bso2 Monitor Reference

Target: W65C02EDU | Source: `bso2.asm` | Generated: 2026-02-14

0) Naming

`bs02` is the preferred written form in this manual.

It is intentionally dual-meaning and retro-styled:

- `b` = Basic
- `s` = System
- `0` = Operations (also read as letter `O`)
- `2` = `/2` and 6502 lineage

Stylized-glyph note: a `6` can be used as a visual stand-in for lowercase `b`, so `6s02` is a branding variant of `bs02`, not a different system.

Expanded meaning: `Basic System Operations/2`.

1) Startup / Prompt Behavior

On reset with valid reset cookie, boot choices are `C/W/M`:

C = clear RAM (confirm Y/N)
W = warm start
M = enter monitor

After clear/startup, the sign-on banner (`BSO2_INIT`) is printed:

```
**** basic system operations/2 ****
****       b s o / 2  v0 . 9   ****
****         6 5 0 2           ****
```

Monitor prompt is a single `-` character on a new line.

2) Command Summary

Cmd	Syntax	Behavior / Output	Flags / Notes
`?`	`?`	Short help line.	Quick command list only.
`H`	`H`	Full multi-line help.	Includes interactive caveats.
`Z`	`Z`	Clear RAM after Y/N confirmation.	Zeroes `$0200-$7EFF`. Does not zero `$0000-$01FF` (ZP/stack) or `$7F00-$7FFF` (I/O area).
`W`	`W`	Warm start back into monitor.	No args.
`D`	`D [START [END]]`	Hex+ASCII dump. `END` is inclusive.	`D` alone repeats last span from next address. Hex/ASCII fields show an 8+8 split.

`U`	`U START END`	Disassemble as 65C02 mnemonics and operands.	`END` is inclusive. Emits `ADDR: MNM OPERAND`.
`A`	`A START [INSN]`	Tiny 65C02 assembler, interactive at next address.	Example: `A 1000` `LDA #FF` then prompt `A 1002:`. `.` exits. No labels/forward refs. Relative branches accept absolute hex targets and are range-checked.
`X`	`X START`	Execute from absolute address.	Transfers control via RTS trampoline. NMI while running under `X` breaks back to monitor; target `RTS` also returns to monitor.
`R`	`R [A=HH] [X=HH] [Y=HH]`	Resume last debug context.	Restores `A/X/Y/P/SP/PC` from latest debug snapshot and resumes via `RTI`. Optional `A/X/Y` overrides are applied first. Up-arrow repeat is useful for single-step resume loops.
`M`	`M [START [B0..B15]]`	Modify/deposit memory. Inline deposit supports up to 16 bytes.	Interactive mode: `CR/LF = next`, `.` ends. CRLF pair counts as one next.
`F`	`F START END B0..B15`	Fill inclusive range with repeating 1..16 byte pattern.	No interactive mode. Verifies each write.
`C`	`C SRC_START SRC_END DST_START`	Copy inclusive source range to destination.	Overlap-safe (forward/backward selection). Verifies each write.
`!`	`!F ...`, `!M ...`, `!C ...`, `!A ...`	Force-prefix for protected commands.	Allows access to protected low RAM (`$0000-$03FF`).
`Q`	`Q`	Enter WAI halt loop.	IRQ masked. Resume by NMI (or Reset). NMI latch returns

			cleanly to monitor.
`V`	`V`	Show vector jump chains.	Displays HW vector to final code path.

3) Interactive Caveats

- `M` interactive: two hex digits are required per byte write (00..FF).
- `A` interactive: type one mnemonic/operand per prompt, `.` exits assembler mode.
- `.` exits interactive modify and retains next-address state for subsequent `M`.
- `CR` or lone `LF` advances to next address.
- `CRLF` pair is consumed as a single next-step.
- `F` does not support interactive mode.
- At an empty monitor prompt, `Up Arrow` (`ESC [ A`) repeats and executes the previous command.
- `F/M/C/A` block access to `$0000-$03FF` unless prefixed with `!`. `D` is always allowed.

4) Verify / Error Outputs

Operation	Message / Behavior
Modify verify fail	`M VERIFY FAILED AT ADDR` + failing address.
Fill verify fail	`F VERIFY FAILED AT ADDR` + failing address.
Copy verify fail	`C VERIFY FAILED AT ADDR` + failing address.
Dump range error	`D RANGE ERROR`.
Unassemble range error	`U RANGE ERROR`.
Assembler branch range error	`A BRANCH RANGE ERROR`.
BRK debug context	Printed as three lines: `PREV:` (instruction at BRK opcode), `TRAP:` (register/status line), then `NEXT:` (instruction at resume PC).
Bad syntax	Per-command usage lines (e.g. `USAGE: M [START [B0..B15]]`).

5) API Reference (Macros and Functions)

Use this section when calling monitor functionality from your own assembly code.

5.1) Macro Reference (`macros.inc`)

Macro	Parameters	Behavior / Notes
`PUSH`	`PUSH p1 [,p2] [,p3] [,p4]`	Pushes listed registers in given order. Supported tokens: `A/X/Y/P` (case-insensitive).
`PULL`	`PULL p1 [,p2] [,p3] [,p4]`	Pops listed registers in given order. Keep ordering compatible with prior `PUSH`.
`REPEAT`	`REPEAT Routine, Count`	Calls `JSR Routine` repeatedly `Count` times. Preserves `X` via push/pull.
`PRT_CSTRING`	`PRT_CSTRING Label`	Prints null-terminated string at `Label` via `PRT_C_STRING`.

`DUMP`	`DUMP Start, EndExclusive`	Convenience wrapper for `MEM_DUMP` with explicit exclusive end.
`FILL`	`FILL Start, EndInclusive, B0 [,B1] [,B2] [,B3] [,B4]`	Loads pattern bytes (1..5) and calls `MEM_FILL_PATTERN`. End is inclusive in macro syntax.
`COPY`	`COPY SrcStart, SrcEndInclusive, DstStart`	Calls overlap-safe `MEM_COPY_RANGE`. Source end is inclusive in macro syntax.
`COPY_BLOCK`	`COPY_BLOCK SrcStart, Length, DstStart`	Compatibility wrapper that expands to `COPY SrcStart,(SrcStart+Length-1),DstStart`.
`CMP_CSTRING`	`CMP_CSTRING AddrA, AddrB`	Wrapper for project-specific string compare symbols/routine (`STRCMP_PTR_*`, `STR_COMPARE`). Use only when those symbols are provided by your build.

5.2) Callable Function Reference

Practical entry points for extensions and integration.

Routine	Input	Output	Flags	ZP / Memory Use
`INIT_SERIAL`	None	UART initialized	Unchanged	None
`WRITE_BYTE`	`A`=char	Char sent to UART, LED updated	Unchanged	None
`READ_BYTE`	None	`A`=received char (ROM read)	ROM-defined	None
`CHECK_BYTE`	None	`A`=status	`C=1` if RX empty	None
`RBUF_INIT`	None	Input ring reset	Unchanged	Uses generic buffer descriptor core
`BUF_INIT`	Active descriptor pointers set	Head/Tail/Count zeroed	Unchanged	Uses `BUF_*_PTR`
`BUF_PUT_A`	`A`=byte	Byte queued	`C=0` stored, `C=1` full	Uses `BUF_*_PTR`, `BUF_SIZE`
`BUF_GET_A`	None	`A`=byte	`C=0` byte, `C=1` empty	Uses `BUF_*_PTR`, `BUF_SIZE`
`CMD_DISPATCH`	`A`=command letter	Handler called from table	`C=0` handled, `C=1` unknown	Uses `CMD_TABLE`, `CMD_POST_ACTION`
`MEM_DUMP`	`PTR_DUMP_CUR`=start (inc), `PTR_TEMP`=end (exc)	Formatted hex+ASCII dump with 8+8 separator	Unchanged	Uses `PTR_DUMP_CUR`, `PTR_DUMP_END`, `PTR_LEG`, `MEM_DUMP_CNT`

`MEM_DISASM_65C02`	`PTR_DUMP_CUR` =start (inc), `PTR_TEMP` =end (inc)	65C02 disassembly output (`ADDR: MNM OPERAND`)	Unchanged	Uses `PTR_DUMP_CUR` , `PTR_DUMP_END` , `PTR_TEMP` , `PTR_LEG` , `DIS_*`
`MEM_FILL_PATTERN`	`PTR_DUMP_CUR` =start (inc), `PTR_DUMP_END` =end (exc), `F_COUNT` =pattern length, `F_PATTERN` =pattern bytes	Fills range with repeating pattern	`C=0` complete, `C=1` aborted (verify/protect)	Uses `PTR_DUMP_CUR` , `PTR_DUMP_END` , `F_COUNT` , `F_PATTERN` , `F_PAT_IDX`
`MEM_COPY_RANGE`	`PTR_LEG` =src start (inc), `PTR_DUMP_END` =src end (exc), `PTR_TEMP` =dst start	Copies source to destination (overlap-safe)	`C=0` complete, `C=1` aborted (verify/protect)	Uses `PTR_LEG` , `PTR_DUMP_CUR` , `PTR_DUMP_END` , `PTR_TEMP` , `CMD_PARSE_VAL`
`CMD_DO_ASM`	`CMD_LINE` = `A START [INSN]`	Interactive tiny assembler	`.` exits	Uses `CMD_LINE` , `PTR_TEMP` , opcode tables, and `ASM_*` / `DIS_*` scratch

6) Parser and Buffer Limits

- `CMD_MAX_LEN = 31` characters (excluding null terminator).
- `RBUF_SIZE = 32` bytes.
- One-command history is kept for up-arrow repeat (`CMD_LAST_LINE`).
- Hex token parser accepts 1..4 hex digits, optional `$` prefix.
- `M` and `F` inline byte lists: max 16 bytes each.
- `!` is consumed as a command prefix, then normal parsing continues.

7) Memory Usage

Build Section Usage (current)

Section	ORG	Size (hex)	Size (dec)
PAGE0	`$0040`	`$4A`	74
CODE	`$8000`	`$1A22`	6690
KDATA	`$E000`	`$B25`	2853
UDATA	`$0200`	`$7A`	122
Total	-	`$260B`	9739

RAM Layout Highlights

- PAGE0 starts at `$0040` . Includes parser state, dump state, debug snapshot, vector hooks, and active buffer descriptor pointers.
- UDATA starts at `$0200` :

```
RBUF_DATA    32 bytes
CMD_LINE     32 bytes (31 + NUL)
CMD_LAST_LINE 32 bytes (31 + NUL)
RESET_COOKIE  4 bytes
F_PATTERN    16 bytes
DBG_TAG_BUF   6 bytes
```

8) Notes for Integrators

- Command parser uppercases incoming command bytes before parse/dispatch.
- Command execution is table-driven via `CMD_TABLE`.
- Input buffering now uses a generic descriptor-based core bound to the ring buffer.
- `Q` path relies on NMI latch (`SYSF_NMI_FLAG_M`) and then re-enters monitor cleanly.

9) Legal Notice

- `WDC`, `W65C02`, and `W65C02EDU` are names associated with Western Design Center, Inc.
- This project is independent and is not affiliated with or endorsed by Western Design Center, Inc.
- This repository does not redistribute WDC tool binaries or WDC ROM images.
- Any third-party source code or text should be included only under its original license terms.