Custom Multi-Cycle 8-bit CPU Data Sheet

Overview

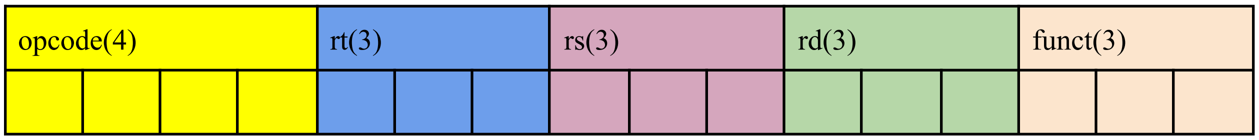
This project implements a simple multi-cycle CPU in Verilog, designed to support a basic RISC-like instruction set. It features a Harvard architecture with separate instruction and data memories and a custom instruction set, an ALU with common operations, a 3-stage fetch/decode/execute control FSM, and a register file with eight 16-bit registers (R0–R7). There is also a debug port that mirrors the contents of R3 for external visibility.

Key Specifications

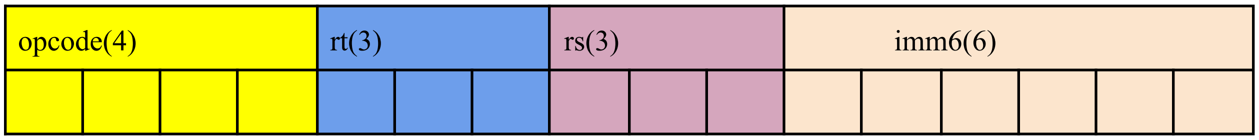
Data Path Width:	16 bits
Address width:	8 bits (256-word instruction & data memory)
Registers:	8 general-purpose registers (R0 hardwired to zero)
Instruction width:	16 bits

Instruction formats:

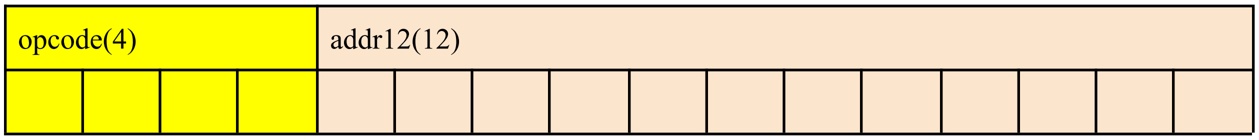
R-type:



I-type:



J-type:



Supported instructions:

- Arithmetic/logical: ADD, SUB, AND, OR, XOR, NOT, ROR, CMP
- Immediate: ADDI, ANDI, ORI, XORI, LDI(load Immediate)
- Memory: LDR (load), STR (store)
- Control: BEQZ (branch if zero), JUMP, HALT

Module Breakdown

1. **PC**: Program counter with halt, branch/jump, and increment control
2. **ins_mem**: Synchronous instruction memory (256 × 16 bits) via
   $readmemh("program.hex", ...)
3. **control_unit**: FSM with FETCH, DECODE, EXECUTE states
4. **reg_file**: 8 × 16-bit register file, synchronous write, asynchronous read
5. **alu_b_mux**: Selects between register operand or immediate
6. **ALU**: Performs arithmetic/logical operations; sets zero_flag
7. **Data_mem**: 256 × 16-bit data memory, synchronous write, asynchronous read
8. **writeback_mux**: Selects ALU result or memory data for register write

FSM Timing

State	Cycle	Action
FETCH	1	inc_PC, fetch ins
DECODE	2	Decode, prepare operands/addresses
EXECUTE	3	ALU/memory/register write

Instruction Encoding Summary

Opcode	Instruction Type	Description

0XXX	I-Type	Immediate arithmetic/logical
1000	R-Type	Register ALU (use funct)
1001	I-Type	LDR; Load from data memory
1010	I-Type	STR
1011	J-Type	JUMP; Unconditional jump
1100	I-Type	BEQZ; Branch if zero on register
1111	—	HALT; Stop CPU

ALU_op	Mnemonic	Operands	Description
000	ADD	{Rt,} Rs, Rd	Add, $Rt \leftarrow Rs + Rd$
001	SUB	{Rt,} Rs, Rd	Substract, $Rt \leftarrow Rs - Rd$
010	AND	{Rt,} Rs, Rd	And, $Rt \leftarrow Rs \& Rd$
011	OR	{Rt,} Rs, Rd	Substract, $Rt \leftarrow Rs \mid Rd$
100	XOR	{Rt,} Rs, Rd	Substract, $Rt \leftarrow Rs \wedge Rd$
101	NOT	{Rt,} Rs	Substract, $Rt \leftarrow {\sim}Rs$
110	ROR	{Rt,} Rs, Rd	Rotate Right, $Rt \leftarrow ROR(Rs, Rd)$
111	CMP	{Rt,} Rs, Rd	Compare, $Rt \leftarrow Rs < Rd$?

Execution Flow:

- Each instruction completes in three clock cycles: fetch, decode, and execute.
- Non-pipelined design ensures simplicity and clarity.

Implementation Details

- **Clock & Reset**: Global synchronous clock; active‑high reset clears PC, registers, FSM
- **Halt**: Latches when HALT decoded (opcode 1111), freezes PC

- **Debug Port**: dbg_reg_out output mirrors R3 contents for easy probe
- **Testbench**: Custom TBs instantiate cpu_datapath, toggle reset, load program.hex, verify register values and memory contents

Sample Program.hex

Instr_mem																
0201:	ADDI				R1			R0			1					
	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1
0401:	ADDI				R2			R0			1					
	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1
8650:	ADD				R3			R1			R2			0		
	1	0	0	0	0	1	1	0	0	1	0	1	0	0	0	0
8650:	HALT															
	1	1	1	1	0	1	1	0	0	0	0	1	0	0	0	0

Contact & Links