**A non-pipeline system takes 50ns to process a task. The same task can be processed in six-segment pipeline with a clockcycle of 10ns. Determine approximately the speedup ratio of the pipeline for 500 tasks.**

**a)6 b)4.95 c)5.7 d)5.5**

**Explanation:-**

**FIRST METHOD:-**

time for non pipeline system 1 task = 50 ns

so 500 tasks = 500x50 = 25000 ns

time with k(=6) segment pipeline

first task 6 x 10 ns subsequent task =10 ns so 499x10 =4990

total =60 + 4990 = 5050

Speedup ratio = time taken without pipeline / time with pipeline

=25000/5050 =4.9

**SECOND METHOD:-**

Number of segment pipeline k = 6

Time period of 1 clock cycle tp = 10ns

Total time required to complete n tasks in k segment pipeline with tp clock cycle time:

= ( k + n − 1 )tp

= ( 6 + 500 − 1 )10 = 5050ns

Speedup ratio = time taken without pipeline / time with pipeline

=25000 / 5050 =4.9

**Amdahl's Law Example**

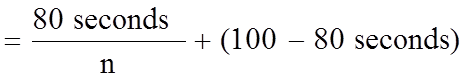
***1) Suppose a program runs in 100 seconds on a machine, with multiply responsible for 80 seconds of this time. How much do we have to improve the speed of multiplication if we want the program to run 4 times faster?***

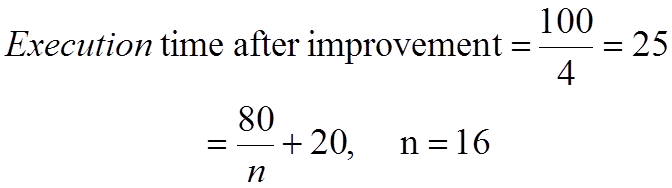
Explanation 1:-

100/4 = 20 + 80/n

n = 16

Explanation 2:-

Execution time after improvement= 



How about 5 times faster?

ans : Not possible

***Ques:- Application execution time is 20sec. 12 seconds are spent performing add operations. If we improve the add operation to run twice as fast, how much faster will the application run?***

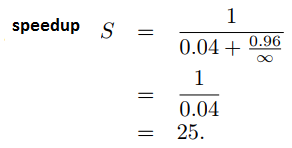
***which speedup could be achieved according to amdhal law for infinite number of processor if 5% of the program is sequential and remaining is ideally parallel?***

Ans : 20

**Which speedup could be achieved according to Amdahl’s law if an “infinite number” of processors is available but 4% of a program is sequential and the remaining part is ideally parallel?**

(a) Infinite speedup (b) 4 (c) 25 (d) 50 (e) None of the answers above is correct.

Explanation:-



***QUES:- Amdahl’s law states that the maximum speedup S achievable by a parallel computer with ‘p’ processors is given by:***

(A) S≤f+(1 – f)/p

(B) S≤f/p+(1-f)

(C) S≤1/[f+(1 – f)/p]

(D) S≤1/[1 – f+f/p] (yes)

***QUES:- The data type created by the data abstraction process is called***

(A) class (B) structure (C) abstract data type(YES) (D) user defined data type

***QUES:- The implicit return type of a constructor is***

A. void B. A class object in which it is defined.(yes)

C. There is no return type. D. None of the above

***QUES:- Implicit return type of a class constructor is :***

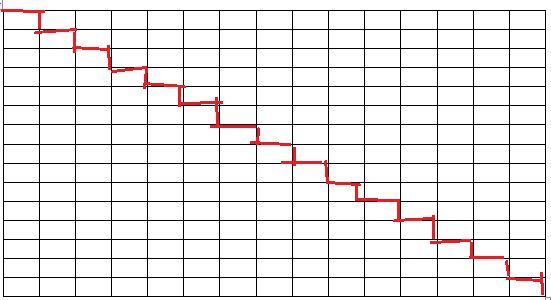
(1) not of class type itself (2) class type itself(yes)

(3) a destructor of class type (4) a destructor not of class type

***QUES:- A multicomputer with 256 CPUs is organized as 16 × 16 grid. What is the worst case delay (in hops) that a message might have to take ?***

(1) 16 (2) 15 (3) 32 (4) 30

**Explanation:-**



In the 16X16 CPU case, the worst-case delay happens when message passes through longest path which is the diagonal of the upper right corner to lower left corner or upper left corner to lower right corner.

Longest path is indicated in red color.

It passes through 2(N-1) = 2(16-1) = 30 hops

***QUES:- Suppose that the time to do a null remote procedure call (RPC) (i.e. 0 data bytes) is 1.0 msec, with an additional 1.5 msec for every 1K of data. How long does it take to read 32 K from the file server as 32 1K RPCs ? (ugc -2016 – july)***

(1) 49 msec (2) 80 msec(yes) (3) 48 msec (4) 100 msec

A single 32K RPC takes 1.5\*32 + 1.0 = 49.0 msec

32 1K RPCs take 1.5\*32 + 1.0\*32 = 80.0 msec

***Which of the following differentiates between overloaded functions and overridden functions ?***

(A) Overloading is a dynamic or runtime binding and overridden is a static or compile time binding.

(B) Overloading is a static or compile time binding and overriding is dynamic or runtime binding.(yes)

(C) Redefining a function in a friend class is called overloading, while redefining a function in a derived class is called as overridden function.

(D) Redefining a function in a derived class is called function overloading, while redefining a function in a friend class is called function overriding.

***The goal of operator overloading is***

(A) to help the user of a class(yes) (B) to help the developer of a class

(C) to help define friend function (D) None of the above

***The speed up of a pipeline processing over an equivalent non-pipeline processing is defined by the ratio :***

(A) S = n tn/ (k + n – 1)tp(yes)

(B) S = n tn/ (k + n + 1)tp

(C) S = n tn/ (k – n + 1)tp

(D) S = (k + n – 1)tp/ n tn

Where n → no. of tasks; tn→ time of completion of each task;

k → no. of segments of pipeline; tp→ clock cycle time;

S → speed up ratio;

**Without pipeline** one task needs tn time.

* n tasks need ntn time.

**With pipeline,**

* First task needs  k cycles to finish. So time will be k tp
* Other n-1 tasks needs tp time only to finish.
* Total time = (k+ n -1 ) tp

**Speed up** = T pipeline / T without pipeline = n t n / (k+n-1) t p

***A set of processors P1, P2, ......, Pk can execute in parallel if Bernstein's conditions are satisfied on a pair wise basis; that is P1 || P2 || P3 || ..... || Pk if and only if:***

(A) Pi || Pj for all i ≠ j(yes) (B) Pi || Pj for all i = j+1

(C) Pi || Pj for all i ≤ j (D) Pi || Pj for all i ≥ j

***What is the result of the expression (1&2)+(3/4) ?***

(A) 1 (B) 2 (C) 3 (D) 0(yes)

***When one-dimensional character array of unspecified length is assigned an initial value :***

(A) an arbitrary character is automatically added to the end of the string

(B) ‘o’ is added to the end of the string(yes)

(C) length of the string is added to the end of the string

(D) ‘end’ is added to the end of the string

***The declaration “unsigned u” indicates :***

(A) u is an unsigned character

(B) u is an unsigned integer(yes)

(C) u is a character

(D) u is a string

***The following ‘C’ statement : int \* f[ ]( ) ; declares :***

(1) A function returning a pointer to an array of integers.

(2) Array of functions returning pointers to integers.(yes)

(3) A function returning an array of pointers to integers.

(4) An illegal statement.

***The following statement in ‘C’ int (\*f())[ ]; declares***

(1) a function returning a pointer to an array of integers.(YES)

(2) a function returning an array of pointers to integers.

(3) array of functions returning pointers to integers.

(4) an illegal statement.

## Calculating Constant Failure Rates

In order to measure failure rates, you need a sample of identical components or systems that can be observed over time. For example, suppose you had five light bulbs connected to an automatic circuit that you could then turn on and off once per hour for 1,000 hours, giving you the following data:

* Bulb 1 burned out after 422 hours.
* Bulb 2 burned out after 744 hours
* Bulb 3 burned out after 803 hours
* Bulb 4 burned out after 678 hours
* Bulb 5 stayed lit for 1000 hours

This gives you 4 failures over a total of 3,647 hours.

To calculate the failure rate, divide the number of failures by the total number of hours, such as 4/3,647 = 0.0011 failures per hour.

In this example, the failure rate per hour is so small that it is almost insignificant. Multiplying the number by 1,000 would make it more meaningful to someone thinking about buying a light bulb, which would be 1.1 failures per 1,000 hours. Since there are 8,760 hours in one year, you can divide 3,647 by 8,760 to get 0.41 failures per year, or about 2 failures every five years.

## Calculating MTBF

Another way to express failure rates is by using the Mean Time Between Failures. MTBF is usually used in high-quality systems where failures are expected to be rare and need to be minimized, like the guidance system on a commercial aircraft or the air bags in a passenger car. Knowing the MTBF allows manufacturers to recommend how often components should be inspected, maintained and replaced.

To calculate the MTBF, you divide the number of hours by the number of failures. In the case of the five light bulbs that were tested, which had a failure rate of 4 per 3,647, you determine the MTF as 3,647/4 = 909. The MTBF is therefore 909 hours.

**QUESTIONS(UGC – JUNE 2020)**

Suppose you are compiling on a machine with 1-byte chars, 2-byte shorts, 4-byte ints, and 8-byte doubles, and with alignment rules that require the address of every primitive data element to be an integer multiple of the element’s size. Suppose further that the compiler is not permitted to reorder fields; padding is used to ensure alignment. How much space will be consumed by the following array?

struct {

short s;

char c;

short t;

char d;

double r;

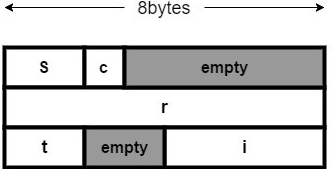
int i;

} A[10]; /\*10-element array of structs\*/

(1) 150 bytes (2) 320 bytes

(3) 240 bytes (4) 200 bytes

**Answer:-**

****

Let assume each block of memory is 8 bytes since the largest element is 8 bytes

requirement: each element placed in the memory has to have an address which is a multiple of its size.

**Step 1 :** I placed element “s” (short) into the memory, which occupies 2 bytes out of the first block of size 8 bytes.

**Step 2:** I placed element “c” (char) into the memory, which occupies 1 bytes out of the first block of size 8 bytes. ( total of 3 bytes occupied till now)

**step 3:** I have to place “r” (real) into the block but since the block only have 5 bytes left and we are not allowed to do reorder, then I have to place the element “r” into the new block.

**step 4:** I placed element “t” (short) into the memory, which occupies 2 bytes out of the third block of size 8 bytes.

**step 5:** I have to place “I” (integer) into the memory, the third block, but since the address has to be multiple of element’s size, I have to jump to 2 bytes and place the element “i” ( 4 bytes) into the last 4 bytes of the third block or in another word into the address 20 to 24. 24 will be multiple of 4.

Since our array has a range of 0-9 ( 10 elements) therefore the total space consumed will be 10 x 24 = 240 bytes.