2006-08-16
Recursion Co., Ltd.
Osamu Tamura

ForCy Grammar

ForCy is a program language for 16 bit virtual stack machine. The program is organized with Word, Variable, Constant and Comment. They are programmed in RPN manner (postfix notation).

class	category	element	example
Word	system defined	string	dup +
	user defined	string	:main
Variable	variable	unsigned 16 bit integer	;x ;y ;count
	array	1-D array of variables	[10]a [0x20]b
Constant	number	unsigned 16 bit integer	34 0x1f 'Q'
	string	byte-string (null-terminated)	"string" "¥x7f¥xff"

Word	... :name	define the start of a subroutine
register	... self ... :name	recursive call
evaluate		execute from the last registered word
Variable		
declare	;name	reserve memory for a variable
refer	name	push the value of variable onto the data stack
substitute	=name	substitute the top for variable
Array		
declare	[size]name	reserve memory for array[size]
refer	index name	push name[index] onto the data stack
substitute	index =name	substitute the top for name[index]
String	"string"	string start position is set to string pointer.
	index @s	push an indexed character of string onto the top.
Comment	/* */	ignore the section
	//	ignore until the EOL
	string,	ignore the *string*

* Delimiters are space, tab, CR and LF.
* The length of word, variable and array should not exceed 15 characters.
* Execution is performed from the last defined word. The name is arbitrary.
* The length of string should not exceed 255 characters.
* The length of comment has no limits. String before ‘,’ should not exceed 14 characters.

Flow control	{ ... } do	iteration
	{ ... *condition* while ... } do	exit if false
	{ ... *condition* break ... } do	exit if true
	{ ... *condition* continue ... } do	restart {} block if true
	{ ... *update_index* ... } limit index for	repeat while index!=limit
	limit index { ... *update_index* ... } for	(*update_index*：e.g. ++,--)
	condition { ... } if	perform {} block if true.
	condition { ... } { ... } ifelse	perform left {} block if true, right {} block if false.
	compare_value { value1 { ... } case value2 { ... } case *default* } switch	perform {} block if value matches otherwise, perform *default*.
	return	return from the current word
	index { w0 w1 ... wk } of	perform indexed word
	{ w0 w1 ... wk } interrupt	set interrupt vector table w0: word called every 2mS w1: word called every 1Sec

* All the elements in ‘of’ and ‘interrupt’ should be user-defined words.

for

* Drop the index and the limit after the iteration.

* Update the index before ‘continue’.

```
   n 0 {
        ...        // update index here
        condition  continue
        ++       // skips if condition is true!
    } for ..
```

* Copy the index when using in loop.

```
    ;i
    10 0 {
        dup =i
        ...
        ++
    } for ..
```

* When the index jumps over the limit,

```
    10 0 {
        ...
        3 +
        10 min
    } for ..
```

Otherwise, use ‘while’.

if, ifelse

* *condition* can be placed after {} block.

* Can’t exit (while, break) nor restart (continue) directly under if/ifelse {} block.

switch

* Can’t exit (while, break) nor restart (continue) directly under switch {}.

* Drop matching value after ‘switch’.

return

* Programmer ought to manage the data stack.

Relational operator	==	(u1 u2 – u1 flag)
	!=	remove ONE element from the stack and push the result.
	<	
	<=	
	>	
	>=	
Unary operator	++	(u1 -- u2)
	--	remove one element from the stack and push the result.
	~	
	!	
Binomial operator	+	(u1 u2 -- u3)
	-	remove two elements from the stack and push the result.
	*	
	/	
	%	
	&	
	|	
	^	
	&&	
	||	
	<<	
	>>	
	min	
	max	

* 1st value remains after comparing two values. Drop if it is not necessary.

```
x y < nip { ... } if
x y < { ... } if .
```

* x 0 >= always returns true, since value is unsigned.

Stack	.	(X --)
Operation		drop X from the stack. Forth:drop
	..	(X1 X0 --)
		drop 2 elements from the stack. Forth:2drop
	@	(Xu .. X0 u -- Xu .. X0 Xu)
		copy the ‘u’th element deep in the stack to the top.
	dup	(X -- X X)
		duplicate X, returning a second copy of it on the stack.
		(equivalent to ‘ 0 @ ‘)
	over	(X1 X0 -- X1 X0 X1)
		push a copy of the second element on the stack.
		(equivalent to ‘ 1 @ ‘)
	=	(Xu+1 .. X1 X0 u -- X0 .. X1)
		insert a copy of the top to the ‘u’th element deep in the stack.
	nip	(X1 X0 -- X0)
		drop the 2nd element on the stack. (equivalent to ‘ 0 = ‘’)
	swap	(X1 X0 -- X0 X1)
		swap the order of the top two stack elements.
	_ds	(Xu-1 .. X0 -- Xu-1 .. X0 u)
		push a number of elements on the data stack to the top. (for debug)
	_rs	(-- u)
		push a number of elements on the return stack to the top. (for debug)

Console I/O	@c?	(-- flag)
		push true if character from console available, otherwise push false.
	@c	(-- char)
		push character from console to the top.
	%c	(char --)
		output the top element character to console.
	%s	(--)
		output string to console.
	%d	(u --)
		output the top element value with decimal format to console.
	%x	(u --)
		output the top element value with hex-decimal format to console.
System	sfr	(u -- u)
		push SFR[u] value to the top.
	=sfr	(u1 u2 --)
		substitute u1 for SFR[u2]
	ep	(u -- u)
		push program_memory[u] to the top.
	=ep	(u1 u2 --)
		substitute u1 for program_memory[u2].
	ed	(u -- u)
		push data_memory[u] to the top.
	=ed	(u1 u2 --)
		substitute u1 for data_memory[u2].
	clk	(-- u)
		push system lapse (2mS) to the top.
	sec	(-- u)
		push system lapse (sec) to the top.