amforth 4.9 Reference Card

Arithmetics

1- `( n1 - n2 )`
optimized decrement

1+ `( n1|u1 - n2|u2 )`
optimized increment

2/ `( n1 - n2 )`
arithmetic shift right

2* `( n1 - n2 )`
arithmetic shift left, filling with zero

abs `( n1 - u1 )`
get the absolute value

>< `( n1 - n2 )`
exchange the bytes of the TOS

cell+ `( a-addr1 - a-addr2 )`
add the size of an address-unit to a-addr1

cells `( n1 - n2 )`
n2 is the size in address units of n1 cells

d2/ `( d1 - d2 )`
shift a double cell value right

d2* `( d1 - d2 )`
shift a double cell left

dabs `( d - ud )`
double cell absolute value

dinvert `( d1 - d2)`
invert all bits in the double cell value

d- `( d1 d2 - d3 )`
subtract d2 from d1

dnegate `( d1 - d2 )`
double cell negation

d+ `( d1 d2 - d3)`
add 2 double cell values

invert `( n1 - n2)`
1-complement of TOS

log2 `( n1 - n2 )`
logarithm to base 2 or highest set bit-number

lshift `( n1 n2 - n3)`
logically shift n1 left n2 times

- `( n1|u1 n2|u2 - n3|u3 )`
subtract n2 from n1

mod `( n1 n2 - n3)`
divide n1 by n2 giving the remainder n3

m* `( n1 n2 - d)`
multiply 2 cells to a double cell

+ `( n1 n2 - n3)`
add n1 and n2

+! `( n a-addr - )`
add n to content of RAM address a-addr

rshift `( n1 n2 - n3 )`
shift n1 n2-times logically right

/ `( n1 n2 - n3)`
divide n1 by n2. giving the quotient

/mod `( n1 n2 - rem quot)`
signed division n1/n2 with remainder and quotient

***** `( n1 n2 - n3 )`
multiply routine

***/** `(n1 n2 n3 - n4)`
signed multiply and division with double precision intermediate

***/mod** `( n1 n2 n3 - rem quot)`
signed multiply n1 * n2 and division with n3 with double precision intermediate and remainder

true `( - -1 )`
leaves the value -1 (true) on TOS

ud/mod `( d1 n - rem ud2 )`
unsigned double cell division with remainder

um/mod `( ud u2 - rem quot)`
unsigned division ud / u2 with remainder

um* `( u1 u2 - d)`
multiply 2 unsigned cells to a double cell

u/mod `(u1 u2 - rem quot)`
unsigned division with remainder

0 `( - 0 )`
place a value 0 on TOS

Character IO

bl `( - 32 )`
put ascii code of the blank to the stack

cr `( - )`
cause subsequent output appear at the beginning of the next line

emit `( c - )`
fetch the emit vector and execute it. should emit a character from TOS

emit? `( - f )`
fetch emit? vector and execute it. should return the ready-to-send condition

key `( - c )`
fetch key vector and execute it, should leave a single character on TOS

key? `( - f)`
fetch key? vector and execute it. should turn on key sender, if it is disabled/stopped

space `( - )`
emits a space (bl)

spaces `( n - )`
emits n space(s) (bl)

type `( addr n - )`
print a RAM based string

Compare

d= `( n1 n2 - flag )`
compares two double cell values

d> `( d1 d2 - flag )`
compares two double cell values (signed)

d< `( d1 d2 - flag)`
checks whether d1 is less than d2

= `( n1 n2 - flag )`
compares two values for equality

0= `( n - flag )`
compare with 0 (zero)

> `( n1 n2 - flag )`
flag is true if n1 is greater than n2

0> `( n1 - flag )`
true if n1 is greater than 0

0< `( n1 - flag)`
compare with zero

max `( n1 n2 - n1|n2 )`
compare two values, leave the bigger one

min `( n1 n2 - n1|n2 )`
compare two values leave the smaller one

<> `( n1 n2 - flag)`
true if n1 is not equal to n2

0<> (n - flag)
true if n is not zero

u> (u1 u2 - flag)
true if u1 > u2 (unsigned)

u>= (u1 u2 - flag)
compare two unsigned numbers, returns true flag if u1 is greater then or equal to u2

u< (u1 u2 - flasg)
true if u1 < u2 (unsigned)

u<= (u1 u2 - flag)
compare two unsigned numbers, returns true flag if u1 is less then or equal to u2

within (n min max - f)
check if n is within min..max

Compiler

2literal (- x1 x2)
(C: x1 x2 -)
compile a cell pair literal in colon definitions

again (-)
(C: dest -)
compile a jump back to dest

**** ("ccc<eol>-)
everything up to the end of the current line is a comment

begin (-)
(C: - dest)
put the next location for a transfer of control onto the control flow stack

['] (- xt)
(C: «space>name-)
what ' does in the interpreter mode, do in colon definitions

code (-)
(C: cchar -)
create named entry in the dictionary, XT is the data field

: (-)
(C: «spaces>name-)
create a named entry in the dictionary, XT is DO-COLON

:noname (- xt)
create an unnamed entry in the dictionary, XT is DO-COLON

constant (- x)
(C: x «spaces>name-)
create a constant in the dictionary

do (n1 n2 -)
(R: - loop-sys)
(C: - do-sys)
start do .. [+]loop

(create) (-)
(C: «spaces>name- voc-link)
parse the input and create an vocabulary entry without XT and data field (PF)

does> (i*x - j*y)
(R: nest-sys1 -)
(C: colon-sys1 - colon-sys2)
replace the runtime semantics

." (-)
(C: "ccc<quote>-)
compiles string into dictionary to be printed at runtime

Edefer (c<name> -)
creates a defer vector which is kept in eeprom.

else (C: orig1 - orig2)
(C: orig1 - orig2)
resolve the forward reference and place a new unresolved forward reference

end-code (-)
finish a code definition

exit (-)
(R: nest-sys -)
end of current colon word

header (addr len wid - voc-link)
creates the vocabulary header without XT and data field (PF) in the wordlist wid

i (- n)
(R: loop-sys - loop-sys)
current loop counter

if (f -)
(C: - orig)
start conditional branch

immediate (-)
set immediate flag for the most recent word definition

j (- n)
(R: loop-sys1 loop-sys2 - loop-sys1 loop-sys2)
loop counter of outer loop

[(-)
enter interpreter mode

leave (-)
(R: loop-sys -)
immediatly leave the current DO..LOOP

literal (- n)
(C: n -)
compile a literal in colon defintions

loop (R: loop-sys -)
(R: loop-sys -)
(C: do-sys -)
compile (loop) and resolve the backward branch

(("ccc<paren>-)
skip everything up to the closing bracket on the same line

+loop (n -)
(R: loop-sys - loop-sys|)
(C: do-sys -)
compile (+loop) and resolve branches

?do (n1|u1 n2|u2 -)
(C: - do-sys)
start a ?do .. [+]loop control structure

] (-)
enter compiler mode

Rdefer (c<name> -)
creates a RAM based defer vector

recurse (-)
compile the XT of the word currently being defined into the dictionary

repeat (-)
(C: orig dest -)
continue execution at dest, resolve orig

s, (addr len -)
compiles a string from RAM to Flash

; (-)
finish colon defintion, compiles (exit) and returns to interpret state

s" (- addr len)
(C: <cchar> -)
compiles a string to flash, at runtime leaves (– flash-addr count) on stack

then (-)
(C: orig -)
finish if

unloop (-)
(R: loop-sys -)
remove loop-sys, exit the loop and continue execution after it

until (f -)
(C: dest -)
finish begin with conditional branch, leaves the loop if true flag at runtime

user `( n cchar - )`
create a dictionary entry for a user variable at offset n

value `( n <name> - )`
create a dictionary entry for a value and allocate 1 cell in EEPROM.

variable `( cchar - )`
create a dictionary entry for a variable and allocate 1 cell RAM

while `( f - )`
`(C: dest - orig dest )`
at runtime skip until repeat if non-true

Conversion

d>s `( d1 - n1 )`
shrink double cell value to single cell.

s>d `( n1 - d1 )`
extend (signed) single cell value to double cell

Dictionary

`( n - )`
compile 16 bit into flash at DP

compile `( - )`
read the following cell from the dictionary and append it to the current dictionary position.

create `( - a-addr )`
`(C: «spaces>name- )`
create a dictionary header. XT is (constant), with the address of the data field of name

' `( «spaces>name- XT )`
search dictionary for name, returns XT or throw an exception -13

Environment

/hold `( - hldsize )`
size of the pictured numeric output buffer in bytes

/pad `( - padsize )`
Size of the PAD buffer in bytes

/user `( - usersize )`
size of the USER area in bytes

wordlists `( - n )`
maximum number of wordlists in the dictionary search order

cpu `( - faddr len )`
flash address of the CPU identification string

forth-name `( - faddr len )`
flash address of the amforth name string

version `( - n )`
version number of amforth

mcu-info `( - faddr len )`
flash address of some CPU specific parameters

Exceptions

abort `( i*x - )`
`(R: j*y - )`
send an exception -1

abort" `( i*x x1 - | i*x )`
`(R: j*y - | j*y )`
`(C: "ccc<quote>- )`
check flag. If true display the parsed text and throw exception -2

catch `( i*x xt - j*x 0 | i*x n )`
execute XT and check for exceptions.

handler `( - a-addr )`
USER variable used by catch/throw

throw `( n - )`
throw an exception

Extended VM

a@ `( - n2 )`
Read memory pointed to by register A (Extended VM)

a@- `( - n )`
Read memory pointed to by register A, decrement A by 1 cell (Extended VM)

a@+ `( - n )`
Read memory pointed to by register A, increment A by 1 cell (Extended VM)

a! `( n - )`
Write memory pointed to by register A (Extended VM)

a!- `( - n2 )`
Write memory pointed to by register A, decrement A by 1 cell (Extended VM)

a!+ `( - n2 )`
Write memory pointed to by register A, increment A by 1 cell (Extended VM)

a> `( n1 - n2 )`
read the A register (Extended VM)

b@ `( - n2 )`
Read memory pointed to by register B (Extended VM)

b@- `( - n )`
Read memory pointed to by register B, decrement B by 1 cell (Extended VM)

b@+ `( - n )`
Read memory pointed to by register B, increment B by 1 cell (Extended VM)

b! `( n - )`
Write memory pointed to by register B (Extended VM)

b!- `( - n2 )`
Write memory pointed to by register B, decrement B by 1 cell (Extended VM)

b!+ `( - n2 )`
Write memory pointed to by register B, increment B by 1 cell (Extended VM)

b> `( n1 - n2 )`
read the B register (Extended VM)

na@ `( n1 - n2 )`
Read memory pointed to by register A plus offset (Extended VM)

na! `( n offs - )`
Write memory pointed to by register A plus offset (Extended VM)

nb@ `( n1 - n2 )`
Read memory pointed to by register B plus offset (Extended VM)

nb! `( n offs - )`
Write memory pointed to by register B plus offset (Extended VM)

>a `( n - )`
Write to A register (Extended VM)

>b `( n - )`
Write to B register (Extended VM)

Interpreter

get-recognizer `( - recn .. rec0 n)`
Get the current recognizer list

rec-find `( addr len - f )`
recognizer searching the dictionary

rec-intnum `( addr len - f )`
recognizer for integer numbers

rec-notfound `( addr len - )`
recognizer for NOT FOUND

set-recognizer `( recn .. rec0 n - )`
replace the recognizer list

Interrupt

int@ (i - xt)
fetches XT from interrupt vector i

-int (- sreg)
turns off all interrupts and leaves SREG in TOS

+int (-)
turns on all interrupts

int! (xt i -)
stores XT as interrupt vector i

int-trap (i -)
trigger an interrupt

#int (- n)
number of interrupt vectors (0 based)

Logic

and (n1 n2 - n3)
bitwise and

negate (n1 - n2)
2-complement

not (flag - flag')
identical to 0=

or (n1 n2 - n3)
logical or

xor (n1 n2 - n3)
exclusive or

MCU

!@spi (n1 - n2)
SPI exchange of 2 bytes, high byte first

baud (- v)
returns usart baudrate settings

bm-clear (bitmask byte-addr -)
clear bits set in bitmask on byte at addr

bm-set (bitmask byte-addr -)
set bits from bitmask on byte at addr

bm-toggle (bitmask byte-addr -)
toggle bits set in bitmask on byte at addr

-jtag (-)
disable jtag at runtime

-wdt (-)
disable watch dog timer at runtime

rx?-isr (- f)
check if unread characters are in the input queue using interrupt driver

rx?-poll (- f)
check if a character can be appended to output queue using register poll

rx-isr (- c)
get 1 character from input queue, wait if needed using interrupt driver

rx-poll (c -)
wait for one character and read it from the terminal connection using register poll

sleep (mode -)
put the controller into the specified sleep mode

c!@spi (txbyte - rxbyte)
SPI exchange of 1 byte

>usart (-)
initialize the user area to use the system terminal for IO

tx?-isr (- f)
check if a character can be appended to output queue.

tx?-poll (- f)
check if a character can be send using register poll

tx-isr (c -)
put 1 character into output queue, wait if needed, enable UDRIE interrupt

tx-poll (c -)
check availability and send one character to the terminal using register poll

+usart (-)
initialize usart

+usartx (-)
initialize the atxmega usart (ATXmega)

wdr (-)
calls the MCU watch dog reset instruction

x-rx?-poll (- f)
check if a character can read from the terminal (Poll, ATXmega)

x-rx-poll (- c)
wait for and get one character from the terminal (Poll, ATXmega)

x-tx?-poll (- f)
check if a character can be sent (Poll, ATXmega)

x-tx-poll (c -)
wait for the terminal becomes ready and put 1 character to it (Poll, ATXmega)

Memory

c@ (a-addr - c1)
fetch a single byte from memory mapped locations

cmove (addr-from addr-to n -)
copy data in RAM, from lower to higher addresses

cmove> (addr-from addr-to n -)
copy data in RAM from higher to lower addresses.

c! (c a-addr -)
store a single byte to RAM address

(!i-nrww) (n f-addr -)
writes n to flash memory using assembly code (code to be placed in boot loader section)

(!i-nvm) (n f-addr -)
writes n to flash at f-addr using NVM (ATXmega)

@ (a-addr - n)
read 1 cell from RAM address

@e (e-addr - n)
read 1 cell from eeprom

@e (e-addr - n)
read 1 cell from eeprom using NVM (ATXmega)

@i (f-addr - n1)
read 1 cell from flash

@u (a-addr - n)
read 1 cell from RAM address

fill (a-addr u c -)
fill u bytes memory beginning at a-addr with character c

! (n addr -)
write n to RAM memory at addr, low byte first

!e (n e-addr -)
write n (2bytes) to eeprom address

!e (n e-addr -)
write n (2bytes) to eeprom address using nvm (atxmega)

!i `( n addr - )`
Deferred action to write a single 16bit cell to flash

!u `( n addr - )`
write n to RAM memory at addr, low byte first

Multitasking

pause `( - )`
Fetch pause vector and execute it. may make a context/task switch

Numeric IO

base `( - a-addr )`
location of the cell containing the number conversion radix

bin `( - )`
set base for number conversion to 2

d. `( d - )`
singed PNO with double cell numbers

d.r `( d w - )`
singed PNO with double cell numbers, right aligned in width w

decimal `( - )`
set base for numeric conversion to 10

digit? `( c - (number|)`
tries to convert a character to a number, set flag accordingly

. `( n - )`
singed PNO with single cell numbers

.r `( n w - )`
singed PNO with single cell numbers, right aligned in width w

hex `( - )`
set base for number conversion to 16

hld `( - addr )`
pointer to current write position in the Pictured Numeric Output buffer

hold `( c - )`
prepend character to pictured numeric output buffer

<# `( - )`
initialize the pictured numeric output conversion process

number `(addr len - [n|d size] f)`
convert a counted string at addr to a number

`( d1 - d2 )`
pictured numeric output: convert one digit

#> `( d1 - addr count )`
Pictured Numeric Output: convert PNO buffer into an string

#s `( d - 0 )`
pictured numeric output: convert all digits until 0 (zero) is reached

sign `( n - )`
place a - in HLD if n is negative

>number `( ud1 c-addr1 u1 - ud2 c-addr2 u2 )`
convert a string to a number c-addr2/u2 is the unconverted string

ud. `( ud - )`
unsigned PNO with double cell numbers

ud.r `( ud w - )`
unsigned PNO with double cell numbers, right aligned in width w

u. `( u - )`
unsigned PNO with single cell numbers

u.r `( u w - )`
unsigned PNO with single cells numbers, right aligned in width w

u0.r `( ud n - )`
Print n digits, fill in preceeding zeros if needed

Search Order

also `( - )`
Duplicate first entry in the current search order list

definitions `( - )`
Make the compilation word list the same as the current first word list in the search order.

forth `( - )`
replace the search order list with the system default list

forth-wordlist `( - wid )`
get the system default word list

get-current `( - wid)`
get the wid of the current compilation word list

get-order `( - widn .. wid0 n)`
Get the current search order word list

only `( - )`
replace the order list with the system default list

order `( - )`
print the wids of the current word list and the search order

previous `( - )`
remove the first entry in the search order list

search-wordlist `( c-addr len wid - [ 0 ] | [ xt [-1|1]] )`
searches the word list wid for the word at c-addr/len

set-current `( wid - )`
set current word list to the given word list wid

set-order `( widn .. wid0 n - )`
replace the search order list

wordlist `( - wid )`
create a new, empty wordlist

Stack

2r> `( - x1 x2 )`
`(R: x1 x2 -)`
move DTOR to TOS

2swap `( x1 x2 x3 x4 - x3 x4 x1 x2 )`
Exchange the two top cell pairs

2>r `( x1 x2 - )`
`(R: - x1 x2)`
move DTOS to TOR

depth `( - n )`
number of single-cell values contained in the data stack before n was placed on the stack.

drop `( n - )`
drop TOS

dup `( n - n n )`
duplicate TOS

nip `( n1 n2 - n2 )`
Remove Second of Stack

nr> `( xn .. x0 n - )`
`(R: - xn .. x0 n)`
move n items from data stack to return stack

n>r `( xn .. x0 n - )`
`(R: - xn .. x0 n)`
move n items from data stack to return stack

over `( x1 x2 - x1 x2 x1 )`
Place a copy of x1 on top of the stack

pick (xu ... x1 x0 u - xu ... x1 x0 xu)
access the stack as an array and fetch the u-th element as new TOS

?dup (n1 - [n1 n1] | 0)
duplicate TOS if non-zero

rot (n1 n2 n3 - n2 n3 n1)
rotate the three top level cells

rp0 (- addr)
start address of return stack

rp@ (- n)
current return stack pointer address

rp! (addr -)
(R: - x*y)
set return stack pointer

r@ (- n)
(R: n - n)
fetch content of TOR

r> (- n)
(R: n -)
move TOR to TOS

sp (- addr)
address of user variable to store top-of-stack for inactive tasks

sp0 (- addr)
start address of the data stack

sp@ (- addr)
current data stack pointer

sp! (addr - i*x)
set data stack pointer to addr

swap (n1 n2 - n2 n1)
swaps the two top level stack cells

>r (n -)
(R: - n)
move TOS to TOR

String

compare (r-addr r-len f-addr f-len - f)
compares two strings in RAM

count (c-addr1 - c-addr2 len)
convert addr of counted string to address of the first characater and length of the string

cscan (addr1 n1 c - addr1 n2)
Scan string at addr1/n1 for the first occurance of c, leaving addr1/n2, char at n2 is first non-c character

cskip (addr1 n1 c - addr2 n2)
skips leading occurancies in string at addr1/n1 leaving addr2/n2 pointing to the 1st non-c character

parse (char "ccc<char>- c-addr u)
in input buffer parse ccc delimited string by the delimiter char.

parse-name («name>- c-addr u)
In the SOURCE buffer parse white-space delimited string. Returns string address within SOURCE.

place (addr1 len1 addr2 -)
copy string as counted string

/string (addr1 u1 n - addr2 u2)
adjust string from addr1 to addr1+n, reduce length from u1 to u2 by n

sliteral (C: addr len -)
(C: addr len -)
compiles a string to flash, at runtime leaves (– flash-addr count) on stack

tolower (C - c)
if C is an uppercase letter convert it to lowercase

toupper (c - C)
if c is a lowercase letter convert it to uppercase

System

accept (addr +n1 - +n2)
receive a string of at most n1 characters at addr until n2 characters are reveived or cr/lf detected.

allot (n -)
allocate or release memory in RAM

cold (-)
start up amforth.

defer@ (xt1 - xt2)
returns the XT associated with the given XT

defer! (xt1 xt2 -)
stores xt1 as the xt to be executed when xt2 is called

execute (xt -)
execute XT

f_cpu (- d)
put the cpu frequency in Hz on stack

interpret (-)
(R: i*x - j*x)
interpret input word by word.

is (xt1 c<char> -)
stores xt into defer or compiles code to do so at runtime

quit (-)
main loop of amforth. accept - interpret in an endless loop

refill (- f)
refills the input buffer

refill-tib (- f)
refills the input buffer

source (- addr n)
address and current length of the input buffer

source-tib (- addr n)
address and current length of the input buffer

warm (nx* -)
(R: ny* -)
initialize amforth further. executes turnkey operation and go to quit

System Value

dp (- f-addr)
address of the next free dictionary cell

edp (- e-addr)
address of the next free address in eeprom

ee-user (- v)
address of the default user area content in eeprom

here (- addr)
address of the next free data space (RAM) cell

turnkey (- n*y)
Deferred action during startup/reset

System Variable

environment (- wid)
word list identifier of the environmental search list

>in (- a-addr)
pointer to current read position in input buffer

latest (- addr)
system LATEST

#tib (- addr)
variable holding the number of characters in TIB

pad `( - a-addr )`
Address of the temporary scratch buffer.

state `( - addr )`
system state

tib `( - addr )`
terminal input buffer address

up@ `( - addr )`
get user area pointer

up! `( addr - )`
set user area pointer

Time

1ms `( - )`
busy waits (almost) exactly 1 millisecond

ms `( n - )`
busy waits the specified amount of milliseconds

Tools

[char] `( - c )`
`(C: «space>name- )`
skip leading space delimites, place the first character of the word on the stack

[compile] `( - c )`
`(C: «space>name- )`
skip leading space delimites, place the first COMPILEacter of the word on the stack

char `( «spaces>name- c )`
copy the first character of the next word onto the stack

.s `( - )`
stack dump

ee>ram `( e-addr r-addr len - )`
copy len cells from eeprom to ram

@e[] `( ee-addr - itemn .. item0 n)`
Get an array from EEPROM

find `( addr - addr 0 | xt -1 | xt 1 )`
search wordlists for entry taken as counted string from addr

find-name `( addr len - 0 | xt -1 | xt 1 )`
search wordlists for the name from string addr/len

icompare `( r-addr r-len f-addr f-len - f)`
compares string in RAM with string in flash

icount `( addr - addr+1 n )`
get count information out of a counted string in flash

init-user `( - )`
setup the default user area from eeprom

itype `( addr n - )`
reads string from flash and prints it

noop `( - )`
do nothing

?stack `( - )`
check stack underflow, throw exception -4

show-wordlist `( wid - )`
prints the name of the words in a wordlist

!e[] `( recn .. rec0 n ee-addr - )`
Write a list to EEPROM

to `( n <name> - )`
store the TOS to the named value (eeprom cell)

unused `( - n )`
Amount of available RAM (incl. PAD)

ver `( - )`
print the version string

word `( c - addr )`
skip leading delimiter character and parse SOURCE until the next delimiter. copy the word to HERE

words `( - )`
prints a list of all (visible) words in the dictionary