BUILDING A
RISC-V CORE

Computer Organization – IUST Spring 2023

Iran University of
Science and Technology

LinkedIn : arvin-delavari

faraz-ghoreishy

E-Mail : arvin7807@gmail.com

farazghoreishy@gmail.com

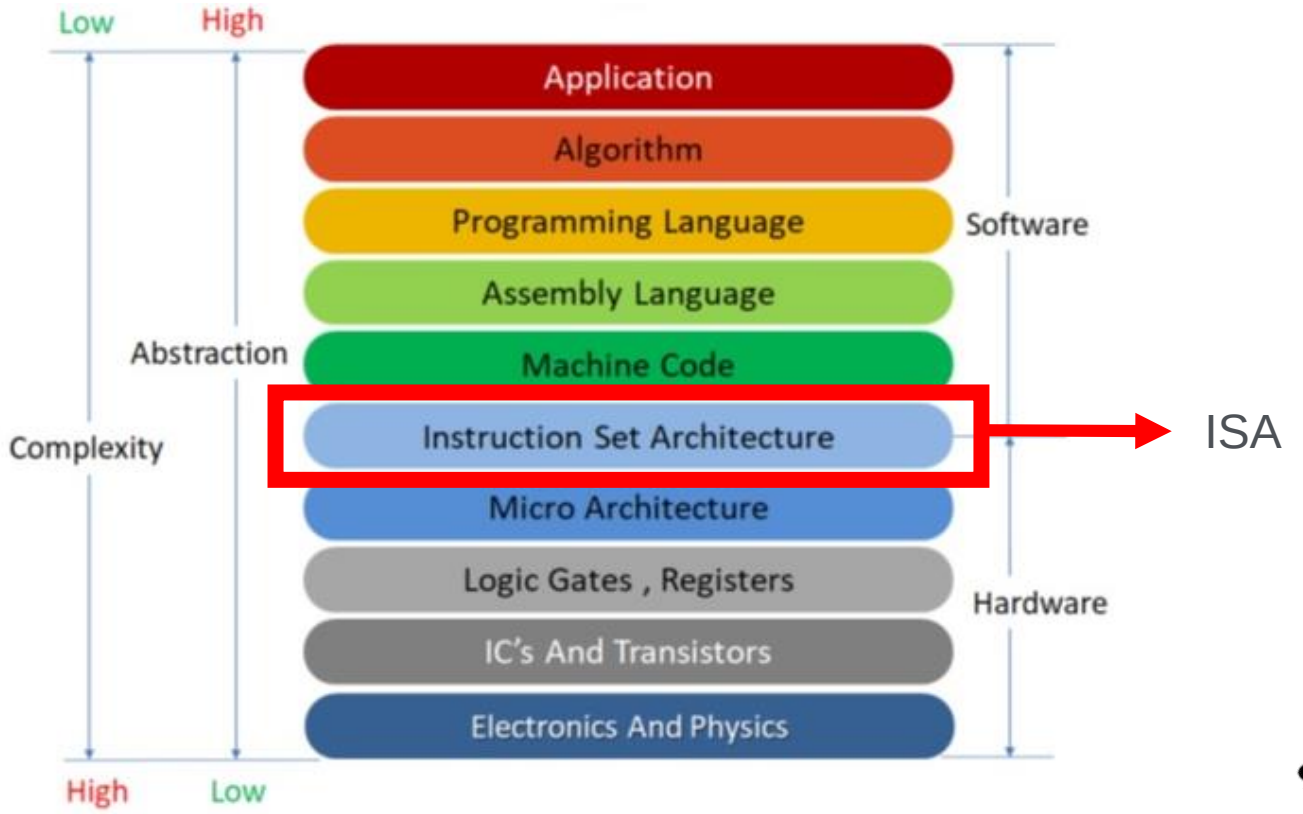
iustCompOrg+4012@gmail.com

Contents :

IUST
Iran University of
Science and Technology

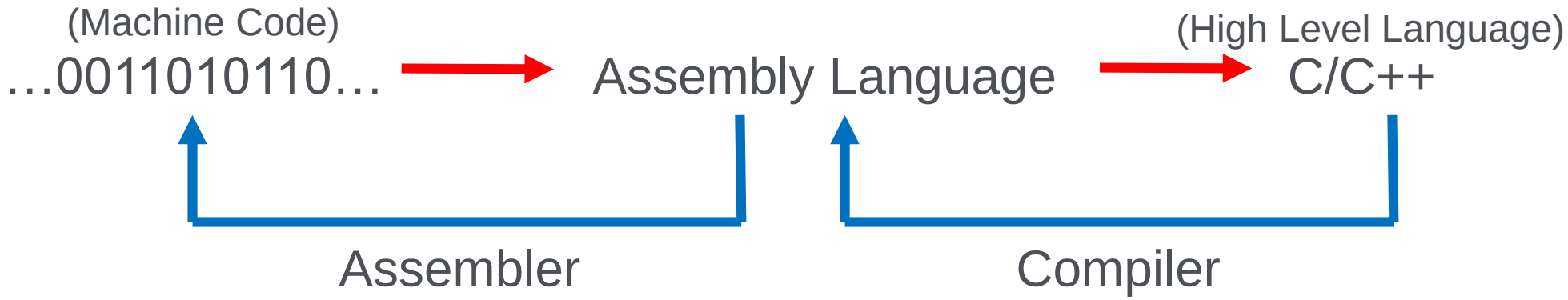
1 – Instruction Set Architecture(ISA)

- In computer science, an instruction set architecture (also called computer architecture) is an abstract model of a computer.
- A device that executes instructions described by that ISA, such as a central processing unit, is called an implementation



Iran University of Science and Technology

1 – Instruction Set Architecture(ISA)



Famous ISAs :

ARM **MIPS** **IA(intel x86)** **RISC-V**

IUST
Iran University of
Science and Technology

1 – Instruction Set Architecture(ISA)

The ISA defines the supported :

- Data types
- The registers
- Memory management (main)
- Key features (such as virtual memory), which instructions a microprocessor can execute
- The Input/Output (I/O)

The ISA can be extended by adding instructions or other capabilities, or by adding support for larger addresses and data values.

IUST
Iran University of
Science and Technology

1 – Instruction Set Architecture(ISA)

Main computer architecture concepts :

RISC(Reduced Instruction Set Computer)

CISC(Complex Instruction Set Computer)

- The primary difference between RISC and CISC architecture is that **RISC-based machines execute one instruction per clock cycle**.
- In a CISC processor, each instruction performs so many actions that it takes several clock cycles to complete.

IUST
Iran University of
Science and Technology

1 – Instruction Set Architecture(ISA)

RISC vs. CISC

CISC	RISC
Emphasis on hardware	Emphasis on software
Multiple instruction sizes and formats	Instructions of same set with few formats
Less registers	Uses more registers
More addressing modes	Fewer addressing modes
Extensive use of microprogramming	Complexity in compiler
Instructions take a varying amount of cycle time	Instructions take one cycle time
Pipelining is difficult	Pipelining is easy

2 – Introduction to RISC-V

- RISC-V is an open standard instruction set architecture based on established RISC principles.
- Unlike most other ISA designs, RISC-V is provided **under open source** licenses that do not require fees to use.

Designer	University of California, Berkeley
Bits	32, 64, 128
Introduced	2010; 12 years ago
Version	unprivileged ISA 20191213,[1] privileged ISA 20211203[2]
Design	RISC
Branching	Compare-and-branch
Endianness	Little[1]:9[a]
Page size	4 KiB

Extensions

M: Multiplication
A: Atomics – LR/SC & fetch-and-op
F: Floating point (32-bit)
D: F.P. Double (64-bit)
Q: F.P. Quad (128-bit)
Zicsr: Control and status register support
Zifencei: Load/store fence
C: Compressed instructions (16-bit)
J: Interpreted or JIT compiled languages support

IUST
Iran University of Science and Technology

2 – Introduction to RISC-V

Name	Description	Version	Status[b]	Instruction count
Base				
RVWMO	Weak Memory Ordering	2.0	Ratified	
RV32I	Base Integer Instruction Set, 32-bit	2.1	Ratified	40
RV32E	Base Integer Instruction Set (embedded), 32-bit, 16 registers	1.9	Open	40
RV64I	Base Integer Instruction Set, 64-bit	2.1	Ratified	15
RV128I	Base Integer Instruction Set, 128-bit	1.7	Open	15
Extension				
M	Standard Extension for Integer Multiplication and Division	2.0	Ratified	8 (RV32) 13 (RV64)
A	Standard Extension for Atomic Instructions	2.1	Ratified	11 (RV32) 22 (RV64)
F	Standard Extension for Single-Precision Floating-Point	2.2	Ratified	26 (RV32) 30 (RV64)
D	Standard Extension for Double-Precision Floating-Point	2.2	Ratified	26 (RV32) 32 (RV64)
Zicsr	Control and Status Register (CSR)	2.0	Ratified	6
Zifencei	Instruction-Fetch Fence	2.0	Ratified	1
G	Shorthand for the IMAFDZicsr_Zifencei base and extensions	—	—	
Q	Standard Extension for Quad-Precision Floating-Point	2.2	Ratified	28 (RV32) 32 (RV64)
L	Standard Extension for Decimal Floating-Point	0.0	Open	
C	Standard Extension for Compressed Instructions	2.0	Ratified	40
B	Standard Extension for Bit Manipulation	1.0	Ratified	43[28]
J	Standard Extension for Dynamically Translated Languages	0.0	Open	
T	Standard Extension for Transactional Memory	0.0	Open	
P	Standard Extension for Packed-SIMD Instructions	0.9.10	Open	
V	Standard Extension for Vector Operations	1.0	Frozen	187[29]
K	Standard Extension for Scalar Cryptography	1.0.1	Ratified	49
N	Standard Extension for User-Level Interrupts	1.1	Open	3
H	Standard Extension for Hypervisor	1.0	Ratified	15
S	Standard Extension for Supervisor-level Instructions	1.12	Ratified	4
Zam	Misaligned Atomics	0.1	Open	
Ztso	Total Store Ordering	0.1	Frozen	

IUST
Iran University of
Science and Technology

2 – Introduction to RISC-V

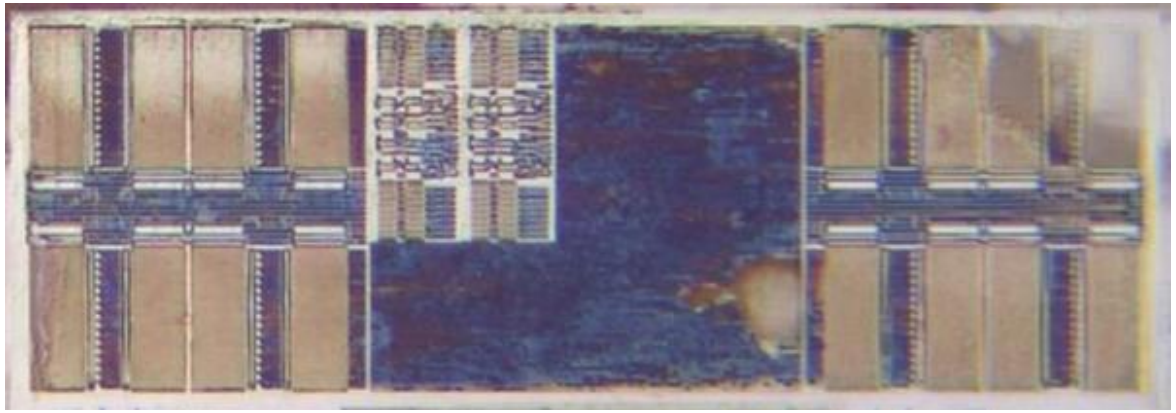
Vanilla-5

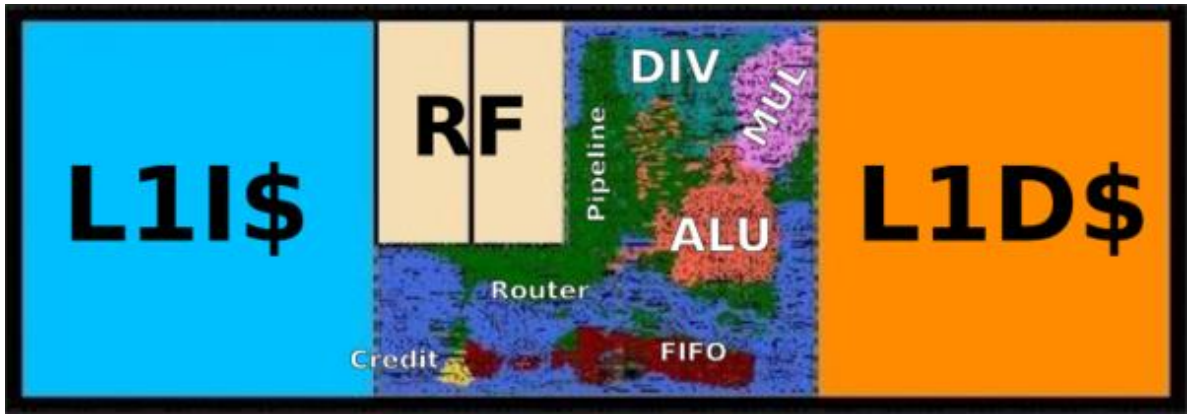
RV32IM ISA

The core is silicon-proven capable of up to 1.4 GHz.

24,251 µm² (0.024 mm²) die area

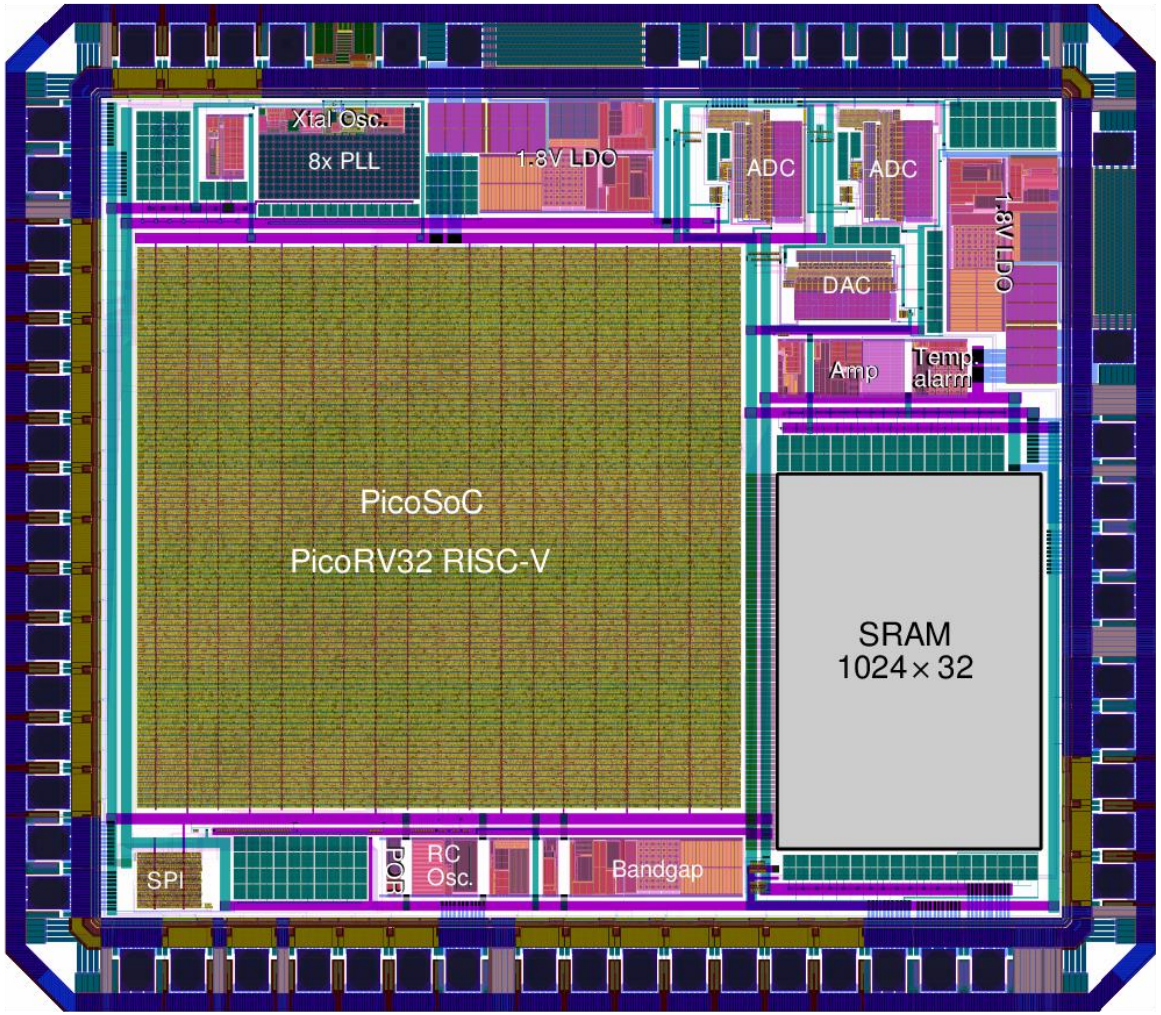
- University of Michigan
- University of Washington
- Cornell University





IUST
Iran University of Science and Technology

2 – Introduction to RISC-V



IUST
Iran University of Science and Technology

2 – Introduction to RISC-V

- Notable companies using RISC-V ISA :

Apple : x86 → ARM → **RISC-V?**

Apple Exploring RISC-V, Hiring RISC-V 'High Performance' Programmers

By Anton Shilov published September 03, 2021

RISC-V gets interest from a major player

Comments (5)

Apple is in the process of switching its PCs to Arm-based SoCs, but the company might not be putting all its eggs into one basket, as it is also exploring the emerging open-source RISC-V architecture. This week the company posted a job alert for RISC-V high-performance programmer(s).

Apple is currently looking for experienced programmers with detailed knowledge of the RISC-V Instruction Set Architecture (ISA) and Arm's Neon vector ISA for its Vector and Numerics Group (VaNG) within its Core Operating Systems group. Apple's VaNG is responsible for developing and improving various embedded subsystems running on iOS, macOS, watchOS, and tvOS.

Known for its secrecy, Apple's listing doesn't disclose exactly what it plans to do with RISC-V, but the job description indicates that the programmer will have to work with machine learning, computational vision, and natural language processing. Among other things, low-level high-performance programming experience is required. Furthermore, the job description also indicates that Apple is already working with RISC-V.

2 – Introduction to RISC-V

IUST
Iran University of Science and Technology

2 – Introduction to RISC-V

RISC-V operands

Name	Example	Comments
32 registers	x0-x31	Fast locations for data. In RISC-V, data must be in registers to perform arithmetic. Register x0 always equals 0.
2^{30} memory words	Memory[0], Memory[4], ..., Memory[4,294,967,292]	Accessed only by data transfer instructions. RISC-V uses byte addresses, so sequential word accesses differ by 4. Memory holds data structures, arrays, and spilled registers.

David A. Patterson, John L. Hennessy
Computer Organization and Design RISC-V Edition
(The Hardware Software Interface-Morgan Kaufmann)

IUST
Iran University of
Science and Technology

2 – Introduction to RISC-V

RISC-V assembly language

Category	Instruction	Example	Meaning	Comments
Arithmetic	Add	`add x5, x6, x7`	`x5 = x6 + x7`	Three register operands; add
	Subtract	`sub x5, x6, x7`	`x5 = x6 - x7`	Three register operands; subtract
	Add immediate	`addi x5, x6, 20`	`x5 = x6 + 20`	Used to add constants
Data transfer	Load word	`lw x5, 40(x6)`	`x5 = Memory[x6 + 40]`	Word from memory to register
	Load word, unsigned	`lwu x5, 40(x6)`	`x5 = Memory[x6 + 40]`	Unsigned word from memory to register
	Store word	`sw x5, 40(x6)`	`Memory[x6 + 40] = x5`	Word from register to memory
	Load halfword	`lh x5, 40(x6)`	`x5 = Memory[x6 + 40]`	Halfword from memory to register
	Load halfword, unsigned	`lhu x5, 40(x6)`	`x5 = Memory[x6 + 40]`	Unsigned halfword from memory to register
	Store halfword	`sh x5, 40(x6)`	`Memory[x6 + 40] = x5`	Halfword from register to memory
	Load byte	`lb x5, 40(x6)`	`x5 = Memory[x6 + 40]`	Byte from memory to register
	Load byte, unsigned	`lbu x5, 40(x6)`	`x5 = Memory[x6 + 40]`	Byte unsigned from memory to register
	Store byte	`sb x5, 40(x6)`	`Memory[x6 + 40] = x5`	Byte from register to memory
	Load reserved	`lr.d x5, (x6)`	`x5 = Memory[x6]`	Load; 1st half of atomic swap
	Store conditional	`sc.d x7, x5, (x6)`	`Memory[x6] = x5; x7 = 0/1`	Store; 2nd half of atomic swap
	Load upper immediate	`lui x5, 0x12345`	`x5 = 0x12345000`	Loads 20-bit constant shifted left 12 bits

David A. Patterson, John L. Hennessy
Computer Organization and Design RISC-V Edition
(The Hardware Software Interface-Morgan Kaufmann)

IUST
Iran University of
Science and Technology

2 – Introduction to RISC-V

Logical	And	`and x5, x6, x7`	`x5 = x6 & x7`	Three reg. operands; bit-by-bit AND
	Inclusive or	`or x5, x6, x8`	`x5 = x6 | x8`	Three reg. operands; bit-by-bit OR
	Exclusive or	`xor x5, x6, x9`	`x5 = x6 ^ x9`	Three reg. operands; bit-by-bit XOR
	And immediate	`andi x5, x6, 20`	`x5 = x6 & 20`	Bit-by-bit AND reg. with constant
	Inclusive or immediate	`ori x5, x6, 20`	`x5 = x6 | 20`	Bit-by-bit OR reg. with constant
	Exclusive or immediate	`xori x5, x6, 20`	`x5 = x6 ^ 20`	Bit-by-bit XOR reg. with constant
Shift	Shift left logical	`sll x5, x6, x7`	`x5 = x6 << x7`	Shift left by register
	Shift right logical	`srl x5, x6, x7`	`x5 = x6 >> x7`	Shift right by register
	Shift right arithmetic	`sra x5, x6, x7`	`x5 = x6 >> x7`	Arithmetic shift right by register
	Shift left logical immediate	`slli x5, x6, 3`	`x5 = x6 << 3`	Shift left by immediate
	Shift right logical immediate	`srli x5, x6, 3`	`x5 = x6 >> 3`	Shift right by immediate
	Shift right arithmetic immediate	`srai x5, x6, 3`	`x5 = x6 >> 3`	Arithmetic shift right by immediate

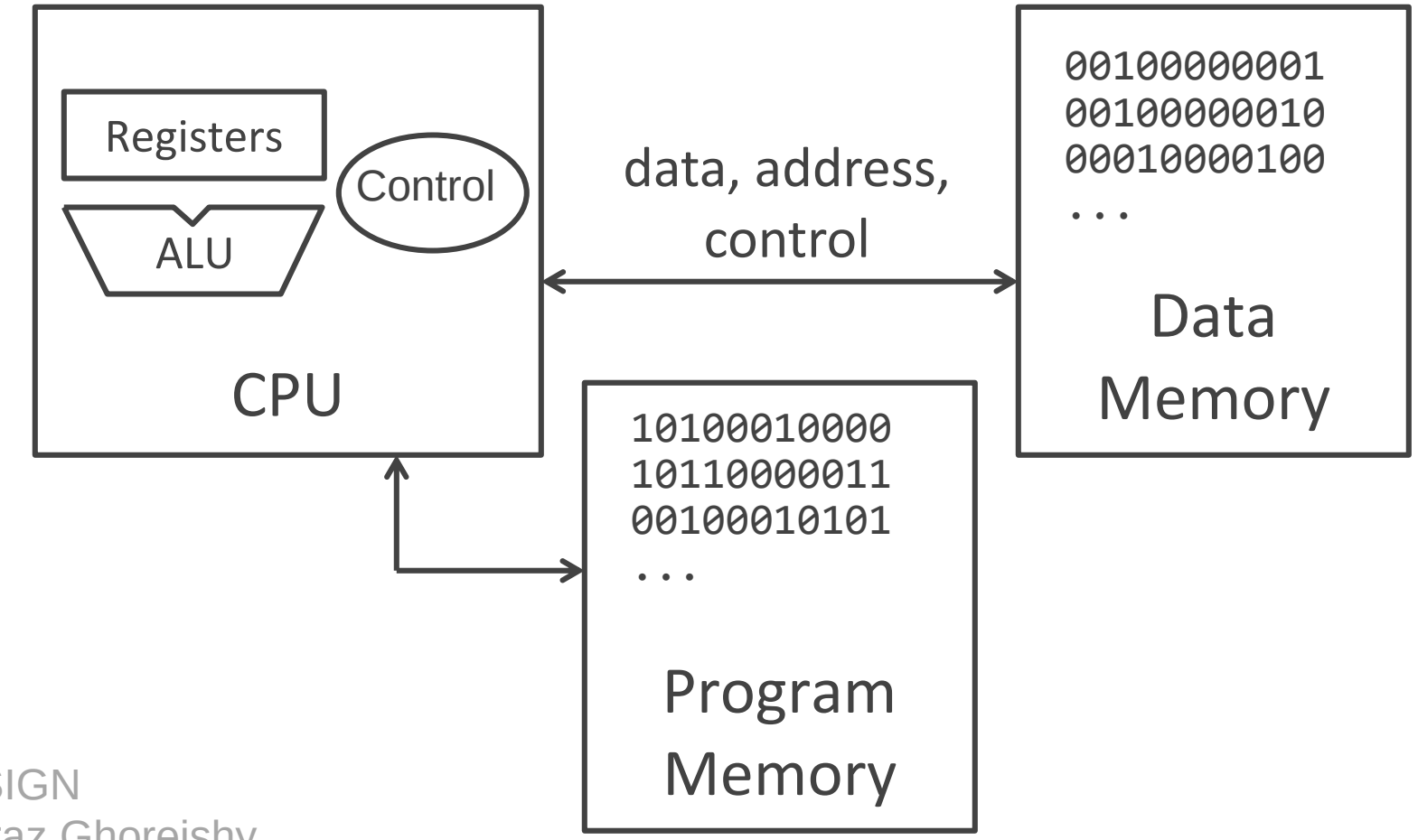
David A. Patterson, John L. Hennessy
Computer Organization and Design RISC-V Edition
(The Hardware Software Interface-Morgan Kaufmann)

IUST
Iran University of
Science and Technology

2 – Introduction to RISC-V

A RISC-V CPU with a (modified) Harvard architecture

- Modified: instructions & data in common address space, separate instr/data caches can be accessed in parallel



3 – Application Binary Interface(ABI)

API (Application programming Interface)

java.text – java
stdio.h – c
std::array – c++

Application Program

Ex : E-mail

Standard libraries

OS

ISA

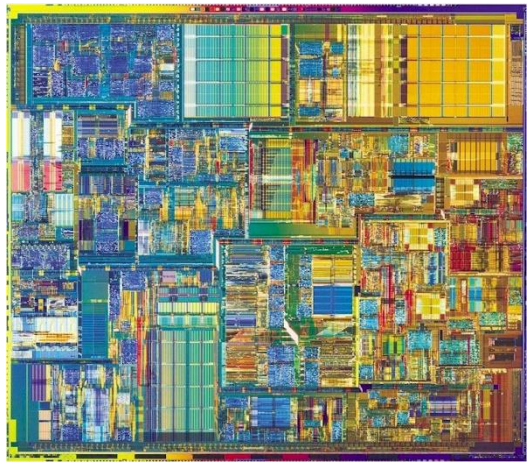
RISC-V, x86, ARM

```
0000000000000000 <main-0x3>:
   0:   00 01 00        # R_RISCV_ALIGN

0000000000000003 <main>:
   3:   00000537                lui     a0,0x0
   7:   fe010113                addi    sp,sp,-32
   b:   00050513                mv      a0,a0
   f:   00113c23                sd      ra,24(sp)
  13:   00813823                sd      s0,16(sp)
  17:   00000097                auipc   ra,0x0
  1b:   000080e7                jalr    ra
  1f:   00000437                lui     s0,0x0
  23:   00810593                addi    a1,sp,8
  27:   00040513                mv      a0,s0
  2b:   00000097                auipc   ra,0x0
  2f:   000080e7                jalr    ra
  33:   00000537                lui     a0,0x0
  37:   00050513                mv      a0,a0
  3b:   00000097                auipc   ra,0x0
  3f:   000080e7                jalr    ra
  43:   00c10593                addi    a1,sp,12
  47:   00040513                mv      a0,s0
  4b:   00000097                auipc   ra,0x0
  4f:   000080e7                jalr    ra
  53:   00c12783                lw      a5,12(sp)
  57:   00812583                lw      a1,8(sp)
  5b:   00000537                lui     a0,0x0
  5f:   00050513                mv      a0,a0
  63:   00f585bb                addw    a1,a1,a5
  67:   00000097                auipc   ra,0x0
  6b:   000080e7                jalr    ra
  6f:   00a00513                li      a0,10
  73:   00000097                auipc   ra,0x0
  77:   000080e7                jalr    ra
  7b:   01813083                ld      ra,24(sp)
  7f:   01013403                ld      s0,16(sp)
  83:   00000513                li      a0,0
  87:   02010113                addi    sp,sp,32
  8b:   00008067                ret
```

RTL

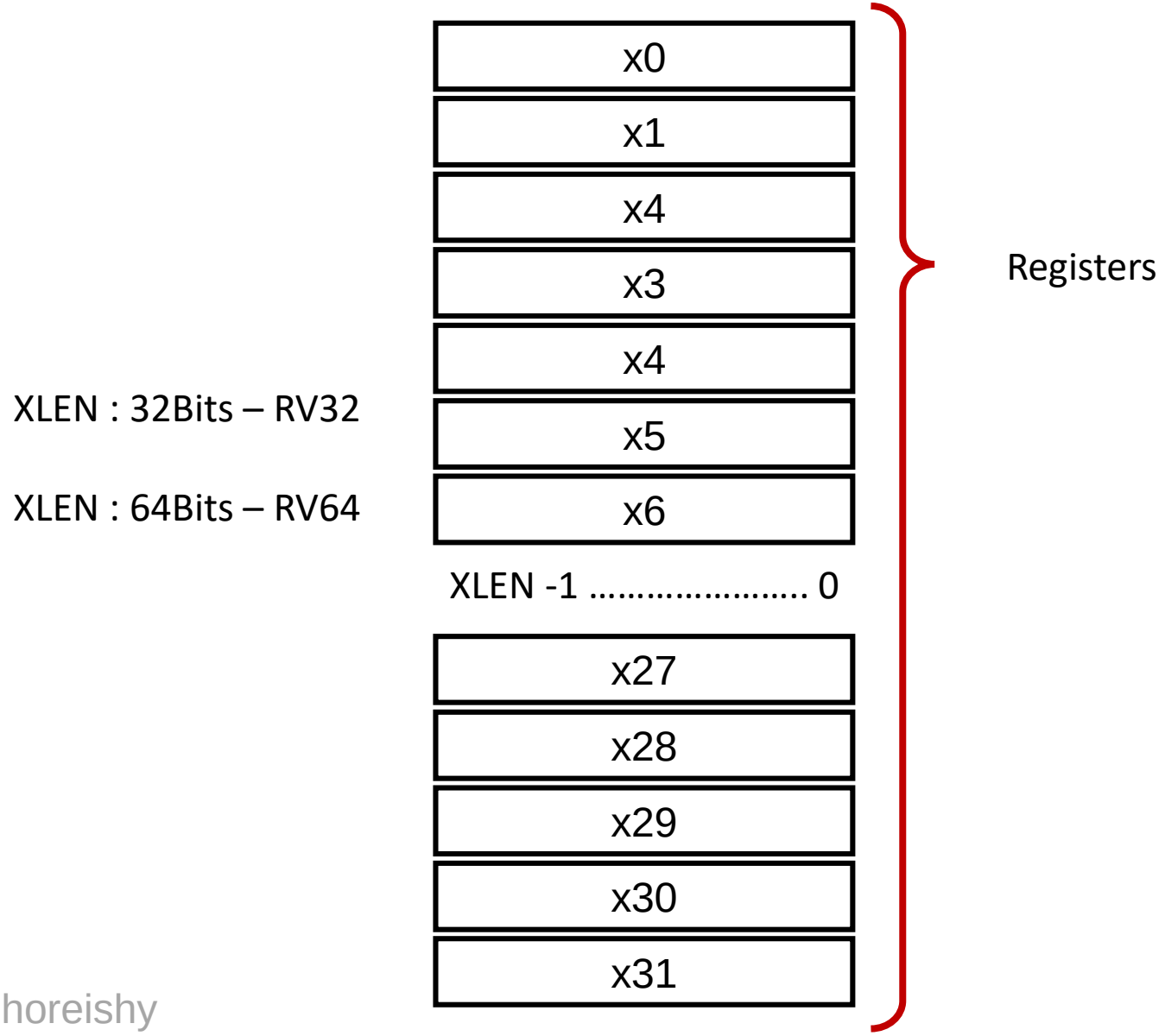
Hardware



Application Binary Interface(ABI)

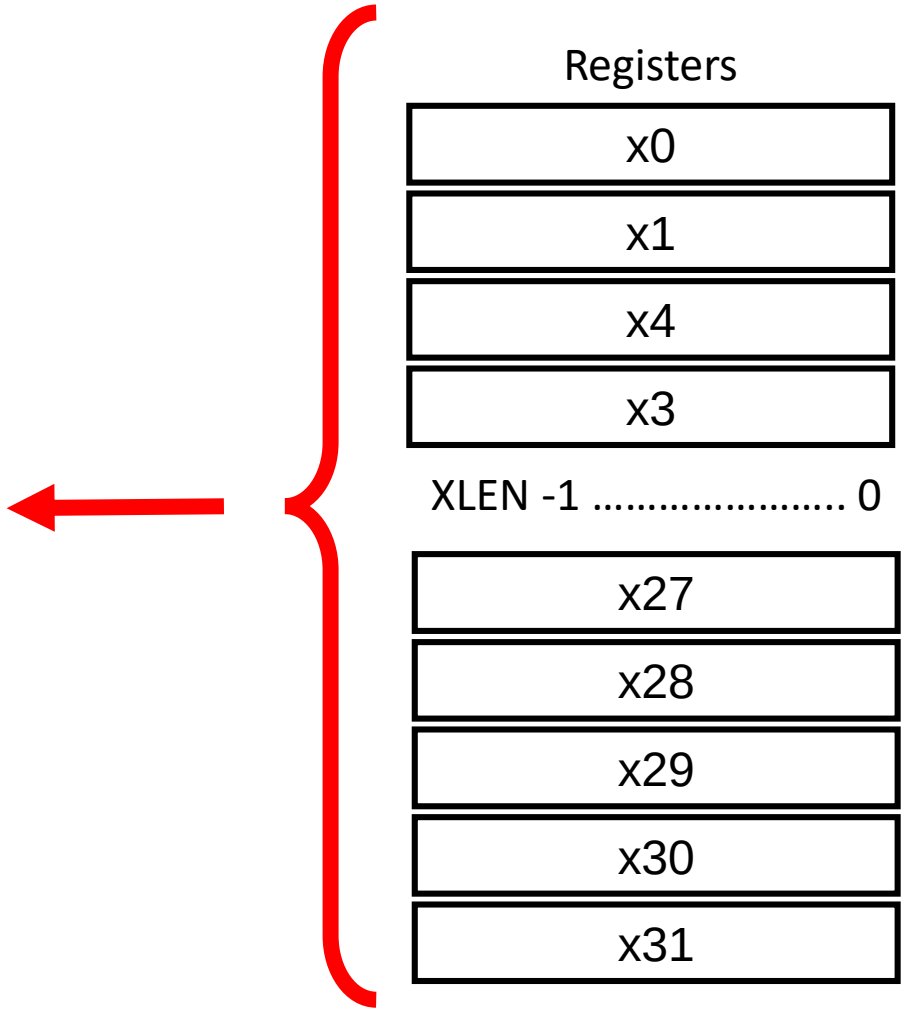
IUST
Iran University of Science and Technology

3 – Application Binary Interface(ABI)

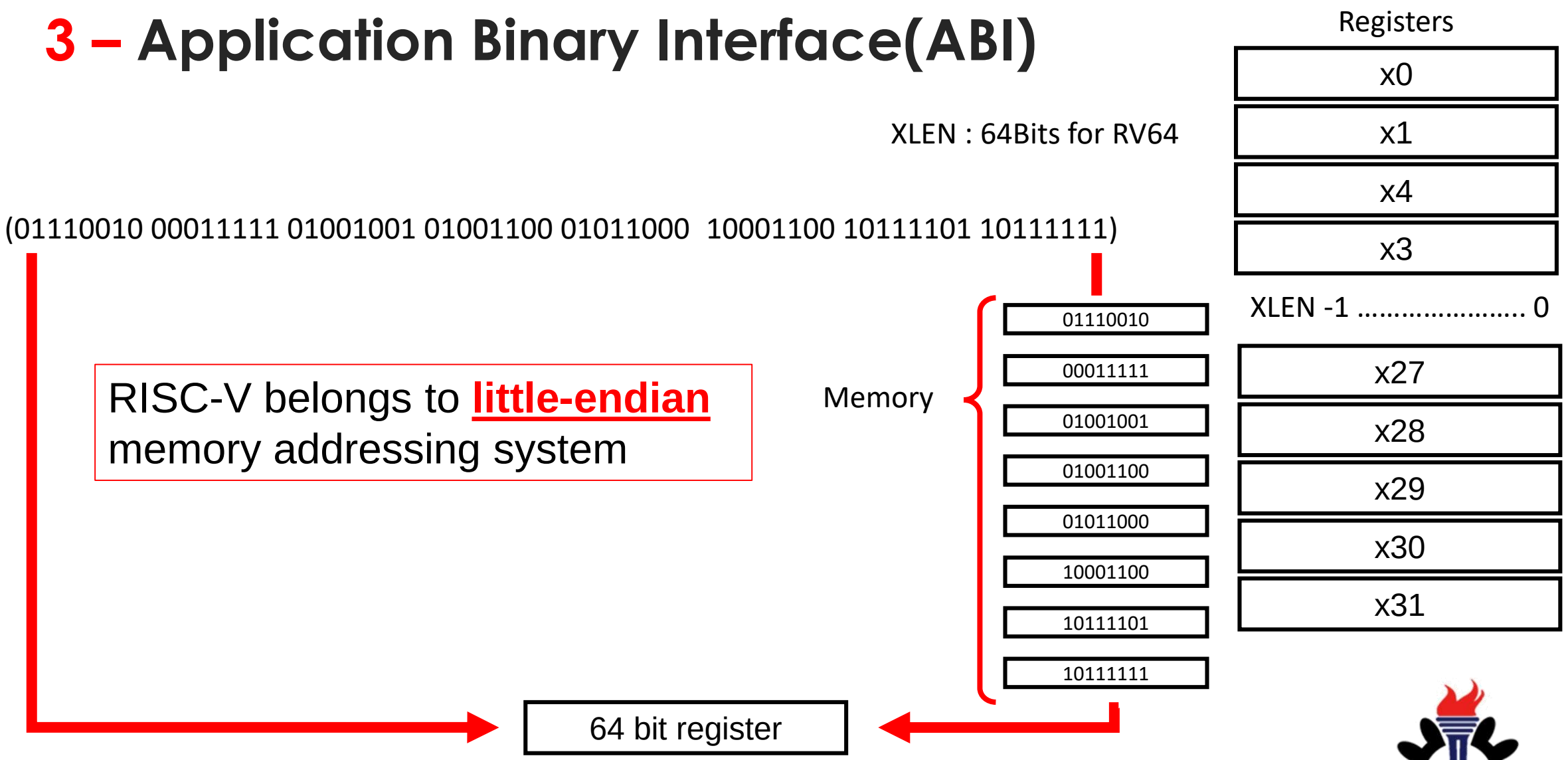


3 – Application Binary Interface(ABI)

Register	ABI name	Usage	Saver
x0	zero	Hard-wired zero	-
x1	ra	Return address	Caller
X2	sp	Stack pointer	Callee
x3	gp	Global pointer	-
x4	tp	Thread pointer	-
x5-x7	t0-2	Temporaries	Caller
x8	s0/fp	Saved register/frame pointer	Callee
x9	s1	Saved register	Callee
x10-11	a0-1	Function arguments/return values	Caller
x12-17	a2-7	Function arguments	Caller
x18-27	s2-11	Saved registers	Callee
x28-31	t3-6	Temporaries	Callee

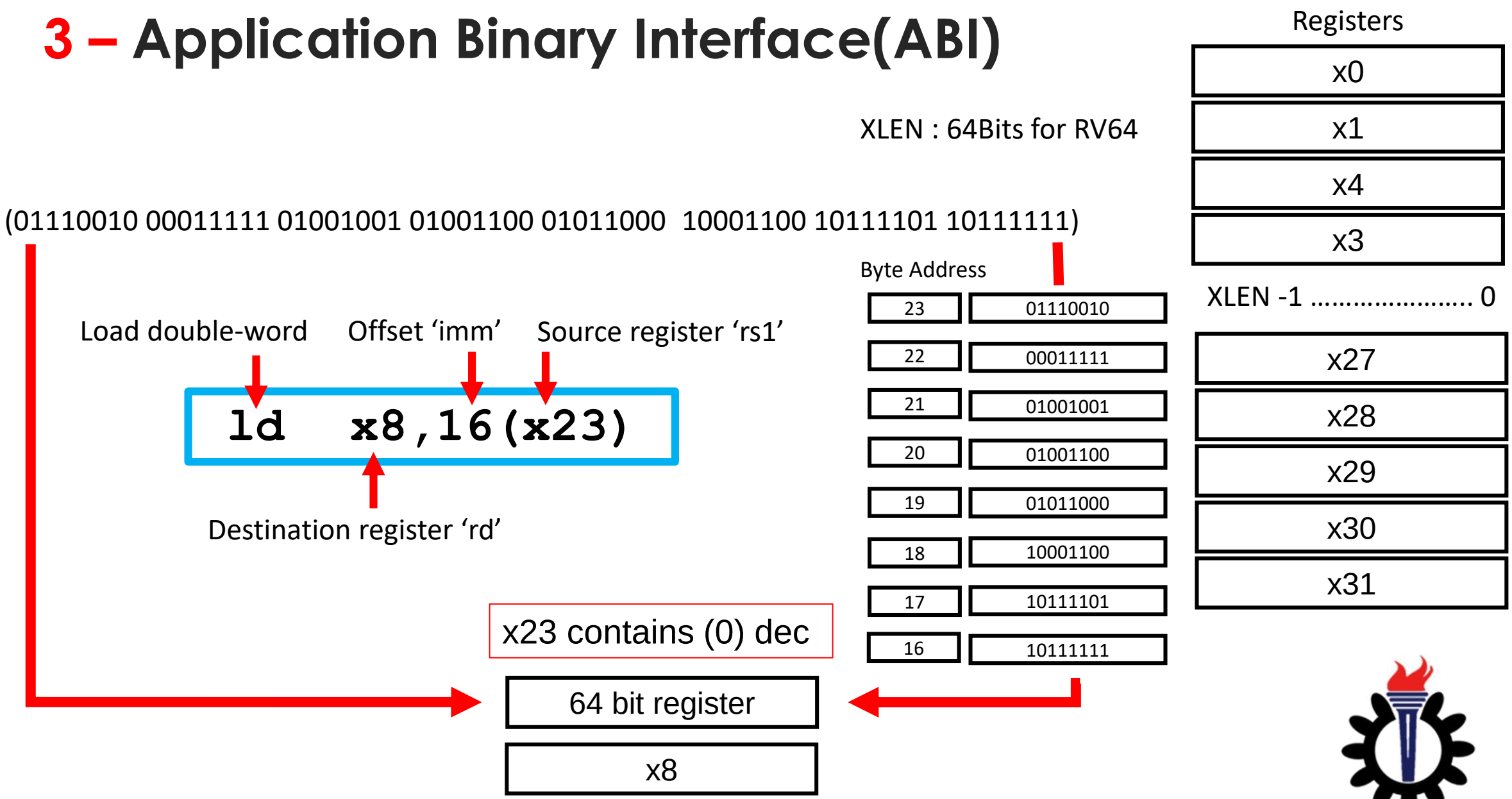


3 – Application Binary Interface(ABI)



IUST
Iran University of Science and Technology

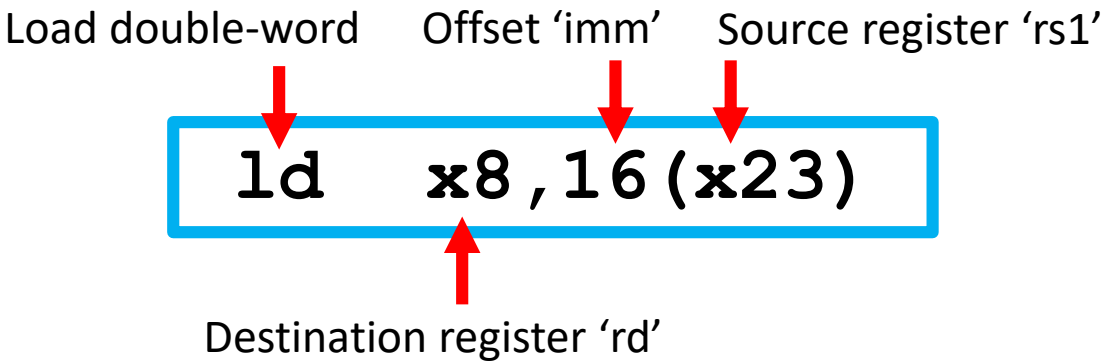
3 – Application Binary Interface(ABI)



IUST
Iran University of Science and Technology

3 – Application Binary Interface(ABI)

(01110010 00011111 01001001 01001100 01011000 10001100 10111101 10111111) XLEN : 64Bits for RV64



immediate	rs1	funct3	rd	opcode
31 30 29 28 27 26 25 24 23 22 21 20	19 18 17 16 15	14 13 12	11 10 9 8 7	6 5 4 3 2 1 0

3 – Application Binary Interface(ABI)

(01110010 00011111 01001001 01001100 01011000 10001100 10111101 10111111) XLEN : 64Bits for RV64

```
ld   x8,16(x23)
add   x8, x24, x8
```

Destination register ‘rd’ Source register ‘rs1’ Source register ‘rs2’

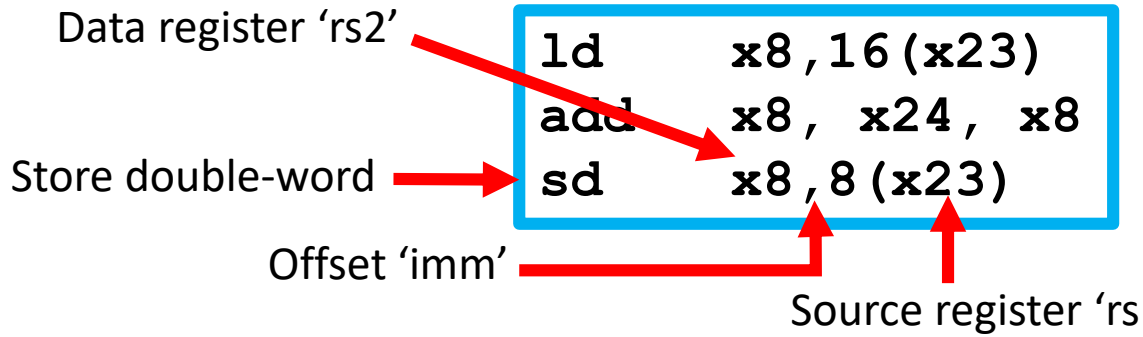
immediate			rs1	funct3	rd	opcode
31 30 29 28 27 26 25	24 23 22 21 20		19 18 17 16 15	14 13 12	11 10 9 8 7	6 5 4 3 2 1 0

funct7	rs2	rs1	funct3	rd	opcode
31 30 29 28 27 26 25	24 23 22 21 20	19 18 17 16 15	14 13 12	11 10 9 8 7	6 5 4 3 2 1 0

IUST
Iran University of Science and Technology

3 – Application Binary Interface(ABI)

(01110010 00011111 01001001 01001100 01011000 10001100 10111101 10111111) XLEN : 64Bits for RV64



immediate			rs1	funct3	rd	opcode
31 30 29 28 27 26 25	24 23 22 21 20		19 18 17 16 15	14 13 12	11 10 9 8 7	6 5 4 3 2 1 0

funct7	rs2	rs1	funct3	rd	opcode
31 30 29 28 27 26 25	24 23 22 21 20	19 18 17 16 15	14 13 12	11 10 9 8 7	6 5 4 3 2 1 0

immediate [11:5]	rs2	rs1	funct3	immediate[4:0]	opcode
31 30 29 28 27 26 25	24 23 22 21 20	19 18 17 16 15	14 13 12	11 10 9 8 7	6 5 4 3 2 1 0

3 – Application Binary Interface(ABI)

(01110010 00011111 01001001 01001100 01011000 10001100 10111101 10111111) XLEN : 64Bits for RV64

```
ld      x8,16(x23)
add     x8, x24, x8
sd      x8,8(x23)
```

Base integer instructions
RV64I

I-Type	immediate		rs1	funct3	rd	opcode
	31 30 29 28 27 26 25 24 23 22 21 20		19 18 17 16 15	14 13 12	11 10 9 8 7	6 5 4 3 2 1 0
R-Type	funct7	rs2	rs1	funct3	rd	opcode
	31 30 29 28 27 26 25	24 23 22 21 20	19 18 17 16 15	14 13 12	11 10 9 8 7	6 5 4 3 2 1 0
S-Type	immediate [11:5]	rs2	rs1	funct3	immediate[4:0]	opcode
	31 30 29 28 27 26 25	24 23 22 21 20	19 18 17 16 15	14 13 12	11 10 9 8 7	6 5 4 3 2 1 0

IUST
Iran University of Science and Technology

3 – Application Binary Interface(ABI)

I-Type : Registers + immediate

R-Type : only registers

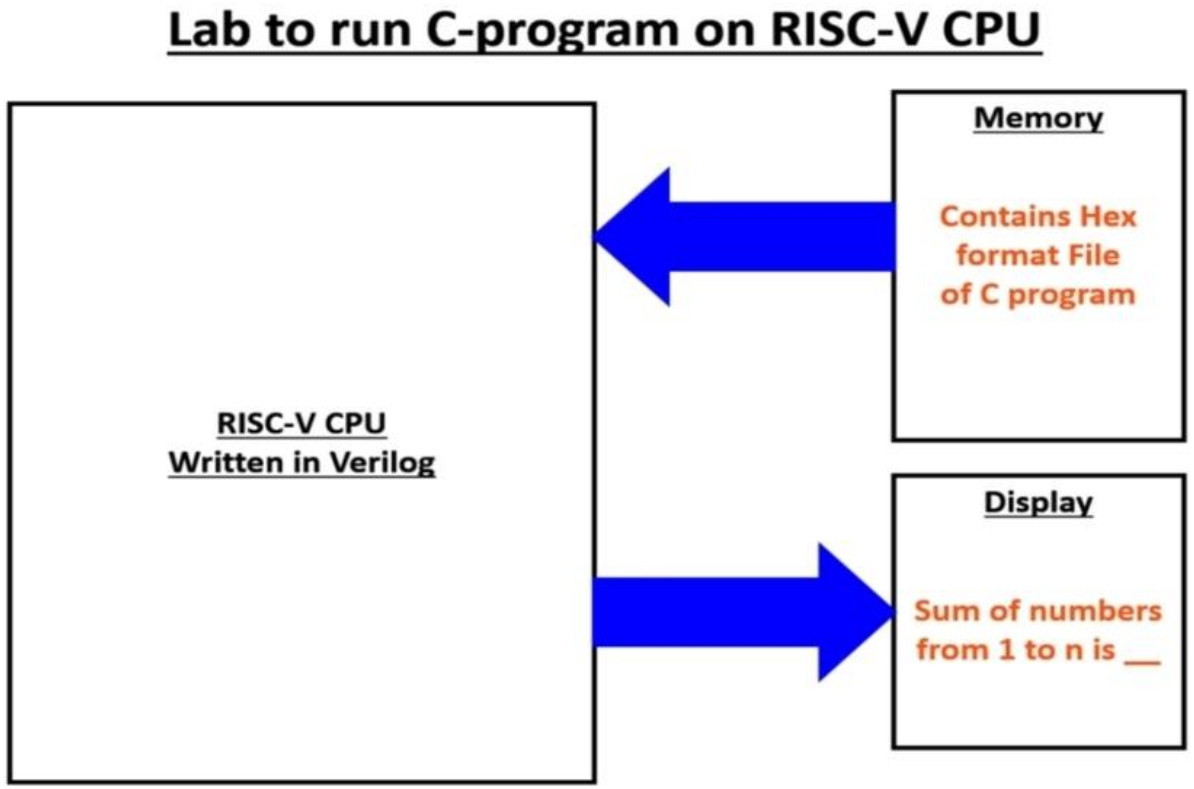
S-Type : only source registers

Base integer instructions
RV64I

I-Type	immediate		rs1	funct3	rd	opcode
	31 30 29 28 27 26 25 24 23 22 21 20		19 18 17 16 15	14 13 12	11 10 9 8 7	6 5 4 3 2 1 0
R-Type	funct7	rs2	rs1	funct3	rd	opcode
	31 30 29 28 27 26 25	24 23 22 21 20	19 18 17 16 15	14 13 12	11 10 9 8 7	6 5 4 3 2 1 0
S-Type	immediate [11:5]	rs2	rs1	funct3	immediate[4:0]	opcode
	31 30 29 28 27 26 25	24 23 22 21 20	19 18 17 16 15	14 13 12	11 10 9 8 7	6 5 4 3 2 1 0

IUST
Iran University of
Science and Technology

3 – Application Binary Interface(ABI)



IUST
Iran University of Science and Technology

3 – Application Binary Interface(ABI)

```
for (i = 0; i < 10; i++)
    printf("go cucs");
```

```
main:   addi x2, x0, 10
        addi x1, x0, 0
loop:   slt x3, x1, x2
        ...
```

10 x2 x0 op=addi

```
00000000101000010000000000010011
00100000000000010000000000010000
00000000001000100001100000101010
```

ALU, Control, Register File, …

High Level Language

- C, Java, Python, ADA, …
- Loops, control flow, variables

Assembly Language

- No symbols (except labels)
- One operation per statement
- "human readable machine language"

Machine Language

- Binary-encoded assembly
- Labels become addresses
- **The language of the CPU**

Machine Implementation (Microarchitecture)

References :

1) David A. Patterson, John L. Hennessy -Computer Organization and Design RISC-V Edition

2) A Practical Approach to VLSI System on Chip (SoC) Design

3) https://riscv.org/wp-content/uploads/2017/05/riscv-spec-v2.2.pdf

4) https://www.cs.cornell.edu/courses/cs3410/2019sp/schedule/slides/06-cpu-notes-bw.pdf

5) https://www.vlsisystemdesign.com/blogs/

6) https://www2.eecs.berkeley.edu/Pubs/TechRpts/2014/EECS-2014-54.pdf

7) VSD-IAT Workshop 2022 – MYTH (By Redwood EDA & VSD)