Основы программирования на языке ассемблера

Микропроцессоор

Микропроцессор — это центральный блок персонального компьютера, предназначенный для управления работой всех остальных блоков и выполнения арифметических и логических операций над информацией.

Microsoft Macro Assembler

MASM является одним из старейших развиваемых ассемблеров. Его развивает компания Microsoft. MASM доступен в рамках такого инструмента для разработки приложений, как Visual Studio. Преимуществом MASM является то, что MASM использует для своих инструкций синтаксис Intel. Недостатком MASM является наличие официальной поддержки только для ОС Windows.

GNU Assembler

GAS поставляется как компонент набора компиляторов GCC. Поскольку компиляторы GCC довольно распространенны и являются кроссплатформенными, то GAC соответственно также можно использовать на разных платформах. Из недостатков можно отметить, что GAS использует синтаксис, отличный от синтаксиса Intel (а именно синтаксис AT&T).

Netwide Assembler

NASM развивается как opensource-проект и использует синтаксис, который похож на синтаксис Intel. Является кросс-платформенным и работает почти на любой платформе.

Flat Assembler

Flat Assembler (FASM) является кросс-платформенным и поддерживает основные ОС (Linux, Windows и MacOS). Тоже развивается как проект open source. Используемый синтаксис похож на NASM. Примечателен тем, что написан на самом FASMe и имеет специальную небольшую IDE для написания программ.

Инструкции

Основу программы на ассемблере составляют инструкции - некоторые действия, например, сложение двух значений, помещение в регистр значения и т.д. При выполнении программы процессор выбирает и интерпретирует каждую инструкцию.

Инструкции

Каждой инструкции сопоставляется определенный машинный двоичный код, который также называется кодом инструкции или кодом операции (опкод, opcode).

Инструкция, которая копирует в регистр RAX число 1, имеет опкод C7 в шестнадцатеричной форме или 11000111 в двоичной форме.

Мнемоники

Написание машинного кода вручную возможно, но излишне громоздко. На практике вместо опкодов применяются так называемые мнемоники - человекочитаемые названия инструкций. Например, инструкция, которая копирует в регистр некоторое значение, имеет мнемонику mov (от слова "move" - помещать, поместить).

Программа

Программа состоит из набора подобных инструкций. Процессор запускает программы через цикл выборки-выполнения (fetch-execute cycle). Компьютер считывает по одной инструкции за раз. Для этого процессор обращается к специальному регистру - указателю команд (или регистр IP), который также называется программным счетчиком (или PC) и который хранит адрес инструкции для выполнения.

Процессор

Выполняет бесконечный цикл следующих операций:

- Считывает инструкцию с адреса памяти, указанного указателем инструкции - регистром IP/PC
- Декодирует инструкцию (т.е. выясняет, что означает инструкция)
- Перемещает указатель инструкций (регистр IP/PC) к следующей инструкции
- Выполняет указанную инструкцию

Команды пересылки данных

MOV — передача данных

Это одна из основных команд для перемещения данных. Она позволяет копировать данные из одного места в другое.

Формат:

```
MOV dest, src
```

MOV — передача данных

Команда перемещает значение из источника (src) в приемник (dest). Источник и приемник могут быть регистром, ячейкой памяти или непосредственным значением.

mov eax, ebx ; Перемещение содержимого EBX в EAX

mov [var], 10 ; Запись числа 10 в переменную var

XCHG — обмен данными

Эта команда меняет местами значения двух операндов.

Формат:

XCHG op1, op2

XCHG — обмен данными

Меняет содержимое op1 и op2. Оба операнда могут быть либо регистрами, либо регистр и память.

```
xchg ax, bx ; Обмен значений AX и BX
```

PUSH/POP — работа со стеком

Эти команды используются для помещения данных в стек и извлечения оттуда.

PUSH:Помещает значение в вершину стека.

POP:Извлекает значение из вершины стека.

PUSH/POP — работа со стеком

```
push eax  ; Поместить EAX в стек
pop ebx   ; Извлечь значение из стека в EBX
```

Способы адресации

Прямая адресация

При прямой адресации указывается конкретный адрес ячейки памяти, куда нужно обратиться.

```
mov al, [0x1000]   ; Чтение байта из ячейки памяти по адресу 0x1000
```

Косвенная адресация через регистр

Здесь адрес хранится в одном из регистров общего назначения.

mov bx, 0x2000 ; Загружаем адрес в регистр BX

mov al, [bx] ; Читаем байт из ячейки памяти по адресу в BX

Базовая + индексная адресация

Этот способ позволяет комбинировать базовый адрес и смещение.

```
mov bx, 0x3000  ; База
mov si, 4  ; Индекс
mov al, [bx+si]  ; Доступ к ячейке по адресу = 0x3000 + 4
```

Масштабируемая индексация

Позволяет использовать масштабирование индекса для расчета смещения.

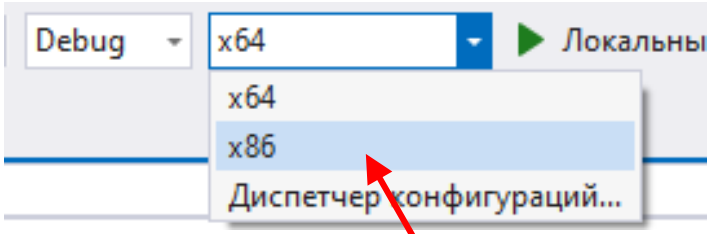
```
mov bx, 0x4000 ; База

mov si, 2 ; Индекс

mov al, [bx+si*2] ; Доступ к ячейке по адресу = 0x4000 + 2 * 2
```

Ассемблерные вставки в C++

Что это

Ассемблерная вставка - это возможность использовать в основном коде программы, написанном на языке высокого уровня, вставки из низкоуровневого кода.

Как использовать




```
#include <iostream>
using namespace std;
int main()
{
    //код на с++
    cout << "Hello World!\n";
    //ассемблерная вставка
    __asm {
        //код на ассемблере
    }
    //код на с++
    cout << "Bye World!\n";
}
```