












spaces		examples
<code> </code>		
<code> </code>		

comment		examples
<code>//_comment</code>		
<code>//_comment</code> ␣		
<code>//</code>		
<code>//</code> ␣		

comma		examples
<code>,</code>		
<code> ,</code>		

lb		examples
<code> </code> ␣ <code> </code> ␣ <code> </code>		

linebreak		examples
<code> </code> ␣ ␣		
<code>//_comment</code> ␣ ␣ <code>//_eh</code>		

linebreak		counterexamples
<code> </code> <code> </code> x		<p>Parsing error: The parser did not consume the entire input.</p> <pre> 1 ␣ ~ Surplus characters on next line Valid <linebreak> Hint: halted due to the following: 2 x ^ Can't match string " "</pre> <p>While trying to parse: <code><linebreak></code> → <code><lb></code>.</p>

linebreak		counterexamples
<pre>//_comment␣ x</pre>	■	<p>Parsing error: The parser did not consume the entire input.</p> <pre>1 //_comment␣ ~~~~~~ Surplus characters on next line Valid <linebreak></pre> <p>Hint: halted due to the following:</p> <pre>2 x ^ Can't match string "</pre> <p>While trying to parse: <linebreak> → <lb>.</p>
<pre>//_comment␣ ␣ x</pre>	■	<p>Parsing error: The parser did not consume the entire input.</p> <pre>2 ␣ ~ Surplus characters on next line Valid <linebreak></pre> <p>Hint: halted due to the following:</p> <pre>3 x ^ Can't match string "</pre> <p>While trying to parse: <linebreak> → <lb>.</p>
<pre>//_comment␣ x</pre>	■	<p>Parsing error: The parser did not consume the entire input.</p> <pre>1 //_comment␣ ~~~~~~ Surplus characters on next line Valid <linebreak></pre> <p>Hint: halted due to the following:</p> <pre>2 x ^ Can't match string "</pre> <p>While trying to parse: <linebreak> → <lb>.</p>

ident-startchar		examples
<pre>f</pre>	■	f
<pre>A</pre>	■	A

ident-startchar		counterexamples
<pre>_</pre>	■	<p>Parsing error: The input does not match the expected format.</p> <pre>1 _ ^ Regex does not match</pre> <p>While trying to parse: <ident-startchar>.</p>

ident-anychar		examples
<pre>A</pre>	■	A
<pre>0</pre>	■	0

ident-anychar		examples
x	■	x
_	■	_

ident		examples
LD_foo	■	(lab: "LD_foo")
main	■	(lab: "main")
loop0	■	(lab: "loop0")

ident		counterexamples
0var	■	<p>Parsing error: The input does not match the expected format.</p> <pre>1 0var ^ Regex does not match</pre> <p>While trying to parse: <ident> → <ident-startchar>.</p>

size		examples
.byte	■	(size: 1)
.word	■	(size: 4)
.hword	■	(size: 2)

size		counterexamples
.bte	■	<p>Parsing error: The input does not match the expected format.</p> <pre>1 .bte ^ Can't match string "word"</pre> <p>While trying to parse: <size>.</p>

hexvalue		examples
0x45	■	(hex: "45")
0xdead	■	(hex: "dead")
0xffffffff	■	(hex: "ffffffff")

hexvalue		counterexamples
0xz	■	<p>Parsing error: The input does not match the expected format.</p> <pre>1 0xz ^ Regex does not match</pre> <p>While trying to parse: <hexvalue>.</p>

decvalue		examples
42	■	(int: "42")
1000	■	(int: "1000")
0	■	(int: "0")


decvalue		counterexamples
45x	■	<p>Parsing error: The parser did not consume the entire input.</p> <pre> 1 45x ~ ^ Surplus characters Valid <decvalue> Hint: halted due to the following: 1 45x ^ Regex does not match While trying to parse: <decvalue>.</pre>


value		examples
0x44	■	(hex: "44")
420	■	(int: "420")
mask	■	(lab: "mask")



value		counterexamples
0var	■	<p>Parsing error: The parser did not consume the entire input.</p> <pre> 1 0var ~ ^^^ Surplus characters Valid <value> Hint: halted due to the following: 1 0var ^ Regex does not match While trying to parse: <value> → <decvalue>.</pre>



concrete-value		examples
.byte_0x42	■	(size: 1, hex: "42")
.word_mask	■	(size: 4, lab: "mask")



concrete-value		counterexamples
.bte_0x42	■	<p>Parsing error: The input does not match the expected format.</p> <pre> 1 .bte_0x42 ^ Can't match string "word" While trying to parse: <concrete-value> → <size>.</pre>

concrete-value		counterexamples
<code>.byte_45x</code>		<p>Parsing error: The parser did not consume the entire input.</p> <pre> 1 .byte_45x ~~~~~ ^ Surplus characters Valid <concrete-value> </pre> <p>Hint: halted due to the following:</p> <pre> 1 .byte_45x ^ Regex does not match </pre> <p>While trying to parse: <concrete-value> → <value> → <decvalue>.</p>

abstract-value		examples
<code>.skip_5</code>		(size: 5)

abstract-value		counterexamples
<code>skip_3</code>		<p>Parsing error: The input does not match the expected format.</p> <pre> 1 skip_3 ^ Can't match string ".skip" </pre> <p>While trying to parse: <abstract-value>.</p>
<code>.skip_0x42</code>		<p>Parsing error: The parser did not consume the entire input.</p> <pre> 1 .skip_0x42 ~~~~~ ^ Surplus characters Valid <abstract-value> </pre> <p>Hint: halted due to the following:</p> <pre> 1 .skip_0x42 ^ Regex does not match </pre> <p>While trying to parse: <abstract-value> → <decvalue>.</p>

data-value		examples
<code>.byte_0x42</code>		(size: 1, hex: "42")
<code>.skip_8</code>		(size: 8)

data-label		examples
<code>a:</code>		(lab: "a")
<code>B:</code>		(lab: "B")

data-contents		examples
<pre> A: B: .word_0x64 .byte_0x42 foo: .hword_42 bar: baz: .byte_1 .byte_2 .byte_3 </pre>		<pre> ((lab: "A"), (lab: "B"), (size: 4, hex: "64"), (size: 1, hex: "42"), (lab: "foo"), (size: 2, int: "42"), (lab: "bar"), (lab: "baz"), (size: 1, int: "1"), (size: 1, int: "2"), (size: 1, int: "3"),) </pre>

data-contents		counterexamples
<pre> .byte_1_c </pre>		<p>Parsing error: The parser did not consume the entire input.</p> <pre> 1 .byte_1_c ~~~~~ ^ Surplus characters Valid <data-contents> Hint: halted due to the following: 1 .byte_1_c ^ Can't match string ":" While trying to parse: <data-contents> → <data-label>. </pre>

data-section		examples
<pre> .data foo: .word_0x42 bar: .word_0x43 </pre>		<pre> (data: ((lab: "foo"), (size: 4, hex: "42"), (lab: "bar"), (size: 4, hex: "43"),),) </pre>

instr-code		examples
ldr		(instr: "ldr", size: 4)
add		(instr: "add")
lsl		(instr: "lsl")
ldrh		(instr: "ldr", size: 2)
b		(instr: "b")

register-number		examples
r0		(reg: 0)

register-number		examples
r9	■	(reg: 9)
r12	■	(reg: 12)

register-alias		examples
sp	■	(reg: 13)
lr	■	(reg: 14)
pc	■	(reg: 15)

register		examples
r8	■	(reg: 8)
sp	■	(reg: 13)
lr	■	(reg: 14)
r0	■	(reg: 0)

constant		examples
#4	■	(int: "4")
#0x1	■	(hex: "1")

deref-offset		examples
,_r1	■	(reg: 1)
,_#3	■	(int: "3")

deref-reg		examples
[r1]	■	(deref: ((reg: 1),))
[_r2_]	■	(deref: ((reg: 2),))
[r1,_r2]	■	(deref: ((reg: 1), (reg: 2)))
[r1,_#1]	■	(deref: ((reg: 1), (int: "1")))
[r1,_r2,_r3]	■	(deref: ((reg: 1), (reg: 2), (reg: 3)))
[r1,_r2,_#2]	■	(deref: ((reg: 1), (reg: 2), (int: "2")))

local-label		examples
.LD_foo	■	(lab: ".LD_foo")

operand		examples
<code>lr</code>	■	(reg: 14)
<code>#1</code>	■	(int: "1")
<code>[r1, #2]</code>	■	(deref: ((reg: 1), (int: "2")))
<code>.LD_xx</code>	■	(lab: ".LD_xx")
<code>=mask</code>	■	(eq: "mask")
<code>loop0</code>	■	(lab: "loop0")

operands		examples
<code>lr, #1</code>	■	((reg: 14), (int: "1"))
<code>r0, r0, [r1, #2]</code>	■	((reg: 0), (reg: 0), (deref: ((reg: 1), (int: "2"))),)

instruction		examples
<code>ldr_r0, [r1]</code>	■	(instr: "ldr", size: 4, ops: ((reg: 0), (deref: ((reg: 1),))),)
<code>add_r0, r1, r2</code>	■	(instr: "add", ops: ((reg: 0), (reg: 1), (reg: 2)))
<code>sub_r1, #1</code>	■	(instr: "sub", ops: ((reg: 1), (int: "1")))
<code>lsl_r1, #8</code>	■	(instr: "lsl", ops: ((reg: 1), (int: "8")))

inline-data		examples
<code>.LD_xx: .word x</code>	■	(lab: "x", size: 4)
<code>.LD_xx: .byte 0x42</code>	■	(lab: ".LD_xx", size: 1, hex: "42")

inline-label		examples
<code>main:</code>	■	(tag: "main")

print-width		examples
<code>/8</code>	■	8
<code>/16</code>	■	16

register-slice		examples
<code>[:]</code>	■	(start: <code>auto</code> , len: <code>auto</code>)
<code>[1:]</code>	■	(start: (int: <code>"1"</code>), len: <code>auto</code>)
<code>[16:8]</code>	■	(start: (int: <code>"16"</code>), len: (int: <code>"8"</code>))

print-register		examples
<code>r0</code>	■	(reg: <code>0</code>)
<code>r0[:8]</code>	■	(reg: <code>0</code> , start: <code>auto</code> , len: (int: <code>"8"</code>))
<code>r0[16:32]</code>	■	(reg: <code>0</code> , start: (int: <code>"16"</code>), len: (int: <code>"32"</code>))


print-list		examples
<code>r0, r1, r2</code>	■	((reg: <code>0</code>), (reg: <code>1</code>), (reg: <code>2</code>))



print-directive		examples
<code>print/8: r0</code>	■	(print: (width: <code>8</code> , regs: ((reg: <code>0</code>),)))
<code>print: r1, r2</code>	■	(print: (width: <code>auto</code> , regs: ((reg: <code>1</code>), (reg: <code>2</code>))))

directive		examples
<code>@_print/8: r0</code>	■	(print: (width: <code>8</code> , regs: ((reg: <code>0</code>),)))

text-contents		examples
<pre>main: ldr r0, [r1] add r0, #1 test loop0: eor r0, r1, r2 .LD_data: .word data</pre>	■	<pre>((tag: "main"), (instr: "ldr", size: 4, ops: ((reg: 0), (deref: ((reg: 1),))),), (instr: "add", ops: ((reg: 0), (int: "1"))), (tag: "loop0"), (instr: "eor", ops: ((reg: 0), (reg: 1), (reg: 2))), (lab: "data", size: 4),)</pre>

text-section		examples
<pre>.text main: mov r0, #1 test add r0, r0, r0</pre>	■	<pre>(text: ((tag: "main"), (instr: "mov", ops: ((reg: 0), (int: "1"))), (instr: "add", ops: ((reg: 0), (reg: 0), (reg: 0))),),)</pre>

arm		examples
<pre> .data A: .byte 0x64 .word 0x95 .hword 45 ␣ ␣ .text main: // main_function ldr r0, .LD_A ldr r0, [r6] ldrb r0, [r6] ldrh r0, [r5, #1] // yay mov r5, r6 mvn r5, r6 mov r5, #0x1 add r5, r6, #1 ␣ lsl r5, #1 lsr r5, #1 // do_some_stuff // testing_comments eor r3, r4, r5 orr r3, r4, r5 and r3, r4, r5 ␣ ␣ .LD_A: .word A ␣ </pre>		<pre> (data: ((lab: "A"), (size: 1, hex: "64"), (size: 4, hex: "95"), (size: 2, int: "45"),), text: ((tag: "main"), (instr: "ldr", size: 4, ops: ((reg: 0), (lab: ".LD_A")),), (instr: "ldr", size: 4, ops: ((reg: 0), (deref: ((reg: 6)))),), (instr: "ldr", size: 1, ops: ((reg: 0), (deref: ((reg: 6)))),), (instr: "ldr", size: 2, ops: ((reg: 0), (deref: ((reg: 5), (int: "1")))),), (instr: "mov", ops: ((reg: 5), (reg: 6))), (instr: "mvn", ops: ((reg: 5), (reg: 6))), (instr: "mov", ops: ((reg: 5), (hex: "1"))), (instr: "add", ops: ((reg: 5), (reg: 6), (int: "1")),), (instr: "lsl", ops: ((reg: 5), (int: "1"))), (instr: "lsr", ops: ((reg: 5), (int: "1"))), (instr: "eor", ops: ((reg: 3), (reg: 4), (reg: 5))), (instr: "orr", ops: ((reg: 3), (reg: 4), (reg: 5))), (instr: "and", ops: ((reg: 3), (reg: 4), (reg: 5))), (lab: "A", size: 4),),) </pre>

parsing		
112	112	0