UNIVERSITÀ
DEGLI STUDI
DI TRIESTE

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

Graduating student - Simone Deiana

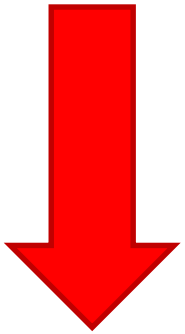
Supervisor - Alberto Carini

Department of Engineering and Architecture

Trieste, 30 August 2024

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

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

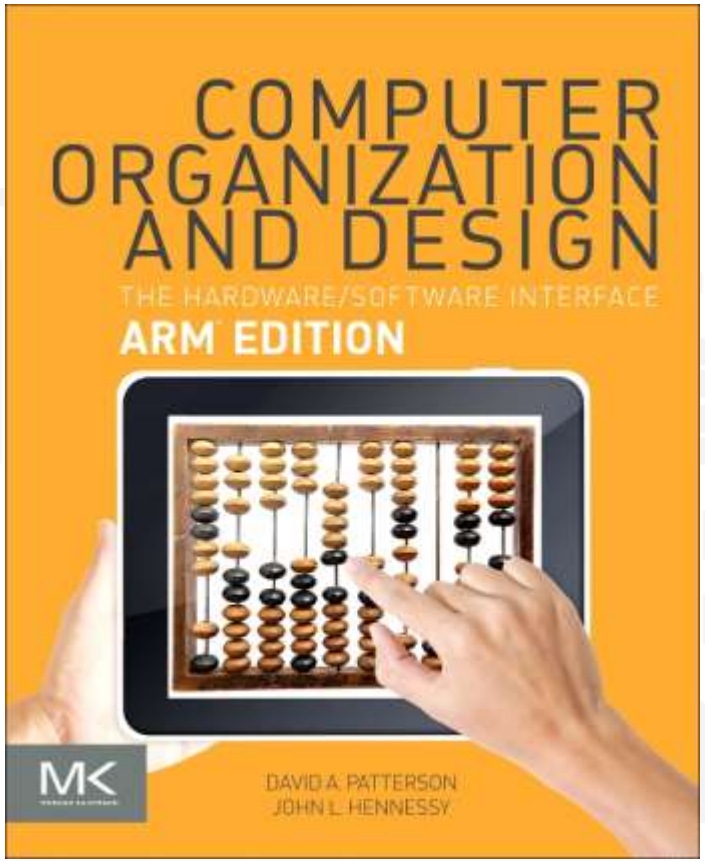


Restoration and development of Arm's Java-based LEGv8 ISA simulator

WHAT IS LEGv8?

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

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

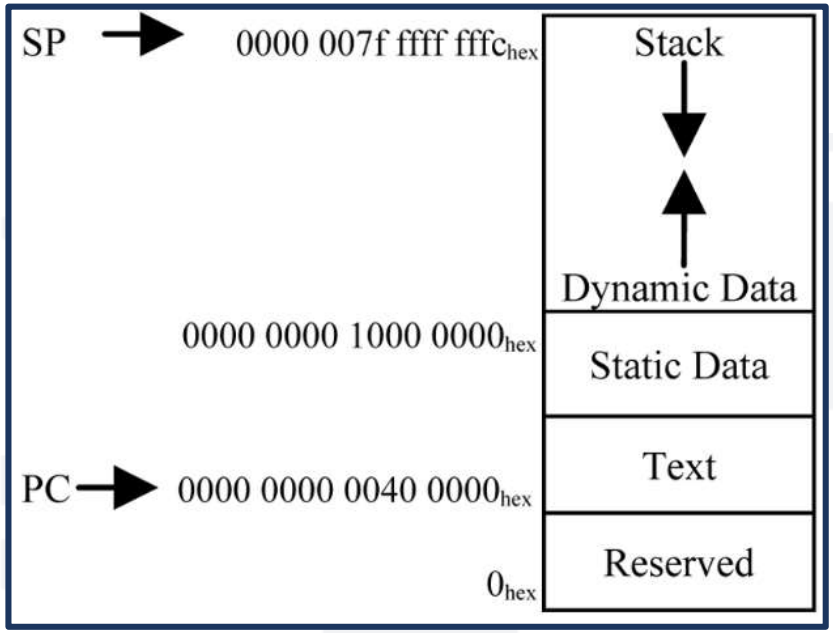
UNIVERSITÀ DEGLI STUDI DI TRIESTE

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



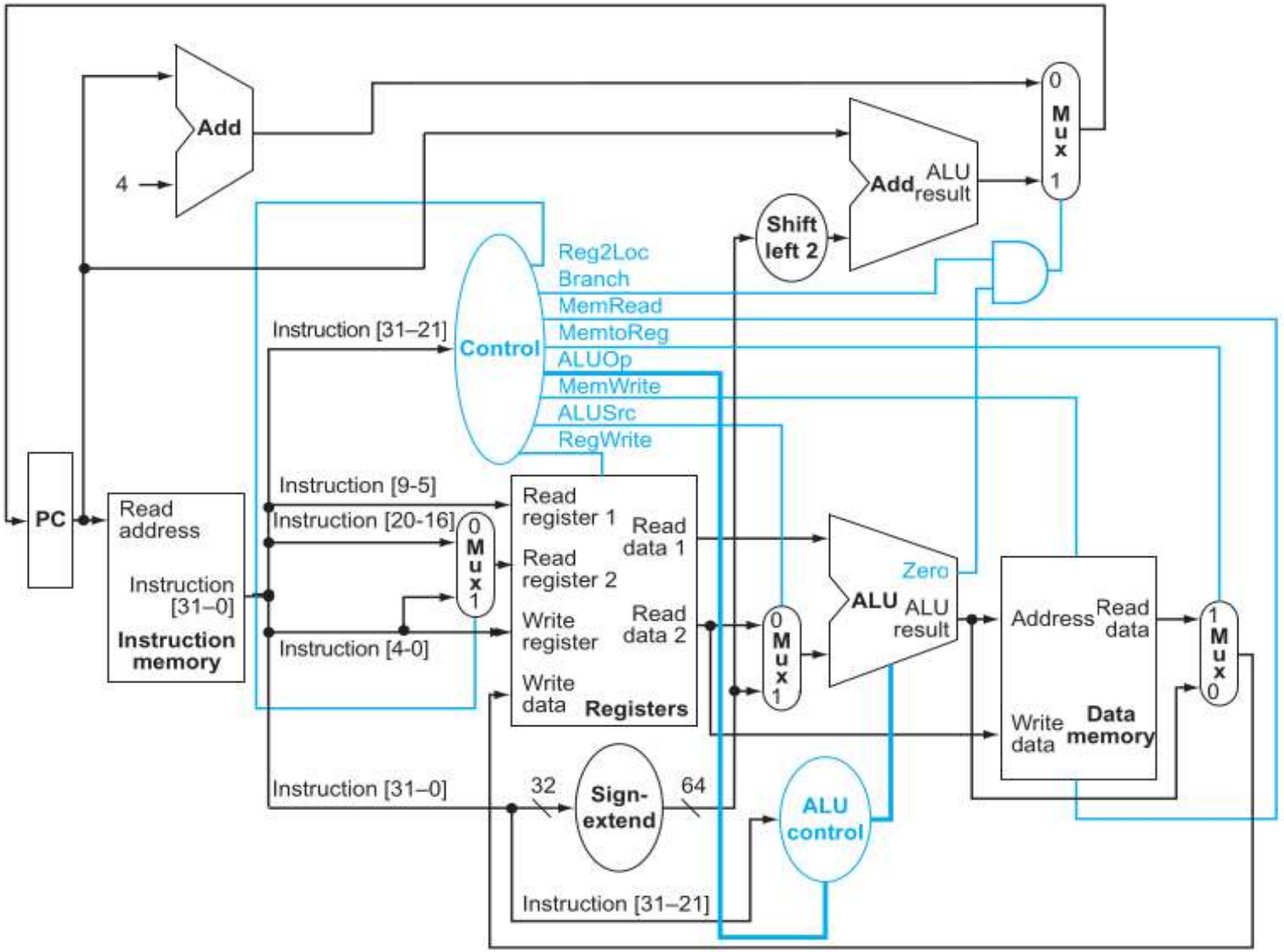
THE REGISTERS

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

THE INSTRUCTIONS

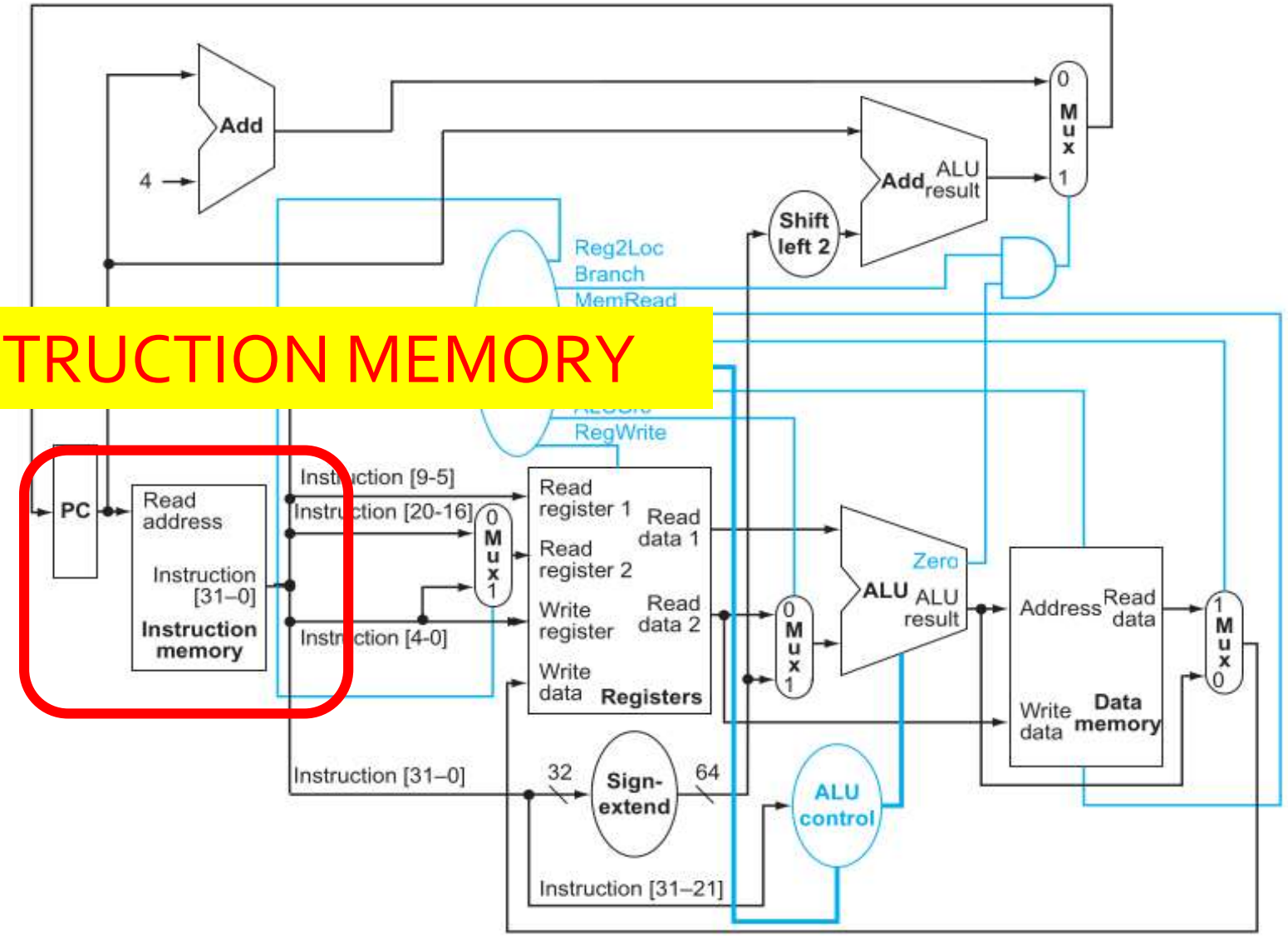
- 64-bit integer and IEEE-754 floating-point arithmetic
- Designed and optimized for pipelined execution

THE DATAPATH



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

THE DATAPAH

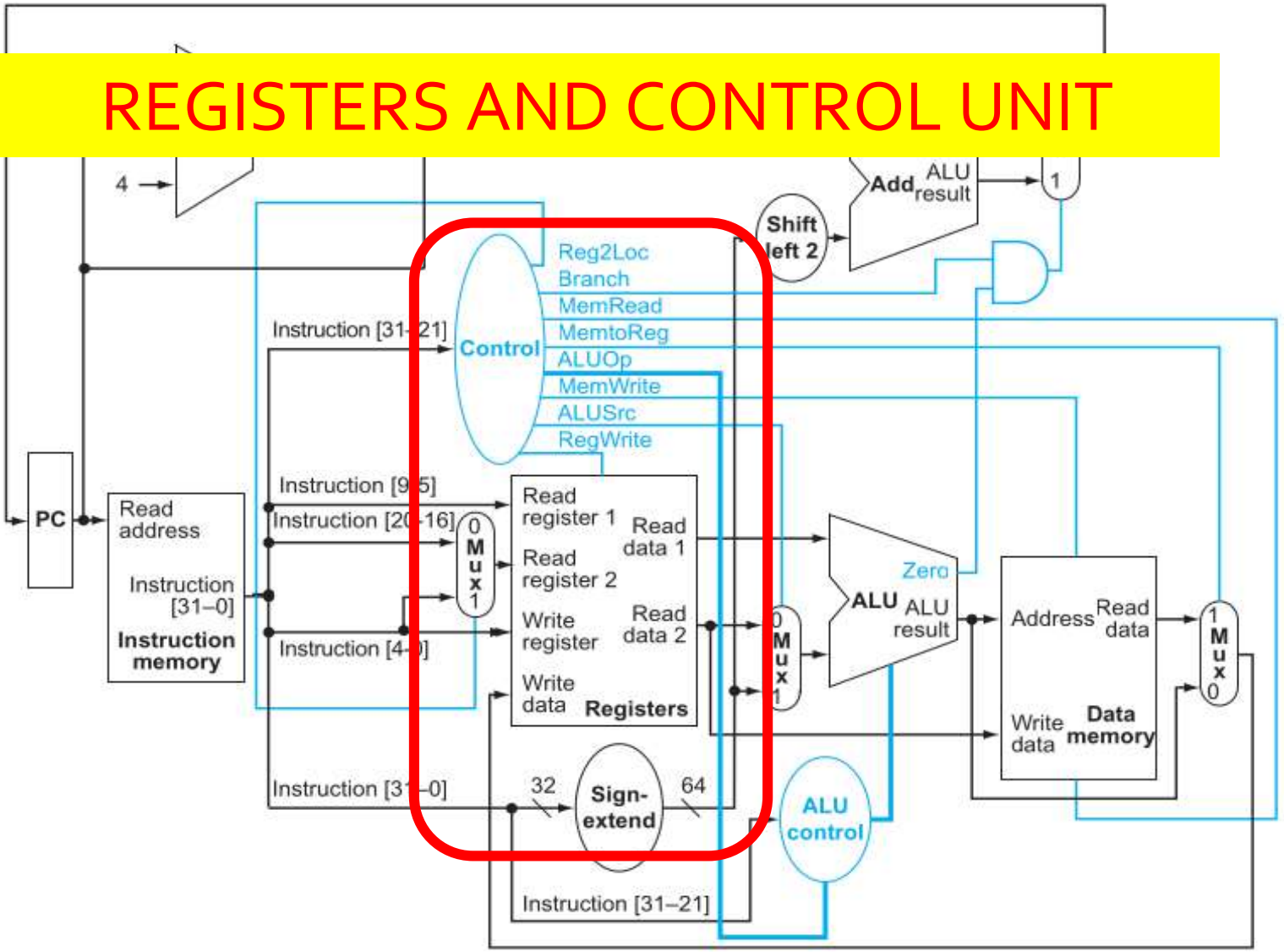


INSTRUCTION MEMORY

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

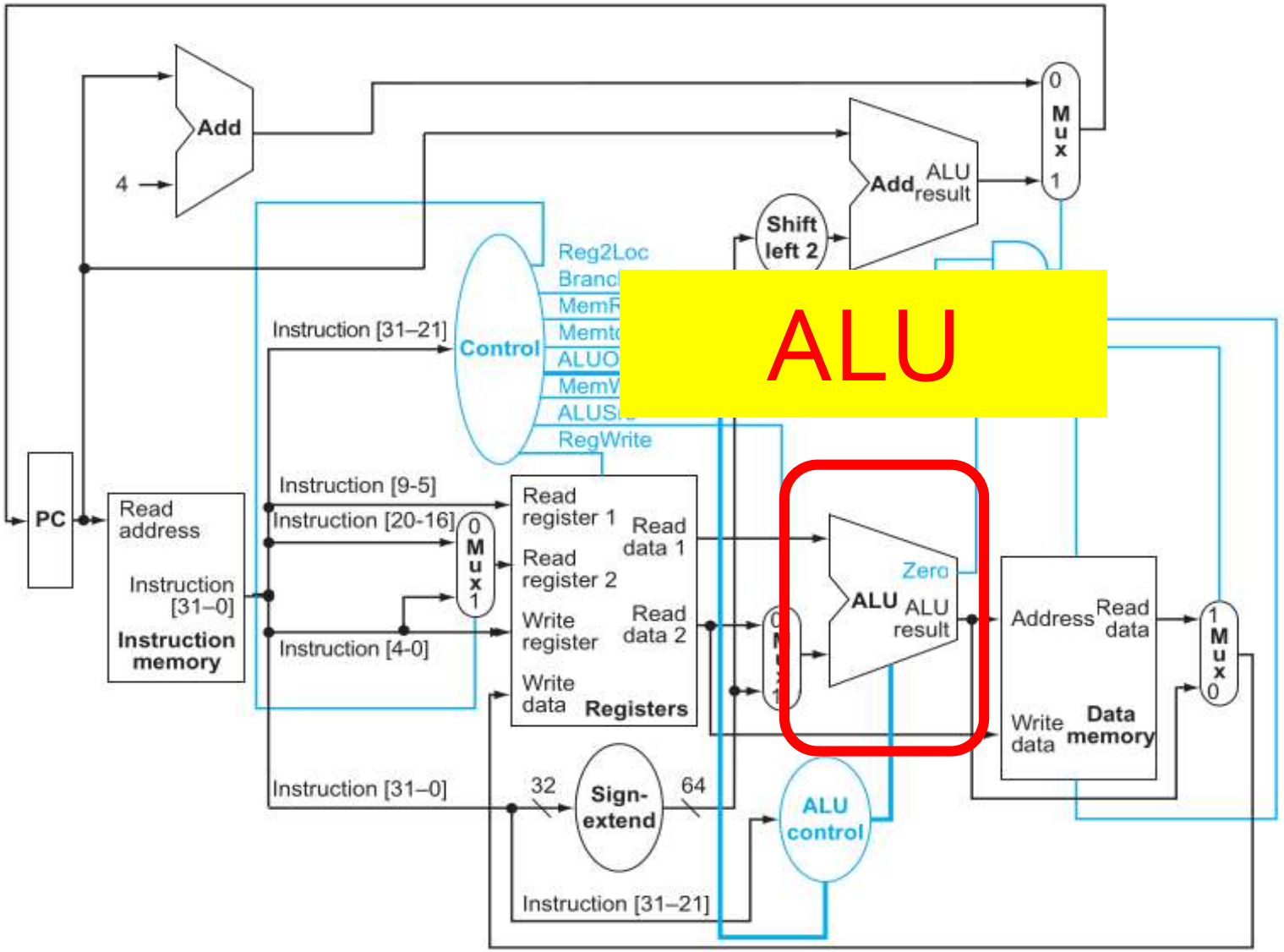
THE DATAPATH

REGISTERS AND CONTROL UNIT



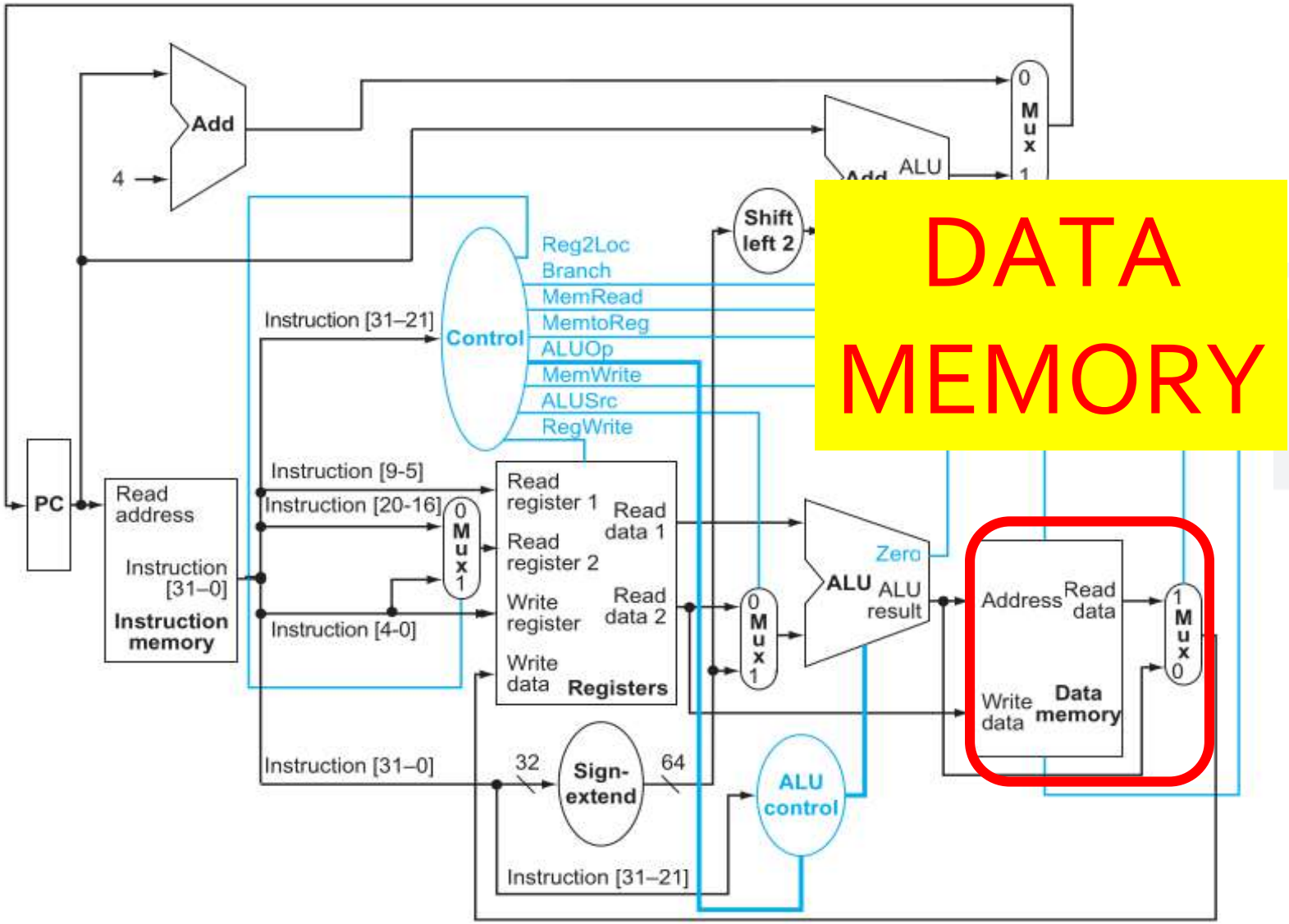
From Computer Organization and Design ARM Edition: The Hardware Software Interface - Patterson, D.A. and Hennessy, J.L.

THE DATAPATH



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

THE DATAPATH



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

WHAT SIMULATOR, AND WHY?

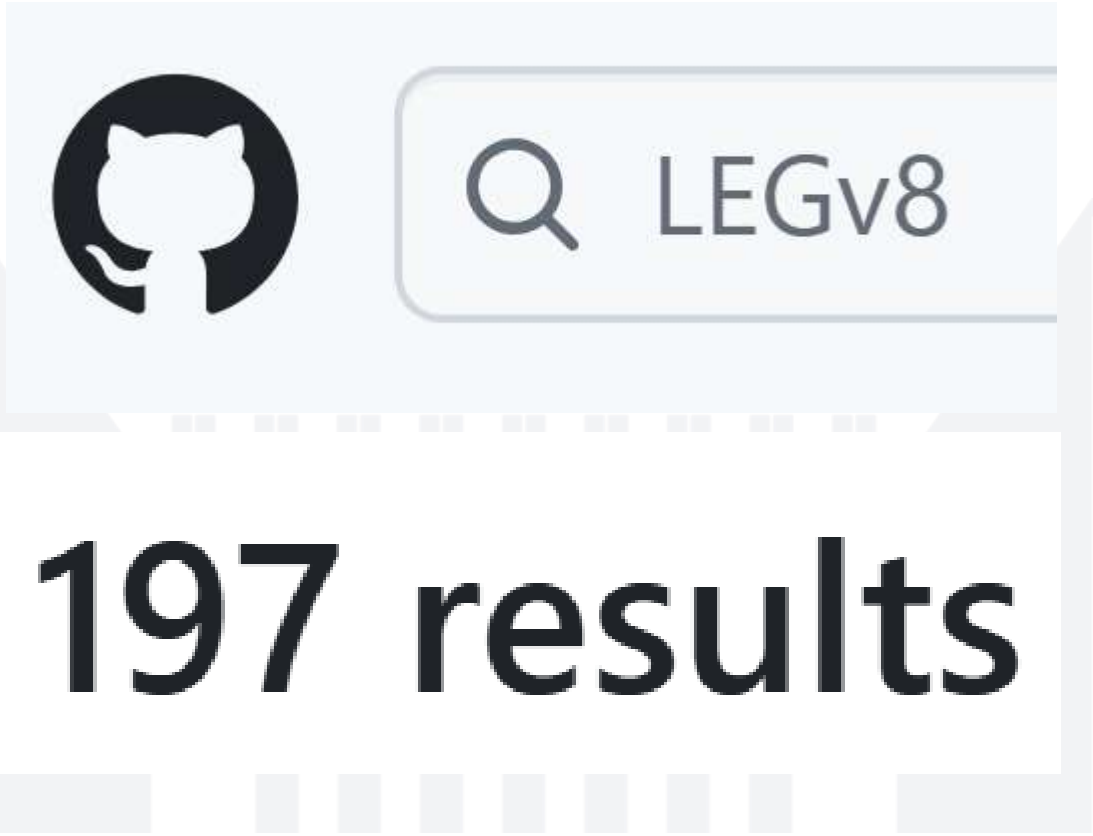
Restoration and development of

Arm's Java-based LEGv8 ISA simulator

NO HARDWARE FOR LEGv8 => NEED A SIMULATOR

Michael H. („Laserlicht“) / Wikimedia Commons

BUT WHICH ONE?



THE PROBLEM:

NO SOFTWARE CAN YET SIMULATE THE ENTIRE LEGv8 ISA!

THE SOLUTION:

- Write one from scratch

OR (BETTER)

- Improve one that already exists

ARM HAS OFFICIALLY MADE A LEGv8 SIMULATOR

GOOD!

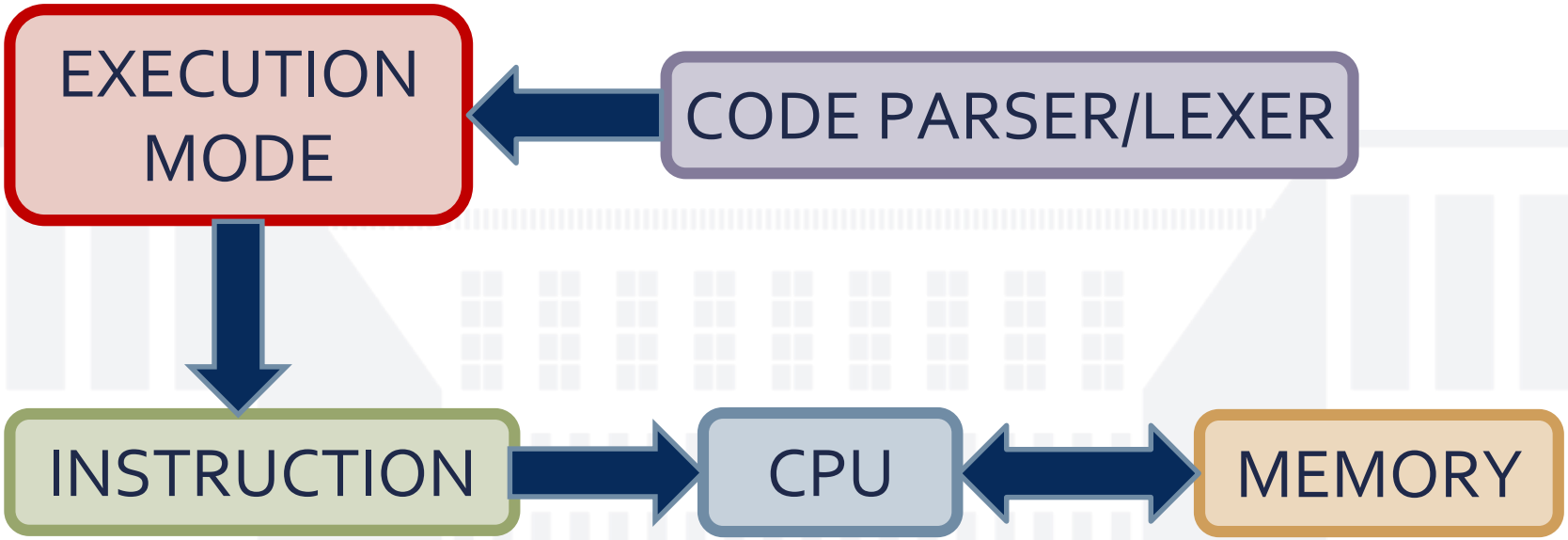
IT'S INCOMPLETE AND BROKEN

BAD!

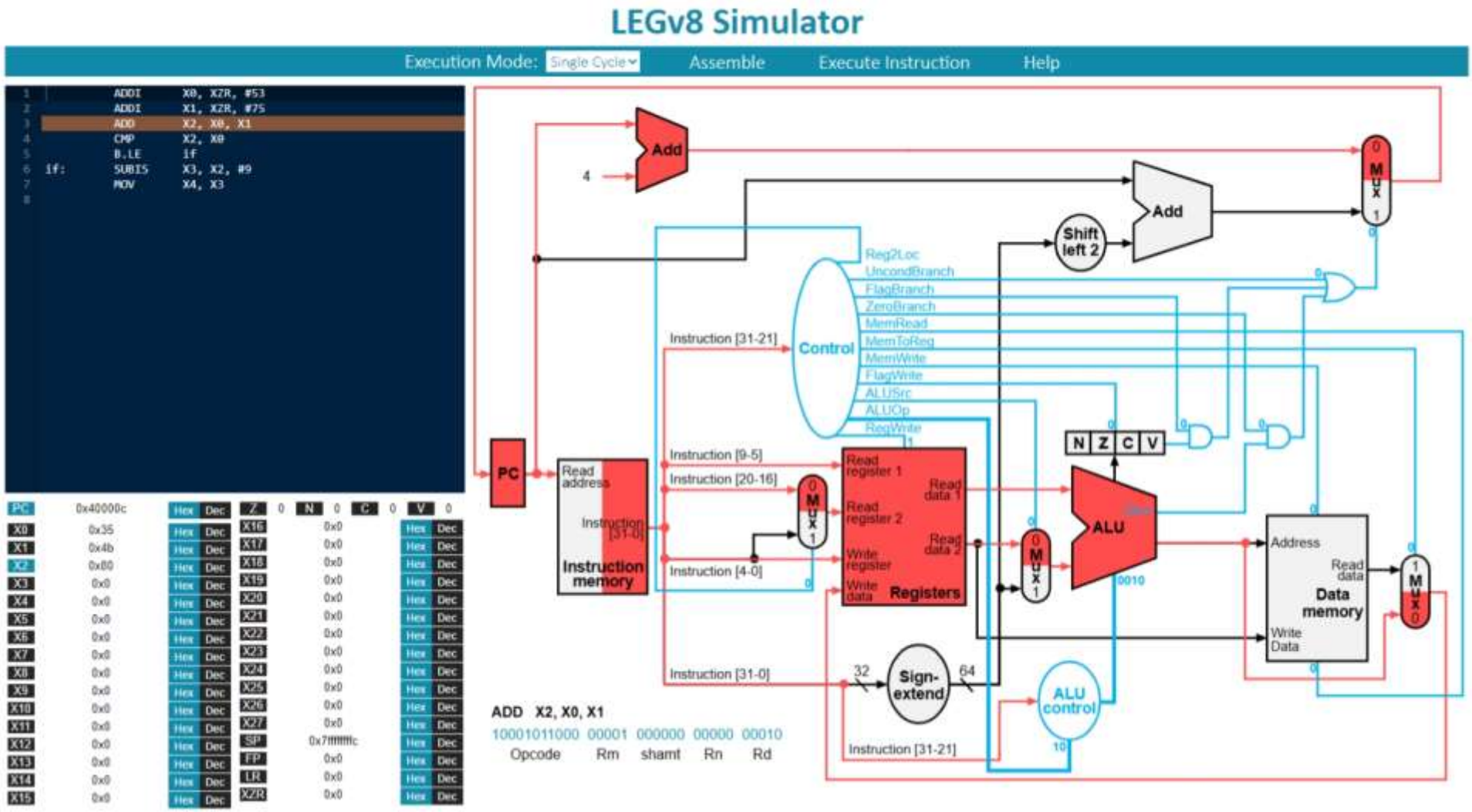
WHAT STANDS OUT:

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

STRUCTURE OF THE SIMULATOR



WHAT SIMULATOR, AND WHY?



UNIVERSITÀ DEGLI STUDI DI TRIESTE

FIXING AND RESTORING THE SIMULATOR

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

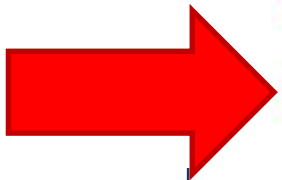
COMPARISONS DON'T WORK!

- 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

UNIVERSITÀ DEGLI STUDI DI TRIESTE

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	

THE FLAGS ARE SET WRONG!

HOW THEY SHOULD BE

≤	B.LE	~(Z=0 & N=V)

HOW THEY ARE

Z	0	N	1	C	0	V	1

¬(Z=0 ∧ N=V) = ¬(TRUE ∧ TRUE) = FALSE

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
```

PROGRAM COUNTER: 0x0

RETURN ADDRESS: 0x0

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
```

PROGRAM COUNTER: 0x4

RETURN ADDRESS: 0x0

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
```

PROGRAM COUNTER: 0xC

RETURN ADDRESS: 0xC

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
```

PROGRAM COUNTER: 0x10

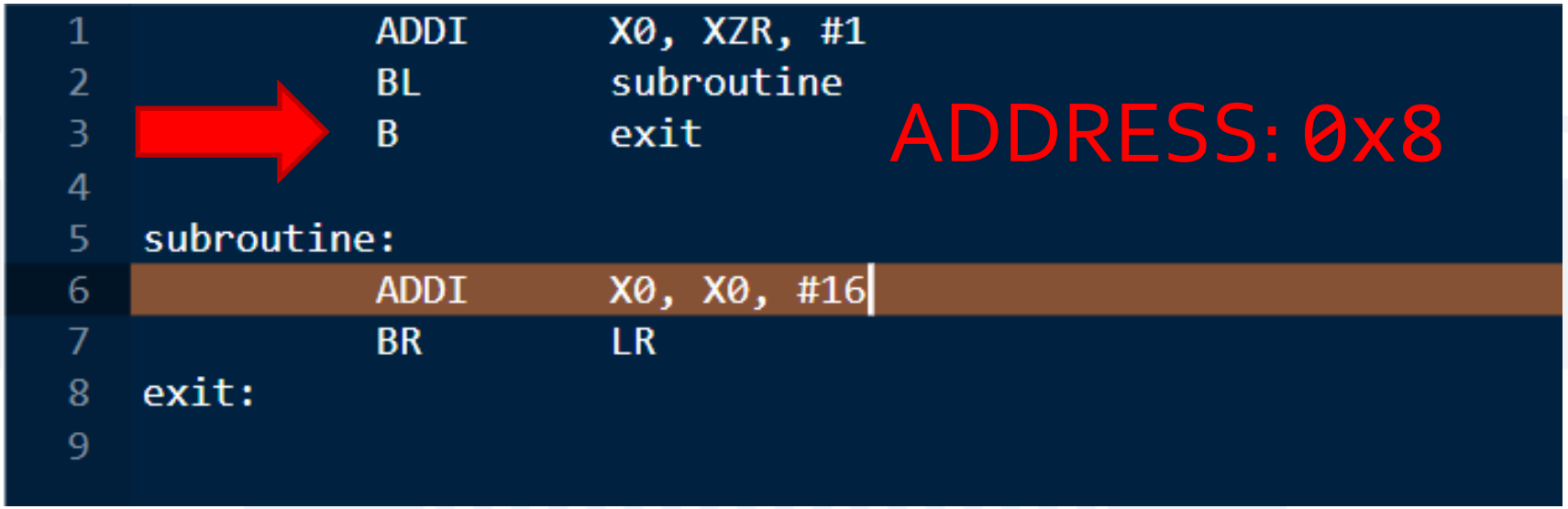
RETURN ADDRESS: 0xC

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
```

IT DOESN'T GO BACK!

IT SHOULD GO HERE!



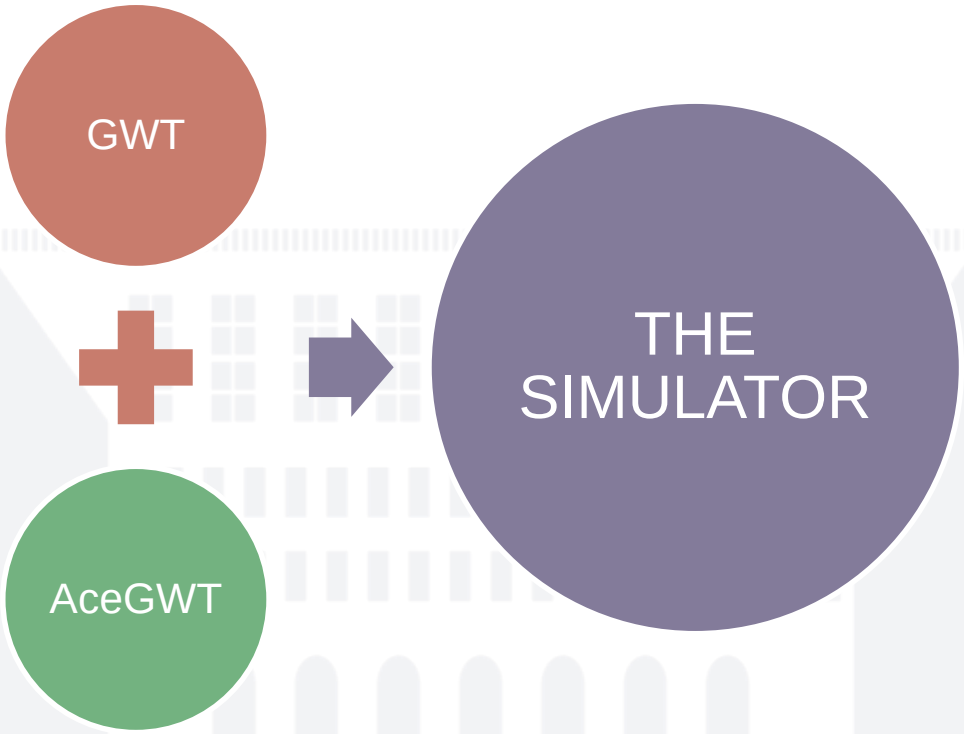
PROGRAM COUNTER: 0xC

RETURN ADDRESS: 0xC

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:

- Need old version of Eclipse, and Eclipse GWT plug-in
- Reverse engineer the dependencies and where they are needed
- Configure the project to stop failing

FILLING THE GAPS

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

WHAT IS THE SIMULATOR MISSING?

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

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

UMULH: A CASE STUDY

- Takes two 64-bit unsigned integer values
- Extends them to 128 bits unsigned
- Performs 128-bit product
- Saves higher 64 bits to destination

PROBLEM 1

- Java does not have primitive 128-bit integer types
- Product of 64-bit integers truncated
- The BigInteger library exists
- GWT 2.7 doesn't emulate it (2.8 does)

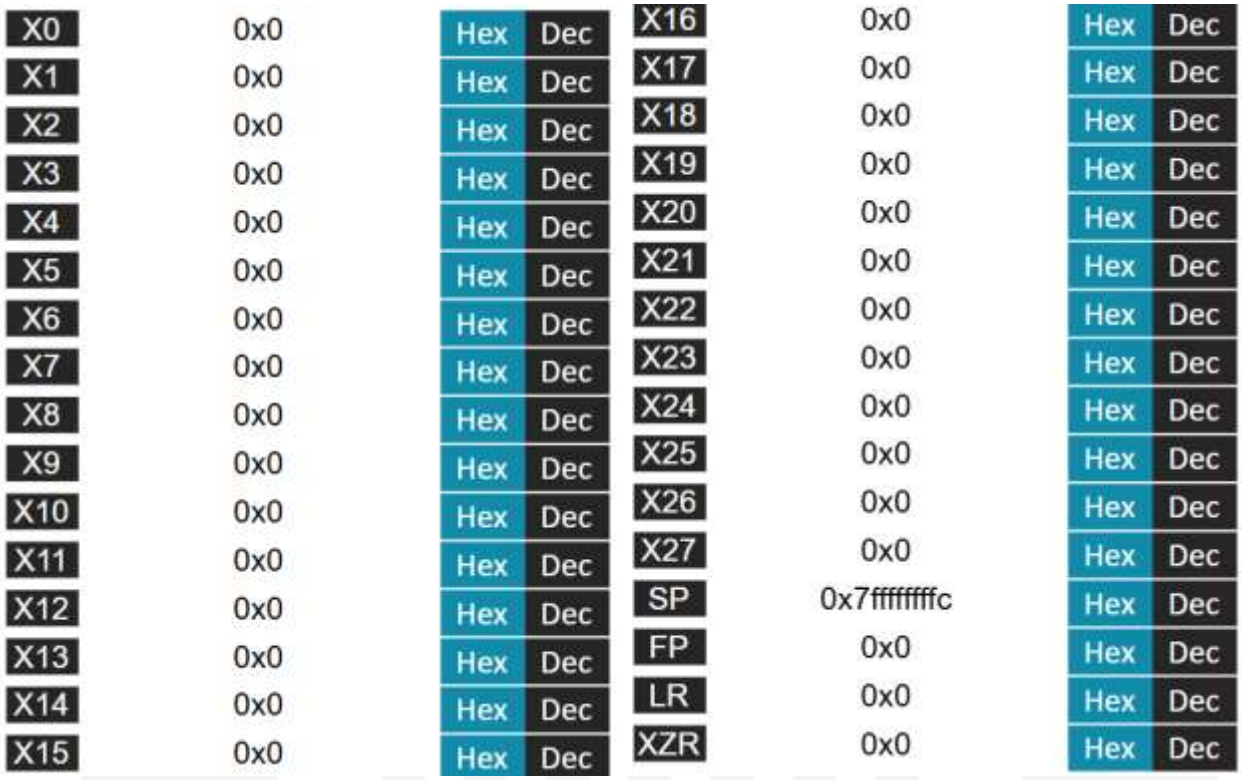
PROBLEM 2

- Primitive integers are signed
- BigInteger also signed
- Bitmask converts 64-bit unsigned integers to 65-bit signed, perform signed multiplication, take the higher bits

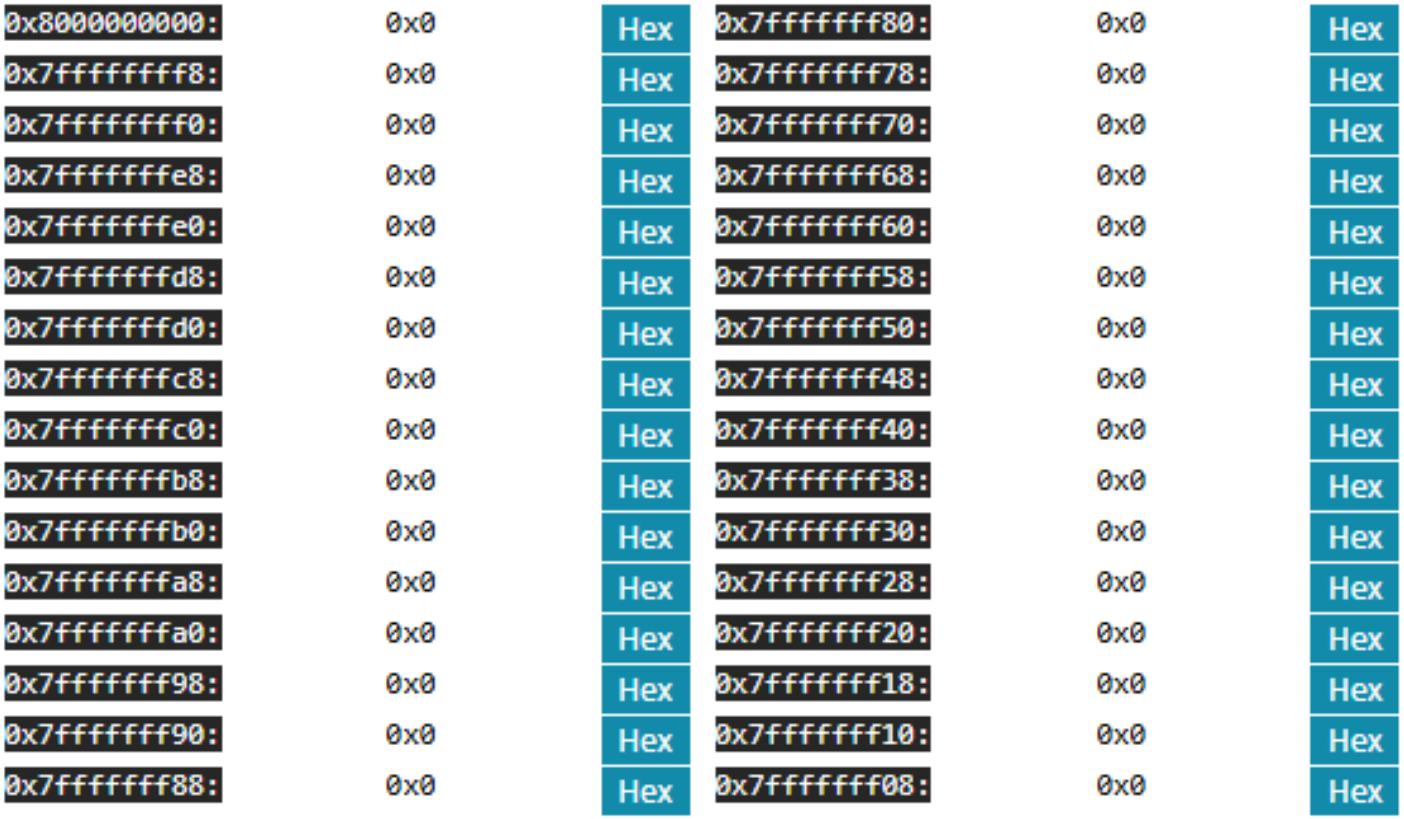
CAN'T SEE THE STACK

- Fundamental for testing and debugging complex programs (now we can write them)
- Useful to understanding LEGv8 and stack management
- Visible in most simulators

THE INTEGER REGISTERS VIEW



THE NEW STACK VIEW



ADDING FLOATING-POINT SUPPORT

- FADDS, FADDD – ADD TWO IEEE-754 VALUES
- FSUBS, FSUBD - SUBTRACT TWO IEEE-754 VALUES
- FMULS, FMULD - MULTPLY 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, FCMPD – COMPARE TWO IEEE-754 VALUES

ARITHMETICAL INSTRUCTIONS (FADDD, FDIVS, …)

- Native Java support for IEEE-754 with *float* and *double* types
- Native Java support for IEEE-754 arithmetical operations
- Straight forward implementation

MEMORY ACCESS INSTRUCTIONS (LDURS, STURD, …)

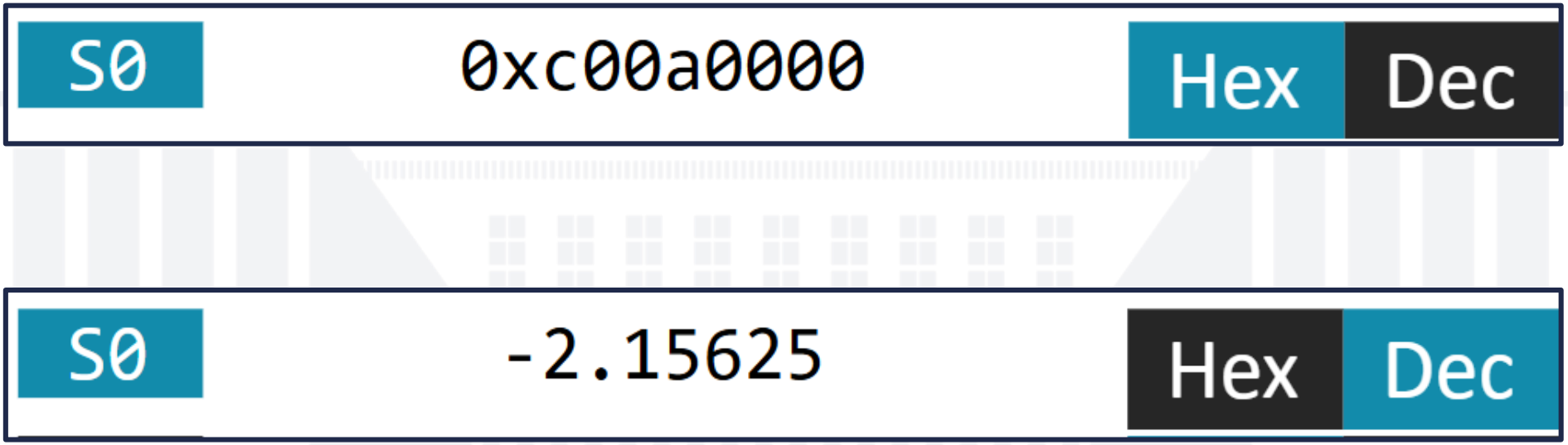
- Simulator uses *long* values to store bits in memory
- Use exising *longs* and *int*s as raw bits
- Use Java *Double.longBitsToDouble* and *Double.doubleToLongBits* to convert before memory

COMPARISON INSTRUCTIONS (FCMPS, FCMPD)

- 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 *(At least one argument was NaN.)*	0	0	1	1

SHOWING THE REGISTERS



THE FINAL VIEW

THE CHERRY ON TOP: MODERNIZING THE BUILD SYSTEM

INTEGRATING MAVEN

- Latest GWT and AceGWT support Maven
- Integrated Maven into the simulator
- Can now use other IDEs, Java 21, GWT 2.11
- To develop, download the code and run *mvn package*. That's it.

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

THANK YOU FOR YOUR ATTENTION

THESIS AVAILABLE HERE

SIMULATOR AVAILABLE HERE