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, 28 August 2024

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

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

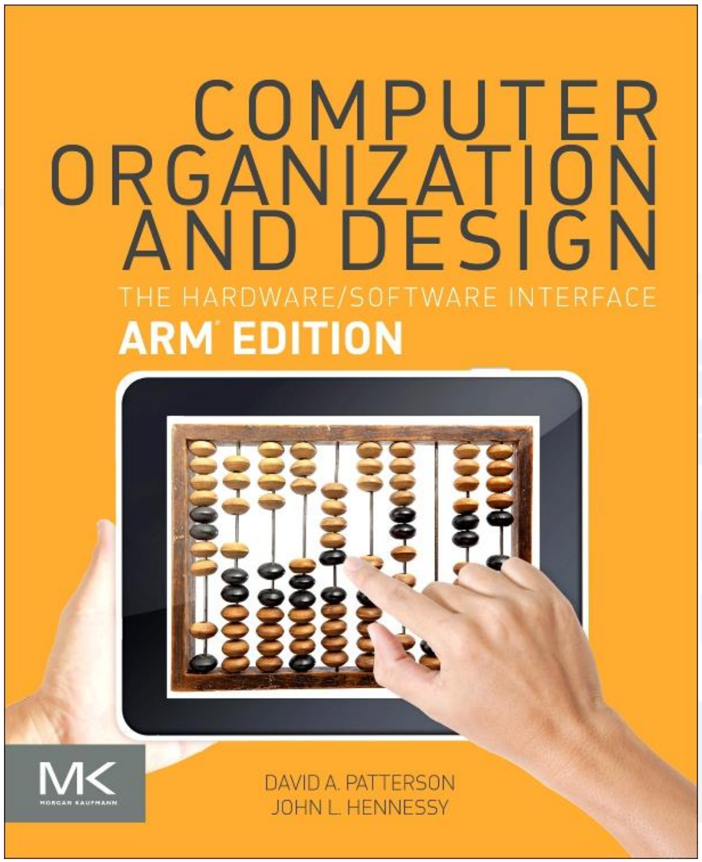
Restoration and development of Arm's Java-based LEGv8 ISA simulator

UNIVERSITÀ DEGLI STUDI DI TRIESTE

WHAT IS LEGv8?

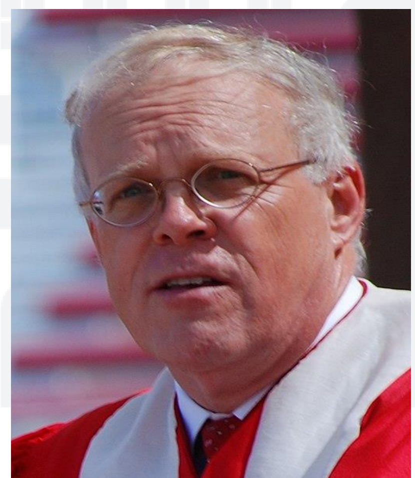
Restoration and development of Arm's Java-based LEGv8 ISA simulator

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

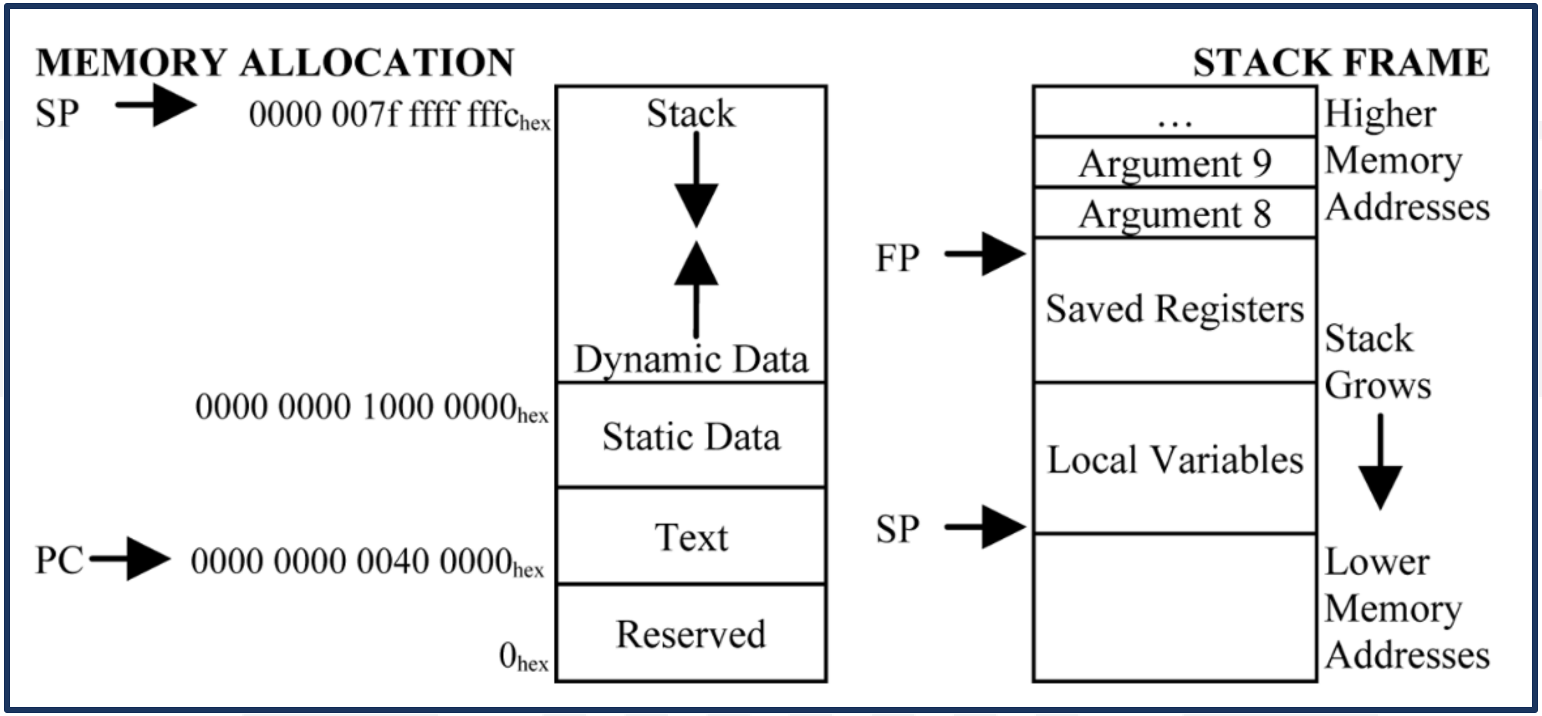


John L. Hennessy

WHAT IS LEGv8?

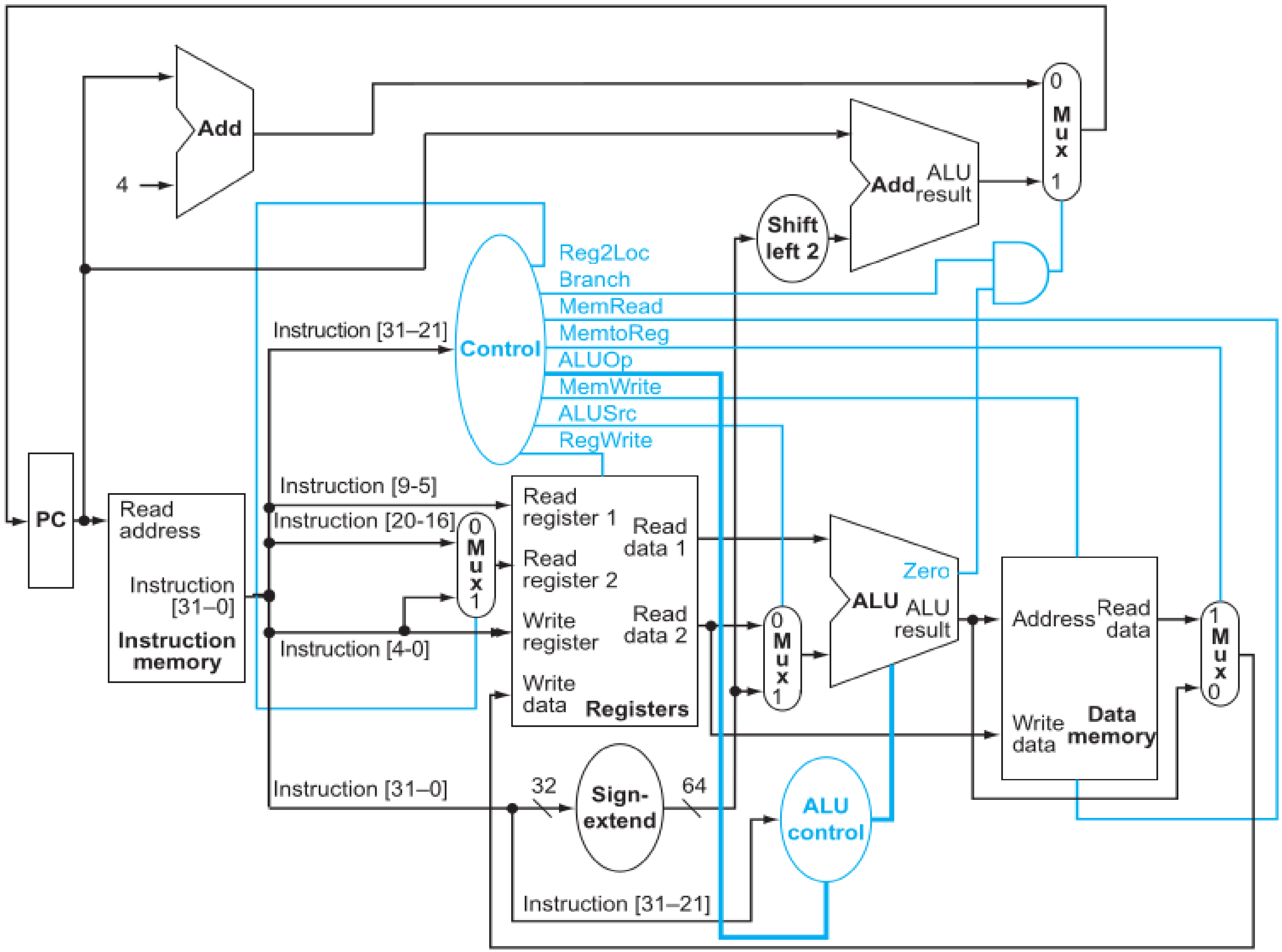
- As simple as it can be…
- … but with a modern design
- Heavily inspired by ARMv8, almost a "subset"

- 64-BIT addressed memory, Harvard model



- 32 64-bit "X" integer registers
- 32 64-bit "D" floating-point registers
- 32 32-bit "S" floating-point "registers"
- 64-bit integer and IEEE-754 floating-point arithmetic
- Designed and optimized for pipelined execution

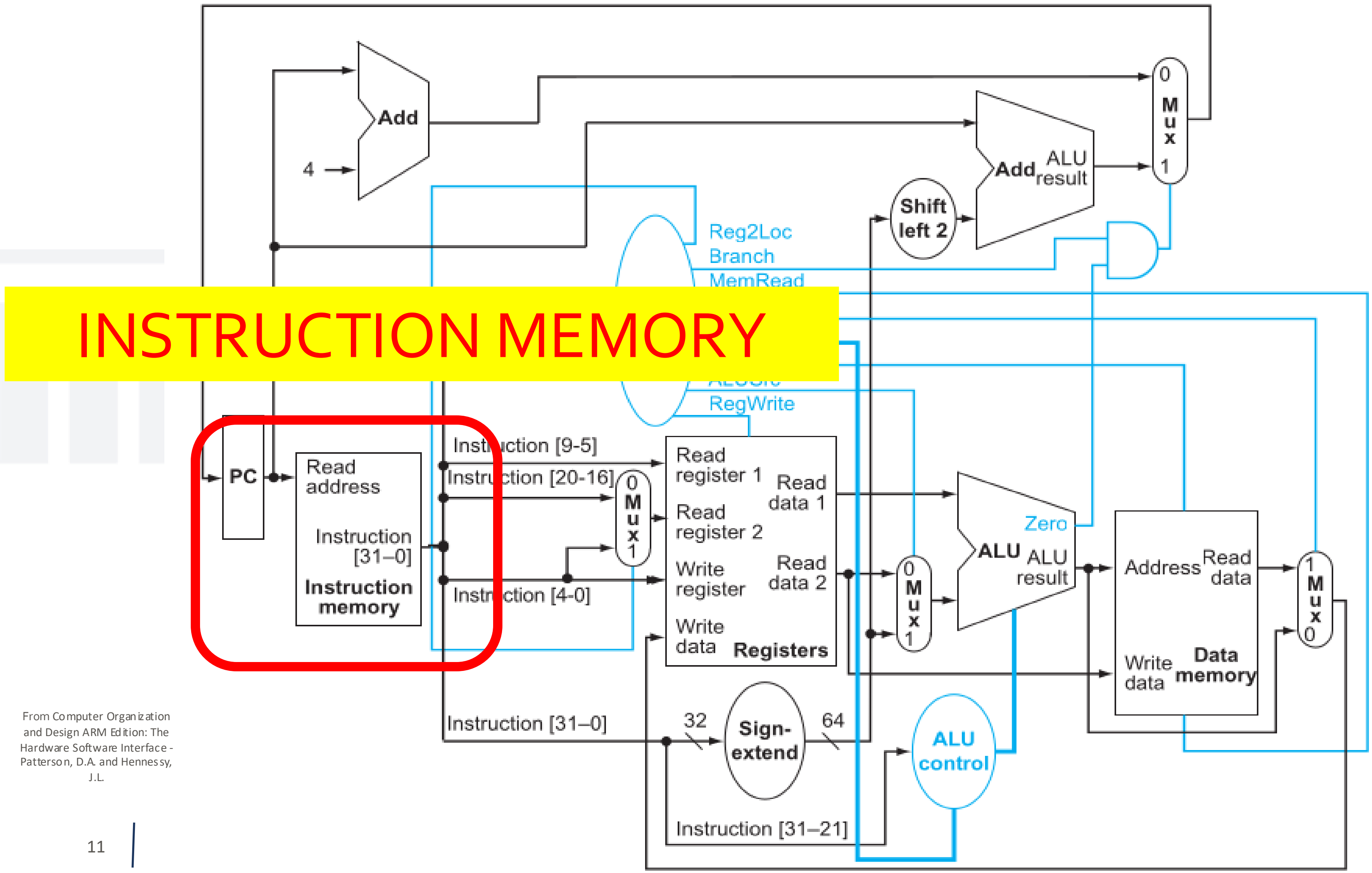
THE DATAPATH



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

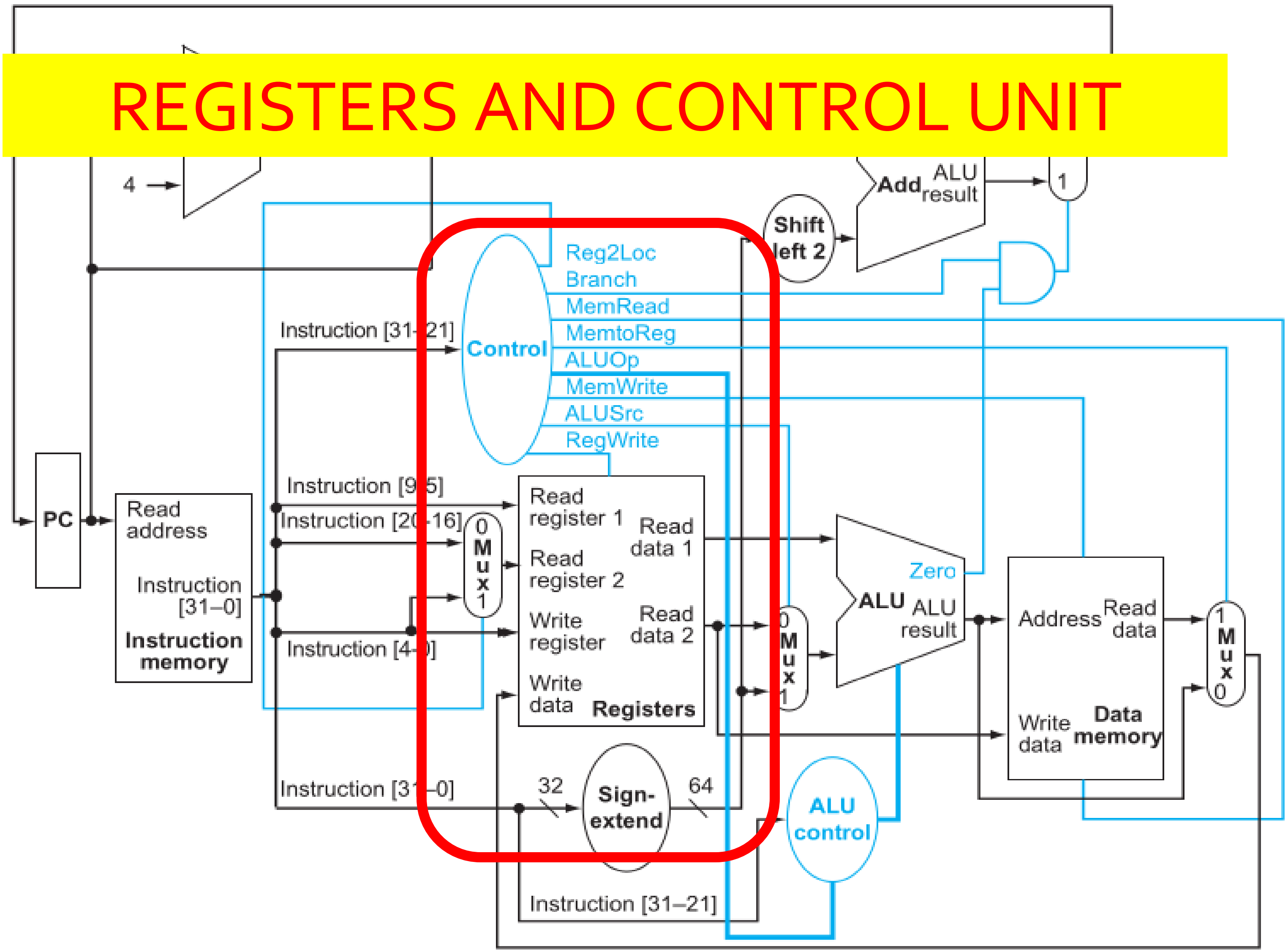
WHAT IS LEGv8?

THE DATAPAH



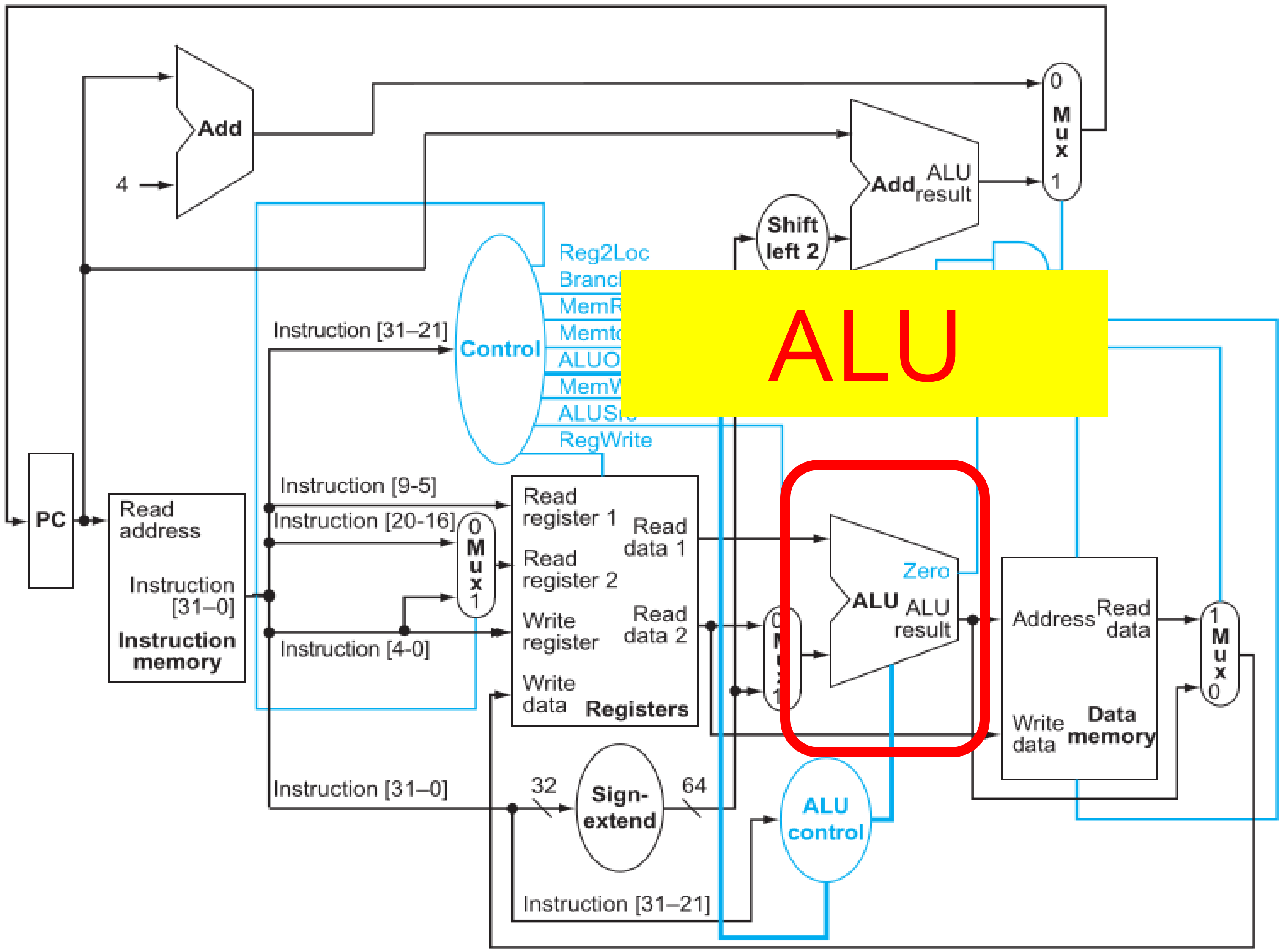
From Computer Organization and Design ARM Edition: The Hardware Software Interface - Patterson, D.A. and Hennessy, J.L.

THE DATAPATH



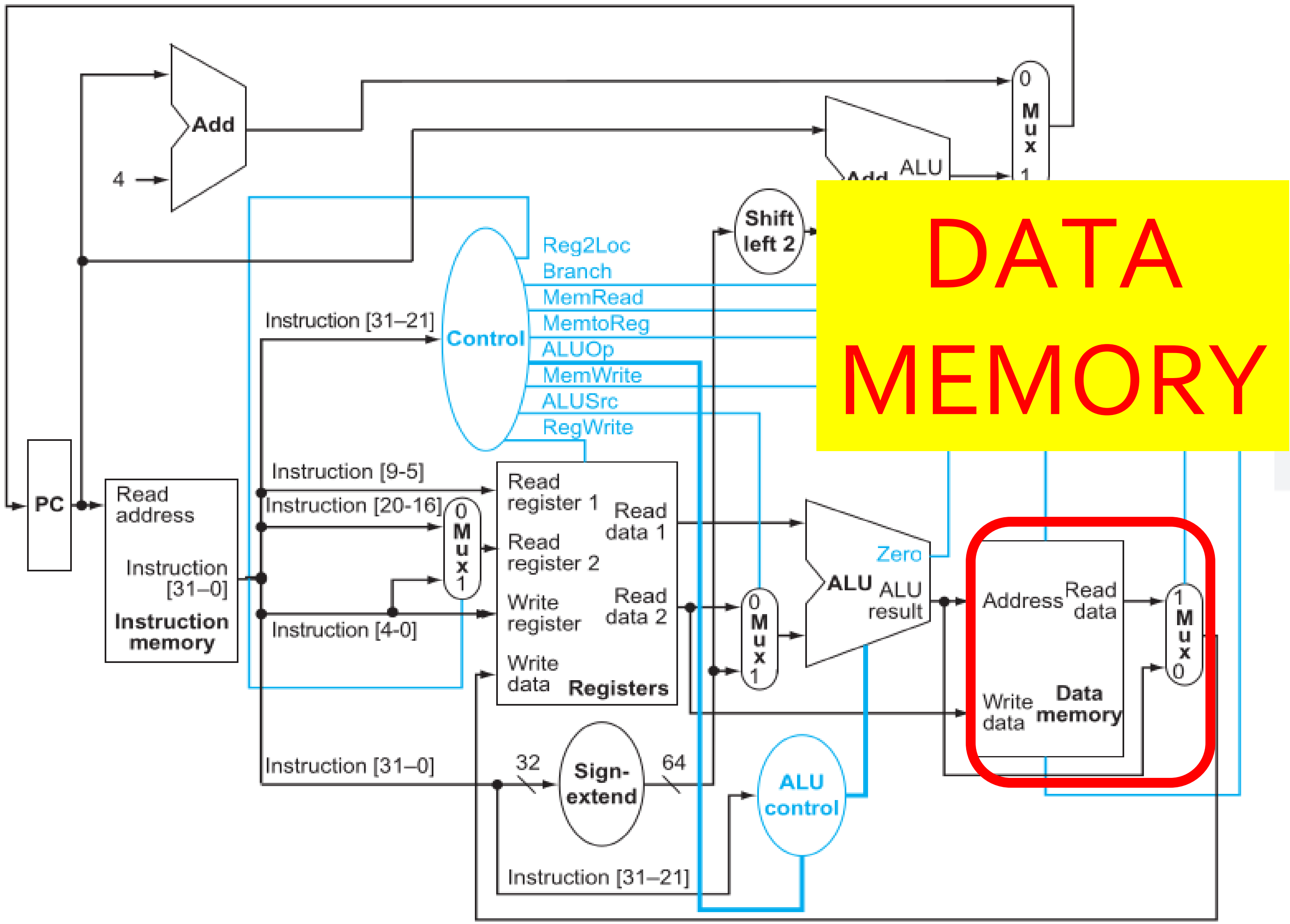
From Computer Organization and Design ARM Edition: The Hardware Software Interface - Patterson, D.A. and Hennessy, J.L.

THE DATAPATH



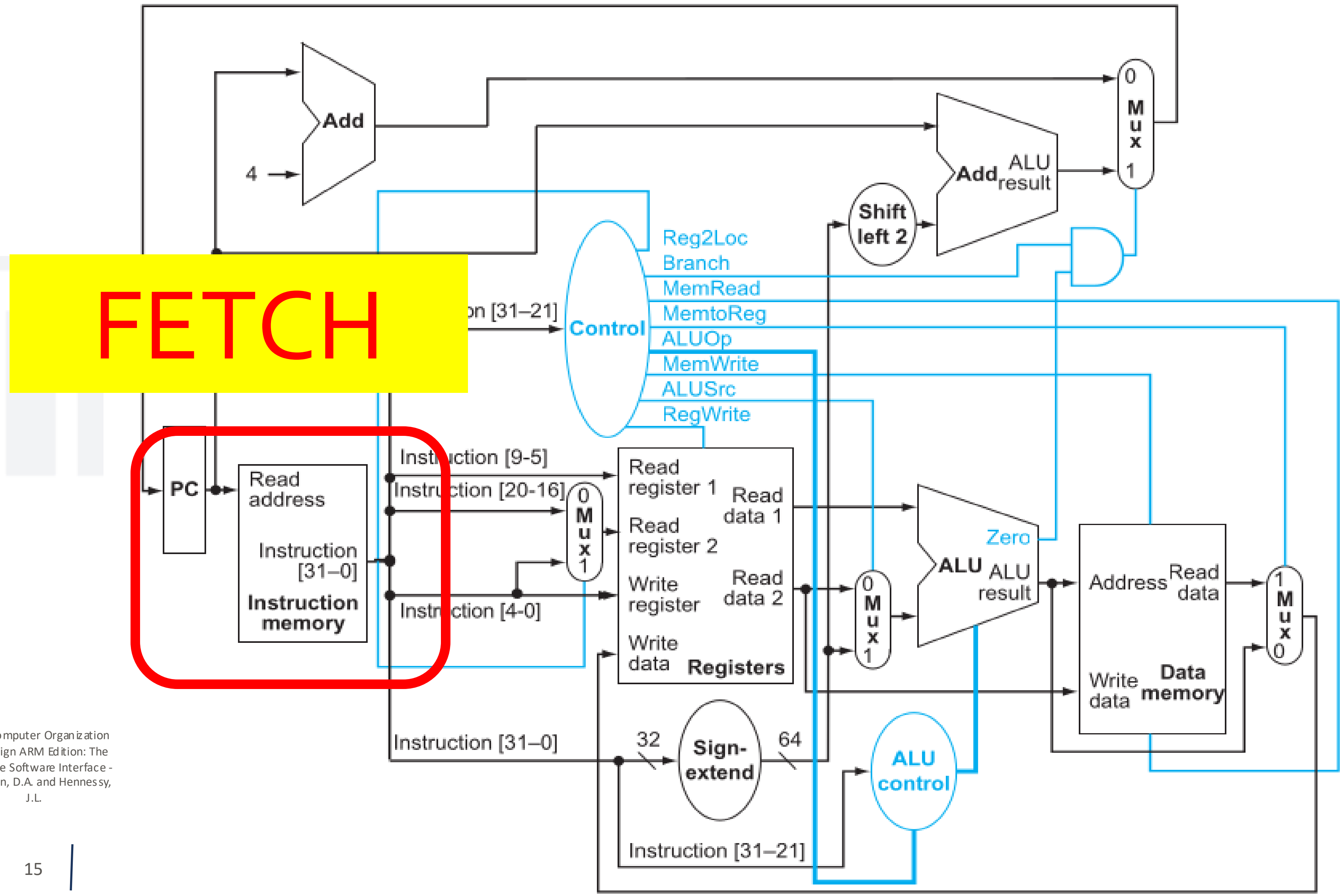
From Computer Organization and Design ARM Edition: The Hardware Software Interface - Patterson, D.A. and Hennessy, J.L.

THE DATAPATH



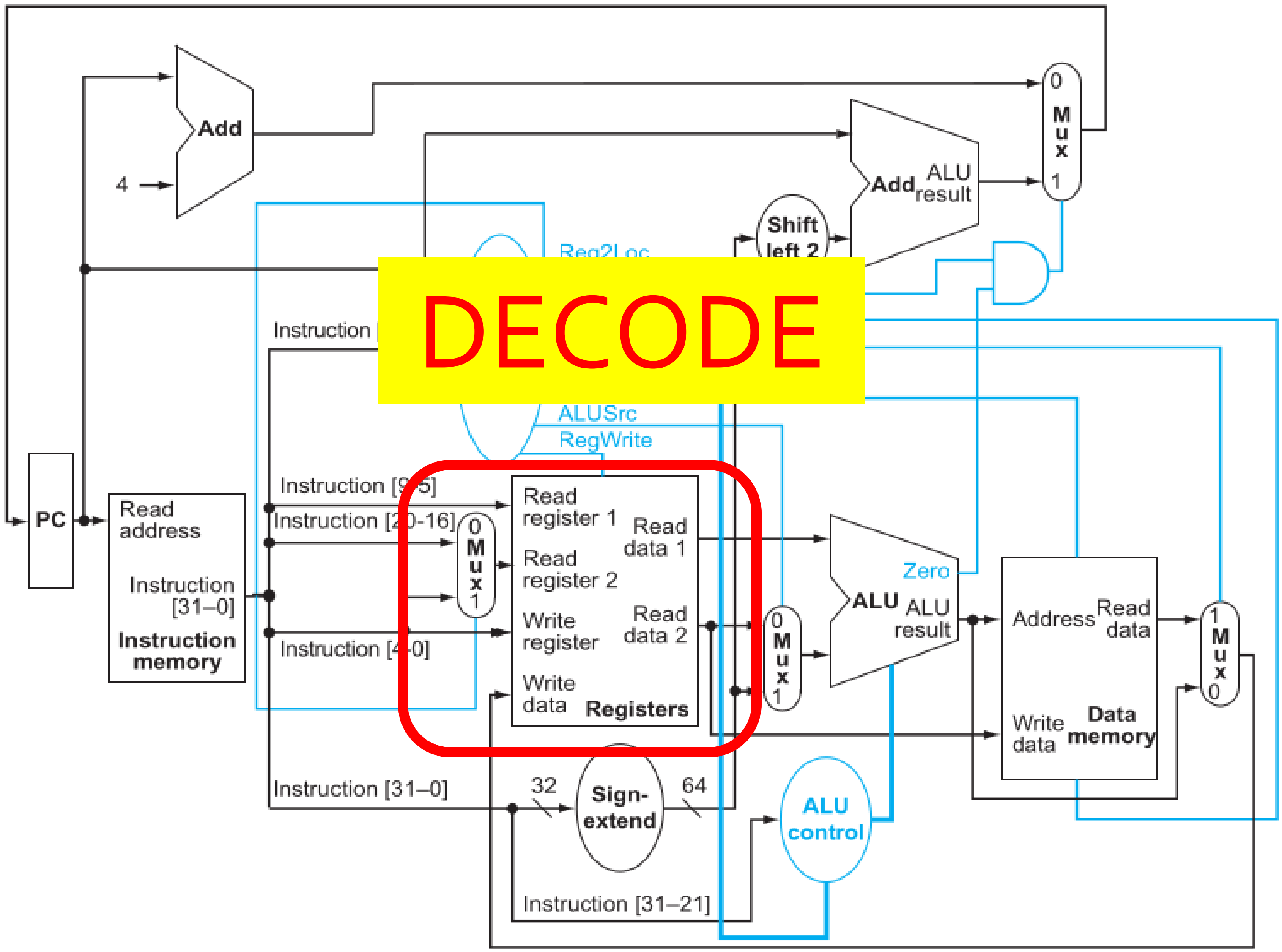
From Computer Organization and Design ARM Edition: The Hardware Software Interface - Patterson, D.A. and Hennessy, J.L.

THE STAGES



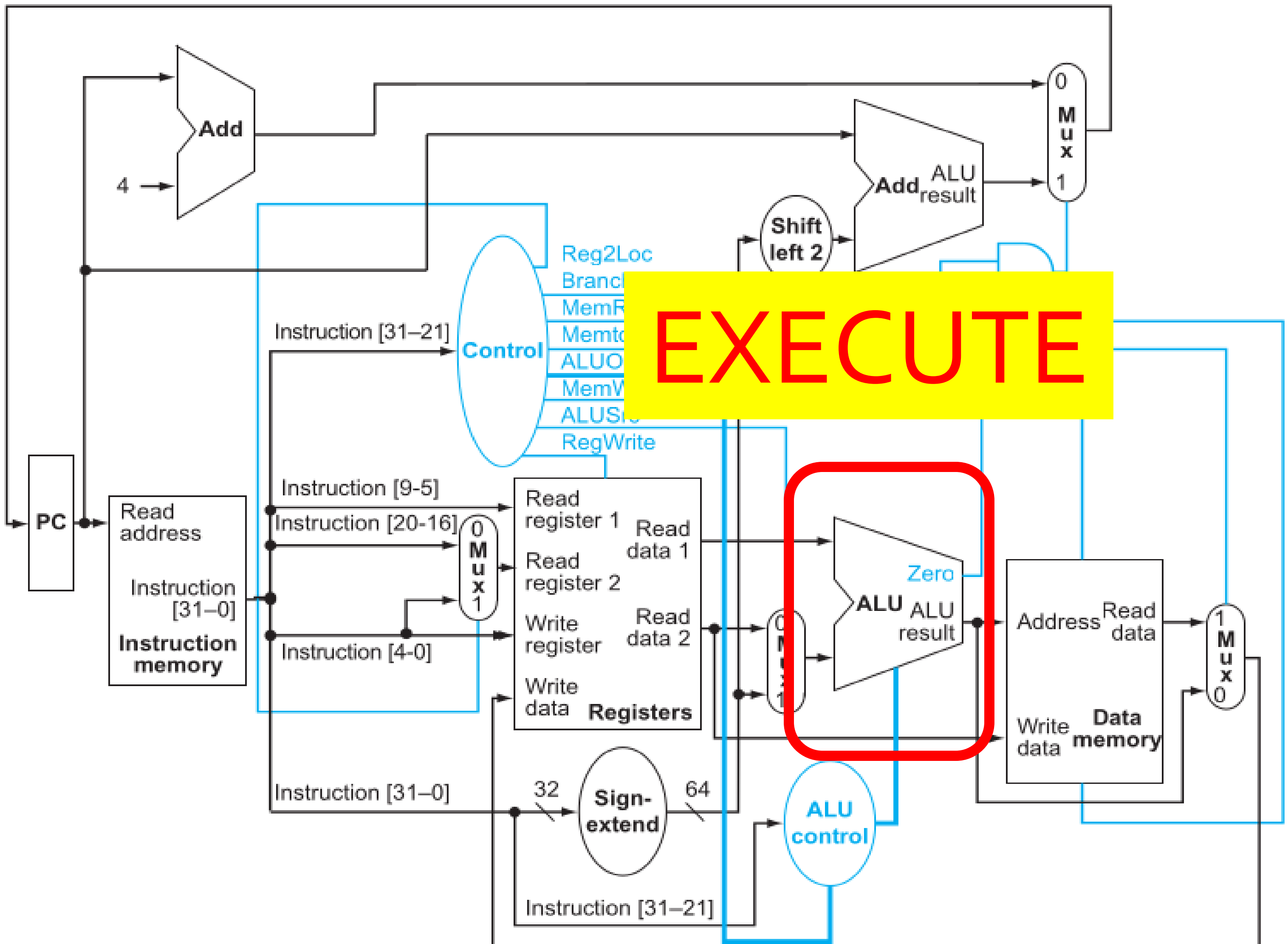
From Computer Organization and Design ARM Edition: The Hardware Software Interface - Patterson, D.A. and Hennessy, J.L.

THE STAGES



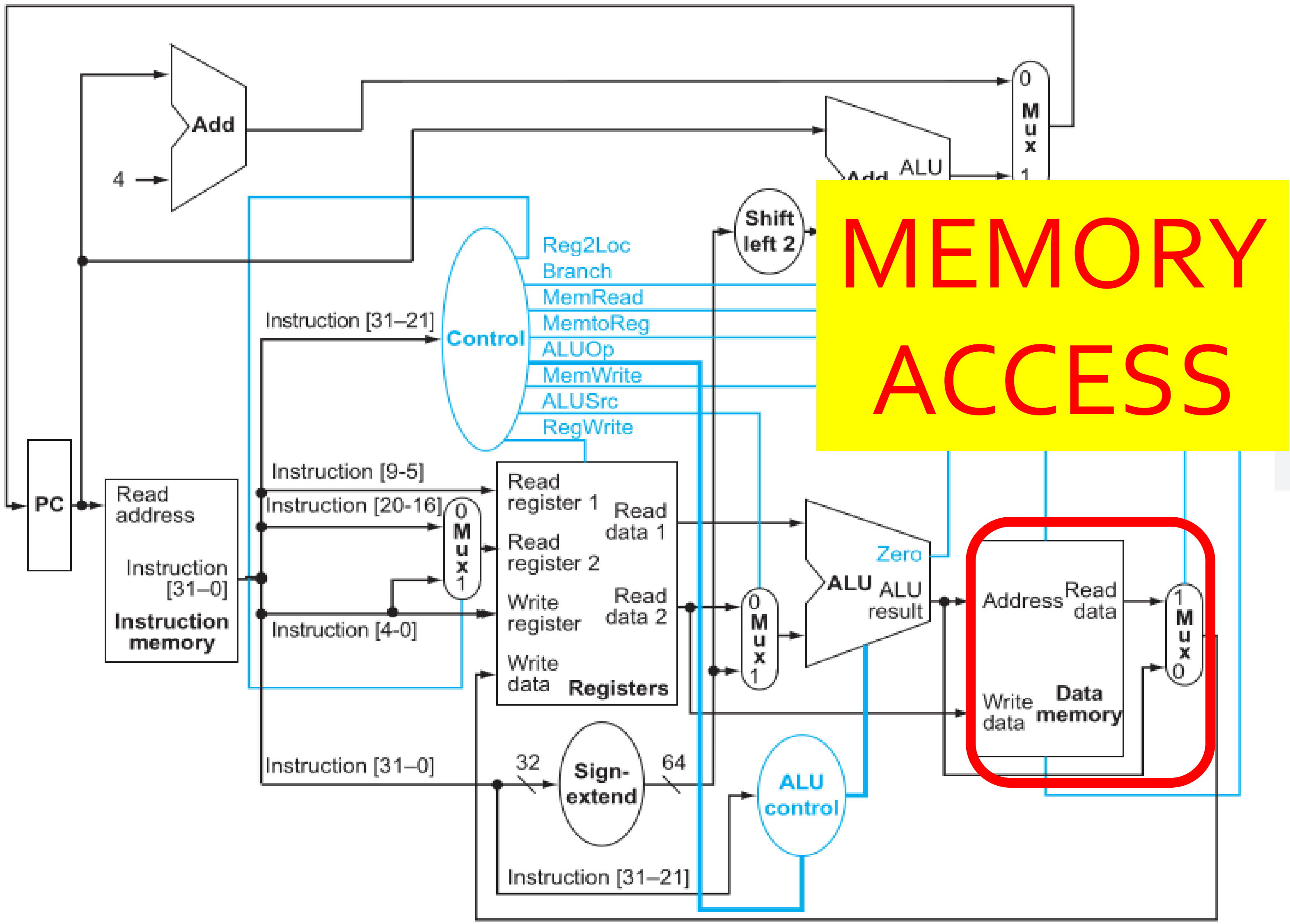
From Computer Organization and Design ARM Edition: The Hardware Software Interface - Patterson, D.A. and Hennessy, J.L.

THE STAGES



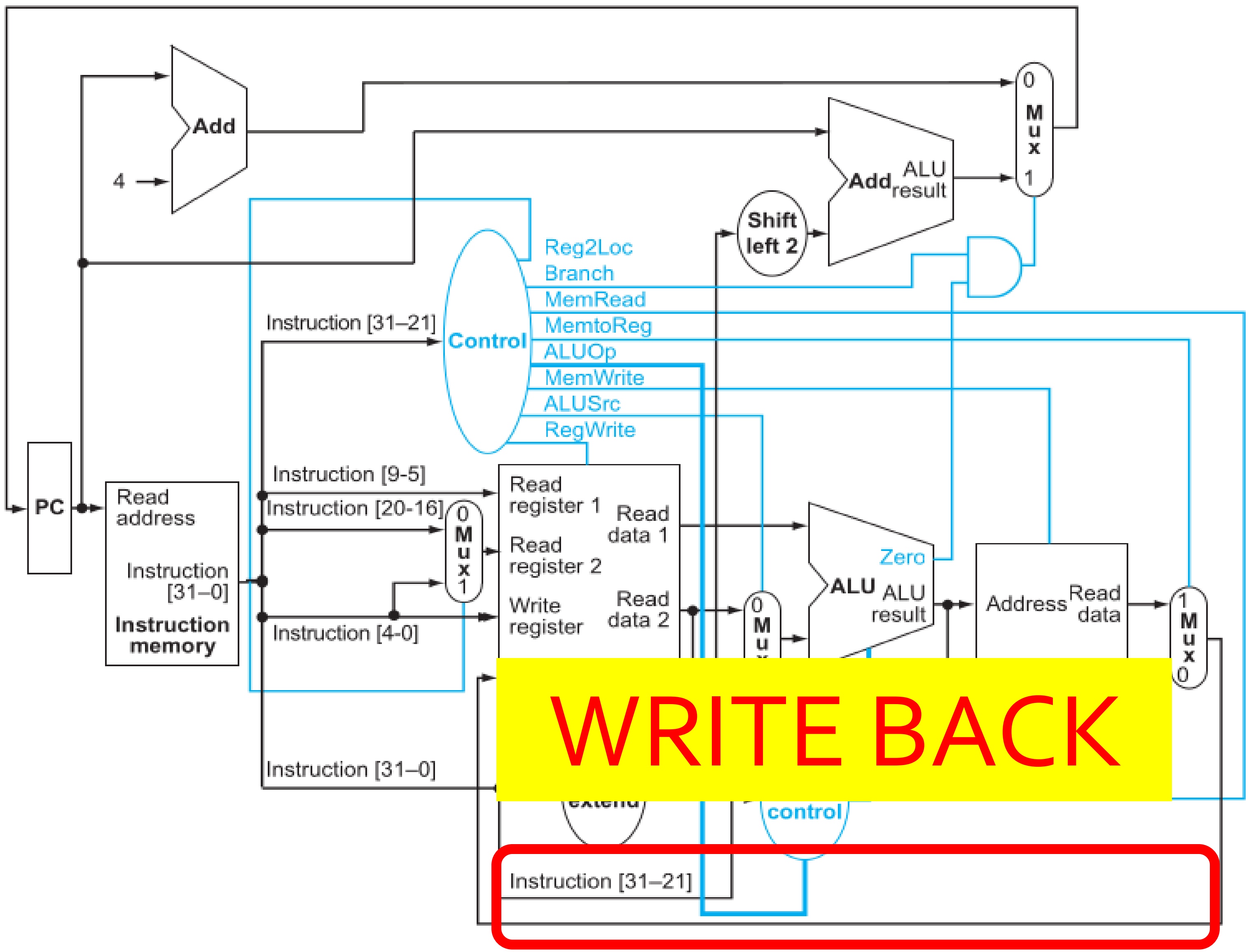
From Computer Organization and Design ARM Edition: The Hardware Software Interface - Patterson, D.A. and Hennessy, J.L.

THE STAGES



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

THE STAGES

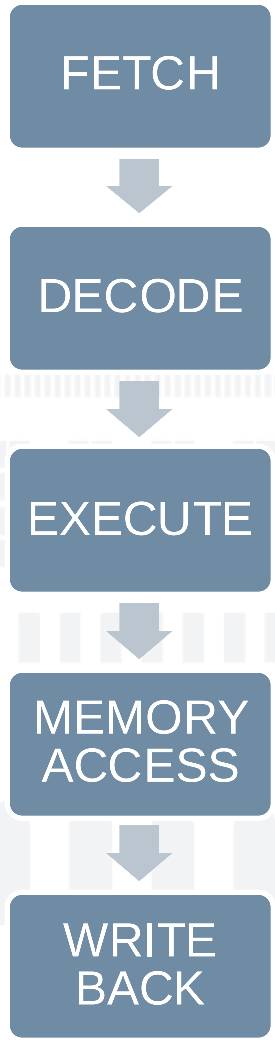


From Computer Organization and Design ARM Edition: The Hardware Software Interface - Patterson, D.A. and Hennessy, J.L.

THE SINGLE CYCLE DATAPATH

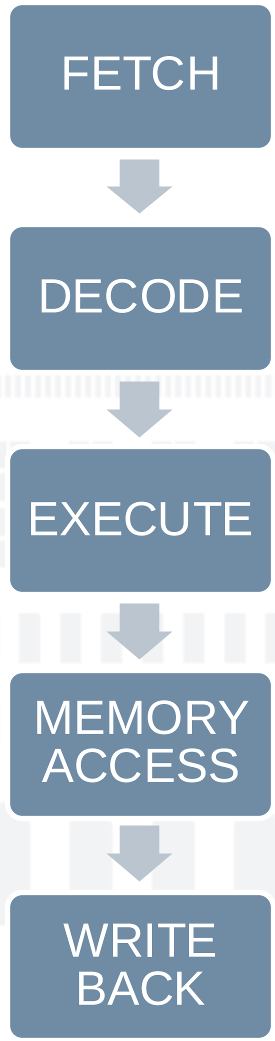
INSTRUCTION 1

THE SINGLE CYCLE DATAPATH



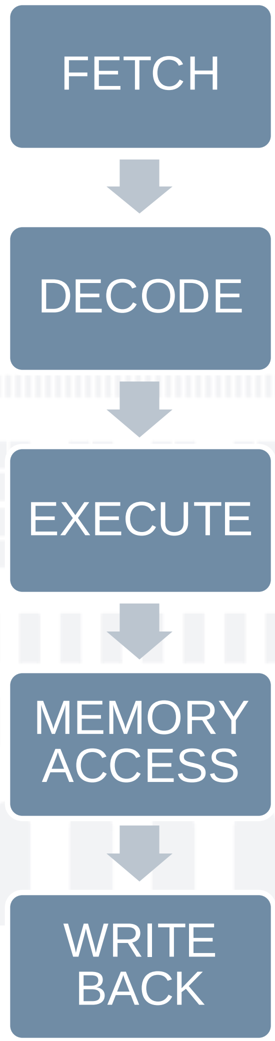
INSTRUCTION 1

THE SINGLE CYCLE DATAPATH



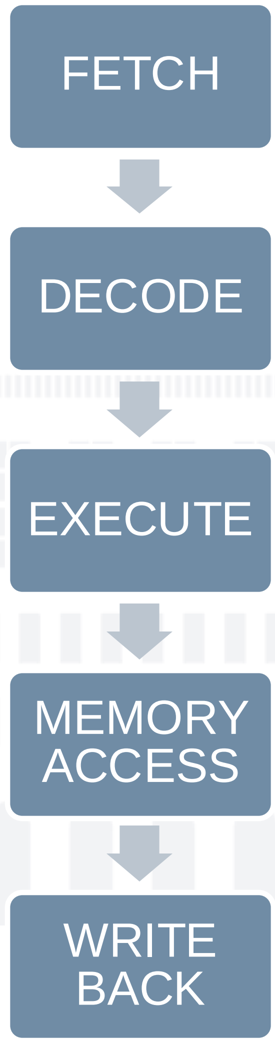
INSTRUCTION 1

THE SINGLE CYCLE DATAPATH



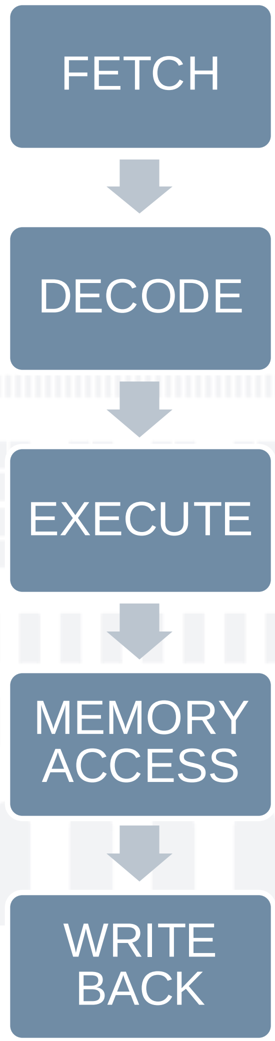
INSTRUCTION 1

THE SINGLE CYCLE DATAPATH



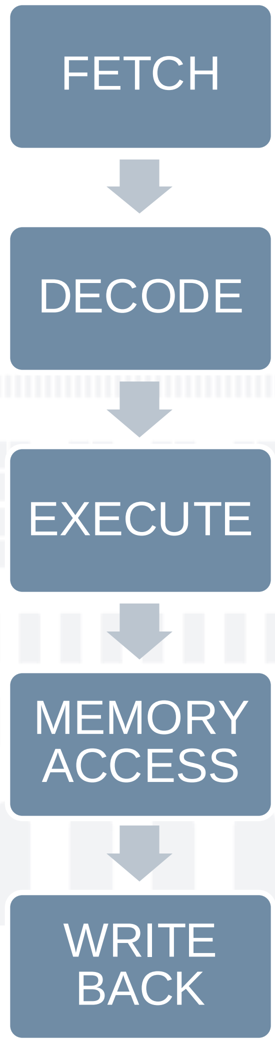
INSTRUCTION 1

THE SINGLE CYCLE DATAPATH



INSTRUCTION 2

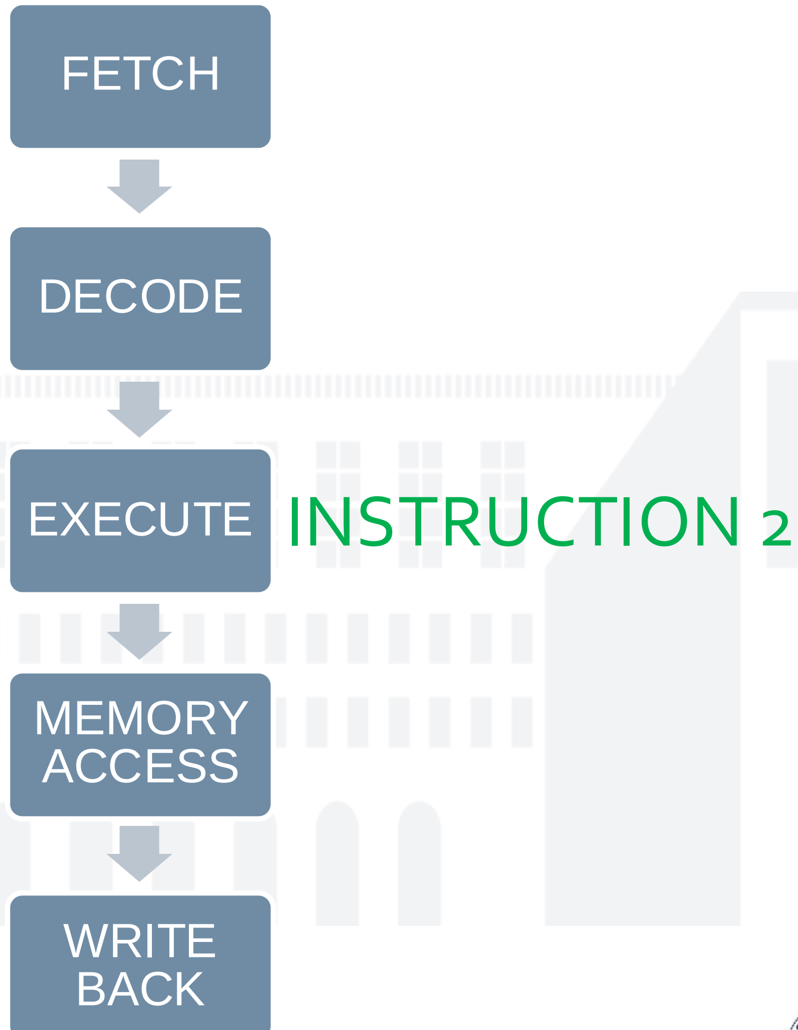
THE SINGLE CYCLE DATAPATH



INSTRUCTION 2

UNIVERSITÀ DEGLI STUDI DI TRIESTE

THE SINGLE CYCLE DATAPATH



THE SINGLE CYCLE DATAPATH

FETCH

↓

DECODE

↓

EXECUTE

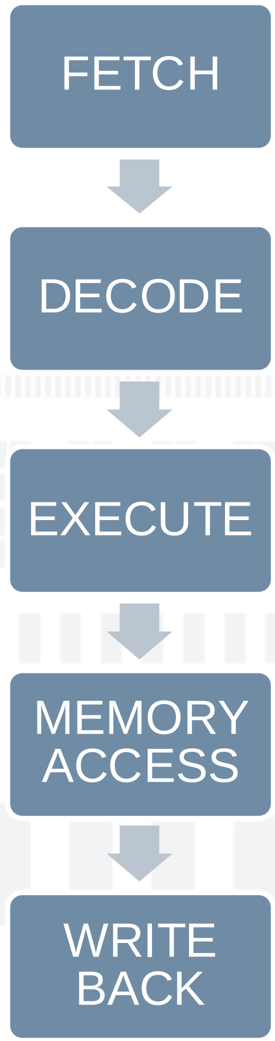
↓

MEMORY ACCESS — INSTRUCTION 2

↓

WRITE BACK

THE SINGLE CYCLE DATAPATH



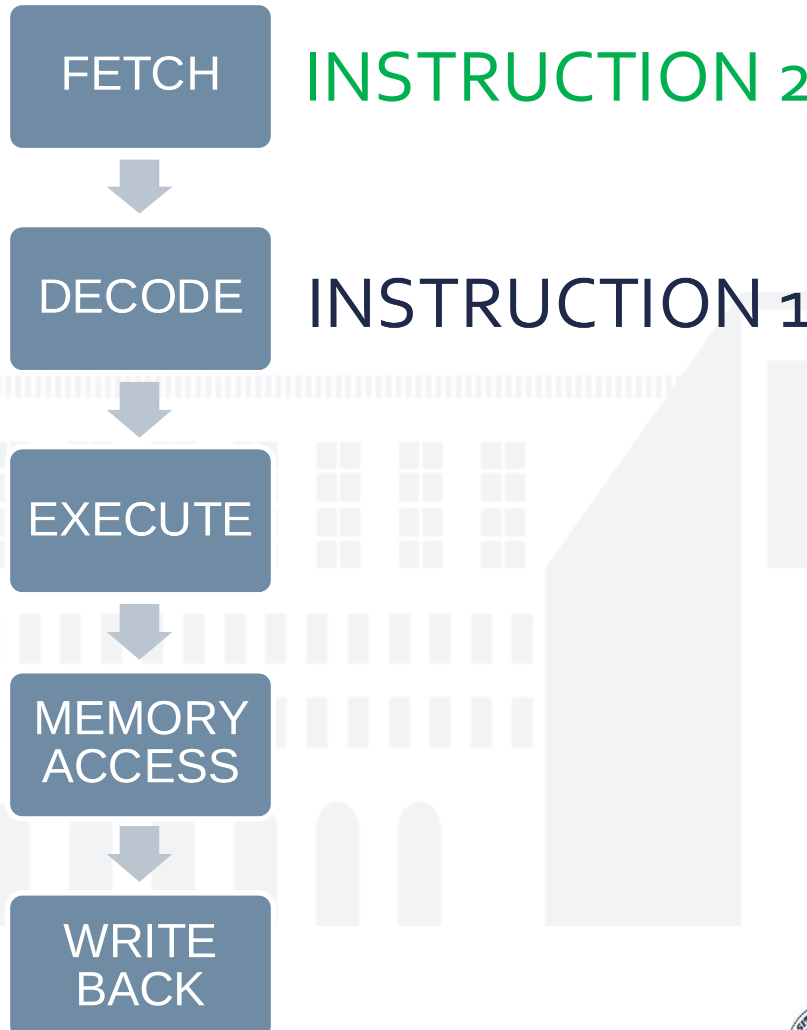
INSTRUCTION 2

THE PIPELINED DATAPATH

FETCH → DECODE → EXECUTE → MEMORY ACCESS → WRITE BACK

INSTRUCTION 1

THE PIPELINED DATAPATH



THE PIPELINED DATAPATH

FETCH → INSTRUCTION 3

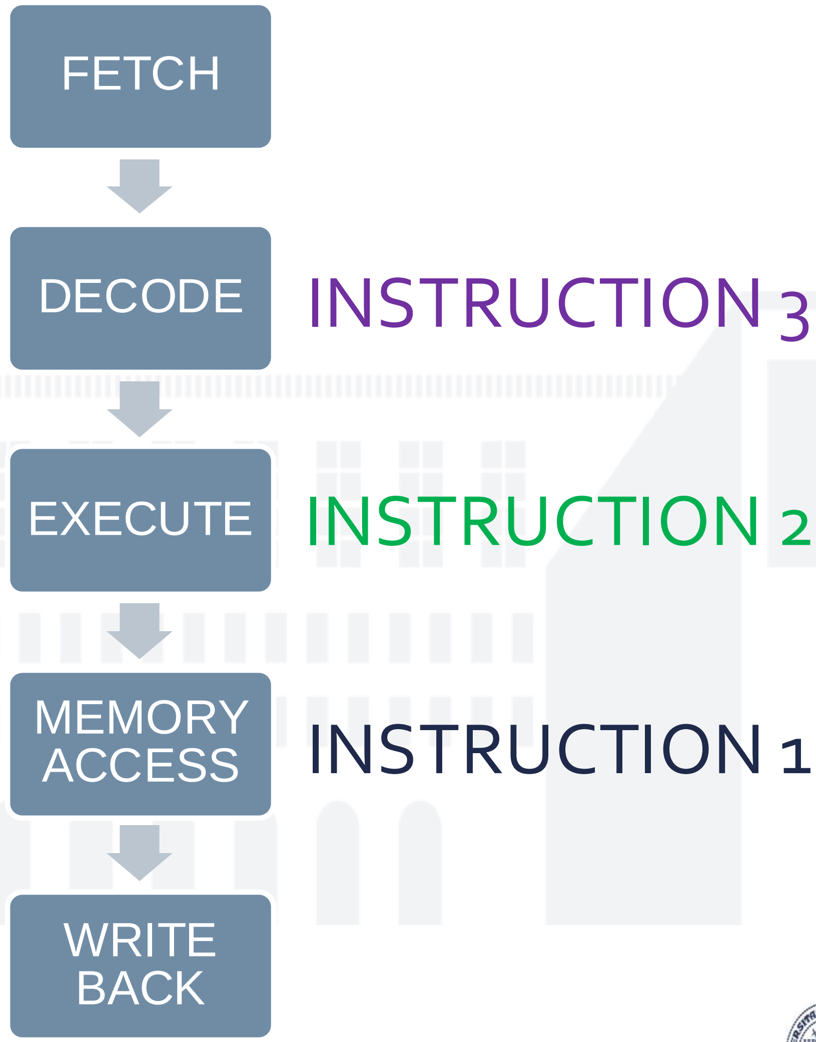
DECODE → INSTRUCTION 2

EXECUTE → INSTRUCTION 1

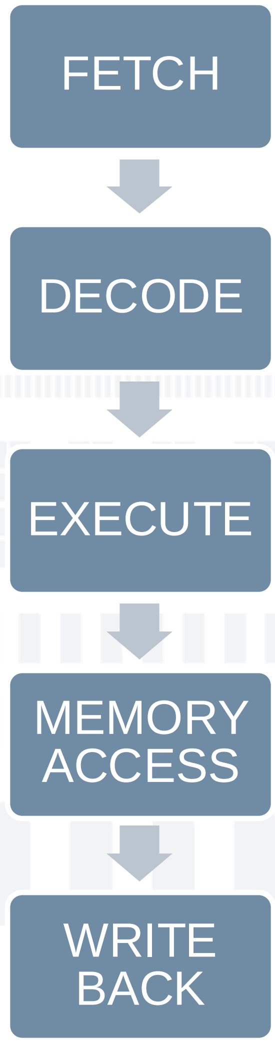
MEMORY ACCESS

WRITE BACK

THE PIPELINED DATAPATH



THE PIPELINED DATAPATH

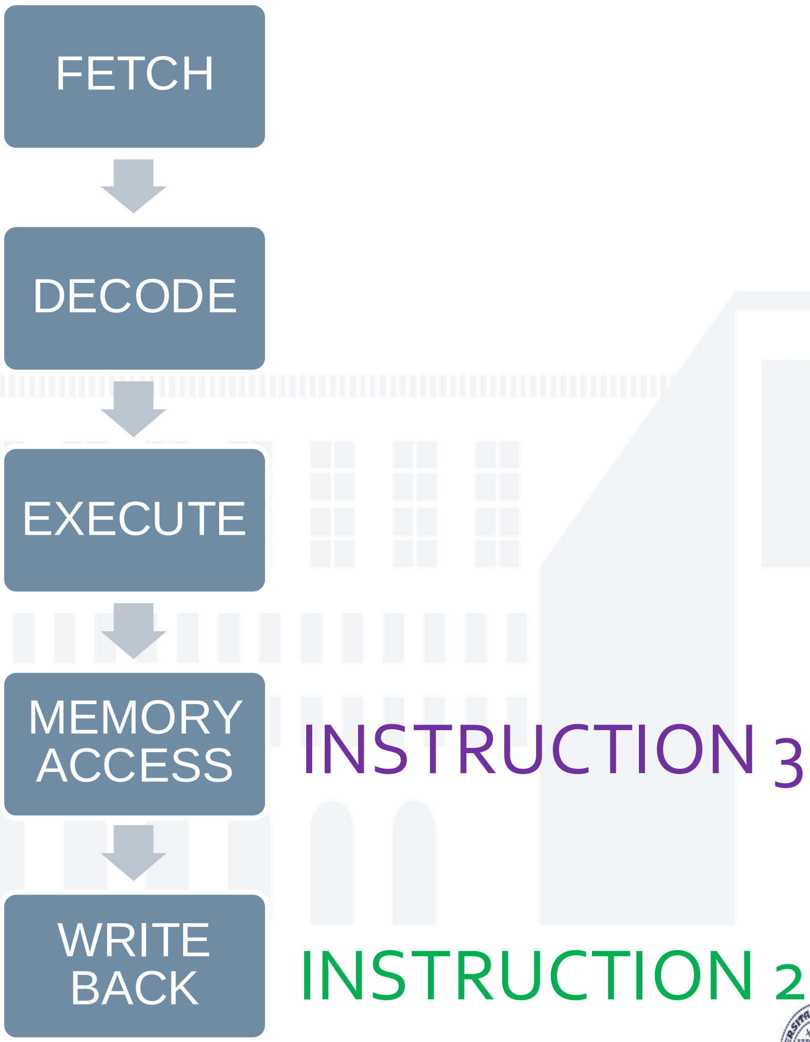


INSTRUCTION 3

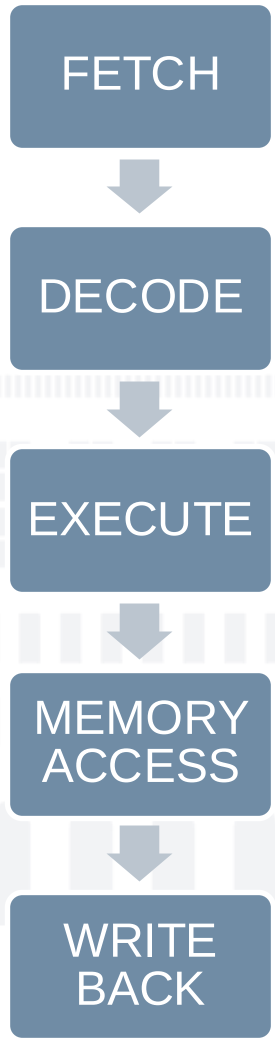
INSTRUCTION 2

INSTRUCTION 1

THE PIPELINED DATAPATH



THE PIPELINED DATAPATH



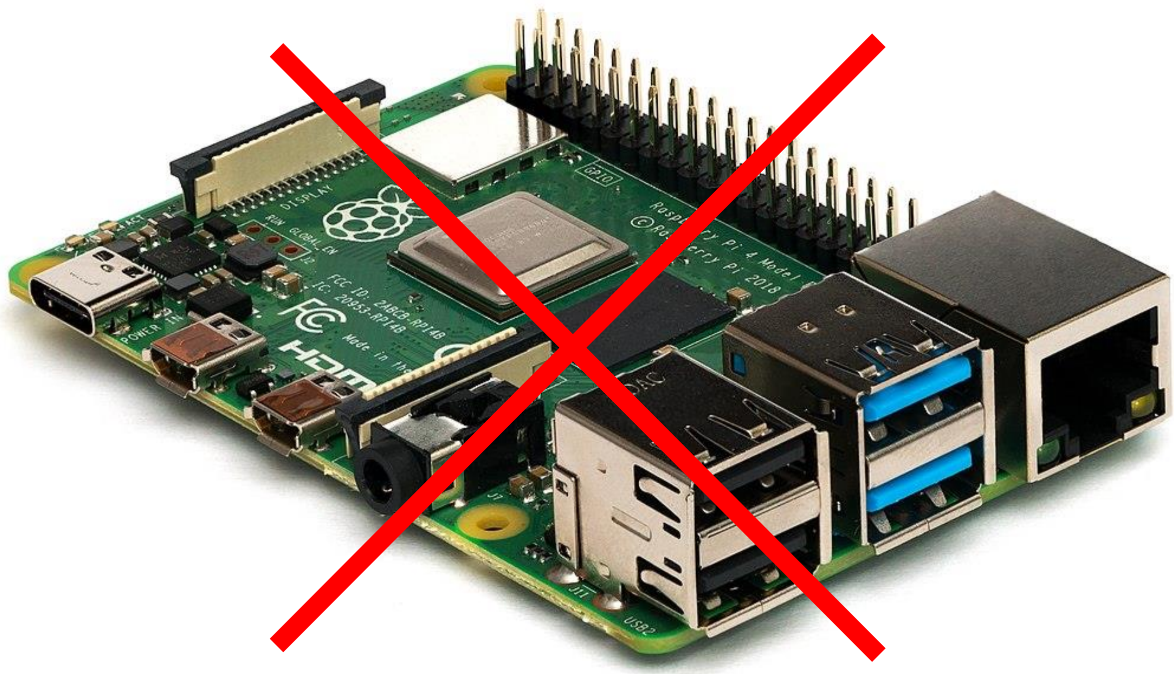
INSTRUCTION 3

UNIVERSITÀ DEGLI STUDI DI TRIESTE

WHAT SIMULATOR, AND WHY?

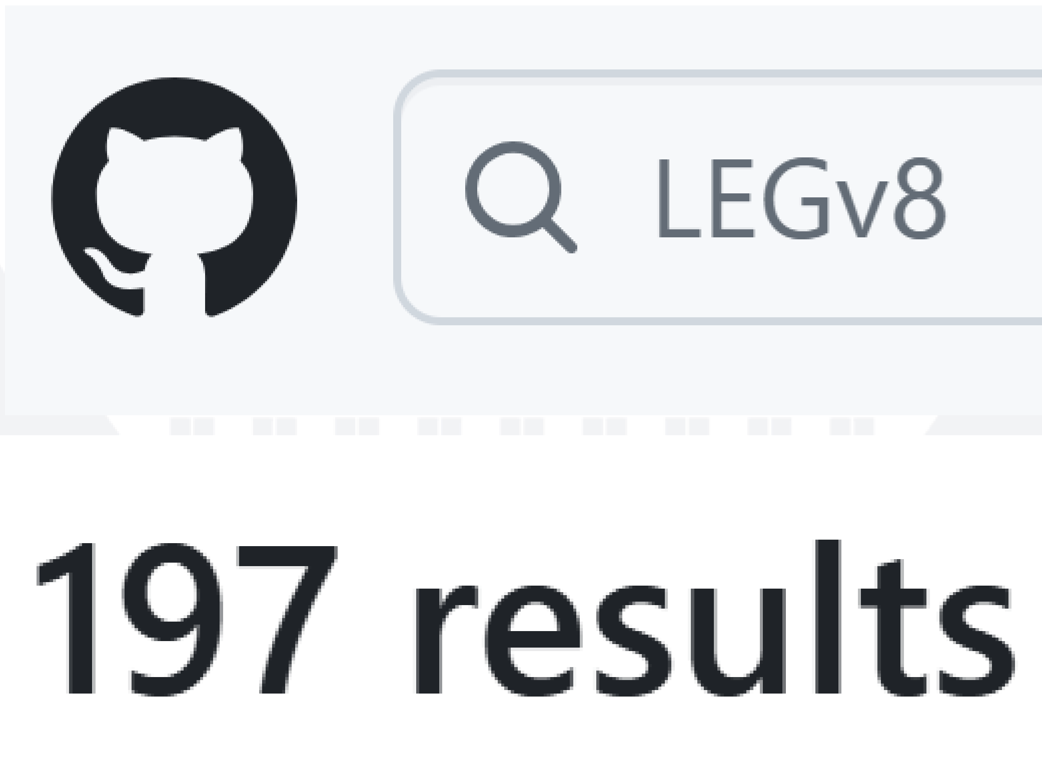
Restoration and development of
Arm's Java-based LEGv8 ISA simulator

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 <u>from scratch</u>

OR (BETTER)

- Improve one that <u>already exists</u>

ARM HAS OFFICIALLY MADE A LEGv8 SIMULATOR

GOOD!

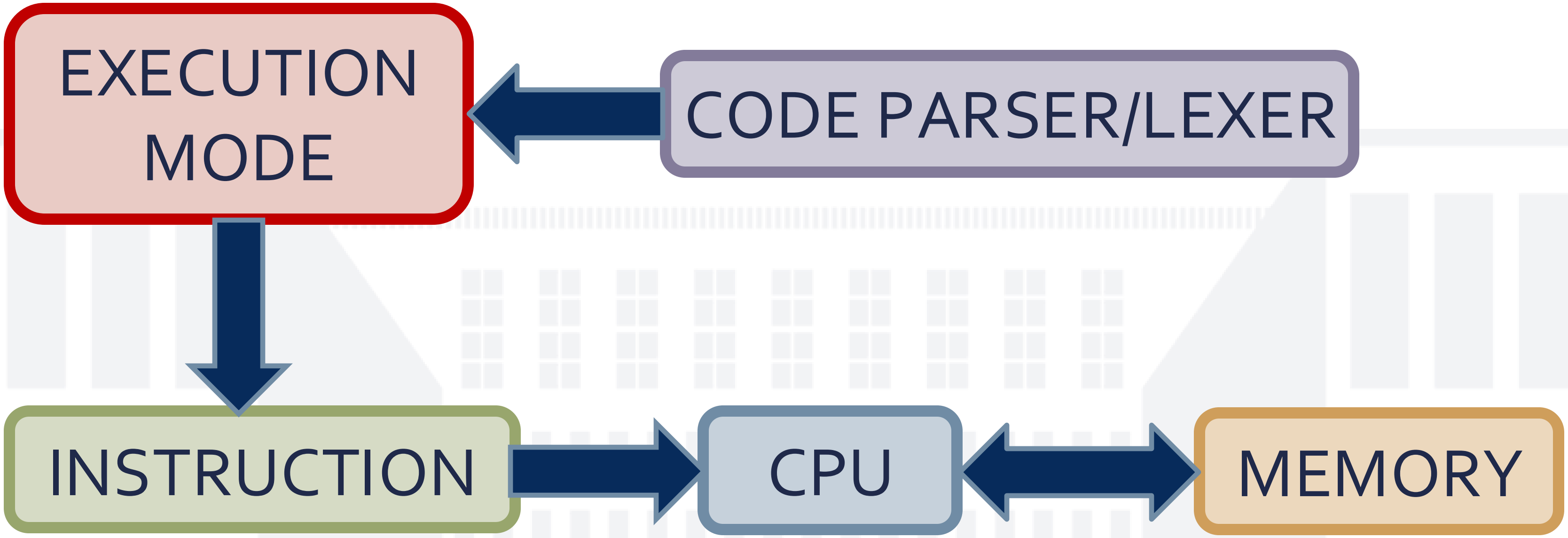
IT'S INCOMPLETE AND BROKEN

BAD!

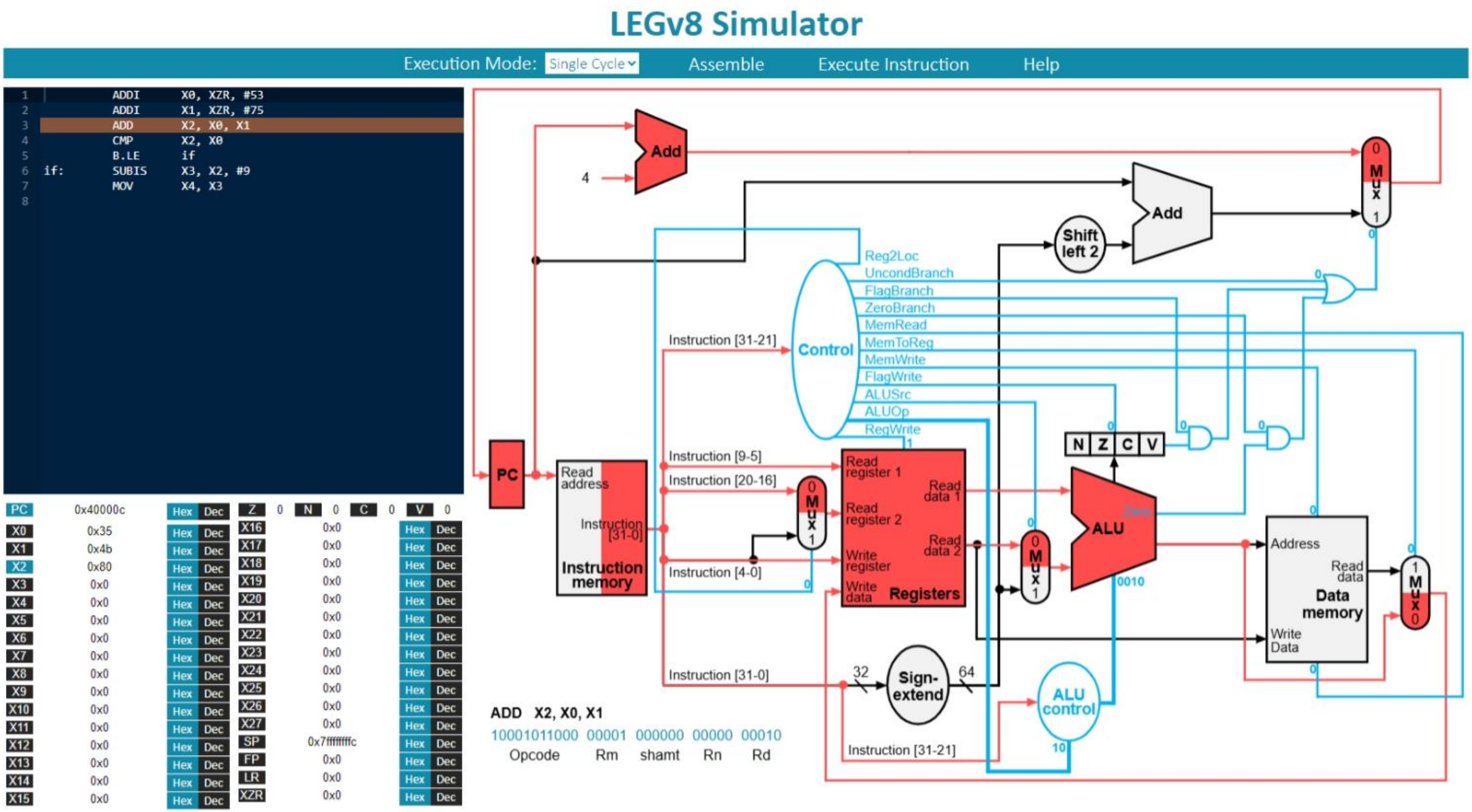
WHAT STANDS OUT:

- 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

WHERE'S THE CATCH?

- Integer comparisons are broken
- Subroutine calls are broken
- Even if fixed: nobody knows how to compile the project!

=> BASICALLY UNUSABLE

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

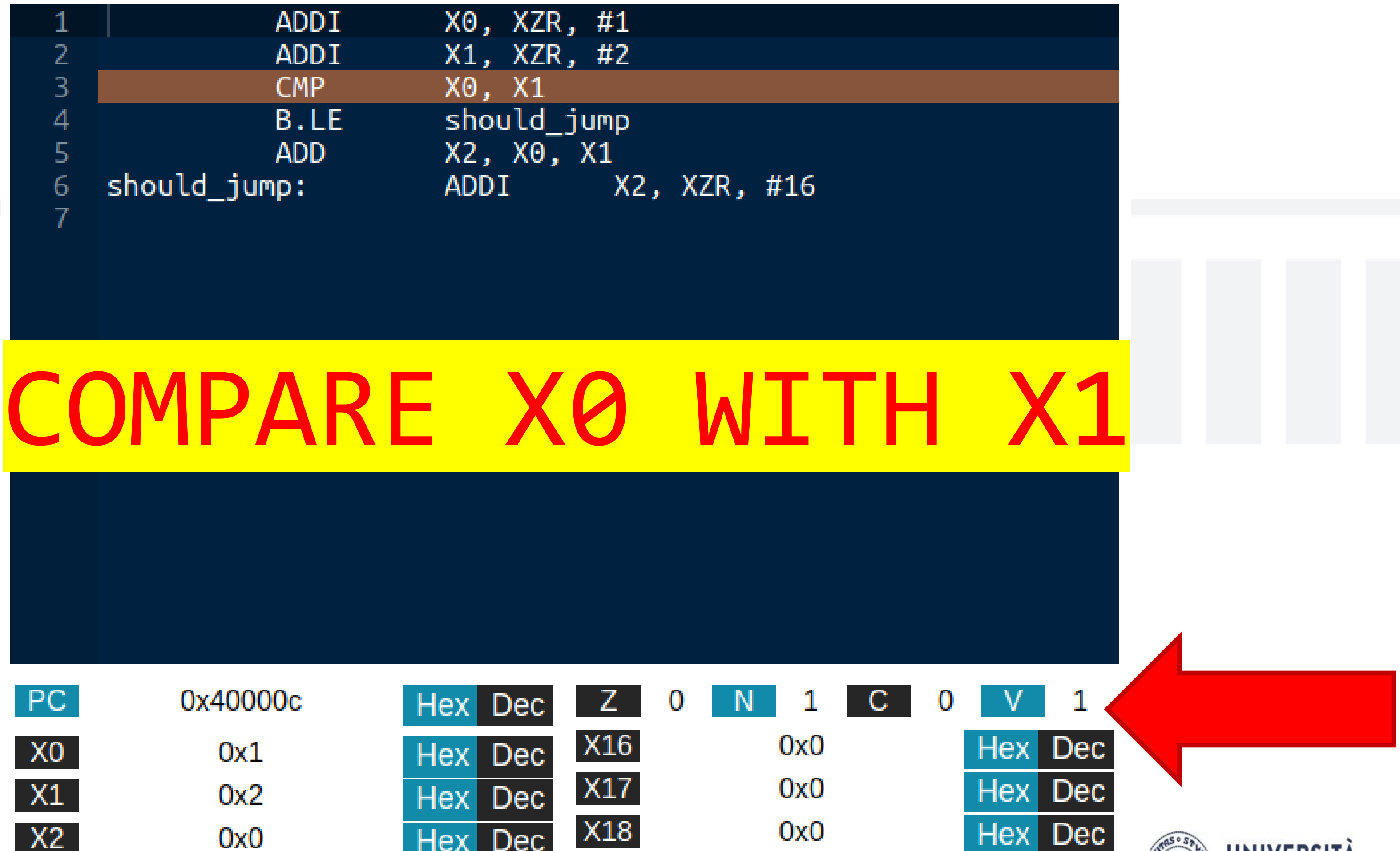
THE COMPARISON BUG

```
            ADDI        X0, XZR, #1
            ADDI        X1, XZR, #2
            CMP         X0, X1
            B.LE        should_jump
            ADD         X2, X0, X1
should_jump:            ADDI        X2, XZR, #16
```

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



UNIVERSITÀ DEGLI STUDI DI TRIESTE

THE COMPARISON BUG

```
            ADDI        X0, XZR, #1
            ADDI        X1, XZR, #2
            CMP         X0, X1
            B.LE        should_jump
            ADD         X2, X0, X1
should_jump:            ADDI        X2, XZR, #16
```

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

```
            ADDI        X0, XZR, #1
            ADDI        X1, XZR, #2
            CMP         X0, X1
            B.LE        should_jump
            ADD         X2, X0, X1
should_jump:            ADDI        X2, XZR, #16
```

...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

$\neg(Z=0 \land N=V) = \neg(TRUE \land TRUE) = FALSE$

BRANCH AND LINKS DON'T WORK!

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

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

CAN'T REUSE CODE

THE BRANCH AND LINK BUG

```
        ADDI    X0, XZR, #1
        BL      subroutine
        B       exit

subroutine:
        ADDI    X0, X0, #16
        BR      LR
exit:

```

PROGRAM COUNTER: 0x0

RETURN ADDRESS: 0x0

THE BRANCH AND LINK BUG

```
            ADDI    X0, XZR, #1
            BL      subroutine
            B       exit

subroutine:
            ADDI    X0, X0, #16
            BR      LR
exit:
```

PROGRAM COUNTER: 0x4

RETURN ADDRESS: 0x0

THE BRANCH AND LINK BUG

```
            ADDI      X0, XZR, #1
            BL        subroutine
            B         exit

subroutine:
            ADDI      X0, X0, #16
            BR        LR
exit:

```

PROGRAM COUNTER: 0xC

RETURN ADDRESS: 0xC

UNIVERSITÀ DEGLI STUDI DI TRIESTE

THE BRANCH AND LINK BUG

```
        ADDI    X0, XZR, #1
        BL      subroutine
        B       exit

subroutine:
        ADDI    X0, X0, #16
        BR      LR
exit:

```

PROGRAM COUNTER: 0x10

RETURN ADDRESS: 0xC

THE BRANCH AND LINK BUG

```
            ADDI        X0, XZR, #1
            BL          subroutine
            B           exit

subroutine:
            ADDI        X0, X0, #16
            BR          LR
exit:

```

IT DOESN'T GO BACK!

UNIVERSITÀ DEGLI STUDI DI TRIESTE

IT SHOULD GO HERE!

```
          ADDI     X0, XZR, #1
          BL       subroutine
          B        exit
subroutine:
          ADDI     X0, X0, #16
          BR       LR
exit:
```

ADDRESS: 0x8

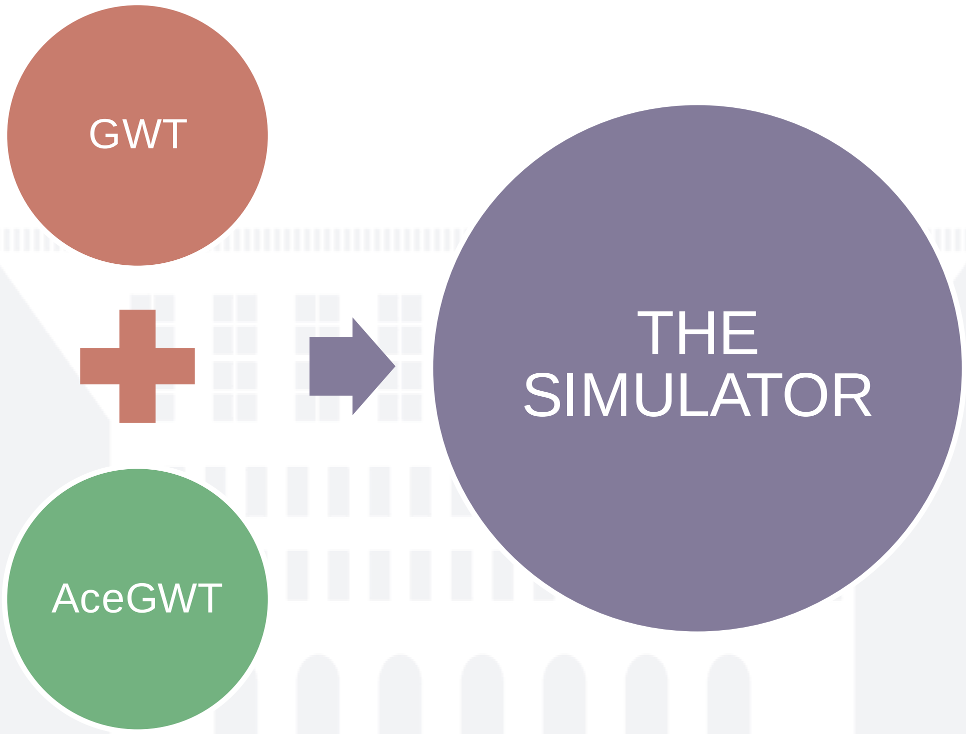
PROGRAM COUNTER: 0xC

RETURN ADDRESS: 0xC

UNIVERSITÀ DEGLI STUDI DI TRIESTE

ALL FIXED? NOW WHAT?

THE PROJECT'S LIBRARIES



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

TO BUILD THE PROJECT:

- 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

UNIVERSITÀ DEGLI STUDI DI TRIESTE

WHAT IS THE SIMULATOR MISSING?

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

UNIVERSITÀ DEGLI STUDI DI TRIESTE

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

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

UNIVERSITÀ DEGLI STUDI DI TRIESTE

THE NEW STACK VIEW

0x8000000000:	0x0	Hex	0x7fffffff80:	0x0	Hex
0x7ffffffff8:	0x0	Hex	0x7fffffff78:	0x0	Hex
0x7ffffffff0:	0x0	Hex	0x7fffffff70:	0x0	Hex
0x7fffffffe8:	0x0	Hex	0x7fffffff68:	0x0	Hex
0x7fffffffe0:	0x0	Hex	0x7fffffff60:	0x0	Hex
0x7fffffffd8:	0x0	Hex	0x7fffffff58:	0x0	Hex
0x7fffffffd0:	0x0	Hex	0x7fffffff50:	0x0	Hex
0x7fffffffc8:	0x0	Hex	0x7fffffff48:	0x0	Hex
0x7fffffffc0:	0x0	Hex	0x7fffffff40:	0x0	Hex
0x7fffffffb8:	0x0	Hex	0x7fffffff38:	0x0	Hex
0x7fffffffb0:	0x0	Hex	0x7fffffff30:	0x0	Hex
0x7fffffffa8:	0x0	Hex	0x7fffffff28:	0x0	Hex
0x7fffffffa0:	0x0	Hex	0x7fffffff20:	0x0	Hex
0x7fffffff98:	0x0	Hex	0x7fffffff18:	0x0	Hex
0x7fffffff90:	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 - 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 *ints* as raw bits
- Use Java *Double.longBitsToDouble* and *Double.doubleToLongBits* to convert before memory

UNIVERSITÀ DEGLI STUDI DI TRIESTE

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

INTEGER REGISTERS

X0	0xc	Hex	Dec
X0	12	Hex	Dec

FLOATING-POINT REGISTERS

S0	0xc00a0000	Hex	Dec
S0	-2.15625	Hex	Dec

THE FINAL VIEW

```
MOVK    X0, #192
LSL     X0, X0, #8
MOVK    X0, #10
LSL     X0, X0, #16
SUBI    SP, SP, #16
STUR    X0, [SP, #0]
LDURS   S0, [SP, #0]
```

0x8000000000:	0x0	Hex	0x7fffffff80:	0x0	Hex
0x7ffffffff8:	0x0	Hex	0x7fffffff78:	0x0	Hex
0x7ffffffff0:	0xc00a0000	Hex	0x7fffffff70:	0x0	Hex
0x7fffffffe8:	0x0	Hex	0x7fffffff68:	0x0	Hex
0x7fffffffe0:	0x0	Hex	0x7fffffff60:	0x0	Hex
0x7fffffffd8:	0x0	Hex	0x7fffffff58:	0x0	Hex
0x7fffffffd0:	0x0	Hex	0x7fffffff50:	0x0	Hex
0x7fffffffc8:	0x0	Hex	0x7fffffff48:	0x0	Hex
0x7fffffffc0:	0x0	Hex	0x7fffffff40:	0x0	Hex
0x7fffffffb8:	0x0	Hex	0x7fffffff38:	0x0	Hex
0x7fffffffb0:	0x0	Hex	0x7fffffff30:	0x0	Hex
0x7fffffffa8:	0x0	Hex	0x7fffffff28:	0x0	Hex
0x7fffffffa0:	0x0	Hex	0x7fffffff20:	0x0	Hex
0x7fffffff98:	0x0	Hex	0x7fffffff18:	0x0	Hex
0x7fffffff90:	0x0	Hex	0x7fffffff10:	0x0	Hex
0x7fffffff88:	0x0	Hex	0x7fffffff08:	0x0	Hex

PC 0x40001c — Z 0 N 0 C 0 V 0

X0	0xc00a0000	X16	0x0	D0	0xc00a0000	D16	0x0	S0	-2.15625
X1	0x0	X17	0x0	D1	0x0	D17	0x0	S1	0x0
X2	0x0	X18	0x0	D2	0x0	D18	0x0	S2	0x0
X3	0x0	X19	0x0	D3	0x0	D19	0x0	S3	0x0
X4	0x0	X20	0x0	D4	0x0	D20	0x0	S4	0x0
X5	0x0	X21	0x0	D5	0x0	D21	0x0	S5	0x0
X6	0x0	X22	0x0	D6	0x0	D22	0x0	S6	0x0
X7	0x0	X23	0x0	D7	0x0	D23	0x0	S7	0x0
X8	0x0	X24	0x0	D8	0x0	D24	0x0	S8	0x0
X9	0x0	X25	0x0	D9	0x0	D25	0x0	S9	0x0
X10	0x0	X26	0x0	D10	0x0	D26	0x0	S10	0x0
X11	0x0	X27	0x0	D11	0x0	D27	0x0	S11	0x0
X12	0x0	SP	0x7ffffffff0	D12	0x0	D28	0x0	S12	0x0
X13	0x0	FP	0x0	D13	0x0	D29	0x0	S13	0x0
X14	0x0	LR	0x0	D14	0x0	D30	0x0	S14	0x0
X15	0x0	XZR	0x0	D15	0x0	D31	0x0	S15	0x0

S16	0x0
S17	0x0
S18	0x0
S19	0x0
S20	0x0
S21	0x0
S22	0x0
S23	0x0
S24	0x0
S25	0x0
S26	0x0
S27	0x0
S28	0x0
S29	0x0
S30	0x0
S31	0x0

THE CHERRY ON TOP: MODERNIZING THE BUILD SYSTEM

- Opens the doors to effortless collaboration
- Decouples project from Eclipse IDE
- Use updated libraries

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.

UNIVERSITÀ DEGLI STUDI DI TRIESTE

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

UNIVERSITÀ DEGLI STUDI DI TRIESTE

SOME FUTURE DEVELOPMENTS

- Refactor codebase with modern practices
- Improve pipelined execution
- Improve UI, make it responsive
- Find modern replacements for GWT, AceGWT
- Get the changes to the official repos

THANK YOU FOR YOUR ATTENTION

[THESIS AVAILABLE HERE]

[SIMULATOR AVAILABLE HERE]

UNIVERSITÀ DEGLI STUDI DI TRIESTE