Jupiter ACE

Reference Card

ACE Forth

Stack Notation	*Cells*	*Description*
c	1	Character (high byte ignored)
flag	1	Boolean (0 = False, 1 = True)
n	1	Signed number
u	1	Unsigned number
x	1	Non-specific single cell
adr	1	Memory address
d	2	Signed double number
ud	2	Unsigned double number
xd	2	Non-specific double cell
f	2	Floating point number

Immediate words in Green
Immediate and compile-only words in Blue

Stack Manipulation

DROP	(x →)	Discard TOS (top of stack)
DUP	(x → x x)	Duplicate TOS
?DUP	(x → x (x))	DUP, if non-zero
OVER	(x_1 x_2 → x_1 x_2 x_1)	Copy second on stack to top
PICK	(x_n...x_1 n → x_n...x_1 x_n)	Copy n^{th} cell to top
ROLL	(x_n...x_1 n → x_{n-1}...x_1 x_n)	Rotate n^{th} cell to top
ROT	(x_1 x_2 x_3 → x_2 x_3 x_1)	Rotate 3rd cell to top
>R	(x →) (R: → x)	Move TOS to Return Stack
R>	(→ x) (R: x →)	Retrieve from Return Stack
SWAP	(x_1 x_2 → x_2 x_1)	Exchange the two top cells

Comparison

<	(n_1 n_2 → *flag*)	True if $n_1 < n_2$
=	(n_1 n_2 → *flag*)	True if $n_1 = n_2$
>	(n_1 n_2 → *flag*)	True if $n_1 > n_2$
0<	(n → *flag*)	True if $n < 0$
0=	(n → *flag*)	True if $n = 0$
0>	(n → *flag*)	True if $n > 0$
U<	(u_1 u_2 → *flag*)	True if $u_1 < u_2$
D<	(d_1 d_2 → *flag*)	True if $d_1 < d_2$
MAX	(n_1 n_2 → n_3)	Leave greater of two numbers
MIN	(n_1 n_2 → n_3)	Leave lesser of two numbers

Logical

AND	(x_1 x_2 → x_3)	Bitwise boolean AND
OR	(x_1 x_2 → x_3)	Bitwise boolean OR
XOR	(x_1 x_2 → x_3)	Bitwise boolean XOR

Integer Arithmetic

+	(n_1 n_2 → n_3)	$n_3 = n_1 + n_2$
-	(n_1 n_2 → n_3)	$n_3 = n_1 - n_2$
*****	(n_1 n_2 → n_3)	$n_3 = n_1 * n_2$
/	(n_1 n_2 → n_3)	$n_4 = n_1 / n_2$
MOD	(n_1 n_2 → n_3)	Remainder of n_1 / n_2 (sign of n_1)
/MOD	(n_1 n_2 → n_3 n_4)	n_3 = remainder of n_1 / n_2 $n_4 = n_1 / n_2$
***/**	(n_1 n_2 n_3 → n_4)	$n_4 = n_1 * n_2 / n_3$
***/MOD**	(n_1 n_2 n_3 → n_4 n_5)	n_4 = remainder of $n_1 * n_2 / n_3$ $n_5 = n_1 * n_2 / n_3$
1+	(n_1 → n_2)	$n_2 = n_1 + 1$
1-	(n_1 → n_2)	$n_2 = n_1 - 1$
2+	(n_1 → n_2)	$n_2 = n_1 + 2$
2-	(n_1 → n_2)	$n_2 = n_1 - 2$
ABS	(n → u)	$u = \|n\|$ (absolute value)
NEGATE	(n_1 → n_2)	$n_2 = -n_1$ (two's complement)
U*	(u_1 u_2 → ud)	$ud = u_1 * u_2$
U/MOD	(ud u_1 → u_2 u_3)	u_2 = remainder of ud / u_1 $u_3 = ud / u_1$
D+	(d_1 d_2 → d_3)	$d_3 = d_1 + d_2$
DNEGATE	(d_1 → d_2)	$d_2 = -d_1$ (two's complement)

Floating Point Arithmetic

INT	(f → n)	Convert floating number to integer
UFLOAT	(u → f)	Convert unsigned integer to float
F+	(f_1 f_2 → f_3)	$f_3 = f_1 + f_2$
F-	(f_1 f_2 → f_3)	$f_3 = f_1 - f_2$
F*	(f_1 f_2 → f_3)	$f_3 = f_1 * f_2$
F/	(f_1 f_2 → f_3)	$f_3 = f_1 / f_2$
FNEGATE	(f_1 → f_2)	$f_2 = -f_1$

Memory

@	(*adr* → x)	Read x (2 bytes) from *adr*
!	(x *adr* →)	Store x (2 bytes) to *adr*
C@	(*adr* → c)	Read c (1 byte) from *adr*
C!	(c *adr* →)	Store c (1 byte) to *adr*

Control Structures

IF	(*flag* →)	Conditional structure IF..(ELSE)..THEN
ELSE	(→)	False condition of an IF structure
THEN	(→)	End of an IF conditional structure
DO	(n_1 n_2 →)	Counted loop structure DO...LOOP (n_2 = count start, n_1 = count end)
LOOP	(→)	Increment loop count, terminate if end
+LOOP	(n →)	Add n to loop count, terminate if end
I	(→ n)	Get current loop count
I'	(→ n)	Get current loop count limit
J	(→ n)	Get outer loop count
LEAVE	(→)	Force a DO...LOOP count to end
BEGIN	(→)	Begin a WHILE or UNTIL loop
UNTIL	(*flag* →)	Loop until *flag* = true (BEGIN..UNTIL)
WHILE	(*flag* →)	Exit loop when *flag* = false (BEGIN..WHILE..REPEAT)
REPEAT	(→)	Jump back to BEGIN in a WHILE loop
EXIT	(→)	Exit current word execution
EXECUTE	(*adr* →)	Execute word with compilation *adr*
CALL	(*adr* →)	Call Z80 code (terminated with **jp(iy)**)
ABORT	(... →)	Quit program, clearing data stack
QUIT	(→)	Quit program, not clearing data stack

Character Input/Output

CR	(→)	Print carriage return and line feed
ASCII *text*	(→ c)	ASCII code of first character in *text*
EMIT	(c →)	Print ASCII c character
SPACE	(→)	Print one space
SPACES	(n →)	Print n spaces, if $n > 0$
."	(→)	Print a string terminated by "
TYPE	(*adr* n →)	Print n characters from *adr*
QUERY	(→)	Accept entry at the input buffer

WORD	(*c* → *adr*)	Take text from input buffer using *c* as delimiter, leave *adr* of length byte
RETYPE	(→)	Allow input buffer editing, turning cursor to **?**
INKEY	(→ *x*)	Read keyboard (0 = no key pressed)

Number Input/Output

BASE	(→ *adr*)	1-byte variable containing system number base
DECIMAL	(→)	Set base to decimal
.	(*n* →)	Print *n* with one trailing space
U.	(*u* →)	Print unsigned with one trailing space
F.	(*f* →)	Print float with one trailing space
CONVERT	(d_1 adr_1 → d_2 adr_2)	Convert string at adr_1 to double number and add into d_1 leaving result d_2
<#	(→)	Initiate formatted output
#	(ud_1 → ud_2)	Convert one digit from ud_1 and HOLD it in the PAD
#S	(*ud* → 0 0)	Convert and HOLD all remaining significant digits
HOLD	(*c* →)	Insert character into formatted string
SIGN	(*n* →)	HOLD minus sign if *n* < 0
#>	(*ud* → *adr n*)	Finish formatted output leaving address & length of the resulting string
NUMBER	(→ *x (adr)*)	Get number from input buffer
	(→ *n 4102*)	Converted to integer
	(→ *f 4181*)	Converted to float
	(→ *0*)	Conversion failed

Word Definition

: *word*	(→)	Start a *word* definition
;	(→)	Terminate a word definition
VARIABLE *name*	(*x* →)	Define a variable with value *x*
variable-name	(→ *adr*)	Get variable *adr*
CONSTANT *name*	(*x* →)	Define a constant with value *x*
constant-name	(→ *x*)	Get constant value
IMMEDIATE	(→)	Mark newest word as immediate
CREATE *name*	(→)	Create a dictionary entry
DEFINER *word*	(→)	Start a defining *word* definition
DOES>	(→ *adr*)	Define the action routine of a defining word
COMPILER *word*	(*n* →)	Start a compiling *word* definition
RUNS>	(→ *adr*)	Defines the action routine of a compiling word
FIND *word*	(→ *adr*)	Find *word* compilation address (0 if not found)
FORGET *word*	(→)	Clear all definitions back to *word*
LIST *word*	(→)	List *word* definition
EDIT *word*	(→)	Edit *word* definition
REDEFINE *word*	(→)	Replace previous *word* with the newest dictionary entry

Vocabulary

VOCABULARY *name*	(→)	Define a new vocabulary
vocabulary	(→)	Set CONTEXT = *vocabulary*
CONTEXT	(→ *adr*)	Get current word search vocabulary address (15411)
CURRENT	(→ *adr*)	Get current word definition vocabulary address (15409)
FORTH	(→)	Set CONTEXT to the FORTH vocabulary
DEFINITIONS	(→)	Set CURRENT vocabulary to CONTEXT
VLIST	(→)	List dictionary to screen

Compiler

,	(*x* →)	Compile *x* into the dictionary
C,	(*c* →)	Compile *c* into the dictionary
ALLOT	(*n* →)	Enclose *n* bytes in the dictionary
LITERAL	(*x* →)	Compile *x* as literal
[	(→)	Enter interpret mode
]	(→)	Enter compile mode

Miscellaneous

(	(→)	Start a comment, terminated by **)**
CLS	(→)	Clear screen
AT	(n_1 n_2 →)	Set print position to row n_1 and column n_2
HERE	(→ *adr*)	Next available dictionary location
PAD	(→ *adr*)	Scratch pad area address (9985)
SLOW	(→)	Normal execution. Enable error checks
FAST	(→)	Faster execution. Disable error checks
BEEP	(u_1 u_2 →)	Play tone u_1 = 1000000 / (8 * frequency [Hz]) u_2 = duration [ms]
IN	(*adr* → *c*)	Read byte from Z80 input port *adr*
OUT	(*c adr* →)	Write byte to Z80 output port *adr*
INVIS	(→)	Disable copy-up mechanism and OK
VIS	(→)	Enable copy-up mechanism and OK
LINE	(→)	Interpret the input buffer as FORTH
PLOT	(n_1 n_2 n_3 →)	Plot at n_1 (X), n_2 (Y) with mode n_3 (0 = unplot, 1=plot, 2=move, 3=change)

Tape Files

LOAD *name*	(→)	Load vocabulary from tape
SAVE *name*	(→)	Save vocabulary to tape
VERIFY *name*	(→)	Verify vocabulary
BLOAD *name*	(*adr u* →)	Load *u* bytes from tape to *adr*
BSAVE *name*	(*adr u* →)	Save *u* bytes from *adr* to tape
BVERIFY *name*	(*adr u* →)	Verify *u* bytes from *adr*

if *adr* = 0 then use file value (BLOAD or BVERIFY)
if *u* = 0 then use file value (BLOAD or BVERIFY)

Error Codes

Error	*Description*
1	Not enough memory
2	Data Stack Underflow
3	BREAK pressed
4	Compile only word
5	Structure imbalance
6	Name size < 1 or > 64
7	PICK or ROLL operand ≤ 0
8	Floating point overflow
9	AT or PLOT to the input buffer
10	Tape error
11	REDEFINE or FORGET error
12	Incomplete definition in dictionary
13	Word not found or is ROM or is FORTH
14	Word not Listable