RISC V

CpE 6110 Term Paper Presentation

Stuart Miller

MISSOURI S&T

RISC-V®

→ Background

Instructions

Memory Model

Pipelining

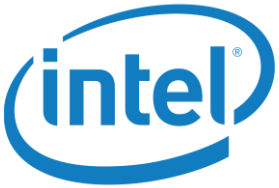
Chip Design

Performance

Conclusions

Background Instruction Set Architectures

- Intel - x86
  - Current generation Windows computers
  - Current generation Apple computers
- ARM Holdings – ARM
  - Most current Android phones
  - Current generation IOS devices
  - Windows 10 IoT
  - 37% of the global market share of all processors
- IBM - PowerPC
  - No widespread consumer use since 2006 Macbook



MISSOURI S&T

Background Open Source ISAs

- OpenRISC
  - Started in the year 2000
  - No commercial usage
  - Most academic implementations are on FPGAs
- SPARC V8
  - Developed by Sun Microsystems in 1987
  - Aimed at server-scale devices
  - No favor for mobile, low-power devices

MISSOURI S&T

Background RISC V

- History
  - Academic endeavor in 2010 at UC-Berkeley
  - Open source and license free
  - User level specification frozen in 2014
  - "Becoming the standard ISA for all computing devices"
- RISC V Foundation
  - "…a non-profit consortium chartered to standardize, protect, and promote the free and open RISC-V instruction set architecture together with its hardware and software ecosystem for use in all computing devices."
  - Promote RISC V for research, education, and commercial use

MISSOURI S&T

Background RISC V

- Tenets
  - Software -- Base instruction set is frozen so programs will function on RISC V cores forever
  - Software – Promote addition and development of instruction set through expansion sets
  - Hardware -- Developers are free to optimize the architecture to their own use case
  - Hardware – Open hardware spec can be inspected and proven for trusted design, ensure functional safety
  - Hardware – No reliance on a single component supplier
  - Licensing – Open source, no fees to use, ever

MISSOURI S&T

Background RISC V

ISA	Base+Ext	Compact Code	Quad FP	Address 32-bit	Address 64-bit	Address 128-bit	Software GCC	Software LLVM	Software Linux	Software QEMU
SPARC V8			✓	✓			✓	✓	✓	✓
OpenRISC				✓	✓		✓	✓	✓	✓
RISC-V	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

MISSOURI S&T

Background Why Open Source?

Field	Standard	Free, open implementation	Proprietary implementation
Networking	Ethernet, TCP/IP	Many	Many
Operating system	Posix	Linux, FreeBSD	Microsoft Windows
Compilers	C	GCC, LLVM	Intel icc, ARMcc
Databases	SQL	MySQL, PostgreSQL	Oracle 12C, Microsoft DB2
Graphics	OpenGL	Mesa3D	Microsoft DirectX
Architecture	None	None	x86, ARM

MISSOURI S&T

Background Why Open Source?

- No major open source hardware initiatives
  - Cost of complexity
  - Produce a product that works in a wide variety of situations or applications
  - Tasked with maintenance (as much as 82% of the project cost)
- Instruction Set Architecture challenges
  - Support a great many hardware platforms
  - Support differing features

MISSOURI S&T

Background Why Open Source?

- RISC V
  - Offsets much of the cost of complexity to academia
  - Academic world already has an established system of receiving corporate or governmental grant funding
  - RISC V Foundation has grant backing from Google, Nvidia, Samsung, Qualcomm, Western Digital, and more
  - Reliance on community developers to report problems and propose resolutions

MISSOURI S&T

Background

→ Instructions

Memory Model

Pipelining

Chip Design

Performance

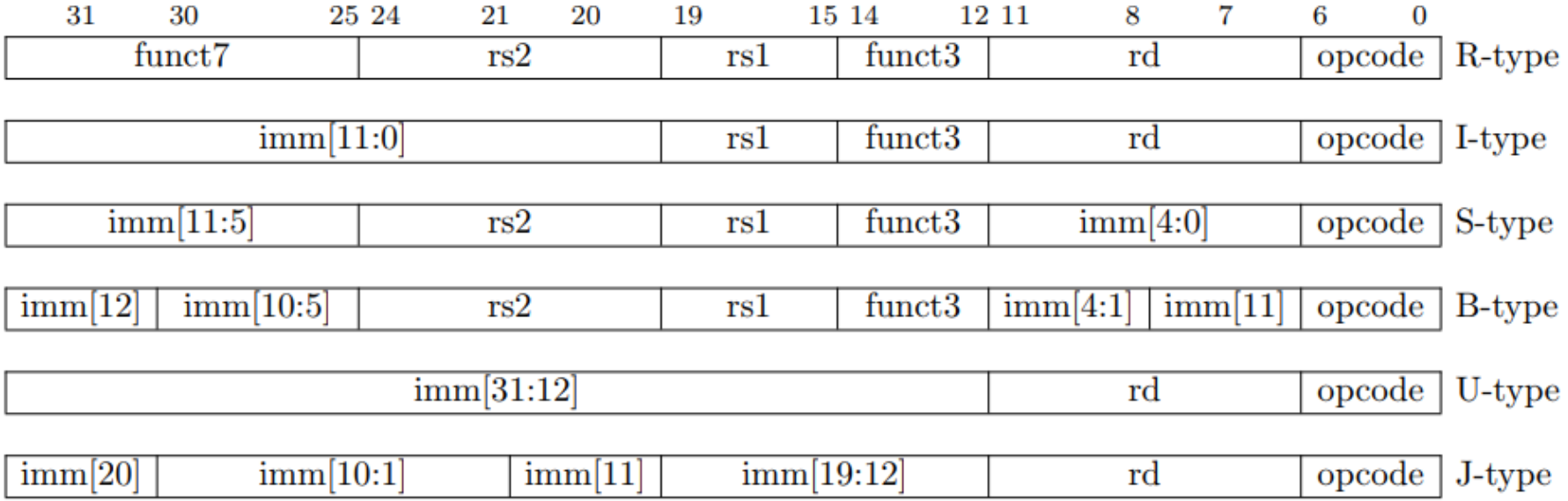
Conclusions

Instructions Base Instructions

- Vaguely similar to MIPS
- Supports both 32-bit and 64-bit instructions
  - Extension set for 128-bit instructions
  - Extension set for compressed 16-bit instructions
- Instruction types
  - Register-to-register (R-Type)
  - Register-immediate (I-type)
  - Store (S-type)
  - Branch (B-type)
  - Extended immediate (U-type)
  - Jump (J-type)

MISSOURI S&T

Instructions Base Instructions

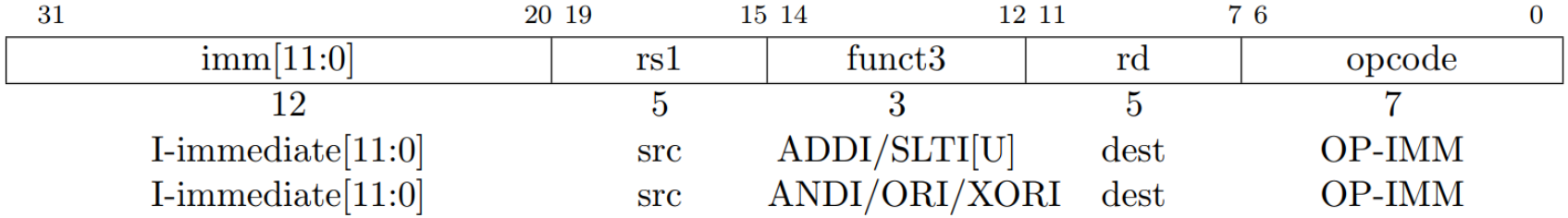


Instructions Base Instructions

31 25	24 20	19 15	14 12	11 7	6 0
funct7	rs2	rs1	funct3	rd	opcode
7	5	5	3	5	7
0000000	src2	src1	ADD/SLT/SLTU	dest	OP
0000000	src2	src1	AND/OR/XOR	dest	OP
0000000	src2	src1	SLL/SRL	dest	OP
0100000	src2	src1	SUB/SRA	dest	OP

- R-Type Instructions
  - "Register to Register"
  - Standard math operations

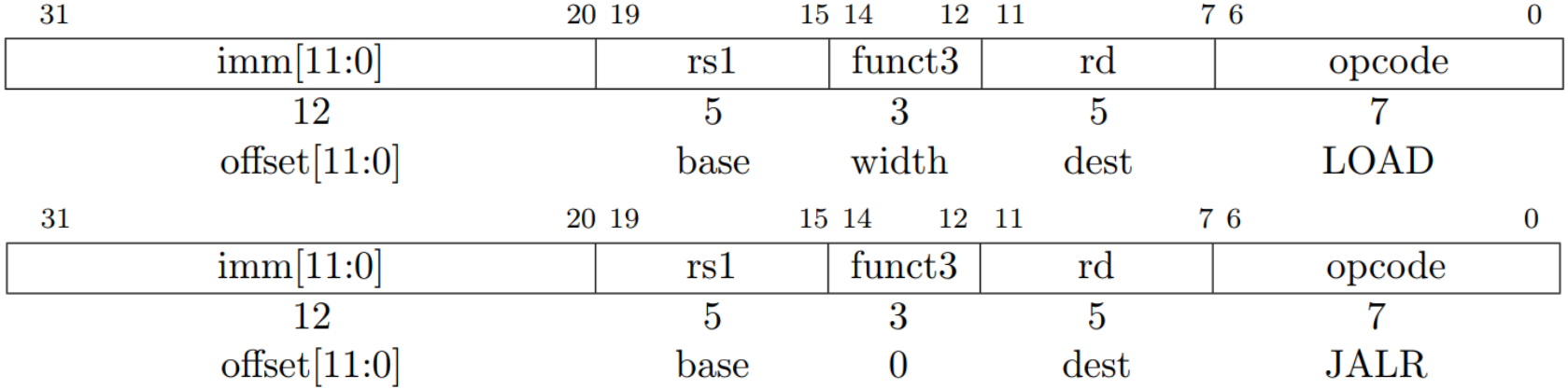
Instructions Base Instructions



31 — 20	19 — 15	14 — 12	11 — 7	6 — 0
imm[11:0]	rs1	funct3	rd	opcode
12	5	3	5	7
I-immediate[11:0]	src	ADDI/SLTI[U]	dest	OP-IMM
I-immediate[11:0]	src	ANDI/ORI/XORI	dest	OP-IMM

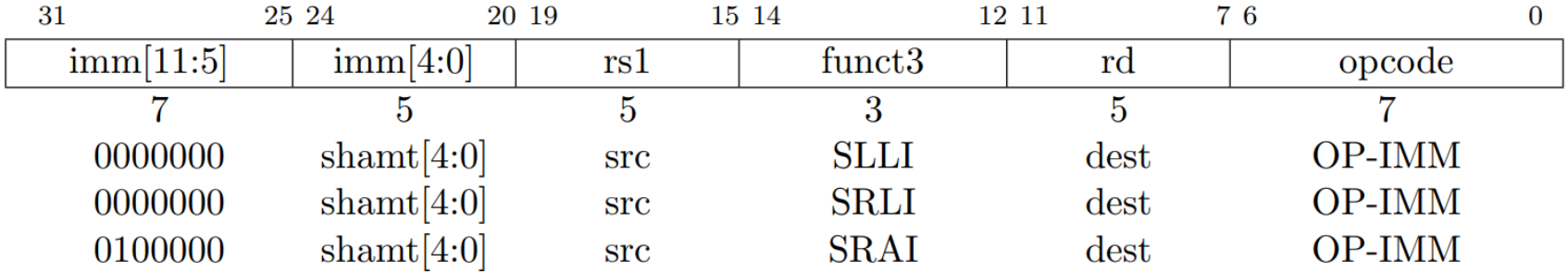
- I-Type Instructions
  - "Register Immediate"
  - Standard math operations using constants

Instructions Base Instructions



- I-Type Instructions (special)
  - "Load"
  - "Jump and Link Register" *(rd = pc = rs1 + imm + 4)*

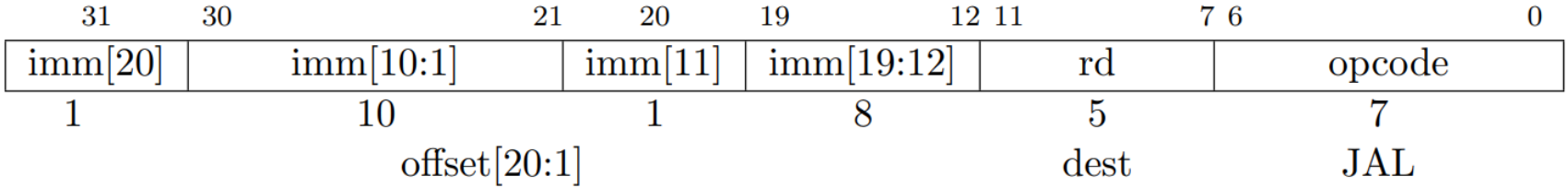
Instructions Base Instructions



31 25	24 20	19 15	14 12	11 7	6 0
imm[11:5]	imm[4:0]	rs1	funct3	rd	opcode
7	5	5	3	5	7
0000000	shamt[4:0]	src	SLLI	dest	OP-IMM
0000000	shamt[4:0]	src	SRLI	dest	OP-IMM
0100000	shamt[4:0]	src	SRAI	dest	OP-IMM

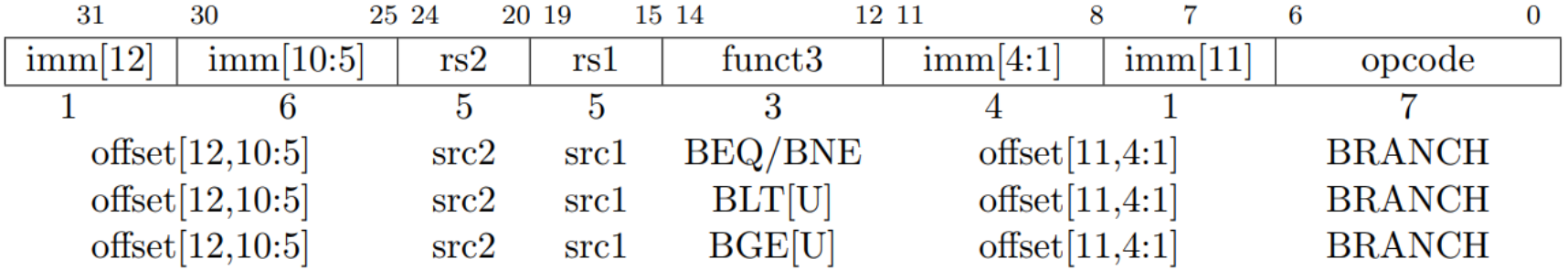
- I-Type Instructions (special)
  - "Shift"
  - Bit shift operations (left/right, logical/arithmetic)

Instructions Base Instructions



- J-Type Instructions (special)
  - "Jump and Link"
  - Plain unconditional jumps

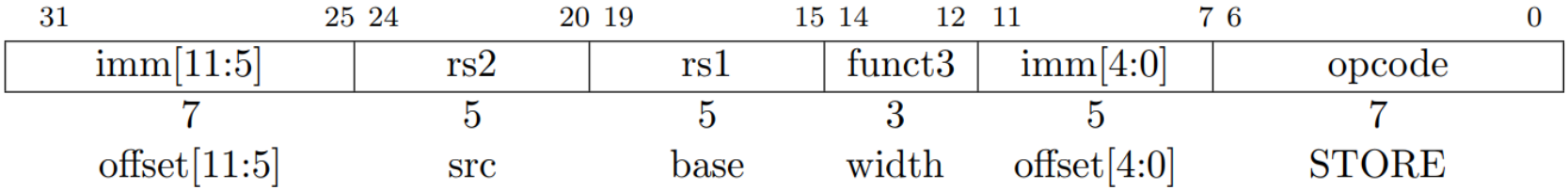
Instructions Base Instructions



31	30 25	24 20	19 15	14 12	11 8	7	6 0
imm[12]	imm[10:5]	rs2	rs1	funct3	imm[4:1]	imm[11]	opcode
1	6	5	5	3	4	1	7
offset[12,10:5]		src2	src1	BEQ/BNE	offset[11,4:1]		BRANCH
offset[12,10:5]		src2	src1	BLT[U]	offset[11,4:1]		BRANCH
offset[12,10:5]		src2	src1	BGE[U]	offset[11,4:1]		BRANCH

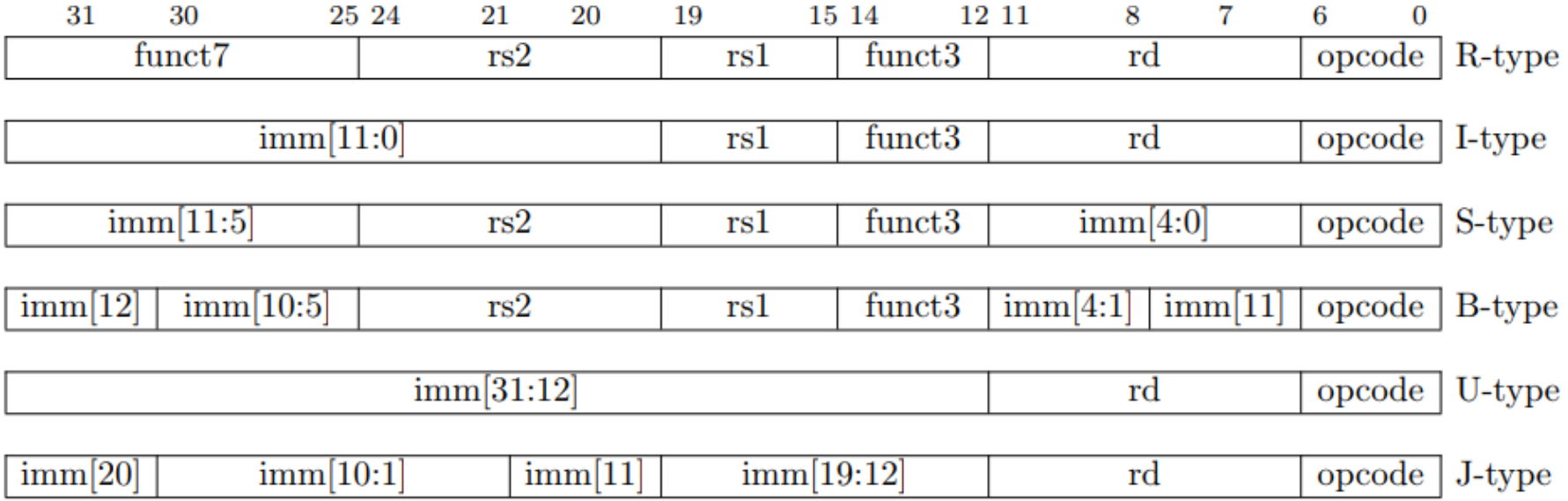
- B-Type Instructions
  - "Conditional Branch"
  - No zero bit, assumed to be 1 (branch target addresses must be multiples of 2)

Instructions Base Instructions

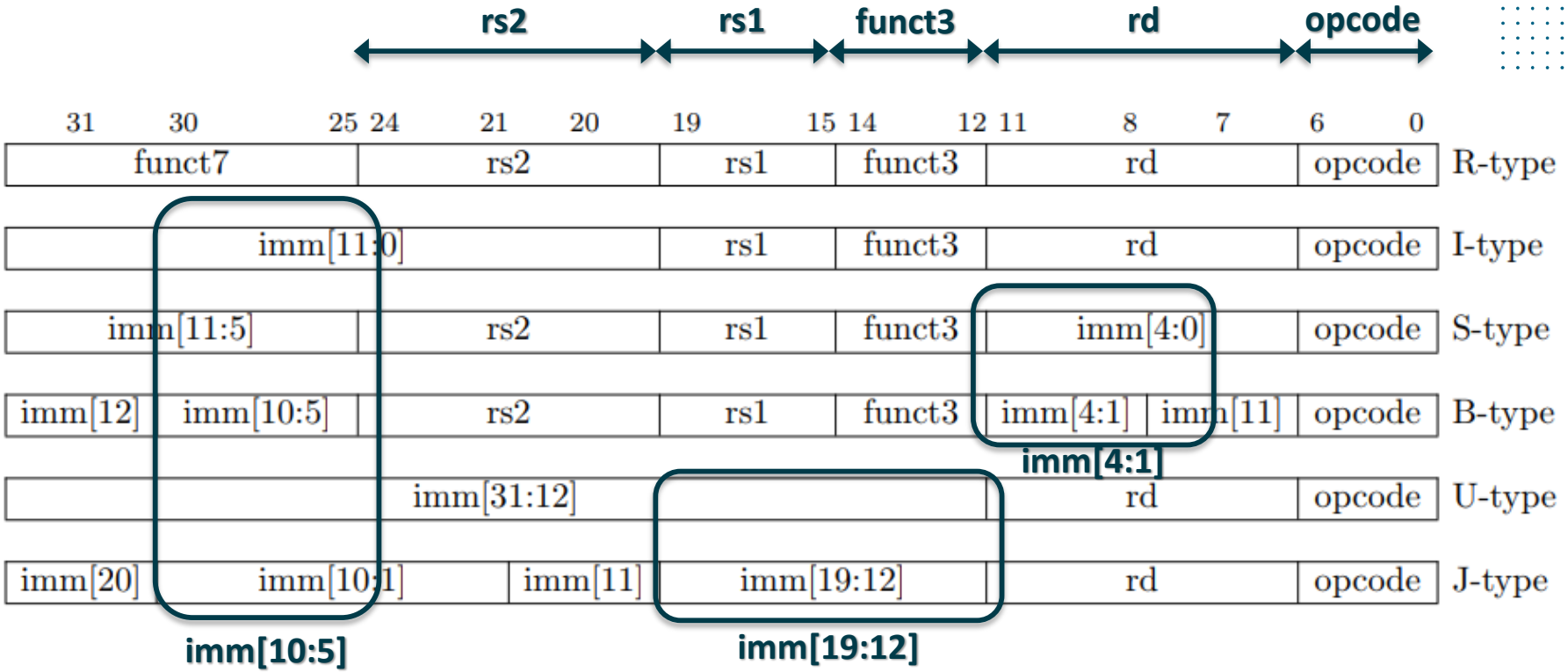


- S-Type Instructions
  - "Store"

Instructions Base Instructions



Instructions Base Instructions



Instructions Compressed Instructions

- "C" Standard Extension for Compressed Instructions, Version 2.0
  - Extension Set "C", currently in draft
  - RISC V Compressed or "RVC"
  - Emphasis on embedded
  - 16-bit encodings for certain instructions (subset of Base Inst. Set)
  - Must combine with 32-bit (or higher) insts, cannot stand alone
  - Typically 50%–60% of insts in program can use compressed variants
    - Results in a 25%–30% code-size reduction

MISSOURI S&T

Instructions Compressed Instructions

Format	Meaning	15 14 13	12	11 10	9 8 7	6 5	4 3 2	1 0
CR	Register	funct4		rd/rs1		rs2		op
CI	Immediate	funct3	imm	rd/rs1		imm		op
CSS	Stack-relative Store	funct3	imm			rs2		op
CIW	Wide Immediate	funct3	imm				rd′	op
CL	Load	funct3	imm		rs1′	imm	rd′	op
CS	Store	funct3	imm		rs1′	imm	rs2′	op
CB	Branch	funct3	offset		rs1′	offset		op
CJ	Jump	funct3	jump target					op

Instructions Compressed Instructions

- 2-bit opcodes
  - All 2-bit instruction opcodes have their two least significant bits set to "11", saving bit patterns "00", "01", and "10" for 16-bit instructions
- 3-bit register addresses
  - Restricts available register range
  - Registers 8-15 for the base integer set
  - Other register subsets for extension sets' registers

RVC Register Number	000	001	010	011	100	101	110	111
Integer Register Number	x8	x9	x10	x11	x12	x13	x14	x15
Integer Register ABI Name	s0	s1	a0	a1	a2	a3	a4	a5
Floating-Point Register Number	f8	f9	f10	f11	f12	f13	f14	f15
Floating-Point Register ABI Name	fs0	fs1	fa0	fa1	fa2	fa3	fa4	fa5

MISSOURI S&T

Instructions Compressed Instructions

Format	Meaning	15 14 13	12	11 10	9 8 7	6 5	4 3 2	1 0
CR	Register	funct4		rd/rs1		rs2		op
CI	Immediate	funct3	imm	rd/rs1		imm		op
CSS	Stack-relative Store	funct3	imm			rs2		op
CIW	Wide Immediate	funct3	imm				rd′	op
CL	Load	funct3	imm		rs1′	imm	rd′	op
CS	Store	funct3	imm		rs1′	imm	rs2′	op
CB	Branch	funct3	offset		rs1′	offset		op
CJ	Jump	funct3	jump target					op

Instructions Compressed Instructions

- Benchmarks - SPEC CPU2006 suite
  - 23% smaller compiled code size than base RISC V
    - x86 code only 15% smaller than base RISC-V – RISC vs CISC?
  - Fetch 25%-30% fewer instruction cache bits
    - Reducing instruction cache misses by 20%-25%
    - Effectively doubled cache size

MISSOURI S&T

Instructions Vector Instructions

- "V" Standard Extension for Vector Operations, Version 0.2
  - Extension Set "V", currently a proposal
  - May support 16-bit, 32-bit, 64-bit, and 128-bit integer, fixed-point, and floating point, plus 8-bit integer and fixed-point
  - 32 vector data registers (v0–v31)
  - 8 vector predicate registers (vp0-vp7)

MISSOURI S&T

Instructions Vector Instructions

- Based on The Hwacha Project
  - "A new vector architecture for future computer systems that are constrained in their power and energy consumption"
  - Pulls vector fetch operations into a separate specialized (faster) processing loop
  - Leaves main loop free to continue

THE HWACHA PROJECT.

MISSOURI S&T

Instructions Embedded Instructions

- "Embedded" Instructions
  - "RV32E" classification
  - Exactly same as normal integer instructions "RV32I"
  - But only allows 16 registers (x0–x15)
    - Removal saves around 25% core area
    - Corresponding core power reduction
  - Exception caused if register (x16-x31) is called

MISSOURI S&T

Instructions Instruction Extensions

Short Name	Type	Purpose	Status	Version
RV32I	Base Integer Instruction Set	Base 32-bit instructions	Frozen	2.0
RV32E	Base Integer Instruction Set	Reduced subset of base 32-bit instructions, for embedded systems	Frozen	1.9
RV64I	Base Integer Instruction Set	Base instructions encoded for 64-bit	Frozen	2.0
RV128I	Base Integer Instruction Set	Base instructions encoded for 128-bit	Frozen	1.7

Instructions Instruction Extensions

Short Name	Type	Purpose	Status	Version
M	Standard Extension	Integer Multiplication and Division	Approved	2.0
A	Standard Extension	Atomic Instructions	Under Revision	2.0
F	Standard Extension	Single-Precision Floating-Point	Approved	2.0
Q	Standard Extension	Quad-Precision Floating-Point	Approved	2.0
L	Standard Extension	Decimal Floating-Point	Placeholder	0.0
C	Standard Extension	Compressed Instructions	Proposal	2.0
B	Standard Extension	Bit Manipulation	Placeholder	0.0
J	Standard Extension	Dynamically Translated Languages	Placeholder	0.0
T	Standard Extension	Transactional Memory	Placeholder	0.0
P	Standard Extension	Packed-SIMD Instructions	Placeholder	0.1
V	Standard Extension	Vector Operations	Proposal	0.2
N	Standard Extension	User-Level Interrupts	Proposal	1.1

Instructions Custom Extensions

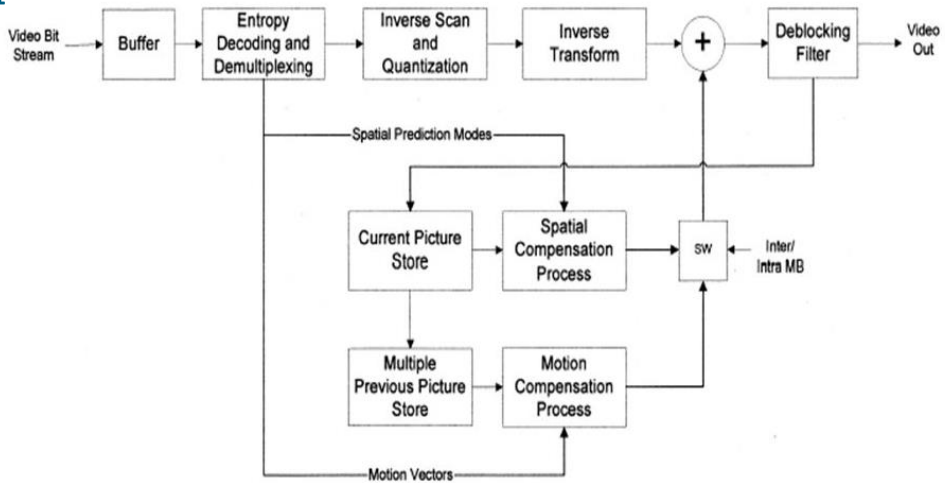
- Allows for user extensibility
- Saves opcodes for custom instructions
- Opcodes are bits [6:0] of instructions
  - Bit patterns [1:0] are reserved for compressed, "11" otherwise
- Operations denoted "custom" are available

inst[4:2] / inst[6:5]	000	001	010	011	100	101	110	111 ($> 32b$)
00	LOAD	LOAD-FP	*custom-0*	MISC-MEM	OP-IMM	AUIPC	OP-IMM-32	$48b$
01	STORE	STORE-FP	*custom-1*	AMO	OP	LUI	OP-32	$64b$
10	MADD	MSUB	NMSUB	NMADD	OP-FP	*reserved*	*custom-2/rv128*	$48b$
11	BRANCH	JALR	*reserved*	JAL	SYSTEM	*reserved*	*custom-3/rv128*	$\geq 80b$

MISSOURI S&T

Instructions Custom Extensions

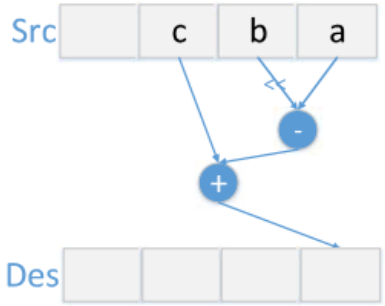
- An example -- H.264 Decoding
  - An extremely popular video encoding/decoding standard
  - Up to 50% lower bitrate at cost of increased algorithmic complexity
  - High demand for embedded processors in purpose-built video devices
  - “Deblocking” operation is computationally expensive



MISSOURI S&T

Instructions Custom Extensions

- Deblocking filter algorithm made heavy use of “clip”, “abs”, and “HAS” functions
  - New assembly instructions for these
  - $Clip(src, minmax) = \begin{cases} minmax & if\ src > minmax \\ -minmax & if\ src < minmax \\ src & otherwise \end{cases}$
  - $Abs(src) = |src|$
  - $HAS(src) = src(a) - 2 * src(b) + src(c)$



FUNCTION LEVEL PROFILING OF THE DEBLOCKING FILTER.

ID	Symbol	Sample	Percentage
1	Main	270002	43.6
2	Hevc_loop,_filter	240920	38.9
3	Clip	72556	13.7
4	Abs	35120	5.7

MISSOURI S&T

Instructions Custom Extensions

- Overall 6.9% speedup
- Minor increase in chip area

Instruction	Hardware Cost	Area Cost (#PLA)
ABS	1 Comp+ 1 Adder	1089
HAS	2 Adder	2112
CLIP	2 Comp + 1 Adder	1122
Total	3 Comp + 4 Adder	4323

ID	Instruction	Speed Up %
1	Original	0
2	ABS	2.1
3	CLIP	4.1
4	HAS	1.0
5	Total	6.9

MISSOURI S&T

Instructions Custom Extensions

- Part of H.264 algorithm requires comparing two 4x4 range of surrounding pixels
  - Added a special 128-bit wide register file with 2 registers
  - Able to store results of row of eight pixels at a time
  - Overall 11% speedup

Instruction		Number of Instruction	Speed Up %
Original		69857562	0
Method 2	Load	18808	-
	Filter	18808	-
	Store	18808	-
	Total	61726266	11.6

MISSOURI S&T

Background

Instructions

→ Memory Model

Pipelining

Chip Design

Performance

Conclusions

Memory Model

- RISC V acknowledges the effects of the memory bottleneck
  - Inherent support for multiple hardware threads
  - Referred to as “harts”
  - Each hart has its own program counter and register set
  - Each hart executes its memory operations sequentially
  - To maintain memory consistency between multiple harts accessing memory, RISC V supports the release consistency model.
  - RISC V uses “Release Consistency”

MISSOURI S&T

Memory Model Weak Consistency

- Weak consistency
  - A more simplistic memory model for MIMD processors
  - Involves regularly synchronizing memory at the end of critical sections
  - Within critical sections, memory accesses are nonblocking
  - Program must be guaranteed not to conflict inside of critical sections

Conditions for Weak Consistency

1. Before an ordinary load or store access is allowed to perform with respect to any other processor, all previous synchronization accesses must be performed, and…
2. Before a synchronization access is allowed to perform with respect to any other processor, all previous ordinary load and store accesses must be performed, and…
3. Synchronization accesses are sequentially consistent with respect to one another.

MISSOURI S&T

Memory Model Release Consistency

- Release consistency
  - Builds on weak consistency
  - All memory synchronizations actively set an acquire and release flag on the section of memory they are operation on
  - Synchronizations are processor dependent based on flag setting
  - Also allows for special access
    - Non-synchronizing special request directly to its target memory location

Conditions for Release Consistency

1. Before an ordinary load or store access is allowed to perform with respect to any other processor, all previous acquire accesses must be performed, and
2. Before a release access is allowed to perform with respect to any other processor, all previous ordinary load and store accesses must be performed, and
3. Special accesses are processor consistent with respect to one another.

MISSOURI S&T

Background
Instructions
Memory Model
→ Pipelining
Chip Design
Performance
Conclusions

Pipelining

- RISC V is merely an instruction set architecture
  - No concrete hardware design required
  - Must support instructions and memory access methods, no more
  - Pipeline is open to hardware designer's interpretation
- 5-stage MIPS pipeline is most common
- 3-stage ARM-style pipelines also somewhat common
- Hard to find public designs
  - RISC-V is open-source, most implementations choose to use closed source

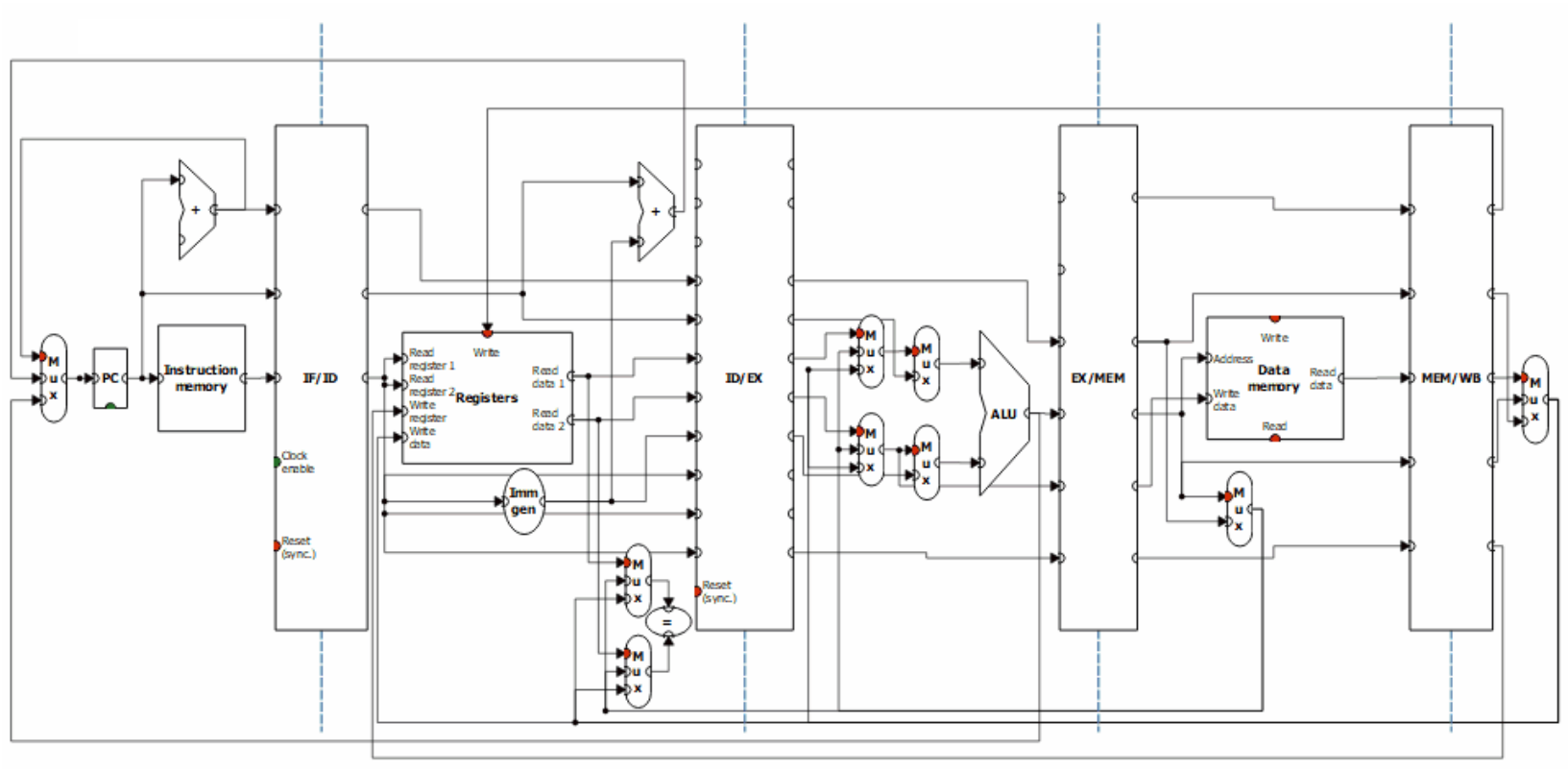
MISSOURI S&T

Pipelining 5-Stage Example

- Ripes
  - A graphical 5-stage processor pipeline simulator
  - Built for simulation, prototyping, and education
  - Can simulate assembly code instructions, good for prototyping new instructions
  - 5-Stage pipeline, nearly indistinguishable from MIPS
  - https://github.com/mortbopet/Ripes

MISSOURI S&T

Pipelining 5-Stage Example

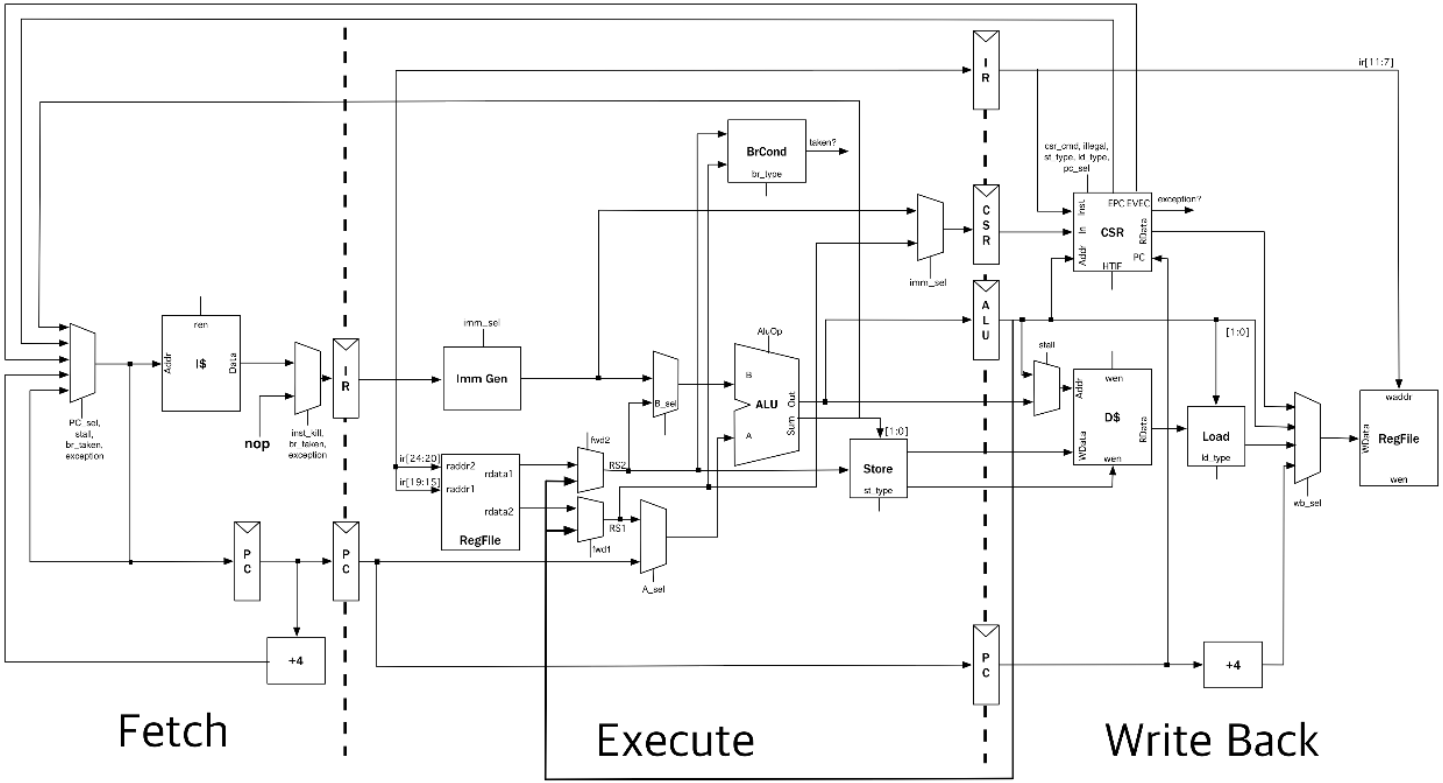


Pipelining 3-Stage Example

- RISCV-Mini
  - A predecessor to UC-Berkeley's Rocket Chip Generator
  - 3-stage pipeline, more akin to that of ARM chips
  - Written in Chisel, a Scala-based hardware language
  - https://github.com/ucb-bar/riscv-mini

MISSOURI S&T

Pipelining 3-Stage Example



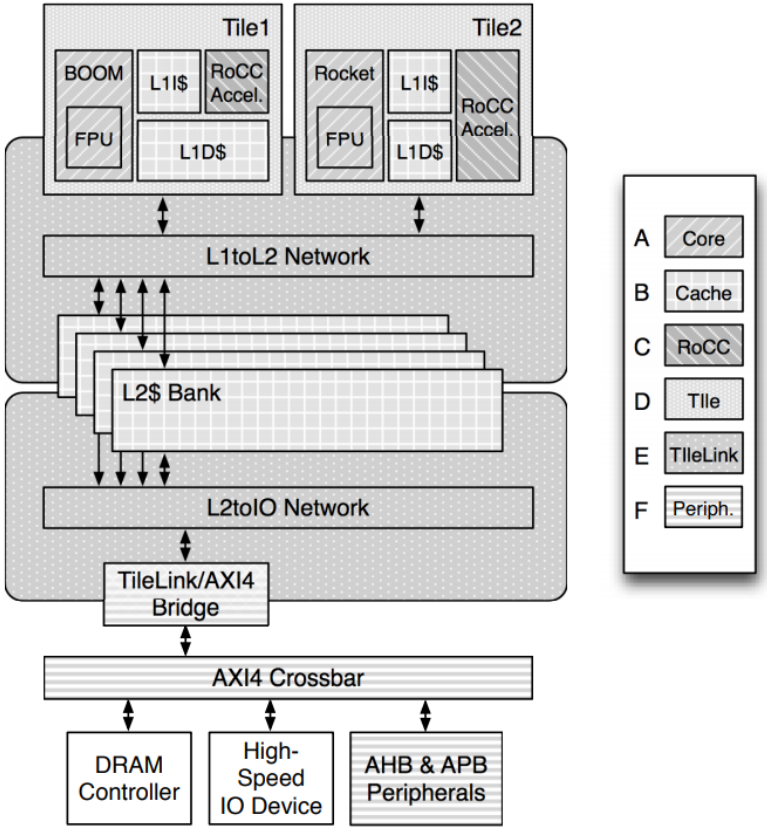
Background
Instructions
Memory Model
Pipelining
→ Chip Design
Performance
Conclusions

Chip Design Rocket Chip Generator Example

- "Made-to-order" silicon
  - Can add or remove ISA extension sets at will
  - Can choose bitness
  - Can "mix and match" ISA features based on application
- Traditionally programmers at mercy of manufacturer options
  - Options are often limited and expensive
- UC-Berkeley's Rocket Chip Generator
  - Custom design chip
  - Also written in Scala (riscv-mini was a precursor to this)

MISSOURI S&T

Chip Design Rocket Chip Generator Example



Chip Design Rocket Chip Generator Example

- Customizes 6 different subsystems separately
  - Processor core, processor cache, coprocessor(s), tile, tile link, and peripherals
- Core design options
  - Rocket core - 5-stage in-order scalar core
  - Z-scale core - 3-stage, single-issue in-order pipeline
  - Berkeley Out-of-Order (BOOM) Core - out-of-order execution

MISSOURI S&T

Rocket Chip Generator 

Chip Design Rocket Chip Generator Example

- Berkeley Out-of-Order (BOOM) Core - out-of-order execution
  - Features both frontend and backend hardware logic
  - Frontend supplies instructions as needed
  - Branch predictor with address vs. last entry (taken, not taken)
  - Backend holds up to 16 instructions
  - Orders instructions, prioritizing older instructions, sends it out over three windows
  - Can accommodate two ALU operations and one memory operation

Berkeley Architecture Research

Chip Design SiFive

- SiFive - A commercial entity focused on promoting RISC V
  - Founded by Andrew Waterman, a creators of RISC V
  - Aims to bring commercial implementations of RISC V to market
  - First product - U54-MC Coreplex
    - 64-bit, quad-core
    - Capable of running Debian and Fedora
    - Intended for artificial intelligence, machine learning, networking, gateways and smart IoT devices

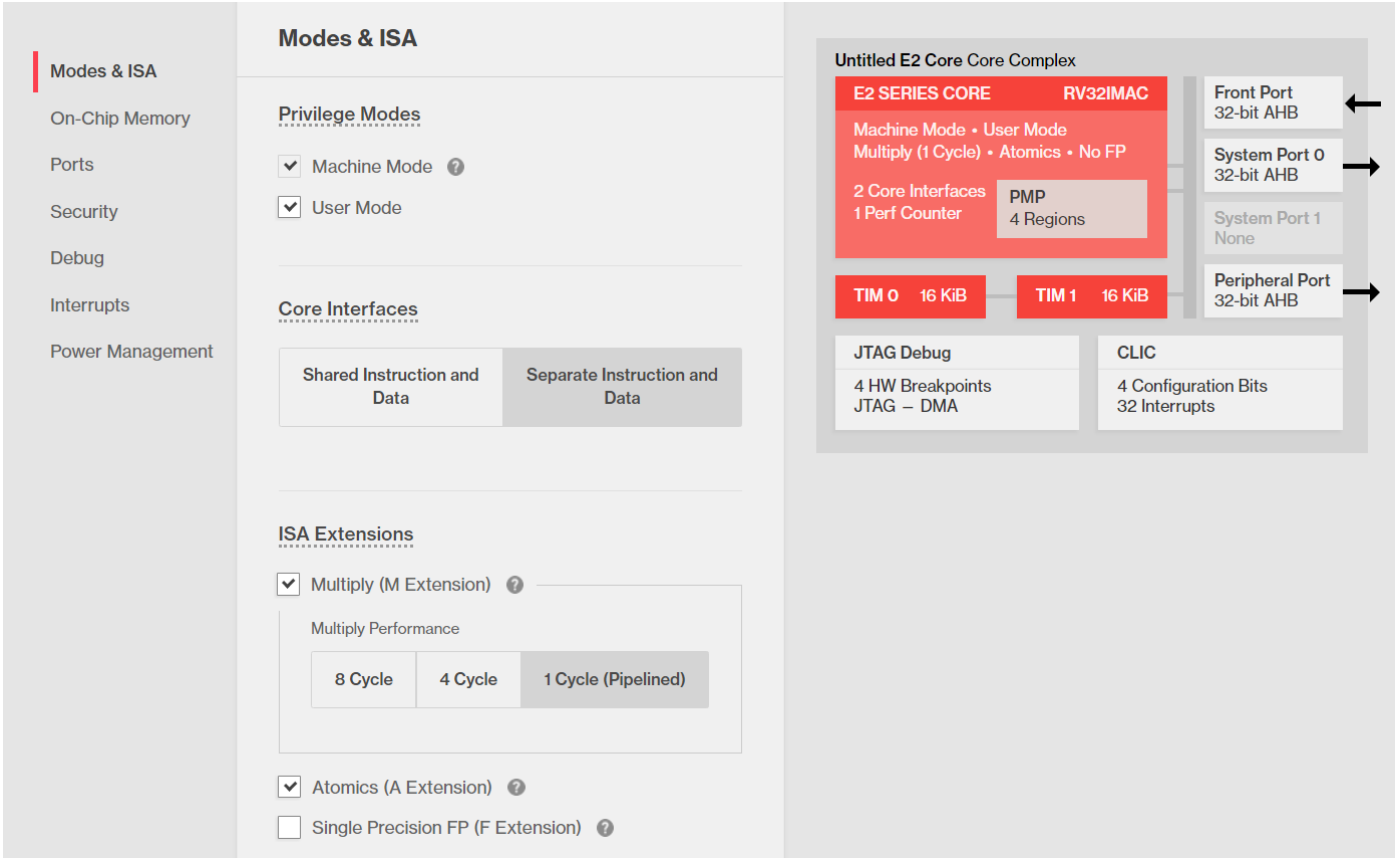
MISSOURI S&T

Chip Design SiFive

- Fully-fledged web-based chip designer
  - Core (CPU) designer available https://scs.sifive.com/core-designer/
  - Chip designer in progress https://www.sifive.com/chip-designer

MISSOURI S&T

Chip Design SiFive



Chip Design SiFive

- Modes & ISA
- On-Chip Memory
- Ports
- Security
- Debug
- Interrupts
- Power Management

On-Chip Memory

Tightly Integrated Memory

TIM 0

Size in KiB

0.5	1	2	4	8	16	32	64	128	256	512

Base Address: 0x8000_0000

Top Address: 0x8000_3FFF

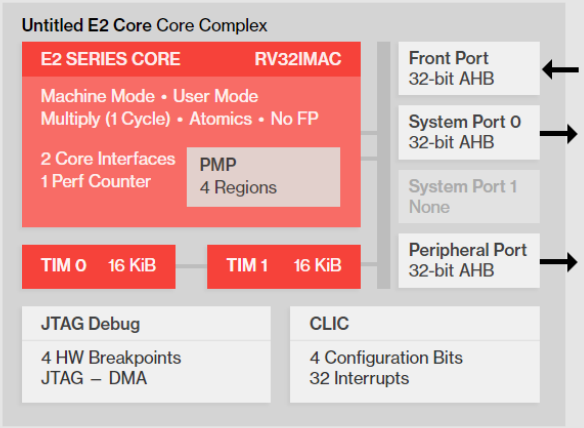
TIM 1

Size in KiB

0.5	1	2	4	8	16	32	64	128	256	512

Base Address: 0x9000_0000

Top Address: 0x9000_3FFF

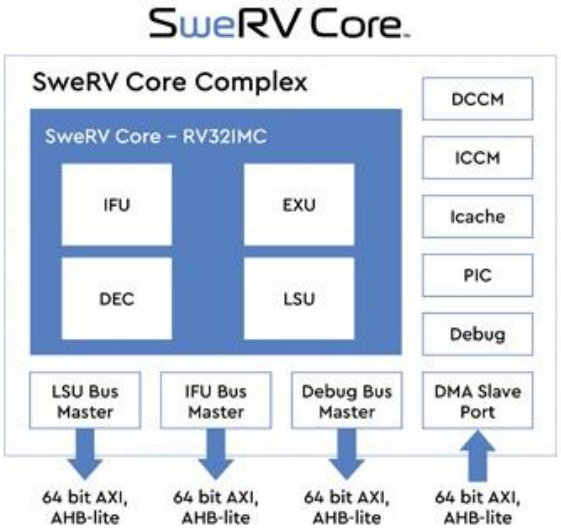


Chip Design Western Digital

- Western Digital signed on as a sponsor of RISC Foundation
  - Proposes using RISC-V cores in future products
  - SweRV Core™
  - Open-source corporate RISC-V

Chip Design Western Digital

- SweRV Core™ EHX1
  - 32-bit, 2-way superscalar, 9 stage pipeline core
  - Simulation performance up to 5.0 CoreMarks/Mhz
  - Aimed at storage controllers, industrial IoT, etc.
  - Clock speeds of up to 1.8Ghz
  - Open-sourced
  - https://github.com/westerndigitalcorporation/swerv_eh1



MISSOURI S&T

Background
Instructions
Memory Model
Pipelining
Chip Design
→ Performance
Conclusions

Performance

- SiFive U54-MC Coreplex
  - 2.75 CoreMark/MHz using the Coremark benchmark
  - In line with Intel Atom N450, Intel Core 2 Duo (Mobile U7600), Microchip's ARM-based PIC32 MX795, and Nvidia Tegra 250
- RISCY – IoT-oriented RISC-V core
  - 3.40 Coremark/MHz
  - Comparable with Cortex-M4
- TAIGA – FPGA-based RISC-V soft-processor framework
  - 33% less of the available FPGA hardware (slices)
  - Could be clocked 39% faster

MISSOURI S&T

Performance

- Benchmarking RISC V leaves a lot to be desired
  - Most software is still custom embedded solutions
  - Not directly portable to other architectures
- SiFive is only company to create working Linux system
  - No benchmarks beyond CoreMark chip score
  - Clearly not a threat to existing Linux systems… yet

MISSOURI S&T

Background
Instructions
Memory Model
Pipelining
Chip Design
Performance
→ Conclusions

Conclusions

- RISC V holds lofty goals breaking into an entrenched market
- “Becoming the standard ISA for all computing devices”
- Perhaps the most forward-thinking and comprehensive ISA among its competitors
- Encompassing a wide range of computing techniques
- Encourages open-source academic community development
- Attractive to commercial entities (SiFive, Western Digital)

MISSOURI S&T

Questions?

MISSOURI S&T