ЛЕКЦИЯ. **Элементы программирования на языке АССЕМБЛЕР**

Ассемблер работает с ячейками памяти и регистрами. Регистрами в языке ассемблер называют ячейки памяти, расположенные непосредственно на кристалле с АЛУ (процессор). Особенностью этого типа памяти является скорость обращения к ней, которая значительно быстрее оперативной памяти ЭВМ. Она также называется сверхбыстрой оперативной памятью (СОЗУ или SRAM).

Существуют следующие виды регистров:   
1. Регистры общего назначения (РОН).   
2. Флаги.   
3. Указатель команд.   
4. Регистры сегментов.

Есть 8 регистров общего назначения, каждый размером в 32 бита.

Доступ к регистрам EAX, ECX, EDX, EBX может осуществляться в 32-битовом режиме, 16-битовом - AX, BX, CX, DX, а также 8-битовом - AH и AL, BH и BL и т. д.

Буква "E" в названиях регистров означает Extended (расширенный). Сами имена же связаны с их названиями на английском:   
Accumulator register (AX) - для арифметических операций.   
Counter register (CX) - для сдвигов и циклов.   
Data register (DX) - для арифметических операций и операций ввода/вывода. Base register (BX) - для указателя на данные.   
Stack Pointer register (SP) - для указателя вершины стека.   
Stack Base Pointer register (BP) - для индикатора основания стека.   
Source Index register (SI) - для указателя отправителя (источника).   
Destination Index register (DI) - для получателя.

Специализация РОН языка ассемблер является условной. Их можно использовать в любых операциях. Однако некоторые команды способны применять только определенные регистры. Например, команды цикла используют ESX для хранения значения счетчика.

В приводимых ниже программах важна также работа со стеком. Для записи в стек используется команда push, для чтения из стека – pop.

Рассмотрим следующую программу на языке с++, в которую добавлен код на ассемблере:

#include "stdafx.h"

#include <iostream>

void sum(int ,int );

int main()

{

int a=12;

int b=34;

\_asm

{

push a

push b

call sum

pop b

pop a

}

getchar();

return 0;

}

void sum(int a,int b)

{ printf("%d\n",a+b); }

Начало кода на ассемблере определяется ключевым слово \_asm. В данном примере код на ассемблере вызывает функцию для суммирования на языке С. Передача аргументов выполняется через стек. Загрузка в стек делается так:

push a

push b

Далее выполняется вызов call sum. Обратим внимание на то, что аргументы не указываются в скобках. Они передаются через стек. Это важно вообще для понимания передачи аргументов при вызовах программ. После получения результата стек следует очистить:

pop b

pop a

Обратимся к другим примерам. В следующем примере показан вызов оператора printf. Это оператор форматированного вывода. Для его выполнения нужно задать строку форматов, и строку для вывода.

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

#include <stdio.h>  
*// definition of constant string*char format**[] =** "%s %s\n"**;**  
char hello**[] =** "Hello"**;**  
char world**[] =** "earth"**;**  
int main**(** void **)  
{**  
\_\_asm *// start ASM code here* **{  
mov** eax**,** offset world *// move the address of world to eax* **push** eax *// Push the address of world to the stack* **mov** eax**,** offset hello *// eax = &amp;(hello[0])* **push** eax *// push hello to the stack* **mov** eax**,** offset format *// eax = format* **push** eax *// push format to the stack* **call** printf *// here is the problem printf is not define  
// at the time of compilation.  
// It will be define in run time.  
//clean up the stack so that main can exit cleanly  
//use the unused register ebx to do the cleanup* **pop** ebx **pop** ebx **pop** ebx **}** *// end of ASM code***}**

Каждый параметр загружается в два действия:

**mov** eax**,** offset world *// move the address of world to eax***push** eax *// Push the address of world to the stack*

Команда mov загружает слово world (массив, идентификатор world ссылается на первый символ) в регистр eax. Offset указывает на то, что грузится адрес.

Команда push eax грузит в стек операнд world. Далее все делаем по аналогии с другими параметрами. Вызов printf

**call** printf

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

Следующий пример показывает вызов из ассемблера системной функции MessageBox. Эта функция используется для отображения диалогового окна сообщения. Сообщение объявляется в секции .data

.data

HelloWorld db "Hello World!", 0

Программа написана на ассемблере MASM. Программу можно набрать в любом текстовом файле с расширением masm. Компиляция выполняется так

ml имя файла.masm

.386

.model flat, stdcall

option casemap :none

include \masm32\include\windows.inc

include \masm32\include\kernel32.inc

include \masm32\include\user32.inc

includelib \masm32\lib\kernel32.lib

includelib \masm32\lib\user32.lib

.data

HelloWorld db "Hello World!", 0

.code

start: invoke MessageBox, NULL, addr HelloWorld, addr HelloWorld, MB\_OK

invoke ExitProcess, 0

end start

Скомпилированную программу следует слинковать и запустить на выполненние:

link имя\_скомпилированного файла

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

Познакомимся теперь с работой математического сопроцессора. Он позволяет обрабатывать вещественные числа и вычислять тригонометрические, гиперболические и логарифмические функции. Вот пример:

#include “stdafx.h”

#include <stdio.h>

int main()

{

float aa=10.0;

\_\_asm{

FINIT

FLD aa

FLD aa

FMUL

FLDPI

FMUL

FST aa

FWAIT

}

printf (“%f”,aa);

getchar();

return 0;

}

Команда **инициализации** сопроцессора - FINIT .

**Загрузка в ST(0)** операнда с именем aa - FLD aa .

**Умножение** ST(0)xST(1) с занесением результата в ST(0) - FMUL

Загрузка в ST(0) числа π - FLDPI

**Запись результата в** ячейку (переменную) памяти aa - FST aa.

**Ожидание** завершения работы сопроцессора – FWAIT.

Остальные команды, которые могут пригодиться в выполнении изучить самостоятельно

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

**.data**

**criteria dt 3.3333e-15** ;could be initialized or set by the program

**temp\_var dt ?** ;could be initialized or set by the program

**.code**

**fld temp\_var** ; загружаем в стек значение **temp\_var**

;=> ST(0)=previous value, ST(1)=current value

**fsub st,st(1)** ; вычисляем разность между операндами

**fabs** ; получаем абсолютное значение разности

**fld criteria** ; загружаем критерий

;=> ST(0)=criteria, ST(1)=abs(difference), ST(2)=current value

**fcompp** ; выполняем сравнение критерия с разницей

;and discard both values from the FPU

;=> ST(0)=current value

**fstsw ax** ; сохраняем результат в регистре AX

**fwait** ;insure the previous instruction is completed

**sahf** ;transfer the condition codes to the CPU's flag register

;In this type of code, the computed values should already have been verified

;to be valid numbers. Their difference should thus be a valid number, as well

;as the criteria. Therefore no need to check for an indeterminate comparison.

**ja criteria\_greater** ;criteria was ST(0) for comparison

**jb criteria\_lower**

**jz criteria\_equal**

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

.dataszApi  db "MessageBox", 0  
szCap  db "Caption",0  
  
.code  
start:  
;equivalent of "invoke MessageBox, 0, addr szApi, addr szCap, MB\_OK"...  
push MB\_OK  
push addr szCap  
push addr szApi  
push 0  
call MessageBox

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++