Packages

aarch64

arm

core

the core definitions

llvm-aarch64

llvm-riscv32

llvm-riscv64

llvm-thumb

pcode

posix

posix runtime defintions

primus

the runtime state of the Primus Machine

program

program-specific definitions (program global variables)

riscv

riscv32

riscv64

target

target-specific definitions (registers, etc)

thumb

user

the default user-space package

Macros

non-zero

(non-zero X) is true if X is not zero

compare

(compare X Y) returns 0 if X = Y, a negative value if X<Y and a positive value if X>Y

array-set

(array-set T P N W) sets to W the N-th element of the array of elements of type T, pointed by P. Returns a pointer to the next element

points-to

(points-to T P V) return true if t P points to a value of type T that is equal to V.

is-in

(is-in X A B C ...) returns true if X is equal A or B or C or ...

nth-byte-of-word

(nth-byte-of-word T N X) returns N-th byte of the word X that has type T

ptr+1

(ptr+1 T P) increments the pointer P to a value of to type T.

incr

(incr X Y ...) increments the value bound with the variables X, Y, ...

when

(when CND BODY) if CND evaluates to true, then BODY is evaluated and the value of the last expression in BODY becomes the value of the whole expression. Otherwise, if CND evaluates to false, nil is returned.

until

(until COND BODY) if COND evaluates to true, then the whole expression evaluates to nil and BODY is not evaluated. Otherwise, if COND evaluates to false, then BODY is evaluated until COND evaluates to true and the value of the last evaluation of BODY becomes the value of the whole expression.

or

(or <expr> ...) evaluates a sequence of expressions EXPR from left to right until it meets the first expression that evaluates to the truth value, that will become the value of the whole form. If no expression returned the truth value, then the result of the whole form is 0:1

decr

(decr X Y ...) decrements the value bound with the variables X, Y, ...

+=

(+= X Y) assigns a sum of X and Y to variable X

+1

(+1 x) returns the successor of X

-1

(-1 X) returns the predecessor of X

assert

(assert COND) terminates program if COND is false

array-get

(array-get T P N) gets the N-th element of the array of elements of type T, pointed by P

case/dispatch

write-word

(write-word T A X) writes the word X of type T to address A returns an address that points to the first byte that follows the just written word.

min

(min X Y ...) returns the lower bound of the (X,Y,...) set

max

(max X Y ...) returns the upper bound of the (X,Y,...) set

unless

(unless CND BODY) if CND evaluates to false, then BODY is evaluated and the value of the last expression in BODY becomes the value of the whole expression. Otherwise, if CND evaluates to true, nil is returned.

sizeof

set$

(set$ s x) set the value of the symbol s to x, returns x. Like set but without implicit quotation.

and

(and X Y ...) evaluates expressions X,Y,... until the first expression that returns false. The value of the whole form is the value of the last evaluated expression.

case

(case K K1 X1 K2 X2 ... Kn Xn [DEF]) evaluates K then consequently evaluates keys K1 ... Kn until a key Ki such that (= K Ki) is found. If such key is found then the whole form evaluates to Xi. If no matching key was found, then if the number of keys is equal to the number of expressions, nil is returned, otherwise, i.e., if an extra expression DEF is provided, then it becomes the value of the whole form. Examples: (defun dispatch-with-default-case (c) (case c 1 'hello 2 'cruel 3 'world 'unknown)) (defun dispatch-with-no-default (c) (case c 1 'hello 2 'cruel 3 'world))

cast

copy-left

(copy-left DST SRC LEN) copies LEN bytes from SRC to DST (right to left)

copy-right

(copy-right DST SRC LEN) copies LEN bytes from SRC to DST (left to right)

copy-byte-shift

(copy-byte-shift DST SRC) copies byte from SRC to DST and increments SRC and DST.

bitwidth

read-word

(read-word T A) reads a word of type T at address A

ptr+

(ptr+ T P N) increments N times the pointer P to a value of type T.

endian

(endian F X Y) expands to (F Y X) if applied in the little endian context OR (endian F X Y) expands to (F X Y) if applied in the big endian context

copy-byte-shift-left

(copy-byte-shift-left DST SRC) copies byte from DST to SRC and decrements SRC and DST.

make-copy

<internal>

sign

returns 1 if X $>$ 0, 0 if X $=$ 0, and -1 if X $<$ 0

fold

(fold F A X Y ...) expands to (F (F A X) Y) ...

Constants

`false`

false is another name for 0:1

`nil`

nil is another name for false

`true`

true is another name for 1:1

Functions

`points-to-null`

(points-to-null P) true if P points to a zero byte

`copy-byte`

(copy-byte DST SRC) copies byte from the address SRC to DST.

`lsb`

(msb X) is the least significant bit of X.

`msb`

(msb X) is the most significant bit of X.

`carry`

(carry RD RN RM) is true if the sum RD = RN + RM causes the carry out of the msb bit.

overflow

(overflow RD RN RM) is true if the sum RD = RN + RM results in two's complement overflow.

Primitives

>

(> X Y ... Z) returns one if all numbers are in monotonically decreasing order.

<

(< X Y ... Z) returns one if all numbers are in monotonically increasing order.

=

(= X Y ... Z) returns one if all numbers are equal in value.

*****

(* X Y ... Z) returns X * Y * ... * Z, performing any necessary type conversions in the process. If no numbers are supplied, 1 is returned.

+

(+ X Y ... Z) returns X + Y + ... + Z, performing any necessary type conversions in the process. If no numbers are supplied, 0 is returned.

/

(/ X Y ... Z) returns X / Y / ... / Z, performing any necessary type conversions in the process. If no numbers are supplied, 1 is returned. If one number is provided returns its reciprocal.

-

(- X Y ... Z) returns X - Y - ... - Z, performing any necessary type conversions in the process. If no numbers are supplied returns 0. If one number is supplied returns its negation.

cast-unsigned

(cast-unsigned S X) performs unsigned extension of X to the size of S bits

exec-addr

(exec-addr ADDR) transfers control flow to ADDR.

is-negative

(is-negative X Y ... Z) returns one if all numbers are negative.

cast-low

(cast-low S X) extracts low S bits from X.

logand

(logand X Y ... Z) returns X land Y land ... land Z, performing any necessary type conversions in the process. Where 'X land Y' is bitwise (logical) / (AND) of X and Y. If no numbers are supplied, 1 is returned. If one number is provided returns that number

load-dword

(load-hword PTR) loads double-word from the address PTR

symbol

(symbol X) returns a symbol representation of X.

logor

(logor X Y ... Z) returns X lor Y lor ... lor Z, performing any necessary type conversions in the process. Where ‘X lor Y‘ is bitwise (logical) (OR) of X and Y. If no numbers are supplied, 1 is returned. If one number is provided returns that number

goto-subinstruction

(goto-subinstruction N) transfers control flow to a subinstruction that is N instructions away from the current (N could be negative).

set-symbol-value

(set-symbol-value S X) sets the value of the symbol S to X. Returns X

extract

(extract HI LO X) extracts bits from HI to LO (both ends including) of X, the returned value has HI-LO+1 bits. Both HI and LO must be static.

word-width

(word-width X Y ... Z) returns the maximum width of its arguments. If no arguments provided returns the size of the machine word.

select

(select B1 B2 ... BN X) returns a word that is a concatenation of bits B1, B2, ... BN of X. All of the B1, ..., BN must be static.

is-positive

(is-positive X Y ... Z) returns one if all numbers are positive.

rshift

(rshift X Y ... Z) returns X >> Y >> ... >> Z, performing any necessary type conversions in the process. Where 'X >> Y' is logical shift right of X by Y bits. If no numbers are supplied, 1 is returned. If one number is provided returns that number

nth

(nth N X) returns the Nth bit of X. N must be static. The function is equivalent to (select N X)

neg

(neg X) returns the negation (2-complement) of X. Same as (- X).

not

(not X Y ... Z) returns one if all numbers are not true. Equivalent to (is-zero X Y Z)

store-byte

(store-byte POS VAL) stores byte VAL at the memory position POS

get-current-program-counter

(get-current-program-counter) is an alias to (get-program-counter)

store-word

(store-word POS VAL) stores VAL at the memory position POS

s>

(s> X Y ... Z) returns one if all numbers are in monotonically decreasing signed order.

s<

(s< X Y ... Z) returns one if all numbers are in monotonically increasing signed order.

s/

(s/ X Y ... Z) returns signed X / Y / ... / Z, performing any necessary type conversions in the process. If no numbers are supplied, 1 is returned. If one number is provided returns its signed reciprocal.

>=

(> X Y ... Z) returns one if all numbers are in monotonically nonincreasing order.

<=

(<= X Y ... Z) returns one if all numbers are in monotonically nondecreasing order.

/=

(/= X Y ... Z) returns one if all numbers are distinct.

logxor

(loxgor X Y ... Z) returns X lxor Y lxor ... lxor Z, performing any necessary type conversions in the process. Where 'X lor Y' is bitwise (logical) exclusive (XOR) of X and Y. If no numbers are supplied, 1 is returned. If one number is provided returns that number

memory-write

(memory-write PTR X) stores byte X at PTR.

s>=

(> X Y ... Z) returns one if all numbers are in monotonically nonincreasing signed order.

s<=

(s<= X Y ... Z) returns one if all numbers are in monotonically nondecreasing signed order.

mod

(/ X Y ... Z) returns mod(X,Y) Y mod ... mod Z, performing any necessary type conversions in the process. Where 'X mod Y' is the remainder of [X / Y]. If no numbers are supplied, 1 is returned. If one number is provided returns that number.

memory-read

(memory-read PTR) loads one byte from the address PTR (synonymous to load-byte)

signed-mod

(/ X Y ... Z) returns signed mod(X,Y) Y mod ... mod Z, performing any necessary type conversions in the process. Where 'X mod Y' is the remainder of [X / Y]. If no numbers are supplied, 1 is returned. If one number is provided returns that number.

load-hword

(load-hword PTR) loads half-word from the address PTR

lnot

(lnot X) returns a bitwise logical negation of X.

load-qword

(load-hword PTR) loads quad-word from the address PTR

lshift

(lshift X Y ... Z) returns X << Y << ... << Z, performing any necessary type conversions in the process. Where 'X << Y' is logical shift left of X by Y bits. If no numbers are supplied, 1 is returned. If one number is provided returns that number

load-bits

(load-word SIZE PTR) loads a SIZE-bit long word from the address PTR

get-program-counter

(get-program-counter) returns the address of the current instruction

is-zero

(is-zero X Y ... Z) returns one if all numbers are zero.

arshift

(arshift X Y ... Z) returns X `>> Y ~>> ... ~>> Z, performing any necessary type conversions in the process. Where `X ~>> Y` is arithmetic shift right of X by Y bits. If no numbers are supplied, 1 is returned. If one number is provided returns that number ** ~load-byte` (load-byte PTR) loads one byte from the address PTR

is-symbol

(is-symbol X) is true if X has a symbolic value.

cast-signed

(cast-signed S X) performs signed extension of X to the size of S bits

load-word

(load-word PTR) loads one word from the address PTR

`concat`

(concat X Y Z ...) concatenates words X, Y, Z, ... into one big word

`cast-high`

(cast-high S X) extracts high S bits from X.