ICP-1 — RISC-V Project

LOVE BÁRÁNY (LOVE.BARANY@EIT.LTH.SE)

Build Your Own RISC-V®

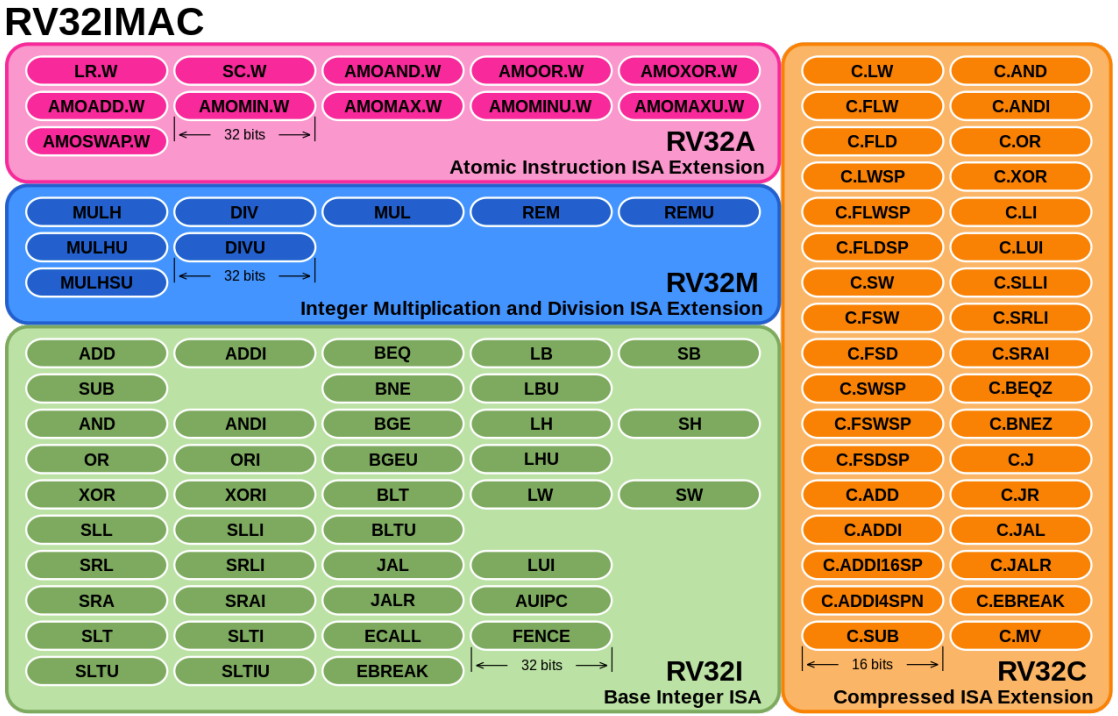
- **RISC** = **R**educed **I**nstruction **S**et **C**omputer
- RISC-V: Open instruction architecture
- Implementation on FPGA only
- Your goal:
  - 32-bit RISC-V with a subset of instructions
  - 5-stage pipeline with control unit
  - Programs sent over a serial link from a computer to an FPGA
    - UART over USB, for more information on UART see e.g. https://www.analog.com/en/resources/analog-dialogue/articles/uart-a-hardware-communication-protocol.html

LUND UNIVERSITY

RV32I	P
RV32E	V
RV64E	*Zbkb
RV64I	*Zbkc
RV128I	*Zbkx
Extension	*Zk
Zifencei	*Zks
Zicsr	*Zvbb
Zicntr	*Zvbc
Zihintntl	*Zvkg
Zihintpause	*Zvkned
Zimop	*Zvknhb
Zicond	*Zvksed
M	*Zvksh
Zmmul	*Zvkt
A	
Zawrs	
Zacas	
RVWMO	
Ztso	
CMO	
F	
D	
Q	
Zfh	
Zfhmin	
Zfa	
Zfinx	
Zdinx	
Zhinx	
Zhinxmin	
C	
*Zce	
B	

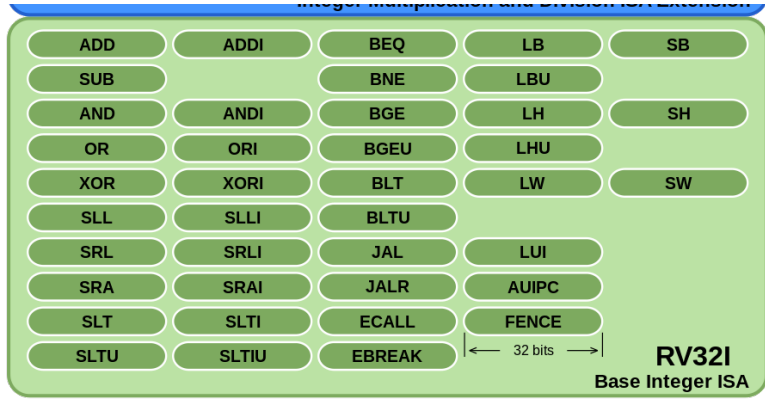
Modular design

- What is actually needed?
  - RV32I can be used as a simplified GPP, even with software support!
- Pick and choose different instructions from different sets depending on your need
- Different extensions complete the base in different ways, for example:
  - RV32C, compressed instructions that are smaller in size
  - RV32F, floating point instructions
  - RV32M, multiplication and division



LUND UNIVERSITY

What is expected



- For all grades: Implement & Verify on FPGA
- **Grade 3:**
  - Most of RV32I (basic integer)
  - Implement hazard detection, stalling, and forwarding
  - Integrate serial UART controller for transfer of programs from computer
  - Branch predictor: local, 2-bit saturation counters, branch history table/branch prediction buffer. See *Computer Architecture – A Quantitative Approach* section C.2 (6th ed) by Hennessy & Patterson

LUND UNIVERSITY

What is expected

- For all grades: Implement & Verify on FPGA
- **Grade 4:**
  - Instructions from RV32C (compressed instructions)
    - Compressed instructions require some extra decoding
  - More advanced branch predictor: 2-level predictor
    - E.g.: gshare, see *Computer Architecture – A Quantitative Approach* section 3.3 (6th ed) by Hennessy & Patterson

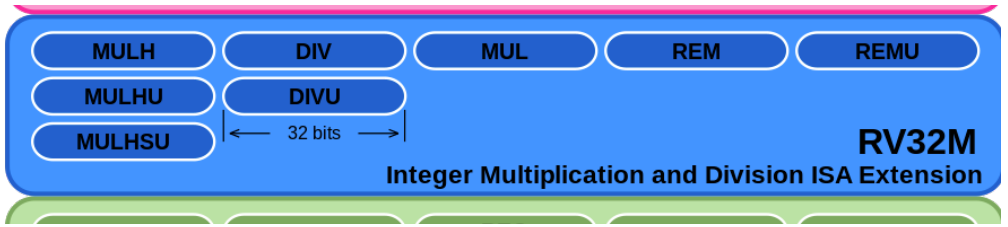
LUND UNIVERSITY

What is expected

- For all grades: Implement & Verify on FPGA
- **Grade 5:**
  - Some of RV32M (multiplication, division etc) and RV32F (floating point operations)
  - Instructions in both M and F require support for multicycle instructions, stalling the pipeline is enough (i.e. no speculation)
  - Floating point numbers are represented differently, and require some care to work with

RV32F / D Floating-Point Extensions

Inst	Name	FMT	Opcode	funct3	funct5	Description (C)
flw	Flt Load Word	*				rd = M[rs1 + imm]
fsw	Flt Store Word	*				M[rs1 + imm] = rs2
fmadd.s	Flt Fused Mul-Add	*				rd = rs1 * rs2 + rs3
fmsub.s	Flt Fused Mul-Sub	*				rd = rs1 * rs2 - rs3
fnmadd.s	Flt Neg Fused Mul-Add	*				rd = -rs1 * rs2 + rs3
fnmsub.s	Flt Neg Fused Mul-Sub	*				rd = -rs1 * rs2 - rs3
fadd.s	Flt Add	*				rd = rs1 + rs2
fsub.s	Flt Sub	*				rd = rs1 - rs2
fmul.s	Flt Mul	*				rd = rs1 * rs2
fdiv.s	Flt Div	*				rd = rs1 / rs2
fsqrt.s	Flt Square Root	*				rd = sqrt(rs1)
fsgnj.s	Flt Sign Injection	*				rd = abs(rs1) * sgn(rs2)
fsgnjn.s	Flt Sign Neg Injection	*				rd = abs(rs1) * -sgn(rs2)
fsgnjx.s	Flt Sign Xor Injection	*				rd = rs1 * sgn(rs2)
fmin.s	Flt Minimum	*				rd = min(rs1, rs2)
fmax.s	Flt Maximum	*				rd = max(rs1, rs2)
fcvt.s.w	Flt Conv from Sign Int	*				rd = (float) rs1
fcvt.s.wu	Flt Conv from Uns Int	*				rd = (float) rs1
fcvt.w.s	Flt Convert to Int	*				rd = (int32_t) rs1
fcvt.wu.s	Flt Convert to Int	*				rd = (uint32_t) rs1
fmv.x.w	Move Float to Int	*				rd = *((int*) &rs1)
fmv.w.x	Move Int to Float	*				rd = *((float*) &rs1)
feq.s	Float Equality	*				rd = (rs1 == rs2) ? 1 : 0
flt.s	Float Less Than	*				rd = (rs1 < rs2) ? 1 : 0
fle.s	Float Less / Equal	*				rd = (rs1 <= rs2) ? 1 : 0
fclass.s	Float Classify	*				rd = 0..9



LUND UNIVERSITY

What is expected

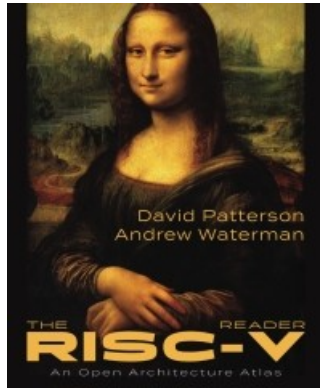
Alternative Grade 5:

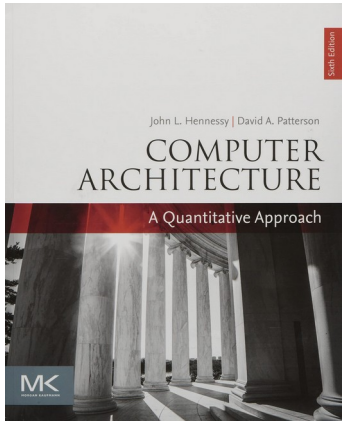
- **PLEASE NOTE: If you choose to do this alternative grade 5, you should expect very little (if any!) supervision available to you for the grade 5 level, compared to if you do the ”normal” (computer architecture related) grade 5. This is due to the limited number of TAs available!**
- Implement and verify for ASIC, components from grade 3+4 (Grade 3+4 are still FPGA only and same for all groups!)
  - Synthesis & PNR
  - Power analysis
  - Testbench with serial interface support

LUND UNIVERSITY

What you're given/Recommended materials to complete project

- A basic 5-stage pipeline in SystemVerilog and VHDL to start with
  - https://github.com/masoud-ata/riscv_sv
  - https://github.com/masoud-ata/PH-RISC-V
  - SV version includes UART controller, can easily be used in VHDL as well
- A simulator and assembler
  - https://github.com/masoud-ata/Masimulator
  - https://github.com/masoud-ata/AssembleRisc
- Various relevant literature





LUND UNIVERSITY

LUND
UNIVERSITY