NAME

mawk-code – dumping mawk's byte-code

SYNOPSIS

At startup, **mawk** compiles the script into byte-code. After that, it interprets the compiled byte-code. Use the **–Wdump** option to show the byte-code.

PROGRAM CODES

As **mawk** executes the program, it maintains a reference to the command to execute in **cdp**. After that there may be data and/or references in **cdp**[0], **cdp**[1], etc.

When an operation requires operands, **mawk** pushes the values (or array/string references) onto the stack, which updates the stack pointer **sp**. When the operation completes, **mawk** consumes those entries on the stack, pushing the result (if any) onto the stack.

While executing user-defined functions, **mawk** maintains a *frame pointer* **fp** to address the function's local variables.

a_cat

Concatenate array-indices.

Usage:

Forms a multiple array index by concatenating the elements of **sp**[1-*cnt*..0], with each element separated by *SUBSEP*.

Parameters:

cdp[0]
: *cnt*, the number of elements to concatenate follows the command.

sp[0]..sp[1-*cnt*]
: hold reference to the elements to concatenate.

Returns the index in sp[0].

a_del

Delete an array item.

Usage:

delete array[expr]

Parameters:

sp[0]
: points to *array*

sp[-1]
: is an *expr*

a_length

Find the length of an array.

Usage:

length(array)

Parameters:

sp[0]
: points to *array*.

Returns the length of the array in sp[0].

a_pusha

Push array address onto stack.

Usage:

This is used to set up a calling argument for a function.

Parameters:

cdp[0]
: array reference follows the command.

Returns the array in sp[0].

a_test

Test if an expression is present in an array.

Usage:
: (expression in array)

Parameters:

sp[0]
: points to *an array*.

sp[-1]
: is an *expression*.

Returns 1 in sp[0] if the expression is found, 0 otherwise.

add

Add two numbers.

Usage:
: *first* + *second*

Parameters:

sp[0]
: holds the *second* value.

sp[-1]
: holds the *first* value.

Returns the sum in sp[0].

add_asg

Combined addition/assignment.

Usage:
: *target* += *source*

Parameters:

sp[0]
: is the *source* expression

sp[-1]
: points to the *target*

Stores the sum in the *target*, leaving sp[0] pointing to the *target*.

ae_pusha

Push reference to array cell, given expression for its index.

Usage:
: *arrayname*[*expression*]

Parameters:

cdp[0]
: an array reference follows the command.

sp[0]
: has an expression, used for the index of a cell in the array.

Returns a reference to the addressed cell in sp[0].

ae_pushi

Push contents of array cell, given expression for its index.

Usage:

arrayname[*expression*]

Parameters:

sp[0]
has an expression, used for the index of a cell in the array.

Returns contents of the addressed cell in sp[0].

aloop

Update reference to next cell for array loop.

Usage:

for (*i* in *arrayname*) *statement*

Parameters:

none

Mawk maintains a stack of array-loop state. It updates the array/cell references in the current loop's state.

assign

Assigns a value.

Usage:

target = *source*

Parameters:

sp[0]
is the *source* expression

sp[-1]
points to the *target*

Stores the sum in the *target*, leaving sp[0] pointing to the *target*.

atan2

Compute arc-tangent of two values.

Usage:

atan2(*first*, *second*)

Parameters:

sp[0]
holds the *second* value

sp[-1]
holds the *first* value

Returns the result in sp[0].

call

Call a function.

Usage:

function()

Parameters:

cdp[0]
is a reference to the function block

cdp[1]
holds number of input arguments

Returns function value in sp[0].

cat

Concatenate two strings.

Usage:
first second

Parameters:

sp[0]
is the *second* string.

sp[-1]
is the *first* string.

Returns the result in sp[0].

close

Close the file or pipe associated with an expression.

Usage:
close(*expression*)

Parameters:

sp[0]
holds the *expression* identifying the file to close

Returns the status from closing the file, 0 on success or −1 on failure.

cos

Compute the cosine of a value in radians.

Usage:
cos(*value*)

Parameters:

sp[0]
is the *value*.

Returns the result in sp[0].

del_a

Delete an array.

Usage:
delete(array)

Parameters:

sp[0]
is the *array* to delete.

div

Divide one number by another.

Usage:
first / *second*

Parameters:

sp[0]
is the *second* value.

sp[-1]
is the *first* value.

Returns the quotient in sp[0].

div_asg

Combined division/assignment.

Usage:
target /= *source*

Parameters:

sp[0]
is the *source*

sp[-1]
points to the *target*

Stores the quotient in the *target*, leaving sp[0] pointing to the target.

eq

Compare two values.

Usage:
first == *second*

Parameters:

sp[0]
is the *second* value

sp[-1]
is the *first* value

Returns 1 in sp[0] if the values are equal, otherwise 0.

exit

Exits mawk with a specific exit-code.

Usage:
exit(*exit_code*)

Parameters:

sp[0]
is the *exit_code*

exit0

Exits mawk with success

Usage:
exit

Parameters:
none

exp

Compute base-e exponential function of a value.

Usage:
exp(*value*)

Parameters:

sp[0]
is the *value*

Returns the result in sp[0].

f_add_asg

Combination addition/assignment to NF.

Usage:

NF += *expression*

Parameters:

sp[0]

is the *expression* to add

f_assign

Assign an expression to NF.

Usage:

NF = *expression*

Parameters:

sp[0]

is the *expression*

f_div_asg

Combination division/assignment to NF.

Usage:

NF /= *expression*

Parameters:

sp[0]

is the *expression*

f_mod_asg

Combination modulus/assignment to NF.

Usage:

NF %= *expression*

Parameters:

sp[0]

is the *expression*

f_mul_asg

Combination multiplication/assignment to NF.

Usage:

NF *= *expression*

Parameters:

sp[0]

is the *expression*

f_post_dec

Post-decrement using NF.

Usage:

NF--

Parameters:

holds a reference to the field to use

f_post_inc

Post-increment using NF.

Usage:

NF++

Parameters:
holds a reference to the field to use

f_pow_asg
Exponentiation using NF.

Usage:
NF ^= *expression*

Parameters:

sp[0]
is the expression to use

f_pre_dec
Predecrement using NF.

Usage:
−−NF

Parameters:

sp[0]
holds a reference to the field to use

f_pre_inc
Preincrement using NF.

Usage:
++NF

Parameters:

sp[0]
holds a reference to the field to use

f_pusha
Push array reference to data split-up as fields..

Usage:
$0 = *expression*
getline

Parameters:

cdp[0]
is a reference to the data to be split/assigned.

Returns the resulting array reference in sp[0].

f_pushi
Push contents of numbered field.

Usage:
$*expression*

Parameters:

cdp[0]
holds a reference to $*expression*

cdp[1]
holds *expression*

Returns the field's value in sp[0].

f_sub_asg
Combination subtraction/assignment to NF.

Usage:
NF -= *expression*

Parameters:

sp[0]
holds a reference to the field to use

fe_pusha
Push reference to numbered field.

Usage:
$*number*

Parameters:

sp[0]
holds the field *number*

Returns a reference to the field in sp[0].

fe_pushi
Push content of numbered field.

Usage:
$*number*

Parameters:

sp[0]
holds the field *number*

Returns the field's content in sp[0].

fflush
Flush the output file or pipe associated with an expression.

Usage:
fflush(*expression*)

Parameters:

sp[0]
is the *expression* value

Returns the result in sp[0].

gt
Test if first value is greater than the second.

Usage:
first > *second*

Parameters:

sp[0]
holds the *second* value.

sp[-1]
holds the *first* value.

Returns 1 in sp[0] if the *first* value is greater than, otherwise 0.

gte
Test if first value is greater than or equal to the second.

Usage:
first >= *second*

Parameters:

sp[0]
holds the *second* value.

sp[-1]
holds the *first* value.

Returns 1 in sp[0] if the *first* value is greater than or equal, otherwise 0.

index

Find the position of the second string in the first.

Usage:
index(*first*, *second*)

Parameters:

sp[0]
is the *second* string

sp[0]
is the *first* string

Returns the position in sp[0] starting at 1 if found, 0 if not found.

int

Returns a value truncated towards zero..

Usage:
int(*value*)

Parameters:

sp[0]
is the value

Returns the result in sp[0].

jmain

Go from BEGIN code to MAIN code.

Usage:
(internal state)

Parameters:

none

jmp

Jump to a new byte-code position, by a given number of bytes.

Usage:
(internal state)

Parameters:

cdp[0]
holds the (signed) number of bytes by which to jump.

jnz

Jump to a new byte-code position if sp[0] is nonzero, by a given number of bytes.

Usage:
(internal state)

Parameters:

cdp[0]
holds the (signed) number of bytes by which to jump.

sp[0]
holds a value to compare against 0.

jz
Jump to a new byte-code position if sp[0] is zero, by a given number of bytes.

Usage:
(internal state)

Parameters:

cdp[0]
holds the (signed) number of bytes by which to jump.

sp[0]
holds a value to compare against 0.

l_pusha
Push a local address onto the evaluation stack.

Usage:
(internal state)

Parameters:

cdp[0]
holds the offset from the *frame pointer* **fp**.

Returns the address in sp[0].

l_pushi
Push contents of a local variable onto the evaluation stack.

Usage:
(internal state)

Parameters:

cdp[0]
holds the offset from the *frame pointer* **fp**.

Returns the contents of the local variable in sp[0].

la_pusha
Pushes a reference to an array onto the evaluation stack.

Usage:
arrayname

Parameters:

cdp[0]
holds the offset from the *frame pointer* **fp** of a reference to an array.

Returns a reference to the array in sp[0].

lae_pusha
Pushes a reference to a given array cell onto the evaluation stack.

Usage:
arrayname[*expression*]

Parameters:

cdp[0]
holds the offset from the *frame pointer* **fp** of a reference to an array.

sp[0]
holds an *expression*

Returns a reference to the specified array cell in sp[0].

lae_pushi
Pushes the contents of a given array cell onto the evaluation stack.

Usage:
arrayname[*expression*]

Parameters:

cdp[0]
holds the offset from the *frame pointer* **fp** of a reference to an array.

sp[0]
holds an *expression*

Returns the contents of the specified array cell in sp[0].

length
Returns the length of a string or array value.

Usage:
length(*value*)

Parameters:

sp[0]
is the string or array reference

Returns the length in sp[0].

ljnz
Special jump for logical-OR, always preceded by **test**.

Usage:
(internal state)

Parameters:

cdp[0]
holds the (signed) number of bytes by which to jump if the value is nonzero.

sp[0]
holds a value to compare against 0.

ljz
Special jump for logical-OR, always preceded by **test**.

Usage:
(internal state)

Parameters:

cdp[0]
holds the (signed) number of bytes by which to jump if the value is zero.

sp[0]
holds a value to compare against 0.

log
Compute the natural logarithm of a value.

Usage:
log(*value*)

Parameters:

sp[0]
is the value

Returns the result in sp[0].

lt

Test if first value is less than the second.

Usage:
first < *second*

Parameters:

sp[0]
holds the *second* value.

sp[-1]
holds the *first* value.

Returns 1 in sp[0] if the *first* value is less than, otherwise 0.

lte

Test if first value is less than or equal to the second.

Usage:
first <= *second*

Parameters:

sp[0]
holds the *second* value.

sp[-1]
holds the *first* value.

Returns 1 in sp[0] if the *first* value is less than or equal, otherwise 0.

match0

Test if $0 matches a given regular expression.

Usage:
$0 ~ *regex*

Parameters:

cdp[0]
holds a reference to a regular expression.

Returns 1 in sp[0] if **$0** matches the regular expression, 0 otherwise.

match1

Test if a given expression matches a given regular expression.

Usage:
expression ~ *regex*

Parameters:

cdp[0]
holds a reference to a regular expression.

sp[0]
holds an expression to test.

Returns 1 in sp[0] if the expression matches the regular expression, 0 otherwise.

match2

Test if an expression in sp[-1] matches the regular expression in sp[0].

Usage:

expression ~ regex

Parameters:

sp[0]
: holds a reference to a regular expression.

sp[-1]
: holds an expression to test.

Returns 1 in sp[0] if the expression matches the regular expression, 0 otherwise.

mktime

Converts a date specification in systime format to a timestamp.

Usage:
: mktime(*string*)

Parameters:

sp[0]
: holds the date-specification string

Returns the result in sp[0].

mod

Compute modulus/remainder with two operands.

Usage:
: *first % second*

Parameters:

sp[0]
: holds the *second* operand

sp[-1]
: holds the *first* operand

Returns the remainder in sp[0].

mod_asg

Assign modulus/remainder with two operands.

Usage:
: *first %= second*

Parameters:

sp[0]
: holds the *second* operand

cdp[0]
: holds the *first* operand

Returns the remainder in sp[0] as well as replacing the *first* value.

mul

Compute product with two operands.

Usage:
: *first * second*

Parameters:

sp[0]
: holds the *second* value

sp[-1]
holds the *first* value

Returns the product in sp[0].

mul_asg
Assign product with two operands.

Usage:
first *= *second*

Parameters:

sp[0]
holds the *second* value

sp[-1]
holds the *first* value

Returns the product in sp[0] as well as replacing the *first* value.

neq
Compare two values.

Usage:
first != *second*

Parameters:

sp[0]
is the *second* value

sp[-1]
is the *first* value

Returns 1 in sp[0] if the values are not equal, otherwise 0.

next
Read the next record, restart pattern testing.

Usage:
next

Parameters:

none

nextfile
Begin processing the next file listed on the command line.

Usage:
nextfile

Parameters:

none

nf_pushi
Push the number of fields (NF) onto the evaluation stack.

Usage:
(internal state)

Parameters:

none

not
Compute a logical negation.

Usage:

! *value*

Parameters:

sp[0]
holds a value to negate.

Returns the result on the evaluation stack, i.e., 0 if the value is nonzero and 1 otherwise.

ol_gl
Read into $0 using getline.

Usage:
getline

Parameters:

none

ol_gl_nr
Read into $0 using getline, updating NR and FNR.

Usage:
getline < *file*

Parameters:

none

omain
Start executing the main section of the script (between BEGIN and END).

Usage:
(internal state)

Parameters:

none

pop
Pop the evaluation stack, discarding the value.

Usage:
(internal state)

Parameters:

none

pop_al
Finish an array "in" loop, deallocating the state information.

Usage:
(internal state)

Parameters:

none

post_dec
Post-decrement a value.

Usage:
value −−

Parameters:

sp[0]
holds the value to decrement

Returns the updated value in sp[0].

post_inc
Post-increment a value.

Usage:
value ++

Parameters:

sp[0]
holds the value to increment

Returns the updated value in sp[0].

pow
Compute the first value raised to the power of the second value.

Usage:
first ^ *second*

Parameters:

sp[0]
holds the *second* value

sp[-1]
holds the *first* value

Returns the result in sp[0].

pow_asg
Assign the first value raised to the power of the second value.

Usage:
variable = *first* ^ *second*

Parameters:

cdp[0]
is a reference to the variable which will be assigned the result

sp[0]
holds the *second* value

sp[-1]
holds the *first* value

pre_dec
Pre-decrement a value.

Usage:
−− *value*

Parameters:

sp[0]
holds the *value* to decrement.

Returns the updated value in sp[0];.

pre_inc
Pre-increment a value.

Usage:
++ *value*

Parameters:

sp[0]
holds the *value* to decrement.

Returns the updated value in sp[0];.

pusha
Push array address onto stack.

Usage:
(internal state)

Parameters:

cdp[0]
array reference follows the command.

Returns the array in sp[0].

pushc
Push a data cell onto the evaluation stack.

Usage:
(internal state)

Parameters:

cdp[0]
is a reference to the data to push

Returns a reference to the result in sp[0].

pushd
Push a double floating value onto the evaluation stack.

Usage:
(internal state)

Parameters:

cdp[0]
is a reference to the data to push

Returns a reference to the result in sp[0].

pushi
Push contents of next referenced variable onto the evaluation stack.

Usage:
(internal state)

Parameters:

cdp[0]
is a reference to the data cell to copy.

Returns a reference to the result in sp[0].

pushint
Reserve the next slot on the evaluation stack, setting its type.

Usage:
(internal state)

Parameters:

cdp[0]
holds the type to set in the new slot, e.g., for data via I/O redirection

Returns a reference to the result in sp[0].

pushs

Push a reference to a string value onto the evaluation stack.

Usage:

(internal state)

Parameters:

cdp[0]
: holds a reference to the string value

Returns a reference to the result in sp[0].

rand

Returns a random number between zero and one..

Usage:

rand()

Parameters:

none

Returns the result in sp[0].

range

Test a range pattern: pat1, pat2 { action }.

Usage:

(internal state)

Parameters:

cdp[0].op
: a flag, test *pat1* if on else *pat2*

cdp[1].op
: offset of *pat2* code from cdp

cdp[2].op
: offset of action code from cdp

cdp[3].op
: offset of code after the action from cdp

cdp[4]
: start of *pat1* code

sp[0]
: holds arguments for the action.

ret

Return a function value.

Usage:

return *value*

Parameters:

sp[0]
: holds the return value

When calling a function, **mawk** saves the current stack, creating a new one. On return, **mawk** restores the previous stack and returns the function value in sp[0].

ret0

Return from a function without providing a return-value.

Usage:

return

Parameters:

sp[0]
is modified to make the value uninitialized.

As in the **ret** operation, **mawk** restores the previous stack. After the return, sp[0] is an uninitialized value.

set_al

Begin an array "in" loop.

Usage:
for (*iterator* in *arrayname*) *statement*

Parameters:

sp[0]
holds a reference to the array

sp[-1]
holds a reference to the iteration variable

Mawk pushes a new entry onto the array loop stack, and updates cdp to point to the statement to execute.

sin

Compute the sine of a value in radians.

Usage:
sin(*value*)

Parameters:

sp[0]
holds the value

Returns the result in sp[0].

sprintf

Returns a string constructed from expression-list according to format.

Usage:
sprintf(*format* [, value1 [,...]])

Parameters:

sp[0]
is the last parameter value; there can be up to 255.

Returns the resulting string in sp[0].

sqrt

Returns the square root of a value.

Usage:
sqrt(*value* 0

Parameters:

sp[0]
is the value

Returns the result in sp[0].

srand

Seeds the random number generator.

Usage:
srand(*value*)
srand( )

Parameters:

sp[0]
: is the seed value, which may be uninitialized

Returns the previous seed value in sp[0].

stop

Finish a range pattern.

Usage:
(internal state)

Parameters:

none

strftime

Formats the given timestamp using the given format.

Usage:
strftime(*format* , *timestamp* , *utc*)
strftime(*format* , *timestamp*)
strftime(*format*)
strftime( )

Parameters:

Zero to three parameters may be on the stack. If all three are used, they are as follows:

sp[0]
: is the *utc* flag

sp[-1]
: is the *timestamp* value

sp[-2]
: is the *format*

Returns the result in sp[0].

sub

Subtract the second value from the first.

Usage:
first − *second*

Parameters:

sp[0]
: holds the *second* value

sp[-1]
: holds the *first* value

Returns the result in sp[0].

sub_asg

Assign the difference of two values to a variable.

Usage:
target = *first* − *second*

Parameters:

cdp[0]
: holds a reference to the variable to which to assign the result

sp[0]
holds the *second* value

sp[-1]
holds the *first* value

Stores the difference in the *target*, leaving sp[0] pointing to the *target*.

substr

eturns the substring of string s, starting at index i, of length n.

Usage:
substr(s,i,n)
substr(s,i)

Parameters:

Two or three parameters may be on the stack. If all three are used, they are as follows:

sp[0]
holds the length *n*.

sp[0]
holds the index *i*.

sp[0]
holds the string *s*.

system

Executes a command, returning the wait-status.

Usage:
status = system(*command*)

Parameters:

sp[0]
is the command to execute

Returns the wait-status in sp[0].

systime

Returns the current time of day as the number of seconds since the Epoch.

Usage:
systime( )

Parameters:

none

Returns the result in sp[0].

test

Test a logical expression.

Usage:
value

Parameters:

sp[0]
holds a value to test.

Returns the result on the evaluation stack, i.e., 1 if the value is nonzero and 0 otherwise.

tolower

Copy a string, converting to lowercase.

Usage:

tolower(*value*)

Parameters:

sp[0]
is the value to convert

Returns the result in sp[0].

toupper
Copy a string, converting to uppercase.

Usage:
toupper(*value*)

Parameters:

sp[0]
is the value to convert

Returns the result in sp[0].

uminus
Unitary minus.

Usage:
− *value*

Parameters:

sp[0]
contains a value to negate. As a side-effect, if the value is a string, it is cast to double floating point.

Returns the result in sp[0].

uplus
Unitary plus.

Usage:
+ *value*

Parameters:

sp[0]
contains a value to use. As a side-effect, if the value is a string, it is cast to double floating point.

Returns the result in sp[0].

REGULAR EXPRESSIONS

M_1J
mandatory jump

M_2JA
optional (undesirable) jump

M_2JB
optional (desirable) jump

M_2JC
pop pos'n, optional jump if advanced

M_ACCEPT
end of match

M_ANY
arbitrary character (.)

M_CLASS
character class

M_END
end of string ($)

M_SAVE_POS
push position onto stack

M_START
start of string (^)

M_STR
matching a literal string

M_U
arbitrary string (.*)