



UNIVERSITÀ
DEGLI STUDI
DI TRIESTE

Restoration and development of Arm's Java-based LEGv8 ISA simulator

Graduating student - Simone Deiana

Supervisor - Alberto Carini

Restoration and development of Arm's Java-based LEGv8 ISA simulator



Restoration and development of Arm's Java-based LEGv8 ISA simulator





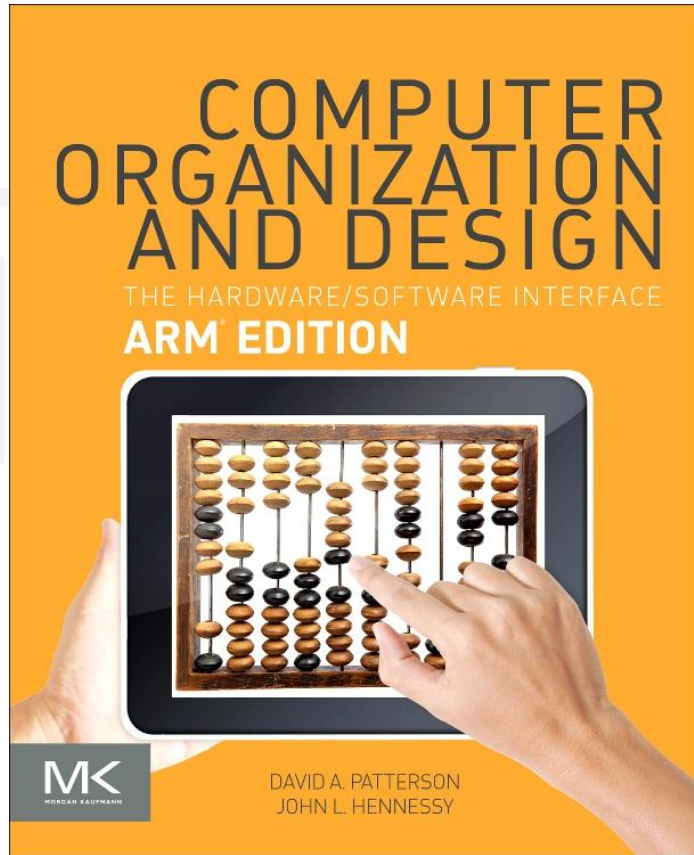
Restoration and development of Arm's Java-based LEGv8 ISA simulator

WHAT IS LEGv8?

Restoration and development of
Arm's Java-based **LEGv8 ISA** simulator

WHAT IS LEGv8?

AN ISA FOR LEARNING COMPUTER ARCHITECTURES



From Computer Organization and Design ARM Edition: The Hardware Software Interface - Patterson, D.A. and Hennessy, J.L.



David A.
Patterson

Peg Skorpinski, CC BY-SA
3.0
<<https://creativecommons.org/licenses/by-sa/3.0/>>, via
Wikimedia Commons



John L.
Hennessy

Eric Chan, CC BY 2.0
<<https://creativecommons.org/licenses/by/2.0/>>, via Wikimedia Commons



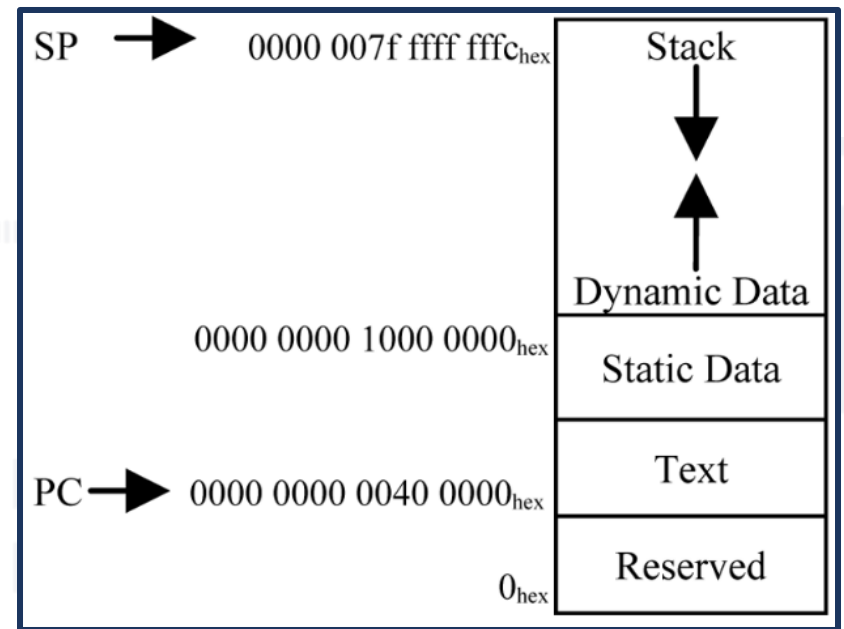
UNIVERSITÀ
DEGLI STUDI
DI TRIESTE

THE DESIGN PHILOSOPHY

- As simple as it can be...
- ... but with a modern design
- Heavily inspired by ARMv8, almost a "subset"

THE MEMORY

- 64-bit addresses
- Harvard model



THE REGISTERS

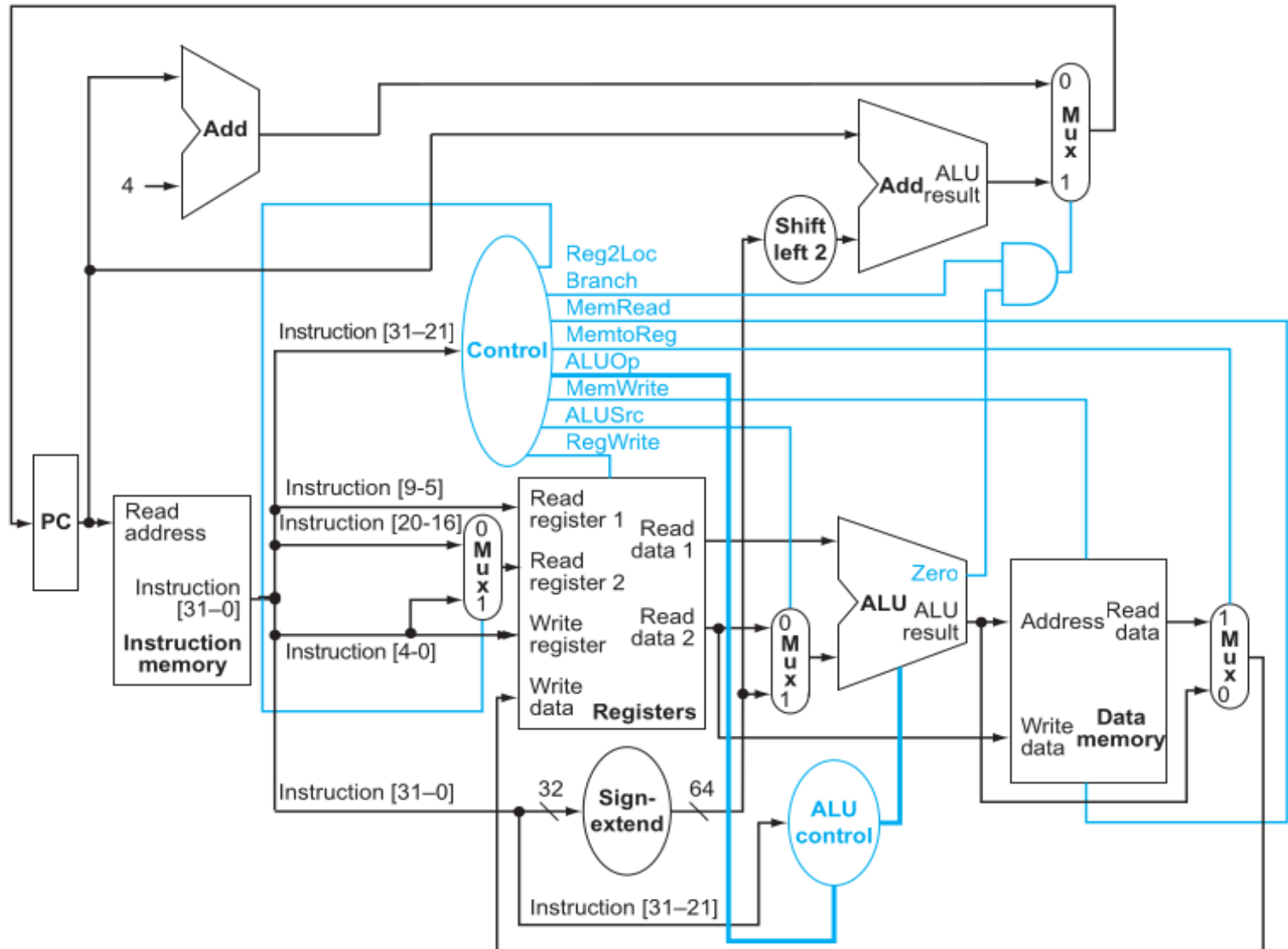
- 32 64-bit "X" integer registers
- 32 64-bit "D" floating-point registers
- 32 32-bit "S" floating-point "registers"

THE INSTRUCTIONS

- 64-bit integer and IEEE-754 floating-point arithmetic
- Designed and optimized for pipelined execution

WHAT IS LEGv8?

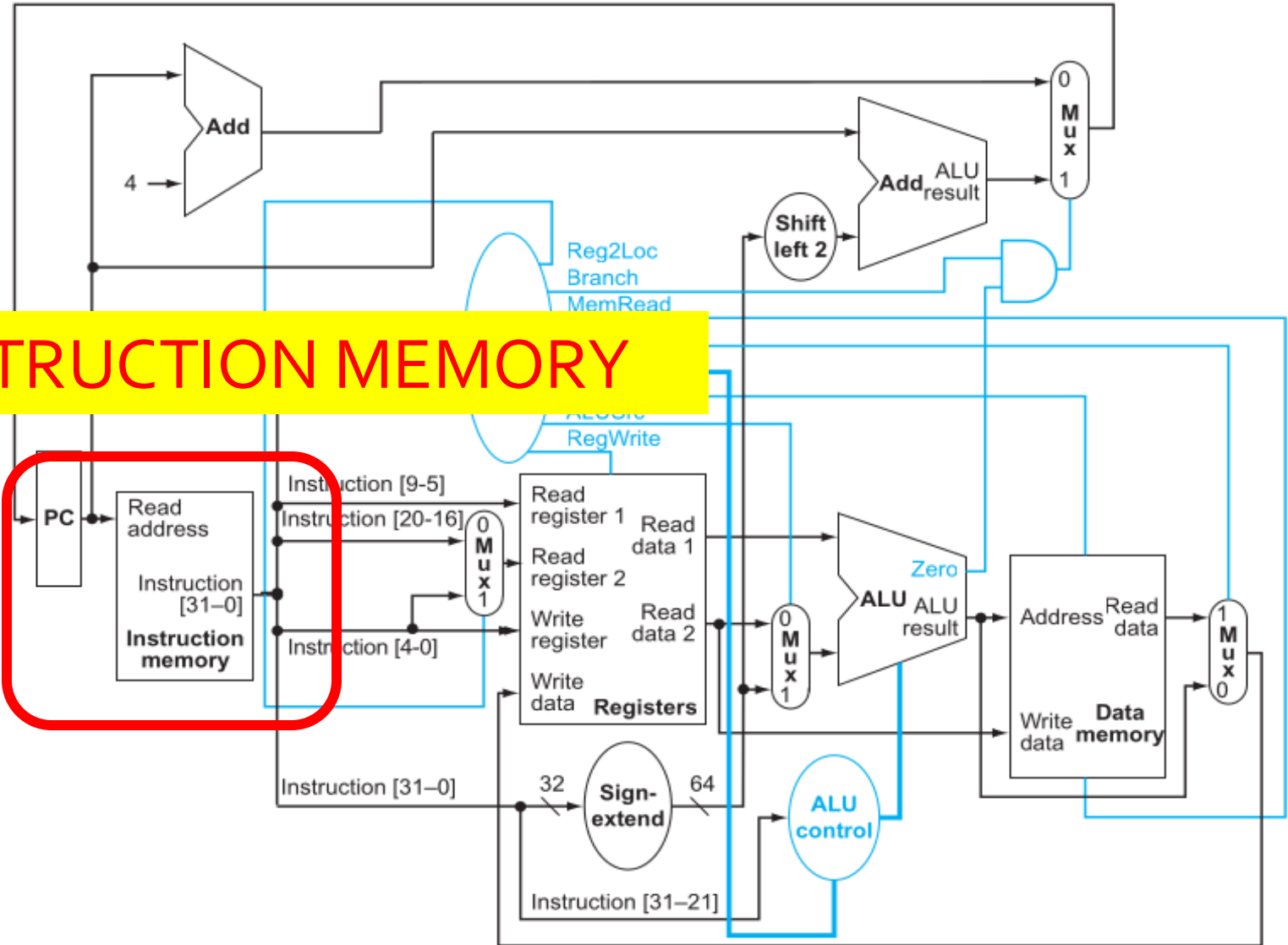
THE DATAPATH



From Computer Organization and Design ARM Edition: The Hardware Software Interface - Patterson, D.A. and Hennessy, J.L.

WHAT IS LEGv8?

THE DATAPAH

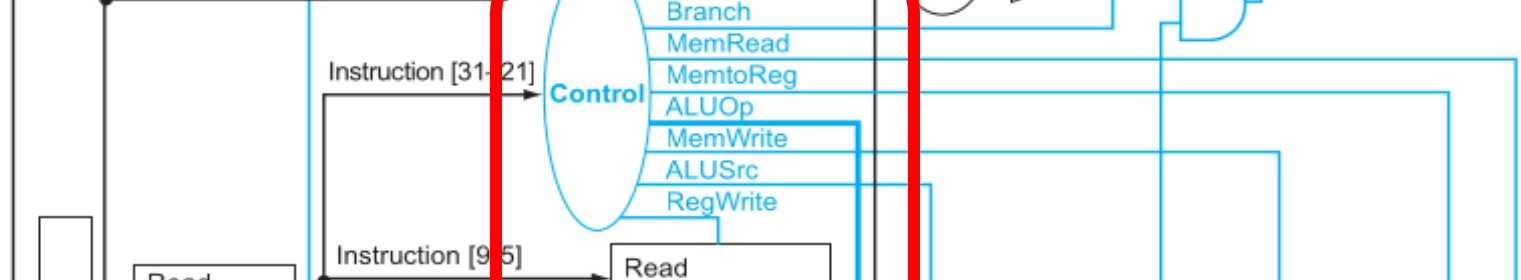


From Computer Organization
and Design ARM Edition: The
Hardware Software Interface -
Patterson, D.A. and Hennessy,
J.L.

Diagram of the Control Unit in a MIPS processor. The Control Unit is a central oval labeled "Control". It receives two inputs: "Instruction [31-21]" and "Instruction [9-5]". It has seven outputs: "Branch", "MemRead", "MemtoReg", "ALUOp", "MemWrite", "ALUSrc", and "RegWrite". The "MemWrite" output is connected to a "Read" block.

Diagram illustrating the Control Unit and its connections:

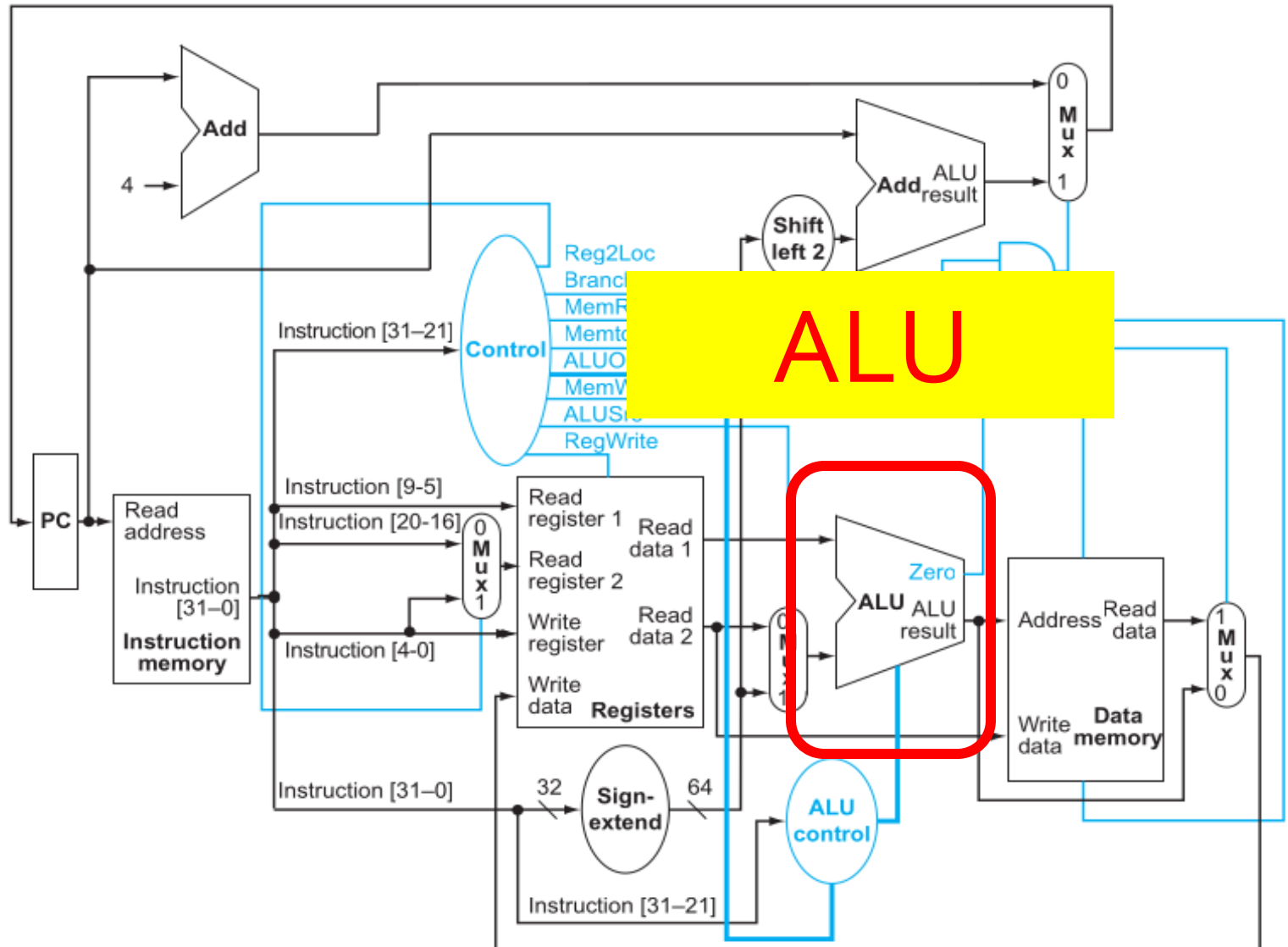
- Control Unit** (Central oval) receives inputs:
 - Instruction [31:21]
 - Instruction [9:5]
- Control Unit** outputs control signals:
 - Branch
 - MemRead
 - MemtoReg
 - ALUOp
 - MemWrite
 - ALUSrc
 - RegWrite
- Read** block receives:
 - Instruction [9:5]
 - RegWrite (from Control Unit)



From Computer Organization and Design ARM Edition: The Hardware Software Interface - Patterson, D.A. and Hennessy, J.L.

WHAT IS LEGv8?

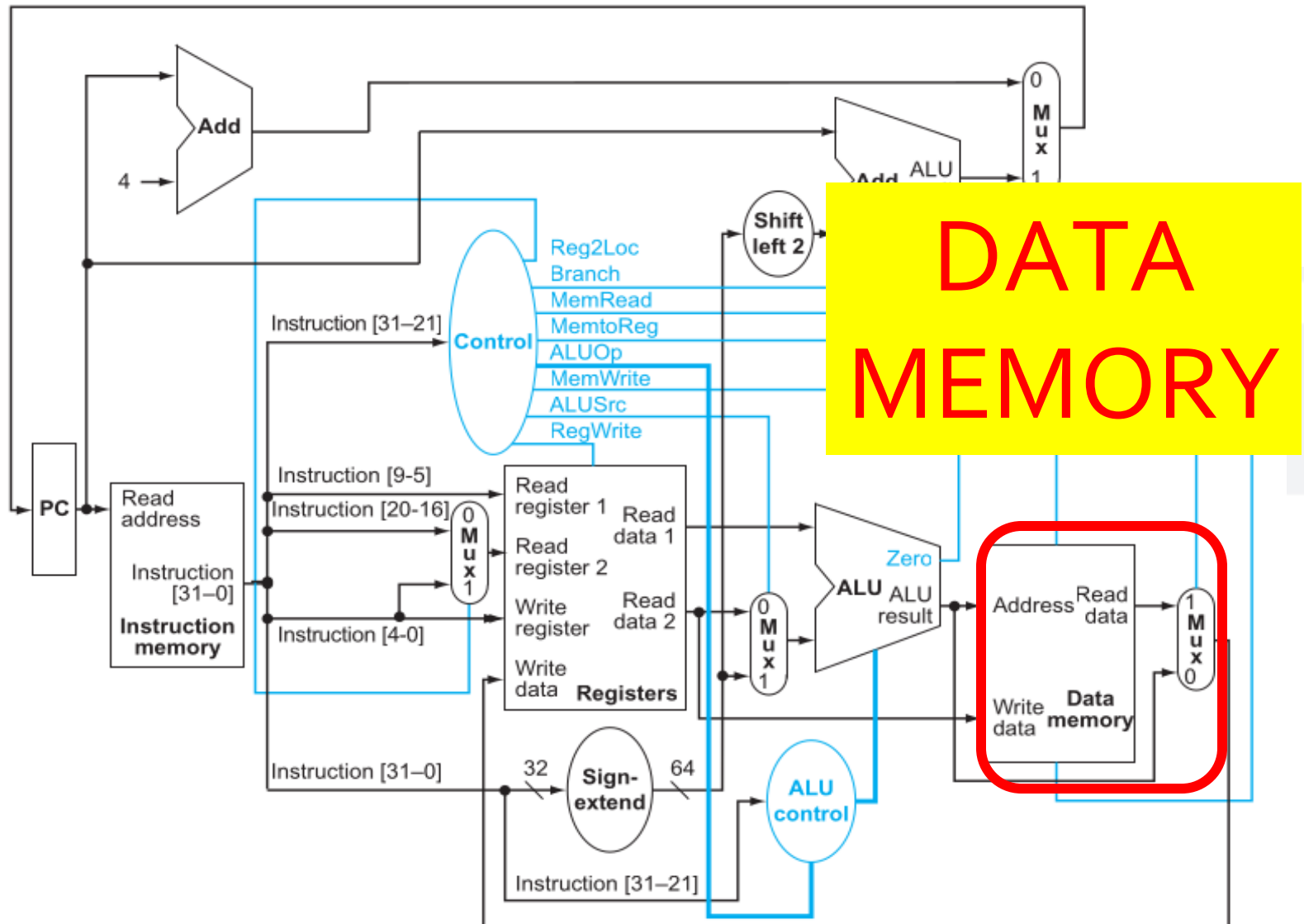
THE DATAPATH



From Computer Organization and Design ARM Edition: The Hardware Software Interface - Patterson, D.A. and Hennessy, J.L.

WHAT IS LEGv8?

THE DATAPATH

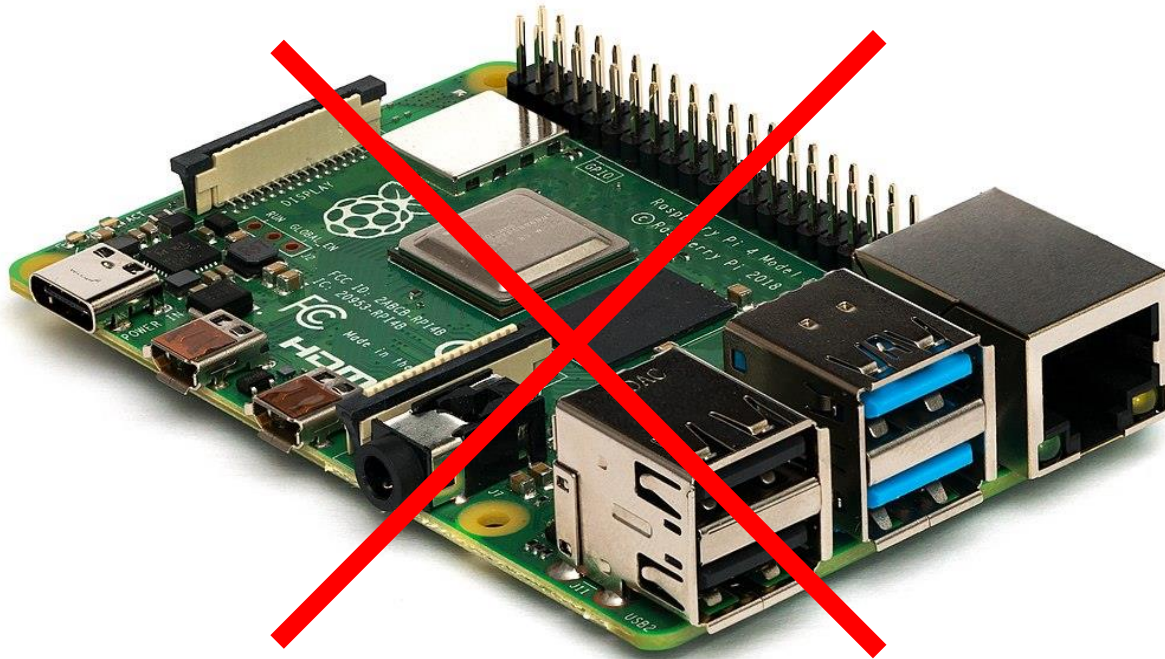


WHAT SIMULATOR, AND WHY?

Restoration and development of
Arm's Java-based LEGv8 ISA simulator

WHAT SIMULATOR, AND WHY?

NO HARDWARE FOR LEGv8 => NEED A SIMULATOR



Michael H. („Laserlicht“) / Wikimedia Commons

WHAT SIMULATOR, AND WHY?

BUT WHICH ONE?



LEGv8

197 results

THE PROBLEM:

NO SOFTWARE CAN YET
SIMULATE THE ENTIRE
LEGv8 ISA!

THE SOLUTION:

- Write one from scratch
- OR (BETTER)
- Improve one that already exists

THE GOOD NEWS

ARM HAS OFFICIALLY MADE A
LEGv8 SIMULATOR

WHAT STANDS OUT:

- Written in Java (high level, extensible)
- Distributed as a web application
- Nice, functional UI
- Closely follows the textbook

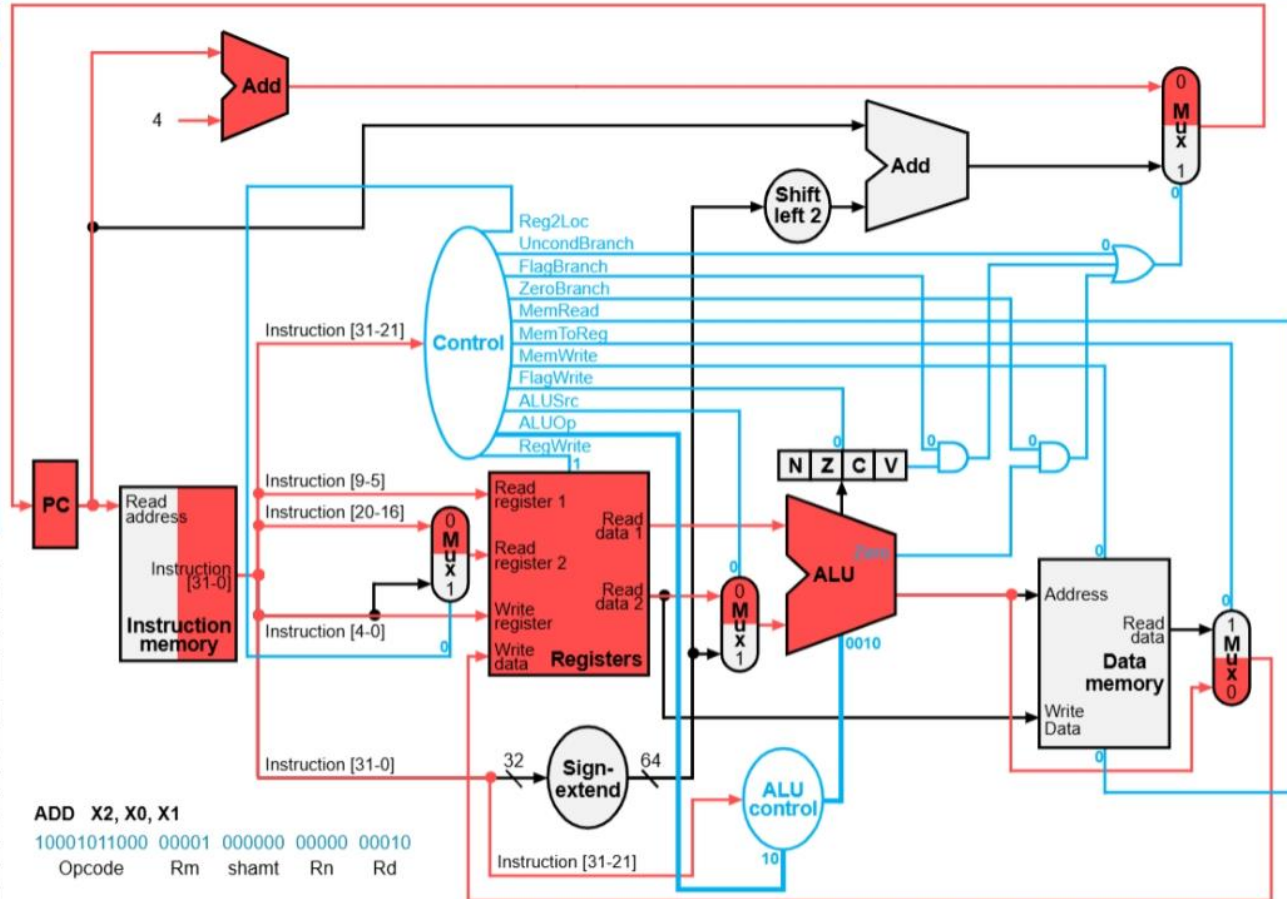
WHAT SIMULATOR, AND WHY?

LEGv8 Simulator

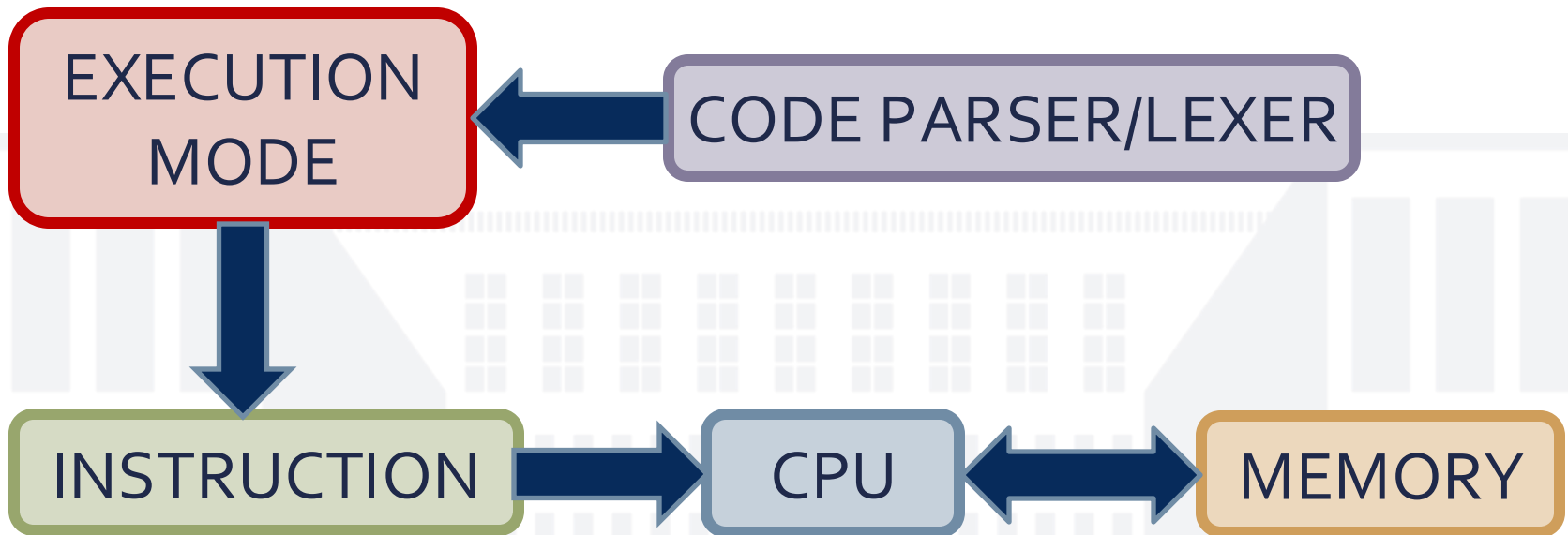
Execution Mode: **Single Cycle** Assemble Execute Instruction Help

```
1 ADDI X0, XZR, #53
2 ADDI X1, XZR, #75
3 ADD X2, X0, X1
4 CMP X2, X0
5 B.LE if
6 if: SUBIS X3, X2, #9
7 MOV X4, X3
8
```

PC	0x40000c	Hex	Dec	Z	0	N	0	C	0	V	0
X0	0x35	Hex	Dec	X16	0x0	Hex	Dec	X16	0x0	Hex	Dec
X1	0x4b	Hex	Dec	X17	0x0	Hex	Dec	X17	0x0	Hex	Dec
X2	0x80	Hex	Dec	X18	0x0	Hex	Dec	X18	0x0	Hex	Dec
X3	0x0	Hex	Dec	X19	0x0	Hex	Dec	X19	0x0	Hex	Dec
X4	0x0	Hex	Dec	X20	0x0	Hex	Dec	X20	0x0	Hex	Dec
X5	0x0	Hex	Dec	X21	0x0	Hex	Dec	X21	0x0	Hex	Dec
X6	0x0	Hex	Dec	X22	0x0	Hex	Dec	X22	0x0	Hex	Dec
X7	0x0	Hex	Dec	X23	0x0	Hex	Dec	X23	0x0	Hex	Dec
X8	0x0	Hex	Dec	X24	0x0	Hex	Dec	X24	0x0	Hex	Dec
X9	0x0	Hex	Dec	X25	0x0	Hex	Dec	X25	0x0	Hex	Dec
X10	0x0	Hex	Dec	X26	0x0	Hex	Dec	X26	0x0	Hex	Dec
X11	0x0	Hex	Dec	X27	0x0	Hex	Dec	X27	0x0	Hex	Dec
X12	0x0	Hex	Dec	SP	0x7ffffffc	Hex	Dec	SP	0x7ffffffc	Hex	Dec
X13	0x0	Hex	Dec	FP	0x0	Hex	Dec	FP	0x0	Hex	Dec
X14	0x0	Hex	Dec	LR	0x0	Hex	Dec	LR	0x0	Hex	Dec
X15	0x0	Hex	Dec	XZR	0x0	Hex	Dec	XZR	0x0	Hex	Dec



STRUCTURE OF THE SIMULATOR



THE BAD NEWS

IT'S INCOMPLETE
AND BROKEN

FIXING AND RESTORING THE SIMULATOR

Restoration and development of
Arm's Java-based LEGv8 ISA simulator

COMPARISONS DON'T WORK!

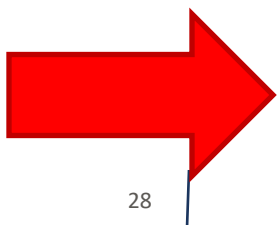
- No "if-else" conditionals
- No "switch-case" conditionals
- No "while" loops
- No "for" loops

THE COMPARISON BUG

```
1      ADDI    X0, XZR, #1
2      ADDI    X1, XZR, #2
3      CMP     X0, X1
4      B.LE    should_jump
5      ADD     X2, X0, X1
6  should_jump:  ADDI    X2, XZR, #16
7
```

X0 = 1, X1 = 2

PC	0x400008	Hex	Dec	Z	0	N	0	C	0	V	0
X0	0x1	Hex	Dec	X16			0x0			Hex	Dec
X1	0x2	Hex	Dec	X17			0x0			Hex	Dec
X2	0x0	Hex	Dec	X18			0x0			Hex	Dec

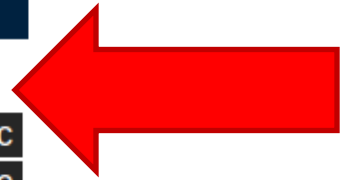


THE COMPARISON BUG

1	ADDI	X0, XZR, #1
2	ADDI	X1, XZR, #2
3	CMP	X0, X1
4	B.LE	should_jump
5	ADD	X2, X0, X1
6	should_jump:	ADDI X2, XZR, #16
7		

COMPARE X0 WITH X1

PC	0x40000c	Hex	Dec	Z	0	N	1	C	0	V	1
X0	0x1	Hex	Dec	X16			0x0			Hex	Dec
X1	0x2	Hex	Dec	X17			0x0			Hex	Dec
X2	0x0	Hex	Dec	X18			0x0			Hex	Dec



THE COMPARISON BUG

```
1 |      ADDI      X0, XZR, #1
2 |      ADDI      X1, XZR, #2
3 |      CMP       X0, X1
4 |      B.LE      should_jump
5 |      ADD       X2, X0, X1
6 |  should_jump:  ADDI      X2, XZR, #16
7 |
```

1 <= 2 ?

OF COURSE, LET'S JUMP!

PC	0x400010	Hex	Dec	Z	0	N	1	C	0	V	1
X0	0x1	Hex	Dec	X16			0x0	Hex	Dec		
X1	0x2	Hex	Dec	X17			0x0	Hex	Dec		
X2	0x0	Hex	Dec	X18			0x0	Hex	Dec		

THE COMPARISON BUG

```
1 |      ADDI      X0, XZR, #1
2 |      ADDI      X1, XZR, #2
3 |      CMP       X0, X1
4 |      B.LE      should_jump
5 |      ADD       X2, X0, X1
6 | should_jump:   ADDI      X2, XZR, #16
7 |
```

...OR NOT

PC	0x400014	Hex	Dec	Z	0	N	1	C	0	V	1
X0	0x1	Hex	Dec	X16			0x0	Hex	Dec		
X1	0x2	Hex	Dec	X17			0x0	Hex	Dec		
X2	0x3	Hex	Dec	X18			0x0	Hex	Dec		

BRANCH AND LINKS DON'T WORK!

~~*void subroutine(arg1, ...)*~~
~~*float function(arg1, ...)*~~

CAN'T REUSE CODE

THE BRANCH AND LINK BUG

```
1      ADDI      X0, XZR, #1
2      BL        subroutine
3      B         exit
4
5  subroutine:
6      ADDI      X0, X0, #16
7      BR        LR
8  exit:
9
```

THE BRANCH AND LINK BUG

```
1      ADDI      X0, XZR, #1
2      BL        subroutine
3      B         exit
4
5  subroutine:
6      ADDI      X0, X0, #16
7      BR        LR
8  exit:
9
```

THE BRANCH AND LINK BUG


```
1      ADDI    X0, XZR, #1
2      BL      subroutine
3      B        exit
4
5  subroutine:
6      ADDI    X0, X0, #16
7      BR      LR
8  exit:
9
```

THE BRANCH AND LINK BUG

```
1      ADDI      X0, XZR, #1
2      BL        subroutine
3      B         exit
4
5  subroutine:
6      ADDI      X0, X0, #16
7      BR        LR
8  exit:
9
```

THE BRANCH AND LINK BUG

1	ADDI	X0, XZR, #1	
2	BL	subroutine	
3	B	exit	
4			
5	subroutine:		
6	ADDI	X0, X0, #16	
7	BR	LR	
8	exit:		
9			

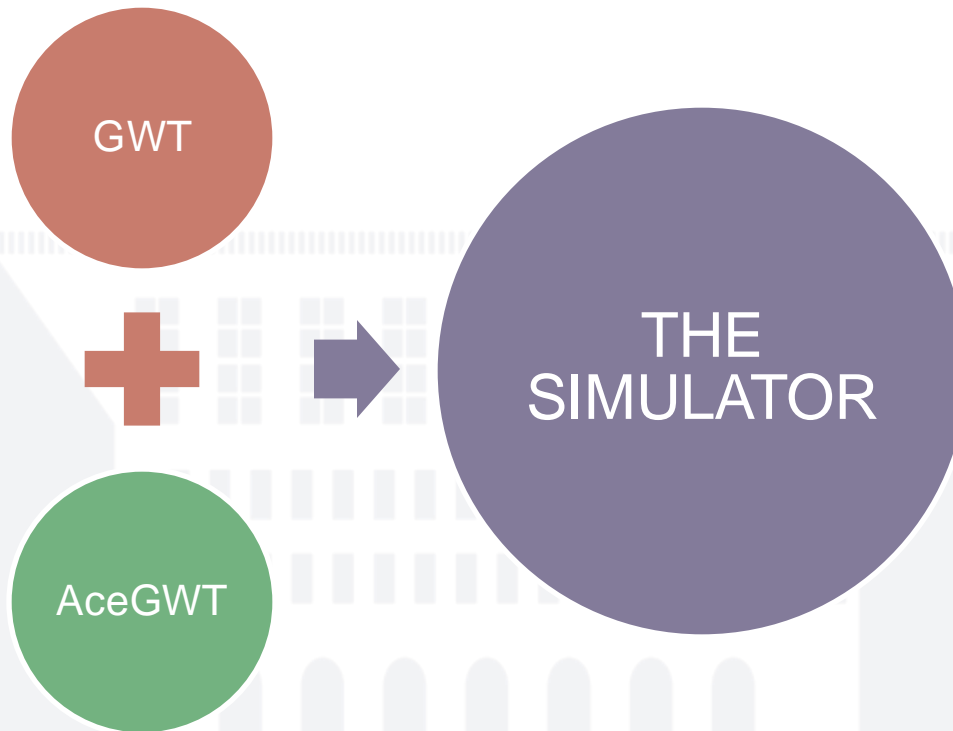


WE SHOULD BE HERE!

IT DOESN'T GO BACK!

ALL FIXED, BUT...
NOBODY KNOWS HOW IT WORKS!

THE PROJECT'S DEPENDENCIES



GWT

- Framework (formerly) from Google
- Generates web applications (client-server, client only) from Java
- Emulates Java's JVM with JavaScript

GWT

- Old, outdated, barely supported
- Convoluted custom build tools
- Limited emulation of JVM
- Basically needs Eclipse plug-in for real development

AceGWT

- Provides GWT bindings for the Ace editor
- Can be used like normal GWT component
- Also old, outdated, and unsupported

WORKING IT OUT:

- Needed older version of Eclipse, and Eclipse GWT plug-in
- Reverse engineered the dependencies and their configuration
- Configured build system to stop failing

AFTER 3 YEARS, WE CAN NOW
PRODUCE NEW VERSIONS
OF THE SIMULATOR!

FILLING THE GAPS

Restoration and **development** of
Arm's Java-based LEGv8 ISA simulator

WHAT IS THE SIMULATOR MISSING?

- Incomplete integer arithmetic
- No visualization for the stack memory
- No IEEE-754 arithmetic and data instructions

THE MISSING INTEGER-BASED INSTRUCTIONS

- **MUL** — LOWER 64 BITS OF THE MULTIPLICATION
- **SMULH** — HIGHER 64 BITS OF THE SIGNED MULTIPLICATION
- **UMULH** — HIGHER 64 BITS OF THE UNSIGNED MULTIPLICATION
- **SDIV** — SIGNED DIVISION
- **UDIV** — UNSIGNED DIVISION
- **LDA** — LOAD ADDRESS OF A LABEL IN A REGISTER

Easy to implement in
the existing codebase

but...

Java doesn't like big
or unsigned numbers!

PROBLEM 1

- Java does not have primitive 128-bit integer types
- The BigInteger library exists
- GWT 2.7 doesn't emulate it (2.8 does)

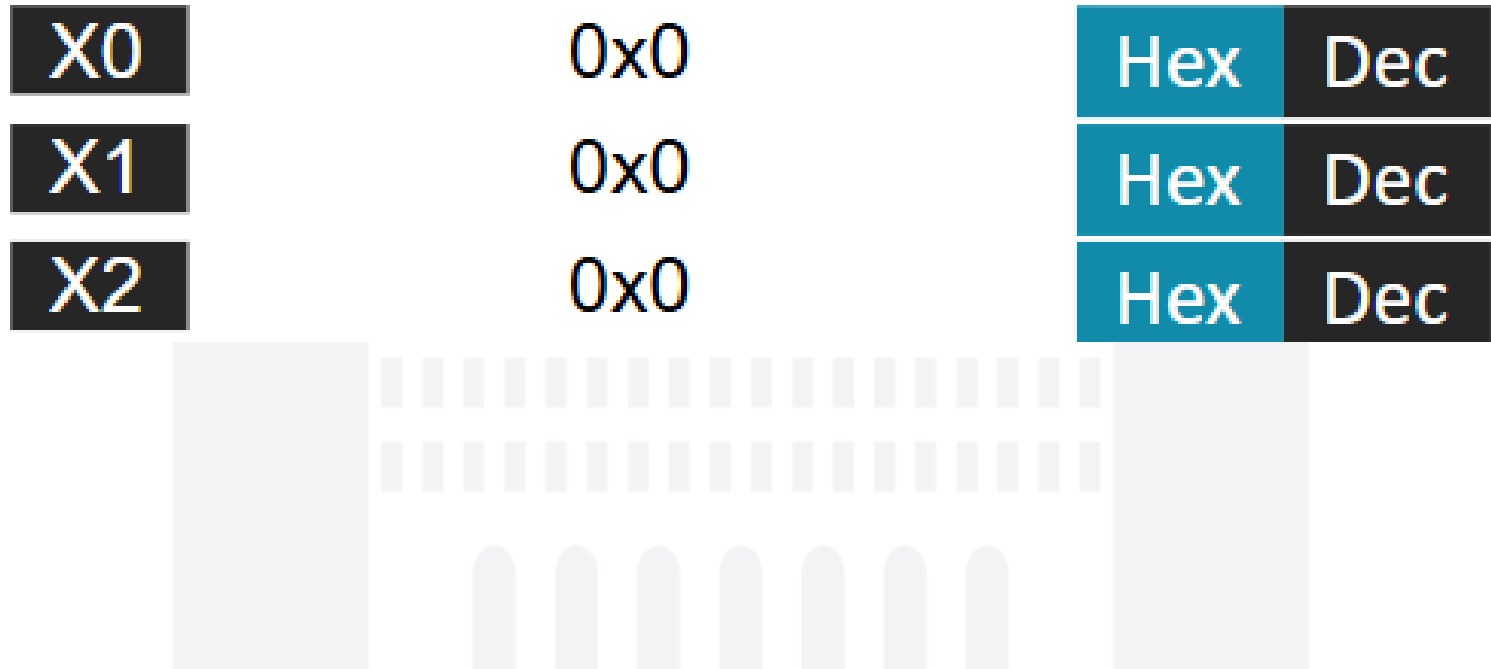
PROBLEM 2

- Primitive integers are signed
- BigInteger also signed
- Bitmask converts 64-bit unsigned integers to 65-bit signed, perform signed operations, back to unsigned

STACK NOT VISUALIZED

- Important for testing and debugging complex programs (now we can write them)
- Useful to understand LEGv8 and stack management

TAKING SOME INSPIRATION



THE NEW STACK VIEW

0x8000000000:	0x0	Hex	0x7fffffff80:	0x0	Hex
0x7fffffff8:	0x0	Hex	0x7fffffff78:	0x0	Hex
0x7fffffff0:	0x0	Hex	0x7fffffff70:	0x0	Hex
0x7fffffff8:	0x0	Hex	0x7fffffff68:	0x0	Hex
0x7fffffff80:	0x0	Hex	0x7fffffff60:	0x0	Hex
0x7fffffff8:	0x0	Hex	0x7fffffff58:	0x0	Hex
0x7fffffff80:	0x0	Hex	0x7fffffff50:	0x0	Hex
0x7fffffff8:	0x0	Hex	0x7fffffff48:	0x0	Hex
0x7fffffff80:	0x0	Hex	0x7fffffff40:	0x0	Hex
0x7fffffff8:	0x0	Hex	0x7fffffff38:	0x0	Hex
0x7fffffff80:	0x0	Hex	0x7fffffff30:	0x0	Hex
0x7fffffff8:	0x0	Hex	0x7fffffff28:	0x0	Hex
0x7fffffff80:	0x0	Hex	0x7fffffff20:	0x0	Hex
0x7fffffff8:	0x0	Hex	0x7fffffff18:	0x0	Hex
0x7fffffff80:	0x0	Hex	0x7fffffff10:	0x0	Hex
0x7fffffff88:	0x0	Hex	0x7fffffff08:	0x0	Hex

ADDING FLOATING-POINT SUPPORT

- **FADDS, FADDD** — ADD TWO IEEE-754 VALUES
- **FSUBS, FSUBD** - SUBTRACT TWO IEEE-754 VALUES
- **FMULS, FMULD** - MULTIPLY TWO IEEE-754 VALUES
- **FDIVS, FDIVD** — DIVIDE TWO IEEE-754 VALUES
- **LDURS, LDURD** — LOAD IEEE-754 VALUE FROM MEMORY
- **STURS, STURD** - STORE IEEE-754 VALUE TO MEMORY
- **FCMPS, FCMPSD** — COMPARE TWO IEEE-754 VALUES

ARITHMETICAL INSTRUCTIONS



Java's
IEEE-754

LEGv8's
IEEE-754

MEMORY ACCESS INSTRUCTIONS

- Simulator designed for integer use
- Memory uses integers to store bytes

IDEA: SEE INTEGERS LIKE RAW BITS



CONVERT FLOATS
TO RAW BITS

CAN BE USED AS
BINARY PROTOCOL



RETROCOMPATIBILITY!

COMPARISON INSTRUCTIONS

- LEGv8 does not specify flag-setting conditions for IEEE-754 comparisons
- Use ARMv8's ones:

IEEE-754 Relationship	ARM APSR Flags			
	N	Z	C	V
Equal	0	1	1	0
Less Than	1	0	0	0
Greater Than	0	0	1	0
Unordered (<i>At least one argument was NaN.</i>)	0	0	1	1

THE FINAL VIEW

1	MOVK	X0, #192	0x80000000:	0x0	Hex	0x7fffffff80:	0x0	Hex
2	LSL	X0, X0, #8	0x7fffffff8:	0x0	Hex	0x7fffffff78:	0x0	Hex
3	MOVK	X0, #10	0x7fffffff0:	0xc00a0000	Hex	0x7fffffff70:	0x0	Hex
4	LSL	X0, X0, #16	0x7fffffff0:	0x0	Hex	0x7fffffff68:	0x0	Hex
5	SUBI	SP, SP, #16	0x7fffffff0:	0x0	Hex	0x7fffffff60:	0x0	Hex
6	STUR	X0, [SP, #0]	0x7fffffff0:	0x0	Hex	0x7fffffff58:	0x0	Hex
7	LDURS	S0, [SP, #0]	0x7fffffff0:	0x0	Hex	0x7fffffff50:	0x0	Hex
8			0x7ffffffc8:	0x0	Hex	0x7fffffff48:	0x0	Hex
			0x7ffffffc0:	0x0	Hex	0x7fffffff40:	0x0	Hex
			0x7ffffffb8:	0x0	Hex	0x7fffffff38:	0x0	Hex
			0x7ffffffb0:	0x0	Hex	0x7fffffff30:	0x0	Hex
			0x7ffffffa8:	0x0	Hex	0x7fffffff28:	0x0	Hex
			0x7ffffffa0:	0x0	Hex	0x7fffffff20:	0x0	Hex
			0x7fffffff98:	0x0	Hex	0x7fffffff18:	0x0	Hex
			0x7fffffff90:	0x0	Hex	0x7fffffff10:	0x0	Hex
			0x7fffffff88:	0x0	Hex	0x7fffffff08:	0x0	Hex

Dec	Z	N	C	V	D0	0xc00a0000	Hex	Dec	D16	0x0	Hex	Dec	S0	-2.15625	Hex	Dec	S16
Dec	X16	0x0	Hex	Dec	D1	0x0	Hex	Dec	D17	0x0	Hex	Dec	S1	0x0	Hex	Dec	S17
Dec	X17	0x0	Hex	Dec	D2	0x0	Hex	Dec	D18	0x0	Hex	Dec	S2	0x0	Hex	Dec	S18
Dec	X18	0x0	Hex	Dec	D3	0x0	Hex	Dec	D19	0x0	Hex	Dec	S3	0x0	Hex	Dec	S19
Dec	X19	0x0	Hex	Dec	D4	0x0	Hex	Dec	D20	0x0	Hex	Dec	S4	0x0	Hex	Dec	S20
Dec	X20	0x0	Hex	Dec	D5	0x0	Hex	Dec	D21	0x0	Hex	Dec	S5	0x0	Hex	Dec	S21
Dec	X21	0x0	Hex	Dec	D6	0x0	Hex	Dec	D22	0x0	Hex	Dec	S6	0x0	Hex	Dec	S22
Dec	X22	0x0	Hex	Dec	D7	0x0	Hex	Dec	D23	0x0	Hex	Dec	S7	0x0	Hex	Dec	S23
Dec	X23	0x0	Hex	Dec	D8	0x0	Hex	Dec	D24	0x0	Hex	Dec	S8	0x0	Hex	Dec	S24
Dec	X24	0x0	Hex	Dec	D9	0x0	Hex	Dec	D25	0x0	Hex	Dec	S9	0x0	Hex	Dec	S25
Dec	X25	0x0	Hex	Dec	D10	0x0	Hex	Dec	D26	0x0	Hex	Dec	S10	0x0	Hex	Dec	S26
Dec	X26	0x0	Hex	Dec	D11	0x0	Hex	Dec	D27	0x0	Hex	Dec	S11	0x0	Hex	Dec	S27
Dec	X27	0x0	Hex	Dec	D12	0x0	Hex	Dec	D28	0x0	Hex	Dec	S12	0x0	Hex	Dec	S28
Dec	SP	0x7fffffff0	Hex	Dec	D13	0x0	Hex	Dec	D29	0x0	Hex	Dec	S13	0x0	Hex	Dec	S29
Dec	FP	0x0	Hex	Dec	D14	0x0	Hex	Dec	D30	0x0	Hex	Dec	S14	0x0	Hex	Dec	S30
Dec	LR	0x0	Hex	Dec	D15	0x0	Hex	Dec	D31	0x0	Hex	Dec	S15	0x0	Hex	Dec	S31
Dec	XZR	0x0	Hex	Dec													

THE CHERRY ON TOP: MODERNIZING THE BUILD SYSTEM

WHY DO THIS?

- Latest GWT and AceGWT support Maven
- Much easier to include if project also supports Maven

THE TRICKLE-DOWN EFFECT

- Libraries are managed automatically
- Project decoupled from Eclipse, can use other IDEs or even terminal
- Can use Java 21, GWT 2.11
- More configurable builds

CONCLUSIONS

- Arm's LEGv8 simulator finally working
- Only one to implement every LEGv8 instruction
- Can now be developed with modern tools, set-up and build in seconds, much easier collaboration

THANK YOU FOR
YOUR ATTENTION

[THESIS AVAILABLE HERE](#)

[SIMULATOR AVAILABLE HERE](#)