Министерство образования и науки Российской федерации

Федеральное государственное учреждение высшего образования

РЫБИНСКИЙ ГОСУДАРСТВЕННЫЙ АВИАЦИОННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ имени П. А. СОЛОВЬЕВА

Институт Информационных технологий и систем управления

Кафедра вычислительных систем

Пояснительная записка

к программной части курсового проекта

по дисциплине «Микропроцессорные системы»

на тему «Счётчик купюр»

Студент группы ИВБ1–20 Морозов А. А.  
Руководитель канд. тех. наук, проф Комаров В. М.  
Нормоконтролер канд. тех. наук, проф Комаров В. М.

Рыбинск 2023

Содержание

[Введение 3](#_Toc136706964)

[1 Анализ технического задания 4](#_Toc136706965)

[2 Декомпозиция программы 6](#_Toc136706966)

[3 Разработка структуры данных программы 9](#_Toc136706967)

[4 Алгоритмизация программы 1](#_Toc136706978)1

[5 Кодирование программы 2](#_Toc136706979)8

[6 Тестирование и отладка программы 2](#_Toc136706980)9

[7 Руководство пользователя 3](#_Toc136706981)1

[Заключение 32](#_Toc136706982)

[Список использованных источников 3](#_Toc136706983)3

[Приложение А 3](#_Toc136706984)4

##### Введение

Использование микропроцессорной техники является одним из средств автоматизации технологических процессов. При современном уровне развития элементной базы вычислительной техники для построения устройств обработки информации целесообразно использовать микропроцессорные системы. Использование микропроцессоров обеспечивает возможность реализации таких устройств на одной небольшой печатной плате.

##### 1 Анализ технического задания

Из анализа технического задания следует, устройство должно обеспечивать:

1. для выбора номинала купюры воспользуемся клавишами;
2. в зависимости от нажатой клавиши будет выбран номинал купюры
3. если сумма купюр будет равна 500000 или количество бракованных

или обычных купюр будет равняться 100, то дальнейший счёт нужно прекратить;

1. отображение количества купюр до 100;
2. при выборе другого номинала во время счёта устройство должно

посчитать используя новый номинал

1. эмуляцию ввода бракованной купюры;
2. отображение количества бракованных купюр до 100;
3. возможность остановки работы устройства;
4. возможность сбросить накопленную сумму и количество купюр.

На основании перечисленных требований можно представить разрабатываемое устройство в виде «черной сферы» (рисунок 1.1), а также изобразить лицевую панель устройства (рисунок 1.2). Исходя из этих представлений начинается техническое проектирование устройства.

****

Рисунок 1.1 – Представление счётчика купюр в виде «чёрной сферы»



Рисунок 1.2 – Лицевая панель устройства охранной сигнализации

##### 2 Декомпозиция программы

Анализируя требования к программе с учётом лицевой панели устройства, получим исходную схему представления поставленной задачи.



Рисунок 2.1 – Исходная схема представления задачи «Счётчик купюр»

Из этой схемы следует, что проектируемая программа должна обрабатывать входную информацию с 7 кнопок выбора номинала купюры и 4 кнопок управления, выходная информация с которых имеет два значения: лог. «0» и лог. «1», отображать выходную информацию на 3 дисплеях, состоящих из знакосинтезирующих семисегментных индикаторов, и совокупности светодиодов.

На первом этапе (рисунок 2.2) представим программу в виде двух задач: «Обработка входной информации», «Обработка выходной информации». Подзадачи, выделенные на первом этапе, являются сложными и требуют дальнейшего разбиения.



Рисунок 2.2 – Статическая модель программы (после первого этапа декомпозиции)

На втором этапе (рисунок 2.3) подзадачу «Обработка выходной информации» разобьём на подзадачи «Эмуляция прокрутки барабана», «Контроль переполнения», «Изменение суммы и количества бракованных и нормальных купюр», «Вывод числовой информации».

Подзадачи, выделенные на втором этапе, являются простыми и не требуют дальнейшей декомпозиции.



Рисунок 2.3 – Статическая модель программы (после второго этапа декомпозиции)

##### 3 Разработка структуры данных программы

Процесс решения любой задачи состоит из активации тех или иных программных модулей, выполняющих некоторую подзадачу. Передача информации между модулями осуществляется с помощью наборов данных.

Дальнейшее проектирование программы заключается в переходе от ее статической модели к динамической, т. е. к ее алгоритмическому описанию.



Таблица 3.1 – Структура данных программы

##### 4 Алгоритмизация программы

Вычислительный процесс, реализуемый программой, представляет собой последовательность дискретных шагов по преобразованию данных.

Для решения поставленной задачи необходимы следующие программные модули с соответствующими символьными именами:

– «Функциональная подготовка» (*FuncPrep*);

– «Ввод с кнопок управления и кнопок выбора номинала» (*ReadInput*);

– «Эмуляция прокрутки барабана» (*Simul*);

– «Контроль переполнения» (*OverflowCheck*);

– «Изменение данных» (*AccumulationSumm*);

– «Вывод числовой информации» (*DisplayOutput*);

Символьные имена присвоены соответствующим программным модулям с целью их дальнейшего использования. Выбранные имена отражают содержательный смысл этих модулей.

Последовательность модулей в логической конструкции «Следование» определяется логикой решения задачи.

Алгоритм макроуровня программы проектируемого устройства в одноуровневом представлении приведен на рисунке 4.1. Для повышения информативности ГСА на ней изображены входные и выходные данные, обрабатываемые каждым программным модулем. Это позволяет проследить, что каждый модуль располагается в той точке программного кольца, в которой существуют все обрабатываемые им данные.

Далее необходимо перейти к двухуровневому представлению алгоритма. Этот переход осуществляется формально. Алгоритм программы проектируемого устройства в двухуровневом представлении приведен на рисунке 4.2. Из этого алгоритма следует, что на макроуровне программы находятся лишь команды вызова программных модулей.

После разработки алгоритма макроуровня требуется разработать алгоритмы всех программных модулей нижнего уровня. Алгоритмы программных модулей и подмодулей приведены на рисунках 4.3 – 4.15.



Рисунок 4.1 – Алгоритм программы «Счётчик купюр» (одноуровневое представление)



Рисунок 4.2 – Алгоритм программы «Счётчик купюр» (двухуровневое представление)

*Модуль «*Ввод с кнопок управления и кнопок выбора номинала*» (ReadInput)*

Этот модуль служит для ввода состояния кнопок управления и номинала. Для удобства хранения номинала купюры используем двоично-десятичный распакованный формат .

ГСА этого модуля приведена на рисунке 4.3.

Электромеханическим переключателям свойственно явление дребезга контактов, который необходимо гасить. Гашение дребезга контактов можно выполнить программным путем. ГСА подмодуля гашения дребезга *VibrDelete* приведена на рисунке 4.4. Критерием окончания дребезга является считывание с кнопок одного и того же состояния заданное количество раз.



Рисунок 4.3 – Алгоритм модуля «Ввод с кнопок постановки под охрану» (общий алгоритм)

**

Рисунок 4.4 – Алгоритм подмодуля «Гашение дребезга»

*Модуль «*Эмуляция прокрутки барабана*» (Simul)*

Этот модуль служит для эмуляции работы барабана и модификации флага суммы. Для эмуляции работы барабана воспользуемся программной задержкой.

ГСА этого модуля приведена на рисунке 4.5.



Рисунок 4.5 – Алгоритм модуля «Эмуляция прокрутки барабана» (общий алгоритм)

*Модуль «*Контроль переполнения*» (OverflowCheck)*

Этот модуль служит для проверки проверки на переполнения количества обычных и бракованных купюр.

ГСА этого модуля приведена на рисунке 4.6.



Рисунок 4.6 – Алгоритм модуля «Контроль переполнения» (общий алгоритм)

Подмодуль «Проверка количества обычных купюр» (*OneHundredProverka*)

Этот подмодуль нужен для проверки на переполнение количества обычных купюр. Для удобства этот модуль разбит на ряд подмодулей.

ГСА этого подмодуля приведена на рисунке 4.7.



Рисунок 4.7 – Алгоритм подмодуля «Проверка количества обычных купюр»

Подмодуль «Проверка количества бракованных купюр» (*OneHundredProverka*)

Этот подмодуль нужен для проверки на переполнение количества бракованных купюр.

ГСА этого подмодуля приведена на рисунке 4.8.



Рисунок 4.8 – Алгоритм подмодуля «Проверка количества бракованных купюр»

*Модуль «*Изменение данных*» (AccumulationSumm).*

Этот модуль служит для изменения суммы купюр и количества купюр в зависимости от флагов. Для удобства сброса он реализован в подмодуле.

ГСА этого модуля приведена на рисунке 4.9.



Рисунок 4.9 – Алгоритм модуля «Изменение данных» (общий алгоритм)

Подмодуль «Сброс к стартовым значениям» (Sbros).

Этот подмодуль нужен для реализации сброса всех значений к стартовым используя подмодуль «Инициализация».

ГСА этого подмодуля приведена на рисунке 4.10.



Рисунок 4.10 – Алгоритм подмодуля «Сброс к стартовым значениям»

Подмодуль «Инициализация» (*Init*).

Этот подмодуль используется для инициализации стартовых значений программы.

ГСА этого подмодуля приведена на рисунке 4.11.



Рисунок 4.11 – Алгоритм подмодуля «Инициализация»

*Модуль «*Вывод числовой информации*» (DisplayOutput)*

Этот модуль предназначен для вывод суммы и количества купюр на индикаторы. Для удобства этот модуль разбит на ряд подмодулей.

ГСА модуля изображена на рисунке 4.12.



Рисунок 4.12 – Алгоритм модуля «Вывод числовой информации»

Подмодуль «Вывод суммы» (*SumOut*).

Этот подмодуль выводит на индикаторы значение суммы из памяти устройства.

ГСА этого подмодуля приведена на рисунке 4.13.



Рисунок 4.13 – Алгоритм подмодуля «Вывод суммы»

Подмодуль «Вывод количества купюр» (*CntOut*).

Этот подмодуль выводит на индикаторы значение суммы из памяти устройства.

ГСА этого подмодуля приведена на рисунке 4.14.



Рисунок 4.14 – Алгоритм подмодуля «Вывод количества купюр»

Модуль «Функциональная подготовка» (*FuncPrep*)

Этот модуль предназначен для начальной установки некоторых наборов данных или служебных ячеек, которые служат для хранения вспомогательной информации. Для удобства этот модуль разбит на ряд подмодулей.

ГСА этого модуля приведена на рисунке 4.15.



Рисунок 4.14 – Алгоритм модуля «Функциональная подготовка»

ГСА подмодуля «Инициализация» приведена на рисунке 4.11.

Подмодуль «Копирование массива цифр и таблицы преобразования из сегмента кода в сегмент данных» (*CopyArr*)

Этот подмодуль служит для копирования данных, необходимых для табличного преобразования значений суммы, количества обычных и бракованных купюр в их изображения, которые нужны для вывода на дисплеи.

ГСА этого модуля приведена на рисунке 4.15.



Рисунок 4.15 – Алгоритм подмодуля «Копирование массива цифр и таблицы преобразования из сегмента кода в сегмент данных»

5 Кодирование программы

Кодирование программы представляет собой запись алгоритмов проектируемой программы на языке ассемблер.

При кодировании программы используются ранее разработанные структура данных (таблица 3.1) и ГСА (рисунки 4.1 – 4.15).

Для полного кодирования программы остается лишь определить таблицу преобразования кодов десятичных цифр из массива отображения в семисегментные коды, которые выводятся на семисегментные знакосинтезирующие индикаторы для отображения соответствующей цифры.

Эта таблица зависит от выбранного варианта подключения сегментов индикатора к разрядам управляющего порта вывода. Для принятого варианта зажигания индикаторов нулём преобразование должно выполняться по таблице 5.1.

Исходный текст проектируемой программы приведен в Приложении А.

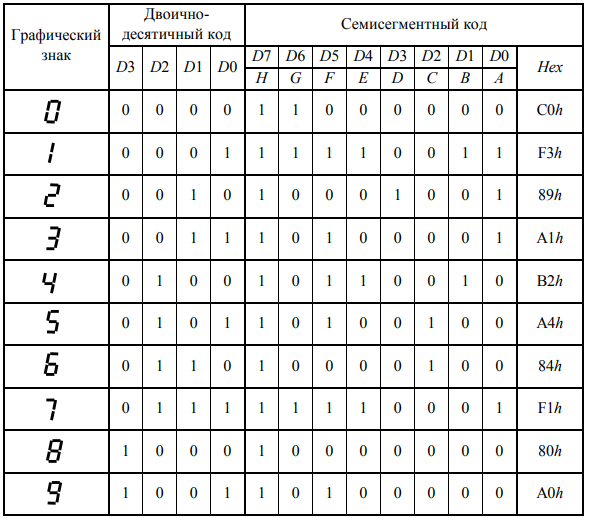


Таблица 5.1 – Кодирования кодов

##### 6 Тестирование и отладка программы

Для тестирования и отладки разработанной программы будем использовать интегрированную программную среду *Demis*. Эта среда позволяет на архитектурном уровне представить проектируемое устройство и работать с ним как с реальным устройством.

Поскольку архитектура процессора и памяти одинаковы практически для всех устройств, то нет никакой необходимости представлять их в отладочной среде. Моделированию подлежит лишь архитектура интерфейса проектируемого устройства рисунок 6.1.

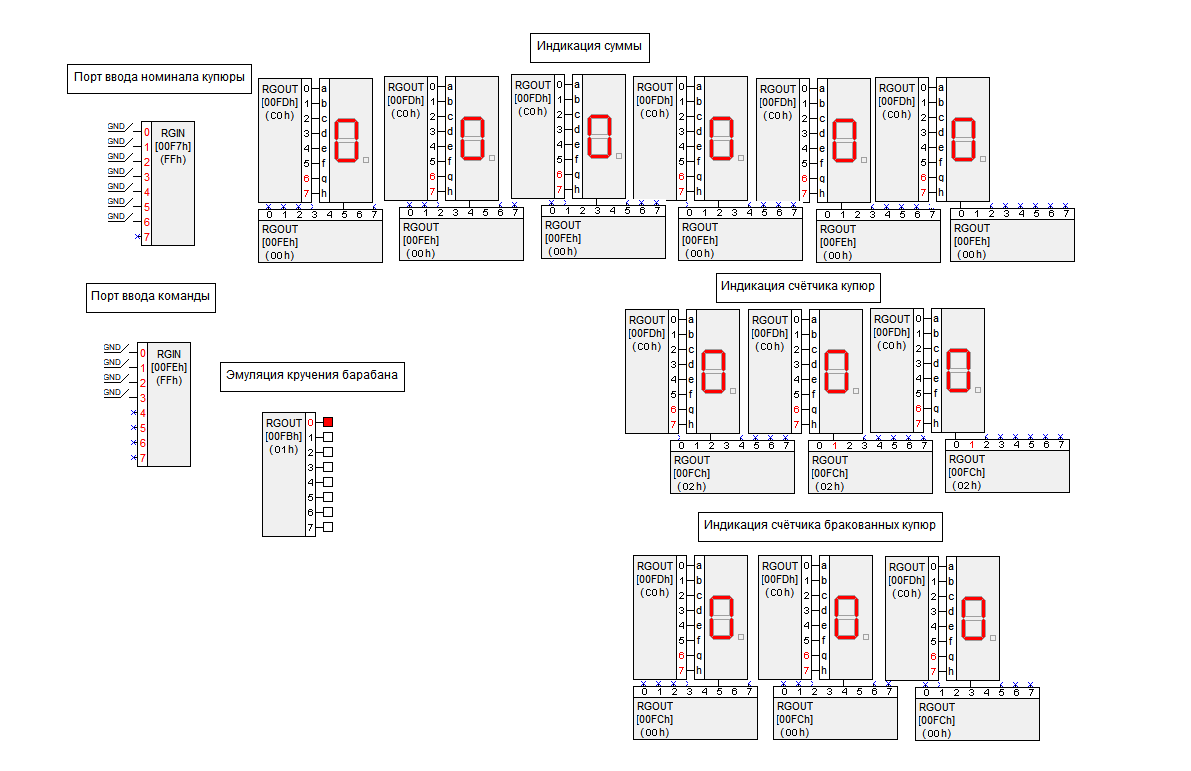


Рисунок 6.1 – Архитектура счётчика купюр в среде *Demis*

В состав архитектуры интерфейса устройства входят 2 порта ввода, 4 порта вывода, 7 кнопок без фиксации – выборка номинала купюры, 4 кнопки без фиксации – управление устройством, 6 знакосинтезирующих индикатора для отображения суммы купюр, 3 знакосинтезирующих индикатора для отображения количества обычных купюр, 3 знакосинтезирующих индикатора для отображения бракованных купюр. Для составления архитектуры устройства выберем необходимые интерфейсные элементы и разместим их на экране, соединяя в соответствии с архитектурой. После этого установим интерфейсным элементам необходимые свойства:

* для портов ввода/вывода зададим адреса в соответствии с исходным текстом программы;
* для двоичных индикаторов выберем уровень зажигания, равный «Зажигается нулём»;
* для кнопок определим исходное состояние, соответствующее ненажатой кнопке, как «нормально незамкнута».

Далее с помощью текстового редактора занесём исходный текст программы.

После занесения программы выполним ее ассемблирование и компоновку, т. е. сборку программы, а затем выполним ее.

##### 7 Руководство пользователя

После запуска программы на всех дисплеях отображаются нули.

При нажатии кнопки «Старт» барабан начнёт крутиться (путём последовательного пересщёлкивания светодиодов). Если перед этим был выбран номинал купюры, то дисплей отображения суммы и счётчик купюр будут увеличиваться на номинал введённой купюры и на единицу соответственно.

Во время прокрутки барабана можно остановить его нажав на кнопку «Стоп». Барабан остановится, а дисплеи будут отображать последнюю информацию.

Есть возможность нажать на кнопку «Сброс». Это приведёт к остановке счёт и сбросу к стартовым данным.

При нажатии на кнопку «Бракованная» считываемая в данный момент купюра будет помечена как бракованная и не будет засчитана в конечную сумма, а счётчик бракованных купюр будет увеличен на 1.

При количестве купюр равным 100 или сумме равной 500000 программа автоматически останавливается. Дисплеи будут отображать последнюю информацию

##### Заключение

##### При разработке программной части курсового проекта была учтена необходимость дальнейшего перехода к аппаратной части, поэтому программа разрабатывалась такой, чтобы претерпеть наименьшие изменения при этом переходе.

##### В процессе разработки программного обеспечения была проведена декомпозиция задачи, разработаны структура данных и статическая модель программы, разработаны алгоритмы отдельных модулей, необходимых для решения поставленной задачи.

##### Программа была проверена и отлажена в среде *Demis*, так же во время отладки были подобраны наиболее оптимальные значения констант. Программа показала себя работоспособной. Таким образом поставленная задача по разработке программной части к курсовому проекту была выполнена.

##### Список использованных источников

1. Комаров В. М. Микропроцессорные системы: Учебное пособие /РГАТА.-Рыбинск, 2006. – 92 с.

2 Комаров В. М. Микропроцессорные системы. Проектирование аппаратного и программного обеспечения: Учебное пособие. – 2 изд. перераб. и доп. – Рыбинск, РГАТА, 2004. – 176 с.

##### Приложение А

.386

;Задайте объём ПЗУ в байтах

RomSize EQU 4096

SumPort = 0FDh ; 2

SumPowerPort = 0FEh ; 1

CntPort = 0FDh ; 2

CntPowerPort = 0FCh ; 3

KbdPort = 0F7h ; 0

IndPort = 0FBh ; 4

ControlPort = 0FEh ; 1

NMax = 100

IntTable SEGMENT use16 AT 0

;Здесь размещаются адреса обработчиков прерываний

IntTable ENDS

Data SEGMENT use16 AT 40h

;Здесь размещаются описания переменных

DataHexArr db 10 dup(?)

DataHexTabl db 10 dup(?)

DataTable dd 7 dup(?)

ErrTable db 5 dup (?)

Res db 6 dup (?)

SelectedNumber DD ?

OldButton db ?

OldCntrl db ?

StopFlag db ?

BrakFlag db ?

OneHundredFlag db ?

SumFlag db ?

SbrosFlag db ?

Buffer dw ?

Cnt DD ?

CntBrak DD ?

Time DD ?

TimeEndFlag DB ?

Data ENDS

;Задайте необходимый адрес стека

Stk SEGMENT use16 AT 2000h

;Задайте необходимый размер стека

DW 16 dup (?)

StkTop Label Word

Stk ENDS

InitData SEGMENT use16

InitDataStart:

;Здесь размещаются описания констант

InitDataEnd:

InitData ENDS

Code SEGMENT use16

;Здесь размещаются описания констант

ASSUME cs:Code,ds:Data,es:Data

HexArr DB 00h,01h,02h,03h,04h,05h,06h,07h,08h,09h

HexTabl DB 0C0H, 0F3h, 89h, 0A1h, 0B2h, 0A4h, 84h, 0F1h, 80h, 0A0h

Table DD 0500h, 010000h, 020000h, 050000h, 01000000h, 02000000h, 05000000h

FuncPrep PROC NEAR

call Init

call CopyArr

FuncPrep ENDP

Init PROC NEAR

xor ax, ax

mov StopFlag, 0FFh

mov BrakFlag, 00h

mov OneHundredFlag, 00h

mov SumFlag, 00h

mov SbrosFlag, 00h

mov word ptr Cnt, ax

mov word ptr Cnt+2, ax

mov word ptr CntBrak, ax

mov word ptr CntBrak+2, ax

mov OldButton, al

mov OldCntrl, al

mov word ptr Res, ax

mov word ptr Res+2, ax

mov word ptr Res+4, ax

mov word ptr SelectedNumber, ax

mov word ptr SelectedNumber+2, ax

mov TimeEndFlag, 01h

mov Buffer, 0100h

mov ax, Buffer

mov al, ah

out IndPort, al

xor ax, ax

RET

Init ENDP

Simul PROC NEAR

MOV CX, AX

MOV AX, Buffer

cmp StopFlag, 0FFh

je Timer1

cmp OneHundredFlag, 0FFh

je Timer1

jmp Timer2

Timer0: ; Таймер

SUB word ptr Time, 1

SBB word ptr Time+2, 0

MOV SI, word ptr Time

OR SI, word ptr Time+2

MOV TimeEndFlag, 0

JNZ Timer1

MOV TimeEndFlag, 01h

Timer2: MOV AL,AH

CMP TimeEndFlag, 01h

JNZ Timer0

Out IndPort, AL

cmp AL, 80h

jne Timer3

mov SumFlag, 0FFh

Timer3: ROL AH, 1

MOV word ptr Time, 0007h

MOV word ptr Time+2, 0000h

JMP Timer0

Timer1: MOV Buffer, AX

MOV AX, CX

ret

Simul ENDP

ReadInput PROC Near

xor ah, ah

mov dx, ControlPort

in al, dx

call VibrDestr

xor ah, ah

cmp al, OldCntrl

jne m3

m6: xor ah, ah

cmp SbrosFlag, 0FFh

je m1

cmp OneHundredFlag, 0FFh

je m1

jmp m4

m3: mov OldCntrl, al

cmp al, 0ffh

je m6

m5: inc ah

shr al, 1

jc m5

dec ah

cmp ah, 03h

jne NoSbros

mov SbrosFlag, 0FFh

NoSbros: cmp ah, 02h

jb m11

mov BrakFlag, 0FFh

jmp m6

m11: cmp ah, 01h

jne m12

mov StopFlag, 0FFh

jmp m6

m12: mov StopFlag, 00h

jmp m6

m4: mov dx, KbdPort

in al, dx

call VibrDestr

xor ah, ah

cmp al, OldButton

je m1

mov OldButton, al

cmp al, 0ffh

je m1 ;Если нет символов для добавления (не нажата ни одна из кнопок)

m2:

inc ah

shr al, 1

jc m2

dec ah

xor al, al

lea BX, Table

lea DI, SelectedNumber

shl ah, 2

MOV CX, 04h

ReadInput1: add al, ah

xlat

mov byte ptr [DI], al

inc BX

inc DI

loop ReadInput1

m1: RET

ReadInput ENDP

AccumulationSumm PROC Near

cmp SbrosFlag, 0FFh

jne M12

call Sbros

M12: cmp OneHundredFlag, 0FFh

je M7

cmp StopFlag, 0FFh

je M7

cmp SumFlag, 00h

je M7

xor ax,ax

cmp BrakFlag, 0FFh

je M10

cmp word ptr SelectedNumber+2, 0

JNZ M8

cmp word ptr SelectedNumber, 0

JZ M7

M8: mov ax, word ptr Cnt

inc ax

AAA

mov word ptr Cnt, ax

M9: mov SumFlag, 00h

mov CX, 04h

lea SI, Res

lea BX, SelectedNumber

AccSum1: mov ax, word ptr [SI]

ADD al, byte ptr [BX]

AAA

mov word ptr [SI], ax

inc SI

inc BX

loop AccSum1

CMP Res+4, 09h

JBE M7

mov Res+4, 0h

INC [Res+5]

JMP M7

M10: mov BrakFlag, 00h

mov SumFlag, 00h

mov ax, word ptr CntBrak

inc ax

AAA

mov word ptr CntBrak, ax

M7:

mov bp, word ptr SelectedNumber

and bp, 00FFh

ret

AccumulationSumm ENDP

OneHundredProverka PROC NEAR

CMP byte ptr Cnt+1, 09h

JBE HundredRet

mov StopFlag, 0FFh

mov OneHundredFlag, 0FFh

mov byte ptr Cnt+1, 00h

mov byte ptr Cnt+2, 01h

HundredRet: ret

OneHundredProverka ENDP

OneHundredProverkaBrak PROC NEAR

CMP byte ptr CntBrak+1, 09h

JBE HundredBrakRet

mov StopFlag, 0FFh

mov OneHundredFlag, 0FFh

mov byte ptr CntBrak+1, 00h

mov byte ptr CntBrak+2, 01h

HundredBrakRet: ret

OneHundredProverkaBrak ENDP

OverflowCheck PROC NEAR

call OneHundredProverka

call OneHundredProverkaBrak

ret

OverflowCheck ENDP

SumOut PROC NEAR ;Выводим сумму на индикаторы

xor cx, cx

mov cl, 01h

lea bx, DataHexTabl

lea SI, Res

SumOut1: mov ah, [SI]

mov al, ah

xlat

;not al ;табличное преобразование

out SumPort, al ;выводим на индикатор

mov al, cl

out SumPowerPort, al ;зажигаем индикатор

mov al,00h

out SumPowerPort, al ;гасим индикатор

shl cl, 1

inc SI

cmp cl, 20h

jbe SumOut1

xor ah, ah

xor cx, cx

SumOutRet: ret

SumOut ENDP

CntOut PROC NEAR

xor cx, cx

mov cl, 01h

lea bx, DataHexTabl

lea SI, byte ptr Cnt

CntOut1: mov ah, [SI]

mov al, ah

xlat ;табличное преобразование

out CntPort, al ;выводим на индикатор

mov al, cl

out CntPowerPort, al ;зажигаем индикатор

mov al,00h

out CntPowerPort, al ;гасим индикатор

shl cl, 1

inc SI

cmp cl, 04h

jbe CntOut1

xor ah, ah

xor cx, cx

mov cl, 08h

lea bx, DataHexTabl

lea SI, byte ptr CntBrak

CntOut2: mov ah, [SI]

mov al, ah

xlat ;табличное преобразование

out CntPort, al ;выводим на индикатор

mov al, cl

out CntPowerPort, al ;зажигаем индикатор

mov al,00h

out CntPowerPort, al ;гасим индикатор

shl cl, 1

inc SI

cmp cl, 20h

jbe CntOut2

xor ah, ah

xor cx, cx

ret

CntOut ENDP

Sbros PROC NEAR

call Init

ret

Sbros ENDP

DisplayOutput PROC NEAR

call SumOut

call CntOut

ret

DisplayOutput ENDP

VibrDestr PROC NEAR

VD1: mov ah,al ;Сохранение исходного состояния

mov ch,0 ;Сброс счётчика повторений

VD2: in al,dx ;Ввод текущего состояния

cmp ah,al ;Текущее состояние=исходному?

jne VD1 ;Переход, если нет

inc ch ;Инкремент счётчика повторений

cmp ch,NMax ;Конец дребезга?

jne VD2 ;Переход, если нет

mov al,ah ;Восстановление местоположения данных

ret

VibrDestr ENDP

CopyArr PROC NEAR

MOV CX, 10 ;Загрузка счётчика циклов

LEA BX, HexArr ;Загрузка адреса массива цифр

LEA BP, HexTabl ;Загрузка адреса таблицы преобразования

LEA DI, DataHexArr ;Загрузка адреса массива цифр в сегменте данных

LEA SI, DataHexTabl ;Загрузка адреса таблицы преобразования в сегменте данных

CopyArr0:

MOV AL, CS:[BX] ;Чтение цифры из массива в аккумулятор

MOV [DI], AL ;Запись цифры в сегмент данных/DataHexArr

INC BX ;Модификация адреса HexArr

INC DI ;Модификация адреса DataHexArr

MOV AH, CS:[BP] ;Чтение графического образа из таблицы преобразования

MOV [SI], AH ;Запись графического образа в сегмент данных/DataHexTabl

INC BP ;Модификация адреса HexTabl

INC SI ;Модификация адреса DataHexTabl

LOOP CopyArr0

MOV CX, 14 ;Загрузка счётчика циклов

LEA BP, Table ;Загрузка адреса таблицы преобразования

LEA SI, DataTable ;Загрузка адреса таблицы преобразования в сегменте данных

CopyArr2:

MOV AH, CS:[BP] ;Чтение графического образа из таблицы преобразования

MOV [SI], AH ;Запись графического образа в сегмент данных/DataTable

MOV AL, CS:[BP+1] ;Чтение графического образа из таблицы преобразования

MOV [SI+1], AL ;Запись графического образа в сегмент данных/DataTable

INC BP ;Модификация адреса Table

INC SI ;Модификация адреса DataTable

INC BP ;Модификация адреса Table

INC SI ;Модификация адреса DataTable

LOOP CopyArr2

xor bp,bp

xor cx, cx

xor ax, ax

ret

CopyArr ENDP

Start:

mov ax,Data

mov ds,ax

mov es,ax

mov ax,Stk

mov ss,ax

lea sp,StkTop

call FuncPrep

MainLoop: call ReadInput

call Simul

call OverflowCheck

call AccumulationSumm

call DisplayOutput

jmp MainLoop

;Здесь размещается код программы

;В следующей строке необходимо указать смещение стартовой точки

org RomSize-16-((InitDataEnd-InitDataStart+15) AND 0FFF0h)

ASSUME cs:NOTHING

jmp Far Ptr Start

Code ENDS

END Start