Учебный курс

Архитектура ЭВМ и язык ассемблера

Лекция 2

заместитель министра связи и массовых коммуникаций РФ, старший преподаватель

Северов Дмитрий Станиславович

Пример

```
TITLE Сложение и вычитание
(AddSub.asm)

; числа 32-разрядные
.MODEL flat,
.386
sdtcall
ExitProcess PROTO,
.STACK 4096
dwExitCode:DWORD

DumpRegs PROTO

main PROC
.code
      mov  eax,1000h
      add  eax,4000h
      sub  eax,2000h
      call DumpRegs
      INVOKE
main ENDP
      ExitProcess,0
END   main
```

Директивы определения данных

- Структура

`[имя] код инициализатор[,инициализаторы]`

- Имя *идентификатор*
  - символическое обозначение адреса данных
- Код
  - Символическое обозначение варианта директивы
  - `BYTE, SBYTE, WORD, SWORD, DWORD, DSWORD`
  - `FWORD, QWORD, TBYTE`
  - `REAL4, REAL8, REAL10`
- Инициализаторы
  - Константное выражение, в т.ч. (символическая) константа,
  - Повтор: `DUP` , неинициализатор: ?
- Порядок следования байтов – сначала младшие

Типы операндов (команд)

- *r8, r16,r32* – 8-, 16-, 32-разрядный РОН
- *reg* – произвольный РОН
- *sreg* – 16-разрядный сегментный регистр
- *imm8,imm16,imm32* – 8-, 16-, 32-разрядное значение, заданное непосредственно в команде
- *r/m8,r/m16,r/m32* – 32-разрядный 8-, 16-, 32-разрядный операнд, кодирующий 8-, 16-, 32-разрядный РОН или адрес 8-, 16-, 32-разрядного операнда в памяти
- *mem*– адрес 8-, 16-, 32-разрядного операнда в памяти

Пересылки простые и неочевидные

`MOV` *`получатель, источник`*

- Длина одинаковая
- Один операнд – обязательно регистр
- Нельзя получать в `CS, IP, EIP`
- Нельзя *`imm16`* `в` *`sreg`*

`MOVZX/MOVESX` расширение (без)знаковое

`LAHF/SAHF` опрос/установка младших флагов

`XCHG` обмен данными

Сложение и вычитание

- Команды

INC reg/mem

DEC reg/mem

NEG reg/mem

ADD *получатель, источник*

SUB *получатель, источник*

- Флаги

ZF– обнуление

CF– выход за границу разрядной сетки

OF– выход за границу дополнительного кода

SF– копия старшего (знакового) бита

Работа с данными и адресами: операторы и директивы

- `OFFSET` – вычислить смещение от начала сегмента в адресном выражении
- `ALIGN` – установить начало очередных данных на границу указанного размера
- `PTR` – установить размер указываемых данных
- `TYPE`, `LENGTHOF`, `SIZEOF` – вычислить размер данных
- `LABEL` – задать имя и тип адреса, без выделения памяти,

Адресация

- Прямая (адрес задан непосредственно)

```
MOV al,var1

MOV al,[var1]

MOV al,[arrayB+1]

MOV al,[arrayD+4]
```

- Косвенная

```
MOV al,[esi]

INC BYTE PTR [esi]
```

Безусловный переход и цикл

`JMP` *`метка_перехода`*

– безусловный переход

`LOOP` *`метка_перехода`*

– `ECX/CX` уменьшается на единицу

– если `ECX/CX` не ноль, то переход по метке

– иначе следующая команда

`LOOPD` всегда `ECX`

`LOOPW` всегда `CX`

Ещё пример

```
TITLE Add and Subtract, Version 2 (AddSub2.asm)
; Сложение и вычитание 32-битных целых переменных
; результат – в переменной.

INCLUDE Irvine32.inc

.data
val1      dword  10000h
val2      dword  40000h
val3      dword  20000h
finalVal  dword  ?
.code
main PROC

   mov eax,val1          ; Загрузить 10000h
   add eax,val2          ; добавить 40000h
   sub eax,val3          ; вычесть 20000h
   mov                   ; записать результат (30000h)
finalVal,eax
   call DumpRegs         ; отобразить регистры

   exit
```

Процесс создания программы

- **Редактор**

⇒Ваш исходный текст

Ввод предписаний

⇐Изменения текста

- **Ассемблер**

⇒Объектный код

Предписания трансляции

⇐Текстовые библиотеки

- **Компоновщик**

⇒Загрузочный код

Предписания компоновки

⇐Статический код

```
link32 AddSub.obj irvine32.lib kernel32.lib
```

- **Загрузчик**

⇒Исполняемый код

Предписания загрузки

⇐Решения ОС

- **ОС+аппаратура**

⇒Результат

Предписания исполнения

⇐Внешние события

⇐Внешние данные и код

Учебная библиотека

ClrScr
CrLf
Delay
DumpMem
DumpRegs
GetCommandTail
GetMseconds
GotoXY
Random32
RandomiZe
ReadHex
ReadInt
ReadString
SetTextColor
WaitMsg
WriteBin
WriteChar
WriteDec
WriteHex
WriteInt

Стек

- Понятие стека
  - LIFO (Last-In, FIst-Out)
- Стековая адресация памяти
  - `SS ESP`
  - «рост» в сторону меньших адресов
- `PUSH/POP r/m16|r/m32|imm32|imm16`
- `PUSHFD/POPFD` – флаги 32 бита
- `PUHSF/POPF` - флаги 16 бит
- `PUSHAD/POPAD` – регистры по 32 бита

`EAX,ECX,EBX,ESP,EBP,ESI,EDI`

- `PUHSA/POPA` - регистры по 16 бит

`AX, CX, BX, SP, BP, SI, DI`

Стек, использование.

```
PUSHF  ; SP<=SP-2,  [SS:SP]<=Flags
POPF   ; Flags<=[SS:SP]; SP<=SP+2,
```

- Полезно:
  - Сохранение регистров
  - Пересылка “без регистров”
  - Доступ к элементам, BP.
- ВАЖНО:
  - Баланс операций `PUSH` и `POP`
  - Контроль границ
  - Соглашения при передаче

```
TITLE Программа реверсирования строк (RevString.asm)
INCLUDE Irvine32.inc
.data
aName   BYTE "abcdefghijklmnopqrstuvwxyz0123456789",0
nameSize = ($ - aName) - 1
.code
main PROC
; Поместим строку посимвольно в стек
    mov   ecx,nameSize
    mov   esi,0
L1: movzx eax,aName[esi]      ; Загрузим символ строки
    push  eax                 ; Поместим его в стек
    inc   esi                 ; в обратном порядке.
    Loop  L1
; Восстановим строку из стека
    mov   ecx,nameSize        ; Загрузим символ из стека
    mov   esi,0
L2: pop eax                   ; Сохраним в массиве
    mov   aName[esi],al       ; строку
    inc   esi
    Loop  L2
    mov   edx,OFFSET aName
    call  WriteString
    call CrLf
    exit
main ENDP
END main
```

Пример работы со стеком

Определение процедур

- PROC и ENDP

```
<имя процедуры>    PROC <FAR|NEAR>
                   <тело процедуры>
<имя процедуры>    ENDP
```

- Документирование
  - Целевые действия
  - Ожидаемые параметры
  - Возвращаемый результат
  - Необходимые условия
- CALL и RET
  - адрес возврата - в стеке
  - CALL <имя процедуры> (адрес втолкнуть и перейти)
  - RET (оказаться по вытолкнутому адресу)

Использование процедур

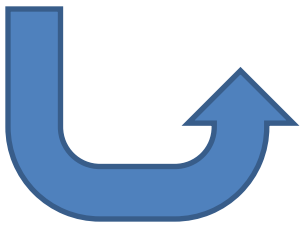
- Вложенные вызовы
- Локальные `L1:` и глобальные `L2::` метки
- Передача параметров через регистры
- Сохранение и восстановление регистров

`PROC USES` *`reg1 reg2`* ...

- Функциональная декомпозиция
  - Разбиение сложного действия на простые
  - Автономная проверка простых действий
  - Обнаружение связей и оценка их «силы»
  - Разделение «структурирования» и кодирования

Пример с процедурами

```
TITLE Программа суммирования целых чисел          (Sum2.asm)
; Запрашивает несколько целых чисел, сохраняет их в массиве, ; вычисляет сумму и отображает
полученный результат
INCLUDE Irvine32.inc
IntegerCount = 3              ; Размер массива
.data
promptl BYTE "Введите целое число со знаком: ",0
prompt2 BYTE "Сумма чисел равна: ",0
array  DWORD IntegerCount DUP(?)
.code
mainPROC
call ClrScr
mov  esi,OFFSET array
mov  ecx,IntegerCount
call PromptForIntegers
call ArraySum
call DisplaySum
exit main ENDP
;-----------------------------------------------------------
PromptForIntegers PROC
; Запрашивает числа и записывает их в массив.
; Передается: ESI = адрес массива двойных слов,
;           ECX= размер массива.
; Возвращается: ничего
; Вызывает: ReadInt, WriteString
;-----------------------------------------------------------
pushad                    ; Сохраним все регистры
mov  edx,OFFSET promptl    ; Адрес приглашения
L1:
call WriteString           ; Выведем приглашение
call ReadInt               ; Прочитаем число (оно в EAX)
mov   [esi],eax            ; Запишем число в массив
add  esi,4                 ; Скорректируем указатель
; на следующий элемент массива
call CrLf                  ; Перейдем на новую строку' на
экране
loop L1
popad                      ; Восстановим все регистры
ret
PromptForIntegers ENDP
```



Пример с процедурами (продолжение)

```
ArraySum PROC
Вычисляет сумму элементов массива 32-разрядных целых чисел
Передается: ESI = адрес массива
ECX = количество элементов массива
Возвращается: EAX = сумма элементов массива
push esi                    ; Сохраним значения регистров ESI и ECX
push, ecx;
mov  eax,O                  ; Обнулим значение суммы
Ы: add   eax,[esi]                ; Прибавим очередной элемент массива
add   esi,4                 ; Вычислим адрес следующего элемента массива
loop L1                     ;Повторим цикл для всех элементов массива
 pop   ecx                  ; Восстановим значения регистров ESI и ECX
 pop   esi
ret                        ; Вернем сумму в регистре EAX
ArravSum EXDP
;-------------------------------------------------------------------------------
DisplavSum PROC
Отображает сумму элементов массива на экране.
 Передается: EAX = сумма элементов массива
Возвращается: ничего
 Вызывает: WriteString, Writelnt
;-------------------------------------------------------------------------------
push edx
mov edx.OFFSET prompt2     ; Выведем пояснение
call WriteString
call Writelnt              ; Отобразим регистр EAX
call CrLf
pop edx
DisplaySum ENDP
END main
```