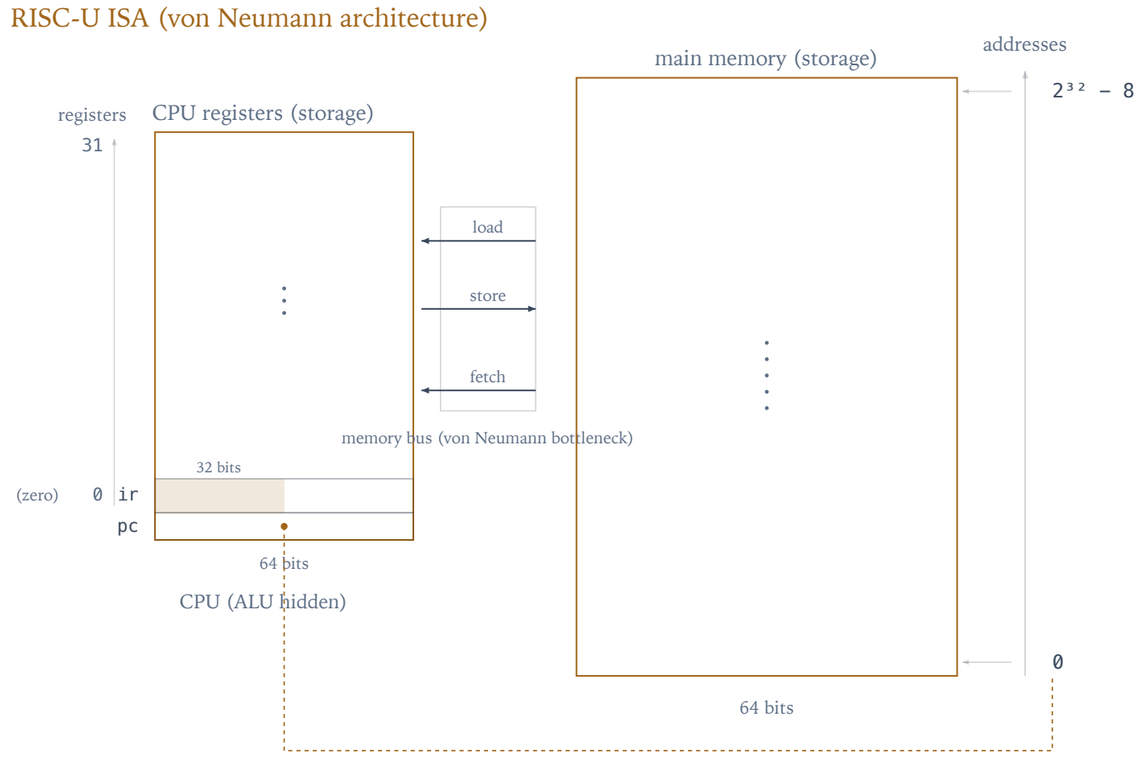
SYSTEMS ENGINEERING · WEEK 2 · STATION II

The Machine, Again

Privilege, exceptions, system calls, machine contexts — and the assembler as a regular language.

THE STATE

Everything a kernel manages is on this figure.



registers, pc, ir, 4 GB of memory

32 registers, a program counter, and 2^{32} bytes: the whole state. Deterministic: the next state is a function of this one and the input. Finite: 2^{35} bits, which the introduction class placed on its axis.

A kernel's job is to give many programs the illusion that each owns this figure. It does that by saving and restoring the registers, and by translating the addresses. Everything else is bookkeeping around those two acts.

And it does that on this same machine, with these same registers. Hold that thought until week 6.

PRIVILEGE AND EXCEPTIONS

Two modes. The machine stops and hands control up whenever a program does what it may not.

Real RISC-V has privilege levels: user mode, where a program runs, and supervisor mode, where the kernel does. Certain instructions and certain memory are supervisor-only. RISC-U has no mode bits; selfie models the boundary in the emulator instead, which is exactly why emulation is the executable specification of the kernel.

An exception is the machine stopping the program and jumping to the kernel's handler: division by zero, an access to memory the program does not own, an illegal instruction, a page fault, a timer. The program's state is intact; the kernel decides what happens next.

THE ONE INSTRUCTION THAT ASKS

`ecall` is an exception the program raises on purpose, with a request in `a7` and arguments in `a0`–`a3`: exit, read, write, open, more memory. It is the whole interface between a program and the world, and the reason isolation is possible at all: everything a program cannot be trusted to do goes through one door.

MACHINE CONTEXTS

A context is the machine, written down.

Selfie keeps a machine context for every machine it emulates or hosts: the registers, the pc, the page table, the program break, the code and data, the exception that stopped it and why, the parent context. Search for the term in `selfie.c`; it is a list of words on the heap with getters and setters, because C* has no structs.

To run a context, mipster loads its registers and pc and starts interpreting. To stop it, it saves them back. That is a context switch, and it is the same operation whether the machine is emulated or virtualized: only who executes the instructions in between differs.

```
// selfie.c, machine contexts
uint64_t* create_context(uint64_t* parent, uint64_t* vct›
void save_context(uint64_t* context);
void restore_context(uint64_t* context);
uint64_t* mipster(uint64_t* to_context, uint64_t timeout)
uint64_t* hypster(uint64_t* to_context, uint64_t timeout)
```

NOTATION, ONCE MORE

RISC-U assembly is a regular language. So its parser is a finite state machine.

```
// selfie -c tiny.c -s tiny.s
ld t0,-16(gp)
addi t1,zero,7
sltu t0,t0,t1
beq t0,zero,6
…
jal zero,-8
```

One instruction per line: a mnemonic, then registers and an immediate in a fixed pattern per format. No nesting, no parentheses to match except the one pair in `imm(rs1)`, no recursion. A single EBNF rule describes it, so a scanner is enough; no pushdown automaton needed.

Assembling is the machine chapter's encoding, reversed from the disassembler: read the line, encode the instruction, emit the word. Selfie already has the encoders; the assembler reuses them.

Which is the assignment: a loop back to the machine via a bit of compiler frontend, so that by week 4 the machine is familiar enough to virtualize.

ASSIGNMENT

`assembler-parser`: option `-a`, and a regular grammar.

1. **Specify.** Design a regular EBNF grammar for RISC-U assembly as selfie prints it with `-s`: mnemonics, registers, immediates, the parentheses of loads and stores, and whitespace.
2. **Model.** A finite state machine for it, drawn before coding. Which characters distinguish the formats?
3. **Implement.** Extend selfie with option `-a` followed by an assembly file: scan and parse it, report syntax errors, and, for now, generate nothing.
4. `./grader/self.py assembler-parser`; look at what it tests. Next week: code generation, and self-assembly.

REUSE

Selfie's scanner has `get_character`, identifier and integer scanning, and the register names; its emulator has the decoders and the disassembler. An assembler is mostly plumbing between things that exist. Take your time designing the grammar; the plumbing is quick afterwards.