ACCRA TECHNICAL UNIVERSITY
SCHOOL OF APPLIED SCIENCE & ARTS
DEPARTMENT OF COMPUTER SCIENCE

FULL TIME: **LEVEL: 300** **SEMESTER: TWO (2)**

ACADEMIC YEAR 2018/2019
COMPUTER ORGANIZATION & ARCHITECTURE
COURSE CODE: CPS 302

MIDSEM

David D. Agbloe
01163844D

Part A.

1. The instruction -> Add LOCA, R0 does _____

   a) Adds the value of LOCA to R0 and stores in the temp register

   b) Adds the value of R0 to the address of LOCA

   c) Adds the values of both LOCA and R0 and stores it in R0

   d) Adds the value of LOCA with a value in accumulator and stores it in R0

2. During the execution of a program which gets initialized first?

   a) MDR

   b) IR

   c) PC

   d) MAR

3. Which of the register/s of the processor is/are connected to Memory Bus?

   a) PC

   b) MAR

   c) IR

   d) Both PC and MAR

4. ISP stands for _____

   a) Instruction Set Processor

   b) Information Standard Processing

   c) Interchange Standard Protocol

   d) Interrupt Service Procedure

5. The internal Components of the processor are connected by _____

   a) Processor intra-connectivity circuitry

   b) Processor bus

   c) Memory bus

   d) Rambus

6. The _____ format is usually used to store data.

   a) BCD

   b) Decimal

   c) Hexadecimal

   d) Octal

7. The 8-bit encoding format used to store data in a computer is _____

   a) ASCII

   b) EBCDIC

   c) ANCI

   d) USCII

8. A source program is usually in _____

   a) Assembly language

   b) Machine level language

   c) High-level language

   d) Natural language

9. Which memory device is generally made of semiconductors?

   a) RAM

   b) Hard-disk

   c) Floppy disk

   d) Cd disk

10. The small extremely fast, RAM's are called as _____

    a) Cache

    b) Heaps

    c) Accumulators

    d) Stacks

11. The ALU makes use of _____ to store the intermediate results.

a) Accumulators

b) Registers

c) Heap

d) Stack

12. _____ are numbers and encoded characters, generally used as operands.

a) Input

b) Data

c) Information

d) Stored Values

13. The Input devices can send information to the processor.

a) When the SIN status flag is set

b) When the data arrives regardless of the SIN flag

c) Neither of the cases

d) Either of the cases

14. _____ bus structure is usually used to connect I/O devices.

a) Single bus

b) Multiple bus

c) Star bus

d) Rambus

15. The I/O interface required to connect the I/O device to the bus consists of _____

a) Address decoder and registers

b) Control circuits

c) Address decoder, registers and Control circuits

d) Only Control circuits

16. To reduce the memory access time we generally make use of _____

a) Heaps

b) Higher capacity RAM's

c) SDRAM's

d) Cache's

17. The smallest entity of memory is called _____

a) Cell

b) Block

c) Instance

d) Unit

18. The instruction, Add #45,R1 does _____

a) Adds the value of 45 to the address of R1 and stores 45 in that address

b) Adds 45 to the value of R1 and stores it in R1

c) Finds the memory location 45 and adds that content to that of R1

d) None of the mentioned

19. The main virtue for using single Bus structure is _____

a) Fast data transfers

b) Cost effective connectivity and speed

c) Cost effective connectivity and ease of attaching peripheral devices

d) None of the mentioned

20. The decoded instruction is stored in _____

a) IR

b) PC

c) Registers

d) MDR

21. A source program is usually in _____

a) Assembly language

b) Machine level language

c) High-level language

d) Natural language

22. The control unit controls other units by generating ____

a) Control signals

b) Timing signals

c) Transfer signals

d) Command Signals

23. The I/O interface required to connect the I/O device to the bus consists of _____

a) Address decoder and registers

b) Control circuits

c) Address decoder, registers and Control circuits

d) Only Control circuits

24. Which registers can interact with the secondary storage?

    a) MAR

    b) PC

    c) IR

    d) R0

25. A processor performing fetch or decoding of different instruction during the execution of another instruction is called _____

    a) Super-scaling

    b) Pipe-lining

    c) Parallel Computation

    d) None of the mentioned

Part B

1. **Describe in detail the different kinds of addressing modes with an example.**

Addressing Mode: The different way in which the location of an operand is specified in an instruction. The addressing mode specifies a rule for interpreting or modifying the address field of the instruction before the operand is actually executed.

Types Of Addressing Modes

i. Immediate
ii. Direct
iii. Indirect
iv. Register
v. Register indirect
vi. Displacement
vii. Stack

Immediate Addressing

The simplest form of addressing is immediate addressing, in which the operand value is present in the instruction. This mode can be used to define and use constants or set initial values of variables. Typically, the number will be stored in twos complement form; the leftmost bit of the operand field is used as a sign bit. When the operand is loaded into a data register, the sign bit is extended to the left to the full data word size. For example: ADD 7, = Add 7 to contents of accumulator. 7 is the operand here.

Direct Addressing

A very simple form of addressing is direct addressing, in which the address field contains the effective address of the operand. The processor will retrieve the data directly from the address specified in the instruction
Single memory reference to access data.
No additional calculations to find the effective address of the operand.

For Example: ADD R1, 4000 - In this the 4000 is effective address of operand.

Indirect Addressing
With indirect addressing, the length of the address field is usually less than the word length, thus limiting the address range. One solution is to have the address field refer to the address of a word in memory, which in turn contains a full-length address of the operand. Indirect addressing means that the address of the data is held in an intermediate location so that the address is first 'looked up' and then used to locate the data itself.
Many programs make use of software libraries that get loaded into memory at run time by the loader. The loader will most likely place the library in a different memory location each time.

For Example: LOAD R1, @100 Load the content of memory address stored at memory address 100 to the register R1.

Register Addressing
Register addressing is similar to direct addressing. The only difference is that the address field refers to a register rather than a main memory address. A register contains the operand. If the contents of a register address field in an instruction is 5, then register R5 is the intended address, and the operand value is contained in R5. Typically, an address field that references registers will have from 3 to 5 bits, so that a total of from 8 to 32 general-purpose registers can be referenced

Register Indirect Addressing
In this addressing the operand's offset is placed in any one of the registers as specified in the instruction. The effective address of the data is in the base register or an index register that is specified by the instruction. The effect is to transfer control to the instruction whose address is in the specified register.

For Example: (A7) to access the content of address register A7.

Displacement Addressing
A very powerful mode of addressing combines the capabilities of direct addressing and register indirect addressing. It is known by a variety of names depending on the context of its use, but the basic mechanism is the same. Displacement addressing requires that the instruction have two address fields, at least one of which is explicit. The value contained in one address field (value = A) is used directly. The other address field, or an implicit reference based on opcode, refers to a register whose contents are added to A to produce the effective address.

Stack Addressing
Computers that are stack-oriented have an instruction to push the contents of a memory location or a register onto the stack. Instructions with no addresses are used in conjunction with a stack. This form of addressing specifies that the two operands are to be popped off the stack, one after another, the operation performed and the result pushed back onto the stack.

2. Explain the various Instruction types with examples.

i. Arithmetic Instructions
ii. Branch Instructions
iii. Data Transfer Instructions
iv. Logic Instructions
v. Bit-oriented Instructions

Arithmetic Instructions
Arithmetic instructions perform several basic operations such as addition, subtraction, division, multiplication etc. After execution, the result is stored in the first operand. For example: ADD A,R1 - The result of addition (A+R1) will be stored in the accumulator.

Branch Instructions
There are two kinds of branch instructions: Unconditional jump instructions: upon their execution a jump to a new location from where the program continues execution is executed. Conditional jump instructions: a jump to a new program location is executed only if a specified condition is met. Otherwise, the program normally proceeds with the next instruction.

Data Transfer Instructions
Data transfer instructions move the content of one register to another. The register the content of which is moved remains unchanged. If they have the suffix "X" (MOVX), the data is exchanged with external memory.

Logic Instructions
Logic instructions perform logic operations upon corresponding bits of two registers. After execution, the result is stored in the first operand.

Bit-Oriented Instructions
Similar to logic instructions, bit-oriented instructions perform logic operations. The difference is that these are performed upon single bits.

3. Briefly explain the organization of ISA computer.

The Instruction Set Architecture (ISA) is the part of the processor that is visible to the programmer or compiler writer. The ISA serves as the boundary between software and hardware.

The 3 most common types of ISAs are:

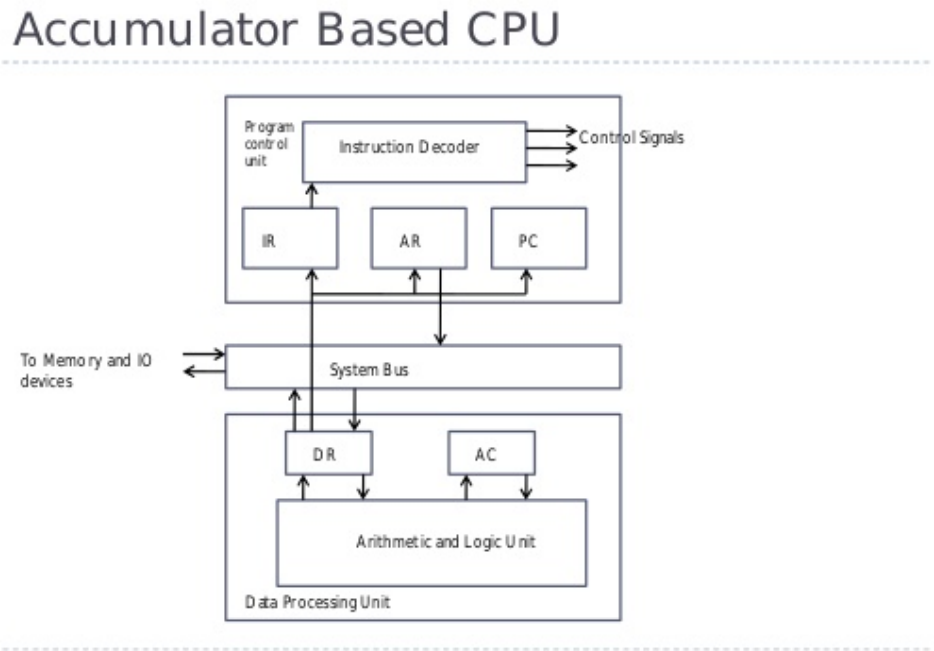
i. ***Stack*** - The operands are implicitly on top of the stack.
ii. ***Accumulator*** - One operand is implicitly the accumulator.
iii. ***General Purpose Register (GPR)*** - All operands are explicitly mentioned, they are either registers or memory locations.

4. With a neat block diagram explain the Accumulator based CPU.

Accumulator
It is an 8-bit register that is part of ALU. This register is used to store 8-bit data & in performing arithmetic & logic operation. The result of operation is stored in accumulator. In a computer's central processing unit (CPU), the accumulator is a register in which intermediate arithmetic and logic results are stored.

Without a register like an accumulator, it would be necessary to write the result of each calculation (addition, multiplication, shift, etc.) to main memory, perhaps only to be read right back again for use in the next operation. Access to main memory is slower than access to a register like the accumulator because the technology used for the large main memory is slower (but cheaper) than that used for a register. Early electronic computer systems were often split into two groups, those with accumulators and those without.

Modern computer systems often have multiple general purpose registers that operate as accumulators, and the term is no longer as common as it once was. However, a number of special-purpose processors still use a single accumulator for their work to simplify their design.



5. **In the von Neumann model, explain the purpose of the processing unit and the program counter**

Processing Unit
The processing unit performs all of the arithmetic and logic functions.

The ALU performs the decision-making (logical) and arithmetic operations. It works in combination with a number of registers that hold the data on which the logical or mathematical operations are to be performed. For decision-making operations, the ALU can determine if a number equals zero, is positive or negative, or which of two numbers is larger or smaller. These operations are most likely to be used as criteria to control the flow of a program. It is also standard for the ALU to perform basic logic functions such as AND, OR, and Exclusive-OR. For arithmetic operations, the ALU often performs functions such as addition, subtraction, multiplication, and division.

Programe Counter
The program counter (PC), commonly called the instruction pointer (IP) and sometimes called the instruction address register (IAR), the instruction counter, or just part of the instruction sequencer, is a processor register that indicates where a computer is in its program sequence. It is is a register in a computer processor that contains the address (location) of the instruction being executed at the current time, the PC initially holds the memory address of ("points to"). it is incremented after fetching an instruction, and the program counter increases its stored value by 1. After each instruction is fetched, the program counter points to the next instruction in the sequence. When the computer restarts or is reset, the program counter normally reverts to 0.Processors usually fetch instructions sequentially from memory,

but control transfer instructions change the sequence by placing a new value in the PC. These include branches (sometimes called jumps), subroutine calls, and returns. A transfer that is conditional on the truth of some assertion lets the computer follow a different sequence under different conditions.

6. What is the difference between

i. Multiprogramming and multiprocessing?

Multiprogramming
A computer running more than one program at a time (like running Excel and Firefox simultaneously. In a multiprogramming system there are one or more programs loaded in main memory which are ready to execute. Only one program at a time is able to get the CPU for executing its instructions (i.e., there is at most one process running on the system) while all the others are waiting their turn. Note that in order for such a system to function properly, the OS must be able to load multiple programs into separate areas of the main memory and provide the required protection to avoid the chance of one process being modified by another one.

The main idea of multiprogramming is to maximize the use of CPU time. Indeed, suppose the currently running process is performing an I/O task (which, by definition, does not need the CPU to be accomplished). Then, the OS may interrupt that process and give the control to one of the other in-main-memory programs that are ready to execute (i.e. *process context switching*). In this way, no CPU time is wasted by the system waiting for the I/O task to be completed, and a running process keeps executing until either it voluntarily releases the CPU or when it blocks for an I/O operation.

Multiprocessing
A computer using more than one CPU at a time. In a uni-processor system, only one process executes at a time. Multiprocessing is the use of two or more CPUs (processors) within a single Computer system. The term also refers to the ability of a system to support more than one processor within a single computer system. Now since there are multiple processors available, multiple processes can be executed at a time. These multi processors share the computer bus, sometimes the clock, memory and peripheral devices also.

In fact, multiprocessing refers to the *hardware* (i.e., the CPU units) rather than the *software* (i.e., running processes). If the underlying hardware provides more than one processor then that is multiprocessing. Several variations on the basic scheme exist, e.g., multiple cores on one die or multiple dies in one package or multiple packages in one system.

ii. Multiprogramming and multithreading?

Multiprogramming
In a modern computing system, there are usually several concurrent application processes which want to execute. Now it is the responsibility of the Operating System to manage all the processes effectively and efficiently. One of the most important aspects of an Operating System is to multi program. In a computer system, there are multiple processes waiting to be executed, i.e. they are waiting when the CPU will be allocated to them and they begin their execution. These processes are also known as jobs. Now the main memory is too small to accommodate all of these processes or jobs into it. Thus, these processes are initially kept in an area called job pool. This job pool consists of all those processes awaiting allocation of main memory and CPU.

CPU selects one job out of all these waiting jobs, brings it from the job pool to main memory and starts executing it. The processor executes one job until it is interrupted by some external factor or it goes for an I/O task.

Multithreading

A thread is a basic unit of CPU utilization. Multi threading is an execution model that allows a single process to have multiple code segments (i.e., threads) running concurrently within the "context" of that process whiles sharing the CPU time. Multi threading is the ability of a process to manage its use by more than one user at a time and to manage multiple requests by the same user without having to have multiple copies of the program.

You can think of threads as child processes that share the parent process resources but execute independently. Multiple threads of a single process can share the CPU in a single CPU system or (purely) run in parallel in a multiprocessing system