SleepyU CPU Manual

Yash Parmar & Nikunj Patel

Professor Hao
CS382-B

11 December 2025

Contents

1 CPU Name and Contributors

Name: SleepyU
Contributors: Yash Parmar & Nikunj Patel

1.1 Contributions

Yash Parmar:

- Designed and organized the stages in the datapath.
- Developed the assembler program with help from Nikunj on opcode encodings.
- Assisted in writing the manual.

Nikunj Patel:

- Defined control signals and opcode encodings with help from Yash on datapath design.
- Developed the demo program for the CPU.
- Assisted in writing the manual.

2 Setup Instructions

1. Run the startup command to assemble the demo program:

```
bash startup.sh
```

1. Press `Simulate` next to `Design`.
2. Press the down tick next to `main` circuit to find the rest.
3. Load `demo_instruction.hex` into Instruction RAM from the `Instruction_Fetching` Circuit.
4. Load `demo_memory.hex` into Data RAM from the `Memory_Access` Circuit.
5. Press `Manual Tick Half Cycle` to half tick into the first cycle.
6. Set initial register values in `Register_File` circuit:
   - X0: 00
   - X1: 00
   - X2: 01
   - X3: 00
7. Run the simulation for 6 clock cycles (Press `Manual Tick Full Cycle` 6 times).
8. The first value in Data RAM from `Memory_Access` circuit should be 43 or 0x43 (or 67 in decimal).

3 Architecture Description

- 4 general purpose registers: X0, X1, X2, X3.
- Supported instructions: BOOST (ADD), FALL (SUB), FETCH (LDR), TUCK (STR).
- PC register increments by 1 after executing each instruction.
- Single-clock-cycle design: all instruction steps (fetch, decode, execute, memory access, write-back) happen in one cycle.

3.1 CPU Components

3.1.1 Instruction Fetching Stage

- Fetch instruction from Instruction RAM using PC.

3.1.2 Instruction Decoding Stage

- Decode opcode and determine registers and control signals:
  - WriteEnable: 1 if BOOST, FALL, FETCH; 0 for TUCK
  - ALUOp: 1 for FALL; 0 for BOOST, FETCH, TUCK
  - MemRead: 1 for FETCH; 0 otherwise
  - MemWrite: 1 for TUCK; 0 otherwise
  - MemToReg: 1 for FETCH; 0 otherwise

  a

3.1.3 Execution Stage

- Perform addition (BOOST, FETCH, TUCK) or subtraction (FALL) as determined by ALUOp.

3.1.4 Memory Access Stage

- Use ALU output as address or computation depending on instruction.
- Load or store data depending on MemRead/MemWrite signals.

3.1.5 Write Back Stgae

- Write either ALU output or memory data to the destination register based on MemToReg.

3.1.6 Register File

- Read source registers and destination register using 2-bit addresses.
- Output 8-bit register values.

4 Naming Conventions

- BOOST: ADD
- FALL: SUB
- FETCH: LDR
- TUCK: STR
- Rd: destination register
- Rs: source register
- Rt: target register
- Rm: operand register 1
- Rn: operand register 2

5 Instruction Format (8-bit)

- `[aa bb cc dd]`:
  - 2-bit opcode (aa): identifies instruction (BOOST, FALL, FETCH, TUCK)
  - 2-bit destination register (bb): X0-X3
  - 2-bit source/operand register (cc): X0-X3
  - 2-bit target/operand register (dd): X0-X3
- Reasoning: 8-bit instruction allows minimal memory usage and easy scalability.

5.1 Instruction Encodings

- BOOST (BOOST Rd, Rm, Rn): opcode = 00
- FALL (FALL Rd, Rm, Rn): opcode = 01
- FETCH (FETCH Rd, [Rs, Rt]): opcode = 10
- TUCK (TUCK Rd, [Rs, Rt]): opcode = 11

Hope you enjoy our SleepyU!