Software-Defined Computer Design Document

Project Name: Software-Defined CPU

Author: Blake Minix

Date: 12-18-2025

1. CPU Overview

   Type: 8-register, 64KB memory, software-defined CPU

   Registers: 8 general-purpose (R0-R7), plus PC, SP, and FP

   Memory: 64KB array

   Stack: LIFO, grows down from top of memory

   Word size: 32-bit registers, 8-bit memory cells (can adjust later)

   Endianness: Little-endian (for multi-byte values)

   I/O: Console output (PRINT)

   Purpose: Minimal CPU capable of running small programs in software

   Diagram (simplified):

   [Registers] <-> [ALU] <-> [Memory/Stack]

   |
   v
   [I/O]

2. Registers

   R0-R7: General-purpose registers

   PC: Program counter (points to next instruction)

   SP: Stack pointer (initially top of memory, stack grows down)

FP: Frame pointer

Z: Zero flag (set when arithmetic result = 0)

3. Memory

   Total size: 64KB

   Instruction fetch reads memory as bytes; multi-byte operands are read sequentially

   Layout:

      0x0000 - 0x7FFF: Code segment

      0x8000 - 0xBFFF: Heap (optional)

      0xC000 - 0xFFFF: Stack

   Access: All memory readable/writable; instructions will enforce boundaries

4. Stack

   Type: LIFO

   Operations: Planned (PUSH, POP, CALL, RET)

   Growth: downwards from 0xFFFF

   Purpose: Temporary storage, function calls

5. Instruction Set (minimal)

   0x01: ADD: Rdest, Rsrc1, Rsrc2: Rdest = Rsrc1 + Rsrc2: ADD R2, R0, R1

   0x02: SUB: Rdest, Rsrc1, Rsrc2: Rdest = Rsrc1 - Rsrc2: SUB R3, R2, R1

   0x03: MOV: Rdest, Immediate: Load Immediate into register: MOV R0, 5

   0x04: PRINT: Rsrc: Print value of register: PRINT R2

   0x05: LOAD: Rdest, Address: Load memory[Address] -> Rdest: LOAD R0, 0x1000

   0x06: STORE: Rsrc, Address: Store Rsrc -> memory[Address]: STORE R1, 0x1000

   0x07: JMP: Address: PC = Address: JMP 0x000A

0x08: JZ: Address: Jump if Z flag set: JZ 0x0010

0x09: JNZ Address: Jump if Z flag not set: JNZ 0x0020

0xFF: HALT: - : Stop execution: HALT

Instructions are byte-encoded in memory.

Example encodings:

ADD R2, R0, R1 -> [opcode][dest][src1][src2]

MOV R0, 5 -> [opcode][dest][imm8]

JMP 0x0010 -> [opcode][addr_lo][addr_hi]

Immediates are 8-bit values (zero-extended to 32-bit registers)

The Z flag is updated by arithmetic and MOV instructions.

Instruction length is determined by opcode and operand count.

6. Program Execution (Fetch-Decode-Execute)

   Loop:

   - Fetch instruction from memory at PC
   - Decode opcode and operands
   - Execute instruction
   - Update PC
   - Repeat until HALT

7. I/O

   PRINT Rsrc: prints value of register to console

   Optional: memory-mapped framebuffer for 2D graphics

   Input (keyboard/mouse) optional

8. Optional/Future Extensions

   Stack operations (PUSH, POP, CALL, RET)

   Status flags: Zero, Negative, Overflow

   Virtual memory or memory protection

   Multitasking/preemptive scheduler

   Graphics framebuffer for games

   JIT compilation for speed

   Debugger/visualizer

Development Path:

1. Implement CPU struct (registers, memory, PC, SP)
2. Implement ADD, SUB, MOV, PRINT, HALT
3. Write fetch-decode-execute loop
4. Run tiny test programs
5. Incrementally add LOAD/STORE, JMP, stack ops
6. Optional: Assembler, I/O, graphics, games