Programming the 8-bit Verilog Micro-controller

Assembly Reference & Tool-chain Guide

Open-source FPGA / ASIC Project

July 11, 2025

Contents

1 Assembler Syntax

Source format (Mini-8 assembler v2):

- **Labels**: identifier ending with `:`. Resolved in pass 1.
- **Comments**: semicolon `;` to end of line.
- **Constants**: hex `$AA`, decimal `170`, char `'A'`.
- **Pseudo-ops**: `ORG`, `DB`, `EQU` (defines a constant byte).

2 Memory Map

Address Range	Region	Notes
`$00-7F`	ROM	Program storage (`rom_128x8_sync`)
`$80-DF`	RAM	96 bytes (`rw_96x8_sync`)
`$E0-EF`	—	*Reserved*
`$F0-FF`	I/O	16 memory-mapped ports

3 CPU Architecture Overview

3.1 Block Diagram

The datapath consists of: PC, IR, A/B accumulators, CCR, MAR and a simple ALU.

4 Instruction-set Reference

Unless otherwise noted an instruction takes 1 cycle fetch + (operand bytes × 2) cycles.

4.1 Mnemonic Quick Sheet

Instr	Hex	Bytes	Addr Mode	Flags	Description	Example
LDA	86	2	Immediate	NZVC	A ← imm	`LDA $AA`
LDA	B6	2	Direct	NZVC	A ← M[addr]	`LDA $80`
LDAB	88	2	Immediate	NZVC	B ← imm	`LDAB $01`
LDAB	B7	2	Direct	NZVC	B ← M[addr]	`LDAB $81`
STAA	96	2	Direct	—	M[addr] ← A	`STAA $F0`
STAB	97	2	Direct	—	M[addr] ← B	`STAB $82`
ADD	42	1	Implied	NZVC	A ← A+B	`ADD`
SUB	43	1	Implied	NZVC	A ← A–B	`SUB`
AND	44	1	Implied	NZ	A ← A&B	`AND`
OR	45	1	Implied	NZ	A ← A|B	`OR`
INC	46	1	Implied	NZVC	A ← A+1	`INC`
DEC	48	1	Implied	NZVC	A ← A–1	`DEC`
BRA	20	2	Relative	—	PC ← PC+off	`BRA *`
BNE	26	2	Relative	—	branch if Z=0	`BNE LOOP`
BEQ	27	2	Relative	—	branch if Z=1	`BEQ DONE`

4.2 Addressing Modes

Immediate Next byte is the operand (`#$AA`).

Direct Absolute byte in zero-page ROM/RAM/I/O (`$80`).

Relative Signed 8-bit offset added to the *next* PC ('BRA', 'BNE', 'BEQ'; 'BRA *' = infinite loop).

Implied No operand byte; accumulator(s) are implicit.

5 Programming Examples

5.1 Blink LED (port 0)

```
        LDA  #$01       ; pattern 0000_0001
loop:
        STAA $F0        ; LED on
        LDA  #$00
        STAA $F0        ; LED off
        BRA  loop       ; forever

        BRA *
```

Listing 1: Minimal blink program

What this demo shows

- **Direct I/O write.** `STAA $F0` shows that an 8-bit value written through the MAR reaches the external LED port, confirming the memory-mapped I/O decoder.
- **Immediate vs. accumulator loads.** Alternating `LDA $01` / `LDA $00` demonstrates that the immediate-mode load correctly overwrites the A-register every cycle.
- **Branch to label.** `BRA loop` exercises relative branch offset calculation across negative distances (backwards jump).

5.2 Fibonacci (4 terms) to RAM

```
;---------- seeds ------------------------------------------
        LDA     #$00        ; F0
        STAA    $80
        LDAB    #$01        ; F1
        STAB    $81

        LDA     #$00        ; A = F(n-1)
        LDAB    #$01        ; B = F(n)

; ---------- temporary scratchpad at $FE ----------------------
temp    EQU     $FE         ; holds the *previous* A each step

; ---------- F2 -----------------------------------------------
        STAA    temp        ; save old A
        ADD                 ; A = A + B  = 1
        STAA    $82         ; store F2
        LDAB    temp        ; B = old A  = 0

; ---------- F3 -----------------------------------------------
        STAA    temp
        ADD                 ; 1 + 0 = 1
```

```
22          STAA    $83
23          LDAB    temp
24
25  ; ---------- F4 ------------------------------------------------
26          STAA    temp
27          ADD                 ; 1 + 1 = 2
28          STAA    $84
29
30  ; ---------- finished ------------------------------------------
31  done:   BRA     *           ; park CPU
```

Listing 2: First 4 terms of Fibonacci Sequence

What this demo proves. The Fibonacci demo (written to first 4 terms for brevity but repo contains the first 16) is intentionally written with only the instruction forms supported by the `assembler.py` toolchain, so it becomes a self-contained acceptance test for both the software and the silicon:

- **Direct addressing.** The loop seeds and later stores every term into $80–$8F, exercising the MAR and zero-page interface.
- **Register-to-register ALU path.** Each new term is produced with the implied-operand `ADD` instruction, showing that the ALU can add the A and B accumulators in a single cycle while updating the $NZVC$ flags.
- **`EQU` constants.** The scratch address `temp EQU $FE` is resolved by the assembler's new pass-1 symbol table, demonstrating the pseudo-op pipeline.
- **Relative branching.** A tight `BNE loop`/`BRA *` pattern confirms correct PC-relative offset calculation (including the special case `BRA *` $\Rightarrow$ offset `0xFE`).
- **8-bit overflow behaviour.** Beyond F_{12} the sequence naturally wraps modulo 256; watching the operand and flag traces in the VCD verifies that the ALU's carry and overflow logic matches expectation.

Together these points cover every datapath element (PC, MAR, A, B, CCR) and every newly-added assembler feature (direct loads, implied ops, `EQU`, extended branches), making the demo a quick but thorough "smoke test" for the entire micro-controller tool-chain.

6 Tool-chain Workflow

1. **Edit** `.asm`. Use `EQU` for constants; labels end with `:`.
2. **Assemble** to a binary image:

```
$ python assembler.py assemble asm/[code name].asm -o build/[preffered
    binary name].bin
```

3. **Load** ROM into the test-bench (automatic in `computer_TB_advanced.sv`).
4. **Simulate** with Icarus or ModelSim to produce `waves.vcd`.
5. **(Optional)** Synthesize in Quartus for FPGA deployment.

7 Debugging Tips

- Increase the watchdog in `computer_TB_advanced.sv` for long programs.
- Use `$dumpvars` selectively to keep VCD size manageable.
- Probe flag behaviour with single-byte ops ('INC', 'DEC', 'AND', ...).

A Complete Op-code Table

Mnemonic	Hex	Bytes/Cycles	N	Z	V	C
LDA (#)	86	2	x	x	x	x
LDA (dir)	B6	2	x	x	x	x
LDAB(#)	88	2	x	x	x	x
LDAB(dir)	B7	2	x	x	x	x
STAA	96	2	—	—	—	—
STAB	97	2	—	—	—	—
ADD	42	1	x	x	x	x
SUB	43	1	x	x	x	x
AND	44	1	x	x	—	—
OR	45	1	x	x	—	—
INC	46	1	x	x	x	x
DEC	48	1	x	x	x	x
BRA	20	2	—	—	—	—
BNE	26	2	—	—	—	—
BEQ	27	2	—	—	—	—

x: flag updated
—: unaffected.