PCS-3225 Sistemas Digitais II — 14 de junho de 2024 - v1

Processador RISC-V

EP03 — Bruno Albertini, Edson Gomi e Ricardo Lera

O objetivo deste EP é implementar um processador RISC-V *on-chip* (sem barramentos) usando uma versão modificada do conjunto de instruções RV32I descrito na Tabela 1.

Este conjunto contém 32 instruções, subdivididas em 5 *Instruction Classes* (ICs):

- Classe A: Operações aritméticas e lógicas (ADD, SUB)
- Classe B: Operações de *load* e *store* (LW, SW)
- Classe C: Operações de *branch* (BEQ, BNE)
- Classe D: Operações incondicionais (JAL, LUI)
- Classe E: Operações de sistema (EBREAK)

Para atender a este conjunto de instruções vamos usar uma ALU de 8 operações e dois bits extras de saída sinalizando "*Zero*" e "*Negativo*". Também utilizaremos um banco de registradores, uma ROM como memória de instruções e uma RAM como memória de dados. Precisamos então projetar a **unidade de controle** e **fluxo de dados** para este processador. No edisciplinas foi disponibilizado o *template* `riscv.v`, contendo três módulos: `control_unit, datapath e riscv`. É necessário:

- Associar os valores de cada sinal de controle:
  - `1 bit:  RegWrite, ALUsrc`
  - `2 bits:  PCsrc, MemWrite`
  - `3 bits:  ALUctl, MemtoReg`
- Descrever os multiplexadores do fluxo de dados para:
  - `PC`
  - `rf_wdata`
  - `aluB`
  - `dm_data_in`

Envie para o juiz o arquivo `riscv.v` contendo **somente** os módulos `control_unit, datapath e riscv`. Os módulos `registerfile, rom e ram` também foram disponibilizados.

Recursos:

- RISC-V Instruction Set Manual: `https://riscv.org/wp-content/uploads/2017/05/riscv-spec-v2.2.pdf`
- RISC-V Encoder/Decoder: `https://luplab.gitlab.io/rvcodecjs/`

Tabela 1: RV32I *"SoC-Mod"* Instruction Set

Code	FMT	Instruction Title	Description (in Verilog)	IC
ADD	R	Add	R[rd] = R[rs1] + R[rs2]	A
ADDI	I	Add Immediate	R[rd] = R[rs1] + imm	
SUB	R	Subtract	R[rd] = R[rs1] - R[rs2]	
AND	R	AND	R[rd] = R[rs1] & R[rs2]	
ANDI	I	AND Immediate	R[rd] = R[rs1] & imm	
OR	R	OR	R[rd] = R[rs1] | R[rs2]	
ORI	I	OR Immediate	R[rd] = R[rs1] | imm	
XOR	R	XOR	R[rd] = R[rs1] ^ R[rs2]	
XORI	I	XOR Immediate	R[rd] = R[rs1] ^ imm	
SLL	R	Shift Left	R[rd] = R[rs1] << R[rs2]	
SLLI	I	Shift Left Immediate	R[rd] = R[rs1] << imm	
SRL	R	Shift Right	R[rd] = R[rs1] >> R[rs2]	
SRLI	I	Shift Right Immediate	R[rd] = R[rs1] >> imm	
SRA	R	Shift Right Arithmetic	R[rd] = R[rs1] >>> R[rs2]	
SRAI	I	Shift Right Arithmetic Imm	R[rd] = R[rs1] >>> imm	
SLT	R	Set if Less Than	R[rd] = (R[rs1] < R[rs2]) ? 1 : 0	
SLTI	I	Set if Less Than Immediate	R[rd] = (R[rs1] < imm) ? 1 : 0	
LW	I	Load Word (32b)	R[rd] = M[R[rs1]+imm]	B
LH	I	Load Halfword (16b)	R[rd] = M[R[rs1]+imm][15:0]	
LB	I	Load Byte (8b)	R[rd] = M[R[rs1]+imm][7:0]	
SW	S	Store Word (32b)	M[R[rs1]+imm] = R[rs2]	
SH	S	Store Halfword (16b)	M[R[rs1]+imm] = R[rs2][15:0]	
SB	S	Store Byte (8b)	M[R[rs1]+imm] = R[rs2][7:0]	
BEQ	SB	Branch if Equal	if (R[rs1] == R[rs2]) PC += imm	C
BNE	SB	Branch if Not Equal	if (R[rs1] != R[rs2]) PC += imm	
BGE	SB	Branch if Greater or Equal	if (R[rs1] >= R[rs2]) PC += imm	
BLT	SB	Branch if Less Than	if (R[rs1] < R[rs2]) PC += imm	
LUI	U	Load Upper Immediate	R[rd] = imm, 12'b0	D
AUIPC	U	Add Upper Immediate to PC	R[rd] = PC + imm, 12'b0	
JAL	UJ	Jump & Link	R[rd] = PC+4; PC += imm	
JALR	UJ	Jump & Link Register	R[rd] = PC+4; PC += R[rs1] + imm	
EBREAK	I	Environment Break	brk = 1	E