7 HLA v2.x Language Reference Manual

7.1 HLA Language Elements

Starting with this chapter we begin discussing the HLA source language. HLA source files must contain only seven-bit ASCII characters. These are text files with each source line record containing a carriage return/line feed (Windows) or a just a line feed (*NIX) termination sequence (HLA is actually happy with either sequence, so text files are portable between OSes without change). White space consists of spaces, tabs, and newline sequences. Generally, HLA does not appreciate other control characters in the file and may generate an error if they appear in the source file.

7.2 Comments

HLA uses "//" to lead off single line comments. It uses "/*" to begin an indefinite length comment and it uses "*/" to end an indefinite length comment. C/C++, Java, and Delphi users will be quite comfortable with this notation.

7.3 Special Symbols

The following characters are HLA lexical elements and have special meaning to HLA:

```
*  /  +  -  (  )  [  ]  {  }  <  >  :  ;  , . =  ? & | ^ ! @ !
```

The following character pairs are HLA lexical elements and also have special meaning to HLA:

```
&&  ||  <=  >=   <>  !=  ==  :=  ..  <<  >>  ##  #(  )#  #{  }#
```

7.4 Reserved Words

Here are the HLA reserved words. You may not use any of these reserved words as HLA identifiers except as noted below (with respect to the #id and #rw operators). HLA reserved words are case insensitive. That is, "MOV" and "mov" (as well as any permutation with respect to case) both represent the HLA "mov" reserved word.

#append	#asm	#closeread	#closewrite
#else	#elseif	#emit	#endasm
#endfor	#endif	#endmacro	#endmatch
#endregex	#endstring	#endtext	#endwhile
#error	#for	#id	#if
#include	#includeonce	#keyword	#linker
#macro	#match	#openread	#openwrite
#print	#regex	#return	#rw
#string	#system	#terminator	#text
#while	#write	@a	@abs
@abstract	@ae	@align	@alignstack
@arb	@arity	@at	@b
@baseptype	@basereg	@basetype	@be
@boolean	@bound	@byte	@c

@cdecl	@ceil	@char	@class
@cos	@cset	@curdir	@curlex
@curobject	@curoffset	@date	@debughla
@defined	@delete	@dim	@display
@dword	@e	@elements	@elementsize
@enter	@enumsize	@env	@eos
@eval	@exactlynchar	@exactlyncset	@exactlynichar
@exactlyntomchar	@exactlyntomcset	@exactlyntomichar	@exceptions
@exp	@external	@extract	@fast
@filename	@firstnchar	@firstncset	@firstnichar
@floor	@forward	@fpureg	@frame
@g	@ge	@global	@here
@index	@insert	@int128	@int16
@int32	@int64	@int8	@into
@isalpha	@isalphanum	@isclass	@isconst
@isdigit	@IsExternal	@isfreg	@islower
@ismem	@isreg	@isreg16	@isreg32
@isreg8	@isspace	@istype	@isupper
		@label	
@isxdigit	@l	@lastobject	@le
@leave	@length	@lex	@linenumber
@localoffset	@localsyms	@log	@log10
@lowercase	@lword	@match	@match2
@matchchar	@matchcset	@matchichar	@matchid
@matchintconst	@matchistr	@matchiword	@matchnumericconst
@matchrealconst	@matchstr	@matchstrconst	@matchtoistr
@matchtostr	@matchword	@max	@min
@mmxreg	@na	@nae	@name
@nb	@nbe	@nc	@ne
@ng	@nge	@nl	@nle
@no	@noalignstack	@nodisplay	@noenter
@noframe	@noleave	@norlesschar	@norlesscset
@norlessichar	@normorechar	@normorecset	@normoreichar
@nostackalign	@nostorage	@np	@ns
@ntomchar	@ntomcset	@ntomichar	@nz
@o	@odd	@offset	@onechar
@onecset	@oneichar	@oneormorechar	@oneormorecset
@oneormoreichar	@oneormorews	@optstrings	@p
@parmoffset	@parms	@pascal	@pclass
@pe	@peekchar	@peekcset	@peekichar
@peekistr	@peekstr	@peekws	@po
@pointer	@pos	@ptype	@qword
@random	@randomize	@read	@real128
@real32	@real64	@real80	@reg
@reg16	@reg32	@reg8	@regex
@returns	@rindex	@s	@section
@sin	@size	@sort	@sqrt
@stackalign	@staticname	@stdcall	@strbrk
@string	@strset	@strspan	@substr

@system
@tab
@tan
@tbyte
@text
@thread
@time
@tokenize
@tostring
@trace
@trim
@type
@typename
@uns128
@uns16
@uns32
@uns64
@uns8
@uppercase
@uptochar
@uptocset
@uptoichar
@uptoistr
@uptostr
@use
@volatile
@wchar
@word
@ws
@wsoreos
@wstheneos
@wstring
@xmmreg
@z
@zeroormorechar
@zeroormorecset
@zeroormoreichar
@zeroormorews
@zerooronechar
@zerooronecset
@zerooroneichar
@zstring
aaa
aad
aam
aas
abstract
adc
add
addpd
addps
addsd
addss
addsubpd
addsubps
ah
al
align
and
andnpd
andnps
andpd
andps
anyexception
arpl
ax
begin
bh
bl
boolean
bound
bp
break
breakif
bsf
bsr
bswap
bt
btc
btr
bts
bx
byte
call
case
cbw
cdq
ch
char
cl
class
clc
cld
clflush
cli
clts
cmc
cmova
cmovae
cmovb
cmovbe
cmovc
cmove
cmovg
cmovge
cmovl
cmovle
cmovna
cmovnae
cmovnb
cmovnbe
cmovnc
cmovne
cmovng
cmovnge
cmovnl
cmovnle
cmovno
cmovnp
cmovns
cmovnz
cmovo
cmovp
cmovpe
cmovpo
cmovs
cmovz
cmp
cmpeqpd
cmpeqps
cmpeqsd
cmpeqss
cmplepd
cmpleps
cmplesd
cmpless
cmpltpd
cmpltps
cmpltsd
cmpltss
cmpneqpd
cmpneqps
cmpneqsd
cmpneqss
cmpnlepd
cmpnleps
cmpnlesd
cmpnless
cmpnltpd
cmpnltps
cmpnltsd
cmpnltss
cmpordpd
cmpordps
cmpordsd
cmpordss
cmppd
cmpps
cmpsb
cmpsd
cmpss
cmpsw
cmpunordpd
cmpunordps
cmpunordsd
cmpunordss
cmpxchg
cmpxchg8b
comisd
comiss
const
continue
continueif
cpuid
cr0
cr1
cr2
cr3
cr4
cr5
cr6
cr7
cseg
cset
cvtdq2pd
cvtdq2ps
cvtpd2dq
cvtpd2pi
cvtpd2ps
cvtpi2pd
cvtpi2ps
cvtps2dq
cvtps2pd
cvtps2pi
cvtsd2si
cvtsd2ss

cvtsi2sd	cvtsi2ss	cvtss2sd	cvtss2si
cvttpd2dq	cvttpd2pi	cvttps2dq	cvttps2pi
cvttsd2si	cvttss2si	cwd	cwde
cx	daa	das	dec
default	dh	di	div
divpd	divps	divsd	divss
dl	do	downto	dr0
dr1	dr2	dr3	dr4
dr5	dr6	dr7	dseg
dup	dword	dx	dx:ax
eax	ebp	ebx	ecx
edi	edx	edx:eax	else

elseif	emms	end	endclass
			endlabel

endconst	endfor	endif	endproc
endreadonly	endrecord	endstatic	endstorage
endswitch	endtry	endtype	endunion
endval	endvar	endwhile	enter
enum	eseg	esi	esp
exception	exit	exitif	external
f2xm1	fabs	fadd	faddp
fbld	fbstp	fchs	fclex
fcmova	fcmovae	fcmovb	fcmovbe
fcmove	fcmovna	fcmovnae	fcmovnb
fcmovnbe	fcmovne	fcmovnu	fcmovu
fcom	fcomi	fcomip	fcomp
fcompp	fcos	fdecstp	fdiv
fdivp	fdivr	fdivrp	felse
ffree	fiadd	ficom	ficomp
fidiv	fidivr	fild	fimul
fincstp	finit	fist	fistp
fisttp	fisub	fisubr	fld
fld1	fldcw	fldenv	fldl2e
fldl2t	fldlg2	fldln2	fldpi
fldz	fmul	fmulp	fnclex
fninit	fnop	fnsave	fnstcw
fnstenv	fnstsw	for	foreach
forever	forward	fpatan	fprem
fprem1	fptan	frndint	frstor
fsave	fscale	fseg	fsin
fsincos	fsqrt	fst	fstcw
fstenv	fstp	fstsw	fsub
fsubp	fsubr	fsubrp	ftst
fucom	fucomi	fucomip	fucomp
fucompp	fwait	fxam	fxch
fxrstor	fxsave	fxtract	fyl2x
fyl2xp1	gseg	haddpd	haddps
hlt	hsubpd	hsubps	idiv
if	imod	imul	in

inc	inherits	insb	insd
insw	int	int128	int16
int32	int64	int8	intmul
into	invd	invlpg	iret
iretd	iterator	ja	jae
jb	jbe	jc	jcxz
je	jecxz	jf	jg
jge	jl	jle	jmp
jna	jnae	jnb	jnbe
jnc	jne	jng	jnge
jnl	jnle	jno	jnp
jns	jnz	jo	jp
jpe	jpo	js	jt
jz	label	lahf	lar
lazy	lddqu	ldmxcsr	lds
lea	leave	les	lfence
lfs	lgdt	lgs	lidt
lldt	lmsw	lock.adc	lock.add
lock.and	lock.btc	lock.btr	lock.bts
lock.cmpxchg	lock.dec	lock.inc	lock.neg
lock.not	lock.or	lock.sbb	lock.sub
lock.xadd	lock.xchg	lock.xor	lodsb
lodsd	lodsw	loop	loope
loopne	loopnz	loopz	lsl
lss	ltreg	lword	maskmovdqu
maskmovq	maxpd	maxps	maxsd
maxss	method	mfence	minpd
minps	minsd	minss	mm0
mm1	mm2	mm3	mm4
mm5	mm6	mm7	mod
monitor	mov	movapd	movaps
movd	movddup	movdq2q	movdqa
movdqu	movhlps	movhpd	movhps
movlhps	movlpd	movlps	movmskpd
movmskps	movntdq	movnti	movntpd
movntps	movntq	movq	movq2dq
movsb	movsd	movshdup	movsldup
movss	movsw	movsx	movupd
movups	movzx	mul	mulpd
mulps	mulsd	mulss	mwait
name	namespace	neg	nop
not	null	or	orpd
orps	out overloads	outsb	outsd
outsw	override	overrides	packssdw
packsswb	packuswb	paddb	paddd
paddq	paddsb	paddsw	paddusb
paddusw	paddw	pand	pandn
pause	pavgb	pavgw	pcmpeqb

pcmpeqd	pcmpeqw	pcmpgtb	pcmpgtd
pcmpgtw	pextrw	pinsrw	pmaddwd
pmaxsw	pmaxub	pminsw	pminub
pmovmskb	pmulhuw	pmulhw	pmullw
pmuludq	pointer	pop	popa
popad	popf	popfd	por
prefetchnta	prefetcht0	prefetcht1	prefetcht2
proc	procedure	program	psadbw
pshufd	pshufhw	pshuflw	pshufw
pslld	pslldq	psllq	psllw
psrad	psraw	psrld	psrldq
psrlq	psrlw	psubb	psubd
psubq	psubsb	psubsw	psubusb
psubusw	psubw	punpckhbw	punpckhdq
punpckhqdq	punpckhwd	punpcklbw	punpckldq
punpcklqdq	punpcklwd	push	pusha
pushad	pushd	pushf	pushfd
pushw	pxor	qword	raise
rcl	rcpps	rcpss	rcr
rdmsr	rdpmc	rdtsc	readonly
real128	real32	real64	real80
record	regex	rep.insb	rep.insd
rep.insw	rep.movsb	rep.movsd	rep.movsw
rep.outsb	rep.outsd	rep.outsw	rep.stosb
rep.stosd	rep.stosw	repe.cmpsb	repe.cmpsd
repe.cmpsw	repe.scasb	repe.scasd	repe.scasw
repeat	repne.cmpsb	repne.cmpsd	repne.cmpsw
repne.scasb	repne.scasd	repne.scasw	repnz.cmpsb
repnz.cmpsd	repnz.cmpsw	repnz.scasb	repnz.scasd
repnz.scasw	repz.cmpsb	repz.cmpsd	repz.cmpsw
repz.scasb	repz.scasd	repz.scasw	result
ret	returns	rol	ror
rsm	rsqrtps	rsqrtss	sahf
sal	sar	sbb	scasb
scasd	scasw	segment	seta
setae	setb	setbe	setc
sete	setg	setge	setl
setle	setna	setnae	setnb
setnbe	setnc	setne	setng
setnge	setnl	setnle	setno
setnp	setns	setnz	seto
setp	setpe	setpo	sets
setz	sfence	sgdt	shl
shld	shr	shrd	shufpd
shufps	si	sidt	sldt
smsw	sp	sqrtpd	sqrtps
sqrtsd	sqrtss	sseg	st0
st1	st2	st3	st4
st5	st6	st7	static

stc	std	sti	stmxcsr
storage	stosb	stosd	stosw
streg	string	sub	subpd
subps	subsd	subss	switch
sysenter	sysexit	tbyte	test
text	then	this	thunk
to	try	type	ucomisd
ucomiss	ud2	union	unit
unpckhpd	unpckhps	unpcklpd	unpcklps
unprotected	uns128	uns16	uns32
uns64	uns8	until	val
valres	var	verr	verw
vmt	wait	wbinvd	wchar
welse	while	word	wrmsr
wstring	xadd	xchg	xlat
xmm0	xmm1	xmm2	xmm3
xmm4	xmm5	xmm6	xmm7
xor	xorpd	xorps	zstring

Note that **@debughla** is also a reserved compiler symbol. However, this is intended for internal (HLA) debugging purposes only. When the compiler encounters this symbol, it immediately stops the compiler with an assertion failure. Obviously, you should never put this statement in your source code unless you're debugging HLA and you want to stop the compiler immediately after the compilation of some statement.

Because the set of HLA reserved words is changing frequently, a special feature was added to HLA to allow a programmer to "disable" HLA reserved words. This may allow an older program that uses new HLA reserved words as identifiers to continue working with only minor modifications to the HLA source code. The ability to disable certain HLA reserved words also allows you to create macros that override certain machine instructions.

All HLA reserved words take two forms: the standard, mutable, form (appearing in the table above) and a special immutable form that consists of a tilde character ('~') followed by the reserved word. For example, 'mov' is the mutable form of the move instruction while '~mov' is the immutable form. By default, the immutable and mutable forms are equivalent when you begin an assembly. However, you can use the **#id** compile-time statement to convert the mutable form to an identifier and you can use the **#rw** compile-time statement to turn it back into a reserved word. Regardless of the state of the mutable form, the immutable form always behaves like the reserved word as far as HLA is concerned. Here's an example of the **#id** and **#rw** statements:

```
#id( mov )  //From this point forward, mov is an identifier, not a
reserved word
mov:
    ~mov( i, eax );  // Must use ~mov while mov is a reserved word!
    cmp( eax, 0 );
    jne mov;
#rw( mov )  // Okay, now mov is a reserved word again.
    mov( 0, eax );
```

Note that use can use the **#id** facility to disable certain instructions. For example, by default HLA handles almost all (32-bit flat model) instructions up through the latest Intel processors. If you want to write code for an earlier processor, you may want to disable instructions available only on later processors to help avoid their use. You can do this by placing the offending instructions in **#id** statements.

The **#rw** statement will not turn an arbitrary identifier into a reserved word. It will only revert a reserved word that was previously converted to an identifier back into a reserved word.

One use of the **#id** statement is to change the syntax of existing HLA instructions. For example, some x86 programmers are completely incapable of handling HLA's (and Gas') "source, dest" syntax and insist on using the original Intel "dest, source" syntax. This isn't a good reason for giving up on HLA because you can easily override HLA's syntax by using the **#id** statement and a set of macros. Consider the following example for the **mov** instruction:

```
#id( mov )
#macro mov( dest, source );
   ~mov( source, dest )
#endmacro
```

By creating an include file (let's calling "intel.hhf") with all the appropriate macros and **#id** statements, you can easily change HLA's syntax to take on a more "Intel" feel.

7.5 External Symbols and Assembler Reserved Words

HLA v2.x, in addition to directly producing object code, offers the option of producing an assembly language file during compilation and invoking an assembler such as MASM, FASM, NASM, or Gas to complete the compilation process. HLA automatically translates normal identifiers you declare in your program to benign identifiers in the assembly language program (in HLA v2.2 these identifiers typically took the form *original_name*__hla_**xxxx** where *original_name* is the original symbol and *xxxx* is a unique four-digit hexadecimal value). However, HLA does not translate **external** symbols, but preserves these names in the assembly language file it produces. Therefore, you must take care not to use external names that conflict with the underlying assembler's set of reserved words or that assembler will generate an error when it attempts to process HLA's output. Obviously, this is not an issue when directly producing object code with HLA (rather than producing an assembly language source file to be assembled by some other assembler).

For a list of assembler reserved words, please see the documentation for the back-end assembler you are using.

7.6 HLA Identifiers

HLA identifiers must begin with an alphabetic character or an underscore. After the first character, the identifier may contain alphanumeric and underscore symbols. There is no technical limit on identifier length in HLA, but you should avoid external symbols greater than about 32 characters in length since the assembler and linkers that process HLA identifiers may not be able to handle such symbols. Also note that if you are generating assembly language source output files, HLA may add some additional characters to the identifiers you use (typically something like "__HLA_*xxxx*" where "*xxxx*" is a 4-digit hexadecimal number) in order to prevent conflicts with the assembler's own reserved word set. As such, you may want to limit yourself to about 20-22 characters if you're using a back-end assembler that has limited identifier lengths.

HLA identifiers are always *case neutral*. This means that identifiers are case sensitive insofar as you must always spell an identifier exactly the same way (with respect to alphabetic case). However, you are not allowed to declare two identifiers whose only difference is alphabetic case.

Although technically legal in your program, do not use identifiers that begin and end with a single underscore. HLA reserves such identifiers for use by the compiler and the HLA standard library. If you declare such identifiers in your program, the possibility exists that you may interfere with HLA's or the HLA Standard Library's use of such a symbol.

By convention, HLA programmers use symbols beginning with two underscores to represent private fields in a class. Therefore, you should avoid such identifiers except when defining such private fields in your own classes.

7.7 External Identifiers

HLA lets you explicitly provide a string for external identifiers. External identifiers are not limited to the format for HLA identifiers. HLA allows any string constant to be used for an external identifier. If you're using a back-end assembler, it is your responsibility to use only those characters that are legal in that assembler. Note that this feature lets you use symbols that are not legal in HLA but are legal in external code (e.g., Win32 APIs use the ’@’ character in identifiers and some non-HLA code may use HLA reserved words as identifiers). See the discussion of the **external** option in the chapters on *HLA Program Structure* and *HLA Procedures* for more details.

7.8 HLA Literal Constants

HLA supports literal numeric, string, character, character set, Boolean, array, record, and union constants. For more details on these HLA language elements, please see the chapters on *HLA Constants and Constant Expressions* and *HLA Data Types*.