Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования
«Севастопольский государственный университет»

ИССЛЕДОВАНИЕ АРХИТЕКТУРЫ И СИСТЕМЫ КОМАНД 16-РАЗРЯДНОГО ПРОЦЕССОРА

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

к лабораторной работе по дисциплине

«Технические средства информационных систем»
для студентов, обучающихся по направлению

09.03.02 Информационные системы и технологии

очной и заочной форм обучения

Севастополь
2022

УДК 004.732

Исследование архитектуры и системы команд 16-разрядного процессора. Методические указания к лабораторным занятиям по дисциплине «Технические средства информационных систем» / Сост. Чернега В.С., Дрозин А.Ю. — Севастополь: Изд-во СевГУ, 2022— 21 с.

Методические указания предназначены для проведения лабораторных работ по дисциплине «Технические средства информационных систем». Целью методических указаний является помощь студентам в выполнении лабораторных работ по исследованию архитектуры 16-разрядных процессоров и персональных ЭВМ, а также по программированию различных задач на языке ассемблера процессора Intel 8086. Излагаются краткие теоретические и практические сведения, необходимые для выполнения лабораторных работ, примеры составления программ, требования к содержанию отчета.

Методические указания рассмотрены и утверждены на методическом семинаре и заседании кафедры информационных систем
(протокол № 1 от 30 августа 2022 г.)

Допущено учебно-методическим центром СевГУ в качестве методических указаний.

Рецензент: Кротов К.В., канд. техн. наук, доцент кафедры ИС

Лабораторная работа

Исследование архитектуры и системы команд 16-разрядных процессоров

1. Цель работы

Исследовать систему команд, архитектуру и основные блоки процессора Intel 8086 и взаимодействие этих блоков процессора при выполнении команд разных типов. Приобрести практические навыки написания ассемблерных программ и отладки их в эмуляторе микропроцессора — экранным отладчиком типа emu8086.

2. Основные теоретические положения

Основной архитектурной особенностью 16-разрядного процессора Intel 8086 (отечественный аналог К1810ВМ86) является 16-разрядная шина данных (ШД), 20-разрядная шина адреса, мультиплексированная с ШД, 16-разрядное АЛУ и 16-разрядные внутренние регистры. Для повышения быстродействия процессор 8086 логически разделен на два независимых блока, работающих параллельно: блок сопряжения с системной шиной BIU (*Bus Interface Unit*) и исполнительный EU (*Execution Unit*). Блок сопряжения считывает коды команд и операндов и сохраняет их в 6-байтовом конвейере команд, а исполнительный блок выбирает команды из конвейера, не дожидаясь, пока BIU доставит очередную команду.

Память в микро-ЭВМ на базе МП Intel 8086 логически организована как одномерный массив *байтов*, каждый из которых имеет адрес в диапазоне 0000 – FFFFFh. Любые два смежных байта могут рассматриваться как 16-битовое слово. Младший байт имеет меньший адрес, старший – больший. *Адресом слова считается адрес его младшего байта.*

В процессорах семейства Intel x86 применяется сегментная адресация памяти, при *которой* все пространство адресов делится на множество сегментов, т.е. пространство является сегментированным. Пространство памяти емкостью 1 Мбайт представляется как набор сегментов, определяемых программным путем. Сегмент состоит из смежных ячеек памяти и является независимой и отдельно адресуемой единицей памяти емкостью 64 Кбайт. Каждому сегменту системной программой назначается начальный (базовый) адрес, являющийся адресом первого байта сегмента в пространстве памяти. В зависимости от вида хранимой информации различают три типа сегментов: для хранения кодов команд используется сегмент кода CS, в качестве стековой памяти используется сегмент стека SS, а для хранения данных служат сегменты DS и ES. Начальные адреса четырех сегментов, выбранных в качестве текущих, записываются в сегментные регистры CS, DS, SS, ES. Для обращения к команде и данным, находящимся в других сегментах, необходимо изменить содержимое сегментных регистров, что позволяет использовать все пространство памяти емкостью 1 Мбайт. Для хранения смещения в сегменте

используются специальные регистры, выбор которых зависит от типа обращения к памяти (таблица 2.1).

Таблица 2.1 – Источники логического адреса

Тип обращения к памяти	Сегмент (по умолчанию)	Вариант	Смещение
Выборка команд	CS	нет	IP
Стековые операции	SS	нет	SP
Переменная	DS	CS, SS, ES	EA
Цепочка-источник	DS	CS, SS, ES	SI
Цепочка-приемник	ES	нет	DI

Команды всегда выбираются из текущего сегмента кода в соответствии с логическим адресом CS:IP. Стековые команды всегда обращаются к текущему сегменту стека по адресу SS:SP. Если при вычислении эффективного адреса EA используется регистр BP, то обращение производится также к стековому сегменту, а ячейки стекового сегмента рассматриваются как ОЗУ с произвольной выборкой. Операнды, как правило, размещаются в текущем сегменте данных, и обращение к ним организуется по адресу DS:EA. Однако программист может заставить МП обратиться к переменной, находящейся в другом текущем сегменте.

Значения *базы сегмента* и *смещения в сегменте представляют собой л*огический адрес. Базовый адрес и смещение в сегменте отображаются 16-разрядными числами. При обращении к памяти BIU на основании логического адреса формирует физический адрес следующим образом: значение базы сегмента смещается на четыре разряда влево, и полученное 20-разрядное число (с четырьмя нулями в младших четырех разрядах) складываются со значением смещения в сегменте. Таким образом, база сегмента (с четырьмя нулями, добавленными в качестве младших разрядов) задает для памяти сегменты длинной 64 Кбайт, а значение сегмента в смещении – расстояние от начала сегмента до искомого адреса памяти. Максимально возможное смещение в сегменте равно 64 Кбайт.

МП содержит три группы регистров. К *первой группе* относятся РОН, используемые для хранения промежуточных результатов. Ко *второй* группе относятся *указатели и индексные регистры*, предназначенные для размещения или извлечения данных из выбранного сегмента памяти. Содержание этих регистров определяет значение смещения в сегменте при задании логического адреса. К *третьей* группе относятся *регистры сегментов*, задающие начальные адреса (базы) самих сегментов памяти. МП имеет регистр флаговых разрядов, используемых для указания состояния АЛУ при выполнении различных команд и управления его работой.

Таким образом в МП располагается тринадцать 16-разрядных регистров и девять флаговых разрядов АЛУ. Последний, тринадцатый регистр,

называется указателем команд *IP (Instruction Pointer*). Он выполняет функции, аналогичные функциям программного счетчика в МП 8080 и не является программно доступным регистром. Доступ к его содержимому может быть осуществлен с помощью команд передачи управления.

Группа РОН включает в себя семь 8-разрядных регистров МП 8080 и один добавленный 8-разрядный регистр для того, чтобы объединить все регистры в четыре 16-разрядные пары. К ним может быть организован программный доступ как к 8- или 16-разрядным регистрам. 16-разрядные регистры обозначаются символами AX, BX, CX и т.д. При обращении к ним, как к 8-разрядным регистрам, их обозначают AL, AH, BL, BH и т.д. Все эти регистры могут быть использованы при выполнении арифметических и логических команд. Однако есть команды, использующие определенные регистры для специфических целей, тогда применяют мнемонические обозначения: аккумулятор (*accumulator*), база (*base*), счет (*count*), данные (*data*).

Группа указателей и индексных регистров состоит из четырех 16-разрядных регистров: регистры указатели SP – Stack Pointer и BP – Base Pointer; индексные регистры SI – Source index и DI – Destination index. Обычно эти регистры содержат информацию о смещении по адресам в выбранном сегменте и позволяют компактно писать программы каждый раз непосредственно, не приводя используемого адреса. С их помощью производится вычисление адресов программ. Чаще всего в регистрах *указателях* записано адресное смещение по отношению к *стековому* сегменту, а в *индексных* регистрах – адресное смещение по отношению к *сегменту данных*.

Регистр состояния (Флаговый регистр) содержит шестнадцать триггеров, из которых используется только 9. Эти триггеры отображают состояние процессора при выполнении последней арифметической или логической команды. Пять триггеров аналогичны МП 580-й серии. Дополнительные триггеры сигнализируют: OVERFLOW – переполнение; DIRECTION – направление – указывает направление работы с массивами (автоматическое увеличение или уменьшение на единицу адреса массива); INTERRUPT – прерывание – определяет для МП реагирования на прерывание; TRAP – (западня, ловушка) устанавливает для МП пошаговый режим.

Система команд ВМ86 содержит мнемокоманды, позволяющие совершать операции над байтами, двухбайтовыми символами, отдельными битами, а также цепочками (строками) байтов и слов. По функциональному признаку система команд МП 8086 разбивается на 6 групп: пересылка данных; арифметические операции; логические операции и сдвиги; передача управления; обработка цепочек; управление процессором. В таблице 2.2 приведена мнемоника почти всех команд МП 8086. Здесь используются следующие обозначения:

dst – регистр-получатель, src – регистр-источник;

mem – адрес памяти (смещение), заданный любым методом адресации;

r1,r2 – регистры общего назначения;

seg – сегментные регистры;

data – непосредственные данные.

Таблица 2.2 – Команды Ассемблера микропроцессора 8086

Команды микропроцессора 1810ВМ86/8086			
Пересылки данных	Арифметические и логические		Сдвига и передачи управления
Mov r,r	Add r1,r2	Cmp r,mem	shl/sal r
Mov r,mem	Add r,mem	Cmp mem,r	shl/sal mem
Mov mem,r	Add mem,r	Cmp r,data	shr r
Mov mem,data	Add r,data	Mul src	sar r
Mov r,data	Add mem,data	Imul src	sar mem
Mov seg,r	Adc r,mem	Div src	
Mov r,seg	Adc mem,r	Idiv src	Rol r
Mov seg,mem	Adc r,data	Daa	Rol mem
Movs dst,src	Adc mem, data	Das	Ror r
Push r	Inc r	Aaa	Ror mem
Push mem	Inc mem	Cbw	Rcl r
Pop r	Sub r1,r2	Cwd	Rcl mem
Pop mem	sub r,mem		Rcr r
Pushf	sub mem,r	And r1,r2	Rcr mem
Popf	sub r,data	And r,mem	
Push seg	sub mem, data	And r,data	Jmp address
Pop seg	Sbb r1,r2	Or r1,r2	Jnz label1
Xchg r,mem	sbb r,mem	Xor r1,r2	Jz label2
Xchg r,r	sbb mem,r	Xor r,mem	Jz address
In port	sbb r,data	Xor r,data	Jnz label3
Out port	sbb mem, data		Jcxz address
In[dx]	Dec r	Test r1,r2	Je label
Out[dx]	Dec mem	Test r,mem	
Lea r	Neg r	Test r,data	
Lds r,mem	Neg mem	Test mem,data	
Lahf	Cmp r1,r2	Not r	
sahf		Not mem	

Подробное описание типов операндов, форматов данных, команд и примеры их использования можно найти во вкладке help системы emu8086, используемой в качестве лабораторной установки.

3. Описание лабораторной установки

Лабораторный стенд для исследования архитектуры и способов программирования на языке ассемблера 16-разрядных микропроцессоров состоит из персонального компьютера, на котором установлен программный эмулятор 16-разрядного микропроцессора типа Intel 8086 (отечественный аналог МП КР1810). Эмулятор отображает на экране персонального компьютера программную модель исследуемого процессора, т.е. регистры общего назначения, регистр флагов, содержимое оперативной памяти. Эмулятор также позволяет создавать и редактировать тексты программ на языке ассемблера МП 8086, выполнять их ассемблирование и исследование процессов модификации регистров процессора, дампов памяти и портов в пошаговом и реальном режимах отладки программ.

3.1. Особенности работы с экранным отладчиком emu8086

При запуске программы появляется окно приглашения (рисунок 3.1), в котором пользователю предлагается выбрать один из вариантов работы: создание нового файла; просмотр примеров программирования; запуск уже использовавшихся файлов или переход в режим помощи.

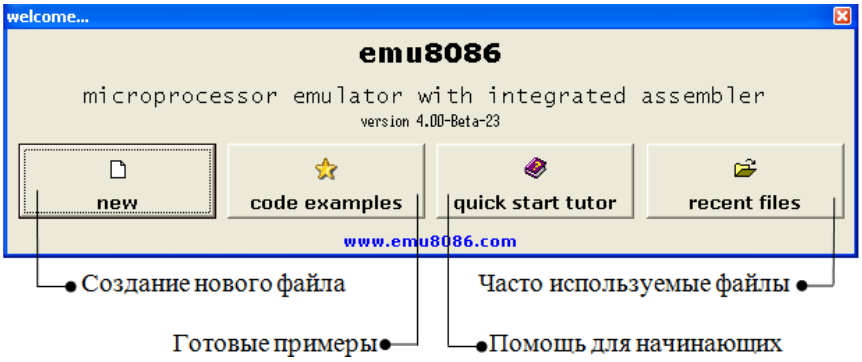


Рисунок 3.1 — Окно приглашения

В эмуляторе имеется довольно обширный учебник по языку программирования Ассемблер для процессора семейства x8086 с наличием множества примеров использования ассемблерных команд. Для получения доступа к этому учебному пособию следует нажать клавишу quick start tutor. Недостатком учебника является то, что он написан на английском языке.

При создании нового файла (рисунок 3.2) существует возможность выбрать шаблон файла.

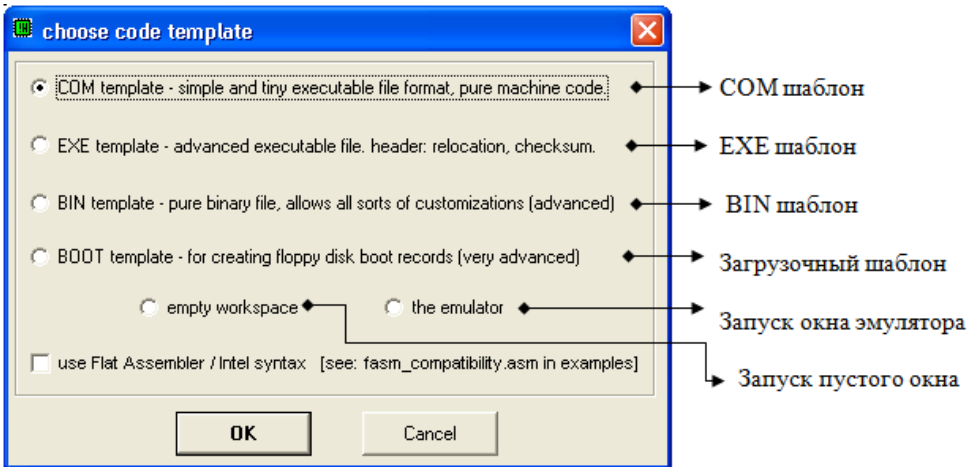


Рисунок 3.2 — Окно выбора шаблона

Например, при выборе варианта шаблона COM-файла, появится окно редактора с соответствующим стандартным кодом (рисунок 3.3). Окно редактора можно разделить на область панели инструментов и рабочую область, используемую для написания и редактирования ассемблерных программ.

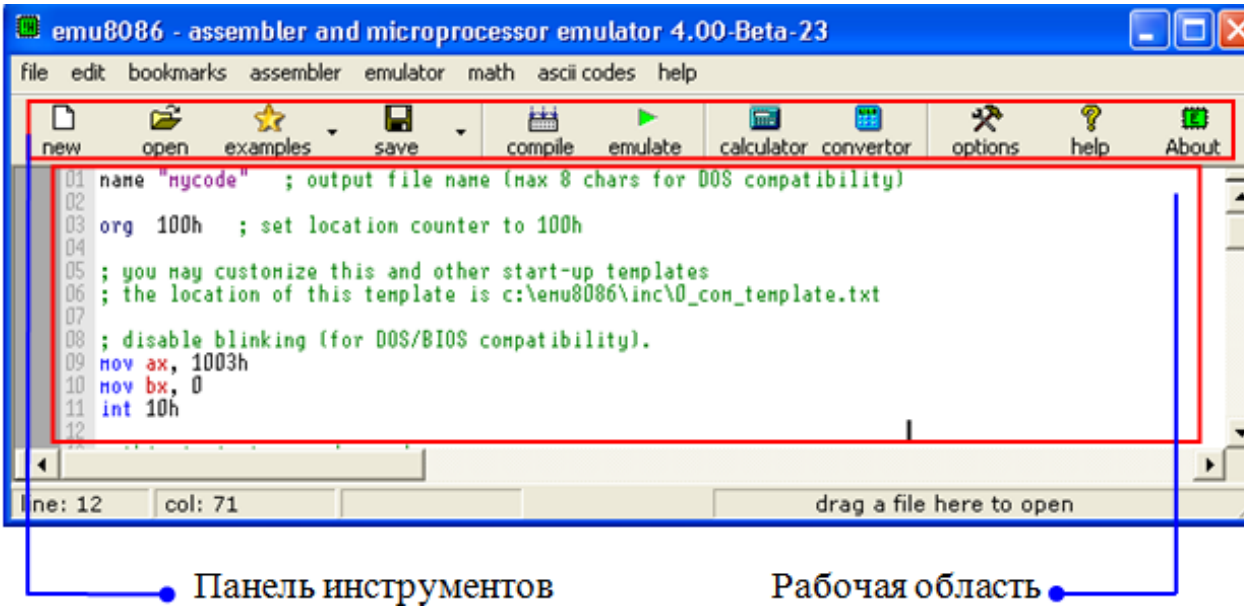


Рисунок 3.3 — Окно создания и редактирования кода

Панель инструментов позволяет осуществить быстрый доступ к самым часто используемым функциям. Пункты меню панели инструментов имеет следующее назначение:

new — Создание нового файла. Создание и ввод текста программы на ассемблере для последующего выполнения. Доступно использование комментариев – перед текстом комментария необходимо ставить “;”.

open — Открытие уже созданного и сохраненного файла для его редактирования или запуска.

examples — Открытие примера. Открывает один из доступных файлов, созданных разработчиками для показа возможностей функций ассемблера.

save — Сохранение на носитель информации файла, который вы создали или отредактировали.

compile —Создание исполняемого файла. Создание файла, который не будет зависеть от эмулятора и будет запускаться из операционной системы без дополнительных программ (*.exe, *.com).

emulate — Запуск программы в режиме эмуляции, в котором возможно отследить за каждым шагом выполнения программы, за содержимым регистров, флагов, ячеек памяти. Используется при проверке на работоспособность программы и при необходимости отыскать ошибки в коде.

calculator — Запуск встроенного калькулятора, который позволяет производить расчеты над числами в двоичной, восьмеричной, десятеричной, шестнадцатеричной формах. Результат можно выводить так же в любой удобной форме. Вид калькулятора представлен на рисунке 3.4.

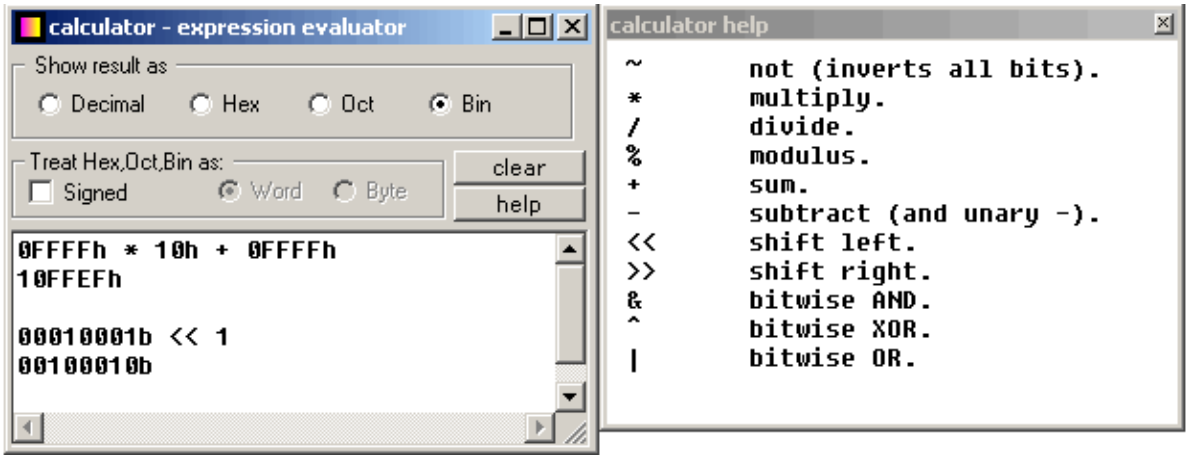


Рисунок 3.4 — Окно калькулятора и окно помощи для работы с ним

convertor — Запуск конвертера систем счисления (двоичная, восьмеричная, десятичная, …). Конвертер основ счисления позволяет легко и удобно переводить числа из одной основы в другую, т.е. осуществлять преобразования типа: bin->hex, dec->bin и т.п. Вид конвертера основ счисления показан на рисунке 3.5.

options — Настройки эмулятора. Позволяют настроить цвет и шрифт кода, комментариев, выделения, фона, панелей и других компонентов эмулятора.

help — Вызов помощи (на англ. языке). Полное описание на английском языке функций работы с ассемблером, прерываниями, циклами и т.д.

About — Информация о создателях. Регистрация продукта, информация о «координатах» создателей, версии и дате создания программы.

Рисунок 3.5 — Окно конвертера

3.2. Работа отладчика в режиме эмуляции

Общий вид окна экранного отладчика показан на рисунке 3.6. В левой части окна изображены программно доступные регистры микропроцессора 8086 и значение их содержимого в 16-ричной системе счисления.

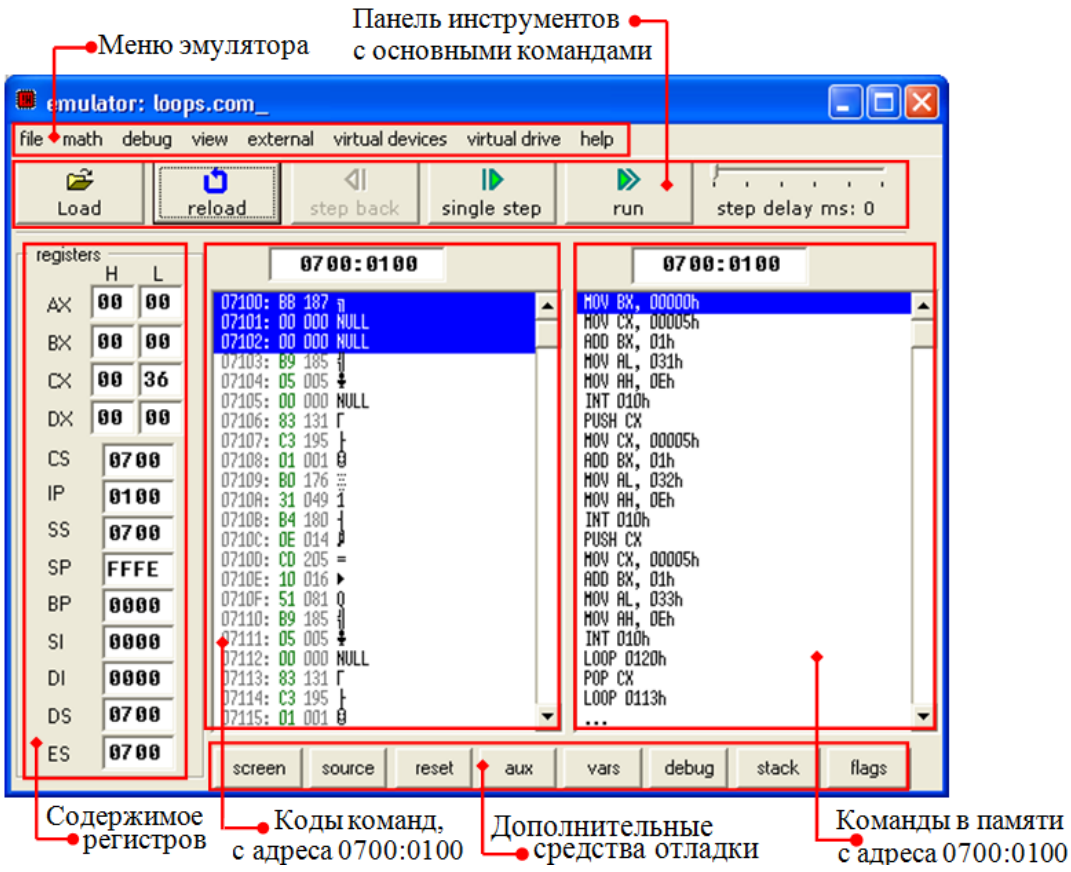


Рисунок 3.6 — Главное окно эмулятора

В правой части окна выведен текст исследуемой программы в мнемонических кодах, а в центре отображается этот же фрагмент в машинных кодах (в 16-ричной системе счисления). Как видно из рисунка, программа размещена в сегменте с базовым адресом 0700h и начальном смещении 0100h.

Двойным щелчком по любому адресу или содержимому регистров вызывается расширенный просмотр значений (рисунок 3.7).

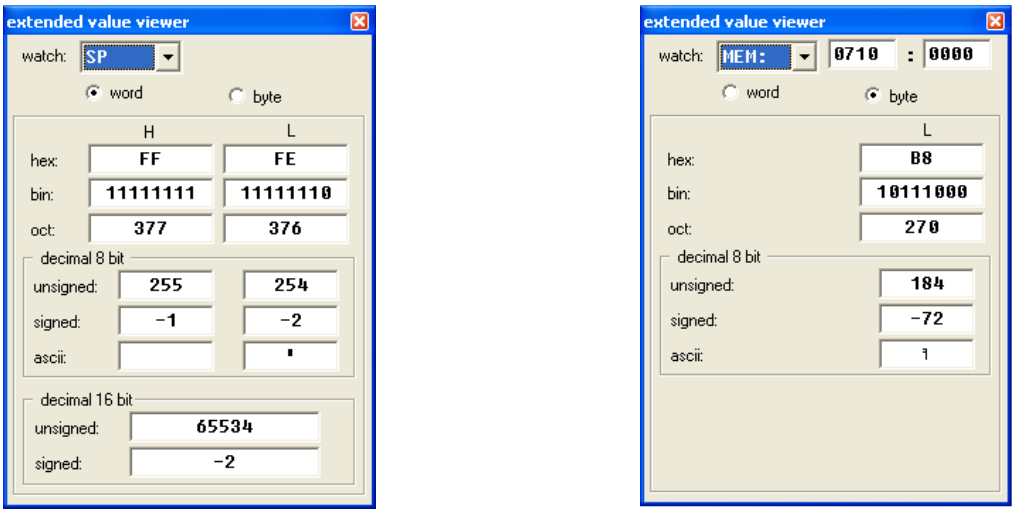


Рисунок 3.7 — Пример окон просмотра значений регистров

На панели инструментов данного окна имеются следующие пункты:

- Load —загрузка существующего файла для его эмуляции.
- Reload — выполнение программы с самого её начала.
- Step back — шаг назад на одну операцию при пошаговом режиме.
- Single step — шаг вперёд на одну операцию при пошаговом режиме.
- Run — эмуляция без остановок между операциями.

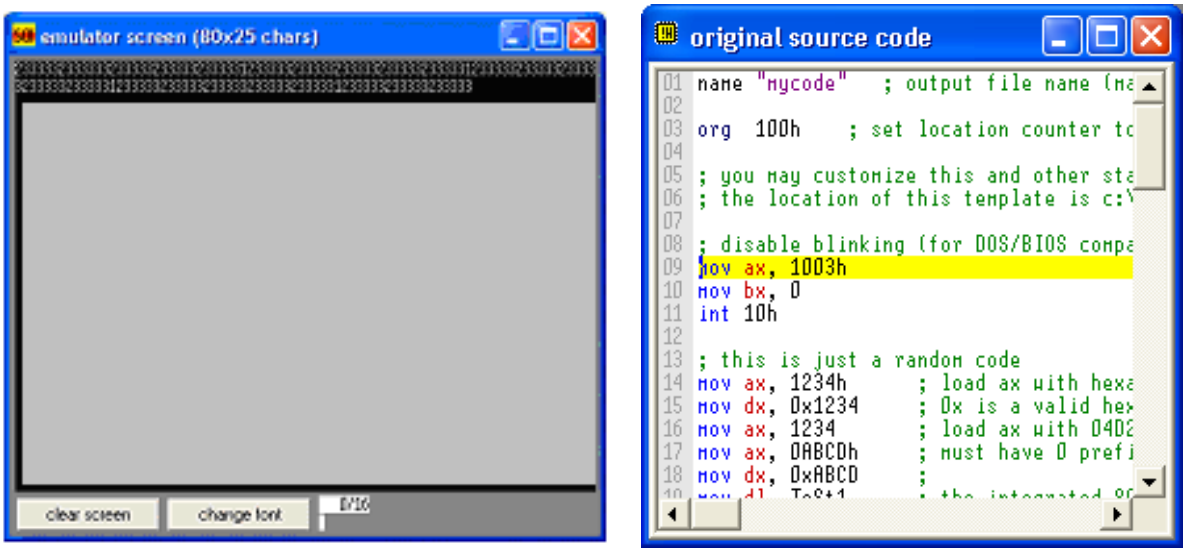
Окно эмулятора (строка внизу) также предоставляет дополнительные возможности для отладки программ, в частности:

screen — Вызов эмулируемого экрана. Если в программе используется работа с экраном (монитором), то выполнение этих операций отображается на эмулируемом экране, показанного на рисунке 3.8,*а*.

source — Вызов окна просмотра оригинального кода. Для ориентации в своей программе и в кодах можно использовать окно, содержащее её оригинальный текст. Пример такого окна показан на рисунке 3.8,б.

reset — Сброс всех значений регистров. Тем самым осуществляется выполнение программы с начала. При этом отчищается эмулируемый экран.

aux — Пункт меню предоставляет дополнительные возможности для просмотра памяти, АЛУ, сопроцессора и др.



а) б)

Рисунок 3.8 — Окно, эмулирующее вывод данных на экран монитора (а) и окно просмотра исходного кода программы (б)

Окно памяти программ и данных **memory**. Используется для удобного просмотра содержимого памяти с произвольным доступом (рисунок 3.9)

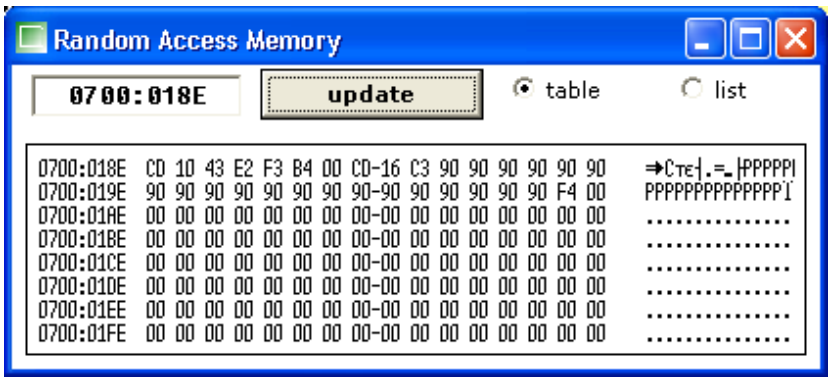


Рисунок 3.9 — Окно просмотра содержимого памяти

В данном окне можно просматривать последовательность располагаемых в памяти данных и перемещаться по памяти, вводя значение адреса вместо показанного на рисунке 0700:018E. Можно выбрать способ просмотра в виде таблицы или списка путём выбора table или list соответственно.

Окно ALU — отображает содержимое арифметико-логического устройства. На рисунке 3.10 показан пример работы АЛУ при сложении: первая строка указывает номер разряда, две следующие строки – это операнды, а последняя – результат.

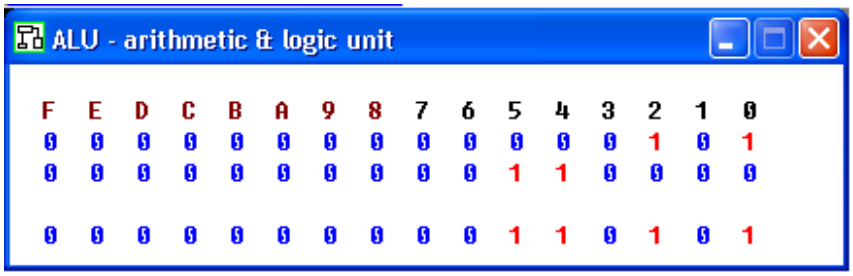


Рисунок 3.10— Окно просмотра содержимого ALU

В окне FPU — отображается содержимое регистров математического сопроцессора.

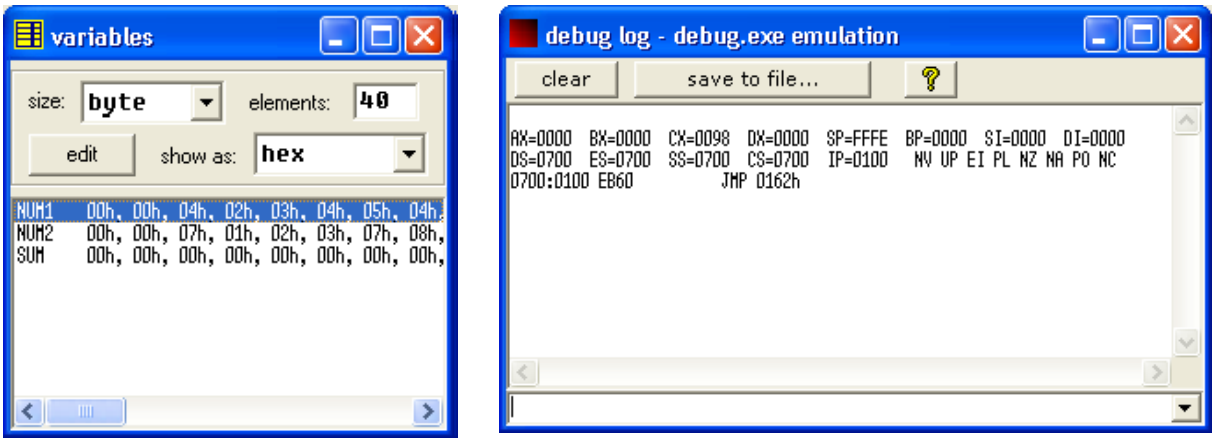
Окно stop on condition —необходимо использовать, если при отладке требуется остановить эмуляцию при каком-то условии значения регистров общего назначения, флагов или ячейки памяти.

Таблица символов **symbol table**. В этой таблице отображаются названия и параметры символов (рисунок 3.11,*а*).

В окне **listing** — отображается программа в машинных кодах. Вся эмулируемая программа представляется в виде кодов команд и адресов, по которым они располагаются.

var — Просмотр переменных. Можно осуществить быстрый и прямой доступ ко всем переменным. В случае необходимости можно изменить их значение двойным щелчком мыши по необходимой переменной.

Журнал отладки **debag** — сохраняет каждое изменение в ходе выполнения программы (рисунок 3.11,*б*).



а) б)

Рисунок 3.11 — Окно просмотра переменных а) и окно журнала отладки б)

Окно stack — оказывает возможность просмотреть содержимое стека или его изменения.

Окно состояние флагов **flags**. Позволяет просмотреть состояние всех флагов и при необходимости изменить их. При нажатии на кнопку “analyse”, появляется окно “lexical flag analyser” (рисунок 3.12), в котором расписывается значение каждого флага процессора.

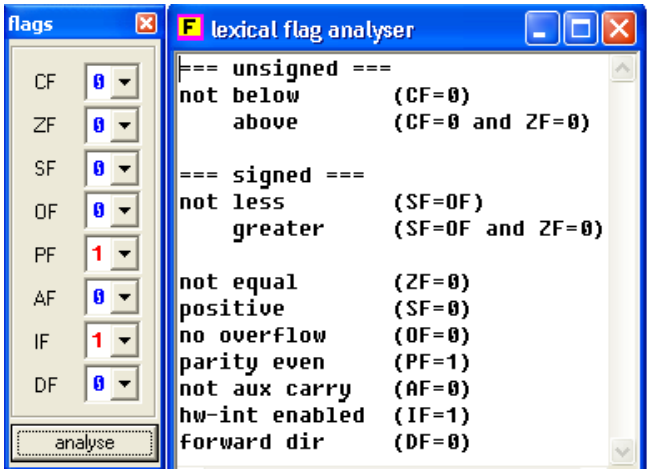


Рисунок 3.12 − Окно состояние флагов

4. Программа работы

3.1. Изучить архитектуру МП 8086, состав регистров и работу процессора с использованием временных диаграмм (выполняется в процессе домашней подготовки к лабораторной работе).

3.2. Изучить основные директивы ассемблера и команды МП 8086 (выполняется в процессе домашней подготовки к лабораторной работе).

3.3. Изучить функции BIOS и DOS и особенности использования их в ассемблерных программах.

3.4. Запустить эмулятор, выбрать шаблон в формате com и ввести в окне редактора пробную программу, приведенную в приложении А.

3.5. Исследовать процесс выполнения программы (т.е. проследить изменение содержимого регистров, оперативной памяти и стека).

3.6. Пояснить в форме комментариев к каждой ассемблерной строке исследуемой программы.

3.7. Запустить эмулятор, выбрать шаблон в формате com и ввести в окне редактора пробную программу, приведенную в приложении Б.

3.8. Исследовать и пояснить изменения регистров при выполнении каждой из команд.

3.9. Рассчитать время выполнения программ.

ПРИМЕЧАНИЕ: В этих программах используются прерывания DOS и BIOS, а сами программы представлены в форматах com или exe. Поэтому, прежде чем начать исследование примеров ассемблерных программ, внима-

тельно ознакомьтесь с методическими указаниями по выполнению данной работы.

5. Методические указания по выполнению работы

Исполняемые модули ассемблерных программ могут быть оформлены в виде exe- com-файлов. Эти файлы, кроме собственно исполняемого кода приложения, должны содержать информацию для операционной системы (DOS) о размещении программы в памяти компьютера, а также команды. которые обеспечат возврат в операционную систему после завершения приложения. Отличие форматов состоит в следующем.

Размер программы. Программа в EXE может иметь любой размер, в то время как COM-файл ограничен размером одного сегмента (≤64к). Файлы типа COM содержат только скомпилированный код без какой-либо дополнительной информации о програме. Размер COM-файла всегда меньше, чем размер соответствующего EXE-файла, так как в нем отсутствует заголовок (512 байт) EXE-файла. В заголовке EXE-файла описывается размер файла, требуемый объем памяти, список команд в программе, использующих абсолютные адреса, которые зависят от расположения программы в памяти, и т.д. Заголовок хранится на диске в начале EXE-файла.

Сегмент стека. В EXE программисту следует задавать сегмент стека, в то время как COM-программа генерирует стек автоматически.

Сегмент данных. В EXE-программах обычно определяется сегмент данных, а регистр DS инициализируется адресом этого сегмента. В COM-программах все данные должны быть определены в сегменте кода.

Инициализация. В EXE-программе следует записать 0-слово в стек и инициализировать регистр DS. Так как в COM стек и сегмент данных не определены, то эти шаги отсутствуют.

При запуске COM-программы DOS заносит в сегментные регистры адрес префикса программного сегмента (256 байт=100H), который резервируется DOS непосредственно перед COM- или EXE –программой в памяти. Так как адресация начинается со смещения 100H от начала префикса, то в программе после директивы SEGMENT следует вносить директиву ORG 100H. Для преобразования EXE-файла в COM-файл используется программа EXE2BIN.

Для облегчения работы программиста разработан ряд стандартных служебных системных программ взаимодействия с клавиатурой, дисплеем и др. устройствами ввода/вывода данных. Эти программы (функции) входят в состав операционной системы (функции DOS) и функции BIOS ─ базовой системы ввода/вывода (Basic Input/Output System). Вызов этих функций осуществляется путем команд прерывания INT.

Все функции DOS вызываются с помощью прерывания 21h (в десятичной нотации 33). Выбор конкретной функции осуществляется путем записи соответствующего номера в регистр **AH**.

Функция 02: вывод одного символа на экран дисплея

Для вывода одного символа на экран ПК используется функция 02 прерывания 21h:

```
mov DL, <код выводимого символа>

mov AH, 2

int 21h
```

Выводимый символ высвечивается в позиции курсора (что бы там ни было записано), после чего курсор сдвигается на одну позицию вправо.

Функция 9: вывод строки на экран дисплея

Для вывода на экран строки (последовательности символов) можно, конечно, использовать функцию 02, однако сделать это можно и за один прием с помощью функции 09 прерывания 21h:

```
mov DX, offset String_name

mov AH, 9

int 21h

. . . . . .

String_name db ’Privet! $’
```

Перед обращением к этой функции в регистр DS должен быть помещен номер того сегмента памяти, в котором находится выводимая строка, а в регистр DX ─ смещение строки внутри этого сегмента. При этом в конце строки должен находиться символ $ (код 24h), который служит признаком конца строки и сам не выводится.

Функция 4Ch: завершение программы

Завершив все свои действия, прикладная программа обязана вернуть управление операционной системе, чтобы пользователь мог продолжить работу на ПК. Такой возврат реализуется функцией 4Ch прерывания 21h, которую помещают в конце программы:

```
mov AL, <код завершения>

mov AH, 4Ch

int 21h
```

Каждая программа обязана сообщить, успешно или нет она завершила свою работу. Так как любая программа вызывается из какой-то другой программы (например, из операционной системы), и иногда вызвавшей программе, чтобы правильно продолжить работу, надо знать, выполнила ли вызванная программа задачу верно, или она проработала с ошибкой. Такая ин-

формация передается в виде кода завершения программы (некоторого целого числа), который должен быть нулевым, если программа проработала правильно, и ненулевым (каким именно - оговаривается в каждом случае особо) в противном случае. (Узнать код завершения вызванной программы можно с помощью функции 4Dh прерывания 21h.) Потребуется этот код или нет, программа все равно должна выдать его.

Для работы с клавиатурой используются функции BIOS, вызываемые прерыванием INT 16h. Они включает в себя функции для выборки кода нажатого символа из буфера с ожиданием нажатия, функции для проверки содержимого буфера и для управления содержимым буфера, функции для изменения скоростных характеристик клавиатуры.

Функция 00h выполняет чтение кода символа из буфера клавиатуры, если он там есть. Если *буфер клавиатуры пуст*, программа переводится в состояние *ожидания* до тех пор, *пока не будет нажата какая-нибудь клавиша*. Скан-код и ASCII-код нажатой клавиши передаются программе.

6. Содержание отчета

5.1. Цель и программа работы.
5.2. Структурная схема МП 8086 и временные диаграммы его функционирования.
5.3. Тексты исследуемых программ с построчными комментариями.
5.4. Результаты проведенных исследований и расчетов.
5.5. Выводы по работе с анализом результатов выполненных исследований и расчетов.

7. Контрольные вопросы

7.1. Расскажите о составе и назначении основных блоков процессора Intel 8086.
7.2. Поясните, за счет чего повышено быстродействие процессора 8086 по сравнению с его предшественником.
7.3. Объясните понятие машинного цикла, перечислите виды машинных циклов МП 8086 и поясните, какие сигналы и в какой последовательности появляются на выводах процессора в каждом из циклов.
7.4. Перечислите основные внешние выходы МП КР1810, расскажите об их назначении.
7.5. Расскажите о флагах процессора и особенностях их использования.
7.6. В чем состоит отличие логического и физического адресов и как формируется физический адрес?

7.7. Расскажите о командах сдвига и с какой целью они используются в ассемблерных программах.
7.8. Как осуществляется инициализация сегментных регистров?
7.9. Что представляет собой стек и где он размещается в программах форматов com и exe?
7.10. Какова роль указателя стека в организации выполнения программы и каково его значение при выполнении первой команде Push?
7.11. Поясните целесообразность включение в состав процессора индексных регистров и приведите пример программы с их использованием.
7.12. Расскажите подробно о работе процессора после включения питания.
7.13. В чем состоит особенность работы процессора при поступлении сигнала прерывания от внешнего устройства?
7.14. Для чего используются внутренние прерывания DOS и BIOS?
7.15. В чем состоит отличие работы процессора в минимальном и максимальном режимах?
7.16. Расскажите об основных возможностях экранного отладчика emu8086.
7.17. Расскажите о режимах исполнения отдельных команд и целых программ в экранном отладчике emu8086.
7.18. Прокомментируйте результат действия каждой из команд в программе сложение-вычитание операндов.
7.19. Опишите возможности взаимодействия микропроцессора с внешними устройствами, реализованные в экранном отладчике emu8086.

Список рекомендованной литературы

1. Абель П. Язык Ассемблера для IBM PC и программирования / Пер. с англ. Ю.В.Сальникова. — М.: Высш. школа, 1992. — 447 с.Новиков Ю.В. Основы микропроцессорной техники: Учебное пособие/Ю.В. Новиков, П.К. Скоробогатов. — М.: Интернет-университет информационных технологий; БИНОМ, 2006. — 359 с.

2. Макуха В.К. Микропроцессорные системы и персональные компьютеры: учебник для вузов/В.к. Макуха, В.А. Микерин. – М.: Изд-во Юрайт, 2022. – 156 с. https://www.urait.ru/book/mikroprocessornye-sistemy-i-personalnye-kompyutery-492153

3. Новожилов О. П. Архитектура ЭВМ и систем в 2 ч. Часть 1: учебное пособие для вузов / О. П. Новожилов. — Москва: Издательство Юрайт, 2023. — 276 с. — (Высшее образование). — ISBN 978-5-534-07717-9. — Текст : электронный // Образовательная платформа Юрайт [сайт]. — URL: https://urait.ru/bcode/516640 (дата обращения: 12.08.2023).

4. Новожилов О. П. Архитектура ЭВМ и систем в 2 ч. Часть 2 : учебное пособие для вузов / О. П. Новожилов. — Москва: Издательство Юрайт, 2023. — 246 с. — (Высшее образование). — ISBN 978-5-534-07718-6. — Текст : электронный // Образовательная платформа Юрайт [сайт]. — URL: https://urait.ru/bcode/516641 (дата обращения: 12.08.2023).

5. Чернега В.С. Технические средства информационных систем. Конспект лекций / В.С. Чернега. – Севастополь: Изд-во СевГУ, 2022 – 160 с.

Приложение А

```
mov ax,0255
inc ax
add ax, alpha
nop
mov bx,ax
dec bx
sub bx, beta
mov dx, bx
sub dx,10
xchg ax,dx
push bx
push ax
pop cx
mov si,cx
mov di,dx
mov 0150h,cx
shl ax,2
mov dx, offset hello
mov ax,0900h
int 21h
mov ax,4c00h
int 21h
ret

alpha dw 25
beta dw 32
hello db 'Privet kafedra IS!$'
```

Приложение Б

Программа печатает переменную размером 16-бит в двоичном виде. Используется команда сдвига влево на 1 бит, после чего анализируется значение флага CF:
Если CF=0, выводится символ ‘0’,
если CF=1 выводим символ ‘1’. Проверка битов осуществляется в цикле.

```
org 100h ;Программа начинается с адреса 100h
jmp start ;Безусловный переход на метку start
;-- Данные ------------------------------------------------------
v dw 12345
pak db 13,10,'Press any key...$'
;----------------------------------------------------------------
start:
mov bx,[v] ;BX = v
mov ah,2 ;Функция DOS 02h - вывод символа
mov cx,16 ;Инициализация счётчика цикла
lp:
shl bx,1 ;Сдвиг BX на 1 бит влево
mov dl,'0' ;dl = '0'
jnc print ;Переход, если выдвинутый бит равен 0
inc dl ;dl = dl + 1 = '1'
print:
int 21h ;Обращение к функции DOS 02h
loop lp ;Команда цикла
mov ah,9 ;\
mov dx,offset pak ; > Вывод строки 'Press any key...'
int 21h ;/
mov ah,8 ;\
int 21h ;/ Ввод символа без эха
mov ax,4C00h ;\
int 21h ;/ Завершение программы
```

Заказ № _____ от «___» _____ 2022 г. Тираж _____ экз.
Изд-во СевГУ