CS 31007 Autumn 2021

COMPUTER ORGANIZATION AND ARCHITECTURE

Instructors

Rajat Subhra Chakraborty (*RSC*)

Bhargab B. Bhattacharya (*BBB*)

Lecture # 7

CPU Performance Equation, Amdahl's Law, Die Yield

19 August 2021

Indian Institute of Technology Kharagpur

Computer Science and Engineering

What determines the execution time of a machine/assembly-level program *P* when it is run on a machine *M*?

- *P* consists of a number of machine-level instructions (IC: *instruction count*);
- Each machine instruction requires several clock cycles to complete (CPI: average number of *clock cycles per instruction*);
- Each clock cycle has certain time period (CCT: *clock cycle time*)

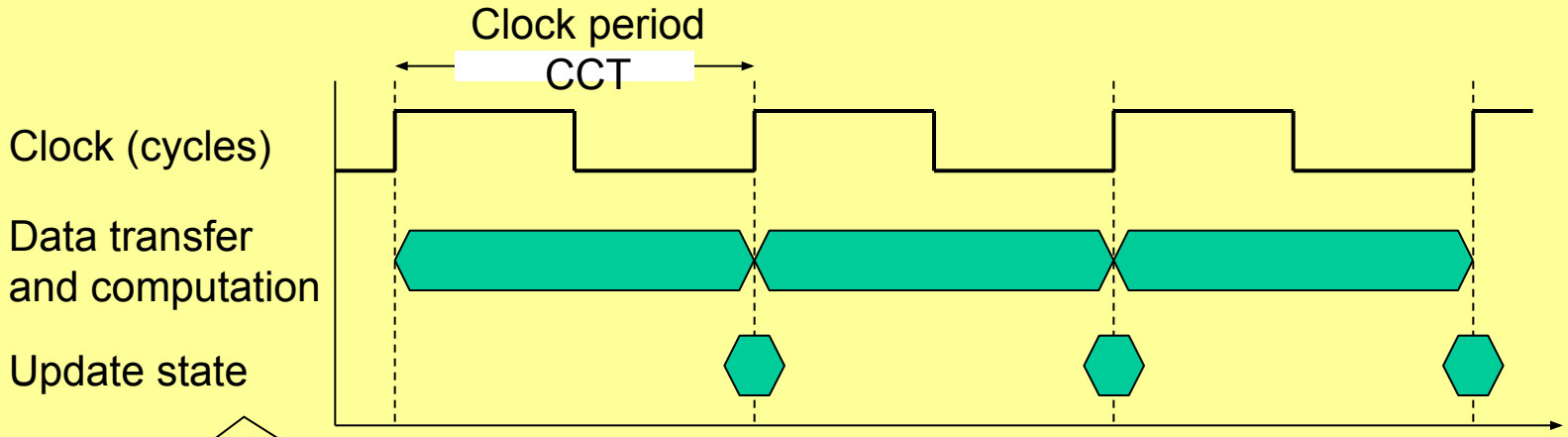
Thus, CPU-time = IC × CPI × CCT

(CPU Performance Equation);

Performance $\propto$ 1/CPU-time

CPU Clocking

- CPUs are driven by constant-rate system clocks:
  - 100 MHz clock frequency (*f*) means the system clock ticks 100 million times every second:



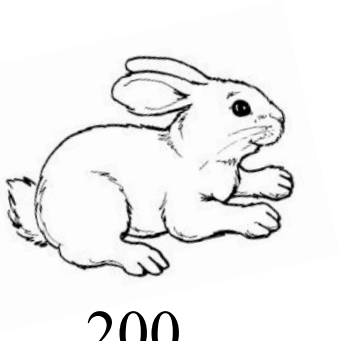
One Clock Cycle Time ("Tick"), i.e., CCT
= 1/100,000,000 *sec*
=1/100 *microsec* = 10 *nanosec*

$CCT = 1/f$

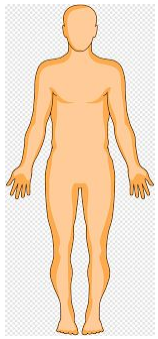
What determines the clock frequency for a processor?

460

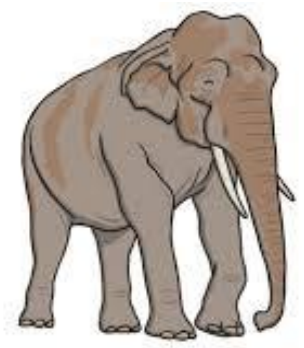
245



200

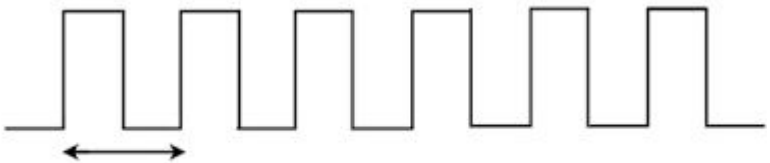


72



30

heart-beats per minute → clock period



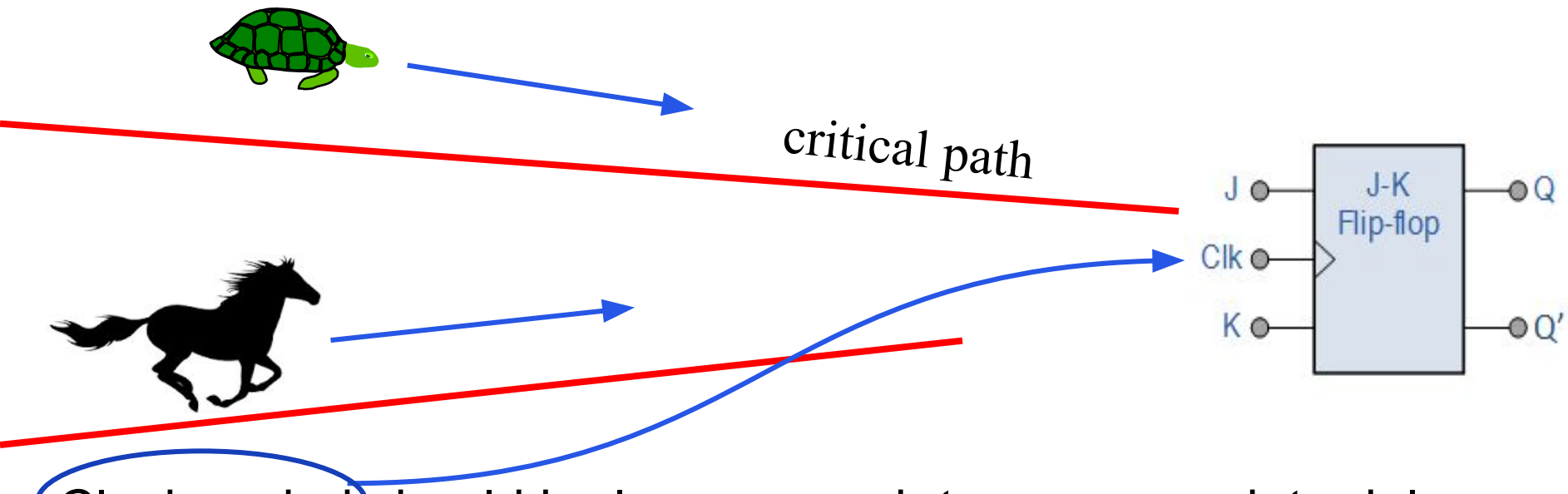
= clock cycle time (CCT)

clock frequency (f) = 1/CCT

Time period of pulsating heart (contraction and relaxation) should be *just* large enough so that blood reaches to the farthest extremities in one pumping cycle

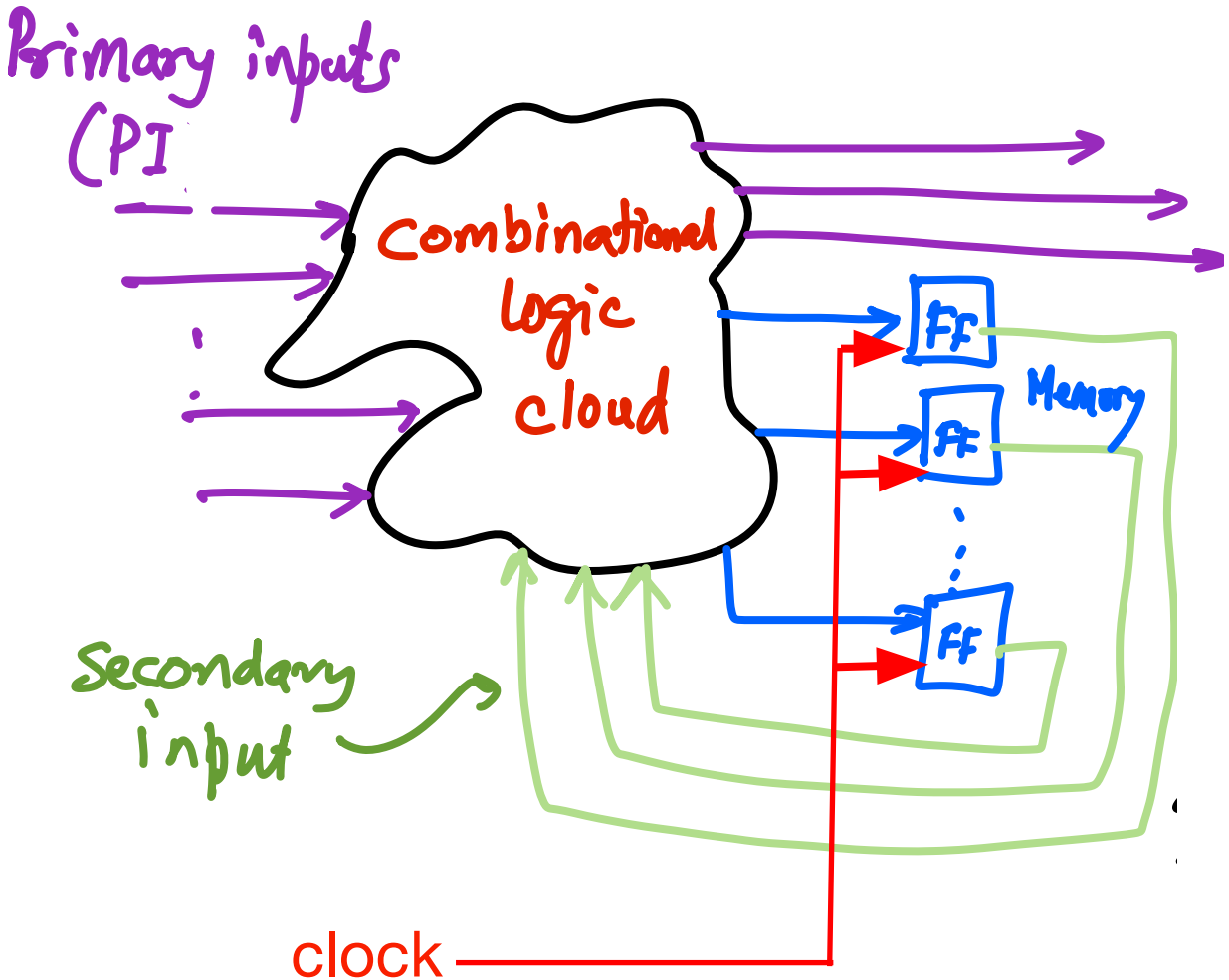
Clock Timing

CCT ≥ signal delay along the longest sensitizable logic path (critical path) among those from PI/outputs(FFs) → PO/inputs(FFs)

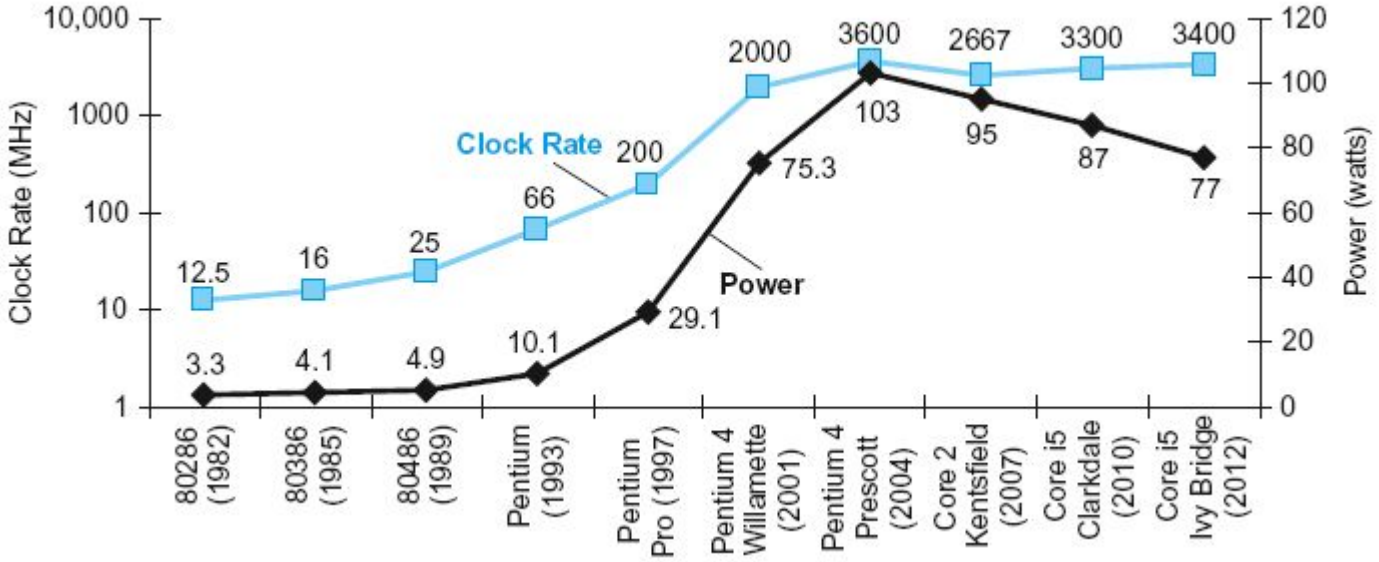


Clock period should be large enough to accommodate delays along critical paths in the circuit (longest ones);
but should not be too large – system slows down unnecessarily

Critical delay in logic circuit $\Longrightarrow$ CCT



Power Issues and CCT

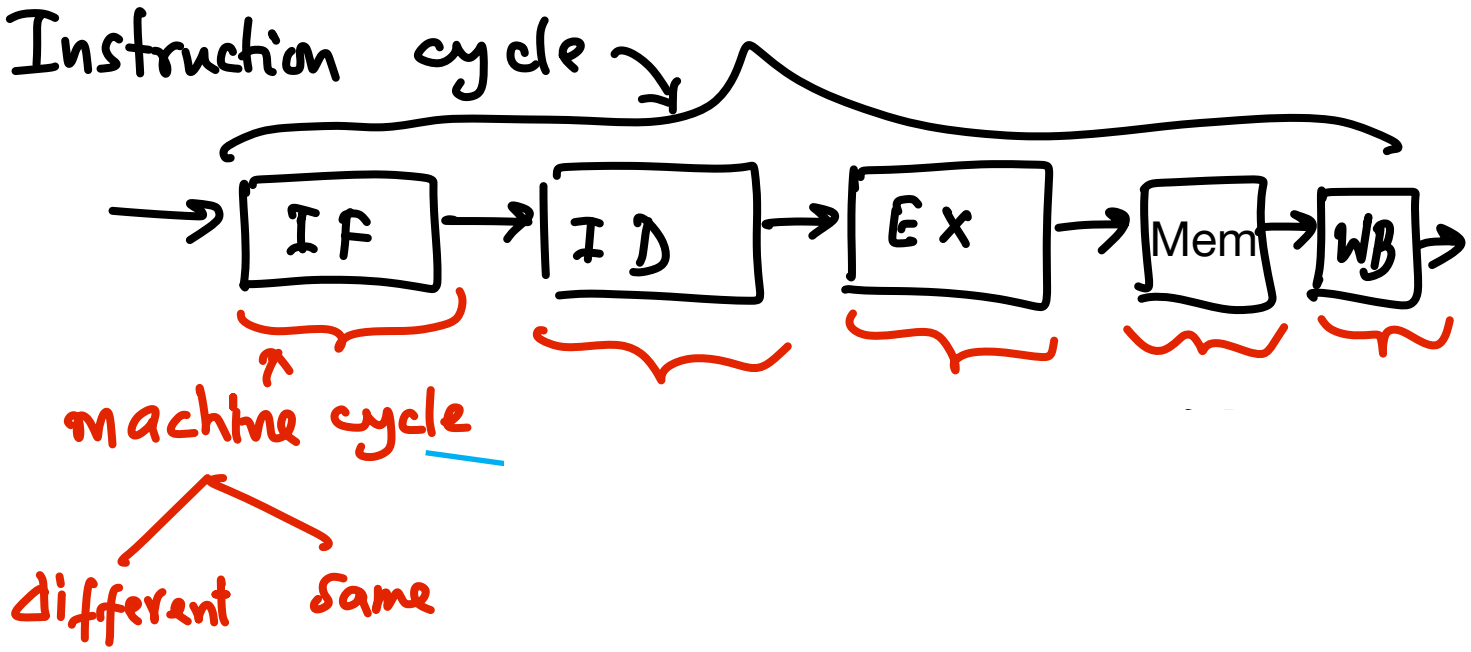


- In CMOS IC technology

$$\text{Power} = \text{Capacitive load} \times \text{Voltage}^2 \times \text{Frequency}$$

~×40 | 5V → 1V | ×1000

CPU_time = IC * CPI * CCT



CPU Time

$$\text{CPU Time} = \text{CPU Clock Cycles} \times \text{Clock Cycle Time}$$

$$= \frac{\text{CPU Clock Cycles}}{\text{Clock Rate}}$$

- Performance improved by
  - Reducing number of clock cycles
  - Increasing clock rate (*a.k.a.* frequency)
  - Hardware designer must often trade off clock rate against cycle count

CPU Time Example

- Computer A: 2GHz clock, 10s CPU time
- Designing Computer B
  - Aim for 6s CPU time
  - Can do faster clock, but causes 1.2 × clock cycles
- How fast must Computer B clock be?

$$\text{Clock Rate}_B = \frac{\text{Clock Cycles}_B}{\text{CPU Time}_B} = \frac{1.2 \times \text{Clock Cycles}_A}{6\text{s}}$$

$$\text{Clock Cycles}_A = \text{CPU Time}_A \times \text{Clock Rate}_A$$

$$= 10\text{s} \times 2\text{GHz} = 20 \times 10^9$$

$$\text{Clock Rate}_B = \frac{1.2 \times 20 \times 10^9}{6\text{s}} = \frac{24 \times 10^9}{6\text{s}} = 4\text{GHz}$$

Instruction Count and CPI

$$\text{Clock Cycles} = \text{Instruction Count} \times \text{Cycles per Instruction}$$

$$\text{CPU Time} = \text{Instruction Count} \times \text{CPI} \times \text{Clock Cycle Time}$$

$$= \frac{\text{Instruction Count} \times \text{CPI}}{\text{Clock Rate}}$$

- Instruction Count for a program
  - Determined by program, ISA and compiler
- Average cycles per instruction
  - Determined by CPU hardware
  - If different instructions have different CPI
    - Average CPI affected by instruction mix

CPI Example

- Computer A: Cycle Time = 250ps, CPI = 2.0
- Computer B: Cycle Time = 500ps, CPI = 1.2
- Same ISA
- Which is faster, and by how much?

$$\text{CPU Time}_A = \text{Instruction Count} \times \text{CPI}_A \times \text{Cycle Time}_A$$
$$= I \times 2.0 \times 250\text{ps} = I \times 500\text{ps}$$

A is faster…

$$\text{CPU Time}_B = \text{Instruction Count} \times \text{CPI}_B \times \text{Cycle Time}_B$$
$$= I \times 1.2 \times 500\text{ps} = I \times 600\text{ps}$$

$$\frac{\text{CPU Time}_B}{\text{CPU Time}_A} = \frac{I \times 600\text{ps}}{I \times 500\text{ps}} = 1.2$$

…by this much

CPI in More Detail

- If different instruction classes take different numbers of cycles

$$\text{Clock Cycles} = \sum_{i=1}^{n} (\text{CPI}_i \times \text{Instruction Count}_i)$$

  - Weighted average CPI

$$\text{CPI} = \frac{\text{Clock Cycles}}{\text{Instruction Count}} = \sum_{i=1}^{n} \left(\text{CPI}_i \times \underbrace{\frac{\text{Instruction Count}_i}{\text{Instruction Count}}}_{\text{Relative frequency}} \right)$$

CPI Example: Who is better?

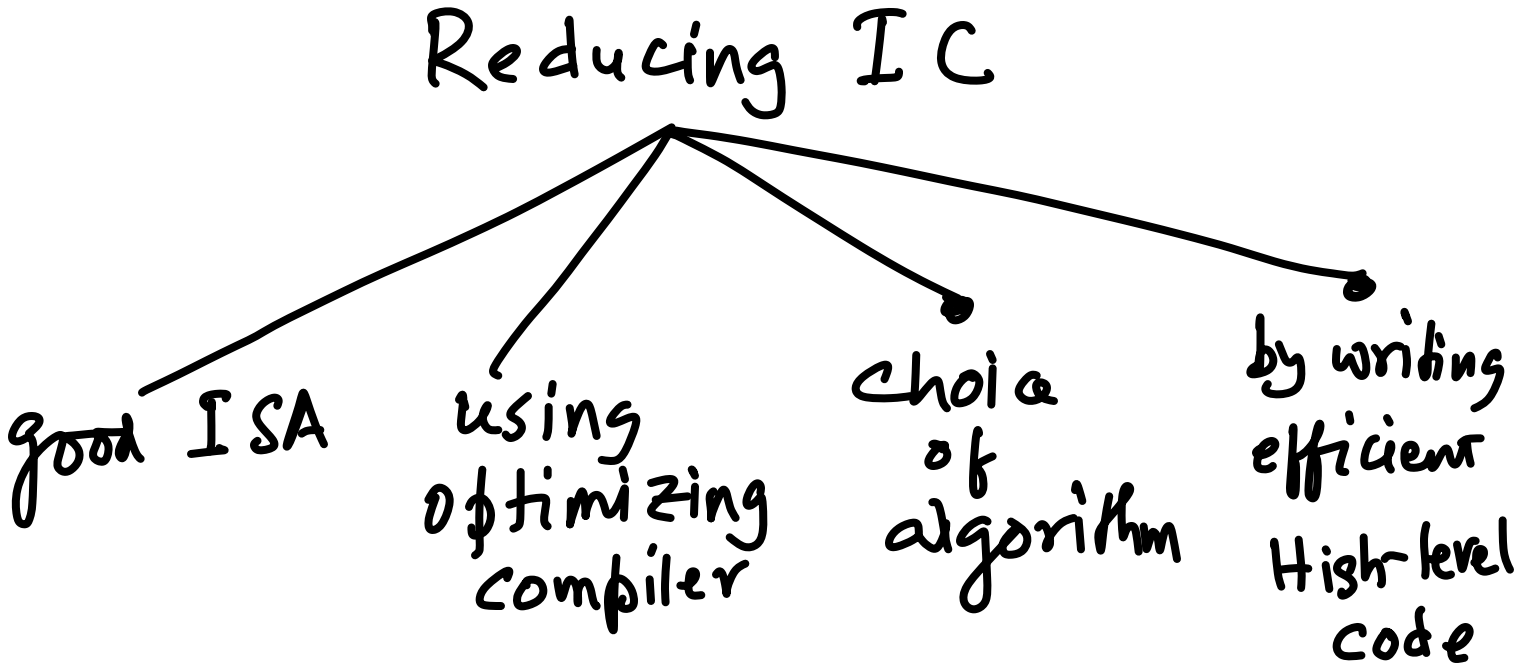
- Two compiled codes for the *same problem,* using instructions in classes A, B, C

Class	A	B	C
CPI for class	1	2	3
IC for Programmer 1	2	1	2
IC for Programmer 2	4	1	1

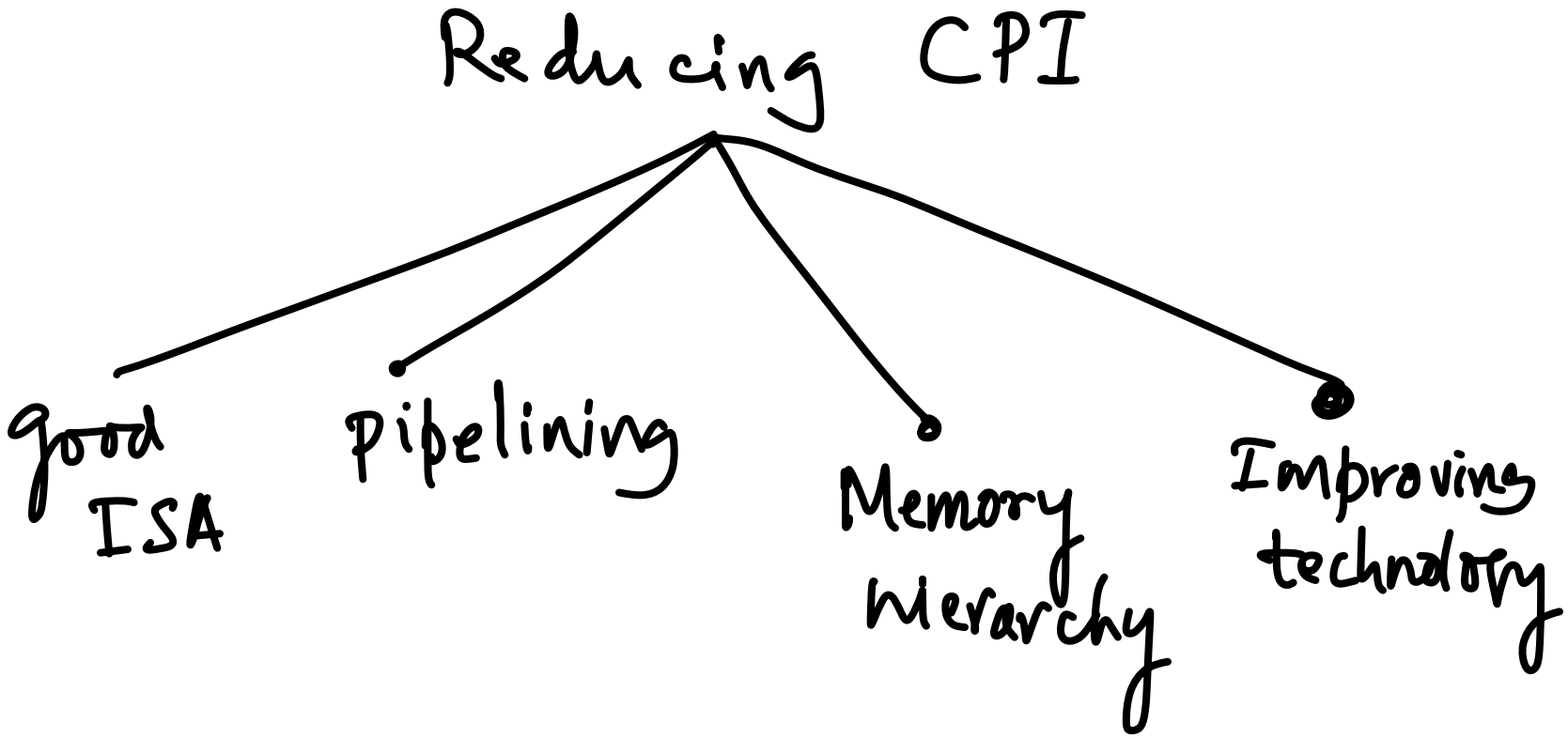
- Sequence 1: IC = 5
  - Clock Cycles
    = 2×1 + 1×2 + 2×3
    = 10
  - Avg. CPI = 10/5 = 2.0
- Sequence 2: IC = 6
  - Clock Cycles
    = 4×1 + 1×2 + 1×3
    = 9
  - Avg. CPI = 9/6 = 1.5

$$CPU\_time = IC * CPI * CCT$$

$$Performance \propto \frac{1}{CPU\_time}$$

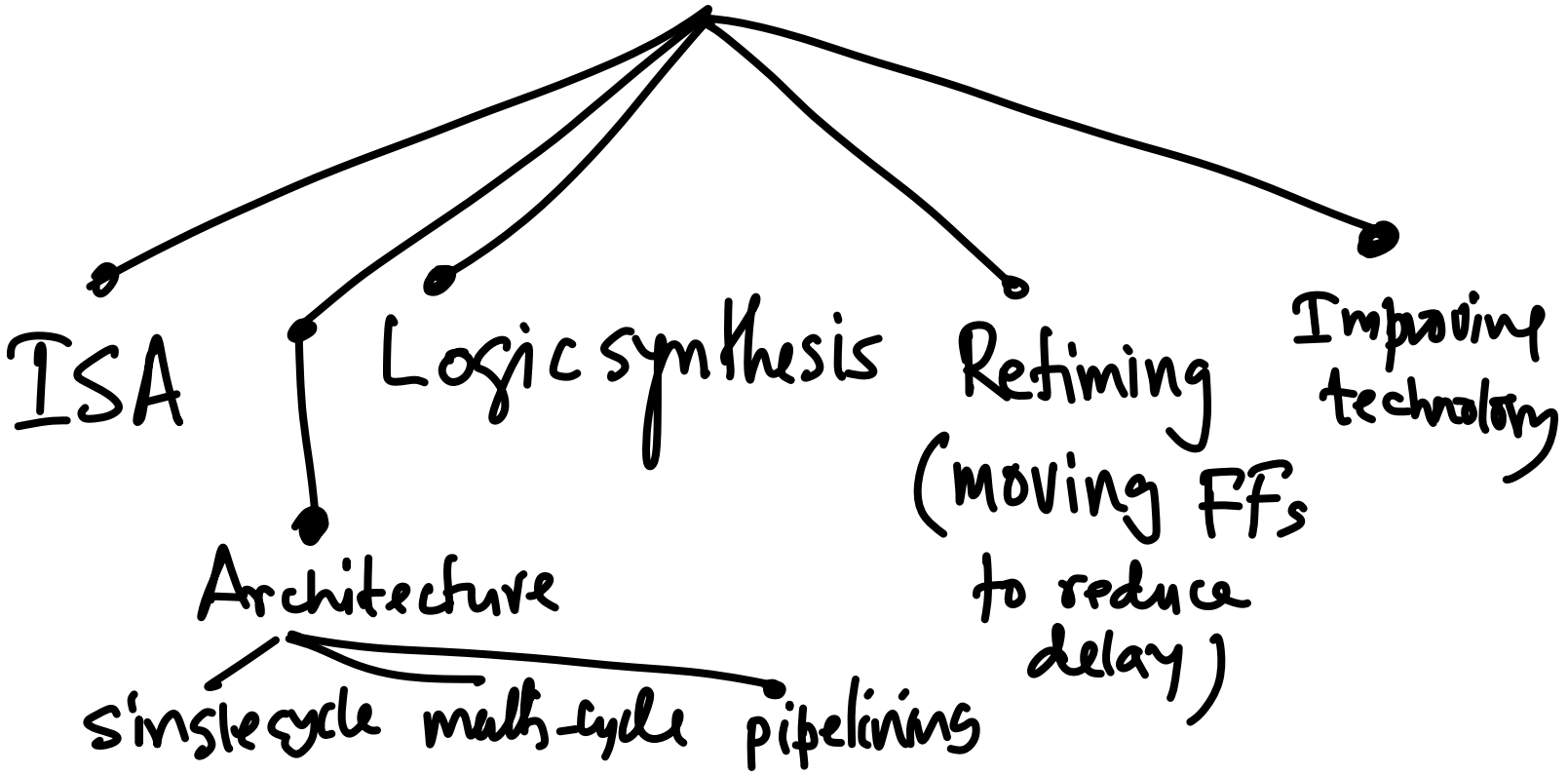


CPU_time = IC * CPI * CCT



CPU_time = IC * CPI * CCT

Reducing CCT



Performance Summary

The BIG Picture: CPU Performance Equation

$$\text{CPU-time} = \text{IC} \times \text{CPI} \times \text{CCT}$$

- Performance depends on
  - Algorithm: affects IC, possibly CPI
  - Programming language: affects IC, CPI
  - Compiler: affects IC, CPI
  - Instruction set architecture: affects IC, CPI, CCT
  - CPI is also affected by memory hierarchy, pipelining; CCT is affected by logic design, technology

Reducing Power

- Suppose a new CPU has
  - 85% of capacitive load of old CPU
  - 15% voltage and 15% frequency reduction

$$\frac{P_{new}}{P_{old}} = \frac{C_{old} \times 0.85 \times (V_{old} \times 0.85)^2 \times F_{old} \times 0.85}{C_{old} \times V_{old}^{\ 2} \times F_{old}} = 0.85^4 = 0.52$$

- The power wall
  - We can't reduce voltage further
  - We can't remove more heat
- How else can we improve performance?

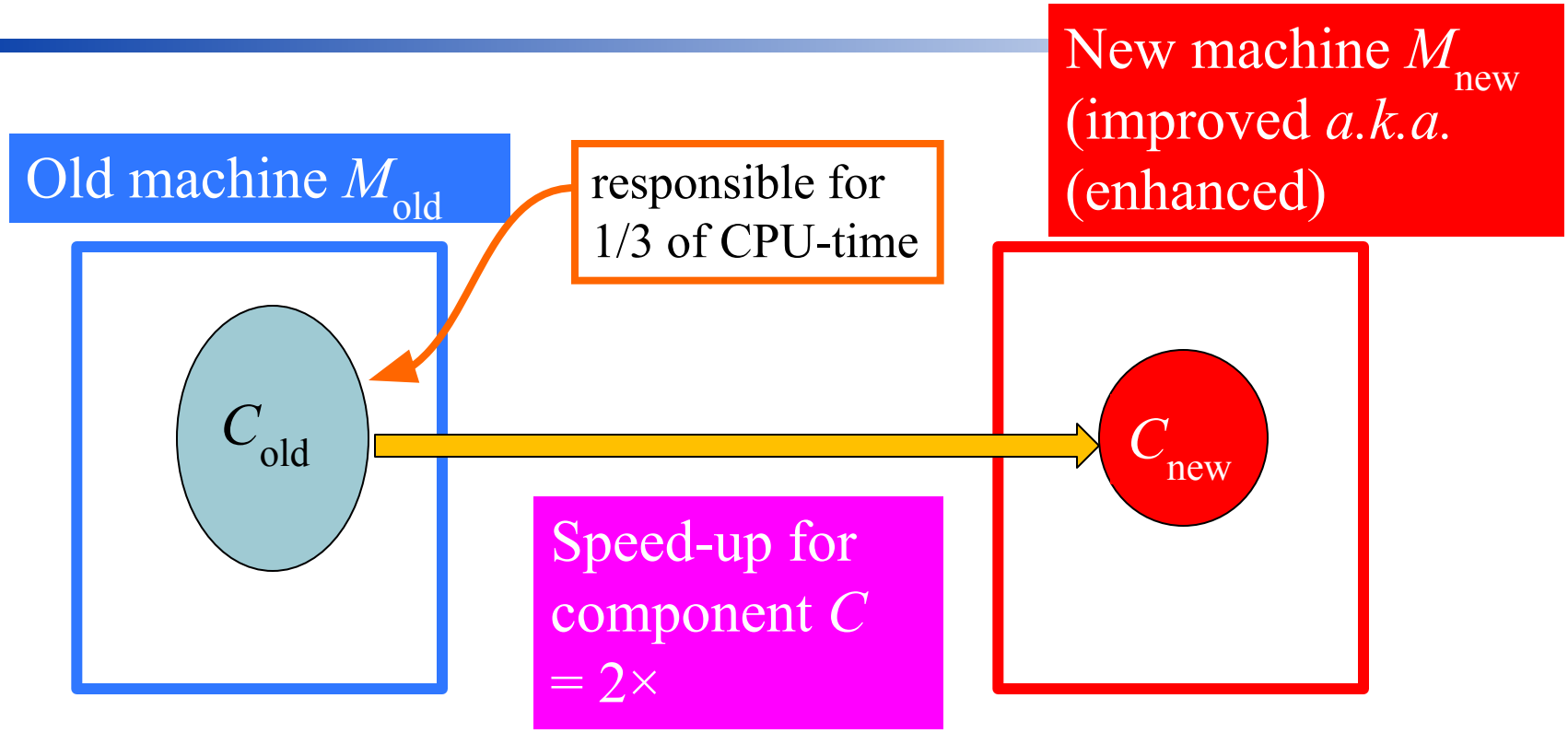
How to improve the performance of a machine by “Enhancement” of one or more components?

- Amdahl’s Law

How to improve the performance of a computing system by adding more processors (cores)?

- Gustavson-Barsis Law

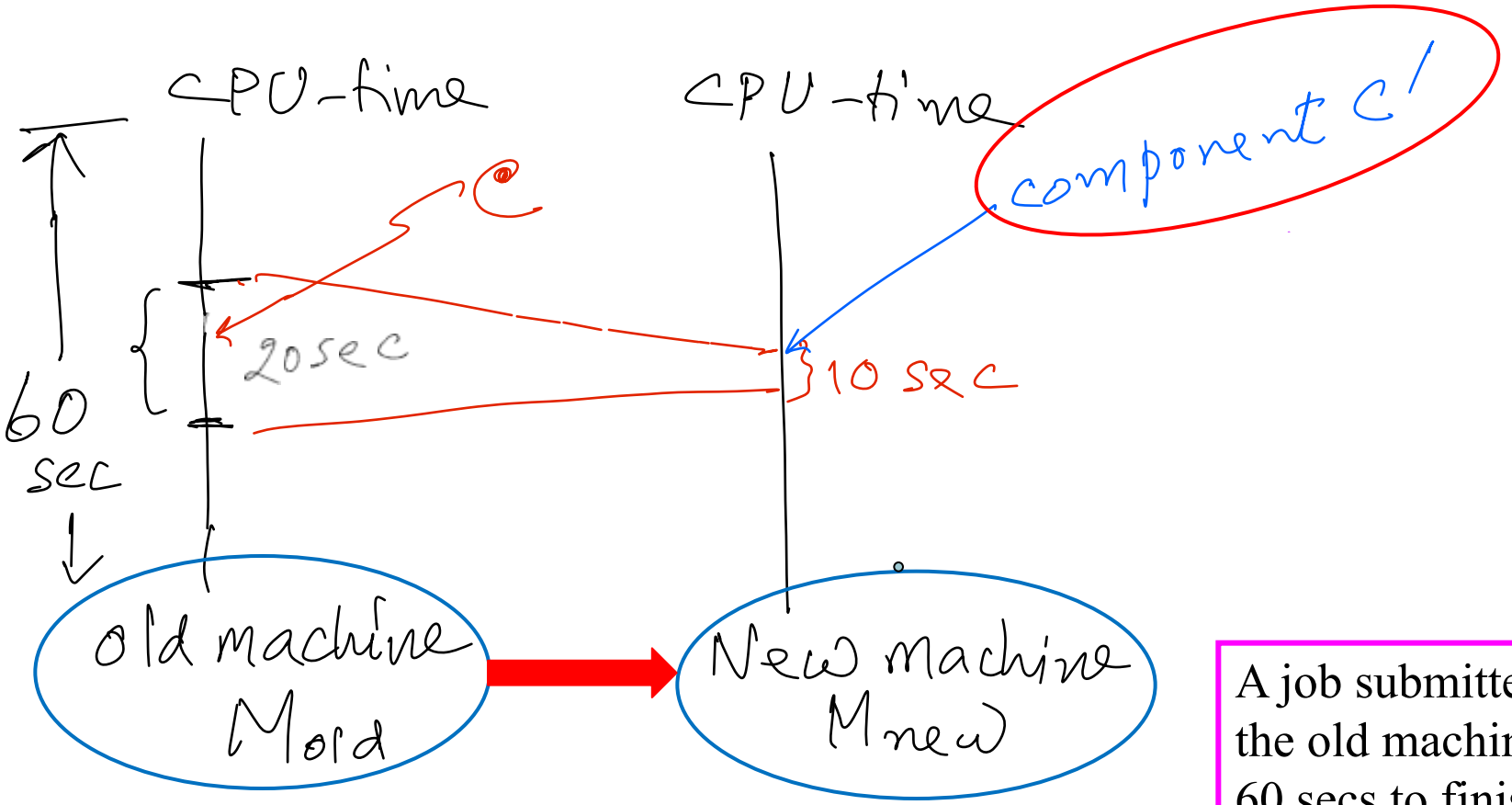
Example: Improving performance in steps



Question: What is the overall speed-up of M_{new} *w.r.t.* M_{old}?

Gradual enhancement of resources for speed-up

Let C be a component (e.g., adder) of an old machine, which is improved to C' in a new machine

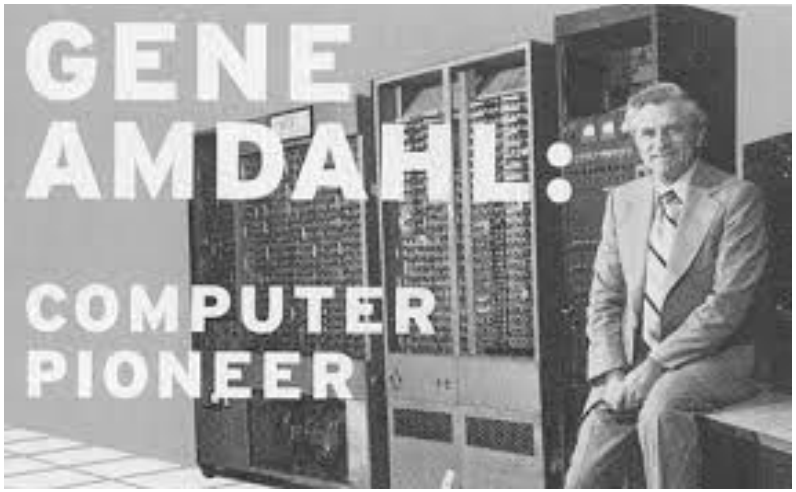


A job submitted to the old machine takes 60 secs to finish

Gradual enhancement of resources for speed-up

Question: what is the overall speed-up?

The general result concerning the overall speed-up, when a component of the old machine is enhanced is captured in "Amdahl's Law"

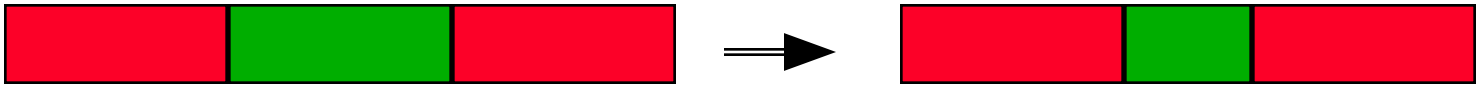


Gene Amdahl
(1922-2015)

Amdahl's Law

Speed-up due to enhancement E:

$$\text{Speed-up(E)} = \frac{\text{CPU-time w/o E}}{\text{CPU-time w/E}} = \frac{\text{Performance w/E}}{\text{Performance w/o E}}$$



F

Suppose that enhancement E accelerates a fraction F of the task by a factor S, and the remainder of the task is unaffected

Amdahl's Law

- Law of diminishing return: unaffected fraction will determine the limiting case!
- Improvement of larger fraction will yield higher overall speed-up Make the common case faster

Example: Amdahl's Law

- Floating-point (FP) instructions are improved to run 2X in a new machine
- 10% of actual instructions are FP

Example: Amdahl's Law

- Floating-point (FP) instructions are improved to run 2× in a new machine
- 10% of actual instructions are FP

$$\text{CPU-time}_{new} = \text{CPU-time}_{old} \times (0.9 + 0.1/2) = 0.95 \times \text{CPU-time}_{old}$$

$$\text{Speed-up}_{overall} = \frac{1}{0.95} = 1.053$$

- For FP Speed-up 100X, $\text{Speed-up}_{overall} = 1.109$

Pitfall: Amdahl's Law

- Improving an aspect of a computer and expecting a proportional improvement in overall performance

$$T_{improved} = \frac{T_{affected}}{improvement\ factor} + T_{unaffected}$$

- Example: multiply accounts for 80s/100s
  - How much improvement in multiply performance to get 5× overall?

$$20 = \frac{80}{n} + 20$$

  - Can't be done!

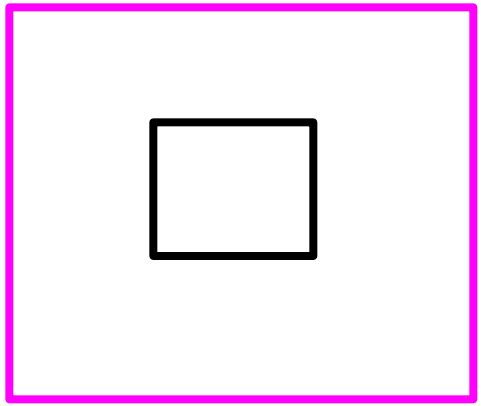
- Corollary: Make the common case fast

Multiprocessors

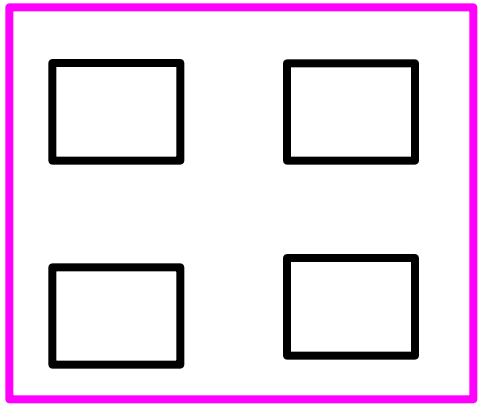
- Multicore microprocessors
  - More than one processor per chip
  - Clock frequency limited
- Requires explicitly parallel programming
  - Compare with instruction level parallelism
    - Hardware executes multiple instructions at once
    - Hidden from the programmer
  - Hard to do
    - Programming for performance
    - Load balancing
    - Optimizing communication and synchronization

Parallelism: Adding multiple processors

Pessimistic view



Single processor



Multi-processor

1. X:= A + B
2. Y:= X * C
3. D:= B + C
4. E:= X – C

Statement 1 & 2 cannot be executed in parallel (serial); 1 and 3 can be executed in parallel; 2 & 4 can be done in parallel

What kind of *speed-up* is expected?

s: time needed by a single processor on serial parts of *P*;
p: time needed by a single processor on the parts of *P* that can be parallelized; $s + p = 1$
By **Amdahl's Law**, the speed-up in the multi-core processor $= 1/(s + p/N)$ ⟹ If $s = 10\%$, the maximum speed-up is 10X

Parallelism: Adding multiple processors
Amdahl's Law *versus* Gustavson-Barsis Law

By *Amdahl's Law*, the speed-up in the multi-core processor = $1/(s + p/N)$, i.e., it is limited by s, increasing N does not help.

In Amdahl's Law, the *size of the job* is assumed to be *constant*; however, multi-cores can solve a large job in the same time. This leads to Gustavson-Barsis Law:

Let s and p represent serial and parallel time on N-core system; The single core processor would take $s + (p \times N)$ *time* to perform the same job P.

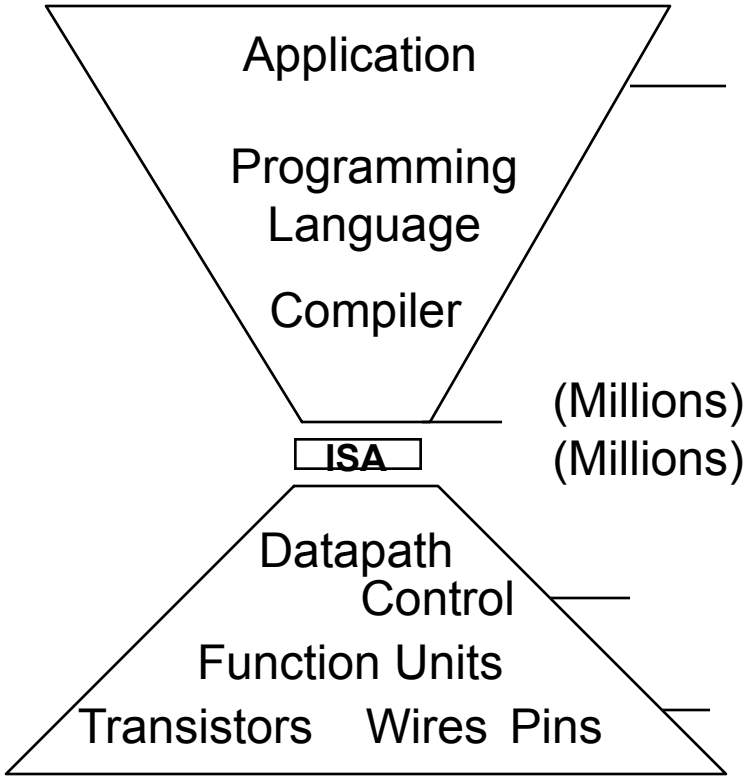
Hence, Speed-up = $(s + (p \times N))/(s + p) = (s + (p \times N)$

Thus, by *Gustavson-Barsis Law*, speed-up **grows** with N

J.L. Gustavson, Reevaluating Amdahl's Law, *CACM*, May 1988

Optimistic view

Metrics of Performance



Throughput per month
Operations per second

(Millions) of Instructions per second: MIPS
(Millions) of (FP) operations per second: MFLOP/s

Megabytes per second

Cycles per second (clock rate)

ISA: RISC *versus* CISC

- RISC (Reduced Instruction Set Computing)
  - Keep the instruction set small and simple
  - Fixed instruction lengths, easy encoding
  - Load-store instruction sets; register-indirect addressing with offset; ALU operations involve only registers

    lw `$t1 -40($t2); add $t0,$s1,$s2`
  - Limited addressing modes
  - Limited operations
  - IC high, CPI low

  Advantage: enables hardware simple and fast; decoding simple; pipelining easy, die-area small

  Performance is optimized focused on software

RISC Example: MIPS, Sun SPARC, HP PA-RISC, IBM PowerPC, Alpha, RISC-V, ARM

- **CISC - Complex Instruction Set Computer**
  - complex instructions, encoding non-uniform
  - different lengths
  - can handle multiple operands
  - complex functionalities
  - IC low, CPI high

pipelining becomes harder; hardware cost increases; compiler design becomes more involved;

- Examples of CISC processors are Intel CPUs, System/360, VAX, AMD

Eight Great Ideas

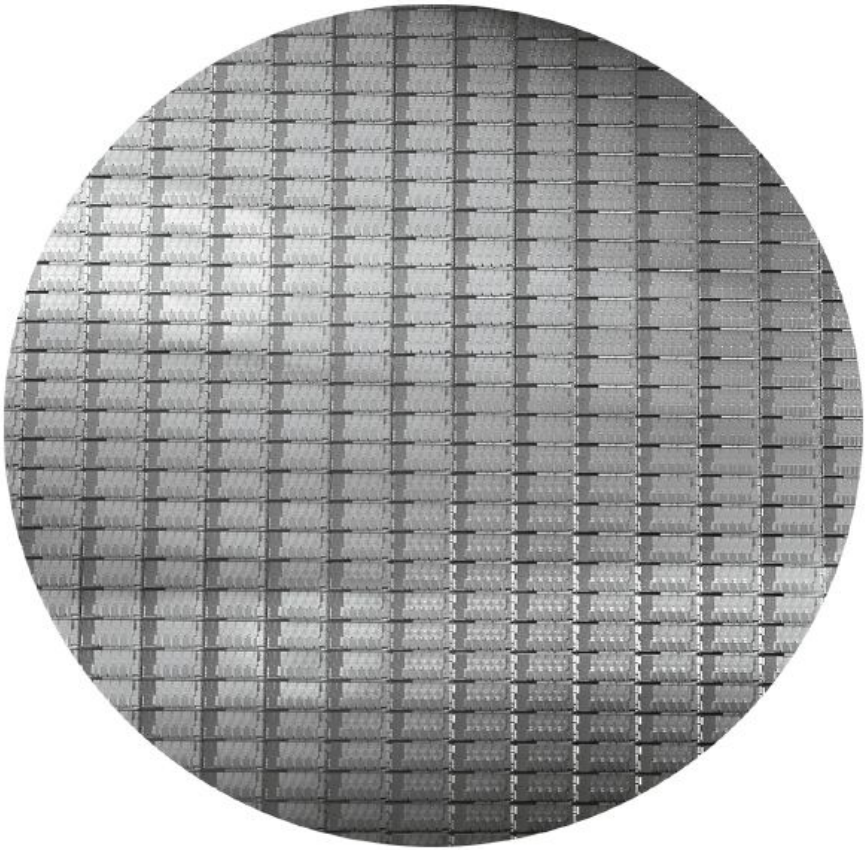
- Design for ***Moore's Law***
- Use ***abstraction*** to simplify design
- Make the ***common case fast***
- Performance *via* ***parallelism***
- Performance *via* ***pipelining***
- Performance *via* ***branch prediction***
- ***Hierarchy*** of memories
- ***Dependability*** *via* redundancy

Amdahl's law

Gustavson-Barsis law

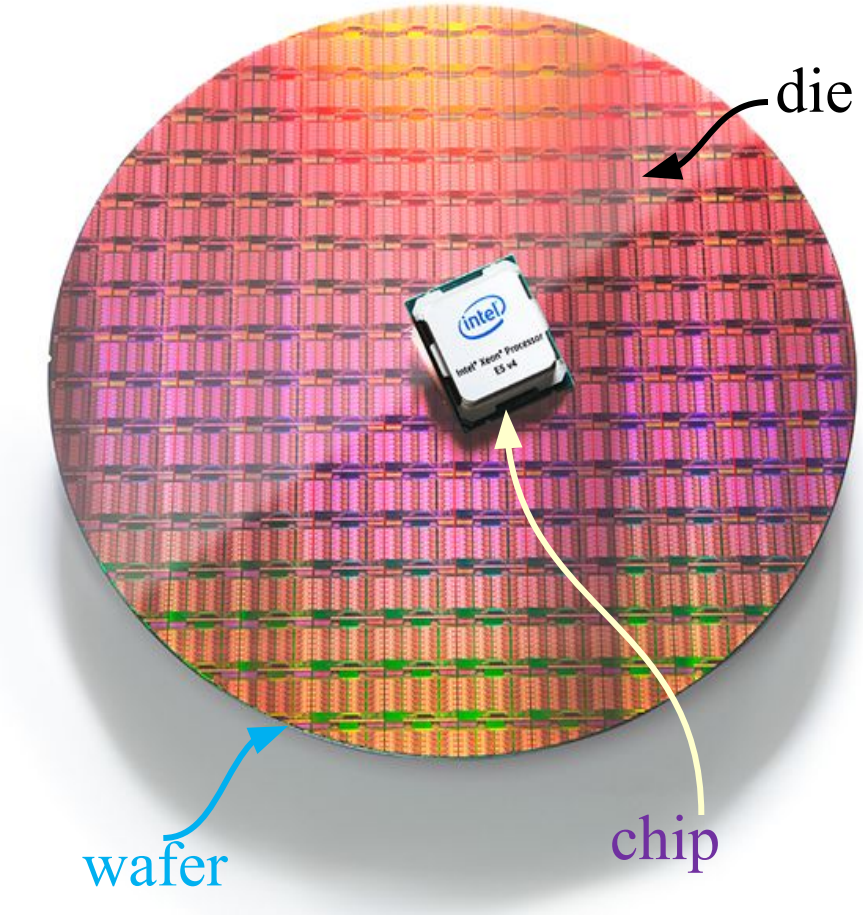
Estimation of Die Yield

Intel Core i7 Wafer

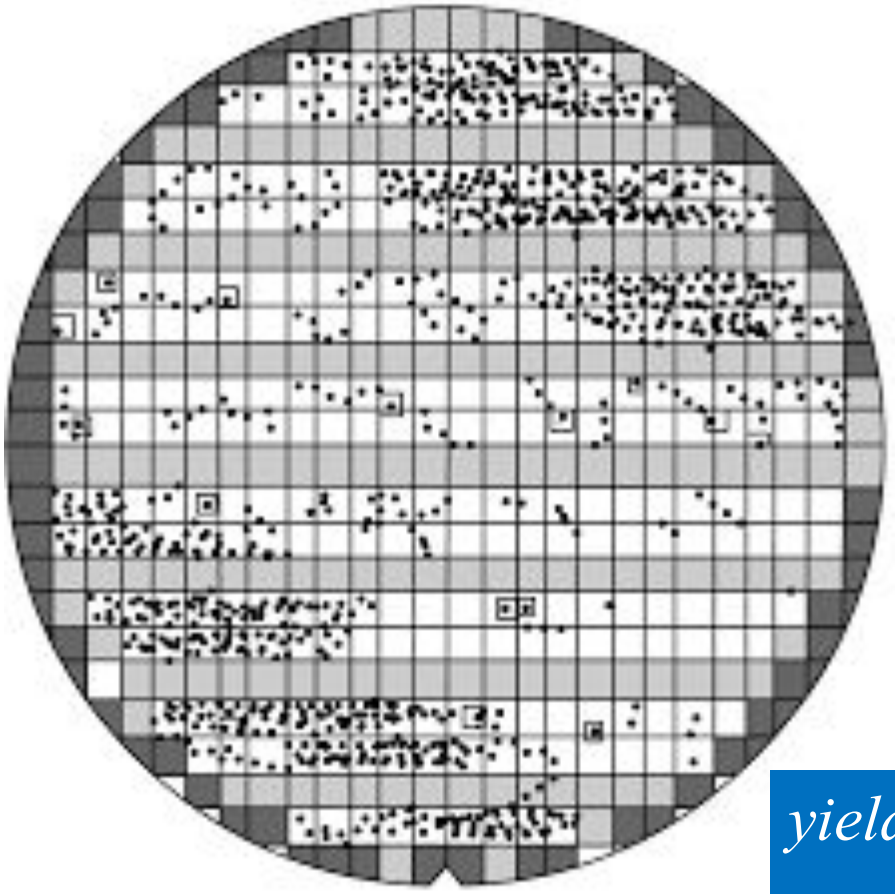


- 300mm wafer, 280 dies, 32*nm* technology, 4-8 cores per die
- Each die is 20.7 x 10.5 *mm*

Intel Xeon E5-2600 v4 chips with 22 processors, 14*nm process technology* (2016)



Defects in IC's and Yield



defects on chips make them unusable;

peripheral chips are incompletely fabricated, and hence unusable (square-peg in round-hole problem);

yield: fraction of good chips produced;

chip-cost depends on yield;

Integrated Circuit Cost

$$\text{Cost per die} = \frac{\text{Cost per wafer}}{\text{Dies per wafer} \times \text{Yield}}$$

$$\text{Dies per wafer} \approx \text{Wafer area}/\text{Die area}$$

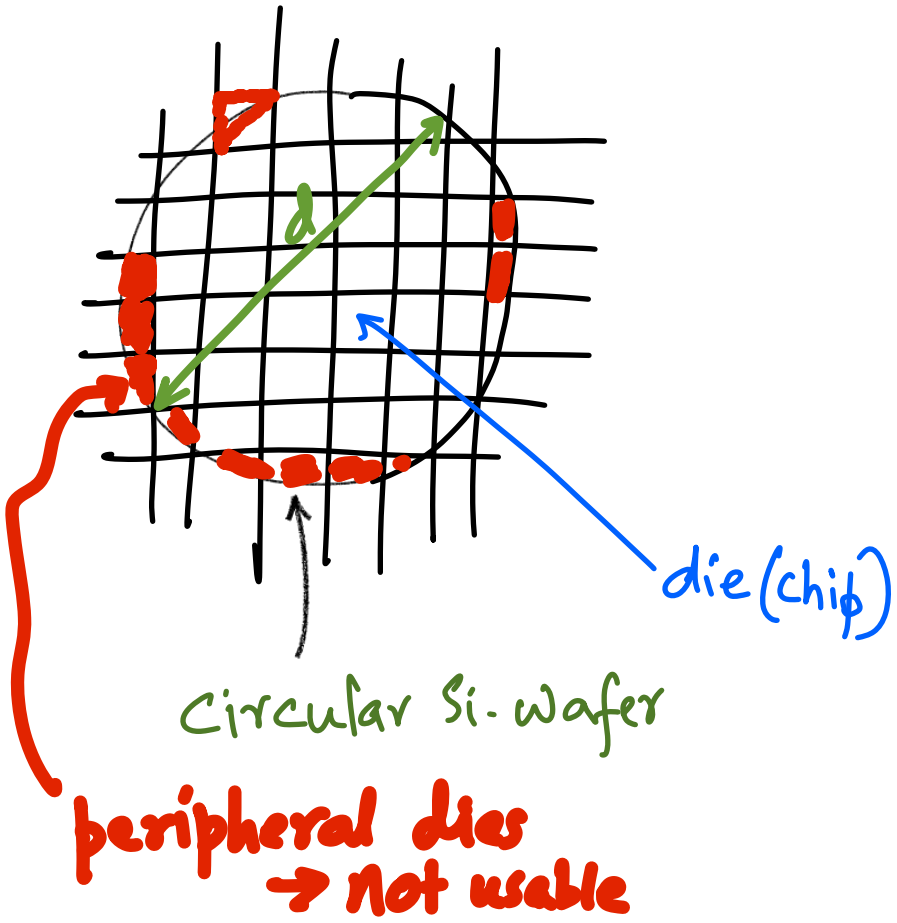
$$\text{Yield} = \frac{1}{[1 + \{(\text{Defects per unit area of chip} \times \text{chip area})/\alpha\}]^{\alpha}}$$

chip-cost ~ (die-area)^4

This is an empirical estimation of yield; α is a parameter that roughly captures the complexity of the manufacturing process

- Nonlinear relation to area and defect rate
  - Wafer cost and area are fixed
  - Defect rate determined by manufacturing process
  - Die area determined by architecture and circuit design

Dies per wafer: More accurate estimation



Hence,

wafer diameter

$$\#\text{Dies per wafer} = \frac{\pi * (d/2)^2}{\text{die.area}} - \frac{\pi \cdot d}{\sqrt{2 * \text{die area}}}$$

Example: Estimate the number of usable dies that can be obtained from a 20-cm diameter wafer, where each die is a square with 1.5 cm on each side.

Answer:

$$\#\text{Dies} = \frac{\pi * 10^2}{2.25} - \frac{\pi * 20}{\sqrt{2 * 2.25}} \approx 110$$

(assuming zero defect) ⊠

Estimating good chips per wafer ⇒ chip cost

good chips per wafer:

$$= \#\text{Dies per wafer} * \text{wafer-yield} * \beta$$

where,

$$\beta = \frac{1}{\left(1 + \left(\frac{\text{defect density} * \text{die area}}{\alpha}\right)\right)^{\alpha}}$$

$\alpha \simeq 2, 3$ for semiconductor processes.

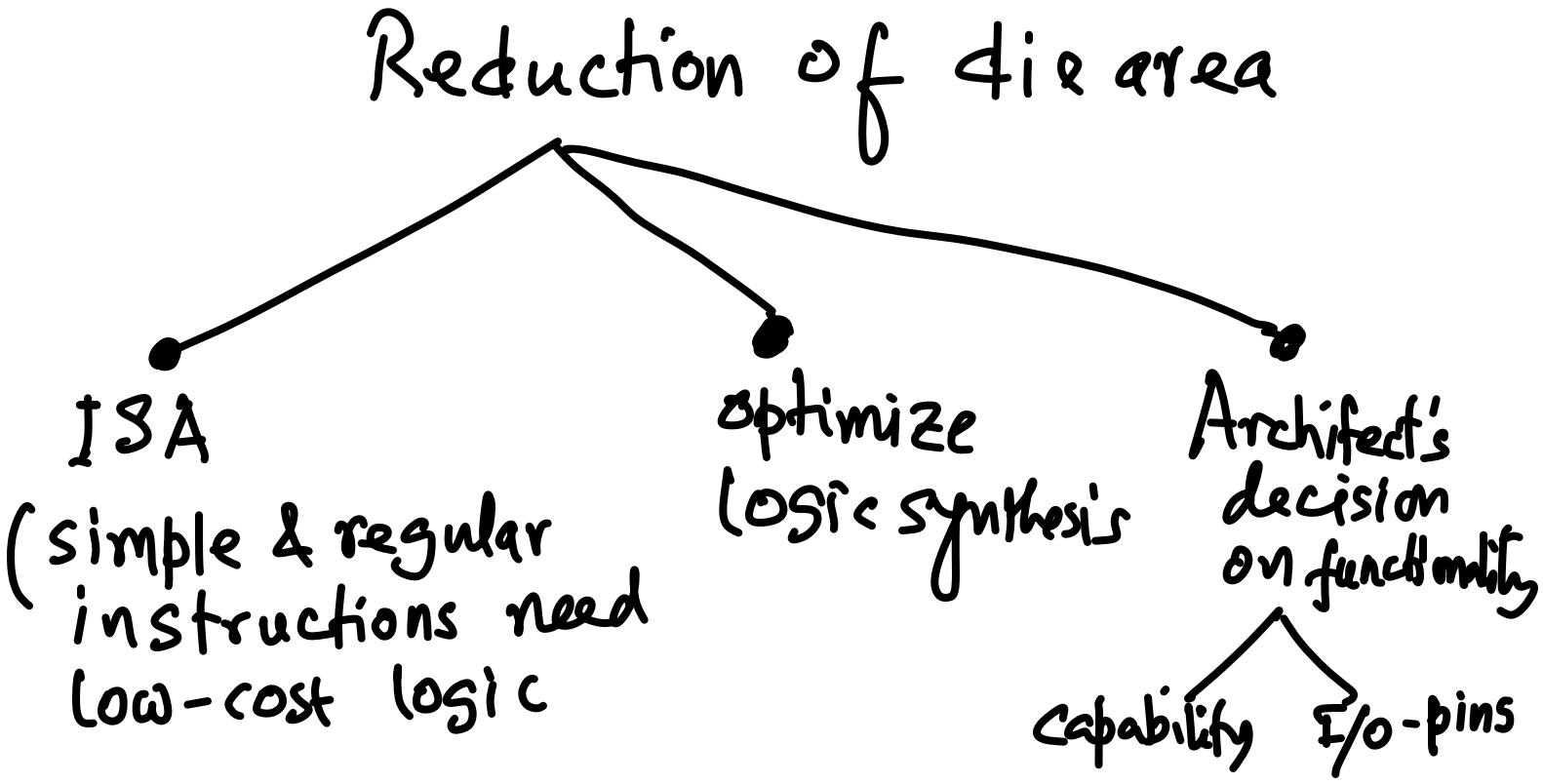
Example : Die yield

Problem: Find the die_yield for a chip that is of square shape with 1 cm on each side. Defectdensity is $0.8/cm^2$ and assume $\alpha = 3$.

Solution $\text{Die\_yield} = \left(1 + \frac{0.8 \times 1}{3}\right)^{-3} = 0.49$

$\Rightarrow 49\%$ (i.e., 51% chips are wasted)

$$\text{Cost per chip} \propto (\text{die area})^4$$



Concluding Remarks

- Cost/performance is improving
  - Due to underlying technology development
- Hierarchical layers of abstraction
  - In both hardware and software
- Instruction set architecture
  - The hardware/software interface
- Execution time: the best performance measure
- Power is a limiting factor
  - Use enhancement and parallelism to improve performance

Next Class

- Introducing MIPS Assembly Language

CS 31007 Autumn 2021

COMPUTER ORGANIZATION AND ARCHITECTURE

Announcement for Test-01

Monday, 06 September 2021, 12 noon – 2 pm

Exam will be conducted via CSE Moodle

Coverage: Processor Design Basics, von Neumann Architecture, ISA, RISC/CISC, CPU-Performance, Die Yield, MIPS Assembly Language Programming

(Material covered till 02 September 2021)

Indian Institute of Technology Kharagpur

Computer Science and Engineering