COMP2300-COMP6300-ENGN2219
Computer Architecture

Convener: Shoaib Akram
shoaib.akram@anu.edu.au

Australian
National
University

Shoaib Akram

School of Computing, (Jan 2020 –), Senior Lecturer

Ph.D., 2019 (Ghent, Belgium)

shbakram.github.io

Shoaib Akram

I am an assistant professor at the Australian National University, Canberra, in the School of Computing. I obtained my Ph.D. degree in Computer Science and Engineering from Ghent University (Belgium). My doctoral advisor was Lieven Eeckhout. I also hold an M.S. in Electrical and Computer Engineering from the University of Illinois at Urbana-Champaign.

I am interested in computer systems in general and memory and storage systems in particular. My research approach intersects the boundary between the hardware, OS, and language runtime.

I lead the **V**ertically **I**ntegrated **C**omputer **S**ystems (**VICS**) research group.

I serve on the reviewing committees of top-tier computer architecture, systems, and programming language conferences.

Curriculum vitae
Google Scholar
My PhD Dissertation

Recent News

[**TACO 2024**] Adnan Hasnat (undergraduate) has a paper accepted on exploiting hybrid memory for accelerating enterprise search engines

[**TOPLAS 2024**] Iacovos (PhD student) has a paper accepted at TOPLAS

[**HPCA 2025**] Serving on the HPCA PC

[**ISCA 2024**] Serving on the ISCA PC

[**ISPASS 2024**] Serving on the ISPASS PC

[**ISMM 2023**] Paper accepted at ISMM (Aditya's undergraduate honours thesis!)

[**MICRO 2023**] Serving on the MICRO PC

[**ASPLOS 2023**] TeraHeap paper will appear at ASPLOS 2023

Teaching

- COMP2300 (2023 – 2025), ENGN2219 (2021 – 2025)
- COMP2310 (2022 – 2023) Systems, Networks, & Concurrency
- Special Topics in Computer Systems, Computer Microarchitecture, 2022 – 2023
- Special Topics in Computer Systems: Parallel Computer Architecture (**2025**)

Research

- **PhD Research**
  - Memory management and computer architecture
  - **PhD Thesis:** Optimizing the language runtime for hardware heterogeneity

- **Current Research**
  - Efficient computer systems for data-intensive applications
    - Information retrieval and databases
    - Integrating new memory and storage technologies

My Group's Recent Work

Vertically **I**ntegrated **C**omputer **S**ystems (**VICS**) research group

Jack Kilrain
Ph.D. Student

Adnan Hasnat
Ph.D. Student

And more …..

[**TACO 2024**] A. Hasnat and S. Akram, SPIRIT: Scalable and Persistent In-Memory Indices for Real-Time Search, ACM Transactions on Architecture and Code Optimization (TACO), 2024.
PDF pre-print
PDF slides

[**TOPLAS 2024**] Iacovos G. Kolokasis, Giannos Evdorou, Shoaib Akram, Anastasios Papagiannis, Foivos Zakkak, Christos Kozanitis, Polyvios Pratikakis, and Angelos Bilas, TeraHeap: Exploiting Flash Storage for Mitigating DRAM Pressure in Managed Big Data Frameworks, ACM Transactions on Programming Languages and Systems (TOPLAS), 2024.
PDF pre-print

[**SOSP 2023 (Poster)**] Iacovos G. Kolokasis, Shoaib Akram, Foivos Zakkak, Polyvios Pratikakis, and Angelos Bilas, DynaHeap: Dynamic Division of DRAM between Heterogeneous Managed Heaps, Symposium on Operating System Principles (SOSP), 2023.
PDF

[**ISMM 2023**] Aditya Chilukuri and Shoaib Akram, Analyzing and Improving the Scalability of In-Memory Indices for Managed Search Engines, ACM SIGPLAN International Symposium on Memory Management (ISMM), 2023.
PDF
PDF slides

[**ASPLOS 2023**] Iacovos G. Kolokasis, Giannos Evdorou, Shoaib Akram, Anastasios Papagiannis, Foivos Zakkak, Christos Kozanitis, Polyvios Pratikakis, and Angelos Bilas, TeraHeap: Reducing Memory Pressure in Managed Big Data Frameworks, Architectural Support for Programming Languages and Operating Systems (ASPLOS), 2023.
PDF version of paper
PDF slides

[**ISPASS 2022**] W. Liu, W. Heirman, S. Eyerman, S. Akram, and L. Eeckhout, Scale-Model Architectural Simulation, IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), 2022.
PDF version of paper

My Group's Recent Work

Vertically **I**ntegrated **C**omputer **S**ystems (**VICS**) research group

Honours Students (ANU)

- Aditya Chilukuri (Graduation: semester 1, 2022) Bachelor of Advanced Computing
- Hou (Harris) Loi (Graduation: semester 2, 2022) Bachelor of Advanced Computing
- Jack Kilrain (Graduation: semester 2, 2022) Bachelor of Software Engineering
- Junming Zhao (Graduation: semester 1, 2023) Bachelor of Engineering and Advanced Computing
  $\longrightarrow$ ***University Medal for Academic Excellence***
- Anson Thai (Graduation: semester 1, 2024) Bachelor of Advanced Computing
  $\longrightarrow$ ***University Medal for Academic Excellence***
- Adnan Hasnat (Graduation: semester 2, 2025) Bachelor of Advanced Computing
  $\longrightarrow$ ***University Medal for Academic Excellence***
  $\longrightarrow$ ***Erin Brent Computer Science Prize for best honours result***
- Kshama Patel (Graduation: semester 2, 2025) Bachelor of Advanced Computing
- Itay Yarom (Graduation: semester 2, 2026) Bachelor of Advanced Computing

PhD Advisor

Lieven Eeckhout

I am a Full Professor (gewoon hoogleraar BOF-ZAP) at Ghent University, Belgium, in the Department of Electronics and Information Systems (ELIS). I obtained my PhD degree from Ghent University in 2002. My research interests include computer architecture, with specific emphasis on performance evaluation and modeling, workload characterization, dynamic resource management, CPU/GPU microarchitecture, and sustainability. I'm an IEEE and ACM Fellow.

Contact information

ELIS – Ghent University
Technologiepark-Zwijnaarde 126
9052 Ghent, Belgium
Tel:+32-9-264.34.58, Fax: +32-9-264.35.94
Email: Lieven.Eeckhout @ UGent.be

My Research Group: PerformanceLab

PhD students:

Benyamin Eslami
Jaime Roelandts
Hossein SeyyedAghaei

PerformanceLab Alumni

Ajeya Naithani (PhD 2019, postdoc) – TU Eindhoven (assistant professor)
Shiqing Zhang (PhD 2023)
Karthik Lakshminarasimhan (PhD 2023)
Wenjie Liu (PhD 2022)
Lu Wang (PhD 2020)
Xia Zhao (PhD 2019)
Shoaib Akram (PhD 2019) – ANU (assistant professor)

PhD Advisor

Awards

- Hall of Fame ISCA (2019), MICRO(2024), HPCA (2022)
- 2024 IEEE CAL Best Paper Award for *'RIP Geomean Speedup, Use Equal-Work (or Equal-Time) Harmonic Mean Speedup Instead'*
- Distinguished Artifact Evaluation Award at ASPLOS 2024 for *'FOCAL: A First-Order Carbon Model to Assess Processor Sustainability'*
- Selection of '*Decoupled Vector Runahead' [MICRO 2023]* as an IEEE Micro Top Pick 2023
- Selection of *'TEA: Time-Proportional Event Analysis' [ISCA 2023]* as an IEEE Micro Top Pick 2023
- IEEE/ACM MICRO 2023 Best Paper Award for *'Decoupled Vector Runahead'*
- BenchCouncil Achievement Award 2023
- "ISCA@50 25-Year Retrospective: 1996-2020" selection of *'Scheduling Heterogeneous Multi-Cores through Performance Impact Estimation (PIE)'* from ISCA 2012
- 2022 Best Paper Award at IEEE Computer Architecture Letters for *'A First-Order Model to Assess Computer Architecture Sustainability'*
- ACM Fellow for contributions in computer architecture performance analysis and modeling (class of 2021)
- Selection of '*Vector Runahead'* as an IEEE Micro Top Pick 2021
- IEEE Fellow for contributions in computer architecture performance analysis and modeling (class of 2018)
- OOPSLA Most Influential Paper Award for Statistically Rigorous Java Performance Evaluation (published at OOPSLA 2007)
- 2017 Maurice Wilkes Award for outstanding contributions to computer architecture performance analysis and modeling
- ISPASS 2013 Best Paper Award
- Laureaat van de Academie in de Klasse van de Technische Wetenschappen, 2010
- Selection of '*Per-Thread Cycle Accounting for SMT Processors'* as an IEEE Micro Top Pick 2010 for *most significant research paper in computer architecture based on novelty and long-term impact*
- Selection of '*A Top-Down Approach to Architecting CPI Component Performance Counters'* as an IEEE Micro Top Pick 2007 for being *most relevant to industry and significant in contribution to the field of computer architecture*
- IBM Belgium Prize for Informatics for PhD thesis in 2003

PhD Advisor

2024

Scalar Vector Runahead
J. Roelandts, A. Naithani, S. Ainsworth, T. M. Jones, and L. Eeckhout
IEEE/ACM International Symposium on Microarchitecture (MICRO), Nov 2024

STAR: Sub-Entry Sharing-Aware TLB for Multi-Instance GPU
B. Li, Y. Wang, T. Wang, L. Eeckhout, J. Yang, A. Jaleel, X. Tang
IEEE/ACM International Symposium on Microarchitecture (MICRO), Nov 2024

Sustainable Hardware Specialization
P. Dangi, T. K. Bandara, S. Sheikhpour, T. Mitra, and L. Eeckhout
IEEE/ACM International Conference on Computer-Aided Design (ICCAD), Oct 2024

Sustainable High-Performance Instruction Selection for Superscalar Processors
S. Sheikhpour, D. Metz, E. Jellum, M. Själander, and L. Eeckhout
IEEE/ACM International Conference on Computer-Aided Design (ICCAD), Oct 2024

R.I.P. Geomean Speedup, Use Equal-Work (or Equal-Time) Harmonic Mean Speedup Instead **[Best of IEEE CAL 2024, presented at HPCA 2025]**
L. Eeckhout
IEEE Computer Architecture Letters, Vol 23, Issue 1, pp. 78-82, Jan-June 2024

FOCAL: A First-Order Carbon Model to Assess Processor Sustainability **[Distinguished Artifact Evaluation Award]**
L. Eeckhout
ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), Vol. 2, pp. 401-415, April 2024

GPU Scale-Model Simulation
H. SeyyedAghaei, M. Naderan-Tahan, and L. Eeckhout
IEEE International Symposium on High-Performance Computer Architecture (HPCA), pp. 1125-1140, March 2024

Sustainable Computer Systems **[Keynote]** [Video recording]
L. Eeckhout
HiPEAC Conference, Jan 2024

Systems @ ANU

- **Computer Architecture**
  - Parallel Computer Architecture
  - Systems, Networks, & Concurrency
  - Compilers (+ Language Runtimes – 2026 S2)
  - Operating Systems
  - Computer Networks
  - Parallel Systems

Systems @ ANU

- **Why do we "still" invest in systems?**

- **Advances in computer systems is the key enabler of modern applications (**Big Data, AI/ML, social media platforms, search engines, communications, mobile apps, drones, among others**)**

- Renewed demand for more compute power and balancing it against cost and energy efficiency, and sustainability

Logistics

- Course webpage: https://comp.anu.edu.au/courses/comp2300/

- **Lectures (**on the website**)**
  - Lecture slides
  - Lecture videos are available via Echo360

- **Policies**
  - General conduct, assignment submissions, support, management, grading, late submissions

- **Resources**
  - Frequently asked questions
  - Writing design documents
  - Stuff needed to finish the assignments

Communication

- We will use **Ed** for
  - Announcements
  - Addressing your concerns
  - Answering your questions

- Students are added automatically
  - If not send an email to **comp2300@anu.edu.au**

Communication

- The course email is an alternative form of communication

  - **comp2300@anu.edu.au**

Tutorials

- Labs are an important component of this course (starting week 2)

- Handouts should be **posted on the website**

- **Relationship to assessment**
  - 20% (Assignment 1)
  - Midterm and Final exam questions (ARM assembly)

- **Excellent tutors with deep technical knowledge of the subject**
  - Almost all of them have taken multiple courses in the systems & architecture specialization

Lectures

- Go well beyond what is covered in the tutorials
- Exams (80%) mostly based on lectures
- Tutorials teach you how to build and program a simple computer to manage complexity
- Lectures will systematically build up from simple to a very complex, state-of-the-art, computer

Assignment Submission

- Assignment submissions are handled via Gitlab
  - You will learn about it in the labs
  - Make a habit of using Gitlab properly!
  - Push often, always pull the latest

- Extensions
  - https://comp.anu.edu.au/courses/comp2300/policies/#extensions

Assessment

Assessment Scheme

Here are the assessments for COMP2300/6300/ENGN2219. **Note**: more specific release and due dates will be provided at a later date.

All times are in **Canberra Time**.

Assessment Item	Release Date	Due Date	Weighting
Assignment 1	Week 6	Week 9	20%
Midsemester Exam	20th April 2026	-	30%
Final Exam	TBD	-	50%

Assessment Pages

No assignments have been released yet.

Textbook

- Freely (?) available online (check Ed or course webpage)
- I will post the chapters/sections on the Lectures page after the lecture

COMP2300

- Asks a question
  - Let’s give it a shot!

COMP2300

How does a computer **actually** work?

COMP2300

How does a computer work **under the hood**?

COMP2300

How does a computer work **under the hood**?

High abstraction level

Low abstraction level

COMP2300

How does a computer **work** **under the hood**?

Traditional Block Diagram of Computer

High abstraction level

Low abstraction level

Let’s be more specific ….

COMP2300

How does a computer perform a useful task?

COMP2300

How does a computer perform a wide variety of useful tasks?

COMP2300

How do we make a computer perform a wide variety of useful tasks?

How do we make electrons perform a wide variety of useful tasks?

How do we make electrons execute a wide variety of useful programs?

How do we make electrons do the work?

Problem Statement: “Save the planet”

Devices

How do we make electrons do the work?

Problem Statement: “Save the planet”

The Algorithm

Devices

How do we make electrons do the work?

Problem Statement: “Save the planet”

The Algorithm

Program in a High-Level Language

Devices

How do we make electrons do the work?

Problem Statement: “Save the planet”

The Algorithm

Program in a High-Level Language

Instruction Set Architecture (ISA)

Devices

How do we make electrons do the work?

Problem Statement: “Save the planet”

The Algorithm

Program in a High-Level Language

Instruction Set Architecture (ISA)

Microarchitecture

Devices

How do we make electrons do the work?

Problem Statement: “Save the planet”
The Algorithm
Program in a High-Level Language
Instruction Set Architecture (ISA)
Microarchitecture
Circuits
Devices

How do we make electrons do the work?

Problem Statement: “Save the planet”
The Algorithm
Program in a High-Level Language
Instruction Set Architecture (ISA)
Microarchitecture
Circuits
Devices

How do we make electrons do the work?

- Using a *sequence of systematic transformations* developed over many decades

- Each step must be studied and improved for the whole **"compute stack"** to work/operate efficiently

Transformation Hierarchy

- We call the steps of the process: **Levels of transformation** OR **transformation hierarchy**

- At each level of the stack, we have choices
  - **Many languages:** Java, Python, Ruby, Scala, C++, C#
  - **ISAs:** ARM, x86, SPARC, PowerPC, RISC-V
  - **Microarchitectures:** Intel, AMD, IBM, Apple

- If we ignore any of the steps, then we cannot
  - Make the best use of computer systems
  - Build the **best** system for a set of programs

Problem

Algorithm

Program

Architecture

micro-arch

circuits

devices

Computer Architecture

Problem Statement: “Save the planet”

The Algorithm

Program in a High-Level Language

Compiler and Third-Party Libraires/Binaries

Operating System

Instruction Set Architecture (ISA)

Microarchitecture

Computer Architecture

Circuits

Devices

Hardware/Software Interface

Problem Statement: "Save the planet"

Software	The Algorithm
	Program in a High-Level Language
	Compiler and Third-Party Libraires/Binaries
	Operating System

Systems software

ISA = Hw/Sw boundary/interface

Instruction Set Architecture (ISA)

	Microarchitecture
	Circuits
Hardware	Devices

ISA vs. Microarchitecture

- **ISA: Specification** of a set of definite instructions that the computer can carry out
  - All computers (CPUs/microprocessors) perform the same set of basic instructions
  - ADD, MULTIPLY, DIVIDE, MOVE, BRANCH
  - Two manufacturers might differ in *which set* of basic instructions
- **Microarchitecture: Implementation** of the ISA using circuits

ISA vs. Microarchitecture

- **ISA:** What the driver needs to know as he sits inside the automobile to make the automobile carry out the driver's wishes
- If the middle pedal (brake) is pressed, the car stops
- Steering wheel, ignition key, the gears, windshield wipers

- **Microarchitecture:** What goes underneath the hood
- Different cost/performance tradeoffs
- Some are turbocharged. Some have disc brakes. Some cost a million $. Some are more fuel efficient than others

Defining Computer Architecture

- **Traditional view**
  - Design of a computer system including *instruction set architecture* and *organization* (*microarchitecture*)
  - What we will cover in this course
  - Traditionally what computer architects do

- **But modern realities require architects to take a more expanded view of building and optimizing computer systems**

Modern Computer Architecture

- **Traditional view**
  - Design of a computer system including instruction set architecture and organization (microarchitecture) and ISA

- **Modern (expanded) view**
  - **Cross-layer design:** co-design across the transformation hierarchy *or* compute stack
  - From algorithms to devices

Transformation Hierarchy

Approach: Cross Layer Design

To achieve the highest efficiency, performance, robustness:

we must take the expanded view

of computer architecture

Broad research
spanning applications, systems, software, logic, circuits
with architecture at the center

Approach: Cross Layer Design

To achieve the highest efficiency, performance, robustness:

we [illegible] the expanded view

o[illegible]r architecture

Co-design across the hierarchy:
Algorithms to devices

Specialize as much as possible within the design goals

Broad research
spanning applications, systems, software, logic, circuits
with architecture at the center

Modern Computer Architecture

- is the science and art of designing computing platforms (hardware, interface, system software, and programming model)

- to achieve a set of design goals
  - E.g., highest performance on earth on workloads X, Y, Z
  - E.g., longest battery life at a form factor that fits in your pocket with cost < $$$ AUD
  - E.g., best average performance across all known workloads at the best performance/cost ratio
  - …

  - Designing a supercomputer is different from designing a smartphone → But, many fundamental principles are similar

Different Platforms, Different Goals

Different Platforms, Different Goals

Different Platforms, Different Goals

Different Platforms, Different Goals

Different Platforms, Different Goals

Figure 3. TPU Printed Circuit Board. It can be inserted in the slot for an SATA disk in a server, but the card uses PCIe Gen3 x16.

Figure 4. Systolic data flow of the Matrix Multiply Unit. Software has the illusion that each 256B input is read at once, and they instantly update one location of each of 256 accumulator RAMs.

Jouppi et al., "In-Datacenter Performance Analysis of a Tensor Processing Unit", ISCA 2017.

Different Platforms, Different Goals

New ML applications (vs. TPU3):

- Computer vision
- Natural Language Processing (NLP)
- Recommender system
- Reinforcement learning that plays Go

250 TFLOPS per chip in 2021
vs 90 TFLOPS in TPU3

1 ExaFLOPS per board

https://spectrum.ieee.org/tech-talk/computing/hardware/heres-how-googles-tpu-v4-ai-chip-stacked-up-in-training-tests

Different Platforms, Different Goals

Different Platforms, Different Goals

- ML accelerator: 260 mm^2, 6 billion transistors, 600 GFLOPS GPU, 12 ARM 2.2 GHz CPUs.
- Two redundant chips for better safety.

Many Other AI/ML Chips in Production

- Alibaba
- Amazon
- Facebook
- Google
- Huawei
- Intel
- Microsoft
- NVIDIA
- Tesla
- Many Others and Many Startups are Building Their Own Chips…

- **Many More to Come…**

2,560-DPU Processing-in-Memory

Benchmarking a New Paradigm: An Experimental Analysis of a Real Processing-in-Memory Architecture

JUAN GÓMEZ-LUNA, ETH Zürich, Switzerland
IZZAT EL HAJJ, American University of Beirut, Lebanon
IVAN FERNANDEZ, ETH Zürich, Switzerland and University of Malaga, Spain
CHRISTINA GIANNOULA, ETH Zürich, Switzerland and NTUA, Greece
GERALDO F. OLIVEIRA, ETH Zürich, Switzerland
ONUR MUTLU, ETH Zürich, Switzerland

Many modern workloads, such as neural networks, databases, and graph processing, are fundamentally memory-bound. For such workloads, the data movement between main memory and CPU cores imposes a significant overhead in terms of both latency and energy. A major reason is that this communication happens through a narrow bus with high latency and limited bandwidth, and the low data reuse in memory-bound workloads is insufficient to amortize the cost of main memory access. Fundamentally addressing this *data movement bottleneck* requires a paradigm where the memory system assumes an active role in computing by integrating processing capabilities. This paradigm is known as *processing-in-memory (PIM)*.

Recent research explores different forms of PIM architectures, motivated by the emergence of new 3D-stacked memory technologies that integrate memory with a logic layer where processing elements can be easily placed. Past works evaluate these architectures in simulation or, at best, with simplified hardware prototypes. In contrast, the UPMEM company has designed and manufactured the first publicly-available real-world PIM architecture. The UPMEM PIM architecture combines traditional DRAM memory arrays with general-purpose in-order cores, called *DRAM Processing Units (DPUs)*, integrated in the same chip.

This paper provides the first comprehensive analysis of the first publicly-available real-world PIM architecture. We make two key contributions. First, we conduct an experimental characterization of the UPMEM-based PIM system using microbenchmarks to assess various architecture limits such as compute throughput and memory bandwidth, yielding new insights. Second, we present *PrIM (Processing-In-Memory benchmarks)*, a benchmark suite of 16 workloads from different application domains (e.g., dense/sparse linear algebra, databases, data analytics, graph processing, neural networks, bioinformatics, image processing), which we identify as memory-bound. We evaluate the performance and scaling characteristics of PrIM benchmarks on the UPMEM PIM architecture, and compare their performance and energy consumption to their state-of-the-art CPU and GPU counterparts. Our extensive evaluation conducted on two real UPMEM-based PIM systems with 640 and 2,556 DPUs provides new insights about suitability of different workloads to the PIM system, programming recommendations for software designers, and suggestions and hints for hardware and architecture designers of future PIM systems.

https://arxiv.org/pdf/2105.03814.pdf

2,560-DPU Processing-in-Memory System

Computer Architecture Today

New demands
from the top
(Look Up)

Many new issues and opportunities
at the bottom
(Look Down)

Our Major Goals

- Understand the basics and principles of design in computer architecture
- Learn how a general purpose computer (CPU) works
- Evaluate tradeoffs of different designs and ideas
- Implement your own designs
- Enable you to come up with novel out-of-box ideas

Two Recurring Themes

- The notion of abstraction

- Hardware + software = computer system

The Notion of Abstraction

- **Abstraction:** Know components from a high level of detail

No human (programmer) can track 10 billion elements. **Computer systems work because of abstraction!**

Apple M1 Chip
Billions of transistors
All working in parallel

The Notion of Abstraction

- Abstraction: View the world from a higher level
- Focus on the important aspects
- It is a way to enhance productivity and efficiency

The Notion of Abstraction

- We will raise the level of abstraction in this course every couple weeks
- We start from the ground (up) because this is how computers have evolved
- Each layer in the *transformation hierarchy* is an abstraction layer
- The ability to un-abstract or deconstruct is an important one

Problem

Algorithm

Program

Architecture

micro-arch

circuits

devices

Hardware versus Software

- Hardware
  - CPU, memory, storage device, disk, SSD, network card, USB device, AI accelerator, FPGA, PLA, motherboard, PCI express bus, SATA drive

- Software
  - Programs, operating systems, compilers, virtual machines, device drivers,

- **One view:** Ok to be an expert at one of these

Hardware versus Software

- Hardware and Software
  - Two interacting parts of the computer system

- Knowing the capabilities and limitations of each leads to better overall systems

Hardware-Software Interplay

APACHE Spark

mongoDB

Hardware-Software Interplay

- Many important and emerging applications are data-intensive services
- It is easier and convenient to produce data than to process, analyze, and store it
- This trend of producing data at high velocity is driving new applications and correspondingly novel hardware

Why are there so many hardware choices?

Hardware: General Purpose vs. Special Purpose

Common building block of computers

General Purpose
CPUs

Apple M1

GPUs

Nvidia GTX 1070

FPGAs

Xilinx Spartan

Special Purpose
ASICs

Cerebras WSE-2

Flexible: Can execute any program
Easy to program & use
Not the best performance & efficiency
C/C++/Java/...

Efficient & High performance
(Usually) Difficult to program & use
Inflexible: Limited set of programs
Domain-specific, special purpose languages

General Purpose vs. Special Purpose

General Purpose

Flexible: Can work with any bolt

Easy to use

Not the best fit, results or efficiency

Special Purpose

Efficient & High performance

(Usually) Difficult to use

Inflexible: Only for fitting bolts

Two Key Ideas

- Key components of a computer are the same

- All computers can compute the same things

Idea 1: Key components are same

Council Bluffs, Iowa data center, Google (115, 000 sq. feet)

Self-flying nano drone
94 milli-watts

Research server for my students with special memory & storage devices

A Canonical Computer System

- **Most computer systems can be viewed as below**
  - Key resources: CPU, memory, I/O devices, storage
  - CPU (processor/microprocessor) does the actual computation
  - Processor can access memory much faster than storage

Idea 1: Key components are same

- **Key Idea 1:** All computer systems, big or small, have a few fundamental components
  - **Microprocessor** (processor or **central processing unit** or CPU) for doing computation
  - **Main memory** for storing temporary information and program data close to the processor
  - **Storage** devices (disks or SSDs) for storing long-term or persistent information
  - **I/O devices** to communicate with the external environment
    - Sensors
    - Peripherals

Idea 2: They all can solve the same problems

- Key Idea 2: All computers regardless of size, cost, and speed can compute the same things if they are given enough time and memory
  - Anything a fast computer can do, a slow computer can do

Computer = Universal Computational Device

- Anything that can be computed, can be computed by a computer provided it has enough time and enough memory
- We instruct the computer how to do **X**, and it obliges by interpreting our instructions
  - Instructions are stored in memory like regular data
- Computer is **programmable** because we can rewrite instructions to make it do something else

Computer = Universal Computational Device

- The idea of a universal computational device is due to Alan Turing (1937)

- He gave the mathematical description of the Turing machine
  - A particular kind of machine called the Turing machine can carry out all computations

Computer = Universal Computational Device

- First, we have two simple example Turing machines
  - One for addition, and one for multiplication

- What we now call a computer is a Turing machine that can simulate all Turing machines
- **What does it need as inputs?**
  - Inputs and the description of the Turing machine to simulate
  - We say that Turing machines are programmable
  - Hence, computers are programmable

Course Plan

- Navigate the transformation hierarchy
  - Bottom-up ✓
  - Top-down

Course Plan

Application Software	Programs
Operating Systems	Device Drivers
Architecture	Instructions Registers
Micro-architecture	Datapaths Controllers
Logic	Adders Memories
Digital Circuits	AND Gates NOT Gates
Analog Circuits	Amplifiers Filters
Devices	Transistors Diodes
Physics	Electrons

Course Plan

Application Software
>"hello world!"
Programs
Operating Systems
Device Drivers
Architecture
Instructions Registers
Software
Micro-architecture
Datapaths Controllers
Hardware
ISA is the boundary (Contract)
Logic
Adders Memories
Digital Circuits
AND Gates NOT Gates
Analog Circuits
Amplifiers Filters
Devices
Transistors Diodes
Physics
Electrons

Course Plan

Application Software (>"hello world!")	Programs
Operating Systems	Device Drivers
Architecture	Instructions Registers
Micro-architecture	Datapaths Controllers
Logic	Adders Memories
Digital Circuits	AND Gates NOT Gates
Analog Circuits	Amplifiers Filters
Devices	Transistors Diodes
Physics	Electrons

C Program

```
int z = z + 1;
for (...) {
```

compiler

Assembly

```
.global _start
_start:
    MOV R0, #0xFFFFFFFF
    MOV R1, #5
    MOV R2, #4
    MOV R3, #3

    ADDS R4,R0,R1
    ADC R5,R2,R3
```

assembler

Machine Code

Software

Hardware

ISA is the boundary (Contract)

01010010
10101010
10101001
10000011

Memory

Course Plan

Application Software (>"hello world!")	Programs
Operating Systems	Device Drivers
Architecture	Instructions Registers
Micro-architecture	Datapaths Controllers
Logic	Adders Memories
Digital Circuits	AND Gates NOT Gates
Analog Circuits	Amplifiers Filters
Devices	Transistors Diodes
Physics	Electrons

C Program

```
int z = z + 1;
for (...) {
```

compiler

Assembly

```
.global _start
_start:
    MOV R0, #0xFFFFFFFF
    MOV R1, #5
    MOV R2, #4
    MOV R3, #3

    ADDS R4,R0,R1
    ADC R5,R2,R3
```

assembler

Machine Code

Software

Hardware

ISA is the boundary (Contract)

01010010
10101010
10101001
10000011

Memory

Instructions stored as 0's and 1's

CPU

Course Plan

Application Software (>"hello world!")	Programs
Operating Systems	Device Drivers
Architecture	Instructions Registers
Micro-architecture	Datapaths Controllers
Logic	Adders Memories
Digital Circuits	AND Gates NOT Gates
Analog Circuits	Amplifiers Filters
Devices	Transistors Diodes
Physics	Electrons

C Program

```
int z = z + 1;
for (...) {
```

compiler

Assembly

```
.global _start
_start:
    MOV R0, #0xFFFFFFFF
    MOV R1, #5
    MOV R2, #4
    MOV R3, #3

    ADDS R4,R0,R1
    ADC R5,R2,R3
```

assembler

Machine Code

Software

Hardware

ISA is the boundary (Contract)

01010010
10101010
10101001
10000011

Memory

Instructions stored as 0's and 1's

CPU

Fetch, decode, execute an instruction every clock cycle

Course Plan

Application Software (>"hello world!")	Programs
Operating Systems	Device Drivers
Architecture	Instructions Registers
Micro-architecture	Datapaths Controllers
Logic	Adders Memories
Digital Circuits	AND Gates NOT Gates
Analog Circuits	Amplifiers Filters
Devices	Transistors Diodes
Physics	Electrons

C Program

```
int z = z + 1;
for (...) {
```

compiler

Assembly

```
.global _start
_start:
    MOV R0, #0xFFFFFFFF
    MOV R1, #5
    MOV R2, #4
    MOV R3, #3

    ADDS R4,R0,R1
    ADC R5,R2,R3
```

assembler

Machine Code

Software

Hardware

ISA is the boundary (Contract)

stored as 0's and 1's

Fetch, decode, execute an instruction every clock cycle

Assignment 1: Build CPU

Old Course Plan

Application Software (>"hello world!")	Programs
Operating Systems	Device Drivers
Architecture	Instructions Registers
Micro-architecture	Datapaths Controllers
Logic	Adders Memories
Digital Circuits	AND Gates NOT Gates
Analog Circuits	Amplifiers Filters
Devices	Transistors Diodes
Physics	Electrons

```
int z = z + 1;
for (...) {
```

C Program

compiler

Assembly

assembler

Machine Code

Assignment 2: Program CPU

Software

Hardware

ISA is the boundary (Contract)

stored as 0's and 1's

Fetch, decode, execute an instruction every clock cycle

Assignment 1: Build CPU

Academic Integrity

- Please follow the best practice guidelines in the assessment handout and policies

- AI use is permitted with acknowledgement
  - Rubric will dictate max. grade with AI use
  - Not a big issue for COMP2300
  - Closed-book exams account for 80% of the grade

- COMP2300 is known for upholding integrity!

2026 Course Plan

Application Software (>"hello world!")	Programs
Operating Systems	Device Drivers
Architecture	Instructions Registers
Micro-architecture	Datapaths Controllers
Logic	Adders Memories
Digital Circuits	AND Gates NOT Gates
Analog Circuits	Amplifiers Filters
Devices	Transistors Diodes
Physics	Electrons

C Program

```
int z = z + 1;
for (...) {
```

compiler

Assembly

assembler

Machine Code

Assignment 2: Program CPU

Software

Hardware

ISA is the boundary (Contract)

stored as 0's and 1's

Fetch, decode, execute an instruction every clock cycle

Assignment 1: Build CPU

2026 Course Plan

Application Software (>"hello world!")	Programs
Operating Systems	Device Drivers
Architecture	Instructions Registers
Micro-architecture	Datapaths Controllers
Logic	Adders Memories
Digital Circuits	AND Gates NOT Gates
Analog Circuits	Amplifiers Filters
Devices	Transistors Diodes
Physics	Electrons

C Program

```
int z = z + 1;
for (...) {
```

compiler

Assembly

assembler

Machine Code

```
.global _start
_start:
    MOV R0, #0xFFFFFFFF
    MOV R1, #5
    MOV R2, #2
    MOV R3, #3
    ADDS R4,R0,R1
    ADC R5,R2,R3
```

Assembly in Labs

New labs

New exam questions

Software

Hardware

ISA is the boundary (Contract)

stored as 0's and 1's

Fetch, decode, execute an instruction every clock cycle

Assignment 1: Build CPU

2026 Course Plan

- Assembly practice in the labs after assignment 1 submission
- New labs focusing on advanced CPU (computer) design
- New exam questions replacing old assignment 2

Why Study Computer architecture ?

- Hardware is increasingly heterogeneous
- Programmers today need a good understanding of what the hardware offers

Past Systems

Why Study Computer architecture ?

- Programmers today need a good understanding of what the hardware offers

Why Study Computer architecture ?

- **Programming models are domain-specific**
  - New ML/AI accelerators
  - Programming models intertwined with hardware details
  - No luxury of a commodity language

New Important Metrics

- Traditional metrics to evaluate a computer
  - Performance
  - Energy efficiency
  - Battery life
  - Cost

New Important Metrics

- Today, we care a lot about
  - Reliability
  - Sustainability
  - Security

Sustainability

- Sustainability is an important concern
  - ICT contribution to global GHG emissions around 4%, and increasing
  - Need to understand emissions during manufacturing of computers and in deployment
  - Sustainability is the latest sub-area in the field of computer architecture

Security Demands Robust Hardware

- Security is desirable over raw performance in many environments
  - Recent cyber attacks target vulnerabilities in CPU design
  - Speculative execution reveals secrets
  - Today's hardware built for performance
- Need to consider security as a first principle in design

Why Study Computer architecture ?

- Writing compilers, operating systems, virtual machines

Why Study Computer architecture ?

- Writing compilers, operating systems, virtual machines

- Programming embedded computers
  - Nano drones, IoT devices, wearable computing

Why Study Computer architecture ?

- Writing compilers, operating systems, virtual machines

- Programming embedded computers
  - Nano drones, IoT devices, wearable computing

- Debugging performance issues better than the average programmer

Our Ultimate Goals

- Make you think broadly *about computing*

- Make you think critically

- Give INSIGHT

Some Comments from Previous Years

- COMP2300 is "empowering"
- **"After building the CPU, I know exactly what my program is doing"**
- "When I engage with a computer expert, I know what they are talking about"
- "I use the knowledge of modern processors from this course every single day" *Now a Security professional*

Some Comments from Previous Years

- Genuinely the best course I have taken yet - the content was really well structured, eg. the transition from circuits to then assembly coding just made so much sense. There was a real sense of satisfaction at the end of assignment 1, looking at how I could understand every individual part of the CPU we built. The lectures expanded on the knowledge that was taught in labs, which helped to solidify the concepts. Also, I liked how the assignments built upon the labs, as it gave a lot of incentive to stay on top of course work. The content is fun too, looking forward to comp2310.

Some Comments from Previous Years

- How the course starts from scratch and progressively abstracts and builds upon concepts to build strong intuition in complex topics. The high quality of explanations in lectures, and the support from tutors and the course conveners on Ed and in labs. The assignment tasks were awesome, especially for the opportunity they provided to take on extension tasks. the exam was excellently designed. Everything about this course is great.

Some Comments from Previous Years

- This is easily the best, most interesting and well-structured course I have done in the two years I've spent at ANU. I definitely have learnt a lot about computers and how they execute programs under the hood. The assignments were quite interesting and I particularly enjoyed making my own CPU and building a snake game on the microbit. I couldn't emphasise any more about how great this course was in my experience, Shoaib was great he often swiftly replied on ed and emails and seemed to actually care about student wellbeing and how students were interpreting the course material.

Some Comments from Previous Years

- Out of my many years at ANU, this course is definitely in my top 3 favourite courses. The content was taught in a well thought out manner that structurally built upon itself at a reasonable pace. Additionally, the content was taught in an engaging manner, I particularly liked how the assignments were done. Shoaib's explanations in the lectures where really good, he took an appropriate amount of time explaining concepts and even explained the same concept in different styles which greatly helped with my understanding of a complicated topic.

Some Comments from Previous Years

- The content was extremely interesting, and it was very well taught. Shoaib was a great lecturer, and clearly understood what he was teaching us very well. He was also pretty funny lol.

Representing Information

Representing Information

- Continuous or analog variables/signals can take an infinite number of values
  - Frequency of oscillation
  - Pressure
  - Position
  - Volume

Analog Signals

- What do we mean by **“analog”** signals?

- Let’s look at some examples

Representing Information: Analog

- Mercury's volume represents temperature

Representing Information: Analog

- Position of needle represents weight

Representing Information: Analog

- Modulated grooves on vinyl record represent sound

Representing Information: Analog

- Chemical properties of film represent captured image

Representing Information: Analog

- Measuring things and storing data by **analogy** has been the **predominant** approach in history

- Analog signals are called **analog** because they are directly proportional to the original information they represent (and can take an infinite # values)

- The term **analog** comes from the Greek word **"analogos"** which means **"proportional"**

Problem with Analog Approach

- **Using continuous variables to represent information is hard for the computer to deal with**
  - Sometimes difficult to measure precisely
  - Difficult to store
  - Difficult to copy

This Course Uses Digital Approach

- If we don't want to represent data as something with **potentially infinitely varying analog values**, what can we do?
  - Use the **Digital** approach
- Represent information using **discrete steps** rather than continuous variation

This Course Uses Digital Approach

- **Digital Systems** represent information with discrete-valued variables

- Discrete variables can have a finite number of distinct values

Early Digital System

- Charles Babbage's analytical engine used **ten** discrete values

- Used mechanical parts such as gears with ten positions 0 – 9

Digital Approach

- Modern digital systems use a binary **(two-valued)** representation

0 1

- Digital designers can focus on **1's** and **0's** without worrying about the physical realization of such variables

Binary Representation

- **High voltage:** Presence of “something” meaning **1**

- **Low voltage:** Absence of “something” meaning **0**

0 1

- **As we will see precise voltage levels do not matter**

Another Binary Representation

0 1

With “some” effort, one can build a computer using billions of coca colas

But there is not a big market for Cola Computers.

- **Computer systems & architecture is closer to engineering than science**

- **Practicality and cost-efficiency are primary concerns**

Why do Computers Use Voltage?

- Mechanical parts are not easy to scale to do large computations – circa 1850 Babbage engine

- Some 1964 computers (CDC 6600 & IBM 360)

- 2020 Apple M1 400 mm^2 and 16 billion transistors

Representing Large Sets in Binary

- **We need more than 0 and 1 to represent large quantities and sets**

- Data is represented using a sequence of symbols where each symbol is 0 or 1

Binary Representation

- **Digital systems internally use "voltages" for representing binary variables**
  - → Low voltage means 0
  - → High voltage means 1

B I N A R Y D I G I T

- A bit is a unit of information
- A binary variable represents one bit of information
- To represent discrete sets with more than two elements, we combine multiple bits into a binary code

Binary Codes

- **Suppose we want to represent 4 colors: {red, blue, green, black}**
  - How many bits of information do I need?
  - (00, 01, 10, 11)
  - The assignment of the **2-bit binary code** to colors is *ad-hoc*
  - Also legitimate is: (10, 11, 00, 01)

- **How many bits of information do we need to represent the alphabet set in English?**
  - For 26 alphabets, we need 5 bits

Information Content in a Binary Code

- Amount of information in a discrete variable with N states

$$\mathrm{D} = Log_2 N \ bits$$

- If **C is a variable representing four colors in the color set**
  - We say that C has or caries two bits of information
- A variable **A represents the English alphabet**
  - It contains 5 bits of information

Why do computers use binary?

- Two symbols enable simplified hardware and improved reliability

- Keep **complexity** and **cost** under control

- Easy to use transistor as a **switch**!
  - We will see later

TRUE and FALSE

0	False	Off
1	True	On
F		
T		

- True and False are called logical values
  - Logical variable is one that can be 1 or 0 (True or False)

Number Systems

Decimal Number System

- Base 10 means 10 digits (0, 1, 2, 3, 4, 5, 6, 7, 8, 9)
- Multiple digits form longer decimal numbers
- Each column of a decimal number has 10 times the weight of the previous column

1000's column
100's column
10's column
1's column

coefficient

power of 10

$$9742 = 9 \times 10^3 + 7 \times 10^2 + 4 \times 10^1 + 2 \times 10^0$$

nine thousands | seven hundreds | four tens | two ones

Range of Decimal Numbers

- An **N-digit** decimal number represents one of 10^N possibilities
  - 0, 1, 2, 3, ..., $10^N - 1$

- **3 digits:** 1000 possibilities in the range 0 – 999

Binary Numbers

- Base **2** means **2** digits (0, 1)
- Multiple bits form longer binary numbers
- Each column of a binary number has **2** times the weight of the previous column

coefficient

power of 2

8's column	4's column	2's column	1's column					
1	0	0	1	=	1×2^3	+	0×2^2	+	0×2^1	+	1×2^0
					one eight		one four		zero two		one one

Range of Binary Numbers

- **An N-bit binary number represents one of 2^N possibilities**
  - 0, 1, 2, 3, ..., $2^N - 1$
  - 3 bits: 8 (= 2 X 2 X 2) possibilities in the range 0 – 7
  - 4 bits: ?
  - 5 bits: ?
  - 10 bits: ?

Powers of 2

Columns #	Power of 2	Weight
0	2^0	1
1	2^1	2
2	2^2	4
3	2^3	8
4	2^4	16
5	2^5	32
6	2^6	64
7	2^7	128
8	2^8	256
9	2^9	512

Columns #	Power of 2	Weight	
10	2^{10}	1024	Kilo
11	2^{11}	2048	
12	2^{12}	4096	
13	2^{13}	8192	
14	2^{14}	16384	
15	2^{15}	32768	
16	2^{16}	65536	

Powers of 2

Power of 2	Decimal Value	Abbreviation	
2^{10}	1024	Kilo (K)	~ 1000
2^{20}	1048576	Mega (M)	~ 1000, 000
2^{30}	1073741824	Giga (G)	~ 1000, 000, 000

What is 2^{24} in decimal?

- 2^{20} X 2^{4} = 1 M X 16 = 16 M

What is 2^{17} in decimal?

- 2^{10} X 2^{7} = 1 K X 128 = 128 K

Terminology

Byte

- 8 bits

Nibble

- 4 bits

Word

- Machine-dependent
- 8 – 16 bits (gadgets)
- 32 – 64 bits (high-end)

Most Significant Bit

1	0	0	0	0	0	0	1

The bit in the highest position

Least Significant Bit

1	0	0	0	0	0	0	1

The bit in the lowest position

Terminology

Most Significant Byte

0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1

*The byte in the **highest** position*

Least Significant Byte

0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1

*The byte in the **lowest** position*

Rev: Binary Codes

Column weight	2^2	2^1	2^0
Column #	2	1	0
	0	0	0
	0	0	1
	0	1	0
	0	1	1
	1	0	0
	1	0	1
	1	1	0
	1	1	1

3-bit
8 elements

- **Write combinations in a systematic way**
- **Note how often the bit flips in each column**
- **Can represent an arbitrary set with a binary code**

Rev: Binary Codes

0
1

1-bit
2 elements

0	0
0	1
1	0
1	1

2-bit
4 elements

0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

3-bit
8 elements

0	0	0	0
0	0	0	1
0	0	1	0
0	0	1	1
0	1	0	0
0	1	0	1
0	1	1	0
0	1	1	1
1	0	0	0
1	0	0	1
1	0	1	0
1	0	1	1
1	1	0	0
1	1	0	1
1	1	1	0
1	1	1	1

4-bit
16 elements

- **Write combinations in a systematic way**
- **Note how often the bit flips in each column**
- **Can represent an arbitrary set with a binary code**

Decimal to Binary Conversion

Method # 1: Find the largest power of 2, subtract, and repeat

Example: Convert 53_{10} to binary

53	32 X 1
53 – 32 = 21	16 X 1
21 – 16 = 5	4 X 1
5 – 4 = 1	1 X 1

2^5	2^4	2^3	2^2	2^1	2^0
1	1	0	1	0	1

Decimal to Binary Conversion

Method # 2: Repeatedly divide by 2, remainder goes in each column

Example: Convert 53_{10} to binary

53/2	=	26	R: 1
26/2	=	13	R: 0
13/2	=	6	R: 1
6/2	=	3	R: 0
3/2	=	1	R: 1
1/2	=	0	R: 1

2^5	2^4	2^3	2^2	2^1	2^0
1	1	0	1	0	1

Hexadecimal Numbers

Motivation: Tedious and error-prone to write long binary numbers

Hexadecimal or base 16: A group of four bits represent 2^4 or 16 possibilities

16 digits: 0 – 9, A, B, C, D, E, F

Column weights: 1, 16, 16^2 (or 256), 16^3 (or 4096)

Hexadecimal Numbers

Hex Digit	Decimal Equivalent	Binary Equivalent
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

Binary to Hexadecimal

Binary	1	1	1	1	0	1	1	1
Hexa	F	7						

Binary	1	1	1	1	1	1	1	0
Hexa	F	E						

Hexadecimal to Binary

Hexa	**D**	**7**	**4**	**1**												
Binary	**1**	**1**	**0**	**1**	**0**	**1**	**1**	**1**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**1**

Binary Addition

carries

		1	1				1	1	
	4	2	7	7		1	0	1	1
+	5	4	9	9	+	0	0	1	1
	9	7	7	6		1	1	1	0

Decimal Addition

Binary Addition

1 + 1 = 2 (10 in binary), but a binary variable can either be 0 or 1

- We record the 1's digit (0), and carry the 2's digit (1) over to the next column

1 + 1 + 1 = 3 (11 in binary), but a binary variable can either be 0 or 1

- We record the 1's digit (1), and carry the 2's digit (1) over to the next column

Overflow

- A = 1111 and B = 1111
  - A + B does not fit in the largest value four bits can represent
- **Overflow:** When the result is too big to fit inside the available bits
- **Detection:** If there is a carry bit out of the most significant column

		1	1	1		
		1	1	1	1	15
+		1	1	1	1	15
	1	1	1	1	0	30

Signed Binary Numbers

Signed Binary Numbers

- We need both positive and negative numbers to solve real-world problems

- *How do we make a string of 1 and 0 represent both positive and negative numbers?*

- If we write all possible combinations of 0 and 1 in a **disciplined** fashion, maybe we can find a way

Most significant bit		least significant bit
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

Signed Binary Numbers

- Use the *most significant bit* to represent the sign: 0 means positive and 1 means negative
- **N bit sign/magnitude system:** 1 bit for sign and N – 1 bits for magnitude (**absolute**)

			Decimal
0	0	0	+0
0	0	1	+1
0	1	0	+2
0	1	1	+3
1	0	0	-0
1	0	1	-1
1	1	0	-2
1	1	1	-3

Drawbacks of Sign/Mag Rep.

- Ordinary binary addition does not work for sign/magnitude numbers
  - What is the sum of +3 and -3 and does the result make sense?
- Zero has two representations (awkward)

			Decimal
0	0	0	+0
0	0	1	+1
0	1	0	+2
0	1	1	+3
1	0	0	-0
1	0	1	-1
1	1	0	-2
1	1	1	-3

One's Complement

- 1's complement was tried in early computers, such as, Control Data Corporation (CDC) 6600
- **Negative number:** Flip all bits of the binary representation of a positive integer
- Suffers from the same problems as the sign/magnitude representation

			Decimal
0	0	0	+0
0	0	1	+1
0	1	0	+2
0	1	1	+3
1	0	0	-3
1	0	1	-2
1	1	0	-1
1	1	1	-0

Two’s Complement

- A third system of representation for signed integers
- Ordinary addition works
- There is a single representation for *zero*
- Used in almost all computers today

Problem: If A + B = C, and A is known, then find B, such that C = 0

			Binary				Decimal
	A	=	0	1	0	1	?
+	B	=	?	?	?	?	?
	C	=	0	0	0	0	0

Problem: If A + B = C, and A is known, then find B, such that C = 0

			Binary				Decimal
	A	=	0	1	0	1	+5
+	B	=	?	?	?	?	?
	C	=	0	0	0	0	0

Problem: If A + B = C, and A is known, then find B, such that C = 0

			Binary				Decimal
	A	=	0	1	0	1	+5
+	B	=	?	?	?	?	-5
	C	=	0	0	0	0	0

Problem: If A + B = C, and A is known, then find B, such that C = 0

			Binary				Decimal
	A	=	0	1	0	1	+5
+	B	=	1	0	1	1	-5
	C	=	0	0	0	0	0

Problem: If A + B = C, and A is known, then find B, such that C = 0

What is the relationship between A and B?

			Binary				Decimal
	A	=	0	1	0	1	+5
+	B	=	1	0	1	1	-5
	C	=	0	0	0	0	0

Some Observations

Observation # 1: If A + B = C, and A is +5, and C is 0, then B must be -5. (**We have found a new representation for negative numbers.**)

Observation # 2: To transform A to B (i.e., +5 to -5), we need to take 1's complement of A and then add +1. We say that B is **2's complement** of A

Observation # 3: Like sign/magnitude numbers, positive numbers have the MSB set to 0, and negative numbers have the MSB set to 1

Some Observations

Observation # 4: Ordinary addition works

- What is the sum of +3 and -3 in two's complement system, and *does the result make sense*?

- Since ordinary addition works, **a circuit to add numbers can handle both addition and subtraction**
  - Recall that, X – A is equivalent to X + (–A)

		1		
	0	1	1	+3
+	1	0	1	-3
	0	0	0	

2’s Complement Circle

More Observations

Observation # 5: There is only one representation for *zero*

Observation # 6: There is one more negative number than positive number

- With 3 bits, this number is 100
- With 4 bits, this number is 1000
- This negative number has no positive counterpart
- It is called the *weird number*
- The 2's complement of the weird number is the weird number (verify!)

2's Complement to Decimal

- If **MSB is 0**
  - It is a positive number. The magnitude is represented by the remaining N – 1 bits

- If **MSB is 1**
  - It is a negative number. Take the two's complement of the (binary) number. The magnitude of the negative number) is represented by the N – 1 bits

Practice and test your understanding using the two's complement circle

Overflow in 2's Complement

- Suppose we have two 5-bit numbers
  - A = 01001 and B = 01011
  - What is A + B?
  - What is the largest value 5 bits can represent in 2's complement?
- **Overflow**
  - The result is too big to fit inside the available bits
  - Sum of two positive integers cannot produce a negative integer!

		1	1			
	0	1	0	0	1	+9
+	0	1	0	1	1	+11
	1	0	1	0	0	-12

Overflow in 2's Complement

- Suppose we have two 5-bit numbers
  - A = 10100 and B = 11010
  - What is A + B?
  - What is the *smallest* value 5 bits can represent in 2's complement?
- **Overflow**
  - The result is too big to fit inside the available bits
  - Sum of two negative integers cannot produce a positive integer!

	1	0	1	0	0	-12
+	1	1	0	1	0	-6
	0	1	1	1	0	14

Overflow in 2's Complement

Observation # 1: If two number being added have the same sign bit and the result has the opposite sign bit (easy!)

Observation # 2: Unlike unsigned numbers, a carry out of the most significant bit does not indicate overflow

Observation # 3: The sum of a negative number and a positive number never produces an overflow (**prove yourself!**)

Range of Number Systems

Number System	Minimum	Maximum
Unsigned	0	$2^N - 1$
Sign/Magnitude	$-2^{N-1} + 1$	$2^{N-1} - 1$
Two's Complement	-2^{N-1}	$2^{N-1} - 1$

N = 3

Unsigned: 0 to 7

Sign/Magnitude: -3 to 3

2's Complement: -4 to 3

N = 4

Unsigned: 0 to ?

Sign/Magnitude: -? to ?

2's Complement: -? to ?

Comparing Number Systems

Binary Representation	Decimal Value Represented			
	Unsigned	Signed Magnitude	1's Complement	2's Complement
000	0	0	0	0
001	1	1	1	1
010	2	2	2	2
011	3	3	3	3
100	4	-0	-3	-4
101	5	-1	-2	-3
110	6	-2	-1	-2
111	7	-3	-0	-1

Quiz: See any errors?

	Unsigned	Signed	1's Comp.	2's Comp.
0 0 0 0	0	0	0	0
0 0 0 1	1	1	1	1
0 0 1 0	2	2	2	2
0 0 1 1	3	3	3	3
0 1 0 0	4	4	4	4
0 1 0 1	5	5	5	5
0 1 1 0	6	6	6	6
0 1 1 1	7	7	7	7
1 0 0 0	8	-0	-7	-1
1 0 0 1	9	-1	-6	-2
1 0 1 0	10	-2	-5	-3
1 0 1 1	11	-3	-4	-4
1 1 0 0	12	-4	-3	-5
1 1 0 1	13	-5	-2	-6
1 1 1 0	14	-6	-1	-7
1 1 1 1	15	-7	-0	-8

Sign Extension in 2's Comp. #s

Question: What is the difference between the 16-bit and 4-bit numbers below?

16-bit number	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1
4-bit number	0	1	0	1												

Answer: None. They both represent the positive number 5
Leading zeros do not impact the magnitude of a binary number

There are times when it is useful to allocate a small number of bits to represent a value

Sign Extension

- What value does the two numbers below represent?

16-bit number (**A**)	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**1**
4-bit number (**B**)	**1**	**1**	**0**	**1**												

- What is the sum of **A** and **B**?
  - **Scenario # 1:** Assume the absence of bits in **B** to be 0
  - **Scenario # 2:** Assume the absence of bits in **B** to be 1

Scenario # 1

	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1	+13
+	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1	-3

X

The assumption that appending 0's will lead to correct addition was wrong

Scenario # 2

	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1	+13
+	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	-3

✓

The assumption that appending 1's will lead to correct addition was right

Sign Extension

- Leading 0's do not change the magnitude of the positive number
- Leading 1's do not change the magnitude of the negative number

When a 2's complement number is extended to more bits, the sign bit must be copied into the most significant bit positions. We refer to the operation as Sign-EXTension or SEXT

Building Block of Computers

Transistors

We are Here.

Transistors

- **Computers are built from billions of small and simple structures called transistors**
  - 1970: Few 1000s of transistors
  - Apple's M2 Max: **50+ Billion** transistors
  - **Moore's Law:** Transistor count double in 18 months
  - Computers with improved capability over time due to a large # transistors at the device level

- **We will cover**
  - How MOS transistor works (as a logic element)?
  - How transistors are connected to form logic gates?
  - How logic gates are interconnected to form larger units that are needed to construct a computer

Moore's Law: The number of transistors on microchips doubles every two years

Our World in Data

Moore's law describes the empirical regularity that the number of transistors on integrated circuits doubles approximately every two years. This advancement is important for other aspects of technological progress in computing – such as processing speed or the price of computers.

Data source: Wikipedia (wikipedia.org/wiki/Transistor_count)

OurWorldinData.org – Research and data to make progress against the world's largest problems.

Licensed under CC-BY by the authors Hannah Ritchie and Max Roser.

Transistors

- Sections 1.6 and 1.7 in Harris & Harris provide more technical explanations than what we will cover

MOS Transistor

- **MOS** stands for
  - Conductors (**M**etal)
  - Insulators (**O**xide)
  - **S**emiconductors

- MOS transistor has three terminals

- We can combine many of these to form logic gates
  - *The electrical properties of metal-oxide semiconductors are well beyond the scope of what we want to understand in this course*
  - They are below our **lowest level of abstraction**
  - If transistors **misbehave**, an architect is at their mercy (unlikely to happen)

Two Types of MOS Transistors

- Two types: n-type and p-type
- They both operate "logically," very similar to the way wall switches work

n-type

p-type

How Does a Transistor work?

- For the lamp to glow, electrons must flow
- For electrons to flow, there must be a closed circuit from the power supply to the lamp and back to the power supply
- The lamp can be turned on and off by simply manipulating the wall switch to make or break the closed circuit

n-type MOS Transistor

- Instead of the wall switch, we could use an n-type or a p-type MOS transistor to make or break the closed circuit

If the gate of the n-type transistor is supplied with a **high** voltage, the connection from source to drain acts like a piece of wire (**we have a closed circuit**)

If the gate of the n-type transistor is supplied with **zero** voltage, the connection between source and drain is broken (**we have an open circuit**)

Schematic of an n-type MOS transistor

- Depending on the technology, high voltage can range from **0.3V** to **3V**

How Does a Transistor work?

- The n-type transistor in a circuit with a battery and a bulb

p-type MOS Transistor

- The p-type MOS transistor works in exactly the opposite fashion from the n-type transistor

The circuit is **open** when the gate is supplied with 3V

The circuit is **closed** when the gate is supplied with 0V

- Depending on the technology, high voltage can range from **0.3V** to **3V**

Some Examples of

Transistors as Building Blocks

Modern Special-Purpose ASIC

- The largest ML accelerator chip (2021)
- 850,000 cores

Cerebras WSE-2
2.6 Trillion transistors
46,225 mm^2

Largest GPU
54.2 Billion transistors
826 mm^2
NVIDIA Ampere GA100

https://www.anandtech.com/show/14758/hot-chips-31-live-blogs-cerebras-wafer-scale-deep-learning

https://www.cerebras.net/cerebras-wafer-scale-engine-why-we-need-big-chips-for-deep-learning/

Apple M1 Ultra (2022)

https://stadt-bremerhaven.de/apple-neuer-m1-ultra-chip-ist-offiziell/

Modern General-Purpose Microprocessor

10nm ESF=Intel 7 Alder Lake die shot (~209mm²) from Intel: https://www.intel.com/content/www/us/en/newsroom/news/12th-gen-core-processors.html
Die shot interpretation by Locuza, October 2021

Intel Alder Lake, 2021

Logic Gates

One Level Higher in Abstraction

- Now, we know how a **MOS transistor** works
- How do we build logic structures out of individual MOS transistors
- We called these logic structures logic gates and they implement simple **Boolean** functions

Boolean Logic

George Boole
English mathematician and philosopher

- A system for describing logical statements/expressions where variables are TRUE or FALSE

- Defines *important but simple* logical operations on binary logical variables

- Boolean algebra defines manipulation rules similar to elementary algebra

Logic

- **Logic comes from reasoning or thinking**
- When presented with some facts, how to derive a valid conclusion
- Assume statements are either TRUE or FALSE
- **When many statements (or conditions) are combined, what is the conclusion or resulting statement?**

Origin of Logic Functions

- Canberra is the **capital** of Australia, **AND today** I am in Canberra, **therefore, today, I am in the capital of Australia**

- Given it is **warm,** and given it is **sunny, OR** it is my birthday, **I conclude that I am going to the beach**

Logic Functions vs. Gates

- Logic gates are the **physical realization** of logic functions (built with transistors)

- Logic operations are the **steppingstone** for composing **sophisticated digital circuits** that perform binary arithmetic

Logic Gates

- Logic gates are digital circuits that take one or more inputs and produce a binary output

- The inputs are to the left, and the outputs are to the right

- The relationship between inputs and outputs is described by a **truth table** or a **Boolean equation**

Truth Table

- A convenient way to describe the behavior of logical functions

- Suppose **A** and **B** are input operands and **Y** is the output
  - **A** can be 0 or 1
  - **B** can be 0 or 1
  - Four combinations (rows)
  - Three columns (2 inputs and an output)

- The Boolean equation for **Y**: **Y** = 0
  - The values of **A** and **B** does not matter

A	B	Y
0	0	0
0	1	0
1	0	0
1	1	0

Truth Table with More Inputs

A	B	C	**Y**
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

- Boolean Equation for output **Y**: **Y** = 1

Note: *Soon we will see more interesting logic functions than Y = 0 and Y = 1*

Logic Gates

Name	NOT	AND	NAND	OR	NOR	XOR	XNOR
Alg. Expr.	$\overline{A}$	AB	$\overline{AB}$	$A+B$	$\overline{A+B}$	$A \oplus B$	$\overline{A \oplus B}$
Symbol							
Truth Table	A | X 0 | **1** 1 | **0**	B A | X 0 0 | **0** 0 1 | **0** 1 0 | **0** 1 1 | **1**	B A | X 0 0 | **1** 0 1 | **1** 1 0 | **1** 1 1 | **0**	B A | X 0 0 | **0** 0 1 | **1** 1 0 | **1** 1 1 | **1**	B A | X 0 0 | **1** 0 1 | **0** 1 0 | **0** 1 1 | **0**	B A | X 0 0 | **0** 0 1 | **1** 1 0 | **1** 1 1 | **0**	B A | X 0 0 | **1** 0 1 | **0** 1 0 | **0** 1 1 | **1**

A Useful Circuit: Tutorial # 1

- **What does this circuit do?**

- Multiplexer
- **Used for decision making and often found inside control logic**

Making Logic Gates Using CMOS Technology

- Modern computers use both n-type and p-type transistors, called Complementary MOS (CMOS) technology
  - **nMOS + pMOS = CMOS**

CMOS Technology Fundamentals

- Let's look at the simplest logic structure that exists in a modern computer
  - What does this circuit do?

CMOS Technology Fundamentals

- What happens when the input is connected to 0V?

CMOS Technology Fundamentals

- **p**-type transistors are good at pulling u**p** the voltage

CMOS Technology Fundamentals

- What happens when the input is connected to 3V?

CMOS Technology Fundamentals

- **n**-type transistors are good at pulling dow**n** the voltage

CMOS NOT Gate (Inverter)

- We have seen a **NOT** gate at the transistor level
  - If A = **0V** then Y = **3V**
  - If A = **3V** then Y = **0V**

- Interpretation of voltage levels
  - Interpret **0V** as logical (binary) 0 value
  - Interpret **3V** as logical (binary) 1 value

A	P	N	Y
0	**ON**	OFF	1
1	OFF	**ON**	0

$$Y = \bar{A}$$

CMOS NOT Gate (Inverter)

A	Y
0	1
1	0

The NOT gate has only one input (unary)

$Y = A'$

$Y = \bar{A}$

Truth Table

NOT Logic Gate

Boolean Equation

NOT Function: *The output Y is the inverse of the input A*

Another CMOS Gate: What is this?

CMOS NAND Gate

A	B	P1	P2	N1	N2	Y

- P1 and P2 are in parallel; only one must be **ON** to pull up the voltage to 3V
- N1 and N2 are connected in series; both must be **ON** to pull down the voltage to 0V

CMOS NAND Gate

A	B	P1	P2	N1	N2	Y
0	0	ON	ON	OFF	OFF	1

- P1 and P2 are in parallel; only one must be **ON** to pull up the voltage to 3V
- N1 and N2 are connected in series; both must be **ON** to pull down the voltage to 0V

CMOS NAND Gate

A	B	P1	P2	N1	N2	Y
0	0	ON	ON	OFF	OFF	1
0	1	ON	OFF	OFF	ON	1

- P1 and P2 are in parallel; only one must be **ON** to pull up the voltage to 3V
- N1 and N2 are connected in series; both must be **ON** to pull down the voltage to 0V

CMOS NAND Gate

A	B	P1	P2	N1	N2	Y
0	0	ON	ON	OFF	OFF	1
0	1	ON	OFF	OFF	ON	1
1	0	OFF	ON	ON	OFF	1

- P1 and P2 are in parallel; only one must be **ON** to pull up the voltage to 3V
- N1 and N2 are connected in series; both must be **ON** to pull down the voltage to 0V

CMOS NAND Gate

A	B	P1	P2	N1	N2	Y
0	0	ON	ON	OFF	OFF	1
0	1	ON	OFF	OFF	ON	1
1	0	OFF	ON	ON	OFF	1
1	1	OFF	OFF	ON	ON	0

- P1 and P2 are in parallel; only one must be **ON** to pull up the voltage to 3V
- N1 and N2 are connected in series; both must be **ON** to pull down the voltage to 0V

CMOS NAND Gate

A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

Truth Table

NAND Logic Gate

$Y = (AB)'$

$Y = \overline{AB}$

Boolean Equation

NAND Function: *The output Y is 1 unless both inputs are 1*

CMOS AND Gate

- We make an AND gate using one NAND gate and one NOT gate
- **Homework:** Can we not use fewer transistors for the AND gate?

CMOS NOT, NAND, and AND Gates

A	Y
0	1
1	0

A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

General CMOS Gate Structure

- We have a general form to construct any inverting logic gate, such as, NOT, NAND, NOR
  - The networks may consist of transistors in series of in parallel
  - When transistors are in **parallel**, the network is **ON** if one of the transistors is **ON**
  - When transistors are in **series**, then network is **ON** only if all transistors are **ON**

pMOS transistors are used for pull-u**p**
nMOS transistors are used for pull-dow**n**

General CMOS Gate Structure

- Exactly one network should be ON, and the other network should be OFF at any given time
- If both networks are ON simultaneously, there is a short circuit → incorrect operation
- If both networks are OFF simultaneously, the output is floating → undefined

pMOS transistors are used for pull-u**p**
nMOS transistors are used for pull-dow**n**

Why This Structure?

- MOS transistors are imperfect switches

- pMOS transistors pass **1's** well but **0's** poorly
  - **p**MOS transistors are good at "pulling u**p**" the output
- nMOS transistors pass **0's** well but **1's** poorly
  - **n**MOS transistors are good at "pulling dow**n**" the output

Latency or Delay

- Which one is faster?
  - Transistor in series
  - Transistors in parallel

- Series connections are slower than parallel connections
  - More **resistance** on the wire

- **Remember: Latency of series vs. parallel circuits extend from transistors to gates and larger circuits**

- See Section 1.7.8 for more details

Gates with More Than Two Inputs

- We can create larger gates with more than 2 inputs
  - 3-input NOR gate or 11-input NAND gate

Figure 1.35 Three-input NAND gate schematic

Manufacturing Tech is the Enabler

- **Precision Manufacturing**
  - Extreme Ultraviolet (EUV) light to pattern <10nm structures

https://www.youtube.com/watch?v=Jv40Viz-KTc

Application Software
>"hello world!"
Programs
Operating Systems
Device Drivers
Architecture
Instructions
Registers
Micro-architecture
Datapaths
Controllers
Logic
Adders
Memories
Digital Circuits
AND Gates
NOT Gates
Analog Circuits
Amplifiers
Filters
Devices
Transistors
Diodes
Physics
Electrons

We are here

The AND Function

A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

Truth Table

AND Logic Gate

$Y = AB$

$Y = A.B$ (product)

$Y = A \cap B$ (intersection)

Boolean Equation

AND Function: *The output Y is 1 if and only if both A and B are 1*

The OR Function

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

Truth Table

OR Logic Gate

$Y = A + B$ (sum)

$Y = A \cup B$ (union)

Boolean Equation

OR Function: *The output Y is 1 if either A or B are 1*

The XOR Function

eXclusive-OR

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

Truth Table

XOR Logic Gate

$Y = A \oplus B$

Boolean Equation

XOR Function: *The output Y is 1 if A or B, but not both, are 1*

OR and XOR

- The term exclusive is used because the output is 1 if only one of the inputs is 1 (mutually exclusive)

- OR produces an output 1, if only one of the two sources is a 1, or both sources are one (inclusive OR)

The NOT Unary Function

A	Y
0	1
1	0

The NOT gate has only one input (unary)

$Y = A'$

$Y = \bar{A}$

Truth Table

NOT Logic Gate

Boolean Equation

NOT Function: *The output Y is the inverse of the input A*

Inverting a Gate's Operation

Any gate can be followed by a bubble to invert its operation

NOT AND → NAND

NOT OR → NOR

NOT XOR → XNOR

NOT NOT → BUF

In Boolean logic, two wrongs make a right!

We say that two bubbles cancel each other's effect

The NAND Function

A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

Truth Table

NAND Logic Gate

Y = (AB)’

Boolean Equation

NAND Function: *The output Y is 1 unless both inputs are 1*

The NOR Function

A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

Truth Table

NOR Logic Gate

$Y = (A + B)'$

Boolean Equation

NOR Function: *The output Y is 1 if neither A nor B is 1*

The XNOR Function

A	B	Y
0	0	1
0	1	0
1	0	0
1	1	1

Truth Table

XNOR Logic Gate

$Y = (A \oplus B)'$

Boolean Equation

XNOR Function: *The output Y is 1 if both A and B are 1*

XOR and XNOR are special

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

XOR

A	B	Y
0	0	1
0	1	0
1	0	0
1	1	1

XNOR

XOR: Output is 1 when inputs are not equal (odd number of 1's)

Parity Gate

XNOR: Output is 1 when inputs are equal (even number of 1's)

Equality Gate

Buffer (BUF)

A	Y
0	0
0	0
1	1
1	1

Truth Table

BUF Logic Gate

$Y = A$

Boolean Equation

Buffer: *The output Y is equal to the input A*

Logic Gates

Name	NOT	AND	NAND	OR	NOR	XOR	XNOR
Alg. Expr.	$\overline{A}$	AB	$\overline{AB}$	$A+B$	$\overline{A+B}$	$A \oplus B$	$\overline{A \oplus B}$
Symbol							
Truth Table	A | X 0 | **1** 1 | **0**	B A | X 0 0 | **0** 0 1 | **0** 1 0 | **0** 1 1 | **1**	B A | X 0 0 | **1** 0 1 | **1** 1 0 | **1** 1 1 | **0**	B A | X 0 0 | **0** 0 1 | **1** 1 0 | **1** 1 1 | **1**	B A | X 0 0 | **1** 0 1 | **0** 1 0 | **0** 1 1 | **0**	B A | X 0 0 | **0** 0 1 | **1** 1 0 | **1** 1 1 | **0**	B A | X 0 0 | **1** 0 1 | **0** 1 0 | **0** 1 1 | **1**

Multiple-Input Gates

A	B	C	Y
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Gates with multiple inputs are possible

Looking at the truth table, can you guess the 3-input gate?

Y = ABC

Multiple-Input Gates

A	B	C	Y
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

Gates with multiple inputs are possible

Looking at the truth table, can you guess the 3-input gate?

$Y = (A + B + C)'$

Bitwise Operations

- All logical operators are applicable to two bit-patterns (group of bits or **bit vectors**) of m bits each, m is any # bits (8, 16, ...)
  - We apply the operation individually to each pair of bits
  - If A and B are **8-bit input sources** (or source operands), then their **AND** or **product**, C, is also 8 bits

C = AB (bit-wise AND)

A	0	0	0	0	1	1	0	1
B	1	1	1	1	1	1	1	1
C								

C = A + B (bit-wise OR)

A	0	0	0	0	1	1	0	1
B	0	0	0	0	0	0	0	0
C								

Bitwise Operations

- All logical operators are applicable to two bit-patterns (group of bits or **bit vectors**) of m bits each, m is any # bits (8, 16, ...)
  - We apply the operation individually to each pair of bits
  - If A and B are **8-bit input sources** (or source operands), then their **AND** or **product**, C, is also 8 bits

C = AB (bit-wise AND)

A	0	0	0	0	1	1	0	1
B	1	1	1	1	1	1	1	1
C	0	0	0	0	1	1	0	1

C = A + B (bit-wise OR)

A	0	0	0	0	1	1	0	1
B	0	0	0	0	0	0	0	0
C	0	0	0	0	1	1	0	1

Bit Masks

	passwd	user id
Packet	0 1 1 0	1 1 0 1

- **B** wants to create a new packet with user id set to **A**'s id and passwd bits set to 0 (e.g., to send a packet to another computer)

Bit Masks

- **Bit mask:** A binary pattern (B) that separates the bits of A into two halves
- Suppose we want to extract the *least significant* 4 bits from A, ignoring the remaining 4 bits

C = AB (bit-wise AND)

A	0	1	1	0	1	1	0	1
B	0	0	0	0	1	1	1	1
C	0	0	0	0	1	1	0	1

Exercises

- Suppose, A = 11000010, and the rightmost two bits are of particular significance. Find a bitmask and a logical operation to mask out the values in the rightmost positions in a new bit pattern B. (**All other bits in B are set to 0.**)

- Suppose, A = 10110010, and the leftmost two bits are of particular significance. Find a bitmask and a logical operation to mask out the values in the leftmost positions in a new bit pattern B. (**All other bits in B are set to 1.**)

Exercise

- How can we find out if two bit-patterns A and B are identical?

- Verify that, 1 AND X = X, where X is a binary variable. Also, verify that, 0 OR X = X.

- Verify that, B AND B = B, where B is a binary variable. Also, verify that, B OR B = B.

- Verify that, B AND B’ = 0, where B is a binary variable. Also, verify that, B OR B’ = 1.

A Useful Circuit: Tutorial # 1

- **What does this circuit do?**

- Multiplexer
- **Used for decision making and often found inside control logic**

Another Useful Circuit

- **What does this circuit do?**

- Half adder (no carry input)
- **Used for making an ALU – Arithmetic & Logic Unit**

Coming Attractions

- We will learn to **systematically build circuits from a specification**

- We **will look at many useful circuits**

- We will study two types of logic circuits – Combinational and sequential