Program	BS Data Science	
Course Code	CC-210	
Course Title	Computer Organization and Assembly Language Programming	
Credit Hours	**Theory**	**Lab**
	3	1
Lecture Duration	90 minutes (1.5 Hours), 2 lectures per week, 3 hours lab session per week	
Semester	4	
Pre-requisites	**Courses**	**Knowledge**
	Digital Logic Design	• Strong grip on Number system • Strong back ground of Combinational and Sequential circuits
Follow Up Courses	Computer Architecture	

Course Learning Outcomes (CLOs)

CLO No	Course Learning Outcome	Bloom Taxonomy
CLO-1	Acquire the basic knowledge of computer organization computer architecture and assembly language.	C2 (Understand)
CLO-2	Understand the concepts of basic computer organization, architecture, and assembly language techniques	C2 (Understand)
CLO-3	Solve the problems related to computer organization and assembly language	C3 (Apply)

Aims and Objectives	1. Students will understand as to how a microprocessor is designed starting from a basic NAND gate to a full-blown computer system 2. Students will learn to write Hardware Description Language of various components of a computer system 3. Students will learn the different ways of interfacing I/O devices with computer 4. Students will learn to design the ISA, machine and assembly language of a microprocessor
	5. Students will understand the organization of the Intel processors, and be able to write basic programs for x86-64 Assembly Language
Learning Outcomes	• Students will understand, design and write the HDL of all the components of a Von-Neumann based computer. Understand the basic concept of computer organization, will design its assembly, machine language and will write its assembler in C, other than writing some basic assembly programs for that designed computer (Understand, Apply, Demonstrate) • Students will have a strong grip of x86-64 assembly language and the tool chain involved (Apply)

Syllabus	**Part-I**: The pre-mid part of the course deals with design of a complete computer system and writing its HDL. **Part-II**: The prost-mid part of the course deals with a detailed discussion on evolution of Intel processors, its programming models, its assembly language and programming tool chain
Contents	**Section-1:** - HDL for Combinational Circuits - HDL for Sequential Circuits - Data Storage in computer system **Section-2:** - Design and HDL of computer memory - Instruction Set Architecture - Design of hardware and writing HDL for Hack Computer - Interfacing I/O devices with Hack Computer **Section-3:** - Design of Machine Language of Hack Computer - Design of Assembly Language of Hack Computer - Design of Data Path (Buses) for Hack Computer - Design and Code of Hack Assembler **Section-4:** - History and Evolution of Intel Microprocessors - Concept of pipelining and improving processor performance - Programming Model of x86-64 processor - NASM and x86-64 assembly - Debugging with GNU debugger (gdb) **Section-5:** - Data transfer instructions - Memory addressing modes - X86-64 Logical and Bit shifting operations
	- Control Transfer Instructions - Function Calling Convention and Function Stack Frames **Section-6:** - Mixing C with Assembly programs - Getting user input - Programming with Arrays and Strings - Floating Point Instructions - Computer performance and parallel processing hardware - Combinational & Sequential Circuits
Teaching-learning Strategies	• Lectures • Case Studies • Project • Assignments
Assignments	Types and Number with calendar

Textbooks	A. The Elements of Computing Systems, Building a modern computer, by Noam Nisan and Shimon Schocken, 2nd Ed, Published in 2020, ISBN- 13: 978-0262640688 B. X86_64 Assembly Language Programming with Ubuntu, by Ed. Jorgensen, January 2020
Reference Material/Suggested Readings	C. Introduction to Computing Systems: from bits and gates to C and beyond, by Yale Patt and Sanjay Patel, 3rd Ed, Published in 2020, ISBN13: 9781260150537 D. X86_64 Assembly Language Programming with Ubuntu, by Ed. Jorgensen, January 2020 X86_64 Assembly Language Programming with Ubuntu, by Ed. Jorgensen, January 2020 E. Dr. Muhammad Arif Butt, COAL -Video Lectures: https://www.youtube.com/c/LearnWithArif/playlists
Notes	

Detailed Lecture wise plan

Week	Lecture	Topic	SourceBook (Ch#)	Recommendation for Learning Activities
1	1	**Introduction to Course:** Computer Organization and Assembly Language Programming and a discussion on the course matrix. The course will be having two parts in the first half of the course we will be designing and writing the HDL of a full-blown computer, will design its machine and assembly language, will write programs and execute those programs on the designed h/w architecture. At the end of the first half of the course, we will also design and write an Assembler (in C) for the designed computer. The second half of the course will deal with the assembly of the all-time famous x86-64 architecture. The links from where the e-books, tools, code snippets, and lecture slides, and other misc resources can be downloaded are mentioned.	Text A-Ch1	

	Lecture	Topic	SourceBook (Ch#)	Recommendation for Learning Activities
	2	**HDL for Combinational Circuits – I:** Review of Boolean logic and gates. Introduction to Hardware Description Languages. Design and code of And, Or, Not gates using the universal NAND gate. Downloading and installing Hardware Simulator and interactive chip testing on this simulator. Designing Xor chip and performing script based testing of Xor chip on h/w simulator. **HDL for Combinational Circuits – II:** Design and HDL code for XOR chip, using And, OR, Not gate chips. A demo of Verification of XOR chip using interactive chip testing in the h/w simulator. A brief overview of script-based chip testing. Writing script for testing of the designed XOR chip. A discussion on key players involved in a hardware construction project.	Text A-Ch1	**Lab:**
2	3	**HDL for Combinational Circuits – III:** Design and HDL code for some standard combinational circuits like Encoder, Decoders, Multiplexers, and De-Multiplexer chips. A demo of Verification / Testing of these standard combinational chips using interactive chip testing in the h/w simulator. **HDL for Combinational Circuits – IV:** Design and HDL code for multi-bit gates. The concept of buses and the design of chips having buses as input. Design and code of And16, Or16, Not16, and Mux16 chips having 16-bits inputs. Design and code of And4way16 and similar chips having four inputs with each input of 16 bits	Text A-Ch1	
	4	**Data Storage – I:** Data Representation in Computers, Unsigned, and Signed Numbers, Sign magnitude representation and its limitations, 1s Complement representation and its limitations, 2s Complement representation, Comparisons and pros and cons of each, Ranges and different Storage Sizes, Overflow in Unsigned and Signed Numbers, How the Hardware Detect an Overflow, Concept of Sign Extension, Encoding Characters and Strings (ASCII and Unicode)	Ref C-Ch2	**Lab:**

Week	Lecture	Topic	SourceBook (Ch#)	Recommendation for Learning Activities
3	5	**Data Storage – II:** Encoding Real Numbers, Fixed Point Representation, Floating Point Representations (IEEE-754), Storage layout, Conversion Examples, Range and Precision, Arithmetic Operations, Overflow and Underflow, IEEE-754 Special Values	Ref C-Ch2	
	6	**Design of ALU - I:** Review of HDL for Combinational Circuits, Designing a single bit Logic Unit Writing HDL for Combinational Arithmetic Circuits like Half Adder, Full Adder, Full Subtractor, 16-bit Binary Adder (Add16 chip), 16-bit Incrementer (Inc16 chip), Demo of above chips on H/W Simulator **Design of ALU - II:** Components of a Computer System, Design of ALU, The Hack ALU, The Hack ALU Operations, Design of Hack ALU, HDL of Hack ALU, Verifying the ALU chip on H/W Simulator	Text A-Ch2	**Lab:**
4	7	**Design of Sequential Circuits:** Why Sequential Circuits? Understanding Time in Circuits, Combinational vs Sequential Circuits, Flip Flops, D flip Flop, SR Flip Flop, JK Flip Flop, T Flip Flop **Design of Registers:** What are Registers, Design of 1-bit Register, HDL for 1-bit Register, Design of 16-bit Register, HDL for 16-bit Register	Text A Ch3	
	8	**Design of Memory:** Concept of Memory Hierarchy, Multi-Byte Read/Write, Design of Random Access Memory, Read/Write Logic of RAM, API of a RAM Chip, HDL of 8 Words RAM, HDL of 64 Words RAM, HDL of 512 Words RAM, HDL of 4K Words RAM, HDL of 16K Words RAM **Design of Counters:** Overview of Hack Computer Components, Overview of Counters, Why do we need Counter for our Hack Computer, Concept of Program Counter, Counter Simulation, Design and Implementation of PC for Hack Computer, Demo on H/W Simulator	Text A-Ch3	**Lab:**
5	9	**ISA-I:** Overview of Computer System, Universality of Computer System, Turing Machine, Von Neumann Architecture, Instruction Set Architecture (ISA)	Ref A Ch3	

Week	Lecture	Topic	SourceBook (Ch#)	Recommendation for Learning Activities
	10	**ISA-II:** Five Dimensions of ISA, Class of ISA, Types and Sizes of Operands, Operations (including control flow instructions), Memory Addressing Models and Addressing Modes, Encoding an ISA	Text A-Ch3	
6	11	**Hack Machine Language – I:** Hack Computer Machine Language, Review of h/w of Hack Computer, Software of Hack Computer, A Instruction, C Instruction, Examples	Text A Ch4	
	12	**Hack Machine Language – II:** Review of Hack Symbolic Machine Instructions, A Instruction, C Instruction, Binary Code Format of Hack Computer Instruction, Encoding of 16 bit A-Instruction, Encoding of 16 bit C-Instruction, Examples, A Complete Hack Program: Assembly Language	Text A-Ch4	
7	13	**Interfacing I/O Devices:** How to interface I/O devices with computer, Interfacing Screen with Hack computer, Demo of built-in Screen chip on h/w Simulator, Interfacing Keyboard with Hack computer, Demo of built-in Keyboard chip on h/w Simulator	Text A Ch 5	
	14	**Hack Assembly Programming – I:** Review of Hack Computer Assembly Instructions, Hack Assembly Programs, A Hello World in Hack assembly, CPU Emulator, Demo, Program Termination **Hack Assembly Programming – II:** Recap previous lecture, Symbols in Hack Assembly Language, Built-in Symbols, Label Symbols, Variable Symbols, Branching, Iteration	Text A-Ch4	
8	15	**Hack Assembly Programming – III:** Review of Hack Assembly Programs, Pointers and Arrays, Input / Output Instructions, Debugging, Review of Hack Assembly Programs, Pointers and Arrays, Input / Output Instructions, Debugging	Text A-Ch4	

Week	Lecture	Topic	SourceBook (Ch#)	Recommendation for Learning Activities
	16	**Data path of Hack CPU-I:** Von Neumann Architecture, Flow of Information inside Computers, Buses, Data Bus, Address Bus, Control Bus, Fetch Execute Cycle, Fetch Execute Clash, Harvard Architecture **Data path of Hack CPU-II:** Review of Hack Computer Architecture, Hack CPU Interface, Hack CPU Implementation, Input/output and Operations of Hack ALU, Control Logic of Hack CPU	Text A-Ch5	
9	17	**Design of Hack Computer:** Recap of Hack Computer Architecture, Implementation of Hack CPU Chip (CPU.hdl), Implementation of Hack Memory Chip (Memory.hdl), RAM16 chip (RAM16K.hdl), Screen chip (Screen.hdl), Keyboard chip (Keyboard.hdl), Implementation of Hack ROM Chip (ROM32K.hdl), Implementation of Hack Computer Chip (Computer.hdl)	Text A-Ch5	
	18	**Design of Hack Assembler:** What is an Assembler? How an Assembler works? Hack Machine Language Specification, Demo of Built-in Hack Assembler, Design of Hack Assembler (w/o Symbols), Design of Hack Assembler (with Symbols), Hack Assembler Implementation in C/C , Executing Hack Machine Code, Hack Computer Chip in h/w Simulator, CPU Emulator	Text A-Ch6	
10	19	**History and Evolution of Intel Microprocessors:** Intel 4004 (1971), Intel 8008, Intel 8080, Intel 8086 (x86), Intel 80286, Intel 80386, Intel 80486, Intel 80586 (Pentium P5), Intel 80686 (Pentium P6), Intel Core (2006) Intel Nehalam (2008), Intel Sandy Bridge, Intel Ivy Bridge, Intel Haswell, Intel Broadwell, Intel Sky Lake, Intel Kaby Lake, Intel Coffee Lake, Intel Coffee Lake Refresh, Intel Comet Lake (2019) **On Improving Processors Performance:** CPU Performance Equation, Single Cycle vs Multi Cycle CPU Architecture, Pipelined CPU Architecture, Pipeline Stages, Even vs Uneven pipelined stages, Pipelined Hazards, Solutions of Pipeline Hazards, CISC vs RISC Architecture	Text B-Ch1	

Week	Lecture	Topic	SourceBook (Ch#)	Recommendation for Learning Activities
	20	**Programming Model of x86 Architecture:** Layout of memory models (flat, segmented) and register set file of Intel 8080, 80386, x86-64. Logical to physical address translation for segmented memory model.	Text B-Ch2	
11	21	**Hello World in x86-64 Assembly:** Overview of microprocessor families and their corresponding assembly languages. Tool chain and programming environment for x86-64 assembly programming. Running the first hello world assembly program. **Structure of x86-64 Assembly Program:** A discussion on the x86-64 assembly language instruction format and the overall structure of assembly	Text B-Ch4	
	22	**Debugging C-Program with gdb:** A review of C-compilation process. What is a debugger? Why use gdb? How to compile, load and run a program inside gdb and get information about the running process. Getting help inside gdb, setting break points, watch points, stepping through the code, examining and modifying variables, convenience variables and setting conditional break points. **Data Types and Endianness in x86-64:** Usage of different data types and special tokens in NASM. Practically understand about the endianness of a machine	Text B-Ch5	
12	23	**Data Transfer Instructions and Process Stack:** Usage of different move instructions like mov, movzx, movsx, lea and xchg. A discussion on the working of process stack and the push and pop instructions. **Memory Addressing Modes:** Theoretical concepts and pros and cons of addressing modes used by different processors. Addressing modes used by x86-64 like Base-Index-Sale-Displacement	Text B-Ch8	

Week	Lecture	Topic	SourceBook (Ch#)	Recommendation for Learning Activities
	24	**Arithmetic Instructions Part-I:** A recap of x86-64 register set and the programming tool chain. Summary of major categories of x86-64 instructions. A practical demo on the use of add, adc, sub, sbb, inc, dec, neg, cmp, clc, stc, and cmc. A discussion on how the flags are effected after these arithmetic instructions **Arithmetic Instructions Part-II:** A recap of x86-64 register set and the programming tool chain. Summary of major categories off x86-64 instructions. A practical demo on the use of mul, div, imul, idiv instructions. A discussion on how the flags are effected after these arithmetic instructions	Text B-Ch7	
13	25	**Logical Operations:** A recap of x86-64 register set and the programming tool chain. Summary of major categories of x86-64 instructions. A practical demo on the use of and, or, not, xor, and test instructions. A discussion on how the flags are effected after these logical instructions. **Bit-Shifting Operations:** A recap of x86-64 register set and the programming tool chain. Summary of major categories of x86-64 instructions. A practical demo on the use of shl, sal, shr, sar, rol, ror, rcl, and rcr instructions. A discussion on how the flags are effected after these logical instructions.	Text B-Ch7	
	26	**Control Instructions - I:** A discussion on control of flow of execution of a program and how to change it. Description of unconditional jump instruction with a demonstration of example programs. Discussion on signed and unsigned conditional jump instructions with demonstration of example programs. Translating if…else code too assembly language. **Control Instructions - II:** A Recap of previous session. Translating high level repetition structure (for, while, ….) to its corresponding x86 assembly code using conditional jump instructions as well as using x86 loop instructions.	Text B-Ch7	

Week	Lecture	Topic	SourceBook (Ch#)	Recommendation for Learning Activities
14	27	**GDB with PEDA Plugin:** A recap of gdb command line and text user interface mode. Downloading, installing, and configuring Python Exploit Development Assistance (PEDA) plugin to enhance the firepower of gdb. Debugging the x86-64 assembly programs using gdb with PEDA and a brief intro of using this plugin for reverse engineering and exploit development **Functions in Assembly Language – I:** What are functions? Why they are used in programming languages? Syntax of defining an assembly function in NASM and MASM. Understanding the working of x86-64 call and ret instruction. The usage of process run-time stack in function call. The concept and requirement of caller-saved and callee-saved registers	Text B-Ch9,12	
	28	**Functions in Assembly Language – II:** Recap of functions in assembly language and use of call and ret instructions. How different operating systems allow x86-64 assembly programmers to pass and return values to and from functions. Designing an assembly function to display a single digit and a multi-digit decimal number on screen. Writing, assembling, linking and executing multi-file assembly programs on x86-64. **Function Calling Convention and FSF:** Recap of assembly language functions. Understanding function calling in high-level languages like C and C . The concept of Function Stack Frame (FSF) or Activation Record used to store data associated with a high-level function on the process run time stack. The x86-64 procedure prolog and procedure epilog for creating and removing FFSF from the stack. A demonstration of stack-based buffer overflow vulnerability and concept of exploiting it.	Text B-Ch9,12	

Week	Lecture	Topic	SourceBook (Ch#)	Recommendation for Learning Activities
15	29	**Mixing C with x86-64 Assembly:** Recap of pushing and popping FSF to and from the process run time stack in high level languages. Calling C-Library functions from within an assembly program. Doing the reverse, i.e., Calling assembly functions from within a C program. **Getting User Input:** User input via system calls, library calls, and command line arguments. What are command line arguments and why we use them in high level languages as well as in assembly programming? Converting the string input received via command line to integer for further processing.	Text B-Ch13 ,16	
	30	**Arrays:** Array address computation (pointer arithmetic) based on its types (bytes, words, double words and quad-words), General pattern for memory references, allocation arrays using malloc, processing arrays, filling with random numbers, printing array elements **String:** Understanding C-strings in assembly, x86 string instructions to store strings in memory, load strings from memory, comparing strings, and scanning strings for substrings	Text B-Ch13 ,16	
16	31	**Floating Point Instructions:** 8087 floating point instructions that use stack of 80 bit floating point registers (ST0, ST1, …). Intel Core iseries floating point instructions that work with Streaming SIMD Extensions (SSE) 128-bit registers (xmm0, xmm1, …). The concept of Advanced Vector Extensions (AVX) that use 256-bit registers (ymm0, ymm1, …) Data movement instructions (movss, movsd) Arithmetic instructions (addss, addsd, subss, subsd, mulss, mulsd, divss, divsd) Integer / floating point conversion instructions (cvtss2sd, cvtsd2ss, cvtss2si, cvtsi2ss, cvtsd2si, cvtsi2sd) Floating point control instructions (ucomiss, ucomisd) Floating point calling convention	Text B-Ch18	
	32	**Computer Performance and Parallel Processing Hardware:.**	Text B-Ch19	