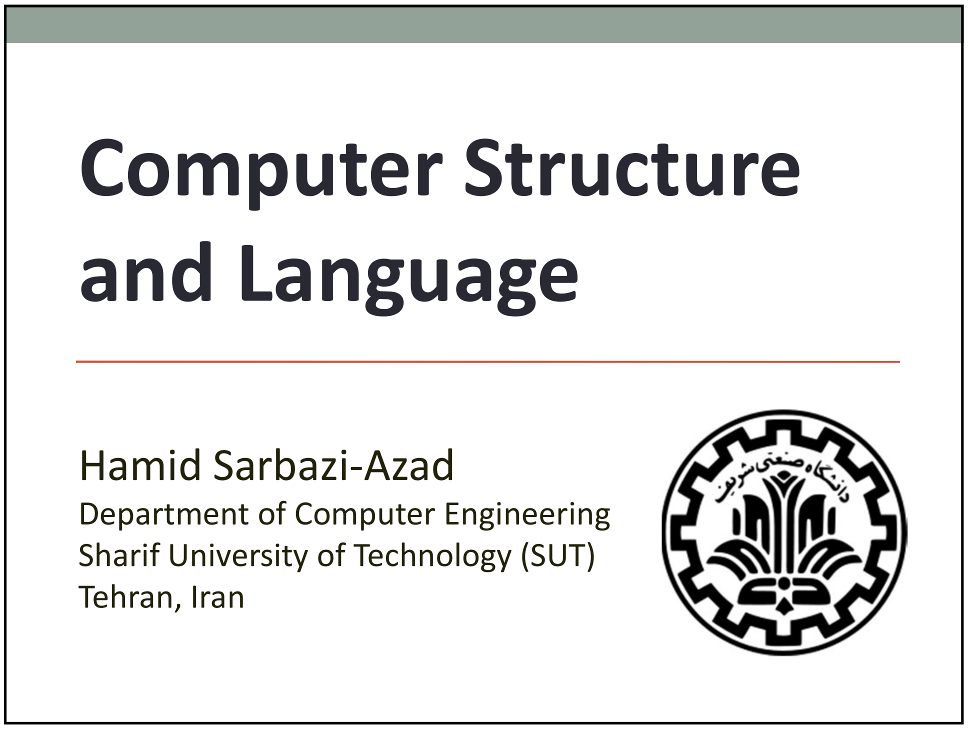
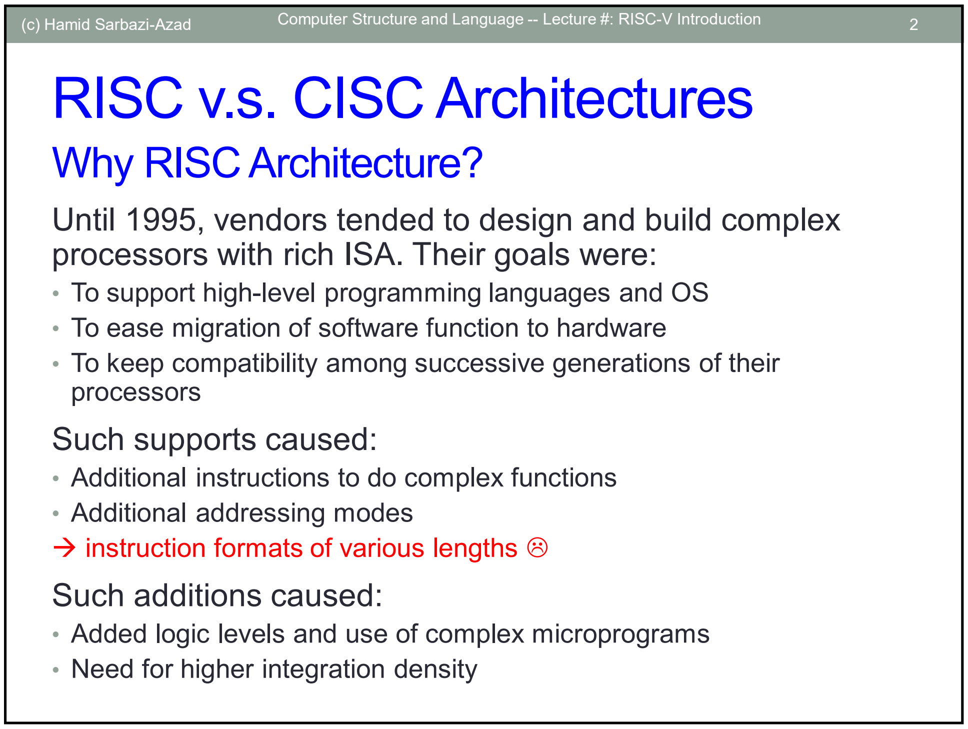


Computer Structure and Language

Hamid Sarbazi-Azad
Department of Computer Engineering
Sharif University of Technology (SUT)
Tehran, Iran

1



RISC v.s. CISC Architectures

Why RISC Architecture?

Until 1995, vendors tended to design and build complex processors with rich ISA. Their goals were:

- To support high-level programming languages and OS
- To ease migration of software function to hardware
- To keep compatibility among successive generations of their processors

Such supports caused:

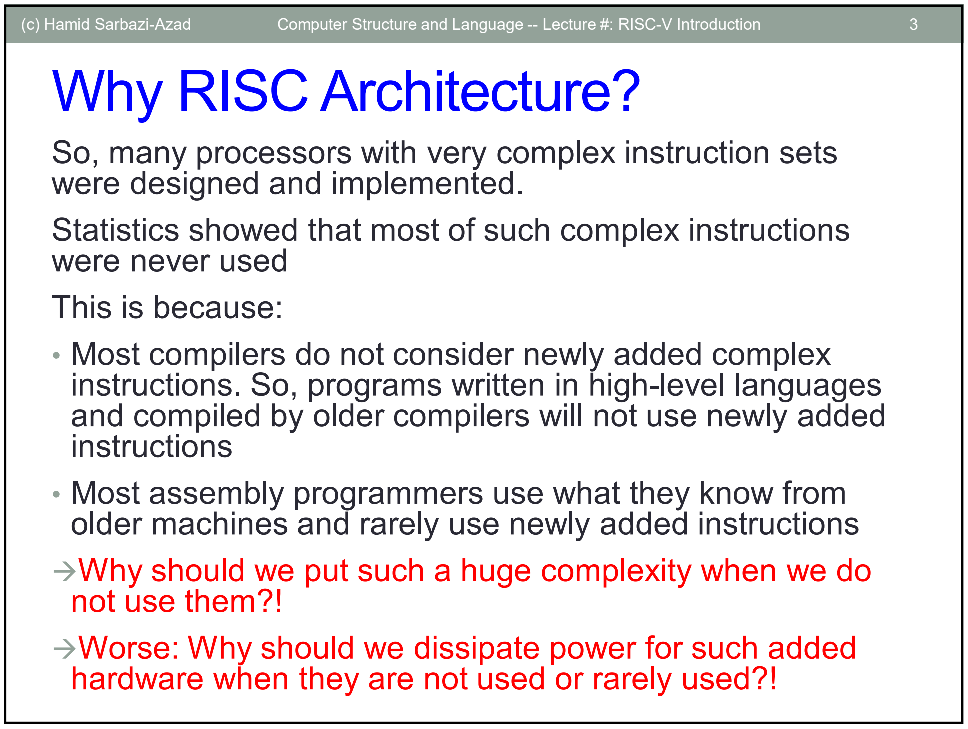
- Additional instructions to do complex functions
- Additional addressing modes

→ instruction formats of various lengths ☹

Such additions caused:

- Added logic levels and use of complex microprograms
- Need for higher integration density

2



Why RISC Architecture?

So, many processors with very complex instruction sets were designed and implemented.

Statistics showed that most of such complex instructions were never used

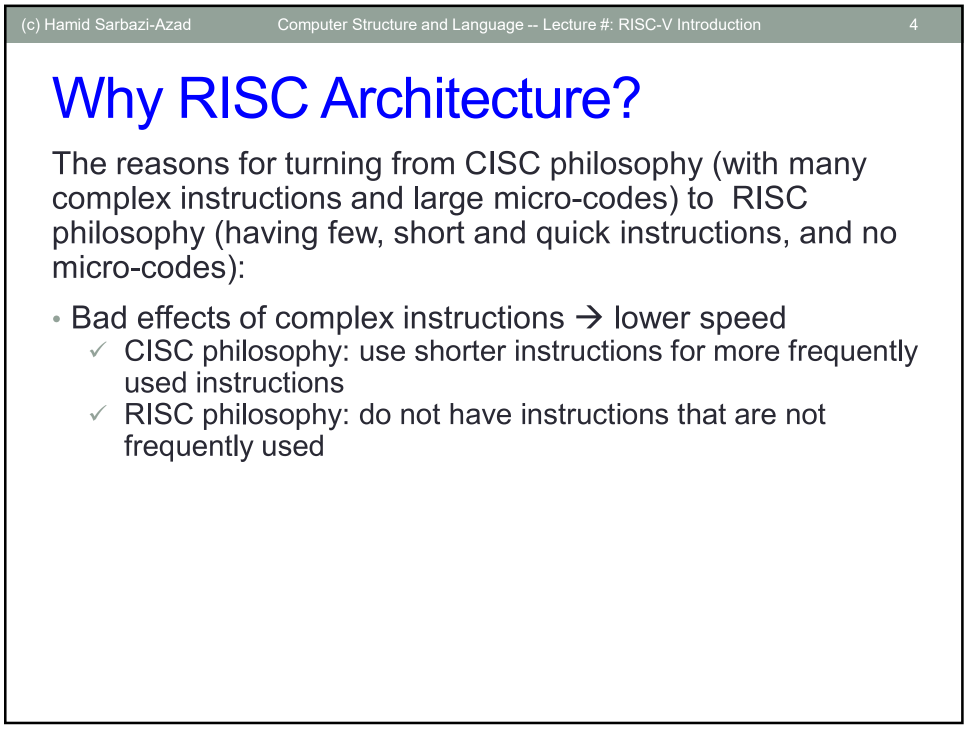
This is because:

- Most compilers do not consider newly added complex instructions. So, programs written in high-level languages and compiled by older compilers will not use newly added instructions
- Most assembly programmers use what they know from older machines and rarely use newly added instructions

→Why should we put such a huge complexity when we do not use them?!

→Worse: Why should we dissipate power for such added hardware when they are not used or rarely used?!

3

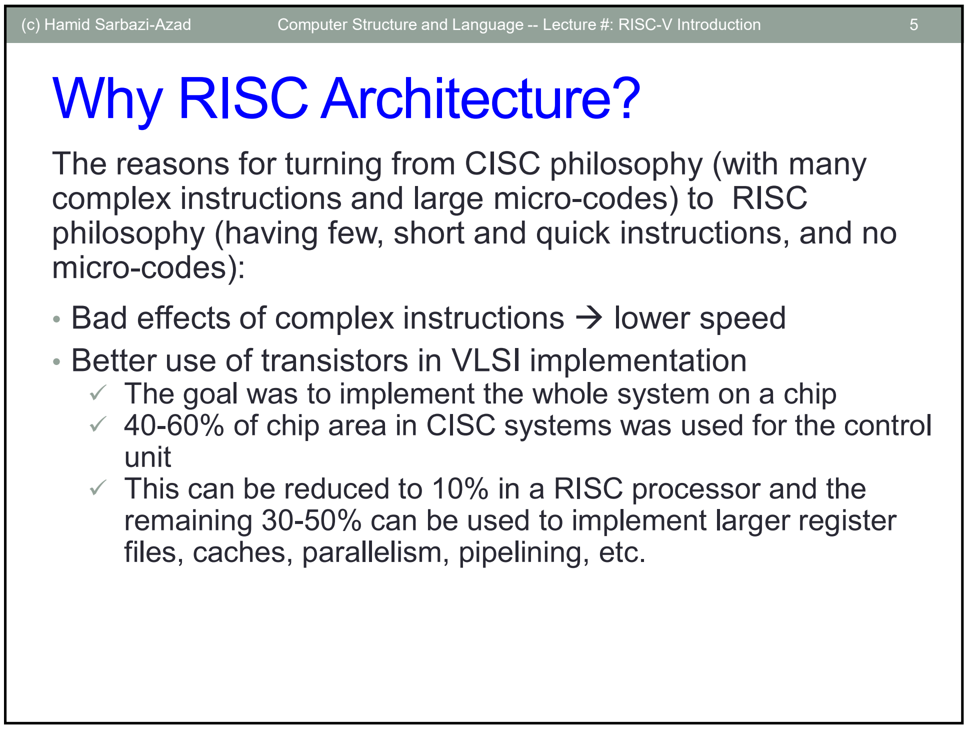


Why RISC Architecture?

The reasons for turning from CISC philosophy (with many complex instructions and large micro-codes) to RISC philosophy (having few, short and quick instructions, and no micro-codes):

- Bad effects of complex instructions → lower speed
  - ✓ CISC philosophy: use shorter instructions for more frequently used instructions
  - ✓ RISC philosophy: do not have instructions that are not frequently used

4

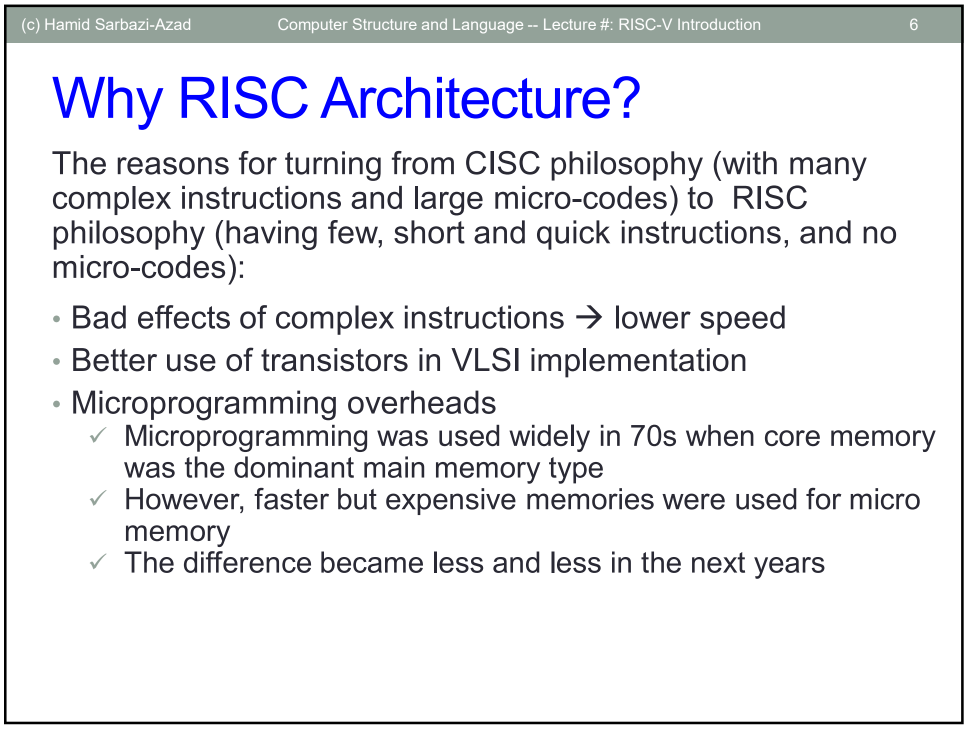


Why RISC Architecture?

The reasons for turning from CISC philosophy (with many complex instructions and large micro-codes) to RISC philosophy (having few, short and quick instructions, and no micro-codes):

- Bad effects of complex instructions → lower speed
- Better use of transistors in VLSI implementation
  - ✓ The goal was to implement the whole system on a chip
  - ✓ 40-60% of chip area in CISC systems was used for the control unit
  - ✓ This can be reduced to 10% in a RISC processor and the remaining 30-50% can be used to implement larger register files, caches, parallelism, pipelining, etc.

5

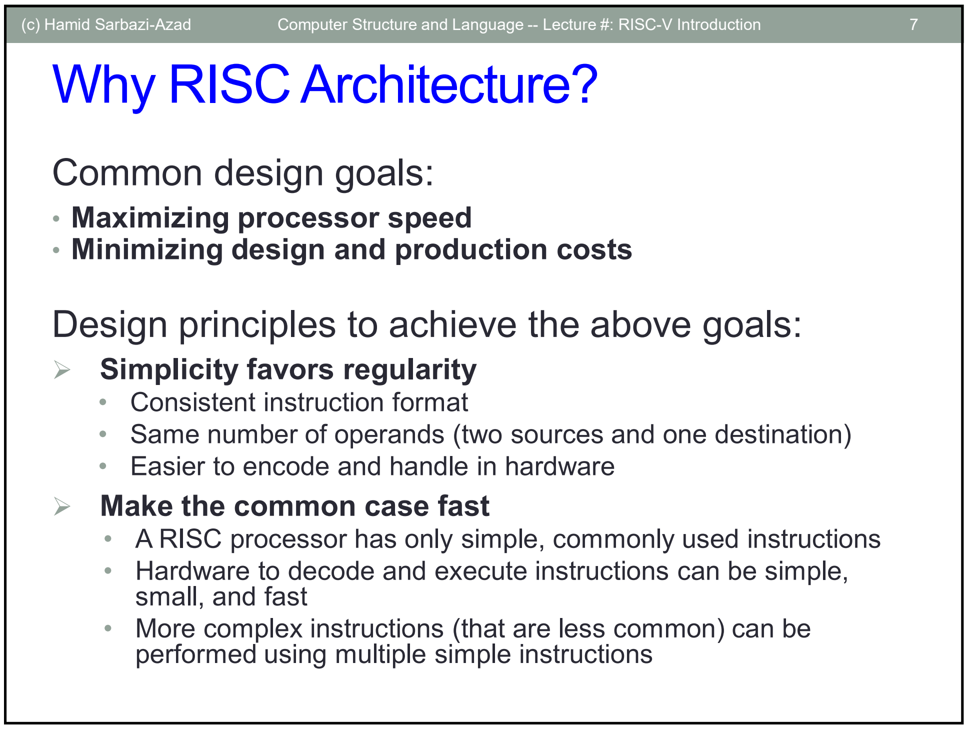


Why RISC Architecture?

The reasons for turning from CISC philosophy (with many complex instructions and large micro-codes) to RISC philosophy (having few, short and quick instructions, and no micro-codes):

- Bad effects of complex instructions → lower speed
- Better use of transistors in VLSI implementation
- Microprogramming overheads
  - ✓ Microprogramming was used widely in 70s when core memory was the dominant main memory type
  - ✓ However, faster but expensive memories were used for micro memory
  - ✓ The difference became less and less in the next years

6



Why RISC Architecture?

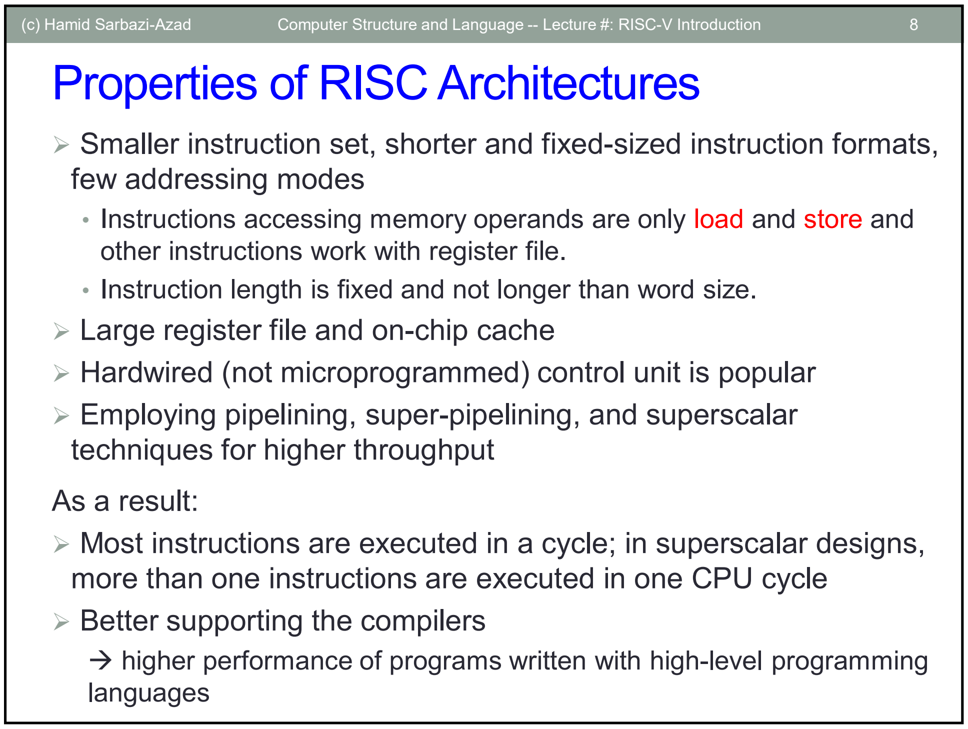
Common design goals:

- **Maximizing processor speed**
- **Minimizing design and production costs**

Design principles to achieve the above goals:

- **Simplicity favors regularity**
  - Consistent instruction format
  - Same number of operands (two sources and one destination)
  - Easier to encode and handle in hardware
- **Make the common case fast**
  - A RISC processor has only simple, commonly used instructions
  - Hardware to decode and execute instructions can be simple, small, and fast
  - More complex instructions (that are less common) can be performed using multiple simple instructions

7



Properties of RISC Architectures

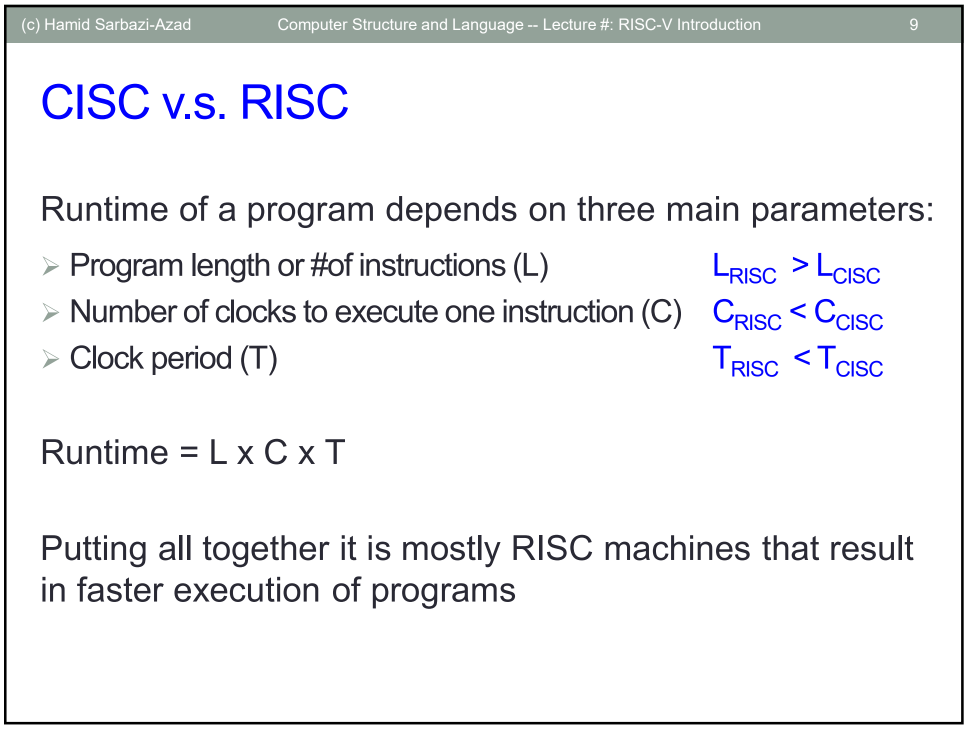
- Smaller instruction set, shorter and fixed-sized instruction formats, few addressing modes
  - Instructions accessing memory operands are only load and store and other instructions work with register file.
  - Instruction length is fixed and not longer than word size.
- Large register file and on-chip cache
- Hardwired (not microprogrammed) control unit is popular
- Employing pipelining, super-pipelining, and superscalar techniques for higher throughput

As a result:

- Most instructions are executed in a cycle; in superscalar designs, more than one instructions are executed in one CPU cycle
- Better supporting the compilers

  → higher performance of programs written with high-level programming languages

8



CISC v.s. RISC

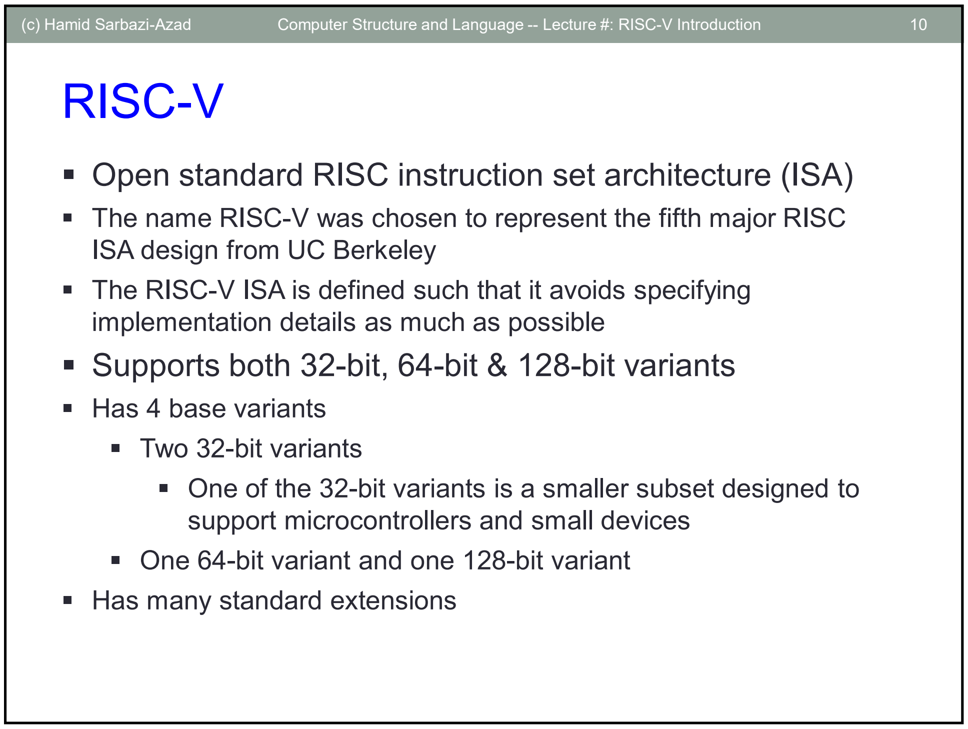
Runtime of a program depends on three main parameters:

➢ Program length or #of instructions (L)	$L_{RISC} > L_{CISC}$
➢ Number of clocks to execute one instruction (C)	$C_{RISC} < C_{CISC}$
➢ Clock period (T)	$T_{RISC} < T_{CISC}$

Runtime = L x C x T

Putting all together it is mostly RISC machines that result in faster execution of programs

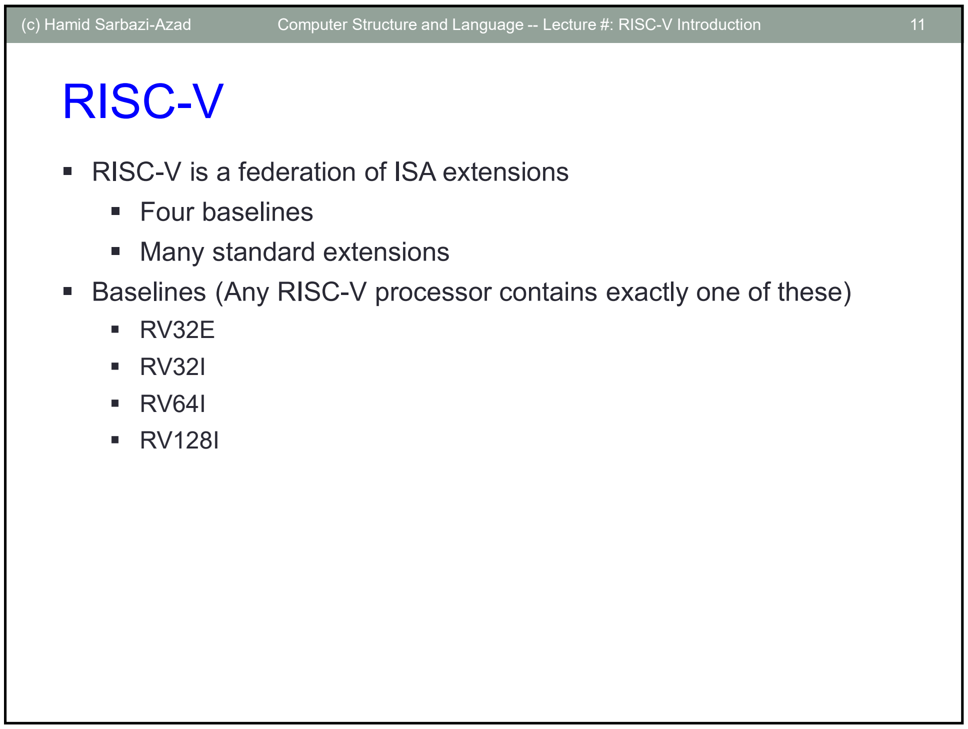
9



RISC-V

- Open standard RISC instruction set architecture (ISA)
- The name RISC-V was chosen to represent the fifth major RISC ISA design from UC Berkeley
- The RISC-V ISA is defined such that it avoids specifying implementation details as much as possible
- Supports both 32-bit, 64-bit & 128-bit variants
- Has 4 base variants
  - Two 32-bit variants
    - One of the 32-bit variants is a smaller subset designed to support microcontrollers and small devices
  - One 64-bit variant and one 128-bit variant
- Has many standard extensions

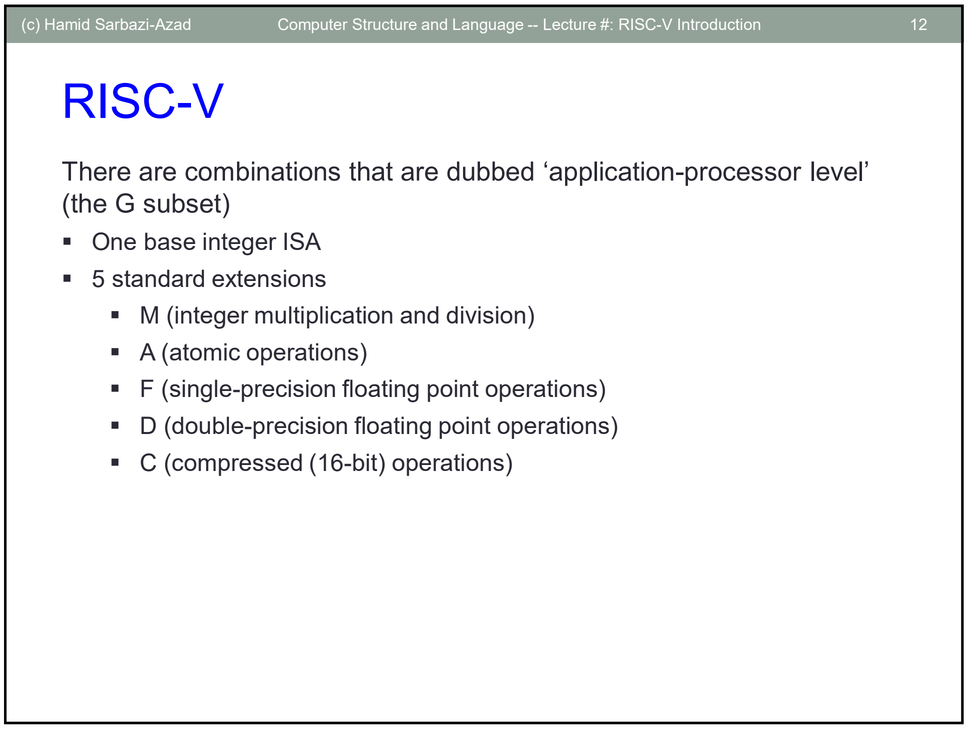
10



RISC-V

- RISC-V is a federation of ISA extensions
  - Four baselines
  - Many standard extensions
- Baselines (Any RISC-V processor contains exactly one of these)
  - RV32E
  - RV32I
  - RV64I
  - RV128I

11



RISC-V

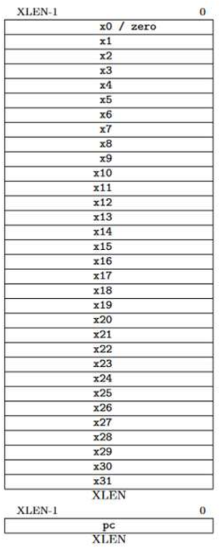
There are combinations that are dubbed ‘application-processor level’ (the G subset)

- One base integer ISA
- 5 standard extensions
  - M (integer multiplication and division)
  - A (atomic operations)
  - F (single-precision floating point operations)
  - D (double-precision floating point operations)
  - C (compressed (16-bit) operations)

12

(c) Hamid Sarbazi-Azad — Computer Structure and Language -- Lecture #: RISC-V Introduction — 13

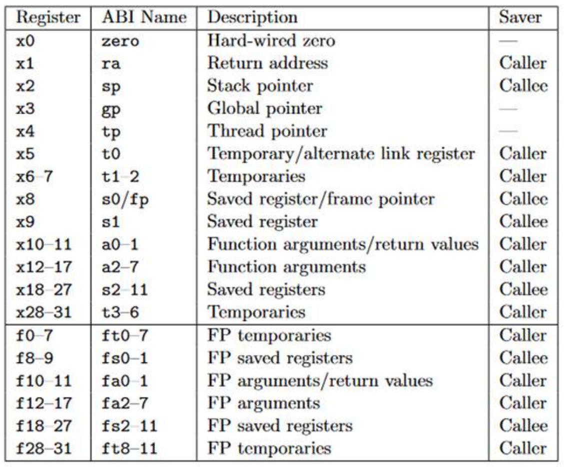
RISC-V - Registers



13

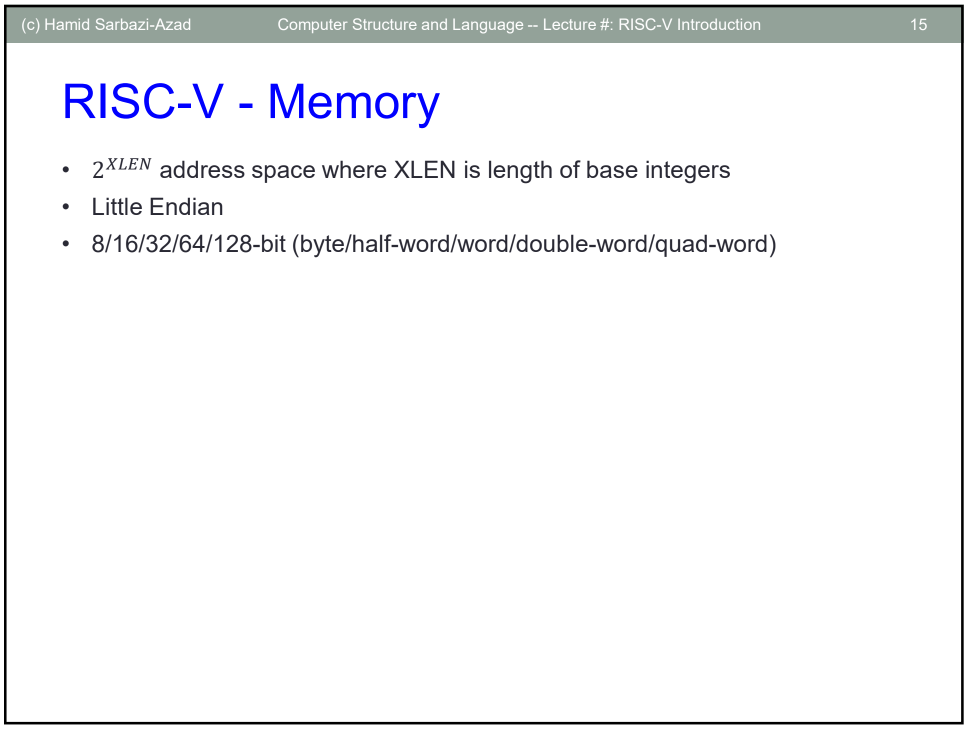
(c) Hamid Sarbazi-Azad — Computer Structure and Language -- Lecture #: RISC-V Introduction — 14

RISC-V - Registers



Register	ABI Name	Description	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	t0	Temporary/alternate link register	Caller
x6–7	t1–2	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	Caller
f0–7	ft0–7	FP temporaries	Caller
f8–9	fs0–1	FP saved registers	Callee
f10–11	fa0–1	FP arguments/return values	Caller
f12–17	fa2–7	FP arguments	Caller
f18–27	fs2–11	FP saved registers	Callee
f28–31	ft8–11	FP temporaries	Caller

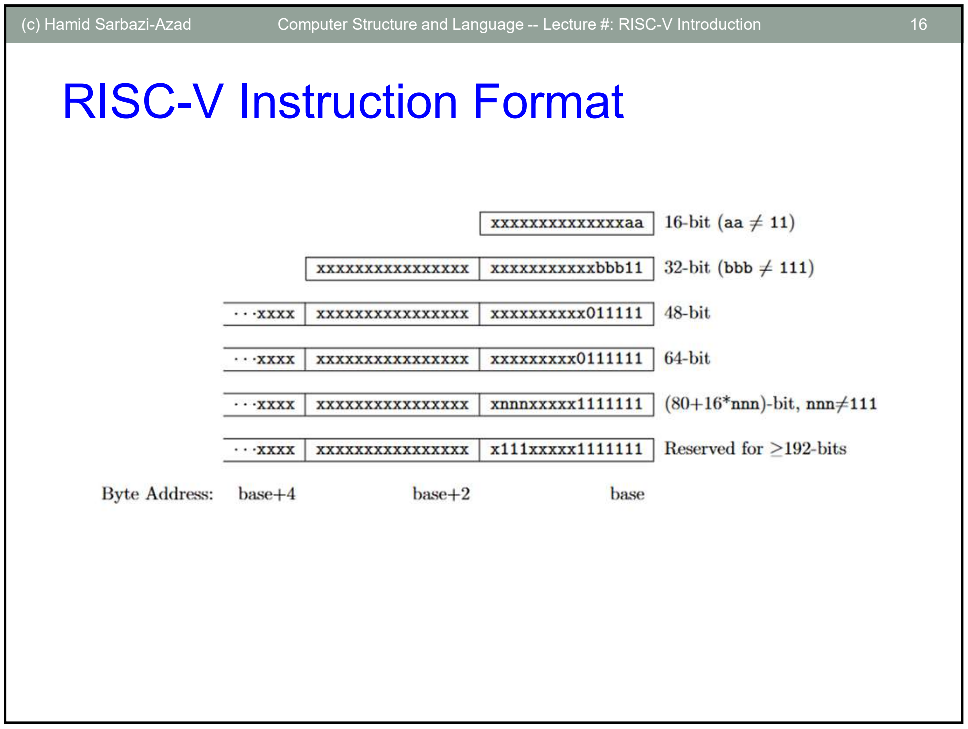
14



RISC-V - Memory

- 2^{XLEN} address space where XLEN is length of base integers
- Little Endian
- 8/16/32/64/128-bit (byte/half-word/word/double-word/quad-word)

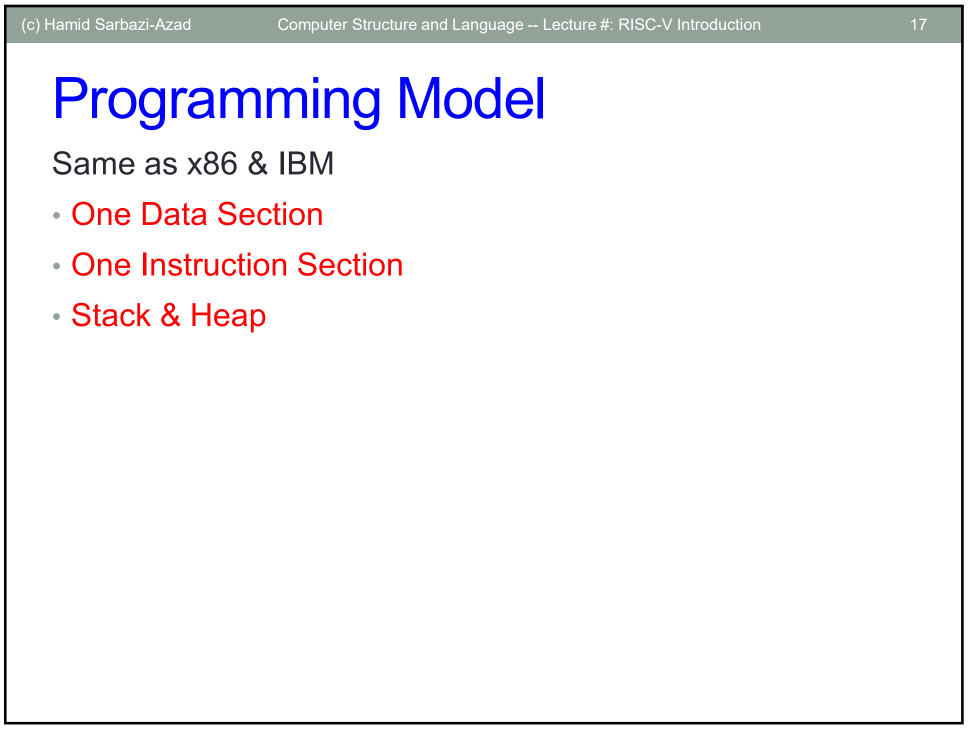
15



RISC-V Instruction Format

			xxxxxxxxxxxxxxaa	16-bit (aa $\neq$ 11)
		xxxxxxxxxxxxxxxx	xxxxxxxxxxxbbb11	32-bit (bbb $\neq$ 111)
	···xxxx	xxxxxxxxxxxxxxxx	xxxxxxxxxx011111	48-bit
	···xxxx	xxxxxxxxxxxxxxxx	xxxxxxxxx0111111	64-bit
	···xxxx	xxxxxxxxxxxxxxxx	xnnnxxxxx1111111	(80+16*nnn)-bit, nnn$\neq$111
	···xxxx	xxxxxxxxxxxxxxxx	x111xxxxx1111111	Reserved for $\geq$192-bits
Byte Address:	base+4	base+2	base	

16

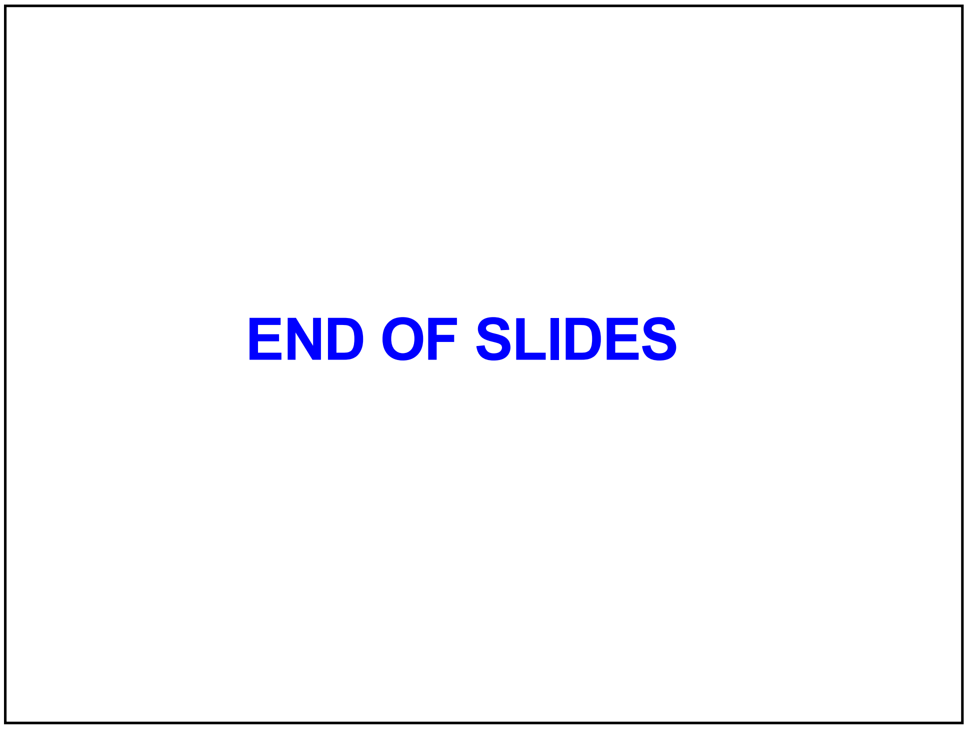


Programming Model

Same as x86 & IBM

- One Data Section
- One Instruction Section
- Stack & Heap

17



END OF SLIDES

18