Computing professionals spend considerable time writing efficient code to solve a particular problem in an application domain. With the imminent ending of Moore and Dennard scaling laws, parallelism at the hardware system level has been increasingly utilized to meet performance requirements in almost all systems, including most commodity hardware. This departure from sequential processing demands a more in-depth understanding of the underlying computer architectures. Architecture can no longer be treated as a black box where principles from one can be applied to another. Instead, programmers should look inside the black box and use specific components to improve system performance and energy efficiency.

The Architecture and Organization (AR) Knowledge Area aims to develop a deeper understanding of the hardware environments upon which almost all computing is based and the relevant interfaces provided to higher software layers. The target hardware comprises low-end embedded system processors up to high-end enterprise multiprocessors.

The topics in this knowledge area will benefit students by appreciating the fundamental architectural principles of modern computer systems, including the challenge of harnessing parallelism to sustain performance and energy improvements into the future. In addition, students need to comprehend computer architecture to develop programs that can achieve high performance at low energy consumption. This KA will help computer science students depart from the black box approach and become more aware of the underlying computer system and the efficiencies specific architectures can achieve.

Changes since CS 2013: This KA has changed slightly since the CS2013 report. Changes and additions are summarized as follows:

- Topics have been revised, particularly in the Knowledge Units AR/Memory Hierarchy and AR/Performance and Energy Efficiency. This change supports recent advances in-memory caching and energy consumption.
- To address emerging topics in Computer Architecture, the newly created KU AR/Heterogeneous Architectures covers the introductory level: In-Memory processing (PIM), domain-specific architectures (e.g. neural network processors) - and related topics.
- Quantum computing, particularly the development of quantum processor architectures, has been gathering pace recently. The new KU AR/Quantum Architectures offer a "toolbox" covering introductory topics in this important emerging area.
- Knowledge units have been merged to better deal with overlaps:
  - AR/Multiprocessing and Alternative Architectures were merged into newly created AR/Heterogeneous Architectures.
### Core Hours

<table>
<thead>
<tr>
<th>Knowledge Unit</th>
<th>CS Core</th>
<th>KA Core</th>
</tr>
</thead>
<tbody>
<tr>
<td>Digital Logic and Digital Systems</td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>Machine-Level Data Representation</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Assembly Level Machine Organization</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Memory Hierarchy</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>Interfacing and Communication</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Functional Organization</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>Performance and Energy Efficiency</td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>Heterogeneous Architectures</td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>Quantum Architectures</td>
<td></td>
<td>3</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>9</strong></td>
<td><strong>16</strong></td>
</tr>
</tbody>
</table>

### Knowledge Units

**AR/Digital Logic and Digital Systems**

*Topics [KA Core]*

- Overview and history of computer architecture
- Combinational vs sequential logic/field programmable gate arrays (FPGAs)
  - Fundamental combinational
  - Sequential logic building block
- Functional hardware and software multi-layer architecture
- Computer-aided design tools that process hardware and architectural representations
- High-level synthesis
  - Register transfer notation
  - Hardware description language (e.g. Verilog/VHDL/Chisel)
- System-on-chip (SoC) design flow
- Physical constraints
  - Gate delays
  - Fan-in and fan-out
  - Energy/power
  - Speed of light
Illustrative learning outcomes [KA Core]

- Comment on the progression of computer technology components from vacuum tubes to VLSI, from mainframe computer architectures to the organization of warehouse-scale computers.
- Comment on parallelism and data dependencies between and within components in a modern heterogeneous computer architecture.
- Explain how the “power wall” makes it challenging to harness parallelism.
- Propose the design of basic building blocks for a computer: arithmetic-logic unit (gate-level), registers (gate-level), central processing unit (register transfer-level), and memory (register transfer-level).
- Evaluate simple building blocks (e.g., arithmetic-logic unit, registers, movement between registers) of a simple computer design.
- Validate the timing diagram behavior of a simple processor, identifying data dependency issues.

AR/Machine-Level Data Representation
Topics [CS Core]

- Bits, bytes, and words
- Numeric data representation and number bases
  - Fixed-point
  - Floating-point
- Signed and twos-complement representations
- Representation of non-numeric data
- Representation of records and arrays

Illustrative learning outcomes [CS Core]

- Comment on the reasons why everything is data in computers, including instructions.
- Follow the diagram and annotate the regions where fixed-length number representations affect accuracy and precision.
- Comment on how negative integers are stored in sign-magnitude and twos-complement representations.
- Articulate with plausible justification how different formats can represent numerical data.
- Explain the bit-level representation of non-numeric data, such as characters, strings, records, and arrays.
- Translate numerical data from one format to another.

AR/Assembly Level Machine Organization
Topics [CS Core]

- von Neumann machine architecture
- Control unit; instruction fetch, decode, and execution
- Introduction to SIMD vs. MIMD and the Flynn taxonomy
- Shared memory multiprocessors/multicore organization
Topics [KA Core]
- Instruction set architecture (ISA) (e.g. x86, ARM and RISC-V)
  - Instruction formats
  - Data manipulation, control, I/O
  - Addressing modes
  - Machine language programming
  - Assembly language programming
- Subroutine call and return mechanisms (xref PL/language translation and execution)
- I/O and interrupts
- Heap, static, stack and code segments

Illustrative learning outcomes [CS Core]
- Contextualize the classical von Neumann functional units in embedded systems, particularly on-chip and off-chip memory.
- Comment on how instruction is executed in a classical von Neumann machine, with extensions for threads, multiprocessor synchronization, and SIMD execution.
- Annotate an example diagram with instruction-level parallelism and hazards to comment on how they are managed in typical processor pipelines.

Illustrative learning outcomes [KA Core]
- Comment on how instructions are represented at the machine level and in the context of a symbolic assembler.
- Map an example of high-level language patterns into assembly/machine language notations.
- Comment on different instruction formats, such as addresses per instruction and variable-length vs fixed-length formats.
- Follow a subroutine diagram to comment on how subroutine calls are handled at the assembly level.
- Comment on basic concepts of interrupts and I/O operations.
- Code a simple assembly language program for string processing and manipulation.

AR/Memory Hierarchy
Topics [CS Core]
- Memory hierarchy: the importance of temporal and spatial locality
- Main memory organization and operations
- Persistent memory (e.g. SSD, standard disks)
- Latency, cycle time, bandwidth and interleaving
- Cache memories
  - Address mapping
  - Block size
  - Replacement and store policy
- Multiprocessor cache coherence
- Virtual memory (hardware support, cross-reference OS/Virtual Memory)
• Fault handling and reliability
• Reliability (cross-reference SF/Reliability through Redundancy)
  o Error coding
  o Data compression
  o Data integrity
• Non-von Neumann Architectures
  o In-Memory Processing (PIM)

Illustrative learning outcomes [CS Core]
• Using a memory system diagram, detect the main types of memory technology (e.g., SRAM, DRAM) and their relative cost and performance.
• Measure the effect of memory latency on running time.
• Enumerate the functions of a system with virtual memory management.
• Compute average memory access time under various cache and memory configurations and mixes of instruction and data references.

AR/Interfacing and Communication
Topics [CS Core]
• I/O fundamentals
  o Handshaking and buffering
  o Programmed I/O
  o Interrupt-driven I/O
• Interrupt structures: vectored and prioritized, interrupt acknowledgement
• External storage, physical organization and drives
• Buses fundamentals
  o Bus protocols
  o Arbitration
  o Direct-memory access (DMA)
• Network-on-chip (NoC)

Illustrative learning outcomes [CS Core]
• Follow an interrupt control diagram to comment on how interrupts are used to implement I/O control and data transfers.
• Enumerate various types of buses in a computer system.
• List the advantages of magnetic disks and contrast them with the advantages of solid-state disks.

AR/Functional Organization
Topics [KA Core]
• Implementation of simple datapaths, including instruction pipelining, hazard detection and resolution
• Control unit
- Hardwired implementation
- Microprogrammed realization
- Instruction pipelining
- Introduction to instruction-level parallelism (ILP)

Illustrative learning outcomes [KA Core]
- Validate alternative implementation of datapaths in modern computer architectures.
- Propose a set of control signals for adding two integers using hardwired and microprogrammed implementations.
- Comment on instruction-level parallelism using pipelining and significant hazards that may occur.
- Design a complete processor, including datapath and control.
- Compute the average cycles per instruction for a given processor and memory system implementation.

AR/Performance and Energy Efficiency
Topics [KA Core]
- Performance-energy evaluation (introduction): performance, power consumption, memory and communication costs
- Branch prediction, speculative execution, out-of-order execution, Tomasulo’s algorithm
- Prefetching
- Enhancements for vector processors and GPUs
- Hardware support for Multithreading
  - Race conditions
  - Lock implementations
  - Point-to-point synchronization
  - Barrier implementation
- Scalability
- Alternative architectures, such as VLIW/EPIC, accelerators and other special-purpose processors
- Dynamic voltage and frequency scaling (DVFS)
- Dark Silicon

Illustrative learning outcomes [KA Core]
- Comment on evaluation metrics for performance and energy efficiency.
- Follow a speculative execution diagram and write about the decisions that can be made.
- Build a GPU performance-watt benchmarking diagram.
- Code a multi-threaded Python program that adds (in parallel) elements of two integer vectors.
- Propose a set of design choices for alternative architectures.
- Comment on key concepts associated with dynamic voltage and frequency scaling.
- Measure improvement of energy savings for an 8-bit integer quantization compared to a 32-bit quantization.
AR/Heterogeneous Architectures

Topics [KA Core]
- SIMD and MIMD architectures (e.g. General-Purpose GPUs, TPUs and NPUs)
- Heterogeneous memory system
  - Shared memory versus distributed memory
  - Volatile vs non-volatile memory
  - Coherence protocols
- Domain-Specific Architectures (DSAs)
  - Machine Learning Accelerator
  - In-networking computing
  - Embedded systems for emerging applications
  - Neuromorphic computing
- Packaging and integration solutions such as 3DIC and Chiplets

Illustrative learning outcomes [KA Core]
- Follow a system diagram with alternative parallel architectures, e.g. SIMD and MIMD, and annotate the key differences.
- Tell what memory-management issues are found in multiprocessors that are not present in uniprocessors, and how these issues might be resolved.
- Validate the differences between memory backplane, processor memory interconnect, and remote memory via networks, their implications for access latency and their impact on program performance.
- Tell how you would determine when to use a domain-specific accelerator instead of just a general-purpose CPU.
- Enumerate key differences in architectural design principles between a vector and scalar-based processing unit.
- List the advantages and disadvantages of PIM architectures.

AR/Quantum Architectures

Topics [KA Core]
- Principles
  - The wave-particle duality principle
  - The uncertainty principle in the double-slit experiment
  - What is a Qubit? Superposition and measurement. Photons as qubits.
- Axioms of QM: superposition principle, measurement axiom, unitary evolution
- Single qubit gates for the circuit model of quantum computation: X, Z, H.
- Two qubit gates and tensor products. Working with matrices.
- The No-Cloning Theorem. The Quantum Teleportation protocol.
- Algorithms
  - Simple quantum algorithms: Bernstein-Vazirani, Simon’s algorithm.
  - Implementing Deutsch-Josza with Mach-Zehnder Interferometers.
- Quantum factoring (Shor’s Algorithm)
- Quantum search (Grover’s Algorithm)
- Implementation aspects
  - The physical implementation of qubits (there are currently nine qubit modalities)
  - Classical control of a Quantum Processing Unit (QPU)
  - Error mitigation and control. NISQ and beyond.
- Emerging Applications
  - Post-quantum encryption
  - The Quantum Internet
  - Adiabatic quantum computation (AQC) and quantum annealing

*Illustrative learning outcomes [KA Core]*

- Comment on a quantum object produced as a particle, propagates like a wave and is detected as a particle with a probability distribution corresponding to the wave.
- Follow the diagram and comment on the quantum-level nature that is inherently probabilistic.
- Articulate your view on entanglement that can be used to create non-classical correlations, but there is no way to use quantum entanglement to send messages faster than the speed of light.
- Comment on quantum parallelism and the role of constructive vs destructive interference in quantum algorithms given the probabilistic nature of measurement(s).
- Annotate the code snippet provided the role of quantum Fourier sampling (QFT) in Shor’s algorithm
- Code Shor’s algorithm in a simulator and document your code highlighting the classical components and aspects in Shor’s algorithm
- Enumerate the specifics of each qubit modality (e.g., trapped ion, superconducting, silicon spin, photonic, quantum dot, neutral atom, topological, color center, electron-on-helium, etc.)
- Contextualize the differences between AQC and the gate model of quantum computation and which kind of problems each is better suited to solve
- Comment on the statement: a QPU is a heterogeneous multicore architecture like an FPGA or a GPU

**Professional Dispositions**

- Self-directed: students should increasingly become self-motivated to acquire complementary knowledge from system documentation.
- Proactive: students need to be proactive and independent to navigate and integrate knowledge from different knowledge areas to understand the underlying computer system.
- Inventive: students should look beyond simple solutions to computer architecture design issues and leverage architecture-specific features whenever possible.
- Professional with ethics: students should exercise discretion and behave ethically. Computer systems, particularly embedded sensors, can directly interface with the user's
body (e.g. real-time glucose monitoring), so the user’s safety and privacy are high-priority.

**Math Requirements**

**Required:**
- Discrete Maths: Sets, Relations, Logical Operations, Number Theory
- Linear Algebra: Arithmetic Operations, Matrix operations

**Desired:**
- Maths/Physics for Quantum Computing: basic probability, trigonometry, simple vector spaces, complex numbers, Euler’s formula
- System performance evaluation: probability and factorial experiment design.

**Shared Concepts and Crosscutting Themes**

**Shared Concepts:**
- Virtual memory (AR/Memory Hierarchy) with OS/Memory Management.
- Error coding, data compression, and data integrity (AR/Memory Hierarchy) with (SF/Software Reliability)
- Communications networks (AR/Interfacing and Communication) with (NC/Introduction)
- Processor architecture design (AR/Heterogeneous Architectures with Platform-based Development (PBD/Mobile and PBD/Industrial)
- Domain-specific architectures (AR/Heterogeneous Architectures) with Intelligent Systems (IS/Advanced Machine Learning)
- Emerging hardware user interfaces (AR/Heterogeneous Architectures) with Human-Computer Interaction (HCI/Human Factors & Security)
- The underlying parallel computer system (AR/Assembly Level Machine Organization) with Parallel and Distributed Computing (PD/Parallel Architecture)

**Crosscutting themes:**
- Parallelism at different levels
- Resource allocation, scheduling and isolation
- Performance and Energy Efficiency
- Social and environmental impacts of underlying computer systems

**Course Packaging Suggestions**

**Computer Architecture - Introductory Course** to include the following:
- SEP/History [2 hours]
- AR/Machine-Level Data Representation [2 hours]
- AR/Assembly Level Machine Organization [2 hours]
- AR/Memory Hierarchy [10 hours]
● OS/Memory Management [10 hours]
● AR/Interfacing and Communication [4 hours]
● AR/Heterogeneous Architectures [5 hours]
● PD/Programs and Executions [4 hours]
● SEP/Methods for Ethical Analysis [3 hours]

Pre-requisites:
● Discrete Maths: Sets, Relations, Logical Operations, Number Theory

Skill statement:
● A student who completes this course should be able to understand the fundamental architectures of modern computer systems, including the challenge of memory caches and memory management.

Systems Course - Advanced to include the following:
● SEP/History [2 hours]
● SEP/Privacy and Civil Liberties [3 hours]
● SE/Software Design [System Design] [4 hours]
● PD/Algorithms and applications [4 hours]
● AR/Heterogeneous Architectures [4 hours]
● OS/Roles and Purpose of Operating Systems [3 hours]
● AR/Memory Hierarchy [7 hours]
● AR/Performance and Energy Efficiency [5 hours]
● NC/Networked Applications [5 hours]

Pre-requisites:
● Discrete Math: Sets, Relations, Logical Operations, Number Theory

Skill statement:
● A student who completes this course should be able to appreciate the fundamental architectures of modern computer systems, including the challenge of harnessing parallelism to sustain performance and energy improvements into the future.

Competency Specifications

● **Task 1**: Assess the performance implications of cache memories in your application.
● **Competency Statement**: Critically analyze the performance of an application concerning caching issues and produce a report summarizing key results.
● **Competency area**: Systems / Application
● **Competency unit**: Development / Deployment / Evaluation / Improvement
● **Required knowledge areas and knowledge units**:
  ○ AR/Memory Hierarchy
Task 2: Evaluate the performance-watt of your machine learning model deployed on an embedded device.

**Competency Statement:** Evaluate the performance and power consumption of the embedded device running the code deployed with a machine learning model trained in the cloud.

**Competency area:** Systems / Application

**Competency unit:** Deployment / Evaluation / Improvement

**Required knowledge areas and knowledge units:**
- AR/Performance and Energy Efficiency
- SPD/Mobile Platforms

**Required skill level:** Evaluate / Develop

**Core level:**

Task 3: Develop a version of your CPU-based application to run on a hardware accelerator (GPU, TPU, NPU).

**Competency Statement:** Apply knowledge from systems design to accelerate an application code and evaluate the code speed-up.

**Competency area:** Software / Systems

**Competency unit:** Requirements / Design / Development / Testing / Documentation

**Required knowledge areas and knowledge units:**
- AR/Heterogeneous Architectures
- PD/Parallel Architecture
- SF/System Design

**Required skill level:** Evaluate / Develop

**Core level:**

Task 4: Develop a benchmarking software tool to assess the performance gain in removing I/O bottlenecks in your code.

**Competency Statement:** Appreciate the importance of taking time to develop or use performance tools to improve the performance of application code.

**Competency area:** Software / Systems

**Competency unit:** Development / Evaluation / Maintenance

**Required knowledge areas and knowledge units:**
- AR/Interfacing and Communication
- AR/Performance and Energy Efficiency
- SF/Resources Allocation and Scheduling
- **Task 5**: Apply knowledge of operating systems to assess page faults in CPU-GPU memory management and their performance impact on the accelerated application.
  - **Competency Statement**:
  - **Competency area**: Systems
  - **Competency unit**: Requirements / Design / Development / Integration / Documentation / Evaluation / Maintenance / Management / Consumer Acceptance / Adaptation to social issues / Improvement
  - **Required knowledge areas and knowledge units**:
    - AR/Memory Hierarchy
    - AR/Performance and Energy Efficiency
    - AR/Heterogeneous Architectures
    - OS/Memory Management
  - **Required skill level**: Apply (used to be Solve) / Evaluate
  - **Core level**:

- **Task 6**: Assess data privacy implications in developing a medical device for continuous patient sensor data collection, and write a white paper on data privacy implications.
  - **Competency Statement**: Apply technical knowledge of systems to understand the embedded sensor device and data accuracy vs data minimization and security at the hardware level, and document the data privacy issues.
  - **Competency area**: Systems / Application
  - **Competency unit**: Requirements / Design / Development / Integration / Documentation / Consumer Acceptance / Adaptation to social issues
  - **Required knowledge areas and knowledge units**:
    - AR/Heterogeneous Architectures
    - SEP/Privacy
    - FPL/Embedded Computing and Hardware Interface
  - **Required skill level**: Evaluate
  - **Core level**:

- **Task 7**: Design software modules for sensor hardware integration.
  - **Competency Statement**: Develop high-level software interfaces and low-level hardware interfaces for system integration.
  - **Competency area**: Systems
  - **Competency unit**: Requirements / Design
  - **Required knowledge areas and knowledge units**:
- **Task 8**: Document a system’s design choices and proposed system hardware and software architecture.
  - **Competency Statement**: Gather application requirements and propose a system architecture solution. Write effective technical documentation.
  - **Competency area**: Systems / Application
  - **Competency unit**: Requirements / Design / Documentation
  - **Required knowledge areas and knowledge units**:
    - AR/Machine-Level Data Representation
    - AR/Performance and Energy Efficiency
    - AR/Heterogeneous Architectures
    - SEP/Professional Communication
    - SE/Product Requirements
  - **Required skill level**: Evaluate / Explain
  - **Core level**: 

- **Task 9**: Work on a multidisciplinary team to effectively develop a sensing-actuator robotics arm for an automated manufacturing cell.
  - **Competency Statement**: Cooperate effectively with team members on the design, development and evaluation of a system.
  - **Competency area**: Software / Systems / Application
  - **Competency unit**: Requirements / Design / Development / Testing / Deployment / Integration / Documentation / Evaluation
  - **Required knowledge areas and knowledge units**:
    - AR/Heterogeneous Architectures
    - SPD/Robot Platforms
    - SE/Teamwork
  - **Required skill level**: Apply (used to be Solve) / Develop
  - **Core level**: 

- **Task 10**: Develop a program explicitly exploiting the underlying CPU cores and memory management system for improved performance.
- **Competency Statement:** Understand the computer architecture principles, parallelism and how memory is managed for efficient data sharing.
- **Competency area:** Systems
- **Competency unit:** Requirements / Design / Development / Required knowledge areas and knowledge units:
  - AR/Assembly Level Machine Organization
  - AR/Memory Hierarchy
- **Required skill level:** Evaluate / Develop
- **Core level:**

---

**Committee**

**Chair:** Marcelo Pias, Federal University of Rio Grande (FURG), Brazil

**Members:**
- Brett A. Becker, University College Dublin, Dublin, Ireland
- Mohamed Zahran, New York University, NY, USA
- Monica D. Anderson, University of Alabama, Tuscaloosa, AL, USA
- Qiao Xiang, Xiamen University, China
- Adrian German, Indiana University, Bloomington, IN, USA

---

**Appendix: Core Topics and Skill Levels**

<table>
<thead>
<tr>
<th>KA</th>
<th>KU</th>
<th>Topic</th>
<th>Skill</th>
<th>Core</th>
<th>Hour(s)</th>
</tr>
</thead>
</table>
| AR | Digital Logic and Digital Systems | ● Overview and history of computer architecture  
    ● Combinational vs. sequential logic  
      ○ Fundamental combinational  
      ○ Sequential logic building block  
    ● Functional hardware and software multi-layer architecture | Explain   |      |         |
|    | Digital Logic and Digital Systems | ● Computer-aided design tools that process hardware and architectural representations  
    ● High-level synthesis  
      ○ Register transfer notation  
      ○ Hardware description language (e.g. Verilog/VHDL/Chisel) | Evaluate  | KA   | 3       |
| AR | Machine-Level Data Representation | - System-on-chip (SoC) design flow  
  - Physical constraints  
    - Gate delays  
    - Fan-in and fan-out  
    - Energy/power [Shared with SPD]  
    - Speed of light  
| A | Apply | CS | 1 |
| AR | Assembly Level Machine Organization | - Bits, bytes, and words  
  - Numeric data representation and number bases  
    - Fixed-point  
    - Floating-point  
  - Signed and two's-complement representations  
  - Representation of non-numeric data  
  - Representation of records and arrays  
| A | Explain | CS | 1 |
| AR | Assembly Level Machine Organization | - von Neumann machine architecture  
  - Control unit; instruction fetch, decode, and execution  
  - Introduction to SIMD vs. MIMD and the Flynn taxonomy  
  - Shared memory multiprocessors/multicore organization [Shared with OS]  
| A | Develop | KA | 2 |
| A | | - Instruction set architecture (ISA) (e.g. x86, ARM and RISC-V)  
  - Instruction formats  
  - Data manipulation, control, I/O  
  - Addressing modes  
  - Machine language programming  
  - Assembly language programming  
  - Subroutine call and return mechanisms (xref PL/language translation and execution) [Shared with OS]  
  - I/O and interrupts [Shared with OS]  
  - Heap, static, stack and code segments [Shared with OS]  
| A | Explain | | |
| A | | - Memory hierarchy: the importance of temporal and spatial locality [Shared with OS]  
  - Main memory organization and operations  
  - Persistent memory (e.g. SSD, standard disks) [Shared with OS]  
| A | | | |
| AR | Memory Hierarchy | - Latency, cycle time, bandwidth and interleaving  
- Virtual memory (hardware support, cross-reference OS/Virtual Memory) [Shared with OS]  
- Fault handling and reliability [Shared with OS]  
- Reliability (cross-reference SF/Reliability through Redundancy)  
  - Error coding  
  - Data compression  
  - Data integrity  
- Non-von Neumann Architectures  
  - In-Memory Processing (PIM)  
- Cache memories [Shared with OS]  
  - Address mapping  
  - Block size  
  - Replacement and store policy  
- Multiprocessor cache coherence | CS | 6 |
|---|---|---|---|
| AR | Interfacing and Communication | - I/O fundamentals [Shared with OS and SPD]  
  - Handshaking and buffering  
  - Programmed I/O  
  - Interrupt-driven I/O  
- Interrupt structures: vectored and prioritized, interrupt acknowledgement [Shared with OS]  
- External storage, physical organization and drives [Shared with OS]  
- Buses fundamentals [Shared with OS and SPD]  
  - Bus protocols  
  - Arbitration  
  - Direct-memory access (DMA)  
- Network-on-chip (NoC) | Explain | CS | 1 |
| AR | Functional Organization | - Implementation of simple datapaths, including instruction pipelining, hazard detection and resolution  
- Control unit  
  - Hardwired implementation  
  - Microprogrammed realization  
- Instruction pipelining  
- Introduction to instruction-level parallelism (ILP) | Develop | KA | 2 |
| AR | Performance and Energy Efficiency | • Performance-energy evaluation (introduction): performance, power consumption, memory and communication costs  
• Branch prediction, speculative execution, out-of-order execution, Tomasulo's algorithm  
• Prefetching | Evaluate | KA | 2 |
| AR | Performance and Energy Efficiency | • Enhancements for vector processors and GPUs  
• Hardware support for Multithreading  
  o Race conditions  
  o Lock implementations  
  o Point-to-point synchronization  
  o Barrier implementation  
• Scalability  
• Alternative architectures, such as VLIW/EPIC, accelerators and other special-purpose processors  
• Dynamic voltage and frequency scaling (DVFS)  
• Dark Silicon | Explain | KA | 1 |
| AR | Heterogeneous Architectures | • SIMD and MIMD architectures (e.g. General-Purpose GPUs, TPUs and NPUs)  
• Heterogeneous memory system  
  o Shared memory versus distributed memory  
  o Volatile vs non-volatile memory  
  o Coherence protocols  
• Domain-Specific Architectures (DSAs)  
  o Machine Learning Accelerator  
  o In-networking computing  
  o Embedded systems for emerging applications  
  o Neuromorphic computing  
• Packaging and integration solutions such as 3DIC and Chiplets | Explain | KA | 3 |
| AR | Quantum Architectures | • Principles  
  o The wave-particle duality principle  
  o The uncertainty principle in the double-slit experiment  
  o What is a Qubit? Superposition and measurement. Photons as qubits. | Explain | KA | 3 |
- Axioms of QM: superposition principle, measurement axiom, unitary evolution
- Single qubit gates for the circuit model of quantum computation: X, Z, H.
- Two qubit gates and tensor products. Working with matrices.
- The No-Cloning Theorem. The Quantum Teleportation protocol.
- Algorithms
  - Simple quantum algorithms: Bernstein-Vazirani, Simon’s algorithm.
  - Implementing Deutsch-Josza with Mach-Zehnder Interferometers.
  - Quantum factoring (Shor’s Algorithm)
  - Quantum search (Grover’s Algorithm)
- Implementation aspects
  - The physical implementation of qubits (there are currently nine qubit modalities)
  - Classical control of a Quantum Processing Unit (QPU)
  - Error mitigation and control. NISQ and beyond.
- Emerging Applications
  - Post-quantum encryption
  - The Quantum Internet
  - Adiabatic quantum computation (AQC) and quantum annealing