DHANALAKSHMI SRINIVASAN ENGINEERING COLLEGE, PERAMBALUR

Two Marks Questions with Answers

CS6303 COMPUTER ARCHITECTURE

II YEAR/ 3rd SEMESTER ECE

UNIT-I	OVERVIEW & INSTRUCTIONS

1. What are the eight great ideas in computer architecture? (MAY-2016)

The eight great ideas in computer architecture are:

1. Design for Moore's Law
2. Use Abstraction to Simplify Design
3. Make the Common Case Fast
4. Performance via Parallelism
5. Performance via Pipelining
6. Performance via Prediction
7. Hierarchy of Memories
8. Dependability via Redundancy

2. What are the five classic components of a computer? (NOV/DEC-2016)

The five classic components of a computer are input, output, memory, datapath, and control, with the last two sometimes combined and called the processor.

3. Define – ISA

The instruction set architecture, or simply architecture of a computer is the interface between the hardware and the lowest-level software. It includes anything programmers need to know to make a binary machine language program work correctly, including instructions, I/O devices, and so on.

4. Define – ABI

Typically, the operating system will encapsulate the details of doing I/O, allocating memory, and other low-level system functions so that application programmers do not need to worry about such details. The combination of the basic instruction set and the operating system interface provided for application programmers is called the application binary interface (ABI).

5. What are the advantages of network computers?

Networked computers have several major advantages:

- Communication: Information is exchanged between computers at high speeds.
- Resource sharing: Rather than each computer having its own I/O devices, computers on the network can share I/O devices.

■ Nonlocal access: By connecting computers over long distances, users need not be near the computer they are using.

6. Define – Response Time (APR/MAY-2016)

Response time is also called execution time. The total time required for the computer to complete a task, including disk accesses, memory accesses, I/O activities, operating system overhead, CPU execution time, and so on is called response time.

7. Define – Throughput (NOV/DEC-2015)

Throughput or bandwidth is the total amount of work done in a given time.

8. Write the CPU performance equation. (MAY-2016)

The Classic CPU 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:

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

9. If computer A runs a program in 10 seconds, and computer B runs the same program in 15 seconds, how much faster is A over B.

We know that A is n times as fast as B if

$$\frac{\text{Performance}_A}{\text{Performance}_B} = \frac{\text{Execution time}_B}{\text{Execution time}_A} = n$$

Thus the performance ratio is

$$\frac{15}{10} = 1.5$$

and A is therefore 1.5 times as fast as B.

10. What are the basic components of performance? (Nov/Dec-2015)

The basic components of performance and how each is measured are:

Components of Performance	Units of measure
CPU execution time for a program	Seconds for the program
Instruction count	Instruction executed for the program
Clock cycles per instruction(CPI)	Average number of clock cycles per instruction
Clock cycle time	Seconds per clock cycle

11. Write the formula for CPU execution time for a program. (Nov/Dec-2016)

$$\text{CPU execution time for a program} = \text{CPU clock cycles for a program} \times \text{Clock cycle time}$$

Alternatively, because clock rate and clock cycle time are inverses,

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

12. Write the formula for CPU clock cycles required for a program. (Apr/May-2014)

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

13. Define – MIPS (Apr/Nov-2015)

Million Instructions Per Second (MIPS) is a measurement of program execution speed based on the number of millions of instructions.

MIPS is computed as:

$$\text{MIPS} = \frac{\text{Instruction count}}{\text{Execution time} \times 10^6}$$

14. What are the fields in an MIPS instruction?

MIPS fields are

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

Where,

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.
funct: Function.

15. Write an example for immediate operand. (Nov/Dec-2014)

The quick add instruction with one constant operand is called add immediate or addi. To add 4 to register $s3, we just write

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

16. Define – Stored Program Concepts

Today's computers are built on two key principles:

1. Instructions are represented as numbers.
2. Programs are stored in memory to be read or written, just like data.

These principles lead to the stored-program concept. Treating instructions in the same way as data greatly simplifies both the memory hardware and the software of computer systems.

17. Define – Addressing Modes. (Nov/Dec-2014)

The different ways in which the operands of an instruction are specified are called as addressing modes.

The MIPS addressing modes are the following:

1. Immediate addressing
2. Register addressing
3. Base or displacement addressing
4. PC-relative addressing
5. Pseudo direct addressing

UNIT-II	ARITHMETIC OPERATIONS

1. Add 6_{10} to 7_{10} in binary and Subtract 6_{10} from 7_{10} in binary. (Nov/Dec-2016)

Addition,

$$\begin{array}{ll} & 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0111_{two} = 7_{ten} \\ + & 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0110_{two} = 6_{ten} \\ \hline = & 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 1101_{two} = 13_{ten} \end{array}$$

Subtraction directly,

$$\begin{array}{ll} & 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0111_{two} = 7_{ten} \\ - & 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0110_{two} = 6_{ten} \\ \hline = & 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_{two} = 1_{ten} \end{array}$$

Or via two's complement of -6,

$$\begin{array}{ll} & 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0111_{two} = 7_{ten} \\ + & 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1010_{two} = -6_{ten} \\ \hline = & 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_{two} = 1_{ten} \end{array}$$

2. Write the overflow conditions for addition and subtraction. (Nov/Dec-2014)

Operation	Operand A	Operand B	Result Indicating overflow
A+B	≥0	≥0	<0
A+B	<0	<0	≥0
A-B	≥0	<0	<0
A-B	<0	≥0	≥0

3. Define – Moore's Law. (Nov/Dec-2015)

Moore's Law has provided so much more in resources that hardware designers can now build much faster multiplication and division hardware. Whether the multiplicand is to be added or not is known at the beginning of the multiplication by looking at each of the 32 multiplier bits.

4. What are the floating point instructions in MIPS? (Nov/Dec-2014)

MIPS supports the IEEE 754 single precision and double precision formats with these instructions:

- Floating-point addition
- Floating-point subtraction
- Floating-point multiplication
- Floating-point division
- Floating-point comparison
- Floating-point branch

5. Define – Guard and Round

Guard is the first of two extra bits kept on the right during intermediate calculations of floating point numbers. It is used to improve rounding accuracy.

Round is a method to make the intermediate floating-point result fit the floating-point format; the goal is typically to find the nearest number that can be represented in the format. IEEE 754, therefore, always keeps two extra bits on the right during intermediate additions, called guard and round, respectively.

6. Define – ULP

Units in the Last Place is defined as the number of bits in error in the least significant bits of the significant between the actual number and the number that can be represented.

7. What is meant by sticky bit?

Sticky bit is a bit used in rounding in addition to guard and round that is set whenever there are nonzero bits to the right of the round bit. This sticky bit allows the computer to see the difference between 0.50 … 00 ten and .... 01 ten when rounding.

8. Write the IEEE 754 floating point format.

The IEEE 754 standard floating point representation is almost always an approximation of the real number.

$$(-1)^S \times (1 + \text{Fraction}) \times 2^{(\text{Exponent} - \text{Bias})}$$

9. What is meant by sub-word parallelism? (Nov/Dec-2016)

Given that the parallelism occurs within a wide word, the extensions are classified as sub-word 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 . 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 sub-word parallelism, which can be used either with ARMv7 or ARMv8.

10. Multiply 1000_{10} * 1001_{10}.

Multiplicand 1000_{ten}
Multiplier × 1001_{ten}

```
  1000
 0000
0000
1000
-------
1001000
```

Product 1001000_{ten}

11. Divide $1{,}001{,}010_{ten}$ by 1000_{ten}.

```
                1001   Quotient
Divisor 1000 | 1001010  Dividend
             -1000
                10
                101
                1010
               -1000
                  10   Remainder
```

12. What are the steps in the floating-point addition? (Nov/Dec-2016)

The steps in the floating-point addition are

1. Align the decimal point of the number that has the smaller exponent.
2. Addition of the significands
3. Normalize the sum.
4. Round the result.

UNIT-III PROCESSOR AND CONTROL UNIT

1. What is meant by data path element?

A data path element is a unit used to operate on or hold data within a processor. In the MIPS implementation, the data path elements include the instruction and data memories, the register file, the ALU, and adders.

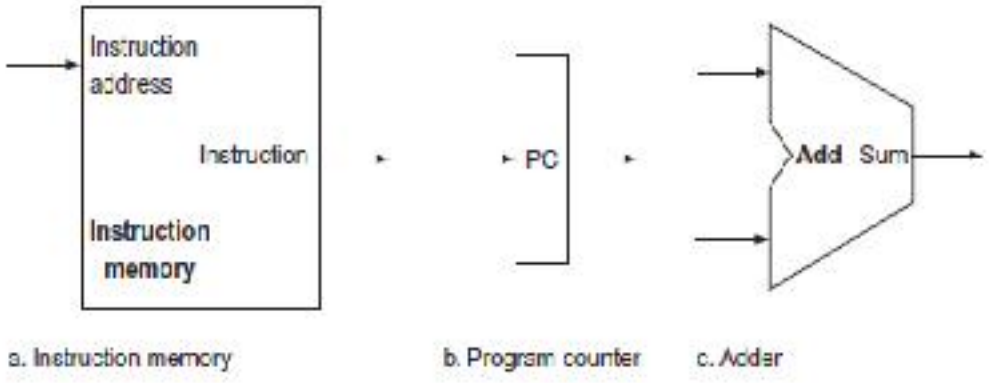
2. What is the use of PC register?

Program Counter (PC) is the register containing the address of the instruction in the program being executed.

3. What is meant by register file?

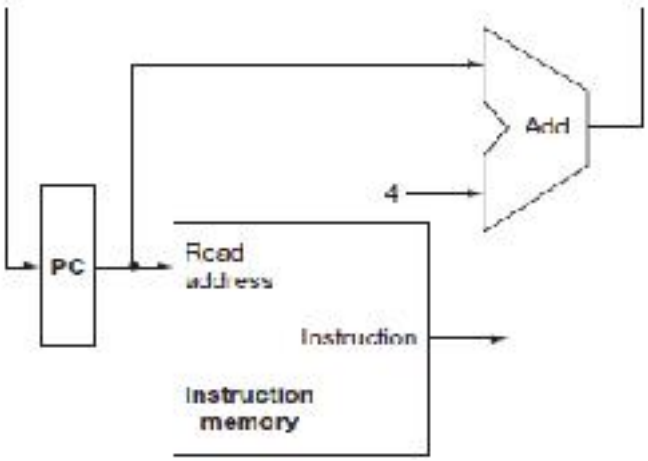
The processor's 32 general-purpose registers are stored in a structure called a register file. A register file is a collection of registers in which any register can be read or written by specifying the number of the register in the file. The register file contains the register state of the computer.

4. What are the two state elements needed to store and access an instruction?



Two state elements are needed to store and access instructions, and an adder is needed to compute the next instruction address. The state elements are the instruction memory and the program counter.

5. Draw the diagram of portion of datapath used for fetching instruction.



A portion of the data path is used for fetching instructions and incrementing the program counter. The fetched instruction is used by other parts of the data path.

6. Define – Sign Extend

Sign-extend is used to increase the size of a data item by replicating the high-order sign bit of the original data item in the high order bits of the larger, destination data item.

7. What is meant by branch target address?

Branch target address is the address specified in a branch, which becomes the new program counter (PC) if the branch is taken. In the MIPS architecture the branch target is given by the sum of the off set field of the instruction and the address of the instruction following the branch.

8. Differentiate branch taken from branch not taken.

Branch taken is a branch where the branch condition is satisfied and the program counter (PC) becomes the branch target. All unconditional jumps are taken branches.

Branch not taken or (untaken branch) is a branch where the branch condition is false and the program counter (PC) becomes the address of the instruction that sequentially follows the branch.

9. What is meant by delayed branch?

Delayed branch is a type of branch where the instruction immediately following the branch is always executed, independent of whether the branch condition is true or false.

10. What are the three instruction classes and their instruction formats?

Field	0	rs	rt	rd	shamt	funct
Bit positions	31:26	25:21	20:16	15:11	10:6	5:0

a. R-type instruction

Field	35 or 43	rs	rt	address
Bit positions	31:26	25:21	20:16	15:0

b. Load or store instruction

Field	4	rs	rt	address
Bit positions	31:26	25:21	20:16	15:0

c. Branch instruction

The three instruction classes (R-type, load and store, and branch) use two different instruction formats.

11. Write the instruction format for the jump instruction.

The destination address for a jump instruction is formed by concatenating the upper 4 bits of the current PC + 4 to the 26-bit address field in the jump instruction and adding 00 as the 2 low-order bits.

Field	000010	address
Bit positions	31:26	25:0

12. What is meant by pipelining?

Pipelining is an implementation technique in which multiple instructions are overlapped in execution. Pipelining improves performance by increasing instruction throughput, as opposed to decreasing the execution time of an individual instruction.

13. What are the five steps in MIPS instruction execution?

1. Fetch instruction from memory.
2. Read registers while decoding the instruction. The regular format of MIPS instructions allows reading and decoding to occur simultaneously.
3. Execute the operation or calculate an address.
4. Access an operand in data memory.
5. Write the result into a register.

14. Write the formula for calculating time between instructions in a pipelined processor.

$$\text{Time between instructions}_{\text{pipelined}} = \frac{\text{Time between instruction}_{\text{nonpipelined}}}{\text{Number of pipe stages}}$$

15. What are hazards? Write its types.

There are situations in pipelining when the next instruction cannot be executed in the following clock cycle. These events are called hazards, and there are three different types.

1. Structural Hazards
2. Data Hazards
3. Control Hazards

16. What is meant by forwarding?

Forwarding, also called bypassing, is a method of resolving a data hazard by retrieving the missing data element from internal buffers rather than waiting for it to arrive from programmer visible registers or memory.

17. What is pipeline stall?

Pipeline stall, also called bubble, is a stall initiated in order to resolve a hazard. They can be seen elsewhere in the pipeline.

18. What is meant by branch prediction?

Branch prediction is a method of resolving a branch hazard that assumes a given outcome for the branch and proceeds from that assumption rather than waiting to ascertain the actual outcome.

19. What are the 5 pipeline stages?

The 5 stages of instruction execution in a pipelined processor are:

1. IF: Instruction fetch
2. ID: Instruction decode and register file read
3. EX: Execution or address calculation
4. MEM: Data memory access
5. WB: Write back

20. What are exceptions and interrupts?

Exception, also called interrupt, is an unscheduled event that disrupts program execution used to detect overflow. Eg. Arithmetic overflow, using an undefined instruction.

Interrupt is an exception that comes from outside of the processor. Eg. I/O device request

21. Define – Vectored Interrupts

Vectored interrupt is an interrupt in that the address to which the control is transferred is determined by the cause of the exception.

12

UNIT-IV	PARALLELISM

1. What is meant by ILP? (Nov/Dec-2016)

Pipelining exploits the potential parallelism among instructions. This parallelism is called instruction-level parallelism (ILP). There are two primary methods for increasing the potential amount of instruction-level parallelism.

1. Increasing the depth of the pipeline to overlap more instructions.
2. Multiple issue.

2. What is multiple issue? Write any two approaches.

Multiple issue is a scheme whereby multiple instructions are launched in one clock cycle. It is a method for increasing the potential amount of instruction-level parallelism. It is done by replicating the internal components of the computer so that it can launch multiple instructions in every pipeline stage. The two approaches are:

1. Static multiple issue (at compile time)
2. Dynamic multiple issue (at run time)

3. What is meant by speculation? (Apr/May-2014)

One of the most important methods for finding and exploiting more ILP is speculation. It is an approach whereby the compiler or processor guesses the outcome of an instruction to remove it as dependence in executing other instructions.

For example, we might speculate on the outcome of a branch, so that instructions after the branch could be executed earlier.

4. Define – Static Multiple Issue

Static multiple issue is an approach to implement a multiple-issue processor where many decisions are made by the compiler before execution.

5. Define – Issue Slots and Issue Packet

Issue slots are the positions from which instructions could be issued in a given clock cycle. By analogy, these correspond to positions at the starting blocks for a sprint.

Issue packet is the set of instructions that issues together in one clock cycle; the packet may be determined statically by the compiler or dynamically by the processor.

6. Define – VLIW

Very Long Instruction Word (VLIW) is a style of instruction set architecture that launches many operations that are defined to be independent in a single wide instruction, typically with many separate opcode fields.

7. Define – Superscalar Processor. (Nov/Dec-2016)

Superscalar is an advanced pipelining technique that enables the processor to execute more than one instruction per clock cycle by selecting them during execution. Dynamic multiple-issue processors are also known as superscalar processors, or simply superscalars.

8. What is meant by loop unrolling?

An important compiler technique to get more performance from loops is loop unrolling, where multiple copies of the loop body are made. After unrolling, there is more ILP available by overlapping instructions from different iterations.

9. What is meant by anti-dependence? How is it removed?

Anti-dependence is an ordering forced by the reuse of a name, typically a register, rather than by a true dependence that carries a value between two instructions. It is also called as name dependence.

Register renaming is the technique used to remove anti-dependence in which the registers are renamed by the compiler or hardware.

10. What is the use of reservation station and reorder buffer?

Reservation station is a buffer within a functional unit that holds the operands and the operation.

Reorder buffer is the buffer that holds results in a dynamically scheduled processor until it is safe to store the results to memory or a register.

11. Differentiate in-order execution from out-of-order execution.

Out-of-order execution is a situation in pipelined execution when an instruction is blocked from executing does not cause the following instructions to wait. It preserves the data flow order of the program.

In-order execution requires the instruction fetch and decode unit to issue instructions in order, which allows dependences to be tracked, and requires the commit unit to write results to registers and memory in program fetch order. This conservative mode is called in-order commit.

12. What is meant by hardware multithreading?

Hardware multithreading allows multiple threads to share the functional units of a single processor in an overlapping fashion to try to utilize the hardware resources efficiently. To permit this sharing, the processor must duplicate the independent state of each thread. It Increases the utilization of a processor.

13. What are the two main approaches to hardware multithreading?

There are two main approaches to hardware multithreading. Fine-grained multithreading switches between threads on each instruction, resulting in interleaved execution of multiple threads. This interleaving is often done in a round-robin fashion, skipping any threads that are stalled at that clock cycle.

Coarse-grained multithreading is an alternative to fine-grained multithreading. It switches threads only on costly stalls, such as last-level cache misses.

14. What is SMT?

Simultaneous Multithreading (SMT) is a variation on hardware multithreading that uses the resources of a multiple-issue, dynamically scheduled pipelined processor to exploit thread-level parallelism. It also exploits instruction level parallelism.

15. Differentiate SMT from hardware multithreading.

Since SMT relies on the existing dynamic mechanisms, it does not switch resources every cycle. Instead, SMT is always executing instructions from multiple threads, leaving it up to the hardware to associate instruction slots and renamed registers with their proper threads.

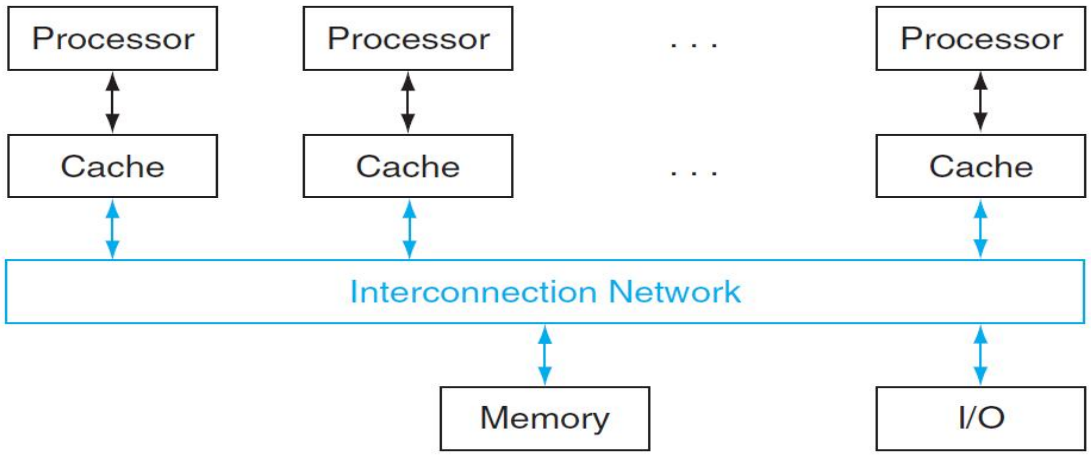
16. What are the three multithreading options? (Nov/Dec-2016)

The three multithreading options are:

1. A superscalar with coarse-grained multithreading
2. A superscalar with fine-grained multithreading
3. A superscalar with simultaneous multithreading

17. Define – SMP. (Nov/Dec-2016)

Shared memory multiprocessor (SMP) is one that offers the programmer a single physical address space across all processors - which is nearly always the case for multicore chips. Processors communicate through shared variables in memory, with all processors capable of accessing any memory location via loads and stores.



18. Differentiate UMA from NUMA.(nov/2015)

Uniform memory access (UMA) is a multiprocessor in which latency to any word in main memory is about the same no matter which processor requests the access.

Non uniform memory access (NUMA) is 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.

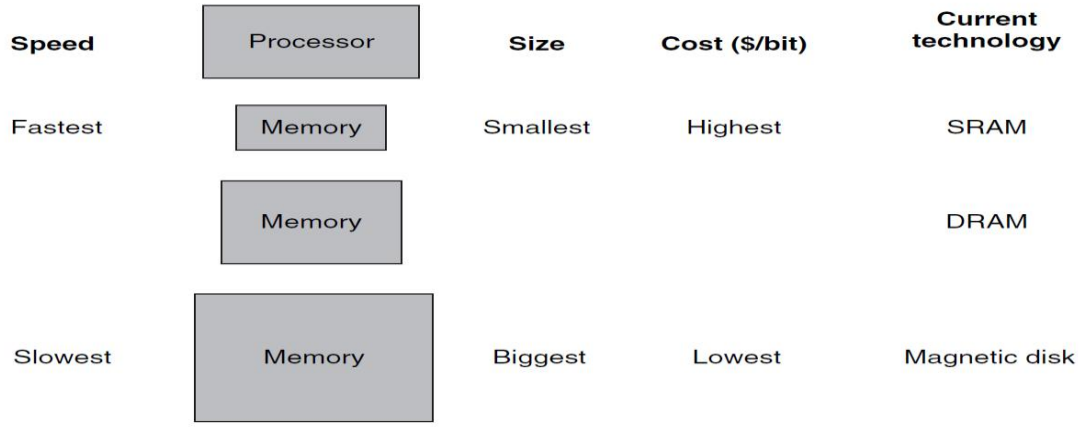
UNIT-V	MEMORY AND I/O SYSTEMS

1. What are the temporal and spatial localities of references? (Nov/Dec-2016)

Temporal locality (locality in time): if an item is referenced, it will tend to be referenced again soon.

Spatial locality (locality in space): if an item is referenced, items whose addresses are close by will tend to be referenced soon.

2. Write the structure of memory hierarchy.



The basic structure of a memory hierarchy.

3. What are the various memory technologies?

The various memory technologies are:

1. SRAM semiconductor memory
2. DRAM semiconductor memory
3. Flash semiconductor memory
4. Magnetic disk

4. Differentiate SRAM from DRAM. (Nov/Dec-2016)

SRAMs are simply integrated circuits that are memory arrays with a single access port that can provide either a read or a write. SRAMs have a fixed access time to any datum. SRAMs don't need to refresh and so the access time is very close to the cycle time. SRAMs typically use six to eight transistors per bit to prevent the information from being disturbed when read. SRAM needs only minimal power to retain the charge in standby mode.

In a dynamic RAM (DRAM), the value kept in a cell is stored as a charge in a

capacitor. A single transistor is then used to access this stored charge, either to read the value or to overwrite the charge stored there. Because DRAMs use only a single transistor per bit of storage, they are much denser and cheaper per bit than SRAM. As DRAMs store the charge on a capacitor, it cannot be kept indefinitely and must periodically be refreshed.

5. What is flash memory?

Flash memory is a type of electrically erasable programmable read-only memory (EEPROM). Unlike disks and DRAM, EEPROM technologies can wear out flash memory bits. To cope with such limits, most flash products include a controller to spread the writes by remapping blocks that have been written many times to less trodden blocks. This technique is called wear levelling.

6. Define − Rotational Latency

Rotational latency, also called rotational delay, is the time required for the desired sector of a disk to rotate under the read/write head, usually assumed to be half the rotation time.

7. What is direct-mapped cache? (Nov/Dec-2016)

Direct-mapped cache is a cache structure in which each memory location is mapped to exactly one location in the cache. For example, almost all direct-mapped caches use this mapping to find a block,

(Block address) modulo (Number of blocks in the cache)

8. Consider a cache with 64 blocks and a block size of 16 bytes. To what block number does byte address 1200 map?

The block is given by,

(Block address) modulo (Number of blocks in the cache)

where the address of the block is

$$\frac{\text{Byte address}}{\text{Bytes per block}}$$

Notice that this block address is the block containing all addresses between

$$\left\lfloor \frac{\text{Byte address}}{\text{Bytes per block}} \right\rfloor \times \text{Bytes per block}$$

and

$$\left\lfloor \frac{\text{Byte address}}{\text{Bytes per block}} \right\rfloor \times \text{Bytes per block} + (\text{Bytes per block} - 1)$$

Thus, with 16 bytes per block, byte address 1200 is block address

$$\left\lfloor \frac{1200}{6} \right\rfloor = 75$$

which maps to cache block number (75 modulo 64) = 11. In fact, this block maps all addresses between 1200 and 1215.

9. How many total bits are required for a direct-mapped cache with 16 KiB of data and 4-word blocks, assuming a 32-bit address?

We know that 16 KiB is 4096 (2^{12}) words. With a block size of 4 words (2^2), there are 1024 (2^{10}) blocks. Each block has 4×32 or 128 bits of data plus a tag, which is $32 - 10 - 2 - 2$ bits, plus a valid bit. Thus, the total cache size is

$$2^{10} \times (4 \times 32 + (32 - 10 - 2 - 2) + 1) = 2^{10} \times 147 = 147 \text{ Kibibits}$$

or 18.4 KiB for a 16 KiB cache. For this cache, the total number of bits in the cache is about 1.15 times as many as needed just for the storage of the data.

10. What are the writing strategies in cache memory?

Write-through is a scheme in which writes always update both the cache and the next lower level of the memory hierarchy, ensuring that data is always consistent between the two.

Write-back is a scheme that handles writes by updating values only to the block in the cache, then writing the modified block to the lower level of the hierarchy when the block is replaced.

11. What are the steps to be taken in an instruction cache miss?

The steps to be taken on an instruction cache miss are

1. Send the original PC value (current PC – 4) to the memory.
2. Instruct main memory to perform a read and wait for the memory to complete its access.
3. Write the cache entry, putting the data from memory in the data portion of

the entry, writing the upper bits of the address (from the ALU) into the tag field, and turning the valid bit on.

4. Restart the instruction execution at the first step, which will refetch the instruction, this time finding it in the cache.

12. Define – AMAT

Average memory access time is the average time to access memory considering both hits and misses and the frequency of different accesses.It is equal to the following:

$$\text{AMAT} = \text{Time for a hit} + \text{Miss rate} \times \text{Miss penalty}$$

13. What are the various block placement schemes in cache memory?

Direct-mapped cache is a cache structure in which each memory location is mapped to exactly one location in the cache.

Fully associative cache is a cache structure in which a block can be placed in any location in the cache.

Set-associative cache is a cache that has a fixed number of locations (at least two) where each block can be placed.

14. Define – MTTF and AFR

Reliability is a measure of the continuous service accomplishment or, equivalently, of the time to failure from a reference point. Hence, mean time to failure (MTTF) is a reliability measure. A related term is annual failure rate (AFR), which is just the percentage of devices that would be expected to fail in a year for a given MTTF.

15. Define – Availability

Availability is then a measure of service accomplishment with respect to the alternation between the two states of accomplishment and interruption. Availability is statistically quantified as

$$\text{Availability} = \frac{\text{MTTF}}{(\text{MTTF} + \text{MTTR})}$$

16. What are the three ways to improve MTTF?

The three ways to improve MTTF are:

1. Fault avoidance: Preventing fault occurrence by construction.
2. Fault tolerance: Using redundancy to allow the service to comply with the service specification despite faults occurring.
3. Fault forecasting: Predicting the presence and creation of faults, allowing the component to be replaced before it fails.

17. Define – TLB

Translation-Lookaside Buffer (TLB) is a cache that keeps track of recently used address mappings to try to avoid an access to the page table.

18. What is meant by virtual memory? (Nov/Dec-2016) (Nov/Dec-2014)

Virtual memory is a technique that uses main memory as a "cache" for secondary storage. Two major motivations for virtual memory: to allow efficient and safe sharing of memory among multiple programs, and to remove the programming burdens of a small, limited amount of main memory.

19. Differentiate physical address from logical address.

Physical address is an address in main memory.

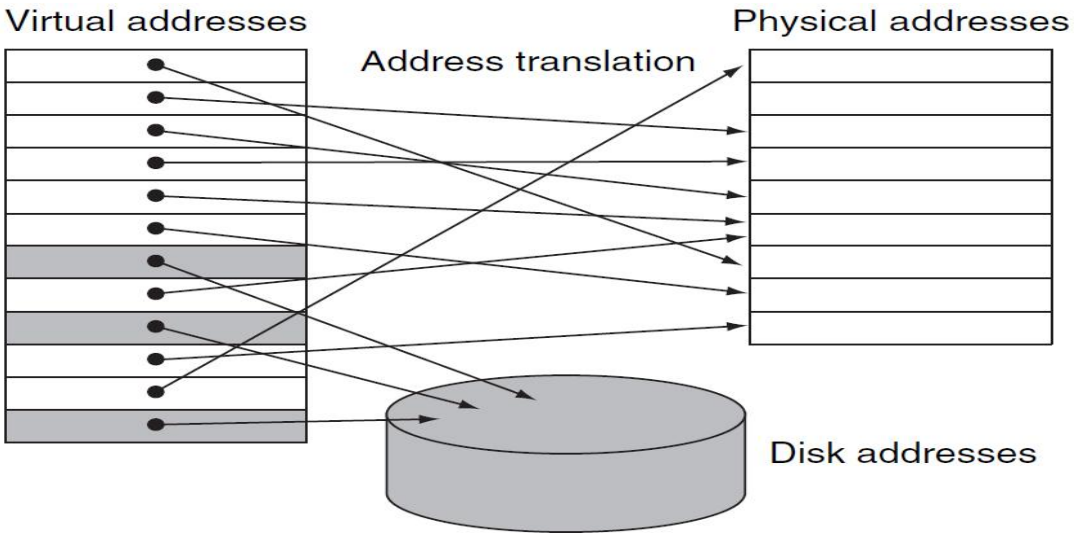
Logical address (or) virtual address is the CPU generated addresses that corresponds to a location in virtual space and is translated by address mapping to a physical address when memory is accessed.

20. Define – Page Fault

Page fault is an event that occurs when an accessed page is not present in main memory.

21. What is meant by address mapping? (Nov/Dec-2016)

Address translation also called address mapping is the process by which a virtual address is mapped to an address used to access memory.



PART B Questions with Answers

CS6303 COMPUTER ARCHITECTURE

II YEAR/ 3rd SEMESTER ECE

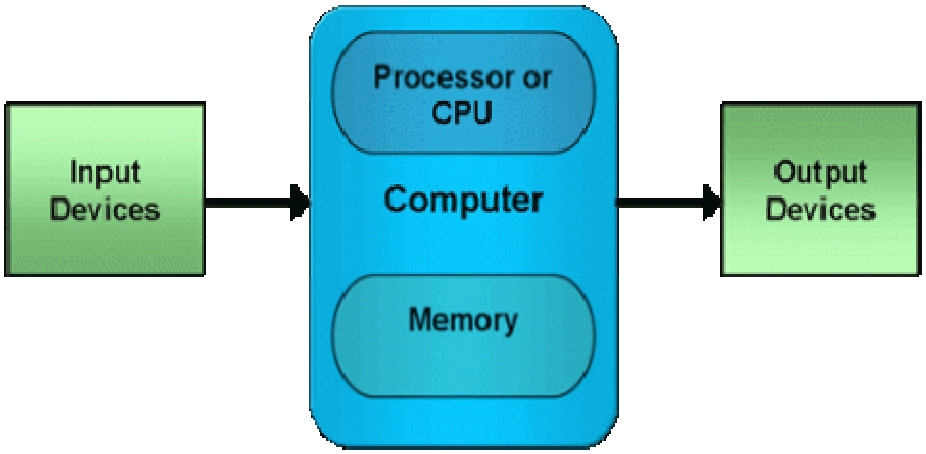
UNIT-I	OVERVIEW & INSTRUCTIONS

1. Discuss about various component of a computer system.**(Nov/2014/2015/2016/2017)**

Components of computer system

1.Input Unit
2.Output Unit
3.Storage Unit
4.Central Processing Unit (CPU)
5.Arithmetic and Logic Unit (ALU)
6.Control Unit

The internal architectural design of computers differs from one system model to another. However, the basic organization remains the same for all computer systems. The following five units (also called ***"The functional units"***) correspond to the five basic operations performed by all computer systems.



Input Unit

Data and instructions must enter the computer system before any computation can be performed on the supplied data. The input unit that links the external environment with the computer system performs this task. Data and instructions enter input units in forms that depend upon the particular device used. For example, data is entered from a keyboard in a manner similar to typing, and this differs from the way in which data is entered through a

mouse, which is another type of input device. However, regardless of the form in which they receive their inputs, all input devices must provide a computer with data that are transformed into the binary codes that the primary memory of the computer is designed to accept. This transformation is accomplished by units that called input interfaces. Input interfaces are designed to match the unique physical or electrical characteristics of input devices to the requirements of the computer system.

In short, an input unit performs the following functions.

1. It accepts (or reads) the list of instructions and data from the outside world.
2. It converts these instructions and data in computer acceptable format.
3. It supplies the converted instructions and data to the computer system for further processing.

Output Unit

The job of an output unit is just the reverse of that of an input unit. It supplied information and results of computation to the outside world. Thus it links the computer with the external environment. As computers work with binary code, the results produced are also in the binary form. Hence, before supplying the results to the outside world, it must be converted to human acceptable (readable) form. This task is accomplished by units called output interfaces.

In short, the following functions are performed by an output unit.

1. It accepts the results produced by the computer which are in coded form and hence cannot be easily understood by us.
2. It converts these coded results to human acceptable (readable) form.
3. It supplied the converted results to the outside world.

Storage Unit

The data and instructions that are entered into the computer system through input units have to be stored inside the computer before the actual processing starts. Similarly, the results produced by the computer after processing must also be kept somewhere inside the computer system before being passed on to the output units. Moreover, the intermediate results produced by the computer must also be preserved for ongoing processing. The **Storage Unit** or the **primary / main storage** of a computer system is designed to do all these things. It provides space for storing data and instructions, space for intermediate results and also space for the final results.

In short, the specific functions of the storage unit are to store:

1. All the data to be processed and the instruction required for processing (received from input devices).
2. Intermediate results of processing.

3. Final results of processing before these results are released to an output device.

Central Processing Unit (CPU)

The main unit inside the computer is the **CPU**. This unit is responsible for all events inside the computer. It controls all internal and external devices, performs ***"Arithmetic and Logical operations"***. The operations a Microprocessor performs are called ***"instruction set"*** of this processor. The instruction set is "hard wired" in the CPU and determines the machine language for the CPU. The more complicated the instruction set is, the slower the CPU works. Processors differed from one another by the instruction set. If the same program can run on two different computer brands they are said to be compatible. Programs written for IBM compatible computers will not run on Apple computers because these two architectures are not compatible.

The control Unit and the Arithmetic and Logic unit of a computer system are jointly known as the Central Processing Unit (CPU). The CPU is the brain of any computer system. In a human body, all major decisions are taken by the brain and the other parts of the body function as directed by the brain. Similarly, in a computer system, all major calculations and comparisons are made inside the CPU and the CPU is also responsible for activating and controlling the operations of other units of a computer system.

Arithmetic and Logic Unit (ALU)

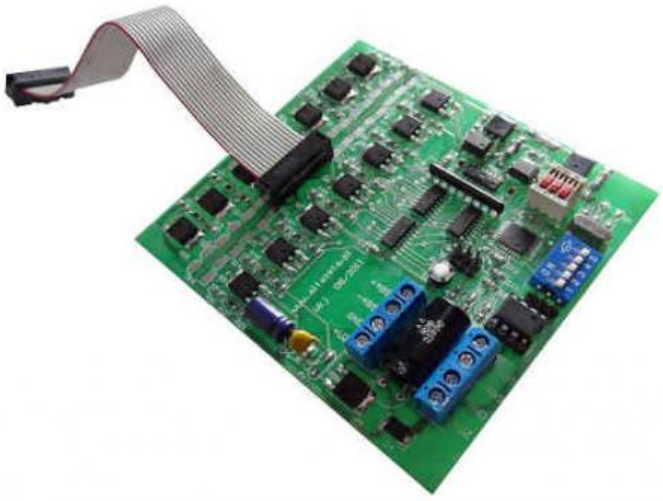
The **arithmetic and logic unit (ALU)** of a computer system is the place where the actual execution of the instructions take place during the processing operations. All calculations are performed and all comparisons (decisions) are made in the **ALU**. The data and instructions, stored in the primary storage prior to processing are transferred as and when needed to the ALU where processing takes place. No processing is done in the primary storage unit. Intermediate results generated in the ALU are temporarily transferred back to the primary storage until needed at a later time. Data may thus move from primary storage to ALU and back again as storage many times before the processing is over. After the completion of processing, the final results which are stored in the storage unit are released to an output device.

The arithmetic and logic unit (ALU) is the part where actual computations take place. It consists of circuits that perform arithmetic operations (e.g. addition, subtraction, multiplication, division over data received from memory and capable to compare numbers (less than, equal to, or greater than).

While performing these operations the ALU takes data from the temporary storage are inside the CPU named registers. Registers are a group of cells used for memory addressing, data manipulation and processing. Some of the registers are general purpose and some are reserved for certain functions. It is a high-speed memory which holds only data from immediate processing and results of this processing. If these results are not needed for the next instruction, they are sent back to the main memory and registers are occupied by the new data used in the next instruction.

All activities in the computer system are composed of thousands of individual steps. These steps should follow in some order in fixed intervals of time. These intervals are generated by the Clock Unit. Every operation within the CPU takes place at the clock pulse. No operation, regardless of how simple, can be performed in less time than transpires between ticks of this clock. But some operations required more than one clock pulse. The faster the clock runs, the faster the computer performs. The clock rate is measured in megahertz (Mhz) or Gigahertz (Ghz). Larger systems are even faster. In older systems the clock unit is external to the microprocessor and resides on a separate chip. In most modern microprocessors the clock is usually incorporated within the CPU.

Control Unit



How the input device knows that it is time for it to feed data into the storage unit? How does the ALU know what should be done with the data once it is received? And how is it that only the final results are sent to the output devices and not the intermediate results? All this is possible because of the control unit of the computer system. By selecting, interpreting, and seeing to the execution of the program instructions, the control unit is able to maintain order and directs the operation of the entire system. Although, it does not perform any actual processing on the data, the control unit acts as a central nervous system for the other

components of the computer. It manages and coordinates the entire computer system. It obtains instructions from the program stored in main memory, interprets the instructions, and issues signals that cause other units of the system to execute them.

The control unit directs and controls the activities of the internal and external devices. It interprets the instructions fetched into the computer, determines what data, if any, are needed, where it is stored, where to store the results of the operation, and sends the control signals to the devices involved in the execution of the instructions.

2. Elaborate the different types of addressing modes with a suitable example**.(2014/2015/2016/2017)**

Addressing Modes

The term ***addressing modes*** refers to the way in which the operand of an instruction is specified. Information contained in the instruction code is the value of the operand or the address of the result/operand. Following are the main addressing modes that are used on various platforms and architectures.

1) Immediate Mode

The operand is an immediate value is stored explicitly in the instruction:

2) Index Mode

The address of the operand is obtained by adding to the contents of the general register (called index register) a constant value. The number of the index register and the constant value are included in the instruction code. Index Mode is used to access an array whose elements are in successive memory locations. The content of the instruction code, represents the starting address of the array and the value of the index register, and the index value of the current element. By incrementing or decrementing index register different element of the array can be accessed.

3) Indirect Mode

The effective address of the operand is the contents of a register or main memory location, location whose address appears in the instruction. Indirection is noted by placing the name of the register or the memory address given in the instruction in parentheses. The register or memory location that contains the address of the operand is a pointer. When an execution takes place in such mode, instruction may be told to go to a specific address. Once it's there, instead of finding an operand, it finds an address where the operand is located.

4) Absolute (Direct) Mode

The address of the operand is embedded in the instruction code.

5) Register Mode

The name (the number) of the CPU register is embedded in the instruction. The register contains the value of the operand. The number of bits used to specify the register depends on the total number of registers from the processor set.

6) Displacement Mode

Similar to index mode, except instead of a index register a base register will be used. Base register contains a pointer to a memory location. An integer (constant) is also referred to as a displacement. The address of the operand is obtained by adding the contents of the base register plus the constant. The difference between index mode and displacement mode is in the number of bits used to represent the constant. When the constant is represented a number of bits to access the memory, then we have index mode. Index mode is more appropriate for array accessing; displacement mode is more appropriate for structure (records) accessing.

7) Auto increment /Auto decrement Mode

A special case of indirect register mode. The register, whose number is included in the instruction code, contains the address of the operand. Auto increment Mode = after operand addressing, the contents of the register is incremented. Decrement Mode = before operand addressing, the contents of the register is decrement.

3. Discuss about the various techniques to represent instruction in a computer system. (2014/2015/2016)

MIPS assembly language

Category	Instruction	Example	Meaning	Comments
Arithmetic	add	add $s1,$s2,$s3	$s1 = $s2 + $s3	Three register operands
	subtract	sub $s1,$s2,$s3	$s1 = $s2 – $s3	Three register operands
	add immediate	addi $s1,$s2,20	$s1 = $s2 + 20	Used to add constants
Data transfer	load word	lw $s1,20($s2)	$s1 = Memory[$s2 + 20]	Word from memory to register
	store word	sw $s1,20($s2)	Memory[$s2 + 20] = $s1	Word from register to memory
	load half	lh $s1,20($s2)	$s1 = Memory[$s2 + 20]	Halfword memory to register
	load half unsigned	lhu $s1,20($s2)	$s1 = Memory[$s2 + 20]	Halfword memory to register
	store half	sh $s1,20($s2)	Memory[$s2 + 20] = $s1	Halfword register to memory
	load byte	lb $s1,20($s2)	$s1 = Memory[$s2 + 20]	Byte from memory to register
	load byte unsigned	lbu $s1,20($s2)	$s1 = Memory[$s2 + 20]	Byte from memory to register
	store byte	sb $s1,20($s2)	Memory[$s2 + 20] = $s1	Byte from register to memory
	load linked word	ll $s1,20($s2)	$s1 = Memory[$s2 + 20]	Load word as 1st half of atomic swap
	store condition. word	sc $s1,20($s2)	Memory[$s2+20]=$s1;$s1=0 or 1	Store word as 2nd half of atomic swap
	load upper immed.	lui $s1,20	$s1 = 20 * 2^{16}	Loads constant in upper 16 bits

Logical	and	and $s1,$s2,$s3	$s1 = $s2 & $s3	Three reg. operands; bit-by-bit AND
	or	or $s1,$s2,$s3	$s1 = $s2 | $s3	Three reg. operands; bit-by-bit OR
	nor	nor $s1,$s2,$s3	$s1 = ~ ($s2 | $s3)	Three reg. operands; bit-by-bit NOR
	and immediate	andi $s1,$s2,20	$s1 = $s2 & 20	Bit-by-bit AND reg with constant
	or immediate	ori $s1,$s2,20	$s1 = $s2 | 20	Bit-by-bit OR reg with constant
	shift left logical	sll $s1,$s2,10	$s1 = $s2 << 10	Shift left by constant
	shift right logical	srl $s1,$s2,10	$s1 = $s2 >> 10	Shift right by constant
Conditional branch	branch on equal	beq $s1,$s2,25	if ($s1 == $s2) go to PC + 4 + 100	Equal test; PC-relative branch
	branch on not equal	bne $s1,$s2,25	if ($s1!= $s2) go to PC + 4 + 100	Not equal test; PC-relative
	set on less than	slt $s1,$s2,$s3	if ($s2 < $s3) $s1 = 1; else $s1 = 0	Compare less than; for beq, bne
	set on less than unsigned	sltu $s1,$s2,$s3	if ($s2 < $s3) $s1 = 1; else $s1 = 0	Compare less than unsigned
	set less than immediate	slti $s1,$s2,20	if ($s2 < 20) $s1 = 1; else $s1 = 0	Compare less than constant
	set less than immediate unsigned	sltiu $s1,$s2,20	if ($s2 < 20) $s1 = 1; else $s1 = 0	Compare less than constant unsigned
Unconditional jump	jump	j 2500	go to 10000	Jump to target address
	jump register	jr $ra	go to $ra	For switch, procedure return
	jump and link	jal 2500	$ra = PC + 4; go to 10000	For procedure call

The *Instruction Set Architecture* (ISA) is the part of the processor that is visible to the programmer or compiler writer. The ISA serves as the boundary between software and hardware. We will briefly describe the instruction sets found in many of the microprocessors used today. The ISA of a processor can be described using 5 catagories:

Operand Storage in the CPU

Where are the operands kept other than in memory?

Number of explicit named operands

How many operands are named in a typical instruction.

Operand location

Can any ALU instruction operand be located in memory? Or must all operands be kept internaly in the CPU?

Operations

What operations are provided in the ISA.

Type and size of operands

What is the type and size of each operand and how is it specified?

Of all the above the most distinguishing factor is the first.

The 3 most common types of ISAs are:

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

Lets look at the assembly code of

A = B + C;

in all 3 architectures:

Stack	Accumulator	GPR
PUSH A	LOAD A	LOAD R1,A
PUSH B	ADD B	ADD R1,B
ADD	STORE C	STORE R1,C
POP C	-	-

Not all processors can be neatly tagged into one of the above catagories. The i8086 has many instructions that use implicit operands although it has a general register set. The i8051 is another example, it has 4 banks of GPRs but most instructions must have the A register as one of its operands.
What are the advantages and disadvantages of each of these approachs?

Stack

Advantages: Simple Model of expression evaluation (reverse polish). Short instructions.
Disadvantages: A stack can't be randomly accessed This makes it hard to generate eficient code. The stack itself is accessed every operation and becomes a bottleneck.

Accumulator

Advantages: Short instructions.
Disadvantages: The accumulator is only temporary storage so memory traffic is the highest for this approach.

GPR

Advantages: Makes code generation easy. Data can be stored for long periods in registers.
Disadvantages: All operands must be named leading to longer instructions.

Earlier CPUs were of the first 2 types but in the last 15 years all CPUs made are GPR processors. The 2 major reasons are that registers are faster than memory, the more data that can be kept internaly in the CPU the faster the program wil run. The other reason is that registers are easier for a compiler to use.

Reduced Instruction Set Computer (RISC)

As we mentioned before most modern CPUs are of the GPR (General Purpose Register) type. A few examples of such CPUs are the IBM 360, DEC VAX, Intel 80x86 and Motorola 68xxx. But while these CPUS were clearly better than previous stack and accumulator based CPUs they were still lacking in several areas:

1. Instructions were of varying length from 1 byte to 6-8 bytes. This causes problems with the pre-fetching and pipelining of instructions.
2. ALU (Arithmetic Logical Unit) instructions could have operands that were memory locations. Because the number of cycles it takes to access memory varies so does the whole instruction. This isn't good for compiler writers, pipelining and multiple issue.
3. Most ALU instruction had only 2 operands where one of the operands is also the destination. This means this operand is destroyed during the operation or it must be saved before somewhere.

Thus in the early 80's the idea of RISC was introduced. The SPARC project was started at Berkeley and the MIPS project at Stanford. RISC stands for Reduced Instruction Set Computer. The ISA is composed of instructions that all have exactly the same size, usualy 32 bits. Thus they can be pre-fetched and pipelined succesfuly. All ALU instructions have 3 operands which are only registers. The only memory access is through explicit LOAD/STORE instructions.
Thus A = B + C will be assembled as:

```
LOAD  R1,A
LOAD  R2,B
ADD   R3,R1,R2
STORE C,R3
```

Although it takes 4 instructions we can reuse the values in the registers.

4. List the eight great ideas invented by computer architects(nov/dec2016)

1. Design for Moore's Law.

2. Use Abstraction to Simplify Design

3. Make the common case fast

4. Performance via parallelism

5. Performance via pipelining

6. Performance via prediction

7. Hierarchy of memories

8. Dependability via redundancy

5. How CPU execution time for a program is calculated?(Nov/Dec 2015)

Performance:

Response time or execution time: The total time required for the computer to complete a task, including disk accesses, memory

accesses, I/O activities, operating system overhead, CPU execution

time, and so on.

Throughput: The total amount of work done in a given time.

Bandwidth: The amount of data that can be carried from one point to another in a given time period (usually a second). This kind of bandwidth is usually expressed in bits (of data) per second (bps). Occasionally, it's expressed as bytes per second (Bps).

clock cycles per instruction (CPI): Average number of clock cycles per

instruction for a program or program fragment.

Performance:

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

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.

UNIT-II	ARITHMETIC OPERATIONS

1. Explain the sequential versions of multiplication algorithm and its hardware.(2014/2016/2017)

Multiplication:

Multiplicand		1000_{ten}
Multiplier	x	1001_{ten}
		1000
		0000
		0000
		1000
Product		1001000_{ten}

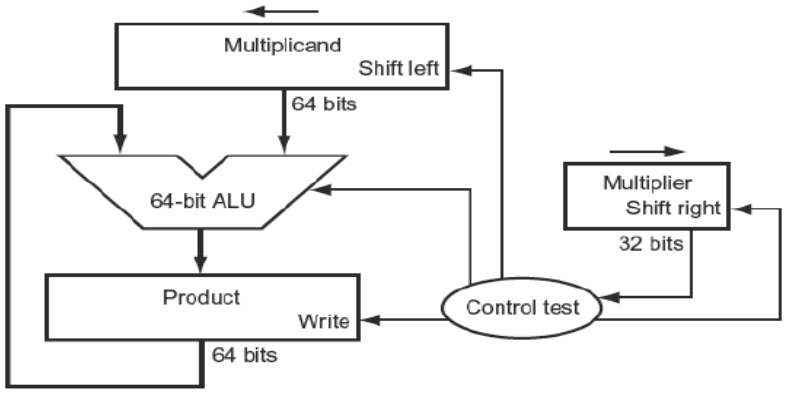
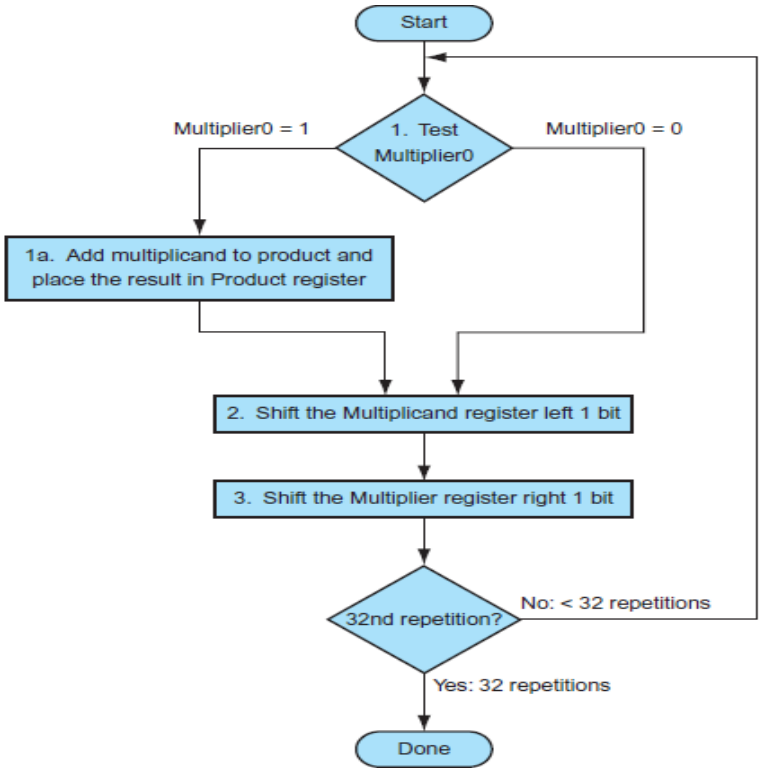


FIGURE **multiplication hardware.** The Multiplicand register, ALU, and Product register are all 64 bits wide, with only the Multiplier register containing 32 bits. The 32-bit multiplicand starts in the right half of the Multiplicand register and is shifted left 1 bit on each step. The multiplier is shifted in the opposite direction at each step. The algorithm starts with the product initialized to 0. Control decides when to shift the Multiplicand and Multiplier registers and when to write new values into the Product register.



2. Explain briefly about floating point addition algorithm and subtraction algorithm?

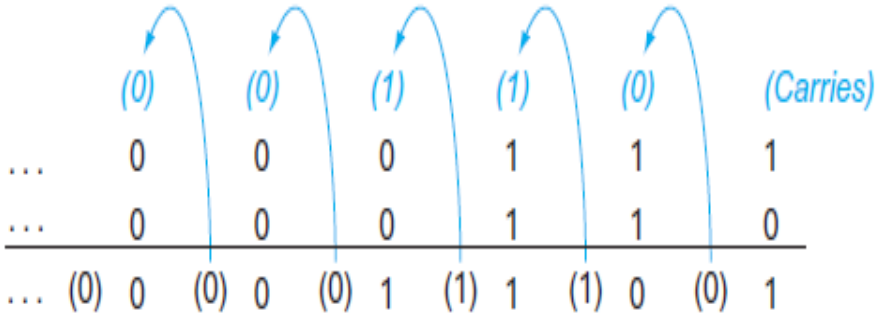
Addition:

Binary Addition and Subtraction

Let's try adding 6_{ten} to 7_{ten} in binary and then subtracting 6_{ten} from 7_{ten} in binary.

$$\begin{array}{ll} & 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0111_{two} = 7_{ten} \\ + & 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0110_{two} = 6_{ten} \\ \hline = & 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 1101_{two} = 13_{ten} \end{array}$$

The 4 bits to the right have all the action; Figure shows the sums and carries. The carries are shown in parentheses, with the arrows showing how they are passed.



Binary addition, showing carries from right to left. The rightmost bit adds 1 to 0, resulting in the sum of this bit being 1 and the carry out from this bit being 0. Hence, the operation for the second digit to the right is 0 + 1 + 1. This generates a 0 for this sum bit and a carry out of 1. The third digit is the sum of 1 + 1 + 1, resulting in a carry out of 1 and a sum bit of 1. The fourth bit is 1 + 0 + 0, yielding a 1 sum and no carry.

Subtraction:

$$\begin{array}{ll} & 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0111_{two} = 7_{ten} \\ - & 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0110_{two} = 6_{ten} \\ \hline = & 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_{two} = 1_{ten} \end{array}$$

or via addition using the two's complement representation of −6:

$$\begin{array}{ll} & 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0111_{two} = 7_{ten} \\ + & 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1010_{two} = -6_{ten} \\ \hline = & 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_{two} = 1_{ten} \end{array}$$

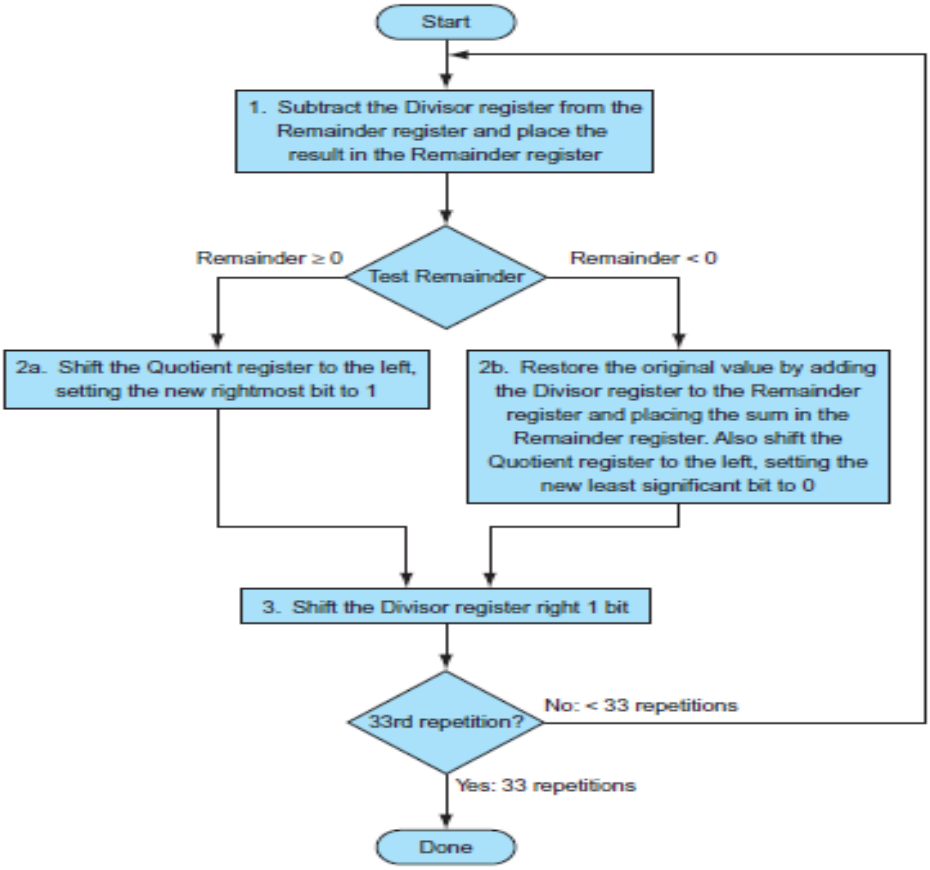
3. Discuss in details about division algorithm in details with diagram and control lines

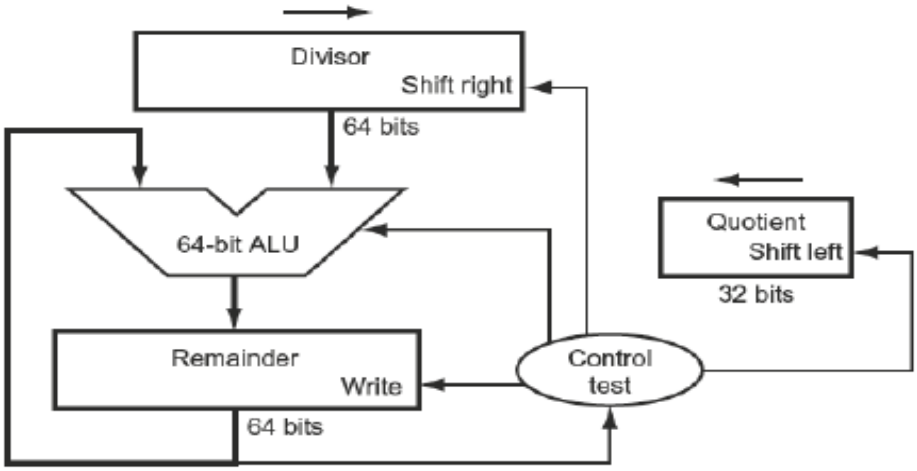
The example is dividing $1{,}001{,}010_{ten}$ by 1000_{ten}:

$$\begin{array}{rll} & 1001_{ten} & \text{Quotient} \\ \text{Divisor } 1000_{ten} & \overline{)\,1001010_{ten}} & \text{Dividend} \\ & -1000 & \\ & \overline{10} & \\ & 101 & \\ & 1010 & \\ & -1000 & \\ & \overline{10_{ten}} & \text{Remainder} \end{array}$$

Divide's two operands, called the dividend and divisor, and the result, called the quotient, are accompanied by a second result, called the remainder. Here is another way to express the relationship between the components:

$$\text{Dividend} = \text{Quotient} \times \text{Divisor} + \text{Remainder}$$





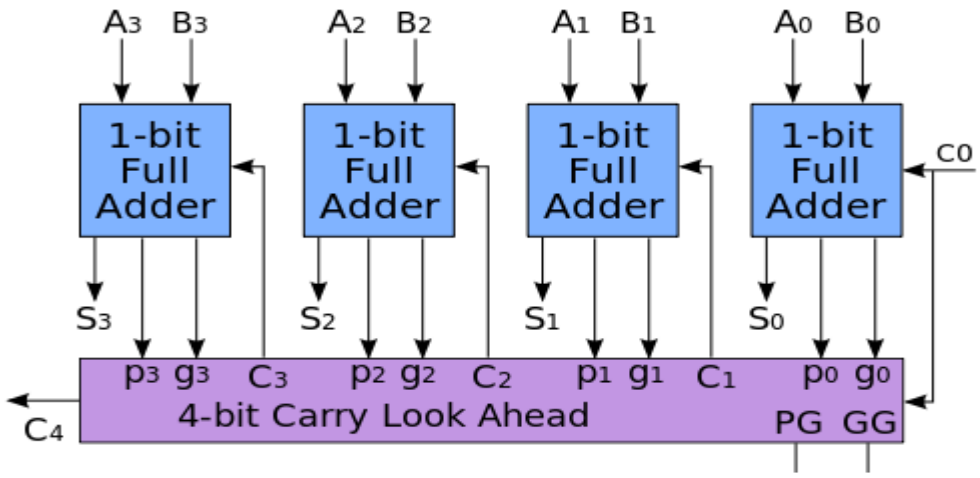
division hardware. The Divisor register, ALU, and Remainder register are all 64 bits wide, with only the Quotient register being 32 bits. The 32-bit divisor starts in the left half of the Divisor register and is shifted right 1 bit each iteration. The remainder is initialized with the dividend. Control decides when to shift the Divisor and Quotient registers and when to write the new value into the Remainder register.

Iteration	Step	Quotient	Divisor	Remainder
0	Initial values	0000	0010 0000	0000 0111
1	1: Rem = Rem – Div	0000	0010 0000	1110 0111
	2b: Rem < 0 ⇒ +Div, sll Q, Q0 = 0	0000	0010 0000	0000 0111
	3: Shift Div right	0000	0001 0000	0000 0111
2	1: Rem = Rem – Div	0000	0001 0000	1111 0111
	2b: Rem < 0 ⇒ +Div, sll Q, Q0 = 0	0000	0001 0000	0000 0111
	3: Shift Div right	0000	0000 1000	0000 0111
3	1: Rem = Rem – Div	0000	0000 1000	1111 1111
	2b: Rem < 0 ⇒ +Div, sll Q, Q0 = 0	0000	0000 1000	0000 0111
	3: Shift Div right	0000	0000 0100	0000 0111
4	1: Rem = Rem – Div	0000	0000 0100	0000 0011
	2a: Rem ≥ 0 ⇒ sll Q, Q0 = 1	0001	0000 0100	0000 0011
	3: Shift Div right	0001	0000 0010	0000 0011
5	1: Rem = Rem – Div	0001	0000 0010	0000 0001
	2a: Rem ≥ 0 ⇒ sll Q, Q0 = 1	0011	0000 0010	0000 0001
	3: Shift Div right	0011	0000 0001	0000 0001

FIGURE **Division example using the algorithm**

4. Explain in detail the principle of of carry look ahead adder. Show how 16 bit CLAs can be constructed from 4 bit adders. (2014/2016/2017)

A **carry-lookahead adder** (**CLA**) or **fast adder** is a type of adder used in digital logic. A carry-lookahead adder improves speed by reducing the amount of time required to determine carry bits. It can be contrasted with the simpler, but usually slower, *ripple carry adder* for which the carry bit is calculated alongside the sum bit, and each bit must wait until the previous carry bit have been calculated to begin calculating its own result and carry bits (see adder for detail on ripple carry adders). The carry-look ahead adder calculates one or more carry bits before the sum, which reduces the wait time to calculate the result of the larger value bits of the adder.



UNIT-III PROCESSOR AND CONTROL

1. Explain in details basic MIPS implementation with necessary multiplexers and control lines.

A Basic MIPS Implementation

We will be examining an implementation that includes a subset of the core MIPS instruction set:

- The memory-reference instructions *load word* (`lw`) and *store word* (`sw`)
- The arithmetic-logical instructions `add`, `sub`, `AND`, `OR`, and `slt`
- The instructions *branch equal* (`beq`) and *jump* (`j`), which we add last

This subset does not include all the integer instructions (for example, shift, multiply, and divide are missing), nor does it include any floating-point instructions.

An Overview of the Implementation

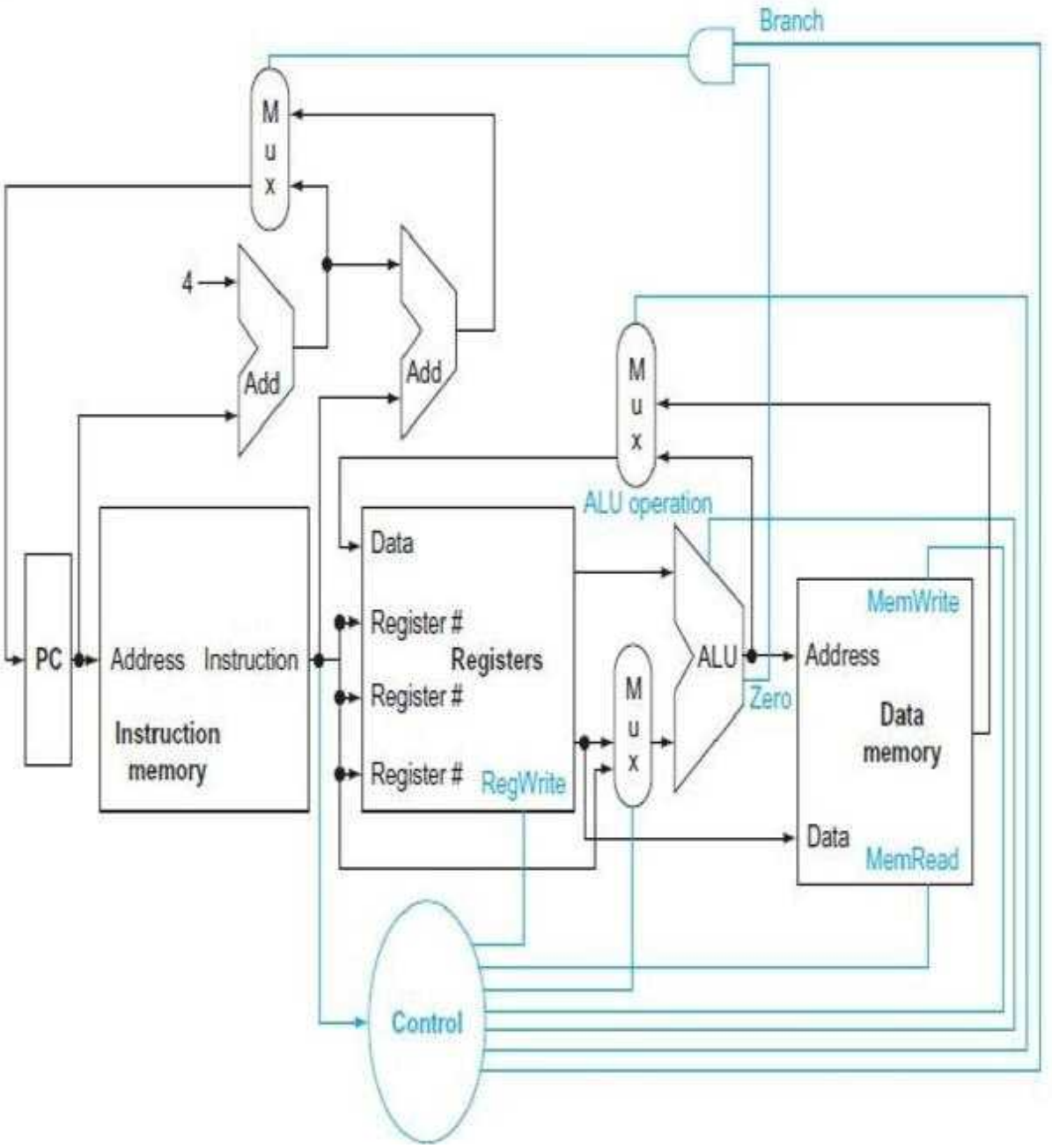
In Chapter 2, we looked at the core MIPS instructions, including the integer arithmetic-logical instructions, the memory-reference instructions, and the branch instructions. Much of what needs to be done to implement these instructions is the same, independent of the exact class of instruction. For every instruction, the first two steps are identical:

1. Send the *program counter* (PC) to the memory that contains the code and fetch the instruction from that memory.
2. Read one or two registers, using fields of the instruction to select the registers to read. For the load word instruction, we need to read only one register, but most other instructions require reading two registers.

After these two steps, the actions required to complete the instruction depend on the instruction class. Fortunately, for each of the three instruction classes (memory-reference, arithmetic-logical, and branches), the actions are largely the same, independent of the exact instruction. The simplicity and regularity of the MIPS instruction set simplifies the implementation by making the execution of many of the instruction classes similar.

For example, all instruction classes, except jump, use the arithmetic-logical unit (ALU) after reading the registers. The memory-reference instructions use the ALU for an address calculation, the arithmetic-logical instructions for the operation execution, and branches for comparison. After using the ALU, the actions required to complete various instruction classes differ. A memory-reference instruction will need to access the memory either to read data for a load or write data for a store. An arithmetic-logical or load instruction must write the data from the ALU or memory back into a register. Lastly, for a branch instruction, we may need to change the next instruction address based on the comparison; otherwise, the PC should be incremented by 4 to get the address of the next instruction.

The basic implementation of the MIPS subset, including the necessary multiplexors and control lines. The top multiplexor ("Mux") controls what value replaces the PC (PC + 4 or the branch destination address); the multiplexor is controlled by the gate that "ANDs" together the Zero output of the ALU and a control signal that indicates that the instruction is a branch. The middle multiplexor, whose output returns to the register file, is used to steer the output of the ALU (in the case of an arithmetic-logical instruction) or the output of the data memory (in the case of a load) for writing into the register file. Finally, the bottommost multiplexor is used to determine whether the second ALU input is from the registers (for an arithmetic-logical instruction or a branch) or from the offset field of the instruction (for a load or store). The added control lines are straightforward and determine the operation performed at the ALU, whether the data memory should read or write, and whether the registers should perform a write operation. The control lines are shown in color to make them easier to see.



2. Explain how the instruction pipeline works? What are the various situations where an instruction pipeline can stall? Illustrate with an example.(Nov/2009/2014/2015)

Data Hazards

Data hazards occur when the pipeline must be stalled because one step must wait for another to complete. Suppose you found a sock at the folding station for which no match existed. One possible strategy is to run down to your room and search through your clothes bureau to see if you can find the match. Obviously, while you are doing the search, loads must wait that have completed drying and are ready to fold as well as those that have finished washing and are ready to dry.

In a computer pipeline, data hazards arise from the dependence of one instruction on an earlier one that is still in the pipeline (a relationship that does not really exist when doing laundry). For example, suppose we have an add instruction followed immediately by a subtract instruction that uses the sum ($s0):

```
add   $s0, $t0, $t1
sub   $t2, $s0, $t3
```

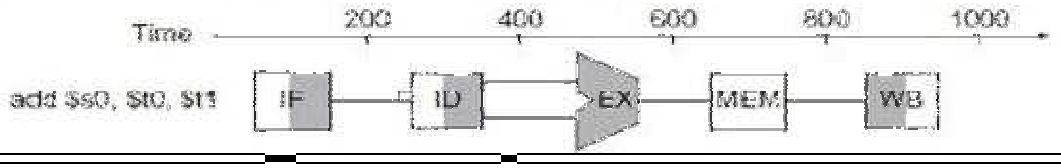
Without intervention, a data hazard could severely stall the pipeline. The add instruction doesn't write its result until the fifth stage, meaning that we would have to waste three clock cycles in the pipeline.

Although we could try to rely on compilers to remove all such hazards, the results would not be satisfactory. These dependences happen just too often and the delay is just too long to expect the compiler to rescue us from this dilemma.

The primary solution is based on the observation that we don't need to wait for the instruction to complete before trying to resolve the data hazard. For the code sequence above, as soon as the ALU creates the sum for the add, we can supply it as an input for the subtract. Adding extra hardware to retrieve the missing item early from the internal resources is called forwarding or bypassing.

Forwarding with Two Instructions

For the two instructions above, show what pipeline stages would be connected by forwarding. Use the drawing in Figure 4.28 to represent the datapath during the five stages of the pipeline. Align a copy of the datapath for each instruction similar to the laundry pipeline in Figure 4.25.



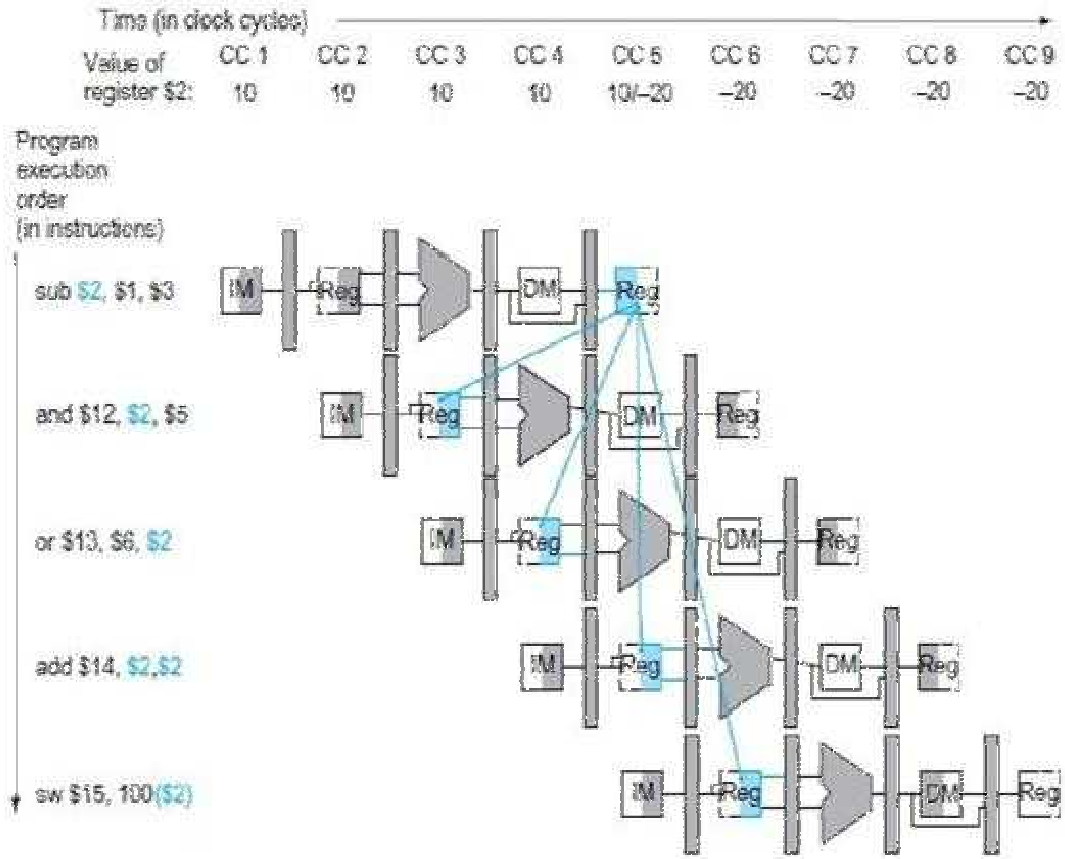
Graphical representation of the instruction pipeline, similar in spirit to the laundry pipeline in Figure 4.25. Here we use symbols representing the physical resources with the abbreviations for pipeline stages used throughout the chapter. The symbols for the five stages: *IF* for the instruction fetch stage, with the box representing instruction memory; *ID* for the instruction decode/register file read stage, with the drawing showing the register file being read; *EX* for the execution stage, with the drawing representing the ALU; *MEM* for the memory access stage, with the box representing data memory; and *WB* for the write-back stage, with the drawing showing the register file being written. The shading indicates the element is used by the instruction. Hence, MEM has a white background because add does not access the data memory. Shading on the right half of the register file or memory means the element is read in that stage, and shading of the left half means it is written in that stage. Hence the right half of ID is shaded in the second stage because the register file is read, and the left half of WB is shaded in the fifth stage because the register file is written.

3. What are hazard? Explain the methods for dealing with data hazards(2016/2017).

Let's look at a sequence with many dependences, shown in color:

```
sub   $2, $1,$3      # Register $2 written by sub
and   $12,$2,$5      # 1st operand($2) depends on sub
or    $13,$6,$2      # 2nd operand($2) depends on sub
add   $14,$2,$2      # 1st($2) & 2nd($2) depend on sub
sw    $15,100($2)    # Base ($2) depends on sub
```

The last four instructions are all dependent on the result in register $2 of the first instruction. If register $2 had the value 10 before the subtract instruction and −20 afterwards, the programmer intends that −20 will be used in the following instructions that refer to register $2.



Pipelined dependences in a five-instruction sequence using simplified datapaths to show the dependences. All the dependent actions are shown in color, and "CC 1" at the top of the figure means clock cycle 1. The first instruction writes into $2, and all the following instructions read $2. This register is written in clock cycle 5, so the proper value is unavailable before clock cycle 5. (A read of a register during a clock cycle returns the value written at the end of the first half of the cycle, when such a write occurs.) The colored lines from the top datapath to the lower ones show the dependences. Those that must go backward in time are *pipeline data hazards*.

UNIT-IV	PARALLELISM

1. Explain in details Flynn's classification of parallel hardware.(2014/2016)

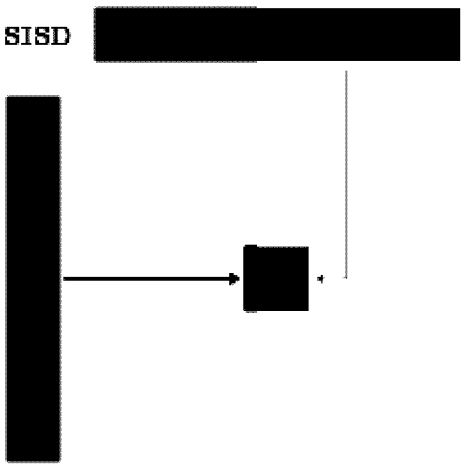
Classifications

- 1.1 Single instruction stream single data stream (SISD)
- 1.2 Single instruction stream, multiple data streams (SIMD)
- 1.3 Multiple instruction streams, single data stream (MISD)
- 1.4 Multiple instruction streams, multiple data streams (MIMD)
- 1.5 Single instruction, multiple threads (SIMT)

Single instruction stream single data stream (SISD)

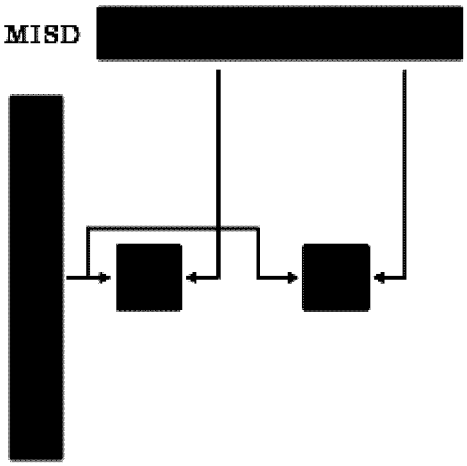
A sequential computer which exploits no parallelism in either the instruction or data streams. Single control unit (CU) fetches single instruction stream (IS) from memory. The CU then generates appropriate control signals to direct single processing element (PE) to operate on single data stream (DS) i.e., one operation at a time.

Examples of SISD architecture are the traditional uniprocessor machines like older personal computers (PCs; by 2010, many PCs had multiple cores) and mainframe computers.



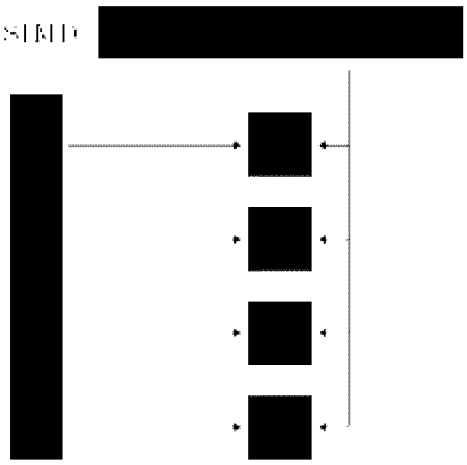
Multiple instruction streams, single data stream (MISD)

Multiple instructions operate on one data stream. This is an uncommon architecture which is generally used for fault tolerance. Heterogeneous systems operate on the same data stream and must agree on the result. Examples include the Space Shuttle flight control computer[4].



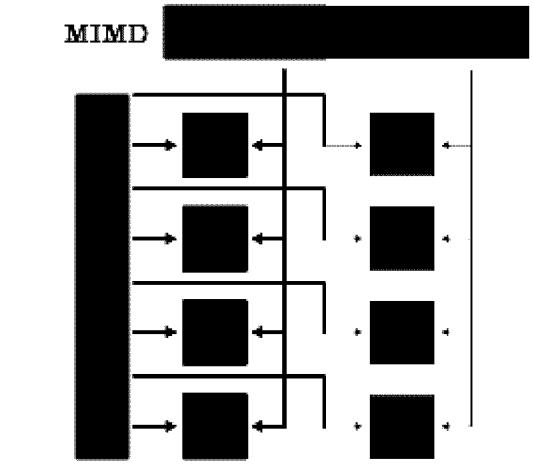
Single instruction stream, multiple data streams (SIMD)

A computer which exploits multiple data streams against a single stream to perform operations which may be naturally parallelized. For example, an array processor or graphics processing unit (GPU)



Multiple instruction streams, multiple data streams (MIMD)

Multiple autonomous processors simultaneously executing different instructions on different data. MIMD architectures include multi-core superscalar processors, and distributed systems, using either one shared memory space or a distributed memory space.



Single instruction, multiple threads (SIMT)

Single instruction, multiple threads (SIMT) is an execution model used in parallel computing where single instruction, multiple data (SIMD) is combined with multithreading. This is not originally part of Flynn's taxonomy but a proposed addition.

These four architectures are shown below visually. Each processing unit (PU) is shown for a uni-core or multi-core computer:

2. Draw a neat sketch of memory hierarchy and explain the need of cache memory.(2014/2015)

Computer memory can be classified in the below given hierarchy:

1) Internal register: Internal register in a CPU is used for holding variables and temporary results. Internal registers have a very small storage; however they can be accessed instantly. Accessing data from the internal register is the fastest way to access memory.

2) Cache: Cache is used by the CPU for memory which is being accessed over and over again. Instead of pulling it every time from the main memory, it is put in cache for fast access. It is also a smaller memory, however, larger than internal register.

Cache is further classified to L1, L2 and L3:

a) L1 cache: It is accessed without any delay.

b) L2 cache: It takes more clock cycles to access than L1 cache.

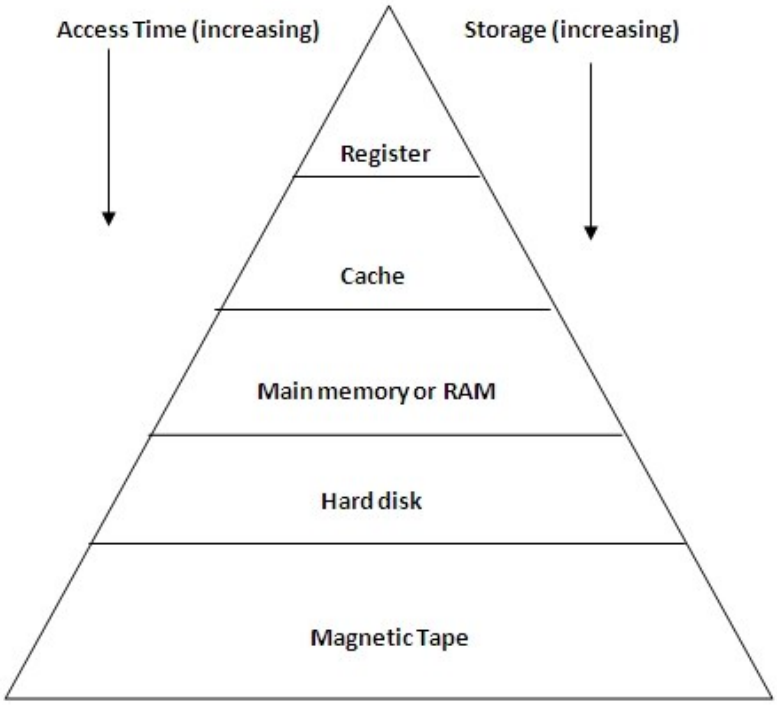
c) L3 cache: It takes more clock cycles to access than L2 cache.

3) Main memory or RAM (Random Access Memory): It is a type of the computer memory and is a hardware component. It can be increased provided the operating system can handle it. Typical PCs these days use 8 GB of RAM. It is accessed slowly as compared to cache.

4) Hard disk: A hard disk is a hardware component in a computer. Data is kept permanently in this memory. Memory from hard disk is not directly accessed by the CPU, hence it is slower. As compared with RAM, hard disk is cheaper per bit.

5) Magnetic tape: Magnetic tape memory is usually used for backing up large data. When the system needs to access a tape, it is first mounted to access the data. When the data is accessed, it is then unmounted. The memory access time is slower in magnetic tape and it usually takes few minutes to access a tape.

Below given figure shows the hierarchy of computer memory:



3. Explain various mechanisms of mapping main memory address into cache memory addresses**.(2014/2015)**

Three techniques can be used:

1. Direct
2. Associative
3. Set Associative.

DIRECT MAPPING

- The simplest technique, known as direct mapping, maps each block of main memory into only one possible cache line. The mapping is expressed as

  i = j modulo m

  where

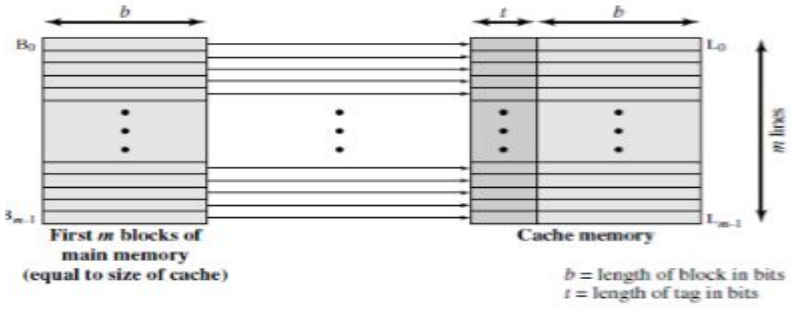
  i cache line number

  j main memory block number

  m number of lines in the cache

- Figure 1 (a) shows the mapping for the first m blocks of main memory. Each block of main memory maps into one unique line of the cache.
- The next m blocks of main memory map into the cache in the same fashion; that is, block Bm of main memory maps into line L0 of cache, block Bm_1 maps into line L1, and so on.
- The mapping function is easily implemented using the main memory address. Figure 2 illustrates the general mechanism. For purposes of cache access, each main memory address can be viewed as consisting of three fields.
- The least significant w bits identify a unique word or byte within a block of main memory; in most contemporary machines, the address is at the byte level. The remaining s bits specify one of the 2s blocks of main memory. The cache logic interprets these s bits as a tag of s – r bits (most significant portion) and a line field of r bits. This latter field identifies one of the m = 2r lines of the cache. To summarize,

Address length = $(s + w)$ bits
Number of addressable units = 2^{s+w} words or bytes
Block size = line size = 2^w words or bytes
Number of blocks in main memory = $\frac{2^{s+w}}{2^w} = 2^s$
Number of lines in cache = $m = 2^r$
Size of cache = 2^{r+w} words or bytes
Size of tag = $(s - r)$ bits



The effect of this mapping is that blocks of main memory are assigned to lines of the cache as follows:

Cache line	Main memory blocks assigned
0	$0, m, 2m, \ldots, 2^s - m$
1	$1, m+1, 2m+1, \ldots, 2^s - m + 1$
⋮	⋮
$m-1$	$m-1, 2m-1, 3m-1, \ldots, 2^s - 1$

- Thus, the use of a portion of the address as a line number provides a unique mapping of each block of main memory into the cache.
- When a block is actually read into its assigned line, it is necessary to tag the data to distinguish it from other blocks that can fit into that line. The most significant s - r bits serve this purpose.

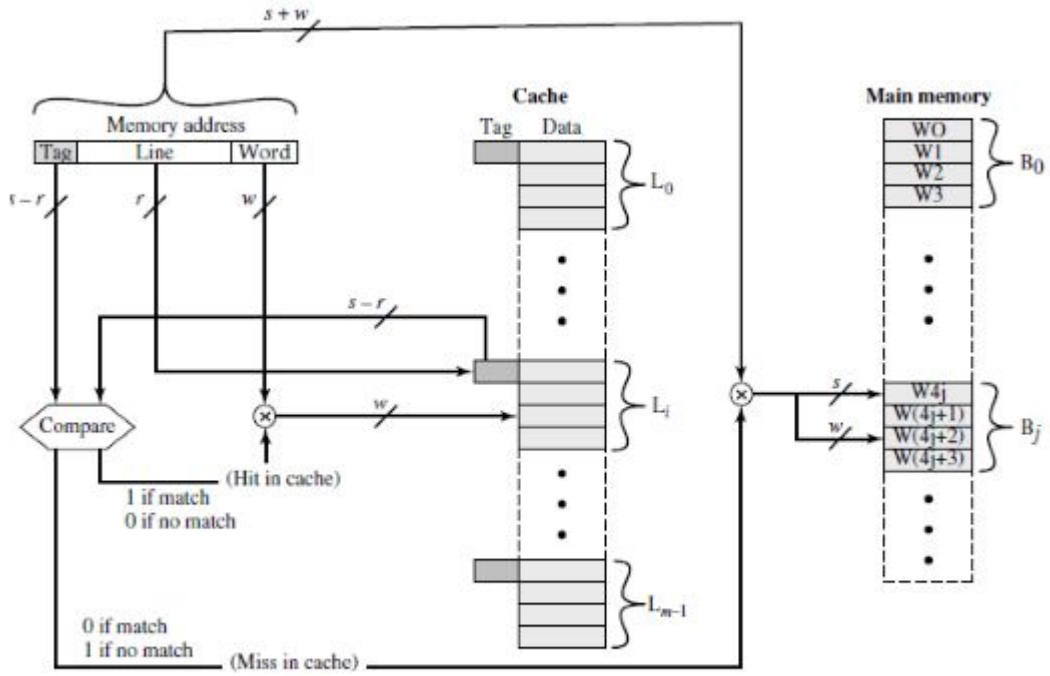


Figure 2

- The direct mapping technique is simple and inexpensive to implement. Its main disadvantage is that there is a fixed cache location for any given block.
- Thus, if a program happens to reference words repeatedly from two different blocks that map into the same line, then the blocks will be continually swapped in the cache, and the hit ratio will be low (a phenomenon known as thrashing).

ASSOCIATIVE MAPPING

- Associative mapping overcomes the disadvantage of direct mapping by permitting each main memory block to be loaded into any line of the cache Figure 1(b).
- In this case, the cache control logic interprets a memory address simply as a Tag and a Word field. The Tag field uniquely identifies a block of main memory.
- To determine whether a block is in the cache, the cache control logic must simultaneously examine every line's tag for a match. Figure 3 illustrates the logic. Note that no field in the address corresponds to the line number, so that the number of lines in the cache is not determined by the address format. To summarize,

Address length = $(s + w)$ bits
Number of addressable units = 2^{s+w} words or bytes
Block size = line size = 2^w words or bytes
Number of blocks in main memory = $\frac{2^{s+w}}{2^w} = 2^s$
Number of lines in cache = undetermined
Size of tag = s bits

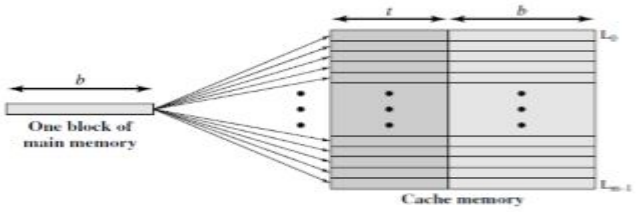


Figure 1 (b)

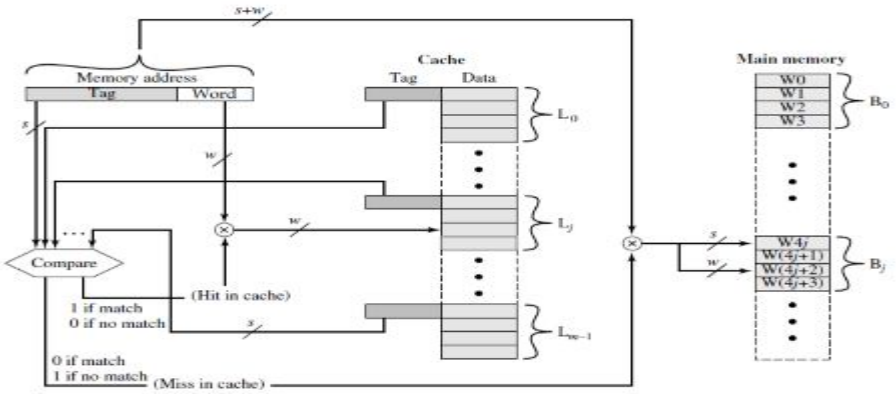


Figure 3

- With associative mapping, there is flexibility as to which block to replace when a new block is read into the cache.
- Replacement algorithms, discussed later in this section, are designed to maximize the hit ratio. The principal disadvantage of associative mapping is the complex circuitry required to examine the tags of all cache lines in parallel.

SET-ASSOCIATIVE MAPPING

- Set-associative mapping is a compromise that exhibits the strengths of both the direct and associative approaches while reducing their disadvantages.
- In this case, the cache consists of a number sets, each of which consists of a number of lines. The relationships are

$$m = n * k$$
$$i = j \text{ modulo } n$$

Where

i = cache set number
j = main memory block number
m = number of lines in the cache
v = number of sets
k = number of lines in each set

- This is referred to as k-way set-associative mapping. With set-associative mapping, block Bj can be mapped into any of the lines of set j.
- Figure 4 (a) illustrates this mapping for the first blocks of main memory. As with associative mapping, each word maps into multiple cache lines.
- For set-associative mapping, each word maps into all the cache lines in a specific set, so that main memory block B0 maps into set 0, and so on.
- Thus, the set-associative cache can be physically implemented as associative caches. It is also possible to implement the set-associative cache a k direct mapping caches, as shown in Figure 4 (b).
- Each direct-mapped cache is referred to as a way, consisting of lines.The first lines of main memory are direct mapped into the lines of each way; the next group of lines of main memory are similarly mapped, and so on.
- The direct-mapped implementation is typically used for small degrees of associativity (small values of k) while the associative- mapped implementation is typically used for higher degrees of associativity.

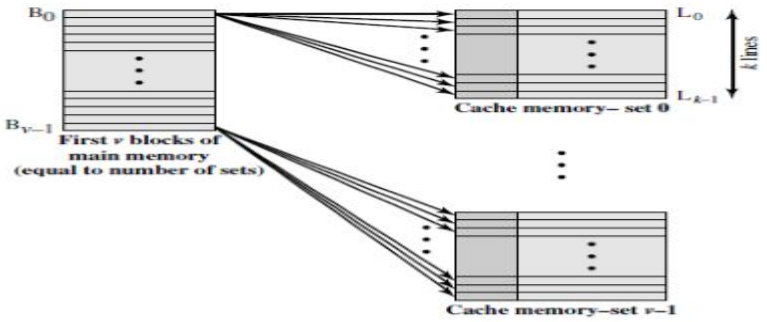


Figure 4 (a)

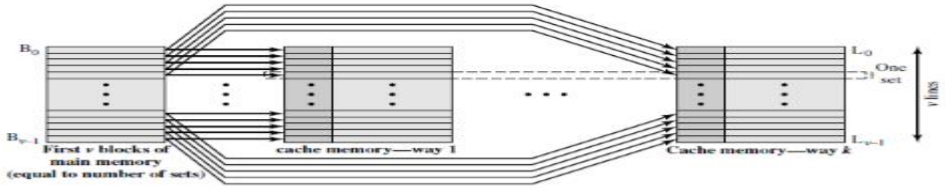


Figure 4 (b)

- For set-associative mapping, the cache control logic interprets a memory address as three fields: Tag, Set, and Word.
- The d set bits specify one of v = 2d sets. The s bits of the Tag and Set fields specify one of the 2s blocks of main memory.
- Figure 5 illustrates the cache control logic.With fully associative mapping, the tag in a memory address is quite large and must be compared to the tag of every line in the cache. With k-way set-associative mapping, the tag in a memory address is much smaller and is only compared to the k tags within a single set.

5. Explain in detail about hardware Multithreading.

multithreading is the ability of a central processing unit (CPU) or a single core in a multi-core processor to execute multiple processes or threads concurrently, appropriately supported by the operating system. This approach differs from multiprocessing, as with multithreading the processes and threads share the resources of a single or multiple cores: the computing units, the CPU caches, and the translation lookaside buffer (TLB).

Where multiprocessing systems include multiple complete processing units, multithreading aims to increase utilization of a single core by using thread-level as well as instruction-level parallelism. As the two techniques are complementary, they are sometimes combined in systems with multiple multithreading CPUs and in CPUs with multiple multithreading cores.

Advantages

If a thread gets a lot of cache misses, the other threads can continue taking advantage of the unused computing resources, which may lead to faster overall execution as these resources would have been idle if only a single thread were executed. Also, if a thread cannot use all the computing resources of the CPU (because instructions depend on each other's result), running another thread may prevent those resources from becoming idle.

If several threads work on the same set of data, they can actually share their cache, leading to better cache usage or synchronization on its values.

Disadvantages

Multiple threads can interfere with each other when sharing hardware resources such as caches or translation lookaside buffers (TLBs). As a result, execution times of a single thread are not improved but can be degraded, even when only one thread is executing, due to lower frequencies or additional pipeline stages that are necessary to accommodate thread-switching hardware.

Types of multithreading:

Coarse-grained multithreading

The simplest type of multithreading occurs when one thread runs until it is blocked by an event that normally would create a long-latency stall. Such a stall might be a cache miss that has to access off-chip memory, which might take hundreds of CPU cycles for the data to

return. Instead of waiting for the stall to resolve, a threaded processor would switch execution to another thread that was ready to run. Only when the data for the previous thread had arrived, would the previous thread be placed back on the list of ready-to-run threads.

For example:

1. Cycle *i*: instruction *j* from thread *A* is issued.
2. Cycle *i* + 1: instruction *j* + 1 from thread *A* is issued.
3. Cycle *i* + 2: instruction *j* + 2 from thread *A* is issued, which is a load instruction that misses in all caches.
4. Cycle *i* + 3: thread scheduler invoked, switches to thread *B*.
5. Cycle *i* + 4: instruction *k* from thread *B* is issued.
6. Cycle *i* + 5: instruction *k* + 1 from thread *B* is issued.

Conceptually, it is similar to cooperative multi-tasking used in real-time operating systems, in which tasks voluntarily give up execution time when they need to wait upon some type of the event. This type of multithreading is known as block, cooperative or coarse-grained multithreading.

The goal of multithreading hardware support is to allow quick switching between a blocked thread and another thread ready to run. To achieve this goal, the hardware cost is to replicate the program visible registers, as well as some processor control registers (such as the program counter). Switching from one thread to another thread means the hardware switches from using one register set to another; to switch efficiently between active threads, each active thread needs to have its own register set. For example, to quickly switch between two threads, the register hardware needs to be instantiated twice.

Additional hardware support for multithreading allows thread switching to be done in one CPU cycle, bringing performance improvements. Also, additional hardware allows each thread to behave as if it were executing alone and not sharing any hardware resources with other threads, minimizing the amount of software changes needed within the application and the operating system to support multithreading.

Many families of microcontrollers and embedded processors have multiple register banks to allow quick context switching for interrupts. Such schemes can be considered a type of block multithreading among the user program thread and the interrupt threads.[*citation needed*]

Interleaved multithreading

The purpose of interleaved multithreading is to remove all data dependency stalls from the execution pipeline. Since one thread is relatively independent from other threads, there is less chance of one instruction in one pipelining stage needing an output from an older instruction in the pipeline. Conceptually, it is similar to preemptive multitasking used in operating systems; an analogy would be that the time slice given to each active thread is one CPU cycle.

For example:

1. Cycle *i* + 1: an instruction from thread *B* is issued.
2. Cycle *i* + 2: an instruction from thread *C* is issued.

This type of multithreading was first called barrel processing, in which the staves of a barrel represent the pipeline stages and their executing threads. Interleaved, preemptive, fine-grained or time-sliced multithreading are more modern terminology.

In addition to the hardware costs discussed in the block type of multithreading, interleaved multithreading has an additional cost of each pipeline stage tracking the thread ID of the instruction it is processing. Also, since there are more threads being executed concurrently in the pipeline, shared resources such as caches and TLBs need to be larger to avoid thrashing between the different threads.

Simultaneous multithreading

The most advanced type of multithreading applies to superscalar processors. Whereas a normal superscalar processor issues multiple instructions from a single thread every CPU cycle, in simultaneous multithreading (SMT) a superscalar processor can issue instructions from multiple threads every CPU cycle. Recognizing that any single thread has a limited amount of instruction-level parallelism, this type of multithreading tries to exploit parallelism available across multiple threads to decrease the waste associated with unused issue slots.

For example:

1. Cycle *i*: instructions *j* and *j* + 1 from thread *A* and instruction *k* from thread *B* are simultaneously issued.
2. Cycle *i* + 1: instruction *j* + 2 from thread *A*, instruction *k* + 1 from thread *B*, and instruction *m* from thread *C* are all simultaneously issued.
3. Cycle *i* + 2: instruction *j* + 3 from thread *A* and instructions *m* + 1 and *m* + 2 from thread *C* are all simultaneously issued.

To distinguish the other types of multithreading from SMT, the term "temporal multithreading" is used to denote when instructions from only one thread can be issued at a time.

In addition to the hardware costs discussed for interleaved multithreading, SMT has the additional cost of each pipeline stage tracking the thread ID of each instruction being processed. Again, shared resources such as caches and TLBs have to be sized for the large number of active threads being processed.

UNIT V

1. Explain about Multicore Processors.(2016/2015)

A **multi-core** CPU is a computer processor which has two or more sections. Each section of the chip executes instructions as if it was a separate computer. The actual processors are still on one chip. On this chip every core looks mostly like the other. They are several mostly independent cores which work together in parallel. A **dual-core** processor is a multi-core processor with two independent microprocessors. A quad-core processor is a multi-core processor with four independent microprocessors. As you might be able to tell from the prefix, the name of the processor is based on the number of the microprocessors on the chip.

Until 2005 single-core processors outnumbered multi-core processors.[*source?*] In the years before there were only multi-core solutions used in individual cases.[*source?*] In the majority of cases they enhanced the frequency. But at a frequency about 4 GHz the CPU would get too hot and take a lot of electricity. This was the point when multi-core processors became more important. Therefore the demand for multi-core processors increased. In the second half of 2006 the best processors were dual-core processors. Since 2006 the development has gone on, so that the new processors get four or more independent microprocessors. Today, single-core processors are not used in new personal computers, but they remain popular in embedded systems.

Advantages

- Having a multi-core processor in a computer means that it will work faster for certain programs.
- The computer may not get as hot when it is turned on.
- The computer needs less power because it can turn off some sections if they aren´t needed.
- More features can be added to the computer.
- The signals between different CPUs travel shorter distances, therefore they degrade less.

Disadvantages

- They do not work at twice the speed as a normal processor. They get only 60-80% more speed.
- The speed that the computer works at depends on what the user is doing with it.
- They cost more than single core processors.
- They are more difficult to manage thermally than lower-density single-core processors.
- Not all operating systems support more than one core.
- Operating systems compiled for a multi-core processor will run slightly slower on a single-core processor.

Operating System Support

The following operating systems support multi-core processors

- Microsoft Windows (Windows XP or newer)
- Linux
- Mac OS X
- Most BSD-based systems
- Solaris

2. What is virtual memory? Explain the steps involved in virtual memory address translation.(2013/2014/2015)

A virtual address is a binary number in virtual memory that enables a process to use a location in primary storage (main memory) independently of other processes and to use more space than actually exists in primary storage by temporarily relegating some contents to a hard disk or internal flash drive

Virtual Memory Address Translation

In a system with virtual memory the main memory can be viewed as a cache for the disk, which serves as the lower-level store. Due to the enormous difference between memory access times and disk access times, a fully-associative caching scheme is used. That is, the entire main memory is a single set - any page can be placed anywhere in main memory. This makes the set field of the address vanish. All that remains is a tag and an offset.

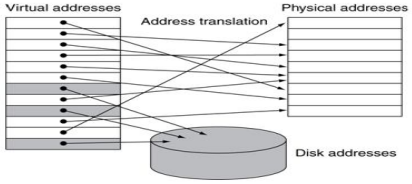
The offset field identifies a particular location within page or frame.

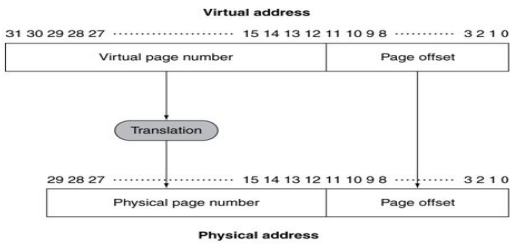
The tag field identifies a page in the logical address space.

Since the tag field just identifies a page it is usually called the page number field.

Virtual Address Translation - 1

1. Extract the page number from the virtual address.
2. Extract the offset from the virtual address
3. Translate the page number into a physical page frame number using the page table.
   - Look up the page number in the page table (using the virtual page number as an index)





Logical Addresses

With a virtual memory system, the main memory can be viewed as a local store for a cache level whose lower level is a disk. Since it is fully associative there is no need for a set field. The address just decomposes into an offset field and a page number field. The number of bits in the offset field is determined by the page size. The remaining bits are the page number.

An Example

A computer uses 32-bit byte addressing. The computer uses paged virtual memory with 4KB pages. Calculate the number of bits in the page number and offset fields of a logical address.

Page Tables

Virtual memory address translation uses a page table. In the diagram to the left, a page table is represented by the large box. It is a structured array in memory. It is indexed by page number.

Each page table entry contains information about a single page. Part of this information is a frame number (green) — where the page is located in physical memory. In addition there are control bits (blue) for controlling the translation process. Address translation concatenates the frame number with the offset part of a logical address to form a physical address.

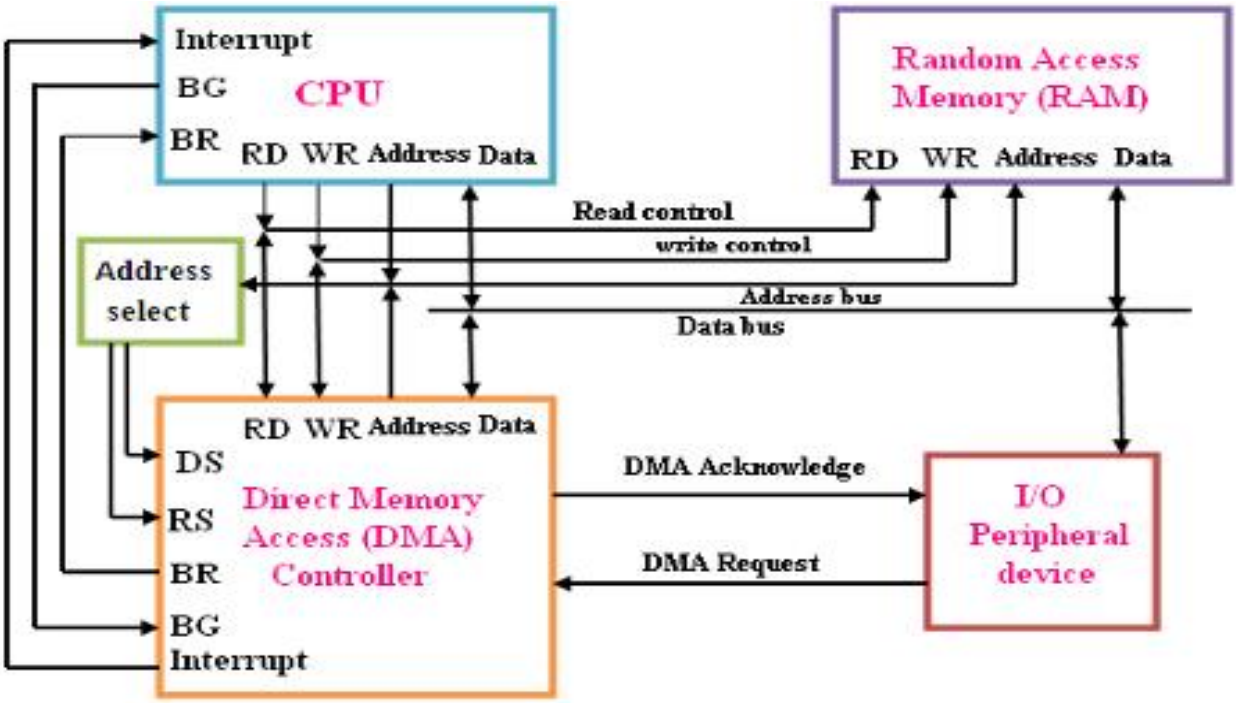
A page table base register (PTBR) holds the base address for the page table of the current process. It is a processor register that is managed by the operating system

3. Explain the use of DMA controllers in a computer system with a neat diagram.(2014/2015/2016)

DMA stands for "**Direct Memory Access**" and is a method of transferring data from the computer's RAM to another part of the computer without processing it using the CPU. While most data that is input or output from your computer is processed by the CPU, some data does not require processing, or can be processed by another device.

In these situations, DMA can save processing time and is a more efficient way to move data from the computer's memory to other devices. In order for devices to use direct memory access, they must be assigned to a DMA channel. Each type of port on a computer has a set of DMA channels that can be assigned to each connected device. For example, a PCI controller and a hard drive controller each have their own set of DMA channels.

DMA DIAGRAM:



4. Explain the IOP Organization and communication between CPU and IOP.(2015/2016/2014)

IOP is a processor with direct memory access capability that communicates with I/O devices. In this configuration, the computer system can be divided into a memory unit, and a number of processors comprised of CPU and one or more IOPs. IOP is similar to CPU except that it is designed to handle the details of I/O processing. Unlike DMA controller which is setup completely by the CPU, IOP can fetch and execute its own instructions. IOP instructions are designed specifically to facilitate I/O transfers. Instructions that are read from memory by an IOP are called commands to differ them from instructions read by CPU . The command words constitute the program for the IOP. The CPU informs the IOP where to find commands in memory when it is time to execute the I/O program.

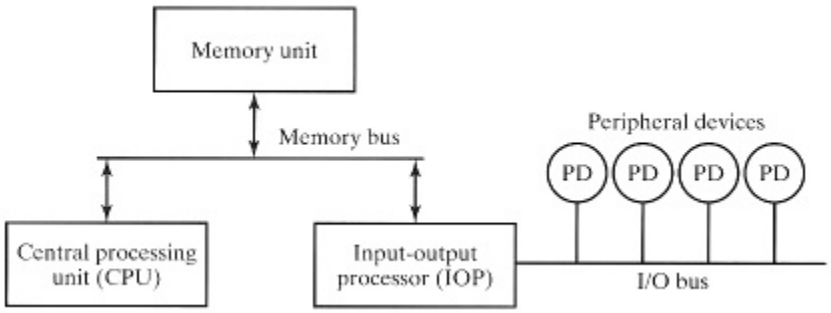


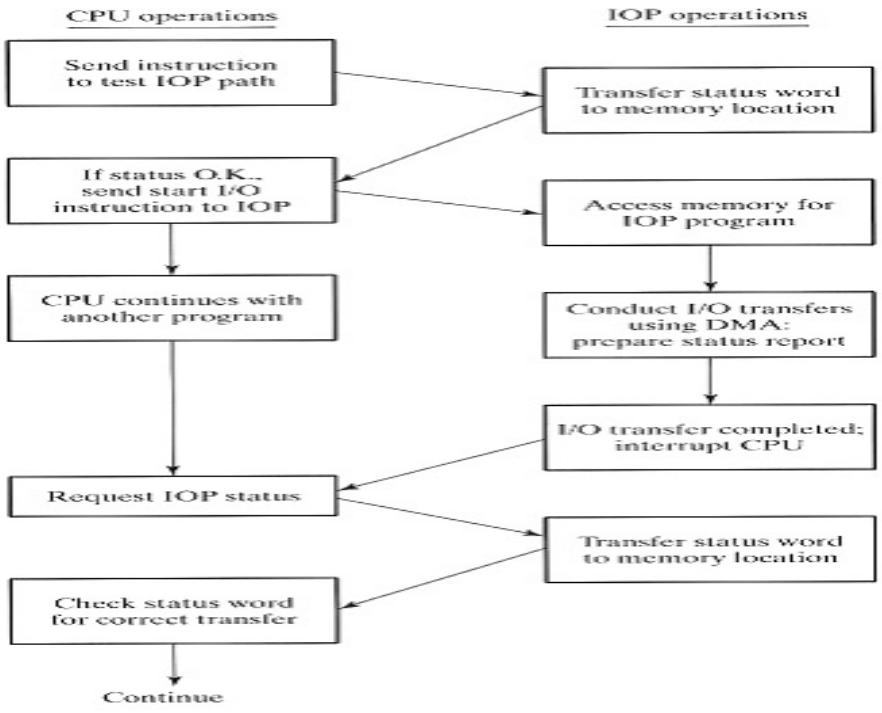
fig. I/O processor

The memory occupies a central position and can communicate with each processor by means of DMA. CPU is usually assigned the task of initiating the I/O program, from the non;IOP operates independent of the CPU and continues to transfer data from external devices and memory.

CPU-IO Communication

Communication between the CPU and IOP may take different forms depending on the particular computer used.Mostly, memory unit acts as a memory center where each processor leaves information for the other.

CPU sends an instruction to test the IOP path. The IOP responds by inserting a status word in memory for the CPU to check. The bits of the status word indicate the condition of IOP and I/O device ("IOP overload condition", "device busy with another transfer" etc). CPU then checks status word to decide what to do next . If all is in order, CPU sends the instruction to start the I/O transfer. The memory address received with this instruction tells the IOP where to find its program. CPU may continue with another program while the IOP is busy with the I/O program. When IOP terminates the transfer (using DMA), it sends an interrupt request to CPU. The CPU responds by issuing an instruction to read the status from the IOP and IOP then answers by placing the status report into specified memory location. By inspecting the bits in the status word, CPU determines whether the I/O operation was completed satisfactorily and the process is repeated again.



5. Explain the used of vectored interrupts in processors. Why is priority handling desired in interrupts controllers? How do the different priority schemes work? (n2014)

When a Process is executed by the CPU and when a user Request for another Process then this will create disturbance for the Running Process. This is also called as the **Interrupt**.

Interrupts can be generated by User, Some Error Conditions and also by Software's and the hardware's. But CPU will handle all the Interrupts very carefully because when Interrupts are generated then the CPU must handle all the Interrupts Very carefully means the CPU will also Provides Response to the Various Interrupts those are generated. So that When an interrupt has Occurred then the CPU will handle by using the Fetch, decode and Execute Operations.

Types of Interrupts

Generally there are three types o Interrupts those are Occurred For Example

1) Internal Interrupt

2) Software Interrupt.

3) External Interrupt.

The External Interrupt occurs when any Input and Output Device request for any Operation and the CPU will Execute that instructions first For Example When a Program is executed and when we move the Mouse on the Screen then the CPU will handle this External interrupt first and after that he will resume with his Operation.

The Internal Interrupts are those which are occurred due to Some Problem in the Execution For Example When a user performing any Operation which contains any Error and which contains any type of Error.

So that Internal Interrupts are those which are occurred by the Some Operations or by Some Instructions and the Operations those are not Possible but a user is trying for that Operation.

And The Software Interrupts are those which are made some call to the System for Example while we are Processing Some Instructions and when we wants to Execute one more Application Programs.