**Федеральное агентство по образованию**

**Государственное образовательное учреждение**

**высшего профессионального образования**

**«Тульский государственный университет»**

***Кафедра «Системы автоматического управления»***

**Курсовая работа**

**по дисциплине**

**ЭВМ И ВЫЧИСЛИТЕЛЬНЫЕ СИСТЕМЫ**

**на тему**

**"РАЗРАБОТКА МПС"**

Тула 2004 г.

# Введение

Одним из крупнейших достижений микроэлектроники и вычислительной техники является создание микропроцессоров (микропроцессорных наборов БИС). В решении актуальных задач ускорения научно-технического прогресса им принадлежит существенная роль.

Высокая производительность, малые габаритные размеры и энергопотребление, эргономичность, развитое, ориентированное на массового пользователя базовое и прикладное программное обеспечение микроЭВМ и ПЭВМ обеспечивают значительное повышение эффективности труда в различных отраслях народного хозяйства. Это особенно важно при автоматизации сложной управленческой, хозяйственной, инженерной детельности, научных исследований и экспериментов, учебного процесса.

Массовый выпуск микропроцессорных наборов БИС с широкими функциональными возможностями, их низкая стоимость, гибкость и точность цифровых методов обработки информации превратили МП в системные элементы, на основе которых создаются системы промышленной автоматики, связи, измерительной техники, управления транспортом и т.д.

Создание МПС на базе МП содержит два этапа: разработка аппаратной части и разработка программного обеспечения на языке низкого уровня (языка кодовых комбинаций, Ассемблера).

В данной работе рассматривается пример разработки микропроцессорной системы, которая является ПИД-регулятором.

**Выбор микропроцессора**

Исходя из данных задания (разрядность, тактовая частота, общий объём памяти) выбираем процессор Z80 фирмы Zilog.

Схема подключения микропроцессора показана на рис. 1



Рисунок 1. Схема подключения микропроцессора

По заданию (вторая часть) на входе МПС – 16‑ти битный сигнал ошибки САУ , синхронный со срезами внешнего синхросигнала постоянной частоты со скважностью  (меандр). Следовательно МПС должна начать обрабатывать новый сигнал, как только он поступил. Этот процесс организован с помощью подключения синхросигнала с выходу маскируемого прерывания .

# Организация памяти

Шестнадцатибитная линия адреса процессора Z80 при разрядности 8 бит позволяет адресовать 64 Кбайта памяти и 512 устройств ввода\вывода (256 ввода, 256 вывода).

По заданию требуется спроектировать блок памяти:

* ПЗУ – 32 Кслов (32Кбайт),
* ОЗУ – 8 Кслов (8Кбайт).

Для хранения 32 Кбайт ПЗУ используем четыре микросхемы 573РФ4 (объём 65536 бит, организация 8192×8)

Для хранения 8 Кбайт ОЗУ – четыре микросхемы 537РУ10 (объём 16384 бит, организация 2048×8).

Составим карту памяти (табл. 1), исходя из того, что ПЗУ занимает нижние адреса памяти, адреса ОЗУ следуют за ПЗУ, остальные 24 Кбайта не используются.

Таблица 1. Карта памяти

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Номер бита | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| ПЗУ | Банк 1(8 кб) | 0000 | **0** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1FFF | **0** | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| Банк 2(8 кб) | 2000 | **0** | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3FFF | **0** | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| Банк 3(8 кб) | 4000 | **0** | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 5FFF | **0** | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| Банк 4(8 кб) | 6000 | **0** | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 7FFF | **0** | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| ОЗУ | Банк 1(2 кб) | 8000 | **1** | **0** | **0** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 87FF | **1** | **0** | **0** | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| Банк 2(2 кб) | 8800 | **1** | **0** | **0** | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 8FFF | **1** | **0** | **0** | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| Банк 3(2 кб) | 9000 | **1** | **0** | **0** | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 97FF | **1** | **0** | **0** | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| Банк 4(2 кб) | 9800 | **1** | **0** | **0** | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 9FFF | **1** | **0** | **0** | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

Из табл. 1 можно сделать следующие выводы по организации памяти:

* Шина данных подключается сразу ко всем восьми микросхемам.
* Для выбора той или иной микросхемы достаточно одного двойного демультиплексора 531ИД14, осуществляющего преобразование 1⇨4. Одна его часть отвечает за выбор схемы ПЗУ, другая – за выбор ОЗУ.
* Бит А15 равен 0, если адресуется ПЗУ и равен 1, когда адресуется ОЗУ.
* ПЗУ. По линии адреса на микросхему подаются биты А0‑А12, биты А13 и А14 определяют, в какой именно банк происходит адресация.
* ОЗУ. По линии адреса на микросхему подаются биты А0‑А10, биты А11 и А12 определяют конкретный банк, биты А13 и А14 всегда равны 0.

Сигналы с выхода демультиплексора подаются непосредственно на входы **** в ПЗУ и на входы **** в ОЗУ.

Для активизации демультиплексора, отвечающего за ПЗУ, на вход **** необходимо подавать активный уровень сигнала (логический 0), если происходит цикл обращения к памяти (****) и **.** Необходимый управляющий сигнал достигается при использовании логического функции **** и элемента дисъюнктора. Справедливы аналогичные рассуждения для демультиплексора, отвечающего за ОЗУ, за исключением того факта, что в дисъюнкцию должно войти отрицание элемента ** – .**

На вход **** ОЗУ необходимо подать логическую функцию **** (логическое **,** элемент конъюнктор), а на вход ** –** выход МП **.**

Напряжения.

Т.к. ПЗУ не находится в режиме программирования, то на входы питания ПЗУ (**,,**) требуется подать +5V, а входы **** и ** –** заземлить (на рис. не показано).

На вход ОЗУ **** также подаётся +5V, а вход **** заземляется.

Схема подключения памяти к шинам представлена на рис. 2.



Рисунок 2. Организация памяти

# Организация ввода\вывода

По заданию МПС должна принимать и выдавать 16 ти битный код. Кроме того у МПС имеются перемычки для задания шестнадцатибитных констант регулятора , , .

На МПС должны быть 48 (16×3) перемычек для задания констант. Схема организации перемычек представлена на рис. слева. Перемычки собираются в группы по восемь (например, младший байт константы ) и подаются на приёмопередатчик 580ВА86. Таким образом задание трёх шестнадцатибитных констант требует шести 8‑ми битных приёмопередатчиков.

Входной и выходной сигналы целесообразно хранить в регистрах, где они будут дожидаться запроса МП на чтение при вводе и запроса внешнего устройства (фронта синхросигнала) при выводе кода. Таким образом требуется наличие четырёх 8‑ми битных регистров.

Всего же требуется адресовать 10 устройств ввода\вывода. При этом задействуются биты А0‑А3. Запишем адреса внешних устройств в табл. 3.

Таблица 2. Адреса внешних устройств

|  |  |  |
| --- | --- | --- |
| **N** | **Внешний сигнал** | **Адрес** |
| **1** | Входной сигнал | 0001:0000 |
| **2** |  | 0011:0010 |
| **3** |  | 0101:0100 |
| **4** |  | 0111:0110 |
| **5** | Выходной сигнал | 1001:1000 |

Обращение к тому или иному внешнему сигналу в зависимости от его адреса осуществим с помощью демультиплексора 555ИД7 1⇨8 и дополнительных логических элементов. На вход демультиплексора подаются биты А0‑А2. Бит А3 отвечает за то, осуществляется ли адресация к портам ввода или вывода.

Рассмотрим случай адресации к младшему байту порта ввода данных или вывода. Обозначим  – сигнал с выхода демультиплексора №0,  – сигнал, который подаётся на регистр, отвечающий за приём входного кода,  – сигнал, отвечающий за запись в регистр выходного кода. Составим таблицу истинности (табл. 2). При этом необходимо помнить, что у  активный уровень  (подключается к входу ****), а  – сигнал положительной логики (подключается к входу ****).

Таблица 3. Таблица истинности

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
| 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 |
| 1 | 1 | 1 | 0 |

Используя СКНФ, СДНФ и теорему де Моргана, запишем логические функции:

**, .**

Очевидно, что демультиплексор активируется, когда происходит цикл обращения к внешним устройствам (****) и хотя бы один из сигналов**** или  равен нулю.

**.**

Сокращённая схема подключения устройств ввода\вывода изображена на рис. 3.

Общий вид МПС достигается совмещением рис. 1–3 на одном и соединением магистралей (шины данных, адреса и управления).



Рисунок 3. Проектирование устройств ввода\вывода

# Разработка программного обеспечения

В соответствии с заданием программа разрабатывается на Ассемблере с использованием системы команд выбранного процессора.

Логически программа состоит из двух частей:

* основная часть программы для задания констант;
* программа обработки прерывания – получения входного шестнадцатибитного кода, выполнение математической операции и выдача шестнадцатибитного кода – результата математической операции:

. (1)

##

## Интеграл и дифференциал

Интеграл и дифференциал считаются по приближённым формулам:

* 
* .

В данных выражениях  – период внешнего синхросигнала. В расчётах этот множитель участвовать не будет, а результат будет выдаваться пропорционально его значению.

## Операция умножения восьмибитных чисел

В системе команд процессора Z80 нет операции умножения, поэтому умножение осуществляется с помощью операций сложения и сдвига. Операция умножения восьмибитных чисел в тексте программы выполнена в виде процедуры MUL8\_8. В данной процедуре представлен один из вариантов умножения байтов – умножение младшими разрядами вперёд со сдвигом частичной суммы вправо.

Расположение операндов:

 – множимое,  – множитель,  – двухбайтный результат, используется в качестве счётчика регистр L.

## Операция умножения шестнадцатибитных чисел со знаком

Операция умножения шестнадцатибитных чисел построена на операции умножения восьмибитных чисел и выполнена в виде процедуры MUL16\_16. Метод вычисления 32‑х битного результата представлен на рис. 4.



Рисунок 4. Схема умножения 16-битных чисел

Т.к. входной код, равно как и константы, могут быть отрицательным числом, записанным в дополнительном коде, необходимо контролировать знак произведения. При этом все отрицательные сомножители входят в произведение по модулю, а знак произведения изменяется на противоположный, если сумма минусов, стоявших перед сомножителями, равна 1 (младший бит равен 1).

Расположение операндов в памяти:

 – множимое,  – множитель, причём коэффициент всегда располагается на месте множителя.

Четырёхбитный результат помещается в память, начиная с адреса, указанного перед процедурой в парном регистре H-L.

## Контроль знаков

Поскольку коэффициенты ПИД-регулятора по заданию постоянны, то нет смысла хранить их со знаком и при каждой операции умножения заново выделять знак и инвертировать при необходимости. Целесообразно хранить модули коэффициентов в одних ячейках памяти (KP, KI, KD), а знаки – в специальной ячейке памяти (далее «знаковой ячейке»), устройство которой представлено на рис. 5. Старшая тетрада является незначащей. Младшие четыре бита отвечают за отрицательность соответсвующего элемента, причём 1 означает, что соответсвующий коэффициент изначально отрицательный. В изображённом случае все коэффициенты положительны, кроме KP. Биты трёх коэффициентов остаются постоянными с момента запуска программы.



Рисунок 5. Организация знаковой ячейки

Значение же коэффициента E () изменяется не только при поступлении нового значения. Эта ячейка отвечает также за знак разности  при приблизительном подсчёте дифференциала. Поэтому дифференциал целесообразно считать в последнюю очередь.

Проверка знака произведения осуществляется в самой процедуре умножения. Т.к. множимое представляет собой либо , либо разность , то ему однозначно соотстветствует младший бит ячейки памяти SIGN. С другой стороны необходимо знать, какой именно из коэффициентов участвует в произведении, чтобы потом определить его знак. Для этого используется другая ячейка памяти «ячейка номера» – NUMBER. В неё вносится номер: 1 соответствует КР, 2 – KI, 3 – KD. Во время процедуры значение этой ячейки полностью определяет, какой именно из коэффициентов участвует в произведении, поэтому перед выполнением процедуры умножения необходимо присвоить какое-либо значение из трёх.

## Использование памяти.

Программа для вычисления (1) требует обращения к ОЗУ. Расположение данных в ОЗУ представлено в табл. 4.

Таблица 4. Использование ОЗУ

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Адрес** | **Переменная** | **Примечание** |
| 1 | 8003: 8002:8001: 8000 | INT\_SUM | Интегральная сумма |
| 2 | 8004 | Sign | Знаковая ячейка |
| 3 | 8005 | Number | Ячейка номера |
| 4 | 8007: 8006 | PE | Предыдущее значение  (число со знаком) |
| 5 | 8009: 8008 | СE | Текущее значение  (число со знаком) |
| 6 | 800B: 800A | KP | Модуль  |
| 7 | 800D: 800C | KI | Модуль  |
| 8 | 800F: 800D | KD | Модуль  |
| WC (Work cell) – рабочие ячейки, необходимые для временного хранения результатов процедуры MUL16\_16 и отдельных частей суммы (1). Далее записывается смещение относительно ячейки WC с адресом 8010. |
| 9 | 8011: 8010 | WC+1: WC+0 | Множимое (блок 2: 1 на рис. 4): текущее значение , разность  |
| 10 | 8013: 8012 | WC+3: WC+2 | Множитель (блок 4: 3): коэффициент |
| 11 | none | WC+4 | Умерший товарищ |
| 12 | 8015: 8014 | WC+6: WC+5 | Блок 1×3 |
| 13 | 8017: 8016 | WC+8: WC+7 | Блок 2×3 |
| 14 | 8019: 8018 | WC+10: WC+9 | Блок 1×4 |
| 15 | 801B: 801A | WC+12: WC+11 | Блок 2×4 |
| 16 | 801F: 801E:801D: 801C | WC+16: WC+15:WC+14: WC+13 | Результат умножения  |
| 17 | 8023: 8022:8021: 8020 | WC+20: WC+19:WC+18: WC+17 | Результат умножения  и  |
| 18 | 9FFF | ST\_PT | Вершина стека |

## Обработка переполнения

Программно переполнение (результат (1) превышает 32 бита) контролируется в трёх случаях:

* на очередном шаге произошло переполнение интегральной суммы;
* переполнение после сложения  и интегральной суммы;
* переполнение при выполнении операции (1).

Результатом переполнения является следующая последовательность действий:

* на выходе сигнал FFFFH;
* обнуляется интегральная сумма.

При этом сам пользователь должен контролировать: чтобы разность двух соседних сигналов и  не превышала 216.

##

## Исходный текст программы

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Метка** | **Мнемокод** | **Тактов** | **Байт** | **Примечания** |
| **Задание констант** |
|  | PRT\_IN\_LO = 00H |  |  |  |
|  | PRT\_IN\_HI = 01H |  |  | Старший байт входного сигнала |
|  | KP\_LO = 02H |  |  |  |
|  | KP\_HI = 03H |  |  |  |
|  | KI\_LO = 04H |  |  |  |
|  | KI\_HI = 05H |  |  |  |
|  | KD\_LO = 06H |  |  |  |
|  | KD\_HI = 07H |  |  |  |
|  | PRT\_OUT\_HI = 08H |  |  |  |
|  | PRT\_OUT\_LO= 09H |  |  |  |
|  | ST\_PT = 9FFFH |  |  | Указатель стека |
| В соотвествии с табл. 4 также указываются значения:Int\_sum, Sign, Number, PE, CE, KP, KI, KD, WC – WC +13, WC + 17 |
| **Основная программа** |
|  | ORG 0000H |  |  | Программа начинается по адресу 0000H |
|  | JMP START | 10 | 3 |  |
|  | NOP | 4 | 1 |  |
|  | NOP | 4 | 1 |  |
|  | NOP | 4 | 1 |  |
|  | NOP | 4 | 1 |  |
|  | JMP INT | 10 | 3 |  |
| START: | DI | 4 | 1 | Запрет прерываний |
|  | **Запись значений коэффициентов (со знаками) в память** |
|  | MVI A, 0 | 7 | 2 | Обнуление ячейки SIGN |
|  | STA SIGN | 13 | 3 |
|  | IN KP\_LO | 10 | 2 | Запись в ячейку с адресом KP значения младшего байта KP |
|  | LXI H, KP | 10 | 3 |
|  | STAX H | 7 | 1 |
|  | IN KP\_HI | 10 | 2 | Запись в ячейку с адресом INT\_SUM+9 значения старшего байта KP |
|  | INX H | 5 | 1 |
|  | STAX H | 7 | 1 |
|  | IN KI\_LO | 10 | 2 | Занесение остальных значений коэффициентов в соответсвующие ячейки памяти, коэффициент KD располагается также в регистрах А: В |
|  | INX H | 5 | 1 |
|  | STAX H | 7 | 1 |
|  | IN KI\_HI | 10 | 2 |
|  | INX H | 5 | 1 |
|  | STAX H | 7 | 1 |
|  | IN KD\_LO | 10 | 2 |
|  | INX H | 5 | 1 |
|  | STAX H | 7 | 1 |
|  | IN KD\_HI | 10 | 2 |
|  | INX H | 5 | 1 |
|  | STAX H | 7 | 1 |
|  |  |  |  | **Определение знаков коэффициентов, выделения модулей и записи знаков в соответствующий бит знаковой ячейки** |
|  | MVI D, 3 | 7 | 2 | Загрузить в регистр D количество циклов |
| @@1: | LDAX H | 7 | 1 | Запись текущего коэффициента в B: C, старший байт также остаётся в аккумуляторе. |
|  | MOV C, A | 5 | 1 |
|  | DCX H | 5 | 1 |
|  | LDAX H | 7 | 1 |
|  | MOV B, A | 5 | 1 |
|  | RAL | 4 | 1 | Занесение в признак переноса старшего бита |
|  | JNC POSNUM | 17 | 3 | Если старший бит = 0, то перейти к POSNUM (positive number) – не требуется уточнение знака и изменения знаковой ячейки |
|  | MOV A, B | 5 | 1 | Загрузка в аккумулятор младшего байта |
|  | XRI FFH | 7 | 2 | Дополнение до двух и прибавление единицы – выделение модуля. |
|  | ADI 1 | 7 | 2 |
|  | MOV B, A | 5 | 1 | Сохранение младшего байта модуля в регистре В |
|  | MOV A, C | 5 | 1 | Помещение в аккумулятор старшего байта |
|  | XRI FFH | 7 | 2 | Дополнение до двух и учёт переноса. |
|  | ACI 0 | 7 | 2 |
|  | MOV C, A | 5 | 1 | Поместить в регистр С старший байт модуля |
|  | XRA A | 7 | 1 | Сброс признака переноса |
|  | MOV E, D | 5 | 1 |  |
|  | MVI A, 1 | 7 | 2 |  |
| @@2: | RAL | 4 | 1 | Смещение единицы в соответствующий разряд и сохранение в регистре Е |
|  | DCX E | 5 | 1 |
|  | JNZ @@2 | 17 | 3 |
| **Итого @@2** | **156** |  |
|  | MOV E, A | 5 | 1 |
|  | LDA SIGN | 13 | 3 | Загрузить в соответствующий бит ячейки SIGN – 1 (прямая адресация) |
|  | ADD E | 4 | 1 |
|  | STA SIGN | 13 | 3 |
| POSNUM: | MOV M, C | 7 | 1 | Сохранить модуль коэффициента в соответсвующих ячейках. |
|  | DCX H | 5 | 1 |
|  | MOV M, B | 7 | 1 |
|  | DCX H | 5 | 1 | Уменьшить адрес на единицу – указатель на старший байт следующего коэффициента |
|  | DCX D | 5 | 1 | Уменьшить счётчик цикла |
|  | JNZ @@1 | 17 | 3 | Если не обработаны все коэффициенты то перейти к @@1. По окончании в регистрах B: С содержится значение коэффициента KP. |
| **Итого@@1** | **750** |  |  |
|  | LXI SP, ST\_PT | 10 | 3 | Установка вершины стека |
|  | EI | 4 | 1 | Разрешение прерываний |
|  | HLT | 7 | 1 | Ожидание прерывания |
| **Итого(START)** | **939** | **129** |  |
| **Программа обработки прерывания** |
|  | **Выявление модуля и знака**  |
| INT: | IN PRT\_IN\_LO | 10 | 2 | Загрузить значение в регистры В: С и в память |
|  | STA СE | 13 | 3 | СE+1: СE (число со знаком) |
|  | MOV C, A | 5 | 1 |  |
|  | IN PRT\_IN\_HI | 10 | 2 |  |
|  | STA СE+1 | 13 | 3 |  |
|  | MOV B, A | 5 | 1 |  |
|  | LXI H, SIGN | 10 | 3 | Обнуление младшего бита SIGN |
|  | LDAX H | 7 | 1 |
|  | ANI 00001110B | 7 | 2 |
|  | STAX H | 7 | 1 |
|  | RAL | 4 | 1 | Проверка знака , операции аналогичны приведённым выше |
|  | JNC POSNUM2 | 17 | 3 |
|  | MOV A, C | 5 | 1 |  |
|  | XRI FFH | 7 | 2 |  |
|  | ADI 0 | 7 | 2 |  |
|  | MOV C, A | 5 | 1 |  |
|  | MOV A, B | 5 | 1 |  |
|  | XRI FFH | 7 | 2 |  |
|  | ACI 0 | 7 | 2 |  |
|  | MOV B, A | 5 | 1 |  |
|  | LDAX H | 7 | 1 | Помещение в младший бит SIGN единицы |
|  | ADI 1 | 7 | 2 |
|  | STAX H | 7 | 1 |
|  | LXI H, WC | 10 | 3 | Помещение значения в память WC+1: WC (модуль) |
| POSNUM2: | MOV A, C | 5 | 1 |  |
|  | STAX H | 7 | 1 |  |
|  | INX H | 5 | 1 |  |
|  | MOV A, B | 5 | 1 |  |
|  | STAX H | 7 | 1 |  |
| **Умножение**  |
|
|  | **Запись коэффициента KP в ячейки памяти WC+3: WC +2** |
|  | LDA KP | 13 | 3 |  |
|  | STA WC+2 | 13 | 3 |  |
|  | LDA KP+1 | 13 | 3 |  |
|  | STA WC+3 | 13 | 3 |  |
|  | MVI A, 1 | 7 | 2 | Запись в NUMBER значение 1 – идентификатор коэффициента KP |
|  | STA NUMBER | 13 | 3 |
|  | LXI H, WC + 13 | 10 | 3 | Необходимый параметр процедуры – адрес младшего байта результата |
|  | CALL MUL16\_16 | 4146 | NONE |  |
| **Умножение**  |
|
|  | MVI A, 2 | 7 | 2 | Запись в NUMBER значение 2 – идентификатор коэффициента KI |
|  | LDA KI | 13 | 3 | (58 тактов, 12 байт) – (42 такта – 12 байт) |
|  | STA WC+2 | 13 | 3 | Запись KI в ячейки множителя |
|  | LDA KI | 13 | 3 |  |
|  | STA WC+3 | 13 | 3 |  |
|  | LXI H, WC + 17 | 10 | 3 |  |
|  | CALL MUL16\_16 | 4146 | NONE |  |
| **Увеличение интегральной суммы** |
|  | LXI H, WC + 17 | 10 | 3 | Инициализация указателей |
|  | LXI D, INT\_SUM | 10 | 3 |
|  | MVI B, 4 | 7 | 2 | Инициализация счётчика |
|  | XRA A | 7 | 1 | Сброс признака переноса |
| LOOP1: | LDAX D | 7 | 1 | Загрузка первого операнда |
|  | ADC M | 7 | 1 | Сложение |
|  | STAX D | 7 | 1 | Запоминание операнда |
|  | DCR B | 5 | 1 | Декремент счётчика |
|  | JZ DONE1 | 17 | 3 | Сложение закончено? |
|  | INX H | 5 | 1 | Переход к следующему байту |
|  | INX D | 5 | 1 |
|  | JMP LOOP1 | 10 | 3 | Организация цикла |
| **Итого LOOP1** | 252 |  |  |
| DONE1: | JC **GLUCK** | 17 | 3 | При переносе перейти к обработке переполнения |
| **Умножение**  |
|
|  | LDA WC | 13 | 3 | Загрузка в регистры B: C |
|  | MOV C, A | 5 | 1 |  |
|  | LDA WC+1 | 13 | 3 |  |
|  | MOV B, A | 5 | 1 |  |
|  | LDA SIGN | 13 | 3 | Проверить знак |
|  | RAR | 4 | 1 |
|  | JNC POSNUM3 | 17 | 3 |  |
|  | MOV A, C | 5 | 1 | Если знак отрицательный, то перевести в дополнительный код. |
|  | XRI FFH | 7 | 2 |
|  | ADI 0 | 7 | 2 |
|  | MOV C, A | 5 | 1 |
|  | MOV A, B | 5 | 1 |
|  | XRI FFH | 7 | 2 |
|  | ACI 0 | 7 | 2 |
|  | MOV B, A | 5 | 1 |
| POSNUM3: | LXI H, PE | 10 | 3 | Загрузить предыдущее значение в регистры В: С |
|  | MOV A, C | 5 | 1 |  |
|  | ADD M | 7 | 1 |  |
|  | MOV C, A | 5 | 1 |  |
|  | INX H | 5 | 1 |  |
|  | MOV A, B | 5 | 1 |  |
|  | ADC M | 7 | 1 |  |
|  | MOV B, A | 5 | 1 |  |
|  | RAL | 4 | 1 | Запись в младший бит SIGN знака разности |
|  | MVI E, 11111110B | 7 | 2 |
|  | MOV A, E | 5 | 1 |
|  | ACI 0 | 7 | 2 |
|  | MOV E, A | 5 | 1 |
|  | LDA SIGN | 13 | 3 |
|  | ANA E | 4 | 1 |
|  | STA SIGN | 13 | 3 |
|  | MOV A, B | 5 | 1 | Проверить ещё раз разность на перенос |
|  | RAL | 4 | 1 |
|  | JNC POSNUM4 | 17 | 3 |  |
|  | MOV A, C | 5 | 1 | Далее процедура инвертирования знака |
|  | XRI FFH | 7 | 2 |  |
|  | ADI 0 | 7 | 2 |  |
|  | MOV C, A | 5 | 1 |  |
|  | MOV A, B | 5 | 1 |  |
|  | XRI FFH | 7 | 2 |  |
|  | ACI 0 | 7 | 2 |  |
|  | MOV B, A | 5 | 1 |  |
| POSNUM4: | STA WC | 13 | 3 | Помещение разности в ячейку множимого |
|  | MOV C, A | 5 | 1 |
|  | STA WC+1 | 13 | 3 |
|  | LDA KD | 13 | 3 | Помещение КD в ячейку множителя |
|  | STA WC+2 | 13 | 3 |
|  | LDA KD+1 | 13 | 3 |
|  | STA WC+3 | 13 | 3 |
|  | LXI H, WC+17 | 10 | 3 |  |
|  | CALL MUL16\_16 | 4146 | NONE |  |
| **Сложение**  |
|
|  | LXI H, INT\_SUM | 10 | 3 | Операция аналогичная увеличению интегральной суммы. Результат сложения в WC+16 – WC+13 |
|  | LXI D, WC + 13 | 10 | 3 |  |
|  | MVI B, 4 | 7 | 2 |  |
|  | XRA A | 4 | 1 |  |
| LOOP2: | LDAX D | 7 | 1 |  |
|  | ADC M | 7 | 1 |  |
|  | STAX D | 7 | 1 |  |
|  | DCR B | 5 | 1 |  |
|  | JZ DONE2 | 17 | 3 |  |
|  | INX H | 5 | 1 |  |
|  | INX D | 5 | 1 |  |
|  | JMP LOOP2 | 10 | 3 |  |
| **Итого LOOP2** | 252 |  |  |
| DONE2: | JC **GLUCK** | 17 | 3 |  |
| **Вычисление (1)** |
|
|  | LXI H, WC + 17 | 10 | 3 | Операция аналогичная увеличению интегральной суммы |
|  | LXI D, WC + 13 | 10 | 3 |  |
|  | MVI B, 4 | 7 | 2 |  |
|  | XRA A | 4 | 1 |  |
| LOOP3: | LDAX D | 7 | 1 |  |
|  | ADC M | 7 | 1 |  |
|  | STAX D | 7 | 1 |  |
|  | DCR B | 5 | 1 |  |
|  | JZ DONE3 | 17 | 3 |  |
|  | INX H | 5 | 1 |  |
|  | INX D | 5 | 1 |  |
|  | JMP LOOP3 | 10 | 3 |  |
| **Итого LOOP3** | 252 |  |  |
| DONE3: | JNC GOOD | 17 | 3 | Если нет переноса, то пропустить обработку переполнения |
| **Обработка переполнений** |
| **GLUCK**: | MVI A, FFH | 7 | 2 | Запись в выдаваемые старшие два байта значения FFFFH |
|  | LXI H, WC + 15 | 10 | 3 |
|  | MOV M, A | 7 | 1 |
|  | INX H | 5 | 1 |
|  | MOV M, A | 7 | 1 |
|  | LXI H, INT\_SUM | 10 | 3 | Обнуление интегральной суммы |
|  | MVI B, 4 | 7 | 2 |
|  | MVI A, 00H | 7 | 2 |
| LOOP4: | STAX D | 7 | 1 |
|  | DCR B | 5 | 1 |
|  | JZ GOOD | 17 | 3 |
|  | INX D | 5 | 1 |
|  | JMP LOOP4 | 10 | 3 |
| **Итого LOOP4** | 176 |  |  |
| **Выдача результата** |
| **GOOD:** | LDA WC + 15 | 13 | 3 | Выдача старших шестнадцати бит суммы |
|  | OUT PRT\_OUT\_LO | 10 | 2 |
|  | LDA WC + 16 | 13 | 3 |
|  | OUT PRT\_OUT\_HI | 10 | 2 |
|  | LDA CE | 13 | 3 | Текущее значение становится предыдущим. |
|  | STA PE | 13 | 3 |
|  | LDA CE+1 | 13 | 3 |
|  | STA PE+1 | 13 | 3 |
|  | **RET** | **10** | **1** |  |
| **Итого RET** | **14444** | **293** |  |
| **Процедура умножения шестнадцатибитных чисел** |
| **MUL16\_16:** |  |  |  | Расположение в памяти исходных данных (4 байта):Множимое – WC+1: WCМножитель – WC+3: WC+2Расположение результата (4 байта):H-L – адрес младшего байта результатаЗадействуются все регистры |
|  | PUSH H | 11 | 3 | Запомнить адрес младшего байта результата |
|  | LXI H, WC | 10 | 3 |  |
|  | LDAX H | 7 | 1 | Вычисление и помещение в память блока 1×3 (см. рис. 4 и табл. 4) |
|  | MOV E, A | 5 | 1 |
|  | INX H | 5 | 1 |
|  | INX H | 5 | 1 |
|  | LDAX H | 7 | 1 |
|  | MOV D, A | 5 | 1 |
|  | CALL MUL8\_8 | 820 | NONE |
|  | MOV A, С | 5 | 1 |
|  | STA WC + 5 | 13 | 3 |
|  | MOV A, B | 5 | 1 |
|  | STA WC + 6 | 13 | 3 |
|  | LDAX H | 7 | 1 | Вычисление и помещение в память блока 2×3 (см. рис. 4 и табл. 4) |
|  | MOV E, A | 5 | 1 |
|  | DCX H | 5 | 1 |
|  | LDAX H | 7 | 1 |
|  | MOV D, A | 5 | 1 |
|  | CALL MUL8\_8 | 820 | NONE |
|  | MOV A, С | 5 | 1 |
|  | STA WC + 7 | 13 | 3 |
|  | MOV A, B | 5 | 1 |
|  | STA WC + 8 | 13 | 3 |
|  | DCX H | 5 | 1 | Вычисление и помещение в память блока 1×4 (см. рис. 4 и табл. 4) |
|  | LDAX H | 7 | 1 |
|  | MOV E, A | 5 | 1 |
|  | INX H | 5 | 1 |
|  | INX H | 5 | 1 |
|  | INX H | 5 | 1 |
|  | LDAX H | 7 | 1 |
|  | MOV D, A | 5 | 1 |
|  | CALL MUL8\_8 | 820 | NONE |
|  | MOV A, С | 5 | 1 |
|  | STA WC + 9 | 13 | 3 |
|  | MOV A, B | 5 | 1 |
|  | STA WC + 10 | 13 | 3 |
|  | LDAX H | 7 | 1 | Вычисление и помещение в память блока 2×4 (см. рис. 4 и табл. 4) |
|  | MOV D, A | 5 | 1 |
|  | DCX H | 5 | 1 |
|  | DCX H | 5 | 1 |
|  | LDAX H | 7 | 1 |
|  | MOV E, A | 5 | 1 |
|  | CALL MUL8\_8 | 820 | NONE |
|  | MOV A, С | 5 | 1 |
|  | STA WC + 11 | 13 | 3 |
|  | MOV A, B | 5 | 1 |
|  | STA WC + 12 | 13 | 3 |
| **Сложение промежуточных результатов** |
|  | LXI H, WC + 6 | 10 | 3 |  |
|  | LDAX H | 7 | 1 | Загрузить ячейку WC+6 |
|  | MOV D, A | 5 | 1 |  |
|  | INX H | 5 | 1 |  |
|  | LDAX H | 7 | 1 | Загрузить ячейку WC+7 |
|  | ADD D | 4 | 1 |  |
|  | MOV D, A | 5 | 1 |  |
|  | INX H | 5 | 1 |  |
|  | LDAX H | 7 | 1 | Загрузить ячейку WC+8 |
|  | ACI 0 | 7 | 2 | Прибавить перенос к третьему байту |
|  | MOV C, A | 5 | 1 |  |
|  | LDA WC+12 | 13 | 3 |  |
|  | ACI 0 | 7 | 2 | Прибавить перенос к четвёртому байту |
|  | MOV B, A | 5 | 1 |  |
|  | INX H | 5 | 1 |  |
|  | LDAX H | 7 | 1 | Загрузить ячейку WC+9 |
|  | MOV E, A | 5 | 1 |  |
|  | MOV A, D | 5 | 1 |  |
|  | ADD E | 4 | 1 |  |
|  | MOV D, A | 5 | 1 | Регистр D содержит второй байт |
|  | MOV A, C | 5 | 1 |  |
|  | ACI 0 | 7 | 2 | Прибавить перенос к третьему байту |
|  | MOV C, A | 5 | 1 |  |
|  | MOV A, B | 5 | 1 |  |
|  | ACI 0 | 7 | 2 | Прибавить перенос к четвёртому байту |
|  | MOV B, A | 5 | 1 |  |
|  | INX H | 5 | 1 |  |
|  | LDAX H | 7 | 1 | Загрузить ячейку WC+10 |
|  | ADD C | 4 | 1 |  |
|  | MOV C, A | 5 | 1 |  |
|  | MOV A, B | 5 | 1 |  |
|  | ACI 0 | 7 | 2 | Прибавить перенос к четвёртому байту |
|  | MOV B, A | 5 | 1 |  |
|  | INX H | 5 | 1 |  |
|  | LDAX H | 7 | 1 | Загрузить ячейку WC+11 |
|  | ADD C | 4 | 1 |  |
|  | MOV C, A | 5 | 1 | Регистр С содержит третий байт |
|  | MOV A, B | 5 | 1 |  |
|  | ACI 0 | 7 | 2 | Прибавить перенос к четвёртому байту |
|  | MOV B, A | 5 | 1 | Регистр В содержит четвёртый байт |
|  | STA NUMBER | 13 | 3 | Загрузка в регистр Е содержимого ячейки памяти NUMBER, числа, которое определеняет, какой из коэффициентов является множителем |
|  | MOV E, A | 5 | 1 |
|  | STA SIGN | 13 | 3 | Загрузить ячейку знаков |
|  | RAR | 4 | 1 | Если знак первого сомножителябыл отрицательный, то регистр H содержит 1, если положительный – 0. |
|  | MVI A, 0 | 10 | 2 |
|  | ACI 0 | 7 | 2 |
|  | MOV H, A | 5 | 1 |  |
| @@3: | RAR | 4 | 1 | Помещение значащего для этого коэффициента бита в признак переноса и последующее сложение с содержимым регистра H |
|  | DCR E | 5 | 1 |
|  | JNZ @@3 | 17 | 3 |
| **Итого @@3** | **78** |  |
|  | MOV A, H | 5 | 1 |
|  | ACI 0 | 7 | 1 |
|  | RAR | 4 | 1 | Сумма минусов чётная? Если нет, то необходимо инвертировать результат |
|  | JNC NO\_INVER | 17 | 3 |
|  | STA WC + 5 | 13 | 3 | Загрузить в аккумулятор первый байт произведения |
|  | XRI FFH | 7 | 2 |  |
|  | ADI 1 | 7 | 2 |  |
|  | POP H | 11 | 1 | Вернуть значение H-L из стека |
|  | STAX H | 7 | 1 | Сохранить значение первого байта по указанному адресу |
|  | INX H | 5 | 1 |  |
|  | MOV A, D | 5 | 1 |  |
|  | XRI FFH | 7 | 2 |  |
|  | ACI 0 | 7 | 2 |  |
|  | STAX H | 7 | 1 | Сохранить значение второго байта |
|  | INX H | 5 | 1 |  |
|  | MOV A, C | 5 | 1 |  |
|  | XRI FFH | 7 | 2 |  |
|  | ACI 0 | 7 | 2 |  |
|  | STAX H | 7 | 1 | Сохранить значение третьего байта |
|  | INX H | 5 | 1 |  |
|  | MOV A, В | 5 | 1 |  |
|  | XRI FFH | 7 | 2 |  |
|  | ACI 0 | 7 | 2 |  |
|  | STAX H | 7 | 1 | Сохранить значение четвёртого байта |
| NO\_INVER: | **RET** | **10** | **1** | Конец процедуры |
| **Итого MUL16\_16** | **4135** | **168** |  |
| **MUL8\_8:** |  |  |  | Осуществляет умножение восьмибитных чиселМножитель – ЕМножимое – DДвухбайтный результат – BC Задействуется регистр: L |
|  | PUSH H | 11 | 1 | Сохранение значение регистровой пары H-L |
|  | LXI B, 0 | 10 | 3 | Сброс частичной суммы |
|  | MVI L, 8 | 7 | 2 | Загрузка счётчика |
| NEXT\_BIT: | MOV A, E | 5 | 1 | Множитель в аккумулятор |
|  | RAR | 4 | 1 | Поместить анализируемый бит в признак переноса |
|  | MOV E, A | 5 | 1 | Возврат сдвинутого множителя |
|  | JNC NO\_ADD | 17 | 3 | Бит множителя равен нулю? |
|  | MOV A, B | 5 | 1 | Нет |
|  | ADD D | 4 | 1 | Прибавление множимого |
|  | MOV B, A | 5 | 1 | Возврат старшей частичной суммы |
| NO\_ADD: | MOV A, B | 5 | 1 |  |
|  | RAR | 4 | 1 | Сдвиг частичной суммы |
|  | MOV B, A | 5 | 1 |  |
|  | MOV A, C | 5 | 1 |  |
|  | RAR | 4 | 1 |  |
|  | MOV C, A | 5 | 1 | Возврат младшей частичной суммы |
|  | DCR L | 5 | 1 |  |
|  | JNZ NEXT\_BIT | 17 | 3 | Организация цикла |
| **Итого NEXT\_BIT** | **760** |  |  |
|  | POP H | 11 | 1 | Возвращение сохранённой регистровой пары H-L |
|  | **RET** | **10** | **1** | Конец процедуры умножения |
| **Итого MUL8\_8** | **809** | **27** |  |
| **Итого вся программа** | **15419** | **627** |  |

# Вывод

В данной работе была разработана МПС на базе восьмиразрядного МП Z80. Довольно простое аппаратное решение дополняется весьма длинной программой на языке Ассемблер, которая занимает в памяти ПЗУ 627 байт. Программа выполняется примерно за 16000 (в самом худшем случае) тактов МП. Это накладывает определённые ограничения на частоту входного синхросигнала. Т.к. входной код синхронизирован по срезу, а выходной должен быть синхронизирован по фронту, то в простейшем случае (реализованный вариант) работа программы должна укладываться в половину периода синхросигнала.

При работе МП на частоте 2МГц один такт длится 500 нс. Программа выполняется за . Соответственно период входного сигнал не должен быть меньше , что накладывает ограничение на частоту входного синхросигнала .

Методы повышения частоты

1. Программный.

В задании сказано, что реализация выходного сигнала может отставать от входного, но не более 10 периодов синхросигнала. Учёт этого условия при пересмотре алгоритма программы (временного хранения поступающий входных сигналов при обработке предыдущих) позволит несколько увеличить частоту. В общем случае возможная частота синхросигнала будет зависеть от количества входных и выходных сигналов.

Примем отставание «выхода от входа» – 9,5 периодов. Тогда результат операции над первым кодом можно выдать через 9,5 периодов, второго – через 10,5 периодов и т.д. – , где  – номер периода синхросигнала, в котором необходимо выдать обработанный код, поступивший на входе в периоде .

Нет оснований предполагать, что на при проведении операций над очередным входным сигналом, количество тактов, за которое обрабатывается программа уменьшится. Из этого предположения следует, что время затраченное на обработку каждого входного кода, постоянно. Следовательно максимальное время, выраженное в количестве периодов синхросигнала, которое может работать программа – , где  – количество входных сигналов, которые планируется подать. От этого времени будет зависеть максимальная частота (примем продолжительность программы прежней).

Подставим конкретные числа:

, , , 

, , , 

Очевидно, что с ростом числа входных кодов  частота будет стемиться к 

1. Аппаратный.

Для значительного повышения максимально возможной частоты рекомендуется перенести часть расчётов из программной части в аппаратную, например, умножать шестнадцатибитные константы в умножителе двоичных чисел.

#

# Список использованной литературы

1. Каган Б.М., Сташин В.В. Основы проектирования микропроцессорных устройств автоматики. – М.: Энергоатомиздат, 1987.
2. Микропроцессорные автоматические системы регулирования. Основы теории и элементы: Учеб. пособие / В.В. Солодовников, В.Г. Коньков, В.А. Суханов, О.В. Шевяков; Под ред. В.В. Солодовникова. – М.: Высш. шк., 1991.
3. Микропроцессоры и микропроцессорные вычислительные системы: Учеб. пособие / О.В. Горячев; Тула, ТулГУ, 1998.
4. Микропроцессоры и микроЭВМ в системах автоматического управления: Справочник / С.Т. Хвоща, Н.Н. Варлинский, Е.А. Попов: Под общ. ред. С.Т. Хвоща. – Л.: Машиностроение, 1987.
5. Микропроцессоры. В 3-х кн. Кн. 1. Архитектура и проетирование микро-ЭВМ. Организация вычислительных процессов: Учеб. для втузов / П.В. Нестеров, В.Ф. Шаньгин, В.Л. Горбунов и др.; Под редакцией Л.Н. Преснухина. М.: Высш. шк., 1986.
6. Мячев А.А., Степанов В.Н. Персональные ЭВМ и микроЭВМ. Основы организации: Справочник / Под ред. А.А. Мячева. – М.: Радио и связь, 1991.
7. Пухальский Г.И., Новосельцева Т.Я. Проектирование дискретных устройств на интегральных микросхемах: Справочник. – М.: Радио и связь, 1990.
8. Скенлон Л. Персональные ЭВМ IBM PC и XT. Программирование на языке ассемблера: Пер. с англ. – М.: Радио и связь. 1989.