Main Points of the Computer Organization and System Software Module

You can find below the topics we have covered during the COSS module. Reading the relevant parts of the textbooks is essential for a good exam result. Revising the exercises and solving exam questions from previous years, see

`http://www.dcs.bbk.ac.uk/~szabolcs/compsys.html`

are highly recommended as part of the preparation for the exam.

Computer Architecture

Instruction Sets

- Machine instruction characteristics:
  - Elements: operation code, source and result operand references, next instruction reference (often implicit); location of operands: memory, register, I/O device
  - Representation: binary, symbolic (e.g. ADD)
  - Instruction types: data processing (arithmetic and logical), movement (I/O) and storage (memory), program flow control (test and branch)
  - Number of addresses: typically between three and zero; implicit addressing (use of accumulator)
  - Instruction set design: operation repertoire, data types, instruction format (length, no. of addresses), registers (which can be referenced by instructions), addressing modes
- Types of operands: addresses: immediate, direct, indirect, displacement, stack; numbers: integer or fixed point, floating point, decimal; characters; logical data
- Types of operations:
  - data transfer: move, store, load, set, push, pop
  - arithmetic: add, subtract, multiply, divide
  - logical: and, not, test, shift, rotate
  - I/O: read, write
  - system control: read or alter control register — kernel mode
  - transfer of control: branch, return, skip; procedure call — use of stack
- Assembly language:

CPU Structure and Function

- Processor organization: registers, ALU unit, control unit
- Interconnection structures: bus; control, address and data lines
- Register organization:
  - user-visible registers: general purpose, data, address (segment pointers, stack pointer), condition codes/flags (positive, negative, zero, equal)

  - control and status registers: program counter PC, instruction register IR, memory registers MAR and MBR, program status word PSW (flags, interrupts enabled, kernel mode)

- Instruction cycle: instruction and data fetch and decode, execute, interrupt; indirect cycle for address calculation; data flow involving registers and bus lines
- Instruction pipelining: overlapping the stages of instruction cycles; prefetching instructions
  - timing diagrams, the effect of dependencies and conditional branching; six-stage pipeline (FI, DI, CO, FO, EI, WO);
  - increased complexity of control logic — dealing with branches: loop buffer (high speed memory containing last few instructions), branch prediction (by opcode, dynamically by logging history)
- Overview of Pentium: register organization, interrupt processing

Reduced Instruction Set Computers

- Instruction execution characteristics: operations (lot of assignments, time-consuming procedure calls), operands (mainly numeric variables)
- RISC characteristics:
  - register-to-register operations
  - simple addressing modes
  - simple instruction formats (single size)
  - small set of instructions
- Use of registers:
  - register windows: parameter, local and temporary registers, circular organization with overlaps, pointers keep track of active procedures
  - compiler based optimization: allocation of physical registers to logical registers, graph coloring according to the active use of registers
- RISC pipelining: re-ordering independent instructions
- Overview of SPARC

Instruction-Level Parallelism and Superscalar Processors

- Independent and concurrent execution of instructions in different pipelines
- Dependencies: true data (write-read), procedural, output (write-write) and antidependencies (read-write), resource conflicts
- Design issues: instruction-level and machine parallelisms
  - instruction issue policies (in-order/out-of-order issue, in-order/out-of-order completion)
  - register renaming to remove output and antidependencies
  - branch prediction
  - superscalar execution: instruction fetch and branch prediction, re-ordering instructions according to dependencies, execution, committing/retiring instructions (re-order according to program logic); speedup (importance of register renaming)
- Overview of Pentium 4

Operating Systems

Processes and Threads

Processes

- Definition: program in execution; usage: multiprogramming, pseudo-parallelism; creation; termination; system calls in UNIX and Windows
- States: running, blocked, ready; possible transitions between states; process switch; modes: kernel and user modes, mode switch
- Implementation: process table, process control block

Threads (aka Lightweight Processes)

- Refined process model: resource grouping + thread of execution
- Multithreading: multiple execution in the same process (memory) environment for co-operation by sharing resources
- Modelling threads: multiple threads inside one process; per process data (address space, open files, global variables, etc.) and per thread data (program counter, registers, stack, state, etc.)
- Advantages of multithreading: pseudo-parallelism with shared address space and data, less overhead with switching than with processes, better performance for I/O-bound applications
- Examples: 1) word processor with interactive and background threads, 2) multithreaded web server with dispatcher and worker threads
- Implementations (and their relative advantages):
  - in user space (+: fast context switches, process-specific scheduling algorithms; -: problems with blocking system calls)
  - in kernel (+: easier semantics for blocking system calls; -: costly context switches)
- Microkernels

Scheduling

- When to schedule: process creation and termination, blocking system calls, interrupts
- Goals:
  - general (fairness, policy enforcement, balance)
  - batch systems (throughput, turnaround time)
  - interactive systems (response time)
  - real-time systems (meeting deadlines)
- Preemptive and non-preemptive algorithms
- Scheduling in batch systems:
  - first-come-first-served (FCFS),
  - shortest-job-first (SJF),
  - shortest-remaining-time-next (SRTN)
- Scheduling in interactive systems:
  - round-robin scheduling,
  - priority scheduling (multiple queues),
  - shortest-process-next (aging, weighted average),
- Comparison of the different scheduling algorithms with respect to the scheduling goals; policy versus mechanism
- Thread scheduling: user space and kernel threads
- Scheduling in UNIX and in Windows

Concurrency

Interprocess Communication

- Race condition between concurrent cooperating processes or threads that share data, examples (spooling directory), consequences
- Critical-section problem; solution should satisfy
  1. mutual exclusion,
  2. no assumption on speed of processes,

3. progress (no deadlock),
4. bounded waiting (no starvation)

- Solutions for two processes using busy waiting
  - software: Peterson's solution
  - help from hardware: test-and-set lock (TSL), exchange (swap)
- Semaphores:
  - definition: integer variable with *up* and *down* (or *signal* and *wait*) operations, sleep and wake up, queues
  - usage: eliminates busy waiting, but possible starvation and deadlock
- Monitors (with condition variables)
- Solutions for classical synchronization problems avoiding busy waiting
  - bounded-buffer (producer–consumer)
  - readers–writers
  - dining philosophers
- Interprocess communication (IPC) via message passing as an alternative to shared memory

Deadlock

- Definition: a set of processes is deadlocked if each process in the set is waiting for an event that only another process in the set can cause
- Examples; usually deadlock may occur when processes are granted exclusive access to resources (hardware or software)
- Characterization; the four necessary conditions for deadlock to occur:
  1. mutual exclusion
  2. hold and wait
  3. no preemption
  4. circular wait
- Modelling deadlocks
  - one resource of each type: resource allocation graph using process and resource nodes and arrows for indicating requests and allocations

  - multiple resources of each type: allocation and request matrices

- Methods for handling deadlocks
  - prevention: deny one of the four conditions above

    mutual exclusion — spooling

    hold and wait — request all required resources initially

    no preemption — take resources away

    circular wait — linearly order resources
  - detection and recovery: detect deadlocks and take recovery action (rollback or kill)
    - single instances of resource types: check for cycles in the resource allocation graph
    - multiple instances of resource types: try to satisfy the needs of at least one process
  - avoidance: make sure that the machine reaches only safe states, i.e., that it is guaranteed that there is some scheduling order in which every process can run to completion even if they request their maximum number of resources immediately
    - single instance of resource type: detecting cycles in the resource allocation graph
    - multiple instances of resource type: Banker's algorithm

Memory Management

Main goal: to allocate (main) memory to processes in an efficient and safe way

- Cache: elements of design (size, write policy, replacement algorithm)
- MM without virtual memory: multiprogramming with fixed partitions, swapping; memory allocation algorithms
- Virtual memory:
  - provides large logical (virtual) address space for programmers
  - only those parts of the program are in main memory that are currently used by a process
  - mapping between logical and physical addresses
  - paging and segmentation
- Paging:

  - page tables (single- and multilevel), translation lookaside buffers (TLB), inverted page tables
  - page fault handling
  - page replacement algorithms: optimal, not recently used (NRU), FIFO, clock, least recently used (LRU)
  - working set and locality of reference, thrashing
  - design issues: local vs. global allocation, load control, page size, cleaning policy
  - paging in UNIX and Windows
- Segmentation: two dimensional address space — virtual memory is divided into logical units (segments); supports protection and sharing
- Segmentation with paging

I/O System

- Devices: block and character; communication between device and machine: bus, controller; direct memory access (DMA); Interrupts
- Principles of I/O software
  - goals: device independence, uniform naming, error handling, buffering
  - programmed I/O: polling, busy waiting
  - interrupt-driven I/O: context switches, interrupts
  - DMA (programmed I/O via the DMA, interrupt on completion)
- I/O software layers
  - interrupt handlers
  - device drivers: their tasks, uniform interface
  - device-independent I/O software: buffering
  - user-space I/O software: spooling
  - life cycle of an I/O request
- Disks
  - hardware: cylinders, tracks, sectors
  - efficiency and reliability using RAID schemas: strips, mirroring, block interleaved parity
  - disk scheduling: FCFS, SSF, Elevator algorithm; their performance with respect to access time (seek time + rotational latency) and fairness
  - disk cache

File Systems

- File: contiguous logical address space, attributes (name, size, type, etc.), operations (open, delete, read, write, etc.)
- Access methods: sequential or random (direct)
- Directory: hierarchical system for organizing files; tree structure or general directed graph structure, relative and absolute path names, operations (create, delete, open, close, etc.), symbolic and hard links
- Implementation — simplicity, fast and flexible access, efficient use of space
  - contiguous allocation
  - linked list and file allocation table (FAT)
  - i-nodes

  Directory entry
- Disk space management: block size, free list
- Example file systems
  - MS-DOS using FAT
  - UNIX using i-nodes
  - Windows 2000 using master file table (MFT)