Учебный курс

Архитектура ЭВМ и язык ассемблера

Лекция 3

заместитель министра связи и массовых коммуникаций РФ, старший преподаватель

Северов Дмитрий Станиславович

Команды сравнения и булевых операций

- Флаги результатов выполнения команд

`ZF`– обнуление

`CF`– выход за границу разрядной сетки

`SF`– копия старшего (знакового) бита

`OF`– нарушение дополнительного кода

`PF`– чётное число бит

- `AND,OR,XOR,NOT`
- `TEST,CMP` – меняем только флаги
- Установка и сброс отдельных флагов
- `BT,BTC,BTS,BTR` работа с семафорами

Команды условных переходов

- `J?? <op>` ; много вариантов
  - Условия – во флагах
  - До 386 метка – ближняя (-128…+127 байт)
- По результатам «сравнения»
  - **Equal, Not Equal**

  Greater, Less, Greater or **Equal, Less** or **Equal** (со знаком)

  Above, Below, Above or **Equal, Below** or **Equal** (без знака)

  Примеры: `JL` - если `SF != OF`, `JB` - если `CF=1`
- По состоянию определённого флага

`[Not] flag {Z|S|C|O|P}F set to 1»`

`JZ, JNZ,…,JP,JNP`

- По состоянию счётчика

`JCXZ, JECXZ`–обход цикла для реализации «предусловия»

Команды циклов

- LOOP* <op>; есть варианты
- LOOP: **if (!--ECX) goto <метка>**;
  - счётчик только в CX/ECX,
    - традиционно: цикл с постусловием!
    - “вошёл с CX/ECX=0”: ещё $2^{16}/2^{32}$раз.
- LOOPE/LOOPZ: Поиск отличного **if(!--ECX || ZF)goto <метка>**
- LOOPNE/LOOPNZ: Поиск требуемого **if(!--ECX || !ZF)goto <метка>**
- **Важно:** расстояние до <op> от -128 до 128 байт

Условные конструкции ЯВУ(1)

```
while(op1<op2)
   {
      op1++;
      if(op2==op3)
         X=2;
      else
         X=3;
   }
```

```
        mov   eax,op1
        mov   ebx,op2
        mov   ecx,op3
L1:     cmp   eax,ebx
        jl    L2
        jmp   L7
L2:     inc   eax
L3:     cmp   ebx,ecx
        je    L4
        jmp   L5
L4:     mov   X,2
        jmp   L6
L5:     mov   X,3
L6:     jmp   L1
L7:     mov   op1,eax
```

Условные конструкции ЯВУ(2)

```
switch(par)
   {
      case 'A':
         Process_A
();
         break;
      case 'B':
         Process_B
();
         break;
      case 'C':
         Process_C
();
         break;
      case 'D':
         Process_D
();
         break;
```

```
…
CaseTable   BYTE   'A'
            DWORD  Process_A
EntrySize       =       ($ - CaseTable)
            BYTE   'B'
            DWORD  Process_B
            BYTE   'C'
            DWORD  Process_C
            BYTE   'D'
            DWORD  Process_D
NumberOfEntries =  ($ -
                   CaseTable)/EntrySize

      mov   al,par
   mov   ebx,OFFSET CaseTable
      mov   ecx,NumberOfEntries
L1:   cmp   al,[ebx]
      jne   L2
      call  NEAR PTR [ebx + 1]
      jmp   L3
L2:   add   ebx,EntrySize
      loop  L1
```

Условные директивы ассемблера

- Условная конструкция

```
.IF условие1
   команды
[.ELSEIF условие2
   команды]
[.ELSE
   команды]
.ENDIF
```

- Цикл с постусловием

```
.REPEAT
   команды
.UNTIL условие
```

- Цикл с преусловием

```
.WHILE условие
   команды
.ENDW
```

- Сравнения
  - Регистров – без знака
  - С переменной – как определена
- Операторы в условиях

```
expr1 == expr2
expr1 != expr2
expr1  > expr2
expr1 >= expr2
expr1  < expr2
expr1 <= expr2
! expr
expr1 && expr2
expr1 || expr2
expr1 &  expr2
CARRY?  OVERFLOW? PARITY?
   SIGN? ZERO?
```

Напоминания

- Циклы (LOOP* **<op>**)
  - **Важно:** расстояние до **<op>** от -128 до 128 байт
- Фокусы оптимизации

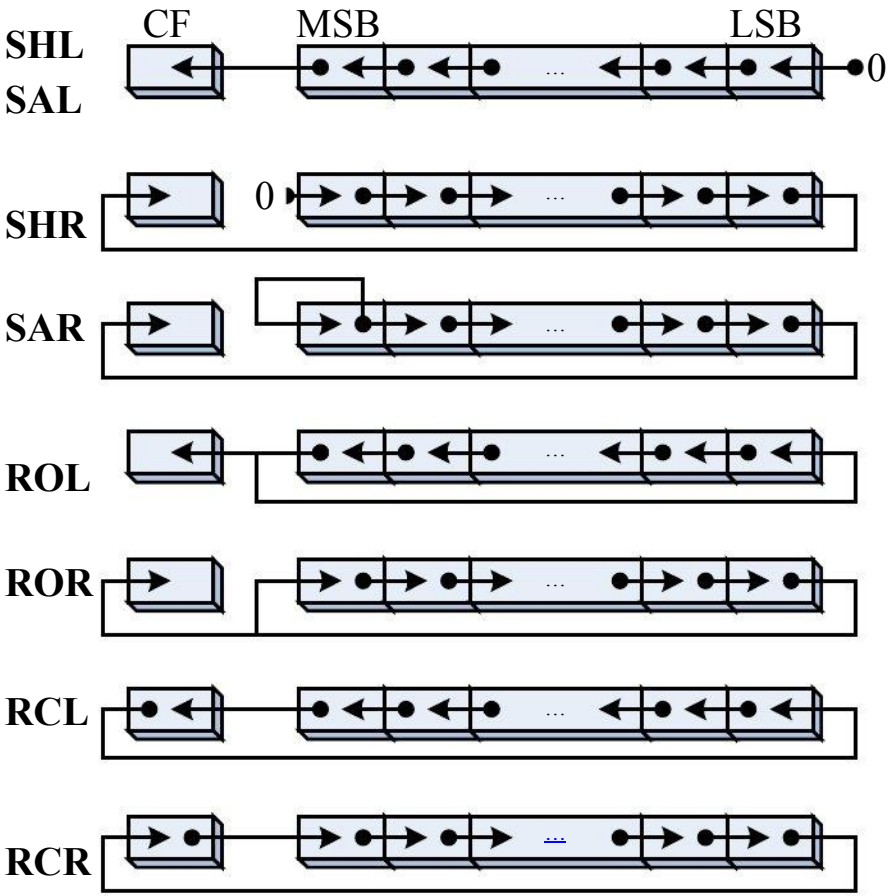
```
while(op1<op2)
    {
       op1++;
       if(op2==op3)
          X=2;
       else
          X=3;
    }
```

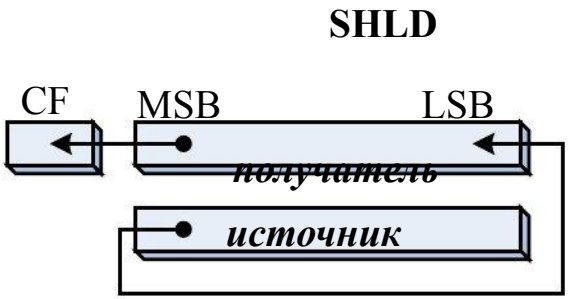
```
        mov   eax,op1
        mov   ebx,op2
L1:     mov   ecx,op3
        cmp   eax,ebx
        jl    L2
L2:     jmp   L7
        inc   eax
L3:     cmp   ebx,ecx
        je    L4
L4:     jmp   L5
        mov   X,2
L5:     jmp   L6
        mov   X,3
L6:     jmp   L1
```

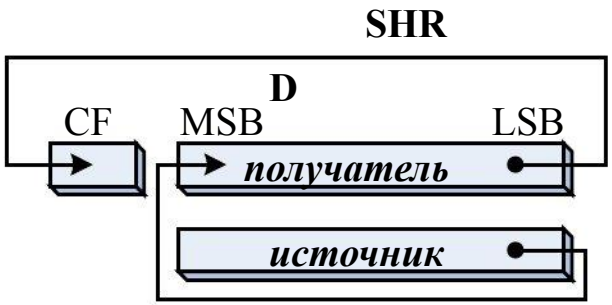
Сдвиги

??? *операнд,счётчик*

SH?D *получатель,источник,счётчик*







Умножение и деление

- MUL *операнд* IMUL *операнд*

Множимое	Множитель	Произведение
AL	*reg/mem8*	AX
AX	*reg/mem16*	DX:AX
Множимое	*reg/mem32*	Произведение

- DIV *операнд* IDIV *операнд*

Делимое	Делитель	Частное	Остаток
AX	*reg/mem8*	AL	AH
DX:AX	*reg/mem16*	AX	DX
Делимое	*reg/mem32*	Частное	Остаток

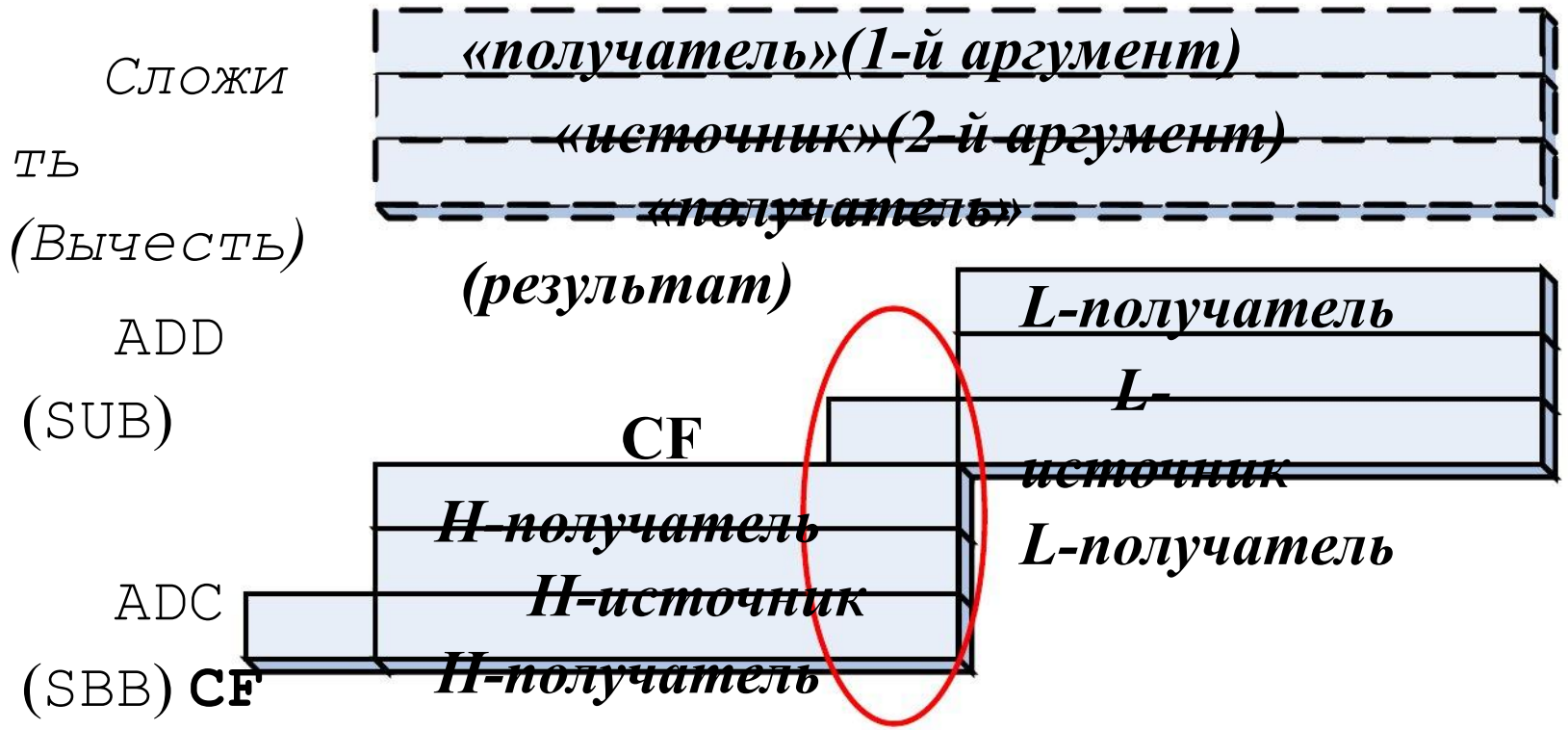
- CBW/CWBE CWD/CDQ -расширение со знаком

Тонкости применения

- Сдвиги
  - Сдвиг массива
  - Быстрое умножение
  - Выделение полей
- Умножение
  - без переполнений
    - CF, OF – большой результат
- Деление
  - исключение - деление на нуль
  - исключение - переполнение

Произвольная точность

- ADC – **ADd** with **Carry**
- SBB – **SuBtract** with **Borrow**



Об основах программирования

- Создание и инициализация автопеременных
- Область действия и время жизни переменных
- Передача параметров
  - механизм передачи: стек
  - способы передачи: «по значению» и «по ссылке»
  - входные, выходные, универсальные параметры
- Стековые фреймы, контекст
- Рекурсия

Локальные переменные

- Назначение
  - Упрощение отладки
  - Переиспользование памяти
  - Переиспользование имён

```
BubbleSort PROC
      LOCAL Temp:DWORD
      LOCAL SwapFlag:DWORD
; ______________________
      RET
BubbleSort ENDP
```

```
 lea esi,SwapFlag
```

```
BubbleSort      :
        push ebp
        mov ebp,esp
        add esp,0FFFFFF8h
        mov esp,ebp
        pop ebp
        ret
```

загрузка эффективного адреса

обращение по адресу

Параметры регистровые и стековые

```
pushad
mov esi,OFFSET array
mov ecx,LENGTHOF array
mov ebx,TYPE array
call DumpMem
popad
```

```
push TYPE  array
push LENGTHOF array
push OFFSET array
call DumpMem
```

INVOKE DumpMem,OFFSET array,LENGTHOF array,TYPE array

Аргумент **INVOKE**	Примеры
Непосредственное значение	`10,300h,OFFSET myList,TYPE array`
Целочисленное выражение	`(10*20), COUNT`
Имя переменной	`myList, array, Temp, SwapFlag`
Адресное выражение	`[myList+2], [ebx+esi]`
Имя регистра	`eax, bl, edi`
Аргумент **INVOKE**	Примеры

`ArraySum PROTO ptrArray:PTR DWORD, sZArray:DWORD`

Разные параметры и переменные

```
TITLE ArraySum
(ArrySum.asm)
INCLUDE Irvine32.inc

.data

Array DWORD 50 DUP(5)

.code
main PROC

   push LENGTHOF Array
   push OFFSET
   call Array
   add  esp,8
   ArraySum
   call  WriteDe
   call  c
         Crlf
    exit
main ENDP


sum
pArray EQU
<[ebp+8]>
count EQU
<[ebp+12]>
```

```
   ArraySum
   PROC
push ebp     ; сохранить EBP
mov          ; указатель на локальные
ebp,esp      ; место для локальной
sub esp,4    ; сохранить ESI
push esi
mov DWORD PTR
sum,0
mov esi,pArray
mov ecx,count
L1:
   mov eax,[esi] ;взять элемент
   addсива       ; добавить с сумме
   sum,eax       ; к следующему элементу
   add esi,4
   loopd L1
   popesi     ;восстановить ESI
   moveax,su  ;результат в EAX
   movm       ;удалить локальные
   popesp,eb  ;восстановить EBP
   retp
      ebp
ArraySum
ENDP
```