

# bso2 Monitor Reference

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

## 0) Naming

`bso2` is the preferred written form in this manual.

It is intentionally dual-meaning and retro-styled:

- `b` = 6 = Basic
- `s` = 5 = System
- `o` = 0 = Operations
- `2` = /2 (homage to IBM System/36)

That makes `6502` a visual shorthand for `bso2`.

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

At reset-cookie boot prompt, a 6-second visible countdown runs; if no key is pressed, default path is `M` (monitor).

A terminal-width prompt follows boot selection: `TERM WIDTH 4=40 8=80 1=132 [8]?`.

Width prompt timeout uses fixed byte `$007B` (`TERM_WIDTH_TIMEOUT`): `00=wait forever`, `01-FF=seconds`, default `08`.

Width persistence policy: `W/M` restore prior width before prompting; `C` starts from default 80 before prompting.

After clear/startup, the sign-on banner (`BS02_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
<code>?</code>	<code>?</code>	Short help line.	Quick command list only.
<code>H</code>	<code>H [A P M S]</code>	Help index or scoped help sections.	<code>H=index, H</code> <code>A=all, H</code> <code>P=protection, H</code> <code>M=memory/tools, H</code> <code>S=steering.</code>
<code>Z</code>	<code>Z</code>	Clear RAM after Y/N confirmation.	Zeroes <code>\$0200-\$7EFF</code> . Does not zero <code>\$0000-\$01FF</code> (ZP/stack) or

			\$7F00-\$FFFF (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. Bare U repeats from saved next-instruction address. 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. Explicit accumulator form like INC A is supported.
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.
N	N	Run to next sequential instruction.	Implements next-stop by patching a temporary BRK at PC+len(opcode) . RAM only; ROM/I/O patch targets are

			rejected. Debug output restores and displays the original stepped-to instruction in CURR: .
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 ..., !N	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.	Spaced format: RST: FFFC > F818 > 8004 > 9F31 > [0080] > 800D . IRQ now appends sub-dispatch lines: BRK: XXXX <name> and HW: YYYY <name> . Bracketed links use [addr16] and indicate a patchable 16-bit RAM trampoline address.

### 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.
- Special repeat behavior: if the previous command was a D ... or U ... form, up-arrow replays bare D/U; D continues by saved span, U by saved next-instruction address.
- F/M/C/A/N block access to \$0000-\$03FF unless prefixed with !. D is always allowed.

- Direct vector-hook edits with !M are non-atomic and debug-only. Writing live bytes at \$0080-\$0088 (especially \$0083-\$0085) can produce mixed-byte jumps, wrong dispatch, hangs/crashes, or temporary vector-name mismatch while patching.
- Game ask hook (GAME ASK, older text: POST ASK) is one-shot: it is set on Reset and when NMI returns to monitor, then cleared after first use.
- Hook flag is fixed/reserved at \$0078 (GAME\_ASK\_PENDING). Manual control: !M 78 01 sets pending; !M 78 00 clears pending.
- Terminal width byte is fixed/reserved at \$007A (TERM\_COLS): 28/50/84 for 40/80/132 columns.
- Terminal-width prompt timeout byte is fixed/reserved at \$007B (TERM\_WIDTH\_TIMEOUT): 00=wait forever, 01-FF=seconds, default 08.
- User ZP range is reserved at \$0090-\$00FF.

## 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 two lines: CURR: and NEXT: on one line, then STATE: on the next line. For N-generated temporary breaks, CURR: shows the restored original instruction.
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 <code>MEM_COPY_RANGE</code> . Source end is inclusive in macro syntax.
COPY_BLOCK	COPY_BLOCK SrcStart, Length, DstStart	Compatibility wrapper that expands to <code>COPY SrcStart,(SrcStart+Length-1),DstStart</code> .
CMP_CSTRING	CMP_CSTRING AddrA, AddrB	Wrapper for project-specific string compare symbols/routine ( <code>STRCMP_PTR_*</code> , <code>STR_COMPARE</code> ). 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 <code>BUF_*_PTR</code>
BUF_PUT_A	A=byte	Byte queued	C=0 stored, C=1 full	Uses <code>BUF_*_PTR</code> , <code>BUF_SIZE</code>
BUF_GET_A	None	A=byte	C=0 byte, C=1 empty	Uses <code>BUF_*_PTR</code> , <code>BUF_SIZE</code>
CMD_DISPATCH	A=command letter	Handler called from table	C=0 handled, C=1 unknown	Uses <code>CMD_TABLE</code> , <code>CMD_POST_ACTI</code> ON
MEM_DUMP	PTR_DUMP_CUR=start (inc), PTR_TEMP=end (exc)	Formatted hex+ASCII dump with 8+8 separator	Unchanged	Uses <code>PTR_DUMP_CUR</code> , <code>PTR_DUMP_END</code> , <code>PTR_LEG</code> , <code>MEM_DUMP_CNT</code>
MEM_DISASM_65C 02	PTR_DUMP_CUR=start (inc), PTR_TEMP=end (inc)	65C02 disassembly output ( ADDR: MNN OPERAND )	Unchanged	Uses <code>PTR_DUMP_CUR</code> , <code>PTR_DUMP_END</code> , <code>PTR_TEMP</code> , <code>PTR_LEG</code> , <code>DIS_*</code>

MEM_FILL_PATTE RN	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	\$0030	\$60	96
CODE	\$8000	\$2725	10021
KDATA	\$A725	\$12A8	4776
UDATA	\$0200	\$7F	127
Total	-	\$3AAC	15020

### RAM Layout Highlights

- PAGE0 starts at \$0030. Includes parser state, dump state, debug snapshot, vector hooks, and active buffer descriptor pointers.
- Guard policy reserves monitor PAGE0 through \$008F; user ZP is reserved at \$0090-\$00FF.
- KDATA floats directly behind CODE (current build start: \$A725).
- Fixed/pinned bytes: GAME\_ASK\_PENDING=\$0078 , BRK\_FLAG=\$0079 , TERM\_COLS=\$007A , TERM\_WIDTH\_TIMEOUT=\$007B , RST\_HOOK=\$0080 , NMI\_HOOK=\$0083 , IRQ\_HOOK=\$0086 , BRK\_HOOK=\$0089 , HW\_HOOK=\$008C .
- Hardware vectors are fixed at the top page: NMI=\$FFFA , RST=\$FFFC , IRQ/BRK=\$FFFF .
- Detailed zero-page map: [ZERO\\_PAGE\\_USAGE.md](#) and [ZERO\\_PAGE\\_USAGE.pdf](#) .
- 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) Planned Commands (Appendix, Provisional)

This appendix documents planned command architecture and roadmap intent only.

**Proviso:** change is constant. These plans are not stable API and may change before publish.

### 9.1) Grammar Direction

- Primary model: `noun verb [args...]` (namespace first, action second).
- Direct-action commands may still exist where practical (for example jump/execute style flow).
- Parser should accept both spaced and fused forms for operator speed.

### 9.2) Canonical Input Compatibility

- Parser policy: token 1 selects namespace and remains locked for that line (no cross-namespace fallback).
- `X S` and `XS` should map to the same internal command key.
- `X R` and `XR` should map to the same internal command key.
- `M D` and `MD` should map to the same internal command key.
- `I O V` and `IOV` should map to the same internal command key.
- `I C` and `IC` should map to the same internal command key.
- One canonical dispatch representation is preferred to avoid duplicate handlers.
- Aliases are spelling variants only (same meaning); command override behavior is intentionally avoided.

### 9.3) Namespace Plan

Root	Planned Role	Notes
<code>B</code>	Bank / FLASH	Reserved for FLASH-related operations (read/program/erase/verify family).
<code>I</code>	Info root	Carries nested subfamilies such as time and I/O.
<code>I T</code>	Time	Time moves under Info; top-level <code>T</code> is freed.
<code>I C</code>	Calculator	Planned calculator entry under Info; prefer RPN input style.
<code>T</code>	Terminal	Repurposed top-level namespace for terminal-related operations.
<code>I O P</code>	PIA	Top-level <code>P</code> is freed; PIA moves under Info/I/O.
<code>I O V</code>	VIA	Top-level <code>V</code> is freed; VIA moves under Info/I/O.
<code>I O V T</code>	VIA timers	Hardware timers are expected under VIA tree.

J	Jump / Execute	Preferred home for execute flow if top-level execute letter changes.
X	Transfer / XMODEM	At minimum: send and receive support.
S	Search	Text and binary search families.
M	Memory family	Supports compact forms such as MD / MM as aliases.
O	Deferred decision	Candidate: chained execution wrapper; decision postponed.

## 9.4) Search Family Detail

- Planned base forms: S C START END <text> and S B START END <pattern...> .
- Current hit display format is <HIT\_ADDR>{ |\*}<ROW\_BASE>: ... ; HIT\_ADDR is exact match start, and ROW\_BASE = HIT\_ADDR & \$FFF0 .
- Separator marker: \* means the match continues into the next 16-byte row (for example \$B8AF\*\$B8A0 implies continuation at \$B8B0) .
- S C mode: unquoted text stops at first whitespace.
- S C mode: quoted delimiters can include ", ', and `.
- S C mode: delimiter escape by doubling delimiter character.
- S C mode wildcards: ? matches exactly one character, \* matches zero or more characters.
- S C mode literals: ?? matches literal ?, and \*\* matches literal \*.
- S B mode tokens: HH byte, HHHH little-endian word, nibble wildcard (?A/A?/??) , and \* byte wildcard.
- Candidate extensions: Pascal strings and high-bit-set text search modes.

## 9.5) XMODEM Requirement

- Before publish, provide both XMODEM receive and send paths.
- Preferred forms: X R ... and X S ... with fused aliases (XR, XS).

## 9.6) Vector + Safety Direction (Pre-Publish Requirement)

- Vector updates must support dynamic atomic update behavior.
- Handler-name contract direction: every patchable target exports <HANDLER> and <HANDLER>\_NAME (ASCIIZ); retarget operations update target address and name pointer together.
- Critical windows include vector commit and FLASH routines.
- During critical windows, all EDU LEDs should flash to signal that NMI should not be pressed.
- NMI path should be guarded/deferred during critical windows instead of normal debug flow.
- Staged-update plus atomic-commit behavior is the intended implementation pattern.
- NMI retargeting direction: patch inactive slot fully, then commit via single-byte active-slot selector flip (no in-place live NMI hook rewrite).
- Direct !M edits to live vector hook bytes are allowed for bring-up/debug but are intentionally outside the production-safe retarget path.
- Mandate (non-changing requirement): any operation that mutates FLASH state or vector state must assert critical indication/guard behavior, including module/transient load paths; implementation detail may change, requirement does not.

## 9.7) Deferred Item

- O command semantics are intentionally deferred.
- If adopted as an operation chain wrapper, error policy and guard policy must be defined explicitly.

## 9.8) Active TODO (Pressing)

Now

- TODO: wrap WDCMONv2 FLASH routine calls behind bso2 wrapper/trampoline entry points.
- TODO: add a post-link map check that enforces END\_KDATA < \$F000 .

**Soon**

- TODO: get the ACIA port on the EDU board running.

**Before Publish**

- TODO: provide XMODEM receive and send paths before publish (X R/XR, X S/XS).
- TODO: implement staged vector update plus atomic commit flow for runtime retargeting.
- TODO: enforce critical-window behavior for FLASH/vector mutation paths (LED warning plus NMI guard/defer).
- TODO: enforce dangerous B operation policy (! required, explicit confirmation, and fail-closed behavior without mutation).
- TODO: add deterministic status reporting for dangerous operations (status code byte plus OK / ABORTED / VERIFY\_FAIL / FLASH\_FAIL / DENIED ).
- Deferred (not current TODO): text compression/decompression, tokenization/RLE, and TX ring architecture while 32K FLASH headroom is sufficient.

**9.9) Flash / Bank Safety Policy (Critical, Non-Negotiable)**

- B must not execute dangerous operations by default.
- bso2 plans to use WDCMONv2 FLASH routines through wrappers/trampolines.
- Integration intent is behavioral/protocol compatibility via wrapper entry points, not direct source-text copy.
- Any dangerous B operation requires both force-prefix ! and explicit user confirmation.
- Dangerous operations include at minimum erase, program/write, monitor self-update, vector commit, and bank activation/commit transitions.
- If ! is absent, dangerous operations fail closed with no side effects.
- Confirmation must be operation-specific (typed intent token), not an implicit continue.
- During dangerous operations: enter critical guard mode before mutation starts, flash all LEDs, and guard/defer NMI debug flow until critical mode exits.
- On verify/check failure: abort mutation, exit critical mode cleanly, and report explicit status.
- Required output for dangerous operations: status code byte plus textual result (OK, ABORTED, VERIFY\_FAIL, FLASH\_FAIL, DENIED ).

**9.10) Board Self-Update Policy**

- Board self-update is always dangerous and always requires ! plus explicit confirmation.
- Before final commit, display target region, byte count, and integrity value (checksum/hash when available).
- Preferred execution shape: preflight validation, stage payload, erase/program, verify, then commit/activate.
- Avoid in-place blind overwrite as the only strategy; preserve a recovery path.
- Self-update is fully covered by the non-changing mandate: any FLASH/vector mutation path (including module/transient load/activation) must assert critical indication and guard behavior.

**9.11) Host Tooling Direction (Linux GNU C)**

- Critical FLASH workflows are expected to have a Linux GNU C host path.
- Python helpers may exist for convenience, but they are not the required path for critical FLASH operations.
- Preferred host model: raw serial protocol wrappers in C with explicit timeout/error handling and deterministic status reporting.

**10) 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.