

## 5.9.2. Memory Commands

---

### 5.9.2.1. base - print or set address offset

---

```
=> help base

base - print or set address offset

Usage:

base

    - print address offset for memory commands

base off

    - set address offset for memory commands to 'off'

=>
```

You can use the `base` command (short: `ba`) to print or set a "base address" that is used as address offset for all memory commands; the default value of the base address is 0, so all addresses you enter are used unmodified. However, when you repeatedly have to access a certain memory region (like the internal memory of some embedded [Power Architecture®](#) processors) it can be very convenient to set the base address to the start of this area and then use only the offsets:

```
=> base

Base Address: 0x00000000

=> md 0 0xc

00000000: 00ff43a6 00000000 ffffffff ffffffff    ..C.....
00000010: 00ff43a6 00000000 ffffffff ffffffff    ..C.....
00000020: 0c904d01 320b4481 1ea3d0a2 c498293a    ..M.2.D.....):

=> base 0x100000

Base Address: 0x00100000

=> md 0 0xc

00100000: 0e0a0e81 bd86200a 60a19054 2c12c402    .....  `...T,...
00100010: c101d028 00438198 7ab01239 62406128    ... (.C..z..9b@a(
00100020: 0c900d05 320b4581 1ca3d0a2 c498293a    ....2.E.....):

=>
```

### 5.9.2.2. crc32 - checksum calculation

---

The `crc32` command (short: `crc`) can be used to calculate a CRC32 checksum over a range of memory:

```
=> crc 0x100004 0x3FC
```

```
CRC32 for 00100004 ... 001003ff ==> 8083764e
=>
```

When used with 3 arguments, the command stores the calculated checksum at the given address:

```
=> crc 0x100004 0x3FC 0x100000
CRC32 for 00100004 ... 001003ff ==> 8083764e
=> md 0x100000 4
00100000: 8083764e bd86200a 60a19054 2c12c402    ..vN.. ``..T,...
=>
```

As you can see, the CRC32 checksum was not only printed, but also stored at address 0x100000.

### 5.9.2.3. cmp - memory compare


```
=> help cmp
cmp - memory compare

Usage:
cmp [.b, .w, .l] addr1 addr2 count
=>
```

With the `cmp` command you can test if the contents of two memory areas are identical or not. The command will either test the whole area as specified by the 3rd (length) argument, or stop at the first difference.

```
=> cmp 0x100000 0x200000 0x400
word at 0x00100000 (0x8083764e) != word at 0x00200000 (0x27051956)
Total of 0 words were the same
=> md 0x100000 0xc
00100000: 8083764e bd86200a 60a19054 2c12c402    ..vN.. ``..T,...
00100010: c101d028 00438198 7ab01239 62406128    ...(.C..z..9b@a(
00100020: 0c900d05 320b4581 1ca3d0a2 c498293a    ....2.E.....):
=> md 0x200000 0xc
00200000: 27051956 552d426f 6f742032 3030392e    '..VU-Boot 2009.
00200010: 31312e31 20284665 62203035 20323031    11.1 (Feb 05 201
00200020: 30202d20 30383a35 373a3132 29000000    0 - 08:57:12)...
=>
```

Like most memory commands the `cmp` can access the memory in different sizes: as 32 bit (long word), 16 bit (word) or 8 bit (byte) data. If invoked just as `cmp` the default size (32 bit or long words) is used; the same can be selected explicitly by typing `cmp.l` instead. If you want to access memory as 16 bit or word data, you can use the variant `cmp.w` instead; and to access memory as 8 bit or byte data please use `cmp.b`.

 Please note that the *count* argument specifies the number of data items to process, i. e. the number of long words or words or bytes to compare.

```
=> cmp.l 0x100000 0x200000 0x400
word at 0x00100000 (0x8083764e) != word at 0x00200000 (0x27051956)
Total of 0 words were the same
=> cmp.w 0x100000 0x200000 0x800
halfword at 0x00100000 (0x8083) != halfword at 0x00200000 (0x2705)
Total of 0 halfwords were the same
=> cmp.b 0x100000 0x200000 0x1000
byte at 0x00100000 (0x80) != byte at 0x00200000 (0x27)
Total of 0 bytes were the same
=>
```

#### **5.9.2.4. cp - memory copy**

```
=> help cp
cp - memory copy

Usage:
cp [.b, .w, .l] source target count
=> help cp
cp - memory copy

Usage:
cp [.b, .w, .l] source target count
=>
```

The `cp` is used to copy memory areas.

```
=> cp 0x100000 0x200000 0x10000
=>
```

The `cp` understands the type extensions `.l`, `.w` and `.b` :

```
=> cp.l 0x200000 0x100000 0x10000
```

```
=> cp.w 0x200000 0x100000 0x20000
=> cp.b 0x200000 0x100000 0x40000
=>
```

### 5.9.2.5. md - memory display

```
=> help md
md - memory display

Usage:
md [.b, .w, .l] address [# of objects]
=>
```

The `md` can be used to display memory contents both as hexadecimal and ASCII data.

```
=> md 0x100000
00100000: 8083764e bd86200a 60a19054 2c12c402    ..vN.. .`..T,...
00100010: c101d028 00438198 7ab01239 62406128    ...(.C...z..9b@a(
00100020: 0c900d05 320b4581 1ca3d0a2 c498293a    ....2.E.....):
=>
00100030: 58f5c828 6029e009 d0718131 154b105b    X..(`)...q.1.K.[
00100040: 9019a424 7423a001 e064013c 016a0070    ...$t#...d.<.j.p
00100050: d0809820 12437140 0064e018 424be2a9    ... .Cq@d..BK..
=>
```

This command, too, can be used with the type extensions `.l`, `.w` and `.b` :

```
=>
=> md.w 0x100000
00100000: 8083 764e bd86 200a 60a1 9054 2c12 c402    ..vN.. .`..T,...
00100010: c101 d028 0043 8198    ...(.C..
=> md.b 0x10000
```

The last displayed memory address and the value of the count argument are remembered, so when you enter `md` again *without arguments* it will automatically continue at the next address, and use the same count again.

```
=> md.b 0x100000 0x20
00100000: 2f 83 00 00 40 9e ff 38 38 60 00 00 4b ff ff 3c    /...@...88`..K..<
00100010: 83 5e 00 0c 80 9e 00 08 2b 9a 00 ff 82 9e 00 10    .^.....+.....
=> md.w 0x100000
```

```

00100000: 2f83 0000 409e ff38 3860 0000 4bff ff3c  /...@...88`..K..<
00100010: 835e 000c 809e 0008 2b9a 00ff 829e 0010  .^.....+.....
00100020: 82be 0014 7f45 d378 409d 000c 3b40 00ff  ....E.x@...;@..
00100030: 38a0 00ff 2b95 00ff 409d 0008 3aa0 00ff  8...+...@....:...
=> md 0x100000

00100000: 2f830000 409eff38 38600000 4bffff3c  /...@...88`..K..<
00100010: 835e000c 809e0008 2b9a00ff 829e0010  .^.....+.....
00100020: 82be0014 7f45d378 409d000c 3b4000ff  ....E.x@...;@..
00100030: 38a000ff 2b9500ff 409d0008 3aa000ff  8...+...@....:...
00100040: 8002021c 3bfb000a 7f9f0040 419d002c  ....;.....@A.,
00100050: 2f9a0000 419e0014 7c1f0050 3925ffff  /...A...|..P9%..
00100060: 7f890040 419d0014 7fe3fb78 4bf1401d  ...@A.....xK.@.
00100070: 7c651b78 48000014 3c00bfff 6000ffff  |e.xH...<...`...
=>

```

### 5.9.2.6. mm - memory modify (auto-incrementing)

```

=> help mm

mm - memory modify (auto-incrementing address)

Usage:
mm [.b, .w, .l] address

=>

```

The `mm` is a method to interactively modify memory contents. It will display the address and current contents and then prompt for user input. If you enter a legal hexadecimal number, this new value will be written to the address. Then the next address will be prompted. If you don't enter any value and just press ENTER, then the contents of this address will remain unchanged. The command stops as soon as you enter any data that is not a hex number (like `.`):

```

=>

=> mm 0x100000

00100000: 8083764e ? 0
00100004: bd86200a ? 0xaabbccdd
00100008: 60a19054 ? 0x01234567
0010000c: 2c12c402 ? .
=> md 0x100000 0x10

00100000: 00000000 aabbccdd 01234567 2c12c402  ....#Eg,...
00100010: c101d028 00438198 7ab01239 62406128  ...(.C..z...9b@a(

```

```

00100020: 0c900d05 320b4581 1ca3d0a2 c498293a    ....2.E.....):
00100030: 58f5c828 6029e009 d0718131 154b105b    X..(`)...q.1.K.[
=>

```

Again this command can be used with the type extensions `.l`, `.w` and `.b` :

```

=>

=> mm.w 0x100000

00100000: 0000 ? 0x0101
00100002: 0000 ? 0x0202
00100004: aabb ? 0x4321
00100006: ccdd ? 0x8765
00100008: 0123 ? .

=> md 0x100000 0x10

00100000: 01010202 43218765 01234567 2c12c402    ....C!.e.#Eg,...
00100010: c101d028 00438198 7ab01239 62406128    ...(.C..z...9b@a(
00100020: 0c900d05 320b4581 1ca3d0a2 c498293a    ....2.E.....):
00100030: 58f5c828 6029e009 d0718131 154b105b    X..(`)...q.1.K.[

=>

=>

=> mm.b 0x100000

00100000: 01 ? 0x48
00100001: 01 ? 0x65
00100002: 02 ? 0x6c
00100003: 02 ? 0x6c
00100004: 43 ? 0x6f
00100005: 21 ? 0x20
00100006: 87 ? 0x20
00100007: 65 ? 0x20
00100008: 01 ? .

=> md 0x100000 0x10

00100000: 48656c6c 6f202020 01234567 2c12c402    Hello    .#Eg,...
00100010: c101d028 00438198 7ab01239 62406128    ...(.C..z...9b@a(
00100020: 0c900d05 320b4581 1ca3d0a2 c498293a    ....2.E.....):
00100030: 58f5c828 6029e009 d0718131 154b105b    X..(`)...q.1.K.[

=>

```

### 5.9.2.7. mtest - simple RAM test

---

```
=> help mtest

mtest - simple RAM read/write test

Usage:

mtest [start [end [pattern [iterations]]]]

=>
```


The mtest provides a simple memory test.


```
=>

=> mtest 0x100000 0x200000

Pattern 00000000 Writing... Reading...Pattern FFFFFFFF Writing... Reading...Pattern
00000001 Writing... Reading...Pattern FFFFFFFE Writing... Reading...Pattern 00000002
Writing... Reading...Pattern FFFFFFFD Writing... Reading...Pattern 00000003
Writing... Reading...Pattern FFFFFFFC Writing... Reading...Pattern 00000004
Writing... Reading...Pattern FFFFFFFB Writing... Reading...Pattern 00000005
Writing... Reading...Pattern FFFFFFFA Writing... Reading...Pattern 00000006
Writing... Reading...Pattern FFFFFFF9 Writing... Reading...Pattern 00000007
Writing... Reading...Pattern FFFFFFF8 Writing... Reading...Pattern 00000008
Writing... Reading...Pattern FFFFFFF7 Writing... Reading...Pattern 00000009
Writing... Reading...Pattern FFFFFFF6 Writing... Reading...Pattern 0000000A
Writing... Reading...Pattern FFFFFFF5 Writing... Reading...Pattern 0000000B
Writing... Reading...Pattern FFFFFFF4 Writing... Reading...Pattern 0000000C
Writing... Reading...Pattern FFFFFFF3 Writing... Reading...Pattern 0000000D
Writing... Reading...Pattern FFFFFFF2 Writing... Reading...Pattern 0000000E
Writing... Reading...Pattern FFFFFFF1 Writing... Reading...Pattern 0000000F
Writing... Reading...

=>
```

 This tests writes to memory, thus modifying the memory contents. It will fail when applied to ROM or flash memory.

 This command may crash the system when the tested memory range includes areas that are needed for the operation of the U-Boot firmware (like exception vector code, or U-Boot's internal program code, stack or heap memory areas).

### 5.9.2.8. mw - memory write (fill)

---

```
=> help mw

mw - memory write (fill)

Usage:

mw [.b, .w, .l] address value [count]
```

```
=>
```

The `mw` is a way to initialize (fill) memory with some value. When called without a count argument, the value will be written only to the specified address. When used with a count, then a whole memory areas will be initialized with this value:

```
=> md 0x100000 0x10

00100000: 0000000f 00000010 00000011 00000012 .....
00100010: 00000013 00000014 00000015 00000016 .....
00100020: 00000017 00000018 00000019 0000001a .....
00100030: 0000001b 0000001c 0000001d 0000001e .....

=> mw 0x100000 0xaabbccdd

=> md 0x100000 0x10

00100000: aabbccdd 00000010 00000011 00000012 .....
00100010: 00000013 00000014 00000015 00000016 .....
00100020: 00000017 00000018 00000019 0000001a .....
00100030: 0000001b 0000001c 0000001d 0000001e .....

=> mw 0x100000 0 6

=> md 0x100000 0x10

00100000: 00000000 00000000 00000000 00000000 .....
00100010: 00000000 00000000 00000015 00000016 .....
00100020: 00000017 00000018 00000019 0000001a .....
00100030: 0000001b 0000001c 0000001d 0000001e .....

=>
```

This is another command that accepts the type extensions `.l`, `.w` and `.b` :

```
=> mw.w 0x100004 0x1155 6

=> md 0x100000 0x10

00100000: 00000000 11551155 11551155 11551155 .....U.U.U.U.U
00100010: 00000000 00000000 00000015 00000016 .....
00100020: 00000017 00000018 00000019 0000001a .....
00100030: 0000001b 0000001c 0000001d 0000001e .....

=> mw.b 0x100007 0xff 7

=> md 0x100000 0x10

00100000: 00000000 115511ff ffffffff ffff1155 .....U.....U
00100010: 00000000 00000000 00000015 00000016 .....
00100020: 00000017 00000018 00000019 0000001a .....
00100030: 0000001b 0000001c 0000001d 0000001e .....
```



```
=>
```

### **5.9.2.9. nm - memory modify (constant address)**

```
=> help nm
nm - memory modify (constant address)

Usage:
nm [.b, .w, .l] address

=>
```

The `nm` command (non-incrementing memory modify) can be used to interactively write different data several times to the same address. This can be useful for instance to access and modify device registers:

```
=>
=> nm.b 0x100000
00100000: 00 ? 0x48
00100000: 48 ? 0x65
00100000: 65 ? 0x6c
00100000: 6c ? 0x6c
00100000: 6c ? 0x6f
00100000: 6f ? .
=> md 0x100000 8
00100000: 6f000000 115511ff ffffffff ffff1155      o....U.....U
00100010: 00000000 00000000 00000015 00000016      .....
=>
```

The `nm` command too accepts the type extensions `.l`, `.w` and `.b`.


### **5.9.2.10. loop - infinite loop on address range**

```
=> help loop
loop - infinite loop on address range

Usage:
loop [.b, .w, .l] address number_of_objects

=>
```

The `loop` command reads in a tight loop from a range of memory. This is intended as a special form of a memory test, since this command tries to read the memory as fast as possible.

 This command will never terminate. There is no way to stop it but to reset the board!

```
=> loop 100000 8
```