	|	**bars make cores**
|_	produces a door (a core with sample)	
|%	produces a core (battery and payload)	
|@	produces a wet core (battery and payload)	
|.	produces a trap (a core with one arm)	
|:	produces a gate with a custom sample	
|-	produces a trap (a core with one arm) and evaluates it	
|^	produces a core whose battery includes a $ arm and computes the latter	
|~	produces an iron gate	
|*	produces a wet gate (a one-armed core with sample)	
|=	produces a dry gate (a one-armed core with sample)	
|?	produces a lead trap	
|$	produces a mold	
	$	**bucs form molds**
$@	structure that normalizes a union tagged by head atom	
$:	forms a cell type	`[a=foo b=bar c=baz]`
$_	structure that normalizes to an example	`_foo`
$%	structure that recognizes a union tagged by head atom	
$*	bunt (irregular form is *)	
$^	structure that normalizes a union tagged by head depth (cell)	
$~	defines a custom type default value	
$-	structure that normalizes to an example gate	
$=	structure that wraps a face around another structure	`foo=bar`
$?	forms a type from a union of other types	`?($foo $bar $baz)`
	%	**cens put the fun in function**
%_	resolves a wing with changes, preserving type	
%.	calls a gate, inverted	
%^	calls a gate with triple sample	
%+	calls a gate with a cell sample	
%-	calls a gate	`(fun arg)`
%:	calls a gate with many arguments	
%~	evaluates an arm in a door	`~(arm core arg)`
%*	evaluates an expression, then resolves a wing with changes	
%=	resolves a wing with changes	`foo(x 1, y 2, z 3)`
	:	**cols make cells**
:_	constructs a cell, inverted	
:^	constructs a cell, 4-tuple	`[a b c d]`
:+	constructs a cell, 3-tuple	`[a b c]`
:-	constructs a cell, 2-tuple	`[a b]`, `a^b`
:~	constructs a null-terminated list	`~[a b c]`
:*	constructs an n-tuple	`[a b c d e …]`
::	marks a comment	
	.	**dots nock**
.+	increments an atom using Nock 4	`+(42)`
.*	evaluates using Nock 2	
.=	tests for equality using Nock 5	`=(a b)`
.?	tests for cell or atom using Nock 3	
.^	loads from namespace using Nock 12	

	^ kets cast	
^|	converts a gold core to an iron core (invariant)	
^.	typecasts on value	
^-	typecasts by explicit type label	`` `foo`bar ``
^+	typecasts by inferred type	
^&	converts a core to a zinc core (covariant)	
^~	folds constant at compile time	
^=	binds name to a value	`foo=bar`
^?	converts a core to a lead core (bivariant)	
^*	produces example type value	
^:	produces a 'factory' gate for a type	
	~ sigs hint	
~|	prints in stack trace if failure	
~$	profiler hit counter	
~_	prints in stack trace, user-formatted	
~%	registers jet	
~/	registers jet with registered context	
~<	raw hint, applied to product ("backward")	
~>	raw hint, applied to computation ("forward")	
~+	caches a computation	
~&	prints (used for debugging)	
~?	prints conditionally (used for debugging)	
~=	detects duplicate	
~!	prints type if compilation failure	
	; mics make	
;:	calls a binary function as an n-ary function	`:(fun a b c d)`
;<	glues a pipeline together (monadic bind)	
;~	glues a pipeline together with a product-sample adapter (monadic bind)	
;;	normalizes with a mold, asserting fixpoint	
;+	(Sail) makes a single XML node	
;*	(Sail) makes a list of XML nodes from Hoon expression	
;=	(Sail) makes a list of XML nodes	
;/	(Sail) yields tape as XML element	
	= tises alter	
=|	combines default type value with the subject	
=.	changes one leg in the subject	
=?	changes one leg in the subject conditionally	
=^	pins the head of a pair; changes a leg with the tail	
=:	changes multiple legs in the subject	
=/	combines a named noun with the subject	
=;	combines a named noun with the subject, inverted	
=<	composes two expressions, inverted	`foo:bar`
=>	composes two expressions	
=-	combines a new noun with the subject	
=*	defines an alias	
=,	exposes namespace	
=+	combines a new noun with the subject	
=~	composes many expressions	

?	**wuts test**	
?|	logical OR	`|(foo bar baz)`
?:	branches on a boolean test	
?.	branches on a boolean test, inverted	
?<	negative assertion	
?>	positive assertion	
?-	switches against a union, no default	
?^	branches on whether a wing of the subject is a cell	
?=	tests pattern match	
?#	tests pattern match	
?+	switches against a union, with default	
?&	logical AND	`&(foo bar baz)`
?@	branches on whether a wing of the subject is an atom	
?~	branches on whether a wing of the subject is null	
?!	logical NOT	`!foo`
!	**zaps wild**	
!:	turns on stack trace	
!.	turns off stack trace	
!,	emits AST of expression	
!;	emits the type for an expression using the type of type	
!>	wraps a noun in its type	
!=	makes the Nock formula for a Hoon expression	
!?	restricts Hoon version	
!!	crashes	
!<	lift dynamic value into static context	
/	**fases ford**	
/$	slams a gate on extra arguments	
/|	takes a series of horns and produces the first one (L-to-R) that succeeds; if none succeed, produces stack traces from arguments	
/=	runs a horn (usually produced by another Ford rune), takes the result of that horn, and wraps a face around it	
/.	produces a null-terminated list from a sequence of horns, terminated by ==	
/,	acts as switch statement, picking a branch to evaluate based on whether the current path matches the path in the switch statement	
/&	pass a horn through multiple marks	
/_	*unfiltered*: takes a horn, producing new horn mapping supplied horn over list of files in current directory; *filtered*: runs a horn on each file matching aura	
/~	produces a horn that evaluates a twig and places the product in the subject	
/:	takes a path and a horn, and evaluates the horn with the current path set to the supplied path	
/^	takes a mold and a horn, and casts the result of the horn to the mold	
/!	produces a mark	
/+	accepts a filename and loads that filename from the `lib` directory	
/-	accepts a filename and loads that filename from the `sur` directory	
//	parses relative path as a hoon twig, and adds the resulting twig to the subject	
/;	takes a twig and a horn; the twig should evaluate to a gate, which is then slammed with the result of the horn as its sample	

/# takes a horn and produces a cell of the dependency hash of the result of the horn, and the result itself
/% forwards extra arguments to enclosed renderers
-/= **terminators terminate**
-- terminates core expression
== terminates running series of Hoon expressions
+ **luses change**
+| labels a chapter
+$ produces a structure arm (type definition)
++ produces a (normal) arm
+* produces a type constructor arm

