EECE7352 - COMPUTER ARCHITECTURE: HOMEWORK 2

VIKAS VEERABATHINI[1]

CONTENTS

LIST OF FIGURES

LIST OF TABLES

[1] *Department of Electrical and Computer Engineering, Northeastern University, Boston, United States*

1 PART A

1.1 Write a RISC-V assembly program for multiplication

Description: Write a RISC-V assembly program to compute the product of two integer values. The two values should be initialized in main(). The main() function should call the product(int x, int y) function, passing the two integer values as arguments. The product function should return the product of the two numbers.

1.1.1 *Develop both a non-recursive and recursive implementation of your assembly program. Submit your assembly code on Blackboard through Turnitin.*

1.1.2 *What is the largest product that can be computed in your program?*

RISC is a 32bit architecture and it also supports signed integers thereby making the range as -2,147,483,647 – 2,147,483,647 , so an iterative multiplication would usually be able to hold a maximum absolute value = 2,147,483,647.
In case of recursive multiplication . Stack size is 1252 words (32-bit = 1 word) . For every call of a multiplication api , it requires around 8words on stack to stuff it's local variables . Thus keeping in mind the stack size , we wont be able to make more than 156 calls . Which would eventually be a cap . Say we have a multiplication of 120 * 160 (It wont be possible , as it would involve calling multiply api 160 times) which would lead to a stack overflow .

1.1.3 *Discuss how you implemented integer multiplication, since it is not directly supported on the simulator. Discuss an alternative implementation for multiplication. Which implementation would you expect to perform better, and why?*

Please find the pseudocode of the integer multiplication api on RISC-V

```
a,b
tempa = a
for i->b
  a = a+tempa ;
```

Above is an iterative version , where we add one operand to it multiple times till we count till second operand . Which is how multiplication works .
Recursive version :

```
mul ( a,b ) :
  if b not equal to zero
  a + mul (a,b-1)
```

Unoptimized Recursive Multiplication follows below algorithm . (It is a demonstration for multiplication of 20 x 10 , source-code : Recursive-Multiplication.S)
As below shows , it takes around 10 recursive calls to achieve the same . As per our assembly code profiling , each recursive call consumes around 32bytes to perform it's operations (to store local variables) . Thus we would be needing 320bytes of stack for this operation to occur .

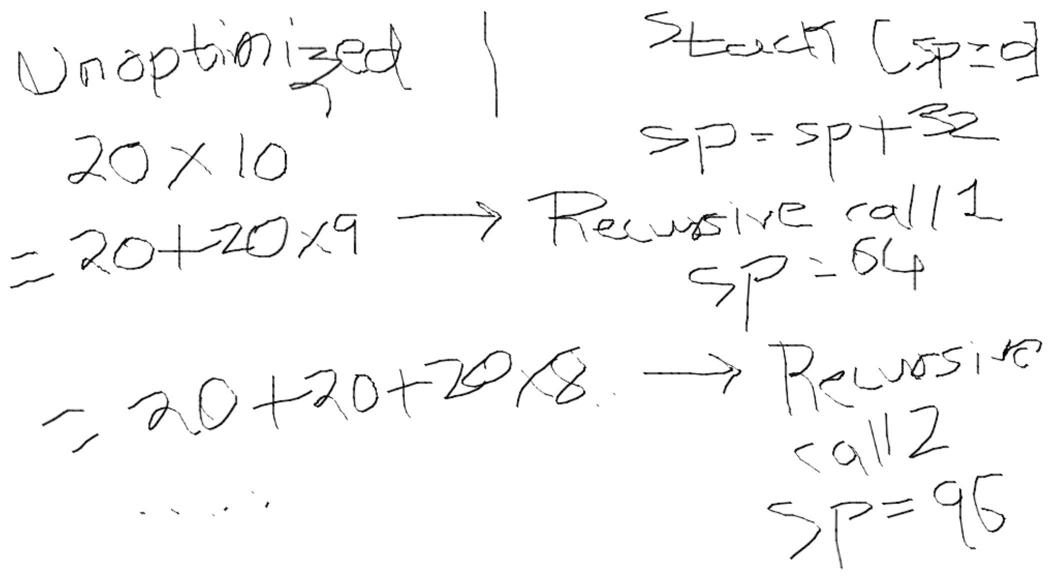

Figure 1

Optimized recursive multiplication
Rather than adding the number to itself recursively , we would proceed further by a factor of '2' for both operands .

```
mul ( a,b ) :
  if b -> 1
    return a
  if b%2 equal to zero
    mul (a<<1,b>>1)
  else
    a+mul(a<<1,b>>1)
```

An alternative Optimized recursive version of the multiplication would occupy less space on stack as well as time as shown below . (source code : Optimized-Recursive-Multiplication.S)

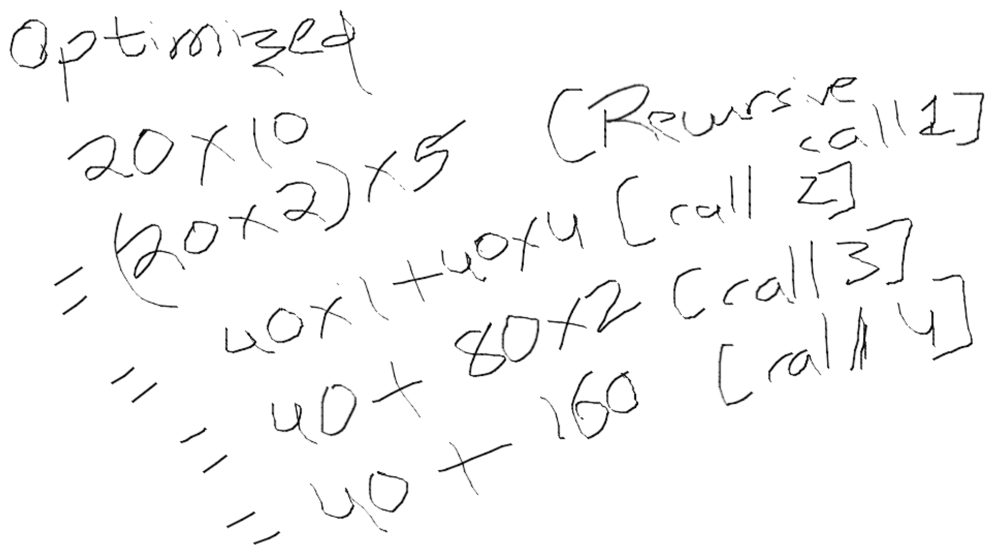

Figure 2

In any case , an iterative version of above would take some more time but would indeed cost lesser stack and would be a best possible solution for multiplication on this architecture . It would also improve the range of numbers to be multiplied as it wouldn't get constrained as in recursion owing to less availability of stack .

1.2 Write a recursive RISC-V assembly program to compute the factorial

Description: Write a recursive RISC-V assembly program to compute the factorial of a value that is initialized in main (). We provide a recursive factorial program in the c program example on Blackboard.

1.2.1 *Submit your assembly code on Blackboard through Turnitin.*

1.3 What is the largest integer value for factorial

Description: What is the largest integer value that you can compute the factorial in your program on the RV32I ISA? Explain why.

1.4 Solution

Largest number for which we can find a factorial is 12 since 13 factorial exceeds the desired the range of Instruction Set Architecture that is 4294967295 , as RISC ISA is 32bits long only . Although we can workaround above to have the higher bits to be stored onto other word , but somehow the RISC-V simulator doesn't allow doing so . Although i was going through RISC-V documents (reference guide) , looks like it is allowed on RISC-V board . Looks like a missing simulator feature .

2 PART B

2.1 Profile Quick-sort

Description: For this problem you will use the qsort.c (quicksort) program provided, and you need to produce a dynamic instruction mix table (similar to Figure A.29 in your textbook) to characterize the execution of the quicksort program. You can perform this study on any architecture of your choice. There are a number of approaches you can take to produce this data. Please make sure to explain how you produced the data in your table and provide details of the tools that you used.

2.1.1 *You could instrument the code to capture the execution frequency of each basic block, and then, using an assembly listing of the program, provide instruction counts (this is slightly imprecise, but very acceptable for this assignment).*

2.1.2 *You could find a tracing program that can capture an instruction trace. You would then have to write a program to count individual instructions (challenging, but not impossible).*

2.1.3 *You could find a tool out on the Internet that provides this capability already for you. While this sounds easy, it may be a bit of work to learn the particular tool you have chosen to use.*

2.2 Solution

I am using Intel PIN (Profiling and Instrumentation tool) to profile our code . It is more like a JIT (just-in-time compiler) or a byte code translator .
Please find the PIN user manual : https://software.intel.com/sites/landingpage/pintool/docs/97998/Pin/html/
SETUP :
i) Download PIN linux
ii) Upload the folder onto linux setup
iii) Provide chmod 777 access to root of PIN directory , so that we don't face access issues .
iv) make qsort.test (note .cpp needs to be replaced by .test in commandline only)
Code modifications :
In a nutshell PIN adds an API as a callback for every instruction execution . , lets assume we have INSTRUCTION A and INSTRUCTION B . During compilation as mentioned in step (iv) , Pin will parse the assembly of <filename.S> and for each instruction encountered , it would embed the subsequent callback assembly instructions between Instruction A and Instruction B . I have made my own callbacks and employed them to increment counters for memory read , memory write , syscall , branch , an allaround instruction counter respectively . That would help to let us know the number of memory access , branch , syscall and total number of instructions .

Please find the dynamic instruction table as below :

Name	**Syscall**	**Stack Access**	**Branches**	**Memory Writes**	**Memory reads**	**ALU operations**	**Total**
Qsort.c	21	3930	3978	154185	398289	986241	1546644
	0.00135%	0.254%	0.257%	9.969%	25.75%	63.7%	

As per above , looks like it is ALU operations intensive , but memory write and read combined make-up for 35% . In my opinion , I am inclined to mention it is bit memory intense too . Somehow if we speed-up memory access by 50% , then we happen to gain approximately 18% speedup in program execution . Coming to ALU operations , it is difficult to crunch it further as we are using -O3 flag for compilation , so the only way to optimize it further is to have some more parallelization to improve the speed .

3 PART C

3.1 Floating point benchmarks

Description: For this part of the assignment, write two different benchmark programs on your own that contain significant floating-point content. Compile the programs on X86 and generate an assembly listing of the benchmarks. Then identify 4 different floating-point instructions used in each program (a total of 8) and explain both the operands used by each instruction and the operation performed on the operands by the instruction.

3.2 Solution

Benchmark 1 : It is a simple testcase which has two components .

1 . String copy (Allocates two pieces of memory . Initialize one piece with a sentence and then copies 50 characters from one piece of memory to other)

2 . Floating point operations (Execute math equation

$$: 3x^3 + 5x^2 + 6x + 10) \tag{1}$$

Above two components are run for around 10000000 iterations .

Results 1

Name	Number of cycles	Time ticks	Time ticks per loop
String copy	10000000	11908715043	1190
Floating point ops	10000000	154662852	15

Idea of above test case is to benchmark two important aspects of processor i.e.. Memory access and ALU operations (Floating point unit)

From Results 1 we can say that it reveals some benchmark about basic features of an architecture , but instructions involved in bench marking are fairly simple and I feel that we rather need a very intense operations in a single iteration which could invade the architecture and stress it to full extent so that we can get reliable benchmark ratings .

So we modified benchmark 1 , with below components :

1 . Allocate 512kb of memory in heap region of 256kb each . Then perform a byte-byte copy of around 64k for each iteration

2 . Two Floating point equations of an order of 3 . One dealing with trigonometry and other with logarithm to ensure that we use gcc math libraries and study it's assembly as well .

Results 2 :

Name	Number of cycles	Time ticks	Time ticks per loop
String copy	100000	19105625432	191056
Floating point ops	100000	94257783	942

We are using chronos library in C to measure the start and end time . It is a very precise timer available inbuilt in C++11 standards . A very great thing about it is we dont need to run the benchmark in billion loops so that we can get the time it ran in seconds , chronos being precise it works in time ticks or clock rate approximately . An important thing about above benchmarks is that we are not trying to add many instructions , just that we are trying to performing complex instructions with different components (Floating point and Memory accesses) many number of times . That would help us gauge on how fast processor really is .

Benchmark 2 floating point assembly instructions :
movss -8(%rbp) , %xmm0
1 . Move the value from stack offset as mentioned below to that of xmm0 register with a signed single precision format . It can be used on either low double-words of xmm registers or from a 32-bit memory location to xmm register it doesn't work on memory-memory transfer
sqrtss -4(%rbp), %xmm0
2 . Compute the square root of value stored at stack location with offset (-4(%rbp)) and stashes them onto xmm0 register - lower double word as a single scalar precision value
cvtsi2ss %eax, %xmm0
3 . Convert a doubleword integer stored in xmm0 (source) to single scalar precision floating value stored which is stashed in eax (destination) . Note that destination can only be a GPR while source can be xmm or a memory location
movaps %xmm0 , %xmm3
4 . Move aligned packed single precision floating point value from xmm3 (source) to xmm0 (destination) . In above case both the operands are registers , but it's major use case is to load a value from any xmm onto a memory location . In that case if the memory location is not 128-bit or 16byte aligned then it raises a general - protection exception
Benchmark 1 floating point assembly instructions :
cvtsi2ssq %rax, %xmm0
1 . Convert a quadword integer stored in xmm0 (source) to single scalar precision floating value stored which is stashed in rax (destination) . Note that destination can only be a GPR while source can be xmm or a memory location
mulss %xmm2, %xmm0
2 . Multiplies the low single-precision floating-point value from the source operand (second operand) by the low single-precision floating-point value in the destination operand (first operand), and stores the single-precision floating-point result in the destination operand. The source operand can be an XMM register or a 32-bit memory location. The destination operand is an XMM register. The three high-order doublewords of the destination operand remain unchanged
addss %xms (destination - xmm only) %xmm2 (source - xmm only) , %xmm0 (source)
3 . Adds the low single-precision floating-point values from the second source operand and the first source operand, and stores the double-precision floating-point result in the destination operand. The second source operand can be an XMM register or a 64-bit memory location. The first source and destination operands are XMM registers.
movups %xmm0 (destination) , %xmm1 (source)
4 . Operation is same as movaps , just that in case if the destination is a memory location and it is accidentally unaligned then it doesn't throw a general protection exception instead the Hardware does the copy with two issues .

4 PART D

4.1 Appendix K

Description: For this problem you will need to read through Appendix K in your text, covering a number of instructions sets, and then answer the following questions:

4.1.1 *Name 2 CISC instruction set architectures and 2 RISC instruction set architectures.*

4.1.2 *Describe 3 characteristics of the DEC Alpha instruction set.*

4.1.3 *Discuss the differences/similarities between MIPS and PowerPC in terms of how they handle conditional branches.*

4.1.4 *Given an example of how register windows work on the SPARC ISA.*

4.1.5 *In your opinion, which generation of the Intel x86 architecture was the most significant advancement from the previous generation of the ISA.*

4.2 Solution

Please find 2 RISC ISAs below :
1 . Power version 3.0 2 . SPARCv9
CISC ISAs are as below :
1 . Intel 8086 processors 2 . VAX architecture
2) Describe 3 characteristics of the DEC Alpha instruction set.
Please find characteristics of DEC Alpha instruction set as follows :

i . Each instruction is 32bit or 4bytes long .

They have a regular format only .

R-type format is as follows :

Rd : destination operand
Rs2 : source 2 operand
Rs1 : source 1 operand

31					
Funct7	rs2	rs1	Funct3	rd	opcode

ii . There are 32 integer registers (R0 through R31) , each 64 bits wide . R31 reads as zero, and writes to R31 are ignored . We encounter many operations in assembly which would adding zero or subtracting zero or has some other thing to do with zero , where R31 can be simply used rather than having a space reserved in stack for the same and then loading it .
iii . There are 32 floating-point registers (F0 through F31), each 64 bits wide. F31 reads as zero, and writes to F31 are ignored. It is worth mentioning that , that all floating point operations happen between floating point registers only . Owing to above restriction , architects had to carve out a separate F31 register which would be holding zero forever , as we cannot use R31 for a zero in floating point operations .

3) Discuss the differences/similarities between MIPS and PowerPC in terms of how they handle conditional branches.

MIPS and powerPC have many differences when it comes to handling of conditional branches and they are as follows

MIPS	**PowerPC**
Is similar to RISC-V	Is similar to SPARC
Only seeks the result as 1 or 0 .	Has four condition codes with eight copies of them in a register
Is constrained to test for equality or against zero	Supports various complex operations (<, <= , >, >=)
Had to embed SLTx (Shift less than instruction) to have a logic for <, <= expressions	Doesn't need such instruction
Conditional branch instructions are fairly simple and doesn't involve ALU at all	Conditional branch instructions are complex and gets a hang of ALU for most instructions

In a nutshell , MIPS has a fairly simpler handling of conditional branches with many restrictions but PowerPC has a complex handling of conditional branches with very less restrictions . Since MIPS conditional branches are fairly simple it doesn't involve predication of branch techniques which greatly reduces a tendency for spectral attacks .

4) Given an example of how register windows work on the SPARC ISA.

Register window has a provision of 8 registers each for local , global , incoming and outgoing params . When a new procedure is called all the parameters related to functions are stored onto these banks which form a circular buffer A fairly good example of a register windows on SPARC processor would be say we have functions A,B,C,D where A calls B , B calls C and C calls D . (something like this : A -> B -> C -> D)

A0 A1 B0 B1 C0 C1 D0 D1 D2

Above is how a register window looks like . (Assuming A,B,C have used 2 banks respectively and D has used 3 banks) . Suppose by a certain means , if a program had a deeper call stack and let the function E be the one at the last which is supposedly exhausting the register banks it would then overwrite into the first index where A resides as the windows are circular in nature .

5) In your opinion, which generation of the Intel x86 architecture was the most significant advancement from the previous generation of the ISA

After a brief golden handcuff period to avoiding playing around with software , Intel came up 80486 architecture that aimed to increase the performance with Pentium and P5 processor . This also gave an advent to having multimedia instructions , SIMD and vectors on intel architecture 1997 (basically Pentium MMX architectures) onwards which I consider as a significant advancement . Having a thought to produce such a great advancement with a risk of having software being jeo-pardized , also reducing the efficiency of clock rate owing to increase in registers is something worthy to be mentioned .

5 PART E

5.1 Amdahl

Description: Read the Amdahl, Blaauw and Brooks 1964 paper on the IBM 360 Architecture. Given the timeframe of the paper, what do you find the most impressive feature of the architecture as described by the authors? Justify why you feel this is such a great feature. Also, discuss the representation of the various data types supported on this important ISA, and contrast it with the RISC-V.

5.2 Solution

IBM360 architecture has a very impressive design considering the time (since it was in 1960 , where there was no much dependency on computers especially for day-day chores)
1 . Using addressed - register scheme rather than push down of stack considering a very important limitation of stack-based mechanism with variable length instructions .
2 . Having a variable length instructions in contrast to that of fixed length ones , inorder to carry business transactions swiftly
And many more ..

But the most striking feature I felt was having defined a channel as a conceptual entity with a standard interface and establishing a concurrency support for up to 256 IØdevices using 256 channels . It can be simply accorded as a major advancement in field of communication systems with multiplexing engaged . It throws open a new pathways to various fields of computers be It networking , servers or an embedded system , which as a requirement of multiple channels of communication to the main processing unit .

IBM360	RISCV
It has a support for 32 as well as 64 bit instructions .Some models Employed a support of 16,32 and 48 - bit instruction formats to improve code density so that business transactions can be faster	Constrained to 32-bits only with no variable length support . Thereby giving advantage to HW designer to develop a simple hardware that would expand any RVC (RISC-V) instruction into a 32-bit machine language code only
Some models of IBM360 (especially smaller ones , for fast business arithmetic) use storage-storage or memory-register ISA	It uses a decimal accumulators or load-store based ISA only
It supports a half-byte data type and thereby remaining other data types short int (32-bit) , float (16,32,48 bit) , char (8-bit) , zone decimal (Each 4-bit pack , repesenting a digit with rightmost 4-bit representing sign) .	It supports . Char , int , float , long , long double , long long .

Zoned or packed decimal , which was coined on IBM360 was a novel thing which was essentially meant to implement Binary Coded Decimal numbers . But was discontinued further as ASCII code came into picture .
Zoned or packed decimal also reduced the compression efficiency of numbers , but it was quite crucial in boosting the arithmetic with particular hardware .
RISC has a support for varied ranges of data-types like char , long double , long long (upto 128 bits of storage) . Above support was absent on IBM360 has there was no need of the same . Because commercial computations at that time usually dealt with a less range of numbers .

6 PART F

6.1 Chapter 1 Problems (Extra credit)

Description: Complete problems 1.7, 1.8, 1.10, and 1.12 (only a and b) from the text.

6.2 Solution

1.7 [10/15/15/10/10] One challenge for architects is that the design created today will require several years of implementation, verification, and testing before appearing on the market. This means that the architect must project what the technology will be like several years in advance. Sometimes, this is difficult to do.
a. [10] According to the trend in device scaling historically observed by Moore's Law, the number of transistors on a chip in 2025 should be how many times the number in 2015?
As per moore's law number of transistor's doubles on a device in couple of years . Thus following the same number of transistors on a chip in 2025 would be 32 times that in 2015 .
b. [15] The increase in performance once mirrored this trend. Had performance continued to climb at the same rate as in the 1990s, approximately what performance would chips have over the VAX-11/780 in 2025?
VAX-11 / 780 in 1977 had a SPECint benchmark of 0 , with growth @ 25% till 1990 . 1990 saw a dawn of architectural and organizational changes which lead to 52% growth till 2002 then afterwards we saw it growing at 20% . Had the perfomance trend havent taken a dip , and we would have seen a same trend as in 1990 (i.e., 52%) throughout . Considering that at end of 2002 it was 4195 (from where a 20% growth started) , and rather than growing on 20% it went onto 52% . We would get a benchmark value of

$$47076290 == 4195(1 + 0.52)pow(23) \tag{2}$$

in SPECint with base reference as VAX-11 / 780 .
c. [15] At the current rate of increase of the mid-2000s, what is a more updated projection of performance in 2025?
At the end of 2005 , performance is around 6505 where a growth of 20% was observed . Assuming we observe the same growth at 20% till 2025 , mathematically we project that at end of 2025 it would be

$$248386 == ((6505 * (1 + 0.2)^9 * (1 + 0.2)^9 * (1 + 0.2)^2)) \tag{3}$$

d. [10] What has limited the rate of growth of the clock rate, and what are architects doing with the extra transistors now to increase performance?
Increasing the clock rate , would switch on and off the transistors at similar speed (billion times per second) . The above could cause a increase in thermal density in transistors , thereby dissipating more heat leading unstability issues . This fact has limited the rate of growth of clock rate . Architects are somehow using this extra transistors count to develop multi-core processors . Inorder to perfectly utilize the efficiency of multi-core computing , programs need to have a high amount of parallelizable nature .
e. [10] The rate of growth for DRAM capacity has also slowed down. For 20 years, DRAM capacity improved by 60% each year. If 8 Gbit DRAM was first available in 2015, and 16 Gbit is not available until 2019, what is the current DRAM growth rate?
On the begin of 2015 , DRAM capacity was 8 Gbit . At the begin of 2019 , it is projected to be 16 Gbit . We can observe that ,

$$8(1 + x)^4 = 16 \tag{4}$$

. . . .. [x : yearly growth rate] Solving above equation would fetch us x to be 20%
1.8 [10/10] You are designing a system for a real-time application in which specific deadlines must

be met. Finishing the computation faster gains nothing. You find that your system can execute the necessary code, in the worst case, twice as fast as necessary. a. [10] How much energy do you save if you execute at the current speed and turn off the system when the computation is complete?

Assuming that system isn't necessarily switched off after the execution of a program and runs with same voltage (or current) . Based on the assumption , it looks like system can do the same computation twice faster , so we save half of energy by switching off the system for other half of the computation

b. [10] How much energy do you save if you set the voltage and frequency to be half as much?

Energy is directly proportional to

$$Voltage^2 \tag{5}$$

. . . . (1) If new Voltage (Vn) = 1/2 * old Voltage (Vo) . . . .(2) Then we would new energy = older energy * 1/4 . . . .. (From equation 1 and 2) Thereby giving a savings of 75% on energy

1.10 [10/10/20] Availability is the most important consideration for designing servers, followed closely by scalability and throughput. a. [10] We have a single processor with a failure in time (FIT) of 100. What is the mean time to failure (MTTF) for this system?

FIT can be defined as failures per billion hours of operation . Thus , from the problem we can state that there are 100 failures per billion

$$(10^9) \tag{6}$$

hours of operation . So basically one failure would take around 10 million hours . (= 1 / FIT) Thereby MTTF (Mean time to failure) = 10 million hours

b. [10] If it takes one day to get the system running again, what is the availability of the system?

Availability is defined mathematically as = Mean time to failure (MTTF) / Mean time to repair (MTTR)

From problem 1.10 (a) (above) , we can observe that MTTF = 10million hours While mean time to repair = 1day or 24hours . Thus , Availability = 416667

c. [20] Imagine that the government, to cut costs, is going to build a supercomputer out of inexpensive computers rather than expensive, reliable computers. What is the MTTF for a system with 1000 processors? Assume that if one fails, they all fail.

We have a single processor with a Failure in time (FIT) of 100 . Having 1000 such processors would yield a Failure in time of 0.1mn (1000 * FIT) As we know that

$$MTTF = 10^9/Failureintime \tag{7}$$

We would get MTTF for the system = 10000

1.12 [20/10/10/10/15] In this exercise, assume that we are considering enhancing a quad-core machine by adding encryption hardware to it. When computing encryption operations, it is 20 times faster than the normal mode of execution. We will define percentage of encryption as the percentage of time in the original execution that is spent performing encryption operations. The specialized hardware increases power consumption by 2%. a. [20] Draw a graph that plots the speedup as a percentage of the computation spent performing encryption. Label the y-axis "Net speedup" and label the x-axis "Percent encryption." Please find the graph below , which is a plot of net speedup obtained vs percentage of program spent in encryption

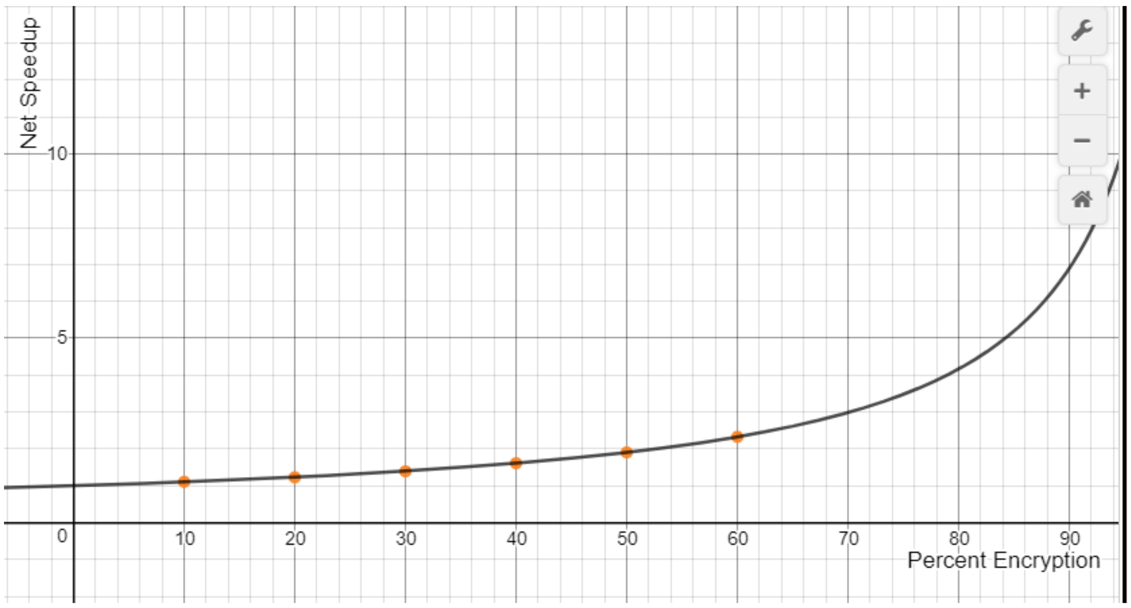

Figure 3

b. [10] With what percentage of encryption will adding encryption hardware result in a speedup of 2?

As per amdahl's law . Speedup = 1 / (1-p+p/N) P : percentage of program on which enhancements work N : Speedup provided by enhancement (Which is 20)

2 = 1 / (1-p+p/20) Solving above equation , we would get p = 10/19 0.48 or 48%

REFERENCES