Section 4: Assembler & Toolchain
Unified Plan (v1–v5)

Computer Ecosystem Project

January 29, 2026

Contents

1 Version Comparison (v1–v5)

Ver.	Primary Goal	Key Additions	Build Output
v1	Minimum end-to-end loop: `.asm` → `.bin` → run	Two-pass assembler, fixed 8-byte encoding, strict diagnostics, flat binary, minimal loader	Single flat `.bin` (base `0x0000`)
v2	Enable structured programs	Stack + calls: `PUSH8`, `POP8`, `CALL_ABS`, `RET`, `CMP`; accept `SP`, `FP`; (opt) `.org`, `.sym`, disasm	Single flat `.bin` + (opt) `.sym`
v3	OS-shaped layout and control transfers	Vectors at `0x0000..0x00FF`, reset at `0x0100`; `SYSCALL`, `IRET`; directives `.vectors`, `.vector`, `.entry`, `.text`; image layout mode	Memory image `.bin` (vectors + text)
v4	Compiler-friendly milestone	BO addressing + wide ops; data directives + `.align`; sections `.text`, `.data`; minimal object format `.o`; minimal linker `ld` + relocations	`.o` objects + linked `.bin` + `.map`
v5	Protection-aware system build	Indirect control flow (`JMP_R`, `JZ_R`, `JNZ_R`, `CALL_R`); `SETI`, `SETK`, `TRAP`; dual-image builds (kernel and user); linker range checks + lints; trap-aware trace and debug	`kernel.bin` + `user.bin` + maps/metadata

2 Detailed Execution Plans

The following sections are included via `\input`. All files are intended to compile together as a single document.

3 Plan of Execution (v1)

3.1 Purpose and Scope

Goal. Deliver the minimum end-to-end loop:

`.asm` → `assembler` → `flat .bin` → `loader` → `emulator run`

Target: CPU v1 contract (fixed-size 8-byte instructions, strict validation, flat memory).

Out of Scope. Linker/object files, macros, includes, multi-file builds, sections, vectors, syscalls.

3.2 Deliverables

1. **Assembler** (`asm`): two-pass assembler producing a flat binary.
2. **Loader** (`loadbin`): loads `.bin` at base address.
3. **Integration demo** (`demo.asm`) that assembles and runs.
4. **Test suite**: unit + golden + end-to-end.
5. **Documentation**: syntax, mnemonics, errors, CLI.

3.3 CLI Contracts

```
# Assemble:
asm input.asm -o program.bin

# Optional:
# --base 0x0000

# Load into emulator memory:
loadbin program.bin --base 0x0000
```

3.4 Assembly Language Spec (Minimal)

- Comments: `;` to end of line
- Labels: `name:`
- Registers: `R0..R15`
- Integers: decimal and `0x` hex

Supported mnemonics (v1): `MOV_RI,MOV_RR,ADD,SUB,LOAD8_ABS,STORE8_ABS,JMP_ABS, JMP_REL,JZ_ABS,JZ_REL,HALT`.

3.5 Encoding Contract

Each instruction is 8 bytes:

`opcode(8) | rd(8) | ra(8) | rb(8) | imm32(32)`

Little-endian for `imm32`.

3.6 Execution Checklist

1. Define instruction table: mnemonic → opcode + operand schema + zero-field constraints.
2. Implement lexer (tokens: ident, reg, number, punctuation, comments).
3. Implement parser (line → optional label + optional instruction).
4. Pass 1: assign addresses, build symbol table (PC starts at base, +8 per instruction).
5. Pass 2: resolve labels, compute REL offsets, validate ranges, emit bytes.
6. Write `.bin` (no header); implement `loadbin`.
7. Build `examples/v1/demo.asm`; add golden bytes test + emulator integration test.

3.7 Diagnostics Requirements

All errors must include file, line/column, category code, message, and (optional) hint.

3.8 Definition of Done

1. Demo assembles to stable bytes and runs to HALT.
2. Unit tests cover lexer/parser/encoder and key failure modes.
3. Docs describe syntax and CLI.

4 Plan of Execution (v2)

4.1 Purpose and Scope

Goal. Extend v1 to enable structured programs:

- Stack/calls: `PUSH8, POP8, CALL_ABS, RET`
- Comparison: `CMP`
- Assembly register names: `SP`, `FP`

Remain backwards compatible with v1 sources.

Out of Scope. Interrupt vectors, syscalls, MMIO layout (v3); sections and data (v4); user and kernel protection (v5).

4.2 Deliverables

1. Updated assembler with v2 mnemonics + `SP, FP`.
2. v2 example programs: `call_ret.asm`, `stack.asm`.
3. Updated tests (encoding + new diagnostics).
4. Recommended: `.sym` output; minimal disassembler.

4.3 CLI Additions

```
asm input.asm -o program.bin
# recommended:
# --sym program.sym
# --dump
```

4.4 Language Additions

New mnemonics: `CMP, PUSH8, POP8, CALL_ABS, RET`.

New register tokens: `SP`, `FP` (internally encoded using special selectors where ISA permits).

Optional directive: `.org <address>` (may require gap-fill with zeros in flat output).

4.5 Execution Checklist

1. Extend instruction table with v2 mnemonics and constraints (e.g., `RET` fields must be zero).
2. Update lexer/parser to accept `SP, FP`.
3. Implement (optional) `.org` in parser + pass1/pass2 with output gap-fill.
4. Implement encoder support for new instructions and strict field validation.
5. Add example programs and golden tests; run emulator integration (SP changes, CALL and RET returns).
6. (Recommended) Write `.sym`; (Recommended) implement disasm reading 8-byte chunks.

4.6 New Diagnostics

- **E_BAD_SPECIAL_REG**: `SP, FP` used where not allowed.
- **E_RET_NONZERO**: `RET` has nonzero fields.
- **E_ORG_BACKWARDS**: `.org` decreases PC (if forbidden).

4.7 Definition of Done

1. All v1 programs still assemble and run.
2. v2 call/stack examples assemble and run.
3. Tests validate new encodings and errors; optional sym/disasm works if implemented.

5 Plan of Execution (v3)

5.1 Purpose and Scope

Goal. Adopt OS-shaped layout and control transfers:

- Vector table at `0x0000..0x00FF`
- Reset entry at `PC=0x0100`
- New mnemonics: `SYSCALL`, `IRET`
- MMIO conventions (console and timer)

Out of Scope. BO addressing + data sections + linker (v4); protection rules (v5).

5.2 Deliverables

1. Assembler supports `SYSCALL` and `IRET`.
2. Layout directives: `.vectors`, `.vector`, `.entry`, `.text`.
3. Image-mode output that places vectors and code at the proper addresses.
4. Loader defaults appropriate for reset at `0x0100`.
5. Examples: vectors + syscall demo (+ timer IRQ demo if emulator supports).

5.3 Binary Layout Convention

Two modes:

- **Raw mode** (compat): behaves like v2 flat blob.
- **Image mode** (recommended): emits a memory image starting at `0x0000` with:
  - vectors at `0x0000`
  - text at `0x0100`
  - zero-filled gaps

5.4 Directive Semantics

```
.vectors
.vector 0, syscall_handler % 0x00: syscall vector
.vector 1, timer_handler % 0x01: timer interrupt vector

.entry start % reset begins at PC=0x0100
.text
```

Vector table contract (matches CPU v3). The vector table occupies `0x0000..0x00FF`. Each vector entry is a 2-byte little-endian address (`u16`) stored at `0x0000 + 2*id`. Vector `0x00` is the syscall handler; vector `0x01` is the timer interrupt handler. On reset the CPU begins at `PC=0x0100` (no reset vector).

5.5 Execution Checklist

1. Extend instruction table: `SYSCALL` (no operands; `rd=ra=rb=imm32=0`; syscall number in `R0`), `IRET` (all non-opcode fields zero).
2. Parse new directives and track state (inside `.vectors`, current section).
3. Implement image-mode layout engine: reserve `0x0000..0x00FF` for vectors; emit vector entries at `0x0000`; place `.text` at `0x0100`; gap-fill zeros.
4. Pass 1: compute symbol addresses with layout rules.

5. Pass 2: emit bytes at correct offsets; patch vector slots as `u16` little-endian handler addresses at `0x0000 + 2*id` (valid ids `0x00..0x7F`).
6. Update loader/run defaults: load at `0x0000`, start execution at `0x0100` (reset entry).
7. Add demos and integration tests: SYSCALL enters handler, IRET returns.

5.6 New Diagnostics

- **E_BAD_VECTOR_ID**: vector id out of range.
- **E_VECTOR_NO_LABEL**: undefined handler label.
- **E_ENTRY_NOT_0100**: `.entry` label does not resolve to `0x0100` in v3 image mode.
- **E_SYSCALL_NONZERO**: `SYSCALL` has any nonzero non-opcode fields.
- **E_VECTOR_UNALIGNED**: handler address is not 8-byte aligned (instruction alignment).
- **E_OVERLAP_VECTORS**: code overlaps `0x0000..0x00FF` when forbidden.
- **E_IRET_NONZERO**: IRET has nonzero fields.

5.7 Definition of Done

1. v1 and v2 programs still assemble and run.
2. v3 image binaries place vectors/text correctly.
3. SYSCALL and IRET demo works (vector 0 dispatch + return) and tests cover directives + vector patching.

6 Plan of Execution (v4)

6.1 Purpose and Scope

Goal. Make the toolchain compiler-friendly:

- CPU v4 support: BO addressing + wide load and store + bitwise and shift
- Data directives and `.align`
- Minimal `.text, .data` sections
- Minimal object output (`.o`) + minimal linker (`ld`) + relocations

Out of Scope. Protection enforcement and user and kernel split rules (v5).

6.2 Deliverables

1. Assembler supports BO operands and v4 mnemonics.
2. Data directives: `.byte, .word16, .word32, .word64, .ascii, .asciz`, plus `.align`.
3. Sections: `.text` and `.data`.
4. Object output: `asm -c file.asm -o file.o`.
5. Linker: `ld a.o b.o -o program.bin` with relocations + `.map`.

6.3 BO Operand Syntax

Recommended:

```
LOAD32_BO R1, [R3 + 16]
STORE8_BO [R4 + -1], R2
```

6.4 Relocations (Minimal Set)

- `R_ABS16`: write 16-bit absolute (LE)
- `R_ABS32`: write 32-bit absolute (LE)
- `R_REL32`: write 32-bit PC-relative (optional, if needed)

6.5 Execution Checklist

1. Add v4 ISA mnemonics and BO operand parsing; encode correctly; unit tests.
2. Add data directives emission (bytes + LE words + strings) and tests.
3. Add `.align N` padding with validation and tests.
4. Implement minimal sections in binary mode (two buffers: text and data; deterministic placement).
5. Add object output `-c`: section bytes + symbols + relocations; add `.global, .extern`.
6. Build minimal linker: merge sections, resolve globals, assign addresses, apply relocations, emit `.bin` + `.map`.
7. Add multi-file example: `lib.o + main.o` links and runs.

6.6 New Diagnostics

- **E_BAD_BO_ADDR**: malformed `[Reg + Off]`.
- **E_ALIGN_ZERO**: invalid `.align 0`.
- **E_BAD_STRING**: invalid string escape.
- **E_DUP_GLOBAL**: duplicate global symbol at link time.
- **E_UNDEF_GLOBAL**: undefined external symbol at link time.
- **E_RELOC_RANGE**: relocation overflow.

6.7 Definition of Done

1. v1–v3 programs still build and run.
2. v4 BO + wide ops + data directives work with tests.
3. `asm -c` and `ld` link multi-file programs and run in emulator.

7 Plan of Execution (v5)

7.1 Purpose and Scope

Goal. Finish Section 4 roadmap alignment with protection-aware builds:

- Indirect control flow: `JMP_R, JZ_R, JNZ_R, CALL_R`
- Mode/trap ops: `SETI, SETK, TRAP`
- Dual-image build model: **kernel** + **user**
- Linker range enforcement + user MMIO lints
- Trap-aware trace and debug improvements (recommended)

Note. `JNZ_R` is introduced in v5 as the register-indirect complement to `JZ_R`.

7.2 Deliverables

1. Assembler supports v5 mnemonics and strict field validation.
2. Linker supports `-target=kernel` and `-target=user`.
3. Range checks:
   - user code and data must fit `-user-start..-user-end`
   - user output cannot reference MMIO absolute region (lint)
4. Minimal syscall and trap stubs library for userland (optional but strongly recommended).
5. End-to-end demos: kernel boots, transitions to user, user performs syscall or trap and returns.

7.3 Two-Image Build Model

- **Kernel image**:
  - owns vectors at `0x0000`
  - reset entry at `0x0100`
  - provides syscall and trap handlers
- **User image**:
  - linked into user-safe region
  - calls services via `SYSCALL` and/or `TRAP`
  - should not directly use MMIO absolute addresses

7.4 Execution Checklist

1. Add v5 mnemonics to instruction table and encoder (reg-only indirect jumps and calls; strict imm and zero-field rules).
2. Extend linker with `-target=kernel` and `-target=user` and separate layouts.
3. Implement region enforcement and MMIO lints for user builds; add negative tests.
4. Provide minimal user syscall stubs (`libsys`) + kernel handlers; buildable as `.o`.
5. Improve trace and debug: trap cause, EPC, BADADDR, mode transitions; symbolized breakpoints (recommended).
6. Create v5 end-to-end demo: kernel + user images, syscall prints, return path validated.

7.5 New Diagnostics

- **E_BAD_INDIRECT**: malformed indirect jump and call operands.
- **E_IMM_NONZERO**: instruction requires imm=0 but found nonzero.
- **E_USER_MMIO**: user binary contains forbidden MMIO absolute access.

- **E_RANGE_OVERFLOW**: output exceeds allowed region.
- **E_BAD_TARGET**: user symbol resolves into kernel-only region.

7.6 Definition of Done

1. v1–v4 builds remain functional.
2. v5 instructions assemble/link/run.
3. Kernel and user images build with enforced ranges and lints.
4. End-to-end demo shows syscall or trap path and (optional) indirect calls.