

Краткое руководство по программированию на ассемблере i386

Ситкарев Г.А., <sitkarev@komitex.ru>

Лаборатория Прикладной Математики и Программирования
http://amplab.syktusu.ru

1.1. Отличия AT&T от Intel

Порядок операндов в командах обратный.

AT&T	Intel
op src,dst	op dst,src

Константные значения, передаваемые как непосредственные операнды, префиксируются знаком «\$», иначе считаются адресами.

AT&T	Intel
movl \$addr,%eax	mov eax,addr
movl addr,%ebx	mov ebx,dword [addr]

Все регистры префиксируются знаком «%».

AT&T	Intel
movl \$25,%eax	mov eax,25

Относительный адрес через базовый регистр, регистр индекса, масштаб (1, 2, 4) и смещение передаётся иначе.

AT&T	Intel
movl 16(%ebx,%ecx,4)	mov eax,[16+ebx+ecx*4]
lea (%ebx,%ecx),%edi	lea edi,[ebx+ecx]
movl 8(%ebp),%esi	mov esi,[ebp+8]

К командам ассемблера добавляется префикс, если нужно обозначить размерность операндов. Соответственно для байта, полуслова, слова и двойного слова это b, s, l и q.

AT&T	Intel
mulb %cl	mul al
subw %dx,%ax	sub ax,dx
movw 8(%edx),%ax	mov ax,word [edx+8]
movl %edi,-8(%ebp)	mov dword [ebp-8],edi

Вызов по косвенному адресу.

AT&T	Intel
call *(addr)	call [addr]

1.2. Регистры общего назначения

Все функции программы имеют доступ к регистрам CPU и FPU. Список регистров и их назначение приведены в табл. 1.

Тип	Регистр	Назначение
Общего применения	%eax	возвр. значение
	%ebx	локальный регистр
	%edx	делимое
	%ecx	счётчик
	%edi	локальный регистр
	%esi	локальный регистр
	%ebp	база стека
Управляющие	%eip	адрес след. инстр.
	%eflags	флаги состояния
FPU	%st(0)	верхушка стека FPU
	%st(1)	след. после %st(0)
	...	
	%st(7)	дно стека FPU

Табл. 1

1.3. Типы данных

Соответствие типов Си и Intel i386 приведено в табл. 2.

Тип	Си	Размер	Выравнивание
Целое	char	1	1
	unsigned char	1	1
	short	2	2
	int	4	4
	long	4	4
Указатели	(любой_тип) *	4	4
	(любой_тип) (*)()	4	4
Плав. точка	float	4	4
	double	8	4
	long double	12	4

Табл. 2

1.4. Вызов функций

Аргументы функциям передают через стек в обратном порядке. После вызова функции, аргументы из стека убирают.

```
.data
:
arg1:
.long 18
arg2:
.long 175
fmt:
.asciz "I am %d years old, %d cm tall.\n"
:
.text
:
pushl arg2
pushl arg1
pushl $fmt
call printf
add $(3*4),%esp / 3*4=12
```

1.4.1. Стек вызова

Кроме регистров, каждая функция располагает кадром стека. Стек растёт сверху-вниз, от больших адресов к меньшим.

Положение	Что содержит	Кадр
4×n+8(%ebp)	аргумент n	Предыдущий
12(%ebp)	...	
8(%ebp)	аргумент 1 аргумент 0	
4(%ebp)	адрес возврата	Текущий
0(%ebp)	предыдущ. %ebp	
-4(%ebp)	область локальных переменных и сохр. регистров	
... 0(%esp)		

Табл. 3

Стек выравнивается по границе слова (4 байта). Если размер передаваемого значения не кратен 4-м байтам, то в стеке пустое место заполняют до границы слова.

1.5. Назначение регистров

Некоторые регистры имеют специальное назначение. В частности, регистры %ebp, %esp, %ebx, %edi, %esi принадлежат вызывающей функции. Функция, которую вызывают, должна вернуть прежние значения этих регистров после своего завершения.

%esp

Указатель стека содержит лимит текущего кадра стека. Это нижняя граница самого последнего слова в стеке. Указатель стека всегда выравнивается по границе слова (4 байта).

%ebp

Указатель кадра содержит базовый адрес текущего кадра стека. Таким образом, у каждой функции есть два регистра, которые содержат адреса начала и конца её кадра. Аргументы, переданные функции, находятся по положительному смещению от %ebp, а локальные переменные по отрицательному смещению от %ebp. Функция должна восстановить предыдущее значение %ebp после возврата.

%eax

Целые значения и указатели возвращаются в %eax. Если функция возвращает структуру или объединение, то она помещает их адрес в %eax. Этот затираемый регистр может использоваться внутри функции как угодно, его значение она не обязана сохранять.

%ebx

Обычно регистр служит для хранения глобальной таблицы смещений (GOT) в базонезависимом коде. Если код базозависимый, то %ebx обычный локальный регистр без специального назначения. Функция должна восстановить предыдущее значение %ebx после возврата.

%ecx, %edx

Затираемые регистры без специального назначения. Функции не обязаны восстанавливать их предыдущее значение после возврата.

%edi, %esi

Локальные регистры без специального назначения. Функция должна восстановить предыдущие значения после возврата.

%st(0)

Возвращаемое значение с плавающей точкой лежит на вершине стека регистров FPU. Внутренний формат хранения в регистрах FPU одинаков для значений одинарной, двойной и расширенной точностей. Если функция не возвращает значения с плавающей точкой, этот регистр должен быть пуст. При входе в функцию этот регистр также должен быть пуст.

%st(1),...,%st(7)

Затираемые регистры FPU без специального назначения. Эти

регистры должны быть пусты при входе и выходе из функции.

`%eflags`

Регистр флагов содержит системные флаги, такие как CF (флаг переноса) и DF (флаг направления). Флаг DF должен быть установлен в направлении «вперёд» (т. е. сброшен в 0) при входе в функцию и при выходе из неё. Прочие флаги не сохраняются и не имеют назначения.

Управляющее слово FPU

Управляющее слово FPU содержит настройки режимов с плавающей точкой, такие как маска и флаги исключений, режим округления, точность и прочие.

1.6. Пролог и эпилог

Стандартный пролог и эпилог для функции, сохраняющей регистры `%ebp`, `%edi`, `%esi`, и выделяющей для локальных переменных в стеке 20 байт.

пролог

```
pushl %ebp / сохраняем старое значение %ebp
movl %esp,%ebp / устанавливаем новый кадр стека
subl $20,%esp / выделяем в стеке 20 байт
pushl %edi / сохраняем %edi
pushl %esi / сохраняем %esi
pushl %ebx / сохраняем %ebx
```

эпилог

```
movl $0,%eax / возвращаем 0
popl %ebx / восстанавливаем %ebx
popl %esi / восстанавливаем %esi
popl %edi / восстанавливаем %edi
addl $20,%esp / убираем локальные переменные
movl %ebp,%esp / поднимаем %esp
pop %ebp / восстанавливаем %ebp
ret / выходим из функции
```

1.7. Системные вызовы

Системные вызовы осуществляются через программное прерывание 0x80. Аргументы системному вызову передают через регистры. Номера системных вызовов содержатся в заголовочном файле `<asm/unistd.h>`. Назначение регистров дано в табл. 4.

Регистр	Назначение
<code>%eax</code>	номер системного вызова и возвращаемое значение
<code>%ebx</code>	1-й аргумент
<code>%edx</code>	2-й аргумент
<code>%ecx</code>	3-й аргумент
<code>%edi</code>	4-й аргумент
<code>%esi</code>	5-й аргумент

Табл. 4

```
#include <asm/unistd_32.h>
.bss
.space 1024

.data
msg:
.asciz "Hello, world!0
.set msg_len, . - msg - 1

.text
.global _start
_start:
    movl $__NR_write,%eax
    movl $1,%ebx
    movl $msg,%ecx
    movl $msg_len,%edx
    int $0x80

    movl $__NR_exit,%eax
    movl $0,%ebx
    int $0x80
```

1.8. Сборка и линковка

Для сборки файлов на ассемблере лучше всего пользоваться обёрткой `gcc`, т. к. в этом случае будут обрабатываться директивы препроцессора. Для этого файл ассемблерного кода должен иметь расширение `*.S`. Если расширение будет другое, то файл препроцессором не обрабатывается.

```
$ gcc -g -m32 -c -o test.o test.S
$ ld -melf_i386 -o test test.o
```

Если в ассемблерном коде вызываются функции библиотеки Си или других библиотек, то в таком случае удобнее собирать его сразу `gcc`.

```
$ gcc -g -m32 -o test test.S -lc -lm
```

В этом случае точка входа в программу меняется со `_start` на `main`.

1.9. Отладка

Отладку построчно выполняют через `gdb`.

```
$ gdb ./test
(gdb)
```

В любом месте программы можно установить точку останова. Если нужно начать с точки входа, то точка останова ставится на адрес следующей инструкции.

```
(gdb) disassemble _start
Dump of assembler code for function _start:
0x08048074 <_start+0>: mov $0x4,%eax
0x08048079 <_start+5>: mov $0x1,%ebx
(gdb) break *(_start+5)
Breakpoint 1 at 0x08048079
```

Программу запускают и проходят пошагово, начиная с точки останова, с заходом в функции (`step`) или без (`next`). В любом месте можно прервать пошаговое выполнение и продолжить нормальное выполнение программы (`continue`). Выполнение программы можно прервать в любой момент (`kill`).

```
(gdb) run
Breakpoint 1, _start () at intro.S:15
15      movl $1,%ebx
Current language: auto; currently asm
(gdb) info line
Line 15 of "intro.S" starts at address 0x08048079 <_start+5> and
ends at 0x0804807e <_start+10>.
(gdb) next
_start () at intro.S:16
16      movl $msg,%ecx
(gdb) info registers
eax      0x4      4
ecx      0x0      0
edx      0x0      0
ebx      0x1      1
esp      0xffffd26aa0 0xffffd26aa0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
...
```

Регистры FPU или другую комбинацию регистров удобнее выводить, определив команду для `gdb`.

```
(gdb) define fpuinfo
Type commands for definition of "fpuinfo".
End with a line saying just "end".
>info registers st0 st1 st2 st3 st4 st5 st6 st7
>end
(gdb) fpuinfo
st0      0      (raw 0x00000000000000000000)
st1      0      (raw 0x00000000000000000000)
st2      0      (raw 0x00000000000000000000)
st3      0      (raw 0x00000000000000000000)
st4      0      (raw 0x00000000000000000000)
st5      0      (raw 0x00000000000000000000)
st6      0      (raw 0x00000000000000000000)
st7      0      (raw 0x00000000000000000000)
```

Определения команд, точек останова и другие часто используемые настройки удобно поместить в файл `.gdbinit` в каталоге программы.

```
$ cat .gdbinit
break *(_start+5)
define fpuinfo
info registers st0 st1 st2 st3 st4 st5 st6 st7
end
```

Для исследования адресов и содержимого по ним используют несколько команд. Команда `x` имеет следующий формат:

```
x/nfu addr
```

где `n`, `f`, `u` — опции, а `addr` — выражение, содержащее адрес.

`n` количество элементов.

`f` формат, `o`(octal), `x`(hex), `d`(decimal), `u`(unsigned decimal), `t`(binary), `f`(float), `a`(address), `i`(instruction), `c`(char), `s`(string).

`u` размер, `b`(byte), `h`(halfword), `w`(word), `g`(giant, 8 байт).

```
(gdb) print (char *) &msg
$3 = 0x8049098 "Hello, world!0
(gdb) x/s 0x8049098
0x8049098 <msg>: "Hello, world!0
(gdb) x/3xb (0x8049098+2)
0x804909a <msg+2>: 0x6c 0x6c 0x6f
```