

Software CPU – Phase 1 Architecture Specification

1. Overview

This document specifies the architecture of a simple 16-bit Software CPU to be implemented in C/C++. It covers:

- Register set
- Condition flags
- Instruction format
- Addressing modes
- Opcode table
- Memory map
- CPU schematic (logical)
- Fetch-decode-execute microarchitecture

Later phases (emulator, assembler, example programs) will build on this design.

2. Data Path and Word Size

- **Word size:** 16 bits
- **Address size:** 16 bits → 64 KiB address space
- **Endianness:** Little-endian (low byte at lower address)
- **Instruction size:**
 - Base instruction: 16 bits (1 word)
 - Some instructions use an additional 16-bit word for:
 - Immediate constants
 - Full addresses
 - PC-relative offsets

3. Registers

3.1 Programmer-Visible Registers

- **General-Purpose Registers (GPRs)**
 - **R0, R1, R2, R3** ($4 \times 16\text{-bit}$)
 - Used for arithmetic, logic, addresses, and temporaries.
- **Program Counter (PC, 16-bit)**
 - Holds the address of the **next instruction word** to fetch.
- **Stack Pointer (SP, 16-bit)**
 - Points to the **top of the stack** in memory.
 - Stack convention: grows **downward** (pre-decrement on push, post-increment on pop).

- **Flags Register (FLAGS, 8-bit)**

- Only lower 4 bits are used:
 - Bit 0: **Z** – Zero
 - Bit 1: **N** – Negative
 - Bit 2: **C** – Carry
 - Bit 3: **V** – Overflow
 - Bits 4–7: reserved (must be 0)

Total programmer-visible registers:

R0–R3, PC, SP, FLAGS.

3.2 Internal Registers (Microarchitecture)

These registers exist conceptually in the CPU implementation/schematic:

- **IR – Instruction Register (16-bit)**
Holds the current instruction word.
 - **MAR – Memory Address Register (16-bit)**
Holds the address for memory access.
 - **MDR – Memory Data Register (16-bit)**
Buffer for data read from or written to memory.
 - **TMP – Temporary Register (optional)**
Used internally by the ALU/control unit.
-

4. Flags and Condition Codes

Most ALU instructions (**ADD, SUB, AND, OR, XOR, CMP, SHL, SHR**) update the flags as follows:

- **Zero Flag (Z)**
 - Set to 1 if the 16-bit result is **0x0000**.
 - Cleared to 0 otherwise.
- **Negative Flag (N)**
 - Interprets result as signed 16-bit.
 - Set to 1 if bit 15 of the result is 1.
 - Cleared to 0 otherwise.
- **Carry Flag (C)**
 - For **ADD**: set if there is an unsigned carry out (result > **0xFFFF**).
 - For **SUB**: set if no borrow is required (i.e., **a >= b** in unsigned arithmetic).
 - For shifts:
 - **SHL**: **C** gets the bit shifted out of the MSB.
 - **SHR**: **C** gets the bit shifted out of the LSB.

- **Overflow Flag (V)**

- Indicates **signed** overflow.
- For **ADD**: set if adding two operands with the same sign produces a result with a different sign.
- For **SUB**: set if signed subtraction overflows.
- Cleared by purely logical ops (**AND**, **OR**, **XOR**) or when overflow is not meaningful.

Conditional branches (e.g. **JZ**, **JNZ**, **JC**, **JNC**, **JN**) test these flags.

5. Instruction Format

5.1 Base Instruction Word Layout (16 bits)

15	11 10	8 7	5 4	2 1	0
+-----+-----+-----+-----+-----+					
OPCODE MODE RD RS UNUSED					
+-----+-----+-----+-----+-----+					
5 bits	3 bits	3 bits	3 bits	2 bits	

- **OPCODE (bits 15–11, 5 bits)**

Encodes the operation (up to 32 instructions).

- **MODE (bits 10–8, 3 bits)**

Selects addressing mode.

- **RD (bits 7–5, 3 bits)**

Destination register index (0–3 → **R0–R3**; others reserved for future use).

- **RS (bits 4–2, 3 bits)**

Source register index (0–3 → **R0–R3**).

- **UNUSED (bits 1–0, 2 bits)**

Reserved (must be 0 for now).

Depending on the instruction, some fields (e.g. **RD**, **RS**, **MODE**) may be ignored.

5.2 Extended Word (If Required)

For some addressing modes, the instruction is followed by an extra **16-bit word**:

- Immediate constant (**#imm16**)
 - Absolute memory address
 - Signed offset (for PC-relative or base+offset addressing)
-

6. Addressing Modes

The **MODE** field (3 bits) selects the addressing mode:

MODE (binary)	Meaning
000	Register
001	Immediate (extra word)
010	Direct memory (absolute, extra word)
011	Register indirect
100	Register + offset (extra word, signed)
101	PC-relative (extra word, signed)
110-111	Reserved

6.1 Register Mode (MODE = 000)

- Operands are in registers.
- Example: ADD R0, R1
 - RD = R0, RS = R1.

6.2 Immediate Mode (MODE = 001)

- Extra word holds a 16-bit constant.
- Example: ADD R0, #10
 - First word: opcode, mode, rd, rs
 - Second word: 0x000A.

6.3 Direct Memory Mode (MODE = 010)

- Extra word holds a 16-bit memory address.
- Example:
 - LOAD R1, [0x2000]
 - STORE [0x3000], R2

6.4 Register Indirect Mode (MODE = 011)

- Memory address is in a register.
- Example:
 - LOAD R0, [R1] (effective address = contents of R1)
 - STORE [R2], R3

6.5 Register + Offset Mode (MODE = 100)

- Extra word holds a signed 16-bit offset.
- Effective address: EA = R[RS] + sign_extend(offset16).
- Example:
 - LOAD R1, [R2 + 4]

6.6 PC-Relative Mode (MODE = 101)

- Extra word holds a signed 16-bit offset.
- Effective target: EA = PC + sign_extend(offset16) (where PC already points to the next instruction word).

- Used primarily for branching:
 - JZ label
 - JNZ label
-

7. Opcode Table (Draft)

OPCODE is shown in binary (5 bits) and decimal.

OPCODE	Dec	Mnemonic	Description
00000	0	NOP	No operation
00001	1	HALT	Stop execution
00010	2	MOV	Move between registers / imm / memory
00011	3	LOAD	Load from memory into RD
00100	4	STORE	Store from register to memory
00101	5	ADD	RD \leftarrow RD + operand
00110	6	SUB	RD \leftarrow RD - operand
00111	7	AND	RD \leftarrow RD & operand
01000	8	OR	RD \leftarrow RD operand
01001	9	XOR	RD \leftarrow RD ^ operand
01010	10	CMP	Compare (like SUB, result discarded; flags only)
01011	11	SHL	Logical shift left
01100	12	SHR	Logical shift right
01101	13	JMP	Unconditional jump
01110	14	JZ	Jump if Z = 1
01111	15	JNZ	Jump if Z = 0
10000	16	JC	Jump if C = 1
10001	17	JNC	Jump if C = 0
10010	18	JN	Jump if N = 1
10011	19	CALL	Call subroutine (push PC; jump)
10100	20	RET	Return from subroutine (pop PC)
10101	21	PUSH	Push register onto stack
10110	22	POP	Pop value from stack into register
10111	23	IN	Read from IO port into RD
11000	24	OUT	Write RS to IO port
11001-11111			Reserved for future use

For your report, you can extend this into a table with:

- Supported addressing modes per instruction.
 - Whether each instruction updates flags.
-

8. Memory Map

The CPU has a 16-bit address space: **0x0000–0xFFFF** (64 KiB).

Proposed layout:

Address Range	Description
0x0000–0x7FFF	RAM (32 KiB) – data + stack
0x8000–0xEFFF	Program area (code + static data)
0xF000–0xF0FF	Memory-mapped I/O
0xF100–0xFFFF	Reserved (e.g., reset vector, ROM)

8.1 RAM

- General-purpose data storage.
- **Stack:**
 - Grows downward within RAM.
 - Example initial value: **SP = 0x7FFF**.

8.2 Program Area

- Program code and constant data loaded by the emulator.
- Example: entry point at **0x8000**.

8.3 Memory-Mapped I/O (**0xF000–0xF0FF**)

Example assignments:

- **0xF000**: Output data register (write a byte/character here to “print”).
- **0xF001**: Input data register (read a byte from input).
- **0xF010–0xF01F**: Timer registers (for future timer example).

The emulator interprets reads/writes in this address range as I/O operations instead of normal memory.

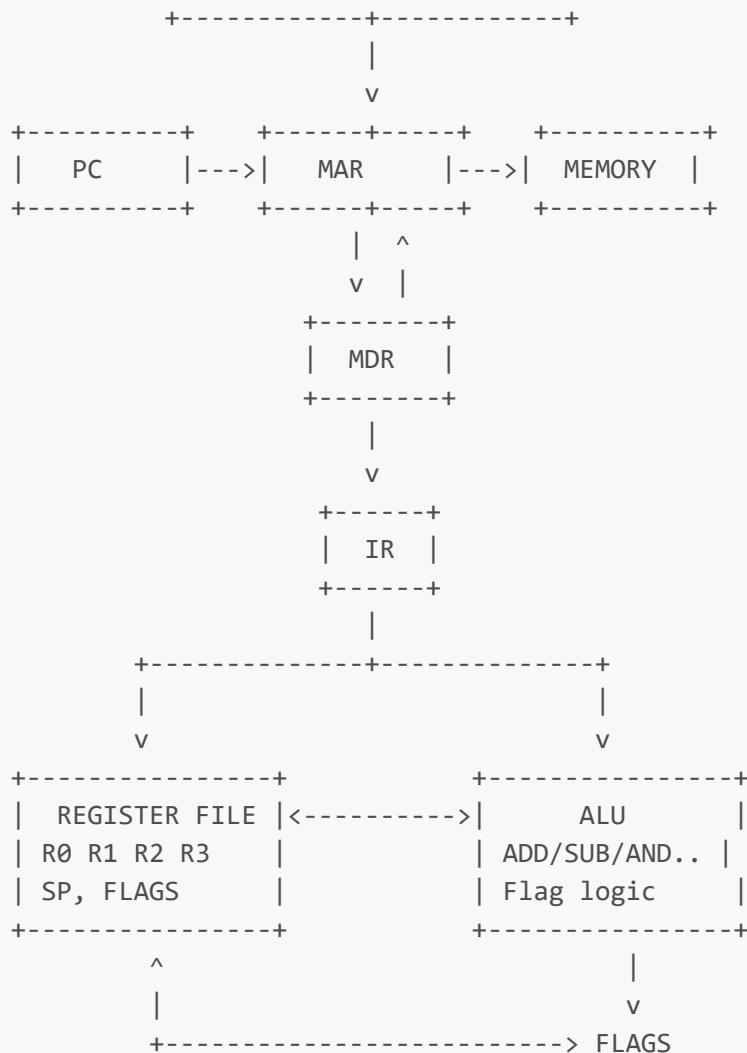
8.4 Reserved / Vectors (**0xF100–0xFFFF**)

- Reserved for future use:
 - Interrupt vectors
 - Reset vector (e.g., initial **PC** at **0xFFFFC–0xFFFFD**)

9. CPU Schematic (Logical Block Diagram)

The following diagram shows the main components and data paths:





- **Control Unit:**
 - Finite-state machine that sequences fetch, decode, and execute.
- **Register File:**
 - Contains GPRs, SP, and provides inputs/outputs to ALU.
- **ALU:**
 - Performs arithmetic/logic, sets Z, N, C, V.
- **Memory Subsystem:**
 - MAR selects address.
 - MDR buffers data.
 - Unified memory for code/data/IO-mapped regions.

10. Fetch–Decode–Execute Microarchitecture

The CPU operates in repeated cycles of **fetch**, **decode**, and **execute**.

10.1 Fetch Phase

1. MAR \leftarrow PC
2. MDR \leftarrow MEM[MAR] (read 16-bit instruction word)
3. IR \leftarrow MDR

4. $\text{PC} \leftarrow \text{PC} + 2$ (advance to next word)

10.2 Decode Phase

- The control unit examines fields in **IR**:
 - $\text{OPCODE} = \text{IR}[15:11]$
 - $\text{MODE} = \text{IR}[10:8]$
 - $\text{RD} = \text{IR}[7:5]$
 - $\text{RS} = \text{IR}[4:2]$
- Based on opcode and mode, it determines:
 - Whether an extra word must be fetched.
 - Source/destination of operands.
 - ALU operation.
 - Whether to update **PC** and flags.

10.3 Execute Phase – Examples

10.3.1 Example: ADD in Register Mode (**MODE = 000**)

1. Read operands:

$A \leftarrow R[\text{RD}], B \leftarrow R[\text{RS}]$

2. $\text{RESULT} \leftarrow A + B$ (via ALU)

3. $R[\text{RD}] \leftarrow \text{RESULT}$

4. Update **Z**, **N**, **C**, **V** based on **RESULT**.

10.3.2 Example: LOAD RD, [addr] in Direct Mode (**MODE = 010**)

1. Fetch extra word for address:

- $\text{MAR} \leftarrow \text{PC}$
- $\text{MDR} \leftarrow \text{MEM}[\text{MAR}]$ (address word)
- $\text{PC} \leftarrow \text{PC} + 2$

2. $\text{MAR} \leftarrow \text{MDR}$ (effective address)

3. $\text{MDR} \leftarrow \text{MEM}[\text{MAR}]$ (load data from memory)

4. $R[\text{RD}] \leftarrow \text{MDR}$

5. Optionally update **Z** and **N** based on loaded value.

10.3.3 Example: JZ with PC-Relative Mode (**MODE = 101**)

1. Fetch offset word:

- $\text{offset} \leftarrow \text{MEM}[\text{PC}]$
- $\text{PC} \leftarrow \text{PC} + 2$

2. If **Z == 1**:

- $\text{PC} \leftarrow \text{PC} + \text{sign_extend}(\text{offset})$

3. Else:

- Do nothing further (branch not taken).

11. Summary

This specification defines the complete Phase 1 architecture for a 16-bit Software CPU, including:

- Register set (**R0–R3, PC, SP, FLAGS**)
- Flag semantics (**Z, N, C, V**)
- Instruction format and bit fields
- Addressing modes
- Core opcode set
- Memory map and I/O regions
- Logical CPU schematic
- Fetch–decode–execute behavior

This design is intended to be straightforward to emulate in C/C++ and to support future phases:

- Emulator implementation
- Assembler
- Example programs (Hello World, Fibonacci, timer demo).