LC-2K Finite State Machine Simulator

13349047 Computer Science 赖少凡

Email:Laishao_yuan163.com Phone:18819461573（Short:61573）

1 Basic design

Several blocks of the simulator:

- fetch block
  - fetch: PC=PC+1 and load the instrReg
  - fetch_delay: wait for the clock
- branch: acts as the controler
- add
  - add: load regA to ALU Operand
  - add_calc: load regB to bus and calculate the RegA+RegB
  - add_done: store the result into regDest
- nand, nand_calc, nand_done: the same as add block
- lw
  - lw: load the regA to ALU Operand
  - lw_addr: calculate the sum of regA and offset
  - lw_read: ready to read the Memory
  - lw_read_delay: wait for the clock and store it into regB
- sw, sw_addr, sw_write, sw_write, sw_write_delay: the same as lw block
- beq
  - beq: load the regA to ALU Operand
  - beq_calc: calculate the difference between regA and regB

  - beq_judge: jump according to (regA == regB)
  - beq_addr: calculate the address
  - beq_pc: set the PC
- jalr
  - jalr: regB = PC
  - jalr_a: PC = regA
- halt: return
- noop: goto fetch

2 Optimize

To simplify the project, I use some macros:

- `define __STATE__(type) type: printState(&state, #type);`
- `define _READMEM() memoryAccess(&state, 1)`
- `define _WRITEMEM() memoryAccess(&state, 0)`
- `#define _DISPATCH(opcode, label) if (((state.instrReg >> 22) & 0x7) == opcode) goto label`
- `define _REG_A state.reg[(state.instrReg >> 19) & 0x7]`
- `#define _REG_B state.reg[(state.instrReg >> 16) & 0x7]`
- `#define _REG_DEST state.reg[state.instrReg & 0x7]`
- `#define _OFFSET convertNum(state.instrReg & 0x0000ffff)`

Those optimization is all following the 10 rules given by the documentation.

3 Result

The simulator is working as well as the simluator in the lab 02.