Foundations of Computer Systems (CS541)

Arijit Mondal

Syllabus

- Introduction to computer architecture
- Instruction set architecture
- CPU design

Books to be followed

- Computer Organization and Design: The Hardware/Software Interface – David A. Patterson, John L. Hennessy
- Computer Organization and Architecture – William Stallings
- Computer Architecture: A Quantitative approach – David A. Patterson, John L. Hennessy

Evaluation policy

- Midsem: 30%
- Endsem: 50%
- Assignments/Quiz: 20%

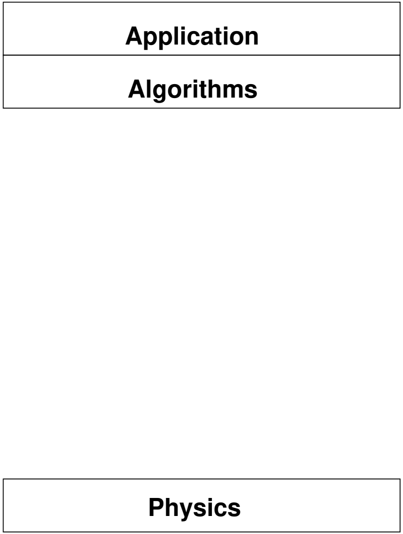
Introduction

Abstraction of computing systems

Application

Physics

Abstraction of computing systems



Abstraction of computing systems

Application
Algorithms
Programming language

Physics

Abstraction of computing systems

Application

Algorithms

Programming language

Operating systems

Physics

Abstraction of computing systems

Application
Algorithms
Programming language
Operating systems
Instruction set architecture

Physics

Abstraction of computing systems

Application

Algorithms

Programming language

Operating systems

Instruction set architecture

Microarchitecture

Physics

Abstraction of computing systems

- Application
- Algorithms
- Programming language
- Operating systems
- Instruction set architecture
- Microarchitecture
- Register transfer level

- Physics

Abstraction of computing systems

- Application
- Algorithms
- Programming language
- Operating systems
- Instruction set architecture
- Microarchitecture
- Register transfer level
- Gates

- Physics

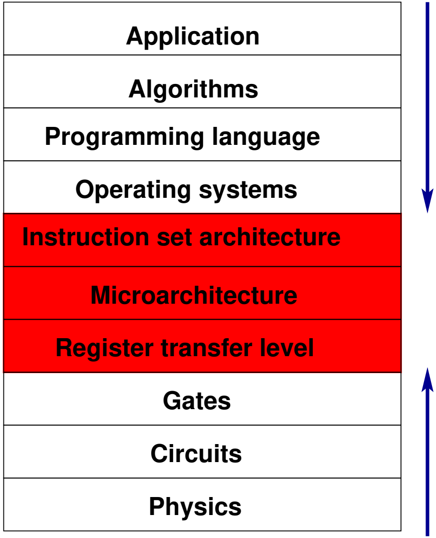
Abstraction of computing systems

Application
Algorithms
Programming language
Operating systems
Instruction set architecture
Microarchitecture
Register transfer level
Gates
Circuits
Physics

Abstraction of computing systems

- Application
- Algorithms
- Programming language
- Operating systems
- Instruction set architecture
- Microarchitecture
- Register transfer level
- Gates
- Circuits
- Physics

Abstraction of 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

Abstraction

- Abstraction helps us to deal with complexity
  - Hide lower level details
- Instruction set architecture
  - Hardware/Software interface
- Application binary interface
  - ISA plus system software
- Implementation
  - The details underlying and interface

Architecture vs Microarchitecture

- 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
  - 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

Levels of Program Code

- High level language
  - Easy to code & debug
  - Close to problem domain
  - Provides productivity

```
g = h * i ;
k = j + i ;
g = h[1] ;
```

Levels of Program Code

- High level language
  - Easy to code & debug
  - Close to problem domain
  - Provides productivity

```
g = h * i ;
k = j + i ;
g = h[1] ;
```

compiler

Levels of Program Code

- High level language
  - Easy to code & debug
  - Close to problem domain
  - Provides productivity

```
g = h * i ;
k = j + i ;
g = h[1] ;
```

compiler

- Assembly language
  - Textual representation of instructions

```
MUL R0, R1, R2 ;
ADD R3, R4, R2
LDR R3, [R0,#4]
```

Levels of Program Code

- High level language
  - Easy to code & debug
  - Close to problem domain
  - Provides productivity
- Assembly language
  - Textual representation of instructions

```
g = h * i ;
k = j + i ;
g = h[1] ;
```

compiler

```
MUL R0, R1, R2 ;
ADD R3, R4, R2
LDR R3, [R0,#4]
```

assembler

Levels of Program Code

- High level language
  - Easy to code & debug
  - Close to problem domain
  - Provides productivity
- Assembly language
  - Textual representation of instructions
- Hardware language
  - Binary data
  - Encoded instruction and data

```
g = h * i ;
k = j + i ;
g = h[1] ;
```

compiler

```
MUL R0, R1, R2 ;
ADD R3, R4, R2
LDR R3, [R0,#4]
```

assembler

```
0000101101000010101
1010101111100101010
1010101011110000011
```

Components of a Computer

- Same components for all kind of computers
  - Server, Desktop, Embedded systems

Components of a Computer

- Same components for all kind of computers
  - Server, Desktop, Embedded systems
- Input-Output support

Components of a Computer

- Same components for all kind of computers
  - Server, Desktop, Embedded systems
- Input-Output support
  - User interface devices – Keyboard, mouse, display
  - Storage devices – Hard disk, CD/DVD, Flash
  - Network adapters for communicating with others

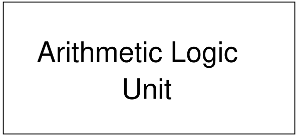
Components of a Computer

- Same components for all kind of computers
  - Server, Desktop, Embedded systems
- Input-Output support
  - User interface devices – Keyboard, mouse, display
  - Storage devices – Hard disk, CD/DVD, Flash
  - Network adapters for communicating with others
- Inside the computer

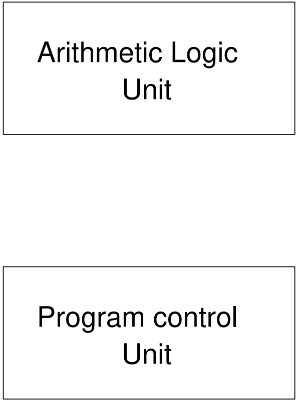
Components of a Computer

- Same components for all kind of computers
  - Server, Desktop, Embedded systems
- Input-Output support
  - User interface devices – Keyboard, mouse, display
  - Storage devices – Hard disk, CD/DVD, Flash
  - Network adapters for communicating with others
- Inside the computer
  - Arithmetic logic unit (ALU)
  - Program control unit
  - Memory
  - Datapath

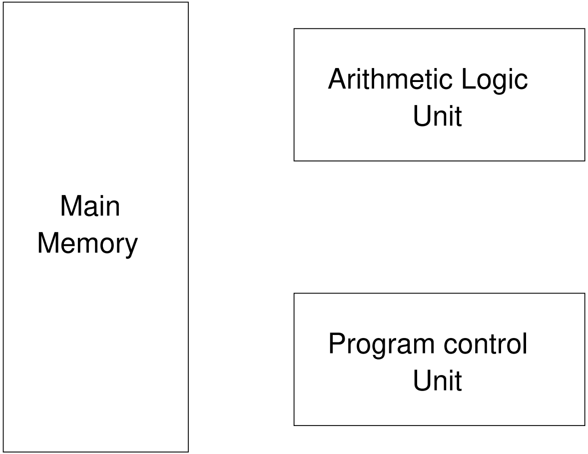
IAS Computer



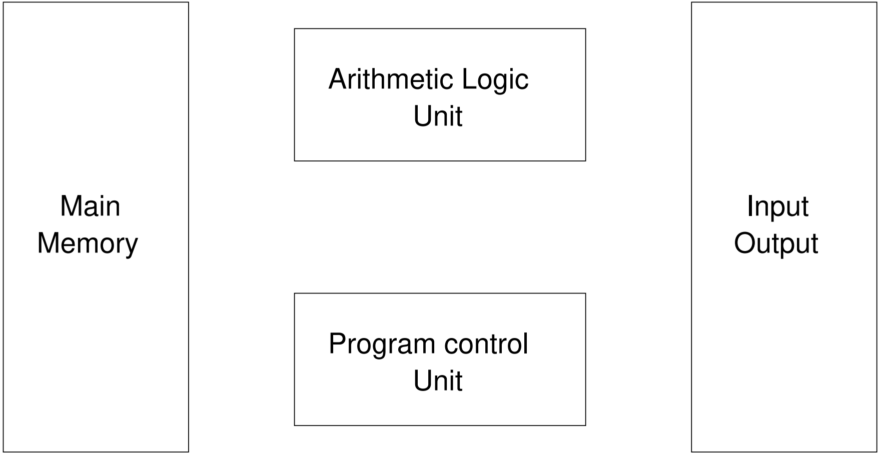
IAS Computer



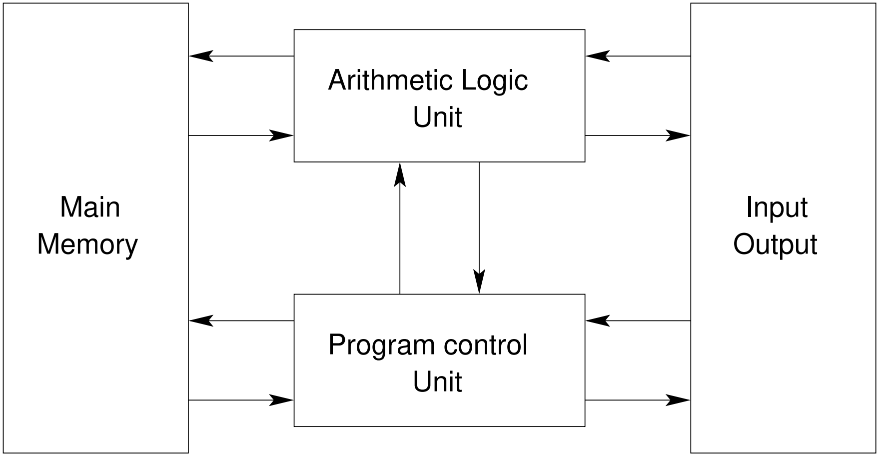
IAS Computer



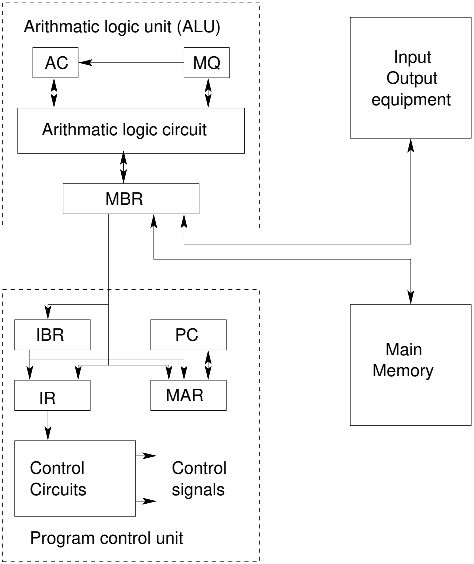
IAS Computer



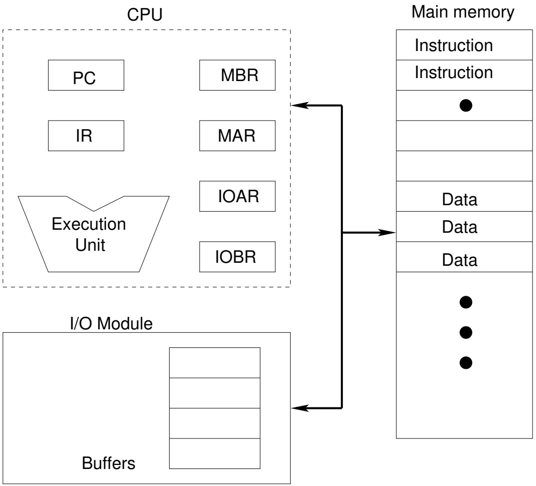
IAS Computer



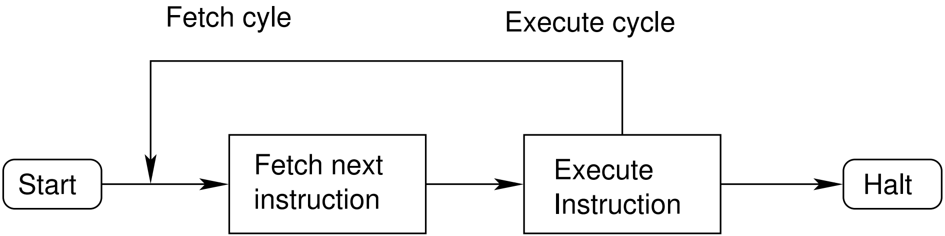
Expanded structure of IAS Computer



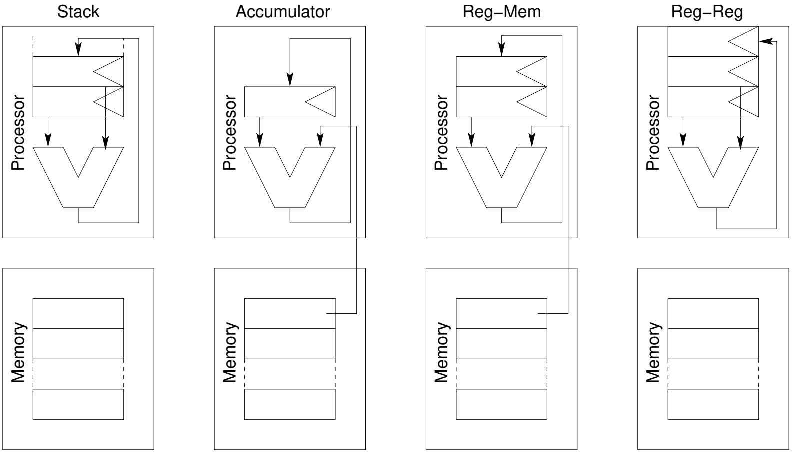
Top level view of computer



Basic instruction cycle



Machine Model



Stack	Accumulator	Reg-Mem	Reg-Reg
push A	load A	load r1, A	load r1,A
push B	add B	add r3,r1, B	load r2,B
add	store C	store r3, C	add r3,r2,r1
pop C			store r3,c

Understanding Performance

- Algorithms
  - Determines number of operation executed
- Programing language, compiler, architecture
  - Determine number of machine instructions is executed per operation
- Processor and memory systems
  - Determines how fast instructions are executed
- I/O systems
  - Determines how fast I/O operations are performed

Performance

- Response time
  - How long it takes to finish a task
- Throughput
  - Total workdone per unit time (eg. task/transaction/per hour)
- Dependency of response time and throughput
  - Replacing the processor with a faster version?
  - Adding more processors?

Relative performance

- Performance is defined as 1/Execution time
- X is n times faster than Y
  - $\text{Performance}_X/\text{Performance}_Y = \text{Execution time}_Y/\text{Execution time}_X = \text{n}$
- Example: Time taken to run a program
  - 10ns in machine X and 15ns in machine Y
  - $\text{Execution time}_Y/\text{Execution time}_X = 15/10 = 1.5$
  - So, X is 1.5 times faster than Y

Measuring performance

- Elapsed time (Wall clock time)
  - Total time to complete a task including I/O, memory access, disk access, OS overhead, etc.
- CPU time
  - The time the CPU spends computing this task
  - Does not include I/O time, other jobs' share
  - Can be further subdivided – *user* CPU time and *system* CPU time
- Different programs are affected differently by CPU and system performance

CPU clocking

- Operation is controlled by a constant rate clock
  - Clock period is duration of clock cycle. (eg. 300ns $= 300 \times 10^{-9}$s)
  - Clock frequency is cycles per second. (eg. 4GHz $= 4 \times 10^{9}$Hz)
  - Clock period $= 1/$Clock frequency

CPU Time

- $\text{CPU time} = \text{CPU clock cycles} \times \text{Clock period} = \dfrac{\text{CPU clock cycle}}{\text{Clock frequency}}$
- Performance can be improved by
  - Reducing number of clock cycle
  - Increasing clock frequency
  - Hardware designer must trade off clock frequency against cycle count

Example

- Machine A: Run time 10s, Clock speed 2GHz
- Design a new machine (B say)
  - Run time is 6s
  - Faster clock require 1.2 times more clock cycles compared to A
- Clock frequency for machine B?

Instruction count and CPI

- Clock cycles $=$ Instruction count $\times$ Cycles per instruction
- CPU time $=$ Instruction count $\times$ CPI $\times$ Clock period $= \dfrac{\text{Instruction count} \times \text{CPI}}{\text{Clock frequency}}$
- Instruction count for a program
  - Depends on ISA, compiler, program
- Average cycles per instruction
  - Determined by CPU hardware
  - Different instruction have different CPI
  - Average CPI is affected by instruction mix

CPI example

- Machine A: Clock period - 250ps, CPI - 2.0
- Machine B: Clock period - 500ps, CPI - 1.2
- Same set of instructions
- Which is faster?

CPI in more detail

- Different instructions take different cycles
- Clock cycles $= \sum_{i=1}^{n} (\text{CPI}_i \times \text{Instruction count}_i)$
- Weighted average CPI $=$

$$\frac{\text{Clock cycle}}{\text{Instruction count}} = \sum_{i=1}^{n} \left(\text{CPI}_i \times \frac{\text{Instruction count}_i}{\text{Instruction count}} \right)$$

CPI example

Instruction	A	B	C
CPI for instruction	1	2	3
IC in Sequence 1	2	1	2
IC in Sequence 2	4	1	1

- Which code sequence executes the most instructions?
- Compute average CPI for each sequence.

Performance summary

- CPU Time $= \dfrac{\text{Instructions}}{\text{Program}} \times \dfrac{\text{Clock cycles}}{\text{Instruction}} \times \dfrac{\text{second}}{\text{Clock cycle}}$
- Performance depends on

Performance summary

- $\text{CPU Time} = \dfrac{\text{Instructions}}{\text{Program}} \times \dfrac{\text{Clock cycles}}{\text{Instruction}} \times \dfrac{\text{second}}{\text{Clock cycle}}$
- Performance depends on
  - Algorithm -

Performance summary

- $\text{CPU Time} = \dfrac{\text{Instructions}}{\text{Program}} \times \dfrac{\text{Clock cycles}}{\text{Instruction}} \times \dfrac{\text{second}}{\text{Clock cycle}}$
- Performance depends on
  - Algorithm - Affects IC, possibly CPI

Performance summary

- CPU Time $= \dfrac{\text{Instructions}}{\text{Program}} \times \dfrac{\text{Clock cycles}}{\text{Instruction}} \times \dfrac{\text{second}}{\text{Clock cycle}}$
- Performance depends on
  - Algorithm - Affects IC, possibly CPI
  - Programming language -

Performance summary

- CPU Time $= \dfrac{\text{Instructions}}{\text{Program}} \times \dfrac{\text{Clock cycles}}{\text{Instruction}} \times \dfrac{\text{second}}{\text{Clock cycle}}$
- Performance depends on
  - Algorithm - Affects IC, possibly CPI
  - Programming language - Affects IC, CPI

Performance summary

- CPU Time $= \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Clock cycles}}{\text{Instruction}} \times \frac{\text{second}}{\text{Clock cycle}}$
- Performance depends on
  - Algorithm - Affects IC, possibly CPI
  - Programming language - Affects IC, CPI
  - Compiler -

Performance summary

- CPU Time $= \dfrac{\text{Instructions}}{\text{Program}} \times \dfrac{\text{Clock cycles}}{\text{Instruction}} \times \dfrac{\text{second}}{\text{Clock cycle}}$
- Performance depends on
  - Algorithm - Affects IC, possibly CPI
  - Programming language - Affects IC, CPI
  - Compiler - Affects IC, CPI

Performance summary

- CPU Time $= \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Clock cycles}}{\text{Instruction}} \times \frac{\text{second}}{\text{Clock cycle}}$
- Performance depends on
  - Algorithm - Affects IC, possibly CPI
  - Programming language - Affects IC, CPI
  - Compiler - Affects IC, CPI
  - Instruction set architecture -

Performance summary

- $\text{CPU Time} = \dfrac{\text{Instructions}}{\text{Program}} \times \dfrac{\text{Clock cycles}}{\text{Instruction}} \times \dfrac{\text{second}}{\text{Clock cycle}}$
- Performance depends on
  - Algorithm - Affects IC, possibly CPI
  - Programming language - Affects IC, CPI
  - Compiler - Affects IC, CPI
  - Instruction set architecture - Affects IC, CPI, Clock period

Performance: Power

- Power $\propto$ Capacitive load $\times$ Voltage^2 $\times$ Frequency
- Suppose a new CPU has the following
  - 85% of capacitive load of old CPU
  - 15% reduction in voltage, 15% reduction in frequency
    - $\frac{P_{new}}{P_{old}} = \frac{0.85 \times C_{old} \times (V_{old} \times 0.85)^2 \times F_{old} \times 0.85}{C_{old} \times (V_{old})^2 \times F_{old}} = 0.85^4 = 0.52$
  - Constraints
    - Further reduction in voltage may not be possible
    - Dissipation of heat

MIPS as performance metric

- MIPS: Millions of Instruction Per Second
  - Does not account for
    - Differences in ISAs in computers
    - Differences in complexity between instructions
- $\text{MIPS} = \dfrac{\text{Instruction count}}{\text{Execution time} \times 10^6} = \dfrac{\text{Instruction count}}{\frac{\text{Instruction count} \times \text{CPI}}{\text{Clock frequency}} \times 10^6}$

  $= \dfrac{\text{Clock frequency}}{\text{CPI} \times 10^6}$
- CPI varies between programs on a given CPU

Multiprocessors

- Multicore multiprocessors
  - More than one processor per chip
- Requires explicit parallel programming
  - Instruction level parallelism
    - Hardware executes multiple instructions simultaneously
    - Hidden from programmer
  - Hard to do
    - Programming for performance
    - Load balancing
    - Optimizing communication and synchronization

Conclusion

- Cost/performance is improving
  - Due to underlying technology development
- Hierarchical layer of abstraction
  - In both hardware and software
- Instruction set architecture
  - The Hardware/Software interface
- Execution time – measure of performance
- Power is a limiting factor
  - Use parallelism to improve performance