**ASSUMPTION UNIVERSITY**

**Vincent Mary School of Science & Technology**

**DEPARTMENT OF COMPUTER SCIENCE**

**COURSE OUTLINE**

|  |
| --- |
| **COURSE ORGANIZATION** |
| **Course Title:** | **Computing Systems** (SC5212)  |
| **Course Status:** | Non-credit |
| **Pre-requisite:** | None |
| **Semester:** | 1/2019 |
| **Classroom, Date & Timing:** | Monday 18:30-21:30 at A82 |
| **Description:** | Fundamental knowledge on computer architecture and operating systems; modern design and technology, and design rational; processor architectures, forms of parallelism, instruction set architecture, memory systems, multiprocessors, process synchronization, distributed computation, deadlock management. |
| **Marks Allocation:** | Term Project (individual)\*Class work (Quizzes & Assignments)Midterm ExaminationFinal Examination (Comprehensive) |  15 % 25 % 20 % 40 % |

|  |
| --- |
| **COURSE INSTRUCTOR** |
| **Instructor:** | Asst. Prof. Dr. Anilkumar K Gopalakrishnan |  |  |  |
| **Office:** | E64 (HuaMak), VMS0508(Suvarnabhumi)  |
| **Email:** | anil@scitech.au.edu |  |  |  |
| **Course materials:** | portal.scitech.au.edu/anilkumar |  |  |  |
| **Mobile Phone:** | 0891351711 |  |  |  |

|  |
| --- |
| **COURSE RESOURCES** |
| **Textbook1:** | * **Digital Design and Computer Architecture**, 2nd Edition, David M.H and Sarah L.H, Morgan Kaufmann, Elsevier, 2013 (ISBN: 978-0-12-394424-5)
 |
| **Textbook2:** | * Computer Architecture: A Quantitative Approach 4th Edition, David A. Patterson and John L. Hennessy, Morgan Kaufmann Publishers, 2005. (ISBN: 0-12-370490-1)
 |
| **Textbook3:** | * **Operating System Concepts**, 9th Edition, Abraham Silberschatz, Peter B. Galvin, Greg Gagne, Wiley, 2013 (ISBN: 978-1-118-09375-7)
 |
| **Reference(s):** | * **Computer Architecture and Implementation**, Harvey G. Cragon, Cambridge University Press, 2000 (ISBN: 0-52-165168-9)
* **Operating Systems :** Internals and Design Principles 8th Edition, William Stallings, Prentice Hall, 2015 (ISBN: 978-0-133-80591-8)
* **Advanced Concept in Operating Systems**, by Mukesh Singhal and Niranjan Shivaratri, McGraw-Hill, 1994.
 |

|  |
| --- |
| **COURSE POLICIES:** |
| 1. Students are required to have 80% of class attendance to be eligible for the final examination.
2. Examination contents will be based on assigned lecture materials and class assignments.
 |

|  |
| --- |
| **COURSE EXAMINATIONS** |
| **Midterm:** | Date: |  | Time: | 12:00-14:00 (2 hrs) |
|   |  |
| **Final:** | Date: |  | Time: | 13:00-16:00 (3 hrs) |
|  |  |

|  |  |
| --- | --- |
| **Week** | **Topic** |
| Week 1,2 | Introduction-Number System & Logic Gates |
| Week 3, 4 | Combinational and Sequential Logics |
| Week 5 | Arithmetic Circuits |
| Week 6 | Operating System Overview |
| Week 7 | Process Synchronization and Thread management |
| Week 8 | Distributed Computation  |
| Week 9 | Deadlock Management and CPU scheduling |
| Week 10 | Instruction Set Architecture |
| Week 11 | Memory Management  |
| Week 12 | Pipeline Architecture |
| Week 13 | Multiprocessor Systems |
| Week 14 | Virtual Memory |
| Week 15 | Revision |

|  |
| --- |
| **INSTRUCTION SET ARCHITECTURE (ISA) PROJECT:** |
| Write a simple CPU Instruction Set Architecture (ISA) simulation program for a selected CPU size (either 16-bit, 24-bit or 32-bit) using any programming language. The ISA design should be included *the user defined instruction sets*, *control registers*, *main memory indication*, CPI (clock per instruction), etc.**Requirements:*** The simulation should support at least 16-bit integer representation.
* The ISA should consist of at least 8 general purpose registers.
* The program should be able to handle (at least) op-codes such as *mov* (move), *add* (addition), *sub* (subtraction), *mul* (multiplication), *div* (division), etc.

**Example ISA**1. User will be allowed to input the instructions in the following way:

Assume that a 24-bit ISA with 16-bit integer data representation and 16-bit 8 GPRs (say r0,…..r7). Select any two registers from the given GPR list as a *multiplication register* for multiplication and a *remainder register* for division. An example instruction set with their decoded and encoded forms are shown below (where **r0** is a multiplication register and **r7** is a remainder register):   Decoded form Instruction meaning Encoded form mov r1 3 r1 🡨 3 [0001001 0000 0000 0000 0011] add r1 3 r1 🡨 r1 + 3 ------  mov r2 r1 r2 🡨 r1 ------  mul r2 -1 r0: r2 🡨 r2 \* -1 (total 32-bit result) -------  mov r3 r2 r3 🡨 r2 ------- div r3 2 r7:r3 🡨 r3 / 2 ------  end 0 0**Values of registers after the execution of the instruction set**During the execution of the above code sequence, the values of registers would be varied in the following way: r1 = 3 [0000 0000 0000 0011] r1 = 6 [0000 0000 0000 0110] r2 = 6 [0000 0000 0000 0110] r0: r2 = -6 [1111 1111 1111 1111 1111 1111 1111 1010] r3 = -3 [1111 1111 1111 1101] r7: 0 [0000 0000 0000 0000]**CPI (Clocks Per Instruction)**CPI of the program depends on the number of clock cycles used by each instruction **Pipelined version**  Pipelined version of the instruction set simulation is an optional one. But, whoever with such  an additional feature will be rewarded greatly.**Scoring Criteria:**The following key quality will be considered:* ISA design & coding 10%
* Presentation 2%
* Report with code 3%
 |
| **Programming Assignments:** | **No.** | **Instruction** | **Marks** |
| 1 | Write a simulation program to simulate the behavior of **FCFS**, **Preemptive SJF,** **Priority**, and **Round Robin** CPU scheduling algorithms (the simulations must be based on the *arrival time* of each process). Show all necessary parameters to keep track of progress step-by-step. |  |
| 2 | Write a simulation program to show behavior of **banker’s algorithm** for multiple resource unit allocation problems. The program should show whether a solution end with a ***safe state*** or not. And also the program should have the facility to collect input dataset from both a text file and from a random generator. Show all necessary parameters to keep track of progress step-by-step. |  |
| 3 | Implements the **FIFO**, **LRU**, and **Optimal** *page replacement algorithms* using any programming language (along with that you should apply the random page-reference string to each algorithm, and record the number of *page f***aults** incurred by each algorithm). |  |