V7/x86 Assembler Reference Manual

Robert Nordier
Nordier & Associates
www.nordier.com

The V7/x86 assembler *as* is an x86 (IA-32) assembler capable of handling both 16-bit and 32-bit mode instructions. It performs span-dependent instruction optimization, even in the presence of align statements, and features fairly full support for x86 general purpose and floating-point instructions; however, multimedia (MMX, SSE, SSE2, *etc.*) instructions are not supported.

Input syntax has been designed to be broadly compatible with existing Unix x86 assemblers (which are themselves mutually incompatible), though in some instances syntax based on that of the original Unix assembler has been preferred.

This document is intended to supplement, rather than to replace, the original Version 7 Assembler Reference Manual, and the reader is encouraged to consult that document to clarify any general points incompletely treated here.

1 Usage

The assembler is invoked using the syntax

```
as [-u] [-o object] source
```

where the "-o" option specifies the name of the object file which otherwise defaults to *a.out*. The option "-u" may be given, but is ignored, since the default behavior is to treat all undefined symbols as undefined-external.

2 Lexical conventions

Assembler tokens comprise registers, identifiers, temporary symbols, constants, string-literals, operators, and punctuators.

Blank (0x20) and tab (0x09) characters may appear between tokens, and may be required to delimit tokens; they are otherwise ignored.

Comments begin with two forward slashes (//) and extend for the rest of the line.

2.1 Registers

The following machine register names are recognised:

8-bit	**al**, **cl**, **dl**, **bl**, **ah**, **ch**, **dh**, **bh**
16-bit	**ax**, **cx**, **dx**, **bx**, **sp**, **bp**, **si**, **di**
32-bit	**eax**, **ecx**, **edx**, **ebx**, **esp**, **ebp**, **esi**, **edi**
segment	**es**, **cs**, **ss**, **ds**, **fs**, **gs**
control	**cr0** ... **cr7**
debug	**dr0** ... **dr7**
test	**tr0** ... **tr7**
FPU	**st**, **st(0)** ... **st(7)**

2.2 Identifiers

Identifiers are drawn from the set of alphanumeric characters together with "." (dot) and "_" (underscore), but may not begin with a digit. Only the first eight characters of an identifier are significant. Case is significant.

2.3 Temporary symbols

These constitute references to numeric labels, and consist of a digit followed by "b" or "f". They are further discussed in Section 5.1.

2.4 Constants

As in the language C, integer constants beginning with the sequence "0X" or "0x" are taken as hexadecimal; those beginning with"0", as octal; and those not beginning with "0", as decimal.

Character constants consist of a single character within single quotes. As in C, simple, octal, and hexadecimal escape sequences are supported. The special escape sequences are

b	backspace
f	form feed
n	newline
r	carriage return
t	horizontal tab
v	vertical tab
\ooo	(octal)
\**x***hh*	(hexadecimal)

and otherwise the sequence **'**\*c***'** has the value of the character *c*.

2.5 String-literals

String literals are sequences of characters, including escape sequences, delimited by double quotes.

2.6 Operators

Operators are discussed in Section 6.

2.7 Punctuators

Punctuators include the following characters: `$ ( ) * , : ; [ ]`. They are treated in various places throughout the manual: for example, the colon is used to indicate a label, and is covered in Section 5.1, in dealing with labels.

3 Segments

Three segments are available: *text*, *data* and *bss*, intended for code, initialized data, and uninitialized data respectively. The current segment defaults to *text*, and the current segment can be changed through the use of pseudo-operations, as discussed in Section 7.

4 The location counter

The special purpose identifier consisting of a single dot (`.`) denotes the location counter relating to the current segment.

5 Statements

Statements are separated by a semi-colon or by a newline, and may be preceded by labels.

5.1 Labels

There are two kinds of labels: *name labels* and *numeric labels*.

A *name label* consists of an identifier (name) followed by a colon. The identifier is assigned the value of the location counter (.) on the first pass, and a phase error occurs on the second pass if the value of the symbol and the location counter differ.

A *numeric label* consists of a digit followed by a colon. Such a label is referred to as [0-9][bf] where, for example, “2b” signifies the nearest numeric label with the value 2 in a backward direction, and “2f” signifies the same, but in a forward direction.

5.2 Null statements

Null statements are possible and permissible: for instance, blank lines or lines containing only labels.

5.3 Keyword statements

All non-null statements are keyword statements. Keyword statements are either pseudo-operations (see Section 7) or machine instructions (see Section 8).

6 Expressions

6.1 Expression operators

Operators are as follow:

$*$	multiplication
/	division
%	remainder
$+$	addition*
$-$	subtraction*
$<<$	left shift
$>>$	right shift
&	bitwise and
|	bitwise or
!	bitwise or not*

where an asterisk indicates operators that are both unary and binary.

Operators have equal precedence. Square brackets ([ ]) may be used for grouping.

7 Pseudo-operations

7.1 **.align** *expression* [, *expression*]

Advances the current location counter to a specified storage boundary, by outputting fill characters as required. The first expression is the desired alignment, which should be an integral power of two, and the second (optional) expression is the fill character value, which defaults to zero.

7.2 **.ascii** *"string"* [, *"string"*] . . .

Generates the characters corresponding to one or more string-literals. These are not terminated by a zero byte. See also **.asciz**.

7.3 **.asciz** *"string"* [, *"string"*] . . .

Generates the characters corresponding to one or more string-literals, adding a terminating zero to each. See also **.ascii**.

7.4 **.bss**

Sets the current segment to *bss*. See also **.data** and **.text**.

7.5 **.byte** *expression* [, *expression*] . . .

Assembles the expressions in successive bytes. See also **.long** and **.word**.

7.6 .code16

Sets the assembler to produce code for a processor operating in 16-bit mode. See also **.code32**.

7.7 .code32

Sets the assembler to produce code for a processor operating in 32-bit mode. See also **.code16**.

7.8 .comm *name*, *expression*

Declares a common symbol of specified length. See also **.lcomm**.

7.9 .data

Sets the current segment to *data*. See also **.bss** and **.text**.

7.10 .globl *name* [, *name*] ...

Makes each of the specified symbols external.

7.11 .lcomm *name*, *expression*

Declares a local common symbol of specified length. The effect is similar to the pseudo-operation **.comm** except that the symbol is not made external. See also **.comm**.

7.12 .long *expression* [, *expression*] ...

Assembles the expressions in successive (4-byte) long words. See also **.byte** and **.word**.

7.13 .set *name*, *expression*

Sets an identifier to a specified value.

7.14 .space *expression* [, *expression*]

Generates n bytes, where n is the value of the first expression, each of which have the value c, where c is the value of the second expression, or zero if no second expression is supplied.

7.15 .text

Sets the current segment to *text*. See also **.bss** and **.data**.

7.16 **.word** *expression* [, *expression*] . . .

Assembles the expressions in successive (2-byte) words. See also **.byte** and **.long**.

8 Diagnostics

Diagnostics are intended to be self-explanatory.