Лабораторная работа №4
Типы данных Ассемблера. Пересылка данных.

Цель работы:

– знакомство с типами переменных Ассемблера и их размещением в памяти;

– изучение команд пересылки данных Ассемблера;

– получение навыков работы с отладчиком.

Немного теории…

Самое главное в процессоре это регистры. Регистры состоят из триггеров. Триггер может иметь два значения «0» или «1». Регистры бывают 8-и, 16-и, 32-х и 64-х разрядные. Если регистр 8-разрядный, то в нем 8 триггеров.

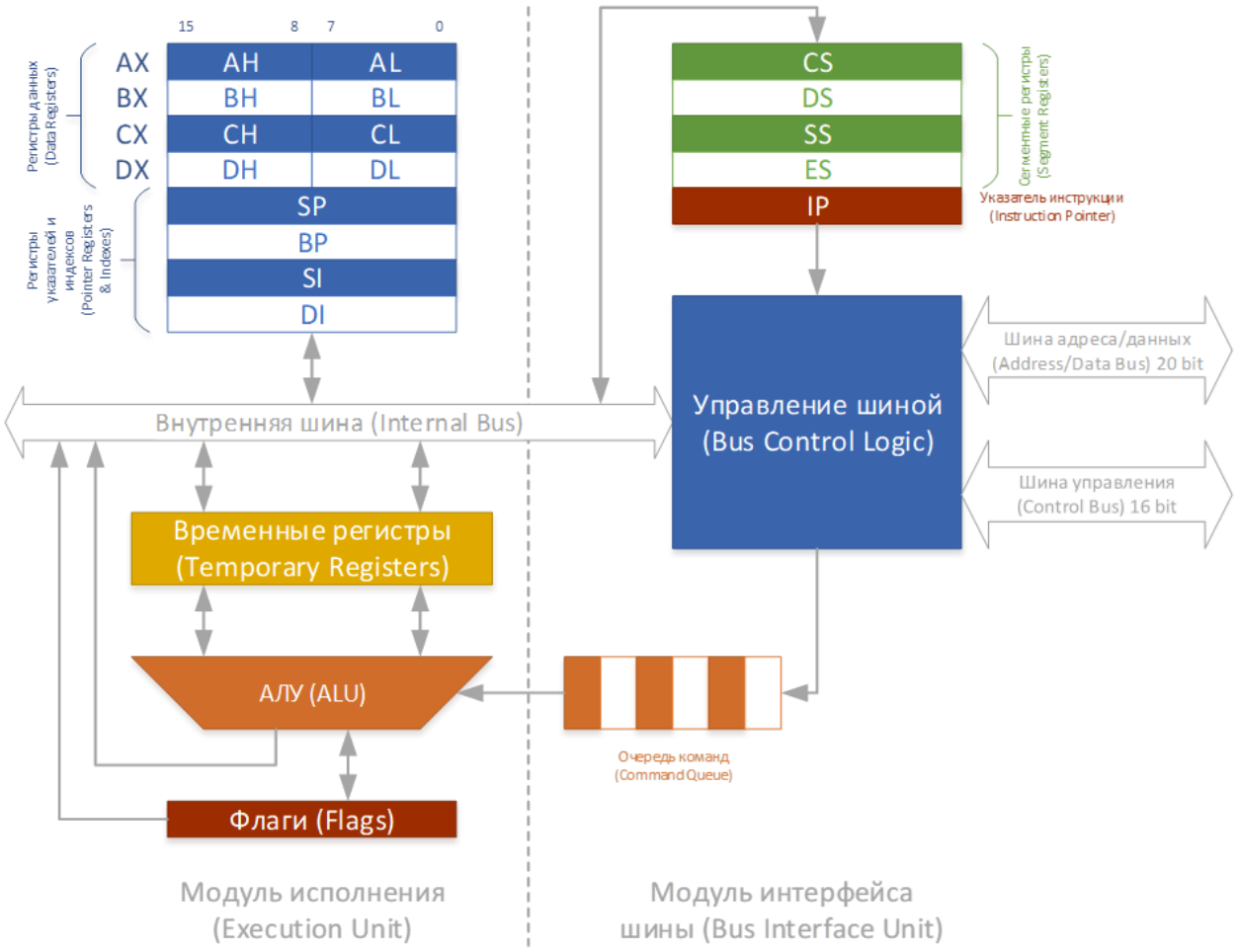

Рис. 4.1. Архитектура МП i8086

Регистр используется для промежуточного хранения информации, некоторые регистры хранят только определённую информацию. Также

есть порты ввода вывода. Доступ к внешним устройствам происходит через порты ввода вывода, с помощью контроллера ввода-вывода. Не путайте порты ввода-вывода с портами LPT, COM и т.д.

Директивы определения данных в Ассемблере.

В общем случае все директивы объявления данных имеют такой синтаксис:

[имя] директива <значение>

Синтаксис параметра <*значение*> может быть следующим:

– *?* (неинициализированные данные);

– *значение* (значение элемента данных);

– DUP(dup_выражение [,dup_выражение]) объявление и инициализация массивов.

К директивам объявления и инициализации простых данных относятся:

db *(Define Byte)* – определить байт. Директивой *db* можно задавать следующие значения:

– выражение или константу, принимающую значение из диапазона -128...+127 для чисел со знаком; 0...255 для чисел без знака;

– 8-битовое относительное выражение, использующее операции *HIGH* и *LOW*;

– символьную строку из одного или более символов? заключенную в кавычки.

dw *(Define Word)* – определить слово. Директивой *dw* можно задавать следующие значения:

– выражение или константу, принимающую значение из диапазона -32768...32767 для чисел со знаком; 0...65535 для чисел без знака;

– 1- или 2-байтовую строку, заключенную в кавычки.

dd *(Define Double word)* - определить двойное слово. Директивой *dd* можно задавать следующие значения:

– выражение или константу, принимающую значение из диапазона -2147483648...+2147483647 для чисел со знаком; для чисел 0...4 294 967 295 без знака;

– относительное или адресное выражение, состоящее из 16-битового адреса сегмента и 16-битового смещения;

– строку длиной до 4 символов, заключенную в кавычки;

dq *(Define Quarter word)* – определить учетверенное слово;

df *(Define Far word)* – определить указатель дальнего слова;

dp *(Define Pointer)* – определить указатель 48 бит;

dt *(Define Ten Bytes)* – определить 10 байт.

Для резервирования памяти под массивы используется директива *dup:*

```
area    dw 10 dup(?)          ; резервируется  память объемом 10 слов
string  db 20 dup('*')        ; строка заполняется кодом символа '*'
array   dw 3 dup(8)           ; массив из 3 слов инициализир. числом 8
t       db 4 dup(5 dup(8))    ; 20 восьмерок
```

Пример применения директив определения данных

```
; определение байта
        v1      db   ?              ; не инициализировано
        v2      db   'ИП21'         ; символьная строка
        v3      db   6              ; десятичная константа
        v4      db   0afh           ; шестнадцатеричная константа
        v5      db   0110100b       ; двоичная константа
;определение слова
        w1      dw   bff3h          ; шестнадцатеричная константа
        w2      dw   11101111b      ; двоичная константа
        w3      dw   2,24,5,7       ; 4 константы
        w4      dw   23 dup(*)      ; 23 звездочки
;определение двойного слова
        d1      dd   ?              ; не определено
        d2      dd   'uAudf'        ; символьная строка
        d3      dd   08734          ; десятичная константа
        d4      dd   087h, 85fh     ; две константы
;определение учетверенного слова
        v1      dq   ?              ; не определено
        v2      dq   a83dh          ; константа
```

Пересылка данных.

Синтаксис команды MOV Ассемблера:

MOV [приёмник], [источник]

С помощью этой команды можно переместить значение из источника в приёмник. То есть, команда *MOV* копирует содержимое источника и помещает это содержимое в приёмник.

Никакие флаги при этом не изменяются.

При использовании этой команды следует учитывать, что имеются некоторые ограничения. А именно, инструкция *MOV* не может:

– записывать данные в регистры CS и IP;

– копировать данные из одного сегментного регистра в другой сегментный регистр (сначала нужно скопировать данные в регистр общего назначения);

– копировать непосредственное значение в сегментный регистр (сначала нужно скопировать данные в регистр общего назначения).

Источником может быть один из следующих:

– область памяти (*mem*);

– регистр общего назначения (*reg*);

– непосредственное значение (например, число) (*imm*);

– сегментный регистр (*sreg*).

Приёмником может быть один из следующих:

– область памяти (*mem*);

– регистр общего назначения (*reg*);

– сегментный регистр (*sreg*).

Пример использования инструкции *MOV*:

```
mov ax,0b800h     ; установить ax = b800h
mov ds,ax         ; копировать значение из ax в ds
mov cl,'a'        ; cl = 41h (ascii-код)
```

Просмотр выполнения программы в отладчике.

Для отладки написанной программы можно воспользоваться отладчиком, например, OllyDbg. Программа *OllyDbg* — 32-битный отладчик (*debugger*), предназначенный для анализа и модификации откомпилированных исполняемых файлов и библиотек, работающих в режиме пользователя. *OllyDbg* отличается интуитивно понятным интерфейсом, подсветкой специфических структур кода, простотой в установке и запуске.

Исполняемый файл программы загружается в отладчик через меню: *File -> Open*.

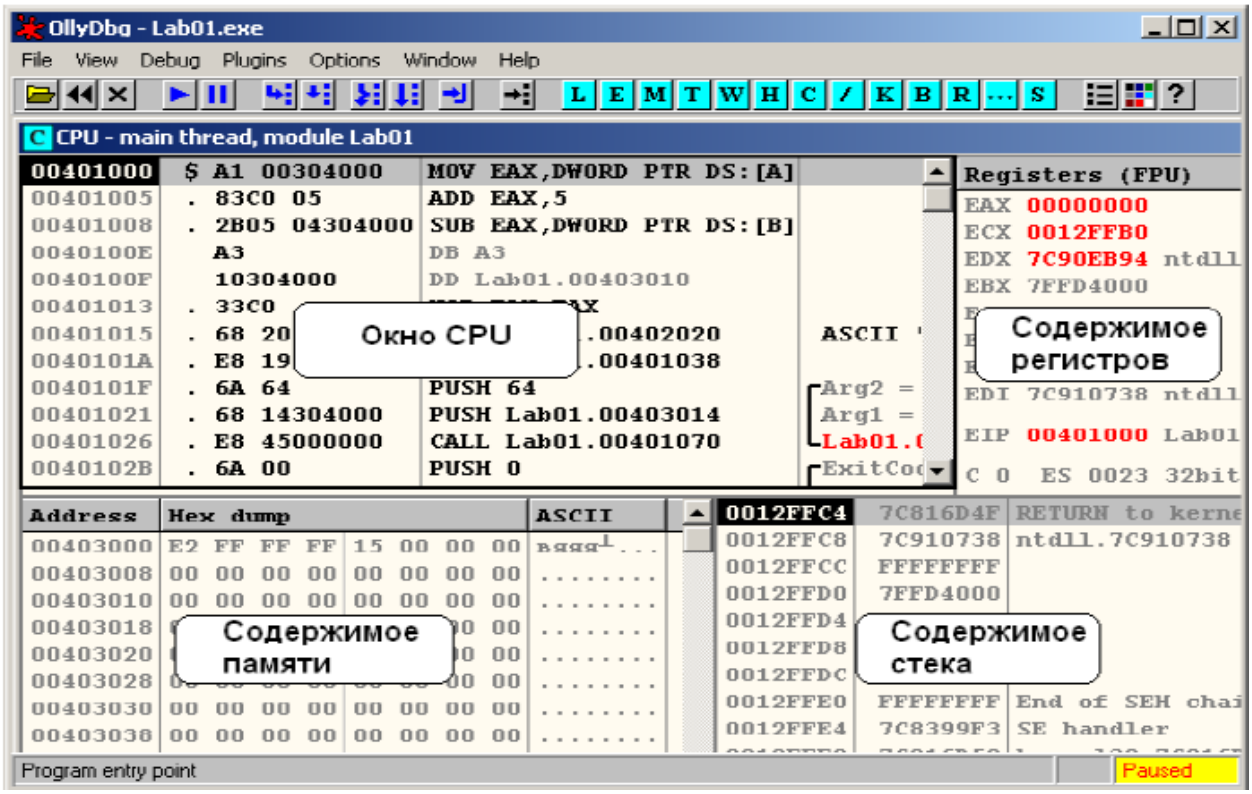

Рис. 4.2. Окно программы *OllyDbg*

В окне программы (рис.4.2) можно выделить четыре области:

– окно CPU, в котором высвечиваются адреса команд исполняемой программы, шестнадцатеричные коды команд программы, результаты дисассемблирования и результаты выделения параметров процедур;

– окно регистров, в котором отображается содержимое регистров процессора, включая регистр флагов;

– окно памяти – окно, в котором высвечиваются адреса памяти (*Address*) и ее шестнадцатеричный (*Hex dump*) и символьный (ASCII) дампы;

– окно стека, в котором отображается содержимое области памяти, отведенной под стек.

Стек – область памяти, в которой хранится информация о каждом обращении к функции. Кроме того, в стеке могут временно храниться данные, например, содержимое регистров.

Стек увеличивается в сторону уменьшения адресов памяти. Когда вы добавляете что-то в стек, то эти данные помещаются по адресу памяти младшему, чем адрес предыдущего элемента. То есть, по мере роста стека адрес вершины стека уменьшается.

Инструкция *push* заносит что-нибудь на верх стека, а *pop* уносит данные оттуда. Например, *push EAX* выделяет место наверху стека и

помещает туда значение из регистра *EAX*, а *pop EAX* переносит любые данные из верхней части стека в *EAX* и освобождает эту область памяти.

Регистр *ESP* — указатель стека. Он хранит адрес вершины стека и изменяет своё значение при записи или чтении из стека.

В исходный момент времени (см. рис.4.2) курсор находится в окне CPU, т.е. программа готова к выполнению. Для выполнения команд программы в пошаговом режиме используют следующие функциональные клавиши отладчика:

– F7 (*step into*) – выполнить шаг с заходом в тело процедуры;

– F8 (*step over*) – выполнить шаг, не заходя в тело процедуры.

Адрес начала программы *00401000h*.

Адрес начала раздела инициированных данных – *00403000h*. Каждое значение занимает столько байт, сколько резервируется соответствующей директивой. В отладчике каждый байт представлен двумя шестнадцатеричными цифрами. Кроме того, используется обратный порядок байт, т.е. младший байт числа находится в младших адресах памяти (перед старшим). Если шестнадцатеричная комбинация соответствует коду символа, то он высвечивается в следующем столбце (ASCII), иначе в нем высвечивается точка.

Не инициированные данные располагаются после инициированных с адреса, кратного 16.

При каждом выполнении команды мы можем наблюдать изменение данных в памяти и/или в регистрах, отслеживая процесс выполнения программы и контролируя правильность промежуточных результатов.

Так после выполнения первой команды число *A* копируется в регистр *EAX*, который при этом подсвечивается красным. При записи в регистр порядок байт меняется на прямой, при котором первым записан старший байт.

Задание 1.

1. Написать программу, где в сегменте данных будут созданы следующие переменные:

Dec = 65;
Neg = -160;
Bin1 = 0b1000111011;
Bin2 = 0b1011110000;
Hex = 0xAD456C4;
Text = ‘Good bye, America’;
Array[5] = {256, 765, 89, 654, 9}.

Для создания переменных выделить минимально необходимый объем памяти.

В тексте программы выполнить очистку регистров *EAX* и *EBX* с помощью команд *MOV* и *XOR* соответственно.

Переместить переменную *Bin1* в регистр *EAX* и логически умножить ее на переменную *Bin2*. Результат сохранить в переменную *Bin3*.

Сохранить переменные *Dec* и *Hex* в стек. Извлечь переменные из стека, поменяв местами их значения.

2. Открыть программу в отладчике. Указать адреса расположения переменных в памяти, заполнив таблицу:

Имя переменной	Адрес	Порядок байт в памяти

3. Выполнить программу в пошаговом режиме. После выполнения каждого шага заносить данные в таблицу:

EAX	EBX	Bin3	ESP	FC	FZ	FP	FS	прочее…

4. По каждому результатам п.3 и п.4 сделайте выводы.

Содержание отчёта.

1. Название работы.
2. Цель работы.
3. Постановка задачи.
4. Текст программ с комментариями.
5. Результаты выполнения программы в отладчике.
6. Выводы.