CSE 112: Computer Organization (Section A)

Instructor: Sujay Deb

Class Room: C101 Meeting: Tue, Thr 12:00 Noon

INDRAPRASTHA INSTITUTE *of* INFORMATION TECHNOLOGY DELHI

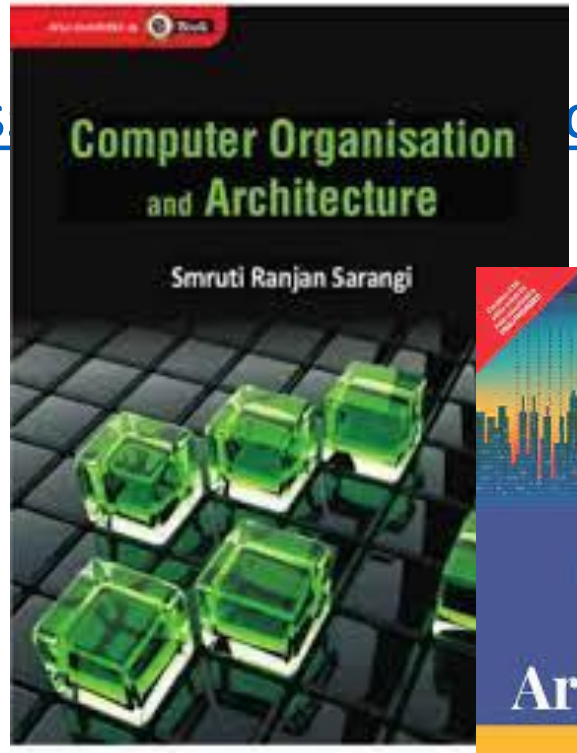
AMS Lab @IIITD

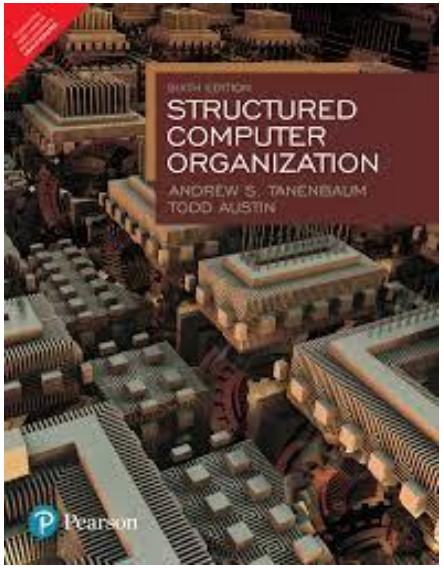
Course Information (Section A)

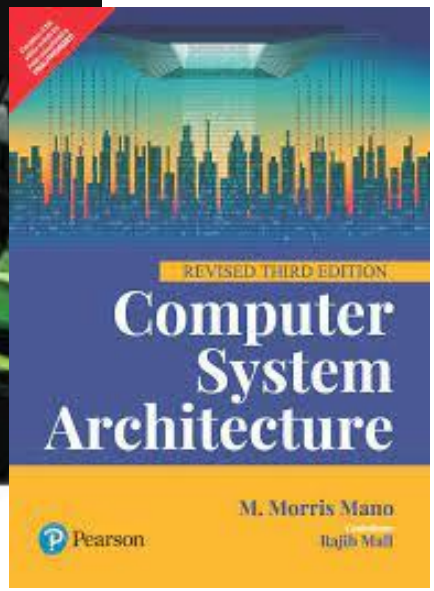
- Instructor:
  - Sujay Deb (sdeb@iiitd.ac.in)
  - Office A- 607 (Office Hours: Thr 3:00 - 4:00 PM)
- TA Support:
- Mentor TA:
  - Each student will have a mentor TA for
  - Tutorials
  - Dedicated Office hours
  - ..

Course Information

- Course web page: We will use Google Classroom for almost everything!
- **Class Code: wugafcs**
- Link: https://clas c/NDg1MTU3NDgyMDUz?cjc=wugafcs
- Textbook:







Grading structure

- Assignment: 10%
- Quizzes: 20%
- Midterm: 30%
- Final: 40%

- Lectures and discussions in class will cover basics of the course
- Tutorials, assignments and practice problems will help you gain a deep understanding of the subject

- Keep yourself in sync with the class, plan to spend at least 10-12 hrs per week on the course
- **Zero tolerance towards violation of Academic Integrity**

Course Objectives

1. Write programs in assembly language for a given machine (e.g., ARM)

2. Ability to analyze pipelines, data-path and interrupts, etc.

3. Simulate and appreciate different memory types, compare performance of cache memory, compare caches with different configuration

Course Description

- Introduction to computer systems
  - general overview of computer abstraction and technology

- Instruction set architecture
  - instruction type
  - format
  - operand
  - addressing mode

Course Description

- Computer arithmetic
  - addition
  - subtraction
  - multiplication and division
  - floating-point representation
- Basics of microprocessor
  - pipeline
  - datapath and control
  - data and control hazards

Course Description

- Parallelism
  - instruction level parallelism
- Memory hierarchy
  - exploiting locality using cache memory
  - virtual memory
- I/O and storage
  - performance of disk and file systems

Let's Pause and ask ourselves

- Why we should study CSE 112?
  - Innovations are based on Computing platforms
  - Know how about these platforms are essential for appropriate decision making
  - This course will help us understand the functioning of a computer

Fastest Computer/ Smallest Computer

- What is the speed of the world's fastest Computer?

TOP 10 Sites for June 2020

For more information about the sites and systems in the list, click on the links or view the complete list.

1-100 | 101-200 | 201-300 | 301-400 | 401-500

Rank	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
1	**Supercomputer Fugaku** - Supercomputer Fugaku, A64FX 48C 2.2GHz, Tofu interconnect D, Fujitsu RIKEN Center for Computational Science Japan	7,299,072	415,530.0	513,854.7	28,335
2	**Summit** - IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM DOE/SC/Oak Ridge National Laboratory United States	2,414,592	148,600.0	200,794.9	10,096
3	**Sierra** - IBM Power System AC922, IBM POWER9 22C 3.1GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM / NVIDIA / Mellanox	1,572,480	94,640.0	125,712.0	7,438

- Link: https://www.top500.org/lists/top500/2020/06/
- What is smallest Computer?

https://www.techrepublic.com/article/is-this-the-worlds-smallest-computer-ibm-chip-is-no-bigger-than-a-grain-of-salt/

CSE 112: Computer Organization (Section A)

Instructor: Sujay Deb

Lecture 2

INDRAPRASTHA INSTITUTE *of*
INFORMATION TECHNOLOGY
DELHI

AMS Lab @IIITD

Agenda

- What is Computer Architecture?
- Evolution of Computing Devices
- Moore’s Law
- Architecture vs Organization

Computer Architecture

Application

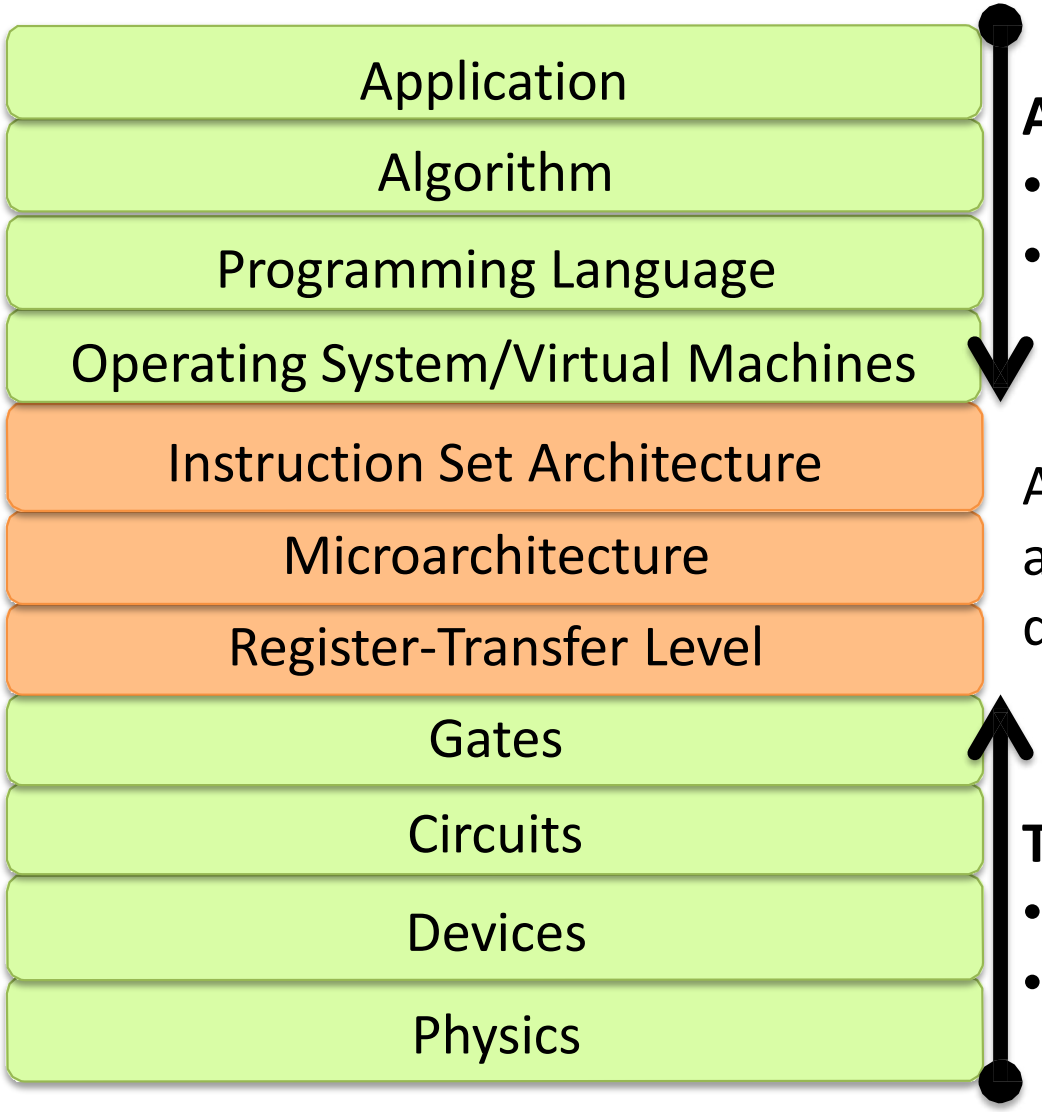
Application Requirements:

- Suggest how to improve architecture
- Provide revenue to fund development

Technology Constraints:

- Restrict what can be done efficiently
- New technologies make new arch possible

Abstractions in Modern Computing Systems



Application Requirements:

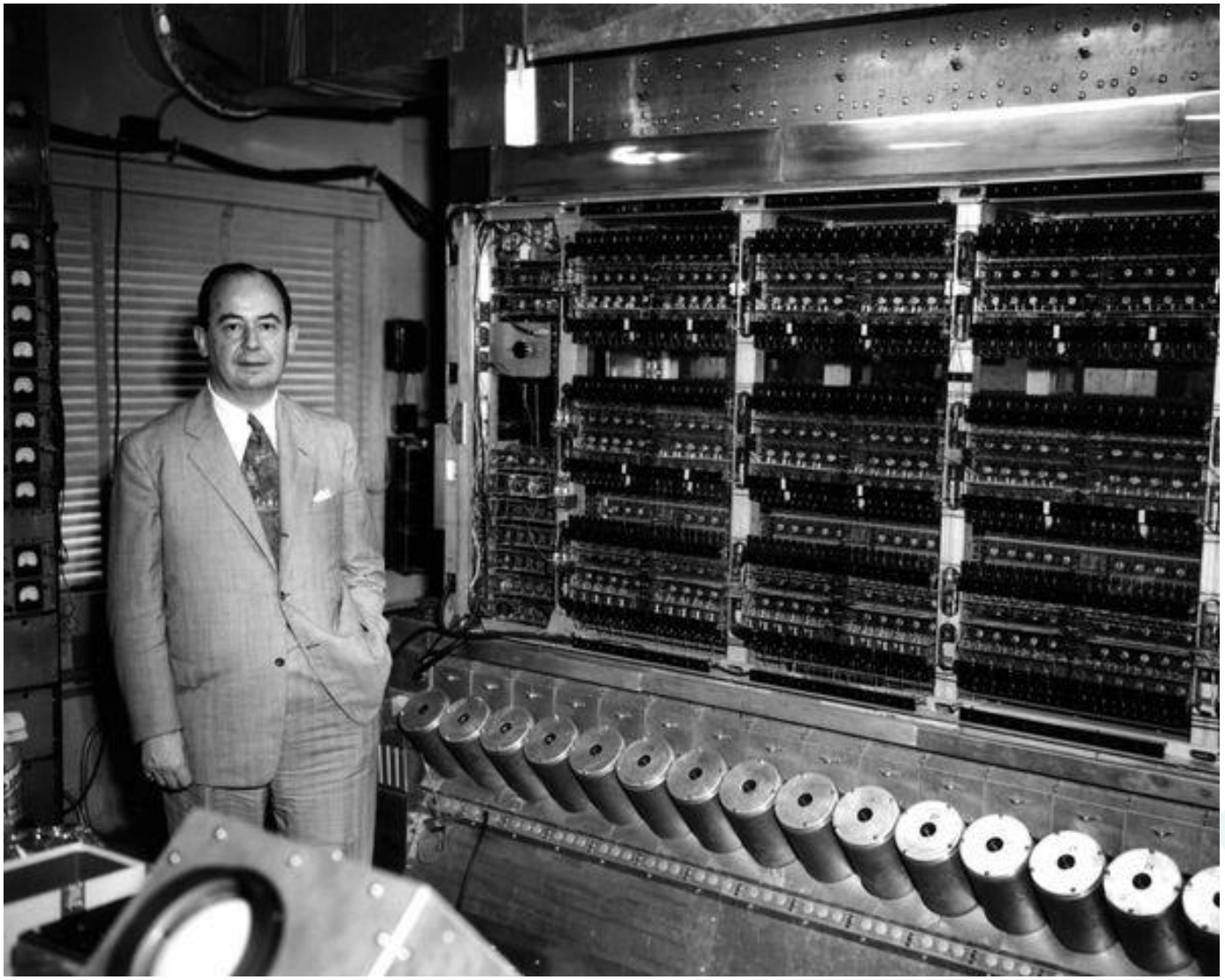
- Suggest how to improve architecture
- Provide revenue to fund development

Architecture provides feedback to guide application and technology research directions

Technology Constraints:

- Restrict what can be done efficiently
- New technologies make new arch possible

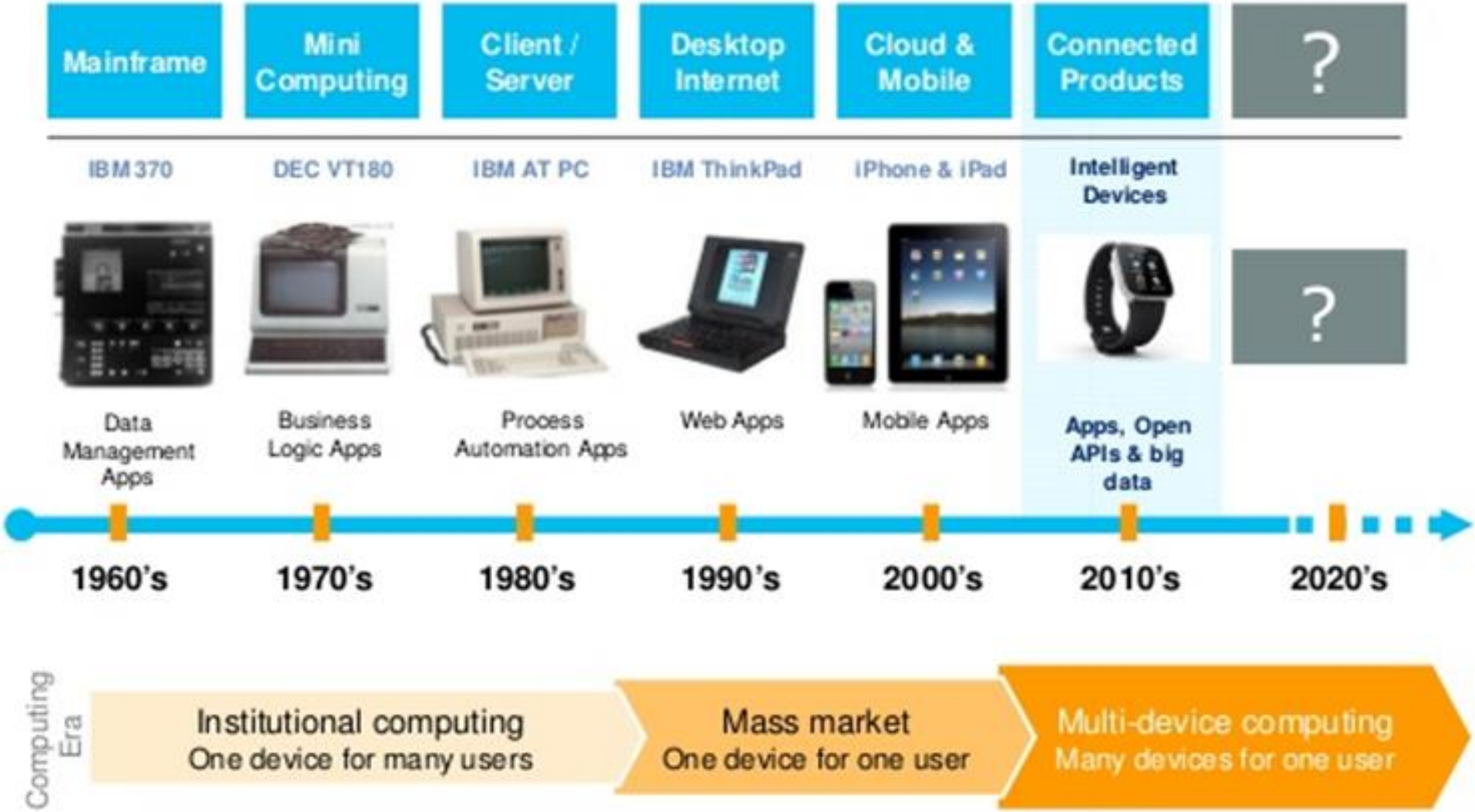
Computers Then..



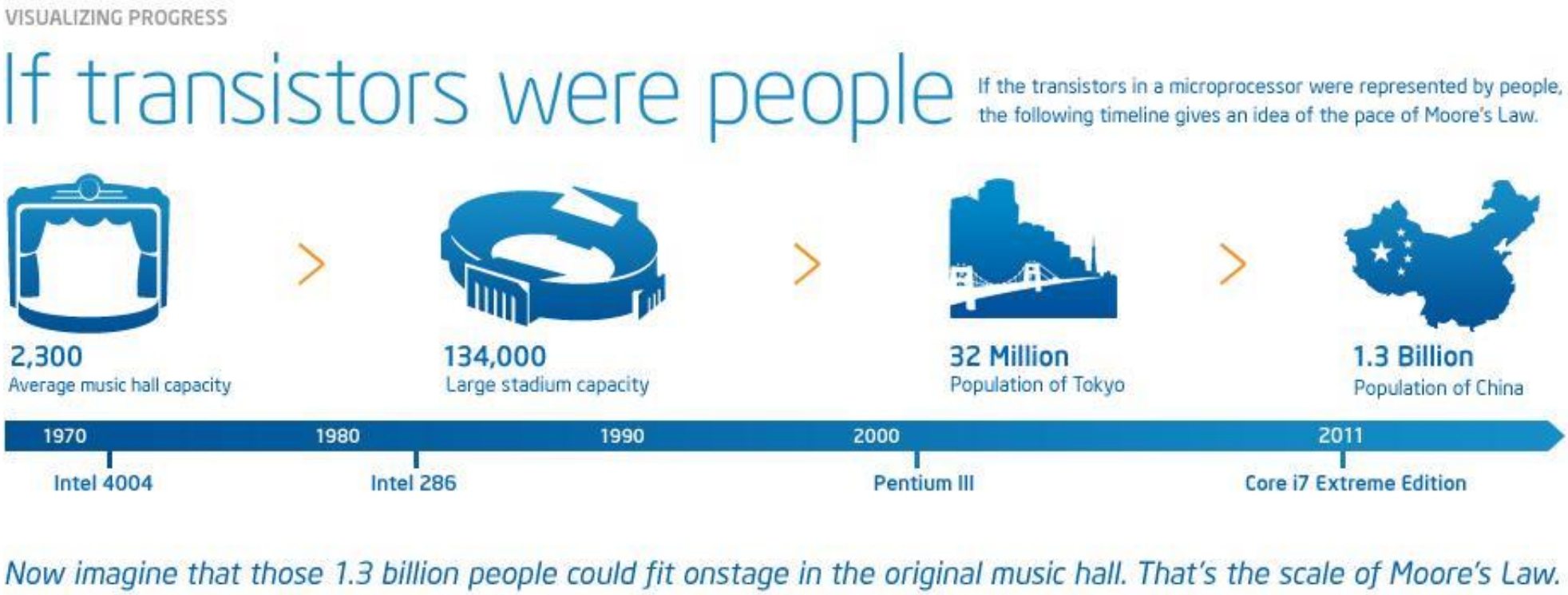
John von Neumann with the IAS Computer (Courtesy of the Shelby White and Leon Levy Archives Center, Institute for Advanced Study (IAS))

Computers Now

Evolution of Computing Devices



Moore's Law



Courtesy:

http://www.intel.com/content/www/us/en/silicon-innovations/moores-law-technology.html

CSE 112: Computer Organization (Section A)

Instructor: Sujay Deb

Lecture 3

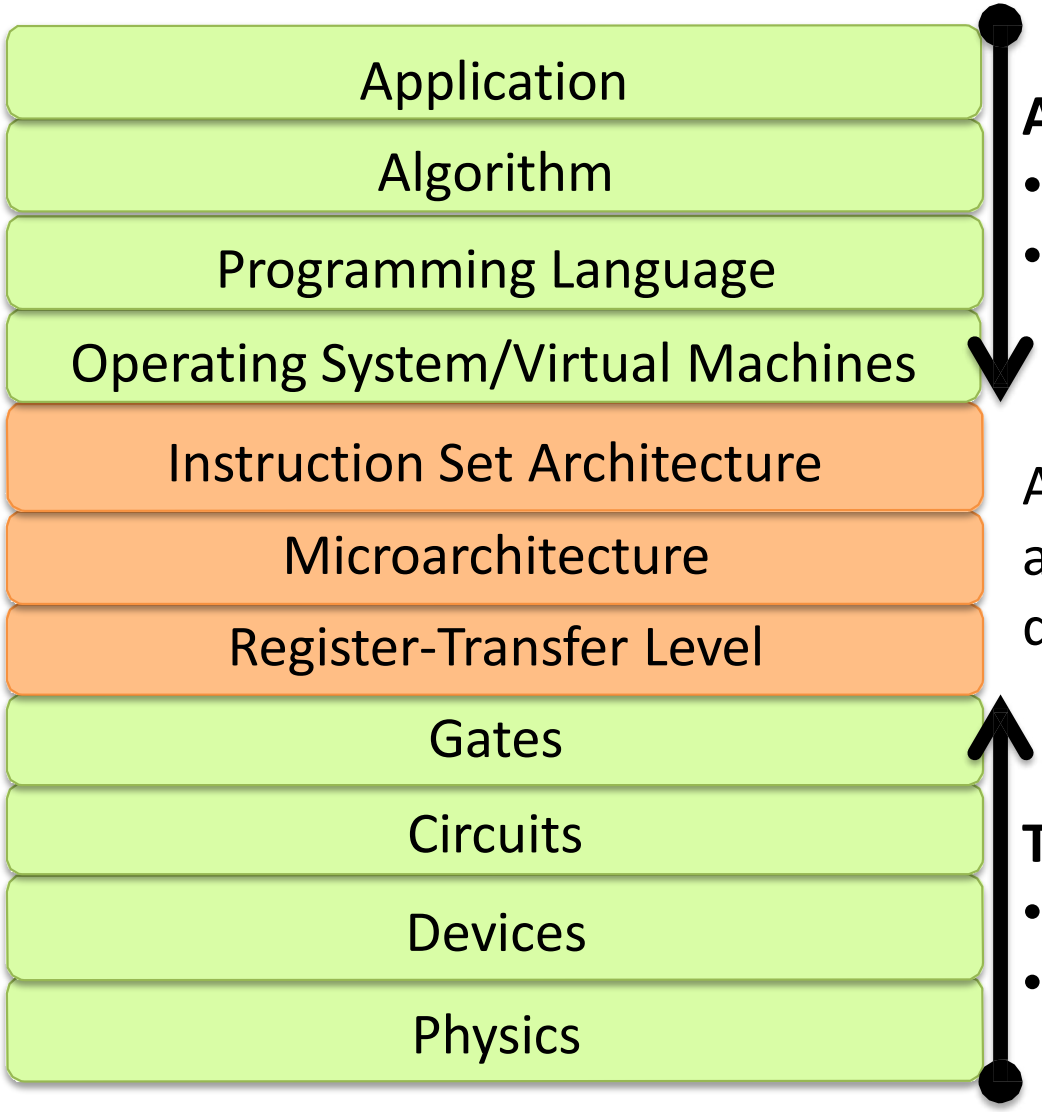
INDRAPRASTHA INSTITUTE *of*
INFORMATION TECHNOLOGY
DELHI

AMS Lab @IIITD

Agenda

- What is Computer Architecture?
- Evolution of Computing Devices
- Moore’s Law
- Architecture vs Organization

Abstractions in Modern Computing Systems



Application Requirements:

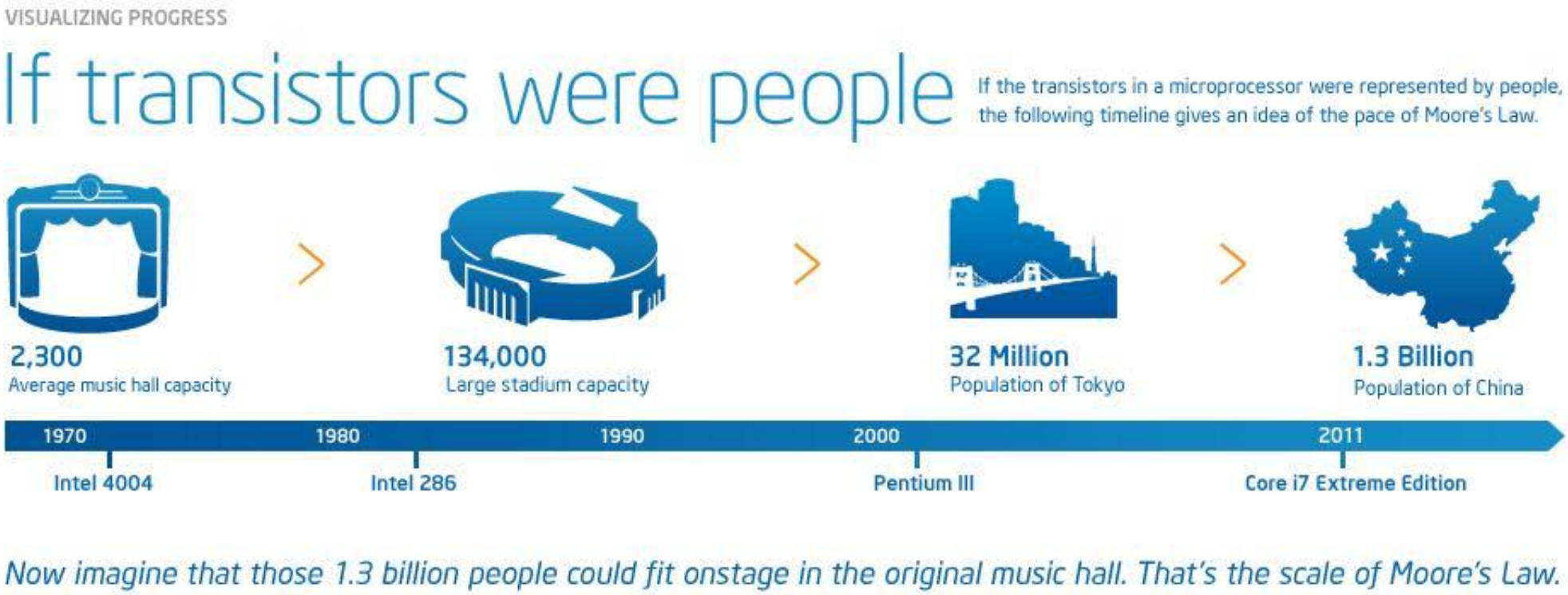
- Suggest how to improve architecture
- Provide revenue to fund development

Architecture provides feedback to guide application and technology research directions

Technology Constraints:

- Restrict what can be done efficiently
- New technologies make new arch possible

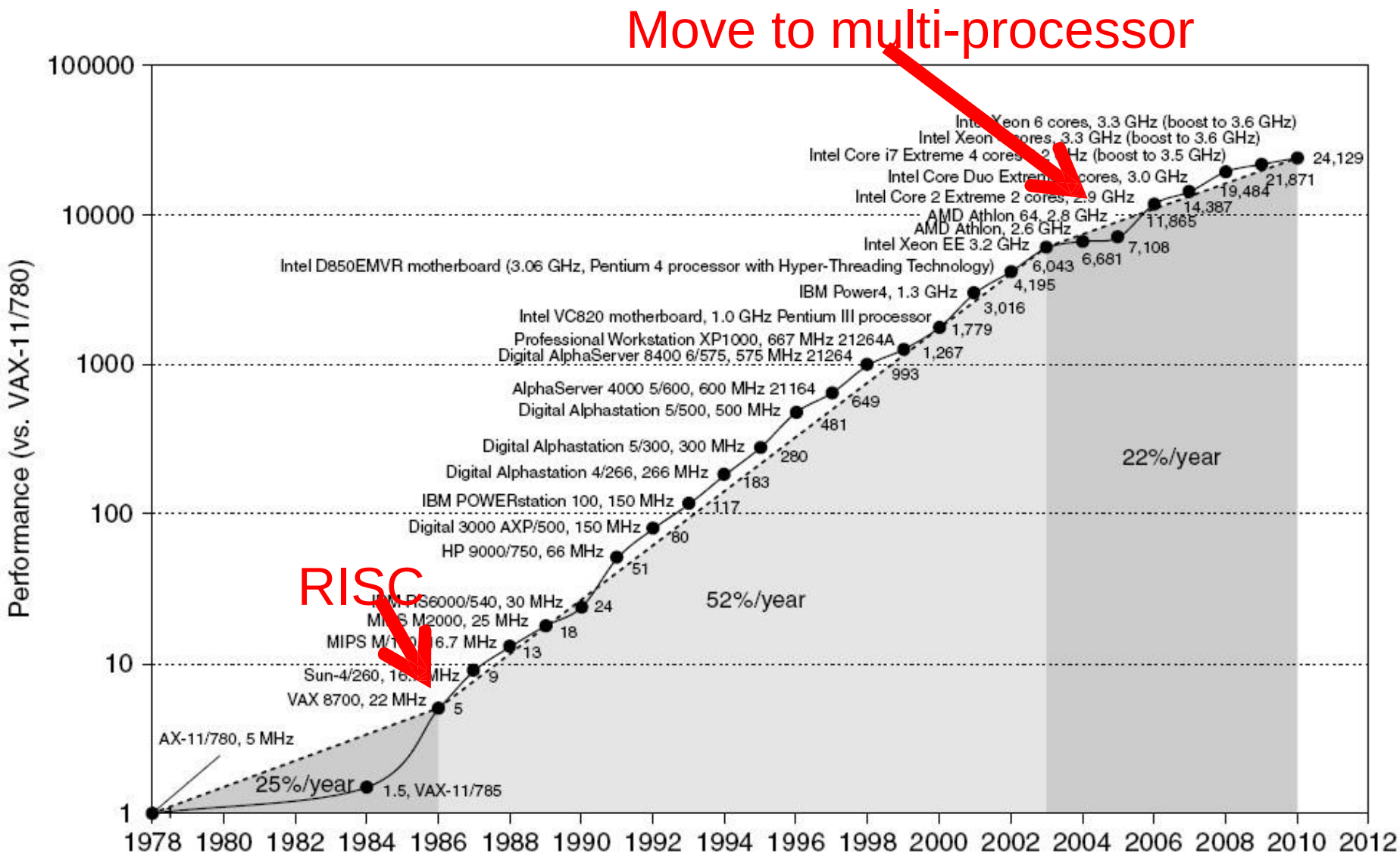
Moore's Law



Courtesy:

http://www.intel.com/content/www/us/en/silicon-innovations/moores-law-technology.html

Sequential Processor Performance

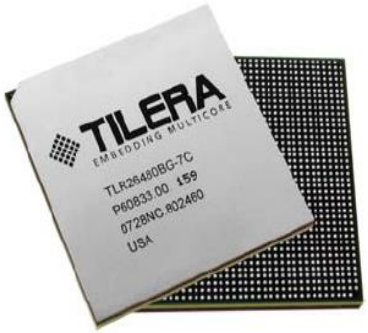


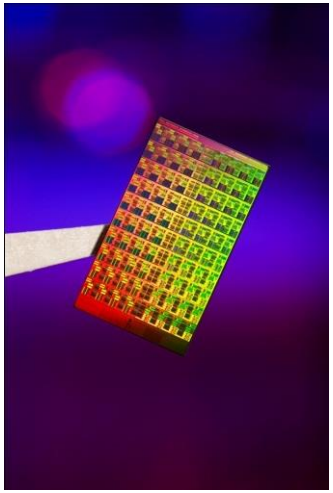
From Hennessy and Patterson Ed. 5 Image Copyright © 2011, Elsevier Inc. All rights Reserved.

The era of Many-Core systems

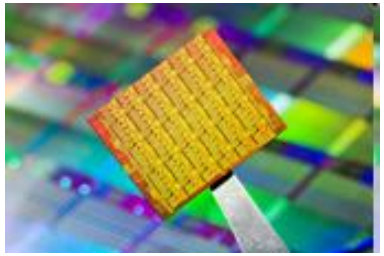
- How to keep up with demands on computational power?
  - Can not scale clock frequency
  - Solution: Increase number of cores - parallelism
    - Mass Market production of Intel, AMD dual-core and quad-core CPUs
    - Custom Systems-on-Chip (SoCs)
  - Many Core chips from Tilera for networking, cloud computing and multimedia applications.

Adapteva's Epiphany





Intel 80 core processor



Single-chip Cloud Computer

'Number of cores will double every 18 months'

- Prof. A. Agarwal, MIT, founder of Tilera Corporation

The era of Many-Core systems

- We are at the early stage of Many-core Processor evolution
  - Many-core is going to be ubiquitous
- Immense possibilities:
  - Server-type performance on handheld devices

Architecture vs. Organization

"Architecture"/ Instruction Set Architecture:

- Programmer visible state (Memory & Register)
- Operations (Instructions and how they work)
- Execution Semantics (interrupts)
- Input/Output
- Data Types/Sizes

Microarchitecture/ Organization:

- Tradeoffs on how to implement ISA for some metric (Speed, Energy, Cost)
- Examples: Pipeline depth, number of pipelines, cache size, silicon area, peak power, execution ordering, bus widths, ALU widths

Same Architecture
Different Microarchitecture

AMD Phenom X4

- X86 Instruction Set
- Quad Core
- 125W
- Decode 3 Instructions/Cycle/Core
- 64KB L1 I Cache, 64KB L1 D Cache
- 512KB L2 Cache
- Out-of-order
- 2.6GHz

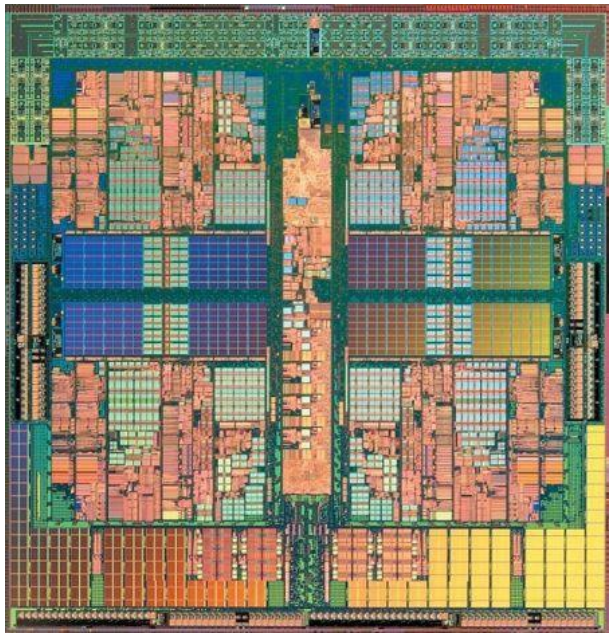


Image Credit: AMD

Intel Atom

- X86 Instruction Set
- Single Core
- 2W
- Decode 2 Instructions/Cycle/Core
- 32KB L1 I Cache, 24KB L1 D Cache
- 512KB L2 Cache
- In-order
- 1.6GHz

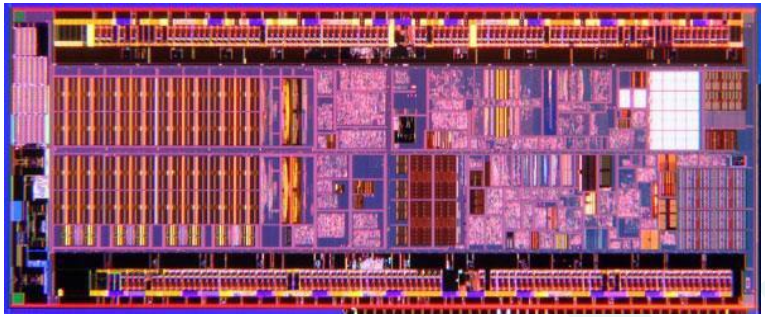


Image Credit: Intel

Different Architecture
Different Microarchitecture

AMD Phenom X4

- X86 Instruction Set
- Quad Core
- 125W
- Decode 3 Instructions/Cycle/Core
- 64KB L1 I Cache, 64KB L1 D Cache
- 512KB L2 Cache
- Out-of-order
- 2.6GHz

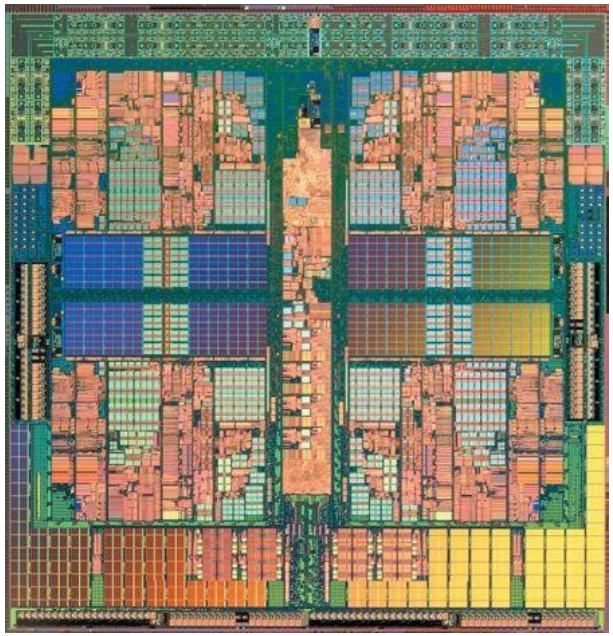


Image Credit: AMD

IBM POWER7

- Power Instruction Set
- Eight Core
- 200W
- Decode 6 Instructions/Cycle/Core
- 32KB L1 I Cache, 32KB L1 D Cache
- 256KB L2 Cache
- Out-of-order
- 4.25GHz

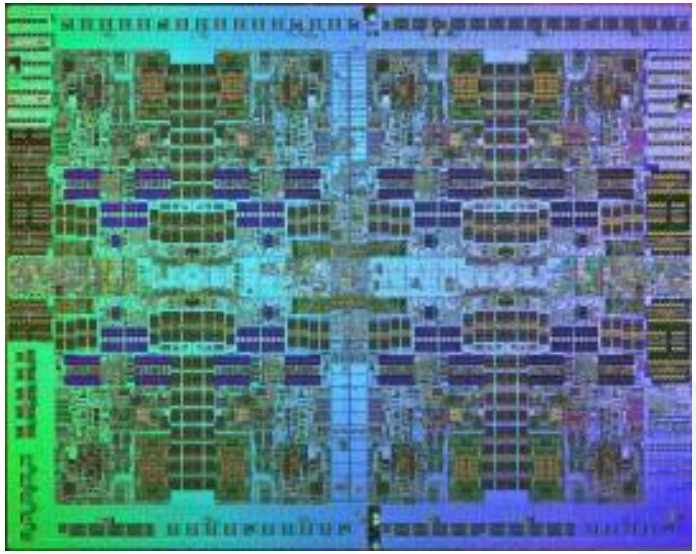


Image Credit: IBM

Courtesy of International Business Machines Corporation, © International Business Machines Corporation.

Recap

- What is Computer Architecture?
- Evolution of Computing Devices
- Moore’s Law
- Architecture vs Organization

CSE112: Computer Organization (Section A)

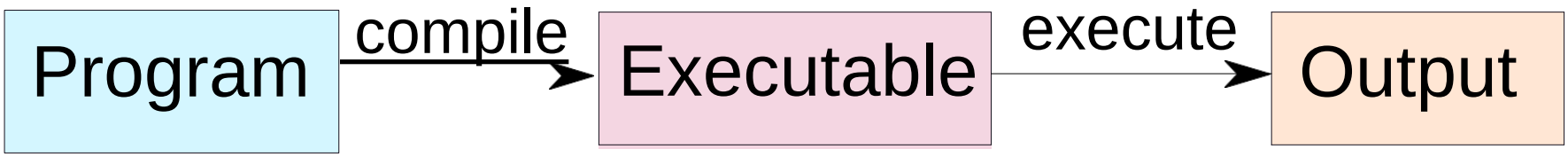
Instructor: Sujay Deb

Lecture 4

INDRAPRASTHA INSTITUTE of INFORMATION TECHNOLOGY DELHI

AMS Lab @IIITD

How to Instruct a Computer?



- Write a program in a high level language – C, C++, Java
- **Compile** it into a format that the computer understands
- Execute the program

Instruction Set Architecture:

The semantics of all the instructions supported by a processor is known as its instruction set architecture (ISA). This includes the semantics of the instructions themselves, along with their operands, and interfaces with peripheral devices.

Features of an ISA

- Example of instructions in an ISA
  - Arithmetic instructions : add, sub, mul, div
  - Logical instructions : and, or, not
  - Data transfer/movement instructions
- Complete
  - It should be able to implement all the programs that users may write.

Features of an ISA – II

- Concise
  - The instruction set should have a limited size. Typically an ISA contains 32-1000 instructions.
- Generic
  - Instructions should not be too specialized, e.g. add14 (adds a number with 14) instruction is too specialized
- Simple
  - Should not be very complicated.

Designing an ISA

- Important questions that need to be answered :
  - How many instructions should we have ?
  - What should they do ?
  - How complicated should they be ?

Two different paradigms : RISC and CISC

RISC
(Reduced Instruction Set Computer)

CISC
(Complex Instruction Set Computer)

RISC vs CISC

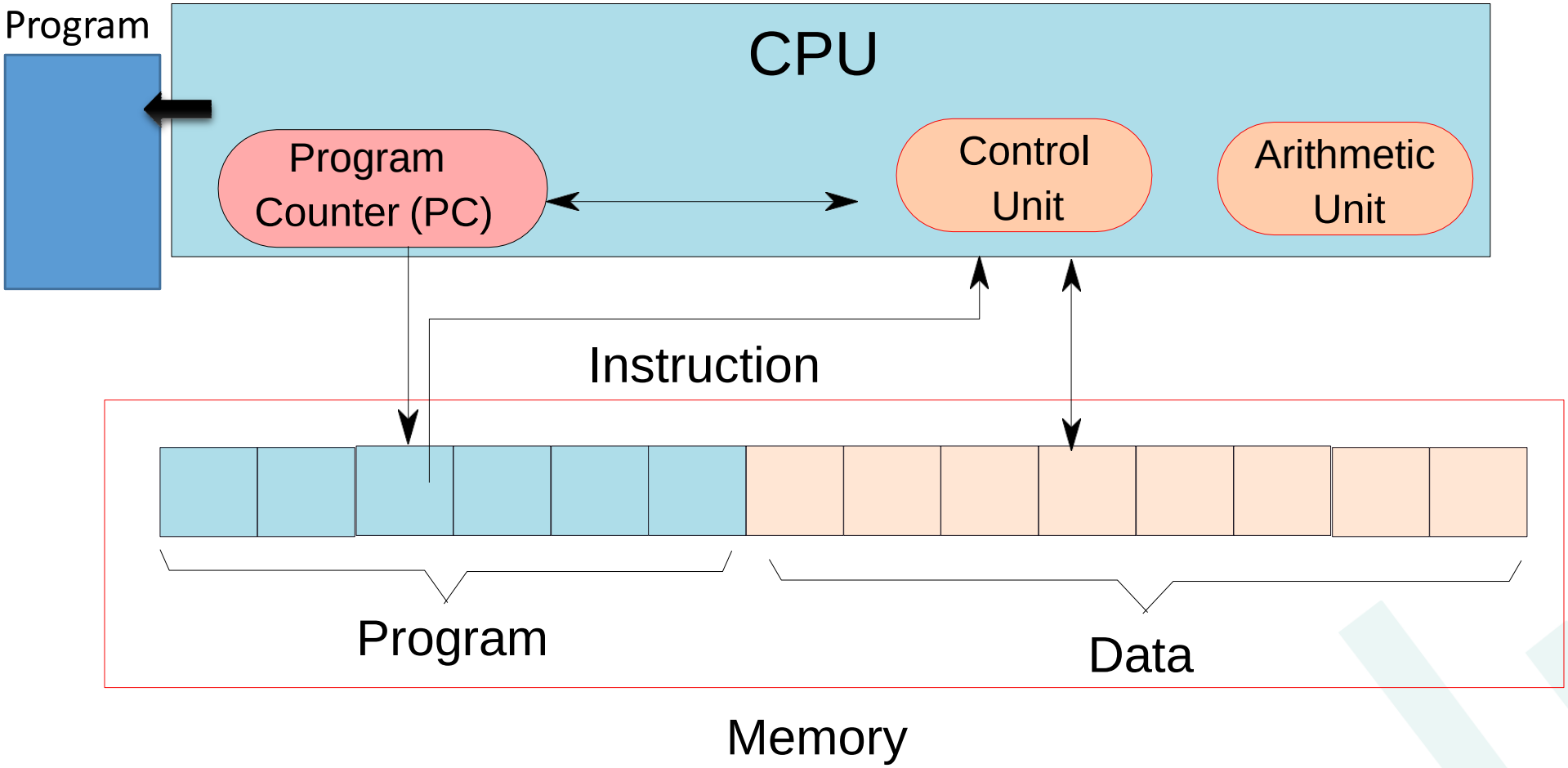
A reduced instruction set computer (RISC) implements simple instructions that have a simple and regular structure. The number of instructions is typically a small number (64 to 128). Examples: ARM, IBM PowerPC, HP PA-RISC

A complex instruction set computer (CISC) implements complex instructions that are highly irregular, take multiple operands, and implement complex functionalities. Secondly, the number of instructions is large (typically 500+). Examples: Intel x86, VAX

Summary so far …

- Computers are dumb yet ultra-fast machines.
- Instructions are basic rudimentary commands used to communicate with the processor. A computer can execute billions of instructions per second.
- The compiler transforms a user program written in a high level language such as C to a program consisting of basic machine instructions.
- The instruction set architecture(ISA) refers to the semantics of all the instructions supported by a processor.
- The instruction set needs to be complete. It is desirable if it is also concise, generic, and simple.

Compute Block Diagram



Elements of a Computer

- Memory (array of bytes) contains
  - The program, which is a sequence of instructions
  - The program data → variables, and constants
- The program counter(PC) points to an instruction in a program
  - After executing an instruction, it points to the next instruction by default
  - A branch instruction makes the PC point to another instruction (not in sequence)
- CPU (Central Processing Unit) contains the
  - Program counter, instruction execution units

Let us now design an ISA …

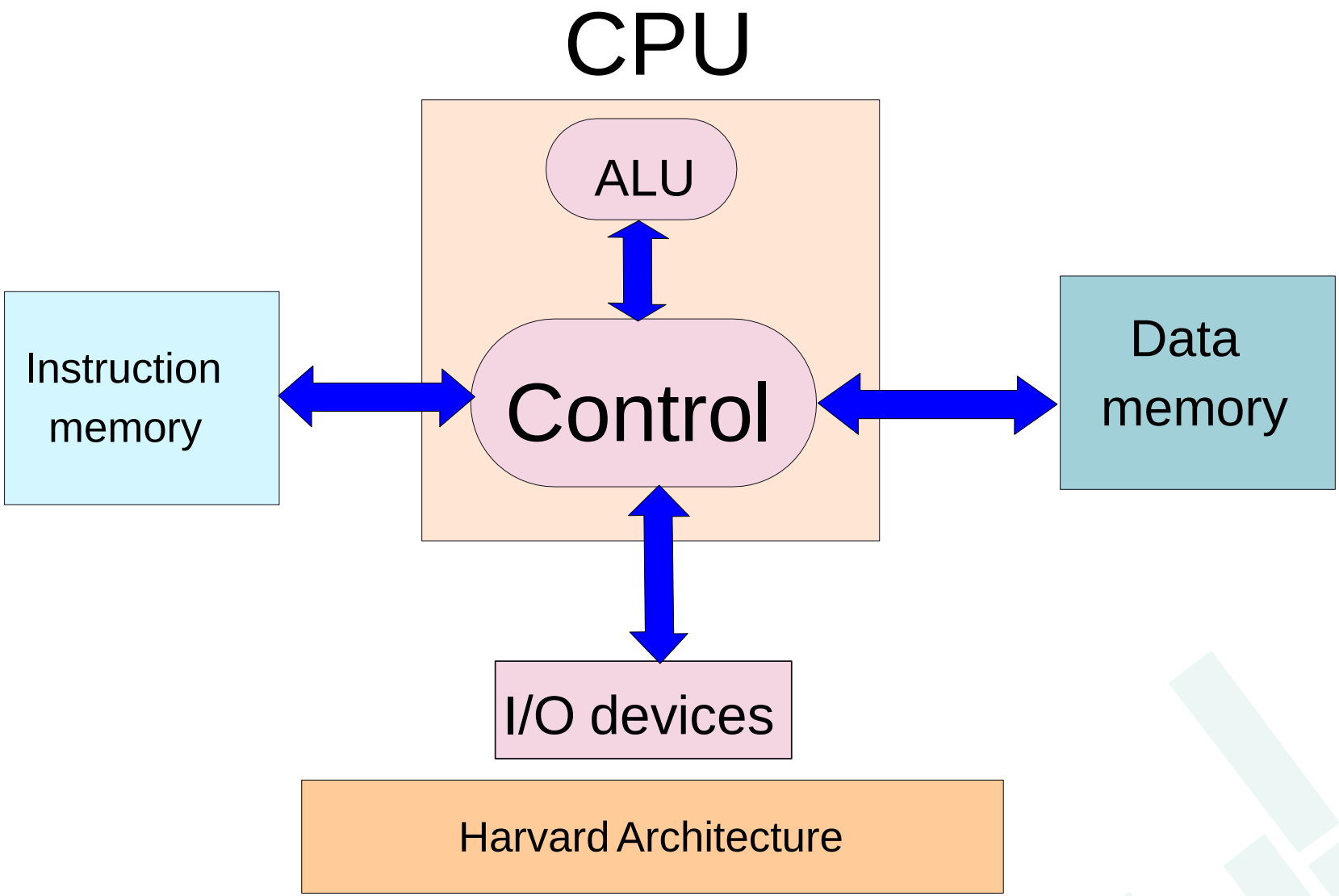
- Single Instruction ISA
  - sbn – subtract and branch if negative
- Add (a + b) (assume temp = 0)

```
1: sbn temp, b, 2
2: sbn a, temp, exit
```

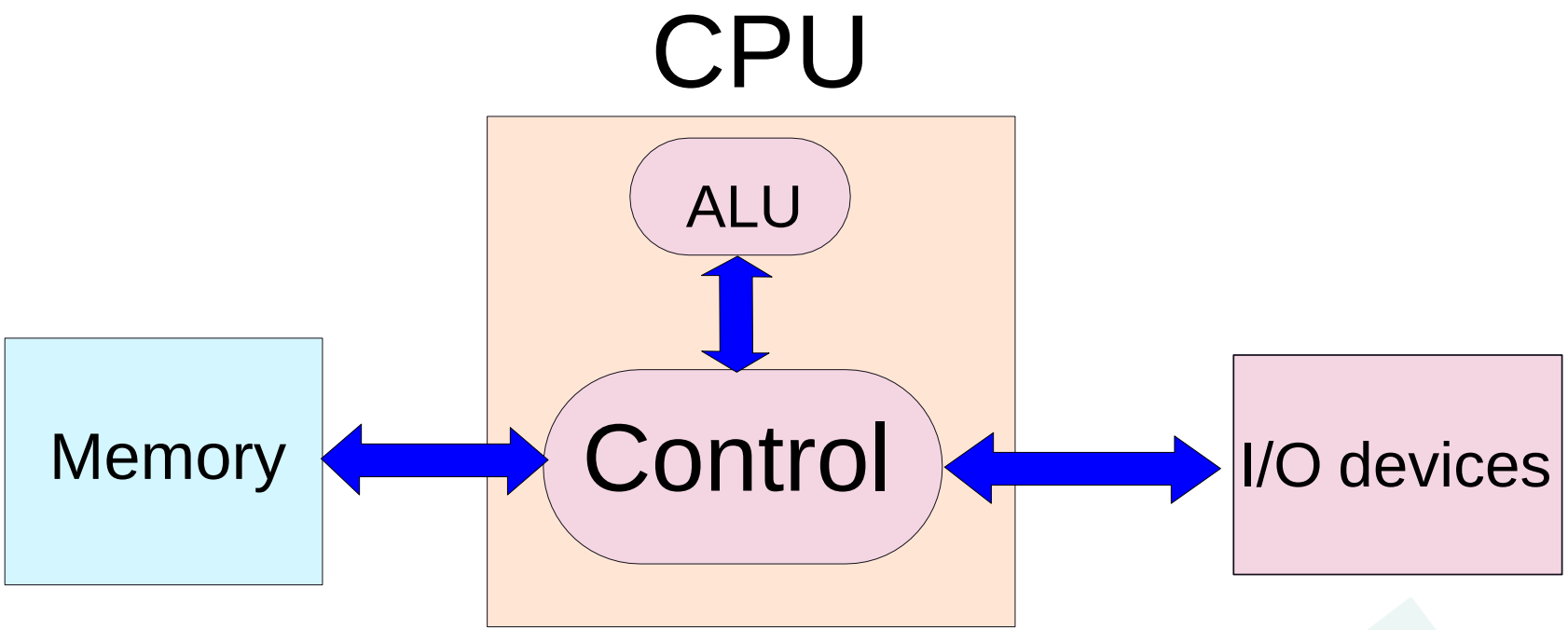
Multiple Instruction ISA

- Arithmetic Instructions
  - add, subtract, multiply, divide
- Logical Instructions
  - or, and, not
- Move instructions
  - Transfer values between memory locations
- Branch instructions
  - Move to a new program location, based on the values of some memory locations

Designing Practical Machines



Von-Neumann Architecture



Problems with Harvard/ Von-Neumann Architectures

- The memory is assumed to be one large array of bytes
  - It is very very slow

General Rule: Larger is a structure, slower it is

- Solution:
  - Have a small array of named locations **(registers)** that can be used by instructions
  - This small array is very fast

Insight: Accesses exhibit locality (tend to use the same variables frequently in the same window of time)

Uses of Registers

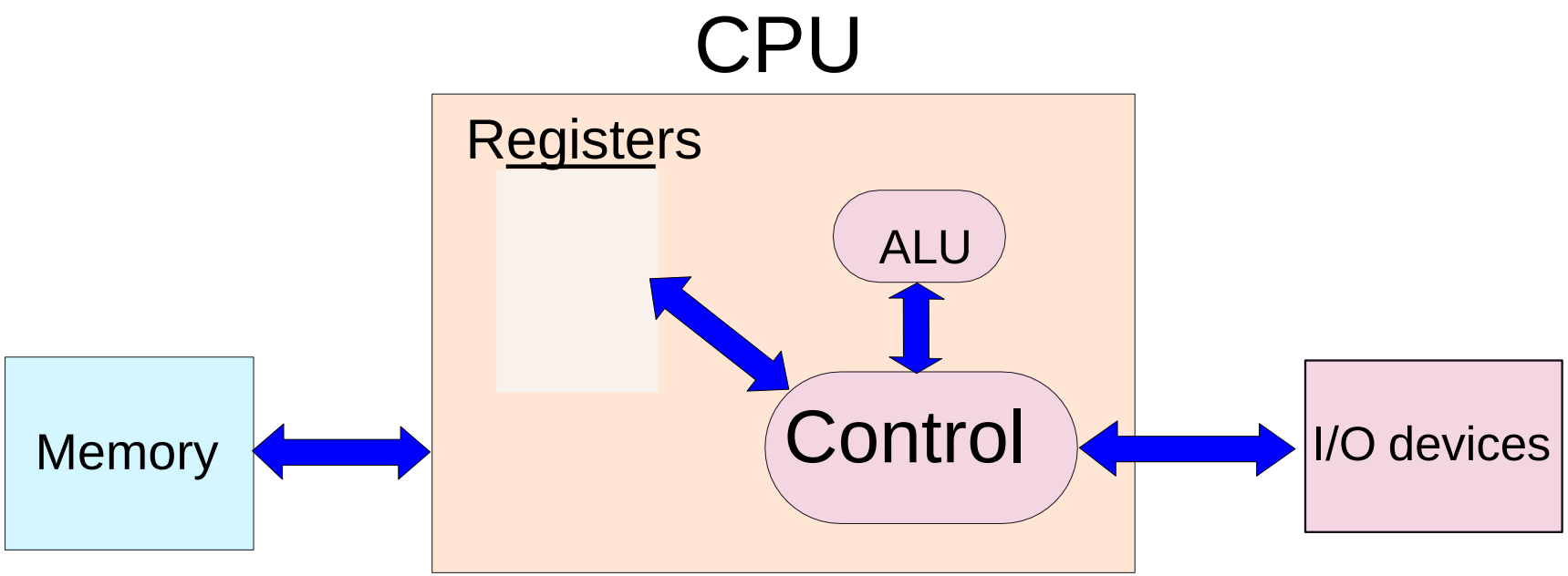
- A CPU (Processor) contains set of registers (16-64)
- These are named storage locations.
- Typically values are loaded from memory to registers.
- Arithmetic/logical instructions use registers as input operands
- Finally, data is stored back into their memory locations.

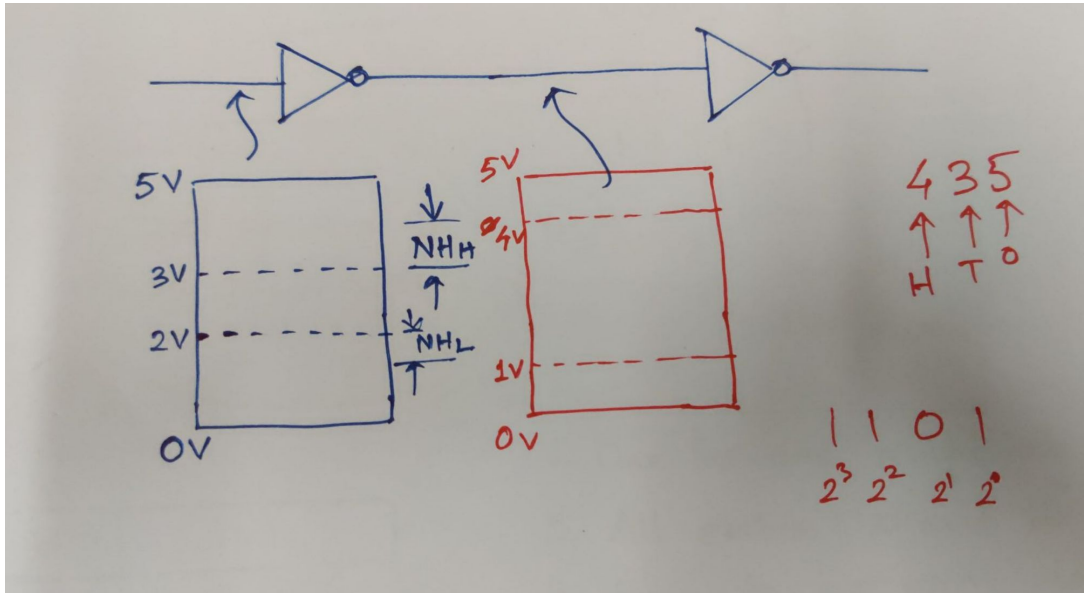
Example of a Program in Machine Language with Registers

```
1: r1 = mem[b] // load b
2: r2 = mem[c] // load c
3: r3 = r1 + r2 // add b and c
4: mem[a] = r3 // save the result
```

- r1, r2, and r3, are registers
- mem → array of bytes representing memory

Machine with Registers




5V
3V
2V
0V
NHH
NHL
5V
4V
1V
0V
4 3 5
H T O
1 1 0 1
2³ 2² 2¹ 2⁰

Negative Integers.

4 bit .

0000
⋮
1111

Set of +ve & -ve numbers $\xleftrightarrow{\text{Mapping}}$ Set of +ve numbers.

S

1. One-to-one mapping
2. All entries should be mapped.
3. Easy to perform addition/subtraction etc..

sign bit	$\lvert u \rvert$

$$\text{Sign bit} = \begin{cases} 1, & u < 0 \\ 0, & u > 0 \end{cases}$$

$5 \rightarrow 0101$

$-5 \rightarrow 1101$

1's complement

$3 \rightarrow 0011$

$-3 \rightarrow 1100$

x	$\bar{x} = 1 - x$
1	0
0	1

$15 - 3$

$= 12$

$= 1100$

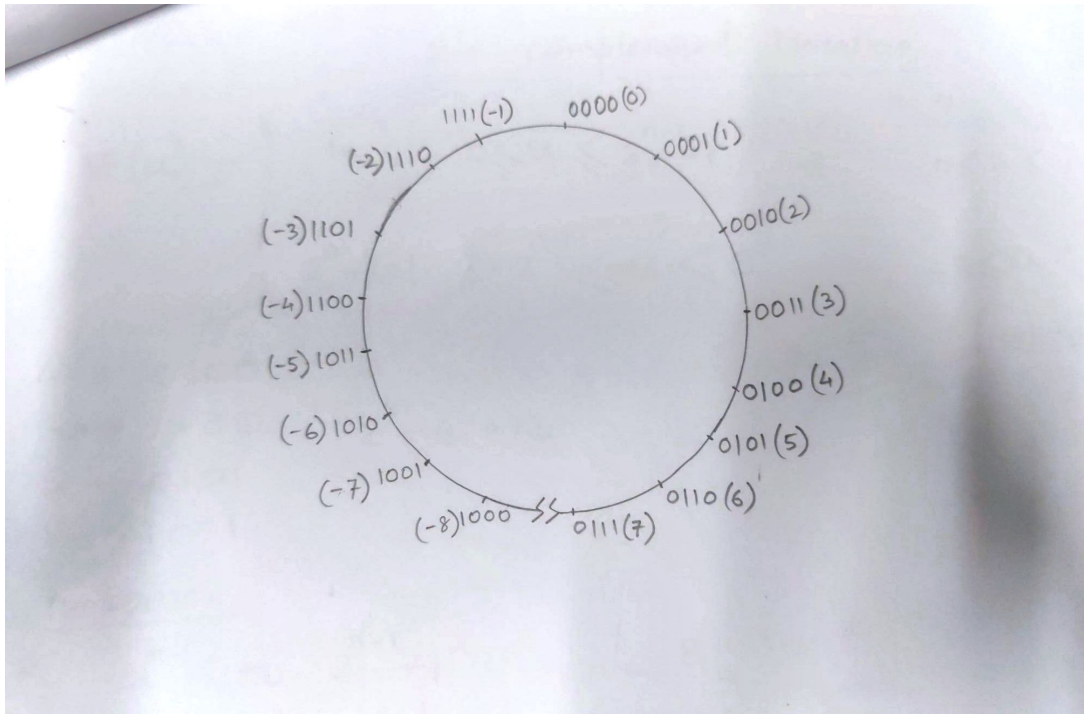
$$\boxed{2^n - 1 - |u|}$$

$$F(u) = \begin{cases} u, & u > 0 \\ \sim(|u|) \text{ or } (2^n - 1 - |u|), & u < 0 \end{cases}$$

-7 to 8

$-3 + 7 = 4 \rightarrow 0100$

$8 + 7 = 15 \rightarrow 1111$


0000 (0)
0001 (1)
0010 (2)
0011 (3)
0100 (4)
0101 (5)
0110 (6)
0111 (7)
(-8) 1000
(-7) 1001
(-6) 1010
(-5) 1011
(-4) 1100
(-3) 1101
(-2) 1110
1111 (-1)

2's complement Notation

$$F(u) = \begin{cases} u, & 0 \leq u \leq 2^{n-1} - 1 \\ 2^n - |u|, & -2^{n-1} \leq u < 0 \end{cases}$$

$0 \leq u \leq 7$

$-8 \leq u < 0$

$n = 4$

$4 \leftarrow 0100$

$-4 \leftarrow 2^4 - 4 = 12 = 1100$

Properties.

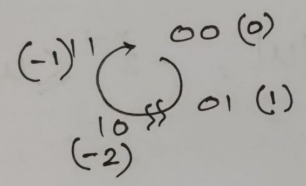
* -2^{n-1} to $2^{n-1} - 1$
* Unique representation of 0 (0000)
* MSB is equal to sign bit.

$\underset{2^3}{1}\,\underset{2^2}{1}\,\underset{2^1}{0}\,\underset{2^0}{1}$

Number Circle

$\boxed{5 + (-2)}$

2 bit



M + N

5 (-2)

$2^n - N$

$2^4 - 2 = 14$

Computing 2's complement

* $2^n - u$

$$= \underbrace{2^n - 1 - u}_{\text{1's complement.}} + 1$$

$$\boxed{2 \times (-3)} =$$

$$\downarrow \qquad \downarrow$$

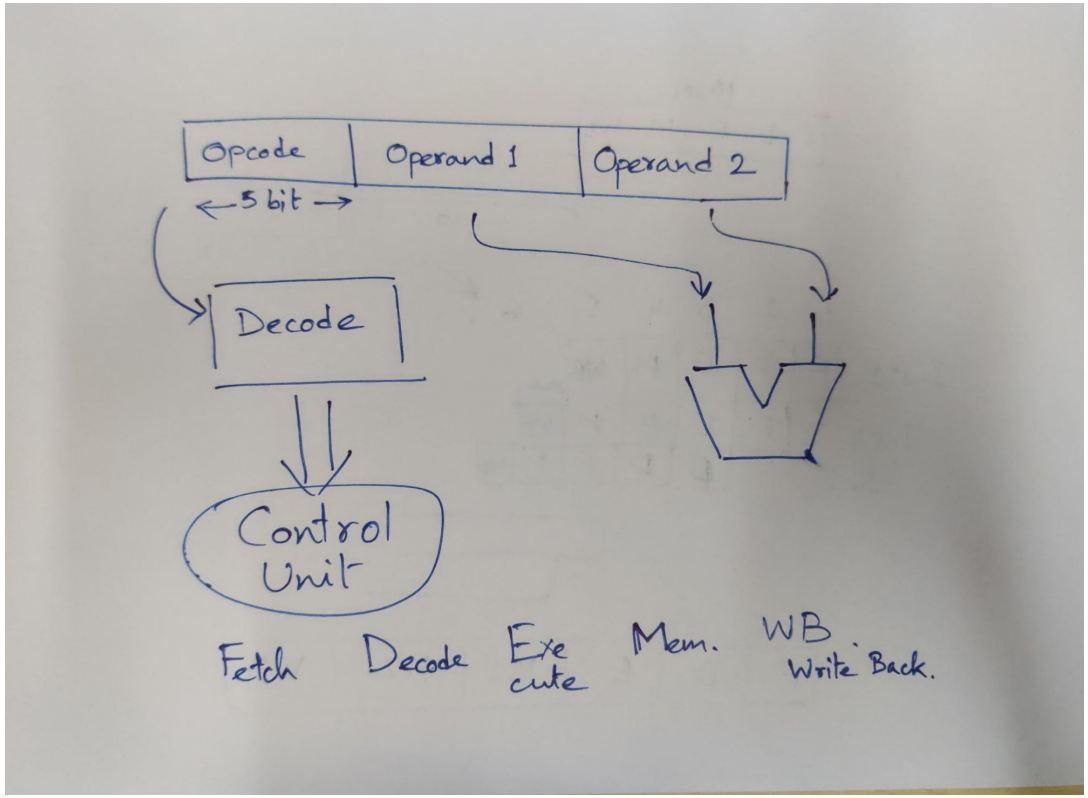
$$0010 \qquad 1101 \qquad = 26 \bmod 2^4 = 10$$

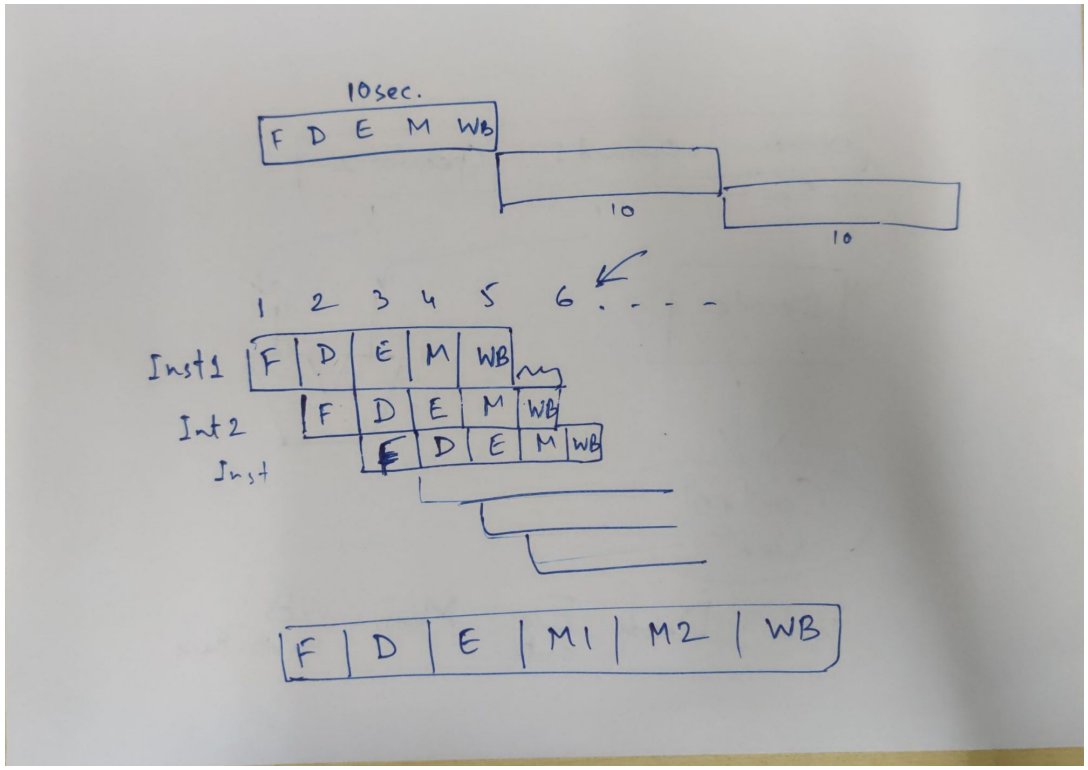
$$= 1010$$

$$a \equiv b \qquad \boxed{a = b \bmod 2^n}$$

$$a = 3 \qquad b = 19 \bmod 16$$

$$= 3$$


Opcode
Operand 1
Operand 2
5 bit
Decode
Control Unit
Fetch
Decode
Exe cute
Mem.
WB
Write Back.


10sec.
F D E M WB
10
10
1 2 3 4 5 6 . . . .
Inst1
F D E M WB
Int2
F D E M WB
Inst
F D E M WB
F D E M1 M2 WB

Sign Extension

Convert a n bit number to a ~~m~~ m - bit number (2's complement)

$$\underset{8}{m} > \underset{4}{n}.$$

+ve

Add $(m-n)$ 0's in the MSB positions.

$0100 \longrightarrow 00000100$

-ve:

$$F(u) = 2^n - |u|$$

$$F'(u) = 2^m - |u|$$

$$2^m - u - (2^n - u)$$

$$= 2^m - 2^n$$

$$= 2^n + 2^{n+1} + 2^{n+2} \ldots\ldots + 2^{m-1}$$

$$= \underbrace{11111111}_{m-n}\ \underbrace{0000\cdots0}_{n.}$$

$$F_8(-4) = 11110000 + 1100$$

$$= 11111\underbrace{100}$$

$$F_4(-4) = 2^4 - 4$$

$$= 1100$$

To convert a -ve number
Add $(m-n)$ 1's in the MSB positions.

* In both cases extend the sign bit by $m-n$ positions.

<u>Overflow theorem:</u>

$u+v$

$u \cdot v < 0$, there will never be an overflow.

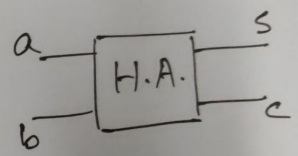
$u \cdot v > 0$, an overflow is possible.

u & v are of same sign.

→ the sign of the result is different from u & v then there is overflow.

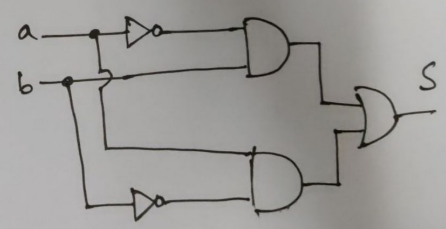
Truth Table of Half adder

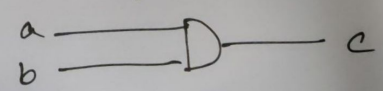
a	b	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



$$S = \bar{a}b + a\bar{b}$$

$$C = a.b.$$





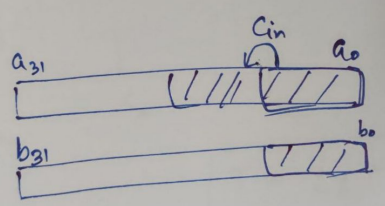
FA truth table

a	b	C_{in}	S	C_{out}
0	0	0	0	0
0	1	0	1	0
1	0	0	1	0
1	1	0	0	1
0	0	1	1	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	1

$$S = a \oplus b \oplus C_{in}$$

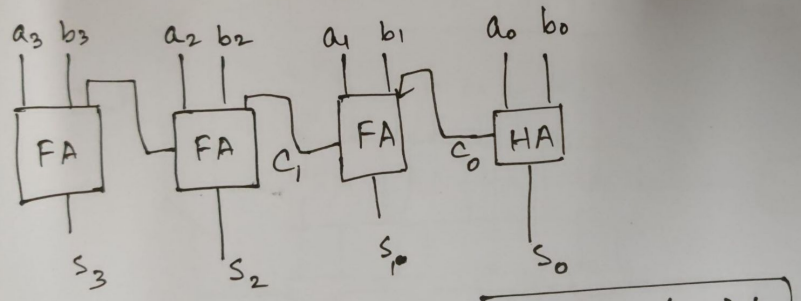
$$= a \cdot \bar{b}\, \bar{C}_{in} + \bar{a} \cdot b \cdot \bar{C}_{in} + \bar{a}\bar{b}\, C_{in} + a \cdot b\, C_{in}$$

$$C_{out} = a \cdot b + a \cdot C_{in} + b \cdot C_{in}$$

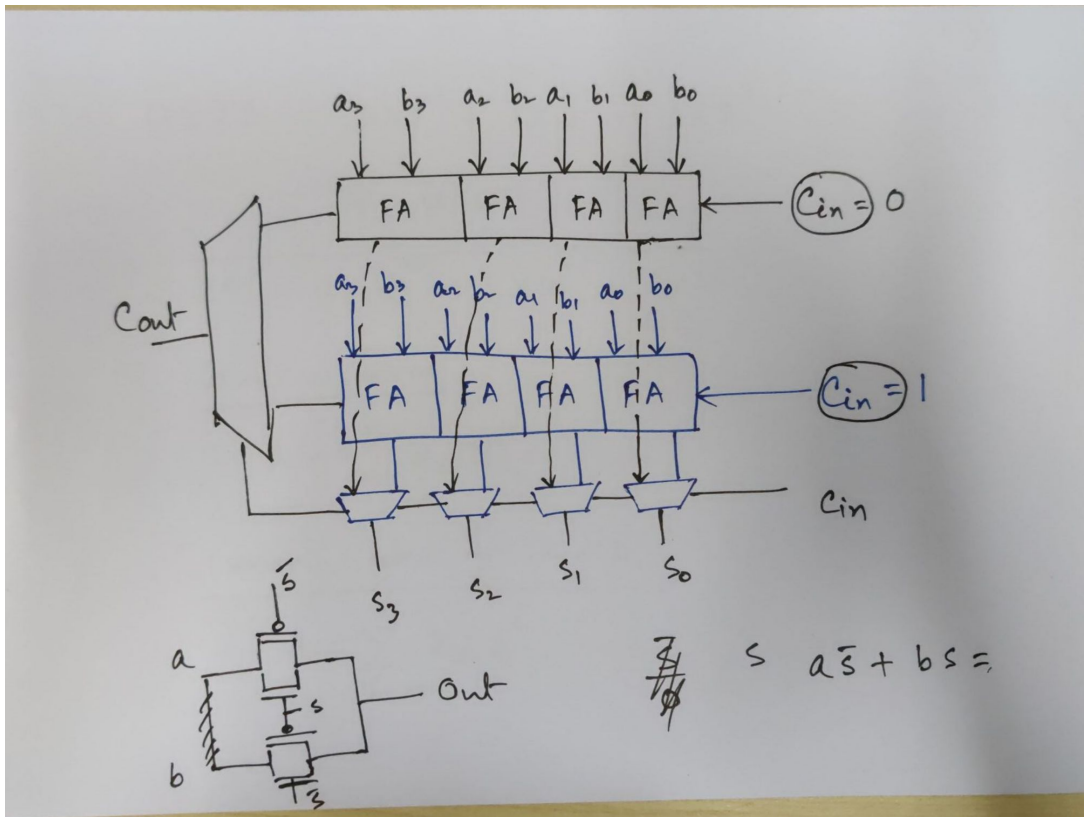


A : $a_3\ a_2\ a_1\ a_0$

B : $b_3\ b_2\ b_1\ b_0$



$$\boxed{T = t_h + (n-1)\,t_f}$$



$s \quad a\bar{s} + bs =$

$$0.375 = 0 \times 2^{-1} + 1 \times 2^{-2} + 1 \times 2^{-3}$$

$$\Rightarrow 0.011$$

$$3.29 = 3 \times 10^{0} + 2 \times 10^{-1} + 9 \times 10^{-2}$$

<u>Generic form</u>

$$A = \sum_{i=-n}^{n} x_i \, 10^{i}$$

<u>Generic form for base 2</u>

$$A = \sum_{i=-n}^{n} x_i \, 2^{i}$$

Floating point Numbers.

Normalized Form

$$A = (-1)^s \times P \times 2^x \quad \left(P = 1 + M,\ 0 \leq M < 1,\ x \in Z\right)$$

significand betn 1 & 2

M → Mantissa

X → exponent

Z → set of integers.

IEEE 754 format

1	8	23
Sign	exponent	Mantissa

$1.\underbrace{xxxx}_{M}$

1	8	23
Sign bit	exponent	Mantissa

bias $\rightarrow$ 127

$$E = x + 127$$

(0 − 255)

−127 to +128

<u>Normalized numbers.</u>

$x \rightarrow -126$ to $+127$

$$A = (-1)^S \times P \times 2^{E - bias}$$

$$-3 = (-1)^1 \times (1 + .5) \Rightarrow \boxed{(-1)^1 \times (1 + 2^{-1}) \times 2^1}$$

1	10000000	10. 0
1	E	

M

2^{-1} 2^{-i} 2^{-23}

$$M = \sum_{i=1}^{23} x_i \cdot 2^{-i}$$

1011001

$\underbrace{1.011001}_{\text{significand}} \times 2^{6}$

0.0001101

1.1101×2^{-4}

$A = P \times 2^{x}$, $(P = 1 + M,\ 0 \leq M < 1,\ x \in \mathbb{Z})$

Mantissa

$$\boxed{A = (-1)^{s} \times P \times 2^{x}}$$

IEEE 754 format.

Sign(s)	Exponent (X)	Mantissa (M)
1	8	23

$E \rightarrow 0$ & 255 are reserved.

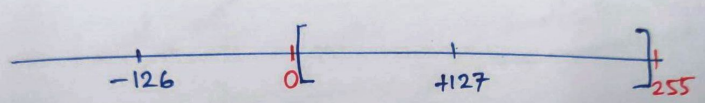
$E \rightarrow 1$ & 254

$E = x + \text{bias}$.

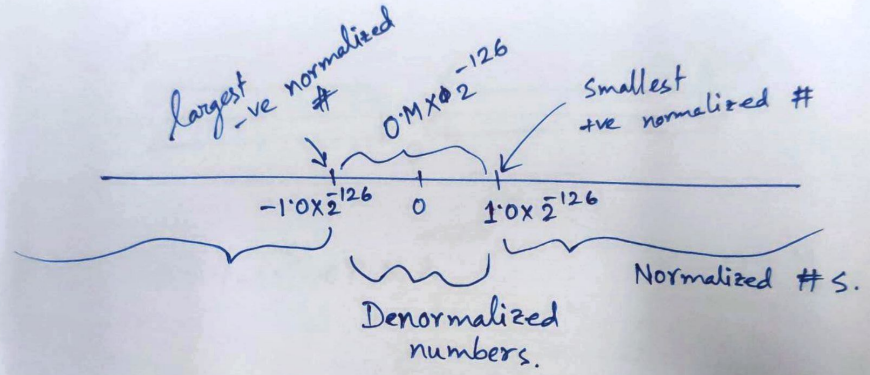
$$A = (-1)^S \times P \times 2^{E-\text{bias}}, \quad (P = 1 + M,\ 0 \leq M < 1,\ 1 \leq E \leq 254)$$

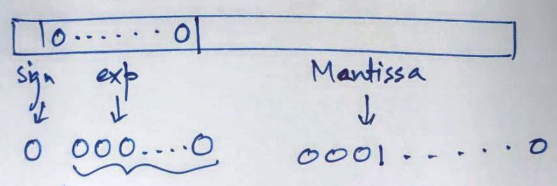
127

E	M	Value
255	0	∞ if $S=0$
255	0	$-\infty$ if $S=1$
255	$\neq 0$	NAN (not a number)
0	0	0
0	$\neq 0$	Denormal number.



For normalized values, you will never find exponent field of all 0's or all 1's.



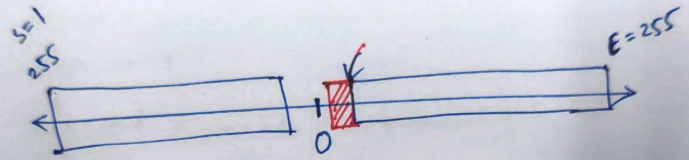


Mantissa $= 0001 \ldots\ldots$

$= 0 + 2^{-4}$

Answer $= 2^{-4} \times 2^{-126}$

$= 2^{-130}$



$$A = (-1)^S \times P \times 2^{-126}$$

$$P = 0 + \underline{M}$$

$$\boxed{P = \underline{0 + M}, \; 0 \leq M < 1}$$

$$2^{-126} \div 2$$

2^{-146}

sign → 0

exponent → −126 (denormal)

0 00000000 00000000000.... 2^{-20}....

Double Precision

S → 1

E → 11

M → 52

Assembly Language } family of low level programming languages.

- Low level programming language
- Specific for an ISA

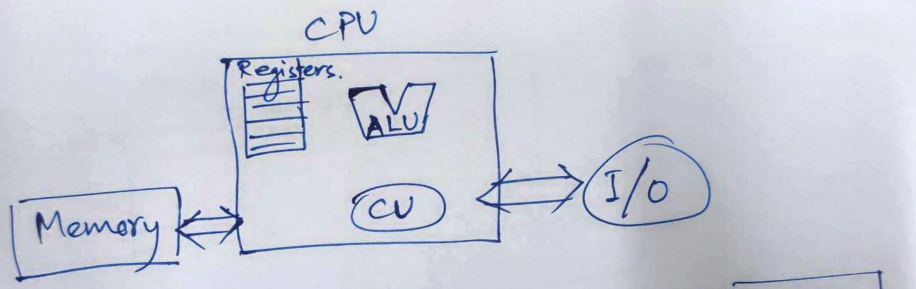
Why learn Assembly language?

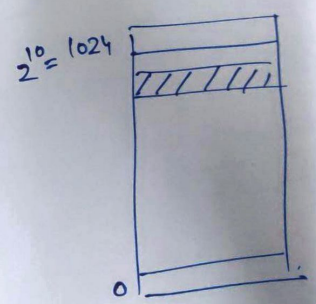
- Highly efficient code
- IOT platforms.

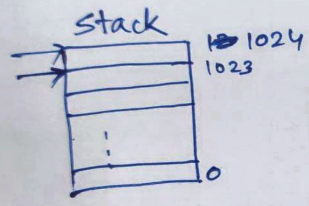
Two parts:

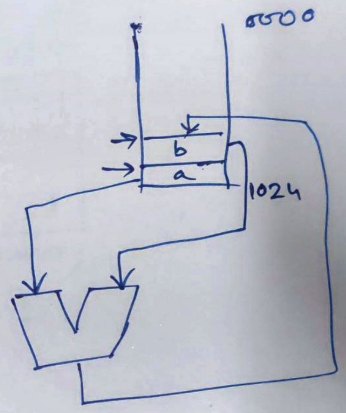
1. Instruction code/opcode
2. a list of operands.

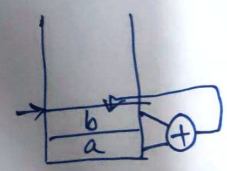
* Where do operands come from & where do results go?









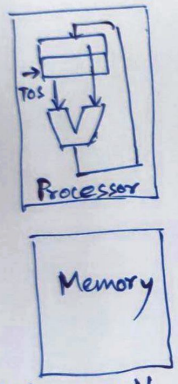


C = A+B.

Push A
Push B.
Add.
~~Pop C.~~
Pop C.

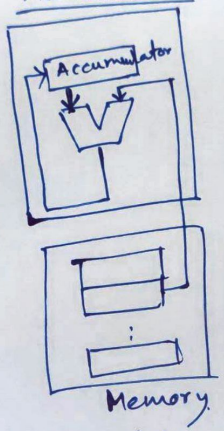
Machine Models.

Stack



# of explicitly named operand '0'.

Accumulator

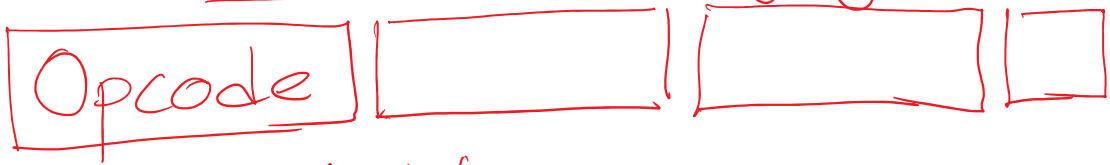


'1'

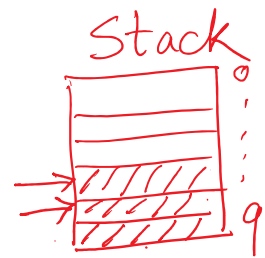
Let 12

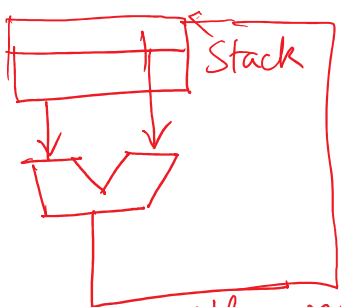
Thursday, June 2, 2022 12:00 PM

Assembly Language



Machine Models

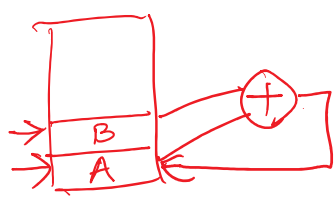




# of explicitly named operand '0'.

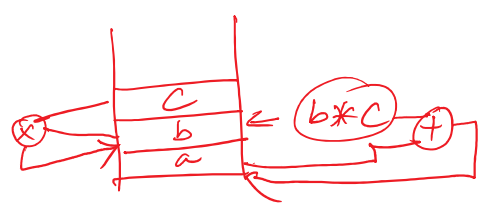
C = A+B

Push A
Push B
Add
Pop C



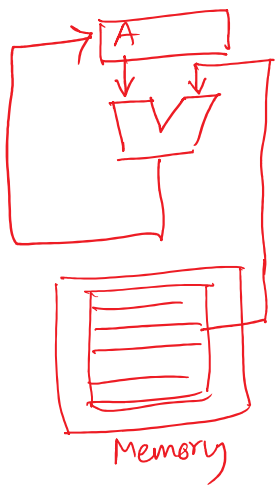
a = b * c

push a



push b
push c
Mul
Add
Pop

Accumulator

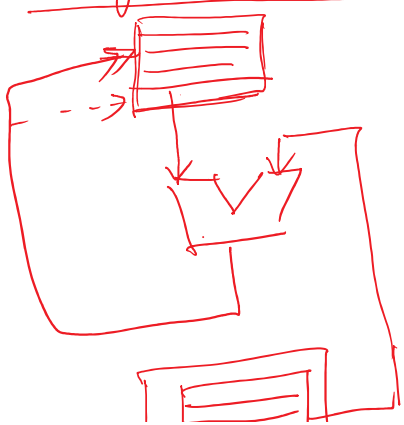


C = A + B

Load A
ADD B
Store C

# of explicitly named operand 1.

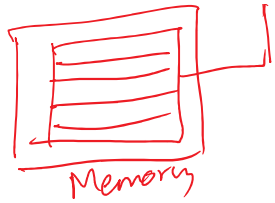
Register - Memory



C = A + B

Load R1, A
ADD R3, R1, B
Store R3, C

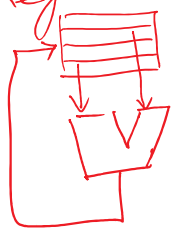
ADD R1, B



# of explicitly named operand 3 or 2

in x86 ISA, 1st operand is always destination as well.

Register-Register



Load R1, A
load R2, B
ADD R3, R1, R2

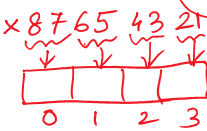
ADD R1, R2 # 2 or 3

Registers: named storage locations.
General Purpose Registers
ARM: r0 .... r15

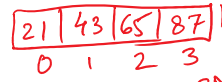
Stack Pointer
Program Counter
Return address
} Special registers.

Power PC, Sparc

0x8765 4321



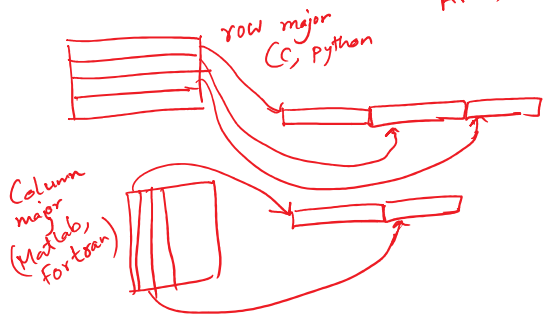
Big Endian



Little Endian
ARM, x86

row major ...than

ARM, "



Opcode Operand1, ...

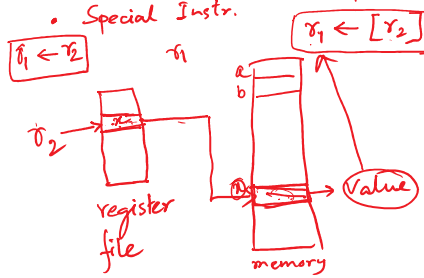
Operand:
- Constant (immediate)
- Register
- Memory

Types of Instructions
- Data Processing Instn:
  add, sub, mul, div, com, logical operations
- Data transfer Instn
  Load, store
- Branch Instr.
- Special Instr.

$r_1 \leftarrow r_2$

$r_1 \leftarrow r_2 + 4$

$r_1 \leftarrow [r_2]$



Lec 13

Tuesday, June 7, 2022 12:00 PM

Addressing Modes

* Immediate addressing mode

  A ← 0X13

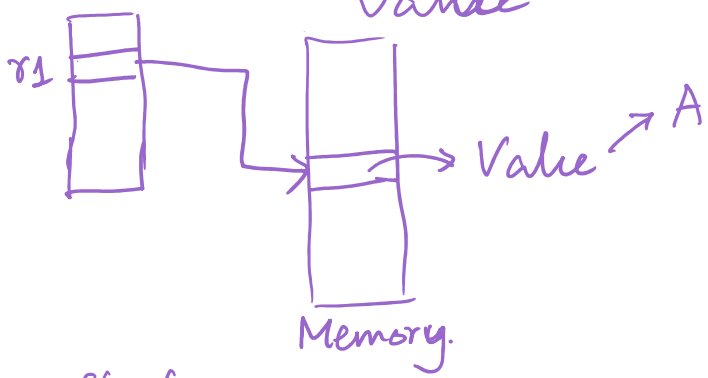
* Register direct " "

  A ← r1

* Register indirect " "

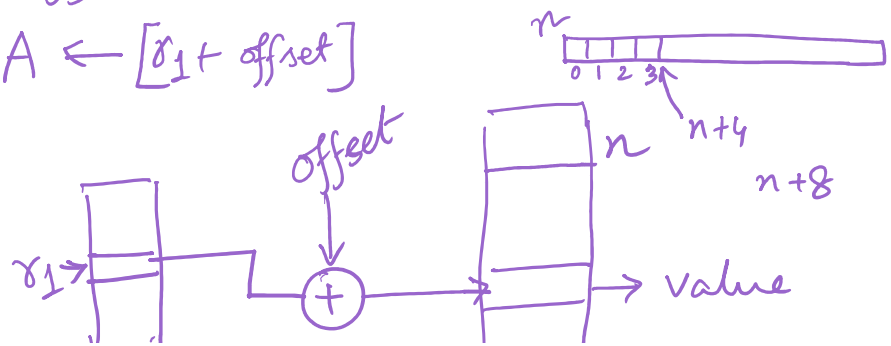
  A ← [r1]

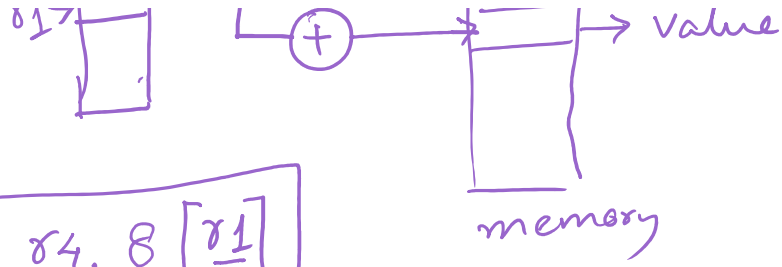
the register saves the address of the memory location that contain the value



Base-offset

A ← [r1 + offset]





st r4, 8 [r1]

SIMPLE RISC

16 register r0 ... r15

r14 ← stack pointer (sp)

r15 ← return address register (ra)

flags.E = 1 (equality), flags.GT = 1 (greater than)

mov

mov r_1, r_2 $r_1 \leftarrow r_2$

mov r1, 3 $r_1 \leftarrow 3$

16 bit immediates.

-2^{15} to $2^{15}-1$

Arithmatic / Logic Instⁿ

Arithmatic / Logic Insu

add, sub, mul, div, mod, cmp

add r_1, r_2, r_3 $\quad r_1 \leftarrow r_2 + r_3$

add $r_1, r_2, 10$

sub r_1, r_2, r_3 $\quad r_1 \leftarrow r_2 - r_3$

mul r_1, r_2, r_3 $\quad r_1 \leftarrow r_2 \times r_3$

div r_1, r_2, r_3 $\quad r_1 \leftarrow r_2 / r_3$ (quotient)

mod r_1, r_2, r_3 $\quad r_1 \leftarrow r_2 \bmod r_3$ (remainder)

cmp r_1, r_2 $\quad$ set flag

$a = 3$ $\qquad a \leftrightarrow r_0$

$b = 5$ $\qquad b = r_1$

$c = a + b$ $\qquad c = r_2$

$d = c - 5$ $\qquad d = r_3$

mov r0, 3

mov r_1, 5

add r_2, r_0, r_1

sub $r_3, r_2, 5$

$a = 3$, $b = 5$ Compare 3 & 5

mov r_0, 3 flags.E = 0

mov r_1, 5 flags.GT = 0

Cmp r_0, r_1

Compute $\boxed{31/29 - 50}$

mov r_1, 31

mov r_2, 29

div r_3, r_1, r_2

sub r_4, r_3, 50

Branch Inst^n

Unconditional Branch

b .foo $\longrightarrow$ branch to .foo

add r_1, r_2, r_3

```
b .foo
  .
  .
  .
.foo: sub r3, r2, r1
```

Conditional Branch Instⁿ

beq .foo → branch to .foo if flags.E = 1

bgt .foo → " " " if flags.GT = 1

* The flags are only set by the cmp instⁿ.

Lec 14

Thursday, June 9, 2022 5:47 AM

Real World Instruction Sets

Arch	Type	# Oper	# Mem	Data Size	# Regs	Addr 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	Mainframe
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	Accum.	1	1/2	8-bit	0	16-bit	Microcontroler

Logical Instn

and r_1, r_2, r_3

$r_2 \rightarrow 1010$

$r_3 \rightarrow 0110$

$r_1 \rightarrow 0010$

$(r_2 \cap r_3)$

Shift Instn

Logical Shift left (lsl)

$<<$

$0010 << 2 \rightarrow 1000$

$<< n$ is same as multiplying by 2^n

Arithmetic Shift right (asr)

$>>$ operator.

$0010 >> 1 \rightarrow 0001$

$0010 >> 2 \rightarrow 0000$

$1000 >> 2 \quad 1110$

same as diving a signed number by 2^n.

Logical shift right (lsr)

($>>>$ operator)

$1000 >>> 2 \rightarrow 0010$

Same as diving an unsigned number 2^n.

Compute $102 * 7.5$

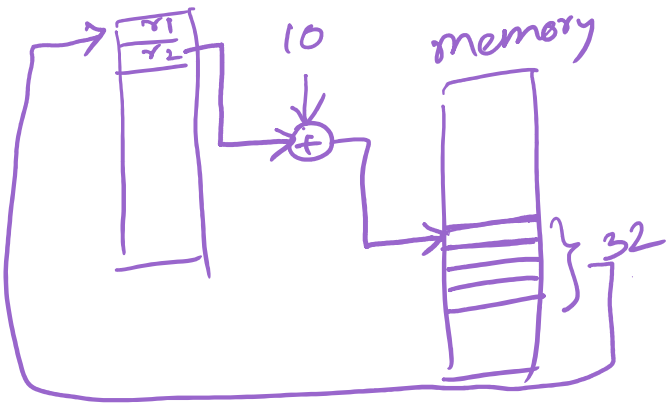
$102(8 - 0.5)$

```
mov r0, 102
lsl r1, r0, 3
lsr r2, r0, 1
```

sub r_3, r_1, r_2

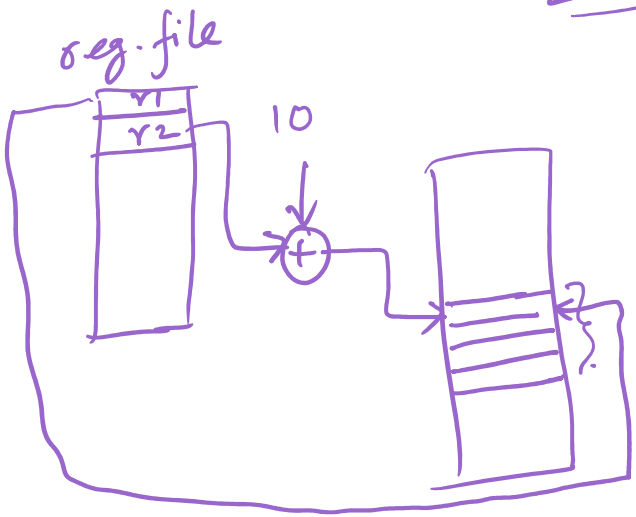
Load - store instn

ld $r1, 10[r2]$ $\qquad r1 \leftarrow [r2+10]$



St $r1, 10[r2]$

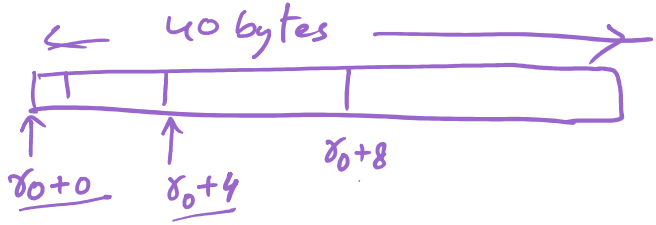
$[r2+10] \leftarrow r1$



Example $\longleftarrow$ 40 bytes $\longrightarrow$

Example

```
int arr[10]
```




```
arr[3] = 5;
arr[4] = 8;
arr[5] = arr[4] + arr[3];
```

assume base of array is saved in r0.

```
mov r1, 5
st r1, 12[r0]
mov r2, 8
st r2, 16[r0]
add r3, r1, r2
st r3, 20[r0].
```

Factorial (num)

```
int prod = 1;
int idx;
for (idx = num; idx > 1; idx--){
    prod = prod * idx
}.
```

Assembly code

```
mov r1, 1          /* prod = 1 */
mov r2, r0         /* idx = num */
.loop:
    mul r1, r1, r2 /* prod = prod * idx */
```

```
mul r1, r1, r2     /* prod = prod * idx */
sub r2, r2, 1      /* idx = idx - 1 */
cmp r2, 1          /* Compare(idx, 1)*/
bgt .loop          /* if (idx>1) goto .loop
                   */
```

<u>Modifiers</u> (for instructions that has an immediate operand)

* default : mov → treat the 16 bit immediate as a signed number (automatic sign extension)

* (u) : movu → treat the 16 bit immediate as unsigned number.

* (h) : movh → left shift the 16 bit immediate by 16 positions.

```
mov r1, 0XAB12
```

sign	bits.	AB	12

```
movu r1, 0XAB12
```

$r_0 \leftarrow$ 0X12AB A92D

```
movh r0, 0X12AB
```

movu r1, 0XAB12

00	00	AB	12

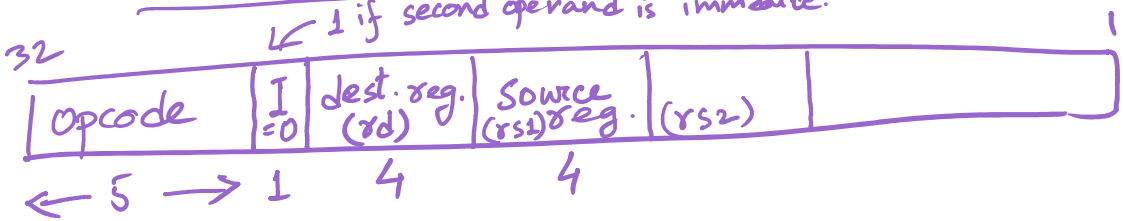
movh r1, 0XAB12

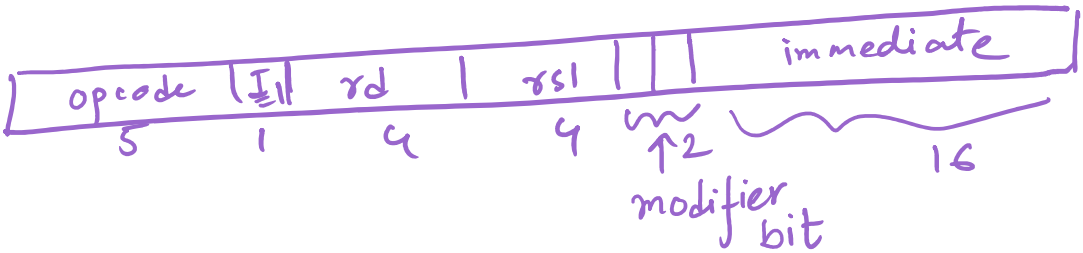
AB	12	00	00

addu, r0, 0XA92D

Instruction format

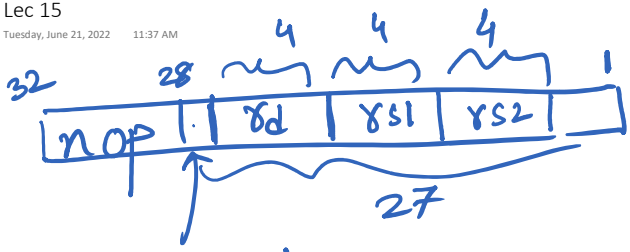
1 if second operand is immedite.





Lec 15

Tuesday, June 21, 2022 11:37 AM



add rd, rs1, (rs2/imm)

Imm. bit

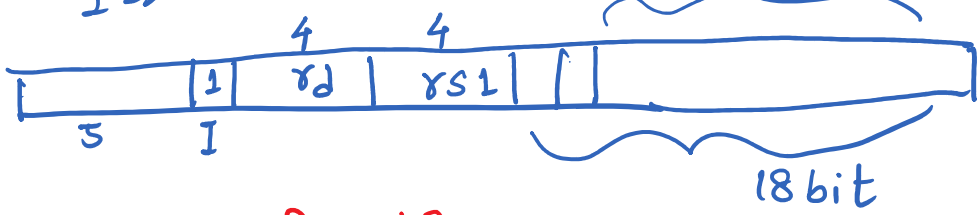
0 → reg.

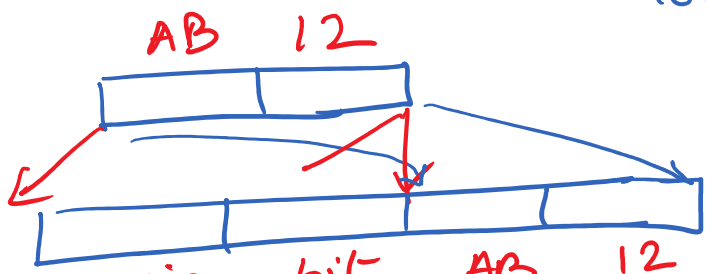
1 → Imm.

00 ← default

01 ← u

10 ← h





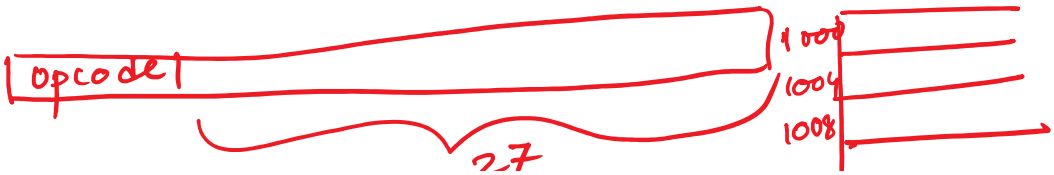
sign bit AB 12 ← default.

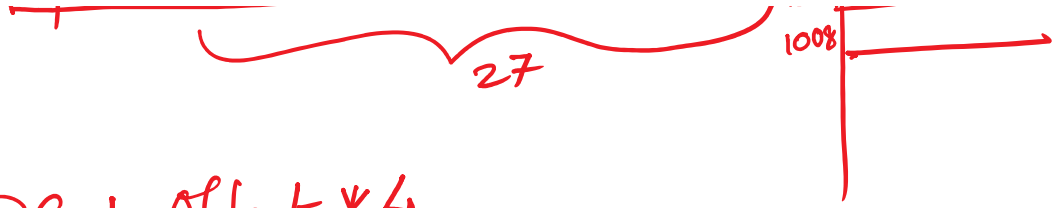
0 0 00 AB 12 ← u

A B 12 0 0 00 ← h

---

b, beq, bgt, call





PC + Offset * 4

St [dest] [Source].

St Imm.[rs1] , rs2

St. [r_d] , Imm [rs1]

r1 → ABCD 0000

ABCD1234

ABCD1234

movh r1, ABCD

add r1, r1, 1234

CSE 112: Computer Organization (Section A)

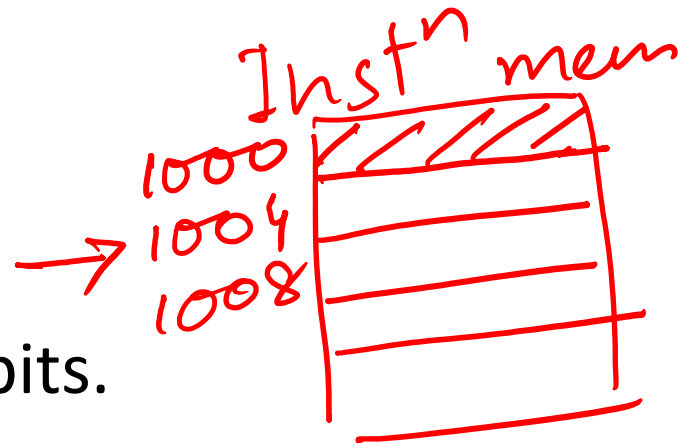
Instructor: Sujay Deb

Lecture 15

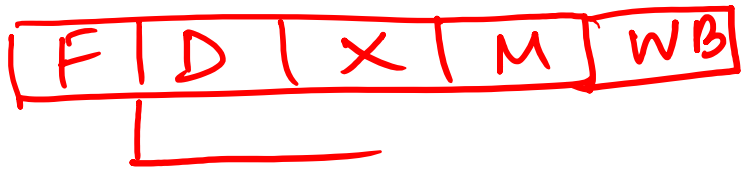
INDRAPRASTHA INSTITUTE *of*
INFORMATION TECHNOLOGY
DELHI

AMS Lab @IIITD

Encoding Instructions



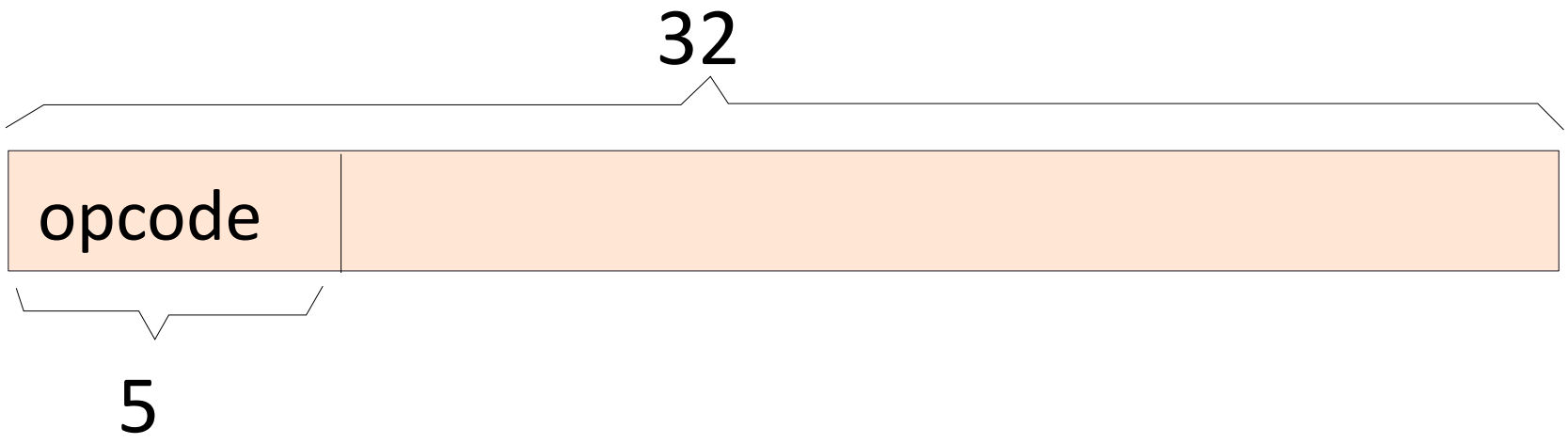
- Encode the SimpleRisc ISA using 32 bits.
- We have 21 instructions. Let us allot each instruction an unique code (opcode)



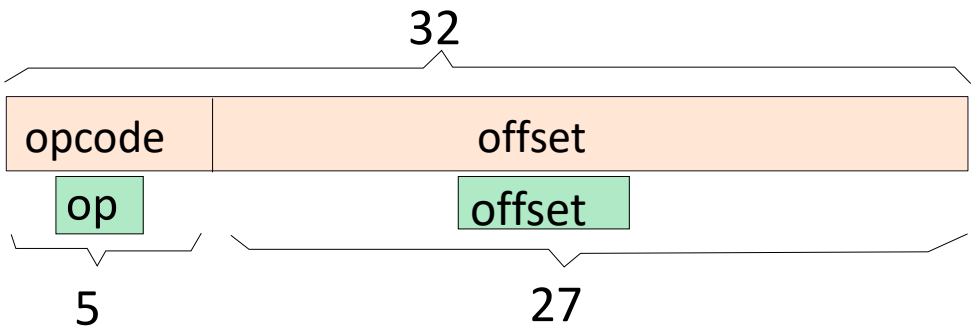
Instruction	Code	Instruction	Code	Instruction	Code
add	00000	not	01000	beq	10000
sub	00001	mov	01001	bgt	10001
mul	00010	lsl	01010	b	10010
div	00011	lsr	01011	call	10011
mod	00100	asr	01100	ret	10100
cmp	00101	nop	01101		
and	00110	ld	01110		
or	00111	st	01111		

0-Address Instructions

- nop and ret instructions



1-Address Instructions

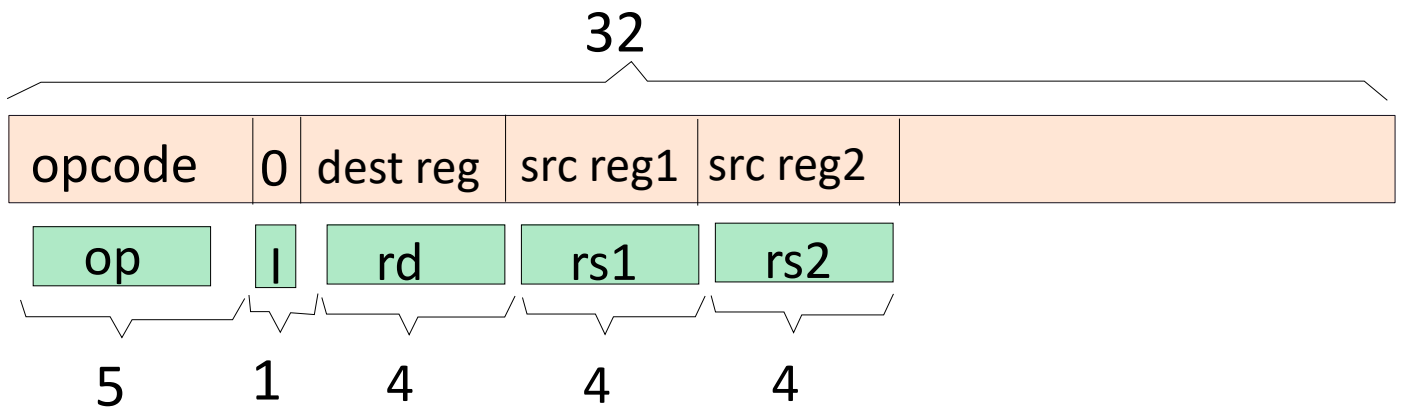


- Instructions – call, b, beq, bgt
- Use the branch format
- Fields :
  - 5 bit opcode
  - 27 bit offset (PC relative addressing)
  - Since the offset points to a 4 byte word address
    - The actual address computed is : PC + offset * 4

3-Address Instructions

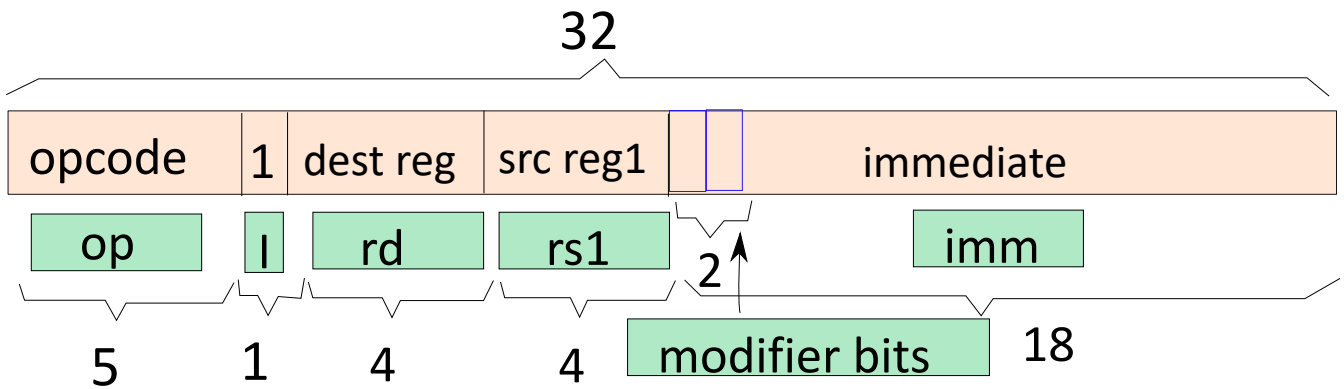
- Instructions – add, sub, mul, div, mod, and, or, lsl, lsr, asr
- Generic 3 address instruction
  - <opcode> rd, rs1, <rs2/imm>
- Let us use the I bit to specify if the second operand is an immediate or a register.
  - I = 0 → second operand is a register
  - I = 1 → second operand is an immediate
- Since we have 16 registers, we need 4 bits to specify a register

Register Format



- opcode → type of the instruction
- I bit → 0 (second operand is a register)
- dest reg → rd
- source register 1 → rs1
- source register 2 → rs2

Immediate Format

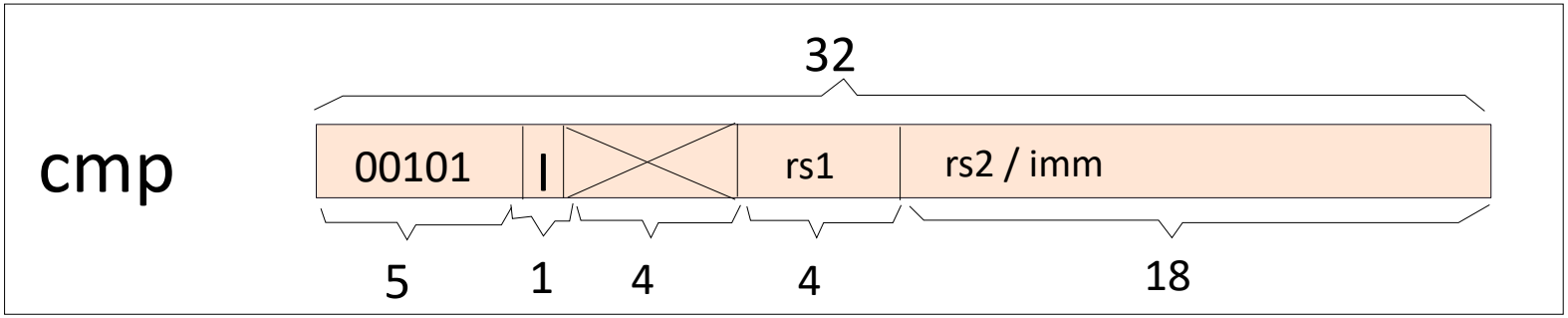


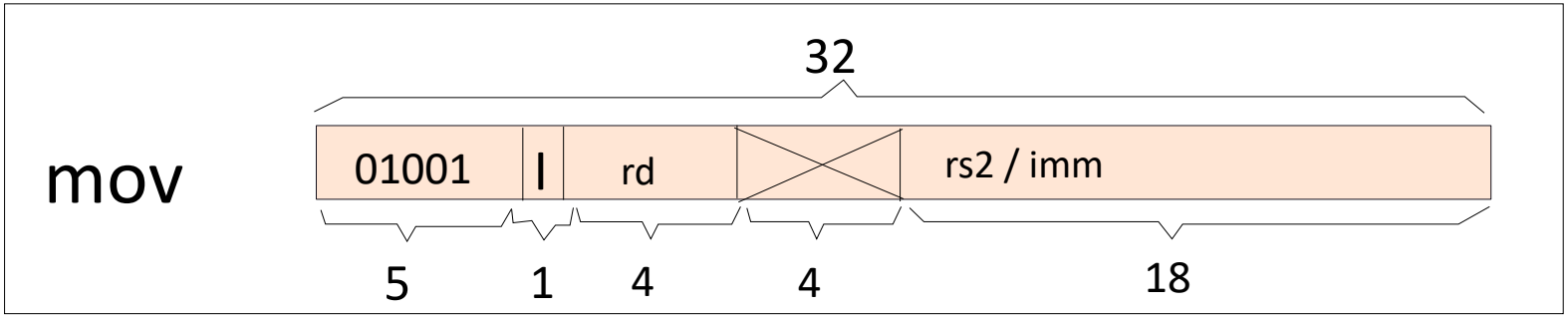
- opcode → type of the instruction
- I bit → 1 (second operand is an immediate)
- dest reg → rd
- source register 1 → rs1
- Immediate → imm
- modifier bits → 00 (default), 01 (u), 10 (h)

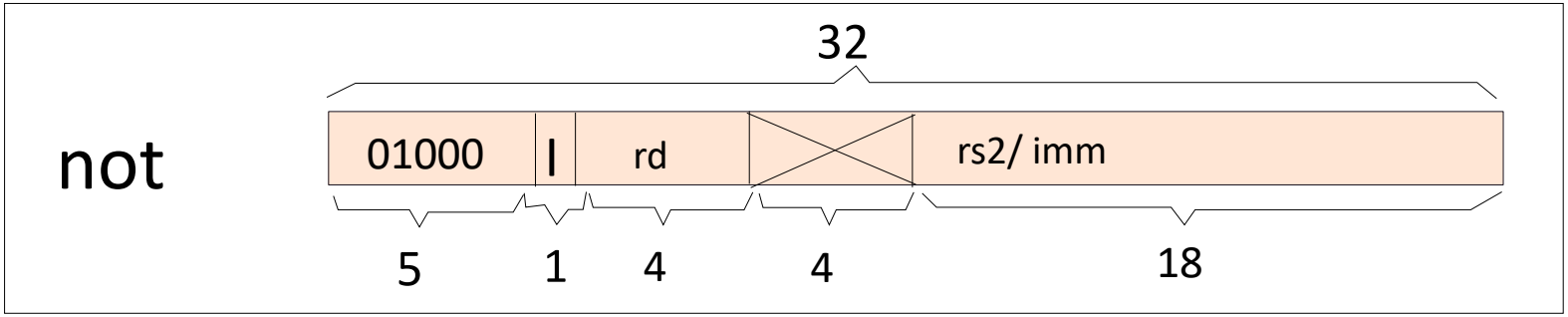
2 Address Instructions

- cmp, not, and mov
- Use the 3 address : immediate or register formats
- Do not use one of the fields

cmp, not, and mov

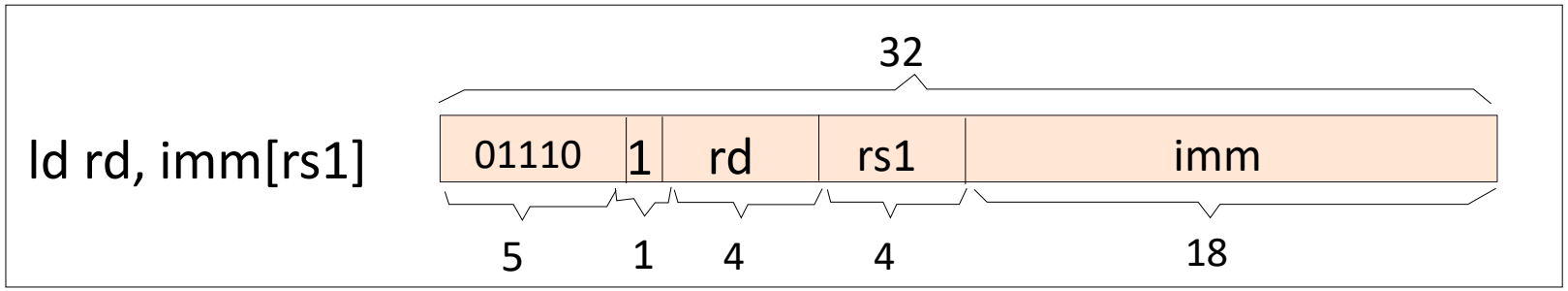






Load and Store Instructions

- ld rd, imm[rs1]
- rs1 → base register
- Use the immediate format.

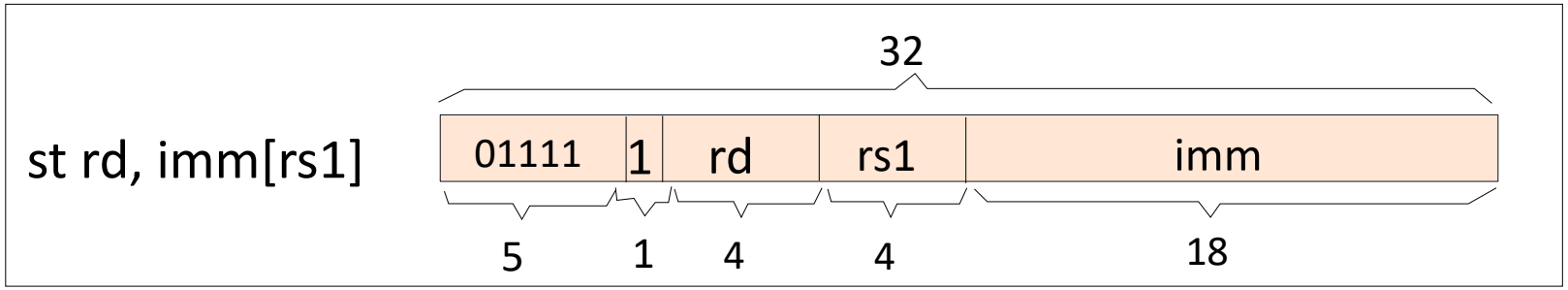


Store Instruction

- Strange case of the store inst.
- st reg1, imm[reg2]
- has two register sources, no register destination, 1 immediate
- Cannot fit in the immediate format, because the second operand can be either be a register OR an immediate (not both)
- Should we define a new format for store instructions ?

Store Instruction

- Let us make an exception and use the immediate format.
- We use the rd field to save one of the source registers
- st rd, imm[rs1]



Summary of Instruction Formats

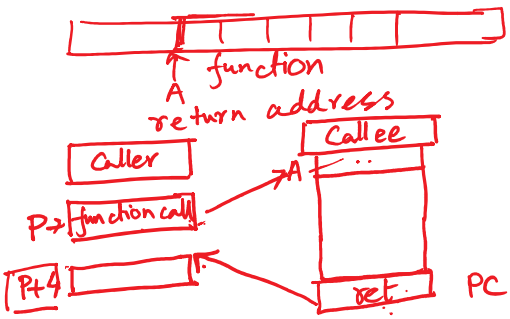
Format	Definition				
branch	*op* (28-32)	*offset* (1-27)			
register	*op* (28-32)	I (27)	*rd* (23-26)	*rs* 1(19-22)	*rs* 2(15-18)
immediate	*op* (28-32)	I (27)	*rd* (23-26)	*rs* 1(19-22)	*imm* (1-18)
op → opcode, *offset* → branch offset, *I* → immediate bit, *rd* → destination register					
*rs*1 → source register 1, *rs*2 → source register 2, *imm* → immediate operand					

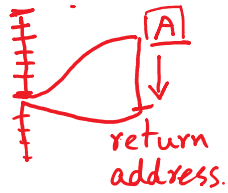
- branch format → nop, ret, call, b, beq, bgt
- register format → ALU instructions
- immediate format → ALU, ld/st instructions

Lec 16

Thursday, June 23, 2022 11:38 AM

Functions:





PC ← P + 4

.addition

2000 → add r2, r0, r1

2004 ret

.main:

return address

1012

1000 mov r0, 3

1004 mov r1, 5

1008 store r3, 100[r4]

10012 Call . addition

1016 mov . . .

* Space problem
* Overwrite " ← Spilling

Register Spilling

Caller saves the set of registers.

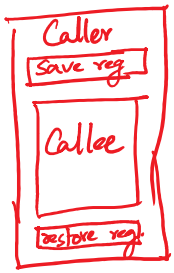
Calls function

Restores the set of registers after function returns.

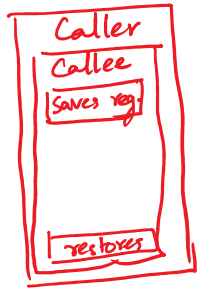
Callee saved scheme

Caller " "

Caller " "



Caller Saved



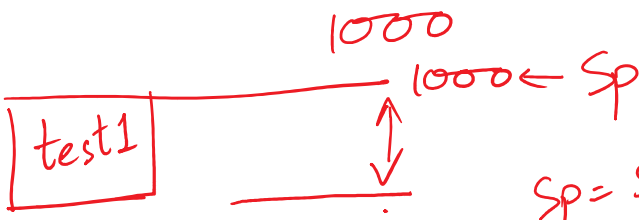
Callee saved.

* strict agreement betn caller & callee regarding the memory location & its use.

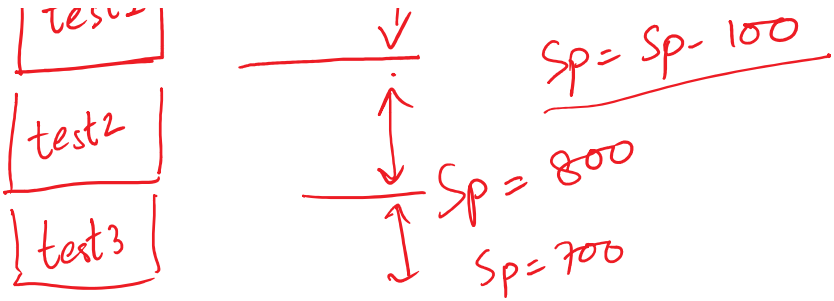
* All the space need to be reclaimed as soon as the function execution is done.

Activation Block :

memory block of a function.



Sp = Sp - 100



Through Stack

* Solve space problem
  - activation block for passing as many parameters.

* Overwrite problem
  - activation block

* management of activation block
  - by stack.

Call & return

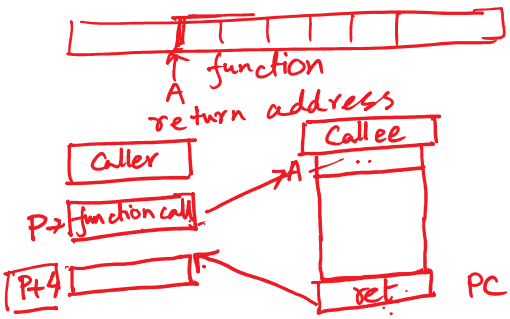
Call ra ← PC+4 ; PC ← address of function.

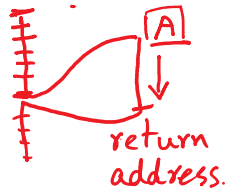
ret PC ← ra

Lec 16

Thursday, June 23, 2022 11:38 AM

Functions:





PC ← P + 4

.addition

2000 → add r2, r0, r1

2004 ret

.main:

return address

1012

1000 mov r0, 3

1004 mov r1, 5

1008 store r3, 100[r4]

10012 Call . addition

1016 mov . . .

* Space problem
* Overwrite " ← Spilling

Register Spilling

Caller saves the set of registers.

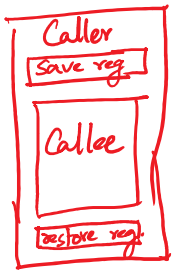
Calls function

Restores the set of registers after function returns.

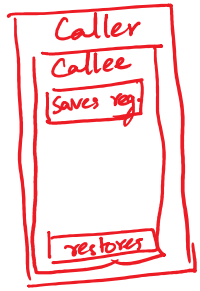
Callee saved scheme

Caller " "

Caller " "



Caller Saved



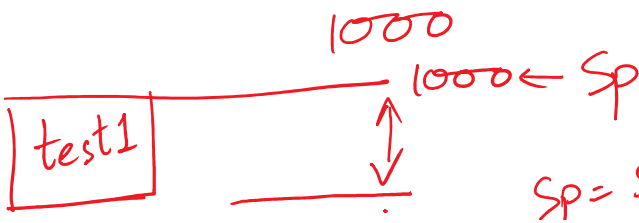
Callee saved.

* strict agreement betn caller & callee regarding the memory location & its use.

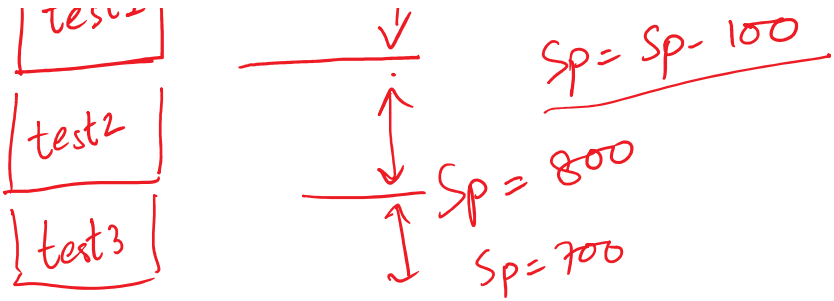
* All the space need to be reclaimed as soon as the function execution is done.

Activation Block :

memory block of a function.



Sp = Sp - 100



Through Stack

* Solve space problem
  - activation block for passing as many parameters.

* Overwrite problem
  - activation block

* management of activation block
  - by stack.

Call & return

Call ra ← PC+4 ; PC ← address of function.

ret PC ← ra

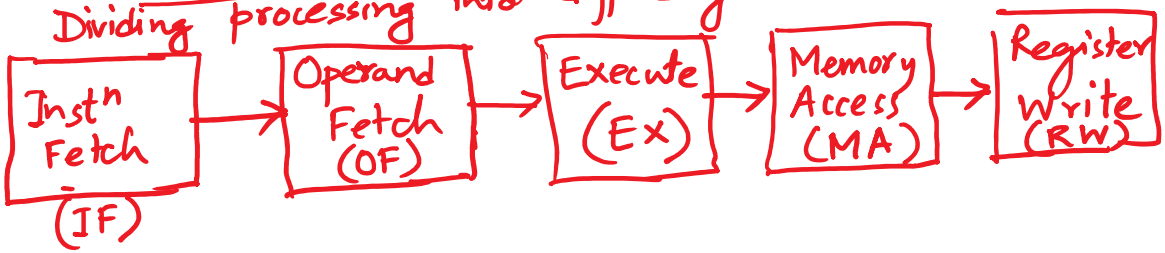
Lec 17

Tuesday, June 28, 2022 11:41 AM

CSE 112

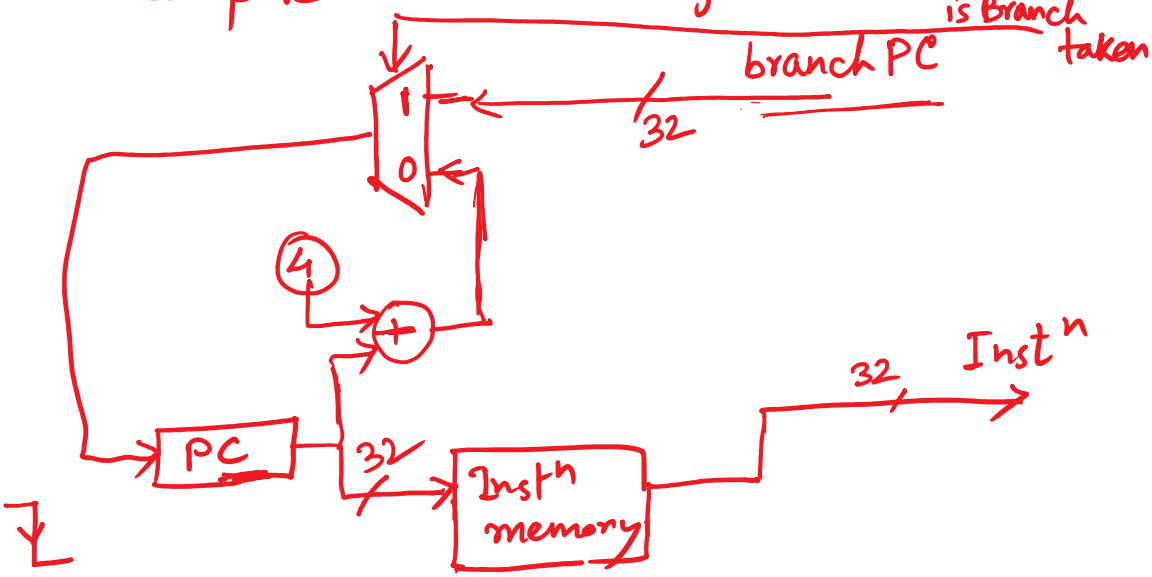
Processor Design

Dividing processing into diff^n stages.



IF

- Fetch an inst^n from Inst^n memory
- Compute the address of the next Inst^n.



OF

- Decode the Inst^n

- Decode the Inst^n
- Fetch the register operands from reg. files.
-

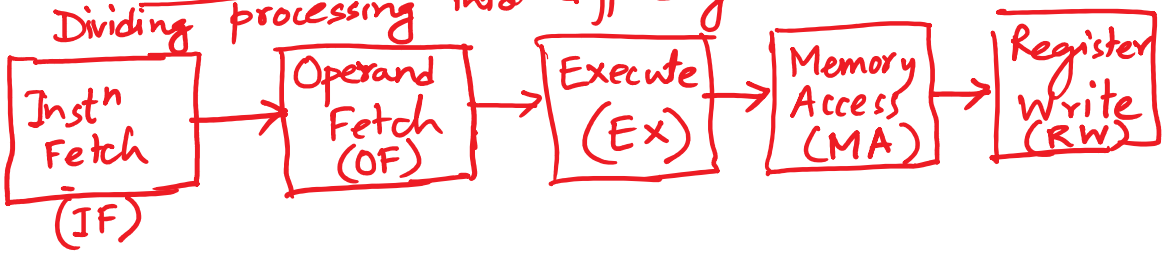
Lec 17

Tuesday, June 28, 2022 11:41 AM

CSE 112

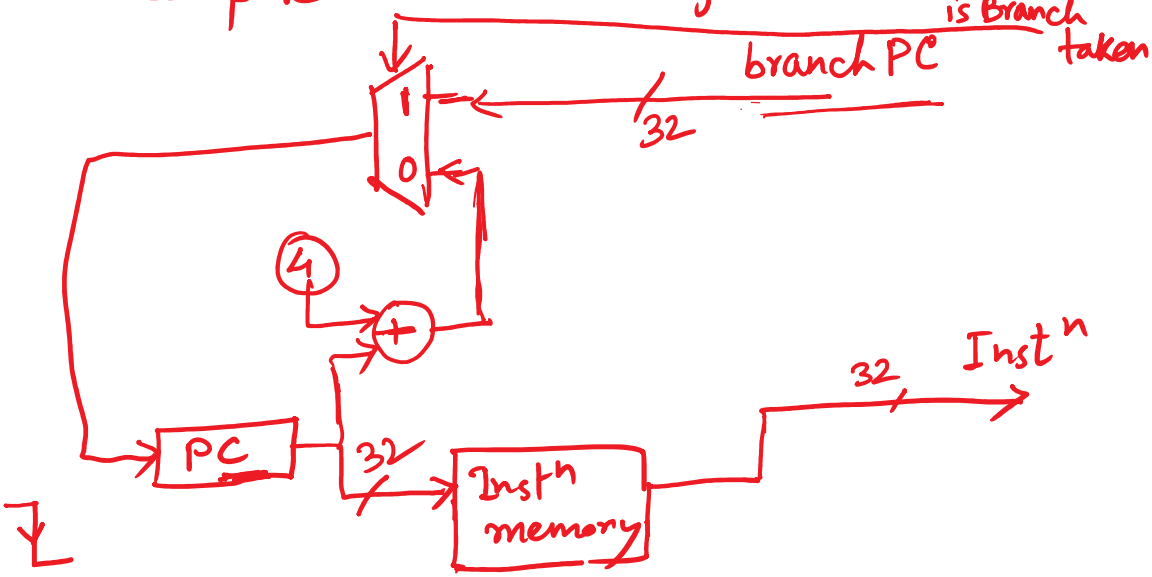
Processor Design

Dividing processing into diff^n stages.



IF

- Fetch an inst^n from Inst^n memory
- Compute the address of the next Inst^n.



OF

- Decode the Inst^n

- Decode the Inst^n
- Fetch the register operands from reg. files.
-

Lec 18

Thursday, June 30, 2022 11:39 AM

OF:

- Decode the Inst^n
- Fetch the register operands from RFs.
- Compute the branch target. (PC + offset)
- Compute the immediate (16 bit + 2 bit modifier) 32 bit number.
- Generate Control Signals.

Ex stage:

Contains ALU

- Perform all arithmatic & logical operation.
- Contains the branch unit for computing branch condition (beq, bgt)
- Contains Flag register (update by cmp inst^n)

Lec 18

Thursday, June 30, 2022 11:39 AM

OF:

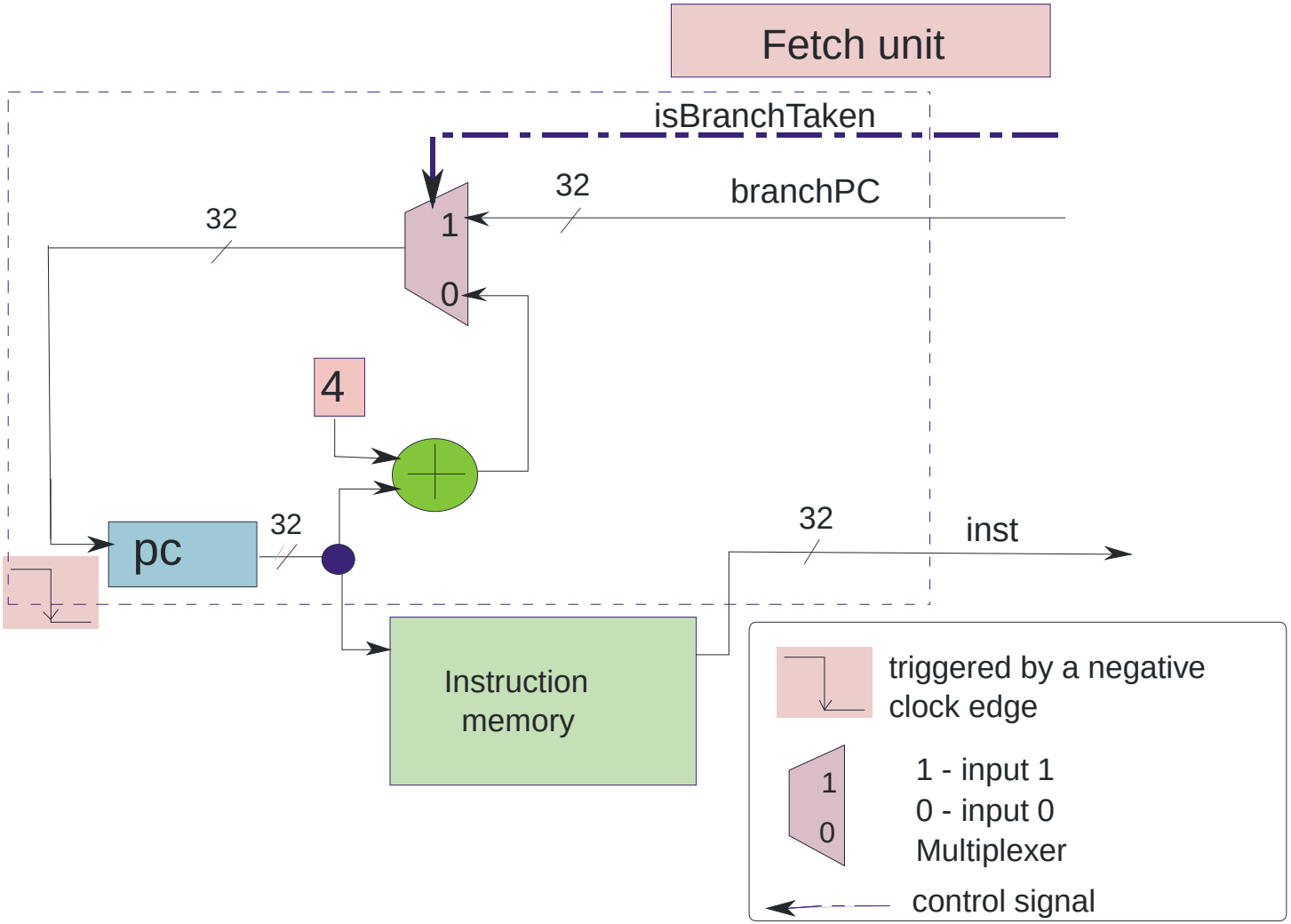
- Decode the Inst^n
- Fetch the register operands from RFs.
- Compute the branch target. (PC + offset)
- Compute the immediate (16 bit + 2 bit modifier) 32 bit number.
- Generate Control Signals.

Ex stage:

Contains ALU

- Perform all arithmatic & logical operation.
- Contains the branch unit for computing branch condition (beq, bgt)
- Contains Flag register (update by cmp inst^n)

Instruction Fetch (IF) Stage



isBranchTaken

- isBranchTaken is a control signal
  - It is generated by the EX unit
- Conditions on isBranchTaken

Instruction	Value of *isBranchTaken*
non-branch instruction	0
call	1
ret	1
b	1
beq	branch taken – 1 branch not taken – 0
bgt	branch taken – 1 branch not taken – 0

Operand Fetch Unit

Inst.	Code	Format	Inst.	Code	Format
add	00000	add rd, rs1, (rs2/imm)	lsl	01010	lsl rd, rs1, (rs2/imm)
sub	00001	sub rd, rs1, (rs2/imm)	lsr	01011	lsr rd, rs1, (rs2/imm)
mul	00010	mul rd, rs1, (rs2/imm)	asr	01100	asr rd, rs1, (rs2/imm)
div	00011	div rd, rs1, (rs2/imm)	nop	01101	nop
mod	00100	mod rd, rs1, (rs2/imm)	ld	01110	ld rd, imm[rs1]
cmp	00101	cmprs1, (rs2/imm)	st	01111	st rd, imm[rs1]
and	00110	and rd, rs1, (rs2/imm)	beq	10000	beq offset
or	00111	or rd, rs1, (rs2/imm)	bgt	10001	bgt offset
not	01000	not rd, (rs2/imm)	b	10010	b offset
mov	01001	mov rd, (rs2/imm)	call	10011	call offset
			ret	10100	ret

Instruction Formats

Format	Definition				
branch	*op* (28-32)	*offset* (1-27)			
register	*op* (28-32)	*I* (27)	*rd* (23-26)	*rs1* (19-22)	*rs2* (15-18)
immediate	*op* (28-32)	*I* (27)	*rd* (23-26)	*rs1* (19-22)	*imm* (1-18)
op → opcode, *offset* → branch offset, *I* → immediate bit, *rd* → destination register					
*rs*1 → source register 1, *rs*2 → source register 2, *imm* → immediate operand					

* Each format needs to be handled separately.

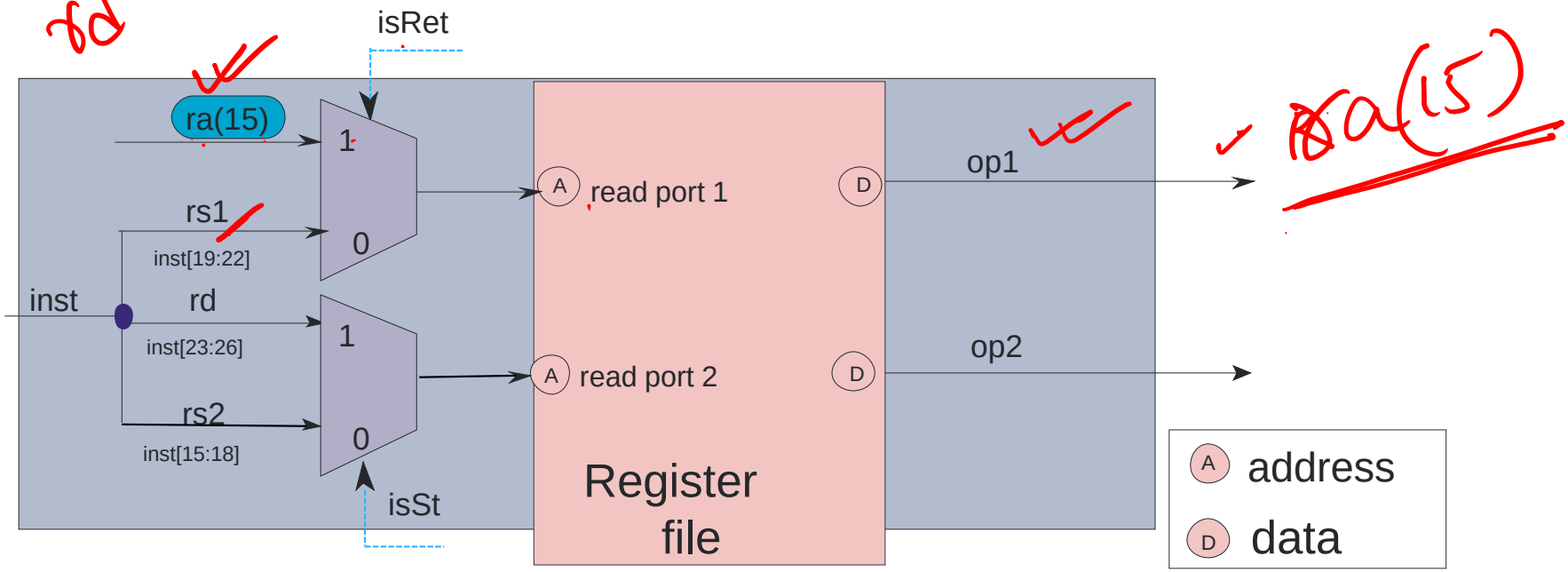
St r1, 20[r2]

st r1, 20[r2]

rd

rs1

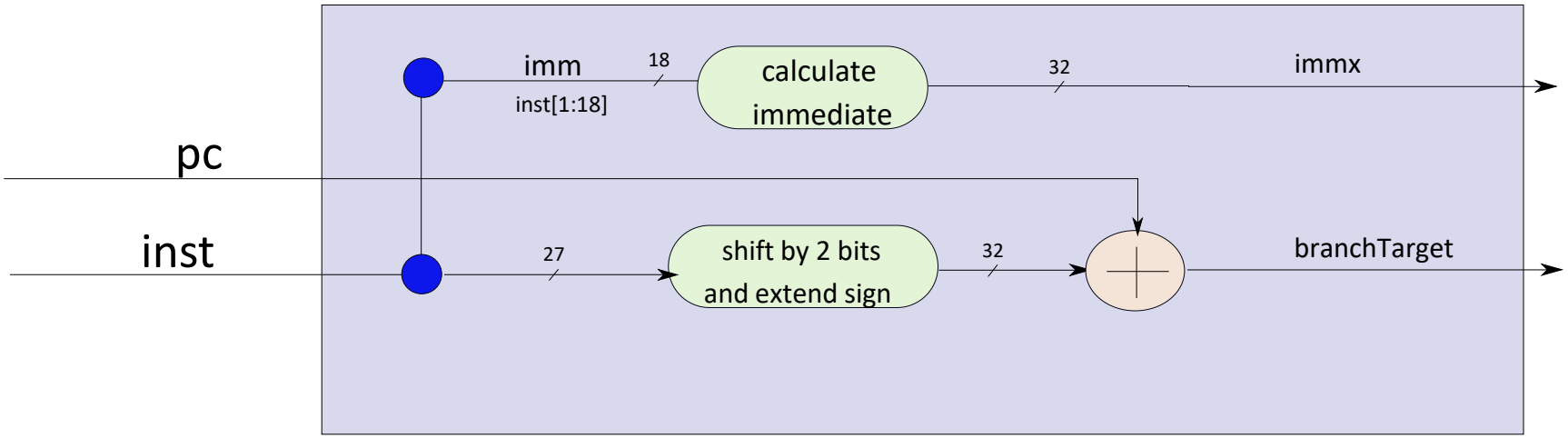
Register File Read



ra(15)

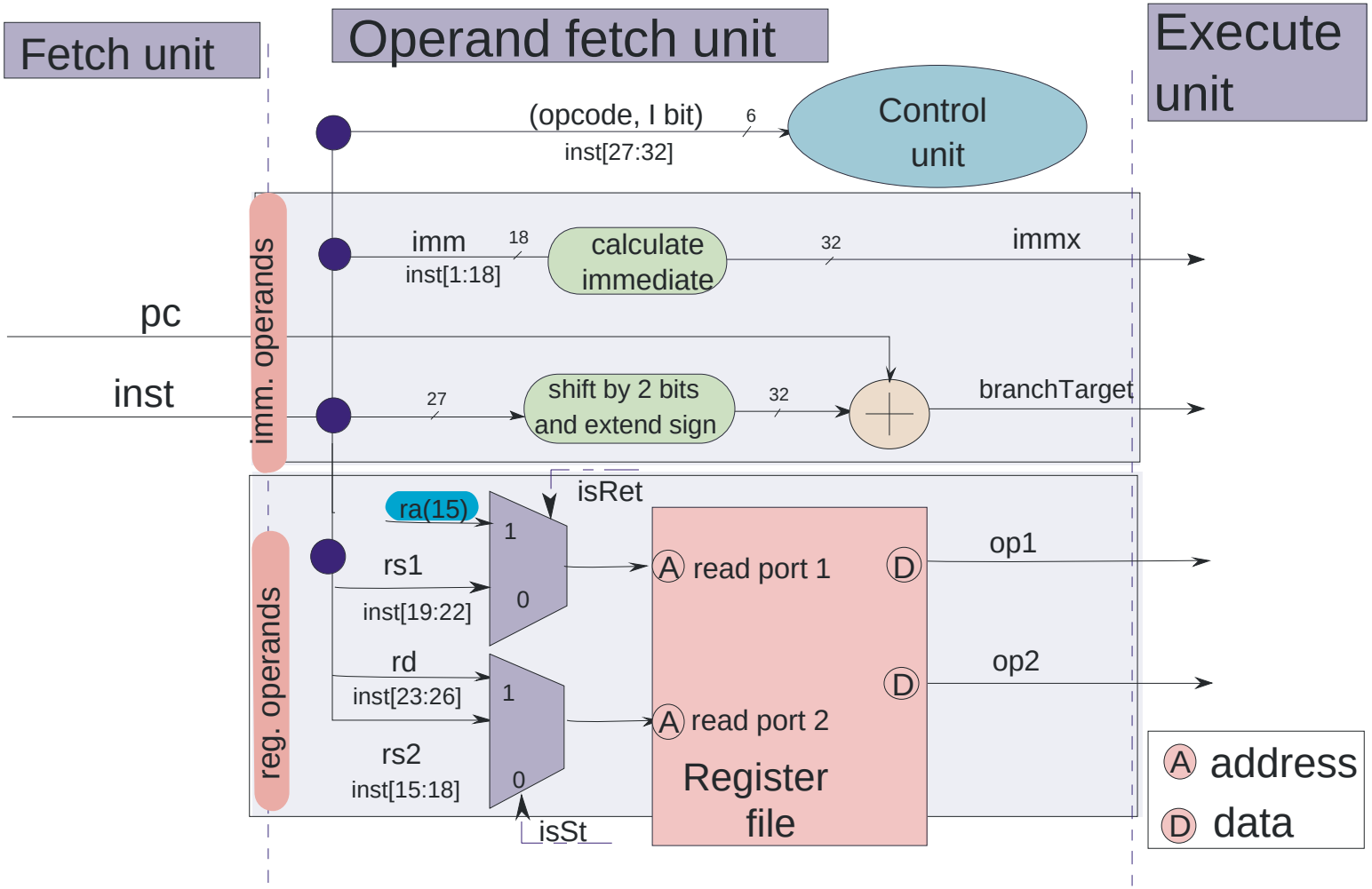
- First input → rs1 or ra(15) (ret instruction)
- Second input → rs2 or rd (store instruction)

Immediate and Branch Unit

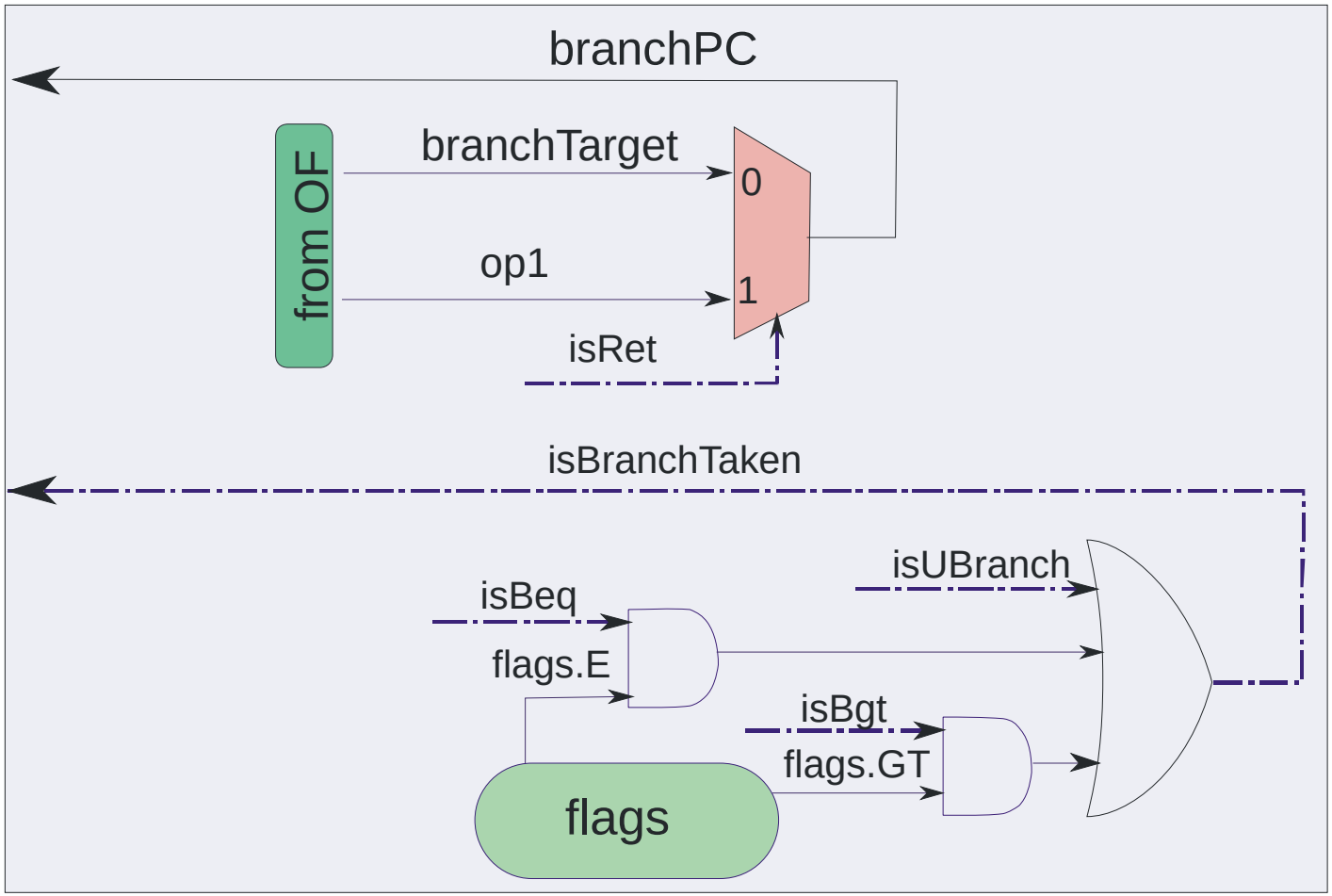


- Compute immx (extended immediate), branchTarget, irrespective of the instruction format.
- For the branchTarget we need to choose between the <u>embedded target</u> and <u>op1</u> (ret)

OF Unit

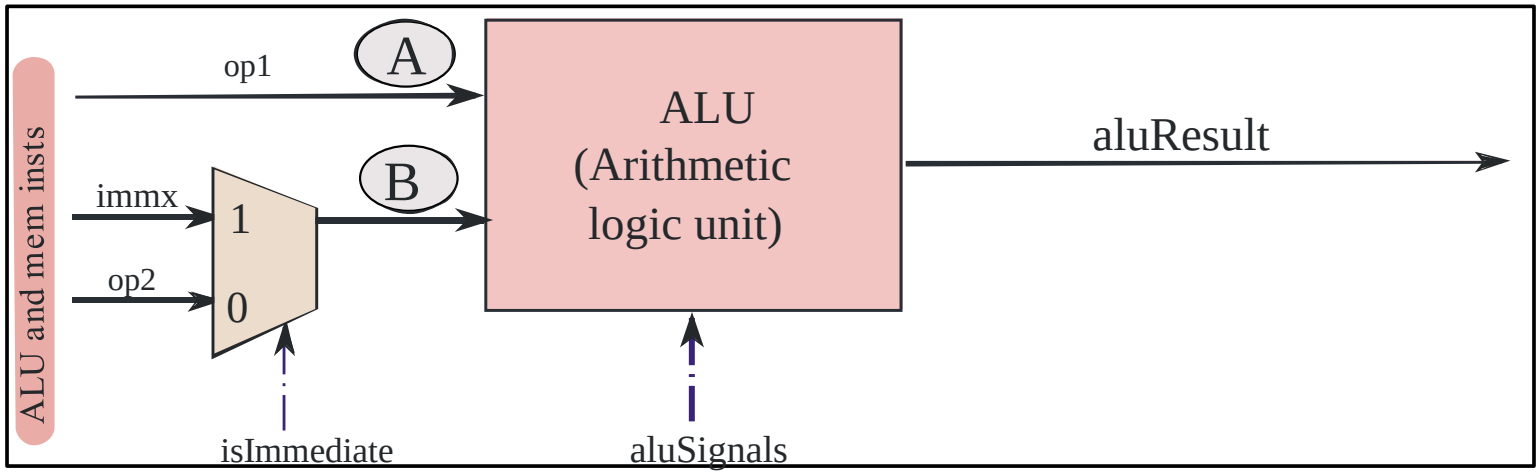

Fetch unit
Operand fetch unit
Execute unit
(opcode, I bit)
inst[27:32]
6
Control unit
imm
inst[1:18]
18
calculate immediate
32
immx
pc
inst
imm. operands
27
shift by 2 bits and extend sign
32
branchTarget
isRet
ra(15)
rs1
inst[19:22]
rd
inst[23:26]
rs2
inst[15:18]
1
0
1
0
isSt
reg. operands
read port 1
read port 2
Register file
op1
op2
address
data

EX Stage – Branch Unit



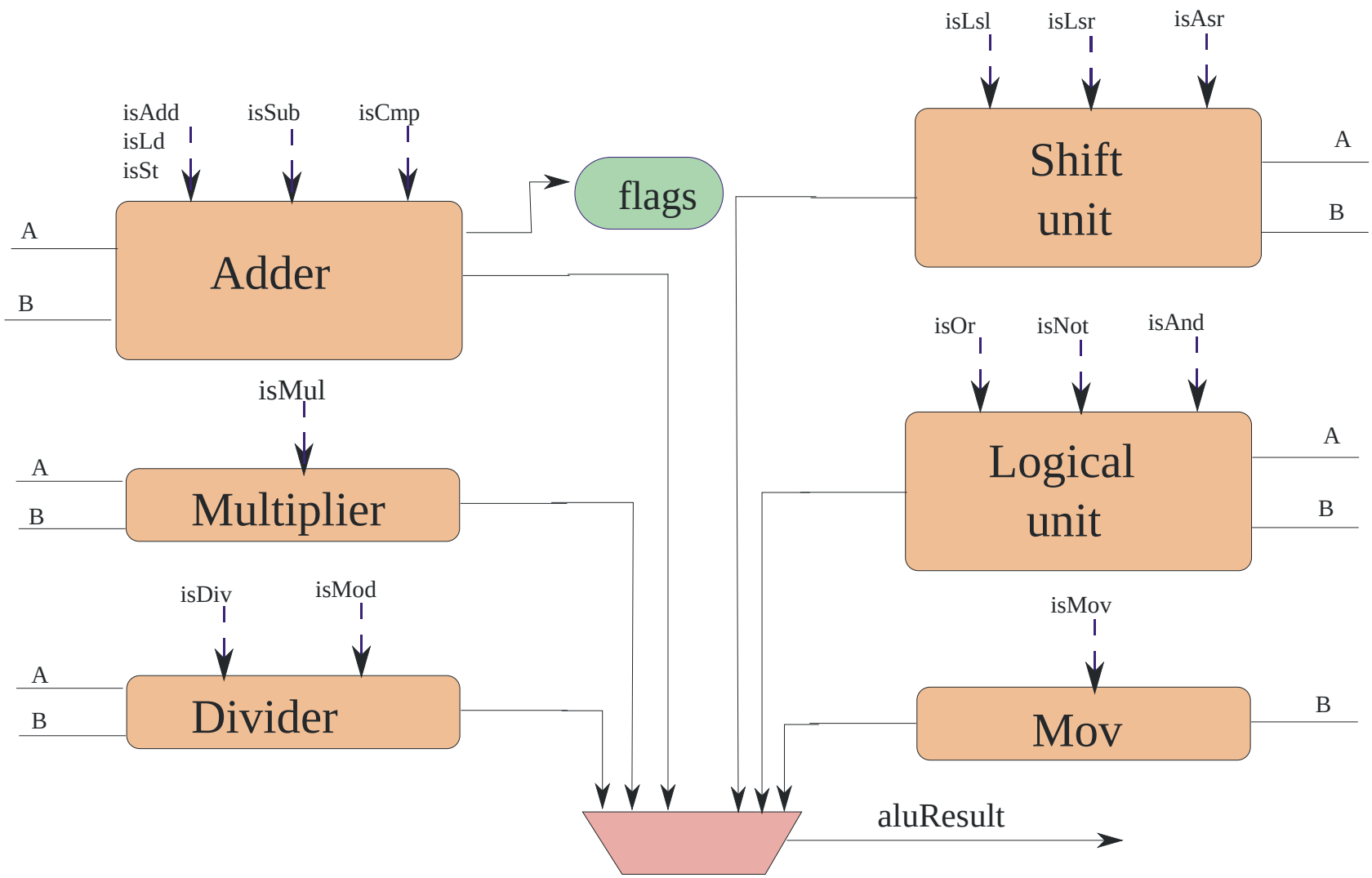
Generates the isBranchTaken Signal

ALU

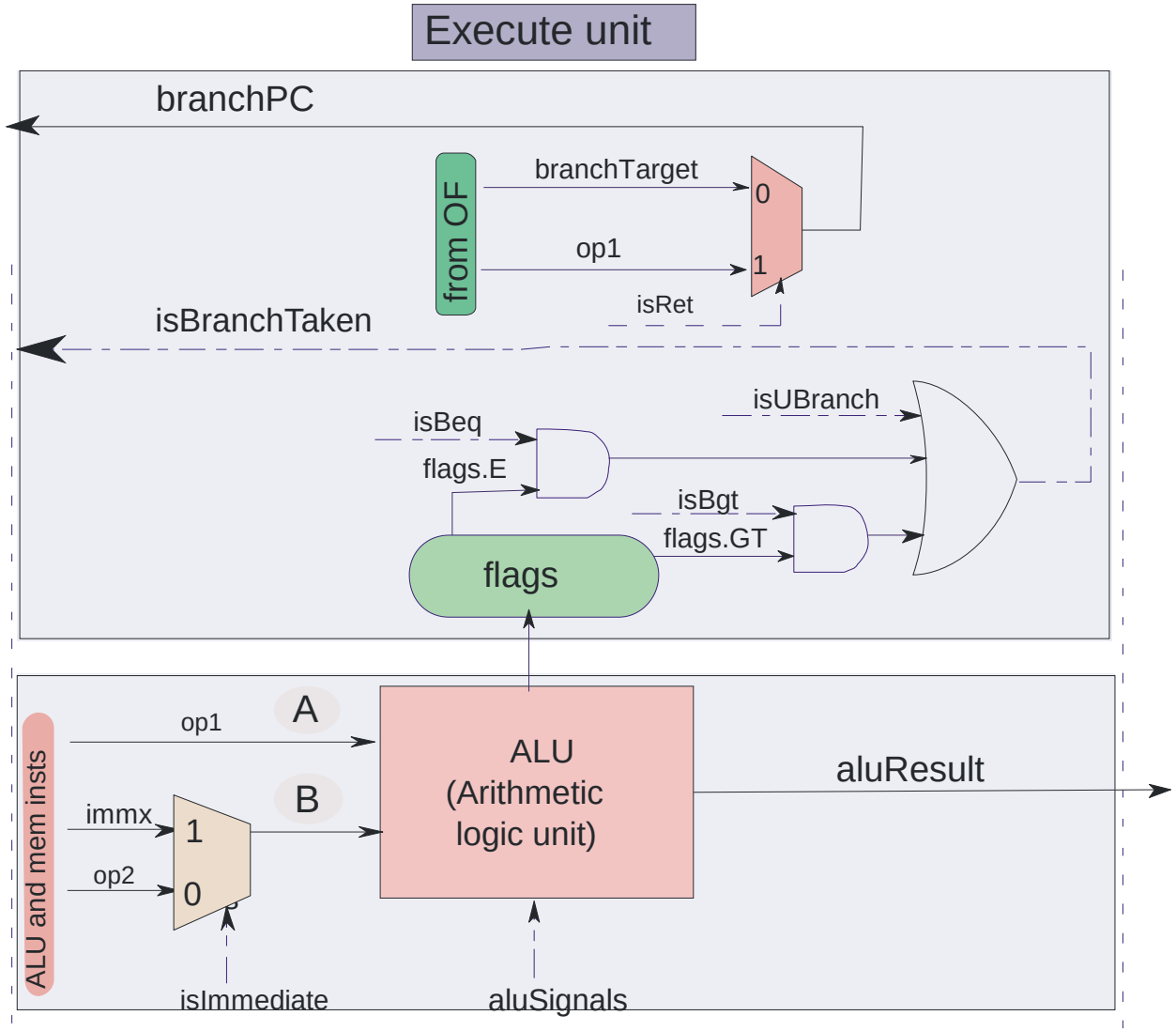


Choose between immx and op2 based on the value of the I bit

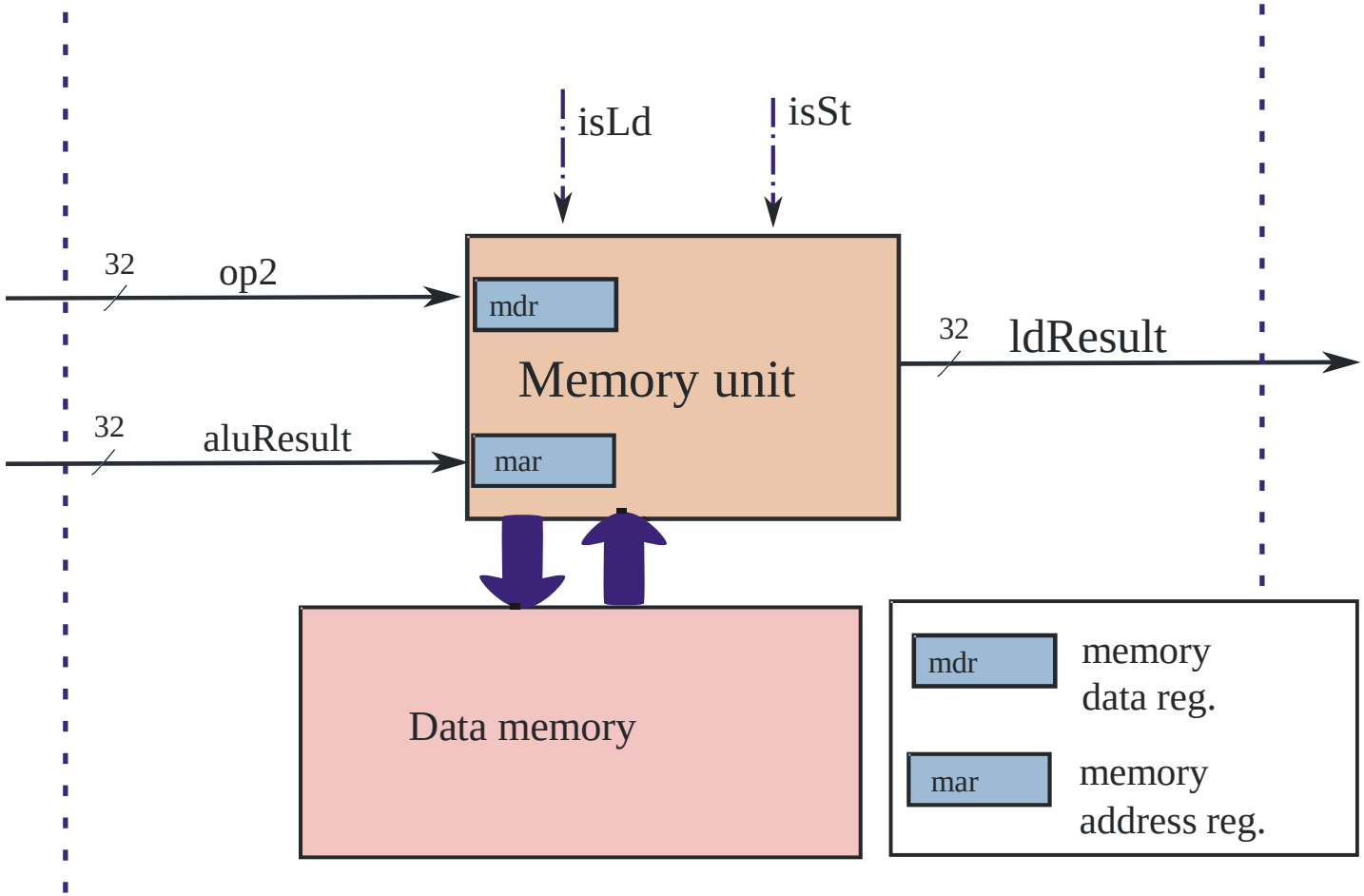
Inside the ALU



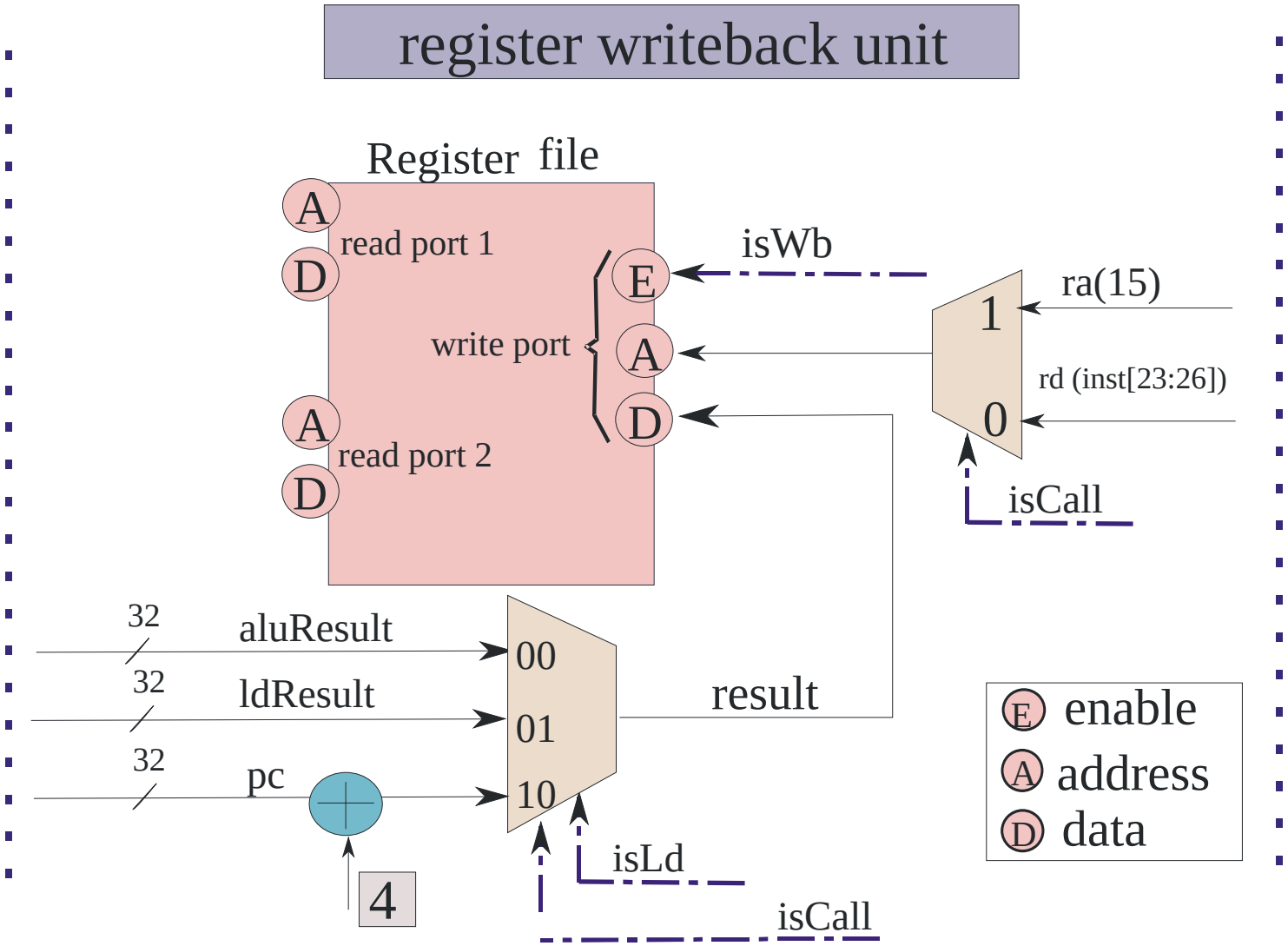
EX Unit

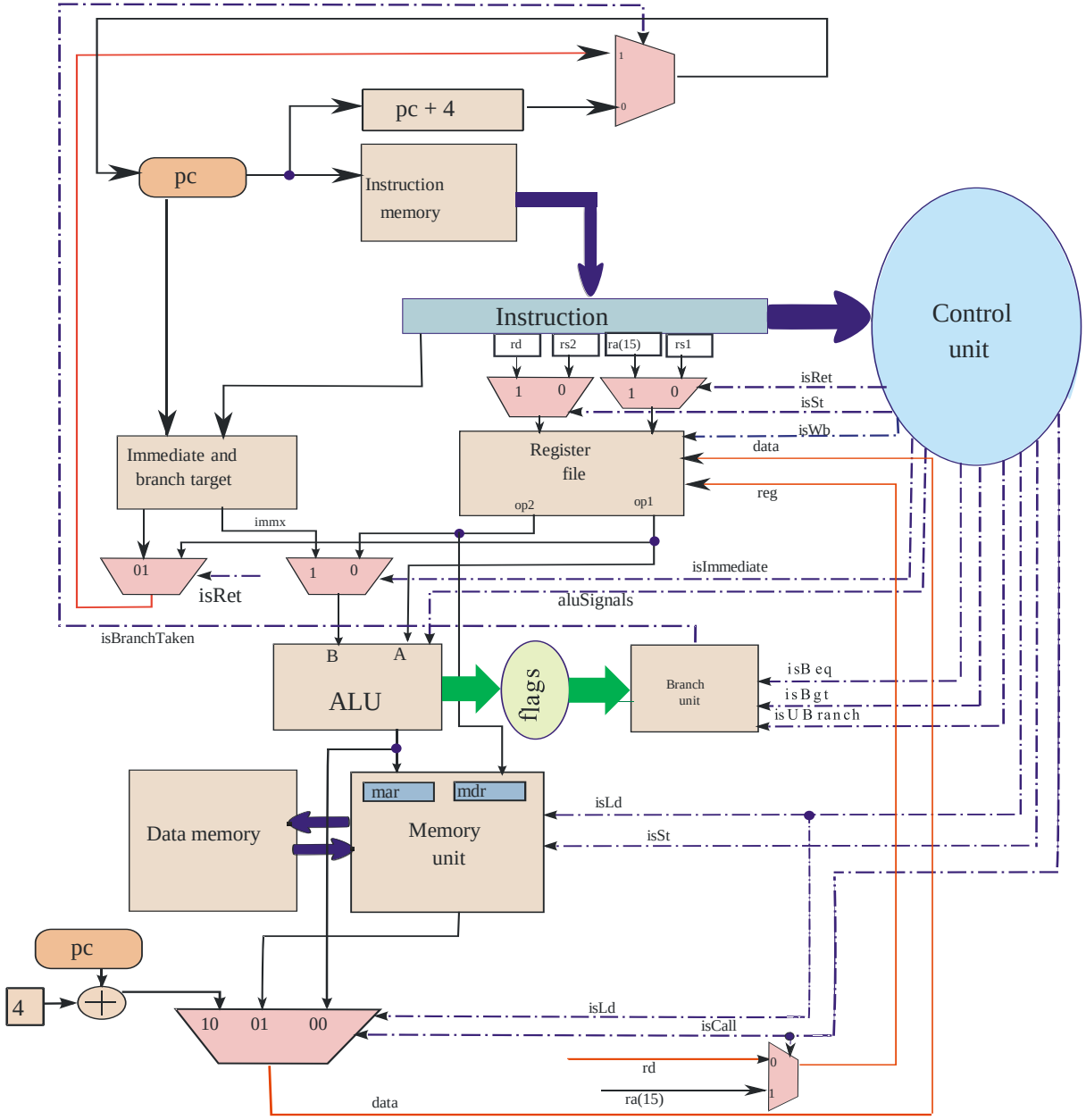

Execute unit
branchPC
from OF
branchTarget
op1
0
1
isRet
isBranchTaken
isUBranch
isBeq
flags.E
isBgt
flags.GT
flags
ALU and mem insts
op1
A
immx
op2
1
0
B
ALU
(Arithmetic
logic unit)
aluResult
isImmediate
aluSignals

MA Unit

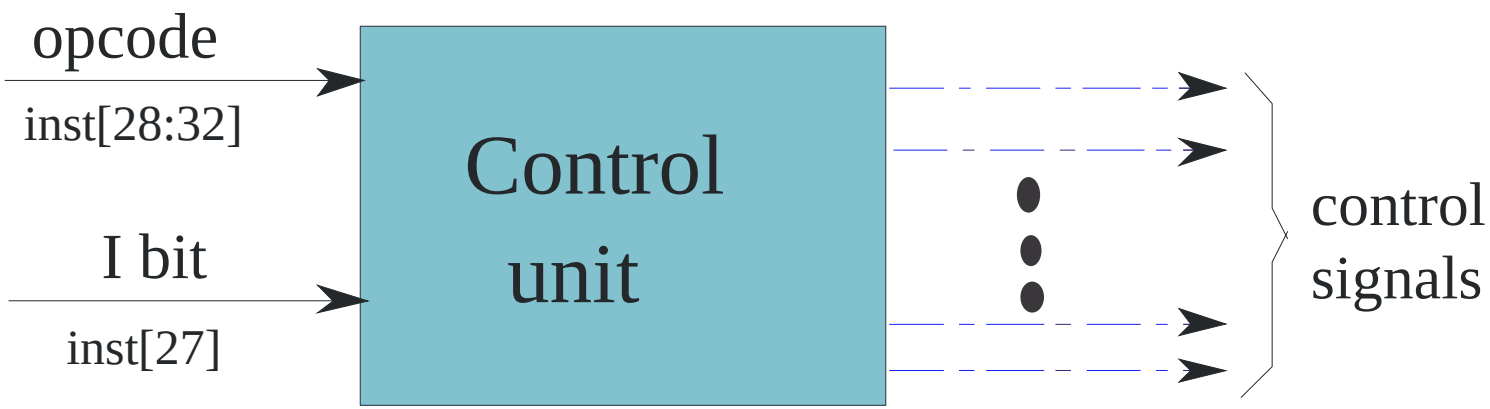

isLd
isSt
32
op2
mdr
Memory unit
32
ldResult
32
aluResult
mar
Data memory
mdr
memory
data reg.
mar
memory
address reg.

RW Unit


register writeback unit
Register file
A
D
read port 1
A
D
read port 2
E
A
D
write port
isWb
ra(15)
1
0
rd (inst[23:26])
isCall
32
aluResult
32
ldResult
32
pc
00
01
10
result
4
isLd
isCall
E enable
A address
D data


pc + 4
pc
Instruction memory
Instruction
Control unit
rd
rs2
ra(15)
rs1
isRet
isSt
isWb
data
reg
Immediate and branch target
Register file
op2
op1
immx
isImmediate
aluSignals
isRet
isBranchTaken
B
A
ALU
flags
Branch unit
isBeq
isBgt
isUBranch
Data memory
mar
mdr
Memory unit
isLd
isSt
pc
4
10
01
00
isLd
isCall
rd
ra(15)
data

The Ha rdwired Control Unit



- Given the opcode and the immediate bit
  - It generates all the control signals

Control Signals

SerialNo.	Signal	Condition
1	*isSt*	Instruction: *st*
2	*isLd*	Instruction: *ld*
3	*isBeq*	Instruction: *beq*
4	*isBgt*	Instruction: *bgt*
5	*isRet*	Instruction: *ret*
6	*isImmediate*	*I* bit set to 1
7	*isWb*	Instructions: *add*, *sub*, *mul*, *div*, *mod*, *and*, *or*, *not*, *mov*, *ld*, *lsl*, *lsr*, *asr*, *call*
8	*isUBranch*	Instructions: *b*, *call*, *ret*
9	*isCall*	Instructions: *call*

Control Signals – II

aluSignal		
10	*isAdd*	Instructions: *add*, *ld*, *st*
11	*isSub*	Instruction: *sub*
12	*isCmp*	Instruction: *cmp*
13	*isMul*	Instruction: *mul*
14	*isDiv*	Instruction: *div*
15	*isMod*	Instruction: *mod*
16	*isLsl*	Instruction: *lsl*
17	*isLsr*	Instruction: *lsr*
18	*isAsr*	Instruction: *asr*
19	*isOr*	Instruction: *or*
20	*isAnd*	Instruction: *and*
21	*isNot*	Instruction: *not*
22	*isMov*	Instruction: *mov*

Control signal Logic

opcode

op_5	op_4	op_3	op_2	op_1

immediate bit

I

Serial No.	Signal	Condition
1	*isSt*	$\overline{op_5}.op_4.op_3.op_2.op_1$
2	*isLd*	$\overline{op_5}.op_4.op_3.op_2.\overline{op_1}$
3	*isBeq*	$op_5.\overline{op_4}.\overline{op_3}.\overline{op_2}.\overline{op_1}$
4	*isBgt*	$op_5.\overline{op_4}.\overline{op_3}.\overline{op_2}.op_1$
5	*isRet*	$op_5.\overline{op_4}.op_3.\overline{op_2}.\overline{op_1}$
6	*isImmediate*	I
7	*isWb*	$\sim(op_5 + \overline{op_5}.op_3.op_1.(op_4 + \overline{op_2})) + op_5.\overline{op_4}.\overline{op_3}.op_2.op_1$
8	*isUbranch*	$op_5.\overline{op_4}.(\overline{op_3}.op_2 + op_3.\overline{op_2}.\overline{op_1})$
9	*isCall*	$op_5.\overline{op_4}.\overline{op_3}.op_2.op_1$

Control Signal Logic - II

aluSignals		
10	*isAdd*	$\overline{op5}.\overline{op4}.\overline{op3}.\overline{op2}.\overline{op1} + \overline{op5}.op4.op3.op2$
11	*isSub*	$\overline{op5}.\overline{op4}.\overline{op3}.\overline{op2}.op1$
12	*isCmp*	$\overline{op5}.\overline{op4}.op3.\overline{op2}.op1$
13	*isMul*	$\overline{op5}.\overline{op4}.\overline{op3}.op2.\overline{op1}$
14	*isDiv*	$\overline{op5}.\overline{op4}.\overline{op3}.op2.op1$
15	*isMod*	$\overline{op5}.\overline{op4}.op3.\overline{op2}.\overline{op1}$
16	*isLsl*	$\overline{op5}.op4.\overline{op3}.op2.\overline{op1}$
17	*isLsr*	$\overline{op5}.op4.\overline{op3}.op2.op1$
18	*isAsr*	$\overline{op5}.op4.op3.\overline{op2}.\overline{op1}$
19	*isOr*	$\overline{op5}.\overline{op4}.op3.op2.op1$
20	*isAnd*	$\overline{op5}.\overline{op4}.op3.op2.\overline{op1}$
21	*isNot*	$\overline{op5}.op4.\overline{op3}.\overline{op2}.\overline{op1}$
22	*isMov*	$\overline{op5}.op4.\overline{op3}.\overline{op2}.op1$