Solution Paper – III
High Performance Computer Architecture

Q. 1. What are benchmarks? Discuss various kinds Bench marks. Give their significance in computer architecture.

Ans. Bench mark: The selection of a computer system to meet an end user requirement needs a formal procedure as specified ahead: The feasibility study of a new installation or the possibility of n/ o grading an existing system is analyzed to determine cost effectiveness in terms of requirements of end-user. The report based on increased productivity of skilled staff, reduced product development times, a more reliable product etc. is prepared. Then report of the feasibility study to be submitted to senior measurement to segment funds to implement the proposed system.
Bench mark programs used to test particular performance factors like whetstone for floating point numerical, ohrystona for integer numerical and string. End user bench mark programs to evaluate performance factors that are important in particular application area like unpack benchmark.

There are following kinds of benchmarks

(i) Whetstone benchmark

(ii) Dhrystone benchmark

(iii) LINPAC Benchmark

(iv) SPEC Benchmark.

(i) Whetstone benchmark: Whetstone benchmarks are a synthetic benchmark used to test compiler optimication and bating point performance. The whetstone benchmark is used to measure computer performance and is designed to stimulate floating point numerical applications.

(a) This benchmark is having a large percentage of floating point data and instruction.

(b) This benchmark spends a high percentage it execution time nearly 50% in mathematical library functions.

(c) Whetstone benchmarks contains a number of very tight loops and use of instruction Caches will enhance performance effectively.

(ii) Dhrystone benchmark: The Dhrystone benchmark is used to test performance factors important in non-numeric system programming like operating systems, compilers, word processor etc.

(a) This benchmark do not contain floating-point operations.

(b) This benchmark spent a considerable percentage of time in string functions making the test very dependent upon the way such operations are performed. Example in line lode, routines written in assembly language making it brone to manufacturers fine-tuning of critics routines.

(iii) LINPAC benchmark: The unpack benchmark is derived from a real application that originated as a collection of linear algebra subroutines implemented in FORTRAN.

(iv) SPEC benchmark: The Standard Performance Evaluation Corporation (SPEC) is non-profit corporation formed to establish, maintain and endorse a standardized set of relevant bench marks that can be applied to the newest generation of High-performance Computers.

Q 2.What is instruction cycles? Explain.

Ans. A program in computer consists of sequence of instructions. Executing these instructions runs the program in computer. Moreover each instruction is further divided into sequences of phases The concept of execution of an instruction through different phases is called instruction cycle. The instruction is divided into sub phases as specified ahead.

1. First of all an instruction is fetched (accessed) from memory.
2. Then decode that instruction.,
3. Decisions is made for memory or register of I/O reference instruction, in case of memory indirect address, read the effective address from the memory.
4. Finally execute the instruction.

1**. Fetch phase** : The sequence counter (SC) is initialized to 0. The program counter (PC) contains the address first instruction of a program under execution. The address of first instruction on front PC is loaded into address register (AR) during first clock esTQ). Then instruction from memory location given by address (AR) is loaded into the instruction.register (IR) and program. coi is increased to address of next instruction in second clock (T1). i. nese micro-operations using register transfer language is shown as ahead.

$T_0 : AR \leftarrow PC.$

$T_1 : IR \leftarrow M[AR], PC \leftarrow PC + 1.$

2. **Decode phase:** All bits of the instruction under execution stored in JR are analysed and decoded as shown ahead in third clock cycle (T2).

T2: D0, D1, D2 ... D1 -i5ecode IR (12 - 14)

I - JR (15), AR - JR Y'

3**. Decision phase** : First off all the [JR (12 — 14)] bit of instruction are decoded.
(i) If decoder output D7 (111) is 1, that means instruction must be register reference or inp peference....Then the last bit of instruction registerjj) stored itj. flip-flop I is decoded If JR (15) = 0 that means thc instruction is register enc1nfrtion.JfIR (15) 1, that theans the Instruction are used far registers or I/O reference instruction code.

$T_2 : D_7 \leftarrow$ Decode IR (12 – 14), $O \leftarrow$ IR (15) (Register reference)

$T_2 : D_7 \leftarrow$ Decode IR (12 – 14)

$I \leftarrow$ IR (15) (I/O reference).

(ii) If decoder output D7 (III) is not ±, that means JR (12—14) are decoded in between D0 to D6. That implies the instruction is memory reference instruction. Then last bit of instruction register JR (15) stored in flip-flop J is decoded. Jt JR (15) = 0, that mean memory reference instruction is direction address instruction If JR (15) = 1, that means memory reference instruction is indirect address instruction. The first twelve bits of instruction are used as address of memory location.

T_2 : D_0, D_1, D_2 .... D0 ← Decode IR (12 – 14)

(Direct address memory regference instruciton)

T_2 : D_0, D_1, D_2 ... D_6 ← Decode IR (15)

(Indirect Address Memory Reference Instruction).

4. **Execution Phase** : Then the instruction is executed as memory or register or input output reference instruction in fourth clock cycle (T3).

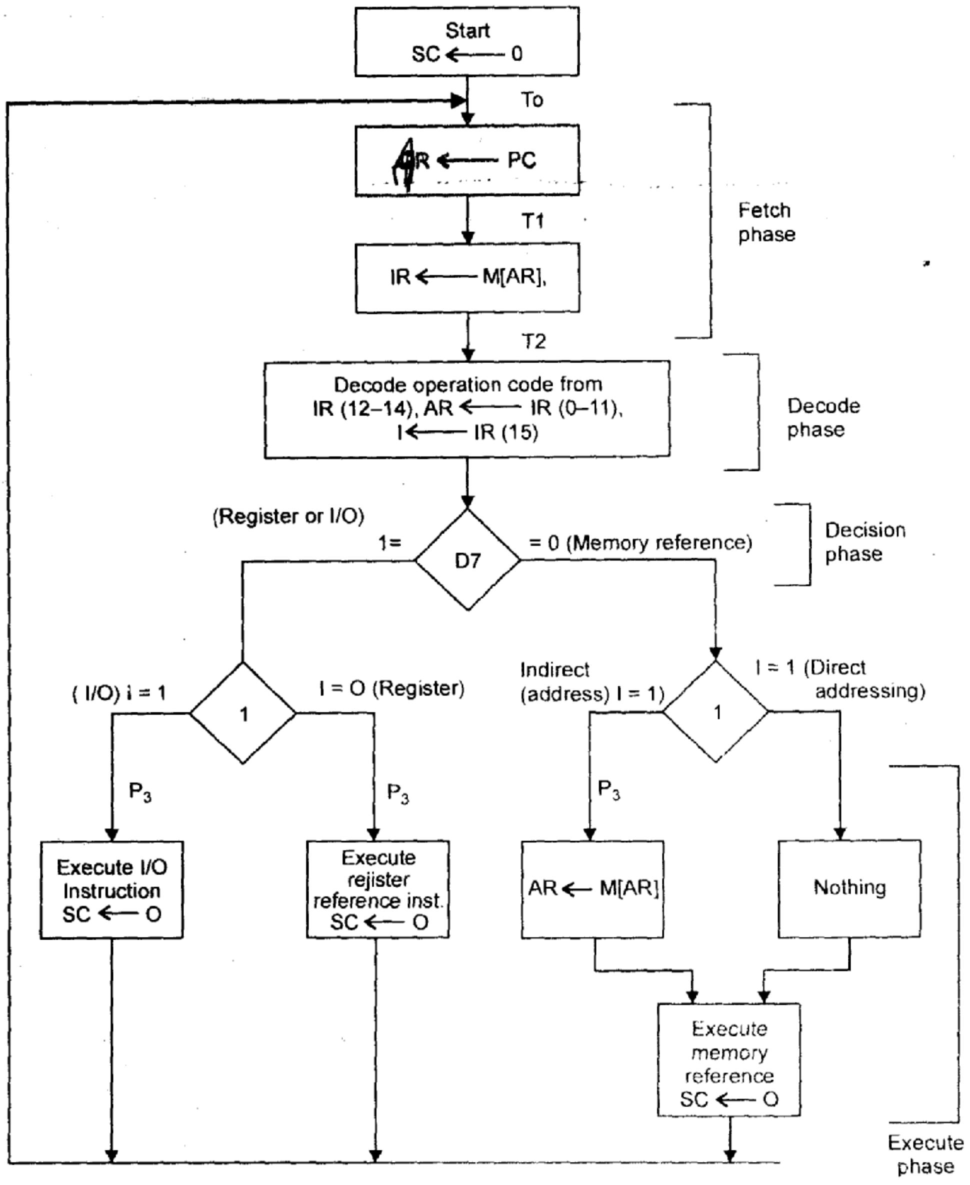
D_7' I' T_3 :Execute direct address memory reference instruction.

D_7' I' T_3 : Calculate the effective addresss (AR ← M[AR]) and execute indirect address nemory reference instruction.

D_7' I' T_3 :Execute register reference instruction.

D_7' I T_3 : Execute an input output reference instruction.

Flow chart of instruction cycle



Start
SC ← 0
To
AR ← PC
T1
IR ← M[AR],
T2
Fetch phase
Decode operation code from IR (12–14), AR ← IR (0–11), I ← IR (15)
Decode phase
(Register or I/O)
1=
D7
= 0 (Memory reference)
Decision phase
(I/O) i = 1
1
I = O (Register)
Indirect (address) I = 1)
1
I = 1 (Direct addressing)
P3
P3
P3
Execute I/O Instruction SC ← O
Execute rejister reference inst. SC ← O
AR ← M[AR]
Nothing
Execute memory reference SC ← O
Execute phase

Q. 3. Write a note on following:
(i) Pentium Processor - (ii) Server System

Ans. (i) **Pentium processor:** Pentium processor with super scalar architecture came as modification of 80486 and 8086. It is based on CISC and uses two pipelines for integer processor so that two instructions are processed simultaneously one pipeline will have same condition then another is compared with hardware 80486 processor had only adder in one chip floating point unit. One the other side Pentium processor is having adder, multiplier and divide in on chip floating point unit. That means Pentium processor can do the multiplication and division fastly. The separate data and code cache of 8KB exits on chip. Dual independent bus (DIB) architecture divides the bus as front side and backside bus. Backside

Bus transfer the data from L2 Cache to CPU and vice-versa. Front side bus is used to transfer the data from CPU to main memory and to other components of system.

Pentium processor user write back policy for cache data, while 80486 uses write through policy for cache data. The detail other common types of processor are AMD and Cyrix although these two types of processor are less powerful as compared to Pentium Processor.

(ii) **Server System :** System is formed as server or client depending upon the software used in that machine suppose window 2003 server operating system is installed on machine, that machine will be termed as sever. If on the same machine Window 95 is installer that machine is termed as client. Although server machine uses specialized hardware meant for faster processing server provides the service to other machine called client attached to server. Different types of servers are Network server, web server, database server, backup server. Sever system is having powerful computing power, high performance and higher clock speed. These systems are having good fault tolerance capability using disk mirroring, disk stripping and RAID concepts. These systems have back up power supply with hot swap. IBM and SUN servers are providing the different server of server for different use.

Q. 4. Compare and contrast super pipelined machine and super scalar machines.

Ans. **Super pipelined machine**: Pipelining is the concept overlapping of multiple instruction during execution time. Super of pipelining splits one task into multiple subtasks. These subtasks of two or more different tasks are executed parallel by different hardware units. It overlaps the multiple instructions in execution. The instruction goes through the four stages during execution phase.

1. Fetch an instruction from memory (FI).
2. Decode the instruction (DI).
3. Calculate the effective address (EA).
4. Execute the instruction (LI).

		1	2	3	4	5	6	7	8
1		FI	DI	EA	EI				
2			FI	DI	EA	EI			
3				FI	DI	EA	EI		
4					FI	DI	EA	EI	
5						FI	DI	EA	EI

Fig. Space time diagram

Super scalar processor/Machine: The scalar machine executes one instruction one set of operands at a time. The super scalar architecture allows on the execution of multiple instruction, at the same time in different pipeline. Here multiple processing elements are used for different instruction at the same time. Pipeline is also implemented in each processing elements. The instruction fetching units fetch multiple instructions at a time from cache. The instruction decoding units check the independence of those instructions at a time from cache. There should be multiple execution units so that multiple instructions can be executed at the same time. The slowest stage among fetch, decode and execute will determine the overall performance of system. Ideally these three stages should be equal fast.

Q. 5. What are reasons of pipeline conflicts in pipelined processor? How are they resolved?

Ans. There are following reasons which create the conflicts in pipelined processor and way by which it is resolved:

1. **Resource conflicts:** It caused by access to memory by two segments at the same time. Most of these conflicts can be resolved by using separate instruction and data memories.

2. **Data dependency conflict**: It arises when an instruction depends on the result of a previous instruction, but this result is not yet available.

3**. Branch difficulties**: It arises from branch and other instructions that change the value of PC.

A difficulty that may cause a degradation of performance in an instruction in pipeline is due to possible collision of data or address. A collision occurs when an instruction cannot proceed because previous instructions did not complete certain operations. A data depending occurs when an instruction needs data that are not yet available. For example, an instruction in the Fetch operand segment may need to fetch an operand that is being generated at the same time by the previous instruction in segment EX (Execute). Therefore, the second instruction must wait for data to become available by the first instruction. Similarly, an address dependency needed by address mode is not available.

For example, an instruction with register indirect mode cannot proceed to fetch the operand if the previous instruction is loading the address into the register. Therefore, the operand access to memory must be delayed until the required address is available. Pipelined computers deal with such conflicts between data dependencies in a variety of ways. The most straight forward method is to insert Hardware inter locks. An interlock is a circuit that detects instructions whose source operands are destinations of instructions farther up in pipeline. This approach maintains the program sequences by using hardware to insert the required delays.

Another technique called operand forwarding uses special hardware to detect a conflict and then avoid it by routing the data through special paths between pipeline segments.'tor example, instead of transforming an ALU result into a destination register, hardware checks the destination operand and if it is needed as a source in the next instruction, it passes the result directly into ALU input, by passing the register file. This method requires additional hardware paths through multiplexers as well as the circuit that detects the conflict.

A procedure employed in some computers is to give the responsibility for solving data conflicts problems to the compiler that translates the high-level programming language into a machine language program.

Q 6.What is meant by super scalar processor? Explain the concept of pipelining in uperscalar processor?

Ans. The scalar processor executes one instruction on one set of operands at a time. The super scalar architecture allows the execution of multiple instructions at the same time in different pipelines. Here multiple processing elements are used for different instruction at same time. Pipeline is also implemented in each processing elements. The instruction fetching units fetch microinstruction at a time from cache. The instruction decoding unit checks the independence of these instructions so that they can be executed in parallel. There should be multiple execution units so the multiple instructions .can be executed at the same time. The slowest stage among fetch, decode and execute will determine the performance of the system. Ideally these three stages should be equally fast. Practically execution stage in slowest and drastically affect the performance of system.

Pipeline overlaps the multiple instructions in execution. The instruction goes through the four stages during the execution phase.

1. Fetch an instruction from memory (FI)

2. Decode the instruction (DI)

3. Calculate the effective address (EA)

4. Execute the Instruction (El).

	1	2	3	4	5	6	7	8
1	FI	DI	EA	EI				
2		FI	DI	EA	EI			
3			FI	DI	EA	EI		
4				FI	DI	EA	EI	
5					FI	DI	EA	EI

In space-time diagram above, five instructions are executed using instruction pipeline.

These five instructions are executed in eight clock cycles. Each instruction had been through four stages. Although the various stages may not be of equal duration in each instruction. The result in waiting at certain stages.

Q. 7. In a two-level memory hierarchy, if the cache has an access time of ns and main memory has an access time of 60ns, what is the hit rate in cache required to give an average access time of 10ns?

Ans. Using the formula, the average access time = (THjt X 1Hit) + (T5x ‘miss)

The average access time 10ns = (8ns x hit rate) + 60 ns x — (hit rate), (The hit and miss rates at a given level should sum to 100 percent). Solving for hit rate, we get required hit rate of 96.2%.

Q. 8. Explain instruction set of SPARC with Descriptions.
Ans. The condition code register on the SPARC has four bits: Z (Zero), N (Negative), C (Carry), and V (Overflow). The standard arithmetic operations (e.g., addition and subtraction) do not update the bits in the condition code register. Instead, there are special operations that update the condition code register. The names for these operations have a suffix of ``cc'' to indicate that they update the bits in the condition code register.

In most cases, the effect that an operation has on the condition codes is just what you would expect. Most of these operations set the Z bit when the result of the operation is zero, and clear this bit when the result is nonzero. Similarly, most of these operations set the N bit when the result of the operation is negative, and clear this bit when the result is nonnegative. The V bit is usually set when the (signed integer) result of the operation cannot be stored in 32 bits, and cleared when the result can be stored in 32 bits. Finally, the C bit is set when the operation generates a carry out of the most significant bit, and cleared otherwise.
In most contexts, you will be most interested in the N and Z bits of the condition code register and we will emphasize these bits in the remainder of this lab.

INSTRUCTIONS

A) Data Movement Instructions

Instruction	Description	Example
ldsb address, rd	Load signed byte from addr	ldsb [%r1], %r2
ld address, rd	Load signed word from addr	ld [%r1], %r2
ldub address, rd	Load unsigned byte from addr	ldub [%r1], %r2
stb rs, address	Stores byte to addr	stb %r1, [%r2]
sth rs, address	Stores halfword to addr	sth %r1, [%r2]
st rs, address	Stores word to addr	st %r1, [%r2]
sethi const22, rd	Sets upper 22 bits of rd with const	sethi 123,%r1

B) Arithmetic Instructions

Instruction	Description	Example
add/addcc rs1, rs2/const, rd	reg[rd] = reg[rs1] + reg[rs2]	add %r1, %r2, %r3
sub/subcc rs1, rs2/const, rd	reg[rd] = reg[rs1] - reg[rs2]	sub %r1, %r2, %r3
addx/addxcc rs1, rs2/const, rd	reg[rd] = reg[rs1] + reg[rs2] (with carry)	addx %r1, %r2, %r3
subx/subxcc rs1, rs2/const, rd	reg[rd] = reg[rs1] - reg[rs2] (with borrow)	subx %r1, %r2, %r3
cmp %r1,%r2	Sets appropriate condition codes	cmp %r1,%r2

C) Logical and Shift Instructions

Instruction	Description	Example
and rs1,rs2/const,rd	reg[rd]=reg[rs1] AND reg[rs2]	and %r1,%r2,%r3
or rs1,rs2/const,rd	reg[rd]=reg[rs1] OR reg[rs2]	or %r1,%r2,%r3

orn rs1,rs2/const,rd	reg[rd]=reg[rs1] NOR reg[rs2]	orn %r1,%r2,%r3

D) Branch and Control Transfer Instructions

Instruction	Description	Example
ba target	branch always	ba loop1
bne target	branch not equal	bne loop1
be target	branch equal	be loop1
bg target	branch greater	bg loop1
ble target	branch less than or equal	ble loop1
bge target	branch greater than or equal	bge loop1
bl target	branch less than	bl loop1
bgu target	branch greater (unsigned)	bgu loop1
bleu target	branch less or equal (unsigned)	bleu loop1

Note: Like most RISC machines, the SPARC uses a branch delay slot. By default, the instruction following a branch instruction is executed whenever the branch instruction is executed.

E) Procedure Instructions

Instruction	Description	Example
call target	Jump to call-address and save PC into %r15	call proc
jmpl rs , rd	Jumps to [rs] and saves PC to rd	jmpl %r1,%r2
save rs1,rs2,rd save rs1,const,rd	Provide new register window . rd = rs1 + rs2 (For stack pointer updation)	save %sp,-16,%sp
restore rs1,rs2,rd restore rs1,const,rd	Restore old register window. rd = rs1 + rs2 (For stack pointer updation)	restore %sp,16,%sp

Q. 9. What is the difference between isolated mapped Input Output and memory mapped Input Output. What are the advantages and disadvantages of each?

Ans. Isolated I/O: In isolated mapped I/O transfer, there will be common address and data bus for main memory and I/O devices. The distinction memory transfer i l7transfer is made through control lines. There will be separate control signals for main memory and I/O device. Those signals are memory read, memory write, I/O read and I/O write. This is an isolated I/O method of communication using a common bus. When CPU fetches and decodes the operation code of input or output instruction, the address associated with instruction is placed on address bus. If that address is meant for I/O devices then 1/0 read or I/O write control signal will he enabled depending upon whether we want to read or write the data from I/O devices. If that address is meant for main memory then memory read or memory write signals will be enabled depending upon whether we want to read or write the data to main memory. Memory Mapped I/O: In memory-mapped I/O, certain address locations are not used by memory and I/O devices use these address. Example : It address from 0 to 14 are not used by main memory. Then these addresses can be assigned as the address of I/O devices. That means with above example we can connect with 15 I/O devices to system having addresses from 0 to 14. So we an have single set of address, data and control buses. It the address on address bus belongs to main memory. This will reduce the available address space for main memory but as most modern system are having large main memory so that is not normally problem. Memory mapped I/O treats I/O parts as memory locations programmer must ensure that a memory-mapped address used by I/O device is used as a regular memory address. There are following main point of difference between isolated mapped I/O and memory mapped I/O.

Isolated Mapped I/O	Memory Mapped I/O
1. Isolated I/O method is for assinging addresses in a common bus.	The assigned addresses for interface cannot be used for memory words, which reduces the memory range available.
2. In isolated I/O configuration, the CPU has distinct input and output instruction and each associated with address of interface resister.	Memory-mapped I/O use memory type instructions to access I/O data.
3. The address locations are used by I/O devices than it is for I/O read or I/O write control signal will enable.	Some address locations are used by memory and I/O device can use these addresss.
4. In this, there is separate control signal for main memory and I/O devices.	There is common control signal through memory mapped I/O for memory and I/O devices.

Advantage/disadvantage

The Advantage is that the load and store instructions used for reading and writing from memory can be used to input and output data from I/O registers. In a typical computer, there are more memory reference instructions than I/O instructions with memory-mapped I/O all instruction that refer to memory are also available for I/O.

Q. 10. Write and explain all classes of interrupts.

Ans. There are two main classes of interrupts explained below:

1. Maskable interrupts.
2. Non-maskable interrupts.

1. **Maskable interrupts:** The commonly used interrupts by number are called maskable interrupts The processor can ask o temporarily ignore such interrupts These interrupts are temporarily 1gnred such that processor can finish the task under execution. The processor inhibits (block) these types of interrupts by use of special interrupt mask bit. This mask bit is part of the condition code register or a

special interrupt request input, it is ignored else processor services the interrupts when processor is free, processor will serve these types of interrupts.

2. **Non-Maskable Interrupts (NMI):** Some interrupts cannot be masked out or ignored by the processor. These are referred to as non-maskable interrupts. These are associated with high priority tasks that cannot be ignored. Example system bus faults.

The computer has a non-maskable interrupts (NMI) that can be used for serious conditions that demand the processor's attentions immediately. The NMI cannot be ignored by the system unless it is shut off specifically. In general most processors support the non-maskable interrupt (NMI). This interrupt has absolute priority. When it occurs the processor will finish the current memory cycle and then branch to a special routine written to handle the interrupt request. When a NMI signal is received the processor immediately stops whenever it as doing and attends to it. That can lead to problem if these type of interrupts are used improperly. The NMI signal is used only for critical problem situation like Hardware errors.