SENG 21213

Computer Architecture and Operating Systems

12-Lecture Series Plan · 3 Hours per Lecture

Lecture Series Summary

L#	Title	Key Theme	OS Stage
L01	Computer System Fundamentals	Basic elements	Architecture
L02	Memory Hierarchy & I/O Architecture	Cache memory	Architecture
L03	Data Representation & Combinational Circuits	Number systems	Architecture
L04	Sequential Circuits & Computer Architectures	Flip-flops	Architecture
L05	Processor Architecture & Instruction Set Architecture	CPU internals	Architecture
L06	Pipelining, Branch Prediction & Parallelism	Instruction pipelining	Architecture
L07	Assembly Language & Hardware-Software Integration	Machine code	Architecture
L08	Operating System Overview & Process Models	OS objectives	Milestone L08
L09	Process Description, Control & Scheduling	PCBs	Milestone L09
L10	Threads & Concurrency Control	Thread model	Milestone L10
L11	Deadlocks, Starvation & Memory Management	Deadlock conditions	Milestone L11
L12	File Systems, Protection Levels & Future Trends	File system design	Milestone L12

Course Information

Course Code	SENG 21213
Course Title	Computer Architecture and Operating Systems
Pre-requisites	SENG 11213, SENG 11223
Contact Hours	12 Lectures × 3 Hours = 36 Contact Hours

Assessment	End-of-Semester Examination + Continuous Assessment (OS Assignment)
Teaching Method	Lectures, Supervised Practical Sessions, Assignment Milestones

Course Learning Outcomes

- Explain how data and programs are represented in computers
- Design combinational circuits to make logical decisions
- Design sequential circuits to perform sequences of actions
- Compare and contrast different computer architectures
- Describe the functionality and working of computer system building blocks
- Demonstrate understanding of assemblers and programming in assembly language
- Explain the key roles played by an operating system
- Identify the major components of operating systems
- Describe the concepts, models, and approaches involved in OS design

OS Assignment: Build Your Own OS

Starting from Lecture 8, students will progressively build a working 32-bit x86 operating system from bare metal. By Lecture 12, the OS will run in QEMU with a full feature set.

Milestone	Lecture	Feature	New Files
Stage 0	L08	Boot + VGA Display + Interactive Shell	boot.asm, kernel.c, vga.c, keyboard.c (Given)
Stage 1	L09	Process Management + Round-Robin Scheduler	process.h, process.c, scheduler.c
Stage 2	L10	Kernel Threads + Mutex + Semaphore	thread.h, thread.c, mutex.c
Stage 3	L11	Physical Memory Manager + kmalloc/kfree	pmm.h, pmm.c, vmm.c
Stage 4	L12	RAM Disk + FAT-style File System	fs.h, fs.c, ramdisk.c

Lecture 01 • Computer System Fundamentals

Basic elements, instruction execution, interrupts & the fetch-execute cycle

Duration: 3 Hours	References: Notes 1 & 2	Reading: Stallings Comp Org Ch.1

Topics Covered

- ► Course overview and assessment structure
- ► Basic elements: Processor, Main Memory, I/O Modules, System Bus
- ► Internal registers: MAR, MBR, I/OAR, I/OBR
- ► Memory module structure and address spaces
- ► Instruction Execution: the fetch–decode–execute cycle
- ► Instruction categories: data movement, arithmetic, control
- ► Interrupts: purpose, types, handling mechanism
- ► Interrupt service routine (ISR) and the interrupt cycle

Learning Outcomes

- ► Describe the fundamental components of a computer system
- ► Explain how the CPU executes instructions step by step
- ► Differentiate between classes of interrupts
- ► Trace a simple instruction through the fetch-execute cycle

Practical / Lab Activities

- ► Lab setup: Docker, QEMU, NASM, GCC toolchain installation
- ► Examine the SENG21213-OS Stage 0 source code
- ► Build and boot the initial OS in QEMU
- ► Observe Protected Mode switch in boot.asm using debugger

💻 **OS Assignment: Stage 0 distributed. Students build and run the initial OS. Explore boot.asm and understand the Real-to-Protected Mode transition.**

📌 *Recommended Reading: Stallings Comp Org Ch.1; Course Notes 1 (Basic Elements)*

Lecture 02 • Memory Hierarchy & I/O Architecture

Cache memory, memory hierarchy, bus architectures, I/O systems

Duration: 3 Hours	References: Notes 1 & 2	Reading: Stallings Comp Org Ch.4–5

Topics Covered

- ▶ Memory hierarchy: registers → cache → RAM → disk
- ▶ Trade-offs: capacity vs. access time vs. cost
- ▶ Cache memory: hit ratio, locality of reference
- ▶ Average access time calculations (two-level memory formula)
- ▶ DRAM vs SRAM technologies
- ▶ Bus architecture: address, data, control buses
- ▶ Synchronous vs asynchronous bus protocols
- ▶ I/O techniques: programmed I/O, interrupt-driven I/O, DMA
- ▶ I/O address register and buffer register operation

Learning Outcomes

- ▶ Calculate average memory access time given a hit ratio
- ▶ Compare memory technologies across the hierarchy
- ▶ Describe bus arbitration and data transfer protocols
- ▶ Distinguish between three I/O techniques

Practical / Lab Activities

- ▶ Modify vga.c: read and understand VGA memory-mapped I/O (0xB8000)
- ▶ Implement vga_draw_box() stub using direct VGA addressing
- ▶ Experiment with outb() port I/O in keyboard.c

💻 OS Assignment: Students explore memory-mapped I/O in vga.c. Complete the vga_draw_box() function. Understand how the kernel writes directly to hardware memory.

📌 *Recommended Reading: Stallings Comp Org Ch.4–5; Course Notes 2 (Memory Hierarchy)*

Lecture 03 • Data Representation & Combinational Circuits

Number systems, binary arithmetic, Boolean algebra, logic gates

Duration: 3 Hours	References: Lecture preparation	Reading: Stallings Comp Org Ch.9–10

Topics Covered

- ► Number systems: binary, octal, hexadecimal, BCD
- ► Number conversions and arithmetic operations
- ► Signed integers: sign-magnitude, two's complement, one's complement
- ► Floating-point representation: IEEE 754 single and double precision
- ► Character encoding: ASCII, Unicode
- ► Boolean algebra: axioms, theorems, De Morgan's laws
- ► Logic gates: AND, OR, NOT, NAND, NOR, XOR, XNOR
- ► Combinational circuit design: truth tables, K-maps, SOP/POS
- ► Half adder, full adder, ripple carry adder
- ► Multiplexers, decoders, encoders

Learning Outcomes

- ► Convert numbers between all common bases
- ► Represent integers and floats in binary
- ► Apply Boolean algebra to simplify logic expressions
- ► Design and analyse combinational circuits

Practical / Lab Activities

- ► Write a C function to display hex values using vga_printf() in the OS shell
- ► Add a 'hexdump' command to kernel shell that prints memory in hex
- ► Implement binary and hex input parsing in the shell

💻 **OS Assignment: Add a 'mem hexdump' command to the OS shell that reads memory addresses and displays their contents in hexadecimal — reinforcing data representation.**

📌 *Recommended Reading: Stallings Comp Org Ch.9–10; Comer Ch.3*

Lecture 04 • Sequential Circuits & Computer Architectures

Flip-flops, registers, FSMs, and comparative processor architectures

Duration: 3 Hours	References: Lecture preparation	Reading: Stallings Comp Org Ch.12, 13

Topics Covered

- ▶ Latches and flip-flops: SR, D, JK, T
- ▶ Sequential circuit analysis: state tables, state diagrams
- ▶ Registers and shift registers
- ▶ Counters: synchronous, asynchronous, ring
- ▶ Finite State Machines (FSM): Mealy vs Moore
- ▶ Instruction Set Architectures: CISC vs RISC
- ▶ Register-memory vs register-register architectures
- ▶ Von Neumann vs Harvard architecture
- ▶ Comparative study: x86, ARM, RISC-V, MIPS
- ▶ Architectural constraints: power, area, performance

Learning Outcomes

- ▶ Analyse and design sequential circuits using FSMs
- ▶ Compare major CPU architecture families
- ▶ Justify architectural choices for embedded vs desktop systems
- ▶ Contrast CISC (x86) and RISC (ARM) design philosophies

Practical / Lab Activities

- ▶ Examine the GDT structure in boot.asm (segmented architecture)
- ▶ Trace the CPU state FSM (Real Mode → Protected Mode) in the bootloader
- ▶ Research ARM Cortex-M vs x86 architectural differences

💻 **OS Assignment: Students examine the GDT descriptor format in boot.asm and document the bit fields for each segment. Relate to ISA design choices.**

📌 *Recommended Reading: Stallings Comp Org Ch.12, 13; Comer Ch.7–8*

Lecture 05 • Processor Architecture & Instruction Set Architecture

CPU internals, ISA design, addressing modes, and the ALU

Duration: 3 Hours	References: Lecture preparation	Reading: Stallings Comp Org Ch.11, 14

Topics Covered

- ► CPU internal organisation: ALU, control unit, registers
- ► Instruction format: opcode, operand fields
- ► Addressing modes: immediate, direct, indirect, register, indexed, relative
- ► Instruction types: data transfer, arithmetic/logic, control flow, I/O
- ► Stack architecture and function call conventions
- ► Condition codes and flags register (EFLAGS in x86)
- ► Microoperations: fetch, indirect, execute, interrupt cycles
- ► Control unit design: hardwired vs microprogrammed

Learning Outcomes

- ► Decode instruction formats for a simple ISA
- ► Apply all major addressing modes
- ► Describe the control unit's role in instruction execution
- ► Trace a function call through the stack

Practical / Lab Activities

- ► Examine kernel_entry.asm: identify call convention, stack setup
- ► Write an inline assembly function in kernel.c to read EFLAGS
- ► Decode sample x86 opcodes using Intel reference manual

💻 **OS Assignment: Students write an inline assembly utility in kernel.c to read the CPU's EFLAGS register and display it via vga_printf() — applying ISA knowledge to real kernel code.**

📌 *Recommended Reading: Stallings Comp Org Ch.11, 14; Comer Ch.5–6*

Lecture 06 • Pipelining, Branch Prediction & Parallelism

Instruction pipelining, hazards, branch prediction strategies, and parallel architectures

Duration: 3 Hours	References: Lecture preparation	Reading: Stallings Comp Org Ch.14

Topics Covered

- ► Instruction pipelining: concept, stages, pipeline diagram
- ► Pipeline performance: speedup, throughput, efficiency
- ► Pipeline hazards: structural, data, control hazards
- ► Hazard resolution: stalling, forwarding, out-of-order execution
- ► Branch prediction: static vs dynamic prediction
- ► Dynamic prediction: 1-bit and 2-bit prediction schemes, BTB
- ► Speculative execution and branch target buffer
- ► Superscalar and VLIW architectures
- ► Multi-core and SMP parallelism overview
- ► Flynn's taxonomy: SISD, SIMD, MISD, MIMD

Learning Outcomes

- ► Calculate pipeline speedup and identify bottlenecks
- ► Identify and resolve pipeline hazards
- ► Compare branch prediction strategies
- ► Classify parallel computing architectures using Flynn's taxonomy

Practical / Lab Activities

- ► Simulate a 5-stage pipeline on paper with a given instruction sequence
- ► Add a simple statistics counter to kernel (instructions executed approximation)

💻 **OS Assignment: Students add a kernel-mode CPU tick counter using RDTSC (Read Time-Stamp Counter) assembly instruction to measure performance. Understand hardware performance counters.**

📌 *Recommended Reading: Stallings Comp Org Ch.14; Comer Ch.9*

Lecture 07 • Assembly Language & Hardware-Software Integration

Machine code, assemblers, assembly programming, and calling conventions

Duration: 3 Hours	References: Lecture preparation	Reading: Stallings Comp Org Ch.11

Topics Covered

- ► Machine language: binary encoding of instructions
- ► Assembler directives: ORG, BITS, DB, DW, DD, EQU, SECTION
- ► Registers in x86: EAX, EBX, ECX, EDX, ESI, EDI, ESP, EBP, EIP
- ► NASM syntax: mov, add, sub, mul, div, jmp, jz, jnz, call, ret
- ► Stack operations: push, pop, call, ret conventions
- ► Interfacing assembly and C: cdecl calling convention
- ► System calls: concept, software interrupt, SYSENTER
- ► Privileged modes: Ring 0 (kernel) vs Ring 3 (user)
- ► Interrupt Descriptor Table (IDT) overview

Learning Outcomes

- ► Read and write basic x86 NASM assembly
- ► Implement simple functions in assembly and call them from C
- ► Explain the cdecl calling convention
- ► Distinguish privileged and unprivileged CPU modes

Practical / Lab Activities

- ► Write a NASM function that computes factorial and call it from kernel.c
- ► Implement a software interrupt handler (INT 0x80 style) in the OS
- ► Add privileged mode checking to the kernel shell

💻 **OS Assignment: Students implement a basic system call interface: a software interrupt (INT 0x80) handler that switches from Ring 3 to Ring 0 — the foundation for user-mode processes in L09.**

📌 *Recommended Reading: Stallings Comp Org Ch.11; Comer Ch.6; NASM Manual*

Lecture 08 • Operating System Overview & Process Models

OS objectives, roles, types, multiprogramming, time-sharing, and process abstraction

Duration: 3 Hours	References: Notes 3, 4, 5	Reading: Stallings OS Ch.1–2

Topics Covered

- ► OS objectives: convenience, efficiency, ability to evolve
- ► OS as interface: user → applications → OS → hardware
- ► OS services: program development, execution, I/O, file access
- ► OS kernel vs user space
- ► Batch, multiprogramming, and time-sharing systems
- ► Real-time and distributed operating systems
- ► Process concept: definition, process image
- ► Process states: New → Ready → Running → Blocked → Terminated
- ► Process Control Block (PCB): all fields and their meaning
- ► Context switching: what is saved, and when

Learning Outcomes

- ► Explain the three core OS objectives
- ► Distinguish batch, multiprogramming, and time-sharing paradigms
- ► Describe all fields in a Process Control Block
- ► Trace a context switch between two processes

Practical / Lab Activities

- ► Design the PCB struct for SENG21213-OS (process.h)
- ► Implement process_create() that initialises a PCB
- ► Add the 'ps' command to display a list of PCBs

💻 OS Assignment: Milestone L09 begins. Students define struct pcb_t and implement process_create(). The 'ps' shell command displays all registered processes with state, PID, and name.

📌 *Recommended Reading: Stallings OS Ch.1–2; Course Notes 3 (OS Overview); Notes 4*

Lecture 09 • Process Description, Control & Scheduling

PCBs, process lifecycle, OS control tables, scheduling algorithms

Duration: 3 Hours	References: Notes 4 & 5	Reading: Stallings OS Ch.3–4

Topics Covered

- ► OS control structures: memory, I/O, file, and process tables
- ► Process image in virtual memory: stack, heap, data, text
- ► Process creation and termination events
- ► Five-state process model (and the suspended states variant)
- ► Seven-state model: Ready/Suspend, Blocked/Suspend
- ► Scheduling levels: long-term, medium-term, short-term
- ► Scheduling criteria: throughput, turnaround, waiting time, response time
- ► Scheduling algorithms: FCFS, SJF, Round Robin, Priority, Multilevel Queue
- ► Dispatcher and context switch overhead

Learning Outcomes

- ► Model process lifecycle using state diagrams
- ► Implement a PCB-based process table
- ► Compare CPU scheduling algorithms for different workloads
- ► Calculate scheduling metrics: average waiting time, turnaround time

Practical / Lab Activities

- ► Implement round-robin scheduler in scheduler.c
- ► Implement process_yield() using inline assembly (save/restore registers)
- ► Test with two co-operative processes printing alternating messages

💻 **OS Assignment: Complete Milestone L09: working round-robin scheduler. Two kernel processes alternate execution via cooperative yielding. 'ps' shows live state transitions.**

📌 *Recommended Reading: Stallings OS Ch.3–4; Course Notes 4 & 5*

Lecture 10 • Threads & Concurrency Control

Thread model, kernel vs user threads, race conditions, mutual exclusion, semaphores

Duration: 3 Hours	References: Notes 6 & 7	Reading: Stallings OS Ch.4 (Threads)

Topics Covered

- ► Thread vs process: resource ownership vs execution
- ► Benefits of threads: responsiveness, resource sharing, economy
- ► User-level threads vs kernel-level threads
- ► Thread states and lifecycle
- ► Concurrency: definition, challenges, race conditions
- ► Critical section problem and requirements for a solution
- ► Mutual exclusion mechanisms: spinlock, test-and-set
- ► Semaphores: binary and counting; wait() and signal()
- ► Monitors and condition variables
- ► Classic problems: Producer-Consumer, Readers-Writers, Dining Philosophers

Learning Outcomes

- ► Distinguish threads from processes and justify when to use each
- ► Identify race conditions in concurrent code
- ► Implement mutex and semaphore primitives
- ► Apply synchronisation to solve classic concurrency problems

Practical / Lab Activities

- ► Implement kernel threads in thread.c (cooperative scheduling)
- ► Implement spinlock using x86 LOCK XCHG instruction
- ► Implement binary semaphore with wait/signal operations
- ► Demonstrate race condition fix using mutex in the OS

💻 OS Assignment: Milestone L10: kernel threads with mutex. Students implement a producer-consumer demo running in two kernel threads with a shared buffer protected by a semaphore.

📌 *Recommended Reading: Stallings OS Ch.4 (Threads); Ch.5 (Concurrency); Course Notes 6 & 7*

Lecture 11 • Deadlocks, Starvation & Memory Management

Deadlock conditions, prevention/avoidance, and memory management strategies

Duration: 3 Hours	References: Notes 8 & 9	Reading: Stallings OS Ch.7–8

Topics Covered

- ► Deadlock: definition, necessary conditions (Coffman conditions)
- ► Resource allocation graphs and cycle detection
- ► Deadlock prevention: eliminate one of the four conditions
- ► Deadlock avoidance: Banker's Algorithm
- ► Deadlock detection and recovery
- ► Starvation: causes and prevention (ageing)
- ► Memory management requirements: relocation, protection, sharing, logical organisation
- ► Fixed partitioning, dynamic partitioning, external/internal fragmentation
- ► Paging: page table, frame, logical vs physical address translation
- ► Segmentation: segment table, protection, sharing
- ► Virtual memory: demand paging, page fault, page replacement algorithms
- ► LRU, FIFO, Optimal page replacement; Belady's anomaly

Learning Outcomes

- ► Identify deadlock conditions using resource allocation graphs
- ► Apply Banker's Algorithm for deadlock avoidance
- ► Compute logical-to-physical address translation with paging
- ► Compare page replacement algorithms and calculate page fault rates

Practical / Lab Activities

- ► Implement a physical memory manager (buddy allocator or bitmap allocator)
- ► Implement kmalloc() / kfree() stubs
- ► Add 'free' command to OS shell showing memory usage
- ► Implement a simple deadlock detector for the process table

💻 **OS Assignment: Milestone L11: Physical Memory Manager. Students implement a bitmap-based page allocator (pmm.c). 'free' command shows total/used/free pages. Optional: detect circular wait in process table.**

📌 *Recommended Reading: Stallings OS Ch.7–8; Course Notes 8 (Deadlock) & 9 (Memory Management)*

Lecture 12 • File Systems, Protection Levels & Future Trends

File system design, access control, notable architectures, and emerging trends

Duration: 3 Hours	References: Lecture preparation	Reading: Stallings OS Ch.11–12

Topics Covered

- ► File system concepts: files, directories, metadata
- ► File system operations: create, read, write, seek, delete
- ► Allocation methods: contiguous, linked, indexed (i-node)
- ► FAT file system: cluster chains, directory entries
- ► Buffer cache and file system performance
- ► Privileged modes: CPU rings, kernel mode vs user mode
- ► Protection mechanisms: capability lists, access control matrices
- ► Protection levels in x86: CPL, DPL, RPL
- ► Notable architectures: IBM System/360, RISC-V, Apple M-series
- ► Current trends: heterogeneous computing, persistent memory (NVM), quantum computing outlook

Learning Outcomes

- ► Design a simple flat file system for a RAM disk
- ► Explain file allocation methods and their trade-offs
- ► Describe CPU privilege rings and why they matter for OS security
- ► Summarise current and emerging trends in computer architecture

Practical / Lab Activities

- ► Implement a RAM disk in the OS (fixed-size byte array in BSS)
- ► Implement a FAT-like file system: format, create, write, read
- ► Add 'ls', 'cat', 'touch' commands to the OS shell
- ► Final milestone: full OS demo with all features running in QEMU

💻 **OS Assignment: Milestone L12 (Final): Complete OS demo. Students present their running OS in QEMU featuring: multi-process scheduling, thread synchronisation, memory allocation, and a working file system.**

📌 *Recommended Reading: Stallings OS Ch.11–12; Comer Ch.15; Course Notes; OSDev Wiki*

Recommended Reading

Primary OS Text: Stallings, W. (2014). Operating Systems: Internals and Design Principles, 8th Ed. Pearson.

Primary Architecture Text: Stallings, W. (2011). Computer Organization and Architecture: Designing for Performance, 10th Ed. Pearson.

Supplementary: Comer, D.E. (2004). Essentials of Computer Architecture. Pearson.

Online Reference: OSDev Wiki – https://wiki.osdev.org

Assembly Reference: NASM 2.x Documentation – https://nasm.us/doc

QEMU Documentation: https://www.qemu.org/docs/master