BASE2 .0

Application Programming Interfaces (API) 64-Bit

Parameters

Args	Param	If lesser units required
Arg1	RAX	EAX/AX/AL
Arg2	RBX	EBX/BX/BL
Arg3	RCX	ECX/CX/CL
Arg4	RDX	EDX/DX/DL
Arg5	RDI	EDI/DI/DIL

Return(s)

RAX(1) RBX(2)

LISTING**

prnreg	prnregdu	prnregd
prnregou	prnrego	dumpreg
dumpregdu	dumpregd	dumprego
dumpregou	dumpregb	dumpseg
flags**	opsize	opcode
stackview	memview	memviewn
memviewc	memviewb	mem_reset
mem_set	mem_swap	mem_insertr
mem_insert	mem_copy	mem_lcopy
mem_tofile	prnintu	prnint
prnhexu	prnhex	prnoct
prnoctu	prnbinu	prnbin
prnbinb	prnint128u	prnoct128u
prnhex128u	fpbin	fpbind
fpbinx		
prndbl	prndble	prnflt
prnflte	prndblx	dblsplit
decodeBASE4	encodeBASE64	int2str
dec2str	dec2stru	oct2str
oct2stru	hex2str	hex2stru
bin2str	bin2stru	dbl2str
dble2str	flt2str	flte2str
dblx2str	dumpmmxrf	dumpmmxf
dumpmmxr	dumpmmx	dumpxmmrf
dumpxmmf	dumpxmmr	dumpxmm
dumpymmrf	dumpymmf	dumpymmr
dumpymm	prndblf	prnfltf
prnhexbf	prnintuf	prnintf
prnintd	prnintdf	prnintw
prnintwf	prnintb	prnintbf
clearxmm	clearymm	sse_flags
sse_round	rstmxcsr	fpdinfo
fpfinfo	fpxinfo	fpu_stack
fpu_sflag	fpu_cflag	fpu_tag
fpu_reg	fpu_copy	fpu_precision
fpu_round	rndint	randm
randq	factorial	add128
mul128	powint	addf
subf	mulf	divf
sine	tangent	cosine
sincos	atangent	secant
cosecant	cotangent	rad2deg
deg2rad	fpow	g_ratio
lg10	ln10	isintREAL4
isintREAL8	iseven	isodd
bconv	bitfield	str2int
str2dbl	str2flt	dbl2int
dblt2int	flt2int	fltt2int
int2dbl	readint	readdbl
readflt	chr_isdigit	chr_isalpha
chr_islower	chr_isupper	chr_change
chr_chcase	chr_tolower	chr_toupper
chr_find	chr_count	chr_shuffle
byte_count	ascii	str_trim
str_wordcnt	str_token	str_copy
str_cmpz	str_cmp	str_find
str_findz	str_append	str_appendz
str_reverse	str_reverse2	str_tolower
str_toupper	str_length	str_lengthd
str_length2	digitprob	digitprobu
digithprob	digithprobu	digitscan
digitscanu	digithscan	digithscanu
digitcount	digitcountu	digithcount
digithcountu	scan_qword	scan_dword
scan_word	scan_byte	
sort_byte	sort_int	sort_intu
sort_dbl	sort_dblx	sort_flt
aprnhbyte	aprnhbyter	
aprnbyte	aprnbyteu	aprnbyter
aprnbyteur	aprnword	aprnwordu
aprnwordr	aprnwordur	aprnintd
aprnintdu	aprnintdr	aprnintdur
aprnint	aprnintu	aprnintr
aprnintur	aprnflt	aprnfltr
aprndbl	aprndblr	aprndblx
aprndblxr	aprnhbytef	aprnhbyterf
aprnbytef	aprnbyteuf	aprnbyterf
aprnbyteurf	aprnwordf	aprnworduf
aprnwordrf	aprnwordurf	aprnintdf
aprnintduf	aprnintdrf	aprnintdurf
aprnintf	aprnintuf	aprnintrf
aprninturf	aprnfltf	aprnfltrf
aprndblf	aprndblrf	

Kernel Specifics

prnchrp**	prnspaces**	prnstreg
prnlines**	prnline	prnspace
prnchrs	prnchar	prnchr
prnstrd	prnstrz	prnstr
readchr	readch	readstr
mem_alloc	mem_free	mem_alloc2
mem_free2	mem_load	file_new
file_open	file_read	file_write
file_close	file_size	file_copy
file_delete	timer_start	timer_stop
delay	get_ticks	halt
exitp	exitx	

** Arguments on stack (callee cleanup)

Note: This API is applicable to both Win64 and Linux64

prnreg(1)

Display 64-bit register
Display formatted hex
RAX | Register to display
Ret | -
Note| 16-digit Hex format
    | Unsigned

prnregdu(1)

Display unsigned 64-bit Register
RAX | Register to display
Ret | -
Note| 20-digit decimal format

prnregd(1)

Display signed 64-bit Register
RAX | Register to display
Ret | -
Note| 19-digit decimal format

prnregou(1)

Display Register in unsigned octal
RAX | Register to display
Ret | -
Note| -

prnrego(1)

Display Register in signed octal
RAX | Register to display
Ret | -
Note| -

dumpreg

Display registers dump (unsigned hex)
Arg |
Ret | -
Note| -

dumpregdu

Display registers dump (decimal)
arg |
Ret | -
Note| -

dumpregd

Display registers dump (signed int)
arg |
Ret | -
Note| -

dumprego

Display registers dump (signed oct)
arg |
Ret | -
Note| -

dumpregou

Display registers dump (unsigned oct)
Arg |
Ret | -
Note| -

dumpregb

Display registers dump (unsigned binary)
Arg |
Ret | -
Note| -

dumpseg

Display segment registers
Arg | -
Ret | -
Note| -

flags(1)*

Display RFLAG
Arg | pushfq
Ret | -
Note| -

opsize(2)/1

Get size between 2 labels
RBX | Label (reg64,add64)
RAX | Label (reg64,add64)
Ret | Size
Note| Order of label is irrelevant

opcode(2)

Encode instrns between 2 labels
RBX | Label 2
RAX | Label 1 (first/lead label)
Ret | -
Note| Label 1 must be the first

stackview(1)

Display stack
RAX | Number of items to view (+up,-down)
Ret | -
Note| content |hex_address
    | 1 item = 8 bytes implied
    | If +ve, last line is TOS
    | If -ve, First line is TOS

memview(2)

View memory dump + offsets
RBX | size (+up,-down)
RAX | label, address
Ret | -
Note| Leftmost is MSB
    | Content |Addr| HexOffs|DecOffs

memviewn(2)

View memory dump in Little-Endian layout
RBX | size (+up,-down)
RAX | label, address
Ret | -
Note| -

memviewc(2)

View memory dump (strings) + offsets
RBX | size (+up,-down)
RAX | label, address
Ret | -
Note| Leftmost is MSB
    | Content |Addr| HexOffs|DecOffs

memviewb(3)

View memory dump in flat layout
RAX | Type: 0-Hex, 1=Decimal
RBX | +size
RAX | label, address
Ret | -
Note| -

mem_reset(2)

Clear memory of RBX bytes
RBX | number of bytes to clear
RAX | offset / location
Ret | -
Note| Clear by bytes

mem_set(3)

Set memory with arg3 byte

```
CL  | Set byte to use
RBX | size of bytes to set
RAX | offset / pointer
Ret | -
Note| Set by bytes
```

mem_swap(2)

Swap two memory content
Swap to array elements

```
RBX | Address of item
RAX | Address of item
Ret | -
Note| Could use LEA / OFFSET
```

mem_insertr(2)

Insert register content into memory

```
RBX | Value to insert
RAX | Memory location
Ret | -
Note| -
```

mem_insert(4)

Insert bytes into memory

```
RDX | Num of source bytes to write
RCX | Source Address
RBX | Destn offsets to write. 0=from start
RAX | Destn address
Ret | -
Note| Overwrites
```

mem_copy(3)

Copy memory content of R8 bytes

```
RCX | Number of bytes to copy
RBX | Source pointer
RAX | Destination pointer
Ret | -
Note| -
```

mem_lcopy(3)

Copy memory content of R8 bytes

```
RCX | Number of bytes to copy (Large n)
RBX | Source pointer
RAX | Destination pointer
Ret | -
Note| -
```

mem_tofile(3)

Copy memory content to a new file

```
RCX | Address of new filename
RBX | Size to copy
RAX | Address of memory source
Ret | A new filled file
Note| Must be a new file or else...
```

prnintu(1)

Display unsigned Decimal

```
RAX | Value to display
Ret | -
Note| -
```

prnint(1)

Display Signed Decimal Integer

```
RAX | Value to display
Ret | -
Note| -
```

prnhexu(1)

Display Unsigned Hexadecimal

```
RAX | Value to display
Ret | -
Note| Imm. must be in valid format
```

prnhex(1)

Display Signed Hexadecimal

```
RAX | Value to display
Ret | -
Note| Imm must be in valid format
```

prnoct(1)

Display Octal (signed)

```
RAX | Value to display
Ret | -
Note| Imm. must be in valid format
```

prnoctu(1)

Display Octal (unsigned)

```
RAX | Value to display.
Ret | -
Note| Imm. must be in valid format
```

prnbinu(1)

Display unsigned Binary

```
RAX | Value to display
Ret | -
Note| Imm. must be in valid format
```

prnbin(1)

Display Signed Binary

```
RAX | Value to display.
Ret | -
Note| Imm. must be in valid format
```

prnbinb(1)

Display Unsigned Binary with separators

```
RAX | Value to display
Ret | -
Note| Imm must be in valid format
    | Full display
```

prnint128u(1)

Display Unsigned 128-bit integer

```
RAX | Addr of 128-bit value to display
Ret | -
Note| Two consecutive QWORDS in memory
```

prnoct128u(1)

Display Unsigned 128-bit octal integer

```
RAX | Addr of 128-bit value to display
Ret | -
Note| Two consecutive QWORDS in memory
```

prnhex128u(1)

Display Unsigned 128-bit Hex

```
RAX | Addr of 128-bit value to display
Ret | -
Note| Two consecutive QWORDS in memory
```

fpbin(1)

Display REAL8 Floating Point Binary

```
RAX | REAL8 value to display
Ret | -
Note| -
```

fpbind(1)

Display REAL4 Floating Point Binary

```
EAX | REAL4 value to display
Ret | -
Note| -
```

fpbinx(1)

Display REAL10 Floating Point Binary

RAX | Address of the REAL10
Ret | -
Note| DT type.

prndbl(1)

Display double precision (REAL8)

RAX | REAL8 value to display
Ret | -
Note| Var should be init as FP (0.0)
| Var must be of type DQ.

prndble(2)

Display REAL8 with decimal places

RBX | Decimal places (1-15)
RAX | REAL8 value to display
Ret | -
Note| Var should be init as FP (0.0)
| Variable must be of type DQ.

prnflt(1)

Display single precision (REAL4)

EAX | REAL4 value to display
Ret | -
Note| Var should be initd to FP (0.0)

prnflte(2)

Display single precision (REAL4) with decimal places

RBX | Decimal places (1-7)
EAX | REAL4 value to display
Ret | -
Note| Var should be initd to FP (0.0)
| EAX is of type DD.

prndblx(1)

Display extended precision (REAL10)

RAX | The address of a DT in memory
Ret | -
Note| Var should be initd to FP (0.0)
| Variable must be of type DT.
| Displays unrounded precision

dblsplit(1)/2

Split a REAL8 into parts

RAX | REAL8 value
Ret | Integral part in RAX
| Fraction part in RBX
Note| Deals normal FP value only

decodeBASE64(2)

Decode 0-ended BASE64 to ASCII

RBX | Address of destination buffer
RAX | Address of string to decode
Ret | -
Note| RBX at least 2/3 the size of RAX
| Returned string are null-ended
| RFC 4648

encodeBASE64(2)

Encode 0-ended ASCII to BASE64 (RFC 4648)

RBX | Address of destination buffer
RAX | Address of string to encode
Ret | -
Note| RBX twice the size of RAX
| Both strings are null-ended

int2str(4)

Convert integer to string

RDX | Signness. 0=unsigned. 1=signed
RCX | Target base (2 to 36 only)
RBX | Buffer's address
RAX | Value
Ret | String in the sent buffer
Note| Buffer size must reflect # of digits
| String will be 0-ended

dec2str(2)

Convert signed int to 0-ended decimal string

RBX | Buffer's address
RAX | Integer to convert
Ret | String in buffer
Note| Buffer must be large enough

dec2stru(2)

Convert unsigned int to 0-ended dec string

RBX | Buffer's address
RAX | Integer to convert
Ret | String in buffer
Note| Buffer must be large enough

oct2str(2)

Convert signed octal to 0-ended string

RBX | Buffer's address
RAX | Integer to convert
Ret | String in buffer
Note| Buffer must be large enough

oct2stru(2)

Convert unsigned octal to 0-ended octal string

RBX | Buffer's address
RAX | Integer to convert
Ret | String in buffer
Note| Buffer must be large enough

hex2str(2)

Convert signed int to 0-ended hex string

RBX | Buffer's address
RAX | Integer to convert
Ret | String in buffer
Note| Buffer must be large enough

hex2stru(2)

Convert unsigned int to 0-ended hex string

RBX | Buffer's address
RAX | Integer to convert
Ret | String in buffer
Note| Buffer must be large enough

bin2str(2)

Convert signed bin to 0-ended bin string

RBX | Buffer's address
RAX | Integer to convert
Ret | String in buffer
Note| Buffer must be large enough (>66)

bin2stru(2)

Convert unsigned bin to 0-ended bin string

RBX | Buffer's address
RAX | Integer to convert
Ret | String in buffer
Note| Buffer must be large enough (>64)

dbl2str(2)

Convert REAL8 to 0-ended string

```
RBX | Buffer's address to save to
RAX | REAL8 value
Ret | 0-ended string in sent buffer.
Note| Buffer > 24
```

dble2str(3)

Convert REAL8 with decimal places to 0-ended string

```
RCX | Buffer's address to save to
RBX | Decimal places (1 to 15)
RAX | REAL8 value
Ret | 0-ended string in sent buffer.
Note| Buffer > 24
```

flt2str(2)

Convert REAL4 to 0-ended string

```
RBX | Buffer's address to save to
EAX | REAL4 value
Ret | 0-ended string in sent buffer.
Note| Buffer > 15
```

flte2str(3)

Convert REAL4 with decimal places to 0-ended string

```
RCX | Buffer's address to save to
RBX | Decimal places. (1-7)
EAX | REAL4 value
Ret | 0-ended string in sent buffer.
Note| Buffer > 24
```

dblx2str(2)

Convert REAL10 to 0-ended string

```
RBX | Buffer address to save to
RAX | Address of a DT to convert
Ret | 0-ended string in sent buffer.
Note| RAX must point to type DT.
    | Buffer >= 32
```

dumpmmxrf(1)

Dump MMX registers,reversed view. Formatted.

```
RAX | Type of display:
       0 - Unsigned integer BYTE
       1 - Signed integer BYTE
       2 - Unsigned integer WORD
       3 - Signed integer WORD
       4 - Unsigned integer DWORD
       5 - Signed integer DWORD
       6 - Unsigned integer QWORD
       7 - Signed integer QWORD
       8 - Unsigned hex BYTES
Note| -
```

dumpmmxf(1)

Dump MMX registers. Formatted.

```
RAX | Type of display:
       0 - Unsigned integer BYTE
       1 - Signed integer BYTE
       2 - Unsigned integer WORD
       3 - Signed integer WORD
       4 - Unsigned integer DWORD
       5 - Signed integer DWORD
       6 - Unsigned integer QWORD
       7 - Signed integer QWORD
       8 - Unsigned hex BYTES
Note| -
```

dumpmmxr(1)

Dump MMX registers,reversed view

```
RAX | Type of display:
       0 - Unsigned integer BYTE
       1 - Signed integer BYTE
       2 - Unsigned integer WORD
       3 - Signed integer WORD
       4 - Unsigned integer DWORD
       5 - Signed integer DWORD
       6 - Unsigned integer QWORD
       7 - Signed integer QWORD
       8 - Unsigned hex BYTES
Note| -
```

dumpmmx(1)

Dump MMX registers

```
RAX | Type of display:
       0 - Unsigned integer BYTE
       1 - Signed integer BYTE
       2 - Unsigned integer WORD
       3 - Signed integer WORD
       4 - Unsigned integer DWORD
       5 - Signed integer DWORD
       6 - Unsigned integer QWORD
       7 - Signed integer QWORD
       8 - Unsigned hex BYTES
Note| -
```

dumpxmmrf(1)

Dump XMM registers,reversed view. Formatted.

```
RAX | Type of display:
       0 - Unsigned integer BYTE
       1 - Signed integer BYTE
       2 - Unsigned integer WORD
       3 - Signed integer WORD
       4 - Unsigned integer DWORD
       5 - Signed integer DWORD
       6 - Unsigned integer QWORD
       7 - Signed integer QWORD
       8 - Singles
       9 - Doubles
       10- Unsigned hex BYTES
Note| -
```

dumpxmmf(1)

Dump XMM registers. Formatted.

```
RAX | Type of display:
       0 - Unsigned integer BYTE
       1 - Signed integer BYTE
       2 - Unsigned integer WORD
       3 - Signed integer WORD
       4 - Unsigned integer DWORD
       5 - Signed integer DWORD
       6 - Unsigned integer QWORD
       7 - Signed integer QWORD
       8 - Singles
       9 - Doubles
       10- Unsigned hex BYTES
Note| -
```

dumpxmmr(1)

Dump XMM registers,reversed view

```
RAX | Type of display:
       0 - Unsigned integer BYTE
       1 - Signed integer BYTE
       2 - Unsigned integer WORD
       3 - Signed integer WORD
       4 - Unsigned integer DWORD
       5 - Signed integer DWORD
       6 - Unsigned integer QWORD
       7 - Signed integer QWORD
       8 - Singles
       9 - Doubles
       10- Unsigned hex BYTES
Note| -
```

dumpxmm(1)

Dump XMM registers

```
RAX | Type of display:
       0 - Unsigned integer BYTE
       1 - Signed integer BYTE
       2 - Unsigned integer WORD
       3 - Signed integer WORD
       4 - Unsigned integer DWORD
       5 - Signed integer DWORD
       6 - Unsigned integer QWORD
       7 - Signed integer QWORD
       8 - Singles
       9 - Doubles
       10- Unsigned hex BYTES
Note| -
```

dumpymmrf(1)

Dump YMM registers,reversed view. Formatted.

```
RAX | Type of display:
       0 - Unsigned integer BYTE
       1 - Signed integer BYTE
       2 - Unsigned integer WORD
       3 - Signed integer WORD
       4 - Unsigned integer DWORD
       5 - Signed integer DWORD
       6 - Unsigned integer QWORD
       7 - Signed integer QWORD
       8 - Singles
       9 - Doubles
       10- Unsigned hex BYTES
Note| Expects AVX CPU.
Note| Byte options need wide display
```

dumpymmf(1)

Dump YMM registers. Formatted.

```
RAX | Type of display:
       0 - Unsigned integer BYTE
       1 - Signed integer BYTE
       2 - Unsigned integer WORD
       3 - Signed integer WORD
       4 - Unsigned integer DWORD
       5 - Signed integer DWORD
       6 - Unsigned integer QWORD
       7 - Signed integer QWORD
       8 - Singles
       9 - Doubles
       10- Unsigned hex BYTES
Note| Expects AVX CPU
    | Byte options need wide display
```

dumpymmr(1)

Dump YMM registers,reversed view

```
RAX | Type of display:
       0 - Unsigned integer BYTE
       1 - Signed integer BYTE
       2 - Unsigned integer WORD
       3 - Signed integer WORD
       4 - Unsigned integer DWORD
       5 - Signed integer DWORD
       6 - Unsigned integer QWORD
       7 - Signed integer QWORD
       8 - Singles
       9 - Doubles
       10- Unsigned hex BYTES
Note| Expects AVX CPU
    | Byte options need wide display
```

dumpymm(1)

Dump YMM registers

```
RAX | Type of display:
       0 - Unsigned integer BYTE
       1 - Signed integer BYTE
       2 - Unsigned integer WORD
       3 - Signed integer WORD
       4 - Unsigned integer DWORD
       5 - Signed integer DWORD
       6 - Unsigned integer QWORD
       7 - Signed integer QWORD
       8 - Singles
       9 - Doubles
       10- Unsigned hex BYTES
Note| Expects AVX CPU
    | Byte options need wide display
```

prndblf(1)

Display Double-precision. Formatted

```
RAX | Double value to display.
Ret | -
Note| Internal use
```

prnfltf(1)

Display Single-precision. Formatted

```
EAX | Single value to display.
Ret | -
Note| Internal use
```

prnhexbf(1)

Display Unsigned Hex Byte (formatted)

```
AL  | Byte value to display.
Ret | -
```

prnintuf(1)

Display Unsigned 64-bit Decimal. Formatted

```
RAX | Qword Value to display.
Ret | -
Note| Internal use
```

prnintf(1)

Display Signed 64-bit Decimal. Formatted

```
RAX | Qword Value to display.
Ret | -
Note| Internal use
```

prnintd(2)

Display 32-bit Decimal

```
RBX | 0-unsigned. 1-signed
EAX | Dword Value to display.
Ret | -
Note| Internal use
```

prnintdf(2)

Display 32-bit Decimal. Formatted

```
RBX | 0-unsigned. 1-signed
EAX | Dword Value to display.
Ret | -
Note| Internal use
```

prnintw(2)

Display 16-bit Decimal

```
RBX | 0-unsigned. 1-signed
AX  | Word value to display.
Ret | -
Note| Internal use
```

prnintwf(2)

Display 16-bit Decimal (formatted)

```
RBX | 0-unsigned. 1-signed
AX  | Word value to display
Ret | -
Note| Internal use
```

prnintb(2)

Display Byte Decimal

```
RBX | 0-unsigned. 1-signed
AL  | Byte value to display.
Ret | -
Note| Internal use
```

prnintbf(2)

Display Byte Decimal. Formatted)

```
RBX | 0-unsigned. 1-signed
AL  | Byte value to display.
Ret | -
Note| Internal use
```

clearxmm

Clear all XMM registers

```
Arg | -
Ret | -
Note| -
```

clearymm

Clear all YMM registers

```
Arg | -
Ret | -
Note| Expects AVX CPU
```

sse_flags

Display MXCSR register

```
Arg | -
Ret | -
Note| -
```

sse_round(1)

Set SSE rounding control

```
RAX | Rounding mode
      0 = near
      1 = down
      2 = up
      3 = zero
Ret | -
Note| -
```

rstmxcsr

Reset SIMD Flags register (mxcsr) to CPU default

```
Arg | -
Ret | -
Note| -
```

fpdinfo(1)

Display Double FP information

```
RAX | REAL8 value
Note| -
Ret | -
```

fpfinfo(1)

Display Float FP information

```
EAX | REAL4 value
Ret | -
Note| -
```

fpxinfo(1)

Display REAL10 FP information

```
RAX | Address of a REAL10
Ret | -
Note| -
```

fpu_stack

Display FPU stack

```
Arg | -
Ret | -
Note| Display is real10
```

fpu_sflag

Display FPU Status Flag

```
Arg | -
Ret | -
Note| -
```

fpu_cflag

Display FPU Control Flag

```
Arg | -
Ret | -
Note| -
```

fpu_tag

Display FPU Tag Register

```
Arg | -
Ret | -
Note| 00 = valid
    | 01 = zero
    | 10 = invalid
    | 11 = empty
```

fpu_reg(1)

Display one FPU stack register

```
RAX | Stack # to display (0-7)
Ret | -
Note| Special values show nothing
```

fpu_copy(1)/1

Copy a FPU register

```
RAX | Stack # to copy (0-7)
Ret | Copied value in RAX
Note| -
```

fpu_precision(1)

Set FPU precision control

```
RAX | Precision mode
      0 = real8
      1 = real10
      2 = real4
Ret | -
Note| FSAVE/FINIT will reset it
```

fpu_round(1)

Set FPU rounding control

```
RAX | Rounding mode
      0 = near
      1 = down
      2 = up
      3 = zero
Ret | -
Note| FSAVE/FINIT will reset it
```

rndint/1

Generate 1 pseudorandom integer

```
Arg | -
Ret | Random integer in RAX
Note| -
```

randm(1)/1

Get 1 unsigned random integer

```
RAX | MAX value (0-MAX)
Ret | Random integer in RAX
Note| 0, MAX inclusive
```

randq(2)

Generate unique random integers

```
from 0 to MAX-1 and save in array
RBX | Addr of array initialized to -1
RAX | MAX value
Ret | Random elements in array
Note| Example: MAX = 50
      elems dq 50 dup(-1)
      elems: TIMES 50 dq -1
```

factorial(1)/1

Get factorial

```
RAX | +Value
Ret | Factorial in RAX
Note| Limit to 20! Only
```

add128(3)

Add two 64-bit values and keep result in buffer. Unsigned

```
RCX | Address of buffer (16 bytes)
RBX | Value
RAX | Value
Ret | -
Note| Use prnint128u to display it
```

mul128(3)

Multiply two 64-bit values and keep result in buffer. Unsigned

```
RCX | Address of buffer (32 bytes)
RBX | Multiplier
RAX | Multiplicand
Ret | -
Note| Use prnint128u to display it
```

powint(2)/1

Calculate 64-bit integral power

```
RBX | +Exponent/power (int)
RAX | Base (int)
Ret | Result in RAX
    | # if overflow
Note| Var must be of type DQ
    | Takes +exponent only
```

addf(2)/1

Add two floating points

```
RBX | Value2 in FP format
RAX | Value1 in FP format
Ret | Double value in RAX
Note| RAX=RAX+RBX
```

subf(2)/1

Sub two floating points

```
RBX | Value2 in FP format
RAX | Value1 in FP format
Ret | Double value in RAX
Note| RAX=RAX-RBX
```

mulf(2)/1

Multiply two floating points

```
RBX | Value2 in FP format
RAX | Value1 in FP format
Ret | Double value in RAX
Note| RAX=RAX*RBX
```

divf(2)/1

Divide two floating points

```
RBX | Value2 in FP format (divisior)
RAX | Value1 in FP format (dividend)
Ret | Double value in RAX
Note| RAX=RAX/RBX
```

sine(1)/1

Get sine radian

```
RAX | FP value in radian
Ret | sine in RAX
Note| -
```

tangent1)/1

Get tangent radian

```
RAX | FP value in radian
Ret | tangent in RAX
Note| -
```

cosine(1)/1

Get cosine radian

```
RAX | FP value in radian
Ret | cosine in RAX
Note| -
```

sincos(1)/2

Get sine & cosine radian

```
RAX | FP value in radian
Ret | Sine in RAX
    | Cosine in RBX
Note| -
```

atangent(1)/1

Get arc-tangent radian

```
RAX | FP value in radian
Ret | Arctangent in RAX
Note| -
```

secant(1)/1

Get secant radian

```
RAX | FP value in radian
Ret | Secant in RAX
Note| -
```

cosecant(1)/1

Get cosecant radian

```
RAX | FP value in radian
Ret | Cosecant in RAX
Note| -
```

cotangent(1)/1

Get cotangent radian

```
RAX | FP value in radian
Ret | cotangent in RAX
Note| -
```

rad2deg(1)/1

Convert Radian to Degree

```
RAX | REAL8 value in RADIAN
Ret | DEGREE value in RAX
Note| Var should be init as FP (0.0)
```

deg2rad(1)/1

Convert Degree to Radian

```
RAX | REAL8 value in DEGREE
Ret | RADIAN value
Note| Var should be init as FP (0.0)
```

fpow(2)/1

Calculate x^n

```
RBX | REAL8 power
      0.3333333333333333 for cubic root
      or 3FD5555555555555h
RAX | +REAL8 Base
Ret | Raised value in RAX (Double)
Note| Args. must be in doubles
    | base must be positive
```

g_ratio(2)/1

Find golden ratio for multi ops

```
RBX | 0 - mul
      1 - div
      2 - pow
RAX | REAL8 value
Ret | golden ratio
Note| 1.618033988749894;848
    | RAX = RAX <op> G.ratio
```

lg10(1)/1

Find Common Log of Base 10

```
RAX | +REAL8 value
Ret | Value in RAX
Note| -
```

ln10(1)/1

Find Natural Log

```
RAX | +REAL8 value
Ret | Value in RAX
Note| -
```

isintREAL4(1)/1

Check if a REAL4 is a qualified integer

```
EAX | REAL4 value
Ret | RAX: 0-no,1-yes
Note| Test normal FP values only
```

isintREAL8(1)/1

Check if a REAL8 is a qualified integer

```
RAX | REAL8 value
Ret | RAX: 0-no,1-yes
Note| Test normal FP values only
```

iseven(1)/1

Check if even number

```
RAX | Integer to check
Ret | 0 if no. 1 if yes
Note|
```

isodd(1)/1

Check if odd number

```
RAX | Integer to check
Ret | RAX - 0 if no. 1 if yes
```

bconv(2)

Convert and display integer

```
RBX | Display base to use (2 to 36)
RAX | Integer to convert
Ret | -
Note| Imm value is limited to 32-bit
```

bitfield(3)

Display a bitfield from a 64-bit data

```
RCX | lower bit
RBX | higher bit
RAX | Value to be extracted from
Ret | -
Note| Display bit range
```

str2int(1)/1

Convert 0-ended string to integer

```
RAX | Address of the 0-ended string
Ret | Integer in RAX
Note| Expected Suffix- b(bin), d(dec)
    | h(hex), o,q(octal). 'd' is optional
    | String must be of type DB
    | Digits string must be valid
```

str2dbl(1)/1

Convert string to double

```
RAX | Address of a 0-ended string
Ret | Double-precision RAX
Note| String must be of type DB
```

str2flt(1)/1

Convert string to float

```
RAX | Address of a 0-ended string
Ret | Single-precision in EAX
Note| Limit digits 7
    | String must be of type DB
```

dbl2int(2)/1

Convert REAL8 to integer

```
RBX | Rounding mode:
       0-near
       1-down
       2-up
       3-zero
RAX | REAL8 value
Ret | Integer in RAX
Note| Input must be in FP format
```

dblt2int(1)/1

Convert REAL8 to integer, truncate to 0

```
RAX | REAL8 value
Ret | Integer in RAX
Note| Input must be in FP format
```

flt2int(2)/1

Convert REAL4 to integer

```
RBX | Rounding mode:
       0-Near
       1-Down
       2-Up
       3-Zero
EAX | REAL4 value
Ret | Integer in EAX
Note| Input must be in FP format
```

fltt2int(1)/1

Convert REAL4 to integer, truncate to 0

```
EAX | REAL4 value
Ret | Integer in EAX
Note| Input must be in FP format
```

int2dbl(1)/1

Convert integer to real

```
RAX | Integer value to convert
Ret | Real in RAX
Note| -
```

readint/1

Get multi suffix integer from keyboard

```
Arg | -
Ret | Integer in RAX
Note| Expected Suffix- b(binary), d(dec)
    | h(hex), o,q(octal). d is optional
    | Digits must be in valid format
```

readdbl/1

Get double-precision from stdin

```
Arg | -
Ret | Double precision in RAX
Note| Doesn't take E format
    | Must use FP format for input
```

readflt/1

Get single-precision from stdin

```
Arg | -
Ret | Single precision in EAX
Note| Doesn't take E format
    | Must use FP format for input
```

chr_isdigit(1)/1

Check if alphanumeric digit

```
AL  | Byte char to check
Ret | RAX - 0 if no. 1 if yes
Note| Digit '0' to '9' only.
```

chr_isalpha(1)/1

Check if alphabet

```
AL  | Byte char to check
Ret | RAX - 0 if no. 1 if yes
Note| 'A'-'Z' and 'a'-'z'
```

chr_islower(1)/1

Check if lowercase

```
AL  | Byte char to check
Ret | RAX - 0 if no. 1 if yes
Note|
```

chr_isupper(1)/1

Check if uppercase

```
AL  | Byte char to check
Ret | RAX - 0 if no. 1 if yes
Note| -
```

chr_change(3)

Change a char from a 0-ended string

```
CL  | Char byte to use
BL  | Char byte to change
RAX | Address of string
Ret | -
Note| -
```

chr_chcase(1)/1

Alternate character case

```
AL  | Byte char to change
Ret | Modified char
Note| -
```

chr_tolower(1)/1

Change to lowercase

```
AL  | Byte to change
Ret | Modified char
Note| -
```

chr_toupper(1)/1

Change to uppercase

```
AL  | Character to change
Ret | Modified char
Note| -
```

chr_find(2)/1

Find byte from a 0-ended string

```
BL  | Character to find
RAX | String address to look from
Ret | Index in RAX. -1 if not found
Note| Will stop at first one found
```

chr_count(2)/1

Count a char from a 0-ended string

```
BL  | Character to count
RAX | String's address
Ret | Count in RAX. 0 if none
Note| -
```

chr_shuffle(1)

Shuffle a 0-ended string

```
RAX | String's address
Ret | Same string shuffled
Note| -
```

byte_count(3)/1

Count byte recurrences

```
CL  | The byte to count
RBX | Size of source buffer
RAX | Source address
Ret | Count in RAX. 0 if none
Note| -
```

ascii(1)

Display ASCII equivalences

```
AL  | Char or Hex value to display
Ret | -
Note| -
```

str_trim(2)

Trim a 0-ended string

```
RBX | # of chars to cut
RAX | Address of the 0-ended string
Ret | The same string trimmed
Note| -
```

str_wordcnt(1)/1

Count words of a 0-ended string

```
RAX | Address of string
Ret | Word count
Note| Single char is a word
```

str_token(2)

Display tokens off a 0-ended string

```
RBX | Addr of 0-ended delimiter string
RAX | Addr of the string
Ret | -
Note| eg, delimiter db '? .-',0
```

str_copy(3)

Copy a string to another array

```
RCX | Size of bytes to copy
RBX | Source string's address
RAX | Target buffer's address
Ret | Copied string at the sent address
Note| Target must fit the size to copy
    | Target buffer is 0-ended
```

str_cmpz(2)/1

Compare two 0-ended strings

```
RBX | String 1 address
RAX | String 2 address
Ret | RAX = 1 if Equal, 0 if not
Note| The two strings mus be 0-ended
```

str_cmp(3)/1

Compare strings with size

```
RCX | Size of bytes to compare
RBX | String 1 address
RAX | String 2 address
Ret | RAX = 1 if Equal, 0 if not
Note| -
```

str_find(3)/1

Search for a sub-string

```
RCX | Size of key string
RBX | Addr of key string
RAX | Addr of 0-ended source string
Ret | Address. -1 -Not found
Note| Case-sensitive
```

str_findz(2)/1

Search for a sub-string from a C string

```
RBX | Addr of 0-ended search/key string
RAX | Addr of 0-ended source string
Ret | Location. -1 = Not found
Note| Case-sensitive
```

str_append(5)/1

Combine two 0-ended strings with size

```
RDI | # of bytes to copy from str2
DL  | Separator byte. 0-if none
RCX | Addr of buffer
RBX | Addr of str2
RAX | Addr of str1 (0-ended)
Ret | 0-ended combined string
    | RAX - Combined string length
Note| 1st string + 2nd string
    | Buffer must be large enough
```

str_appendz(4)/1

Combine two 0-ended strings

```
DL  | Separator byte. 0-if none
RCX | Addr of buffer
RBX | Addr of second string
RAX | Addr of first string
Ret | 0-ended combined string
    | RAX - Combined string length
Note| 1st string + 2nd string
    | Buffer must be large enough
```

str_reverse(1)

Reverse 0-ended string

```
RAX | Address of the 0-ended string
Ret | The same string reversed
Note| -
```

str_reverse2(1)

Reverse 0-ended aligned-16 string

```
RAX | Address of the 0-ended string
Ret | The same string reversed
Note| String must be align16
```

str_tolower(1)

Change 0-ended string to lower case

```
RAX | Address of the 0-ended string
Ret | The same string converted
Note| -
```

str_toupper(1)

Change 0-ended string to upper case

```
RAX | Address of the 0-ended string
Ret | The same string converted
Note| -
```

str_length(1)/1

Find length of a 0-ended string

```
RAX | String's address
Ret | size in RAX.
Note| -
```

str_lengthd(2)/1

Get length of a 0-ended string with delimiter

```
BL  | Delimiter byte
RAX | String's address
Ret | size in RAX
Note| -
```

str_length2(1)/1

Find length of a 0-ended string (aligned)

```
RAX | String's address
Ret | size in RAX.
Note| String must be aligned 16
```

digitprob(2)/1

Probe decimal digit at specified location of a signed decimal value.

```
RBX | Digit position to probe (#1=MSD)
RAX | The decimal value to probe from
Ret | Decimal digit. -1 = Error
Note| -
```

digitprobu(2)/1

Probe decimal digit at specified location of an unsigned decimal value.

```
RBX | Digit position to probe (#1=MSD)
RAX | The decimal value to probe from
Ret | Decimal digit. -1 = Error
Note| -
```

digithprob(2)/1

Probe hex digit at specified location of a signed hexadecimal value.

```
RBX | Digit position to probe (#1=MSD)
RAX | The decimal value to probe from
Ret | Hex digit. -1 = Error
Note| -
```

digithprobu(2)/1

Probe hex digit at specified location of an unsigned hexadecimal value.

```
RBX | Digit position to probe (#1=MSD)
RAX | The decimal value to probe from
Ret | Hex digit. -1 = Error
Note| -
```

digitscan(2)/1

Find a decimal digit from a signed decimal value.

```
RBX | Decimal digit to find
RAX | The decimal value to find from
Ret | Digit position. #1 = MSD. -1 = Error
Note| -
```

digitscanu(2)/1

Find a decimal digit from a unsigned decimal value.

RBX | Decimal digit to find
RAX | The decimal value to find from
Ret | Digit position. #1 = MSD. -1 = Error
Note| -

digithscan(2)/1

Find a hex digit from a signed hexadecimal value.

RBX | Hex digit to find
RAX | The hex value to find from
Ret | Digit position. #1 = MSD. -1 = Error
Note| -

digithscanu(2)/1

Find a hex digit from a unsigned hexadecimal value.

RBX | Hex digit to find
RAX | The Hex value to find from
Ret | Digit position. #1 = MSD. -1 = Error
Note| -

digitcount(1)/1

Count the number of digits of a signed decimal value.

RAX | The decimal value to count
Ret | Digits count
Note| -

digitcountu(1)/1

Count the number of digits of an unsigned decimal value.

RAX | The decimal value to count
Ret | Digits count
Note| -

digithcount(1)/1

Count the number of digits of a signed hexadecimal value.

RAX | The hexadecimal value to count
Ret | Digits count
Note| -

digithcountu(1)/1

Count the number of digits of an unsigned hexadecimal value.

RAX | The hexadecimal value to count
Ret | Digits count
Note| -

scan_qword(3)/1

Find first matching quadword

RCX | Size of bytes to scan
RBX | The qword immediate
RAX | Starting location
Ret | Offset. -1 if not found
Note| -

scan_dword(3)/1

Find first matching dword

RCX | Size of bytes to scan
EBX | The dword immediate (DWORD)
RAX | Starting location
Ret | Offset. -1 if not found
Note| -

scan_word(3)/1

Find first matching two-byte

RCX | Size of bytes to scan
BX | The word immediate (WORD)
RAX | Starting location
Ret | Offset. -1 if not found
Note| -

scan_byte(3)/1

Find first matching byte

RCX | Size of bytes to scan
BL | The byte immediate (BYTE)
RAX | Starting location
Ret | Offset. -1 if not found
Note| -

sort_byte(3)

Sort char/byte array (signed)

RCX | 0-ascending, 1-descending
RBX | Number of array elements
RAX | Address of array (DB)
Ret | Return the sorted array
Note| Elements are of type DB

sort_int(3)

Sort array of signed int (bubble sort)

RCX | 0-ascending, 1-descending
RBX | Number of array elements
RAX | Address of array (DQ)
Ret | Return the sorted array
Note| Elements are of type DQ

sort_intu(3)

Sort array of unsigned int (bubble sort)

RCX | 0-ascending, 1-descending
RBX | Number of array elements
RAX | Address of array (DQ)
Ret | Return the sorted array
Note| Elements are of type DQ

sort_dbl(3)

Sort array of doubles (bubble sort)

RCX | 0-ascending, 1-descending
RBX | Number of array elements
RAX | Address of array (DQ)
Ret | Return the sorted array
Note| Elements are of type DQ

sort_dblx(3)

Sort array of REAL10 (bubble sort)

RCX | 0-ascending, 1-descending
RBX | Number of array elements
RAX | Address of array (DT)
Ret | Return the sorted array
Note| Elements are of type DT

sort_flt(2)

Sort array of singles (bubble sort)

RCX | 0-ascending, 1-descending
RBX | Number of array elements
RAX | Address of array (DD)
Ret | Return the sorted array
Note| Elements are of type DD

aprnhbyte(3)

Display array of unsigned hex bytes. Little-Endian

```
CL  | Separator byte. 0 if none
RBX | Number of elements
RAX | Address of array
Ret | -
Note| Elements are of type DB
```

aprnhbyter(3)

Display array of unsigned bytes. Reversed-view

```
CL  | Separator byte. 0 if none
RBX | Number of elements
RAX | Address of array
Ret | -
Note| Elements are of type DB
```

aprnbyte(3)

Display array of signed bytes. Little-Endian

```
CL  | Separator byte. 0 if none
RBX | Number of elements
RAX | Address of array
Ret | -
Note| Elements are of type DB
```

aprnbyteu(3)

Display array of unsigned bytes. Little-Endian

```
CL  | Separator byte. 0 if none
RBX | Number of elements
RAX | Address of array
Ret | -
Note| Elements are of type DB
```

aprnbyter(3)

Display array of signed bytes. Reversed-view

```
CL  | Separator byte. 0 if none
RBX | Number of elements
RAX | Address of array
Ret | -
Note| Elements are of type DB
```

aprnbyteur(3)

Display array of unsigned bytes. Reversed-view

```
CL  | Separator byte. 0 if none
RBX | Number of elements
RAX | Address of array
Ret | -
Note| Elements are of type DB
```

aprnword(3)

Display array of signed words. Little-Endian

```
CL  | Separator byte. 0 if none
RBX | Number of elements
RAX | Address of array
Ret | -
Note| Elements are of type DW
```

aprnwordu(3)

Display array of unsigned words. Little-Endian

```
CL  | Separator byte. 0 if none
RBX | Number of elements
RAX | Address of array
Ret | -
Note| Elements are of type DW
```

aprnwordr(3)

Display array of signed words. Reversed-view

```
CL  | Separator byte. 0 if none
RBX | Number of elements
RAX | Address of array
Ret | -
Note| Elements are of type DW
```

aprnwordur(3)

Display array of unsigned words. Reversed-view

```
CL  | Separator byte. 0 if none
RBX | Number of elements
RAX | Address of array
Ret | -
Note| Elements are of type DW
```

aprnintd(3)

Display array of signed ints. Little-Endian

```
CL  | Separator byte. 0 if none
RBX | Number of elements
RAX | Address of array
Ret | -
Note| Elements are of type DD
```

aprnintdu(3)

Display array of unsigned ints. Little-Endian

```
CL  | Separator byte. 0 if none
RBX | Number of elements
RAX | Address of array
Ret | -
Note| Elements are of type DD
```

aprnintdr(3)

Display array of signed ints. Reversed-view

```
CL  | Separator byte. 0 if none
RBX | Number of elements
RAX | Address of array
Ret | -
Note| Elements are of type DD
```

aprnintdur(3)

Display array of unsigned ints. Reversed-view

```
CL  | Separator byte. 0 if none
RBX | Number of elements
RAX | Address of array
Ret | -
Note| Elements are of type DD
```

aprnint(3)

Display array of signed longs. Little-Endian

CL | Separator byte. 0 if none
RBX | Number of elements
RAX | Address of array
Ret | -
Note| Elements are of type DQ

aprnintu(3)

Display array of unsigned longs. Little-Endian

CL | Separator byte. 0 if none
RBX | Number of elements
RAX | Address of array
Ret | -
Note| Elements are of type DQ

aprnintr(3)

Display array of signed longs. Reversed view

CL | Separator byte. 0 if none
RBX | Number of elements
RAX | Address of array
Ret | -
Note| Elements are of type DQ

aprnintur(3)

Display array of unsigned longs. Reversed-view

CL | Separator byte. 0 if none
RBX | Number of elements
RAX | Address of array
Ret | -
Note| Elements are of type DQ

aprnflt(3)

Display array of REAL4s. Little-Endian

CL | Separator byte. 0 if none
RBX | Number of elements
RAX | Address of array
Ret | -
Note| Elements are of type DD/REAL4

aprnfltr(3)

Display array of REAL4s. Reversed view

CL | Separator byte. 0 if none
RBX | Number of elements
RAX | Address of array
Ret | -
Note| Elements are of type DD/REAL4

aprndbl(3)

Display array of REAL8s. Little-Endian

CL | Separator byte. 0 if none
RBX | Number of elements
RAX | Address of array
Ret | -
Note| Elements are of type DQ/REAL8

aprndblr(3)

Display array of REAL8s. Reversed view

CL | Separator byte. 0 if none
RBX | Number of elements
RAX | Address of array
Ret | -
Note| Elements are of type DQ/REAL8

aprndblx(3)

Display array of REAL10s. Little-Endian

CL | Separator byte. 0 if none
RBX | Number of elements
RAX | Address of array
Ret | -
Note| Elements are of type DT/REAL10

aprndblxr(3)

Display array of REAL10s. Reversed view

CL | Separator byte. 0 if none
RBX | Number of elements
RAX | Address of array
Ret | -
Note| Elements are of type DT/REAL10

aprnhbytef(3)

Display array of unsigned hex bytes. Little-Endian. Formatted.

CL | Separator byte. 0 if none
RBX | Number of elements
RAX | Address of array
Ret | -
Note| Elements are of type DB

aprnhbyterf(3)

Display array of unsigned bytes. Reversed-view. Formatted.

CL | Separator byte. 0 if none
RBX | Number of elements
RAX | Address of array
Ret | -
Note| Elements are of type DB

aprnbytef(3)

Display array of signed bytes. Little-Endian. Formatted.

CL | Separator byte. 0 if none
RBX | Number of elements
RAX | Address of array
Ret | -
Note| Elements are of type DB

aprnbyteuf(3)

Display array of unsigned bytes. Little-Endian. Formatted.

CL | Separator byte. 0 if none
RBX | Number of elements
RAX | Address of array
Ret | -
Note| Elements are of type DB

aprnbyterf(3)

Display array of signed bytes. Reversed-view. Formatted.

CL | Separator byte. 0 if none
RBX | Number of elements
RAX | Address of array
Ret | -
Note| Elements are of type DB

aprnbyteurf(3)

Display array of unsigned bytes. Reversed-view. Formatted.

CL | Separator byte. 0 if none
RBX | Number of elements
RAX | Address of array
Ret | -
Note| Elements are of type DB

aprnwordf(3)

Display array of signed words. Little-Endian. Formatted.

CL | Separator byte. 0 if none
RBX | Number of elements
RAX | Address of array
Ret | -
Note| Elements are of type DW

aprnworduf(3)

Display array of unsigned words. Little-Endian. Formatted.

CL | Separator byte. 0 if none
RBX | Number of elements
RAX | Address of array
Ret | -
Note| Elements are of type DW

aprnwordrf(3)

Display array of signed words. Reversed-view. Formatted.

CL | Separator byte. 0 if none
RBX | Number of elements
RAX | Address of array
Ret | -
Note| Elements are of type DW

aprnwordurf(3)

Display array of unsigned words. Reversed-view. Formatted.

CL | Separator byte. 0 if none
RBX | Number of elements
RAX | Address of array
Ret | -
Note| Elements are of type DW

aprnintdf(3)

Display array of signed ints. Little-Endian. Formatted.

CL | Separator byte. 0 if none
RBX | Number of elements
RAX | Address of array
Ret | -
Note| Elements are of type DD

aprnintduf(3)

Display array of unsigned ints. Little-Endian. Formatted.

CL | Separator byte. 0 if none
RBX | Number of elements
RAX | Address of array
Ret | -
Note| Elements are of type DD

aprnintdrf(3)

Display array of signed ints. Reversed-view. Formatted.

CL | Separator byte. 0 if none
RBX | Number of elements
RAX | Address of array
Ret | -
Note| Elements are of type DD

aprnintdurf(3)

Display array of unsigned ints. Reversed-view. Formatted.

CL | Separator byte. 0 if none
RBX | Number of elements
RAX | Address of array
Ret | -
Note| Elements are of type DD

aprnintf(3)

Display array of signed longs. Little-Endian. Formatted.

CL | Separator byte. 0 if none
RBX | Number of elements
RAX | Address of array
Ret | -
Note| Elements are of type DQ

aprnintuf(3)

Display array of unsigned longs. Little-Endian. Formatted.

CL | Separator byte. 0 if none
RBX | Number of elements
RAX | Address of array
Ret | -
Note| Elements are of type DQ

aprnintrf(3)

Display array of signed longs. Reversed view. Formatted.

CL | Separator byte. 0 if none
RBX | Number of elements
RAX | Address of array
Ret | -
Note| Elements are of type DQ

aprninturf(3)

Display array of unsigned longs. Reversed-view. Formatted.

CL | Separator byte. 0 if none
RBX | Number of elements
RAX | Address of array
Ret | -
Note| Elements are of type DQ

aprnfltf(3)

Display array of REAL4s. Little-Endian. Formatted.

CL | Separator byte. 0 if none
RBX | Number of elements
RAX | Address of array
Ret | -
Note| Elements are of type DD/REAL4

aprnfltrf(3)

Display array of REAL4s. Reversed view. Formatted.

CL | Separator byte. 0 if none
RBX | Number of elements
RAX | Address of array
Ret | -
Note| Elements are of type DD/REAL4

aprndblf(3)

Display array of REAL8s. Little-Endian. Formatted.

```
CL  | Separator byte. 0 if none
RBX | Number of elements
RAX | Address of array
Ret | -
Note| Elements are of type DQ/REAL8
```

aprndblrf(3)

Display array of REAL8s. Reversed view. Formatted.

```
CL  | Separator byte. 0 if none
RBX | Number of elements
RAX | Address of array
Ret | -
Note| Elements are of type DQ/REAL8
```

prnchrp(2)**

Display char pattern

```
push| Push the char to use
push| Push number of times to print
Ret | -
Note| -
```

prnspaces(1)*

Print multiple whitespace

```
push| Push number of whitespaces
Ret | -
Note| Arguments on stack
```

prnstreg(1)

Display short string off register RAX

```
RAX | The string
Ret | -
Note| -
```

prnlines(1)**

Print multiple lines

```
push| Push number of lines
Ret | -
Note| -
```

prnline

Print a newline

```
Arg | -
Ret | -
Note| -
```

prnspace

Print a whitespace

```
Arg | -
Ret | -
Note| -
```

prnchrs(1)**

Display a char byte from stack

```
push| Push the char value
Ret | -
Note| Use "qword" for DB variable
```

prnchar(1)

Display a char byte from memory

```
RAX | The char's address
Ret | -
Note| Use "qword" for DB var
```

prnchr(1)

Display a char byte in RAX

```
AL  | The char value
Ret | -
Note| -
```

prnstrd(2)

Display a delimiter-ended string

```
BL  | Delimiter byte value
RAX | The string address
Ret | -
Note| -
```

prnstrz(1)

Display a 0-ended string

```
RAX | The string address
Ret | -
Note| -
```

prnstr(2)

Display string with size

```
RBX | Size of bytes to display
RAX | The string address
Ret | -
Note| -
```

readchr(1)

Get a char from keyb and save to memory

```
RAX | The address to save the char to
Ret | The char in the sent address
Note| Var should be of type DB/rb or resb
```

readch/1

Get a char from keyb into AL, with echo

```
Arg | -
Ret | The key in AL
Note| Internal use
    | Use halt if needed
```

readstr(1)/1

Get string from keyboard (echoed)

```
RAX | Buffer's address to save string
Ret | 0-ended string in sent buffer
    | RAX= # of actual bytes in
Note| Press ENTER to finish
    | Buffer must be large enough
    | Ret -1 signals error
    | String will be 0-ended
```

mem_alloc(1)/1

Request memory of n bytes

```
RAX | Bytes requested
Ret | Pointer to the mem block. -1 = none
Note| -
```

mem_free(1)/1

Free memory allocated by mem_alloc

```
RAX | Pointer from mem_alloc
Ret | 1 = success. -1 = error
Note| -
```

mem_alloc2(1)/1

Request memory of n bytes

```
RAX | Bytes requested
Ret | Pointer to the mem block. -1 = none
Note| Uses HeapAlloc
```

mem_free2(1)/1

Free memory allocated by mem_alloc2

```
RAX | Pointer from mem_alloc
Ret | 1 = success. -1 = error
Note| Uses HeapFree
```

mem_load(1)/2

Load a file to memory

```
RAX | Filename path string
Ret | RAX - block pointer
    | RBX - Size
Note| Should re-claim memory at RAX
    | File name must be 0-ended
```

file_new(1)/1

Create a new file

```
RAX | File name path string
Ret | -1 = Error. 0 = success
Note| Filename string must be 0-ended
```

file_open(2)/1

Open an existing file for reading/writing

```
RBX | Operation:
       0-Read
       1-Write
RAX | File name path string
Ret | File descriptor. -1 if error
Note| Filename string must be 0-ended
```

file_read(3)/1

Read from an existing opened file

```
RCX | Number of bytes to read
RBX | Input buffer
RAX | File handle from file_open
Ret | Number of actual read
Note| Filename string must be 0-ended
    | Input buffer must be >= RAX
```

file_write(3)/1

Write to an existing opened file

```
RCX | Number of bytes to write
RBX | Source buffer's address
RAX | File handle from file_open
Ret | Number of actual writes
Note| Filename string must be 0-ended
    | Handle must be opened-write
```

file_close(1)

Close handle issued by file_open

```
RAX | File handle
Ret | -
Note| RAX will be zeroed
```

file_size(1)/1

Get filesize of an opened file

```
RAX | File handle from file_open
Ret | Size in bytes. -1 says error
Note| -
```

file_copy(2)

Copy a file to a new file

```
RBX | Address of new file name
RAX | Address of source file name
Ret | -
Note| Both must be 0-ended string
```

file_delete(1)

Delete a file

```
RAX | Address of Filename string
Ret | 0-Fail (Win64), -ve (Linux64)
Note| Handles must be closed
```

timer_start

Start timer

```
Arg | -
Ret | R14-freq,R15-timer (silent ret)
Note| Should be followed by timer_stop
    | Destroys R14 and R15
```

timer_stop

Stop current timer

```
R14 | Freq (ignore if not used)
R15 | Timer (ignore if not used)
Ret | Time difference in seconds. RAX
Note| Ignore arguments if you are not
      using them in-between
    | Should come after timer_start
    | Destroys R14 and R15
```

delay(1)

Put execution at delay

```
RAX | Milliseconds
Ret | -
Note| 1000ms = 1s
```

get_ticks/1

Get tick count

```
-   | -
Ret | Ticks (ms)
Note| 1000ms = 1s
```

halt

Pause screen

```
Arg | -
Ret | -
Note| Hit Enter to continue
```

exitp

Pause then exit to system

```
Arg | -
Ret | -
Note| Hit Enter to exit
```

exitx

Exit to system

```
Arg | -
Ret | -
Note| -
```

BASE2.0 (base86)

32-bit Application Programming Interfaces (API) for base86

This API applies to both Win32 and Linux32 versions of base86. base86 uses STDCALL calling convention*:

- Parameters use stack memory
- Parameter size defaults to four bytes (dword)
- Arguments are passed / pushed on the stack; in correct sequence, types and numbers
- Double-precision and quadwords arguments are passed by values as two consecutive dwords. Larger arguments are passed by reference.
- Except for float-specific (real4) routines, all real arguments are passed as doubles on the stack
- Integer returns in EAX. Real returns in ST0 (FPU registers are clobbered)
- Quadword values are returned in EAX(low) and EDX(high).
- Callee unwind / clean-up stack

LISTING*

prnreg	prnregdu	prnregd
prnregou	prnrego	dumpreg
dumpregdu	dumpregd	dumprego
dumpregou	dumpregb	dumpseg
flags	opsize	opcode
stackview	memview	memviewn
memviewc	memviewb	mem_reset
mem_set	mem_swap	mem_insertr
mem_insert	mem_copy	mem_lcopy
mem_tofile	prnintu	prnint
prnhexu	prnhex	prnoct
prnoctu	prnbinu	prnbin
prnbinb	fpbin	fpbind
fpbinx		
prnintq	prnintqu	prnhexq
prnhexqu	prnoctq	prnoctqu
prndbl	prndble	prnflt
prnflte	prndblx	fltsplit
decodeBASE4	encodeBASE64	int2str
dec2str	dec2stru	oct2str
oct2stru	hex2str	hex2stru
bin2str	bin2stru	dbl2str
dble2str	flt2str	flte2str
dblx2str	dumpmmxrf	dumpmmxf
dumpmmxr	dumpmmx	dumpxmmrf
dumpxmmf	dumpxmmr	dumpxmm
dumpymmrf	dumpymmf	dumpymmr
dumpymm	prndblf	prnfltf
prnhexbf	prnintuf	prnintf
prnintqf	prnintquf	prnintbf
prnintb	prnintwf	prnintw
clearxmm	clearymm	sse_flags
sse_round	rstmxcsr	fpdinfo
fpfinfo	fpxinfo	fpu_stack
fpu_sflag	fpu_cflag	fpu_tag
fpu_reg	fpu_precision	fpu_round
rndint	randm	randq
factorial	powint	addf
subf	mulf	divf
sine	tangent	cosine
sincos	atangent	secant
cosecant	cotangent	rad2deg
deg2rad	fpow	g_ratio
lg10	ln10	isintREAL4
isintREAL8	iseven	isodd
bconv	bitfield	str2int
str2dbl	str2flt	dbl2int
dblt2int	flt2int	fltt2int
int2flt	readint	readdbl
readflt	chr_isdigit	chr_isalpha
chr_islower	chr_isupper	chr_change
chr_chcase	chr_tolower	chr_toupper
chr_find	chr_count	chr_shuffle
byte_count	ascii	str_trim
str_wordcnt	str_token	str_copy
str_cmpz	str_cmp	str_find
str_findz	str_append	str_appendz
str_reverse	str_reverse2	str_tolower
str_toupper	str_length	str_lengthd
str_length2	digitprob	digitprobu
digithprob	digithprobu	digitscan
digitscanu	digithscan	digithscanu
digitcount	digitcountu	digithcount
digithcountu	scan_dword	scan_word
scan_byte	sort_byte	sort_int
sort_intu	sort_dbl	sort_dblx
sort_flt	aprnhbyte	aprnhbyter
aprnbyte	aprnbyteu	aprnbyter
aprnbyteur	aprnword	aprnwordu
aprnwordr	aprnwordur	aprnintd
aprnintdu	aprnintdr	aprnintdur
aprnint	aprnintu	aprnintr
aprnintur	aprnflt	aprnfltr
aprndbl	aprndblr	aprndblx
aprndblxr	aprnhbytef	aprnhbyterf
aprnbytef	aprnbyteuf	aprnbyterf
aprnbyteurf	aprnwordf	aprnworduf
aprnwordrf	aprnwordurf	aprnintdf
aprnintduf	aprnintdrf	aprnintdurf
aprnintf	aprnintuf	aprnintrf
aprninturf	aprnfltf	aprnfltrf
aprndblf	aprndblrf	

Kernel Specifics

prnchrp	prnspaces	prnstreg**
prnlines	prnline	prnspace
prnchar	prnchr	prnstrd
prnstrz	prnstr	readchr
readch	readstr	mem_alloc
mem_free	mem_load	file_new
file_open	file_read	file_write
file_close	file_size	file_copy
file_delete	timer_start	timer_stop
delay	get_ticks	halt
exitp	exitx	

* A number of routines are not publicly visible due to their supportive nature.
** Argument in EAX

prnreg(1)

Display 32-bit register in unsigned hex

```
push1| Register to display
Ret  | -
Note | -
```

prnregdu(1)

Display 32-bit register in unsigned int

```
push1| Register to display
Ret  | -
Note | -
```

prnregd(1)

Display 32-bit register in signed int

```
push1| Register to display
Ret  | -
Note | -
```

prnregou(1)

Display 32-bit register in unsigned octal

```
push1| Register to display
Ret  | -
Note | -
```

prnrego(1)

Display 32-bit register in signed octal

```
push1| Register to display
Ret  | -
Note | -
```

dumpreg

Display registers dump (unsigned hex)

```
Arg  |
Ret  | -
Note | -
```

dumpregdu

Display registers dump (decimal)

```
Arg  |
Ret  | -
Note | -
```

dumpregd

Display registers dump (signed int)

```
Arg  |
Ret  | -
Note | -
```

dumpregou

Display registers dump (unsigned oct)

```
Arg  |
Ret  | -
Note | -
```

dumprego

Display registers dump (signed oct)

```
Arg  |
Ret  | -
Note | -
```

dumpregb

Display registers dump (unsigned binary)

```
Arg  |
Ret  | -
Note | -
```

dumpseg

Display segment registers

```
Arg  | -
Ret  | -
Note | -
```

flags(1)

Display EFLAG register

```
push1| pushfd
Ret  | -
Note | -
```

opsize(2)/1

Get size between 2 labels

```
push2| Label
push1| Label
Ret  | Size
Note | Order of label is irrelevant
```

opcode(2)

Encode instrns between 2 labels

```
push2| Label 2
push1| Label 1 (first/lead label)
Ret  | -
Note | Label 1 must be the first
```

stackview(1)

Display stack

```
push1| Number of items to view (+up,-down)
Ret  | -
Note | content |hex_address
     | 1 item = 4 bytes implied
     | If +ve, last line is TOS
     | If -ve, First line is TOS
```

memview(2)

View memory dump + offsets

```
push2| label, address
push1| size (+up,-down)
Ret  | -
Note | Leftmost is MSB
     | Content |Addr| HexOffs|DecOffs
```

memviewn(2)

View memory dump in Little-Endian layout

```
push2| label, address
push1| size (+up,-down)
Ret  | -
Note | -
```

memviewc(2)

View memory dump (strings) + offsets

```
push2| label, address
push1| size (+up,-down)
Ret  | -
Note | Leftmost is MSB
     | Content |Addr| HexOffs|DecOffs
```

memviewb(3)

View memory dump in flat layout

```
push3| Type: 0-Hex, 1=Decimal
push2| +size
push1| label, address
Ret  | -
Note | -
```

mem_reset(2)

Clear memory by arg2 bytes

```
push2| number of bytes to clear
push1| offset / location
Ret  | -
Note | Clear by bytes
```

mem_set(3)

Set memory with arg3 byte

```
push3| Set byte to use
push2| size of bytes to set
push1| Address to set
Ret  | -
Note | Set by bytes
```

mem_swap(2)

Swap two dword memory content
Swap two dword array elements

```
push2| Address of item
push1| Address of item
Ret  | -
Note | Could use LEA / OFFSET
```

mem_insertr(2)

Insert register content into memory

```
push2| Value to insert
push1| Target location
Ret  | -
Note | Put value in register first
```

mem_insert(4)

Insert bytes into memory

```
push4| Num of source bytes to write
push3| Source Address
push2| Destn offsets to write.0=from start
push1| Destn address
Ret  | -
Note | Overwrites
```

mem_copy(3)

Copy memory content of R8 bytes

```
push3| Number of bytes to copy
push2| Source pointer
push1| Destination pointer
Ret  | -
Note | -
```

mem_lcopy(3)

Copy memory content of arg3 bytes

```
push3| Number of bytes to copy (Large n)
push2| Source pointer (from)
push1| Destination pointer (to)
Ret  | -
Note | -
```

mem_tofile(3)

Copy memory content to a new file

```
push3| Address of new filename
push2| Size to copy
push1| Address of memory source
Ret  | A new filled file
Note | Must be a new file or else...
```

prnintu(1)

Display unsigned Decimal

```
push1| Value to display
Ret  | -
Note | -
```

prnint(1)

Display Signed Decimal Integer

```
push1| Value to display
Ret  | -
Note | -
```

prnhexu(1)

Display Unsigned Hexadecimal

```
push1| Value to display
Ret  | -
Note | Imm. must be in valid format
```

prnhex(1)

Display Signed Hexadecimal

```
push1| Value to display
Ret  | -
Note | Imm must be in valid format
```

prnoct(1)

Display Octal (signed)

```
push1| Value to display
Ret  | -
Note | Imm. must be in valid format
```

prnoctu(1)

Display Octal (unsigned)

```
push1| Value to display.
Ret  | -
Note | Imm. must be in valid format
```

prnbinu(1)

Display unsigned Binary

```
push1| Value to display
Ret  | -
Note | Imm. must be in valid format
```

prnbin(1)

Display Signed Binary

```
push1| Value to display.
Ret  | -
Note | Imm. must be in valid format
```

prnbinb(1)

Display Unsigned Binary with separators

```
push1| Value to display
Ret  | -
Note | Imm must be in valid format
     | Full display
```

fpbin(2)

Display REAL8 Floating Point Binary

```
push2| A DQ/double/REAL8, upper DWORD
push1| A DQ/double/REAL8, lower DWORD
Ret  | -
Note | push1 and push2 are the same double
```

fpbind(1)

Display REAL4 Floating Point Binary

```
push1| REAL4 value to display
Ret  | -
Note | -
```

fpbinx(1)

Display REAL10 Floating Point Binary

```
push1| Address of the REAL10
Ret  | -
Note | DT type.
```

prnintq(2)

Display Signed 64-bit integer (qword)

```
push2| The DQ's upper dword
push1| The DQ's lower dword
Ret  | -
Note | -
```

prnintqu(2)

Display Unsigned 64-bit integer (qword)

```
push2| The DQ's upper dword
push1| The DQ's lower dword
Ret  | -
Note | -
```

prnhexq(2)

Display Signed 64-bit hexadecimal (qword)

```
push2| The DQ's upper dword
push1| The DQ's lower dword
Ret  | -
Note | -
```

prnhexqu(2)

Display Unsigned 64-bit hex (qword)

```
push2| The DQ's upper dword
push1| The DQ's lower dword
Ret  | -
Note | -
```

prnoctq(2)

Display Signed 64-bit octal (qword)

```
push2| The DQ's upper dword
push1| The DQ's lower dword
Ret  | -
Note | -
```

prnoctqu(2)

Display Unsigned 64-bit octal (qword)

```
push2| The DQ's upper dword
push1| The DQ's lower dword
Ret  | -
Note | -
```

prndbl(2)

Display double precision (REAL8)

```
push2| REAL8 upper dword
push1| REAL8 lower dword
Ret  | -
Note | Var should be init as FP (0.0)
```

prndble(3)

Display REAL8 with decimal places

```
push3| Decimal places (1-15)
push2| REAL8 upper dword
push1| REAL8 lower dword
Ret  | -
Note | Var should be init as FP (0.0)
```

prnflt(1)

Display single precision (REAL4)

```
push1| REAL4 value to display
Ret  | -
Note | Var should be initd to FP (0.0)
```

prnflte(2)

Display single-precision (REAL4) with decimal places

```
push2| Decimal places (1-7)
push1| REAL4 value to display
Ret  | -
Note | Var should be initd to FP (0.0)
```

prndblx(1)

Display unrounded extended-precision (REAL10)

```
push1| The address of a DT in memory
Ret  | -
Note | Var should be initd to FP (0.0)
     | Variable must be of type DT.
```

fltsplit(1)/2

Split a REAL4 into parts

```
push1| REAL4 value
Ret  | Integer in EAX. Fraction in ST0.
Note | Takes normal FP value only
     | Destroys FPU
```

decodeBASE64(2)

Decode 0-ended BASE64 to ASCII (RFC4648)

```
push2| Address of destination buffer
push1| Address of string to decode
Ret  | -
Note | Arg2 at least 2/3 the size of Arg1
     | Returned string are null-ended
```

encodeBASE64(2)

Encode 0-ended ASCII to BASE64 (RFC 4648)

```
push2| Address of destination buffer
push1| Address of string to encode
Ret  | -
Note | Arg2 twice the size of Arg1
     | Both strings are null-ended
```

int2str(4)

Convert integer to 0-ended string

```
push4| Signness. 0=unsigned. 1=signed
push3| Target base (2 to 36 only)
push2| Buffer's address
push1| Value
Ret  | String in the sent buffer
Note | Arg2 size must reflect # of digits
```

dec2str(2)

Convert signed int to 0-ended decimal string

```
push2| Buffer's address
push1| Integer to convert
Ret  | String in buffer
Note | Buffer must be large enough
```

dec2stru(2)

Convert unsigned int to 0-ended dec string

```
push2| Buffer's address
push1| Integer to convert
Ret  | String in buffer
Note | Buffer must be large enough
```

oct2str(2)

Convert signed octal to 0-ended string

```
push2| Buffer's address
push1| Integer to convert
Ret  | String in buffer
Note | Buffer must be large enough
```

oct2stru(2)

Convert unsigned octal to 0-ended octal string

```
push2| Buffer's address
push1| Integer to convert
Ret  | String in buffer
Note | Buffer must be large enough
```

hex2str(2)

Convert signed int to 0-ended hex string

```
push2| Buffer's address
push1| Integer to convert
Ret  | String in buffer
Note | Buffer must be large enough
```

hex2stru(2)

Convert unsigned int to 0-ended hex string

```
push2| Buffer's address
push1| Integer to convert
Ret  | String in buffer
Note | Buffer must be large enough
```

bin2str(2)

Convert signed bin to 0-ended bin string

```
push2| Buffer's address
push1| Integer to convert
Ret  | String in buffer
Note | Buffer must be large enough (>34)
```

bin2stru(2)

Convert unsigned bin to 0-ended bin string

```
push2| Buffer's address
push1| Integer to convert
Ret  | String in buffer
Note | Buffer must be large enough (>32)
```

dbl2str(3)

Convert REAL8 to 0-ended string

```
push3| Address of bufffer
push2| Double's upper DWORD
push1| Double's lower DWORD
Ret  | 0-ended string in sent buffer.
Note | Buffer > 24
```

dble2str(4)

Convert REAL8 with decimal places to 0-ended string

```
push4| Buffer's address
push3| Decimal places
push2| Double's upper DWORD
push1| Double's lower DWORD
Ret  | 0-ended string in sent buffer.
Note | Buffer > 24
```

flt2str(2)

Convert REAL4 to 0-ended string

```
push2| Buffer's address to save to
push1| REAL4 value
Ret  | 0-ended string in sent buffer.
Note | Buffer > 15
```

flte2str(3)

Convert REAL4 with decimal places to 0-ended string

```
push3| Buffer's address to save to
push2| Decimal places. (1-7)
push1| REAL4 value
Ret  | 0-ended string in sent buffer.
Note | Buffer > 15
```

dblx2str(2)

Convert REAL10 to 0-ended string

```
push2| Buffer address to save to
push1| Address of a DT to convert
Ret  | 0-ended string in sent buffer.
Note | push1 must point to type DT.
     | Buffer >= 32
     | Unrounded
```

dumpmmxrf(1)

Dump MMX registers,reversed view. Formatted.

```
push1| Type of display:
       0 - Unsigned integer BYTE
       1 - Signed integer BYTE
       2 - Unsigned integer WORD
       3 - Signed integer WORD
       4 - Unsigned integer DWORD
       5 - Signed integer DWORD
       6 - Unsigned integer QWORD
       7 - Signed integer QWORD
       8 - Unsigned hex BYTES
Note | -
```

dumpmmxf(1)

Dump MMX registers. Formatted.

```
push1| Type of display:
       0 - Unsigned integer BYTE
       1 - Signed integer BYTE
       2 - Unsigned integer WORD
       3 - Signed integer WORD
       4 - Unsigned integer DWORD
       5 - Signed integer DWORD
       6 - Unsigned integer QWORD
       7 - Signed integer QWORD
       8 - Unsigned hex BYTES
Note | -
```

dumpmmxr(1)

Dump MMX registers,reversed view

```
push1| Type of display:
       0 - Unsigned integer BYTE
       1 - Signed integer BYTE
       2 - Unsigned integer WORD
       3 - Signed integer WORD
       4 - Unsigned integer DWORD
       5 - Signed integer DWORD
       6 - Unsigned integer QWORD
       7 - Signed integer QWORD
       8 - Unsigned hex BYTES
Note | -
```

dumpmmx(1)

Dump MMX registers

```
push1| Type of display:
       0 - Unsigned integer BYTE
       1 - Signed integer BYTE
       2 - Unsigned integer WORD
       3 - Signed integer WORD
       4 - Unsigned integer DWORD
       5 - Signed integer DWORD
       6 - Unsigned integer QWORD
       7 - Signed integer QWORD
       8 - Unsigned hex BYTES
Note | -
```

dumpxmmrf(1)

Dump XMM registers,reversed view. Formatted.

```
push1| Type of display:
       0 - Unsigned integer BYTE
       1 - Signed integer BYTE
       2 - Unsigned integer WORD
       3 - Signed integer WORD
       4 - Unsigned integer DWORD
       5 - Signed integer DWORD
       6 - Unsigned integer QWORD
       7 - Signed integer QWORD
       8 - Singles
       9 - Doubles
       10- Unsigned hex BYTES
Note | -
```

dumpxmmf(1)

Dump XMM registers. Formatted.

```
push1| Type of display:
       0 - Unsigned integer BYTE
       1 - Signed integer BYTE
       2 - Unsigned integer WORD
       3 - Signed integer WORD
       4 - Unsigned integer DWORD
       5 - Signed integer DWORD
       6 - Unsigned integer QWORD
       7 - Signed integer QWORD
       8 - Singles
       9 - Doubles
       10- Unsigned hex BYTES
Note | -
```

dumpxmmr(1)

Dump XMM registers,reversed view

```
push1| Type of display:
       0 - Unsigned integer BYTE
       1 - Signed integer BYTE
       2 - Unsigned integer WORD
       3 - Signed integer WORD
       4 - Unsigned integer DWORD
       5 - Signed integer DWORD
       6 - Unsigned integer QWORD
       7 - Signed integer QWORD
       8 - Singles
       9 - Doubles
       10- Unsigned hex BYTES
Note | -
```

dumpxmm(1)

Dump XMM registers

```
push1| Type of display:
       0 - Unsigned integer BYTE
       1 - Signed integer BYTE
       2 - Unsigned integer WORD
       3 - Signed integer WORD
       4 - Unsigned integer DWORD
       5 - Signed integer DWORD
       6 - Unsigned integer QWORD
       7 - Signed integer QWORD
       8 - Singles
       9 - Doubles
       10- Unsigned hex BYTES
Note | -
```

dumpymmrf(1)

Dump YMM registers. Reversed. Formatted.

```
push1| Type of display:
       0 - Unsigned integer BYTE
       1 - Signed integer BYTE
       2 - Unsigned integer WORD
       3 - Signed integer WORD
       4 - Unsigned integer DWORD
       5 - Signed integer DWORD
       6 - Unsigned integer QWORD
       7 - Signed integer QWORD
       8 - Singles
       9 - Doubles
       10- Unsigned hex BYTES
Note | Expects AVX CPU.
Note | Byte options need wide display
```

dumpymmf(1)

Dump YMM registers. Formatted.

```
push1| Type of display:
       0 - Unsigned integer BYTE
       1 - Signed integer BYTE
       2 - Unsigned integer WORD
       3 - Signed integer WORD
       4 - Unsigned integer DWORD
       5 - Signed integer DWORD
       6 - Unsigned integer QWORD
       7 - Signed integer QWORD
       8 - Singles
       9 - Doubles
       10- Unsigned hex BYTES
Note | Expects AVX CPU
     | Byte options need wide display
```

dumpymmr(1)

Dump YMM registers. Reversed

```
push1| Type of display:
       0 - Unsigned integer BYTE
       1 - Signed integer BYTE
       2 - Unsigned integer WORD
       3 - Signed integer WORD
       4 - Unsigned integer DWORD
       5 - Signed integer DWORD
       6 - Unsigned integer QWORD
       7 - Signed integer QWORD
       8 - Singles
       9 - Doubles
       10- Unsigned hex BYTES
Note | Expects AVX CPU
     | Byte options need wide display
```

dumpymm(1)

Dump YMM registers

```
push1| Type of display:
       0 - Unsigned integer BYTE
       1 - Signed integer BYTE
       2 - Unsigned integer WORD
       3 - Signed integer WORD
       4 - Unsigned integer DWORD
       5 - Signed integer DWORD
       6 - Unsigned integer QWORD
       7 - Signed integer QWORD
       8 - Singles
       9 - Doubles
       10- Unsigned hex BYTES
Note | Expects AVX CPU
     | Byte options need wide display
```

prndblf(2)

Display Double-precision. Formatted

```
push1| Double's upper dword.
push1| Double's lower dword.
Ret  | -
Note | -
```

prnfltf(1)

Display Single-precision. Formatted

```
push1| Single value to display.
Ret  | -
Note | -
```

prnhexbf(1)

Display Unsigned Hex Byte (formatted)

```
push1| Byte value to display.
Ret  | -
Note | Internal use
```

prnintuf(1)

Display 32-bit unsigned int (formatted)

```
push1| Dword Value to display
Ret  | -
Note | Internal use
```

prnintf(1)

Display 32-bit signed integer (formatted)

```
push1| Dword Value to display
Ret  | -
Note | Internal use
```

prnintqf(2)

Display 64-bit signed int (formatted)

```
push1| The quad's upper dword
push1| The quad's lower dword
Ret  | -
Note | Internal use
```

prnintquf(2)

Display 64-bit unsigned int (formatted)

```
push1| The quad's upper dword
push1| The quad's lower dword
Ret  | -
Note | Internal use
```

prnintbf(2)

Display Byte Decimal (formatted)

```
push2| 0-unsigned. 1-signed
push1| Byte value to display.
Ret  | -
Note | Use "dword" override for push1
```

prnintb(2)

Display Byte Decimal

```
push2| 0-unsigned. 1-signed
push1| Byte value to display.
Ret  | -
Note | Use "dword" override for push1
```

prnintwf(2)

Display 16-bit Decimal (formatted)

```
push2| 0-unsigned. 1-signed
push1| Word value to display
Ret  | -
Note | Use "dword" override for push1
```

prnintw(2)

Display 16-bit Decimal

```
push2| 0-unsigned. 1-signed
push1| Word value to display.
Ret  | -
Note | Use "dword" override for push1
```

clearxmm

Clear all XMM registers

```
Arg  | -
Ret  | -
Note | -
```

clearymm

Clear all YMM registers

```
Arg  | -
Ret  | -
Note | Expects AVX CPU
```

sse_flags

Display MXCSR register

```
Arg  | -
Ret  | -
Note | -
```

sse_round(1)

Set SSE rounding control

```
push1| Rounding mode
       0 = near
       1 = down
       2 = up
       3 = zero
Ret  | -
Note | -
```

rstmxcsr

Reset SIMD Flags (mxcsr) to CPU default

```
Arg  | -
Ret  | -
Note | -
```

fpdinfo(2)

Display Double FP information

```
push2| REAL8's upper dword
push1| REAL8's lower dword
Note | -
Ret  | -
```

fpfinfo(1)

Display Float FP information

```
push1| REAL4 value
Ret  | -
Note | -
```

fpxinfo(1)

Display REAL10 FP information

```
push1| Address of a REAL10
Ret  | -
Note | -
```

fpu_stack

Display FPU stack

```
Arg  | -
Ret  | -
Note | Display is real10
```

fpu_sflag

Display FPU Status Flag

```
Arg  | -
Ret  | -
Note | -
```

fpu_cflag

Display FPU Control Flag

```
Arg  | -
Ret  | -
```

fpu_tag

Display FPU Tag Register

```
Arg  | -
Ret  | -
Note | 00 = valid
     | 01 = zero
     | 10 = invalid
     | 11 = empty
```

fpu_reg(1)

Display one FPU stack register

```
push1| Stack # to display (0-7)
Ret  | -
Note | Special values show nothing
```

fpu_precision(1)

Set FPU precision control

```
push1| Precision mode:
       0 = real8
       1 = real10
       2 = real4
Ret  | -
Note | FSAVE/FINIT will reset it
```

fpu_round(1)

Set FPU rounding control

```
push1| Rounding mode:
       0 = near
       1 = down
       2 = up
       3 = zero
Ret  | -
Note | FSAVE/FINIT will reset it
```

rndint/1

Generate 1 pseudorandom integer

```
Arg  | -
Ret  | Random integer in EAX
Note | -
```

randm(1)/1

Get 1 unsigned random integer

```
push1| MAX value (0-MAX)
Ret  | Random integer in EAX
Note | 0, MAX inclusive
```

randq(2)

Generate unique random integers from 0 to MAX-1 and save and array

```
push2| Addr of array initialized to -1
push1| MAX value
Ret  | Random elements in array
Note | Example: MAX = 50
       elems dd 50 dup(-1)
       elems: TIMES 50 dd -1
```

factorial(1)/1

Get factorial

```
push1| +Value
Ret  | Factorial in EAX
Note | Limit to !12 Only
```

powint(2)/1

Calculate 32-bit integral power

```
push2| +Exponent/power (int)
push1| Base (int)
Ret  | Result in EAX
     | # if overflow
Note | Var must be of type DD
     | Takes +exponent only
```

addf(4)/1

Add two floating points

```
push4| Value2 upper dword
push3| Value2 lower dword
push2| Value1 upper dword
push1| Value1 lower dword
Ret  | Value in ST0
Note | ST0 = Value1+Value2
     | Value1 and Value2 are doubles
     | Destroys FPU
```

subf(4)/1

Sub two floating points

```
push4| Value2 upper dword
push3| Value2 lower dword
push2| Value1 upper dword
push1| Value1 lower dword
Ret  | Value in ST0
Note | ST0 = Value1 - Value2
     | Value1 and Value2 are doubles
     | Destroys FPU
```

mulf(4)/1

Multiply two floating points

```
push4| Value2 upper dword
push3| Value2 lower dword
push2| Value1 upper dword
push1| Value1 lower dword
Ret  | Value in ST0
Note | ST0 = Value1 * Value2
     | Value1 and Value2 are doubles
     | Destroys FPU
```

divf(4)/1

Divide two floating points

```
push4| Value2 upper dword (divisor)
push3| Value2 lower dword (divisor)
push2| Value1 upper dword (dividend)
push1| Value1 lower dword (dividend)
Ret  | Value in ST0
Note | ST0 = Value1 / Value2
     | Value1 and Value2 are doubles
     | Destroys FPU
```

sine(2)/1

Get sine radian

```
push2| Radian's upper dword
push1| Radian's lower dword
Ret  | Sine in ST0
Note | Arguments are a double
     | Destroys FPU
```

tangent(2)/1

Get tangent radian

```
push2| Radian's upper dword
push1| Radian's lower dword
Ret  | Tangent in ST0
Note | Arguments are a double
     | Destroys FPU
```

cosine(2)/1

Get cosine radian

```
push2| Radian's upper dword
push1| Radian's lower dword
Ret  | Cosine in ST0
Note | Arguments are a double
     | Destroys FPU
```

sincos(2)/2

Get sine and cosine radian

```
push2| Radian's upper dword
push1| Radian's lower dword
Ret  | Sine in ST0
     | Cosine in ST1
Note | Arguments are a double
     | Destroys FPU
```

atangent(2)/1

Get arc-tangent radian

```
push2| Radian's upper dword
push1| Radian's lower dword
Ret  | Arctangent in ST0
Note | Arguments are a double
     | Destroys FPU
```

secant(2)/1

Get secant radian

```
push2| Radian's upper dword
push1| Radian's lower dword
Ret  | Secant in ST0
Note | Arguments are a double
     | Destroys FPU
```

cosecant(2)/1

Get cosecant radian

```
push2| Radian's upper dword
push1| Radian's lower dword
Ret  | Cosecant in ST0
Note | Arguments are a double
     | Destroys FPU
```

secant(2)/1

Get cotangent radian

```
push2| Radian's upper dword
push1| Radian's lower dword
Ret  | Cotangent in ST0
Note | Arguments are a double
     | Destroys FPU
```

rad2deg(2)/1

Convert Radian to Degree

```
push2| REAL8 value upper dword, in RADIAN
push1| REAL8 value lower dword, in RADIAN
Ret  | DEGREE value in ST0
Note | Var should be init as FP (0.0)
     | Destroys FPU
     | push1 & push2 is same double arg.
```

deg2rad(2)/1

Convert Degree to Radian

```
push2| REAL8 value upper dword, in DEGREE
push1| REAL8 value lower dword, in DEGREE
Ret  | RADIAN value in ST0
Note | Var should be init as FP (0.0)
     | push1 & push2 is same double arg.
     | Destroys FPU
```

fpow(4)/1

Calculate x^n

```
push4| pow upper DWORD (+4)
push3| pow lower DWORD
push2| +base upper DWORD (+4)
push1| +base lower DWORD
Ret  | Raised value in ST0 (Double)
Note | base must be positive
     | push1 and push2 is one double
     | push3 and push4 is one double
     | For cubic root, use
       pow=0.3333333333333333
     | Destroys FPU
```

g_ratio(3)/1

Find golden ratio for multi ops

```
push3| 0 - mul
       1 - div
       2 - pow
push2| REAL8 value, upper DWORD
push1| REAL8 value, lower DWORD
Ret  | golden ratio in ST0 (double)
Note | 1.618033988749894848
     | ST0 = arg <op> g.ratio
     | Destroys FPU
```

lg10(2)/1

Find Common Log of Base 10

```
push2| REAL8 value, upper DWORD
push1| REAL8 value, lower DWORD
Ret  | Value in ST0
Note | Destroys FPU
```

ln10(2)/1

Find Natural Log

```
push2| REAL8 value, upper DWORD
push1| REAL8 value, lower DWORD
Ret  | Value in ST0
Note | Destroys FPU
```

isintREAL4(1)/1

Check if a REAL4 is a qualified integer

```
push1| REAL4 value
Ret  | EAX: 0-no,1-yes
Note | Test normal FP values only
```

isintREAL8(2)/1

Check if a REAL8 is a qualified integer

```
push2| REAL8 value, upper DWORD
push1| REAL8 value, lower DWORD
Ret  | EAX: 0-no,1-yes
Note | Test normal FP values only
```

iseven(1)/1

Check if integer is even number

```
push1| Integer to check
Ret  | 0 if no. 1 if yes
Note | -
```

isodd(1)/1

Check if integer is odd number

```
push1| Integer to check
Ret  | 0 if no. 1 if yes
Note | -
```

bconv(2)

Convert and display integer

```
push2| Display base to use (2 to 36)
push1| Integer to convert
Ret  | -
Note | -
```

bitfield(3)

Display a bitfield from a 64-bit data

```
push3| lower bit
push2| higher bit
push1| Value to be extracted from
Ret  | -
Note | Display bit range
```

str2int(1)/1

Convert 0-ended string to integer

```
push1| Address of the 0-ended string
Ret  | Integer in EAX
Note | Expected Suffix- b(bin), d(dec)
     | h(hex), o,q(octal). 'd' is optional
     | String must be of type DB
     | Digits string must be valid
```

str2dbl(1)/1

Convert string to double

```
push1| Address of a 0-ended string
Ret  | Double-precision ST0
Note | String must be of type DB
     | Destroys FPU
```

str2flt(1)/1

Convert string to float

```
push1| Address of a 0-ended string
Ret  | Single-precision in ST0
Note | Limit digits 7
     | String must be of type DB
     | Destroys FPU
```

dbl2int(3)/2

Convert REAL8 to integer

```
push3| Rounding mode
       0 = near
       1 = down
       2 = up
       3 = zero
push2| REAL8 value upper DWORD
push1| REAL8 value lower DWORD
Ret  | 64-bit integer in;
       EAX(low)
       EDX(High)
Note | Precision will be truncated
       Input must be in FP format
```

dblt2int(2)/2

Convert REAL8 to integer, truncate to 0

```
push2| REAL8 value upper DWORD
push1| REAL8 value lower DWORD
Ret  | 64-bit integer in;
       EAX(low)
       EDX(High)
Note | Precision will be truncated
     | Input must be in FP format
```

flt2int(2)/1

Convert REAL4 to integer

```
push2| Rounding mode:
       0-Near
       1-Down
       2-Up
       3-Zero
push1| REAL4 value
Ret  | Integer in EAX
Note | Precision will be truncated
     | Input must be in FP format
```

fltt2int(1)/1

Convert REAL4 to integer, truncate to 0

```
push1| REAL4 value
Ret  | Integer in EAX
Note | Input must be in FP format
```

int2flt(1)/1

Convert integer to real4

```
push1| Integer value to convert
Ret  | Real4 in ST0
Note | Destroys FPU
```

readint/1

Get multi suffix integer from keyboard

```
Arg  | -
Ret  | Integer in EAX
Note | Expected Suffix- b(binary), d(dec)
     | h(hex), o,q(octal). d is optional
     | Digits must be in valid format
```

readdbl/1

Get double-precision from stdin

```
Arg  | -
Ret  | Double precision in ST0
Note | Doesn't take E format
     | Must use FP format for input
     | Destroys FPU
```

readflt/1

Get single-precision from stdin

```
Arg  | -
Ret  | Single precision in ST0
Note | Doesn't take E format
     | Must use FP format for input
     | Destroys FPU
```

chr_isdigit(1)/1

Check if alphanumeric digit

```
push1| Byte char to check
Ret  | EAX - 0 if no. 1 if yes
Note | Digit '0' to '9' only.
```

chr_isalpha(1)/1

Check if alphabet

```
push1| Byte char to check
Ret  | EAX - 0 if no. 1 if yes
Note | 'A'-'Z' and 'a'-'z'
```

chr_islower(1)/1

Check if lowercase

```
push1| Byte char to check
Ret  | EAX - 0 if no. 1 if yes
Note | -
```

chr_isupper(1)/1

Check if uppercase

```
push1| Byte char to check
Ret  | EAX - 0 if no. 1 if yes
Note | -
```

chr_change(3)

Change char occurences from a 0-ended string to char in arg3

```
push3| Char byte to use
push2| Char byte to change
push1| Address of string
Ret  | -
Note | -
```

chr_chcase(1)/1

Alternate character case

```
push1| Byte char to change
Ret  | Modified char
Note | -
```

chr_tolower(1)/1

Change to lowercase

```
push1| Byte to change
Ret  | Modified char in AL
Note | -
```

chr_toupper(1)/1

Change to uppercase

```
push1| Character to change
Ret  | Modified char in AL
Note | -
```

chr_find(2)/1

Find char byte from a 0-ended string

```
push2| String address to look from
push1| Character byte to find
Ret  | Index in EAX. -1 if not found
Note | Will stop at first one found
```

chr_count(2)/1

Count a char from a 0-ended string

```
push2| Character to count
push1| String's address
Ret  | Count in EAX. 0 if none
Note | -
```

chr_shuffle(1)

Shuffle a 0-ended string

```
push1| String's address
Ret  | Same string shuffled
Note | -
```

byte_count(3)/1

Count byte recurrences

```
push3| The byte to count
push2| Size of source buffer
push1| Source address
Ret  | Count in EAX. 0 if none
Note | -
```

ascii(1)

Display ASCII equivalences

```
push1| Char or Hex value to display
Ret  | -
Note | -
```

str_trim(2)

Trim a 0-ended string

```
push2| # of chars to cut
push1| Address of the 0-ended string
Ret  | The same string trimmed
Note | String will be 0-ended
```

str_wordcnt(1)/1

Count words of a 0-ended string

```
push1| Address of string
Ret  | Word count
Note | Single char is a word
```

str_token(2)

Display tokens off a 0-ended string

```
push2| Addr of 0-ended delimiter string
push1| Addr of the string
Ret  | -
Note | eg, delimiter db '? .-',0
```

str_copy(3)

Copy a string to another array

```
push3| Size of bytes to copy
push2| Source string's address
push1| Target buffer's address
Ret  | Copied string at the sent address
Note | Target must fit the size to copy
     | Target buffer is 0-ended
```

str_cmpz(2)/1

Compare two 0-ended strings

```
push2| String 1 address
push1| String 2 address
Ret  | EAX = 1 if Equal, 0 if not
Note | The two strings mus be 0-ended
```

str_cmp(3)/1

Compare strings with size

```
push3| Size of bytes to compare
push2| String 1 address
push1| String 2 address
Ret  | EAX = 1 if Equal, 0 if not
Note | -
```

str_find(3)/1

Search for a sub-string

```
push3| Size of key string
push2| Addr of key string
push1| Addr of 0-ended source string
Ret  | Address. -1 -Not found
Note | Case-sensitive
```

str_findz(2)/1

Search for a sub-string from a C string

```
push2| Addr of 0-ended search/key string
push1| Addr of 0-ended source string
Ret  | Location. -1 = Not found
Note | Case-sensitive
```

str_append(5)/1

Combine two 0-ended strings with size & copy to a new 0-ended buffer

```
push5| # of bytes to copy from str2
push4| Separator byte. 0-if none
push1| Addr of new buffer
push3| Addr of str2
push2| Addr of str1 (0-ended)
Ret  | 0-ended combined string
     | EAX - Combined string length
Note | 1st string + 2nd string
     | Buffer must be large enough
```

str_appendz(4)/1

Combine two 0-ended strings

```
push4| Separator byte. 0-if none
push3| Addr of buffer
push2| Addr of second string
push1| Addr of first string
Ret  | 0-ended combined string
     | EAX - Combined string length
Note | 1st string + 2nd string
     | Buffer must be large enough
```

str_reverse(1)

Reverse 0-ended string

```
push1| Address of the 0-ended string
Ret  | The same string reversed
Note | -
```

str_reverse2(1)

Reverse 0-ended aligned-16 string

```
push1| Address of the 0-ended string
Ret  | The same string reversed
Note | String must be align16
```

str_tolower(1)

Change 0-ended string to lower case

```
push1| Address of the 0-ended string
Ret  | The same string converted
Note | -
```

str_toupper(1)

Change 0-ended string to upper case

```
push1| Address of the 0-ended string
Ret  | The same string converted
Note | -
```

str_length(1)/1

Find length of a 0-ended string

```
push1| String's address
Ret  | size in EAX.
Note | -
```

str_lengthd(2)/1

Get length of a 0-ended string with delimiter

```
push2| Delimiter byte
push1| String's address
Ret  | size in EAX
Note | -
```

str_length2(1)/1

Find length of a 0-ended string (aligned)

```
push1| String's address
Ret  | size in EAX.
Note | String must be aligned 16
```

digitprob(2)/1

Probe decimal digit at specified location of a signed decimal value.

```
push2| Digit position to probe (#1=MSD)
push1| The decimal value to probe from
Ret  | Decimal digit. -1 = Error
Note | -
```

digitprobu(2)/1

Probe decimal digit at specified location of an unsigned decimal value.

```
push2| Digit position to probe (#1=MSD)
push1| The decimal value to probe from
Ret  | Decimal digit. -1 = Error
Note | -
```

digithprob(2)/1

Probe hex digit at specified location of a signed hexadecimal value.

```
push2| Digit position to probe (#1=MSD)
push1| The decimal value to probe from
Ret  | Hex digit. -1 = Error
Note | -
```

digithprobu(2)/1

Probe hex digit at specified location of an unsigned hexadecimal value.

```
push2| Digit position to probe (#1=MSD)
push1| The decimal value to probe from
Ret  | Hex digit. -1 = Error
Note | -
```

digitscan(2)/1

Find a decimal digit from a signed decimal value.

```
push2| Decimal digit to find
push1| The decimal value to find from
Ret  | Digit position.#1 = MSD. -1 = Error
Note | -
```

digitscanu(2)/1

Find a decimal digit from a unsigned decimal value.

```
push2| Decimal digit to find
push1| The decimal value to find from
Ret  | Digit position.#1 = MSD. -1 = Error
Note | -
```

digithscan(2)/1

Find a hex digit from a signed hexadecimal value.

```
push2| Hex digit to find
push1| The hex value to find from
Ret  | Digit position.#1 = MSD. -1 = Error
Note | -
```

digithscanu(2)/1

Find a hex digit from a unsigned hexadecimal value.

```
push2| Hex digit to find
push1| The Hex value to find from
Ret  | Digit position.#1 = MSD. -1 = Error
Note | -
```

digitcount(1)/1

Count the number of digits of a signed decimal value.

```
push1| The decimal value to count
Ret  | Digits count
Note | -
```

digitcountu(1)/1

Count the number of digits of an unsigned decimal value.

```
push1| The decimal value to count
Ret  | Digits count
Note | -
```

digithcount(1)/1

Count the number of digits of a signed hexadecimal value.

```
push1| The hexadecimal value to count
Ret  | Digits count
Note | -
```

digithcountu(1)/1

Count the number of digits of an unsigned hexadecimal value.

```
push1| The hexadecimal value to count
Ret  | Digits count
Note | -
```

scan_dword(3)/1

Find first matching dword

```
push3| Size of bytes to scan
push2| The dword key immediate
push1| Starting location
Ret  | Offset. -1 if not found
Note | -
```

scan_word(3)/1

Find first matching two-byte
push3| Size of bytes to scan
push2| The word key immediate
push1| Starting location
Ret | Offset. -1 if not found
Note | -

scan_byte(3)/1

Find first matching byte
push3| Size of bytes to scan
push2| The byte key immediate
push1| Starting location
Ret | Offset. -1 if not found
Note | -

sort_byte(3)

Sort char/byte array (signed)
push3| 0-ascending, 1-descending
push2| Number of array elements
push1| Address of array (DB)
Ret | Return the sorted array
Note | Elements are of type DB

sort_int(3)

Sort array of signed int (bubble sort)
push3| 0-ascending, 1-descending
push2| Number of array elements
push1| Address of array
Ret | Return the sorted array
Note | Elements are of type DWORD

sort_intu(3)

Sort array of unsigned int (bubble sort)
push3| 0-ascending, 1-descending
push2| Number of array elements
push1| Address of array
Ret | Return the sorted array
Note | Elements are of type DWORD

sort_dbl(3)

Sort array of doubles (bubble sort)
push3| 0-ascending, 1-descending
push2| Number of array elements
push1| Address of array
Ret | Return the sorted array
Note | Elements are of type DQ

sort_dblx(3)

Sort array of REAL10 (bubble sort)
push3| 0-ascending, 1-descending
push2| Number of array elements
push1| Address of array
Ret | Return the sorted array
Note | Elements are of type DT

sort_flt(2)

Sort array of singles (bubble sort)
push3| 0-ascending, 1-descending
push2| Number of array elements
push1| Address of array (DD)
Ret | Return the sorted array
Note | Elements are of type DD

aprnhbyte(3)

Display array of unsigned hex bytes. Little-Endian
push3| Separator byte. 0 if none
push2| Number of elements
push1| Address of array
Ret | -
Note | Elements are of type DB

aprnhbyter(3)

Display array of unsigned hex bytes. Reversed-view
push3| Separator byte. 0 if none
push2| Number of elements
push1| Address of array
Ret | -
Note | Elements are of type DB

aprnbyte(3)

Display array of signed int bytes. Little-Endian
push3| Separator byte. 0 if none
push2| Number of elements
push1| Address of array
Ret | -
Note | Elements are of type DB

aprnbyteu(3)

Display array of unsigned int bytes. Little-Endian
push3| Separator byte. 0 if none
push2| Number of elements
push1| Address of array
Ret | -
Note | Elements are of type DB

aprnbyter(3)

Display array of signed int bytes. Reversed-view
push3| Separator byte. 0 if none
push2| Number of elements
push1| Address of array
Ret | -
Note | Elements are of type DB

aprnbyteur(3)

Display array of unsigned bytes. Reversed-view
push3| Separator byte. 0 if none
push2| Number of elements
push1| Address of array
Ret | -
Note | Elements are of type DB

aprnword(3)

Display array of signed words. Little-Endian
push3| Separator byte. 0 if none
push2| Number of elements
push1| Address of array
Ret | -
Note | Elements are of type DW

aprnwordu(3)

Display array of unsigned words. Little-Endian

```
push3| Separator byte. 0 if none
push2| Number of elements
push1| Address of array
Ret  | -
Note | Elements are of type DW
```

aprnwordr(3)

Display array of signed words. Reversed-view

```
push3| Separator byte. 0 if none
push2| Number of elements
push1| Address of array
Ret  | -
Note | Elements are of type DW
```

aprnwordur(3)

Display array of unsigned words. Reversed-view

```
push3| Separator byte. 0 if none
push2| Number of elements
push1| Address of array
Ret  | -
Note | Elements are of type DW
```

aprnintd(3)

Display array of signed dwords. Little-Endian

```
push3| Separator byte. 0 if none
push2| Number of elements
push1| Address of array
Ret  | -
Note | Elements are of type DD
```

aprnintdu(3)

Display array of unsigned dwords. Little-Endian

```
push3| Separator byte. 0 if none
push2| Number of elements
push1| Address of array
Ret  | -
Note | Elements are of type DD
```

aprnintdr(3)

Display array of signed dwords. Reversed-view

```
push3| Separator byte. 0 if none
push2| Number of elements
push1| Address of array
Ret  | -
Note | Elements are of type DD
```

aprnintdur(3)

Display array of unsigned dwords. Reversed-view

```
push3| Separator byte. 0 if none
push2| Number of elements
push1| Address of array
Ret  | -
Note | Elements are of type DD
```

aprnint(3)

Display array of signed longs. Little-Endian

```
push3| Separator byte. 0 if none
push2| Number of elements
push1| Address of array
Ret  | -
Note | Elements are of type DQ
```

aprnintu(3)

Display array of unsigned longs. Little-Endian

```
push3| Separator byte. 0 if none
push2| Number of elements
push1| Address of array
Ret  | -
Note | Elements are of type DQ
```

aprnintr(3)

Display array of signed longs. Reversed view

```
push3| Separator byte. 0 if none
push2| Number of elements
push1| Address of array
Ret  | -
Note | Elements are of type DQ
```

aprnintur(3)

Display array of unsigned longs. Reversed-view

```
push3| Separator byte. 0 if none
push2| Number of elements
push1| Address of array
Ret  | -
Note | Elements are of type DQ
```

aprnflt(3)

Display array of REAL4s. Little-Endian

```
push3| Separator byte. 0 if none
push2| Number of elements
push1| Address of array
Ret  | -
Note | Elements are of type DD/REAL4
```

aprnfltr(3)

Display array of REAL4s. Reversed view

```
push3| Separator byte. 0 if none
push2| Number of elements
push1| Address of array
Ret  | -
Note | Elements are of type DD/REAL4
```

aprndbl(3)

Display array of REAL8s. Little-Endian

```
push3| Separator byte. 0 if none
push2| Number of elements
push1| Address of array
Ret  | -
Note | Elements are of type DQ/REAL8
```

aprndblr(3)

Display array of REAL8s. Reversed view

```
push3| Separator byte. 0 if none
push2| Number of elements
push1| Address of array
Ret  | -
Note | Elements are of type DQ/REAL8
```

aprndblx(3)

Display array of REAL10s. Little-Endian

```
push3| Separator byte. 0 if none
push2| Number of elements
push1| Address of array
Ret  | -
Note | Elements are of type DT/REAL10
```

aprndblxr(3)

Display array of REAL10s. Reversed view

```
push3| Separator byte. 0 if none
push2| Number of elements
push1| Address of array
Ret  | -
Note | Elements are of type DT/REAL10
```

aprnhbytef(3)

Display array of unsigned hex bytes. Little-Endian. Formatted.

```
push3| Separator byte. 0 if none
push2| Number of elements
push1| Address of array
Ret  | -
Note | Elements are of type DB
```

aprnhbyterf(3)

Display array of unsigned hex bytes. Reversed-view. Formatted.

```
push3| Separator byte. 0 if none
push2| Number of elements
push1| Address of array
Ret  | -
Note | Elements are of type DB
```

aprnbytef(3)

Display array of signed bytes. Little-Endian. Formatted.

```
push3| Separator byte. 0 if none
push2| Number of elements
push1| Address of array
Ret  | -
Note | Elements are of type DB
```

aprnbyteuf(3)

Display array of unsigned bytes. Little-Endian. Formatted.

```
push3| Separator byte. 0 if none
push2| Number of elements
push1| Address of array
Ret  | -
Note | Elements are of type DB
```

aprnbyterf(3)

Display array of signed bytes. Reversed-view. Formatted.

```
push3| Separator byte. 0 if none
push2| Number of elements
push1| Address of array
Ret  | -
Note | Elements are of type DB
```

aprnbyteurf(3)

Display array of unsigned bytes. Reversed-view. Formatted.

```
push3| Separator byte. 0 if none
push2| Number of elements
push1| Address of array
Ret  | -
Note | Elements are of type DB
```

aprnwordf(3)

Display array of signed words. Little-Endian. Formatted.

```
push3| Separator byte. 0 if none
push2| Number of elements
push1| Address of array
Ret  | -
Note | Elements are of type DW
```

aprnworduf(3)

Display array of unsigned words. Little-Endian. Formatted.

```
push3| Separator byte. 0 if none
push2| Number of elements
push1| Address of array
Ret  | -
Note | Elements are of type DW
```

aprnwordrf(3)

Display array of signed words. Reversed-view. Formatted.

```
push3| Separator byte. 0 if none
push2| Number of elements
push1| Address of array
Ret  | -
Note | Elements are of type DW
```

aprnwordurf(3)

Display array of unsigned words. Reversed-view. Formatted.

```
push3| Separator byte. 0 if none
push2| Number of elements
push1| Address of array
Ret  | -
Note | Elements are of type DW
```

aprnintdf(3)

Display array of signed dwords. Little-Endian. Formatted.

```
push3| Separator byte. 0 if none
push2| Number of elements
push1| Address of array
Ret  | -
Note | Elements are of type DD
```

aprnintduf(3)

Display array of unsigned dwords. Little-Endian. Formatted.

```
push3| Separator byte. 0 if none
push2| Number of elements
push1| Address of array
Ret  | -
Note | Elements are of type DD
```

aprnintdrf(3)

Display array of signed dwords. Reversed-view. Formatted.

```
push3| Separator byte. 0 if none
push2| Number of elements
push1| Address of array
Ret  | -
Note | Elements are of type DD
```

aprnintdurf(3)

Display array of unsigned dwords. Reversed-view. Formatted.

```
push3| Separator byte. 0 if none
push2| Number of elements
push1| Address of array
Ret  | -
Note | Elements are of type DD
```

aprnintf(3)

Display array of signed longs. Little-Endian. Formatted.

```
push3| Separator byte. 0 if none
push2| Number of elements
push1| Address of array
Ret  | -
Note | Elements are of type DQ
```

aprnintuf(3)

Display array of unsigned longs. Little-Endian. Formatted.

```
push3| Separator byte. 0 if none
push2| Number of elements
push1| Address of array
Ret  | -
Note | Elements are of type DQ
```

aprnintrf(3)

Display array of signed longs. Reversed view. Formatted.

```
push3| Separator byte. 0 if none
push2| Number of elements
push1| Address of array
Ret  | -
Note | Elements are of type DQ
```

aprninturf(3)

Display array of unsigned longs. Reversed-view. Formatted.

```
push3| Separator byte. 0 if none
push2| Number of elements
push1| Address of array
Ret  | -
Note | Elements are of type DQ
```

aprnfltf(3)

Display array of REAL4s. Little-Endian. Formatted.

```
push3| Separator byte. 0 if none
push2| Number of elements
push1| Address of array
Ret  | -
Note | Elements are of type DD/REAL4
```

aprnfltrf(3)

Display array of REAL4s. Reversed view. Formatted.

```
push3| Separator byte. 0 if none
push2| Number of elements
push1| Address of array
Ret  | -
Note | Elements are of type DD/REAL4
```

aprndblf(3)

Display array of REAL8s. Little-Endian. Formatted.

```
push3| Separator byte. 0 if none
push2| Number of elements
push1| Address of array
Ret  | -
Note | Elements are of type DQ/REAL8
```

aprndblrf(3)

Display array of REAL8s. Reversed view. Formatted.

```
push3| Separator byte. 0 if none
push2| Number of elements
push1| Address of array
Ret  | -
Note | Elements are of type DQ/REAL8
```

prnchrp(2)

Display char pattern

```
push| Push the char to use
push| Push number of times to print
Ret  | -
Note | -
```

prnspaces(1)

Print multiple whitespace

```
push| Push number of whitespaces
Ret  | -
Note | Arguments on stack
```

prnstreg(1)***

Display short string off register EAX

```
EAX  | The string immediate
Ret  | -
Note | -
```

prnlines(1)

Print multiple lines

```
push1| Push number of lines
Ret  | -
Note | -
```

prnline

Print a newline

```
Arg  | -
Ret  | -
Note | -
```

prnspace

Print a whitespace

```
Arg  | -
Ret  | -
Note | -
```

prnchar(1)

Display a char byte from memory

```
push1| The char's address
Ret  | -
Note | -
```

prnchr(1)

Display a char byte

```
push1| The char value
Ret  | -
Note | Use “dword” override if so needed
```

prnstrd(2)

Display a delimiter-ended string

```
push2| Delimiter byte value
push1| The string address
Ret  | -
Note | -
```

prnstrz(1)

Display a 0-ended string

```
push1| The string address
Ret  | -
Note | -
```

prnstr(2)

Display string with size

```
push2| Size of bytes to display
push1| The string address
Ret  | -
Note | -
```

readchr(1)

Get a char from keyb and save to memory

```
push1| The address to save the char to
Ret  | -
Note | Var should be of type DB/rb/resb
```

readch/1

Get a char from keyb into AL, with echo

```
Arg  | -
Ret  | The key in AL
Note | Internal use
     | Use halt if needed
```

readstr(1)/1

Get string from keyboard (echoed)

```
push1| Buffer's address to save string
Ret  | 0-ended string in sent buffer
     | EAX= # of actual bytes in
Note | Press ENTER to finish
     | Buffer must be large enough
     | Ret -1 signals error
     | String will be 0-ended
```

mem_alloc(1)/1

Request memory of n bytes

```
push1| Bytes requested
Ret  | Pointer to the mem block. -1 = none
Note | -
```

mem_free(1)/1

Free memory allocated by mem_alloc

```
push1| Pointer from mem_alloc
Ret  | 1 = success. -1 = error
Note | -
```

mem_load(1)/2

Load a file to memory

```
push1| Filename path string
Ret  | EAX - block pointer
     | EDX - Size
Note | Should re-claim memory at EAX
     | File name must be 0-ended
```

file_new(1)/1

Create a new file

```
push1| File name path string
Ret  | -1 = Error. 0 = success
Note | Filename string must be 0-ended
```

file_open(2)/1

Open an existing file for reading/writing

```
push2| Operation:
       0-Read
       1-Write
push1| File name path string
Ret  | File descriptor. -1 if error
Note | Filename string must be 0-ended
```

file_read(3)/1

Read from an existing opened file

```
push3| Number of bytes to read
push2| Input buffer
push1| File handle from file_open
Ret  | Number of actual read
Note | Filename string must be 0-ended
     | Input buffer must be >= arg3
```

file_write(3)/1

Write to an existing opened file

```
push3| Number of bytes to write
push2| Source buffer's address
push1| File handle from file_open
Ret  | Number of actual writes
Note | Filename string must be 0-ended
     | Handle must be opened-write
```

file_close(1)

Close handle issued by file_open

```
push1| File handle
Ret  | -
Note | EAX will be zeroed
```

file_size(1)/1

Get filesize of an opened file

```
push1| File handle from file_open
Ret  | Size in bytes. -1 says error
Note | -
```

file_copy(2)

Copy a file to a new file

```
push2| Address of new file name
push1| Address of source file name
Ret  | -
Note | Both must be 0-ended string
```

file_delete(1)

Delete a file on disk

```
push1| Address of filename string
Ret  | 0-Fail (Win64), -ve (Linux64)
Note | Handle must be already closed
```

timer_start/1

Start timer

```
Arg  | -
Ret  | Pointer to memory
Note | Must be followed by timer_stop
       to clear allocated memory
```

timer_stop(1)

Stop current timer and display time lapse in seconds

```
push1| pointer from timer_start
Ret  | -
Note | Should come after timer_start
```

delay(1)

Put execution at delay

```
push1| Milliseconds
Ret  | -
Note | 1000ms = 1s
```

get_ticks/1

Get tick count

```
-    | -
Ret  | Ticks (ms) in EAX
Note | 1000ms = 1s
```

halt

Pause screen

```
Arg  | -
Ret  | -
Note | Hit Enter to continue
```

exitp

Pause then exit to system

```
Arg  | -
Ret  | -
Note | Hit Enter to exit
```

exitx

Exit to system

```
Arg  | -
Ret  | -
Note | -
```