельца блока содержится нулевое значение. В том случае, если блок свободен, вызывается функция тестирования, которой передаются в качестве входных параметров адрес начала и конца блока.

Фрагмент программы определения свободных блоков памяти для Convertional и Upper Memory Blocks:

```
typedef struct          // Структура блока MCB.
 { unsigned char marker;              // Тип блока MCB
   unsigned int owner, SizePara;      // Адрес владелеца и Размер блока MCB
   unsigned char dummy[3], name[8];   // Резерв и Имя владельца блока MCB
 } mcb;
```

```
mcb far *ptr;
unsigned int segm, count=0;          // Адрес сегмента, счетчик блоков MCB
disable();                           // Запрет всех аппаратных прерываний
regs.h.ah = 0x52; intdosx(&regs, &regs, &segregs);    // Запрос перечня блоков MCB
segm = peek(segregs.es,regs.x.bx-2);     // Адрес первого блока MCB
do{  ptr = (mcb far*)MK_FP(segm,0);  // Указатель на следующий блок MCB
  if( ptr->marker == 'Z') count++;       // Очередной блок типа 'Z'
  if(!ptr->owner)                        // Блок не имеет владельца - свободен
     Mem_Test(segm, segm + ptr->sizePara+1);   // Вызов функции теста памяти
  segm += segm + ptr->sizePara + 1;
  } while(count<2);
enable();                        // Разрешение всех аппаратных прерываний
```

Для участка памяти выше 1Мб. – Extended Memory (XMS) можно воспользоваться методом обмена (чтения – записи) блоков памяти между XMS и Convertional Memory при помощи функций прерывания 2Fh.

Фрагмент программы для пересылки блока из Convertional Memory в XMS имеет вид

```
void far (*XMS_Entry_Point)(void);          // Функция - диспечер XMS
struct XMS_MOVE
 { unsigned long char length;               // Длина блока в байтах
   unsigned int sorrce_handle, source_off;  // Адрес источника
   unsigned int  dest_handle;               // Адрес приемника
   unsigned long dest_off;
 } xmov;
int handle;                                 // Указатель на занятый блок в XMS
 _AX=0x4310;  geninterrupt(0x2F);           // Инсталяция режима обмена
XMS_Entry_Point = (void far(*)(void))MK_FP(_ES,_BX);
 _AH=9; _DX=200; XMS_Entry_Point();         // Записать в XMS 200 байт
handle=_DX;                                 // Запомнить заголовок блока XMS
xmov.length = ptr->SizePara;                // Длина текущего блока
xmov.source_handler = 0;        xmov.source_off = (unsigned long)(ptr+1);
xmov.dest_handler = handle;     xmov.dest_off = 0;
_SI=FP_OFF(&xmov); _AH=0xB;
XMS_Entry_Point();          // Перенос блока памяти в XMS
```

Таким образом, задача тестирования XMS– памяти частично сводится к тесту блоков Convertional Memory, которые отображаются на соответствующих участках XMS.

2. Алгоритм тестирования памяти методом последовательной записи и считывания.

Суть алгоритма заключается в том, что в оперативную память с ее начала, последовательно, во все ячейки записываются нули; а затем производится считывание информации и сравнение с ранее записанной. Далее, в оперативную память с ее начала записываются единицы и вышеописанные действия повторяются [1].

Фрагмент функции теста памяти методом последовательной записи и считывания:

```
for(i=Beg+1;i<End;i++)
  for(j=0;j<16;j++)  pokeb(i,j,0);
for(i=Beg+1;i<End;i++)
  for(j=0;j<16;j++)
```

```
{ Test=peekb(i,j);
  if(Test!=0) { printf("Error – Adr: %04x:%04x.",i,j); exit(0); }
}
```

Нужно отметить, что этот тест обладает слабыми контрольными свойствами, так как он проверяет лишь схемы записи-считывания информации из оперативной памяти.

3. Алгоритм тестирования памяти методом шахматного кода.

Суть алгоритма заключается в том, что в оперативную память с ее начала, во все ячейки последовательно записываются противоположная информация, а затем проверяется правильность считывания этих данных [1].

Фрагмент функции тестирования памяти по алгоритму шахматного кода:

```
for(i=Beg+1;i<End;i++)        // Запись по всему блоку шахматной последовательность.
  for(j=0;j<16;j+=2) { pokeb(i,j,0xAA); pokeb(i,j+1,0x55);}
  for(i=Beg+1;i<End;i++)   // Чтение из блока значений,
  for(j=0;j<16;j+=2)
  { Test=peek(i,j);
    if(Test!=0xAA) { printf("Error – Adr: %04x:%04x.",i,j); exit(0); }
    Test=peek(i,j+1);
    if(Test!=0x55) { printf("Error – Adr: %04x:%04x.",i,j+1); exit(0); }
  }
```

Этот тест выявляет короткие замыкания между соседними ячейками памяти, а так же проверяет правильность операции записи и чтения из памяти.

4. Алгоритм тестирования памяти методом считывания и записи в прямом и обратном направлении.

Суть алгоритма заключается в том, что в оперативную память с ее начала, последовательно, во все ячейки записываются нули, а затем производится считывание информации, сравнение ее с нулями и запись единиц. Когда достигается конец оперативной памятя, от конца к началу происводится считывание ранее записанных единиц и запись нулей [1].

Фрагмент функции тестирования памяти по алгоритму считывания и записи в прямом и обратном направлении:

```
for(i=Beg+1;i<End;i++)
  for(j=0;j<16;j++)
  { pokeb(i,j,0x80);   Test=peek(i,j);
    if(Test!=0x80) { printf("Error – Adr:%04x:%04x. in 7 bit",i,j);   exit(0); }
    ...
for(i=End-1;i>Beg+1;i--)
  for(j=15;j<-1;j--)
  { pokeb(i,j,0xFE);  Test=peek(i,j);
    if(Test!=0xFE)  { printf("Error – Adr:%04x:%04x. in 0 bit",i,j);  exit(0);}
  }
```

Этот тест позволяет проверить правильность операции записи и считывания из оперативной памяти, а так же способность ячеек памяти переходить в противоположное состояние.

5. Алгоритм тестирования памяти методом четности (нечетности) адреса.

Суть алгоритма заключается в том, что в оперативную память с ее начала, последовательно, во все ее ячейки, имеющие четный адрес записываются нули, а в ячейки с нечетным адресом – единицы. Затем, с начала оперативной памяти производится считывание информации и если адрес четный то производится сравнение с нулем, в противном случае – с единицей [1].

Фрагмент функции тестирования по алгоритму четности (нечетности) адреса:

```
for(i=Beg+1;i<End;i++)          // Запись кодов
  for(j=0;j<16;j++)
  { lx=ldiv(j,2);
    if(lx.rem==0)  pokeb(i,j,0);
    else   pokeb(i,j,0xFF);     }
for(i=Beg+1;i<End;i++)
  for(j=0;j<16;j++)
  { lx=ldiv(j,2);
    if(lx.rem==0)            // Чтение из четного адреса
     { Test=peekb(i,j);
       if(Test!=0)  { printf("Error – Adr: %04x:%04x.",i,j);   exit(0); }
     }
```

Этот тест помимо проверки правильности операции чтения – записи позволяет проверить правильность установки адреса памяти.

6. Алгоритм тестирования памяти методом бегущей единицы.

Суть метода заключается в том, что в оперативную память первоначально записываются нули, затем в первую ячейку памяти записывается единица, и происходит чтение всех ячеек от конца памяти к ее началу. Когда достигнуто начало памяти – во вторую ячейку записывается нуль, происходит переход к следующей ячейке и вышеописанные действия повторяются [1].

Фрагмент функции тестирования памяти по алгоритму бегущей единицы:

```
// Запись во весь блок, кодовой последовательности и ее чтение.
 for(i=Beg+1;i<End;i++)              // Запись во весь блок, кодовой последовательности
и
   for(j=0;j<16;j++) pokeb(i,j,0x00);    // ее чтение.
     for(i=Beg+1;i<End;i++)          // Чтение и анализ.
   for(j=0;j<16;j++)
   { pokeb(i,j,0xFF);
     for(k=End-1;k>Beg+1;k--)
       for(z=15;z<-1;z--)
       {Test = peekb(k,z);
         if((k==i)&&(z==j))
         { if(Test!=0xFF)  { printf("Error – Adr: %04x:%04x.",i,j);   exit(0); }
           pokeb(i,j,0);
         }
          else   if(Test!=0)    ...
       }
```

Этот тест помимо проверки правильности операции чтения – записи позволяет проверить правильность работы ОЗУ при записи противоположной информации.

7. Алгоритм тестирования памяти методом попарного считывания.

Этот тест обеспечивает любые адресные переходы с различным изменением информации при считывании. Суть теста состоит в следующем: в начальный адрес записывается единица на фоне всех остальных нулей, а затем считываются адреса первый со вторым, первый с третьим и так далее до последнего, затем второй адрес с первым, второй с третьим и так далее [1].

Заключение

Для организации комплексной проверки состояния оперативной памяти компьютера, предлагаемые алгоритмы тестирования чаще всего используются совместно. Качественные характеристики каждого из рассмотренных методов тестирования приведены в таблице 1.

Обобщенные характеристики методов тестирования оперативной памяти компьютера

Таблица 1

	Проверка схем Записи- считывания	Проверка КЗ соседних ячеек памяти	Переход в противопо-ложн. сост.	Проверка установки адреса памяти
Метод последоват. записи и считывния	+			
Метод шахматного кода	+	+		
Считывание-запись в прямом и обратн. напр.	+		+	
Метод четности и нечетности адреса	+			+
Метод бегущей единицы	+		+	
Метод попарного считывания	+	+	+	+

Помимо рассмотренных алгоритмов, существует еще масса других методов тестирования памяти персонального компьютера. Однако, как показывает практика, процесс тестирования памяти предлагаемым здесь способом, обычно приводит к положительным результатам, при сравнительно небольших затратах времени.

Следует отметить, что указанные алгоритмы тестирования охватывают большинство неисправностей памяти компьютера, поддающихся диагностике.

Литература

1. "Полупроводниковые БИС запоминающих устройств" : Справочник под ред. А.Ю. Гордонова и Ю.Н. Дьякова. – М. : Радио и связь, 1986. – 360 с.
2. Фролов А.В. Фролов Г.В. "MS – DOS для программиста" М: Диалог МИФИ 1992. – 254с.