Computer Architecture

Instruction Set Architecture

Virendra Singh
Professor
Computer Architecture and Dependable Systems Lab
Department of Electrical Engineering
Indian Institute of Technology Bombay
http://www.ee.iitb.ac.in/~viren/
E-mail: viren@ee.iitb.ac.in

CS-683: Advanced Computer Architecture

Lecture 3 (03 August 2021)

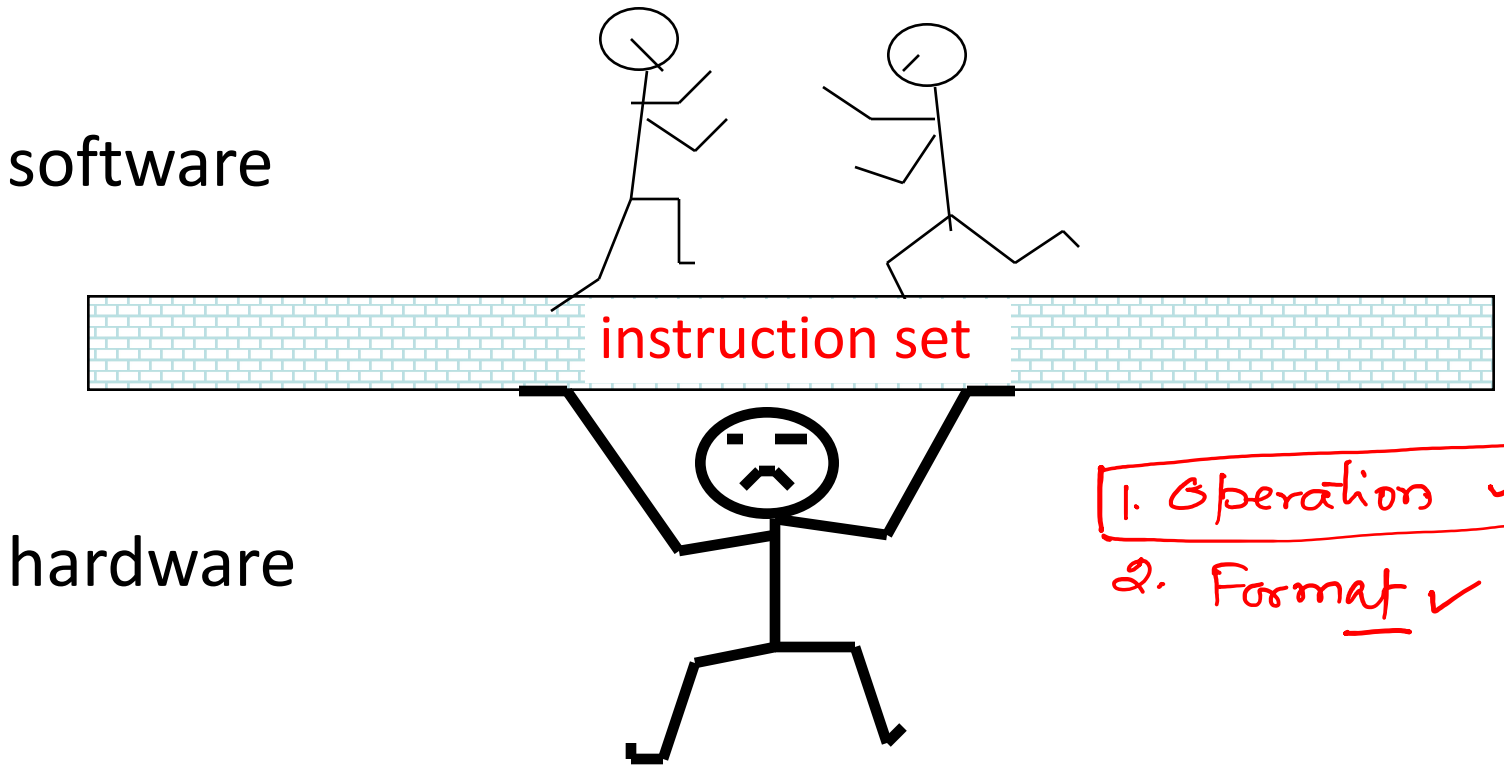
CADSL

Instruction Set Architecture

- Instruction set architecture is the structure of a computer that a machine language programmer must understand to write a correct (timing independent) program for that machine.

- The instruction set architecture is also the machine description that a hardware designer must understand to design a correct implementation of the computer.

Instruction Set Architecture (ISA)



1. Operation ✓
2. Format ✓

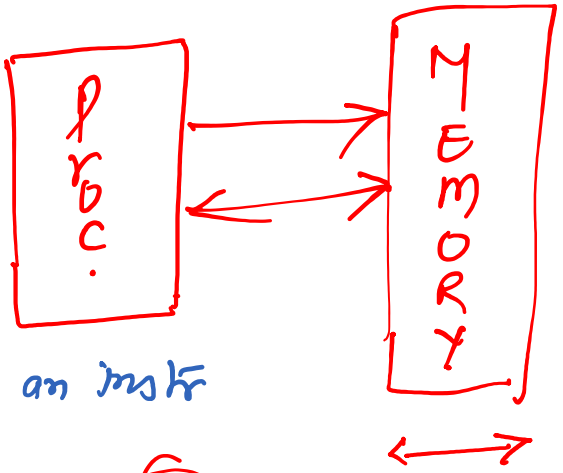
Why not Bigger Instructions?

- Why not "f = (g+h) – (i+j)" as one instruction?
- Church's thesis: A very primitive computer can compute anything that a fancy computer can compute – you need only logical functions, read and write to memory, and data dependent decisions
- Therefore, ISA selection is for practical reasons
  - Performance and cost not computability
- Regularity tends to improve both
  - E.g, H/W to handle arbitrary number of operands is complex and slow, and UNNECESSARY

Instructions Can Be Divided into 3 Classes (I)

- Data movement instructions
  - Move data from a memory location or register to another memory location or register without changing its form
  - *Load*—source is memory and destination is register
  - *Store*—source is register and destination is memory
- Arithmetic and logic (ALU) instructions
  - Change the form of one or more operands to produce a result stored in another location
  - *Add*, *Sub*, *Shift*, etc.
- Branch instructions (control flow instructions)
  - Alter the normal flow of control from executing the next instruction in sequence
  - *Br Loc*, *Brz Loc2*,—unconditional or conditional branches

Instruction Set Architecture



1. memory size
   (static)
   (entire instruction should fit into memory)
   Minimize no. of bits in an instr

2. Slow memory
   * (lot of work packed in an instruction)

3. Programming
   - Complex data structure
     Multiple addressing modes

f: a+b*c
MAC

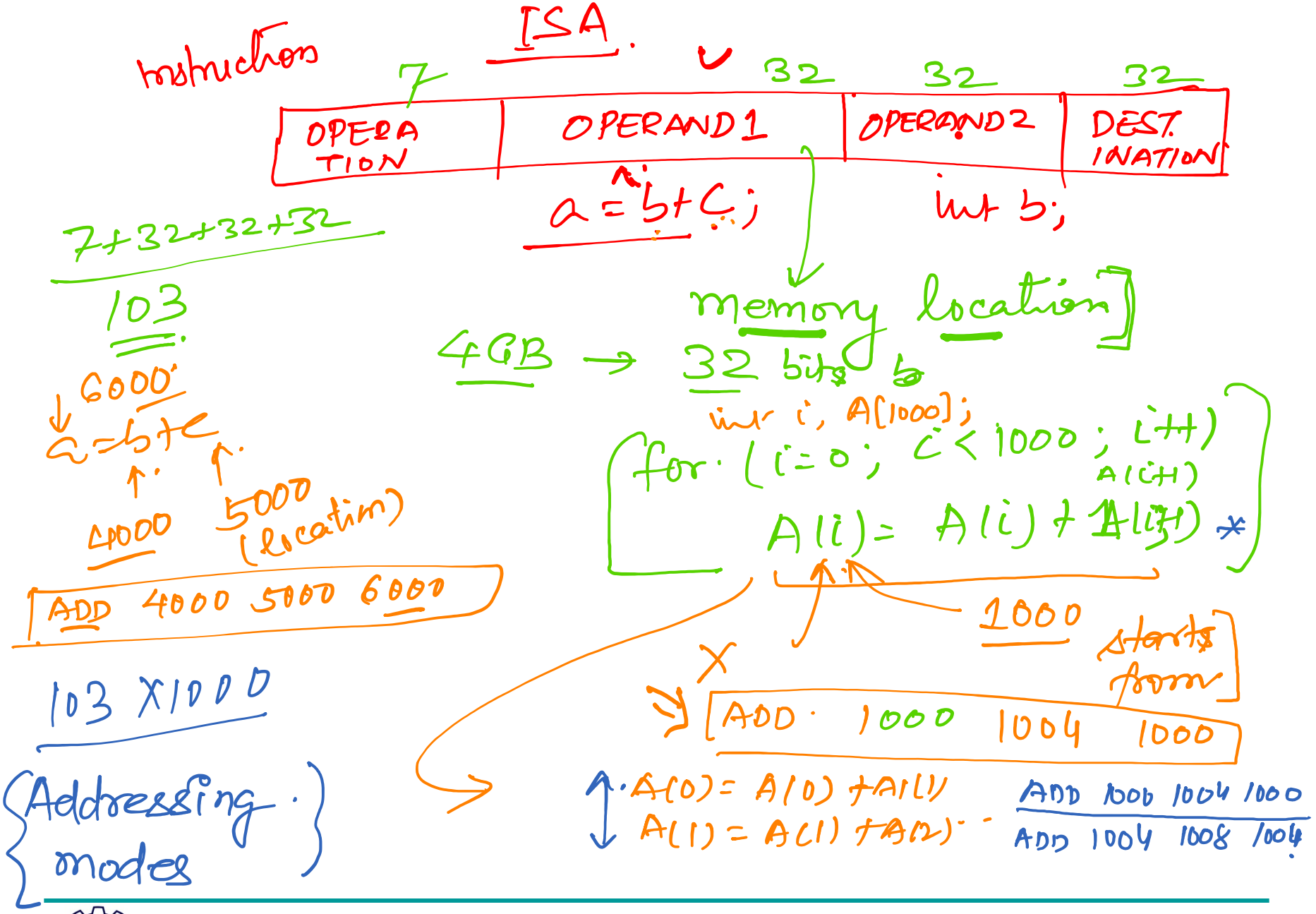
I1 — I10

1000+10+1000+10+
- - - - + 1000+10

I1 I2 I10

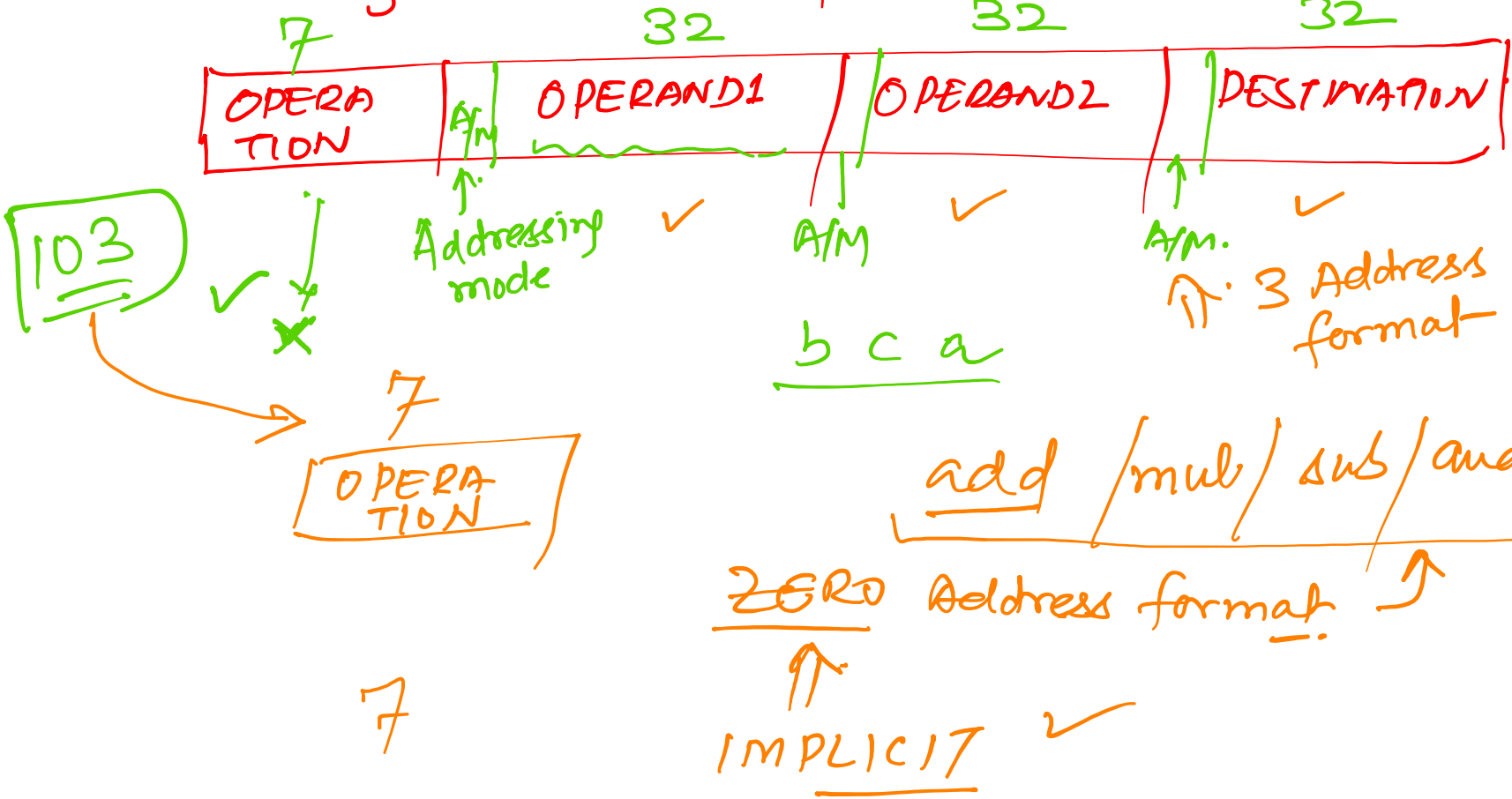
= 1010 x 10 = 10100

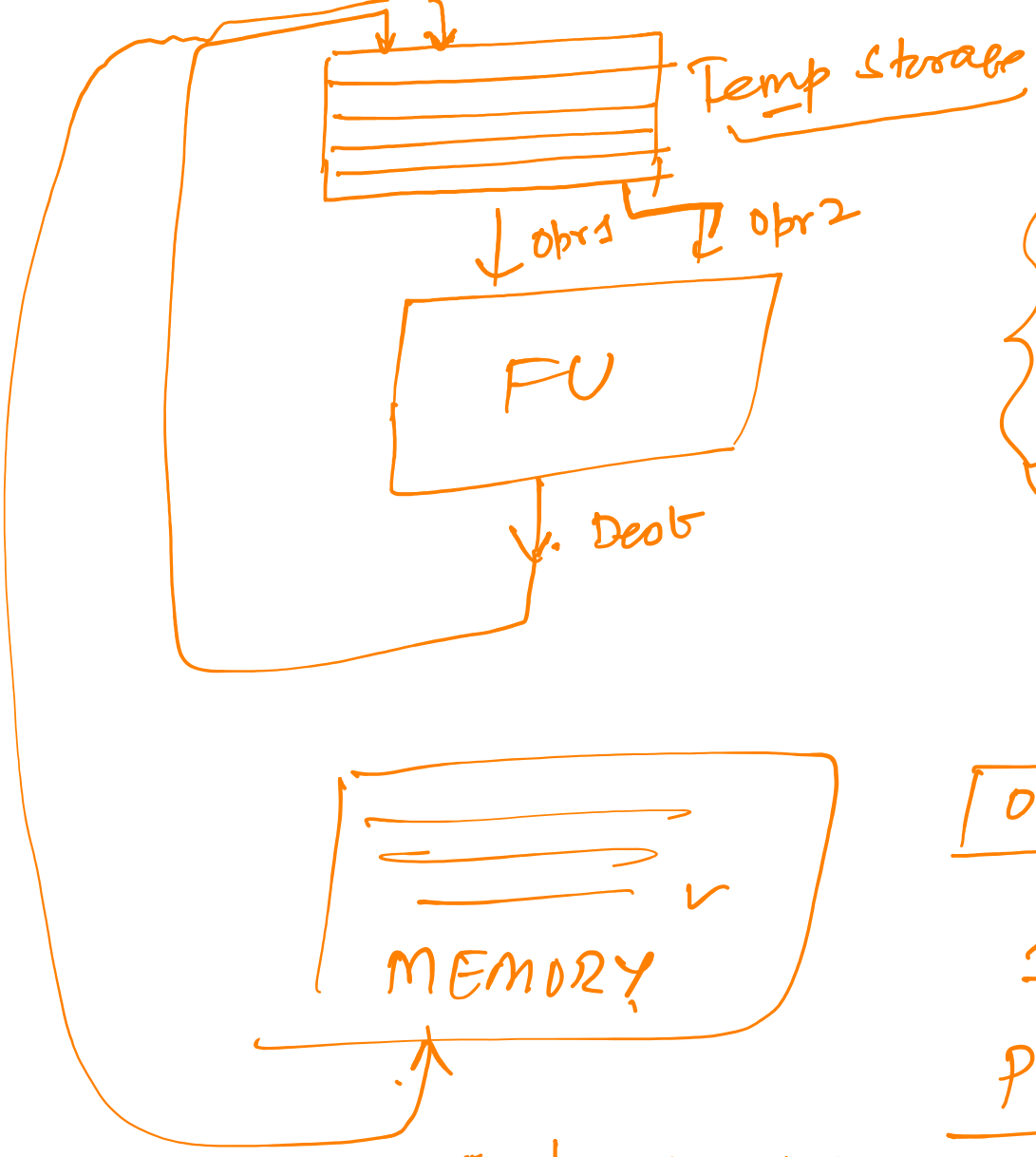
= 1001 x 10 = 10010


Instruction
ISA
7
32
32
32
OPERATION
OPERAND1
OPERAND2
DESTINATION
a = b + c;
int b;
7+32+32+32
103
memory location
4GB → 32 bits
int i, A[1000];
for (i=0; i<1000; i++)
A(i) = A(i) + A(i+1) *
A(i+1)
1000
starts from
ADD 1000 1004 1000
X
103 X 1000
ADD 4000 5000 6000
6000
a = b + c
4000
5000 (location)
Addressing modes
A(0) = A(0) + A(1)
A(1) = A(1) + A(2)
ADD 1000 1004 1000
ADD 1004 1008 1004

ISA

Minimize no. of bits per instruction





take operand from top of Temp storage & write result on top

STACK LIFO

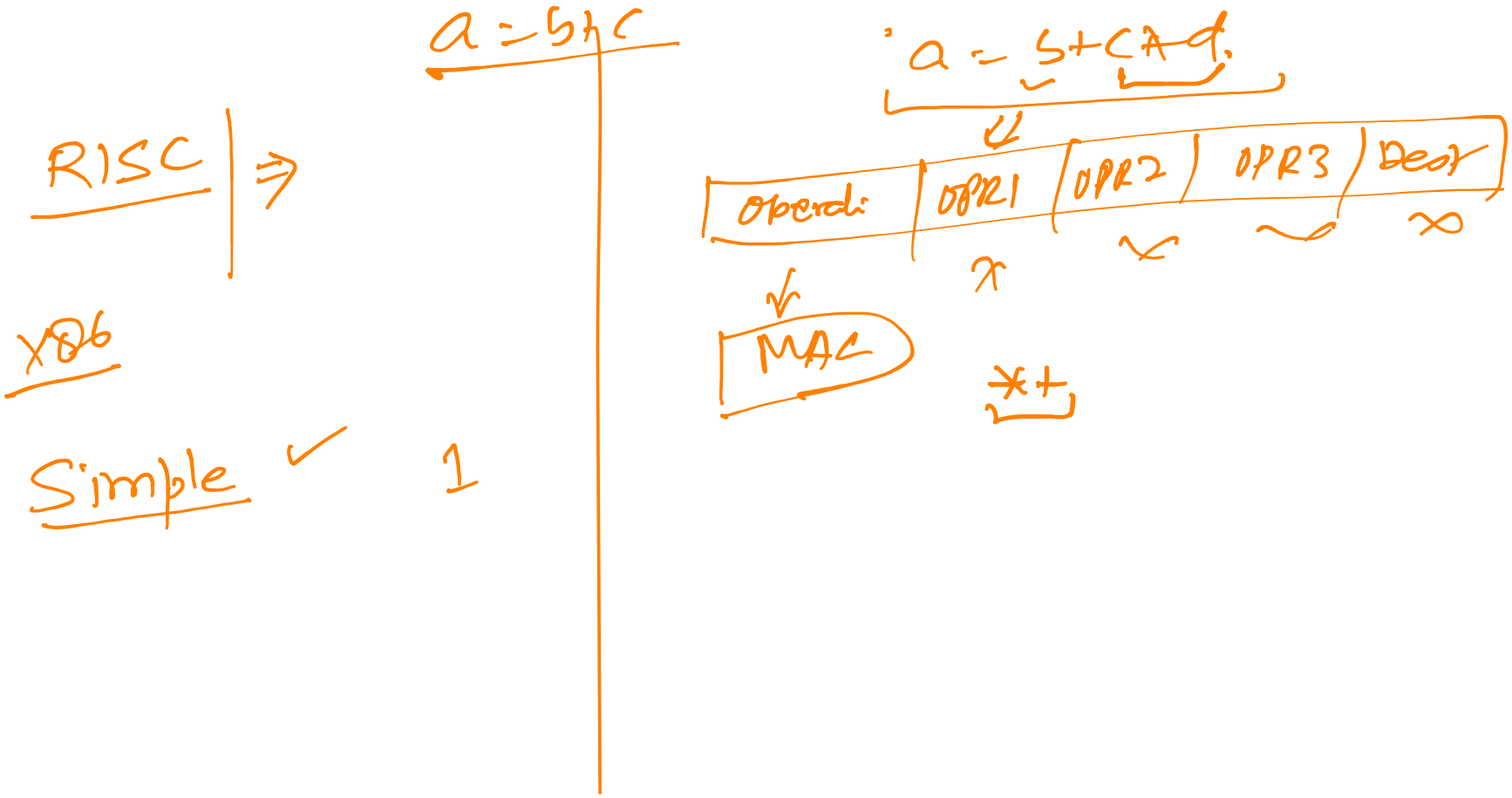
OPERATION	OPERAND
7	32

= 39

1 Address

PUSH / POP

a = b + c | b c + v


a = b * c
RISC ⇒
x86
Simple ✓
1
a = b + c + d
Operation
OPR1
OPR2
OPR3
Dest
x
∞
MAC
* +

Stack Architectures

- Instruction set:

  add, sub, mult, div, . . .

  push A, pop A

- Example: A*B - (A+C*B)

  push A

  push B

  mul

  push A

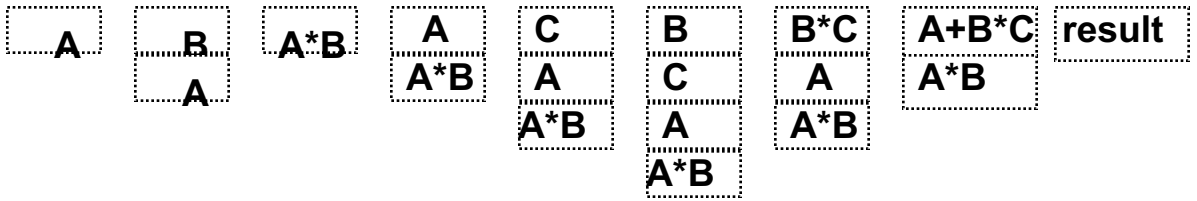
  push C

  push B

  mul

  add

  sub



Stacks: Pros and Cons

- Pros
  - Good code density (implicit operand addressing→ top of stack)
  - Low hardware requirements
  - Easy to write a simpler compiler for stack architectures
- Cons
  - Stack becomes the bottleneck
  - Little ability for parallelism or pipelining
  - Data is not always at the top of stack when need, so additional instructions like TOP and SWAP are needed
  - Difficult to write an optimizing compiler for stack architectures

Thank You