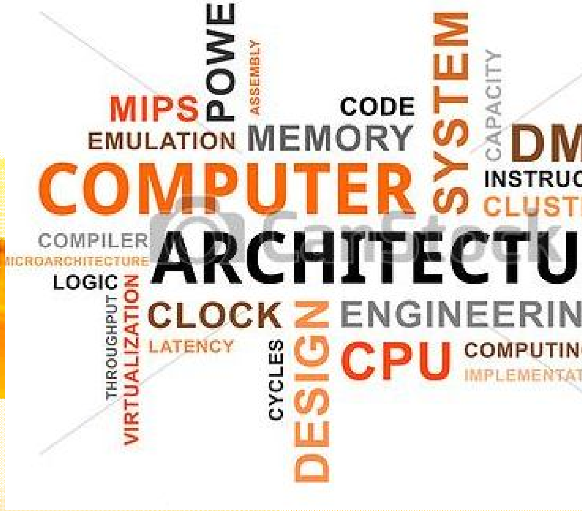
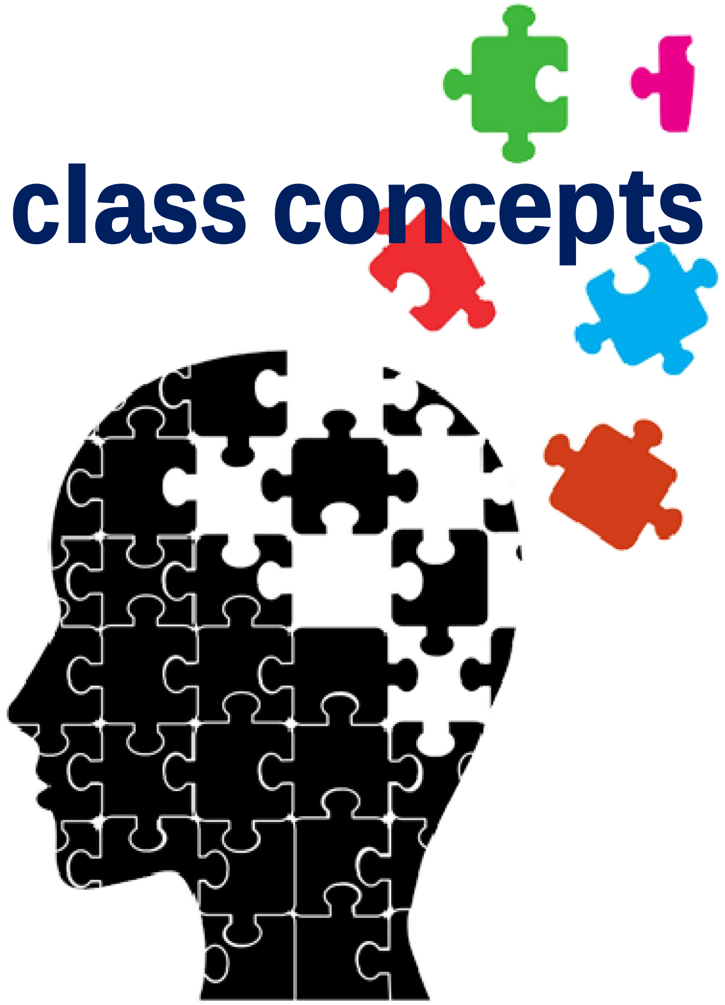
UNIT I
BASIC STRUCTURE OF COMPUTERS

Functional units – Basic operational concepts – **Bus Structures** – **Performance** – Memory locations and addresses – Memory operations – Instruction and Instruction sequencing –– Addressing modes – Assembly language – Case study : RISC and CISC Architecture.

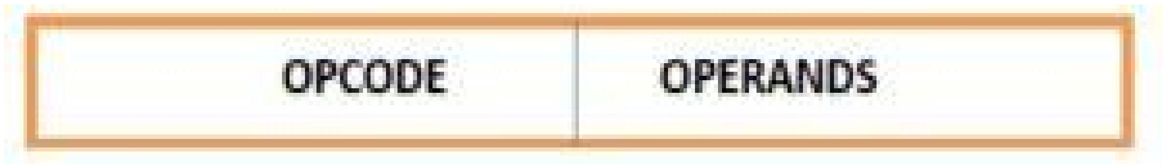


Recall the previous class concepts



Basic Operational Concepts

- Instruction consists of 2 parts



OPCODE	OPERANDS

- Example

```
ADD   LOCA, R0
```

```
Load LOCA, R1
Add R1, R0
```

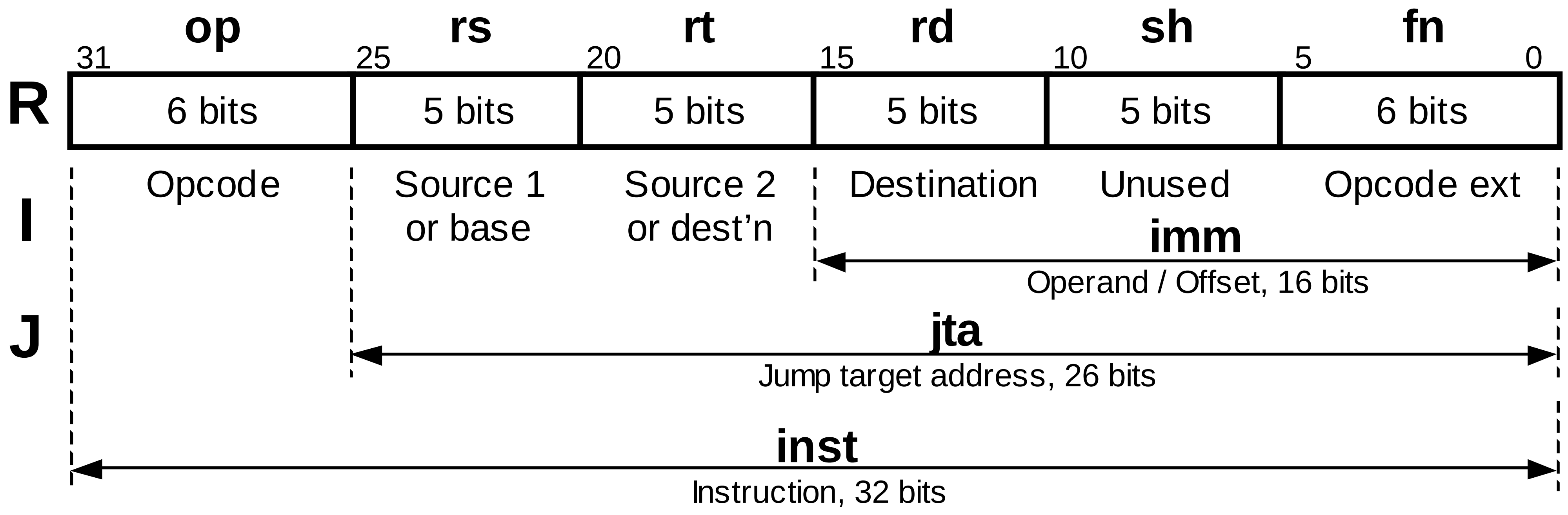
Instructions Format

op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

op	rs	rt	constant or address
6 bits	5 bits	5 bits	16 bits

op	Target Address
6 bits	26 bits

Instructions Format



Translating Arm Assembly Instructions into Machine Instructions

op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

add $t0, $s1, $s2

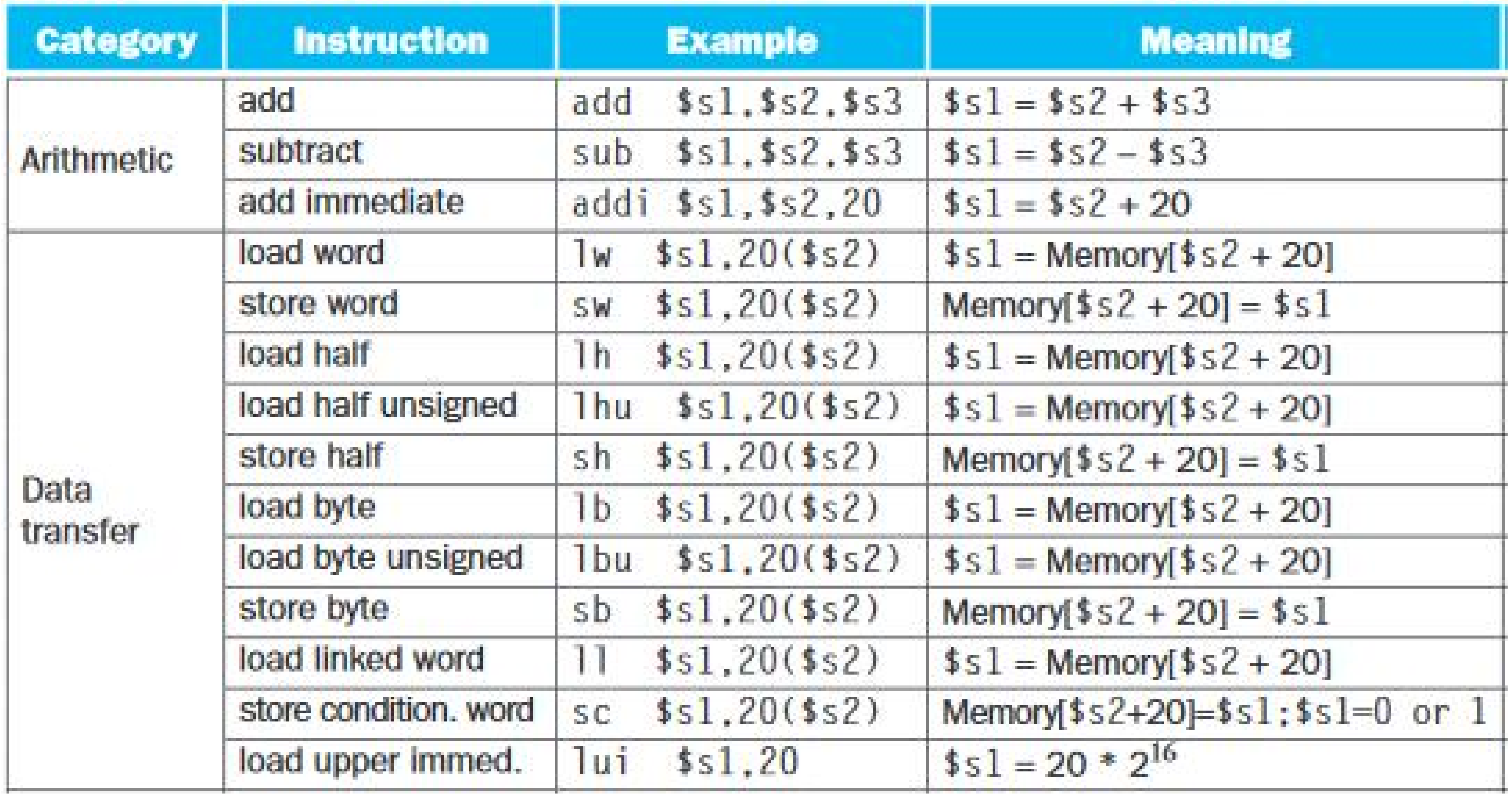
special	$s1	$s2	$t0	0	add

0	17	18	8	0	32

000000	10001	10010	01000	00000	100000

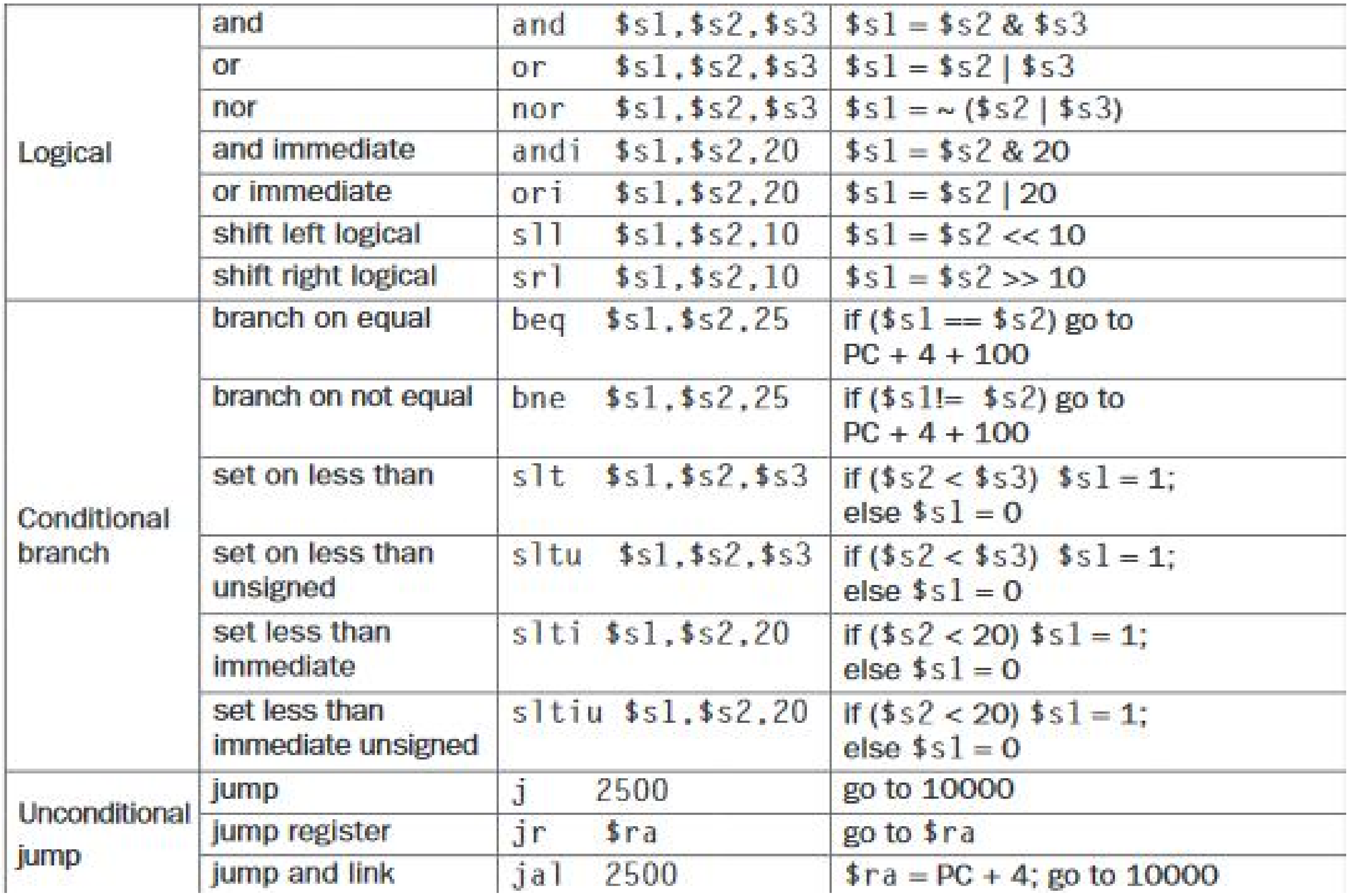
$$00000010001100100100000000100000_2 = 02324020_{16}$$

ARM Assembly Language



Category	Instruction	Example	Meaning
Arithmetic	add	`add $s1,$s2,$s3`	$s1 = $s2 + $s3
	subtract	`sub $s1,$s2,$s3`	$s1 = $s2 – $s3
	add immediate	`addi $s1,$s2,20`	$s1 = $s2 + 20
Data transfer	load word	`lw $s1,20($s2)`	$s1 = Memory[$s2 + 20]
	store word	`sw $s1,20($s2)`	Memory[$s2 + 20] = $s1
	load half	`lh $s1,20($s2)`	$s1 = Memory[$s2 + 20]
	load half unsigned	`lhu $s1,20($s2)`	$s1 = Memory[$s2 + 20]
	store half	`sh $s1,20($s2)`	Memory[$s2 + 20] = $s1
	load byte	`lb $s1,20($s2)`	$s1 = Memory[$s2 + 20]
	load byte unsigned	`lbu $s1,20($s2)`	$s1 = Memory[$s2 + 20]
	store byte	`sb $s1,20($s2)`	Memory[$s2 + 20] = $s1
	load linked word	`ll $s1,20($s2)`	$s1 = Memory[$s2 + 20]
	store condition. word	`sc $s1,20($s2)`	Memory[$s2+20]=$s1;$s1=0 or 1
	load upper immed.	`lui $s1,20`	$s1 = 20 * 2^{16}

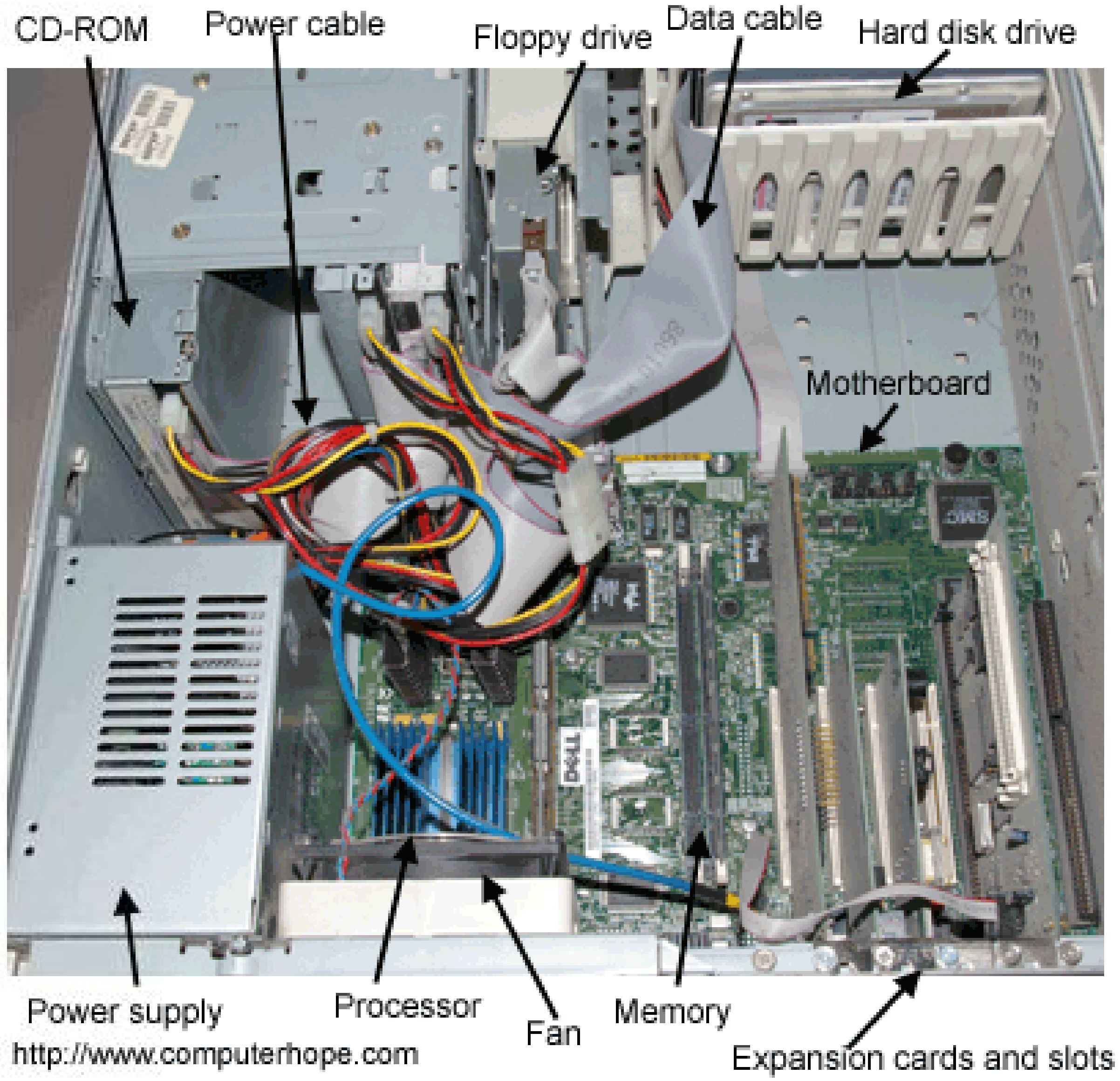
ARM Assembly Language



Logical	and	`and $s1,$s2,$s3`	$s1 = $s2 & $s3
	or	`or $s1,$s2,$s3`	$s1 = $s2 | $s3
	nor	`nor $s1,$s2,$s3`	$s1 = ~ ($s2 | $s3)
	and immediate	`andi $s1,$s2,20`	$s1 = $s2 & 20
	or immediate	`ori $s1,$s2,20`	$s1 = $s2 | 20
	shift left logical	`sll $s1,$s2,10`	$s1 = $s2 << 10
	shift right logical	`srl $s1,$s2,10`	$s1 = $s2 >> 10
Conditional branch	branch on equal	`beq $s1,$s2,25`	if ($s1 == $s2) go to PC + 4 + 100
	branch on not equal	`bne $s1,$s2,25`	if ($s1!= $s2) go to PC + 4 + 100
	set on less than	`slt $s1,$s2,$s3`	if ($s2 < $s3) $s1 = 1; else $s1 = 0
	set on less than unsigned	`sltu $s1,$s2,$s3`	if ($s2 < $s3) $s1 = 1; else $s1 = 0
	set less than immediate	`slti $s1,$s2,20`	if ($s2 < 20) $s1 = 1; else $s1 = 0
	set less than immediate unsigned	`sltiu $s1,$s2,20`	if ($s2 < 20) $s1 = 1; else $s1 = 0
Unconditional jump	jump	`j 2500`	go to 10000
	jump register	`jr $ra`	go to $ra
	jump and link	`jal 2500`	$ra = PC + 4; go to 10000

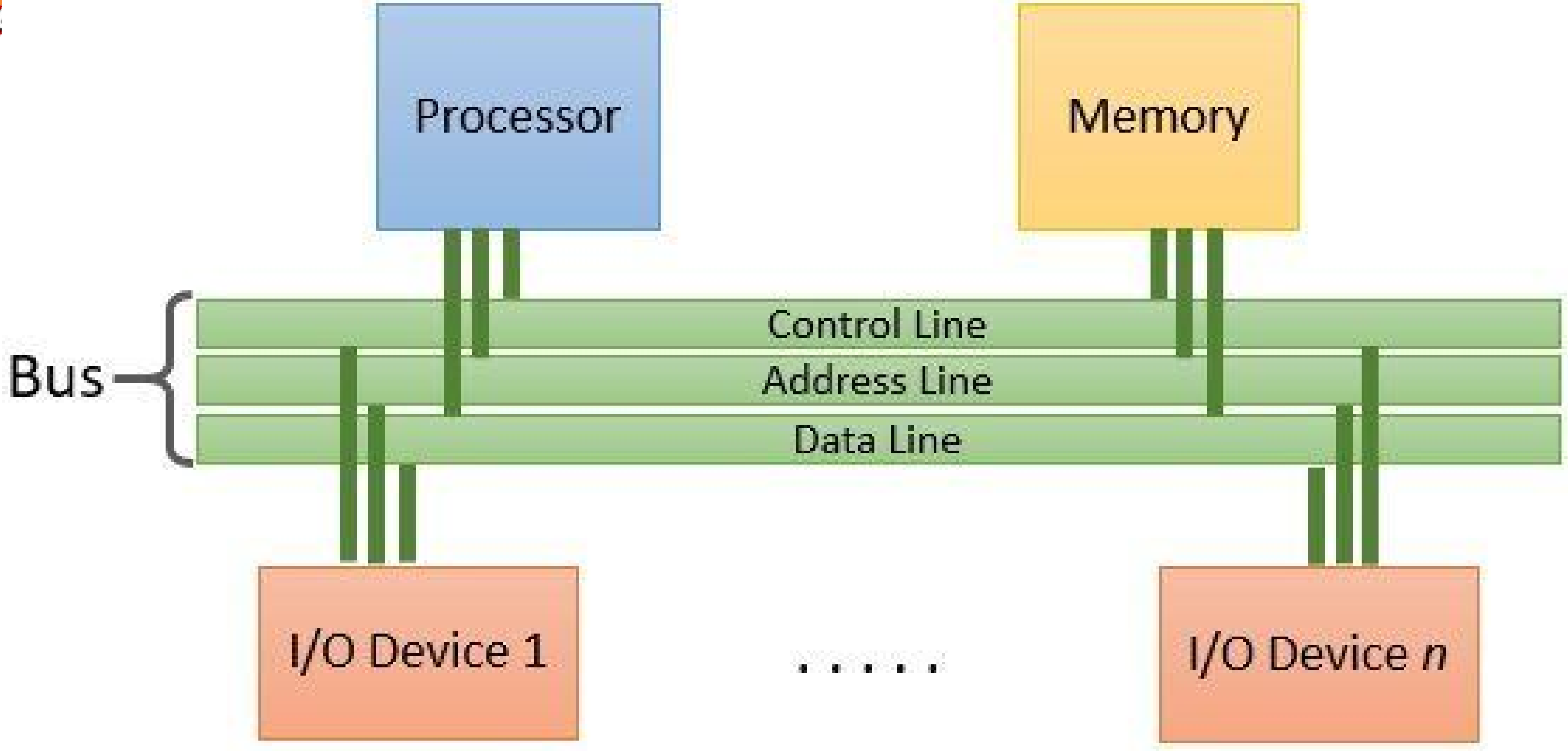
Bus Structure

Inside of a computer case



Bus Structure

- connecting the internal components of the computer
- Shared transmission medium
- **Multiple components or devices use the same bus structure to transmit the information signals to each other.**



Bus Structure

Data Bus

- Data lines coordinate in transferring the data among the system components – collectively called data bus.
- 32 lines, 64 lines, 128 lines, or even more lines.
- The number of lines present in the data bus defines the *width* of the data bus.
- Each data line is able to transfer only one bit at a time.
- The number of data lines in a data bus determines how many bits it can transfer at a time.
- The **performance** of the system also depends on the width of the data bus.

Address Bus

- determines the source or destination of the data present on the data bus. - Together is referred to as **address bus.**
- The number of address lines in the address bus determines its width
- width of the address bus determines the **memory capacity of the system.**
- The content of address lines is also used for addressing I/O ports.
- The **higher-order bits determine the bus module** and the **lower ordered bits determine the address of memory locations or I/O ports.**

Control Lines

- to control the use and access of data and address lines.
- The control signal consists of the **command and timing information.**

Control Lines

- **Memory Write:** This command causes the data on the data bus to be placed over the addressed memory location.
- **Memory Read:** This command causes the data on the addressed memory location to be placed on the data bus.
- **I/O Write:** The command over this control line causes the data on the data bus to be placed over the addressed I/O port.
- **I/O Read:** The command over this control line causes the data from the addressed I/O port to be placed over the data bus.
- **Transfer ACK:** This control line indicates the data has been received from the data bus or is placed over the data bus.

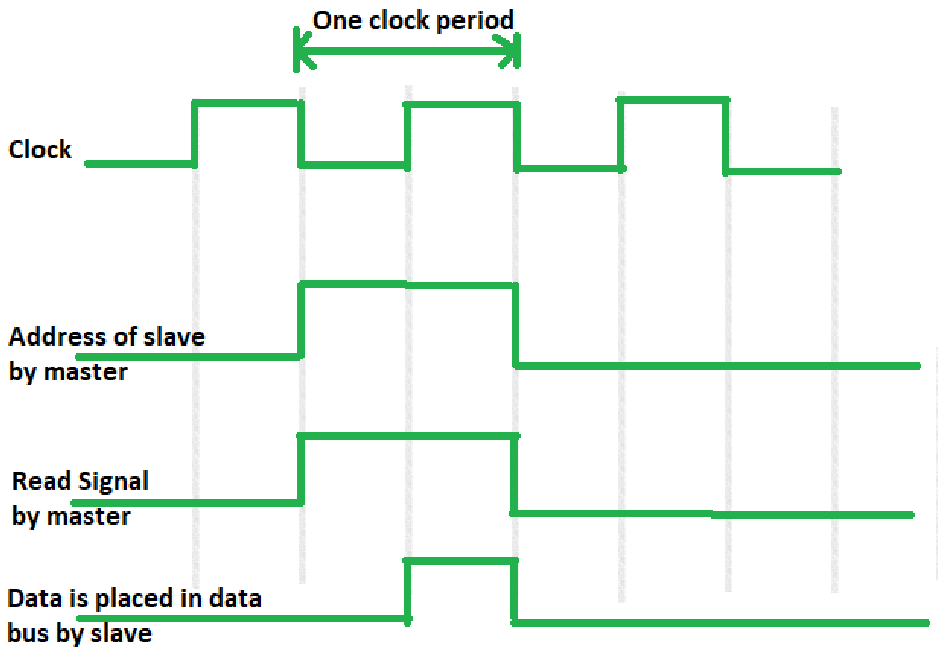
Control Lines

- **Bus Request:** This control line indicates that the component has requested control over the bus.
- **Bus Grant:** This control line indicates that the bus has been granted to the requesting **Interrupt** component.
- **Request:** This control line indicates that interrupts are pending.
- **Interrupt ACK:** This control line provides acknowledgment when the pending interrupt is serviced.
- **Clock:** This control line is used to synchronize the operations.
- **Reset:** The bit information issued over this control line initializes all the modules.

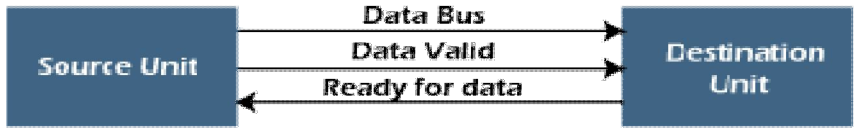
Timing in Bus

- An synchronous bus works at a fixed clock rate whereas an
- Asynchronous bus data transfer is not dependent on a fixed clock.

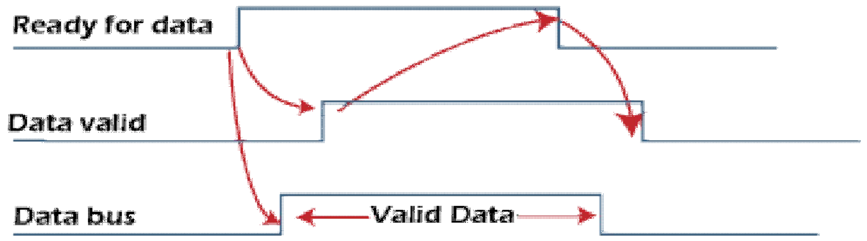
Timing in Bus


One clock period
Clock
Address of slave by master
Read Signal by master
Data is placed in data bus by slave

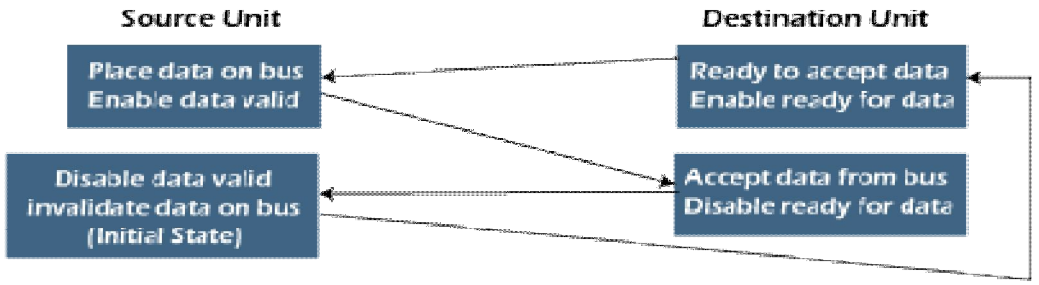
Timing diagram for Synchronous Read Operation



(a) Block Diagram

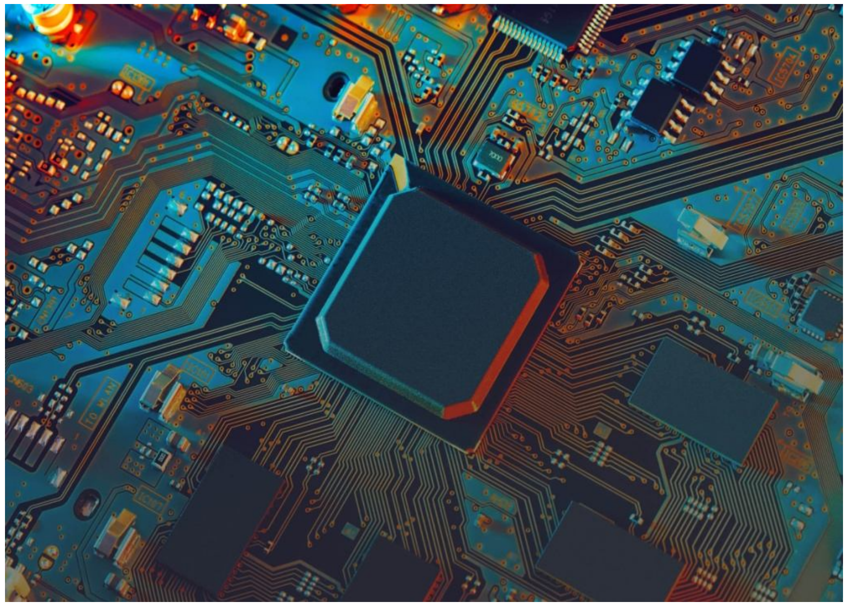


(b) Timing Diagram



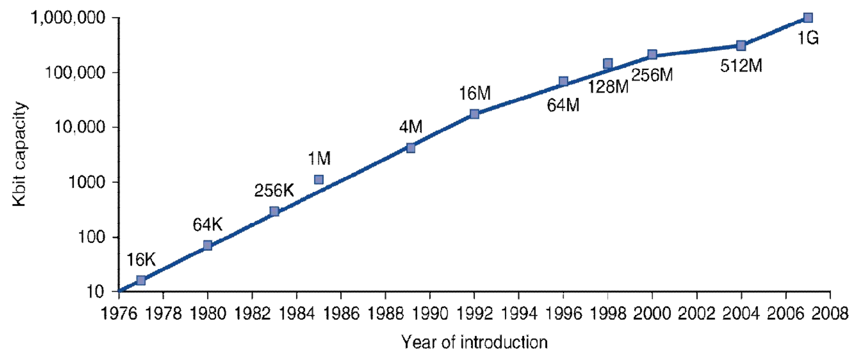
(c) Sequence Diagram (Sequence of events)

Technology & Performance



Technology Trends

- Electronics technology continues to evolve
  - Increased capacity and performance
  - Reduced cost



DRAM capacity

Year	Technology	Relative performance/cost
1951	Vacuum tube	1
1965	Transistor	35
1975	Integrated circuit (IC)	900
1995	Very large scale IC (VLSI)	2,400,000
2005	Ultra large scale IC	6,200,000,000

Response Time and Throughput

Response time : How long it takes to do a task

Throughput : Total work done per unit time

Example : tasks/transactions/… per hour

How are response time and throughput affected by

- Replacing the processor with a faster version?
- Adding more processors?

Decreasing a response Time

Relative Performance

Define Performance = 1/Execution Time

"X is *n* time faster than Y"

$$\begin{aligned}&\text{Performance}_X / \text{Performance}_Y \\ &= \text{Execution time}_Y / \text{Execution time}_X = n\end{aligned}$$

If computer A runs a program in 10 seconds and computer B runs the same program in 15 seconds, how much faster is A than B?

Performance ratio n = 15 /10 = 1.5

A is therefore 1.5 times faster than B.

Measuring Execution Time

Elapsed time : Total response time - Determines system performance

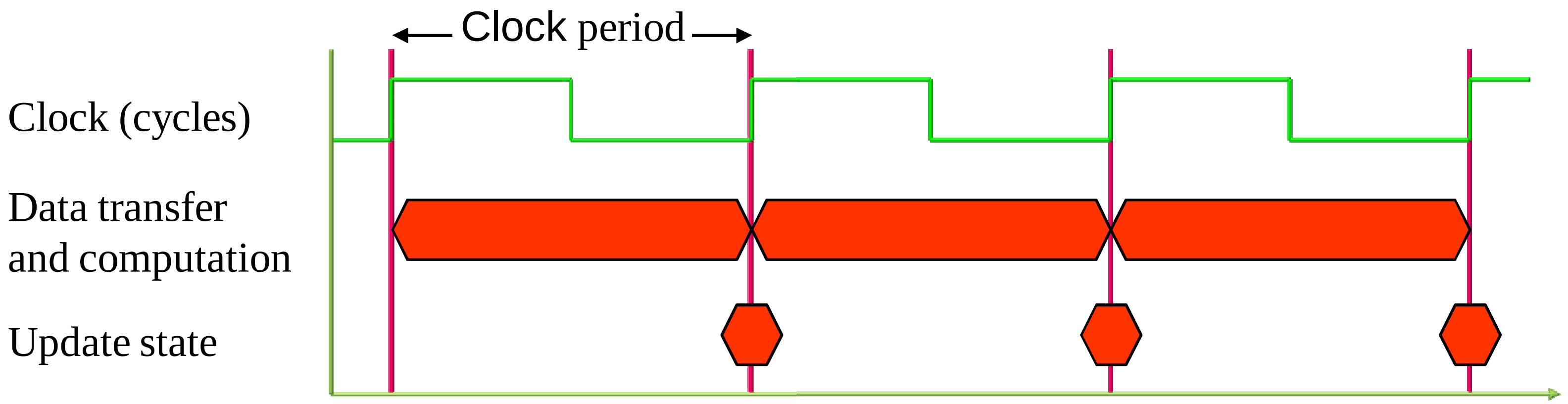
Example : Processing, I/O, OS overhead, idle time

CPU time: Time spent processing a given job . Comprises user CPU time and system CPU time

Different programs are affected differently by CPU and system performance

CPU Clocking

Operation of digital hardware governed by a constant-rate clock



Clock period: duration of a clock cycle

Clock frequency (rate): cycles per second

CPU Performance & its Factor

$$\boxed{\begin{aligned}\text{CPU Execution Time} &= \text{CPU Clock Cycles} \times \text{Clock Cycle Time} \\ &= \frac{\text{CPU Clock Cycles}}{\text{Clock Rate}}\end{aligned}}$$

Performance improved by

- Reducing number of clock cycles
- Increasing clock rate
- Hardware designer must often trade off clock rate against cycle count

Improving Performance

Formula

Our favorite program runs in **10 seconds on computer A**, which has a **2 GHz clock**. We are trying to help a computer designer build a **computer, B, which will run this program in 6 seconds.** The designer has determined that a substantial increase in the clock rate is possible, but this increase will affect the rest of the CPU design, causing **computer B to require 1.2 times as many clock cycles as computer A for this program**. What clock rate should we tell the designer to target?

FORMULA

$$\text{CPU time}_A = \frac{\text{CPU clock cycles}_A}{\text{Clock rate}_A}$$

$$10 \text{ seconds} = \frac{\text{CPU clock cycles}_A}{2 \times 10^9 \frac{\text{cycles}}{\text{second}}}$$

20×10^{9} cycles

$$\text{CPU time}_B = \frac{1.2 \times \text{CPU clock cycles}_A}{\text{Clock rate}_B}$$

$$6 \text{ seconds} = \frac{1.2 \times 20 \times 10^9 \text{ cycles}}{\text{Clock rate}_B}$$

Clock Rate $_B$ = 4 GHz

To run the program in 6 seconds , B must have twice the clock rate of A

Instruction Performance

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

Suppose we have two implementations of the same instruction set architecture. Computer A has a clock cycle time of 250 ps and a CPI of 2.0 for some program, and computer B has a clock cycle time of 500 ps and a CPI of 1.2 for the same program. Which computer is faster for this program and by how much?

CPU clock cycles $_A$ = I x 2.0
CPU clock cycles $_B$ = I x 1.2

Compute the CPU time for each computer

CPU time $_A$ = CPU clock cycles $_A$ × Clock cycle time

CPU time $_A$ = 500 x I ps
CPU time $_B$ = 600 x I ps

$$\frac{\text{CPU performance}_A}{\text{CPU performance}_B} = \frac{\text{Execution time}_B}{\text{Execution time}_A} = $$ 600 x I ps / 500 x I ps

Computer A is 1.2 times as fast as computer B

Classical CPU Performance Equation

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

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

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

Comparing Code Segments

A compiler designer is trying to decide between two code sequences for a particular computer. The hardware designers have supplied the following facts:

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

Sequence 1: IC = 5

Clock Cycles = 10
(2×1 + 1×2 + 2×3)

CPI = clock cycle / IC = 10/5 = 2.0

Sequence 2: IC = 6

Clock Cycles = 9
(4×1 + 1×2 + 1×3)

CPI = 9/6 = 1.5

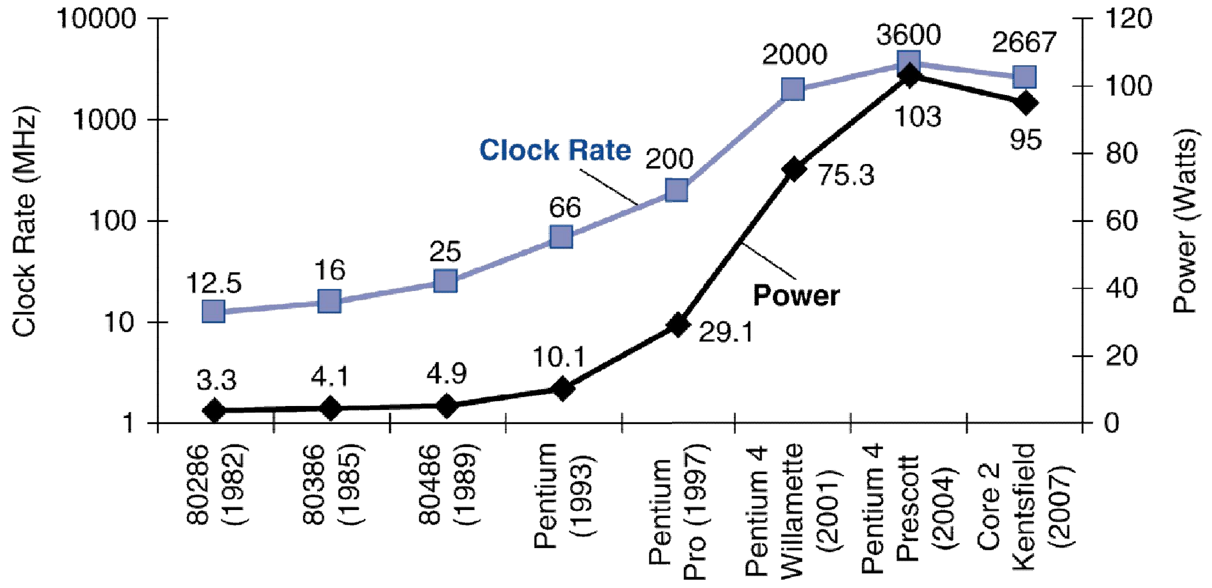
Performance Summary

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

$$\text{CPU Time} = s\frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Clock cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Clock cycle}}$$

- Performance depends on
  - Algorithm: affects IC, possibly CPI
  - Programming language: affects IC, CPI
  - Compiler: affects IC, CPI
  - Instruction set architecture: affects IC, CPI, T_c

Power Trends



In CMOS IC technology

Complementary Metal-Oxide Semiconductor

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

×30

5V → 1V

×1000

TEXT BOOK

Carl Hamacher, Zvonko Vranesic and Safwat Zaky, “Computer Organization”, McGraw-Hill, 6th Edition 2012.

REFERENCES

1. David A. Patterson and John L. Hennessey, “Computer organization and design”, MorganKauffman ,Elsevier, 5th edition, 2014.
2. William Stallings, “Computer Organization and Architecture designing for Performance”, Pearson Education 8th Edition, 2010
3. John P.Hayes, “Computer Architecture and Organization”, McGraw Hill, 3rd Edition, 2002
4. M. Morris R. Mano “Computer System Architecture” 3rd Edition 2007
5. David A. Patterson “Computer Architecture: A Quantitative Approach”, Morgan Kaufmann; 5th edition 2011

THANK YOU