

Министерство образования и науки РФ
Государственное образовательное учреждение
высшего профессионального образования
Воронежская государственная лесотехническая академия

ПРОГРАММИРОВАНИЕ
МИКРОПРОЦЕССОРОВ

Методические указания
к выполнению лабораторных работ для студентов специальности
230400 – «Информационные системы и технологии» (квалификация
бакалавр); для студентов, получающих дополнительную квалификацию
«Разработчик профессионально ориентированных компьютерных технологий»

Воронеж
2011

УДК 004.4

Конарев, М.В. Программирование микропроцессоров [Текст]: методические указания к выполнению лабораторных работ для студентов специальности 230400 – «Информационные системы и технологии» (квалификация бакалавр); для студентов, получающих дополнительную квалификацию «Разработчик профессионально ориентированных компьютерных технологий»; Министерство образования и науки РФ, Гос. образовательное учреждение высш. проф. образования, Воронеж. гос. лесотехн. акад. – Воронеж, 2011. – 67 с.

Печатается по решению редакционно-издательского совета ВГЛТА

Рецензент - начальник лаборатории ФГУП НИИ Электронной техники, к.т.н., А.И. Яньков.

ВВЕДЕНИЕ

Методические указания содержат базовые сведения об архитектуре и методах программирования процессора цифровой обработки сигналов TMS320C40. Рассмотрена среда программирования процессора Texas Instruments Code Composer, система команд данного процессора, средства поддержки и отладки.

Целью методических указаний является научить студентов программировать процессор на низкоуровневом языке ассемблера, а также на высокоуровневом языке Си. Студенты должны научиться инициализировать процессор, ассемблировать, компоновать, создавать исполняемые файлы и запускать программы, программировать ветвления, циклы, макросы и подпрограммы на языке ассемблера, программировать периферийные устройства и обрабатывать прерывания.

Материал изложен от простого к сложному - от знакомства со средой программирования и составления простейших программ до программирования на высокоуровневом языке Си. Весь материал излагается на примерах, в конце каждой лабораторной работы имеются самостоятельные задания для закрепления материала и контрольные вопросы.

Для успешного выполнения лабораторных работ требуются базовые знания языка программирования Си и операционной системы Windows.

ЛАБОРАТОРНАЯ РАБОТА № 1

Тема: «Знакомство со средой программирования ПЦОС TMS320C40 TI Code Composer»

Цель работы: Познакомиться со средой программирования процессоров TMS320 C30/C40. Изучить интерфейс, настройки, общие принципы и методы работы с Code Composer. Научиться загружать и выполнять программы.

Установка параметров конфигурации системы

Перед тем как начать работу со средой Code Composer необходимо сконфигурировать систему, то есть задать модель процессора, их количество, режим работы и т.д. Для конфигурации системы запустите программу Setup Code Composer. В появившемся окне (Рис. 1.1) выберете C4x Simulator и добавьте его в систему (кнопка Add to system configuration).

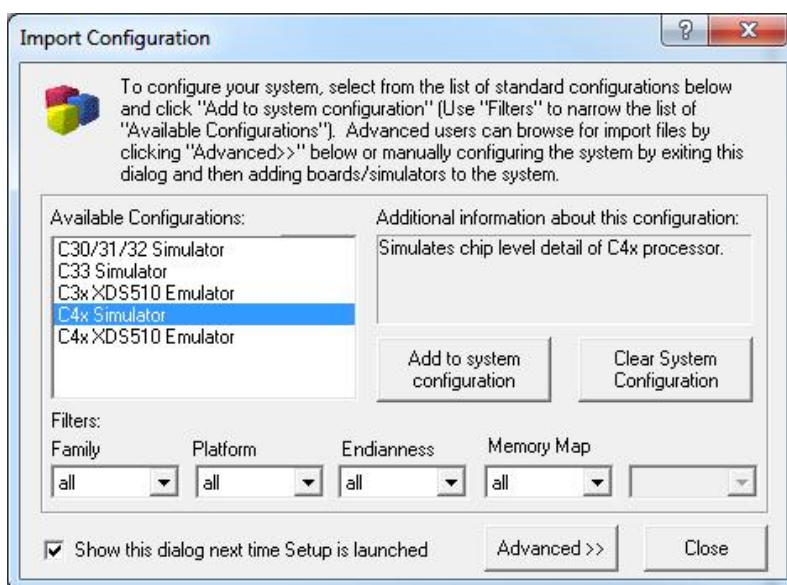


Рисунок 1.1 – Окно Import Configuration

Сохраните параметры конфигурации системы (меню File >> Save) и закройте Setup Code Composer. Запустите программу Code Composer 'C30 - 'C40. Главное окно программы представлено на рисунке 1.2.

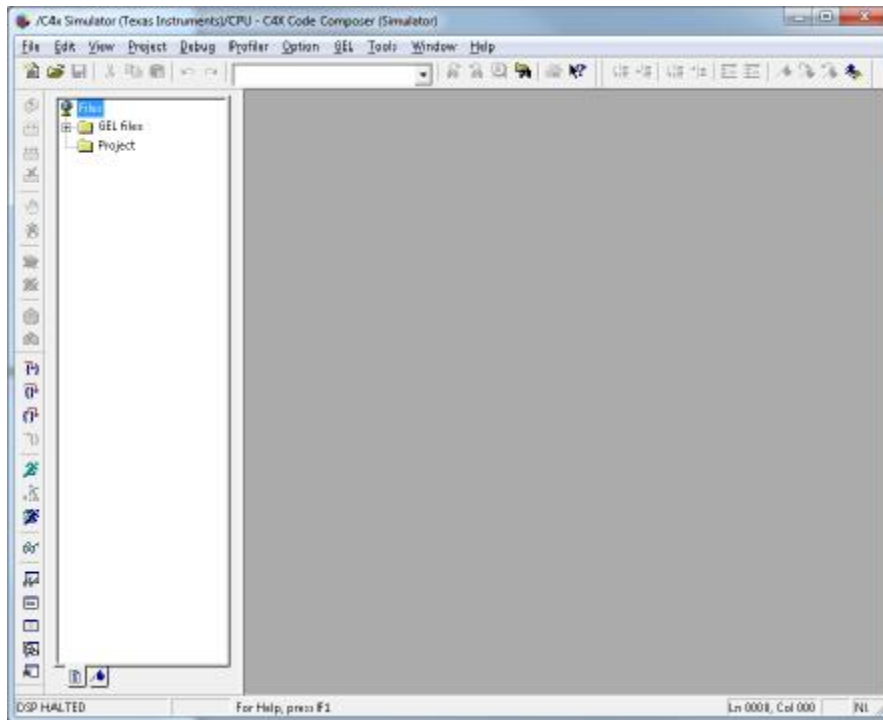



Рисунок 1.2 – Главное окно Code Composer

Оно состоит из главного меню, панели инструментов, окна структуры проекта и редактора кода. Для вызова справки выберите меню Help. При первом запуске программы проект не загружен, а окно редактора кода пусто.

Упражнение 1. Использование окна Memory (Память)

Для просмотра и редактирования содержимого памяти ПЦОС воспользуйтесь окном Memory. Для этого выберите View >> Memory или воспользуйтесь кнопкой  на панели инструментов. При этом появится диалоговое окно (Рис. 1.3) для выбора опций просмотра памяти: стартового адреса, формата представления данных и др.

Занесите в ячейки с адресами 0x2FF800 - 0x2FF80F значения 0x0 – 0xF. Для редактирования содержимого ячейки памяти дважды кликните на ней.

Посмотрите содержимое этих же ячеек после редактирования в формате 32 bit binary. Для изменения формата представления данных в открытом окне Memory воспользуйтесь пунктом Properties контекстного меню.

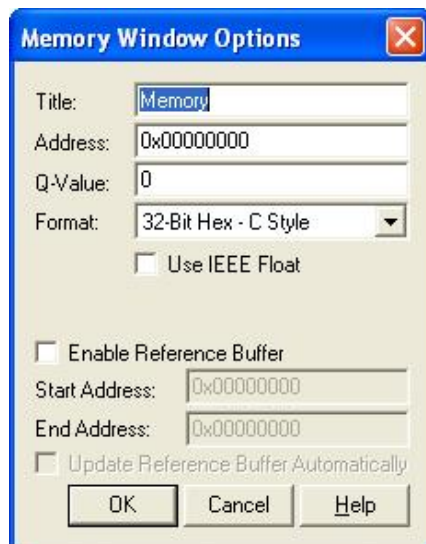



Рисунок 1.3– Опции памяти

Упражнение 2. Использование окна CPU Registers (Регистры)

Для просмотра и редактирования содержимого регистров ПЦОС выберите View >> CPU Registers >> CPU Register или воспользуйтесь кнопкой . Появится окно, представленное на рисунке 1.4. В этом окне вы можете видеть и редактировать содержимое регистров процессора.

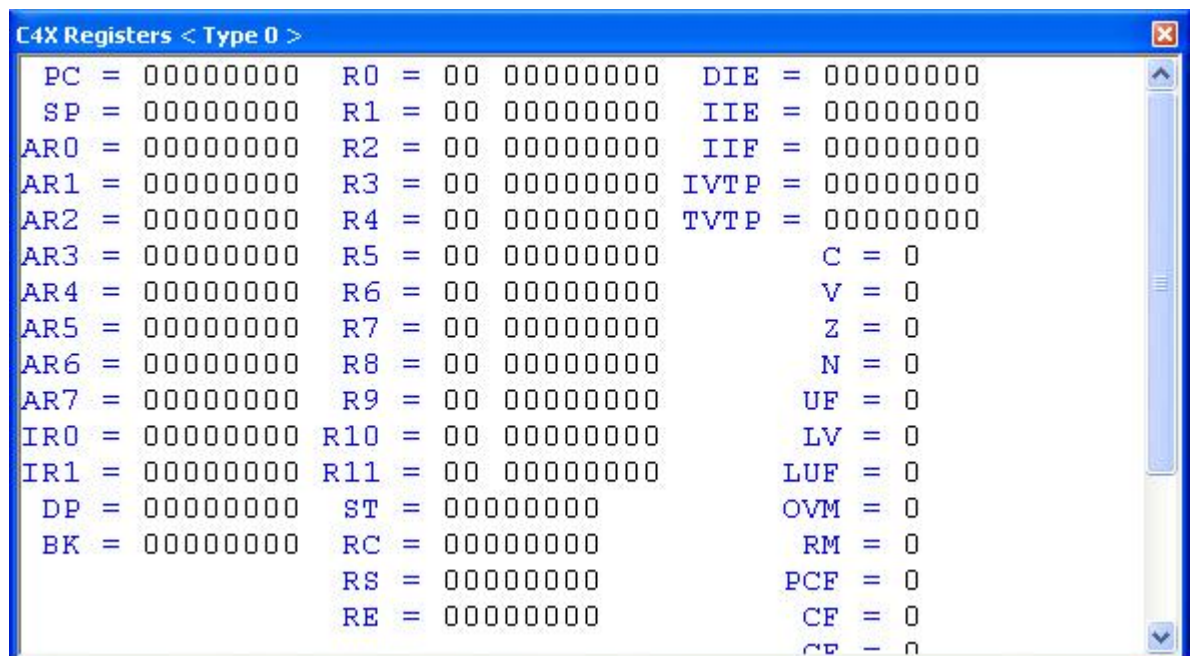


Рисунок 1.4– Окно регистров процессора

Занесите в регистры общего назначения AR0-AR7 значения 0xA0 – 0xA7. Для этого необходимо дважды кликнуть на содержимом регистра и ввести нужное значение в поле Value.

Упражнение 3. Загрузка и выполнение программы

Для загрузки программы выберите File >> Load Program.. Загрузите файл main.out, при этом автоматически откроется окно дизассемблера (Рис. 1.5).

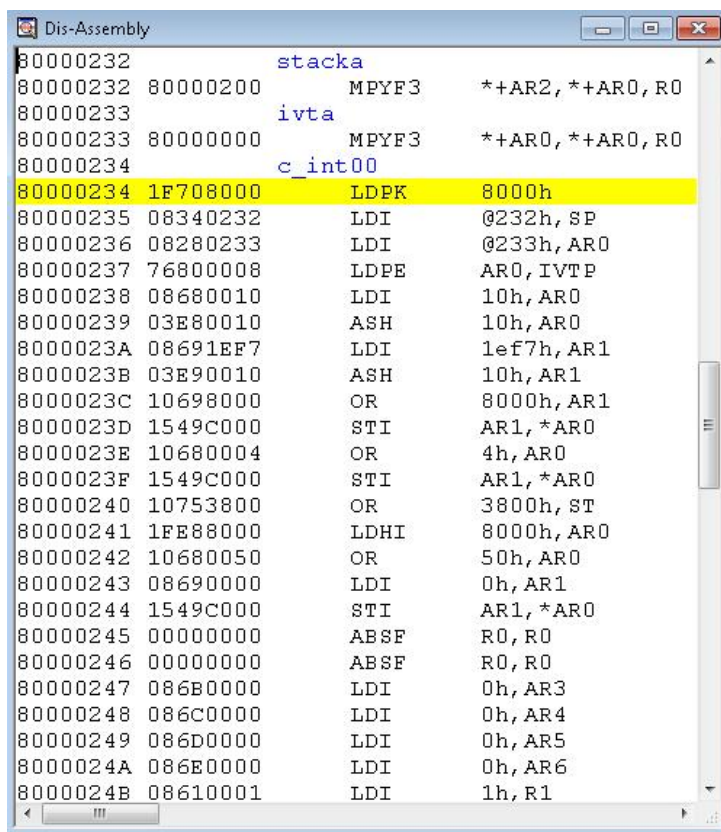







Рисунок 1.5 – Окно дизассемблера

В этом окне показаны: адрес инструкции, код инструкции, символьное представление инструкции для отладки программы. Окно дизассемблера можно вызвать кнопкой  панели инструментов.

Выполните несколько инструкций загруженной программы, для этого нажмите клавишу F8 или кнопкой Step Into  панели инструментов.

Установите курсор ниже на строке с адресом большим, чем текущее значение счетчика команд (PC), затем воспользуйтесь функцией Run to Cursor . Процессор выполнит команды до строчки, где установлен курсор.

Кнопка Run , а также соответствующая функция меню Debug, позволяет запустить программу на выполнения с текущего значения счетчика команд.

Кнопка Halt , а также соответствующая функция меню Debug, останавливает выполнение запущенной программы.

Запустите программу main.out на выполнение с помощью функции Run, а через некоторое время остановите выполнение программы с помощью Halt.

Упражнение 4. Инициализация процессора

Меню Debug также содержит такие полезные функции как Reset DSP и Restart.

Reset DSP – инициализирует содержимое регистров процессора в состояние, предусмотренное при включении питания. Выполнение программы при этом прекращается.

Restart – восстанавливает счетчик команд адресом точки входа в программу. При этом выполнение программы *не начинается*.

Выполните по очереди команды Reset DSP и Restart.

Упражнение 5. Контрольные точки (Breakpoints)

Контрольные точки останавливают выполнение программы. Когда программа остановлена, вы можете просмотреть состояние процессора: проверить и модифицировать переменные, просмотреть содержимое стека и т.д.

Установите в запущенной программе контрольную точку и выполните программу до неё. Для этого выберите в меню функцию Debug >> Breakpoints... В появившемся диалоговом окне (Рис. 1.6) введите адрес точки останова и нажмите Add. Формат адреса должен соответствовать корректному выражению на языке C, например, 0x2FF828 для абсолютного адреса. Можно также указать имя функции или символа. Для исходного

файла на C имеется возможность указать файл и номер строки для контрольной точки.

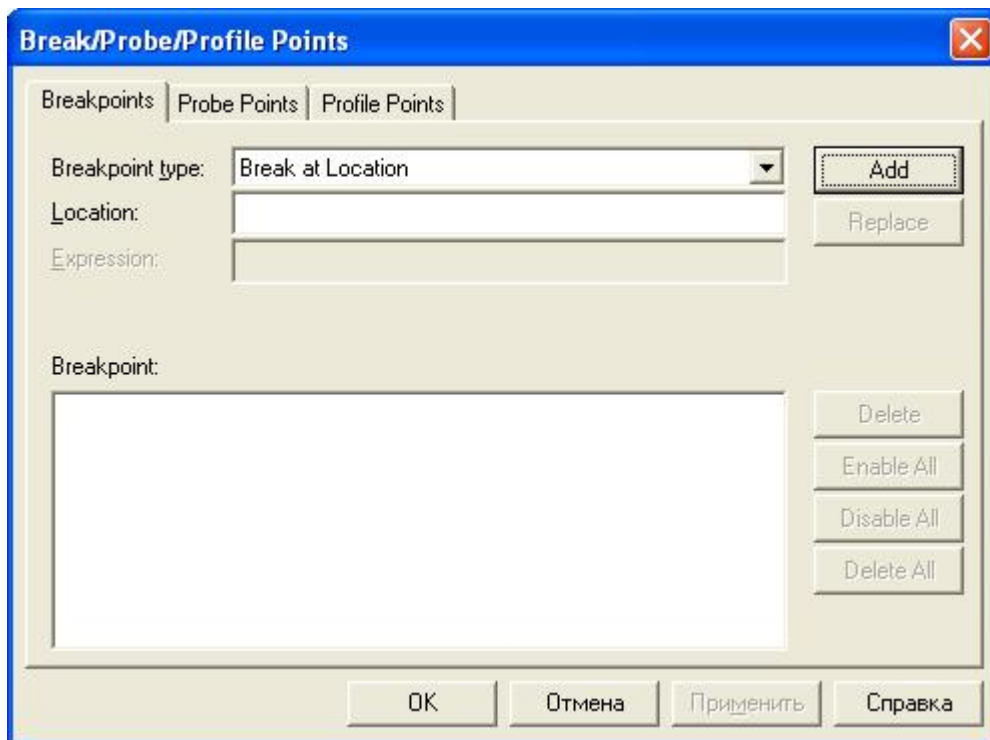




Рисунок 1.6 – Добавление контрольной точки

Убрать и добавить контрольные точки можно с помощью кнопки  на панели инструментов. Кнопка  позволяет убрать все контрольные точки. Диалоговое окно на рисунке 1.6 позволяет разрешить или запретить установленные контрольные точки. Вы также можете установить условные контрольные точки, указав логическое выражение в поле Expression. В этом случае процессор остановится на контрольной точке, только в случае истинности условия.

Упражнение 6. Заполнение области памяти

Чтобы заполнить область памяти определённым значением воспользуйтесь функцией Filling Memory.

1. Выберите Edit >> Memory >> Fill из главного меню программы.
2. В появившемся диалоговом окне (Рис. 1.7) введите стартовый адрес, длину и значение.

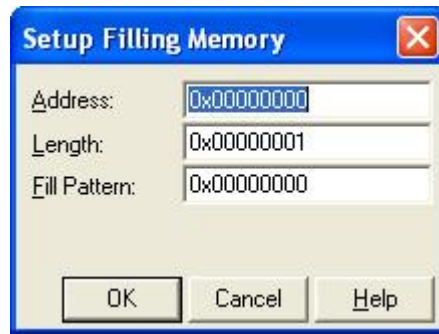


Рисунок 1.7 – Заполнение памяти


При этом в ячейки со стартового адреса, по (стартовый адрес + длина – 1) заполняются значением из Fill Pattern.

Заполните ячейки 0x300000 - 0x300FFF значением 0x55555555, 0x301000 - 0x301FFF значением 0xAAAAAAAA.

Просмотрите содержимое памяти с помощью функции View >> Memory.

Упражнение 7. Использование окна Watch Window

Окно Watch позволяет просматривать и редактировать переменные и выражения на языке C, данные при этом можно представить в различных форматах.

Для того чтобы открыть окно Watch (Рис. 1.8) выберете в меню View >> Watch window или воспользуйтесь кнопкой  на панели инструментов.

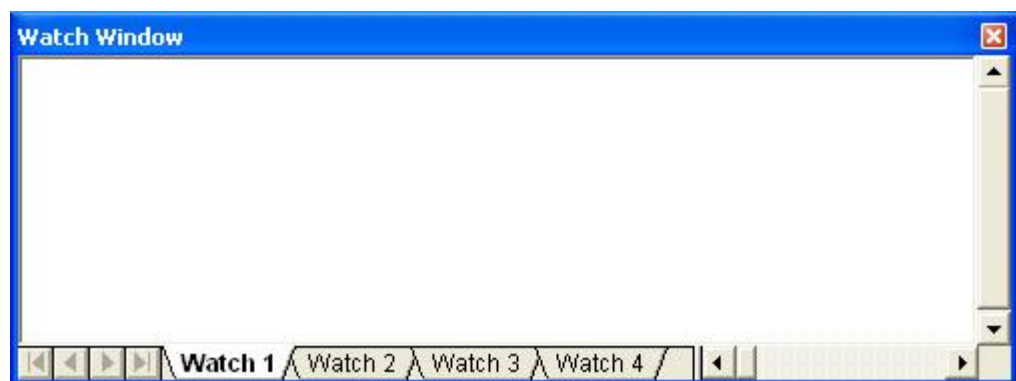



Рисунок 1.8 – Окно Watch

Для добавления переменных в контекстном меню выберете Insert New

Expression. В появившемся окне  введите выражение.

Добавить переменные в окно Watch можно из окна дизассемблера, выбрав функцию Add to Watch Window из контекстного меню.

Формат представления данных в окне Watch по умолчанию десятичный, изменить формат можно при помощи специальных символов. Например, MyVar1, x определяет шестнадцатеричный формат представления для MyVar1. Список символов указан в таблице 1.1.

Таблица 1.1
Форматы представления данных

Символ	Формат представления данных
d	Десятичный
e	Вещественный в экспоненциальной форме
f	Вещественный, десятичный
x	Шестнадцатеричный
o	Восьмеричный
u	Целый, без знака
c	ASCII символ

Добавьте в окно Watch символ c_int00 в шестнадцатеричном формате.

Убедитесь, что в окне появился верный адрес символа.

Добавьте в окно Watch регистр R1. Убедитесь, что изменение значения в регистре отображается и в окне Watch.

Контрольные вопросы

1. Каким образом осуществляется конфигурирование системы перед первым запуском Code Composer?
2. Для чего служит окно Memory, как его вызвать?
3. Как отредактировать содержимое ячейки памяти ПЦОС?
4. Каким образом можно посмотреть и отредактировать содержимое регистров ПЦОС?
5. Как осуществить загрузку программы в процессор?
6. Какая информация представлена в окне дизассемблера?
7. Для чего служат функции Run и Halt?
8. В чём заключается отличие функции Restart от Reset DSP?
9. Для чего служат контрольные точки в программе?
10. Как заполнить область памяти определённым значением?
11. Для чего служит Watch Window?

ЛАБОРАТОРНАЯ РАБОТА № 2

Тема: «Создание и выполнение программы, инициализирующей ПЦОС TMS320C40»

Цель работы: написать и выполнить программу инициализации процессора. Для этого изучить редактор кода Code Composer, научиться ассемблировать программу, компоновать и запускать программу. Изучить основные принципы создания командных файлов компоновщика (Linker).

Для того чтобы создать и выполнить программу необходимо:

- Написать исходный код.
- Получить с помощью ассемблера файлы .obj в формате COFF.

Ассемблер – служит для перевода текстового файла с исходным кодом в объектный файл с инструкциями процессора.

- Получить исполняемый файл .out в формате COFF с помощью компоновщика. **Компоновщик** (Linker) объединяет все необходимые программные объектные файлы в один исполнимый объектный файл .out, при этом размещает секции в памяти и определяет все внешние ссылки.

- Полученный .out файл можно выполнить в Code Composer в режиме симулятора или эмулятора.

Формат файлов COFF упрощает программирование тем, что позволяет оперировать блоками кода или данных. Эти блоки называют секциями. Ассемблер и компоновщик имеют в своем составе директивы, которые позволяют создавать и манипулировать секциями.

Секция – это базовый модуль объектного файла, который содержит инструкции или данные, и при этом занимает обязательно смежное пространство в памяти ПЦОС.

COFF файл всегда содержит три предопределённые секции:

- .text** – обычно содержит исполняемый код программы;
- .data** – обычно содержит инициализированные данные;
- .bss** – обычно содержит неинициализированные данные.

Кроме этих секций в программе могут быть и другие секции, определённые с помощью специальных директив `.sect` или `.usect`. Каждую секцию можно разместить в памяти независимо друг от друга.

Создание инициализированной секции начинается с директивы `.sect`

`.sect "Имя секции"`

Создание неинициализированной секции начинается с директивы `.usect`

Символ `.usect "Имя секции"`, *Размер секции* [необязательный флаг для выравнивания]

Символ - адрес первого зарезервированного в данной секции слова. Фактически определяет имя переменной, для которой резервировалось место. На этот символ можно ссылаться в программе.

Размер секции – количество 32 битных слов, зарезервированных для данной секции.

Упражнение 1 Создание программы, инициализирующей процессор

Далее создадим программу инициализации ПЦОС. При поступлении сигнала reset TMS320C40 переходит на адрес, сохраненный в reset векторе, и начинает выполнение программы с этого места. Reset вектор обычно содержит адрес подпрограммы инициализации системы. Такая подпрограмма обычно выполняет следующие задачи:

- инициализирует регистр DP (Data Pointer);
- инициализирует указатель стека;
- устанавливает указатель на таблицу программных и аппаратных прерываний;
- инициализирует регистры управления памятью;
- очищает и разрешает кэш память.

В примере ниже инициализирующая программа переводит C40 в следующее состояние:

- Кэш разрешен;
- DP указывает на `.text` секцию;

- Указатель стека указывает на начало секции *mystack*;
- Регистры управления памятью инициализированы значением 0x1EF78000.

; Пример инициализации TMS320C40

; Создание таблицы векторов прерываний

;

_myvect .sect "myvect" ; Создаём секцию для векторов прерываний

reset .word _c_int00 ; RESET вектор

.space 1 ; Резервируем место для NMI прерывания

;

; Создаём таблицу векторов программных прерываний

;

_mytrap .sect "mytrap" ; Секция для векторов программных прерываний

;

; Создаём стек

;

_mystack .usect "mystack", 100 ; резервируем 100 слова под стек

.text

stacka .word _mystack ; адрес mystack секции

ivta .word _myvect ; адрес myvect секции

tvta .word _mytrap ; адрес mytrap секции

ieval .word 1 ; значения регистра разрешения прерываний

gctrl .word 1EF78000h ; значения регистра управления памятью

lctrl .word 1EF78000h ; значения регистра управления памятью

mctrla .word 100000h ; адрес регистра управления глобальной памятью

_c_int00: ; точка входа с программы, при reset начинаем отсюда

;

; Инициализация регистра DP

;

ldp stacka

;

; Устанавливаем указатель на таблицу прерываний

;

ldi @ivta, AR0

ldpe AR0, IVTP

;

; Устанавливаем указатель на таблицу программных прерываний

;

```

        ldi @tvta, AR0
        ldpe AR0, TVTP
;
; Инициализируем регистр управления глобальной памятью
;
        ldi @mctrla, AR0
        ldi @gctrl, R0
        sti R0, *AR0
;
; Инициализируем регистр управления локальной памятью
;
        ldi @lctrl, R0
        sti R0, *+AR0(4)
;
; Инициализируем указатель стека
;
        ldi @stacka, SP
;
; Инициализируем регистр разрешения прерываний
; Запись 1 в ПЕ разрешает прерывание таймера
;
        ldi @ieval, ПЕ
;
; Глобально разрешаем прерывания, очищаем и разрешаем кэш
;
        or 3800h, ST

        BR begin ; переход к началу приложения
        nop
        nop
        nop
begin ; начало программы

        ;<< Вставьте сюда код вашего приложения >>

stop    br stop; конец программы
        nop
        nop
        nop
.end

```

Обратите внимания на следующие моменты.

В программе создано 4 секции, из них 3 инициализированные (myvest, mytrap и .text) и неинициализированная секция mystack.

Комментарий в ассемблерной программе начинается с символа ; или *, но только, если * - первый символ в строке.

Каждая строка кода может начинаться с метки или комментария. Символьному представлению инструкции должен предшествовать пробел или табуляция.

Символ _c_int00 является стандартным для определения точки входа в программу. Перед этим символом в начале секции .text определены данные. С _c_int00 и до метки begin находится программный код инициализации TMS320C40. Начиная с метки begin, можно вставить код вашего приложения.

В программе Code Composer выберите File >> New Source File. Сохраните файл под именем init.asm. Далее введите в редакторе кода исходный код, приведённый выше. После этого откомпилируйте программу, для этого выберите функцию Project >> Compile File из главного меню программы или используйте кнопку Compile active source file на панели инструментов. Если возникли сообщения об ошибках при компиляции, то исправьте их.

В результате компилирования программы должен быть создан файл init.obj – это неисполняемый COFF файл.

Упражнение 2 Компоновка программы и создание исполняемого файла

Для того чтобы выполнить программу на ПЦОС необходимо создать исполняемый COFF файл с помощью компоновщика (Linker). Компоновщик использует секции в неисполняемом файле как строительные блоки, он объединяет секции из различных файлов в один, а также определяет адрес в памяти для размещения каждой секции. Для выполнения этих действий используются две директивы: MEMORY и SECTIONS.

Директива **MEMORY** позволяет определить карту памяти целевой системы. Вы можете именовать часть адресного пространства, указать начальный адрес и длину блоков памяти.

Директива **SECTIONS** указывает компоновщику, как объединять входящие секции из различных неисполняемых COFF файлов, и как их размещать в памяти в выходном исполняемом файле.

Запустить компоновщик можно следующим образом:

```
lnk30 LinkerComandFile.cmd
```

Где LinkerComandFile.cmd – это командный файл для компоновщика.

Создадим командный файл для компоновки программы инициализации ПЦОС.

```
Init.obj
```

```
-o Init.out
```

```
MEMORY
```

```
{  ROM:          origin = 0h          length = 01000h
   LOCAL:        origin = 0300000h    length = 0700000h
   MY_VAR        origin = 080000000h   length = 0100h
   GLOBAL:       origin = 080000200h   length = 050000000h
   BLK0:         origin = 02FF800h     length = 0400h
   BLK1:         origin = 02FFC00h     length = 0400h }
```

```
SECTIONS
```

```
{  myvect: > ROM
   .data >LOCAL
   .text: > LOCAL
   mytrap: > BLK1
   mystack: > BLK1 }
```

Init.obj – это имя входного файла. Init.out – имя выходного файла, для его задания используется директива –o.

Далее в файле в блоке **MEMORY** расписана карта памяти целевой системы. Каждый фрагмент начинается с имени, далее указывается

стартовый адрес (после ключевого слова `origin`) и его длина (после ключевого слова `length`).

В блоке `SECTIONS` указаны правила размещения секций. Так секция `myvsect` будет размещена, начиная с нулевого адреса. Секция `.text` – с адреса `0x300000`. Секции `mytrap` и `mystack` во внутренней памяти TMS320C40, одна непосредственно за другой.

Создайте командный файл компоновщика и сохраните (например, под именем `link.cmd`) его в папке с исходным кодом и COFF файлом программы инициализации процессора.

Создайте `bat` файл (например, `makeout.bat`) со следующим содержанием:

```
lnk30 Link.cmd
```

```
PAUSE
```

Для создания `bat` файла кликните правой кнопкой мыши в проводнике Windows и выберите Создать >> Текстовый документ. Измените имя файла с “Текстовый документ.txt” на “`makeout.bat`”.

Запустите `makeout.bat` на выполнение, при этом создастся исполняемый файл `Init.out`, а на экране будет выведено следующее сообщение (Рис. 2.1).

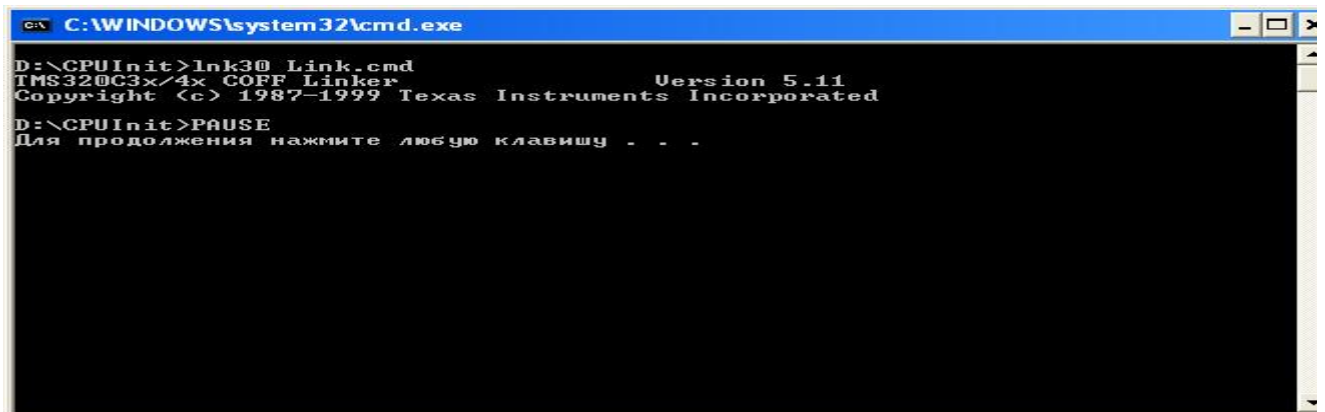


Рисунок 2.1 – Компоновка программы

Полученный исполняемый файл запустите на выполнение в Code Composer.

Обратите внимание, что по нулевому адресу лежит значение `0x300007` – это адрес символа `_c_int00`, который является точкой входа в программу.

Выполните Debug >> Reset DSP, при этом будет выполнен переход к точке входа в программу (Рис. 2.2).

```

Dis-Assembly
00300005 1EF78000      LDA      8000h, IIE
00300006          mctrla
00300006 00100000      ABSF      R0, DP
00300007          c_int00
00300007 50700030      LDIU      30h, DP
00300008 08280001      LDI        @1h, AR0
00300009 76800008      LDPE      AR0, IVTP
0030000A 08280002      LDI        @2h, AR0
0030000B 76810008      LDPE      AR0, TVTP
0030000C 08280006      LDI        @6h, AR0
0030000D 08200004      LDI        @4h, R0
0030000E 1540C000      STI        R0, *AR0
0030000F 08200005      LDI        @5h, R0
00300010 15400004      STI        R0, *+AR0 (4)
00300011 08340000      LDI        myvect, SP
00300012 08370003      LDI        @3h, IIE
00300013 10753800      OR        3800h, ST
00300014 60000003      BR        stop
00300015 0C800000      NOP
00300016 0C800000      NOP
  
```

Рисунок 2.2 - Программа инициализации процессора

Выполните программу по шагам (клавиша F8).

Контрольные вопросы

1. Какого основное назначение компоновщика?
2. Что такое секция?
3. Какие predetermined секции содержит COFF файл, что содержат эти секции?
4. С помощью каких директив можно определить секции?
5. Какие действия обычно выполняет программа инициализации процессора?
7. Как в ассемблерной программе оформляются комментарии?
8. Может ли строка кода на ассемблере начинаться с инструкции?
9. Какой символ является стандартным для точки входа в программу TMS320C40?

10. Какие расширения у исполняемого и неисполняемого COFF файла?
11. Какое назначение директив MEMORY и SECTIONS?

ЛАБОРАТОРНАЯ РАБОТА № 3

Тема: «Программы с использованием арифметических и логических инструкций, инструкций загрузки и сохранения»

Цель работы: изучить основные режимы адресации ПЦОС TMS320C40: регистровый, прямой, косвенный, непосредственный. Научиться использовать инструкции загрузки и сохранения, а также логические и арифметические инструкции. Научиться писать и выполнять программы с использованием изученных классов инструкций.

Основные режимы адресации ПЦОС TMS320C40

Регистровая адресация. В этом режиме операндом является регистр процессора. Для операций с вещественными числами следует использовать регистры R0-R11, для целочисленных операций – любой другой. Список основных регистров приведён в таблице 3.1.

Таблица 3.1
Основные регистры ПЦОС TMS320C40

Синтаксис ассемблера	Назначение регистров
R0 – R11	Регистры повышенной точности 0 - 11
AR0 – AR7	Вспомогательные регистры 0 - 7
DP	Указатель страницы данных
IR0-IR1	Индексные регистры
BK	Регистр размера блока
SP	Указатель системного стека
ST	Регистр состояния процессора
DIE	Регистр разрешения прерывания ПДП
IE	Регистр разрешения внутреннего прерывания
IF	ПОВ выводы и регистр флага прерывания
RS	Регистр адреса начала повторения
RE	Регистр адреса конца повторения
RC	Счетчик повторений
IVTP	Указатель векторной таблицы системных прерываний
TVTP	Указатель векторной таблицы программных прерываний

Прямой режим адресации. В этом режиме адресации операнд есть содержимое 32 разрядного адреса, синтаксически определяемого как @address. Старшие 16 разрядом определяются содержимым регистра DP (Data Pointer), а младшие словом инструкции.

Например: DP = 30h, тогда обращение @1h означает обращение по адресу 30 0001h.

В карте памяти TMS320C40 пространство 2F F800h – 2F FFFFh отведено внутренней ОЗУ, а 30 0000h – 7FFF FFFFh и 8000 0000h – FFFF FFFFh отведены под внешнюю память (local bus и global bus соответственно).

Косвенный режим адресации. Адрес операнда определяется содержимым вспомогательного регистра (AR0-AR7).

Например, при AR0=80001FFFh, *AR0 определяет содержимое по адресу 80001FFFh.

В таблице 3.2 приведены некоторые формы косвенной адресации.

Таблица 3.2
Некоторые режимы косвенной адресации

Синтаксис	Операция	Описание
*ARn	addr=ARn	Без смещения
*+ARn(displacement)	addr=ARn+displacement	С добавлением предварительного смещения
*-ARn(displacement)	addr=ARn-displacement	С предварительным вычитанием смещения
*++ARn(displacement)	addr=ARn+displacement ARn=ARn+displacement	С предварительным добавлением смещения и изменением
*--ARn(displacement)	addr=ARn-displacement ARn=ARn-displacement	С предварительным вычитанием смещения и изменением
*ARn++(displacement)	addr=ARn ARn=ARn+displacement	С последующим добавлением смещения и изменением
*ARn--(displacement)	addr=ARn ARn=ARn-displacement	С последующим вычитанием смещения и изменением

Например, пусть AR1=300001h, обращение *AR1++(1) определяет обращение по адресу 300001h. После обращения по этому адресу значение AR1 увеличится на (1) и станет AR1=300002h. Величину инкремента, равную

единице, можно не указывать, то есть записи $*AR1++(1)$ и $*AR1++$ эквивалентны.

Непосредственный режим адресации. В этом режиме операнд задаётся непосредственно *шестнадцатиразрядным* числом. Операнд может быть целым (со знаком или без) или вещественным в зависимости от типа инструкции.

Система команд ПЦОС TMS320C40

Система команд содержит 113 команд, организованных в следующие функциональные группы:

- Команды загрузки и сохранения;
- Двухоперандные арифметико-логические команды;
- Трёхоперандные арифметико-логические команды;
- Команды программного управления;
- Команды управления блокировкой;
- Команды параллельных операций.

В таблице 3.3 приведён список основных команд загрузки и сохранения. Эти инструкции могут:

- Загружать слово из памяти в регистр;
- Сохранять слово из регистра в память;
- Управлять данными в системном стеке;
- Передавать данные между основными и расширенными регистрами.

Таблица 3.3
Некоторые инструкции загрузки и сохранения данных

Команда	Описание
LDI src, dst	src -> dst Загружает целое
ldi 5, AR0 ;теперь AR0=5 ldi *AR0, AR7; загружаем из адреса 0005 (AR0=5) данные в AR7	
LDHI src, dst	src -> 16 MSBs of dst Непосредственная загрузка 16 разрядов без знака в 16 старших разрядов
ldhi 8000h, AR0 ; теперь AR0 = 80000000h ldhi 30h, AR1 ; теперь AR1 = 00300000h	

STI src, dst	src -> dst Сохраняет целое
ldhi 30h, AR1 ; теперь AR1 = 00300000h ldi 0FFFFh, AR7 sti AR7, *AR1++ ; сохраняем содержимое AR7 по адресу 300000h, после этого AR1=300001 инкрементируется	
LDE src, dst	src(exp) -> dst(exp) Загружает экспоненту числа с плавающей запятой (ПЗ)
LDM src, dst	src (man) ->dst (man) Загружает мантиссу с ПЗ
LDF src, dst	src -> dst Загружает значение с ПЗ
STF src, dst	src -> dst Сохраняет значение с ПЗ
LDPK src	src -> DP Непосредственная загрузка DP регистра
ldpk 30h ;DP=30h ldi @5, AR0 ;записываем данные из 300005h в AR0 ldpk 8000h ;DP=8000h ldi @5, AR0 ;записываем данные из 80000005h в AR0	
POP dst	*SP— -> dst Выталкивает целое из стека
PUSH src	src -> *++SP Загружает целое в стек
ldi 1, AR1; AR1=1 ldi 2, AR2; AR2=2 Push AR1 Push AR2 Pop AR3; AR3=2 Pop AR4; AR4=1	

В таблице 3.4 приведены основные арифметические и логические инструкции, а также примеры их использования.

Таблица 3.4
Некоторые арифметические и логические инструкции

Команда	Описание
ABSF src, dst	src -> dst Абсолютное значение числа с плавающей запятой (ПЗ)
ABSI src, dst	src -> dst Абсолютное значение целого
ldi -1, AR0 ; AR0= -1 (0FFFFFFFh) absi AR0, AR0 ; AR0 = 1	

ADDC src, dst	dst + src + C -> dst Сложить целое с переносом
ADDF src, dst	dst + src -> dst Сложить значение с ПЗ
ADDI src, dst	dst + src -> dst Сложить целые
ldi 9, AR7 addi 3, AR7; AR7=3+9=0Ch	
AND src, dst	dst AND src -> dst Поразрядное логическое И
ldi 5555h, AR0 ; AR0 = 101010101010101b and 0FFh, AR0 ; AR0 = (101010101010101)and(11111111) = 01010101 = 55h	
FIX src, dst	fix(src) -> dst Преобразовать число с ПЗ в целое
FLOAT src, dst	float (src) -> dst Преобразовать целое в число с ПЗ
SUBF src, dst	dst - src -> dst Вычитание значений с ПЗ
SUBI src, dst	dst - src -> dst Вычесть целое
ldi 1000h, AR7 subi 1, AR7 ; AR7=FFF	
CMPI src, dst	dst - src Результат вычитания никуда не записывается, но влияет на флаги регистра состояния ST. Команда применяется для сравнения src с dst
MPYF src, dst	dst * src -> dst Умножить значения с ПЗ
MPYI src, dst	dst * src -> dst Умножить целые
ldi 2, AR0 mpyi 3, AR0 ; AR0=6	
NEGI src, dst	0 – src -> dst Отрицание целого
NOT src, dst	~src -> dst Поразрядное логическое дополнение
ldi 5555h, AR7 ; AR7=0000000000000000 0101010101010101 not AR7, AR7; AR7 = 1111111111111111 1010101010101010	
OR src, dst	dst OR src -> dst Поразрядное логическое ИЛИ
ldhi 8000h, AR0 ; AR0=80000000h or 1000h, AR0 ; AR0=80001000h	

RCPF src, dst	16-разрядная обратная величина от src → dst Обратная величина числа с ПЗ
XOR src, dst	dst XOR src -> dst Поразрядное исключающее ИЛИ

Упражнение 1

Создадим программу, вычисляющую значение следующего выражения

$$F(x,y) = ax^2 + bxy + cy^2.$$

Программу будем создавать на основе приложения инициализации процессора, рассмотренного в лабораторной работе №2. Добавьте в него следующий код:

```

BR begin ; переход к началу приложения
пор
пор
пор
;Вычисление значение F(x,y)=ax^2+bxy+cy^2
;Размещаем данные
.data
a          .word 2
b          .word 3
c          .word 1
x          .word 5
y          .word 4

          ;Код программы снова в секции .text
          .text

begin ; начало программы

          ;устанавливаем DP
          ldp a

          ldi 0, R0 ;R0 используем для хранения результата
          ;вычисляем a*x*x
          ldi @a, AR0 ; AR0=a

```

```

mpyi @x, AR0 ; AR0=a*x
mpyi @x, AR0 ; AR0=a*x*x
;вычисляем b*x*y
ldi @b, AR1 ; AR1=b
mpyi @x, AR1 ; AR1=b*x
mpyi @y, AR1 ; AR1=b*x*y
;вычисляем c*y*y
ldi @c, AR2 ; AR2=c
mpyi @y, AR2 ; AR2=c*y
mpyi @y, AR2 ; AR2=c*y*y
;Сейчас AR0=ax^2 AR1=bxy AR2=cy^2
;Сложим эти значения
addi AR1, AR0; AR0=AR0+AR1
addi AR2, AR0; AR0=AR0+AR2
;AR0 содержит результат вычисленного выражения
;сохраним его по адресу 80001000h
ldhi 8000h,AR3
or 1000h,AR3 ;AR3=80001000h
sti AR0, *AR3
stop br stop; конец программы

```

Обратите внимание на следующие моменты.

Директива `.word` размещает указанные в ней 32 разрядные значения в текущей секции. То есть,

```

.data
a      .word 2
b      .word 3
c      .word 1
x      .word 5
y      .word 4

```

означает размещение слов 2, 3, 1, 5, 4 сначала секции .data. Так как в командном файле указано размещать секцию .data в памяти (LOCAL: origin = 0300000h length = 0700000h), то компоновщик разместит указанные значения, начиная с адреса 300000h (Рис. 3.1).

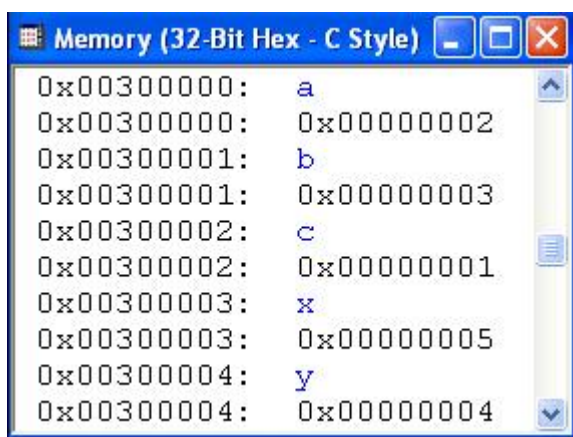


Рисунок 3.1 – Размещение секции .data

При этом обращение a означает адрес 300000h, @a есть обращение к данным по адресу метки (то есть, к 2). Перед обращением к данным @a необходимо установить корректное значение в регистр DP командой “ldr a”.

Самостоятельное задание 1

Вычислите значение следующих выражений. Полученный результат сохраните по адресу 301000h

Вариант	Задание
1	$F(x,y,z) = 5x - axyz + z^2$
2	$F(x,y,z) = x^3 + y^3 - axz$
3	$F(x,y,z) = -x - xy - ax^2 y^2 z^2$
4	$F(x,y,z) = ax^2 - by^2 + az$

Самостоятельное задание 2

Вариант	Задание
1	Вычислите определитель матрицы $\begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix}$
2	Вычислите матрицу $C=A*B$, где $A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$, $B = \begin{pmatrix} q & w \\ x & y \end{pmatrix}$
3	Вычислите произведение матрицы A на вектор B, где

	$A = \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix}, B = (x, y, z)$
4	Вычислите определитель матрицы $C = (A+B)$, где $A = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \quad B = \begin{pmatrix} e & f \\ g & h \end{pmatrix}$

Контрольные вопросы

1. Перечислите основные режимы адресации ПЦОС TMS320C40.
2. Перечислите основные регистры ПЦОС TMS320C40, каково их назначение?
3. Какие регистры следует использовать для операций с вещественными числами?
4. Как адресуется операнд при использовании прямого режима адресации?
5. Как считать данные с адреса 80001F00h при использовании прямого режима адресации?
6. Какие адреса карты памяти ПЦОС TMS320C40 отведены под внешнюю и внутреннюю память?
7. Как осуществляется адресация в косвенном режиме?
8. Перечислите основные режимы косвенной адресации.
9. Какая максимальная разрядность операнда при непосредственной адресации?
10. Назовите основные функциональные группы системы команд ПЦОС TMS320C40.
11. Приведите примеры использования основных инструкций загрузки и сохранения данных (ldi, sti, ldpk, pop, push).
12. Назовите основные инструкции арифметической и логической группы.
13. Каково назначение директивы .word?

14. Как обратиться к данным, размещенным по адресу соответствующей метки?

ЛАБОРАТОРНАЯ РАБОТА № 4

Тема: «Программы с использованием ветвлений и циклов на языке ассемблера. Утилиты Absolute Lister и Hex Conversion Utility»

Цель работы: изучить команды группы программного управления и команды повторения. С использованием этих команд научиться реализовывать ветвления и циклы в ассемблерных программах. Ознакомиться с утилитами Absolute Lister и Hex Conversion Utility.

Инструкции программного управления

Группа команд программного управления TMS320C40 состоит из 24 команд, влияющих на выполнение программы. Режим повторения обеспечивает повторение блока команд (RPTB или RPTBD) или отдельной команды (RPTS). Поддерживаются как стандартные, так и задержанные (одноцикловые) переходы.

Некоторые из команд программного управления могут зависеть от кодов условий, они приведены в таблице 4.1. Состояние флагов вы можете увидеть в последнем столбце окна View CPU Registers программы Code Composer. Сами флаги находятся в регистре состояния ST.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	NMI bus grant	xx	xx	ANALYSIS
R	R	R	R	R	R	R	R	R	R	R	R	R/W	R	R	R
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SC	PGIE	GIE	CC	CE	CF	PCF	RM	OVM	LUF	LV	UF	N	Z	V	C
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Таблица 4.1
Коды условий и флаги

Условие	Код	Описание	Флаг
(a) Безусловные сравнения			
U	00000	Безусловный	Безусловное
(b) Беззнаковые сравнения			
LO	00001	Меньше чем	C
LS	00010	Меньше или равно	C ИЛИ Z

Условие	Код	Описание	Флаг
HI	00011	Больше чем	$\sim C$ И $\sim Z$
HS	00100	Больше или равно	$\sim C$
EQ	00101	Равно	Z
NE	00110	Не равно	$\sim Z$
(с) Знаковые сравнения			
LT	00111	Меньше чем	N
LE	01000	Меньше или равно	N ИЛИ Z
GT	01001	Больше чем	$\sim N$ И $\sim Z$
GE	01010	Больше или равно	$\sim N$
EQ	00101	Равно	Z
NE	00110	Не равно	$\sim Z$
(d) Сравнение с нулем			
Z	00101	Ноль	Z
NZ	00110	Не ноль	$\sim Z$
P	01001	Положительное	$\sim N$ И $\sim Z$
N	00111	Отрицательное	N
NN	01010	Не отрицательное	$\sim N$
(e) Сравнение с флагами условий			
NN	01010	Не отрицательное	$\sim N$
N	00111	Отрицательное	N
NZ	00110	Не ноль	$\sim Z$
Z	00101	Ноль	Z
NV	01100	Нет переполнения	$\sim V$
V	01101	Переполнение	V
NUF	01110	Нет отрицательного переполнения	$\sim UF$
UF	01111	Отрицательное переполнение	UF
NC	00100	Нет переноса	$\sim C$
C	00001	Перенос	C

В таблице 4.2 приведены основные инструкции группы программного управления.

Таблица 4.2
Некоторые команды программного управления

Команда	Описание
Bcond src	Если условие истинно: Если src - регистр, то src \rightarrow PC. Если src - метка, то смещение + PC + 1 \rightarrow PC. Если условие ложно, то нет перехода. Переход по условию (стандартный)
BcondD src	Переход по условию (задержанный) Задержанный означает

	выполнение следующих трёх инструкции перед выполнением перехода.
BR src	PC + 1 + смещение -> PC Безусловный переход (стандартный)
BRD src	PC + 3 + смещение -> PC Безусловный переход (задержанный)
CALL src	Следующий PC в стек PC + 1 + смещение -> PC Вызов подпрограммы
CALLcond src	Если условие истинно то: Следующий PC в стек Если src - регистр, то src -> PC. Если src - метка, то смещение + PC + 1 -> PC. Если условие ложно, то вызова не происходит. Вызов подпрограммы по условию
NOP	Нет операции
RETlcond src	Если условие истинно то PC загружается из стека ST(PGIE) -> ST(GIE) ST(PCF) -> ST(CF) Возврат из прерывания по условию
RETScond src	PC загружается из стека Возврат из подпрограммы по условию
RPTB src	Повтор блока команд
RPTS	Повтор отдельной команды
TRAPcond src	Если условие истинно, то: ST(GIE) -> ST(PGIE) ST(CF) -> ST(PCF) 0 -> ST(GIE) 1 -> ST(CF) next PC -> *(++SP) trap вектор N -> PC Условное программное прерывание

Пример 1 Использования RPTS

ldi 1, R1

rpts 3

addi 1, R1

Команда addi выполнится 4 раза (3, 2, 1, 0 – соответствующие значения регистра повторений). По окончании выполнения rpts R1=5.

Пример 2 Использование RPTB

```
ldi 3, RC ; загружаем счетчик повторений - 4
rptb EndRBlock
```

; выполняем команды до метки EndRBlock RC+1 раз

```
addi 1, R1
```

```
addi 2, R2
```

```
EndRBlock nop
```

Пример 3 Реализация ветвления

Реализуем следующий условный оператор:

If ($x < y$) $z = z + 1$

else $z = z - 1$

; загружаем данные в регистры

```
ldi @x, AR0
```

```
ldi @y, AR1
```

```
ldi @z, AR7
```

```
cmpi AR0, AR1 ; сравниваем x и y
```

; переходим на метку thenlbl, если $AR1 = y > AR0 = x$

```
bgt thenlbl
```

; переходим на elselbl, если $AR1 \leq AR0$

```
br elselbl
```

```
thenlbl
```

```
addi 1, AR7 ; выполняем команды ветки then
```

```
sti AR7, @z
```

```
br endiflbl ; и перепрыгиваем ветку else
```

```
elselbl
```

```
subi 1, AR7 ; выполняем команды ветки else
```

```
sti AR7, @z
```

```
endiflbl
```

Пример 4 Реализация цикла

Реализуем следующий циклический оператор


```
while (x!=0) {y=y+1; x=x-1 }
```

```
    ; загружаем данные в регистры
```

```
    ldi @x, AR0
```

```
    ldi @y, AR1
```

```
    ; сравниваем AR0=x с нулем
```

```
    cmpi 0, AR0
```

```
    bzs end ; если AR0=0, то не выполняем операторы цикла
```

```
whilestart ; операторы цикла
```

```
    addi 1, AR1
```

```
    subi 1, AR0
```

```
    cmpi 0, AR0 ; снова сравниваем x=AR0 с нулём
```

```
    bnzs whilestart ; если не ноль, то возвращаемся на метку whilestart
```

и снова выполняем операторы тела цикла.

```
end:
```

Упражнение 1

Напишем программу, определяющую максимальный элемент массива из N элементов. В Code Composer создайте и сохраните новый проект, для этого выберите Project >> New... в главном меню программы. В папку, где вы сохранили проект, скопируйте (.asm) файл программы инициализации процессора и (.cmd) командный файл компоновщика, созданные в лабораторной работе № 2. Добавьте эти файлы в проект в Code Composer с помощью функции Project >> Add Files to Project... главного меню программы.

Добавьте в программу инициализации процессора следующий код:

```
BR begin ; переход к началу приложения
```

```
por
```

```
por
```

```
por
```

```
;Размещаем данные
```

```
.data
```

```

;длина массива
N      .word 10

;данные массива
M      .word 45h, 30h, 2h, 5h, 1h, 222h, 12h, 11h, 44h, 55h
Madr   .word M+1 ;сохраняем адрес второго слова

.text

begin ; начало программы

    ldp N

    ldi @N, AR7 ;загружаем длину массива

    ldi @M, AR0 ;загружаем первое слово и считаем его максимальным

        ldi @Madr, AR1 ;загружаем адрес второго слова

        ;проверяем слова со второго по N

        subi 2, AR7

        ldi AR7, RC ;счетчик

        rptb loop

        cmpi *AR1, AR0 ;сравниваем очередное слово с максимальным

        bgt l1 ;если текущее максимальное AR0 больше, то переход на l1

        ldi *AR1, AR0 ;изменяем текущее максимальное

l1      addi 1, AR1 ;увеличиваем текущий адрес на 1

loop   nop

        ;сохраняем результат по адресу 80001000h

        ldhi 8000h, AR1

        or 1000h, AR1

        sti AR0, *AR1

stop   br stop; конец программы

```

Откомпилируйте, скомпонуйте и запустите программу на выполнение.
 Выполните программу с различными исходными данными массива М.

Создание листинга программы

Утилита **Absolute Lister** является средством отладки. Входными данными для неё является исполняемый объектный файл .out, выходными данными является файл .abs, этот файл можно ассемблировать для получения листинга, содержащего абсолютные адреса объектного кода.

Для создания листинга создайте в папке с проектом bat файл со следующими командами:

```
abs30 FileName.out
asm30 -v40 -a FileName.abs
PAUSE
```

Укажите корректные имена файлов, далее запустите bat файл. При этом будут созданы файлы FileName.abs и FileName.lst. Откройте и ознакомьтесь с lst файлом в любом текстовом редакторе, например, в WordPad.

Создание hex кода программы

Ассемблер и компоновщик создают исполняемый COFF файл – это бинарный файл. Для некоторых задач (например, запись программы на EPROM) необходимо получить текстовое представление в одном из hex форматов. Для этих целей и служит **Hex Conversion Utility**.

Создайте hex код вашего приложения в формате ASCII. Для этого создайте в папке с проектом bat файл со следующими командами:

```
hex30 InputFile.out -a -romwidth 32 -o OutputFile.hex
PAUSE
```

Скорректируйте имена входного и выходного файла и запустите bat файл. При этом будет создан hex файл, откройте его в текстовом редакторе. Файл будет иметь примерно следующее содержимое:

```
00 30 00 13 00 00 00 00
$A300000,
00 00 00 0A 00 00 00 45 00 00 00 30 00 00 00 02 00 00 00 05 00 00 00 01
00 00 02 22 00 00 00 12 00 00 00 11 00 00 00 44 00 00 00 55 00 30 00 02
00 2F FC 00 00 00 00 00 00 2F FC 00 00 00 00 01 1E F7 80 00 1E F7 80 00
00 10 00 00 50 70 00 30 08 28 00 0D 76 80 00 08 08 28 00 0E 76 81 00 08
```

08 28 00 12 08 20 00 10 15 40 C0 00 08 20 00 11 15 40 00 04 08 34 00 0C
 08 37 00 0F 10 75 38 00 60 00 00 03 0C 80 00 00 0C 80 00 00 0C 80 00 00
 50 70 00 30 08 2F 00 00 08 28 00 01 08 29 00 0B 18 6F 00 02 08 1B 00 0F
 64 00 00 04 04 C8 C1 00 6A 09 00 01 08 48 C1 00 02 69 00 01 0C 80 00 00
 1F E9 80 00 10 69 10 00 15 48 C1 00 60 FF FF FF 0C 80 00 00 0C 80 00 00
 0C 80 00 00

В отдельных строчках, начиная с символа \$A, указан адрес начала секции. В остальных строчках находится текстовое представление содержимого секций (команд и данных). Первая секции начинается с нулевого адреса.

Самостоятельное задание

Вариант	Задание
1	Дан массив из n элементов. Найдите сумму его элементов. Значение n и элементы задаются в ячейках памяти. Полученный результат сохраните по адресу 301000h.
2	Дан массив из n целых чисел. Проверить, есть ли в нём заданное целое число. Значение n , искомое целое и элементы задаются в ячейках памяти. Полученный результат сохраните по адресу 80001000h
3	Дан массив из n целых чисел. Подсчитайте количество отрицательных элементов в нём. Значение n и элементы задаются в ячейках памяти. Полученный результат сохраните по адресу 80000050h
4	В ячейке памяти задано натуральное n , заполните следующие n адресов памяти по правилу: $a[i] = (100 - i) * (2 + i), i = 1..N$
5	В ячейке памяти задаётся целое x . Вычислите значение $f(x)$, и сохраните его по адресу 80000150h $f(x) = \begin{cases} 5x^3, & x > 10 \\ 4x^2, & -10 \leq x \leq 10 \\ 2x, & x < -10 \end{cases}$
6	В ячейках памяти задано натуральное n , x , значения массива $a[n]$. Вычислите значение выражения, и сохраните его по адресу 80001150h $y = a[1]x^1 + a[2]x^2 + \dots + a[10]x^{10}$
7	В ячейках памяти задано натуральное n и значения массива $a[n]$.

	Проверьте, образуют ли его элементы арифметическую прогрессию. Если образуют, то запишите 1 по адресу 80001000h, в противном случае запишите -1.
8	В ячейке памяти задаётся целое x . Вычислите значение $f(x)$, и сохраните его по адресу 80000250h $f(x) = \begin{cases} 1, & x > 0 \\ 0, & x = 0 \\ -1, & x < 0 \end{cases}$

Контрольные вопросы

1. В каком регистре находятся флаги, используемые командами программного управления?
2. Назовите условия знаковых и беззнаковых сравнений.
3. Назовите условия сравнения с нулём.
4. Какую команду следует использовать для перехода на метку?
5. С помощью какой команды можно вызвать подпрограмму?
6. Каково назначение инструкции NOP?
7. Для чего служат инструкции reti и rets?
8. Приведите примеры использования команд повторения (одной команды и блока).
9. Что означает "задержанная" (delated) инструкция?
10. Какое назначение утилиты Absolute Lister?
11. Как создать hex код исполняемого файла программы?

ЛАБОРАТОРНАЯ РАБОТА № 5

Тема: «Программы с использованием стека подпрограмм»

Цель работы: изучить команды вызова подпрограмм и возврата из них, команды работы со стеком. Научиться писать и выполнять программы с использованием подпрограмм.

Стек - структура данных с методом доступа к элементам LIFO (англ. Last In - First Out, «последним пришёл - первым вышел»). Чаще всего принцип работы стека сравнивают со стопкой тарелок: чтобы взять вторую сверху, нужно снять верхнюю.

Добавление элемента, называемое также проталкиванием (push), возможно только в вершину стека (добавленный элемент становится первым сверху). Удаление элемента, называемое также выталкивание (pop), возможно также только из вершины стека, при этом, второй сверху элемент становится верхним.

Для добавления элемента следует использовать инструкцию PUSH, а для считывания элементов из стека – инструкцию POP. Например:

push AR0; помещаем в стек

pop AR0; считываем из стека

; для 40 разрядных регистров с вещественными числами следует применять следующие команды

push R1;

pushf R1;

popf R1;

pop R1;

Подпрограмма (англ. subroutine) - поименованная часть программы, содержащая описание определённого набора действий. Подпрограмма может быть многократно вызвана из разных частей программы.

Ассемблерную подпрограмму обычно оформляют следующим образом:

SubrName ;метка – имя подпрограммы

;тело подпрограммы – команды

push AR4

ldi 1, AR4

pop AR4

rets

pop

pop

pop

Вызов подпрограммы осуществляется командой call SubrName, при этом в стеке сохраняется адрес возврата. Подпрограмма должна

заканчиваться командой `rets`, которая осуществляет возврат к вызвавшей подпрограмму программе (загружает адрес возврата из стек в счетчик команд PC). Если в подпрограмме модифицируются какие-либо регистры, то рекомендуется их сохранять в стек в начале выполнения подпрограммы и восстанавливать перед выходом из подпрограммы, чтобы не испортить данные главной программы.

После инструкций, модифицирующих счетчик команд, настоятельно рекомендуется вставить три инструкции `nop`.

Упражнение 1

Создадим подпрограмму инициализации вспомогательных регистров, а также подпрограммы сохранения контента и его восстановления.

В Code Composer создайте новый файл с исходным кодом (File >> New >> Source File). Сохраните его под именем с расширением `asm`.

Добавьте в него код следующей подпрограммы, инициализирующей вспомогательные регистры.

`;Инициализация вспомогательных регистров`

`InitAR`

`ldi 0, AR0`

`ldi 0, AR1`

`ldi 0, AR2`

`ldi 0, AR3`

`ldi 0, AR4`

`ldi 0, AR5`

`ldi 0, AR6`

`ldi 0, AR7`

`rets`

`nop`

`nop`

`nop`

В приложении (например, в программе инициализации процессора) добавьте команду вызова этой подпрограммы:

```
call InitAR
```

Чтобы подпрограмма из другого исходного файла была видна, сделайте перед точкой входа в приложение следующую ссылку:

```
.include "subr.asm"
```

```
_c_int00:
```

В кавычках следует указать путь к файлу с подпрограммой.

Директивы `.include` / `.coru` указывают ассемблеру на необходимость считывания кода из другого файла. При использовании директивы `.coru` этот код будет включен в листинг программы, а при использовании `.include` – не будет.

Скомпилируйте, скомпонуйте и выполните программу.

Самостоятельное задание 1

Напишите подпрограмму, инициализирующую область памяти длины *len* слов начиная с адреса *adr* значением *pat*. Перед выходом из подпрограммы восстановите значения используемых в ней регистров. Продемонстрируйте работу подпрограммы.

Вариант	Параметры
1	adr = 2FF800h len = 100h pat = 0AAAAAAAAAh
2	adr = 80001000h len = 200h pat = 0FFFFFFFFh
3	adr = 80001200h len = 100h pat = 055555555h
4	adr = 80001500h len = 50h pat = 012345678h
5	adr = 80001800h len = 100h pat = 0AAAA5555h

6	adr = 80001100h len = 200h pat = 0555AAAAh
7	adr = 80001000h len = 50h pat = 01111111h
8	adr = 80001400h len = 100h pat = 087654321h

Макросы в ассемблерной программе

Ассемблер TMS320C40 поддерживает язык макросов, которые позволяют создать собственные “инструкции”. Это особенно полезно, чтобы выполнить одни и те же действия с различными параметрами.

Вызов макроса осуществляется указанием его имени в коде программы. Перед вызовом необходимо определить макрос. Макросы могут быть описаны в самом начале файла с исходным кодом программы, в файлах из директив `.copy` / `.include` или в специальных библиотеках макросов.

Макрос определяется следующим образом:

```
macname .macro [parameter1 ] [, ... , parameterN ]
```

```
    ;код подпрограммы
```

```
[.mexit] ;необязательный параметр перехода к концу макроса
```

```
.endm ;конец макроса
```

Пример 1 Следующий макрос записывает 16 разрядное `inf` по адресу `hw` (старшие 16 разрядом адреса) `lw` (младшие 16 разрядом адреса)

```
Write .macro inf, hw, lw
```

```
    ldi inf, AR6
```

```
    ldhi hw, AR0
```

```
    or lw, AR0
```

```
    sti AR6, *AR0
```

```
.endm
```

При использовании меток в макросе необходимо добавлять знак ? после имени метки для обеспечения её уникальности при многократном вызове макроса.

Упражнение 2

Ниже приведены примеры макросов, которые сохраняют и восстанавливают программный контент.

;Сохранение контента приложения

SaveContent .macro

```

    PUSH    R0      ; State save
    PUSHF   R0      ;
    PUSH    R1      ;
    PUSHF   R1      ;
    PUSH    R2      ;
    PUSHF   R2      ;
    PUSH    R3      ;
    PUSHF   R3      ;
    PUSH    R4      ;
    PUSHF   R4      ;
    PUSH    R5      ;
    PUSHF   R5      ;
    PUSH    R6      ;
    PUSHF   R6      ;
    PUSH    R7      ;
    PUSHF   R7      ;
    PUSH    R8      ;
    PUSHF   R8      ;
    PUSH    R9      ;
    PUSHF   R9      ;
    PUSH    R10     ;
    PUSHF   R10     ;

```

```

    PUSH    R11    ;
    PUSHF    R11    ;
    PUSH    AR0    ;
    PUSH    AR1    ;
    PUSH    AR2    ;
    PUSH    AR3    ;
    PUSH    AR4    ;
    PUSH    AR5    ;
    PUSH    AR6    ;
    PUSH    AR7    ;
    PUSH    DP      ;
    PUSH    IR0     ;
    PUSH    IR1     ;
    PUSH    BK      ;
    PUSH    DIE     ;
    PUSH    IIE;
    PUSH    IIF;
    PUSH    RS      ;
    PUSH    RE      ;
    PUSH    RC      ;
    LDEP    IVTP,R1  ;
    PUSH    R1      ;
    LDEP    TVTP,R1  ;
    PUSH    R1      ;
    .endm

```

LoadContent .macro

```

    POP     R1      ; Restore IVTP and TVTP
    LDPE    R1, TVTP
    POP     R1

```

LDPE	R1, IVTP
POP	RC
POP	RE
POP	RS
POP	IIF
POP	IIE
POP	DIE
POP	BK
POP	IR1
POP	IR0
POP	DP
POP	AR7
POP	AR6
POP	AR5
POP	AR4
POP	AR3
POP	AR2
POP	AR1
POP	AR0
POPF	R11
POP	R11
POPF	R10
POP	R10
POPF	R9
POP	R9
POPF	R8
POP	R8
POPF	R7
POP	R7
POPF	R6

```

POP      R6
POPF     R5
POP      R5
POPF     R4
POP      R4
POPF     R3
POP      R3
POPF     R2
POP      R2
POPF     R1
POP      R1
POPF     R0
POP      R0

.endm

```

Добавьте эти макросы в файл “subr.asm”, созданный в упражнении 1, а в приложении вызовите макросы:

```

SaveContent
call InitAR
LoadContent

```

В отличие от подпрограмм для обращения к макросам не нужна команда `call`, достаточно указать только имя и параметры макроса.

Самостоятельное задание 2

Создайте макрос с функциями, аналогичными подпрограмме из самостоятельного задания 1. Начальный адрес памяти *adr*, длину *len* и данные *pat* передайте макросу в качестве параметров.

Контрольные вопросы

1. Что такое стек?
2. Какие команды служат для работы со стеком?
3. Как поместить в стек и считать из стека 40 разрядные регистры R0-R11?

4. Для чего служат подпрограммы?
5. Как вызвать подпрограмму?
6. Для чего служит команда `rets`?
7. Какое назначение директив `.copy` и `.include`, в чём заключается их функциональное различие?
8. В чём отличие макросов от подпрограмм?
9. Как определяется макрос?

ЛАБОРАТОРНАЯ РАБОТА № 6

Тема: «Программы с обработкой прерываний»

Цель работы: изучить типы прерываний процессора TMS320C40 и методы их обработки. Научиться составлять и выполнять программы с обработкой прерываний.

TMS320C40 поддерживает четыре внешних прерывания (PIF3-0), несколько внутренних прерываний, немаскируемое внешнее прерывание, прерывание RESET, прерывания контроллера прямого доступа к памяти и коммуникационных портов.

Для корректной обработки прерываний в программе необходимо выполнить следующие действия:

1. Разместить таблицу векторов прерываний в памяти.
2. Инициализировать регистр IVTP.
3. Создать программы обработки прерываний.
4. Создать программный стек.
5. Разрешить необходимые прерывания.
6. Разрешить прерывания глобально.
7. Сгенерировать сигнал прерывания.
8. Рассмотрим каждое из этих действий.

1. Таблица векторов прерываний (IVT) содержит векторы прерываний в порядке убывания их приоритета. Вектор прерывания – это адрес программы обработки прерывания, то есть той подпрограммы, которая будет вызвана

при возникновении прерывания. Таблица IVT должна быть расположена в границах 512 слов. Таблица прерываний приведена на рисунке 6.1.

000h	Reserved	01Dh	ICFULL4
001h	NMI	01Eh	ICRDY4
002h	TINT0	01Fh	OCRDY4
003h	IIOF0	020h	OCEMPTY4
004h	IIOF1	021h	ICFULL5
005h	IIOF2	022h	ICRDY5
006h	IIOF3	023h	OCRDY5
007h	Unused	024h	OCEMPTY5
.		025h	DMA INT0
00Ch		026h	DMA INT1
00Dh	ICFULL0	027h	DMA INT2
00Eh	ICRDY0	028h	DMA INT3
00Fh	OCRDY0	029h	DMA INT4
010h	OCEMPTY0	02Ah	DMA INT5
011h	ICFULL1	02Bh	TINT1
012h	ICRDY1	02Ch	Unused
013h	OCRDY1	.	
014h	OCEMPTY1	.	
015h	ICFULL2	.	
016h	ICRDY2	.	
017h	OCRDY2	.	
018h	OCEMPTY2	.	
019h	ICFULL3	.	
01Ah	ICRDY3	.	
01Bh	OCRDY3	.	
01Ch	OCEMPTY3	03Eh	Unused
		03Fh	Reserved

Рисунок 6.1 – Таблица векторов прерываний

Нулевой адрес зарезервирован для вектора reset. В ячейке NMI должен быть адрес подпрограммы обработки немаскируемого внешнего прерывания, далее TINT0 – адрес обработчика прерывания таймера, затем обработчики внешних прерываний, прерываний коммуникационных портов, контроллера прямого доступа к памяти (DMA), второго прерывания таймера.

В программе векторы прерываний можно разместить следующим образом.

```
_myvect .sect "myvect" ;Создаём секцию для векторов прерываний
reset      .word _c_int00 ;RESET вектор
```

```

NMI      .space 1 ;Резервируем место для NMI прерывания
TINT0    .word TINT0subr

```

Для прерывания reset адрес обработки указан как адрес метки `_c_int00`, то есть точка входа в программу. Для прерывания таймера TINT0 – адрес TINT0subr – заголовок подпрограммы обработки прерывания. Для прерывания NMI адрес процедуры обработки прерывания не определён. Директива `.space` резервирует место для указанного после неё числа 32 разрядных слов.

2. Регистр IVTP должен содержать адрес начала таблицы векторов прерываний IVT. Инициализировать этот регистр можно, например, так:

```

;в секции .data или в .text перед точкой входа в программу _c_int00
ivta    .word _myvect ; адрес myvect секции размещаем по метке ivta
; в .text после точки входа в программу
ldp     ivta ;устанавливает Data Pointer
ldi     @ivta, AR0 ;содержимое по метке в регистр
ldpe    AR0, IVTP ;инициализирует регистр IVTP

```

3. Программа обработки прерывания создаётся аналогично подпрограммам, при этом заканчивается командой `reti`, а не `rets`. Так как мы указали для прерывания таймера адрес заголовка TINT0subr, то его необходимо предварительно создать:

```

TINT0subr
        пор ;здесь размещаются команды обработки прерывания
        reti

```

4. При работе с прерываниями необходим программный стек. Это можно создать следующим образом:

```

;создаём неинициализированную секцию под стек размером 100 слов
_mystack .usect "mystack",100 ; резервируем 100 слова под стек
;помещаем адрес этой секции в ячейку с меткой stacka
stacka   .word _mystack ; адрес mystack секции

```


;загружаем этот адрес в регистр SP

ldp stacka

ldi @stacka, SP

5. Внутренние прерывания разрешаются в регистре ПЕ (Рис 6.2). Для разрешения прерывания необходимо выставить в единицу соответствующий бит регистра.

31	30	29	28	27	26	25	24	23	22	21
ETINT1	EDMA INT5	EDMA INT4	EDMA INT3	EDMA INT2	EDMA INT1	EDMA INT0	EOC-EMPTY5	EOC-RDY5	EIC-RDY5	EIC-FULL5
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
20	19	18	17	16	15	14	13	12	11	10
EOC-EMPTY4	EOC-RDY4	EIC-RDY4	EIC-FULL4	EOC-EMPTY3	EOC-RDY3	EIC-RDY3	EIC-FULL3	EOC-EMPTY2	EOC-RDY2	EIC-RDY2
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
9	8	7	6	5	4	3	2	1	0	
EOC-EMPTY1	EOC-RDY1	EIC-RDY1	EIC-FULL1	EOC-EMPTY0	EOC-RDY0	EIC-RDY0	EIC-FULL0	ETINT0		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	

Рисунок 6.2 – Регистр ПЕ

Внешние прерывания разрешаются в регистре ПФ (Рис. 6.3), в этом же регистре находятся флаги внутренних прерываний (устанавливаются в 1 при возникновении прерывания).

31	30	29	28	27	26	25	24
TINT1	DMAINT5	DMAINT4	DMAINT3	DMAINT2	DMAINT1	DMAINT0	TINT0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
23	22	21	20	19	18	17	16
xx	xx	xx	xx	xx	xx	xx	NMI
R	R	R	R	R	R	R	R
15	14	13	12	11	10	9	8
EIIOF3	FLAG3	TYPE3	FUNC3	EIIOF2	FLAG2	TYPE2	FUNC2
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
7	6	5	4	3	2	1	0
EIIOF1	FLAG1	TYPE1	FUNC1	EIIOF0	FLAG0	TYPE0	FUNC0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Рисунок 6.3 - Регистр ПФ

6. Глобально прерывания разрешаются установкой в единицу бита GIE (бит 13) статусного регистра ST, запись в этот бит нуля запрещает прерывания (все, кроме немаскируемого).

or 2000h, ST

Для установки в 1 нужного бита, обычно выполняют логические И с числом, у которого в нужном разряде 1, а в остальных – 0. Так, 2000h = 100000000000000b.

7. Для возникновения сигнала прерывания необходимо, чтобы возникли условия, специфические для каждого прерывания. Внешние возникают путём подачи сигнала на соответствующий вход процессора (ПОФ). Можно вызвать прерывание программно установкой в единицу соответствующего бита регистра ПФ.

Упражнение 1 Создание тестовой программы с обработкой прерывания таймера

Создайте в Code Composer новый проект, добавьте в него ассемблерный файл с исходным кодом программы инициализации процессора, а также командный файл компоновщика.

Добавьте в программу следующий код:

```
begin ; начало программы
    or 2000h, ST ;глобально разрешаем прерывания
    or 1, PE    ;разрешаем прерывание таймера
    ldhi 100h, AR0 ;маска для вызова прерывания таймера
    or AR0, PF ;вызов прерывания таймера
stop
    br stop; конец программы
nop
nop
nop
TINT0subr ;обработчик прерывания таймера
    ldhi 8000h, AR7
    or 1000h, AR7
    stik 1, *AR7
    reti
.end
```

Укажите адрес TINT0subr в качестве вектора прерывания таймера TINT0. Убедитесь, что все пункты 1-7, необходимые для корректной обработки прерываний, выполнены. Откомпилируйте и скомпонуйте программу. Запустите программу с помощью функции Debug >> Run (F5). Через некоторое время остановите выполнение программы Debug >> Halt (shift + F5). Убедитесь, что подпрограмма обработки прерывания отработала (80001000h=1).

Самостоятельное задание 1

Создайте программу, которая программно вызывает и обрабатывает следующее прерывание:

Вариант	Прерывание
1	Немаскируемое прерывание NMI
2	Прерывание контроллера прямого доступа к памяти DMA0
3	Прерывание контроллера прямого доступа к памяти DMA1
4	Прерывание контроллера прямого доступа к памяти DMA2
5	Прерывание контроллера прямого доступа к памяти DMA3
6	Прерывание контроллера прямого доступа к памяти DMA4
7	Прерывание контроллера прямого доступа к памяти DMA5
8	Прерывание таймера TINT1

Для разрешения прерываний DMA запишите значение 0FFFFFFFh в регистр DIE

Контрольные вопросы

1. Какие прерывания поддерживает ПЦОС TMS320C40?
2. Какие действия необходимо выполнить для корректной обработки прерываний?
3. Какое назначение регистра IVTP?
4. Какое назначение директивы .space?
5. Какая команда служит для возврата из обработчика прерывания?
6. В каких регистрах разрешаются внутренние и внешние прерывания?
7. В каком регистре расположены флаги прерываний?
8. Как глобально разрешить/запретить все прерывания кроме немаскируемого?

9. Как программно вызвать прерывание?

ЛАБОРАТОРНАЯ РАБОТА № 7

Тема: «Программирование периферийных устройств»

Цель работы: ознакомиться с периферийными устройствами TMS320C40, научиться программировать контроллер прямого доступа к памяти (ПДП) и таймер, составить программы обработки прерываний периферийных устройств.

Все периферийные устройства TMS320C40 контролируются через регистры, картированные в карте памяти ПЦОС. Периферийные устройства TMS320C40 включают два таймера и шесть коммуникационных портов, контроллер ПДП.

Программирование контроллера ПДП

Контроллер ПДП – встроенное программируемое устройство, позволяющее одновременно осуществлять передачу данных из одной области памяти в другую и выполнять операции процессора с наименьшими потерями.

Контроллер ПДП поддерживает 6 каналов ПДП, которые выполняют передачу данных из любого места карты памяти TMS320C40.

Каждый канал ПДП управляется девятью регистрами, которые расположены в периферийном адресном пространстве TMS320C40 (Рис. 7.1).

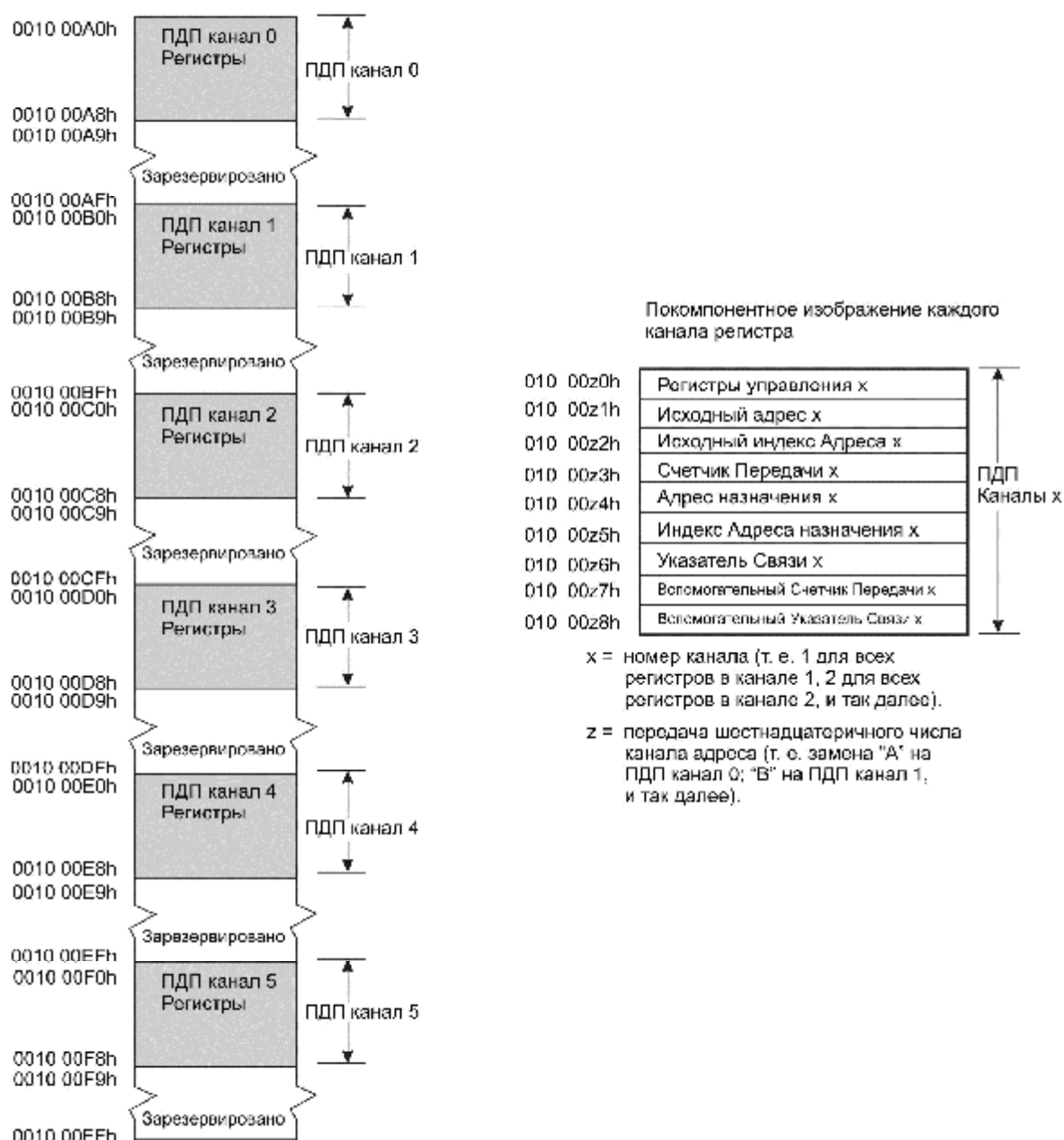


Рисунок 7.1 – Регистры ПДП

Регистр управления: содержит информацию о состоянии и режиме канала ПДП.

Регистр начального адреса: содержит адрес памяти, по которому будут считываться данные.

Индексный регистр начального адреса: содержит значение шага (знаковое 32-разрядное число), используемое для инкрементирования или декрементирования регистра начального адреса.

Регистр конечного адреса: содержит адрес памяти, куда будут записываться данные.

Индексный регистр конечного адреса: содержит значение шага (знаковое 32-разрядное число), используемое для инкрементирования или декрементирования регистра конечного адреса.

Регистр-счетчик передачи: содержит размер блока данных.

Вспомогательный регистр-счетчик передачи: содержит размер блока данных, передаваемых в режиме разделения (вспомогательный канал).

Регистр-указатель: содержит адрес памяти с данными для автоинициализации регистров канала ПДП. Используется в основном режиме или режиме разделения для основного канала.

Вспомогательный регистр-указатель: содержит адрес памяти с данными для автоинициализации регистров канала ПДП. Используется в режиме разделения для вспомогательного канала.

После сброса регистр управления, счетчик передачи, вспомогательный счетчик передачи устанавливаются в 0, а остальные регистры в неопределенное состояние.

Подробное описание каждого бита регистров и различных режимов работы ПДП приведено в техническом описании на процессор.

Для пересылки данных необходимо выполнить следующие действия.

Инициализация регистров

В регистр начального адреса канала ПДП загружается адрес, откуда будет производиться чтение.

В регистр конечного адреса того же канала ПДП загружается адрес, куда будет производиться запись.

В счетчик передачи загружается число передаваемых слов.

В индексный регистр начального/конечного адреса загружается шаг обновления регистра начального/конечного адреса.

В регистр управления каналом ПДП загружаются установки соответствующего режима для синхронизации чтения и записи сопроцессора

ПДП с прерываниями. Регистр DIE определяет, какое прерывание используется для синхронизации.

Запуск ПДП

Сопроцессор ПДП запускается посредством поля DMA START (биты 22-23) в регистре управления каналом ПДП.

После завершения передачи данных сопроцессор ПДП может быть запрограммирован для следующего:

Остановлен до перепрограммирования (разряды TRANSFER MODE=01₂ – биты 2-3 регистра управления)

Продолжение передачи данных (разряды TRANSFER MODE=00₂)

Сгенерировать прерывание для сигнализации ЦПУ, что передача данных завершена (разряд TCC=1₂ – 18 бит регистра управления)

Автоинициализироваться для запуска передачи следующего блока данных (разряды TRANSFER MODE=10₂ или 11₂).

Упражнение 1 Создадим приложение, которое демонстрирует базовую функцию контроллера ПДП – пересылку данных из одной области памяти в другую, не загружая при этом процессор.

Создайте и сохраните новый проект в Code Composer. Добавьте в проект файл с исходным кодом инициализации процессора, а также командный файл компоновщика.

Добавьте в программу следующий код:

BR begin ; переход к началу приложения

nor

nor

nor

.data

DMA0Controla .word 01000A0h ;адрес регистра DMA0Control

Srcadr .word 301000h ;адрес данных для пересылки

Destadr .word 80001000h ;адрес назначения

Trcounter .word 100h ;число пересылаемых слов

```

Control      .word 0C40007h ;значение для регистра управления
               .text
begin ; начало программы
               ;загружаем данные
               ldhi 30h, AR7
               or 1000h, AR7
               ldhi 7654h, AR6
               or 3210h, AR6
               rpts 99h
               sti AR6, *AR7++
               ;инициализируем регистры DMA
               ldp DMA0Controla
               ldi @DMA0Controla, AR0 ;загружаем адрес регистра управления
               ldi @Srcadr, R0
               sti R0, *+AR0(1);загружаем адрес данных для пересылки
               stik 1, *+AR0(2);загружаем индекс
               ldi @Trcounter, R0
               sti R0, *+AR0(3);загружаем счетчик пересылаемых слов
               ldi @Destadr, R0
               sti R0, *+AR0(4);загружаем адрес назначения
               stik 1, *+AR0(5);индекс назначения
               ;запускаем DMA
               ldi @Control, R0
               sti R0, *AR0
stop
               br stop; конец программы

```

Откомпилируйте, скомпонуйте и выполните программу. Убедитесь, что данные с 301000h успешно отправлены и записаны с адреса 80001000h.

Самостоятельное задание 1

1. Создайте программу, в которой с помощью контроллера ПДП пересылаются *count* слов с начального адреса *srcadr* в область памяти с адреса *destadr* по каналу *DMAx*. Область памяти с адреса *srcadr* проинициализируйте значением *pat*.

2. Создайте подпрограмму обработки прерывания для соответствующего канала ПДП. В этой подпрограмме подсчитайте контрольную сумму переданных данных, и сохраните её в ячейке памяти с адресов 80000900h.

Указание: для разрешения прерывания канала ПДП необходимо разрешить соответствующие прерывания в регистре ПЕ (биты 30-23 для EDMAInt5- EDMAInt0 соответственно), установить в 1 бит 18 регистра управления ПДП.

Вариант	Параметры
1	DMAx = 5 srcadr = 301150h destadr = 80001500h pat = 01234567h count = 170h
2	DMAx = 4 srcadr = 301500h destadr = 80001200h pat = 0AAAAAAAAAh count = 100h
3	DMAx = 3 srcadr = 301200h destadr = 80001400h pat = 55555555h count = 180h
4	DMAx = 2 srcadr = 301300h destadr = 80001100h pat = 0FFFFFFFh count = 80h
5	DMAx = 1 srcadr = 80001400h destadr = 301100h pat = 033335555h count = 400h

6	DMAx = 0 srcadr = 80001300h destadr = 301200h pat = 0A5A5A5A5h count = 350h
7	DMAx = 1 srcadr = 80001200h destadr = 301300h pat = 022229999h count = 50h
8	DMAx = 0 srcadr = 80001100h destadr = 301400h pat = 0FFFF0001h count = 200h

Программирование таймера

Таймеры TMS320C40 - 32-разрядные счетчики общего назначения, работающие как таймер или счетчик событий, с двумя режимами сигнализации и внутренним или внешним тактированием.

Каждый таймер состоит из 32-разрядного счетчика, компаратора, селектора входных тактов, импульсного генератора и дополнительной аппаратной части.

Таймер считает циклы входных тактов. Когда этот счет (регистр счетчик) достигает до значения, сохраненного в регистре периода, счетчик обращается в 0 и генерирует выходной сигнал.

Входной тактовый сигнал таймера может быть $N1/2$ внутренней частоты TMS320C40 или внешних тактов на $TCLKx$.

Таймеры управляются тремя регистрами:

- Регистр управления. Этот регистр определяет режим работы таймера, отражает его состояние и управляет функциональностью выводов входа/выхода.
- Регистр периода. Этот регистр определяет частоту выдачи сигналов таймером.

- Регистр-счетчик. Этот регистр содержит текущее значение инкрементируемого счетчика.

Регистр	Адрес периферии	
	Таймер 0	Таймер 1
Регистр управления таймером	100020h	100030h
Регистр счетчика таймера	100024h	100034h
Регистр периода таймера	100028h	100038h

Подробное описание функций и режимов работы таймера приведено в техническом описании TMS320C40.

Регистр управления таймером

На рисунке 7.2 показаны разряды регистра, их имена и функции. Разряды 3-0 - разряды управления портом; разряды 11-6 - разряды глобального управления таймера. Необходимо обратить внимание, что при сбросе все разряды устанавливаются в 0, за исключением DATIN (устанавливается по значению на TCLK).

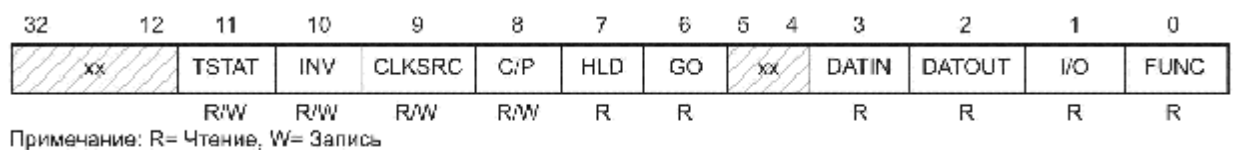


Рисунок 7.2 - Регистр управления таймером

GO Разряд GO. Разряд GO сбрасывает и запускает счетчик таймера. Когда GO=1 и таймер не удерживается, счетчик обнуляется и начинает инкрементироваться по следующему возрастающему фронту тактового входа таймера. GO очищается по тому же возрастающему фронту. GO=0 не влияет на таймер.

HLD Разряд удержания счетчика. Когда этот разряд равен нулю, счетчик запрещен и удерживается в его текущем состоянии. Когда таймер управляется TCLK, состояние TCLK тоже удерживается. Внутренний счетчик (с делением на два) тоже удерживается таким образом, что счетчик может продолжиться с состояния останова при установке HLD в 1. Регистры таймера могут считываться и модифицироваться, пока

счетчик удерживается. RESET имеет более высокий приоритет, чем HLD.

GO	HLD	Результат
0	0	Останов всех операций таймера. Сброс не происходит (Значение сброса).
0	1	Таймер обрабатывается из состояния, предшествующего записи.
1	0	Все операции таймера остановлены, включая обнуление счетчика. Разряд GO не очищается до того, как таймер выйдет из состояния удержания.
1	1	Таймер сбрасывается и запускается.

Упражнение 2 Создайте и выполните следующее приложение, демонстрирующее работу таймера.

```

or 1, ПЕ; разрешаем прерывание таймера 0
OR 3800h, ST; разрешаем прерывания и КЭШ
ldpk 10h; устанавливаем DP
;останавливаем все операции таймера
ldi 0, AR0
sti AR0, @20h
;обнуляем счетчик
sti AR0, @24h
ldi 0FFFh, AR0
sti AR0, @28h; установка периода 0FFFh
;запускаем таймер
ldi 3c1h, AR0
sti AR0, @20h

```

Самостоятельно создайте подпрограмму обработки прерываний таймера, которая бы считала число собственных вызовов. Сохраните число вызовов в ячейке памяти.

Контрольные вопросы

1. Назовите основные периферийные устройства TMS320C40.
2. Какое основное назначение контроллера прямого доступа к памяти?

3. Перечислите основные регистры контроллера ПДП, какое их основное назначение?
4. Какие основные действия необходимо выполнить для пересылки данных с помощью контроллера ПДП?
5. Что выполняет контроллер ПДП после завершения передачи данных?
6. Какое основное назначение таймеров TMS320C40?
7. С помощью каких регистров осуществляется управление таймером, какое их основное назначение?

ЛАБОРАТОРНАЯ РАБОТА № 8

Тема: «Программирование ПЦОС TMS320C40 на С»

Цель работы: изучить основные приёмы программирования микропроцессора TMS320C40 на высокоуровневом языке программирования С, научиться составлять, отлаживать и выполнять программы на С.

TMS320C40 С-компилятор полностью реализует ANSI С стандарт. В комплект к компилятору входят восемь библиотек, реализующих различные функции: работа со строками, динамическое выделение памяти, тригонометрические функции и др. Компилятор поддерживает две модели памяти: Big и Small. В первом случае можно использовать весь доступный объем памяти, во втором случае регистр DP загружается один раз, тем самым увеличивается быстродействие. Имеется возможность вызова ассемблерных функций из С программы и наоборот, в соответствии с соглашениями, описанными в руководстве пользователя С компилятора для TMS320C40.

Создание исполняемой программы на С состоит из нескольких этапов:

1. написание исходного текста программы;
2. компиляция С кода (compile);
3. компиляция ассемблерного кода (assemble);
4. компоновка для создания исполняемого объектного файла (link).

Упражнение 1

В качестве примера создадим программу, проверяющую блок RAM1 0x2FFC00 – 0x2FFFFF. Программа записывает значения в память и подсчитывает контрольную сумму. Если сумма верна, то 0x2FFFFF=1, в противном случае - 0x2FFFFF=2.

Для написания исходного кода программы можно использовать любой текстовый редактор. Рекомендуется использовать Code Composer. Создайте в Code Composer новый проект и файл с исходным кодом следующего содержания.

```

/** Файл test.c Исходный код программы */
const int RAMStartAdr = 0x2FFC00; /*Начальный адрес памяти*/
const int RAMlen = 0x400; /*Длина блока памяти*/
const int ControlSum1 = 0x7FFF800; /*Контрольная сумма для AAAA*/
const int ControlSum2 = 0x3FFFC00; /*Контрольная сумма для 5555*/

/* Заполняем значением dd область памяти, начиная с st, длиной len.
Возвращает контрольную сумму*/
int fillmem(int st, int len, int dd){
    int* adr = (int *)st;
    int i;
    int sum=0; /*Сумма записанных данных*/
    for (i=1; i<=len; i++){
        *adr = dd;
        sum += *adr; /*Добавляем записанное значение*/
        *adr++; /*К следующей ячейке*/
    }
    return sum;
}

main()
{
    int sum;

```

```

int* adr = (int *) (0x2FFFFFF); /*Адрес для записи результата*/
sum = fillmem(RAMStartAdr, RAMlen, 0x5555) + fillmem(RAMStartAdr,
RAMlen, 0xAAAA);
if (sum == (0x1555400+0x2AAA800)) *adr = 1; /*Контрольная сумма
верна*/
else *adr = 2; /*Тест пройден с ошибкой*/
}

```

Для компилирования С кода и полученного ассемблерного служит утилита cl30. Для её запуска необходимо в командной строке выполнить:

```
cl30 [-options] [filenames] [-z [link_options] [object files]]
```

Описание опций приведено в руководстве пользователя С компилятора.

Для нашего примера создайте BAT файл со следующим кодом:

```
cl30 test.c -g -v40 -as -ss test
```

PAUSE

test.c – имя файла;

-ss запускает interlist утилиту, позволяющую получить для каждого С файла соответствующий текстовый файл с ассемблерным кодом;

-v40 служит для указания версии процессора;

-as -g опции дебагера.

Результатом исполнения команды является объектный файл test.obj. Исполняемый файл test.out получим с помощью компоновщика (linker), запуск которого производится с помощью утилиты lnk30:

```
lnk30 [-options] filename1 ... filenamen
```

Для нашего примера создайте BAT файл со следующим содержанием:

```
lnk30 test.obj -o test.out c40lnk.cmd
```

c40lnk.cmd – командный файл компоновщика.

Создайте командный файл со следующим содержанием:

```
/* c40lnk.cmd Командный файл*/
```

-c

```
/* LINK USING C CONVENTIONS */
```

```
-stack 0x40          /* STACK */
-heap 0x400          /* 1K HEAP */
-lrts40.lib          /* GET RUN-TIME SUPPORT */
```

```
/* КАРТА ПАМЯТИ */
```

MEMORY

```
{
    ROM: org = 0x000000 len = 0x1000          /* INTERNAL ROM */
    RAM0: org = 0x2FF800 len = 0x400          /* RAM BLOCK 0 */
    RAM1: org = 0x2FFC00 len = 0x400          /* RAM BLOCK 1 */
    LOCAL: org = 0x300000 len = 0x7D00000     /* LOCAL BUS */
    GLOBAL: org = 0x8000000 len = 0x8000000   /* GLOBAL BUS */
}
```

```
/* РАЗМЕЩЕНИЕ СЕКЦИЙ В КАРТЕ ПАМЯТИ */
```

SECTIONS

```
{
    .text: > RAM0          /* CODE */
    .cinit: > RAM0         /* INITIALIZATION TABLES */
    .const: > RAM0         /* CONSTANTS */
    .stack: > RAM0         /* SYSTEM STACK */
    .sysmem: > RAM0        /* DYNAMIC MEMORY (HEAP) */
    .bss: > RAM0, block 0x100 /* VARIABLES */
}
```

С компилятор автоматически создаёт шесть секций:

.bss (неинициализированная) – глобальные и статические переменные;
 .cinit (инициализированная) – значения для инициализируемых
 глобальных и статических констант;

.const (инициализированная) – глобальные и статические;

.stack (неинициализированная) – программный стек;

`.text` (инициализированная) – исполняемый код и вещественный константы;

`.system` (неинициализированная) память для `malloc` функций.

В командном файле необходимо указывать область памяти для размещения этих секций, а также размер `stack` и `heap`. Секции `.bss` и `.const` в модели памяти по умолчанию (`small`) должны быть размером не более 64К слов и не должны пересекать какие-либо 64К адресные границы.

Для компоновки программы написанной на С необходимо всегда указывать опцию `-c` (или `-cr`), а также RUN-TIME библиотеку. При этом линкер устанавливает точкой входа в программу символ `c_int00`. Функция `c_int00` выполняет инициализацию С окружения: определяет стек, инициализирует `SP` и `FP`, инициализирует глобальные переменные и `DP`, вызывает функцию `main` для исполнения С программы.

Скомпилируете, скомпонуйте и выполните программу из упражнения 1.

Самостоятельное задание

Выполните следующие упражнения с массивами на языке С.

Вариант	Задание
1	Дан массив из 20 элементов. Найдите среднее арифметическое его элементов.
2	Дан массив из 20 целых чисел. Подсчитайте, сколько в нём элементов больше заданного.
3	Дан массив из 20 целых чисел. Подсчитайте количество отрицательных элементов в нём.
4	Дан массив из 100 целых чисел. Отсортируйте его по убыванию.
5	Дан массив целых из 100 целых чисел. Проверьте, отсортирован ли массив.
6	Дан массив из 10 целых чисел. Вычислите значение выражения: $y = a[1]x^1 + a[2]x^2 + \dots + a[10]x^{10}$
7	Дан массив из 20 целых чисел. Проверьте, образуют ли его элементы арифметическую прогрессию.
8	Дан массив из 20 целых чисел. Поменяйте в нём местами максимальный и минимальный элементы.

Контрольные вопросы

1. Какие модели памяти поддерживает С компилятор, чем они различаются?
2. Из каких основных этапов состоит создание программы на языке С для TMS320C40?
3. Какая утилита используется для компилирования С кода?
4. Какие особенности имеет командный файл компоновщика для программы, написанной на С?
5. Какие секции автоматически создаёт С компилятор, какое их назначение?
6. Какая функция инициализирует С окружение? Назовите основные действия, которые она выполняет.

Библиографический список

Основная литература

1. Кузин, А. В. Микропроцессорная техника [Текст] : доп. М-вом образования Рос. Федерации в качестве учеб. для студентов образоват. учреждений сред. проф. образования, обучающихся по группам специальностей 2200 "Информатика и выч. техника", 1800 "Электротехника" / Александр Владимирович, Михаил Анатольевич ; А. В. Кузин, М. А. Жаворонков. - 3-е изд., стер. - М. : Академия, 2007. - 304 с.
2. Александров, Е.А. Микропроцессорные системы [Текст] : доп. М-вом образования Рос. Федерации в качестве учеб. пособия для студентов высш. учеб. заведений, обучающихся по направлению подготовки бакалавров и магистров "Информ. и выч. техника" / Е. А. Александров, Р. И. Грушвицкий, М. С. Куприянов, О. Е. Мартынов, Д. И. Панфилов, Т. В. Ремизевич [и др.]; под общ. ред. Д. В. Пузанкова. - СПб. : Политехника, 2002. - 934,[2] с.

Дополнительная литература

1. Зольников, В.К. Программирование и основы алгоритмизации [Текст]: учебное пособие / В.К.Зольников, В.И.Анциферова, Н.Н.Литвинов, П.Р.Машевич – 2007 – Воронеж ВГЛТА – 326 с.

2. Мясников, В.А. Микропроцессоры : системы программирования и отладки [Текст] / В. А. Мясников, М. Б. Игнатьев, А. А. Кочкин, Ю. Е. Шейнин [и др.]; под ред. В. А. Мясникова, М. Б. Игнатьева. - М. : Энегроиздат, 1985. - 272 с. Корнеев, В.В. Современные микропроцессоры [Текст]:/ В.В. Корнеев, А.В. Киселев. – 2003 - СПб.: БХВ-Петербург – 445 с.

3. Микропроцессоры и микропроцессорные комплекты интегральных микросхем : справ. : в 2 т. Т. 1 / под ред. В. А. Шахнова. - 1988. - 368 с.

4. Кочетов, В. И. Микропроцессоры и микроЭВМ [Текст] : метод. указания к лаб. работе для студентов 3 курса специальности 210214 - "Автоматизация технол. процессов и производств ЛПК" / Владимир Иванович, Марина Александровна ; В. И. Кочетов, М. А. Кривотулова; Воронеж. гос. лесотехн. акад. - Воронеж, 1998. - 8 с.