




















spaces		examples
		
		



comment		examples
		
		
		
		


comma		examples
		
		





linebreak		examples
		
 		




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


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




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


ident-startchar		counterexamples
<pre>_</pre>		<p><b>Parsing error:</b> The input does not match the expected format.</p> <pre>1    _    ^ Character is out of range</pre> <p>While trying to parse: &lt;ident-startchar&gt;.</p>




ident-anychar		examples
<pre>A</pre>		A
<pre>0</pre>		0
<pre>x</pre>		x
<pre>_</pre>		_


ident		examples
<pre>LD_foo</pre>		(lab: "LD_foo")
<pre>main</pre>		(lab: "main")
<pre>loop0</pre>		(lab: "loop0")




ident		counterexamples
<code>0var</code>		<b>Parsing error:</b> The input does not match the expected format. 1   0var ^ Character is out of range While trying to parse: <ident> → <ident-startchar>.


size		examples
<code>.byte</code>		(size: 1)
<code>.word</code>		(size: 4)
<code>.hword</code>		(size: 2)




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


hexvalue		examples
<code>0x45</code>		(hex: "45")
<code>0xdead</code>		(hex: "dead")
<code>0xffffffff</code>		(hex: "ffffffff")



hexvalue		counterexamples
<code>0xz</code>		<b>Parsing error:</b> The input does not match the expected format. 1   0xz ^ Character is out of range While trying to parse: <hexvalue>.



decvalue		examples
<code>42</code>		(int: "42")
<code>1000</code>		(int: "1000")
<code>0</code>		(int: "0")


decvalue		counterexamples
45x		<b>Parsing error:</b> The parser did not consume the entire input. 1   45x ~ ^ Surplus characters   Valid <decvalue> Hint: halted due to the following: 1   45x ^ Character is out of range While trying to parse: <decvalue>.



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



value		counterexamples
0var		<b>Parsing error:</b> The parser did not consume the entire input. 1   0var ~ ^^^ Surplus characters   Valid <value> Hint: halted due to the following: 1   0var ^ Character is out of range While trying to parse: <value> → <decvalue>.



concrete-value		examples
.byte 0x42		(size: 1, hex: "42")
.word mask		(size: 4, lab: "mask")


concrete-value		counterexamples
.bte 0x42		<b>Parsing error:</b> The input does not match the expected format. 1   .bte_0x42 ^ Can't match string "word" While trying to parse: <concrete-value> → <size>.
.byte 45x		<b>Parsing error:</b> The parser did not consume the entire input. 1   .byte_45x ~~~~~ ^ Surplus characters   Valid <concrete-value> Hint: halted due to the following: 1   .byte_45x ^ Character is out of range While trying to parse: <concrete-value> → <value> → <decvalue>.

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

abstract-value		counterexamples
<code>skip 3</code>		<b>Parsing error:</b> The input does not match the expected format. <pre>1   skip_3     ^ Can't match string ".skip"</pre> While trying to parse: <abstract-value>.
<code>.skip 0x42</code>		<b>Parsing error:</b> The parser did not consume the entire input. <pre>1   .skip_0x42     ~~~~~ ^^^ Surplus characters             Valid &lt;abstract-value&gt;</pre> Hint: halted due to the following: <pre>1   .skip_0x42     ^ Character is out of range</pre> While trying to parse: <abstract-value> → <decvalue>.

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
<code>.byte 1 c</code>	■	<b>Parsing error:</b> The parser did not consume the entire input. <pre>1   .byte_1_c       ~~~~~ ^ Surplus characters             Valid &lt;data-contents&gt;</pre> Hint: halted due to the following: <pre>1   .byte_1_c       ^ Can't match string ":"</pre> While trying to parse: <data-contents> → <data-label>.

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
<code>ldr</code>	■	(instr: "ldr", size: 4)
<code>add</code>	■	(instr: "add")
<code>lsl</code>	■	(instr: "lsl")
<code>ldrh</code>	■	(instr: "ldr", size: 2)
<code>b</code>	■	(instr: "b")

register-number		examples
<code>r0</code>	■	(reg: 0)
<code>r9</code>	■	(reg: 9)
<code>r12</code>	■	(reg: 12)

register-alias		examples
<code>sp</code>	■	(reg: 13)
<code>lr</code>	■	(reg: 14)
<code>pc</code>	■	(reg: 15)

register		examples
<code>r8</code>	■	(reg: 8)

register		examples
<code>sp</code>	■	(reg: 13)
<code>lr</code>	■	(reg: 14)
<code>r0</code>	■	(reg: 0)

constant		examples
<code>#4</code>	■	((int: "4"),)
<code>#0x1</code>	■	((hex: "1"),)

deref-offset		examples
<code>, r1</code>	■	((reg: 1),)
<code>, #3</code>	■	((int: "3"),,)

deref-reg		examples
<code>[r1]</code>	■	(deref: ((reg: 1),))
<code>[ r2 ]</code>	■	(deref: ((reg: 2),))
<code>[r1, r2]</code>	■	(deref: ((reg: 1), (reg: 2)))
<code>[r1 , #1]</code>	■	(deref: ((reg: 1), (int: "1")))
<code>[r1, r2, r3]</code>	■	(deref: ((reg: 1), (reg: 2), (reg: 3)))
<code>[r1, r2, #2]</code>	■	(deref: ((reg: 1), (reg: 2), (int: "2")))

local-label		examples
<code>.LD_foo</code>	■	(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: auto, len: auto)
<code>[1:]</code>	■	(start: (int: "1"), len: auto)
<code>[16:8]</code>	■	(start: (int: "16"), len: (int: "8"))

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



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


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



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

directive		examples
<code>@ print/8: r0</code>	■	<code>((print: (width: 8, regs: ((reg: 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]     ldrrh 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>

total	111
	111
	0