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

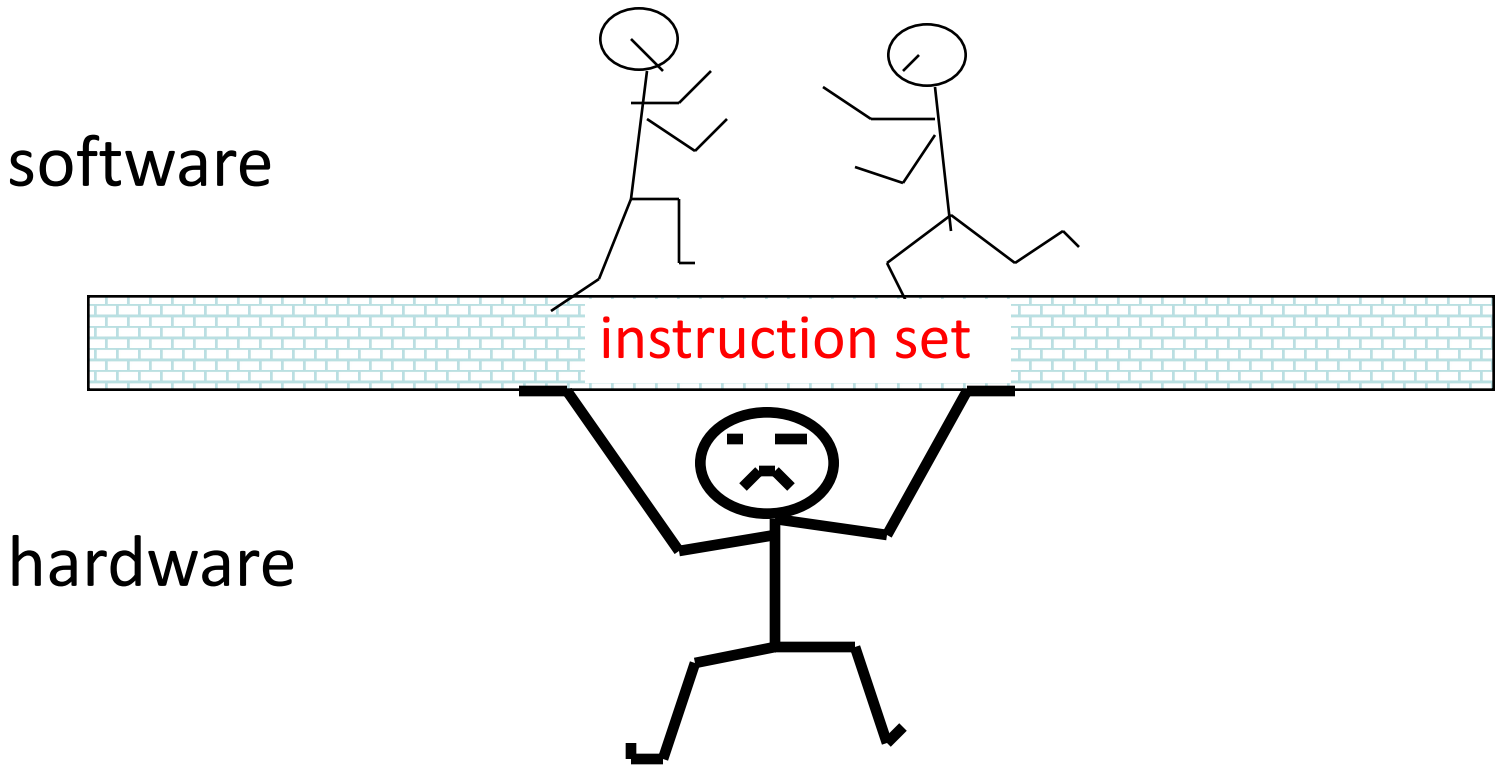
Lecture 4 (05 August 2021) CADSL

Instruction Set Architecture

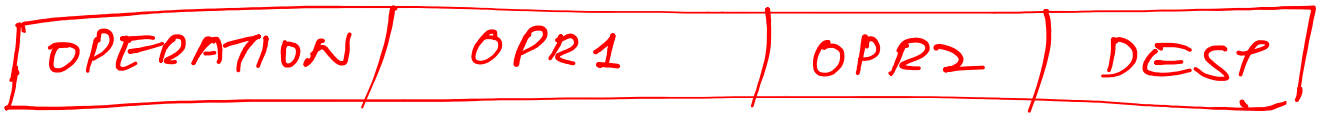
- Instruction set architecture is the structure of a computer that a machine language programmer must understand to write a correct (timing independent) program for that machine.

- 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)



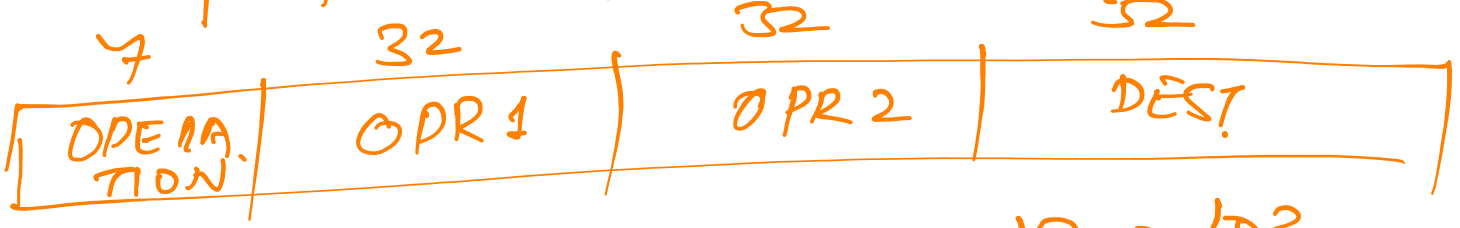
ISA



1. Scarcity of Memory → reduce no. of bits per instruction
2. Slow memory → pack lot of work in a single instruction (insertion in linked list, FFT.)
3. Ease of Programming → multiple addressing modes.

a := std.
a := std.
a := std.

Scarcity of memory



7 + 32 + 32 + 32 = 103.

4 GB → 32 bit

3 Address format

0- Address format ← Implicit reference to operand

ADD / MUL / SUB / AND

1- Address inst.

PUSH a | POP b

LIFO

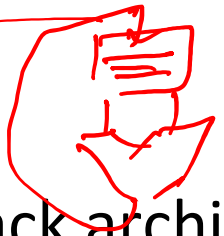
STACK-BASED MACHINES

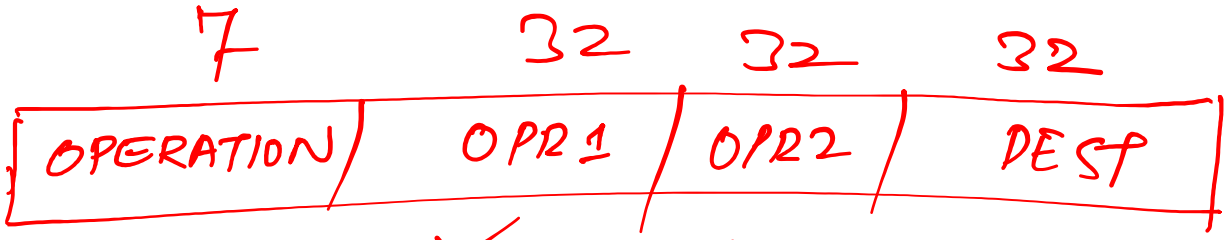
ISA Classification

- Type of internal storage in a processor is the most basic differentiator
- Stack Architecture
- Accumulator Architecture
- General Purpose Register Architecture

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





ONE-Address format

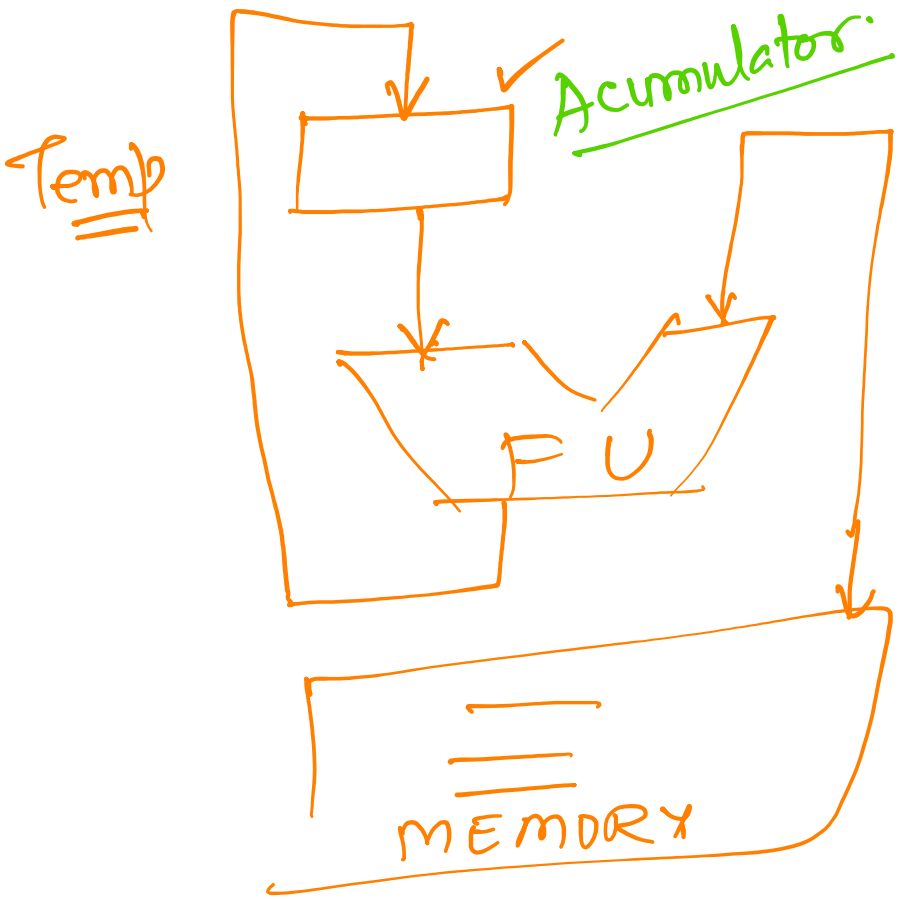
size $7+32 = 39$ ✓

$103 \rightarrow 39$

Add @ ← memory location

Sub b

Mul x



$a = b + c$

temp ← b
temp = temp + c
a ⟵ temp

$a = b + c + d$

temp ← b
temp ← temp + c
temp ← temp + d
a ⟵ temp

Accumulator Architectures

- Instruction set:

  add A, sub A, mult A, div A, . . .

  load A, store A

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

  load B

  mul C

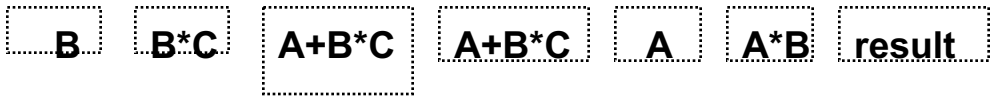
  add A

  store D

  load A

  mul B

  sub D



Accumulators: Pros and Cons

- Pros
  - Very low hardware requirements
  - Easy to design and understand

- Cons
  - Accumulator becomes the bottleneck
  - Little ability for parallelism or pipelining
  - High memory traffic

Memory-Memory Architectures

- Instruction set:

  (3 operands) add A, B, C

  sub A, B, C mul A, B, C

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

    mul D, A, B

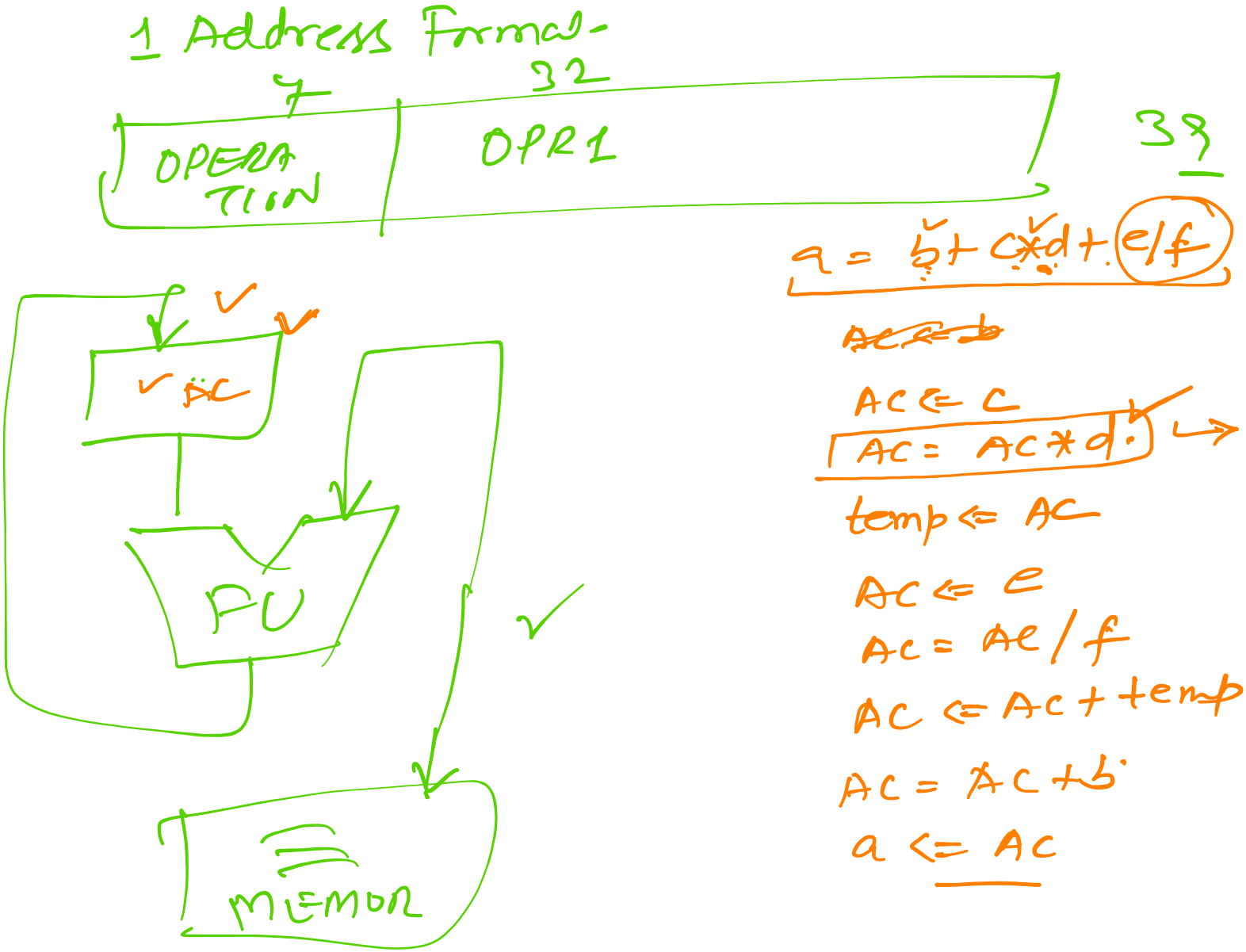
    mul E, C, B

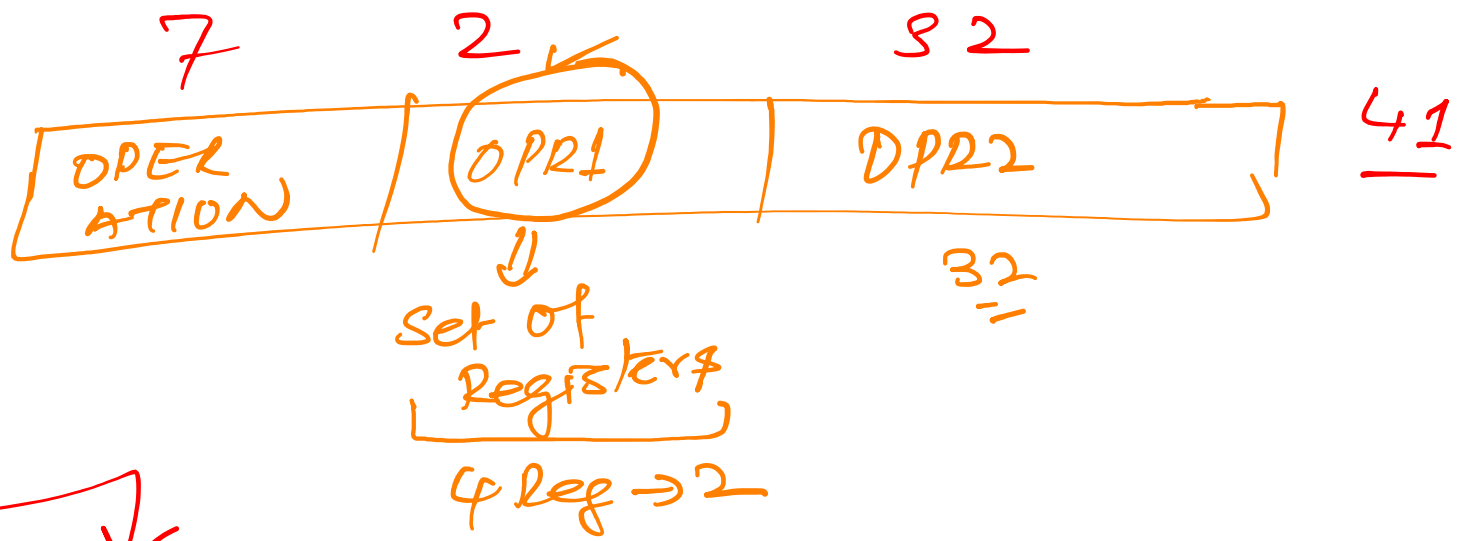
    add E, A, E

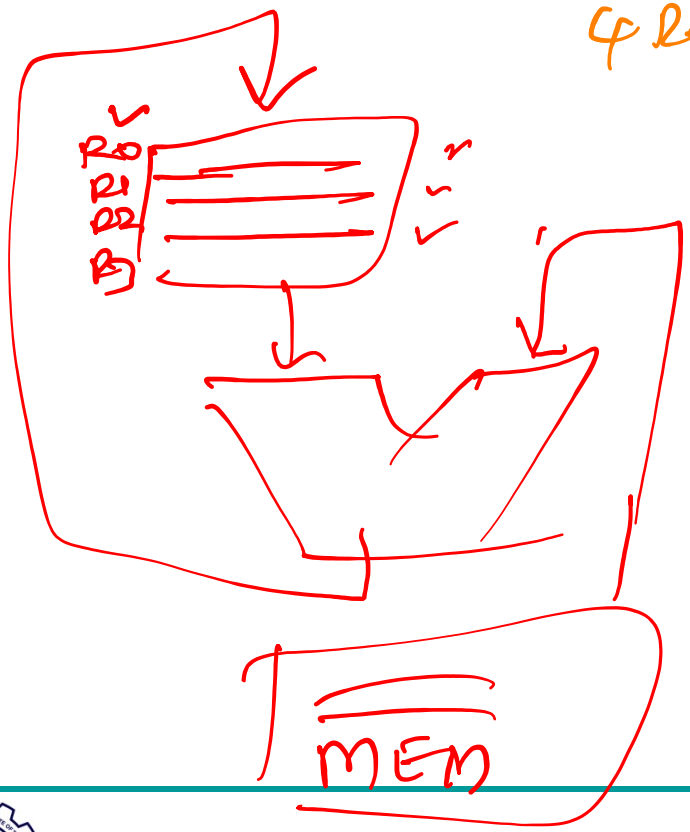
    sub E, D, E

Memory-Memory: Pros and Cons

- Pros
  - Requires fewer instructions (especially if 3 operands)
  - Easy to write compilers for (especially if 3 operands)
- Cons
  - Very high memory traffic (especially if 3 operands)
  - Variable number of clocks per instruction (especially if 2 operands)
  - With two operands, more data movements are required


1 Address Format-
7
32
39
OPERATION
OPR1
AC
PU
MEMORY
a = b + c*d + e/f
AC ⇐ b
AC ⇐ c
AC = AC * d
temp ⇐ AC
AC ⇐ e
AC = AC / f
AC ⇐ AC + temp
AC = AC + b
a ⇐ AC


7
2
32
41
OPER ATION
OPR1
OPR2
32
Set of Registers
4 Reg → 2


R0
R1
R2
R3
MEM

Register-Memory Architectures

- Instruction set:

  add R1, A sub R1, A mul R1, B

  load R1, A store R1, A

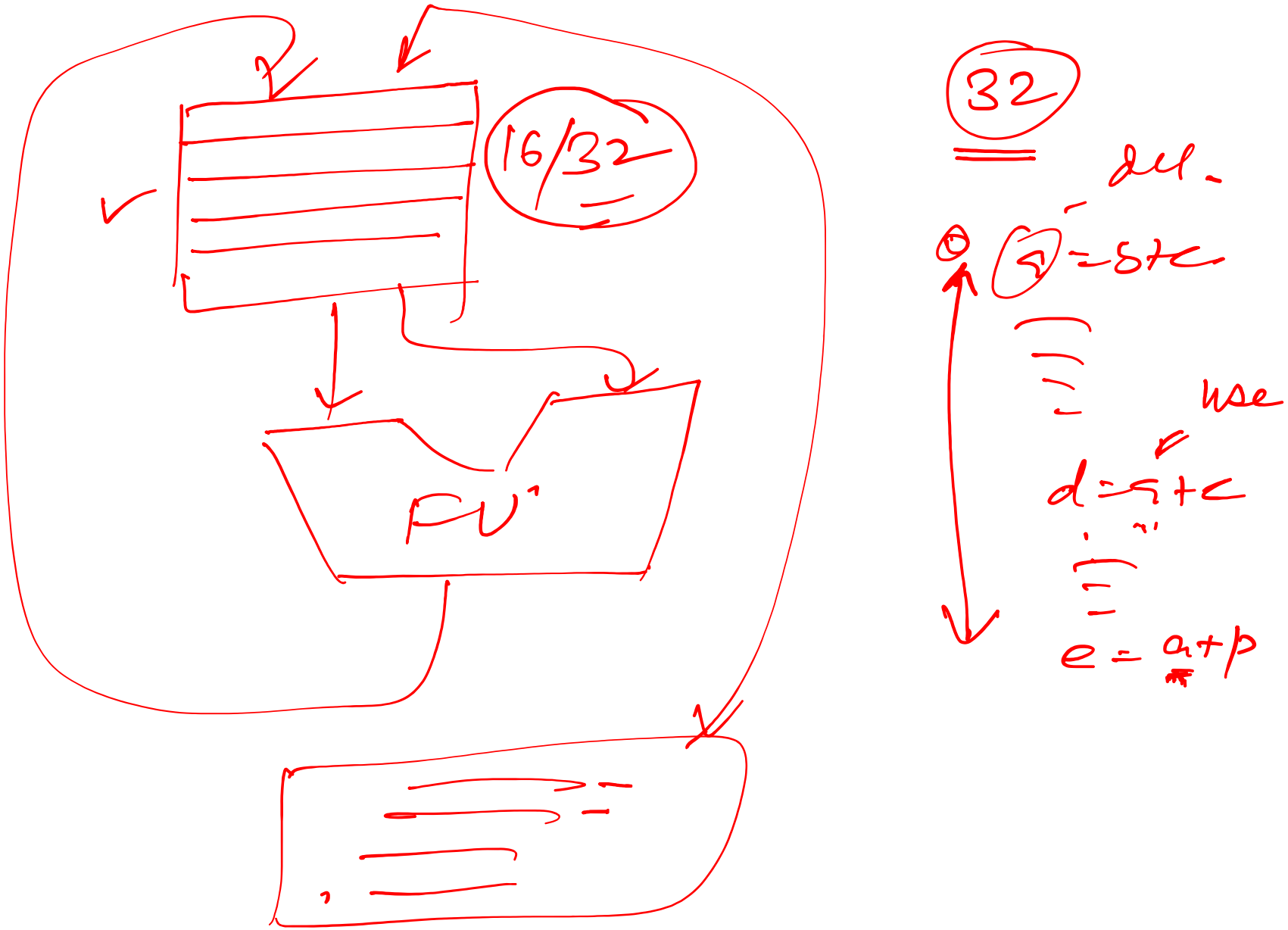
- Example: A*B - (A+C*B)

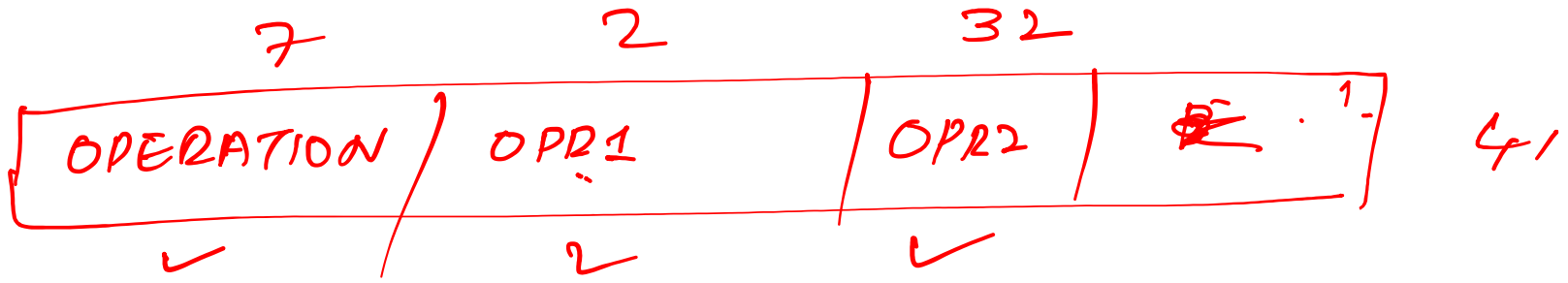
```
load R1, A
mul R1, B          /*    A*B          */
store R1, D
load R2, C
mul R2, B          /*    C*B          */
add R2, A          /*    A + CB             */
sub R2, D          /*    AB - (A + C*B)     */
```

Memory-Register: Pros and Cons

- Pros
  - Some data can be accessed without loading first
  - Instruction format easy to encode
  - Good code density

- Cons
  - Operands are not equivalent (poor orthogonality)
  - Variable number of clocks per instruction
  - May limit number of registers


16/32
PV
32
def.
=stc
use
d=
e=a+p


7
2
32
OPERATION
OPR1
OPR2
41

Thank You