В.Г. Баула
Типичные ошибки при работе на Ассемблере

То, что неясно, следует выяснить. То, что трудно, следует делать с великой настойчивостью.

Конфуций, V век до н.э.

---

1. *Да не будешь ты смешивать в одной команде операнды разной длины.*
2. *Да не будешь ты путать команды для знаковых и беззнаковых чисел.*
3. *Да не будешь ты использовать в одной команде два явных операнда из памяти.*
4. *Да будет у тебя каждая команда знать длину своих операндов.*
5. *Да будешь ты различать адрес ячейки памяти и её содержимое.*

Заповеди программиста на Ассемблере

Программировать на Ассемблере сложно. Разберём типичные ошибки начинающих программистов. Для них можно также порекомендовать прочитать на нашем сайте пособие "**Ассемблер для блондинок (и блондинов).pdf**".

1. Большинство команд (сложения, умножения и т.д.) могут обрабатывать данные длиной (1, 2, 4 или 8 байт). Чтобы настроиться на длину операндов, команда должна эту длину знать. Некоторые операнды сами указывают свою длину, для других надо эту длину явно указывать, например:

```
X: db ?
Y: dw ?
   mov eax,77; Длина константы 77 четыре байта
   add X,77; Длина константы 77 один байт
   cmp [ebx],77; ошибка, длина переменной по адресу [EBX], и, следовательно,
               ; константы 77 неизвестна, надо, например, писать
   cmp word ptr [ebx],77; Длина переменной по адресу [EBX] два байта
   sub ebx,X; Ошибка, длины операндов не совпадают
   adc [ebx],ax;  Длина переменной по адресу [EBX] два байта
   mov ebx,dword ptr X; Будет EBX:=<X,X+1=Y,X+2=Y+1,X+3=&>, т.е.
;  это сама переменная X и три байта за ней; хотя здесь и нет синтаксической
;  ошибки, но, вероятно, программист хотел просто расширить X
;  до четырёх байт и записать на ebx; это надо делать командами
   movsx ebx,X; если считать X знаковой переменной, или
   movzx ebx,X; если считать X, беззнаковой переменной
   movzx ebx,[eax]; ошибка, переменная по [EAX] один или два байта ?
```

2. Некоторые команды существуют и в знаковой, и в без знаковой форме (**mul** и **imul**, **div** и **idiv**), их не надо путать. Особенно часто путают операции для знаковых и беззнаковых величин, например, команды переходов (**jl** и **jb**), макрокоманды вывода (**outint** и **outword**) и т.д.

3. Типичной является и ошибка использования несуществующего формата команд, например

```
   mov X,[eax]; ошибка, запрещённый формат память-память
   mul 15; ошибка, нет такого формата, умножать можно только на регистр и память
         ; (очень хочется, но такого нет !!!);
```

4. Много ошибок допускается в командах умножения и деления. Умножаются всегда операнды одинаковой длины, и произведение всегда имеет в два раза бóльшую длину. А вот при делении, наоборот, делимое имеет удвоенную длину, а частное и остаток – одинарную. Не понимая этого, часто пишут

```
mov eax,X; первый сомножитель
cdq       ; <eax:edx>:=int64(X); Зачем ????? Сомнения о знаниях студента 🙁
imul Y;     второй сомножитель (знаковый)
```

5. Для вызова "чужих" (написанных другими людьми) процедур и функция используются стандартные соглашения о связях, их надо знать. Для 32-битных ЭВМ есть несколько таких соглашений, наиболее распространённое имеет имя **stdcall**, его поддерживают большинство систем программирования и, что тоже важно, библиотеки системных вызовов (процедур и функций) операционных систем (Windows, Unix и других).

По этому соглашению фактические параметры перед вызовом записываются в стек. Как Вы знаете, параметры можно передавать по ссылке и по значению. При передаче по ссылке в стек записывается 32-битный адрес фактического параметра (это адрес переменной !). При передаче по значению в стек записывается само значение фактического параметра, дополненное до длины, кратной 4 байтам. Непонимание может возникнуть при разных длинах формального и фактического параметров. Например, рассмотрим на языке Free Pascal

```
var A: byte; B: longword;
procedure P(X: word); begin ... end;
```

Если Вы хорошо учили Паскаль, то должны знать, что можно делать вызовы `P(A)` и `P(B)`. Действительно, передача по значению допустима, если есть совместимость по присваиванию между формальным и фактическим параметрами. Паскаль будет делать эти вызовы так:

```
movzx eax,A          push B
push  eax            call P; P(B)
call  P; P(A)
```

А вот процедура `P` будет брать свой параметр `X` (он длиной 2 байта) из стека, например, на регистр `AX`, так

```
mov ax,[ebp+8]
```

Это первые два (из-за перевёрнутого представления параметра в памяти), т.е. младшие байты фактического параметра длиной 4 байта.

Другая частая ошибка – не тот порядок записи параметров в стек. По соглашению **stdcall** сначала в стек записывается последний параметр, затем предпоследний и т.д. Часто ошибаются и при работе с функциями, которые возвращают свой результат на регистрах `AL`, `AX` или `EAX`. В прологе функции студенты ставят команду **push** `eax`, а в эпилоге **pop** `eax`, уничтожая результат работы своей функции.

Много ошибок допускается при работе с параметрами, переданными по ссылке. Например, процедура с заголовком

```
procedure P(var X: byte);
```

Будет, например, вызываться так:

```
A  db ?
   push  offset A
   call P
```

Будет ошибкой при в процедуре достать значение фактического параметра `A` так:

```
mov al,[ebp+8]; al:=(младший байт адреса A) !
```

Или так

```
mov al,[[ebp+8]]; Обращение к памяти двойной косвенности в машине нет !
```

Правильно доставать значение `A`, например, так:

```
mov  eax,[ebp+8]
mov al,[eax]; al:=A
```

Аналогично, когда параметр `X` передан по ссылке, будет ошибкой возвращать в нём значение процедуры (пусть, например, это значение на регистре `EBX`) так:

```
mov [ebp+8],ebx; это (Адрес X):=ebx, а надо X:=ebx
```

это надо делать так:

```
mov eax,[ebp+8]
```

```
mov [eax],ebx; это X=eax↑:=ebx
```

6. Многие, плохо выучив Ассемблер, пишут лишние команды, например:

```
mov eax,X
add eax,ebx
mov X,eax; Вместо просто add X,ebx
```

или

```
X  db ?
mov eax,0
mov al,X; Вместо просто movzx eax,X
```

или

```
sub ecx,ecx
mov ecx,[ebp+12]; Это ecx:=0; ecx:=N выглядит странно 🙁.
```

7. Локальные переменные процедуры порождаются в стеке сразу перед (стек растёт вверх!) сохранённым регистром EBP:

```
push ebp
mov  ebp,esp
sub  esp,8; var X: Longint; Y: byte
X  equ  dword ptr [ebp-8]; var X: Longint
Y  equ  byte ptr [ebp-4];  var Y: byte; Y: [Y] [?] [?] [?] 4 байта
```

Все локальные переменные уничтожаются одной командой перед восстановлением старого значения регистра EBP:

```
mov esp,ebp; Уничтожение всех локальных переменных
pop ebp
```