

Status Ahmed 2025-10-29



Agenda

- As discussed last week:
- Ahmed will give a 10 min presentation on the current status of his work.
- We will discuss his presentation with him (5 min).
- Finally, we will define the next steps (teachers only, max. 15 min).

Presentation Ahmed

The image shows a presentation slide titled "Introduction" on the right side. The slide contains a bulleted list of objectives:

- Understand a simplified one-byte instruction set architecture
- Observe fetch decode execute cycle and flag behavior in real time
- Author, assemble, and run small assembly programs using the Editor and Emulator
- Develop debugging strategies using stepping, tracing, and memory inspection Speaker notes

On the left side of the image, there is a blue-tinted photograph of a person sitting at a desk with a laptop, surrounded by various icons representing data, clouds, and connectivity. Below this image, there is a list of agenda items:

- Ahmed is new to bare metal, low-level code
- this slide and the next just repeat the task 😞
- Demo

A large number "2" is visible in the bottom right corner of the slide area.

The screenshot displays two main sections of the CP Emulator interface:

- Main Menu (Top Section):**
 - Header: CP Emulator — Main Menu
 - Address bar: http://127.0.0.1:5500/cpuemulator/index.html
 - Navigation: Back, Forward
 - Section: CP Small Example Processor (Tiny 8-bit teaching CPU — Main Menu)
 - Get started: Open the editor to write assembly, or run the emulator directly. Buttons: Open Emulator, Quick Demo.
 - Resources: Documentation, lab exercises, and presentation for professors. Buttons: Documentation, Exercises.
 - About this project: This demo shows a minimal CPU with 1-byte instructions: LD, ST, ADD, JMP. Use the editor to create programs and the emulator to step/animate execution.
 - Quick links: Program mem: 32 bytes, Data mem: 32 bytes, Registers: r0,r3.
- Runtime (Bottom Section):**
 - Header: CP Emulator — Runtime
 - Step / Run / Animate a tiny CPU
 - Assembler (optional):
 - Text area: LD [00], r0
LD [01], r1...
 - Buttons: Assemble → Program Memory, Clear, Load Example.
 - Text: Import .asm / .hex
 - Text: Drag .asm or .hex files here to load them into the emulator
 - Text: Machine Code (hex): b0 01 A1 12 10 C0
 - Buttons: Load → Program Memory
 - Program Memory:

Addr	Hex	Disasm
00	00	LD [00], r0
01	00	LD [00], r0
02	00	LD [00], r0
03	00	LD [00], r0
04	00	LD [00], r0
05	00	LD [00], r0
06	00	LD [00], r0
07	00	LD [00], r0
08	00	LD [00], r0
09	00	LD [00], r0
0A	00	LD [00], r0
0B	00	LD [00], r0
0C	00	LD [00], r0
0D	00	LD [00], r0
0E	00	LD [00], r0
0F	00	LD [00], r0
10	00	LD [00], r0
 - Registers:
 - Text: r0: 00 | r1: 00 | r2: 00 | r3: 00
 - Text: PC: 00 Flags: Z:0 C:0
 - Buttons: Step, Run, Animate, Reset, Stop.
 - Text: Anim delay (ms): 400
 - Execution Trace: (Empty section)

- looks quite complete??

Proposed Editor Enhancements

Syntax-aware editor: syntax highlighting for mnemonics, registers, hex bytes; auto-indent; line numbers



Live diagnostics: inline parse errors, warnings for out-of-range addresses, and suggested fixes

UX improvements: undo/redo, search/replace, clipboard paste-to-assemble, keyboard shortcuts (Ctrl+S assemble)

- Ahmed wants to implement an editor with syntax completion ... - do we want that??
- Ahmed proposes to add group functions - do we want that??

Conclusion

Project summary: Built a compact browser-based toolchain consisting of an Editor and an Emulator that together let students write assembly, assemble to one-byte machine code, load .asm/.hex files, and observe execution state in real time.

Core learning outcomes achieved: students can assemble and run small programs, follow the fetch-decode-execute cycle, inspect registers and memory, and reason about flags and edge cases such as carry and wrap-around.

Key features delivered: syntax-agnostic assembler, program/data memory preview, step/run/animate controls, execution trace, drag-and-drop file import, notes with persistent save, and safe animation loop.

Discussion with Ahmed

- Stefan is happy!
- some remarks on slide 2 - simulation is not as precise as indicated
- please change memory layout to von Neumann (data and program memory in the same memory) - now it is Harvard (separate memories); maybe add a mode switch in the future
- please extend memory to 256 bytes; if disassembler outputs nonsense for unused memory addresses, it may do so 😊
- Stefan likes demo!
- how to save memory? / how to save assembly? / .hex-file is nice, binary would be sufficient
- in the improved editor
 - no indentation needed
 - line numbers will be useful
 - highlight pc-address and operand address and used registers (source and destination in different colours)
- assembler is not correct!!
 - byte operands behind the opcode are missing
 - LD assembles but disassembly shows ST
- Will there be time to extend the instruction set?? - this is a lower priority, but the code should be written in a way that allows future extensions for different instruction sets
- With notes and program files and ... can we have something like a project that comprises everything to save and reload?
- currently we have basically a local javascript running in the browser; do we need server functionality?
 - Salah is happy, as long Stefan is happy 😊
 - Ahmed has the idea to add server functionality, but so far there is no real need for server functions

next meeting 5.11. 16:00 - no Malek