Reliable and Secure Computer Architecture

Midterm Exam – Fall 2014

Submitted By: Manish Choudhary (mchoudhary8)

Answer 1

A) Is the processor experiencing transient, permanent, or intermittent faults?

Processor is experiencing intermittent faults. Particle strike causes a transient fault generally. But, if there are lots of particle strikes, it becomes an intermittent fault as the fault can occur whenever a particle strikes some bit which is happening very often here. The particle can strike any of the state bits and thus, whenever some bit is stroked, it has a fault. It is not a permanent fault as the bit is not damaged permanently and particle strikes will happens till that substance exists only. Parity also helps to recover from the fault and thus, it is not continuous and stable. The probability of the particle hitting the state bits is much high and thus, can be considered as intermittent fault (as the fault condition is that the particle should hit the state bit).

B) If the correct service for the processor is defined as "the processor is executing instructions", what is its MTTF, MTTR, and availability?

The chances of particle strike are 75%. Let's assume that there are four 60 sec intervals from start. Particle strikes some state bit at 60 seconds and at 120 seconds both causing a failure. If the logic gate bit is stroked at 180 seconds, the next strike at 240 seconds would be definitely on the state bit as per the probability distribution. Therefore, we observed 3 failures in a period of 240 seconds.

So, the MTTF would be 240/3 = 80 seconds.

MTTR = 10 seconds (as the processor executes instructions even at the reboot time and thus, the processor is providing correct service during reboot)

Availability = MTTF/ (MTTF+MTTR) = 80/90 = 88.89%

C) If the correct service for the system is defined as "the system is ready to run user applications", what is its MTTF, MTTR, and availability?

The MTTF calculation is same as explained in the above part as the probability of particle strike on the state bits is same.

MTTR = 40 seconds (as the system won't be ready for the use by application during the reboot period and thus, we have to add that to MTTR)

Availability = MTTF/ (MTTF + MTTR) = 80/120 = 66.67%

Answer 2

A) Is there a scenario in which the overall-output vote would provide better MTTF than bit-wise vote? Describe such a scenario if it exists, or explain why it cannot happen.

Both the voting schemes will output the correct result until 3 ALUs have failed. After the failure of the 3 ALUs, the correctness of the voting result would depend on the different input scenarios.

Over-all output vote can provide better MTTF if the voting inputs and output are of following type. Let's assume that 3 ALUs have failed and are producing random output bits. The outputs produced by each are 1001, 1001, 0110, 0010 and 0100. The overall-output voter will result 1001 which is correct value and thus, the processor will continue to work correctly. But, the bit-wise voter would output 0000 which is wrong and assuming that this output has external impact, it would lead to failure of the processor. Therefore, overall-output voter is able to handle such scenario and thus, provides better MTTF by keeping the system in the correct state and increasing the time to failure. If such scenarios are produced often, the MTTF would be better in case of overall-output voter.

In general, with 2 correct ALUs, probability of overall-output vote being correct is higher as two values will match and rests are random having (15/16^3) probability that other 3 wrong outputs will be same (tie can be handled by re-voting). In bit-wise vote wrong bits at just one position have to outvote the correct bit value and the probability for this is higher (1/8). This results in better MTTF for overall-output vote.

B) Is there a scenario in which the overall-output vote would provide worse MTTF than bit-wise vote? Describe such a scenario if it exists, or explain why it cannot happen.

There are some scenarios in which the overall-output voter is not capable of maintaining the system in correct state. Let's assume that the expected output is 0011. But, the ALUs output following values, 0011, 0001, 0001, 0110 and 1010. The overall-output voter would produce 0001 in result which is wrong. The bit-wise voter would output 0011 which is the right result.

Consider a scenario in which ALU 3 and ALU 4 fail one after another. In that case, the overall-output voter doesn't enjoy the benefit mentioned in (A). Here, 1 ALU is producing correct result. In case of bit-wise vote, any 2 of the remaining 4 bits (for different ALU outputs) at a position have to be equal to the bit at that position in the correct output. The probability for this is ½. So, the total probability of voting the correct result is 1/16. In case of overall-output vote, 1/16 is the probability that another ALU will output same value as outputted by the correct ALU. But, this may have to face tie or outvoting by other 3 ALUs. Therefore, the bit-wise voter has higher probability of succeeding and thus, can have MTTF better than the overall-output vote.

C) With bit-wise voting, what is the probability that the probe fails during the first year?

First, let's calculate the probability of the processor working perfectly (with max 2 ALUs failed).

It will be:

0.5^5 + (5: 4) * 0.5^4 *0.5 + (5: 3) * (0.5 ^ 3) * (0.5 ^ 2)

= 0.5

Now, in the case of 2 correct ALUs, to make the vote output correct, any one of the three bits at each position of outputs has to be equal to the corresponding bit in correct output value and the probability for which will be 0.0625 approx.

In case of 1 correct ALU, any 2 bits of the 4 remaining bits at each position have to be equal to the correct bit and probability for that is 0.063.

So, approx total probability is 0.5 +0.0625 + 0.063 = 0.5688

The failure probability is 1 – 0.5688 = 0.4312 (This approx as some other scenarios can be considered.)

D) With overall-output voting, what is the probability that the probe fails during the first year?

First, let's calculate the probability of the processor working perfectly (with max 2 ALUs failed).

0.5^5 + (5: 4) * 0.5^4 *0.5 + (5: 3) * (0.5 ^ 3) * (0.5 ^ 2)

= 0.5

In case of 2 correct ALUs, to get the majority, another ALU has to produce the same output for which probability is 1/16.

In case of 1 correct ALU, the probability of correct vote output is 1/16^2 (when 2 other ALUs produce the same output).

In case of all failed, the probability of correct value is 1/16^3.

So, total probability is 0.5 + 0.0625 + 0.00391 + 0.00024 = 0.567

This is an approx probability as other cases in which 2 outputs becoming the majority can be considered.

Therefore, the probability of failure is:

1 – 0.567 = 0.43346

Answer -3

A) What is the probability that the system will halt after the first two seconds of operation?

Number of bits per line = 128 + 48 + 1 + 1 + 32 = 210 bits
Total number of bits in the cache = 1024 * 210 bits
The system will halt if the two bit-flips in the first two seconds result in a double error as the system halts when the detected error can't be corrected. So, we need to calculate the probability of double error at two seconds.
Let's assume that one (any) of all the bits gets flipped at first second. To have a double error, another bit in the same line must be flipped at next second.
So, the probability of this is 209/ (1024*210) or 0.0009719.

B) If this cache is duplicated, one bit-flip occurs each second in each of the two copies, and each copy independently chooses the bit to be flipped. Since there are only two versions of each line, bit-wise majority voting cannot be used to correct errors that the SECDED code cannot fix. But duplication can help us correct most of these errors. How?

As the bits to be flipped are chosen independently, the probability of the chosen bits (for a particular flip) falling in the same cache lines for both the copies is less. For double error, the 2 consecutive bits flipped should be in same line for which chances of happening for the same cache lines in both the copies are even lesser.

Therefore, if there is a double error in one copy of line, the other copy may be correct (or has a single bit error which can be corrected internally). The double error in the line can be detected if the code produced non-zero correction position but a parity match. We can check in the other line if it has the double error. If not, we can take the value from this copy and correct the other corrupted copy.

In this way, we can correct many of the double errors which can't be corrected using SECDED. We will miss out only those cases in which same lines in both copies have the double error and thus, we can't copy a cache line from one copy to other. But, the probability of this happening is very low and thus, we can correct majority of the double errors.

Answer 4

A) What is the FIT rate for processor errors if none of the components are protected?

The FIT rate for processor = Sum of effective FIT rates for the components

= Effective FIT for cache + Effective FIT for Latches + Effective FIT for logic gates

(The effective FIT rate is equal to raw FIT rate * Vulnerability factor.)

= (2^3 * 2^20 * 2^3) * 2^-10 * (2^0 * 2^-1) + (2^7 * 2^10 * 2^3) * 2^-10 * (2^-1 * 2^-2) + (2^28) *2^-10 *(2^-3 * 2^-5)

= 2^15 + 2^7 + 2^10

= 2^32 FITs

B) The only FIT reduction mechanisms available are the three whose cost is specified above, and each mechanism can either be applied to the entire component or not applied at all (e.g. you cannot choose to reduce the architectural vulnerability for only half of the cache). We need to reduce the FIT rate for processor errors (which you computed in part A) by at least 50%, while increasing the CPU cost as little as possible. How would you do it?

Reduced FIT rate should be 50% of (2^32 FIT) at max. i.e., maximum 2^31 errors can be allowed in 1 billion hours. But, as we can apply the solution to an entire component only, the best way would be to make the AVF for latches to become 0. This would make the effective FIT rate for latches to be 0 and thus, the overall FIT rate for the processor would become 2^25. This is the best option as it results in the least possible increase in the CPU cost ($4).

If we had considered to make AVF for caches equal to zero, it would have resulted in the FIT rate for the processor to be 2^17 but would have increased the cost of CPU by $8.

The cost of making architectural vulnerability equal to zero for logic gates is too high and thus, can't be an option.