CELEBRATING 25 YEARS CMRIT CMR INSTITUTE OF TECHNOLOGY, BENGALURU. ACCREDITED WITH A+ GRADE BY NAAC

Internal Assessment Test 1

September 2019

Sub:	Advanced Computer Architecture				Sub Code	15CS72	Branch:	CSE
Date:	23/09/2019	Duration:	90 mins	Max Marks:	50	Sem / Sec:	VII A,B,C	OBE

	Answer any FIVE FULL Questions	MARK S	CO	RBT
1 (a)	**Explain Flynn's Classification of Computer architecture along with neat diagram.**	08	CO1	L2

Michael Flynn introduced a classification of various computer architectures based on notions of instruction and data streams.

Single Instruction Stream Single Data Stream(SISD)

- It is uniprocessor system
- Single instruction is executed by CPU in one clock cycle.
- Instructions are executed sequentially
- Workstations of DEC, MP & IBM, IBM 701, IBM 1620, IBM 7090 etc.

Single Instruction Stream Multiple Data stream(SIMD)

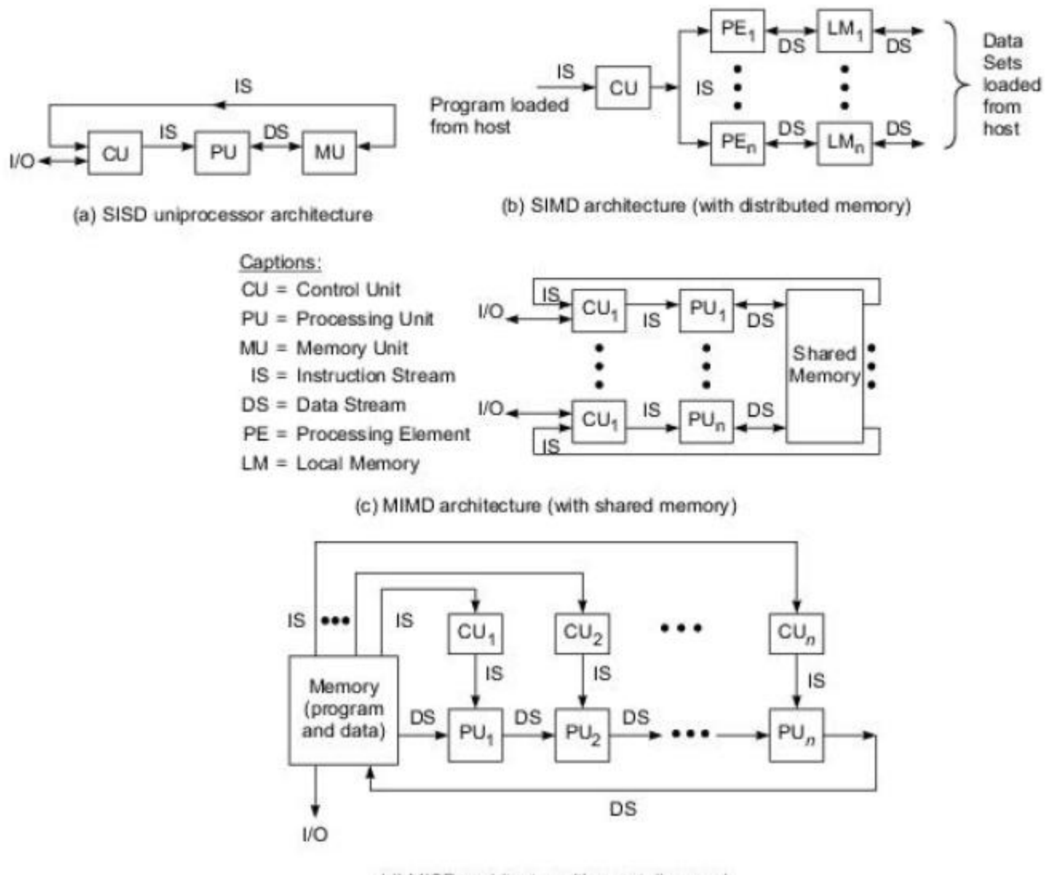
- A single instruction is executed by multiple processing elements or processors. Each processing element operates on different data.
- Data level parallelism is achieved.
- Example: Vector supercomputer in early 1970 like CDC star -100, Connection Machine CM2, Maspar MP-1, IBM 9000, Cray C90, Fujitsu VP.

Multiple Instruction Stream Single Data Stream(MISD)

- The same data stream flows through a linear array of processor executing different instruction streams. This architecture is known as systolic arrays for pipelined execution of specific instructions.
- Few actual examples of this class of parallel computer have ever existed. One is the experimental Carnegie Mellon C.MPP computer (1971).
- Least popular model to be applied in commercial machines.

Multiple Instruction Stream and Multiple Data Stream(MIMD)

- Most popular computer model
- Every processor may be executing different instruction stream and every processor uses different data stream.
- They are also called as parallel computers.
- Example: IBM 370/168MP, Univac 1100/80
- Parallel computers operate in MIMD mode.
- There are two types of MIMD i.e. shared memory multiprocessor and distributed memory multicomputer.
- In shared memory multiprocessor system all processors have common shared

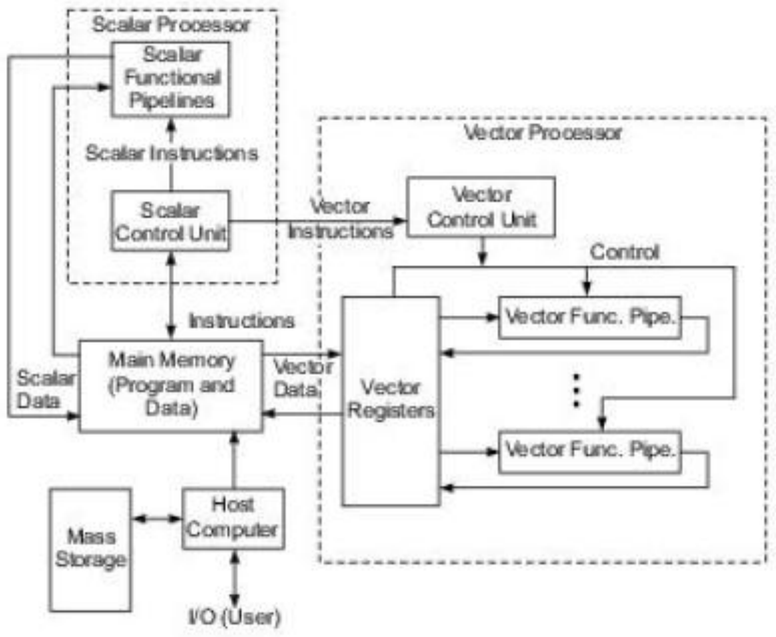
	memory and can communicate through shared variables. • In distributed multicomputer system each computer node has local memory unshared with other nodes. Inter-processor communication is done though message passing among the nodes.  IS I/O CU PU MU DS Program loaded from host PE1 LM1 PEn LMn Data Sets loaded from host Captions: CU = Control Unit PU = Processing Unit MU = Memory Unit IS = Instruction Stream DS = Data Stream PE = Processing Element LM = Local Memory CU1 PU1 CUn PUn Shared Memory CU2 PU2 Memory (program and data) (a) SISD uniprocessor architecture (b) SIMD architecture (with distributed memory) (c) MIMD architecture (with shared memory) (d) MISD architecture (the systolic array)			
1. (b)	**Describe the 5-tuple operational model of SIMD supercomputers.** The operational model of SIMD machine is specified by a 5-tuple M=(N,C,I,M,R) • N is the number of processing elements (PEs) in the machine. For example, the Illiac IV had 64 PEs and the Connection Machine CM-2 had 65,536 PEs. • C is the set of instructions directly executed by the control unit (CU). • I is the set of instructions broadcast by the CU to all PEs for parallel execution. • M is the set of masking schemes, where each mask partitions the set of PEs into enabled and disabled subsets. • R specifies the data routing schemes to be followed during inter PE communication.	(02)	CO4	L2

2 (a)	**Explain the architecture of Vector Supercomputer with a neat diagram.**	(08)	CO1	L3

- The program and data are loaded into main memory from the host computer.
- All instructions are first decoded by the scalar control unit. If the decoded instruction is a scalar operation it will be directly executed by the scalar processor using the scalar functional pipelines.
- If the instruction is decoded as vector operation, it will be sent to vector control unit. The vector control unit manages the flow of vector data between vector functional units and main memory.
- There are multiple vector functional units which are pipelined. Data is forwarded from one vector functional unit to another i.e. called vector chaining.

There are two types of vector processor

- Register Register vector processor
- Memory Memory vector processor

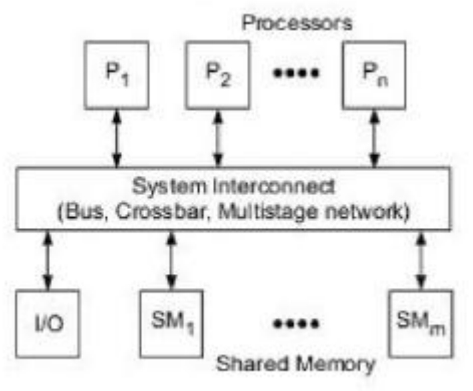


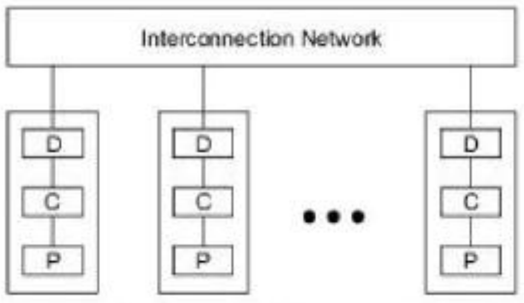
<u>Register Register Vector Processor</u>

- Vector registers are used to hold the vector operands, intermediate and final vector results.
- The vector functional pipelines retrieve operands from and put results into the vector registers. All vector registers are programmable for user instructions.
- The length of each vector register is usually fixed, say, sixty-four 64-bit component registers in a vector register in a Cray Series supercomputer.

<u>Memory to Memory Vector Processor</u>

- Vector operands and results are directly retrieved from and stored into the main memory in super words, say, 512 bits as in Cyber 205.

2 (b)	**Describe $AT2$ model for VLSI.** The parallel computers use VLSI chips for fabricating processor arrays, memory arrays etc. <u>AT^2 Model</u> Let A be the chip area. The latency T is time required from when the inputs are applied until all outputs are produced for a single problem instance. Let s be the size of the problem. Then there exists a lower bound f(s) such that $A * T^2$ >= O (f(s)). The chip is represented by the base area in the two horizontal dimensions. The vertical dimension corresponds to time. Therefore, the three-dimensional solid represents the history of the computation performed by the chip.	(02)	CO1	L2
3 (a)	**Explain UMA Model and COMA Model for shared memory multiprocessor systems with neat diagram.** <u>UMA Model(Uniform Memory Access)</u> • Here the physical memory is uniformly shared by all the processors. • All processors have equal access time to all memory words, which is why it is called uniform memory access. Refer the diagram given below.  Processors P$_1$ P$_2$ •••• P$_n$ System Interconnect (Bus, Crossbar, Multistage network) I/O SM$_1$ •••• SM$_m$ Shared Memory The UMA multiprocessor model • The system interconnect is through bus, crossbar or multistage network. When all processors have equal access to all peripheral devices, the system is called a symmetric multiprocessor. In this case, all the processors are equally capable of running the executive programs, such as OS kernel and I/O service routines. • In asymmetric multiprocessor only master processor can execute the operating system and handle I/O. The remaining processors have no I/O capability and thus are called Attached Processors. Attached processors execute user codes under the supervision of the master processor. <u>COMA(Cache Only Memory Architecture) Model</u> • Here each processor has a cache memory. • There is no memory hierarchy at each processor node. All the caches form global address space. • Remote access to any cache is assisted by distributed cache directory. • Whenever data is accessed in remote cache it gets migrated to where it will be	(06)	CO1	L3

	used. This reduces the number of redundant copies and allows a more efficient use of memory resources. • Examples: Kendall Square Research's KSR-1 machine.  The COMA model of a multiprocessor (P: Processor, C: Cache, D: Directory; e.g. the KSR-1)			
3 (b)	**Describe the Bernstein's conditions of Parallelism.** Bernstein's Conditions • There are certain conditions when two processes are executed in parallel. I_i (Input set) is the set of input variables for process P_i. and O_i (Output set) consists of all output variables generated after the execution of process P_i. Now, consider two processes P_1 and P_2 with their input sets I_1 and I_2 and output sets O_1 and O_2, respectively. These two processes can execute in parallel and are denoted as P1|| P2 if they are independent and follow these three Bernstein's conditions as given below • $I_1 \cap O_2 = \phi$ • $I_2 \cap O_1 = \phi$ • $O_1 \cap O_2 = \phi$ • Bernstein's conditions simply imply that two processes can execute in parallel if they are flow-independent, anti-independent, and output-independent. • In general, a set of processes, $P_1, P_2, \ldots, P_K$ can execute in parallel if Bernstein's conditions are satisfied on a pair wise basis; that is, $P_1 \parallel P_2 \parallel P_3 \parallel P_4 \parallel \ldots \parallel P_K$ if and only if $P_i \parallel P_j$ for all $i \neq j$	(04)	CO4	L2
4	**Explain different types of Dependences in program. Analyze the dependences for following code segment and draw dependence graph and assume there is only one functional unit for Load and Store. Note M (10) contains value 64.** **S1: Load R1,1024** **S2: Load R2,M(10)** **S3: Add R1,R2** **S4: Store M(1024),R1** **S5: Store M((R2)),1024** Data Dependence: There are five types of Data Dependence as shown below • Flow Dependence: A statement S2 is flow dependent on S1 if at least one output of S1 feeds in as input to S2. Flow Dependence is denoted as S1→ S2 • Anti Dependence: Statement S2 is anti-dependent on statement S1 if S2 follows S1 in program order and if the output of S2 overlaps the input to S1. It is denoted	(10)	CO1	L4

as follows

$S1 \nrightarrow S2$

- Output Dependence: Two statements are output-dependent if they produce the same output variable. It is denoted as follows

$S1 \circ\!\!\rightarrow S2$

- I/O Dependence: The read and write statements are I/O statements. The I/O dependence occurs when the same file is referenced by both I/O statements.
- Unknown dependence: The dependence relation between two statements cannot be determined in the following situations:
  - ➢ The subscript of a variable is itself subscribed(indirect addressing mode) LOAD R1, @100
  - ➢ The subscript does not contain the loop index variable.
  - ➢ A variable appears more than once with subscripts having different coefficients of the loop variable.
  - ➢ The subscript is nonlinear in the loop index variable.

<u>Control Dependence</u>

- The conditional statements are evaluated at run time and hence the execution path followed would be different.
- Different paths taken after a conditional branch may introduce or eliminate data dependencies among instructions.
- Dependence may also exist between operations performed in successive iterations of a looping procedure. In the following, we show one loop example with and another without control-dependent iterations.
- The following loop has independent iterations

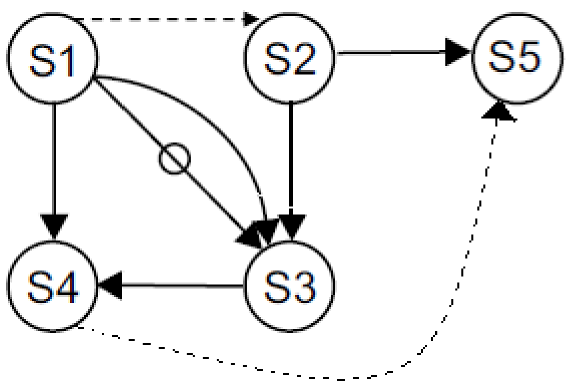
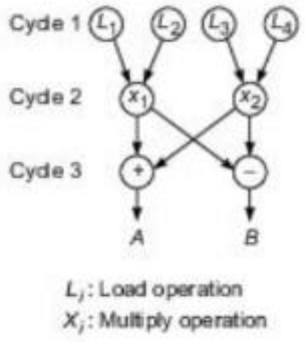
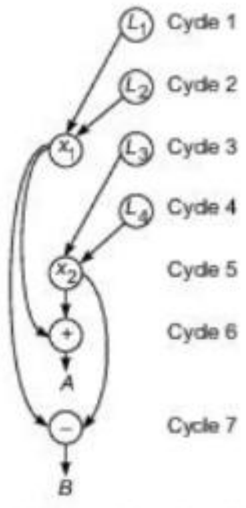
```
      Do    20I = 1, N
            A(I) = C(I)
            IF (A(I) .LT. 0) A(I) = 1
20    Continue
```

- The following loop has dependent iterations

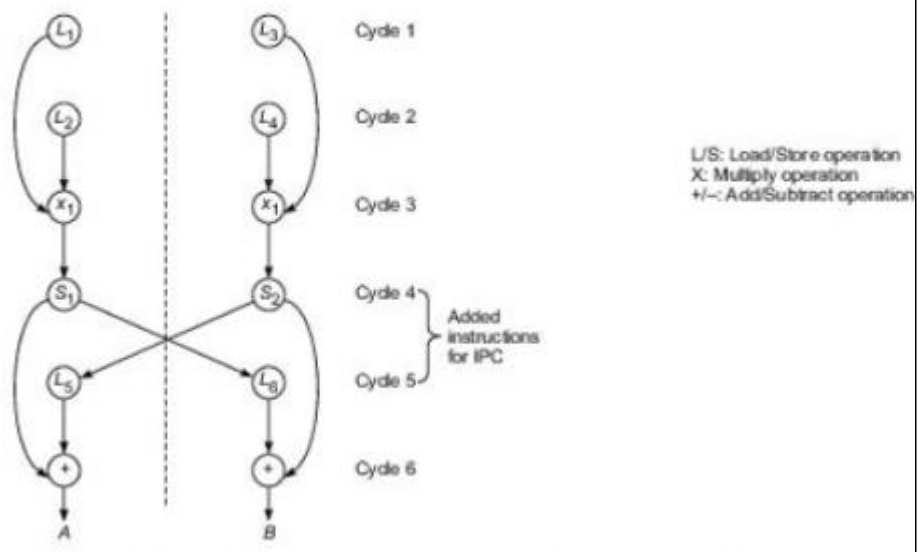
```
      Do    10I = 1, N
            IF (A(I – 1) .EQ. 0) A(I) = 0
10    Continue
```

<u>Resource Dependence</u>

- The Resource Dependence occurs due to conflicts in using shared resources like integer units, floating point units, register or memory areas etc.
- When the conflict is due to ALU unit it is called as ALU dependence and when the conflict is due to storage it is called as storage dependence.

				
5	**Explain Hardware and Software Parallelism with an example.** In this section we will discuss about the hardware and software support needed for parallelism. Example for Software Parallelism and Hardware Parallelism • Consider there are eight instructions (four loads and four arithmetic operations) to be executed in three consecutive machine cycles. Four load operations are performed in the first cycle, followed by two multiply operations in the second cycle and two add/subtract operations in the third cycle. Therefore the parallelism varies from 4 to 2 in three cycles. The average software parallelism is equal to 8/3 = 2.67 instructions per cycle. It is shown in figure given below • Consider the execution of same instructions by two issue processors which can execute one memory access (load or write) and one arithmetic operation simultaneously. With this hardware restriction, the program must execute in seven cycles as shown in figure given below. Therefore hardware parallelism displays an average value of 8/7 = 1.14 instructions executed per cycle. This demonstrates a mismatch between the software parallelism and the hardware parallelism.  Cycle 1 Cycle 2 Cycle 3 L_1 L_2 L_3 L_4 X_1 X_2 + − A B L_i: Load operation X_i: Multiply operation (a) Software parallelism  L_1 Cycle 1 L_2 Cycle 2 X_1 L_3 Cycle 3 L_4 Cycle 4 X_2 Cycle 5 + Cycle 6 A − Cycle 7 B (b) Hardware parallelism • Consider a dual processor system where each processor is a single issue processor. The hardware parallelism is shown in figure given below for 12	(10)	CO1	L2

instructions executed by two processors A and B. S1 and S2 and L5 and L6 are added instructions for Interprocessor communication.



To solve the mismatch problem between software parallelism and hardware parallelism, one approach is to develop compilation support, and the other is through hardware redesign for more efficient exploitation of parallelism.

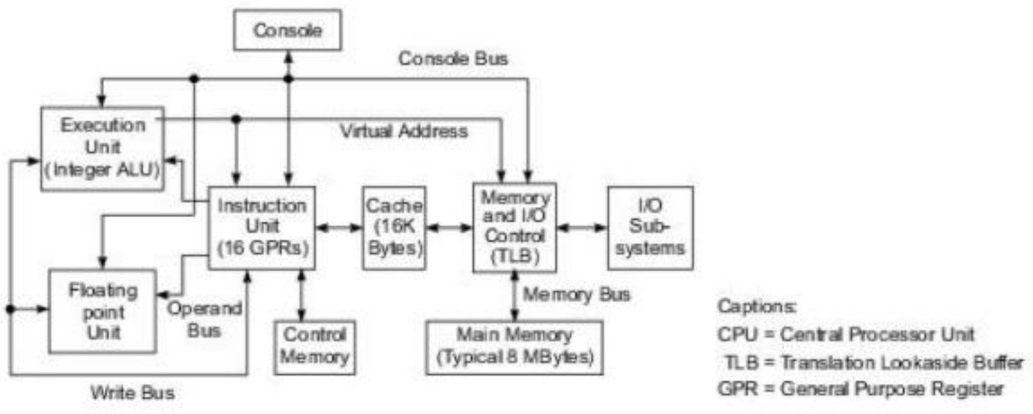
6 (a) **Compare RISC and CISC with respect to its characteristics and its architectural distinctions.** (06) CO2 L2

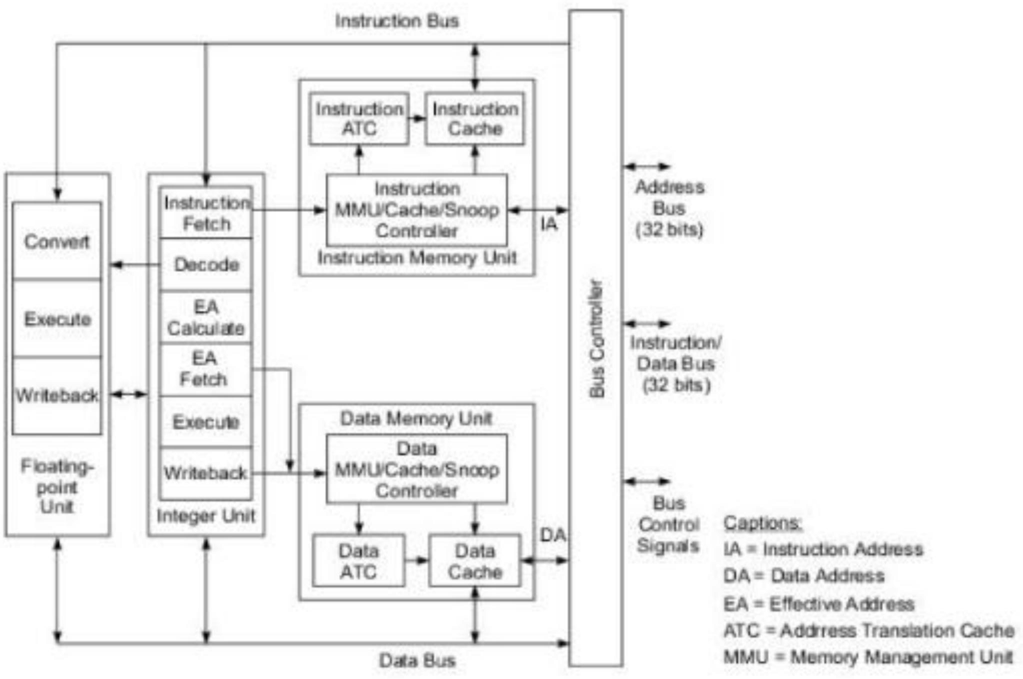
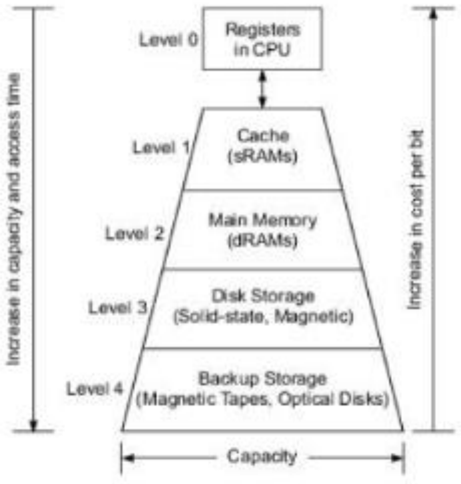
Architectural Consideration	**CISC**	**RISC**
Instruction set size and format	Large set of instructions with variable format(16-64 bits per instruction)	Small set of instructions with fixed format(32 bit per instruction)
Addressing modes	12-24	3-5
General purpose register and cache design	8-24 GPR and unified cache	32-192 GPR and split cache
CPI	Between 2 and 16	Average CPI<1.5
CPU control	Micro programmed	Hardwired

6 (b) **Differentiate VLIW and Superscalar Processor.** (04) CO3 L2

Differences between VLIW and superscalar

- The decoding of VLIW instruction is much easier than superscalar instructions.
- The code density of superscalar machine is better than VLIW when the instruction level parallelism is less.
- The CPI of VLIW processor can be even lower than that of superscalar processor. Example: Multiflow trace computer allows up to seven operations to

	be executed concurrently with 256 bits per VLIW instruction. • The success of VLIW processor depends on the efficiency of code compaction. • By explicitly encoding parallelism in the long instruction, VLIW processor eliminates the need for hardware or software to detect parallelism. Also it has simple hardware design and instruction set. It performs well for scientific applications where the program behavior is more predictable. Not popular among the general purpose applications.			
7	**Explain CISC Scalar processor with example.** The CISC scalar processor may have both integer unit, floating point unit and even multiple such units. It also has support for pipelining. The some early representative CISC scalar processors are VAX 8600,Motorola MC68040,Intel i486 etc Digital Equipment VAX 8600 processor architecture 1. It was introduced by Digital Equipment Corporation in 1985. It has micro programmed control. 2. The instruction set has about 300 instructions with 20 addressing modes. 3. It contains two functional units' i.e. integer unit and floating point unit which are pipelined. 4. It has unified cache. The pipeline is designed with six stages. 5. The Translation Lookaside Buffer (TLB) is built in memory unit for fast generation of physical address from the virtual address. The CPI ranges from 2 to 20 clock cycles.  Console Console Bus Execution Unit (Integer ALU) Virtual Address Instruction Unit (16 GPRs) Cache (16K Bytes) Memory and I/O Control (TLB) I/O Sub-systems Floating point Unit Operand Bus Control Memory Memory Bus Main Memory (Typical 8 MBytes) Write Bus Captions: CPU = Central Processor Unit TLB = Translation Lookaside Buffer GPR = General Purpose Register The Motorola MC68040 microprocessor architecture 1. It has over 100 instructions and supports 18 addressing modes and has 16 general purpose registers. 2. It has split cache.i.e. 4-K byte data cache and 4-K byte instruction cache with separate memory management unit (MMU) and Address Translation cache (ATC). 3. The integer unit is pipelined with 6 stages and floating point unit is pipelined with 3 stages. 4. It is more than 1.2 million transistors. 5. The data formats range from 8 to 80 bits, with provision for the IEEE floating point standard. 6. Snooping logic is built into the memory units for monitoring bus events for cache	(10)	CO2	L2

	invalidation.  Instruction Bus Instruction ATC Instruction Cache Instruction MMU/Cache/Snoop Controller Instruction Memory Unit IA Address Bus (32 bits) Convert Execute Writeback Floating-point Unit Instruction Fetch Decode EA Calculate EA Fetch Execute Writeback Integer Unit Bus Controller Instruction/ Data Bus (32 bits) Data Memory Unit Data MMU/Cache/Snoop Controller Data ATC Data Cache DA Bus Control Signals Data Bus Captions: IA = Instruction Address DA = Data Address EA = Effective Address ATC = Address Translation Cache MMU = Memory Management Unit			
8	**What is memory hierarchy? Explain Inclusion, Coherence, Locality properties with neat diagram.** • The storage devices like register, cache, main memory, disk devices and backup storage devices are organized in a form of hierarchy as shown below. The cache is at level 1, main memory at level 2, disk at level 3 and backup storage at level 4. • Memory devices at a lower level are faster to access, smaller in size, and more expensive per byte, having a higher bandwidth and using a smaller unit of transfer as compared with those at a higher level. • The access time t_i refers to the round-trip time from the CPU to the ith-level memory. The memory size s_i is the number of bytes or words in level i. The cost of the ith-level memory is estimated by the product $c_i s_i$. The bandwidth b_i refers to the rate at which information is transferred between adjacent levels. The unit of transfer x_i refers to the grain size for data transfer between levels i and i+1. • Also $t_{i-1}<t_i, s_{i-1}<s_i, c_{i-1}>c_i, b_{i-1}>b_i$ and $x_{i-1}<x_i$ for i=1,2,3,4 in the hierarchy.  Level 0 Registers in CPU Level 1 Cache (sRAMs) Level 2 Main Memory (dRAMs) Level 3 Disk Storage (Solid-state, Magnetic) Level 4 Backup Storage (Magnetic Tapes, Optical Disks) Capacity Increase in capacity and access time Increase in cost per bit	(10)	CO2	L3

Inclusion

The inclusion property is stated as $M_1 \subset M_2 \subset M_3 \subset \ldots \ldots \subset M_N$. The inclusion relationship implies that all information items are originally stored in the outermost level M_N. During the processing, subsets of M_N, are copied into M_{N-1}. Similarly, subsets of M_{N-1} are copied into M_{N-2} and so on. Hence if word is found in M_i then same word can be found in M_{i+1}, M_{i+2} and so on but may not be found in M_{i-1}

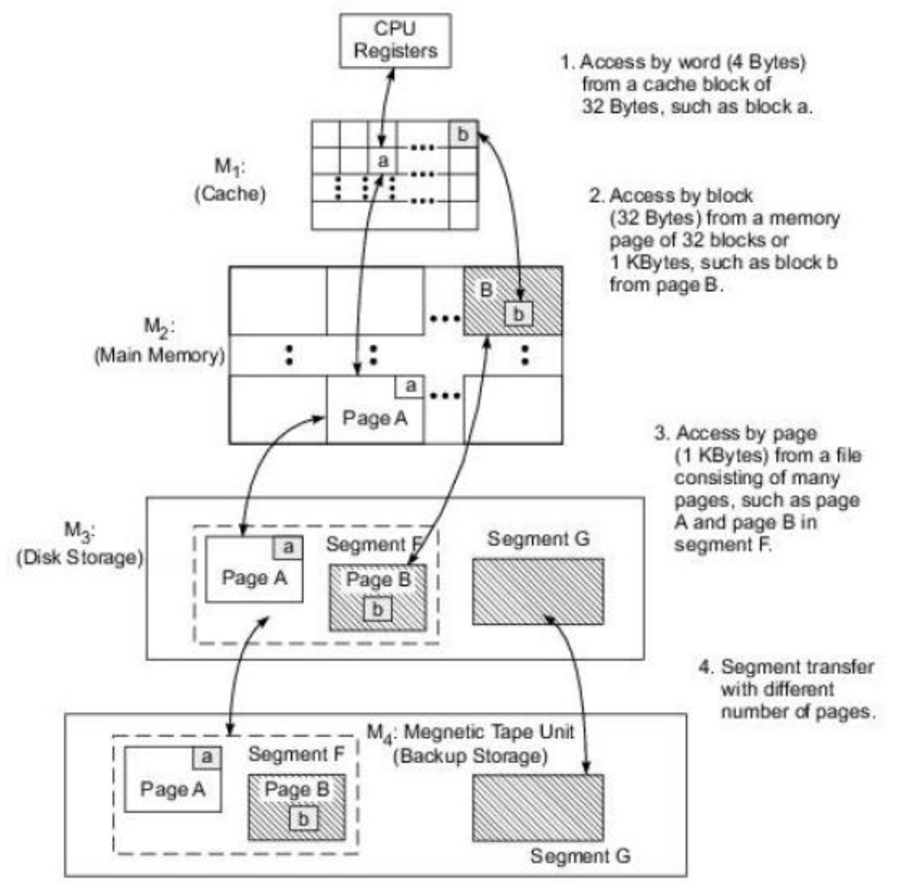

CPU Registers
1. Access by word (4 Bytes) from a cache block of 32 Bytes, such as block a.
M1: (Cache)
a
b
2. Access by block (32 Bytes) from a memory page of 32 blocks or 1 KBytes, such as block b from page B.
M2: (Main Memory)
B
b
Page A
a
3. Access by page (1 KBytes) from a file consisting of many pages, such as page A and page B in segment F.
M3: (Disk Storage)
Page A
a
Segment F
Page B
b
Segment G
4. Segment transfer with different number of pages.
M4: Megnetic Tape Unit (Backup Storage)
Page A
a
Segment F
Page B
b
Segment G

Fig. 4.18 The inclusion property and data transfers between adjacent levels of a memory hierarchy

Coherence

The coherence property requires that copies of the same information item at successive memory levels must be consistent. If a word is modified in the cache, copies of that word must be updated immediately or eventually at all higher levels. In general, there are two strategies for maintaining the coherence in a memory hierarchy.

The first method is called write-through (WT), which demands immediate update in M_{i+1} if a word is modified in M_i.

The second method is write-back (WB), which delays the update in M_{i+1} until the word being modified in M_i is replaced or removed from M_i.

Locality of References

The CPU refers memory to either access the instructions or data. The memory references can be clustered according to time, space or ordering. Hence there are three dimensions for locality of reference i.e. temporal, spatial and sequential locality.

Temporal Locality: Recently referenced items i.e. instructions or data are likely to be referenced again in the near future. This is often caused by special program constructs

	such as iterative loops, process stacks, temporary variables, or subroutines. Once a loop is entered or a subroutine is called, a small code segment will be referenced repeatedly many times. <u>Spatial Locality:</u> This refers to the tendency for a process to access items whose addresses are near one another. For example, operations on tables or arrays involve accesses of a certain clustered area in the address space. <u>Sequential Locality:</u> In typical programs, the execution of instructions follows a sequential order unless branch instructions create out-of-order executions. The ratio of in-order execution to out-of-order execution is roughly 5 to 1 in ordinary programs. Besides, the access of large data array also follows a sequential order.			