CS8491 COMPUTER ARCHITECTURE L T P C 3 0 0 3

- To learn the basic structure and operations of a computer.
- To learn the arithmetic and logic unit and implementation of fixed-point and floating point arithmetic unit.
- To learn the basics of pipelined execution.
- To understand parallelism and multi-core processors.
- To understand the memory hierarchies, cache memories and virtual memories.
- To learn the different ways of communication with I/O devices

UNIT I BASIC STRUCTURE OF A COMPUTER SYSTEM 9

Functional Units – Basic Operational Concepts – Performance – Instructions: Language of the Computer – Operations, Operands – Instruction representation – Logical operations – decision making – MIPS Addressing.

UNIT II ARITHMETIC FOR COMPUTERS 9

Addition and Subtraction – Multiplication – Division – Floating Point Representation – Floating Point Operations – Subword Parallelism

UNIT III PROCESSOR AND CONTROL UNIT 9

A Basic MIPS implementation – Building a Datapath – Control Implementation Scheme – Pipelining – Pipelined datapath and control – Handling Data Hazards & Control Hazards – Exceptions.

UNIT IV PARALLELISIM 9

Parallel processing challenges – Flynn‘s classification – SISD, MIMD, SIMD, SPMD, and Vector Architectures – Hardware multithreading – Multi-core processors and other Shared Memory Multiprocessors – Introduction to Graphics Processing Units, Clusters, Warehouse Scale Computers and other Message-Passing Multiprocessors.

UNIT V MEMORY & I/O SYSTEMS 9

Memory Hierarchy – memory technologies – cache memory – measuring and improving cache performance – virtual memory, TLB‘s – Accessing I/O Devices – Interrupts – Direct Memory Access – Bus structure – Bus operation – Arbitration – Interface circuits – USB.

TOTAL : 45 PERIODS

OUTCOMES:

On Completion of the course, the students should be able to:

Understand the basics structure of computers, operations and instructions.

Design arithmetic and logic unit.

Understand pipelined execution and design control unit.

Understand parallel processing architectures.

Understand the various memory systems and I/O communication.

TEXT BOOKS:

1. David A. Patterson and John L. Hennessy, Computer Organization and Design: The Hardware/Software Interface, Fifth Edition, Morgan Kaufmann / Elsevier, 2014.
2. Carl Hamacher, Zvonko Vranesic, Safwat Zaky and Naraig Manjikian, Computer Organization and Embedded Systems, Sixth Edition, Tata McGraw Hill, 2012.

REFERENCES:

1. William Stallings, Computer Organization and Architecture – Designing for Performance, Eighth Edition, Pearson Education, 2010.
2. John P. Hayes, Computer Architecture and Organization, Third Edition, Tata McGraw Hill, 2012.
3. John L. Hennessey and David A. Patterson, Computer Architecture – A Quantitative Approach‖, Morgan Kaufmann / Elsevier Publishers, Fifth Edition, 2012.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
QUESTION BANK

SUBJECT : CS8491 COMPUTER ARCHITECTURE
SEM /YEAR : IV/II

UNIT -1- BASIC STRUCTURE OF A COMPUTER SYSTEM

Functional Units – Basic Operational Concepts – Performance – Instructions: Language of the Computer – Operations, Operands – Instruction representation – Logical operations – decision making – MIPS Addressing.

PART A

Q.No	QUESTIONS
1.	**Define computer architecture BTL1** Computer architecture is defined as the functional operation of the individual h/w unit in a computer system and the flow of information among the control of those units.
2.	**Define computer h/w BTL1** Computer h/w is the electronic circuit and electro mechanical equipment that constitutes the Computer
3.	**What are the functions of control unit? BTL2** • The memory arithmetic and logic, and input and output units store and process information and perform i/p and o/p operation • The operation of these unit must be coordinate in some way this is the task of control unit the cu is effectively the nerve center that sends the control signal to other units and sense their states.
4.	**What is an interrupt? BTL2** An interrupt is an event that causes the execution of one program to be suspended and another program to be executed.
5.	**What are the uses of interrupts? BTL2** • Recovery from errors • Debugging • Communication between programs • Use of interrupts in operating system
6.	**What is the need for reduced instruction chip? BTL2** • Relatively few instruction types and addressing modes. • Fixed and easily decoded instruction formats. • Fast single-cycle instruction execution. • Hardwired rather than microprogrammed control.
7.	**Explain the following the address instruction? BTL3** • Three-address instruction-it can be represented as add a,b,c operands a,b are called source operand and c is called destination operand. • Two-address instruction-it can be represented as add a,b • One address instruction-it can be represented as add a • Zero address instruction-it can be represented as Push down stack

8.	**Differentiate between RISC and CISC BTL4** RISC & CISC reduced instruction set computer 1. complex instruction set computer simple instructions take one cycle per operation complex instruction take multiple cycles per operation. few instructions and address modes are used. many instruction and address modes. fixed format instructions are used. variable format instructions are used instructions are compiled and then executed by hardware. instructions are interpreted by the microprogram and then executed. RISC machines are multiple register set. CISC machines use single register set.
9.	**Specify three types of data transfer techniques. BTL1** • Arithmetic data transfer • Logical data transfer • Programmed control data transfer
10.	**What is absolute addressing mode? BTL1** The address of the location of the operand is given explicitly as a part of the instruction. Eg. move a , 2000
11.	**What is the role of MAR and MDR? BTL1** • The MAR (memory address register) is used to hold the address of the location to or from which data are to be transferred • The MDR(memory data register) contains the data to be written into or read out of the addressed location.
12.	**Define CPI BTL1** • The term clock cycles per instruction which is the average number of clock cycles each instruction takes to execute, is often abbreviated as CPI. CPI= CPU clock cycles/instruction count.
13.	**Define throughput and throughput rate. BTL1** • throughput -the total amount of work done in a given time. • throughput rate-the rate at which the total amount of work done at a given time.
14.	**State and explain the performance equation? BTL2** Suppose that the average number of basic steps needed to execute one machine instruction is S,where each basic step is completed in one clock cycle. if the clock cycle rate is R cycles per second, the program execution time is given by T = (N x S) / R this is often referred to as the basic performance equation.

15. **What are the various types of operations required for instructions? BTL1**

- Data transfers between the main memory and the CPU registers
- Arithmetic and logic operation on data
- Program sequencing and control
- I/O transfers

16. **What are the various units in the computer? BTL1**

- Input unit
- Output unit
- Control unit
- Memory unit
- Arithmetic and logical unit

PART B

1 **Explain in detail, the eight ideas in computer architecture. (13m) BTL4**

Answer: U-1 in refer notes

Definition(2m)

Diagram(4m)

Explanation(7m)

- Design for Moore's Law
- Use Abstraction to simplify design
- Make the common case fast
- Performance via parallelism
- Performance via pipelining
- Performance via prediction
- Hierarchy of memories
- Dependability via redundancy

2 **Explain in detail, the components of a computer system. (13m) (Apr/may 2018) BTL4**

Answer: U-1 Refer notes

Explanation(8m)

Diagram(5m)

The five classic components of a computer are input, output, memory, datapath, and control.

3 **Explain in detail, the technologies for building processor and memory. (13m) BTL4**

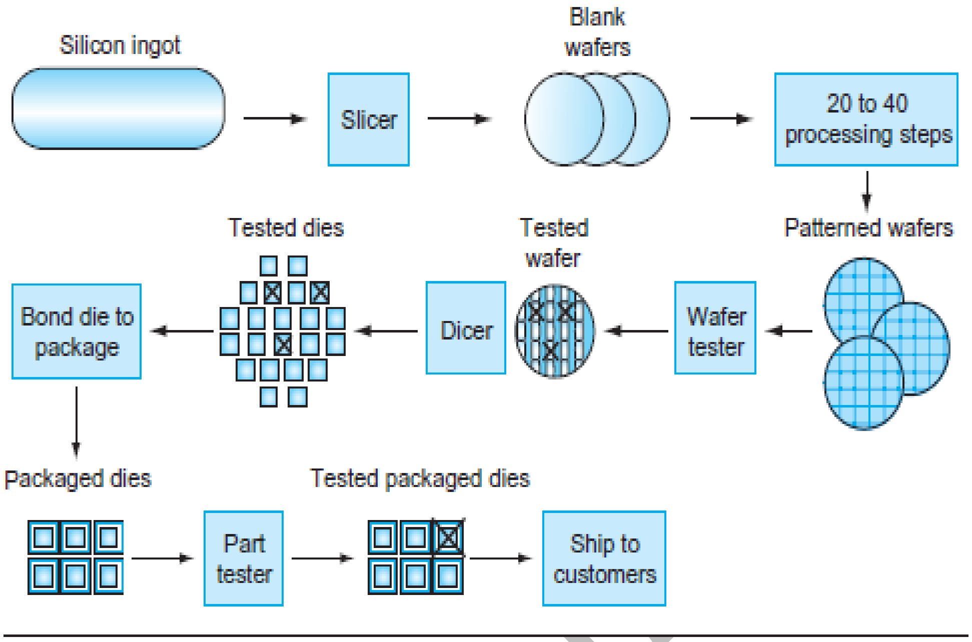
Technologies. (3m)

Answer: U-1 Refer notes

The manufacturing process for integrated circuits: (7m)

- The manufacture of a chip begins with silicon, a substance found in sand. Because silicon does not conduct electricity well, it is called a semiconductor. With a special chemical process, it is possible to add materials to silicon that allow tiny areas to transform into one of three devices:
- Excellent conductors of electricity (using either microscopic copper or aluminum wire)
- Excellent insulators from electricity (like plastic sheathing or glass)
- Areas that can conduct or insulate under special conditions (as a switch) Transistors fall in the last category.
- A VLSI circuit, then, is just billions of combinations of conductors, insulators, and switches manufactured in a single small package. The manufacturing process for integrated circuits is critical to the cost of the chips and hence important to computer designers.
- The process starts with a silicon crystal ingot, which looks like a giant sausage. Today, ingots are 8–12 inches in diameter and about 12–24 inches long. An ingot is finely sliced into wafers no more than 0.1 inches thick.

- These wafers then go through a series of processing steps, during which patterns of chemicals are placed on each wafer, creating the transistors, conductors, and insulators discussed earlier.
- Today's integrated circuits contain only one layer of transistors but may have from two to eight levels of metal conductor, separated by layers of insulators.



The chip manufacturing process:

- After being sliced from the silicon ingot, blank wafers are put through 20 to 40 steps to create patterned wafers.
- These patterned wafers are then tested with a wafer tester, and a map of the good parts is made. Then, the wafers are diced into dies.
- The yield of good dies are then bonded into packages and tested one more time before shipping the packaged parts to customers. One bad packaged part was found in this final test.

Defect: A microscopic flaw in a wafer or in patterning steps that can result in the failure of the die containing that defect.

Die: The individual rectangular sections that are cut from a wafer, more informally known as chips. **Yield:** The percentage of good dies from the total number of dies on the wafer.

The cost of an integrated circuit rises quickly as the die size increases, due both to the lower yield and the smaller number of dies that fit on a wafer. To reduce the cost, using the next generation process shrinks a large die as it uses smaller sizes for both transistors and wires.

$$\text{Cost per die} = \frac{\text{Cost per wafer}}{\text{Dies per wafer} \times \text{yield}}$$

$$\text{Dies per wafer} \approx \frac{\text{Wafer area}}{\text{Die area}}$$

$$\text{Yield} = \frac{1}{(1 + (\text{Defects per area} \times \text{Die area}/2))^2}$$

Diagram(3m)

4 **Explain in detail, the performance of a computer. (13m) BTL4**

Defining Performance:

- If you were running a program on two different desktop computers, you'd say that the faster one is the desktop computer that gets the job done first. If you were running a datacenter that had several servers running jobs submitted by many users, you'd say that the faster computer was the one that completed the most jobs during a day.

- As an individual computer user, you are interested in reducing response time—the time between the start and completion of a task—also referred to as execution time. Datacenter managers are often interested in increasing throughput or bandwidth—the total amount of work done in a given time
- Hence, in most cases, we will need different performance metrics as well as different sets of applications to benchmark personal mobile devices, which are more focused on response time, versus servers, which are more focused on throughput. To maximize performance, we want to minimize response time or execution time for some task. Thus, we can relate performance and execution time for a computer X:

$$\text{Performance}_X = \frac{1}{\text{Execution time}_X}$$

This means that for two computers X and Y, if the performance of X is greater than the performance of Y, we have

$$\text{Performance}_X > \text{Performance}_Y$$

$$\frac{1}{\text{Execution time}_X} > \frac{1}{\text{Execution time}_Y}$$

$$\text{Execution time}_Y > \text{Execution time}_X$$

- That is, the execution time on Y is longer than that on X, if X is faster than Y. To relate the performance of two different computers quantitatively. We will use the phrase "X is n times faster than Y"—or equivalently "X is n times as fast as Y"—to mean

$$\frac{\text{Performance}_X}{\text{Performance}_Y} = n$$

If X is n times as fast as Y, then the execution time on Y is n times as long as it is on X:

$$\frac{\text{Performance}_X}{\text{Performance}_Y} = \frac{\text{Execution time}_Y}{\text{Execution time}_X} = n$$

Measuring Performance: Time is the measure of computer performance:

- The computer that performs the same amount of work in the least time is the fastest. Program execution time is measured in seconds per program. However, time can be defined in different ways, depending on what we count.
- The most straightforward definition of time is called wall clock time, response time, or elapsed time. These terms mean the total time to complete a task, including disk accesses, memory accesses, input/output (I/O) activities, operating system overhead—everything.
- CPU execution time also called CPU time: The actual time the CPU spends computing for a specific task. user CPU time The CPU time spent in a program itself. system CPU time the CPU time spent in the operating system performing tasks on behalf of the program.
- A simple formula relates the most basic metrics (clock cycles and clock cycle time) to CPU time:

$$\begin{matrix}\text{CPU execution time}\\ \text{for a program}\end{matrix} = \frac{\text{CPU clock cycles for a program}}{\text{Clock rate}}$$

Instruction Performance : One way to think about execution time is that it equals the number of instructions executed multiplied by the average time per instruction.

Therefore, the number of clock cycles required for a program can be written as

$$\text{CPU clock cycles} = \text{Instructions for a program} \times \begin{matrix}\text{Average clock cycles}\\ \text{per instruction}\end{matrix}$$

clock cycles per instruction (CPI) Average number of clock cycles per instruction for a program or program

The Classic CPU Performance Equation: The basic performance equation in terms of instruction count (the number of instructions executed by the program),

CPI, and clock cycle time:

$$\text{CPU time} = \text{Instruction count} \times \text{CPI} \times \text{Clock cycle time}$$

or, since the clock rate is the inverse of clock cycle time:

$$\text{CPU time} = \frac{\text{Instruction count} \times \text{CPI}}{\text{Clock rate}}$$

The basic components of performance and how each is measured. These factors are combined to yield execution time measured in seconds per program:

$$\text{Time} = \text{Seconds/Program} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Clock cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Clock cycle}}$$

Instruction mix: A measure of the dynamic frequency of instructions across one or many programs. The performance of a program depends on the algorithm, the language, the compiler, the architecture, and the actual hardware.

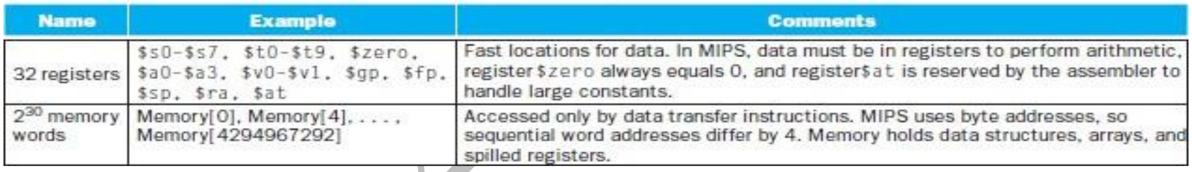
PART-C

1 **Write short notes on : i) Operations and operands ii) Representing instructions iii) Logical and control operations (15m) BTL2**

Operations of the Computer Hardware:

- Every computer must be able to perform arithmetic. The MIPS assembly language Notation add a, b, c instructs a computer to add the two variables b and c and to put their sum in a.

MIPS operands



Name	Example	Comments
32 registers	\$s0-\$s7, \$t0-\$t9, \$zero, \$a0-\$a3, \$v0-\$v1, \$gp, \$fp, \$sp, \$ra, \$at	Fast locations for data. In MIPS, data must be in registers to perform arithmetic, register \$zero always equals 0, and register\$at is reserved by the assembler to handle large constants.
2^{30} memory words	Memory[0], Memory[4], . . . , Memory[4294967292]	Accessed only by data transfer instructions. MIPS uses byte addresses, so sequential word addresses differ by 4. Memory holds data structures, arrays, and spilled registers.

- The natural number of operands for an operation like addition is three: the two numbers being added together and a place to put the sum. Requiring every instruction to have exactly three operands, no more and no less, conforms to the philosophy of keeping the hardware simple: hardware for a variable number of operands is more complicated than hardware for a fixed number.

- Three underlying principles of hardware design:

Design Principle 1: Simplicity favors regularity.

Design Principle 2: Smaller is faster.

Design Principle 3: Good design demands good compromises.

operands of the Computer Hardware:

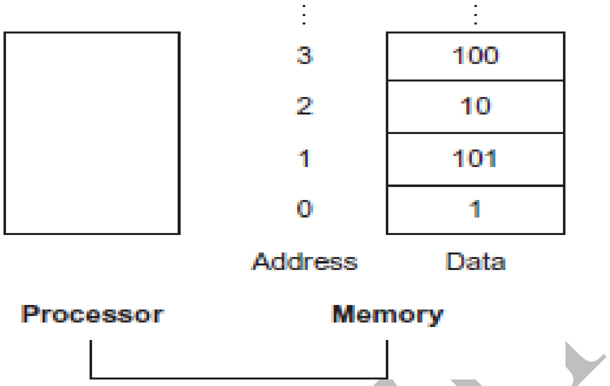
- Unlike programs in high-level languages, the operands of arithmetic instructions are restricted; they must be from a limited number of special locations built directly in hardware called registers.

- Registers are primitives used in hardware design that are also visible to the programmer when the computer is completed, so you can think of registers as the bricks of computer construction.

- The size of a register in the MIPS architecture is 32 bits; groups of 32bits occur so frequently that they are given the name word in the MIPS architecture.

- One major difference between the variables of a programming language and registers is the limited number of registers, typically 32 on current computers, like MIPS.
- The reason for the limit of 32 registers is due to design principles of hardware technology: Smaller is faster.
- A very large number of registers may increase the clock cycle time simply because it takes electronic signals longer when they must travel farther

Memory Operands:



- Data transfer instruction is a command that moves data between memory and registers. Address A value used to delineate the location of a specific data element within a memory array.

Memory addresses and contents of memory at those locations.

- The data transfer instruction that copies data from memory to a register is traditionally called load. The actual MIPS name for this instruction is lw, standing for load word.

lw $t0,8($s3) # Temporary reg $t0 gets A[8]

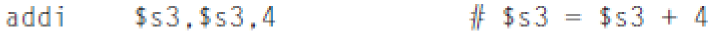
- The instruction complementary to load is traditionally called store; it copies data from a register to memory. The actual MIPS name is sw, standing for store word.

sw $t0,48($s3) # Stores h + A[8] back into A[12]

- Load word and store word are the instructions that copy words between memory and registers in the MIPS architecture.

Constant or Immediate Operands:

- Many times a program will use a constant in an operation—for example, incrementing an index to point to the next element of an array.
- This quick add instruction with one constant operand is called add immediate or addi. To add 4 to register $s3,



```
addi    $s3,$s3,4             # $s3 = $s3 + 4
```

- Computer programs calculate both positive and negative numbers, so we need a representation that distinguishes the positive from the negative.
- The most obvious solution is to add a separate sign, which conveniently can be represented in a single bit; the name for this representation is sign and magnitude.

Signed and Unsigned Numbers:

- Signed versus unsigned applies to loads as well as to arithmetic. The function of a signed load is to copy the sign repeatedly to fill the rest of the register—called sign extension—but its purpose is to place a correct representation of the number within that register.

- Unsigned loads simply fill with 0s to the left of the data, since the number represented by the bit pattern is unsigned.

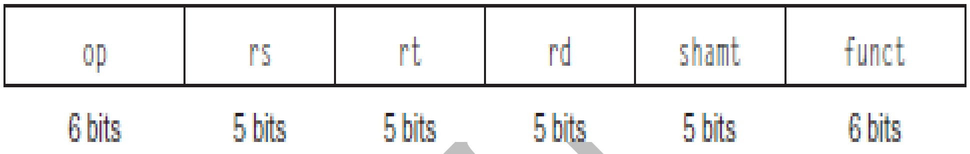
i) Representing instructions

- Instructions are kept in the computer as a series of high and low electronic signals and may be represented as numbers.
- In fact, each piece of an instruction can be considered as an individual number, and placing these numbers side by side forms the instruction.

Instruction format: A form of representation of an instruction composed of fields of binary numbers.

Machine language: Binary representation used for communication within a computer system. Hexa decimal Numbers in base 16.

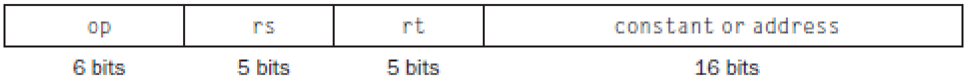
MIPS Fields:



op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

Here is the meaning of each name of the fields in MIPS instructions:

- op: Basic operation of the instruction, traditionally called the opcode.
- rs: The first register source operand.
- rt: The second register source operand.
- rd: The register destination operand. It gets the result of the operation.
- shamt: Shift amount. (Section 2.6 explains shift instructions and this term; it will not be used until then, and hence the field contains zero in this section.)



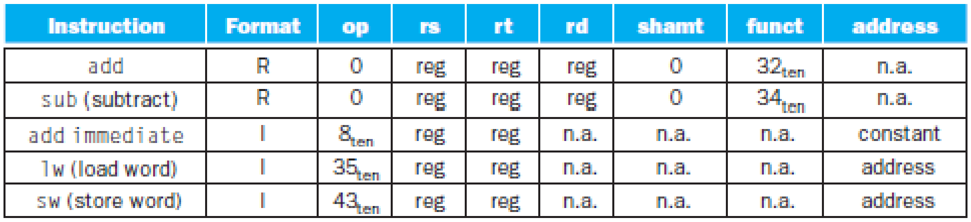
op	rs	rt	constant or address
6 bits	5 bits	5 bits	16 bits

of the operation in the

ngth, thereby requiring

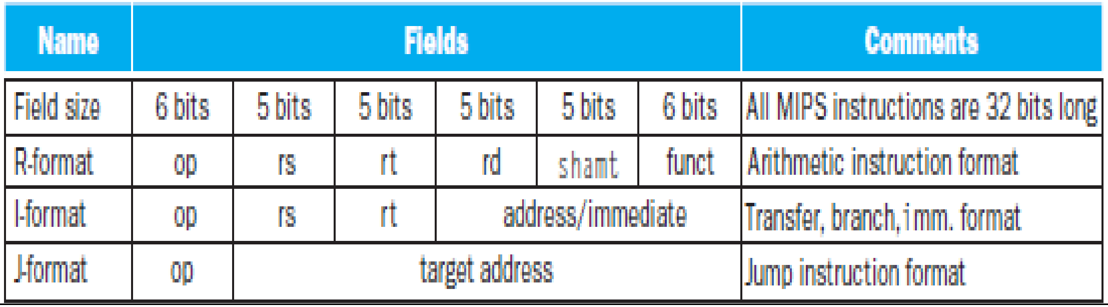
le, the format above is

used by the immediate



Instruction	Format	op	rs	rt	rd	shamt	funct	address
add	R	0	reg	reg	reg	0	32_{ten}	n.a.
sub (subtract)	R	0	reg	reg	reg	0	34_{ten}	n.a.
add immediate	I	8_{ten}	reg	reg	n.a.	n.a.	n.a.	constant
lw (load word)	I	35_{ten}	reg	reg	n.a.	n.a.	n.a.	address
sw (store word)	I	43_{ten}	reg	reg	n.a.	n.a.	n.a.	address

MIPS instruction encoding.

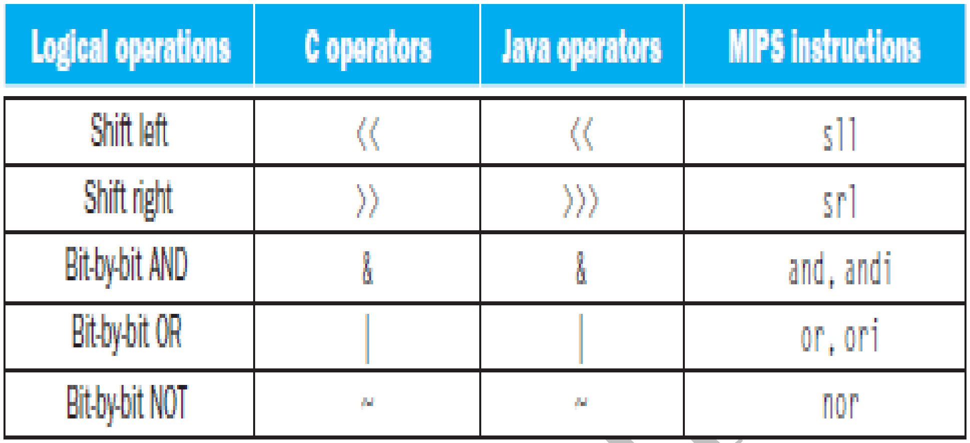


Name	Fields						Comments
Field size	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	All MIPS instructions are 32 bits long
R-format	op	rs	rt	rd	shamt	funct	Arithmetic instruction format
I-format	op	rs	rt	address/immediate			Transfer, branch, imm. format
J-format	op	target address					Jump instruction format

MIPS instruction formats.

(iii) Logical Operations

- The instructions used for the packing and unpacking of bits into words are called logical operations.
- The first class of such operations is called shift s. They move all the bits in a word to the left or right, filling the emptied bits with 0s. For example, if register $s0 contained



Logical operations	C operators	Java operators	MIPS instructions
Shift left	<<	<<	sll
Shift right	>>	>>>	srl
Bit-by-bit AND	&	&	and, andi
Bit-by-bit OR	|	|	or, ori
Bit-by-bit NOT	~	~	nor

0000 0000 0000 0000 0000 0000 0000 1001two = 9ten and the instruction to shift left by 4 was executed, the new value would be: 0000 0000 0000 0000 0000 0000 1001 0000two = 144ten

- The dual of a shift left is a shift right. The actual name of the two MIPS shift instructions are called shift left logical (sll) and shift right logical (srl).

AND: A logical bit by- bit operation with two operands that calculates a 1 only if there is a 1 in both operands. And $t0,$t1,$t2 # reg $t0 = reg $t1 & reg $t2

OR: A logical bit-by bit operation with two operands that calculates a 1 if there is a 1 in either operand.

or $t0,$t1,$t2 # reg $t0 = reg $t1 | reg $t2

NOT: A logical bit-by bit operation with one operand that inverts the bits; that is, it replaces every 1 with a 0, and every 0 with a 1.

NOR: A logical bit-by bit operation with two operands that calculates the NOT of the OR of the two operands. That is, it calculates a 1 only if there is a 0 in both operands.

Instructions for Making Decisions:

- MIPS assembly language includes two decision-making instructions, similar to an if statement with a go to. The first instruction is

beq register1, register2, L1

- This instruction means go to the statement labeled L1 if the value in register1 equals the value in register2. The mnemonic beq stands for branch if equal.
- The second instruction is bne register1, register2, L1 It means go to the statement labeled L1 if the value in register1 does not equal the value in register2.
- The mnemonic bne stands for branch if not equal. These two instructions are traditionally called conditional branches.

the compiled MIPS code for this C if statement if (i == j) f = g + h; else f = g – h; is given as bne $s3,$s4,Else # go to Else if i ≠ j conditional branch

- An instruction that requires the comparison of two values and that allows for a subsequent transfer of control to a new address in the program based on the outcome of the comparison.

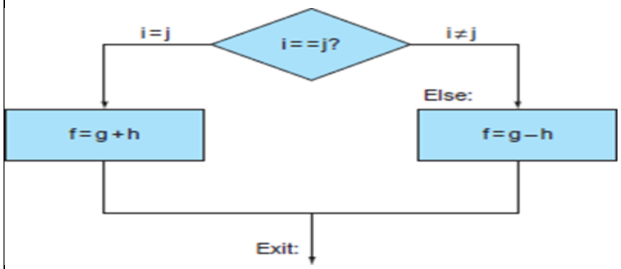
Loops:

- Decisions are important both for choosing between two alternatives—found in ifstatements—and for iterating a computation—found in loops.

Eg1: Loop: sll $t1,$s3,2 # Temp reg $t1 = i * 4

Eg 2: j Loop # go to Loop

Exit:



Case/Switch Statement:

- Most programming languages have a case or switch statement that allows the programmer to select one of many alternatives depending on a single value.
- Jump address table also called jump table. A table of addresses of alternative instruction sequences.

2 **Explain in detail, the Addressing & Addressing Modes. (15m) (Apr/may 2018) BTL4**

Answer: U-1 Refer notes **Carl hamacher book Pageno:48 (10m)**

Immediate addressing, where the operand is a constant within the instruction itself

1. Register addressing, where the operand is a register
2. Base or displacement addressing, where the operand is at the memory location whose address is the sum of a register and a constant in the instruction
3. PC-relative addressing, where the branch address is the sum of the PC and a constant in the instruction
4. Pseudodirect addressing, where the jump address is the 26 bits of the instruction concatenated with the upper bits of the PC.

Diagram(5m)

- Immediate Addressing Mode
- Absolute(Direct) Addressing Mode
- Indirect Addressing Mode
- Register Addressing Mode
- Base with index Addressing Mode
- Base with index & offset Addressing Mode
- Additional Modes(Increment & Decrement Addressing Mode)

UNIT 2- ARITHMETIC FOR COMPUTERS

Addition and Subtraction – Multiplication – Division – Floating Point Representation – Floating Point Operations – Subword Parallelism

PART A

1 **State the principle of operation of a carry look-ahead adder. BTL2**

- The input carry needed by a stage is directly computed from carry signals obtained from all the preceding stages i-1,i-2,…..0, rather than waiting for normal carries to supply slowly from stage to stage.
- An adder that uses this principle is called carry look-ahead adder.

2 **What are the main features of booth's algorithm? BTL1**

- It handles both positive and negative multipliers uniformly.
- It achieves some efficiency in the number of addition required when the multiplier has a few large blocks of 1s.

3 **How can we speed up the multiplication process? BTL3**

There are two techniques to speed up the multiplication process:

- The first technique guarantees that the maximum number of summands that must be added is n/2 for n-bit operands.
- The second technique reduces the time needed to add the summands.

4 **What is bit pair recoding? give an example. BTL1**

- Bit pair recoding halves the maximum number of summands.
- Group the booth-recoded multiplier bits in pairs and observe the following: the pair (+1 -1) is equivalent to the pair (0 +1)that is instead of adding -1 times the multiplicand m at shift position i to +1 the same result is obtained by adding +1

5 **What is the advantage of using booth algorithm? BTL1**

- It handles both positive and negative multiplier uniformly.
- It achieves efficiency in the number of additions required when the multiplier has a few large blocks of 1's.
- The speed gained by skipping 1's depends on the data.

6 **Write the algorithm for restoring division BTL3**

Do the following for n times:

- shift a and q left one binary position.
- subtract m and a and place the answer back in a.
- if the sign of a is 1, set q0 to 0 and add m back to a.

where a- accumulator, m- divisor, q- dividend.

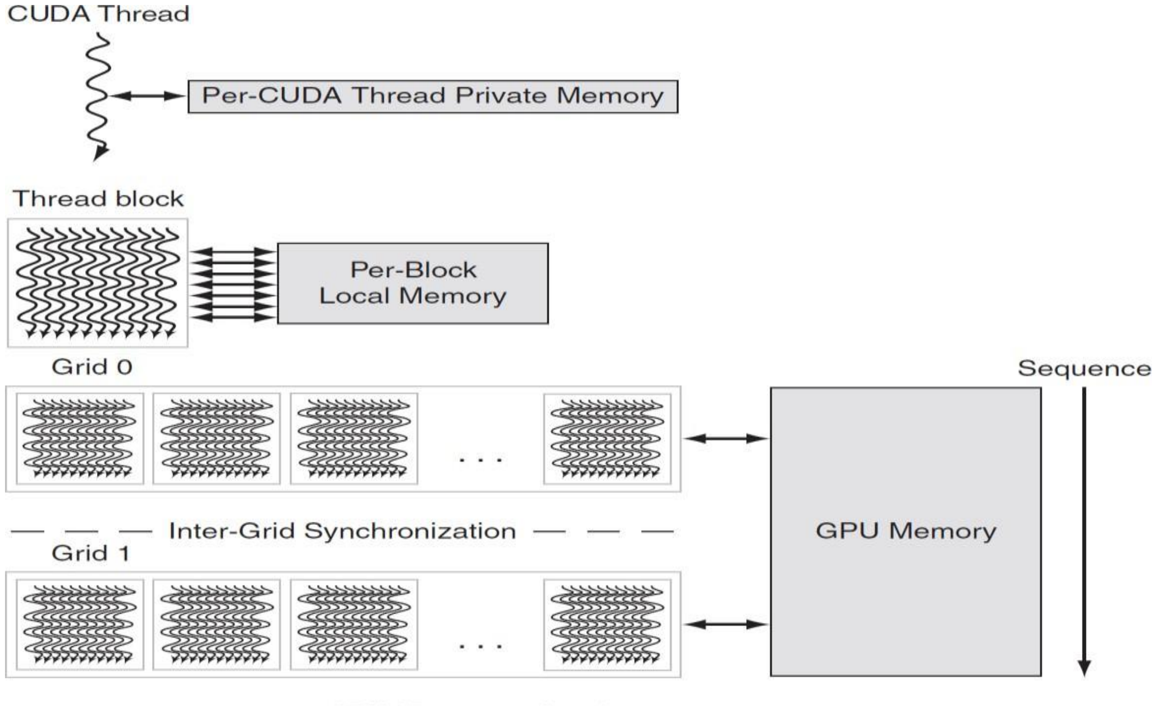
7 **Write the algorithm for non restoring division. BTL3**

Do the following for n times:

	step 1: do the following for n times: • If the sign of a is 0, shift a and q left one bit position and subtract m from a; otherwise, shift a and q left and add m to a. • Now, if the sign of a is 0, set q0 to 1; otherwise, set q0 to0. step 2: if the sign of a is 1, add m to a.
8	**Explain about the special values in floating point numbers. BTL2** The end values 0 to 255 of the excess-127 exponent e are used to represent special values such as: when e= 0 and the mantissa fraction m is zero the value exacts 0 is represented. when e= 255 and m=0, the value is represented. when e= 0 and m=0, denormal values are represented. when e= 2555 and m=0, the value represented is called not a number.
9	**Write the add/subtract rule for floating point numbers. BTL3** • Choose the number with the smaller exponent and shift its mantissa right a number of steps equal to the difference in exponents. • Set the exponent of the result equal to the larger exponent. • Perform addition/subtraction on the mantissa and determine the sign of the result • Normalize the resulting value, if necessary.
10	**Write the multiply rule for floating point numbers. BTL3** • Add the exponent and subtract 127. • Multiply the mantissa and determine the sign of the result. • Normalize the resulting value , if necessary.
11	**What is the purpose of guard bits used in floating point arithmetic BTL1** Although the mantissa of initial operands are limited to 24 bits, it is important to retain extra bits, called as guard bits
12	**What are generate and propagate function? BTL1** • The generate function is given by Gi=XiYi • The propagate function is given as Pi=Xi+Yi.
13	**What is floating point numbers? BTL1** • In some cases, the binary point is variable and is automatically adjusted as computation proceeds. • In such case, the binary point is said to float and the numbers are called floating point numbers.

14	**In floating point numbers when so you say that an underflow or overflow has occurred? BTL5** • In single precision numbers when an exponent is less than -126 then we say that an underflow has occurred. • In single precision numbers when an exponent is less than +127 then we say that an overflow has occurred.
15	**In floating point numbers when so you say that an underflow or overflow has occurred? BTL5** • In single precision numbers when an exponent is less than -126 then we say that an underflow has occurred. • In single precision numbers when an exponent is less than +127 then we say that an overflow has occurred.
PART B	
1	**Summarize about the sub word parallelism. (13m) BTL2** • Since every desktop microprocessor by definition has its own graphical displays, as transistor budgets increased it was inevitable that support would be added for graphics operations. • Many graphics systems originally used 8 bits to represent each of the three primary colors plus 8 bits for a location of a pixel. The addition of speakers and microphones for teleconferencing and video games suggested support of sound as well. Audio samples need more than 8 bits of precision, but 16 bits are sufficient. • Every microprocessor has special support so that bytes and halfwords take up less space when stored in memory (see Section 2.9), but due to the infrequency of arithmetic operations on these data sizes in typical integer programs, there was little support beyond data transfers. Architects recognized that many graphics and audio applications would perform the same operation on vectors of this data. • By partitioning the carry chains within a 128-bit adder, a processor could use parallelism to perform simultaneous operations on short vectors of sixteen 8-bit operands, eight 16-bit operands, four 32-bit operands, or two 64-bit operands. The cost of such partitioned adders was small. • Given that the parallelism occurs within a wide word, the extensions are classified as subword parallelism. It is also classified under the more general name of data level parallelism. They have been also called vector or SIMD, for single instruction, multiple data (see Section 6.6). The rising popularity of multimedia applications led to arithmetic instructions that support narrower operations that can easily operate in parallel. • For example, ARM added more than 100 instructions in the NEON multimedia instruction extension to support subword parallelism, which can be used either with ARMv7 or ARMv8. It added 256 bytes of new registers for NEON that can be viewed as 32 registers 8 bytes wide or 16 registers 16 bytes wide. NEON supports all the subword data types you can imagine except 64-bit floating point numbers: • 8-bit, 16-bit, 32-bit, and 64-bit signed and unsigned integers • 32-bit floating point numbers

Figure shows the memory structures of an NVIDIA GPU. We call the onchip memory that is local to each multithreaded SIMD processor Local Memory.



GPU Memory structures.

- It is shared by the SIMD Lanes within a multithreaded SIMD processor, but this memory is not shared between multithreaded SIMD processors.
- We call the off -chip DRAM shared by the whole GPU and all thread blocks GPU Memory.
- Rather than rely on large caches to contain the whole working sets of an application, GPUs traditionally use smaller streaming caches and rely on extensive multithreading of threads of SIMD instructions to hide the long latency to DRAM, since their working sets can be hundreds of megabytes.
- Thus, they will not fit in the last level cache of a multicore microprocessor.
- Given the use of hardware multithreading to hide DRAM latency, the chip area used for caches in system processors is spent instead on computing resources and on the large number of registers to hold the state of the many threads of SIMD instructions

2 **Explain in detail, the multiplication algorithm, with a neat diagram.(13m) (Apr/may2018) BTL4**

Answer: U-2 Refer notes **carl hamacher book-page no:376**

Explanation:(5m) &Algorithm:(5m)

Step 1: bit=0, shift right C,A &Q

Step 2: bit=1, C,A<-A+B shift right C,A, &Q

Step 3:Check Q0 bit

Diagram:(3m)

3	**Explain in detail, the division algorithm, with a neat diagram. (13m) (Apr/may 2018) BTL4** **Answer: U-2** Refer notes **carl hamacher book-page no:390** **Explanation:(5m) & Algorithm:(5m)** **Step 1: Shift A&Q left 1 binary bit position** **Step 2: Subtract Divisor A<-A-B** **Step 3: Check Sign bit of A & Set Q0** **Diagram:(3m)**
4	**Explain in detail, the flow chart of floating-point multiplication. (13m) BTL4** **Answer: U-2** Refer notes **carl hamacher book-page no:398** **Explanation:(5m) &Algorithm:(5m),** **Step 1: If either multiplicand or multiplier is 0, result will be 0** **Step 2: Add the exponents & subtract bias.** **Step 3: Multiply the mantissas & determine the sign of the result** **Step 4: Result must be normalized** **Diagram:(3m)**
PART C	
1	**Explain in detail, the block diagram of an arithmetic unit for floating-point addition & subtraction. (15m) (Apr/may 2018) BTL4** **Answer: U-2** Refer notes **carl hamacher book-page no:393** **Explanation & Algorithm:(10m),** **Step 1: Change the sign of Q for subtraction & check zero.** **Step 2: Align mantissa** **Step 3: Addition** **Step 4: Normalization** **Diagram:(5m)**
2	**Explain in detail, the addition and subtraction operation. (15m) BTL4** **Answer: U-2** Refer notes **Explanation:(10m),** • **Half adder** • **Full adder** • **Subtractor** • **ALU** • **Examples** **Diagram:(5m)**

UNIT-3 PROCESSOR AND CONTROL UNIT

A Basic MIPS implementation – Building a Datapath – Control Implementation Scheme – Pipelining – Pipelined datapath and control – Handling Data Hazards & Control Hazards – Exceptions.

PART A

1 **Define MIPS. BTL1**

MIPS: one alternative to time as the metric is MIPS (million instruction per second)
MIPS=instruction count/ (execution time x1000000).
This MIPS measurement is also called native MIPS to distinguish it from some alternative definitions of MIPS.

2 **Define MIPS rate. BTL1**

The rate at which the instructions are executed at a given time

3 **Define Pipelining. BTL1**

Pipelining is a technique of decomposing a sequential process into sub operations with each sub process being executed in a special dedicated segment that operates concurrently with all other segments.

4 **Define Instruction pipeline. BTL1**

- The transfer of instructions through various stages of the CPU instruction cycle, including fetch opcode, decode opcode, compute operand addresses.
- Fetch operands, execute instructions and store results. this amounts to realizing most (or) all of the CPU in the form of multifunction pipeline called an instruction pipelining.

5 **What are Hazards? BTL1**

- A hazard is also called as hurdle.
- The situation that prevents the next instruction in the instruction stream from executing during its designated clock cycle. stall is introduced by hazard. (ideal stage).

6 **State different types of hazards that can occur in pipeline. BTL1&2**

The types of hazards that can occur in the pipelining were,

- Data hazards.
- Instruction hazards.
- Structural hazards.

7 **Define Data hazards. BTL1**

A data hazard is any condition in which either the source or the destination operands of an instruction are not available at the time expected in pipeline, as a result some operation has to be delayed, and the pipeline stalls.

8 **Define Instruction hazards. BTL1**

- The pipeline may be stalled because of a delay in the availability of an instruction.
- For example, this may be a result of miss in cache, requiring the instruction to be fetched from the main memory. such hazards are called as instruction hazards or control hazards

9 **Define Structural hazards. BTL1**

- The structural hazards is the situation when two instructions require the use of a given hardware resource at the same time.
- The most common case in which this hazard may arise is access to memory.

10 **How data hazard can be prevented in pipelining? BTL5**

Data hazards in the instruction pipelining can prevented by the following techniques.

- Operand forwarding
- Software approach

11 **How addressing modes affect the instruction pipelining? BTL5**

- Degradation of performance is an instruction pipeline may be due to address dependency

where operand address cannot be calculated without available information needed by addressing mode.

- For e.g. an instruction with register indirect mode cannot proceed to fetch the

operand if the previous instructions is loading the address into the register. hence operand access is delayed degrading the performance of pipeline.

12 **How compiler is used in pipelining? BTL5**

- A compiler translates a high level language program into a sequence of machine instructions.
- To reduce n, we need to have suitable machine instruction set and a compiler that makes good use of it.
- An optimizing compiler takes advantages of various features of the target processor to reduce the product n*s, which is the total number of clock cycles needed to execute a program.
- The number of cycles is dependent not only on the choice of instruction, but also on the order in which they appear in the program.
- The compiler may rearrange program instruction to achieve better performance of course, such changes must not affect of the result of the computation.

13 **List out the methods used to improve system performance. BTL1**

The methods used to improve system performance are

- Processor clock
- Basic performance equation
- Pipelining
- Clock rate
- Instruction set
- Compiler

14 **How the interrupt is handled during exception? BTL5**

- CPU identifies source of interrupt

	• CPU obtains memory address of interrupt handles • PC and other CPU status information are saved • PC is loaded with address of interrupt handler and handling program to handle it.
15	**What is branch delay slot? BTL1** The location containing an instruction that may be fetched and then discarded because of the branch is called branch delay slot.
16	**List out the advantages of pipelining Apr/May 2016 BTL1** 1. The Instruction cycle time of the processor is reduced increasing, instruction throughput. **2.** Increase in pipeline stages increase number of instructions that can be processed at once which reduces delay between completed instructions.
17	**Define Exception. Apr/May 2016 BTL1** Exceptions are internally generated unscheduled events that disrupt program execution & they are used to detect overflow. On the other hand, interrupt comes from outside of the processor.
18	**Web server is to be enhanced with a new CPU which is 10 times faster on computation than old CPU The original CPU spent 40% its time processing and 60% of its time waiting for I/O. What will be the overall speedup**? **Nov/Dec 2018 BTL1** Overall speedup= $\frac{0.4*10+0.6}{0.4+0.6} = 4.6$
19	**List the types of Exception BTL1** Precise Exception- partially executed instructions are discarded. Imprecise Exception- instructions executed to completion
20	**List out the common steps to implement any type of instruction Nov/Dec 2018 BTL1** Fetch & Decode
	PART B
1	**Explain in detail, the basic implementation of MIPS. (13m) BTL4** **Answer: U-3** refer notes pageno:3 Explanation:8m The Basic MIPS Implementation An Overview of the Implementation Diagram:5m
2	**Explain in detail, the steps involved in building a data path unit. (13m) (Apr/May 2018) BTL4** **Answer: U-3 Refer** Notes pageno:1 Explanation:8m

	• Building a datapath • Types of Elements in the Datapath • Datapath Segment for ALU, LW & SW, Br. Instructions Diagram:5m
3	**Explain in detail about the operation of datapath & Control Nov/Dec2017 BTL4** **Building a datapath/Operation (7)** • Building a datapath • Types of Elements in the Datapath • Datapath Segment for ALU, LW & SW, Br. Instructions • Diagram **Control (6)** • Control Implementation scheme • ALU Control • Designing the main control unit • Format for R, L&S, Br. Instructions • Important observations about this Ins. Format • Table/Cmp- Functions of Seven Single bit control Lines • Diagram
4	**Explain in detail, the design of the main control unit. (13m) BTL4** **Answer: U-3** Refer Notes Explanation(8m) • Control Implementation scheme • ALU Control • Designing the main control unit • Format for R, L&S, Br. Instructions • Important observations about this Ins. Format • Table/Cmp- Functions of Seven Single bit control Lines Diagram:(5m)
5	**Explain in detail, the pipelined data path and control. (13m) (Apr/May 2018) BTL5** **Answer: U-3** Refer Notes **carl hamacher book-page no:479** **Explanation(8m)** • Implementation of 2 stage instruction pipelining • Organization of CPU with 4 stage Instruction pipelining • Implementation of MIPS Instruction Pipeline **The Pipelined Control & datapath(5m)** • Instruction fetch: • Instruction decode and register file read:

	• Execute or address calculation • Memory access: • Write-back: Diagram
6	**Discuss the modified datapath to accommodate pipelined executions with a diagram Apr/ May 2017 (13m) BTL2** **Explanation (8m)** • Data Hazard • Operand Forwarding **Diagram (5m)**
7	(i)**Discuss the hazards caused by unconditional branching statements (6m) Apr/ May 2017 BTL2** Explanation (3) Control Hazards Unconditional Branching- Effect of Branching in 2- stage pipelining Branch penalty Diagram(3) (ii**) Describe operand forwarding in a pipeline processor with a diagram (6m)** Explanation (4) • Data Hazard • Operand Forwarding Diagram(3m)

8	**Explain in detail, the instruction hazards. (13m) BTL4**
	Answer: U-3 Refer Notes, **Carl hamacher book pageno:465**
	Explanation(10m)
	Diagram(3m)

9	**Why is branch prediction algorithm needed? Differentiate between the static & dynamic techniques Nov/dec 2016 BTL2&3** **Explanation (10)** • Branch Prediction • Branch prediction strategies • Difference between the static & dynamic branch strategy • A typical state diagram used in dynamic branch prediction **Diagram (3)**
10	**Explain in detail how exceptions are handled in MIPS Architecture Apr/May 2015 BTL4** **Explanation (11)** • Example of Except & Interrupt(2m) • types of Exception • Response to an Exception • Methods used to communicate the reason for an Exception • Exceptions & Interuppts are classified into two types • Precise • Imprecise
	PART C
1.	**Explain the overview of pipelining. (15m) BTL4** **Answer: u-3** Refer Notes **carl hamacher book-page no:454** Explanation(10m) Diagram(5m) An Overview of Pipelining:

Designing Instruction Sets for Pipelining:
Pipeline Hazards:

2. **(i) Explain in detail, the pipeline hazards. (9m) BTL4**

Answer : U-3Refer notes

Explanation(7m)
Pipeline Hazards
Structural Hazards
Data Hazards
Control Hazards
Diagram(2m)

(ii) **A pipelined processor uses delayed branch technique. Recommend any one of the following possibilities for the design of the processor. In the 1st possibility, the processor has a 4- stage pipeline and one delay slots. In the 2nd possibility, it has a 6- stage pipeline & two delay slots. Compare the performance of these two alternatives, taking only the branch penalty into account. Assume that 20% of the instructions are branch instructions and that an optimizing compiler has an 80% success rate in filling in the single delay slot. For the Second alternative, the compiler is able to fill the second slot 25% of the time.** Apr/May 2017 BTL4

Answer: Given 20% of ins. Are br. Ins. & compiler can fill 80% of 1st delay slot & 25% of 2nd delay slot.

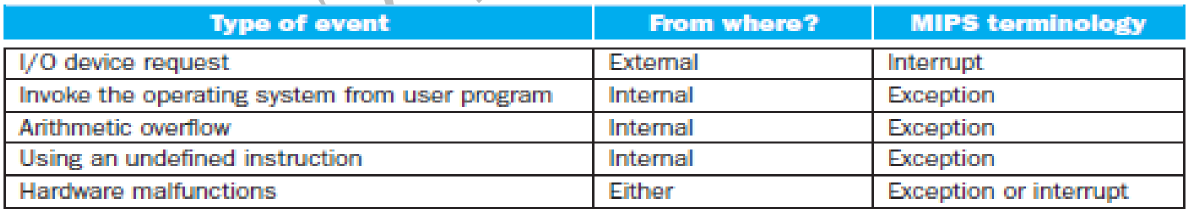
Throughput improvement due to pipeline is n, where n is the number of pipeline stages.

Stage	**No. of cycles needed to execute one instruction**	**Throughput**
4-Stage	1+0.2-0.8*0.2=1.04	4/1.04 = 3.85
6-Stage	1+(0.2*2)-0.8*0.2-0.25*0.2=1.19	6/1.19 = 5.04

3 **Summarize about the exceptions. (15m) (Apr/May 2018) BTL2**

Answer: U-3 Refer Notes, **carl hamacher book-page no:218**

Explanation (10m)



Type of event	From where?	MIPS terminology
I/O device request	External	Interrupt
Invoke the operating system from user program	Internal	Exception
Arithmetic overflow	Internal	Exception
Using an undefined instruction	Internal	Exception
Hardware malfunctions	Either	Exception or interrupt

Example of Except & Interrupt(2m)

- types of Exception
- Response to an Exception
- Methods used to communicate the reason for an Exception
- Exceptions & Interuppts are classified into two types
- Precise
- Imprecise

Diagram(3m)

4 **Interpret a processor has 5 individual stages, namely. IF, ID, EX, MEM, WB and their latencies are 250ps, 350ps, 150ps, 300ps & 200ps respectively. The frequency of the instructions executed by the processor are as follows: ALU: 40%, branch 25%, Load 20% and store 15%. What is the clock cycle time in a pipelined & non-pipelined processor? If you can split one stage of the pipelined datapath into two new stores,**

each with half the latency of the original stage, which stage would you split & what is the new clock cycle time of the processor? Assuming there are no stalls or hazards, what is the utilization of the write-register port of the "Registers" unit? Nov/Dec 2018 BTL3

Answer

(a) Clock cycle tome in a pipelined processor=350ps
Clock cycle time in non-pipelined processor= 250+350+150+300+200=1250ps

(b) We have to split one stage of the pipelined datapath which has a maximum latency i.e, ID
After splitting ID stage with latencies ID1=175ps
ID2=175ps
We have new clock cycle time of the processor equal to 300ps

(c) Assuming there are no stalls or hazards, the utilization of the
data memory= 20% to 15%=35%

(d) Assuming there are no stalls or hazards, the utilization of the write reg. port of the reg. Unit = 40%+25% = 65%

5 **Summarize the following sequence of instructions are executed in the basic 5- stage pipelined processor Apr/May 2018 BTL3/4 (14m)**

OR r1, r2, r3

OR r2, r1, r4

OR r1, r1, r2

(i) Indicate dependences & their type

Answer:

RAW- dependency in r1 between Instruction 1,2 & 3

RAW- dependency in r2 between Instruction 2 & 3

WAR- in r2 from Instructions 1 to 2

WAR- in r1 from Instructions 2 to 3

WAR- in r1 from Instructions 1 to 3

(ii) Assume there is no forwarding in this pipelined processor. Indicate hazards & add NOP instructions to eliminate them.

Answer:

No hazards form WAR, WAW, Since there are 5 stages RAW cause data Hazards

OR r1, r2, r3
NOP
NOP
OR r2, r1, r4
NOP
NOP
OR r1, r1, r2

(iii) Assume there is full forwarding. Indicate hazards & add NOP instructions to eliminate

	them. **Answer:** In full forwarding the data hazards above are eliminated, thus there is no need for NOP instructions.
7	**Explain about the Parallelism via Instructions. (15m) BTL4** **Answer: U-3** Refer Notes Pageno:11 **Explanation(13m)** ILP Implementing a Multiple Issue Processor Speculation Static Multiple Issue Dynamic Multiple Issue ➢ True Data dependency ➢ Procedural Dependency ➢ Resource Conflict ➢ Output dependency ➢ Antidependency Recovery mechanism Instruction- Issue Policy ➢ In-order issue with In-order completion ➢ In-order issue with Out-order completion ➢ Out-order issue with Out-order completion Register Renaming Branch Prediction Diagram(2m)

UNIT 4- PARALLELISM

Parallel processing architectures and challenges, Flynn's Classification, Hardware multithreading, Multicore and shared memory multiprocessors, Introduction to Graphics Processing Units, Clusters and Warehouse scale computers – Other Message passing Multiprocessors

PART A

1 **What is instruction level parallelism? BTL1**

Pipelining is used to overlap the execution of instructions and improve performance. this potential overlap among instructions is called instruction level parallelism (ILP).

2 **List various types of dependences in ILP. BTL1**

- Data dependences
- Name dependences
- Control dependences

3 **What is Multithreading? BTL1**

Multithreading allows multiple threads to share the functional units of a single processor in an overlapping fashion.to permit this sharing, the processor must duplicate the independent state of each thread.

4 **What are multiprocessors? mention the categories of multiprocessors? BTL1**

Multiprocessor are used to increase performance and improve availability. the different categories are SISD, SIMD, MISD, MIMD.

5 **What are two main approaches to multithreading? BTL1**

- fine-grained multithreading
- coarse-grained multithreading

6 **What is the need to use multiprocessors? BTL2**

- Microprocessors as the fastest CPUs collecting several much easier than redesigning
- Complexity of current microprocessors do we have enough ideas to sustain 1.5x/yr?

can we deliver such complexity on schedule?

- Slow (but steady) improvement in parallel software (scientific apps, databases, os)
- Emergence of embedded and server markets driving microprocessors in addition to desktops embedded functional parallelism, producer/consumer model server figure of merit is tasks per hour vs. latency

7 **Write the software implications of a multicore processor? BTL2**

- Multi-core systems will deliver benefits to all software, but especially multi-threaded programs.
- All code that supports the technology or multiple processors, for example, will benefit automatically from multicore processors, without need for modification. most server-side enterprise packages and many desktop productivity tools fall into this category

8 **Define parallel processing. BTL1**

Processing data concurrently is known as parallel processing

9 **Define multiprocessor system. BTL1**

A computer system with atleast two processor is called multiprocessor system

10 **Define parallel processing program. BTL1**

A single program that runs on multiple processors simultaneously

11 **What is cluster? BTL1**

A set of computers connected over a local area network that function as single large multiprocessor is

	called cluster
12	**What is multicore? BTL1** A multicore is an architectural design that places multiple processors on a single computer chip to enhance performance and allow simultaneous process of multiple tasks more efficiently. Each processor is called core
13	**List the Flynn's Classification BTL1 Dec2014** SISD SIMD MISD MIMD
13	**Differentiate between Strong Scaling & weak Scaling BTL2 Dec-17** Strong scaling: Speedup achieved on a multiprocessor without increasing the size of the problem. Weak Scaling: Speedup achieved on a multiprocessor while increasing the size of the problem proportionally to the increase in the number of processors.
14	**Compare UMA and NUMA multiprocessor BTL2 Dec-15** **UMA:** A multiprocessor in which latency to any word in main memory is about the same no matter which processor requests the access. **NUMA:** A type of single address space multiprocessor in which some memory accesses are much faster than others depending on which processor asks for which word
15	**What is Fine grained multithreading? BTL1 May-16** A version of hardware multithreading that suggests switching between threads after every instruction is called fine-grained multithreading
16	**Distinguish implicit multithreading and explicit multithreading BTL2 May-17** Implicit multithreading refers to the concurrent execution of multiple threads extracted from a single sequential program. Explicit Multithreading refers to the concurrent execution execution of instructions from different explicit threads, either by interleaving instructions from different threads on shared pipelines or by parallel execution on parallel pipelines
17	**State the Amdahl's law? BTL1 Dec-14** It states that the performance improvement to be gained from using some faster mode of execution is limited by the fraction of the time the faster mode can be used
18	**What is SaaS(Software as a Service)** Saas is a software that runs at a remote site and made available over the internet typically via a Web interface to customers. SaaS customers are charged based on use versus on ownership.
19	**Protein string matching code has 4 days execution time on current machine doing integer instructions in 20% of time, doing I/O in 35% of time and other operations in the remaining time.**

Which is better tradeoff among the following two proposals? First: Compiler optimization that reduces number of integer instructions by 25%(assume each integer instruction takes the same amount of time); Second: Hardware optimization that reduces the latency of each I/O operations from 6µs to 5µs. BTL2 May-18

Solution:

If we can speed up X of the program by S times, Amdahl's law gives the total speedup, S_{tot}.

$S_{tot}=1/(X/S+(1-X)$

First case: Speed integer instruction time

X=0.2

S=1/(1-0.25)=1.33

$S_{INT}=1/[(0.2/1.33)+(1-0.2)]$

= 1.052

Second case: Speedup I/O operation time.

X=0.35

S=**6µs/5µs**

= 1.2

$S_{IO}=1/[(0.35/1.2)+(1-0.35)]$

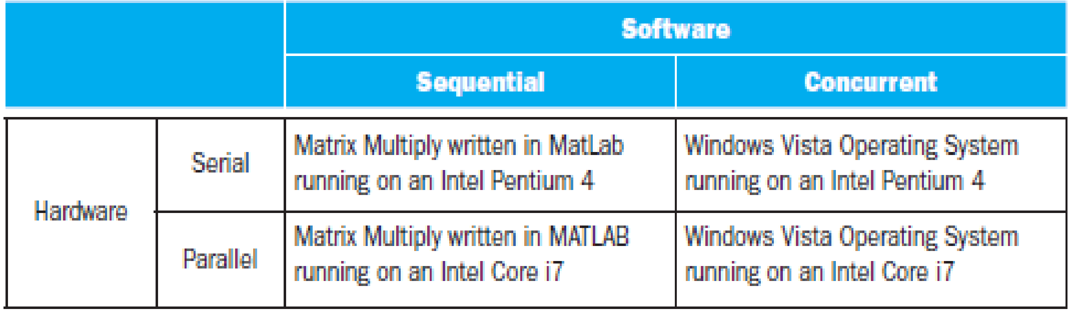
= 1.062

Thus, speeding up I/O operations is done.

PART B

1 **Explain the challenges in parallel processing. (13m) (Apr/May 2018) BTL4**

- The tall challenge facing industry is to create hardware and software that will make it easy to write correct parallel processing programs that will execute efficiently in performance and energy as number of cores per chip scales.
- Only challenge of parallel revolution is figuring out how to make naturally sequential software have high performance on parallel hardware, but it is also to make concurrent programs have high performance on multiprocessors as number of processors increases.
- The difficulty with parallelism is not hardware; it is that too few important application programs have been rewritten to complete tasks sooner on multiprocessors.
- It is difficult to write software that uses multiple processors to complete one task faster, and problem gets worse as number of processors increases.
- The first reason is that you must get better performance or better energy efficiency from a parallel processing program on a multiprocessor; why is it difficult to write parallel processing programs that are fast, especially as number of processors increases



		Software	
		Sequential	Concurrent
Hardware	Serial	Matrix Multiply written in MatLab running on an Intel Pentium 4	Windows Vista Operating System running on an Intel Pentium 4
	Parallel	Matrix Multiply written in MATLAB running on an Intel Core i7	Windows Vista Operating System running on an Intel Core i7

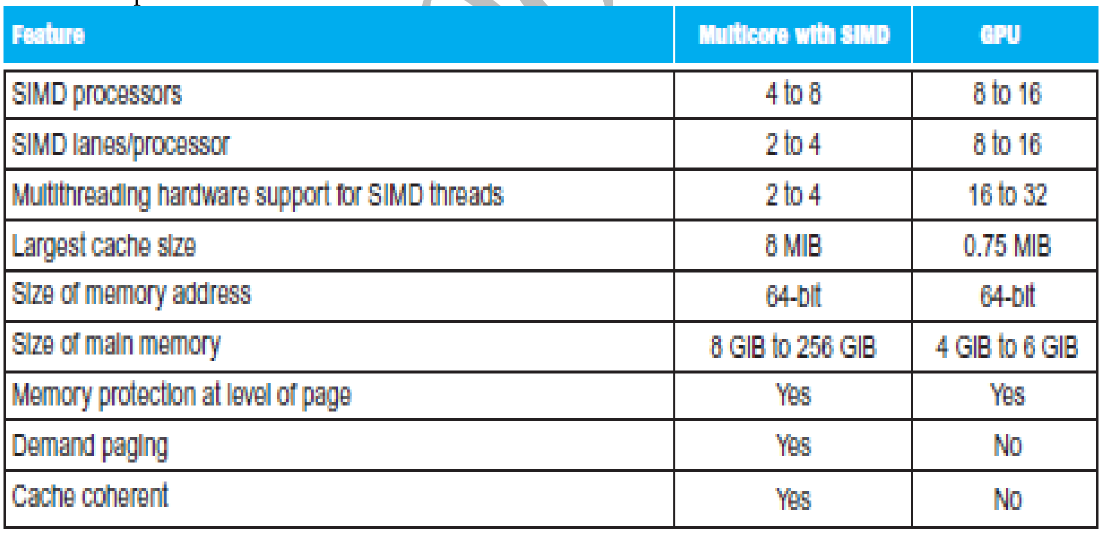
- For both analogy and parallel programming, challenges include scheduling, partitioning work into parallel pieces, balancing load evenly between workers, time to synchronize, and overhead for communication between parties.
- The challenge is stiffer with more reporters for a newspaper story and with more processors for parallel programming.
- Another obstacle, namely Amdahl's Law. It reminds us that even small parts of a program

	must be parallelized if program is to make good use of many cores. Speed-up Challenge: - Suppose you want to achieve a speed-up of 90 times faster with100 processors. - What percentage of original computation can be sequential? Amdahl's Law in terms of speed-up versus original execution time: $\text{Speed-up} = \dfrac{\text{Execution time before}}{(\text{Execution time before} - \text{Execution time affected}) + \dfrac{\text{Execution time affected}}{\text{Amount of improvement}}}$ 0.1% **Speed-up Challenge: Balancing Load** $\text{Speed-up} = \dfrac{1}{(1 - \text{Fraction time affected}) + \dfrac{\text{Fraction time affected}}{\text{Amount of improvement}}}$ - Example demonstrates importance of balancing load, for just a single processor with twice load of the others cuts speed-up by a third, and five times load on just one processor reduces speed-up by almost a factor of three.
2	**Explain in detail, hardware multithreading unit. (13m) (Apr/May 2018) BTL4** **Answer: U-5** Refer Notes Pageno:5 **Explanation(10m)** - Interleaved - Blocked - Simultaneous(SMT) - Chip processing - Scalar - Superscalar - VLSW **Diagram(3m)**
3	**Summarize about the Introduction to Graphics Processing Units (GPU) (13m) BTL2** - The original justification for adding SIMD instructions to existing architectures was that many microprocessors were connected to graphics displays in PCs and workstations, so an increasing fraction of processing time was used for graphics. - As Moore's Law increased number of transistors available to microprocessors, it therefore made sense to improve graphics processing. - A major driving force for improving graphics processing was computer game industry, both on PCs and in dedicated game consoles such as Sony PlayStation. - The rapidly growing game market encouraged many companies to make increasing investments in developing faster graphics hardware, and positive feedback loop led graphics processing to improve at a faster rate than general-purpose processing in mainstream microprocessors. - Given that graphics and game community had different goals than microprocessor development community, it evolved its own style of processing and terminology. - As graphics processors increased in power, they earned name Graphics Processing Units or GPUs to distinguish themselves from CPUs. For a few hundred dollars, anyone can buy a GPU today with hundreds of parallel floating-point units, which makes high-performance computing more accessible. - The interest in GPU computing blossomed when potential was combined with a

programming language that made GPUs easier to program. Hence, many programmers of scientific and multimedia applications today are pondering whether to use GPUs or CPUs.

Here are some of key characteristics as to how GPUs vary from CPUs:

- GPUs are accelerators that supplement a CPU, so y do not need be able to perform all tasks of a CPU.
- This role allows m to dedicate all their resources to graphics. It's fine for GPUs to perform some tasks poorly or not at all, given that in a system with both a CPU and a GPU, CPU can do m if needed.
- The GPU problems sizes are typically hundreds of megabytes to gigabytes, but not hundreds of gigabytes to terabytes. These differences led to different styles of architecture:
- Perhaps biggest difference is that GPUs do not rely on multilevel caches to overcome long latency to memory, as do CPUs.
- Instead, GPUs rely on hardware multithreading (Section 6.4) to hide latency to memory. That is, between time of a memory request and time that data arrives, GPU executes hundreds or thousands of threads that are independent of that request.
- The GPU memory is thus oriented toward bandwidth rather than latency. There are even special graphics DRAM chips for GPUs that are wider and have higher bandwidth than DRAM chips for CPUs.
- In addition, GPU memories have traditionally had smaller main memories than conventional microprocessors. In 2013, GPUs typically have 4 to 6 GiB or less, while CPUs have 32 to 256 GiB.
- Finally, keep in mind that for general-purpose computation, you must include time to transfer data between CPU memory and GPU memory, since GPU is a coprocessor.
- Given reliance on many threads to deliver good memory bandwidth, GPUs can accommodate many parallel processors (MIMD) as well as many threads.
- Hence, each GPU processor is more highly multithreaded than a typical CPU, plus y have more processors.



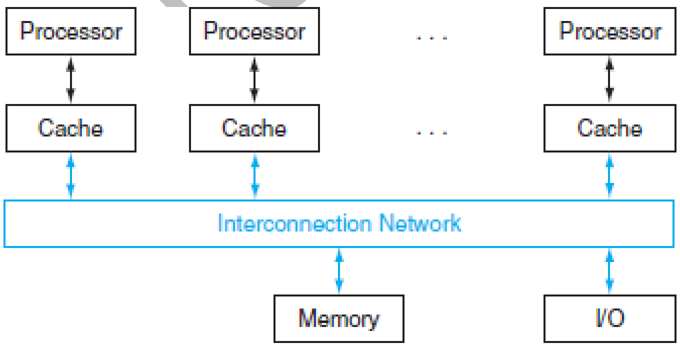
Feature	Multicore with SIMD	GPU
SIMD processors	4 to 8	8 to 16
SIMD lanes/processor	2 to 4	8 to 16
Multithreading hardware support for SIMD threads	2 to 4	16 to 32
Largest cache size	8 MiB	0.75 MiB
Size of memory address	64-bit	64-bit
Size of main memory	8 GiB to 256 GiB	4 GiB to 6 GiB
Memory protection at level of page	Yes	Yes
Demand paging	Yes	No
Cache coherent	Yes	No

- Similarities and differences between multicore with Multimedia SIMD extensions and recent GPUs.
- At a high level, multicore computers with SIMD instruction extensions do share similarities with GPUs.
- Both are MIMDs whose processors use multiple SIMD lanes, although GPUs have more processors and many more lanes.
- Both use hardware multithreading to improve processor utilization, although GPUs have hardware support for many more threads.
- Both use caches, although GPUs use smaller streaming caches and multicore

	computers use large multilevel caches that try to contain whole working sets completely. • Both use a 64-bit address space, although physical main memory is much smaller in GPUs. While GPUs support memory protection at page level, y do not yet support demand paging. • SIMD processors are also similar to vector processors. • The multiple SIMD processors in GPUs act as independent MIMD cores, just as many vector computers have multiple vector processors.
4.	**Explain in detail about the multicore & shared memory multiprocessors with a neat diagram(13m) BTL4** **Answer:** Refer notes • Introduction • Type1, Type2, Type3 • Diagram • Shared memory • UMA • NUMA • Diagram
5	**Describe about the Flynn's classification with a neat diagram (13m) BTL2** **Answer:** Refer notes Explanation- 9m Diagram – 4m • Introduction • SISD • SIMD • MISD • MIMD
	PART C
1	**Explain in detail, the GPA with a neat diagram. (15m) BTL4** **Answer: U-5** refer notes Explanation (12m) • Introduction • GPU vs CPU • Connection between CPU & GPU • GPU Architecture • An Introduction to the NVIDIA GPU Architecture Diagram (3m)
2	**Explain in detail about the introduction to Multiprocessor network topologies. (15m) BTL1** **Answer: Carl Hamacher book pageno:624** Explanation(10m) • Time shared Bus or common bus • Crossbar Switch • Multiport memory • Multistage Switching networks • Hypercube Interconnection Diagram(5m)

3 **Explain in detail, the shared memory multiprocessor, with a neat diagram. (15m) (Apr/May 2018) BTL4**

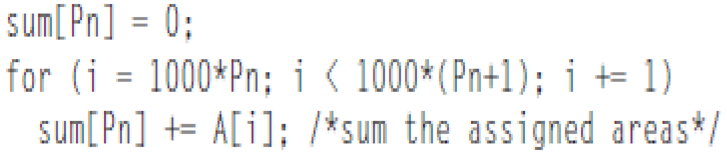
- Shared memory multiprocessor (SMP) is one that offers programmer a single physical address space across all processors-which is nearly always case for multicore chips
- Although a more accurate term would have been shared-address multiprocessor. Processors communicate through shared variables in memory, with all processors capable of accessing any memory location via loads and stores.
- Note that such systems can still run independent jobs in their own virtual address spaces, even if y all share a physical address space.
- Single address space multiprocessors come in two styles. In first style, latency to a word in memory does not depend on which processor asks for it.
- Such machines are called uniform memory access (UMA) multiprocessors. In second style, some memory accesses are much faster than others, depending on which processor asks for which word, typically because main memory is divided and attached to different microprocessors or to different memory controllers on same chip.
- Such machines are called non uniform memory access (NUMA) multiprocessors. As you might expect, programming challenges are harder for a NUMA multiprocessor than for a UMA multiprocessor, but NUMA machines can scale to larger sizes and NUMAs can have lower latency to nearby memory.
- As processors operating in parallel will normally share data, you also need to coordinate when operating on shared data; otherwise, one processor could start working on data before another is finished with it.
- This coordination is called synchronization, When sharing is supported with a single address space, there must be a separate mechanism for synchronization. One approach uses a lock for a shared variable.
- Only one processor at a time can acquire lock, and or processors interested in shared data must wait until original processor unlocks variable.



Classic organization of a shared memory multiprocessor

- OpenMP An API for shared memory multiprocessing in C, C++, or Fortran that runs on UNIX and Microsoft platforms. It includes compiler directives, a library, and runtime directives.
- A Simple Parallel Processing Program for a Shared Address Space Suppose we want to sum 64,000 numbers on a shared memory multiprocessor computer with uniform memory access time. Let's assume we have 64 processors.
- The first step is to ensure a balanced load per processor, so we split set of numbers into subsets of same size. We do not allocate subsets to a different memory space, since re is a single memory space for machine; we just give different starting addresses to each processor.

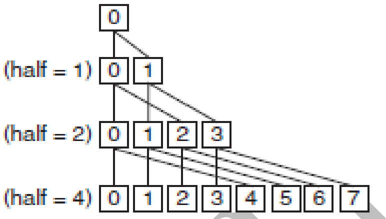
- Pn is number that identifies processor, between 0 and 63. All processors start program by running a loop that sums their subset of numbers:



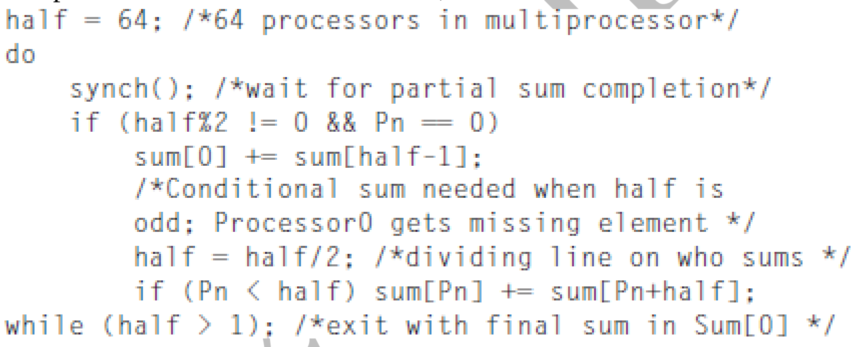
```
sum[Pn] = 0;
for (i = 1000*Pn; i < 1000*(Pn+1); i += 1)
  sum[Pn] += A[i]; /*sum the assigned areas*/
```

- The next step is to add se 64 partial sums.
- This step is called a reduction, where we divide to conquer.
- Half of processors add pairs of partial sums, and n a quarter add pairs of new partial sums,

and so on until we have single, final sum.



- Each processor to have its own version of loop counter variable i, so we must indicate that it is a private variable. Here is the code,



```
half = 64; /*64 processors in multiprocessor*/
do
    synch(); /*wait for partial sum completion*/
    if (half%2 != 0 && Pn == 0)
        sum[0] += sum[half-1];
        /*Conditional sum needed when half is
        odd; Processor0 gets missing element */
        half = half/2; /*dividing line on who sums */
        if (Pn < half) sum[Pn] += sum[Pn+half];
while (half > 1); /*exit with final sum in Sum[0] */
```

- Some writers repurposed acronym SMP to mean symmetric multiprocessor, to indicate that latency from processor to memory was about same for all processors

UNIT 5- MEMORY AND I/O SYSTEM

Memory Hierarchy – memory technologies – cache memory – measuring and improving cache performance – virtual memory, TLB‘s – Accessing I/O Devices – Interrupts – Direct Memory Access – Bus structure – Bus operation – Arbitration – Interface circuits – USB.

PART A

1 **Define memory access time. BTL1**

- The time that elapses between the initiation of an operation and completion of that

operation, for example, the time between the read and the MFC signals.

- This is referred to as memory access time.

2 **Define memory cycle time. BTL1**

- The minimum time delay required between the initiations of two successive memory

operations, for example, the time between two successive read operations.

3 **Define Static memories. BTL1**

Memories that consist of circuits capable of retaining the state as long as power is applied are known as static memories.

4 **What is locality of reference? What are its types? May 14 BTL1**

- Many instructions in localized area of the program are executed repeatedly during some

time period and the remainder of the program is accessed relatively infrequently.

- This is referred as locality of reference.
- Two types they are, Temporal & Spatial Locality

5 **Explain virtual memory technique. BTL2**

Techniques that automatically move program and data blocks into the physical memory, when they are required for execution are called virtual memory technique

6 **What are virtual and logical addresses? BTL1**

The binary addresses that the processor issues for either instruction or data are called

virtual or logical addresses.

7 **Define translation buffer. BTL1**

- Most commercial virtual memory systems incorporate a mechanism that can avoid the

bulk of the main memory access called for by the virtual to physical addresses translation buffer.

- This may be done with a cache memory called a translation buffer.

8 **What is optical memory? BTL1**

- Optical or light based techniques for data storage, such memories usually employ optical disk which resemble magnetic disk in that they store binary information in concentric tracks on

an electromechanically rotated disks.

- The information is read as or written optically, however with a laser replacing the read write arm of a magnetic disk drive. optical memory offer high storage capacities but their access rate is are generally less than those of magnetic disk

9 **What are static and dynamic memories? BTL1**

static memory are memories which require periodic no refreshing. dynamic memories are memories, which require periodic refreshing.

10 **What are the components of memory management unit? BTL1**

- A facility for dynamic storage relocation that maps logical memory references into physical memory addresses.
- A provision for sharing common programs stored in memory by different users .

11 **What are the multimedia applications which use caches? BTL2**

Some multimedia application areas where cache is extensively used are

- Multimedia entertainment
- Education
- Office systems
- Audio and video mail

12 **What do you mean associative mapping technique? BTL1**

- The tag of an address received from the CPU is compared to the tag bits of each block of

the cache to see

- If the desired block is present. this is called associative mapping technique.

13 **What is an i/o channel? BTL1**

An i/o channel is actually a special purpose processor, also called peripheral processor.the main processor initiates a transfer by passing the required information in the input output channel. the channel then takes over and controls the actual transfer of data.

14 **Why program controlled i/o is unsuitable for high-speed data transfer? BTL5**

- In program controlled i/o considerable overhead is incurred, because several program

instruction have to be executed for each data word transferred between the external devices and

main memory.

- Many high speed peripheral; devices have a synchronous modes of operation, that is data

transfer are controlled by a clock of fixed frequency, independent of the CPU.

15 **what is the function of i/o interface? BTL1 Dec-06/07 May-07/09**

The function is to coordinate the transfer of data between the CPU and external devices.

What is the necessity of an interface?

Handle data transfer between much slower peripherals & CPU or memory

Match signal levels of different I/O protocols with computer signal levels

	Provides necessary driving capabilities – sinking & sourcing currents
16	**What is the need to implement memory as a hierarchy May 15 BTL1** Ideally, computer memory should be fast, large and inexpensive. Unfortunately, it is impossible to meet all the three of these requirements using one type of memory.
17	**Name some of the IO devices. BTL1** • Video terminals • Video displays • Alphanumeric displays • Graphics displays • Flat panel displays • Printers • Plotters
18	**What is an interrupt?** An interrupt is an event that causes the execution of one program to be suspended and another program to be executed
19	**What is the difference between Serial interface & Parallel interface Dec15 BTL2** **Serial Interface** It transfer data one bit at a time Lower data transfer rate. Needs less number of wires to connect devices in the system Well suited for long distances, because fewer wires are used as compared to a parallel bus. **Parallel Interface** It can transmit more than one data bit at a time. Faster data transfer rate. Needs more number of wires to connect devices in the system. The interconnection penalty increases as distances increase.
20	**What is DMA? Or What is DMA operation? State its advantages or why we need DMA Dec 16/May 15/Dec 17 BTL1** A Special control unit may be provided to enable transfer a block of data directly between an external device and memory without contiguous intervention by the CPU. This approach is called DMA. The data transfer using such approach is called DMA operation. Two main Advantages of DMA operation are: The data transfer is very fast. Processor is not involved in the data transfer operation and hence it is free to execute other tasks.
21	**What is the use of DMA controller Dec15 BTL1** DMA is used to connect a high speed network to the computer bus. The DMA control handles the data transfer between high speed network & the computer system. It is also used to transfer data between processor & floppy disk with the help of Floppy disk controller
22	**What is meant by interleaved memory? May 13&17 BTL1** The memory interleaving is a technique to reduce memory access time by dividing memory into a number of memory modules and the addresses are arranged such that the successive words in the address space are placed in different modules. Most of the times CPU access consecutive memory locations. In such situations accesses will be to the different modules. Since these modules can be accessed in parallel, the average access time of fetching word from the main memory can be reduced
23	**What is meant by address mapping?** The virtually addressed memory with pages mapped to main memory. This process is called address

mapping or address translation

24 **Define hit rate/hit ratio Dec 15 BTL1**

The percentage of accesses where the processor finds the code or data word it needs in the ache memory is called the hit rate or hit ratio

25 **How DMA can improve I/O speed. May 15 BTL1**

DMA is a hardware controlled data transfer. It doesn’t spend testing I/O device status and executing a number of instructions for I/O data transfer.

In DMA transfer, data is transferred directly from the disk controller to the memory location without passing through the processor or the DMA controller

26 **What is the purpose of dirty/Modified bit in cache memory Dec 14 BTL1**

The data in the cache is called dirty data, if it is modified within cache but not modified in main memory. Whereas, dirty bit(modified bit) is a cache line condition(status)identifier, its purpose is to indicate whether contents of a particular cache line are different to what is stored in operating memory.

27 **How many total bits are required for a direct-mapped cache with 16kb of data and 4-word blocks, assuming a 32-bit address? Dec 17 BTL2**

Solution:
16kb=4k words=2^12 words
Block size of 4 words= 2^10 blocks
Each block has 4*32=128 bits of data + tag + valid bit
Tag + Valid bit= (32 – 10 – 2 - 2) +1=19
Total cache size= 2^10 (128 + 19) = 2^10 * 147

PART B

1 **Differentiate programmed I/O from memory mapped I/O. (13m) (Apr/May 2018) BTL4**

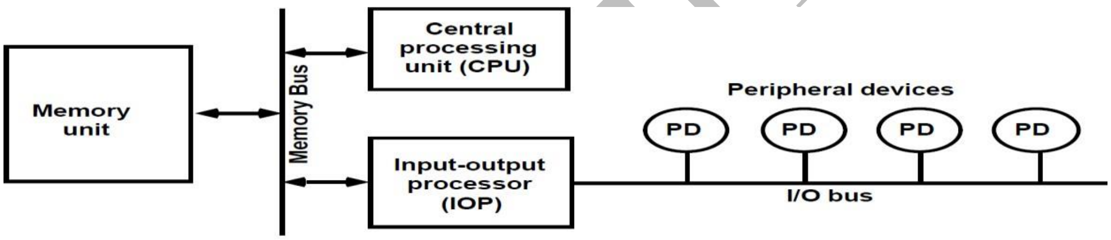
	Isolated-mapped I/O	**Memory-mapped I/O**
1.	Each port is treated as an independent unit.	Each port is treated as an independent unit.
2.	Separate address spaces for memory and input/output ports.	CPU’s memory address space is divided between memory and input/output ports.
3.	Usually, processor provides less address lines for accessing I/O. Therefore, less decoding is required.	Usually, processor provides more address lines for accessing memory. Therefore more decoding is required control signals.
4.	I/O control signals are used to control read and write operations.	Memory control signals are used to control read and write I/O operations.
5.	I/O address bus width is smaller than memory address bus width.	Memory address bus width is greater than I/O address bus width.
6.	Two instructions are necessary to transfer data between memory and port.	Single instruction can transfer data between memory and port.
7.	Data transfer is by means of instruction like MOVE.	Each port can be accessed by means of IN or OUT instructions.
8.	I/O bus shares only I/O address range.	Memory address bus shares entire address range.

2 **Explain in detail, the architecture of I/O Processors. (13m) BTL4**

- The I/O processor (IOP) has an ability to execute I/O instructions and it can have complete control over I/O operation.
- The I/O instructions are stored in main memory. When I/O transfer is required, the CPU initiates an I/O transfer by instructing the I/O channel to execute an I/O program stored in the main memory.
- The I/O program specifies the device or devices, the area of memory storage, priority and actions to be taken for certain error conditions.

Features and Functions of IOP

1. An IOP can fetch and execute its own instructions.
2. Instructions are specially designed for I/O processing.
3. In addition to data transfer, IOP can perform arithmetic and logic operations, branches, searching and translation.
4. IOP does all work involved in I/O transfer including device setup, programmed I/O, DMA operation.
5. IOP can transfer data from an 8-bit source to 16-bit destination and vice versa.
6. Communication between IOP and CPU is through memory based control blocks. CPU defines tasks in the control blocks to locate a program sequence, called a channel program.
7. IOP supports multiprocessing environment. IOP and CPU can do processing simultaneously. This distributed processing approach improves system performance and flexibility.

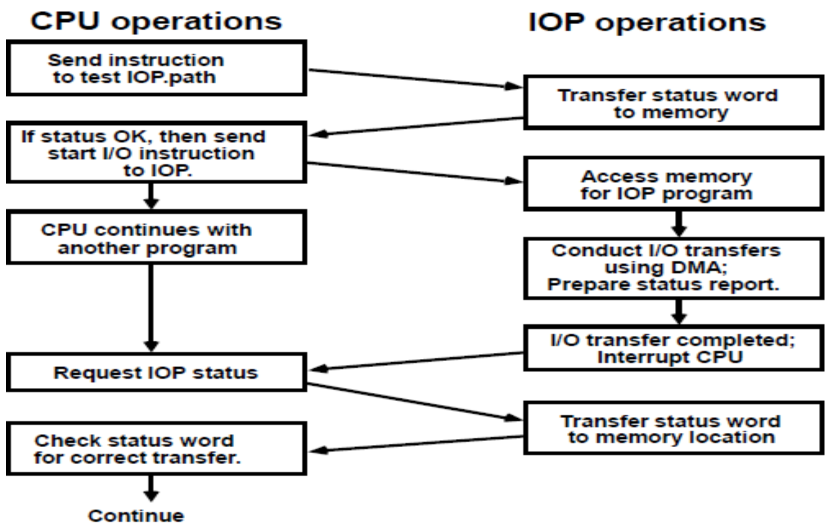


Block diagram of a computer with I/O processor

- The Figure shows the block diagram of computer system with an I/O processor.
- The CPU and I/O processor work independently and communicate with each other using centrally located memory and DMA.
- The CPU does the processing of needed in the solution of computational tasks and IOP does the data transfer between various peripheral devices and the memory unit.

CPU and IOP Communication

- The communication between CPU and IOP may be different for different processor and IOP configurations. However, in most of cases the memory based control blocks are used to store the information about the task to be performed.
- The processor uses these blocks to leave information in it for the other processor. The memory control block are linked, i.e., the address of the next memory based control blocks is available in the previous memory based control block.

	 CPU operations IOP operations Send instruction to test IOP.path Transfer status word to memory If status OK, then send start I/O instruction to IOP. Access memory for IOP program CPU continues with another program Conduct I/O transfers using DMA; Prepare status report. I/O transfer completed; Interrupt CPU Request IOP status Transfer status word to memory location Check status word for correct transfer. Continue **CPU and IOP communication** • The figure shows the flowchart of sequence of operations that are carried out during the CPU and IOP communication. The sequence of operations carried out during CPU and IOP communication are: 1. CPU checks the existence of I/O path by sending an instruction. 2. In response to this IOP puts the status word in the memory stating the condition f IOP and I/O device (Busy, ready, etc.) 3. CPU checks the status word and if all conditions are OK, it sends the instruction to start I/O transfer along with the memory address where the IOP program is stored. 4. After this CPU continues with another program. 5. IOP now conducts the I/O transfer using DMA and prepares status report. 6. On completion of I/O transfer, IOP sends an interrupt request to the CPU.The CPU responds to the interrupt by issuing an instruction to read the status from the IOP. The status indicates whether the transfer has been completed or is any errors occurred during the transfer.
3	**Compare & Design the mapping techniques & functions in involved in cache memory (13m) (Apr/May2018) BTL4&6** **Answer: U-4** Refer Notes, **Carl hamacher book Pageno:316** **Explanation(8m)** • Direct mapping • Associative mapping(Fully Associative) • Set- Associative mapping **Diagram(5m)**
4	**Explain about the mass storage. (13m) BTL4** **Answer: U-4** Refer notes, **Carl hamacher book Pageno:358** **Explanation(8m)** • **Magnetic** disk • **Floppy disk** • RAID Disk arrays • Magnetic tapes • Optical Disk

	Diagram(5m)
5	**Expain about Interuppt Handling / Write the sequence of operations carried out by a processor. When interrupted by a peripheral device connected to it. /Design & Explain a parallel priority interrupt hardware for a system with 8 interuupt sources. Dec 15/May 17 BTL4** **Answer:** Explanation (10m) Interrupt Driven I/O • Enabling & disabling interrupts • Vectored Interuppts • Interuppt Nesting • Interuppt Priority Recognition of interrupt & Response to interrupt Diagram (3m) • Response to an interrupt with the flowchart & diagram
6	**Explain about virtual memory & steps involved in Virtual Memory address translation BTL2** **Answer:** Explanation (10m) • Virtual memory • Concept of paging • Virtual to Physical Address Translation • Segment Translation • Page Translation Diagram (3m)
7	**Explain memory technologies in detail May17 BTL4** **Answer:** Explanation: (10m) RAM & ROM Technologies • Static RAM cell • Read operation • Write operation • CMOS Cell • Read operation • Write operation • DRAM • ROM, PROM, EPROM, EEPROM Diagram (3m)
8	**Explain Bus Arbitration techniques in DMA Dec 14/ May 17** **Answer:** Explanation (10m) Approaches to Bus Arbitration • Centralized bus arbitration ➢ Daisy Chaining ➢ Polling method ➢ Independent request • Distributed bus arbitration Diagram (3m)
9	**Describe about the i/p & o/p devices in detail with a neat diagram. (15m) BTL1**

	Answer: U-4 Refer notes, **Carl hamacher book Pageno:554-558** Explanation:10m Diagram:5m **I/P devices:** Keyboard, Mouse,… **O/P devices:** Printer, Plotter,…
	PART C
1	**Explain in detail, the concepts of virtual memory. (15m) (Apr/May 2018) BTL4** **Answer: U-4** Refer Notes, **Carl hamacher book Pageno:337** Explanation:10m Diagram:5m
2	**Explain in detail, the methods to improve cache performance. (15m) BTL4** **Answer: U-4** Refer Notes, **Carl hamacher book Pageno:329** Explanation:10m Diagram:5m
3	**Explain in detail, the cache memory and the accessing methods (15m) BTL4** **Answer: U-4** Refer Notes, **Carl hamacher book Pageno:314** Explanation:10m Diagram:5m
4	**Explain about DMA/ DMA Operations/ DMA Controller** **Answer:** **Explanation (10m)** • DMA Operation • DMA Block diagram • Cycle stealing mode(Single transfer mode) • Block transfer mode • Demand transfer mode
5	(i) Consider web browsing application assuming both client & server are involved in the process web browsing application, where can caches be placed to speed up the process design a memory hierarchy for the system show the typical size & the latency at various levels of the hierarchy. What is the relationship between cache size & its access latency? What are the units of data transfers between hierarchies? What is the relationship between the data location, data size & transfer latency? Answer: a) Assuming both client & server are involved in the process of web browsing application, caches can be placed on both sides-Web browser & server b) Memory hierarchy for the system is as follows: 1. Browser cache, size=fraction of client computer disk, latency= local disk latency 2. Proxy cache, size-proxy disk, Latency= LAN + proxy disk latencies 3. Server-side cache= fraction of server disk, Latency= WAN + server disk 4. Server storage, size= server storage, latency= WAN + server storage. Latency is not directly related to cache size. (C) The units of data transfers between hierarchies are pages. (d) Latency grows with page size as well as distance (ii) The following sequence of instructions are executed in the basic 5-stage pipelined processor I1: lw \$1, 40(\$6) I2: add \$6, \$2, \$2 I3: sw \$6, 50(\$1)

Indicate dependencies & their type, Assuming there is no forwarding in pipelined processor. Indicate hazards & add NOP instructions to eliminate them.

Answer:

(a) I1: RAW Dependency on **$1 from I1 to I3**
I2: RAW Dependency on **$6 from I2 to I3**
I3: RAW Dependency on **$6 from I1 to I2 to I3**

(b) If register read happens in the second half of the clock & the register write happens in the first half. The code that eliminates these hazards by inserting nop instruction is:

```
I1: lw $1, 40($6)
I2: add $6, $2, $2
nop; delay I3 to avoid RAW hazard on $1 from I1
I3: sw $6, 50($1)
```

6 Assume the miss rate of an instruction cache is 2% & miss rate of data cache is 4% If a processor has a CPI of 2 without any memory stalls & miss penalty 100 cycles for all misses, determine how much faster a processor would run with a perfect cache that never missed. Assume the frequency of all loads & stores is 36%

Solution: The number of memory miss cycles for instructions in terms of the instruction count(I) is
Instruction miss cycle=I*2% * 100 = 2.00*I
As the frequency of all loads & stores is 36%, we can find the number of memory miss cycles for data refernces:
Data miss cycles= I*36%*4%*100=1.44 *I
The total number of memory-stall cycles is 2.00 I + 1.44 I = 3.44 I. This is move then 3 cycles of memory stall per instruction. Accordingly, the total CPI including memory stalls is 2+3.44 = 5.44. Since there is no change in instruction count or clock rate, the ratio of the CPU execution times is
CPU time with stalls/CPU time with perfect cache = I*CPI_{stall} * Clock cycle/ I*$CPI_{perfect}$ * Clock cycle
= CPI_{stall} /$CPI_{perfect}$
= 5.44/2
The performance with the perfect cache is better by 2.72

Hit time is the time to access the upper level of the memory hierarchy, which includes the time needed to determine whether the access is a hit or miss.

If a larger cache is used, there is increase in the access time i.e, the hit time. But at a certain point, the increase in hit time due to larger cache results into decrease in miss rate i.e, the hit rate increases and so the cache performance also increases.

AMAT(Average Memory Access Time) is the average time to access memory considering both hits & misses & the frequency of different accesses

AMAT= Time for a hit + Miss rate * Miss penalty