

Пензенский государственный университет
Кафедра «Вычислительная техника»

ОТЧЕТ

по лабораторной работе № 9

по дисциплине: "Арифметические и логические основы вычислительной техники"
на тему: "Деление в цифровых процессорах"

Выполнили:
студенты группы 20ВВ2
xxxxxxxxxxxxxxxxxxxx

Принял:
xxxxxxxxxxxxxxxxxxxx

Пенза, 2021

Лабораторное задание:

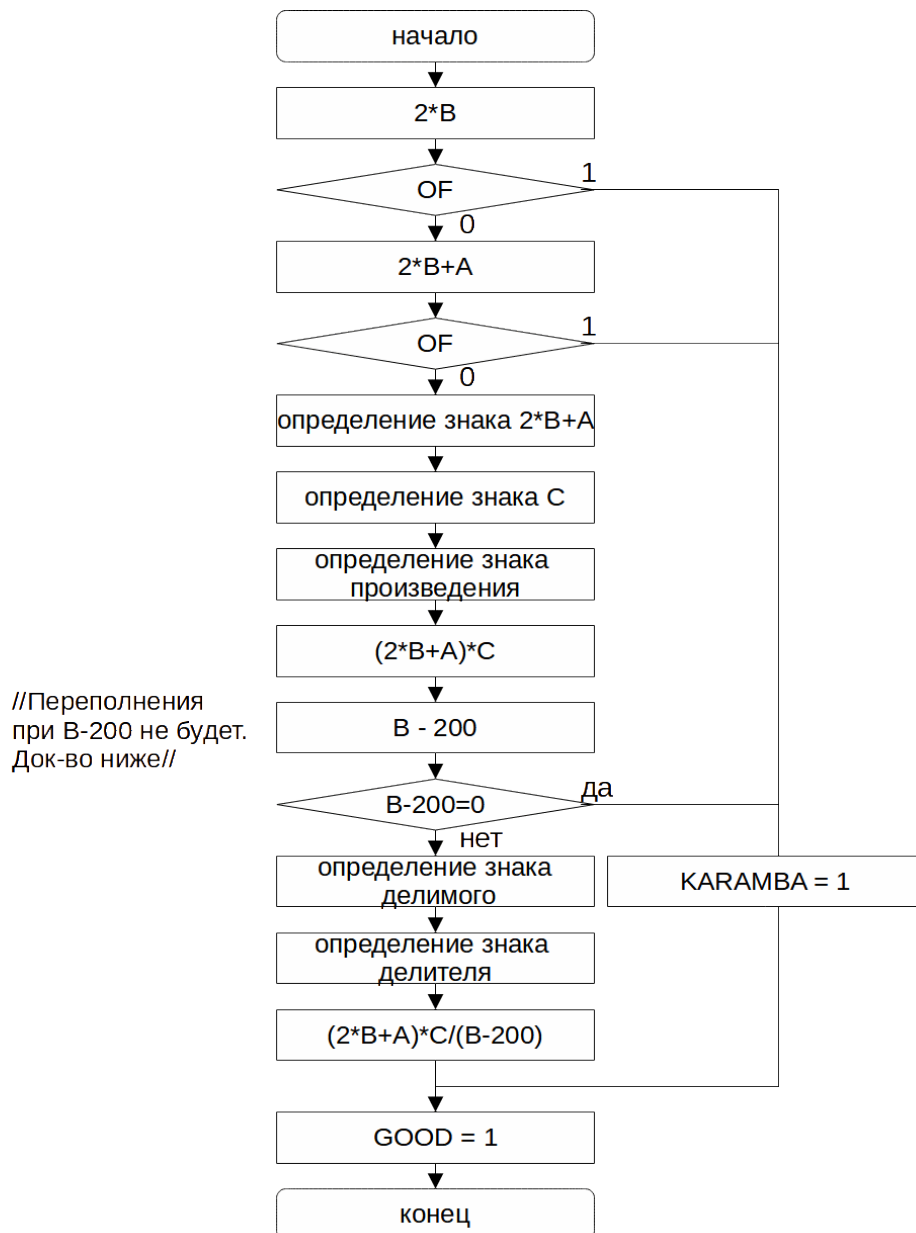
Написать на языке ассемблера программу вычисления выражения $Y = (2 * B + A) * C / (B - 200)$.

Процессор имеет разрядность – 16 бит. Исходные данные (значения переменных заданного выражения - целые, 16-битовые со знаком) располагаются в оперативной памяти, результат вычисления также поместить в оперативную память.

Деление переменных при вычислении выражения осуществлено с помощью операций сложения, вычитания, сдвига.

Умножение и деление на константы осуществлено с помощью операций сдвига, сложения и вычитания.

Общий алгоритм вычисления выражения приведен на блок-схеме:



Все возможные исключения при выполнении $(2*B+A)*C$ рассмотрены в лабораторной работе №8.

Поэтому обрабатываем исключения при вычислении знаменателя $(B-200)$ и делении на него:

Переполнение при В-200 не произойдет:

$$B - 200 = B + (-200)$$

$$200_{10} = C8_{16} = 1100\ 1000_2$$

$$[-200]_2 = 1111\ 1111\ 0011\ 1000_2$$

Условие переполнения:

$$\begin{array}{r} 1000\ 0000\ 0000\ 0000 \\ + 1111\ 1111\ 0011\ 1000 \\ \hline 10111\ 1111\ 0011\ 1000 \end{array}$$

$$CF = 1; SF = 0; OF = 1; ZF = 0$$

$$B = -8000_{16} = -1000\ 0000\ 0000\ 0000_2$$

$$[B]_2 = 1000\ 0000\ 0000\ 0000_2$$

При таком значении В переполнение произойдет при $2*B$ в процессе вычисления делимого:

$$2*B = 0000\ 0000\ 0000\ 0000$$

Значит, программа прервется до шага В – 200 и обработка переполнения здесь не требуется.

При $B = -4000_{16}$ переполнения при $2*B$ не произойдет, но и на шаге В – 200 переполнения не будет.

$$B = -4000_{16} = -100\ 0000\ 0000\ 0000_2$$

$$[B]_2 = 1100\ 0000\ 0000\ 0000_2$$

$$\begin{array}{r} 1100\ 0000\ 0000\ 0000 \\ + 1111\ 1111\ 0011\ 1000 \\ \hline 11011\ 1111\ 0011\ 1000 \end{array}$$

$$CF = 1; SF = 1; OF = 0; ZF = 0$$

- 1) В-200= 0 и аварийное завершение (деление на ноль).
- 2) Переполнение при делении, частное выходит за пределы 16 бит.
- 3) Нормальное завершение при положительном делимом и делителе, $Y > 0$, без остатка.
- 4) Нормальное завершение при положительном делимом и отрицательном делителе, $Y < 0$, без остатка.
- 5) Нормальное завершение при положительном делимом и делителе, $Y > 0$, с остатком.
- 6) Нормальное завершение при положительном делимом и отрицательном делителе, $Y < 0$, с остатком.

Общий алгоритм вычисления выражения приведен на блок схеме:

Листинг программы:

```
data segment
```

```
A dw ?
```

```
B dw ?
```

```
C dw ?
```

```
Y1 dw ?
```

```
Y2 dw ?
```

```
chast dw ?
```

```
ost dw ?
```

```
GOOD db ?
```

KARAMBA db ?

del dw ?

del_neg dw ?

mask dw 8000h ;маска для определения знака=1000 0000
0000 0000

data ends

code segment

assume cs: code, ds:data, ss: nothing

start:

mov ax, data

mov ds, ax

mov GOOD, 0 ;флаг нормального завершения

mov KARAMBA, 0 ;флаг аварийного завершения

;расчет суммы

mov ax, B ;ax=B

sal ax, 1 ;ax=2*B

jno next ;аварийное завершение при переполнении 2*B

inc KARAMBA

next:

mov bx, A

add ax, bx ;2*B+A

jno next2

inc KARAMBA

next2:

mov bx, C

```

;проверка знаков
mov dx, mask
mov cx, dx

;определение знака 2*B+A
and dx, ax ;выполняет логическое И между всеми битами
операндов; если 1 И 1 = 1, значит, 2*B+A - отрицательное
число

jz plus_SUM

not ax;инверсия и +1 (след. шаг), если число
отрицательное

inc ax
mov dx, 1

plus_SUM:
and cx, bx ;определение знака C
jz plus_C

not bx ;инверсия и +1 (след. шаг), если число
отрицательное

inc bx
mov cx, 1

plus_C:
xor dx, cx ;1*1=0, 0*0=0, 1*0=1
push dx ;сохранение знака произведения в стек

;выполнение умножения
xor dx, dx
mov cx, 15

;множимое 2*B+A в ax, множитель C в bx
mul_C:

```

```

    rcr bx,1 ;начало цикла алгоритма умножения – сдвиг
регистра множителя

    jnc a_1 ;проверка выдвинутого разряда – если бит равен
0, то сдвиг

    add dx, ax ;иначе сложение

a_1:
    rcr dx, 1 ;сдвиг сумматора
    loop mul_C ;уменьшение счетчика на 1

    rcr bx,1 ;дополнительный сдвиг регистра множителя
    rcr dx,1 ;дополнительный сдвиг регистра сумматора
    rcr bx,1 ;дополнительный сдвиг регистра множителя

;извлечение знака
pop cx
test cl, 1
jz LABELRESULT ;если знак 0, то выход
not dx
not bx
inc bx

LABELRESULT:
; Итог: в dx – старшая часть, в bx – младшая часть
произведения

mov Y1, dx ;загрузка старшей части в память
mov Y2, bx ;загрузка младшей части в память

;вычисление делителя
mov dx, B

```

```

sub dx, 200 ;dx=B-200 (делитель)
cmp dx, 0 ;проверка делителя на равенство 0
jne checking
inc KARAMBA
jmp МЕТКА_KARAMBA

;проверяем на отрицательность делимое
checking:
xor si, si ;очистка регистров, которые будут
использованы для сохранения знаков
xor di, di
mov ax, Y1 ;в регистре ax находится старший разряд
делимого
mov bx, Y2 ;в регистре bx находится младший разряд
делимого ЦЕЛАЯ ЧАСТЬ
test ax, 8000h
jz znakdel
xor ax, 0FFFFh ;то же, что и инверсия
xor bx, 0FFFFh
add bx, 1 ;добавление 1 к инвертированному числу
adc ax, 0
add si, 1 ;флаг того, что делимое отрицательно
znakdel:
test dx, 8000h ;проверка делителя на знак
jz predDIV ;если положительное то переход на
предварительное деление
neg dx ;если делитель отрицательный, переводим его
дополнительный код
add si, 2 ;сохраняем знак
;если si = 1, то отрицательно делимое

```



```

;если si = 2, то отрицателен делитель
;если si = 3, то отрицательны и делитель, и делимое
predDIV:
    mov cx, 16 ;цикл 10h = 10000h = 16, число в цикле
одинаковое
    mov del, dx ;делитель
    neg dx ;инверсия делителя
    mov del_neg, dx ;отрицательный делитель
    add ax, del_neg ;пробное деление
    jc METKA_KARAMBA ;аварийное завершение
    neg dx
    jmp delenie

save:
    popf ;восстановление из стека значения флагов

delenie:
    rcl bx,1 ;циклический сдвиг влево младших разрядов и
флага CF на 1 бит
    rcl ax,1 ;циклический сдвиг влево старших разрядов и
флага CF на 1 бит
    jc plus ;если CF=1 (полученный остаток отрицательный),
то к старшим разрядам делимого прибавляется содержимое
делителя
    add ax,del_neg ;иначе из старших разрядов делимого
вычитается делитель
    jmp cycle

plus:
    add ax, del

```

cycle:

pushf ;сохраняем в стек значения флагов

loop save

ostatok:

popf

rcl bx, 1

test ax, 8000h ;проверяем остаток на знак

jz ostatok2 ;если положительное, то восстанавливаем знак делителя

add ax, del ;иначе восстанавливаем остаток

ostatok2:

test si, 1 ;восстанавливаем знак делимого

jz ostpolozh ;если положительное, то переход на result

neg ax ;иначе инверсия остатка

add di, 1 ;флаг того, что результат отрицателен

ostpolozh:

test si, 2 ;восстанавливаем знак делителя

jz chastpolozh

add di, 1 ;флаг того, что результат отрицателен

chastpolozh:

cmp di, 1

jne result

neg bx

result:

```
mov chast, bx ;загрузка частного в память
mov ost, ax ;загрузка остатка в память
jmp МЕТКА_GOOD ;переход на нормальное завершение
```

МЕТКА_KARAMBA:

```
mov KARAMBA, 1 ;аварийное завершение
```

МЕТКА_GOOD:

```
mov GOOD, 1 ;нормальное завершение
```

quit:

```
mov ax, 4c00h ;возврат в операционную систему
```

```
int 21;
```

```
code ends
```

```
end start
```

Прогоны программы:

1) $B-200=0$ и аварийное завершение (деление на ноль).

$A = 9914_{10} = 26BA_{16} = 10\ 0110\ 1011\ 1010_2$

$B = 200_{10} = C8_{16} = 1100\ 1000_2$

$C = 15425_{10} = 3C41_{16} = 11\ 1100\ 0100\ 0001_2$


```

File View Run Breakpoints Data Options Window Help
[.] CPU 80486 ds:000F = 00 1-[ ] [ ]
cs:0000 B8CF4A mov ax,4ACF ax 4ACF c=0
cs:0003 8ED8 mov ds,ax bx 0000 z=0
cs:0005 C6060E0000 mov byte ptr [000E],00 cx 0000 s=0
cs:000A C6060F0000 mov byte ptr [000F],00 dx 0000 o=0
cs:000F A10200 mov ax,[0002] si 0000 p=0
cs:0012 D1E0 shl ax,1 di 0000 a=0
cs:0014 7104 jno 001A bp 0000 i=1
cs:0016 FE060F00 inc byte ptr [000F] sp 0000 d=0
cs:001A 8B1E0000 mov bx,[0000] ds 4ACF
cs:001E 03C3 add ax,bx es 4ABF
cs:0020 7104 jno 0026 ss 4ACE
cs:0022 FE060F00 inc byte ptr [000F] cs 4AD1
cs:0026 8B1E0400 mov bx,[0004] ip 000A
cs:002A 8B161400 mov dx,[0014]
cs:002E 8BCA mov cx,dx

A B C
ds:0000 54 2D CA 04 47 25 00 00 T-4*G%
ds:0008 00 00 00 00 00 00 00 00
ds:0010 00 00 00 00 00 00 00 00 A
ds:0018 00 00 00 00 00 00 00 00
ds:0020 B8 CF 4A 8E D8 C6 06 0E 1-J0+|+n
ss:0008 6864
ss:0006 742E
ss:0004 706C
ss:0002 6568
ss:0000 6474

F1-Help F2-Bkpt F3-Mod F4-Here F5-Zoom F6-Next F7-Trace F8-Step F9-Run F10-Menu

```

```

File View Run Breakpoints Data Options Window Help
[.] CPU 80486 1-[ ] [ ]
cs:0104 891E0A00 mov [000A],bx ax 03FC c=1
cs:0108 A30C00 mov [000C],ax bx C258 z=0
cs:010B EB06 jmp 0113 cx 0010 s=0
cs:010D 90 nop dx FBFE o=0
cs:010E C6060F0001 mov byte ptr [000F],01 si 0000 p=1
cs:0113 C6060E0001 mov byte ptr [000E],01 di 0000 a=1
cs:0118 B8004C mov ax,4C00 bp 0000 i=1
cs:011B CD15 int 15 sp 0000 d=0
cs:011D 0000 add [bx+si],al ds 4ACF
cs:011F 0000 add [bx+si],al es 4ABF
cs:0121 0000 add [bx+si],al ss 4ACE
cs:0123 0000 add [bx+si],al cs 4AD1
cs:0125 0000 add [bx+si],al ip 0118
cs:0127 0000 add [bx+si],al
cs:0129 0000 add [bx+si],al

A B C Y1
ds:0000 54 2D CA 04 47 25 FE 07 T-4*G%
ds:0008 58 C2 00 00 00 00 01 01 % ошибка
ds:0010 Y2 i chast lost 06 / успешное завершение
ds:0018 00 00 00 00 00 00 00 00
ds:0020 B8 CF 4A 8E D8 C6 06 0E 1-J0+|+n
ss:0008 6864
ss:0006 742E
ss:0004 706C
ss:0002 6568
ss:0000 6474

F1-Help F2-Bkpt F3-Mod F4-Here F5-Zoom F6-Next F7-Trace F8-Step F9-Run F10-Menu

```

3) Нормальное завершение при положительном делимом и делителе, $Y > 0$, без остатка.

$$A = 1997_{10} = 7CD_{16} = 111\ 1100\ 1101_2$$

$$B = 744_{10} = 2E8_{16} = 10\ 1110\ 1000_2$$

$$C = 64_{10} = 40_{16} = 100\ 0000_2$$

```

File View Run Breakpoints Data Options Window Help
[.] CPU 80486 ds:0002 = 02E8 1=[ ] [ ]
cs:0003 8ED8 mov ds,ax ax 4ACF c=0
cs:0005 C6060E0000 mov byte ptr [000E],00 bx 019A z=0
cs:000A C6060F0000 mov byte ptr [000F],00 cx 0000 s=0
cs:000F A10200 mov ax,[0002] dx 0220 o=0
cs:0012 D1E0 shl ax,1 si 0000 p=0
cs:0014 7104 jno 001A di 0000 a=0
cs:0016 FE060F00 inc byte ptr [000F] bp 0000 i=1
cs:001A 8B1E0000 mov bx,[0000] sp 0000 d=0
cs:001E 03C3 add ax,bx ds 4ACF
cs:0020 7104 jno 0026 es 4ABF
cs:0022 FE060F00 inc byte ptr [000F] ss 4ACE
cs:0026 8B1E0400 mov bx,[0004] cs 4AD1
cs:002A 8B161400 mov dx,[0014] ip 000F
cs:002E 8BCA mov cx,dx
cs:0030 2700 and dx,ax

A B C
ds:0000 CD 07 E8 02 40 00 00 00 --w00
ds:0008 00 00 00 00 00 00 00 00
ds:0010 00 00 00 00 00 00 00 00 A
ds:0018 00 00 00 00 00 00 00 00
ds:0020 B8 CF 4A 8E D8 C6 06 0E 1-J0+|+n
ss:0000 0495
ss:0006 0000
ss:0004 BDF8
ss:0002 111E
ss:0000 01D4

F1-Help F2-Bkpt F3-Mod F4-Here F5-Zoom F6-Next F7-Trace F8-Step F9-Run F10-Menu

```

```

File View Run Breakpoints Data Options Window Help
[.] CPU 80486 1=[ ] [ ]
cs:0113 C6060E0001 mov byte ptr [000E],01 ax 4C00 c=1
cs:0118 B8004C mov ax,4C00 bx 019A z=0
cs:011D CD15 int 15 cx 0000 s=1
cs:011D 0000 add [bx+si],al dx 0220 o=0
cs:011F 0000 add [bx+si],al si 0000 p=1
cs:0121 0000 add [bx+si],al di 0000 a=1
cs:0123 0000 add [bx+si],al bp 0000 i=1
cs:0125 0000 add [bx+si],al sp 0000 d=0
cs:0127 0000 add [bx+si],al ds 4ACF
cs:0129 0000 add [bx+si],al es 4ABF
cs:012B 0000 add [bx+si],al ss 4ACE
cs:012D 0000 add [bx+si],al cs 4AD1
cs:012F 0000 add [bx+si],al ip 011B
cs:0131 0000 add [bx+si],al
cs:0133 0000 add [bx+si],al

A B C Y1
ds:0000 CD 07 E8 02 40 00 03 00 --w00
ds:0008 40 67 9A 01 00 00 01 00 - ошибка
ds:0010 Y2 2 chast lost 00 успешное завершение
ds:0018 00 00 00 00 00 00 00 00
ds:0020 B8 CF 4A 8E D8 C6 06 0E 1-J0+|+n
ss:0000 6864
ss:0006 742E
ss:0004 706C
ss:0002 6568
ss:0000 6474

F1-Help F2-Bkpt F3-Mod F4-Here F5-Zoom F6-Next F7-Trace F8-Step F9-Run F10-Menu

```

Проверка:

$$Y = (2 \cdot 744 + 1997) \cdot 64 / (744 - 200) = 410_{10} = 19A_{16}$$

4) Нормальное завершение при положительном делимом и отрицательном делителе, $Y < 0$, без остатка.

$$\begin{aligned}
A &= 3926_{10} = F56_{16} = 1111\ 0101\ 0110_2 \\
B &= -1200_{10} = -4B0_{16} = -100\ 1011\ 0000_2 \\
[B]_2 &= 1111\ 1011\ 0101\ 0000_2 = FB50_{16} \\
C &= 2000_{10} = 7D0_{16} = 111\ 1101\ 0000_2
\end{aligned}$$

The image displays two screenshots of an 80486 CPU emulator interface. The top screenshot shows the assembly code window with instructions like `mov ax, 4ACF`, `mov ds, ax`, and `mov byte ptr [000E], 00`. The bottom screenshot shows the execution of the `int 15` instruction, which is labeled as `CD15`. The register window on the right shows the state of the CPU registers, including `ax`, `bx`, `cx`, `dx`, `si`, `di`, `bp`, `sp`, `ds`, `es`, `ss`, `cs`, and `ip`. The memory dump at the bottom shows the contents of memory locations, including the instruction `U*PJH.` and the message `успешное завершение` (successful completion).

Проверка:

$$Y = (2 * (-1200) + 3926) * 2000 / (-1200 - 200) = -2180_{10} = -884_{16} = -1000 \ 1000 \ 0100_2$$

$$[Y]_2 = 1111 \ 0111 \ 0111 \ 1100_2 = F77C_{16}$$

- 5) Нормальное завершение при положительном делимом и делителе, $Y > 0$, с остатком.

$$A = 160_{10} = A0_{16} = 1010 \ 0000_2$$

$$B = 744_{10} = 2EB_{16} = 11 \ 1110 \ 1011_2$$

$$C = 100_{10} = 64_{16} = 0110\ 0100_2$$

The first screenshot shows the initial state of the CPU 80486. The assembly code is as follows:

```

cs:0003 8ED8      mov     ds,ax
cs:0005 C6060E0000    mov     byte ptr [000E],00
cs:000A C6060F0000    mov     byte ptr [000F],00
cs:000F A10200      mov     ax,[0002]
cs:0012 D1E0      shl     ax,1
cs:0014 7104      jno     001A
cs:0016 FE060F00    inc     byte ptr [000F]
cs:001A 8B1E0000    mov     bx,[0000]
cs:001E 03C3      add     ax,bx
cs:0020 7104      jno     0026
cs:0022 FE060F00    inc     byte ptr [000F]
cs:0026 8B1E0400    mov     bx,[0004]
cs:002A 8B161400    mov     dx,[0014]
cs:002E 0BCA      mov     cx,dx
cs:0030 2704      rnd     dx,ax

```

The second screenshot shows the execution of a loop that calculates Y and the remainder (ost). The assembly code is as follows:

```

cs:0113 C6060E0001    mov     byte ptr [000E],01
cs:0118 B0004C      mov     ax,4C00
cs:011B CD15      int     15
cs:011D 0000      add     [bx+si],al
cs:011F 0000      add     [bx+si],al
cs:0121 0000      add     [bx+si],al
cs:0123 0000      add     [bx+si],al
cs:0125 0000      add     [bx+si],al
cs:0127 0000      add     [bx+si],al
cs:0129 0000      add     [bx+si],al
cs:012B 0000      add     [bx+si],al
cs:012D 0000      add     [bx+si],al
cs:012F 0000      add     [bx+si],al
cs:0131 0000      add     [bx+si],al
cs:0133 0000      add     [bx+si],al

```

Проверка:

$$Y = (2 \cdot 744 + 160) \cdot 100 / (744 - 200) = 302_{10} = 12E_{16} = 1\ 0010\ 1110_2$$

$$\text{ost} = (2 \cdot 744 + 160) \cdot 100 - 302 \cdot (744 - 200) = 512_{10} = 200_{16} = 10\ 0000\ 0000_2$$

- 6) Нормальное завершение при положительном делимом и отрицательном делителе, $Y < 0$, с остатком.

$$A = 3926_{10} = F56_{16} = 1100\ 1000_2$$

$$B = -1200_{10} = -4B0_{16} = -100\ 1011\ 0000_2$$

$[B]_2 = 1111\ 1011\ 0101\ 0000_2 = FB50_{16}$

$C = 2013_{10} = 7DD_{16} = 111\ 1011\ 1011_2$

The screenshot shows the 80486 CPU emulator interface. The assembly window displays the following code:

```

cs:0000 B8CF4A    mov     ax,4ACF
cs:0003 8ED8      mov     ds,ax
cs:0005 C6060E0000    mov     byte ptr [000E],00
cs:000A C6060F0000    mov     byte ptr [000F],00
cs:000F A10200      mov     ax,[0002]
cs:0012 D1E0      shl     ax,1
cs:0014 7104      jno     001A
cs:0016 FE060F00    inc     byte ptr [000F]
cs:001A 8B1E0000      mov     bx,[0000]
cs:001E 03C3      add     ax,bx
cs:0020 7104      jno     0026
cs:0022 FE060F00    inc     byte ptr [000F]
cs:0026 8B1E0400      mov     bx,[0004]
cs:002A 8B161400      mov     dx,[0014]
cs:002E 8B161400      mov     cx,dx
  
```

The memory dump shows the state of memory segments:

Segment	Address	Value
ds	0000	56 0F 50 FB DD 07 00 00
ds	0008	00 00 00 00 00 00 00 00
ds	0010	00 00 00 00 00 00 00 00
ds	0018	00 00 00 00 00 00 00 00
ds	0020	B8 CF 4A 0E 08 C6 06 0E
ss	0000	6864
ss	0006	742E
ss	0004	706C
ss	0002	6568
ss	0000	6474

The status bar at the bottom shows: 1-Help F2-Bkpt F3-Mod F4-Here F5-Zoom F6-Next F7-Trace F8-Step F9-Run F10-Menu

The screenshot shows the 80486 CPU emulator interface. The assembly window displays the following code:

```

cs:0110 C6060E0001    mov     byte ptr [000E],01
cs:0115 B8004C      mov     ax,4C00
cs:0118 CD15      int     15
cs:011A 0000      add     [bx+si],al
cs:011C 0000      add     [bx+si],al
cs:011E 0000      add     [bx+si],al
cs:0120 0000      add     [bx+si],al
cs:0122 0000      add     [bx+si],al
cs:0124 0000      add     [bx+si],al
cs:0126 0000      add     [bx+si],al
cs:0128 0000      add     [bx+si],al
cs:012A 0000      add     [bx+si],al
cs:012C 0000      add     [bx+si],al
cs:012E 0000      add     [bx+si],al
cs:0130 0000      add     [bx+si],al
  
```

The memory dump shows the state of memory segments:

Segment	Address	Value
ds	0000	56 0F 50 FB DD 07 2E 00
ds	0008	5E DF 6E F7 EE 00 01 00
ds	0010	00 00 00 00 00 00 00 00
ds	0018	00 00 00 00 00 00 00 00
ds	0020	B8 CF 4A 0E 08 C6 06 0E
ss	0000	6864
ss	0006	742E
ss	0004	706C
ss	0002	6568
ss	0000	6474

The status bar at the bottom shows: 1-Help F2-Bkpt F3-Mod F4-Here F5-Zoom F6-Next F7-Trace F8-Step F9-Run F10-Menu

Проверка:

$Y = (2*(-1200)+3926)*2013/(-1200-200) = -2194_{10} = -892_{16} = -1000\ 1001\ 0010_2$

$[Y]_2 = 1111\ 0111\ 0110\ 1110$

$ost = (2*(-1200)+3926)*2013-(-2194)*(-1200-200) = 238_{10} = EE_{16}$

Вывод: создали программу на языке ассемблера для вычисления выражения $Y=(2*B+A)*C/(B-200)$. Умножение и деление были реализованы через операции сдвига, сложения и вычитания. Возможные исключения обработаны.