CPU Documentation

Implementation

State Machine

Note: Avi said that it was alright to not include schematics since I'd already moved past that part of the process and it didn't make a lot of sense to backtrack and re-create them.

The CPU's main state machine lives in `rv32i_multicycle_core.sv`. Each state is active for exactly one clock cycle, then transitions to another state (it is legal for a state to transition back to itself in some limited cases, such as `S_HALT` and `S_ERROR`). The states are as follows:

Name	Description	Register File	Memory Access	ALU	Other	Notes
`S_FETCH`	Fetch instruction at `PC` from memory	-	Read @ `PC`	`PC + 4`	`PC_old = PC; PC = PC + 4.` Instruction register updates.	
`S_DECODE`	Decode instruction	-	-	-	Instruction decoder enabled, all outputs update.	Register File reading is directly connceted to the `rs1` and `rs2` outputs of the instruction decoder, so this triggers register reads.
`S_EXECUTE`	Execute the (first step of) instruction	Reads `rs1` and `rs2` (if present). Writes to `rd` (for I-Type, R-Type, `jal`/`jalr`, `lui`)	-	Depends on instruction	`PC = alu_result (= rs1/PC + imm)` for `jal`/`jalr`	All instructions start with `S_EXECUTE`, but may involve more than one state.
`S_BRANCH_JUMP`	For taken branches, calculate the target address and execute the jump	-	-	Calculate target address: `PC + imm`	`PC = alu_result (= PC + imm)`	
`S_LOAD`	For load instructions, load the data from memory	Stores data read from memory into `rd`.	Read @ target address	-	-	Target address stored is calculated during `S_EXECUTE`, stored in non-architectural register (`load_store_address`).
`S_STORE`	For store instructions, store the data into memory	-	Store `rs2` @ target address	-	-	Target address stored is calculated during `S_EXECUTE`, stored in non-architectural register (`load_store_address`).
`S_HALT`	Stop execution immediately. No further activity occurs.	-	-	-	-	Never leaves this state. In simulation, immediately exits.
`S_ERROR`	Encountered a fatal error. No further activity occurs.	-	-	-	-	Never leaves this state. Somewhat duplicative/inconsistent with `PANIC` macro (see below).

`PANIC`

The `PANIC` macro is used throughout the CPU when an unrecoverable error has ocurred (ex. an illegal instruction). In simulation, this prints an error message and ends the simulation. The behavior in synthesis is undefined.

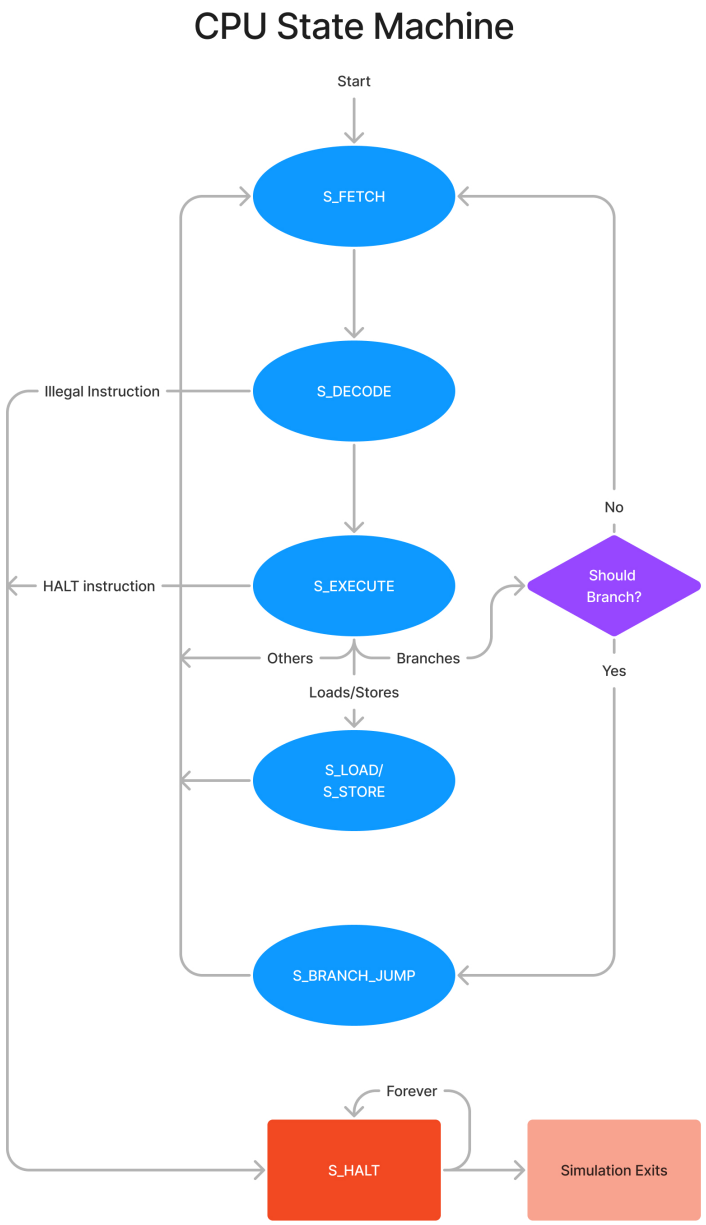

Figure 1: CPU finite state machine diagram

Instructions

This CPU supports the entire `rv32i` instruction set, *except* for `lh[u]`, `lb[u]`, `sh`, and `sb`. It also supports the following non-standard instruction(s):

op	funct3	funct7	Type	Instruction	Description	Operation	Notes
0000000 (0)	000	-	N/A	`halt`	Stop CPU execution immediately	`HALT`	Instruction is all `0`s

Input and Output

In simulation, upon halting, the CPU will print the value of the `a0` register as a return value. See the assembler documentation for how to use this with C.

Additionally, if an `ARGV` option is provided to `make` for any `[waves|test]_rv32i[_c]_<name>` target, `a0` will be initialized with that value as an argument. If no argument is provided, it will default to zero.

```
# plus_one.s

addi a0, a0, 1
```

```
$ make test_rv32i_plus_one ARGV=4
... lots of output ...
Halting! Program Returned:         5
```