Saeed Safari,

Advanced Computer Architecture

Introduction

ECE School,

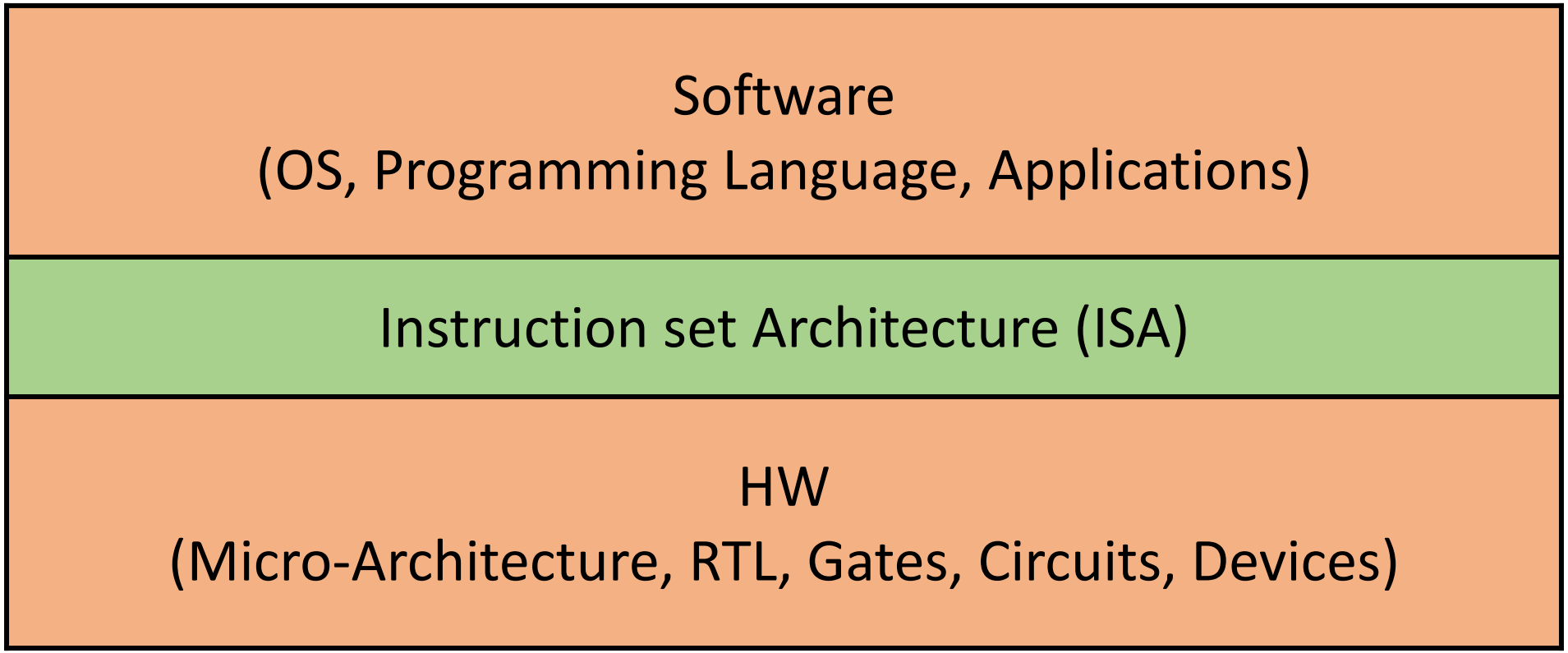
Faculty of Engineering,

University of Tehran

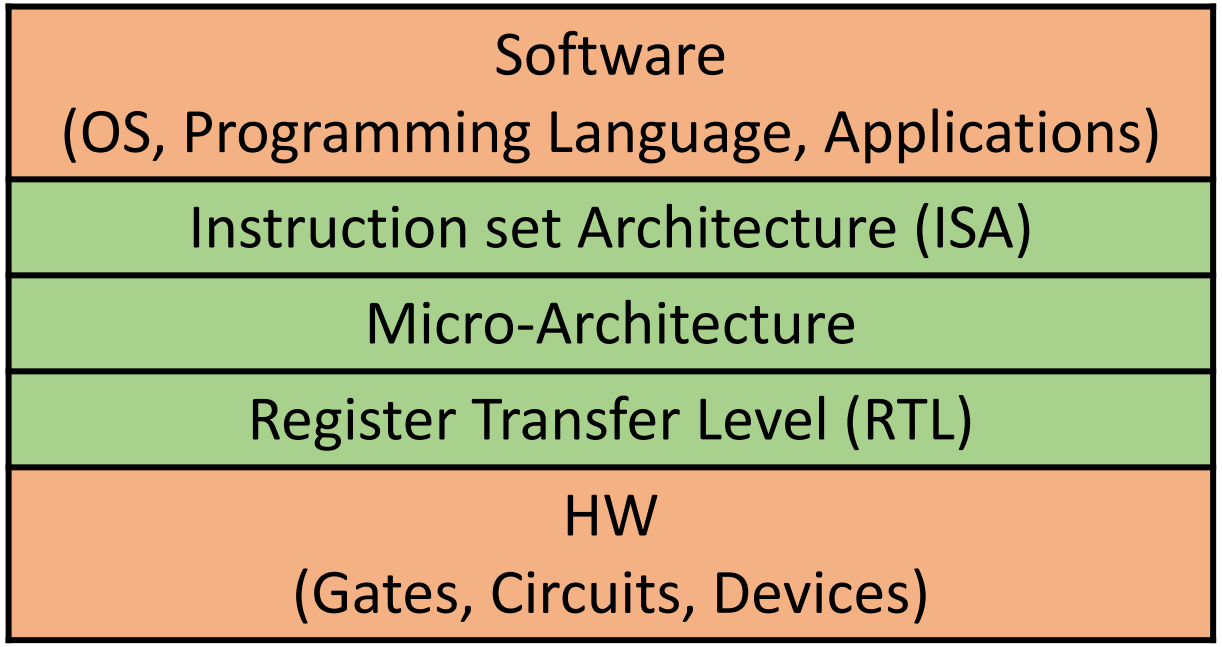
Outline

- Architecture
  - CPU classes
  - Addressing modes
  - Operands type and size
  - Operations
  - Instruction encoding
- Micro-Architecture

Computing Systems Abstraction Levels

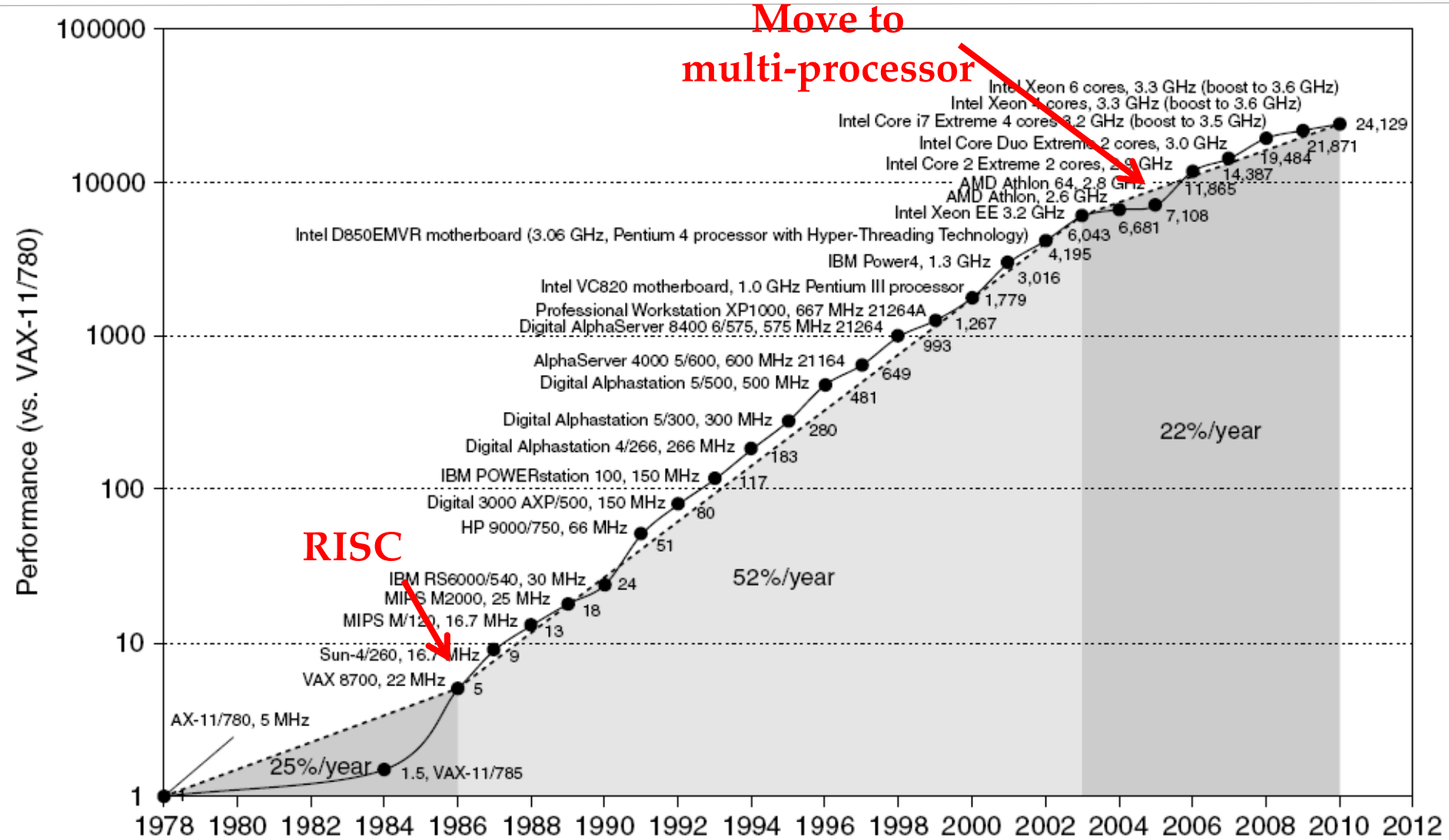


Computing Systems Abstraction Levels



Computer Architecture
and
Advanced Computer Architectrure

Single Processor Performance



Saeed Safari,
ECE School, Faculty of Engineering,
University of Tehran

Adopted from Hennessy and Patterson, 6th Ed.

Architecture vs. Micro-Architecture

- Architecture / Instruction Set Architecture (ISA)
  - Programmer visible state (Registers, Memory)
  - Data types (Integer, FP)
  - Addressing modes
  - Operations
  - Instruction encoding
  - …

Architecture vs. Micro-Architecture

- Micro-Architecture / Organization
  - Tradeoffs on ISA implementation (Speed, cost, power, …)
  - **Examples:**
    - Pipeline depth,
    - Number of pipelines,
    - ALU width,
    - Cache size,
    - Execution ordering,
    - …

Same Architecture Different Micro-Architecture

- **Intel Atom**
  - X86 Instruction Set
  - Single Core
  - 2 W
  - Decode 2 Instructions/Cycle/Core
  - 32 KB L1 I Cache, 32 KB L1 D Cache
  - 512 KB L2 Cache
  - In-Order
  - 47,000,000 Tr.
  - 1.6 GHz

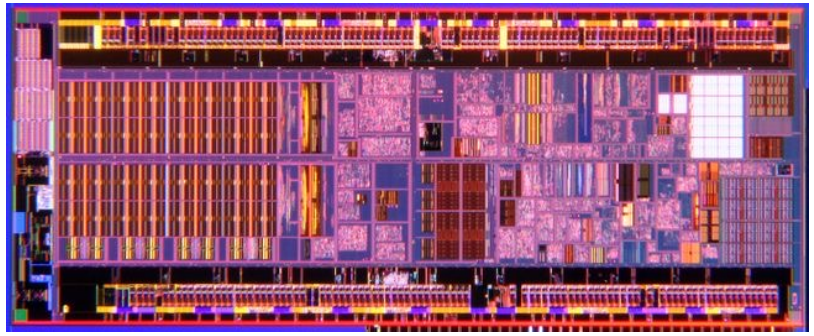


Image Credit: Intel

- **AMD Phenom X4**
  - X86 Instruction Set
  - Quad Core
  - 125 W
  - Decode 3 Instructions/Cycle/Core
  - 64 KB L1 I Cache, 64 KB L1 D Cache
  - 512 KB L2 Cache
  - Out-Of-Order
  - 758,000,000 Tr.
  - 2.6 GHz

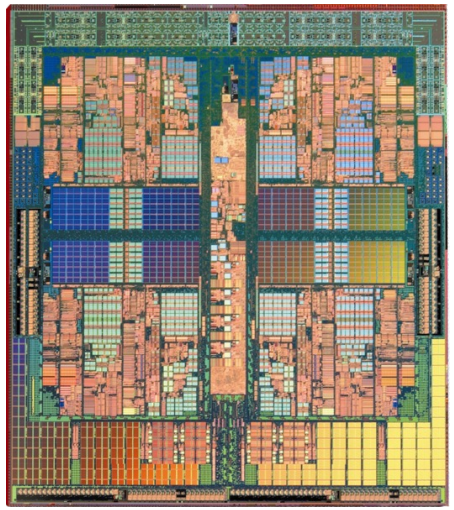


Image Credit: AMD

Different Architecture Different Micro-Architecture

- **IBM POWER7**
  - Power Instruction Set
  - Eight Core
  - 200 W
  - Decode 6 Instructions/Cycle/Core
  - 32 KB L1 I Cache, 32 KB L1 D Cache
  - 256 KB L2 Cache
  - Out-Of-Order
  - 1,200,000,000 Tr.
  - 4.25 GHz

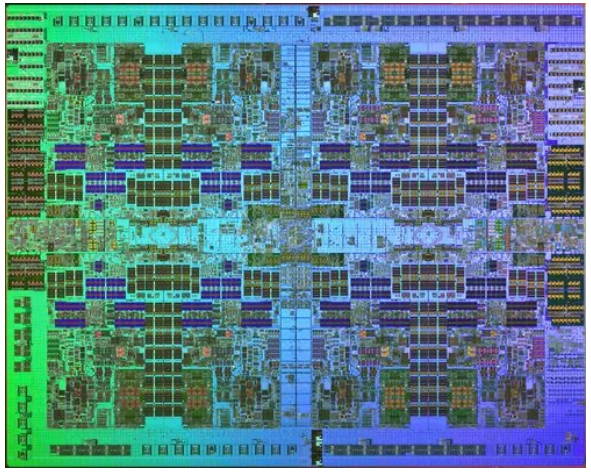


Image Credit: IBM

- **AMD Phenom X4**
  - X86 Instruction Set
  - Quad Core
  - 125 W
  - Decode 3 Instructions/Cycle/Core
  - 64 KB L1 I Cache, 64 KB L1 D Cache
  - 512 KB L2 Cache
  - Out-Of-Order
  - 758,000,000 Tr.
  - 2.6 GHz

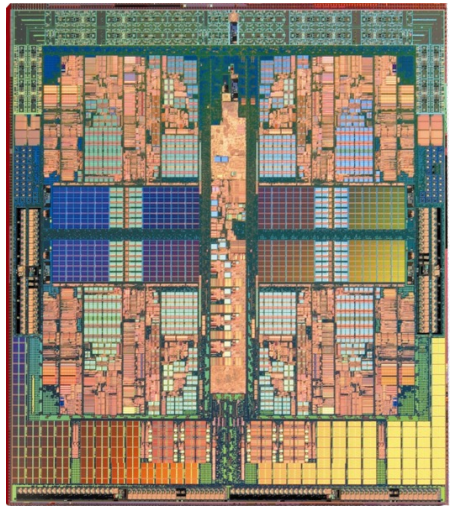


Image Credit: AMD

Architecture

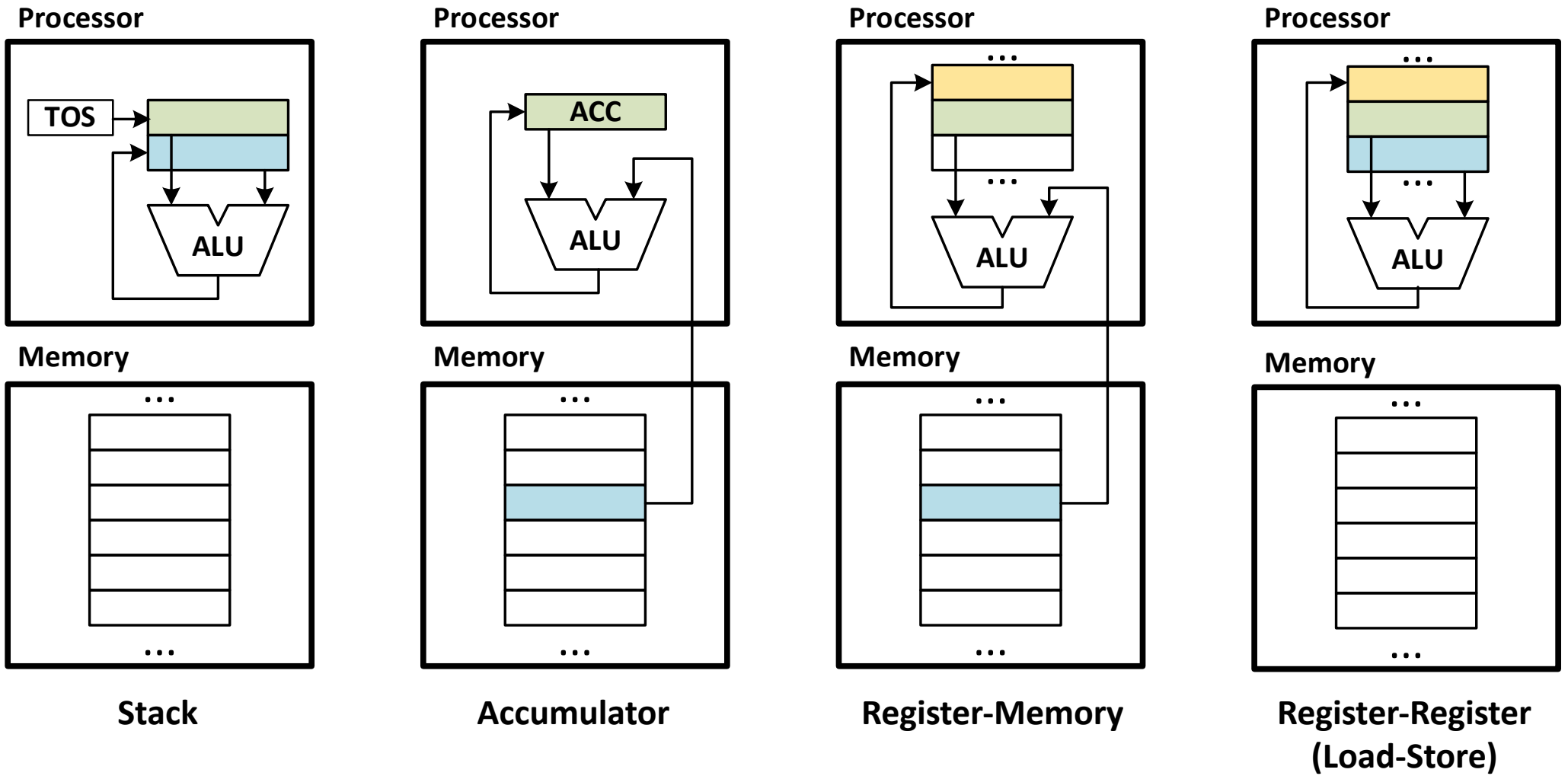
- **Complete:** should be able to implement all user program
- **Concise:** limited size of the instruction set
- **Generic:** instructions should capture the common case
- **Simple:** instructions should be simple

  - It is necessary to be complete, whereas the rest of the properties are desirable

Architecture

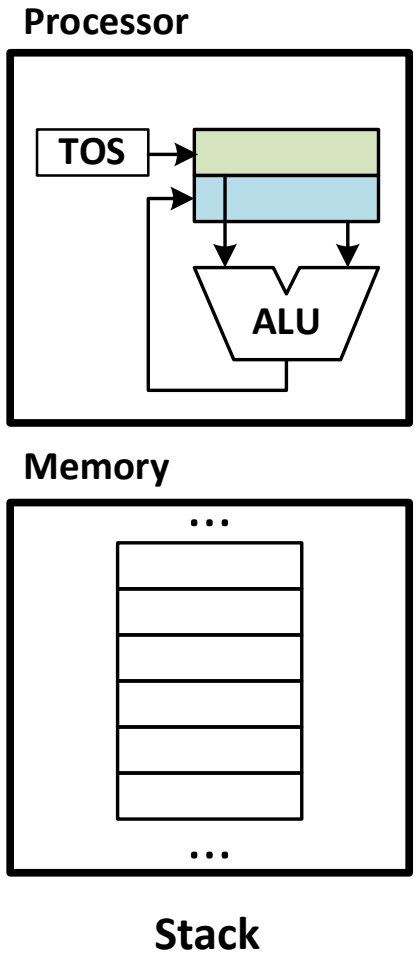
- Machine model
- Memory addressing
- Type and size of operands
- Operations
- Encoding

Machine Models



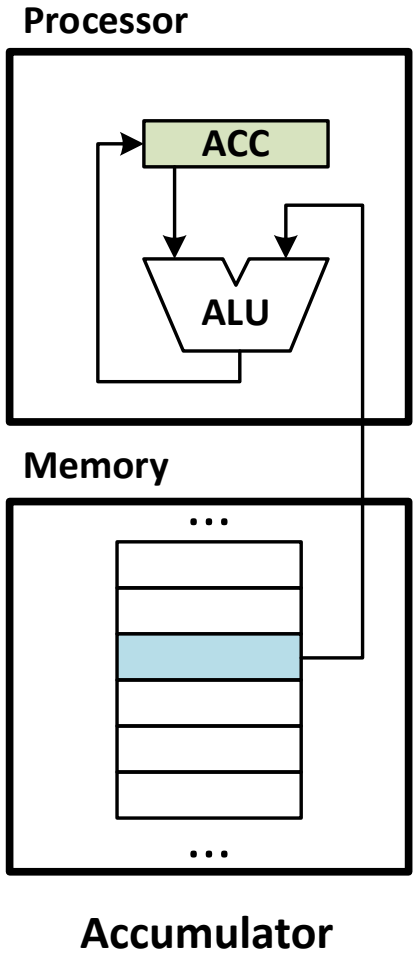
Saeed Safari,
ECE School, Faculty of Engineering,
University of Tehran

Machine Models



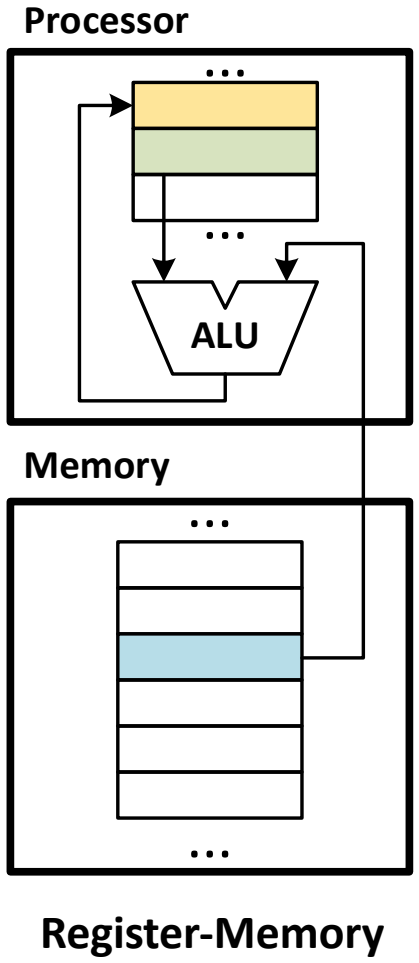

```
C = A + B;


PUSH    A
PUSH    B
ADD
POP     C
```

Machine Models




```
C = A + B;


LOAD    A
ADD     B
STORE   C
```

Machine Models

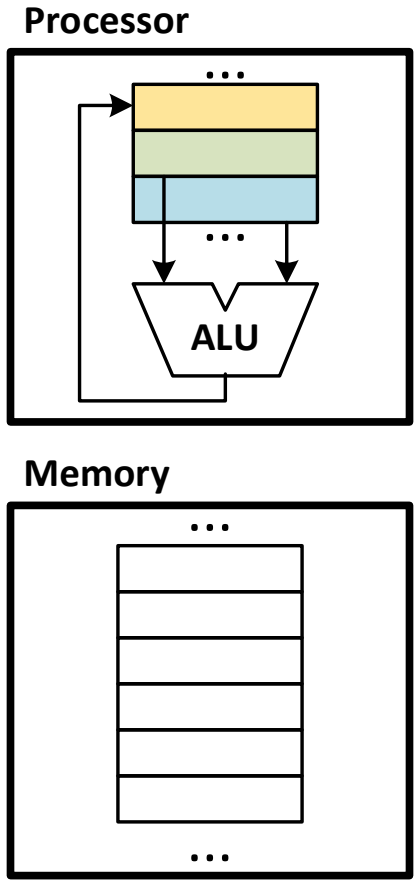


Register-Memory

```
C = A + B;
```

```
LOAD    R1,A
ADD     R3, R1, B
STORE   R3, C
```

Machine Models



Register-Register
(Load-Store)

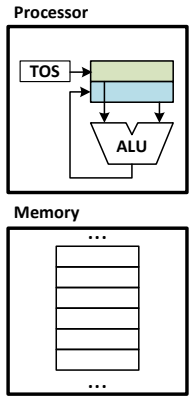
```
C = A + B;


LOAD    R1, A
LOAD    R2, B
ADD     R3, R1, R2
STORE   R3, C
```

Saeed Safari,
ECE School, Faculty of Engineering,
University of Tehran

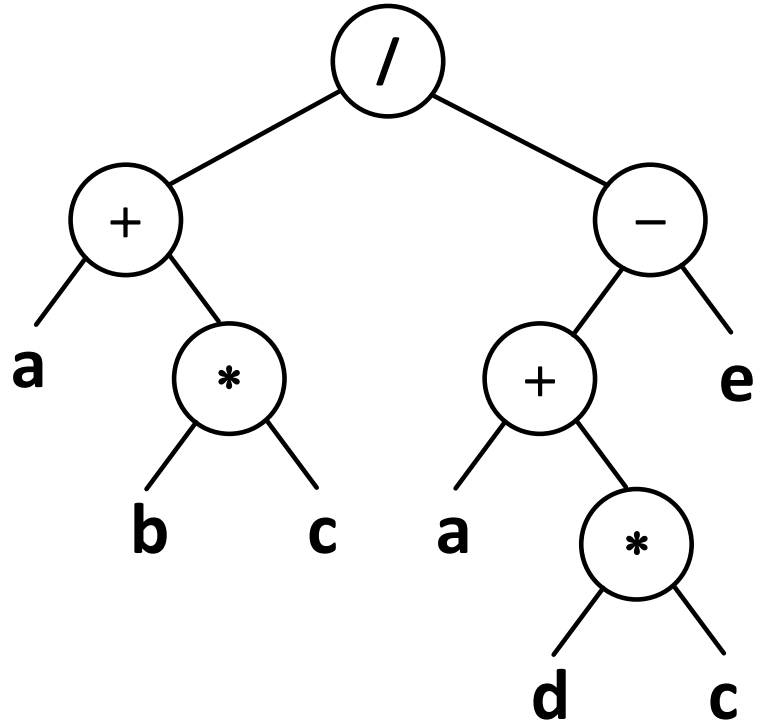
Machine Models

Architecture Type	Example
Stack	HP 3000, Intel X87 FPU
Load-store	ARM, MIPS, PowerPC, SPARC, RISC-V
Register-memory	IBM 360/370, Intel X86, Motorola 68000

Stack-Based Processor



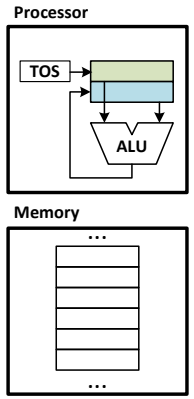

```
y = (a + b * c) / (a + d * c - e)
```



- Reverse Polish

```
a b c * + a d c * + e - /
```

Stack-Based Processor



a b c * + a d c * + e - /

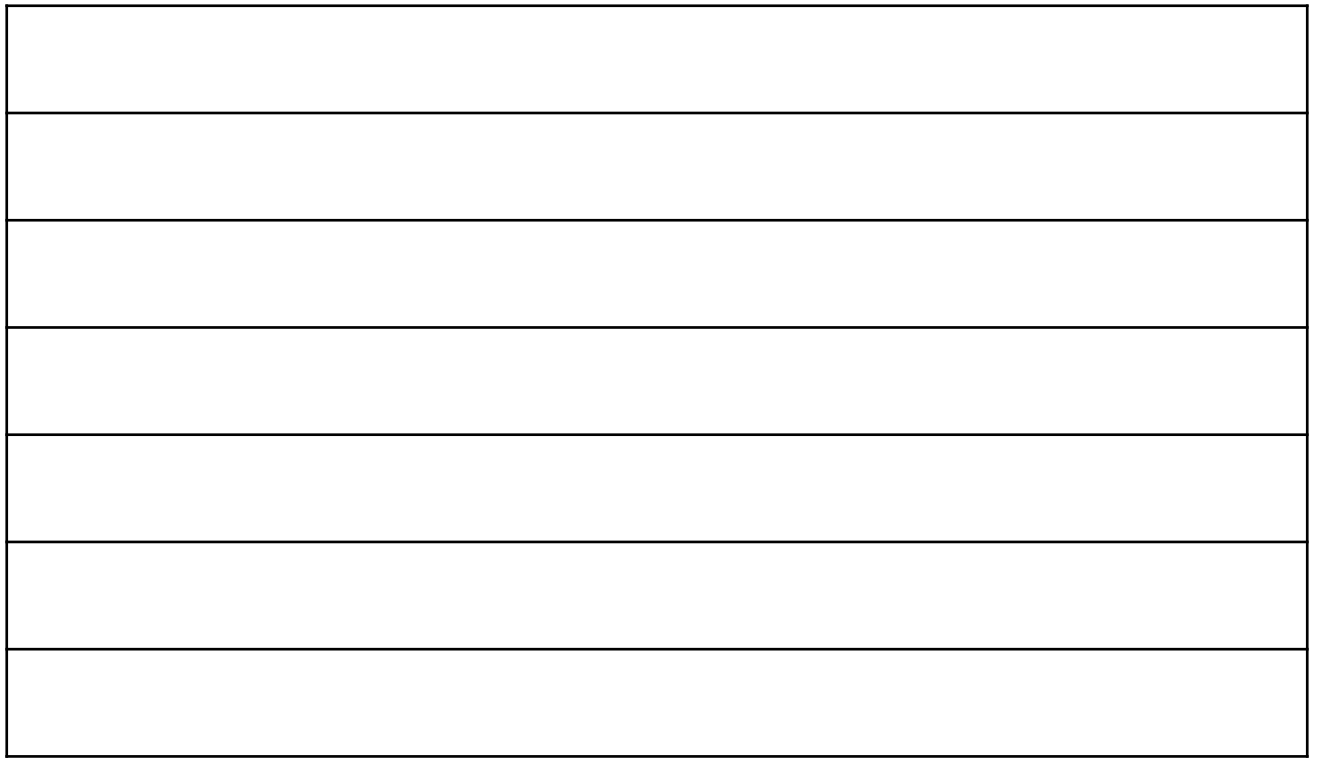
```
PUSH    a
PUSH    b
PUSH    c
MUL
ADD
PUSH    a
PUSH    d
PUSH    c
MUL
ADD
PUSH    e
SUB
DIV
POP     y
```

Stack

Stack-Based Processor

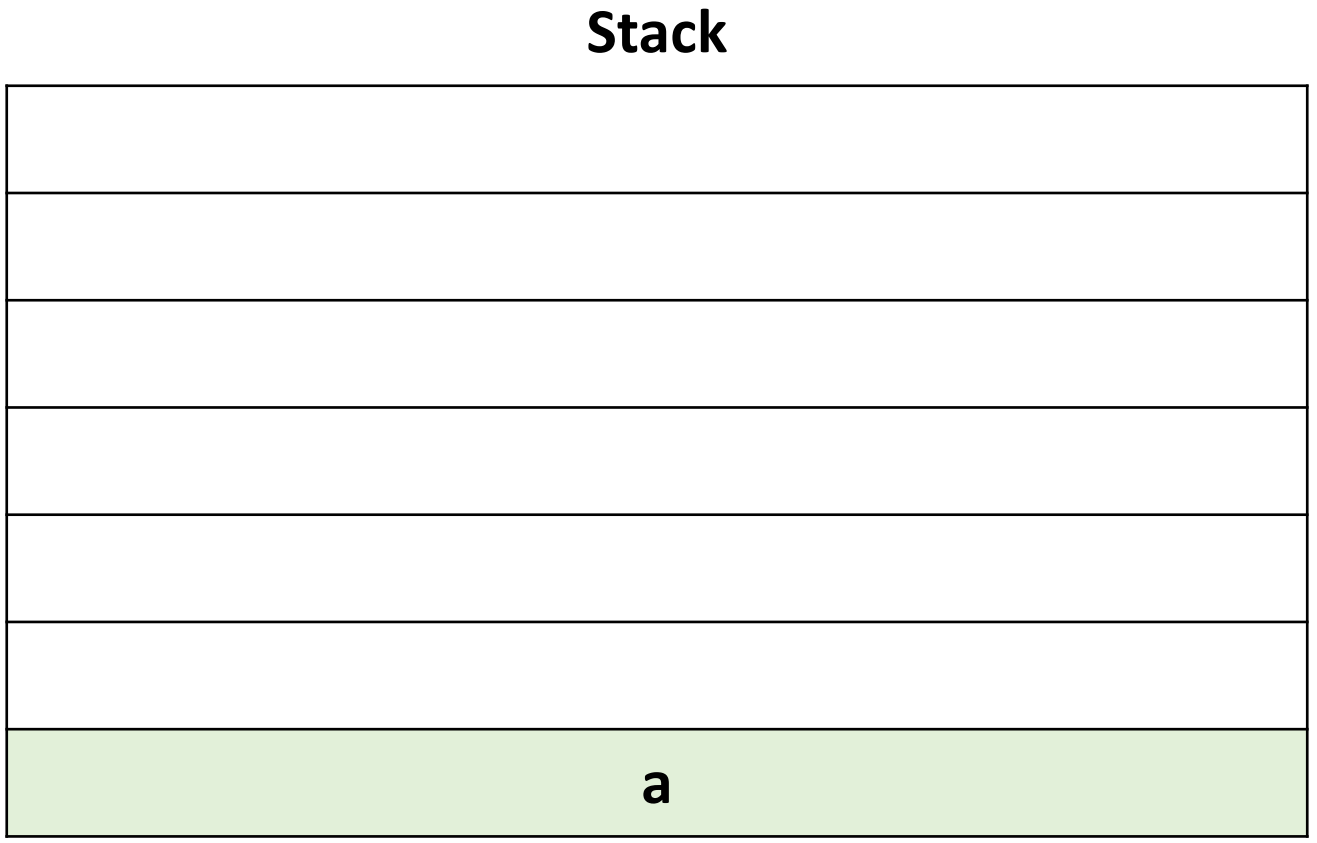
```
PUSH    a
PUSH    b
PUSH    c
MUL
ADD
PUSH    a
PUSH    d
PUSH    c
MUL
ADD
PUSH    e
SUB
DIV
POP     y
```

Stack



Stack-Based Processor

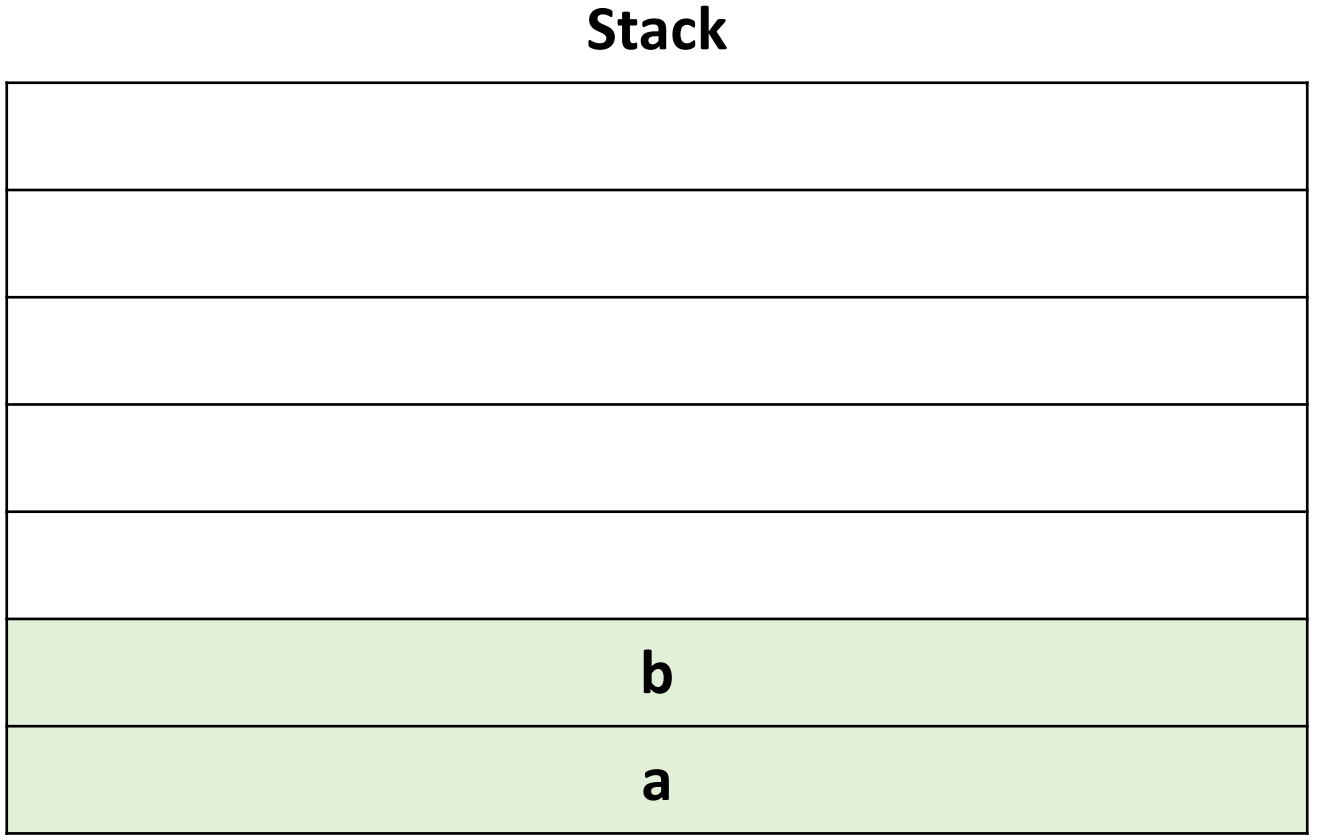
```
PUSH    a
PUSH    b
PUSH    c
MUL
ADD
PUSH    a
PUSH    d
PUSH    c
MUL
ADD
PUSH    e
SUB
DIV
POP     y
```



Saeed Safari,
ECE School, Faculty of Engineering,
University of Tehran

Stack-Based Processor

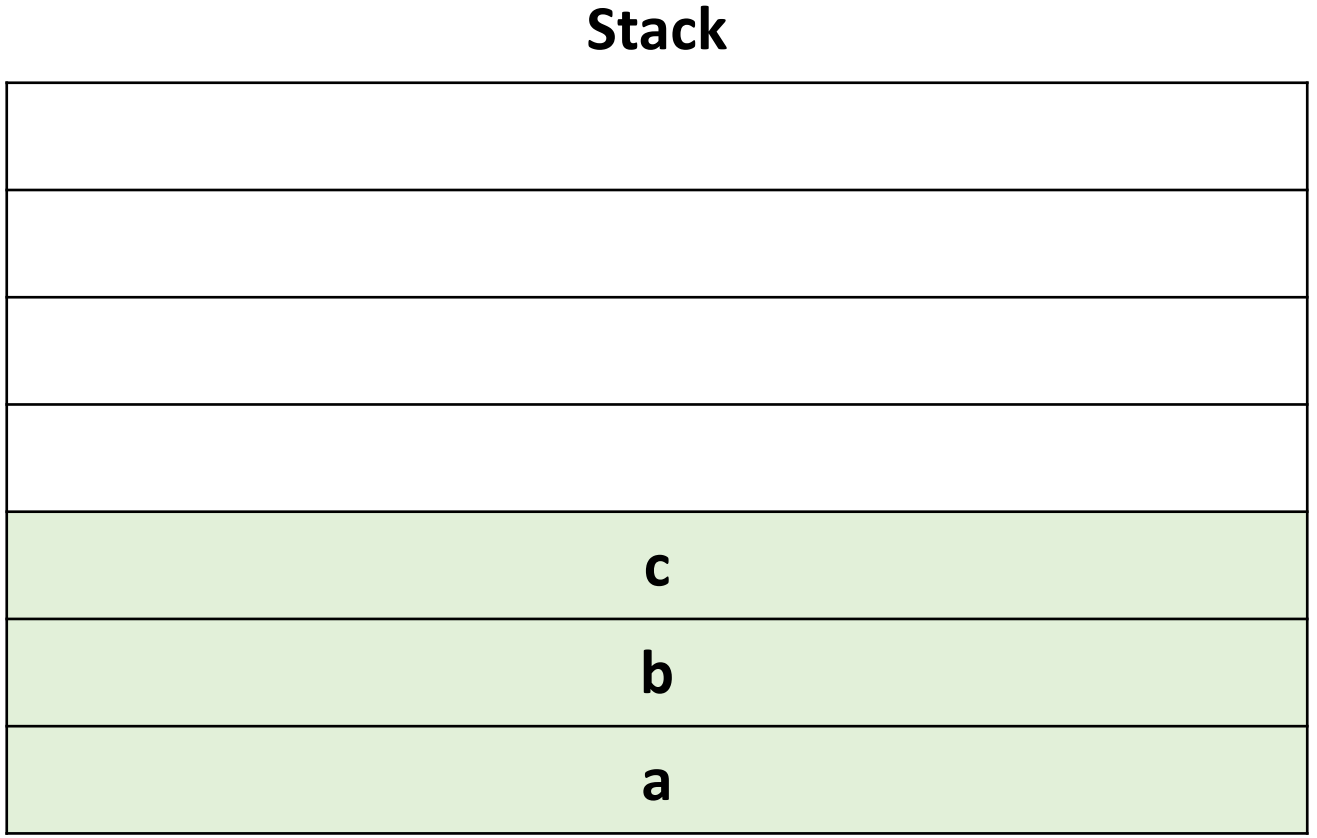
```
PUSH    a
PUSH    b
PUSH    c
MUL
ADD
PUSH    a
PUSH    d
PUSH    c
MUL
ADD
PUSH    e
SUB
DIV
POP     y
```



Saeed Safari,
ECE School, Faculty of Engineering,
University of Tehran

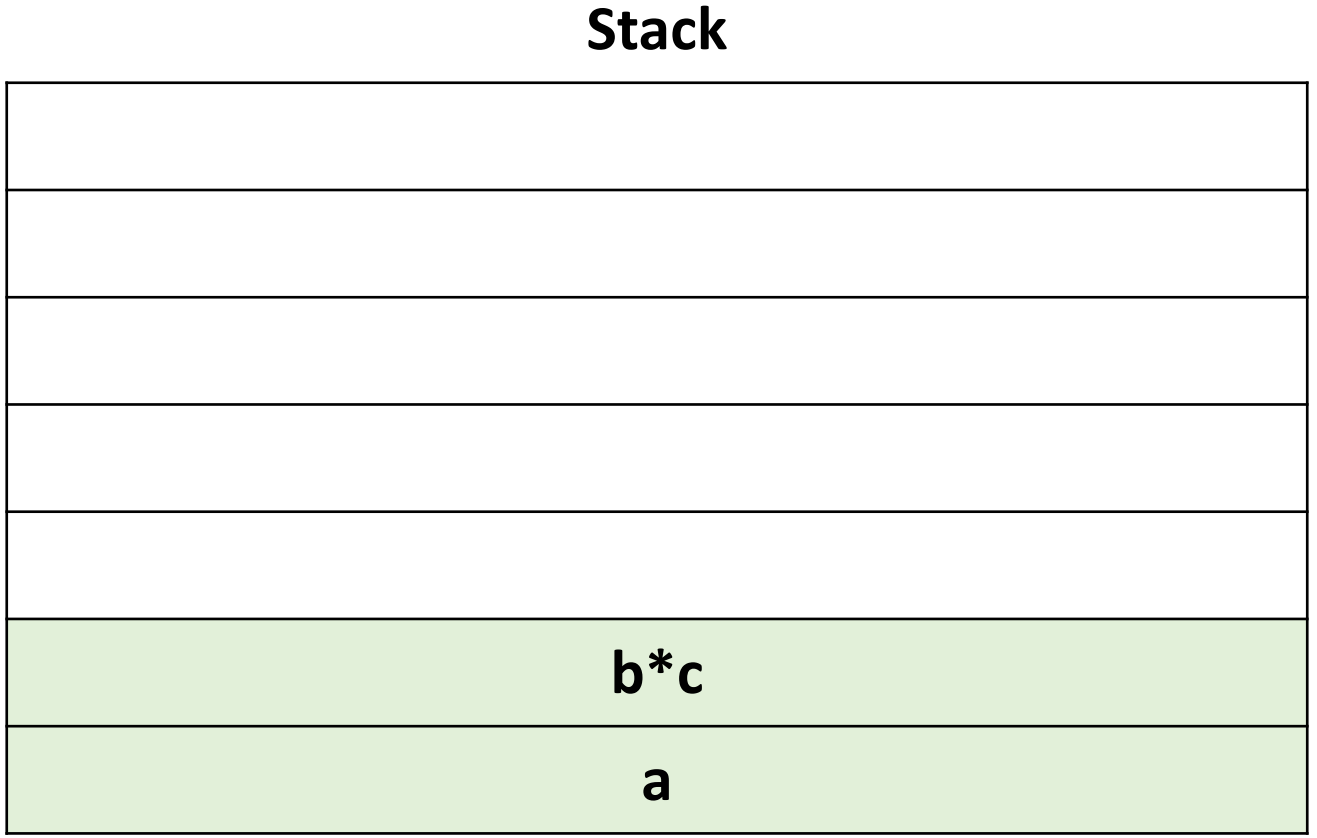
Stack-Based Processor

```
PUSH    a
PUSH    b
PUSH    c
MUL
ADD
PUSH    a
PUSH    d
PUSH    c
MUL
ADD
PUSH    e
SUB
DIV
POP     y
```



Stack-Based Processor

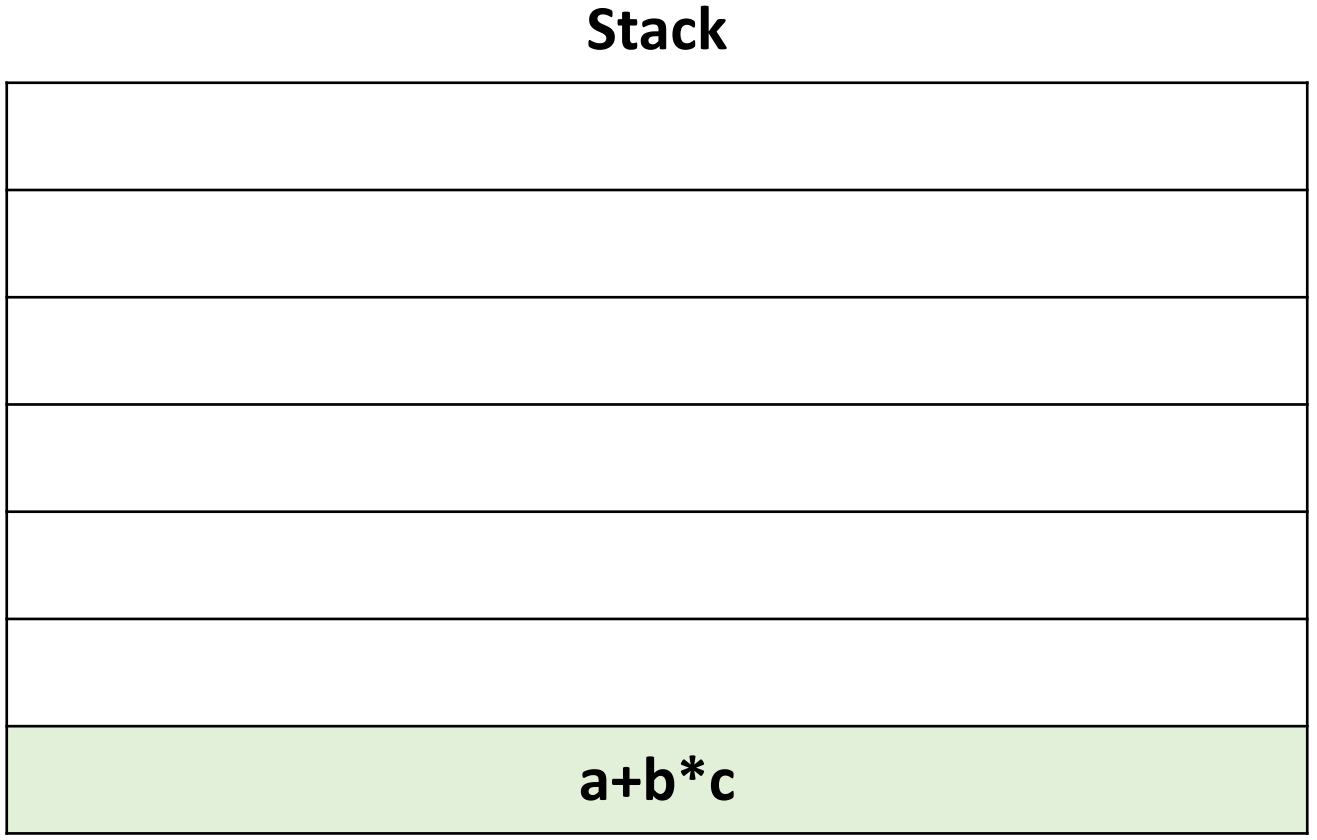
```
PUSH    a
PUSH    b
PUSH    c
MUL
ADD
PUSH    a
PUSH    d
PUSH    c
MUL
ADD
PUSH    e
SUB
DIV
POP     y
```



Saeed Safari,
ECE School, Faculty of Engineering,
University of Tehran

Stack-Based Processor

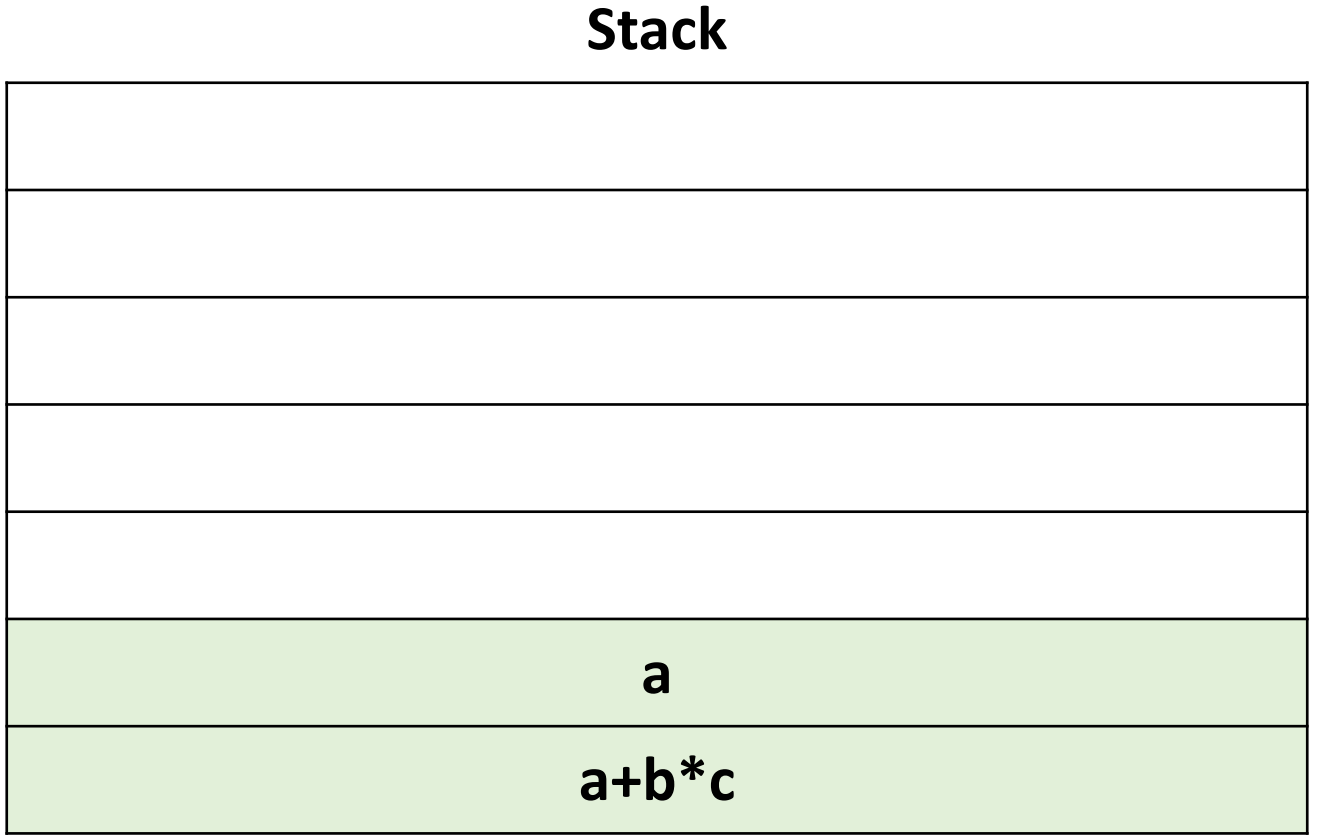
```
PUSH    a
PUSH    b
PUSH    c
MUL
ADD
PUSH    a
PUSH    d
PUSH    c
MUL
ADD
PUSH    e
SUB
DIV
POP     y
```



Saeed Safari,
ECE School, Faculty of Engineering,
University of Tehran

Stack-Based Processor

```
PUSH    a
PUSH    b
PUSH    c
MUL
ADD
PUSH    a
PUSH    d
PUSH    c
MUL
ADD
PUSH    e
SUB
DIV
POP     y
```



Saeed Safari,
ECE School, Faculty of Engineering,
University of Tehran

Stack-Based Processor

```
PUSH    a
PUSH    b
PUSH    c
MUL
ADD
PUSH    a
PUSH    d
PUSH    c
MUL
ADD
PUSH    e
SUB
DIV
POP     y
```

Stack

d
a
a+b*c

Saeed Safari,
ECE School, Faculty of Engineering,
University of Tehran

Stack-Based Processor

```
PUSH    a
PUSH    b
PUSH    c
MUL
ADD
PUSH    a
PUSH    d
PUSH    c
MUL
ADD
PUSH    e
SUB
DIV
POP     y
```

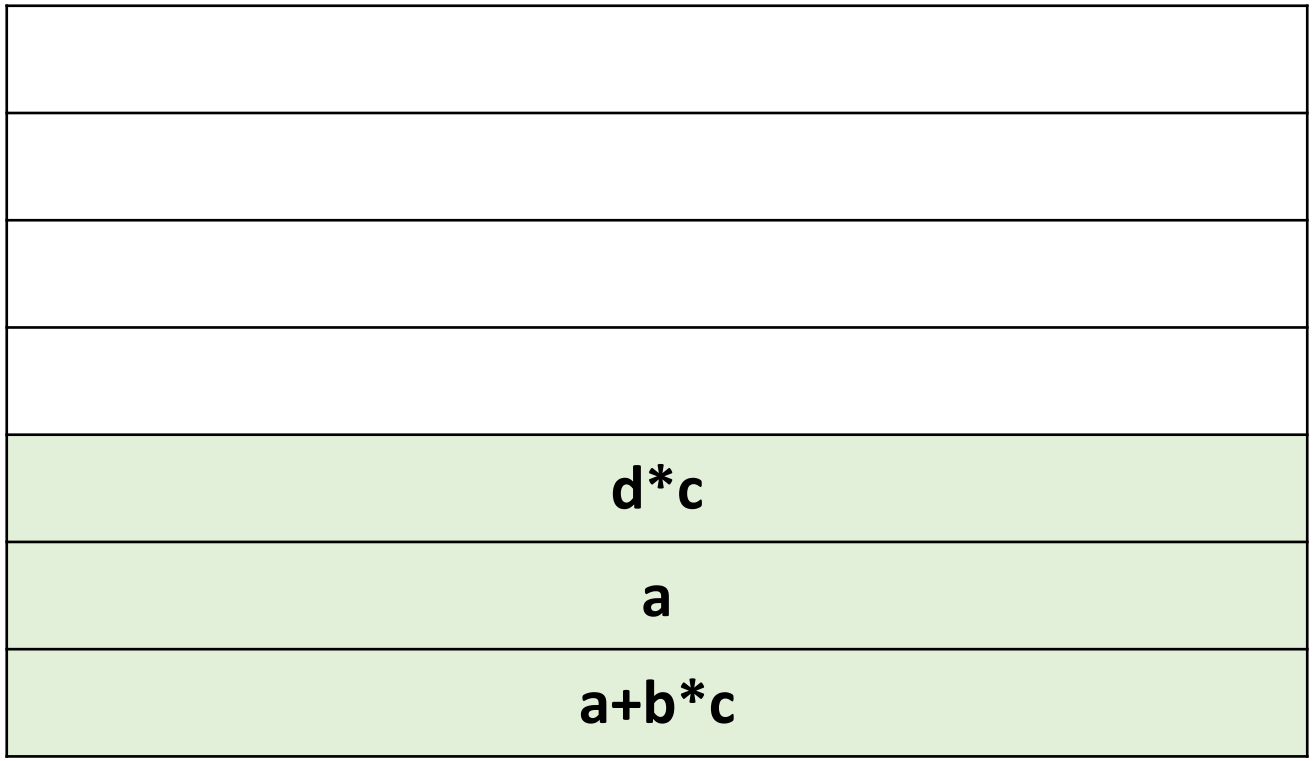
Stack

c
d
a
a+b*c

Stack-Based Processor

```
PUSH    a
PUSH    b
PUSH    c
MUL
ADD
PUSH    a
PUSH    d
PUSH    c
MUL
ADD
PUSH    e
SUB
DIV
POP     y
```

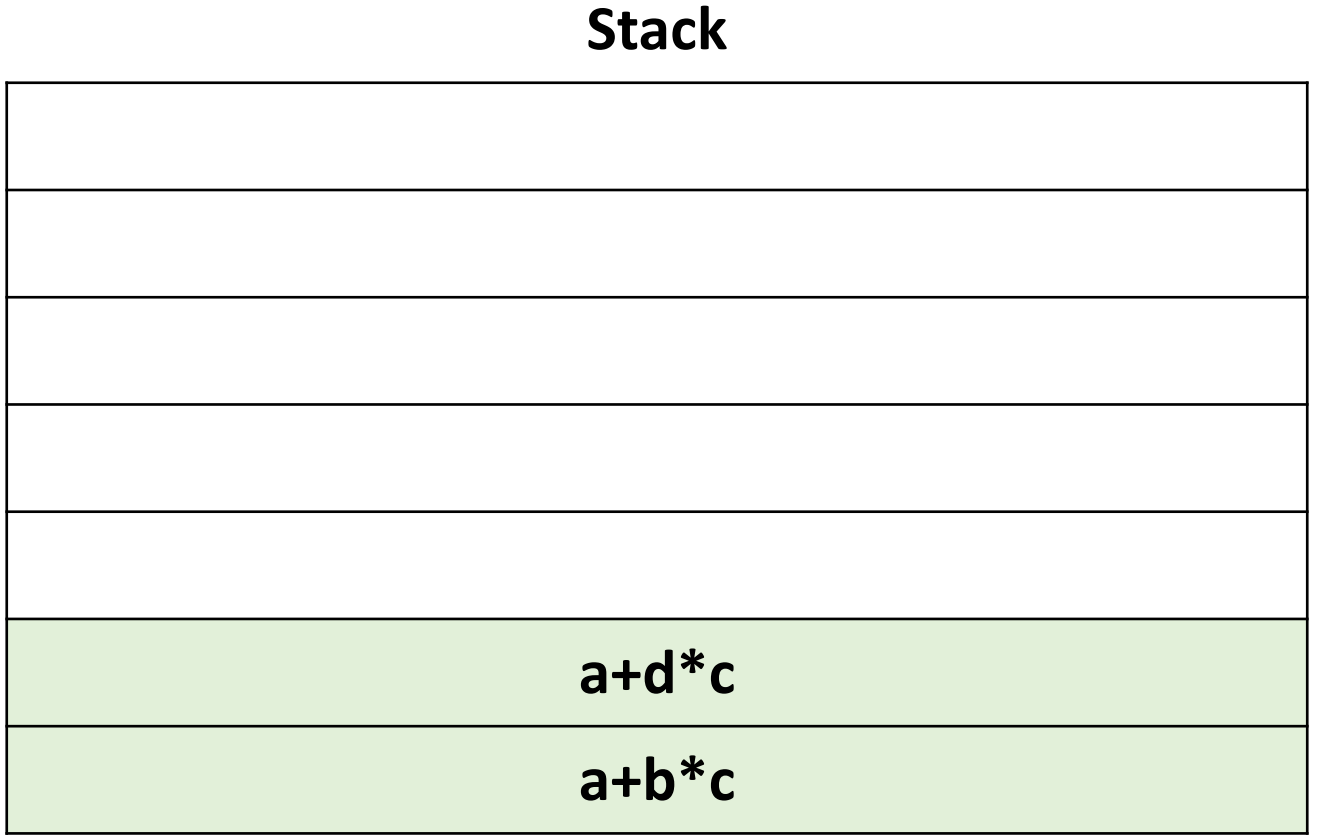
Stack



Saeed Safari,
ECE School, Faculty of Engineering,
University of Tehran

Stack-Based Processor

```
PUSH    a
PUSH    b
PUSH    c
MUL
ADD
PUSH    a
PUSH    d
PUSH    c
MUL
ADD
PUSH    e
SUB
DIV
POP     y
```



Stack-Based Processor

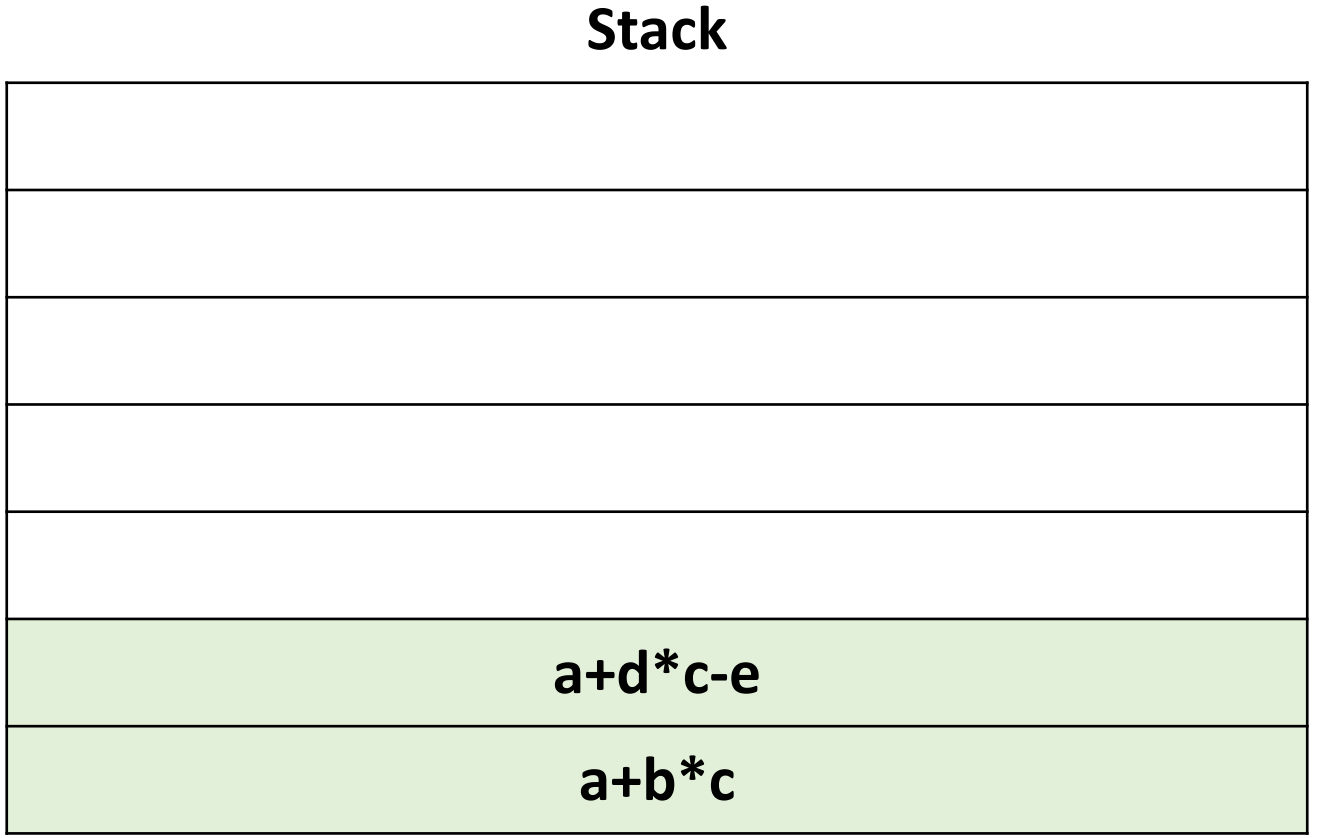
```
PUSH    a
PUSH    b
PUSH    c
MUL
ADD
PUSH    a
PUSH    d
PUSH    c
MUL
ADD
PUSH    e
SUB
DIV
POP     y
```

Stack

e
a+d*c
a+b*c

Stack-Based Processor

```
PUSH    a
PUSH    b
PUSH    c
MUL
ADD
PUSH    a
PUSH    d
PUSH    c
MUL
ADD
PUSH    e
SUB
DIV
POP     y
```



Stack-Based Processor

```
PUSH    a
PUSH    b
PUSH    c
MUL
ADD
PUSH    a
PUSH    d
PUSH    c
MUL
ADD
PUSH    e
SUB
DIV
POP     y
```

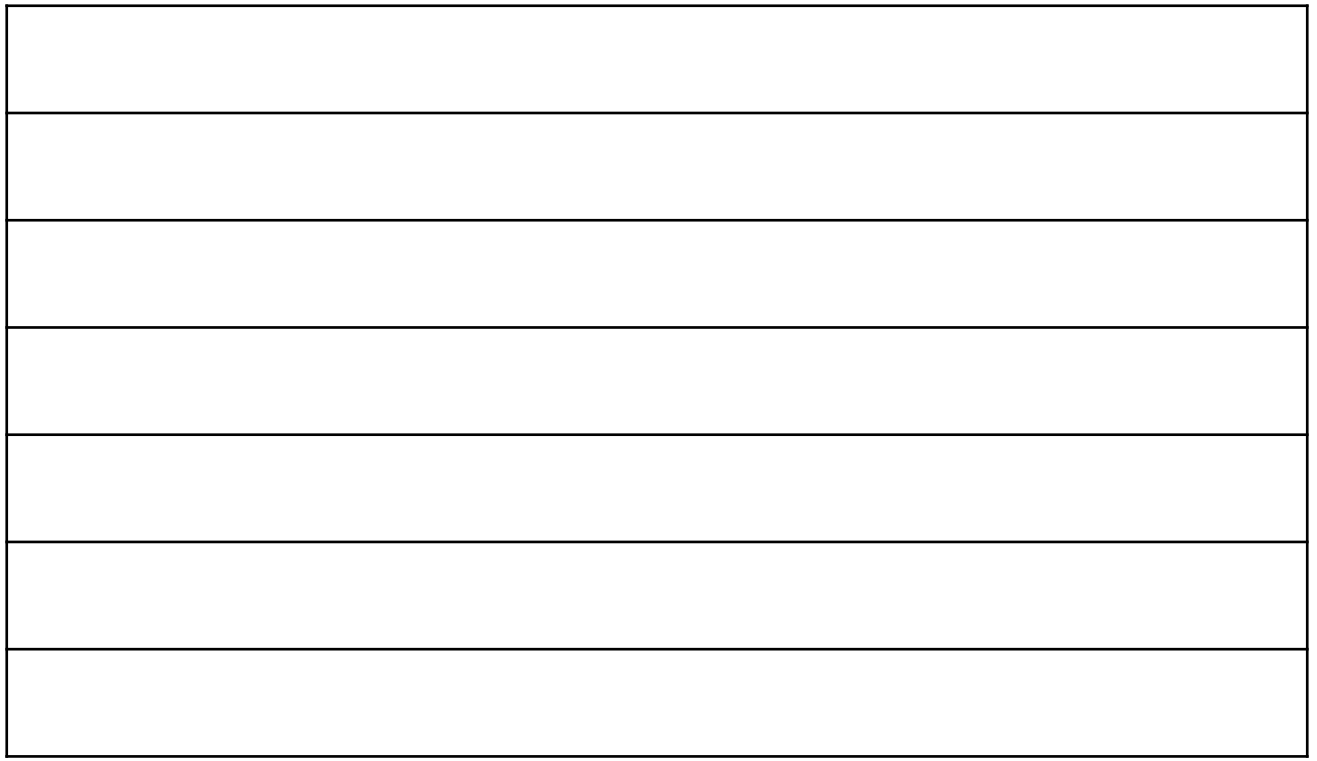
Stack

(a+b*c)/(a+d*c-e)

Stack-Based Processor

```
PUSH    a
PUSH    b
PUSH    c
MUL
ADD
PUSH    a
PUSH    d
PUSH    c
MUL
ADD
PUSH    e
SUB
DIV
POP     y
```

Stack



Stack Hardware Organization

- Stack is part of the processor state
  - Stack must be bounded and small
  - Number of registers
  - Smaller than the main memory
- Conceptually stack is unbounded
  - A part is in the processor state
  - The rest is in the main memory

Stack Hardware Organization

- Each PUSH (or POP) operation needs 1 memory reference
- Assumption
  - Stack size = 2 (4)
  - The 2 (4) top elements stored in registers
  - The rest is in main memory

Stack Hardware Organization

Program	Stack (Size = 2)	Memory References
PUSH a	R0	a
PUSH b	R0 R1	b, a
PUSH c	R0 R1 R2	c, b, ss(a)
MUL	R0 R1	b*c, sf(a)
ADD	R0	a+b*c
PUSH a	R0 R1	a, a+b*c
PUSH d	R0 R1 R2	d, a, ss(a+b*c)
PUSH c	R0 R1 R2 R3	c, d, ss(a)
MUL	R0 R1 R2	d*c, sf(a)
ADD	R0 R1	a+d*c, sf(a+b*c)
PUSH e	R0 R1 R2	e, a+d*c, ss(a+b*c)
SUB	R0 R1	a+d*c-e, sf(a+b*c)
DIV	R0	(a+b*c)/(a+d*c-e)

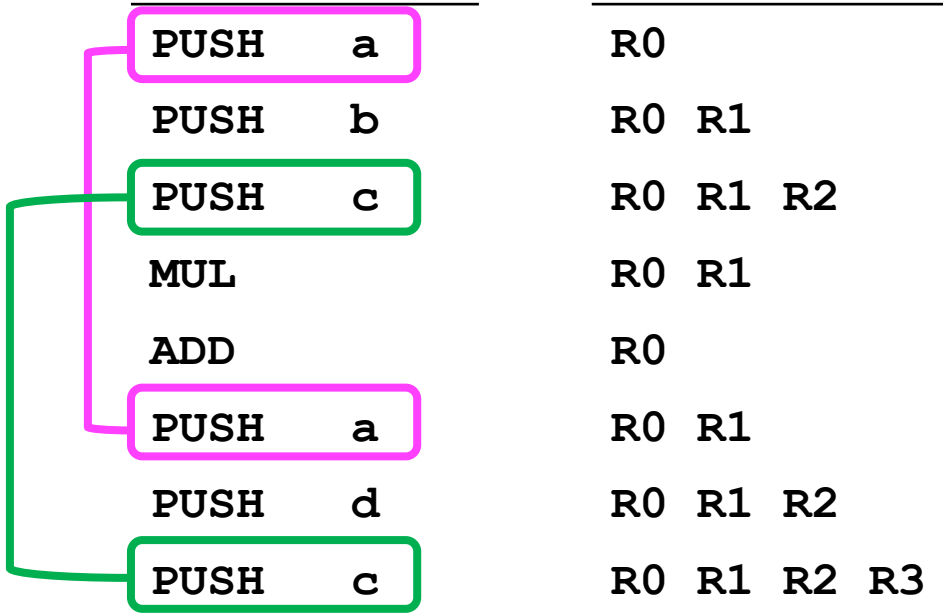
4 implicit stores

4 implicit fetches

Saeed Safari,
ECE School, Faculty of Engineering,
University of Tehran

Stack Hardware Organization

a and **c** are loaded twice



Program		Stack (Size = 4)
PUSH	a	R0
PUSH	b	R0 R1
PUSH	c	R0 R1 R2
MUL		R0 R1
ADD		R0
PUSH	a	R0 R1
PUSH	d	R0 R1 R2
PUSH	c	R0 R1 R2 R3
MUL		R0 R1 R2
ADD		R0 R1
PUSH	e	R0 R1 R2
SUB		R0 R1
DIV		R0

Memory Addressing

- How many address lines?
  - 32-bit
  - 64-bit
- Byte/Word addressable?

Memory Addressing

- Little/Big Endian?
  - Little endian

  - Big endian

Memory Addressing

- Aligned / Unaligned?

$$Adr \bmod size = \begin{cases} 0 & aligned \\ 1 & unaligned \end{cases}$$

Width of objects	0	1	2	3	4	5	6	7
1 byte (byte)	Aligned	Aligned	Aligned	Aligned	Aligned	Aligned	Aligned	Aligned
2 bytes (half word)	Aligned		Aligned		Aligned		Aligned	
2 bytes (half word)		Unaligned		Unaligned		Unaligned		Unaligned
4 bytes (word)	Aligned				v			
4 bytes (word)		Unaligned				Unaligned		
4 bytes (word)			Unaligned				Unaligned	
4 bytes (word)				Unaligned				Unaligned

Memory Addressing

Addressing mode	Example	Meaning
Register	Add R1, R2	$Regs[R1] \leftarrow Regs[R1] + Regs[R2]$
Immediate	Add R1, 3	$Regs[R1] \leftarrow Regs[R1] + 3$
Displacement	Add R1, 100(R2)	$Regs[R1] \leftarrow Regs[R1] + Mem[100 + Regs[R2]]$
Register Indirect	Add R1, (R2)	$Regs[R1] \leftarrow Regs[R1] + Mem[Regs[R2]]$
Indexed	Add R1, (R2+R3)	$Regs[R1] \leftarrow Regs[R1] + Mem[Regs[R2] + Regs[R3]]$
Direct (Absolute)	Add R1, (100)	$Regs[R1] \leftarrow Regs[R1] + Mem[100]$
Memory Indirect	Add R1, @(R2)	$Regs[R1] \leftarrow Regs[R1] + Mem[Mem[Regs[R2]]]$

Memory Addressing

Addressing mode	Example	Meaning
Autoincrement	Add R1, (R2)+	$Regs[R1] \leftarrow Regs[R1] + Mem[Regs[R2]]$ $Regs[R2] \leftarrow Regs[R2] + \mathrm{d}$
Autodecrement	Add R1, –(R2)	$Regs[R2] \leftarrow Regs[R2] - \mathrm{d}$ $Regs[R1] \leftarrow Regs[R1] + Mem[Regs[R2]]$
Scaled	Add R1, 100(R2)[R3]	$Regs[R1] \leftarrow Regs[R1] +$ $Mem[100 + Regs[R2] + Regs[R3] * d]$

Type and Size of Operands

- Types:
  - Integer (Signed/Unsigned)
  - Binary Coded Decimal (BCD)
  - Floating Point
    - IEEE-754
    - Intel Extended Precision (80-bit)
  - Packed Vector Data

Type and Size of Operands

- Sizes:
  - Integer (8-bit, 16-bit, 32-bit, 64-bit)
  - Floating Point
    - Single Precision (SP: 32-bit)
    - Double Precision (DP: 64-bit)
    - Intel Extended Precision (80-bit)

Operations

- Data transfer:
  - LD, ST, MFC1, MTC1
- ALU operation:
  - ADD, SUB, AND, OR, XOR, MUL, DIV, SLT, LUI
- Floating Point:
  - ADD.S, SUB.D, MUL.D, CVT.S.W, CVT.W.S
- Control Flow:
  - J, JR, BEQ, JAL

Operations

- Multimedia (SIMD):
  - ADD.PS, SUB.PD, MUL.PS
- String
  - REP MOVSB (x86)

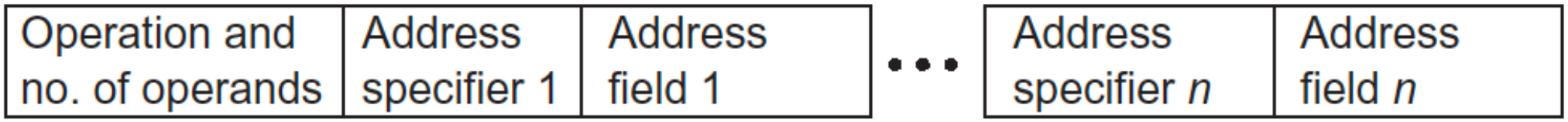
ISA Encoding

- Fixed Width:
  - Every instruction has the same width
  - Easy to decode
  - RISC architectures :
    - MIPS, PowerPC, SPARC, ARM, RISC-V, …
  - Example: MIPS, every instruction has 4-bytes

Operation	Address field 1	Address field 2	Address field 3

ISA Encoding

- Variable Length:
  - Instruction can have different width
  - Less space in cache/memory
  - CISC architectures:
    - IBM 360, Intel x86, Motorola 68K, VAX, …
  - Example: X86, instructions are 1-byte up to 17-bytes



ISA Encoding

- Mostly Fixed:
  - Instruction has two different widths
  - Example:
    - MIPS16: two types of instructions: 2-bytes and 4-bytes
    - THUMBS (ARM16): two types of instructions: 2-bytes and 4-bytes

ISA Encoding

- Very Long Instruction Word (VLIW):
  - Multiple instructions in a fixed length bundle
  - Example:
    - Multiflow
    - HP/ST* Lx
    - TI C6000

* Semiconductor Technology

Real World Instruction Sets

Arch	Type	# Opnd	# Mem	Data Size	# Regs	Adr Size	Use
Alpha	Reg-Reg	3	0	64-bit	32	64-bit	Workstation
ARM	Reg-Reg	3	0	32/64-bit	16	32/64-bit	Cell Phones, Embedded
MIPS	Reg-Reg	3	0	32/64-bit	32	32/64-bit	Workstation, Embedded
SPARC	Reg-Reg	3	0	32/64-bit	24-32	32/64-bit	Workstation
TI C6000	Reg-Reg	3	0	32-bit	32	32-bit	DSP
IBM 360	Reg-Mem	2	1	32-bit	16	24/31/64	Mainframes
x86	Reg-Mem	2	1	8/16/32/64-bit	4/8/24	16/32/64	Personal Computers
VAX	Mem-Mem	3	3	32-bit	16	32-bit	Minicomputer
Mot. 6800	Accumulator	1	1/2	8-bit	0	16-bit	Microcontroller

Summary

- Topics covered
  - Architecture vs Micro-architecture
  - Architecture
    - Machine model
    - Memory addressing
    - Data types
    - Instructions
    - Encoding
  - Micro-architecture