Complications in the Integration of Commercial Microprocessors in Harsh Radiation Environments

Dr. Heather Quinn
Technical Staff Member
Los Alamos National Laboratory

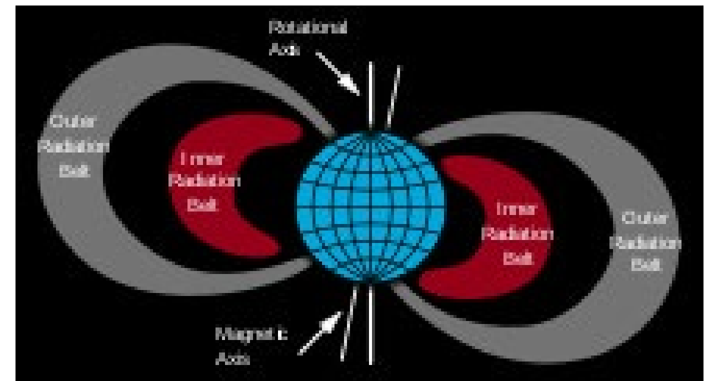
https://www.gapingvoid.com/blog/2011/01/04/the-processor-is-an-expression-of-human-potential/

Managed by Triad National Security, LLC for the U.S. Department of Energy's NNSA

LA-UR-19-22575

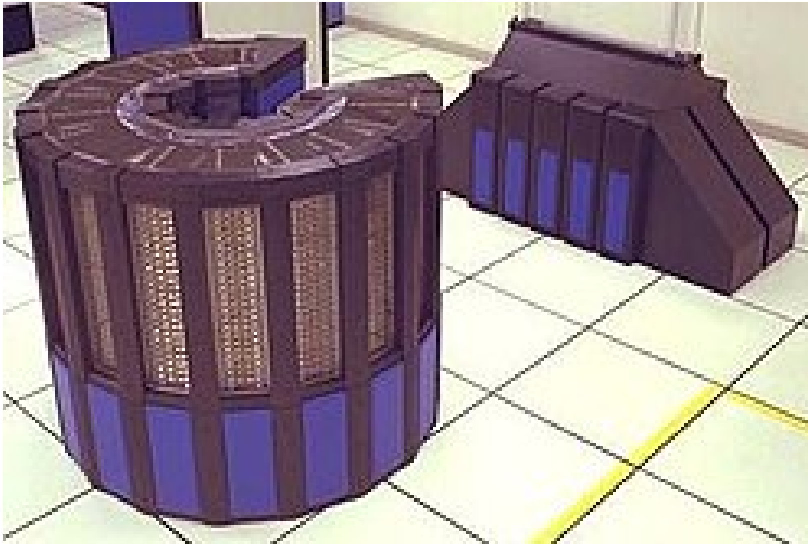
LANL’s 56 Years in Space

- LANL operational space systems monitor the nuclear test ban treaties
  - 1400 sensors, 400 instruments, 74 satellites
- Here are some other things we’ve done in 56 years:
  - Detected the first gamma-ray burst
  - Collected 56 years of space weather data in the magnetosphere
  - Used lasers to shoot rocks on Mars to learn that Mars once had water
  - Flew the first Xilinx Virtex field-programmable gate arrays (FPGAs) in space
- …And now we are trying to find sharks on Europa, Jupiter’s icy moon (…or radiation-resistant bacteria)



https://en.wikipedia.org/wiki/Van_Allen_radiation_belt#/media/File:Van_Allen_radiation_belt.svg

…But We Also Have Giant Supercomputers



https://en.wikipedia.org/wiki/Cray#/media/File:Cray2.jpeg

- Our supercomputers are larger than our satellites by orders of magnitude
- From the early 2000s (130-150nm):
  - The Q supercomputer had 24.0 radiation-induced faults per week in the BTAG memory [1]
  - The Cibola Flight Experiment satellite had 3.5 radiation-induced faults per week in the FPGAs [2]
- Turns out to be an old problem:
  - We determined recently the first radiation-induced fault in occurred at LANL in a Cray-2 supercomputer
- But it is a current problem, too
  - The designers and the programmers are adjusting to the reality that the nodes crash and hardware needs replacement

[1] https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=1545893
[2] https://dl.acm.org/citation.cfm?id=2629556

…And Giant Particle Accelerators

- We accelerate protons to 800 MeV to smash into a tungsten plug to make neutrons
- Around the tungsten is a harsh radiation environment of neutrons and protons
- The entire system is surrounded in a scourge of low-energy neutrons and radio frequency emissions that make it a horrible environment for electronics, which are needed for diagnostic and beam control

https://en.wikipedia.org/wiki/Los_Alamos_Neutron_Science_Center#/media/File:Los_Alamos_Neutron_Science_Center_01.jpg

Here Is What We Have Learned From These Missions

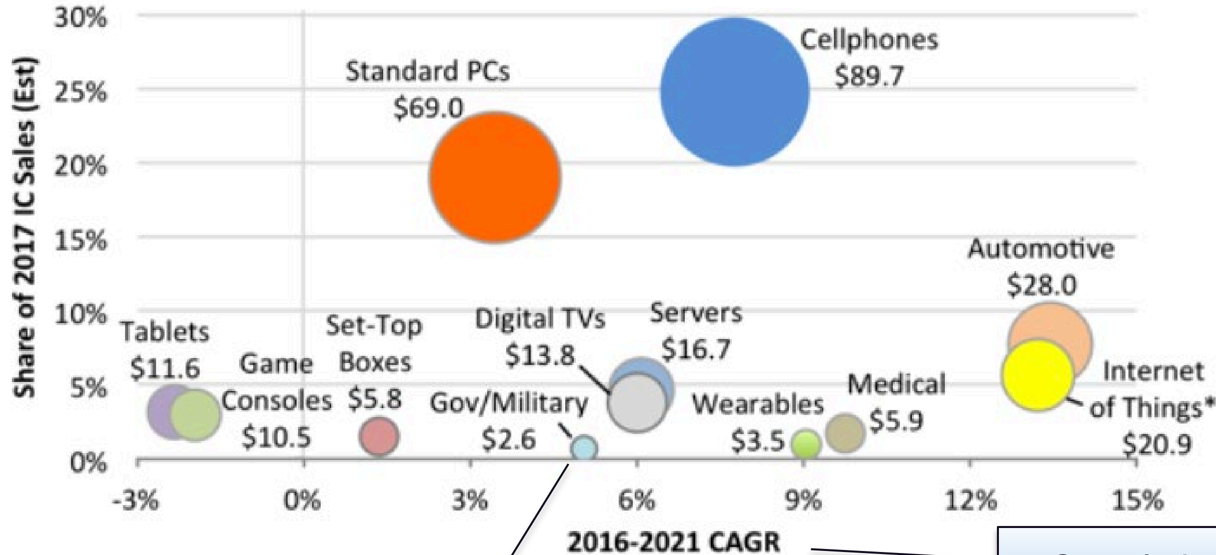
- Science and national security concerns move at a rapid pace, and even satellites need to be flexible
  - While this fact seems obvious now, it was not obvious in 1998
  - People yelled at us that we were “destroying space as we know it” by putting all of the sensor data through the FPGA
  - In 2012 we proved that we were right – the FPGA would not destroy the data and increasing compute speeds 100x
  - So maybe we <u>did</u> destroy space as it was known
- The supercomputers and particle accelerators are more accessible, but not necessarily easier to fix
  - The accelerator is radioactive and a destructive failure means turning the system off
  - The supercomputers are large, and destructive failures means humans tracking down and swapping out hardware

Outline

- The case for using commercial microprocessors in harsh radiation environments
- What's the catch? It's a challenge to test microprocessors!
  - The old way of testing: time-consuming programming in assembly
  - The new way of testing: high-level languages
- We are concerned about the results are getting, and I have 21 open questions for anyone looking for a project

The Case for Using Commercial Microprocessors in Harsh Radiation Environments

The Rad Hard Market is Not Sleeping



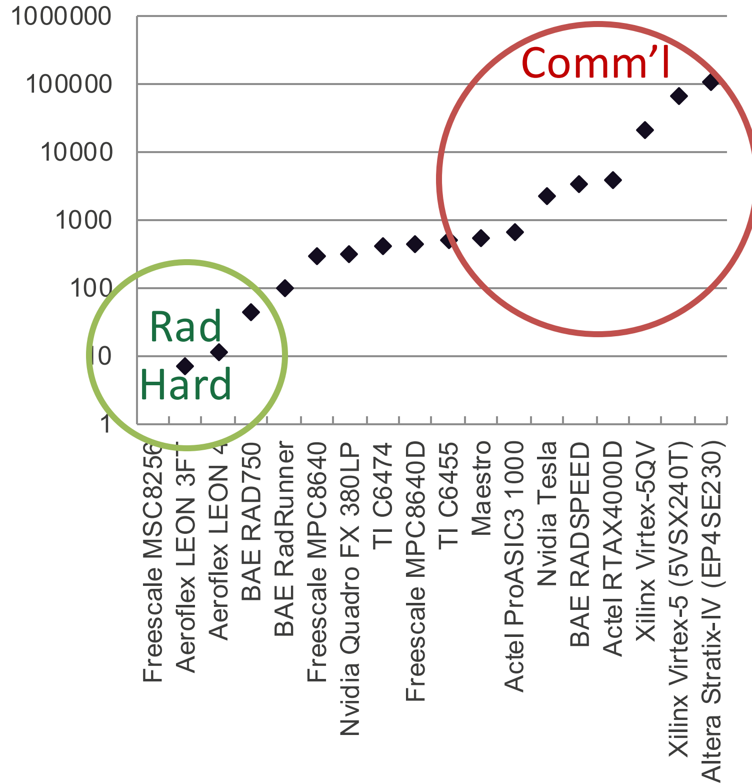
https://semiengineering.com/foundry-challenges-in-2018/

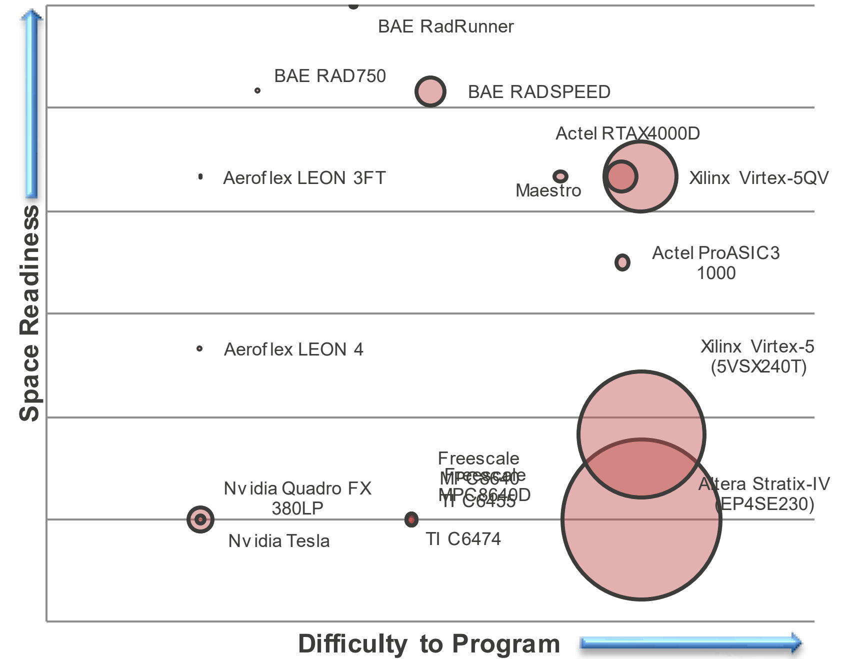
Rad hard market

Cumulative Annual Growth Rate (CAGR)

- In the 1970s, the market share for government and military was 1/3-1/2
- It has been consistently <1% for 20 years
- The only economically feasible solution: fly cell phone hardware

The Current Approach to Radiation-Hardened Compute is Inadequate





Radiation-Hardened Compute is Four Generations Behind…Right Now…And Not Catching Up

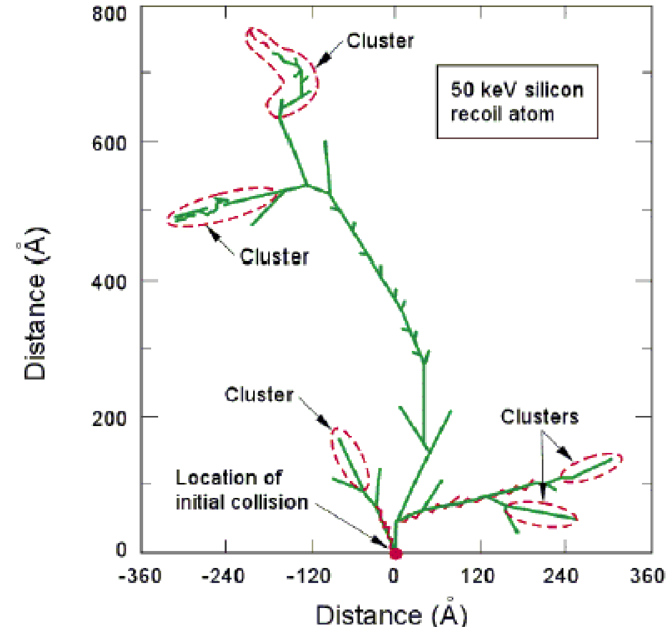
- The BAE RAD750 is the most commonly used radiation-hardened microprocessor in space
  - It is based on the Power PC 750, which was designed in 1997 and used as the original iMAC processor in 1998
  - BAE recently released the RAD750 replacement, which is the RAD5545
    - It is a 45nm 4-core microprocessor, which is four generations behind the state of the art
    - It is better than all of its competitors, which are shown in the lower left corner of the previous graph but still worse than any current commercial microprocessor

- The dynamic tension:
  - Radiation-hardened microprocessors are necessary for mission processing
  - Radiation-hardened microprocessors are not fast enough for other types of processing

What's the Catch?

Three Ways to Destroy a Deployed System

- Total ionizing dose (TID):
  – Accumulation of ionizing radiation causes the transistors' parametrics to prematurely age
- Displacement damage (DD):
  – Accumulation of radiation destroys the transistors through knock-on effects
- Single-event effects (SEEs):
  – An umbrella category of destructive and non-destructive effects caused by single particles



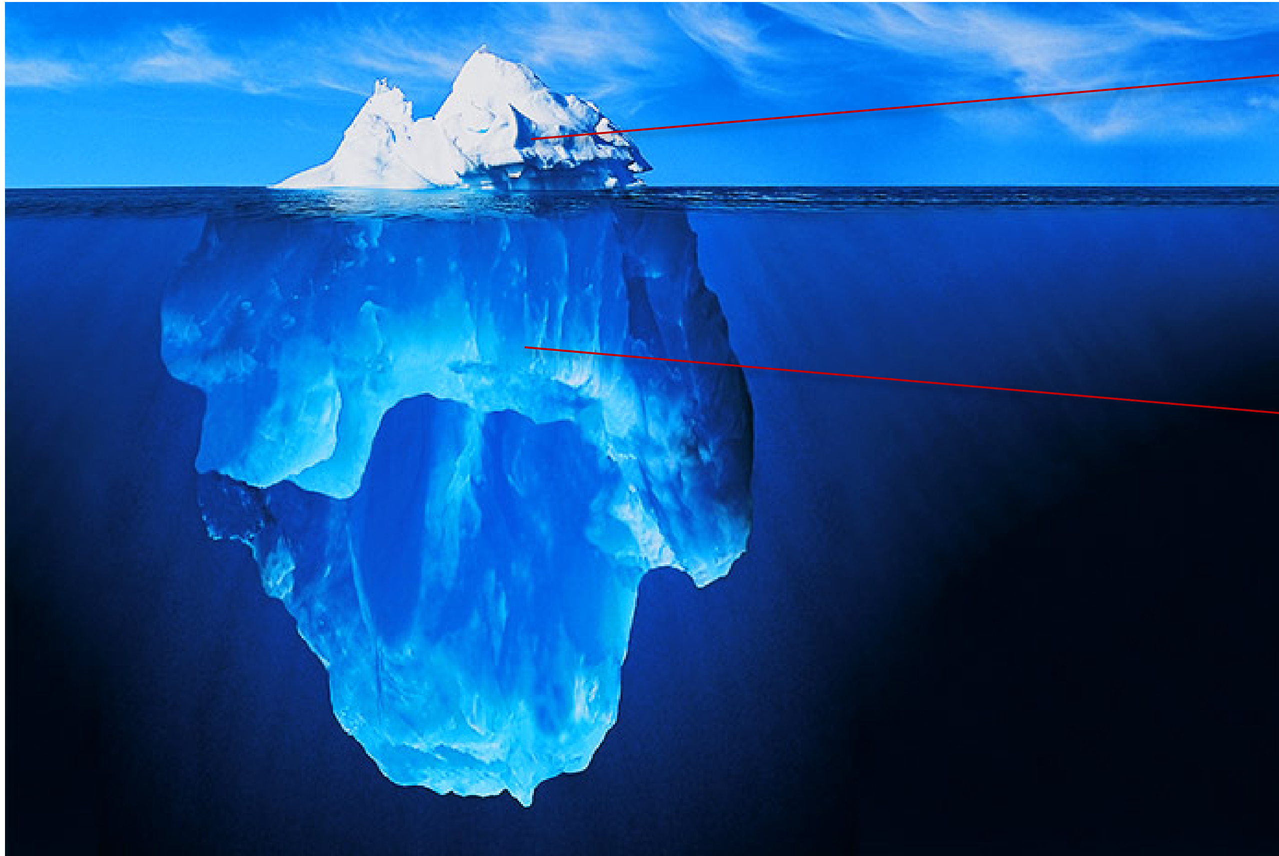
Displacement Cascade Damage in Silicon ["Space Radiation Effects on Microelectronics," NASA Jet Propulsion Laboratory]

To insert commercial microprocessors into satellites and accelerators, need to determine how radiation affects the part.

How Bad Could It Be?

- TID and SEE are known issues with microprocessors that could cause a destructive failure to the system
  - While we can swap out parts in the supercomputer and the accelerator, you might not want to
  - As for satellites…good luck explaining why your hand-picked compute system on your $100M artisanal payload that took a decade to design is dead
- Even non-destructive failures are messy in microprocessors
  - We were able to determine, quantify, and mitigate most of the major non-destructive failure modes in FPGAs in seven years with a team of 20
  - We are still documenting and quantifying the microprocessor problems, and only have a single solution after a decade of research
    - Bad News: there are really only about fifteen people working on this problem worldwide
    - Good News: for the last five years all of us get together twice a month to discuss our progress
    - Even Better News: we’re satellite people, so we are used to being patient
    - Best News: we have at least 21 open questions for grad students to pursue

Why Is This Taking So Long?



What we know about the actual microprocessor architecture

The information deficit: what we do not know about that is going to affect the test

Manufacturers unwilling to help

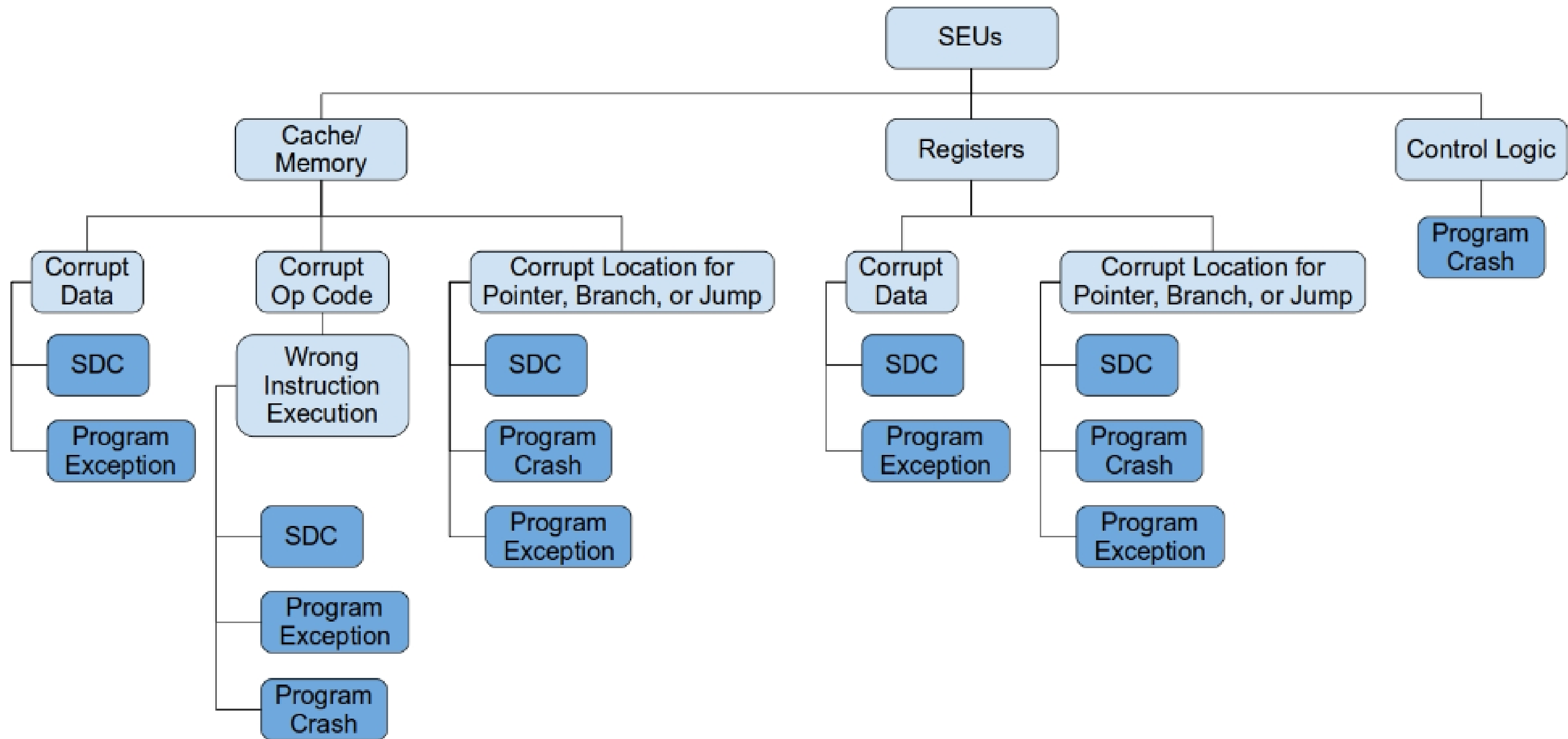
http://www.spindrift-racing.com/jules-verne/drupal/en/out-of-the-classroom/sfs-otc-icebergs-en

Preparing Commercial Microprocessors for Harsh Radiation Environments

Qualification of Parts for Harsh Environments

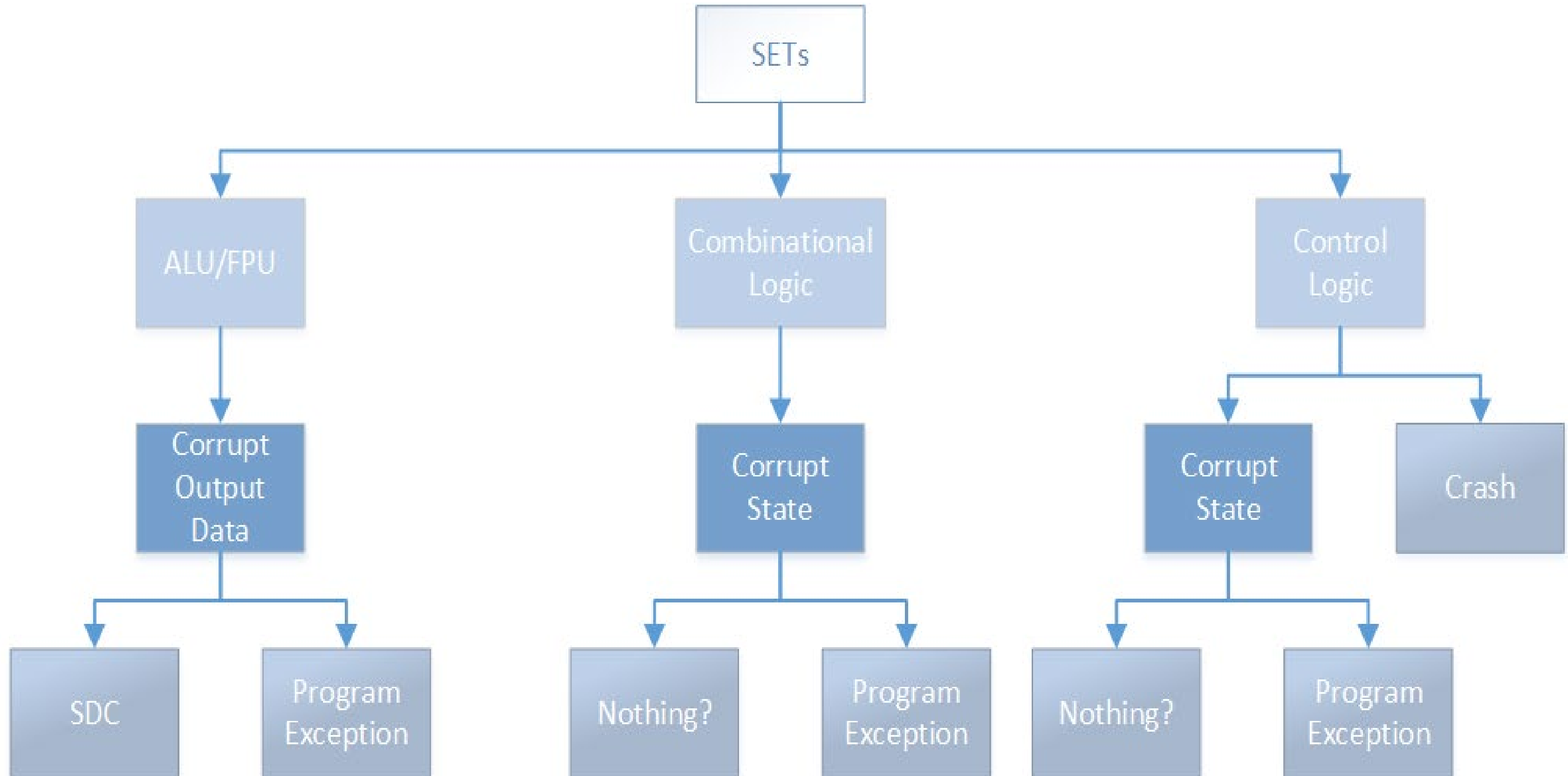
- The part is characterized for how it works in the expected radiation environment
- The idea is to test additively under the assumption that the performance of the full microprocessor system can be partitioned into sub-systems
  - Memory
  - Logic
  - Control
  - Operating System
  - System Software
- This assumption is arguably incorrect:
  - There is no known method for deconstructing or partitioning a system into tractable sub-systems
  - There is no known method of modeling microprocessor systems
  - There is no known method for predicting untested microprocessor systems or software using test results from known microprocessors or software
- On the other hand, this type of testing is the only known method for testing complex systems so until someone develops a better methodology….

Taxonomy of Memory Failure Modes in Microprocessors



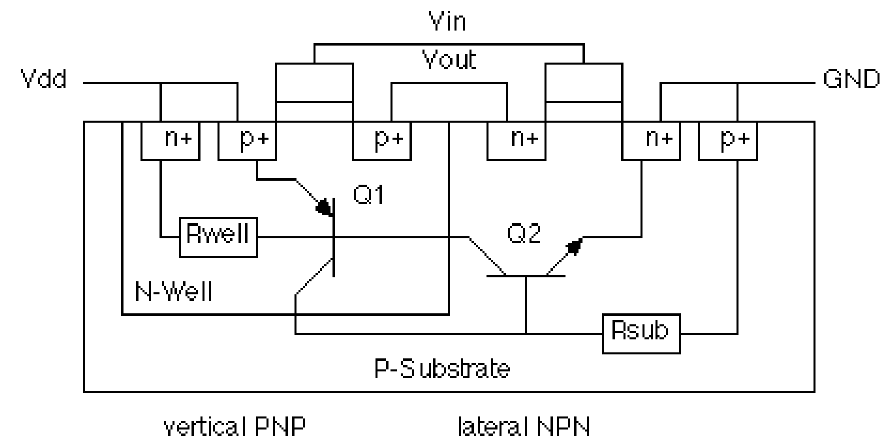
H. Quinn, et al, "Software resilience and the effectiveness of software mitigation in microcontrollers", *TNS.*, vol. 62, no. 6, pp. 2532-2538, Dec. 2015.

Taxonomy of Logic Failure Modes in Microprocessors



Single-Event Latchup (SEL)

- Can naturally occur in any CMOS circuits, due to the parasitic thyristors in the layout
  - Cannot avoid inserting the thyristor
  - Resistors can keep the thyristor from turning on, but not used anymore
- Once ON the thyristor can experience snap back and try to pull an infinite current.
- Many microprocessors have single-event latchup (SEL) sensitivities
  - Low-end microcontrollers often very sensitive to SEL
  - Most of the time we are weeding out these parts, and not trying to remediate



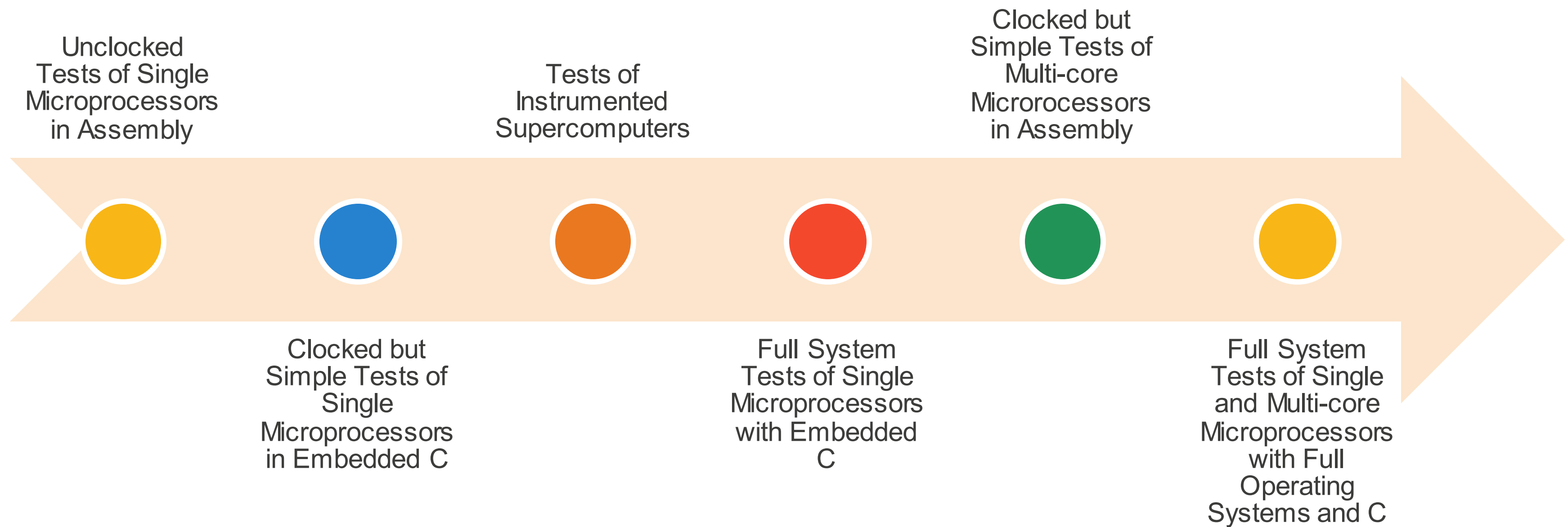
http://www.ece.drexel.edu/courses/ECE-E431/latch-up/latch-up.html

Issues in Testing Microprocessors and Open Questions

The Foundational Questions:

1. What is the underlying sensitivity to radiation in the hardware?
   - Are we testing correctly?
   - Could modeling help us?
2. How does software translate hardware faults into silent data corruption and crashes?
   - It is all how the software uses the architecture?
   - Do the unused parts of the architecture affect either silent data corruption or crashes?
   - How does the full system stack affect errors?

Evolution of Radiation Testing Microprocessors



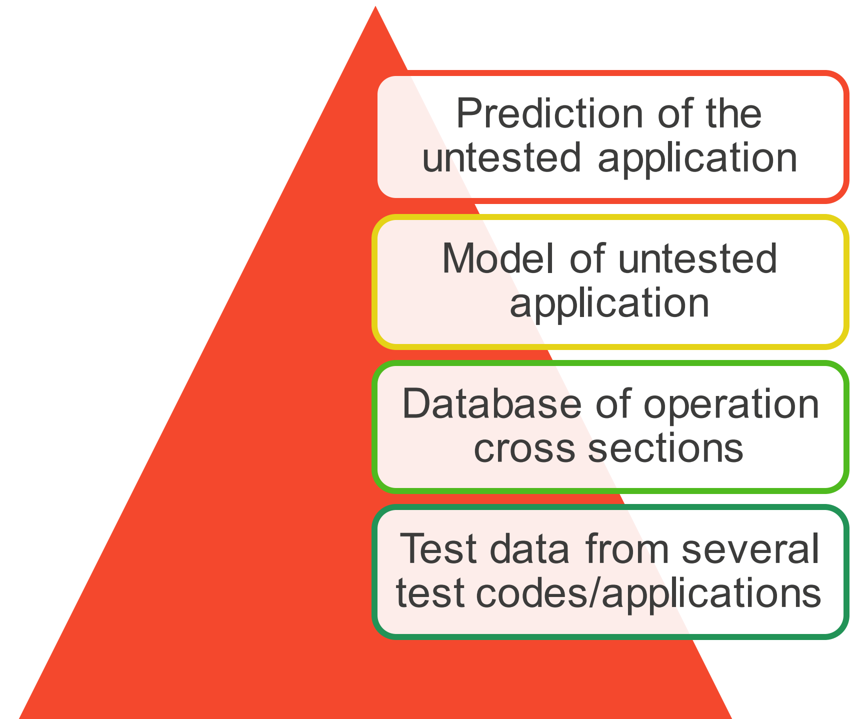
Did this rapid evolution of testing methodology improve the quality of test results?

Benchmarking and Test Standards for Repeatable, Comparable Test Results

- We have been working toward standardized test circuits and codes for testing mitigation for awhile: FPGA benchmark done, software benchmark limping along
- Can we develop a standard set of codes for characterizing all microprocessors or groups of microprocessors?

Predicting the Unpredictable: Modeling Untested Codes

- How do we accurately predict untested codes using information about the hardware sensitivity?
- Additive testing
  - Already discussed
- Use machine learning to extract information about individual operations from existing codes
  - Determine the cross section for different applications
  - Determine which operations are used via profiling
  - Use machine learning to extract cross sections for different operations
- And then a miracle occurs on the modeling side…

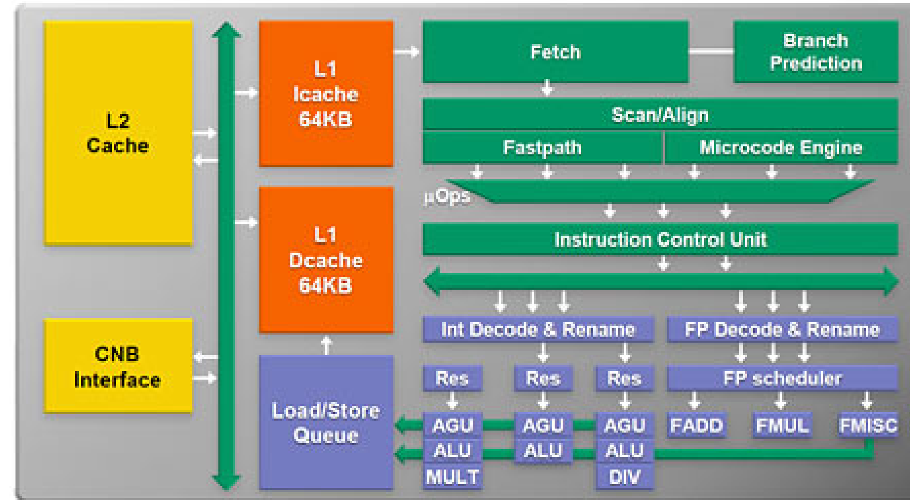


Fault Injection Tools for Microprocessors

- All the testing is occurring in the beam right now
- The community of microprocessor testers (all fifteen of us) needs a good fault injection tool that allows us to insert faults in a manner that is:
  - Fast
  - Uniform
  - Validated
- The problems:
  - The information deficient affects the ability to do complete fault injection
  - The lack of fault categorization affects the validity of doing fault-type-specific fault injection
- If you have a good idea on how to design one, we will help you validate it!
  - We also promise to love you for the rest of your life

Is There a Correct Way to Test Upper Caches?

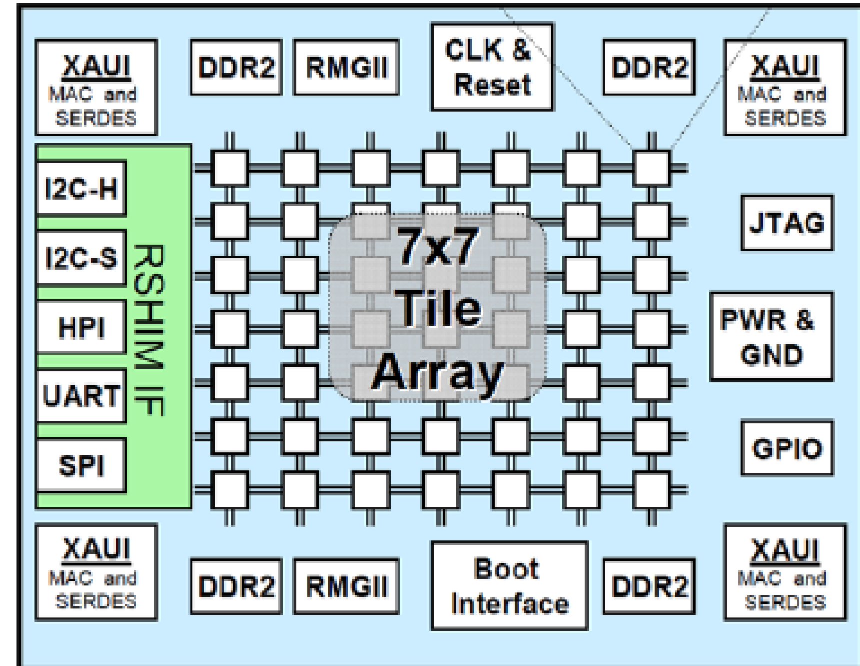
- Data moves from upper to lower caches, lower caches to registers, and registers to ALU
  - Even if the caches are turned off, the data will move through this datapath.
    - Data are overwritten constantly, instead of being cached
- If the tests gradually build from registers only to registers and L1 cache and so on, it is possible to disentangle the results?
- Maybe best to mimic the system software's memory usage to determine whether there is an issue with how data movement occurs in the flight software?



http://hardware89.blogspot.com/2015/11/hardware-prefetcher.html

Measuring Failure Modes in Multi-core Microprocessors

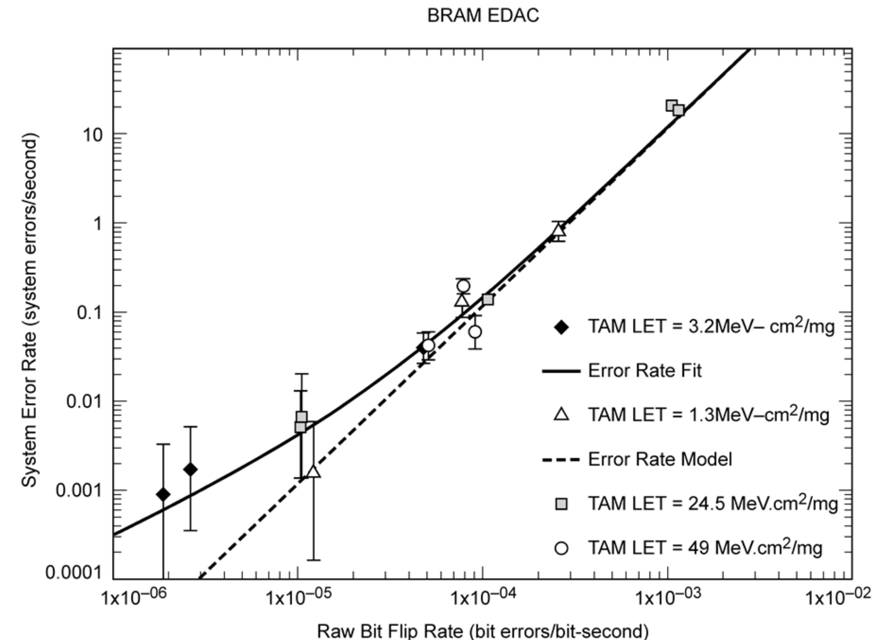
- Multi-core microprocessors have
  - Shared failure modes
  - Intra-core interferences
- What knowledge from single-core tests is useful in multi-core systems?
- Need work on test standards



Steven M. Guertin, Brian Wie, Michael K. Plante, Antwong Berkley, Lonnie S. Walling, and Manuel Cabanas-Holmen. SEE Test Results for Maestro Microprocessor. RADECS 2012.

Measuring Failure Modes in Fault-Tolerant Circuitry

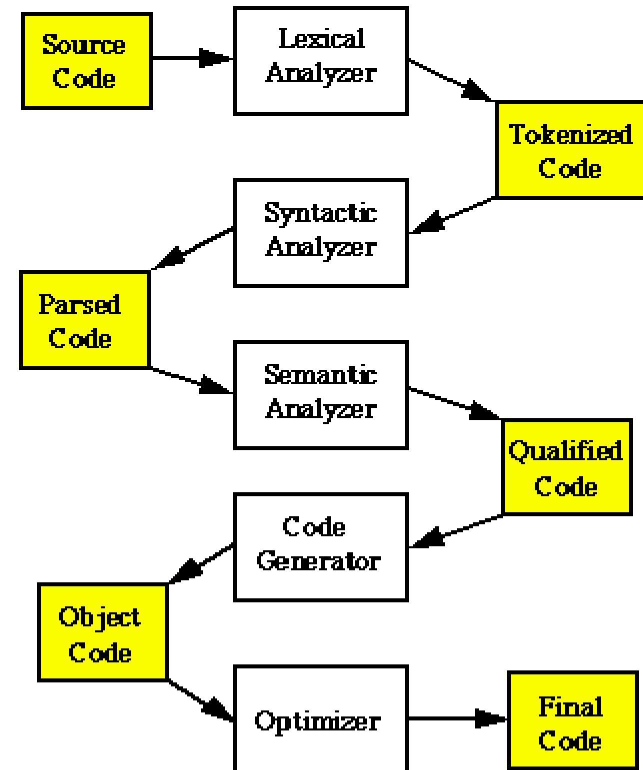
- A fault-tolerant circuit should be tested to assure it is more fault tolerant than the unmitigated circuit
  - Corrects increased fault rate
  - Corrects the types of faults that occur in the circuit
  - Corrects the deployed fault rate
- There are a lot of details that have to be right to make certain the mitigation technique will work
  - Everyone gets it wrong occasionally, and testing keeps everyone honest



S. M. Guertin, "SOC SEE Test Guideline Development," presented at the Single-Event Effects Symposium, San Deigo, CA, 2013.

Compilers and Interpreters

- Side effect of testing software: compilers and interpreters
- Compilers have a first order effect on the test executable.
  - Optimization steps can remove and reorder code: make certain the test, the mitigation or the checker are not removed
  - The effect of compiler optimization on the code's resilience needs quantification
  - The effect of different compilers and compiler versions needs quantification
  - The effect of how the compiler uses the architecture needs quantification
  - The difference between assembly and compiled code needs quantification
- The effect of interpreters needs quantification and study
  - Guidelines are needed for use during tests
  - Older test protocols might need modification



https://courses.cs.vt.edu/~cs1104/Compilers/Compilers.020.html

Programming Languages

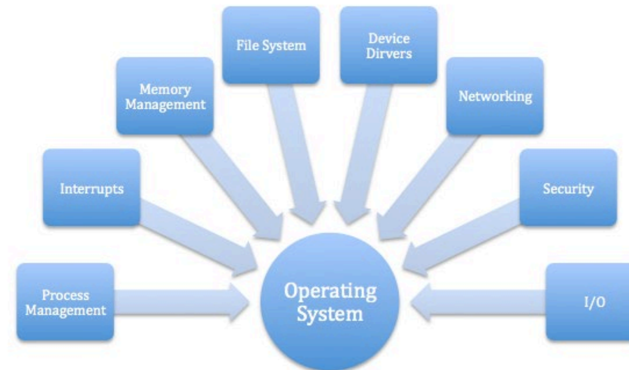
- Historically: assembly
- Currently: C, CUDA, and OpenCL
  - C is a luxury: easier to develop and maintain than assembly
  - Flight software development is likely also done in high-level languages, so the effect of the programming language is important
- The programming languages affect compilers and interpreters
  - Might need to be studied alongside the programming language
- Language differences is not understood and needs quantification
  - The effect of C vs. assembly needs quantification
  - Similar comparison between the GPU programing languages need quantification
  - The effect of the programming language choice on system stability, vulnerability, and robustness needs quantification

Questions vs. Projects

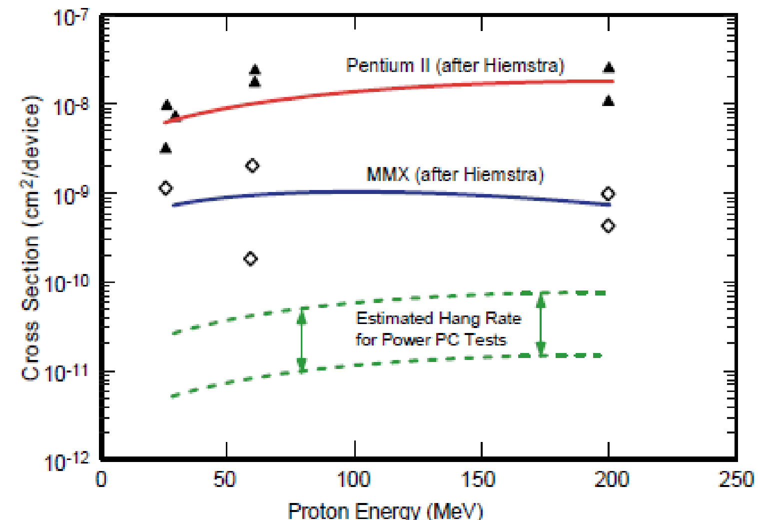
https://sdtimes.com/msft/sd-times-news-digest-redmonks-programming-rankings-vs-code-1-21-apache-cordova-6-0/

Operating Systems

- The operating system as "big software"
  - Sits between the test code and the hardware
  - Controls what executes when
- Widely believed that the operating system sets the crash rate for the system
  - Farokh Irom: "The hang rate was so high that it was not possible to determine the error rate for registers in those tests"
  - Mike Wirthlin (BYU): "The OS increases the crashing by 10x"
  - LANL research: 100,000x increase in crashing
- All operating systems are problematic
  - A solution is needed so we can stabilize deployed system
  - BYU is looking at doing a small, radiation-resistant OS using their automated software mitigation tool (COAST)



https://pbalasundar.wordpress.com/2012/06/11/introduction-to-operating-systems/



F. Irom, "Guideline for Ground Radiation Testing of Microprocessors in the Space Radiation Environment," 2008.

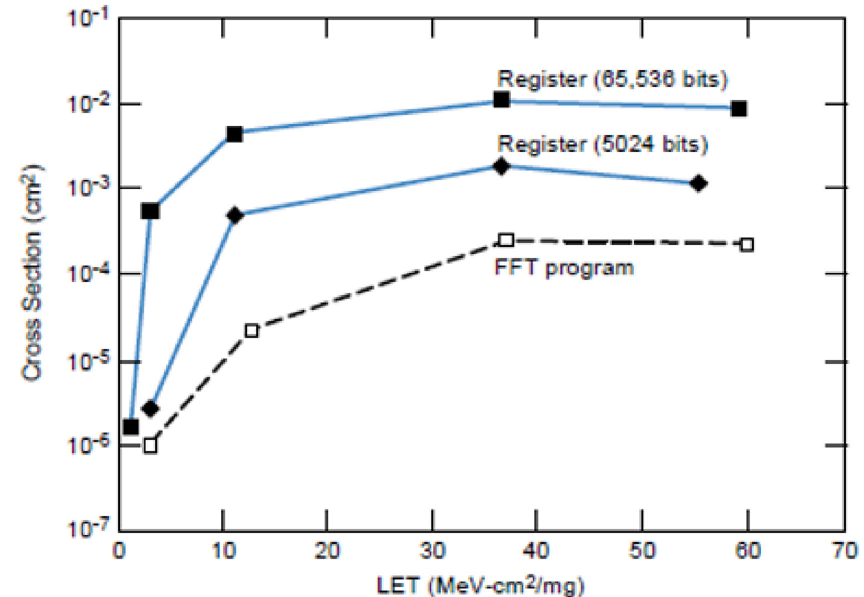
File Systems

- A sub-problem with operating systems
  - LANL has experienced a number of problems with file systems during tests, because we did not know that the ancillary hardware should be protected from radiation during tests
  - We now have the same issues in our cubesats, which are our first satellites with file systems
  - Other testers are now seeing at LANSCE
- Eventually figured out radiation changes files open in memory
  - Operating system writes out the changed file to permanent storage
  - Once enough critical files are changed, the secondary machine will crash and not reboot properly
  - Can fix by reformatting the hard drive, but loss of data and time is problematic
- The problem does not need quantification, but it does need a solution

http://www.shreddersandshredding.net/heavy-shredders.html

Measuring the Effect of Faults in Software

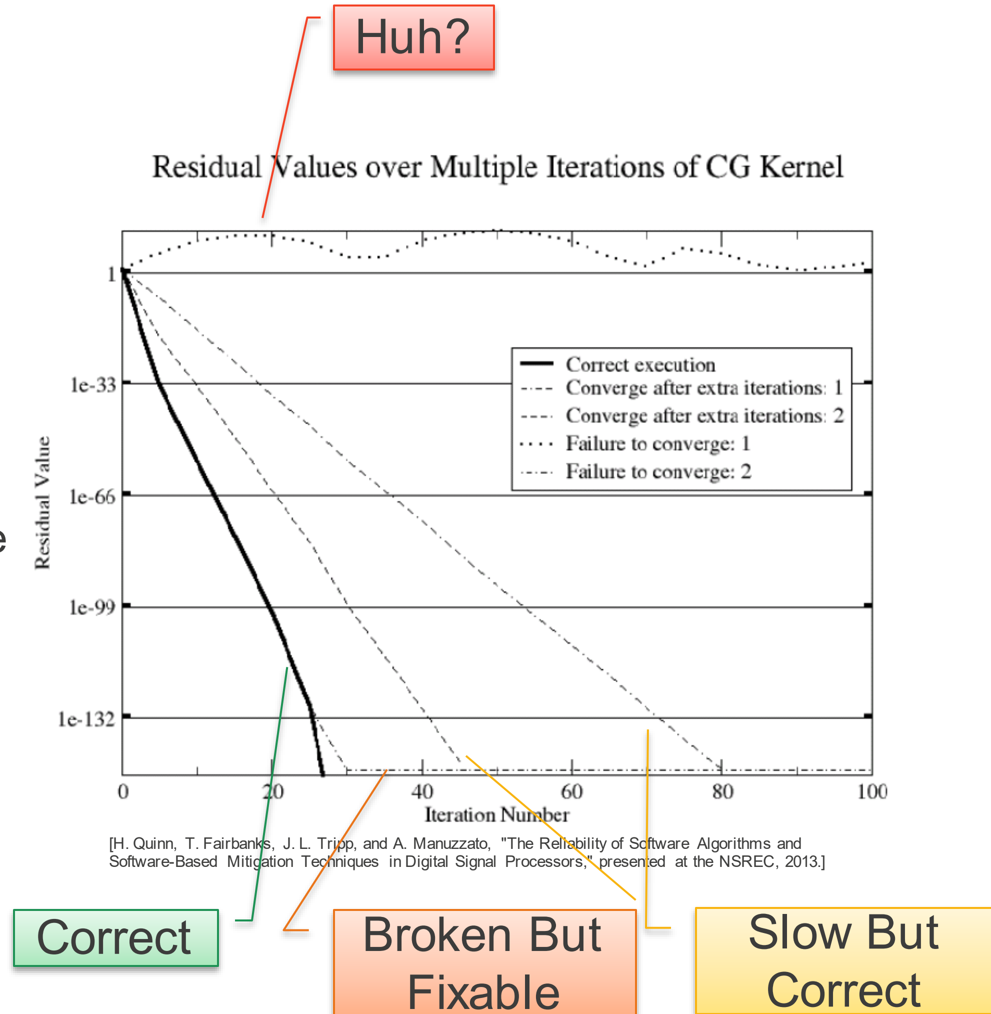
- The software sensitivity is dependent on how the logic and memory are used:
  - The sensitivity from memory is *usually* smaller than the full architecture
  - Currently no way to compare the sensitivity of the full architecture versus how software uses it
- Need a method to translate the sub-component testing into knowledge about the software sensitivity



F. Irom, "Guideline for Ground Radiation Testing of Microprocessors in the Space Radiation Environment," 2008.

Algorithm Design

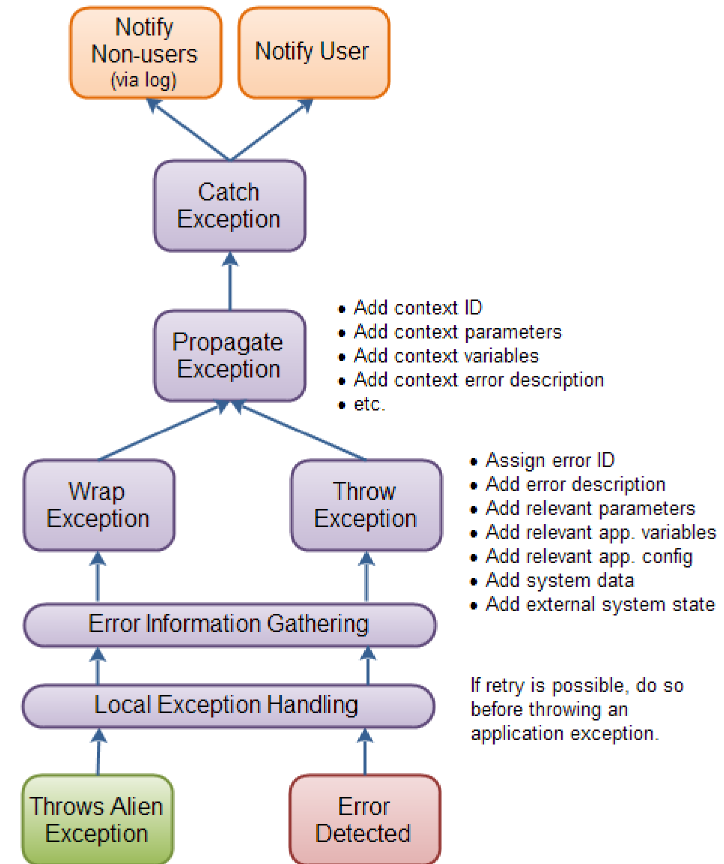
- Algorithm performance:
  - Faster algorithms are exposed to less radiation per execution, therefore fewer errors. Right? Not always.
- Need to determine what makes resilient algorithms:
  - Is where the model of how the architecture is used comes in?
- In the mid-2000s: iterative algorithms are resilient!
  - Assumption: iterative algorithms keep running until the algorithm converged
  - Assumption: recursion is bad
  - Test data: faults cause issues with convergence
  - **In the future, lets agree on this principle: In god we trust, all others must bring data**



[H. Quinn, T. Fairbanks, J. L. Tripp, and A. Manuzzato, "The Reliability of Software Algorithms and Software-Based Mitigation Techniques in Digital Signal Processors," presented at the NSREC, 2013.]

Exception Handling

- Many researchers believe that one of the causes underlying crashes is unhandled exceptions
  - Error correction codes cause some crashes, due to uncorrectable faults
  - JPL has been able to catch exceptions and roll back the fault before the crash…but it only works in assembly
- The effect of exception handling on system resilience needs quantification
  - Exception handling in high-level languages need quantification
  - Better exception handling in the operating system is needed



http://tutorials.jenkov.com/exception-handling-strategies/overview.htm

Categorization of Faults

- The percentage of faults that are from data vs. logic vs. control have not been quantized
- Assumption: the breakdown changes with software
  - Does Quicksort have more control flow errors due to context switching?
  - Does Advanced Encryption Standard (AES) have more logic faults because it takes very little memory and does a lot of bitwise math?
  - Or is it all just memory faults, because there is so much memory?
  - Worst thought ever: how many faults are caused by unused portions of the architecture?

Summary

A Guide for the Perplexed

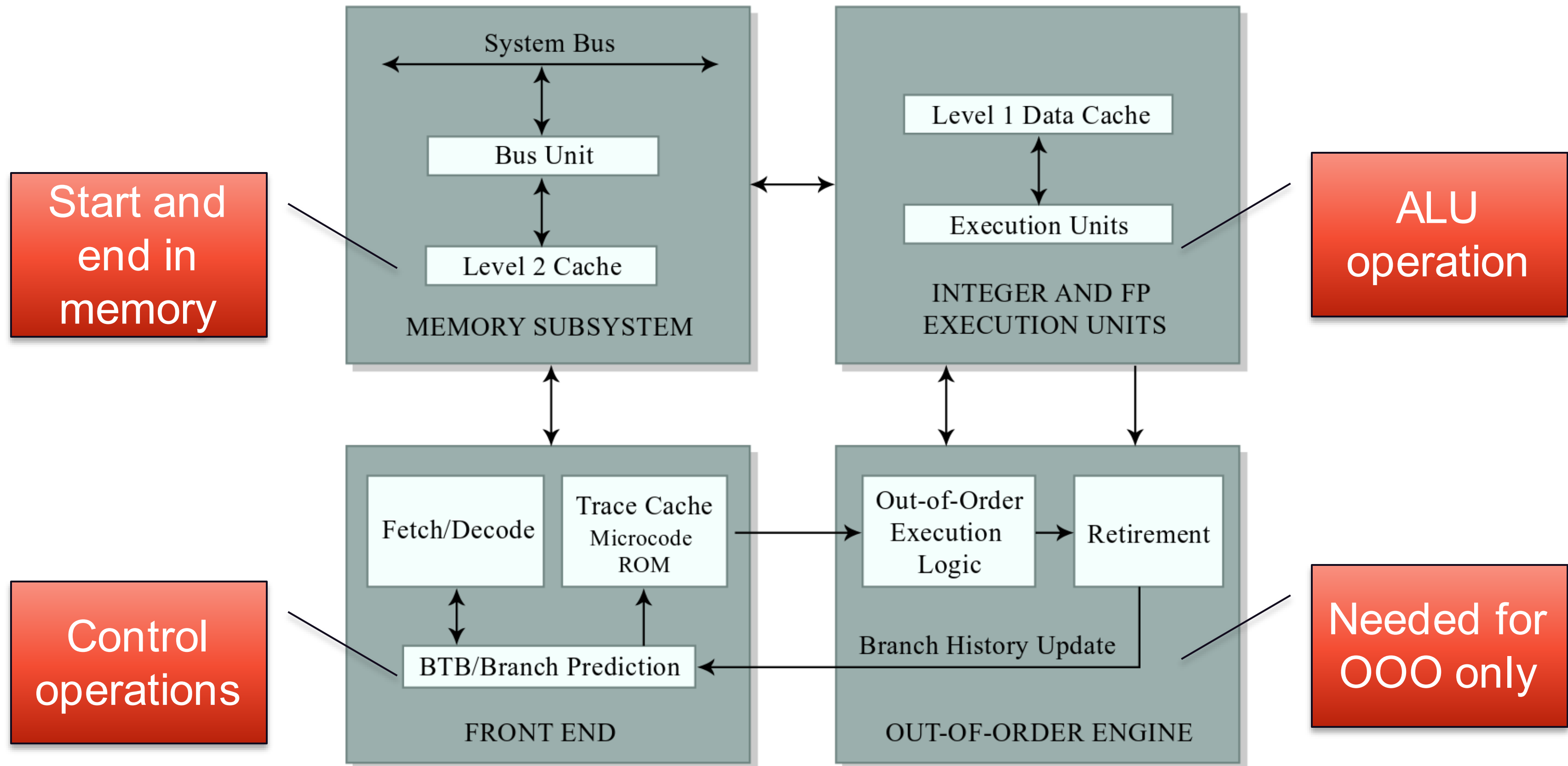
- It's a challenge to test microprocessors!
- We had an older way of testing in assembly, but it is time-consuming to design tests and hard to scale results
- We have newer ways to test using higher level languages, but we are still concerned about the results are getting
- There is a benchmark for testing mitigated codes, but that might not be enough
  - The benchmark group is looking at new methods of testing: you should join the effort!
  - Take one of our 21 open questions. We cannot answer all of them.

The Most Perplexing of the Open Questions

- Compilers and interpreters: (1) How does compiler optimization affect the code? (2) How do different compilers and compiler versions affect the code? (3) What is the difference between assembly and compiled code?
- Programming languages: (4) Does the language matter?
- Operating system: (5) Is it just large software? (6) Does it make software crash more or less frequent? (7) Are all operating systems equal? (8) Can we design a better operating system?
- File systems: (9) Can we design a robust file system that does not lose or corrupt your files?
- Algorithm design: (10) Does the actual algorithm matter? (11) Does the design of the algorithm matter? (12) Are iterative algorithms better? (13) Are recursive algorithms worse?
- Exception handling: (14) Is exception handling in high-level languages? (15) Can we build a universal exception handler as a middleware?
- Categorization of faults: (16) Can we predict the percentage of data vs. logic vs. control faults?
- Testing: (17) Is there a way to standardize characterization of microprocessors? (18) Is there a better way to test than additively? (19) Is there a way to determine if we are testing correctly?
- Modeling: (20) Is there a way to model untested systems using tested systems?
- Fault injection: (21) Is it really feasible?

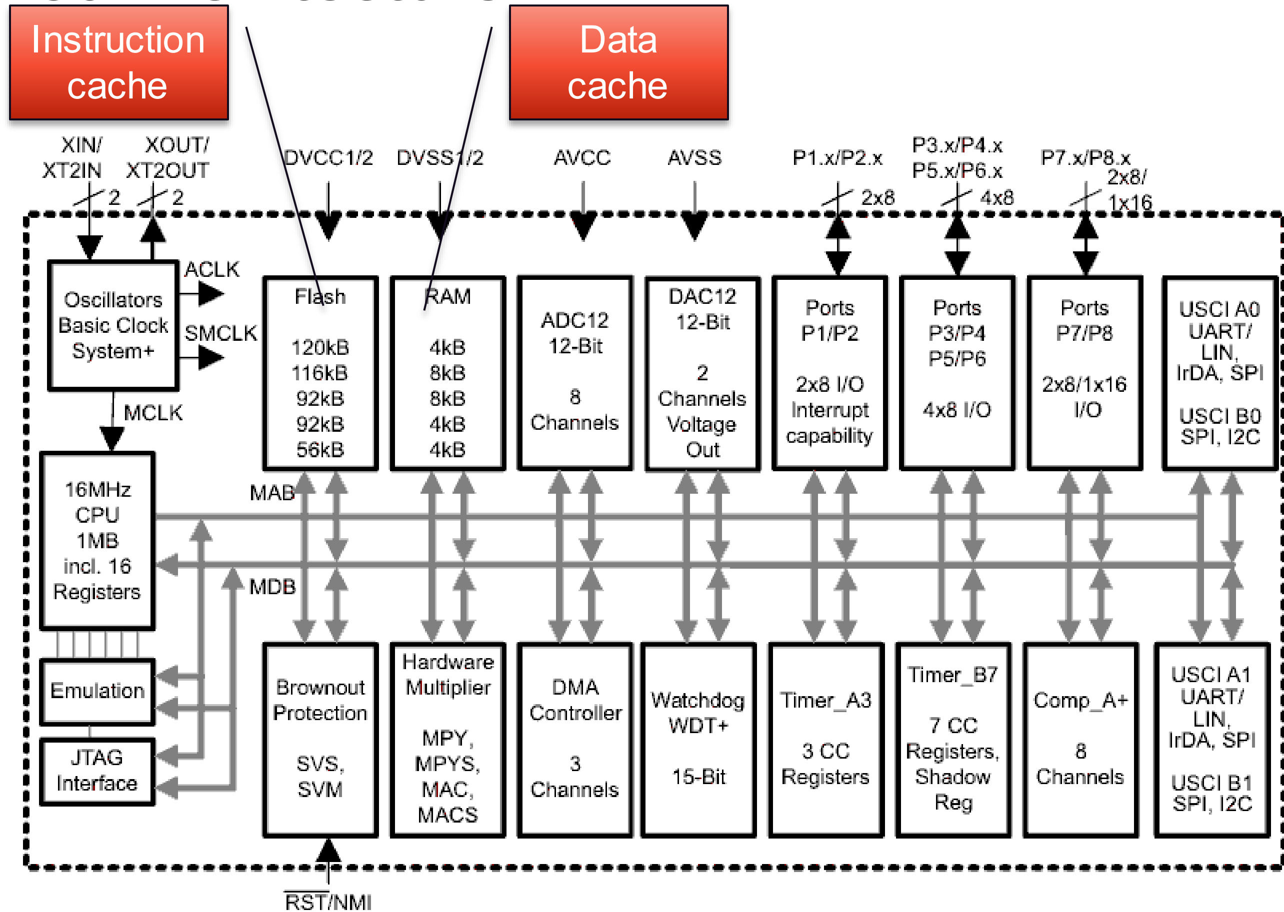
Backup Slides

Another View of Executing of Instructions



From Joel Emer's Computer Architecture Slides:
https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/
6-823-computer-system-architecture-fall-2005/lecture-notes/l15_micro_evlutn.pdf

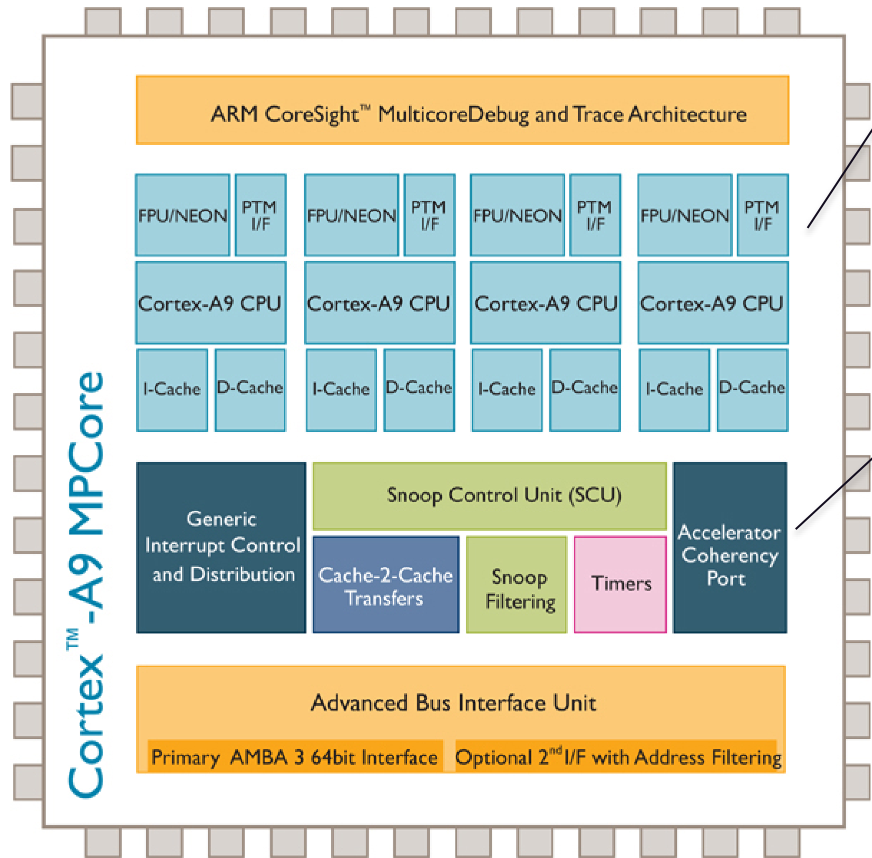
MSP430 Architecture



Several peripherals with memory mapped functionality

http://www.ti.com/product/MSP430F2619

Cortex A9 Architecture



Multiple cores with separate or shared caches

Logic for handling cache coherence

http://www.embeddedinsights.com/epd/arm/arm-cortex-a9.php

Caches

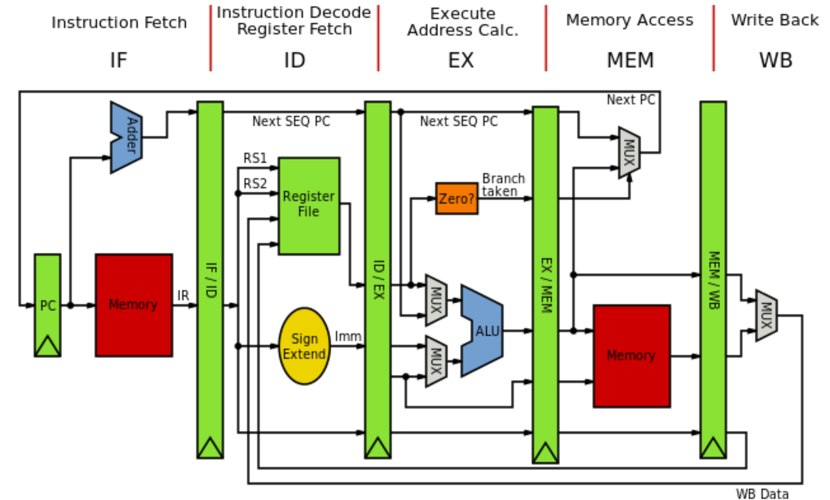
- Most microprocessors have on-chip managed caches, or unmanaged static random access memory (SRAM) blocks.
- Caches for general-purpose microprocessors:
  - Stage the data variables and instructions for quick execution
  - Can include multiple levels of cache
  - The control circuitry around a managed cache is immense
- Unmanaged SRAM for microcontrollers:
  - Compiler and the linker determine where memory values are stored
  - Memory is not staged for computation
  - Less control circuitry

Register File

- The set of memory closest to the ALU in a microprocessor
  - The fastest to access due to cell design and locality
  - The design of the memory cell is often the same, but functionality differs
- Two types of main registers: general-purpose registers for integers, and floating-point registers for floating point
  - These registers store
    - Input values before processing
    - Output data before moving back into the cache and permanent memory
    - Branch/jump location before the program moves
  - Register assignments determined by the machine code
- The special-purpose registers store the program state
  - The program counter stores the location of the next instruction to be executed
  - The instruction register stores the instruction being executed
  - The accumulator register stores the output of the ALU before it is stored in a general-purpose register
  - These registers are not part of the user-defined memory and are controlled by the microprocessor during operation of the program

Pipeline Registers and Other Flip-Flops

- Pipeline registers is the most common other form of memory:
  - Break the execution phases of the microprocessor into distinct phases (instruction fetch and decode)
  - Adders and multipliers can have registers for pipelining
- In both cases the registers play the same role: store the intermediate of the calculation so that the inputs can be changed every clock cycle
- The memory cell design for all of these registers is a latch design.
- Microprocessor could have 10-20 pipeline stages
  - Pipelining registers are used throughout the architecture, including the control logic



https://en.wikipedia.org/wiki/Computer_architecture

Potential Radiation-induced Failures in Mathematical Units/Logic

- The second largest set of units are mathematical operations
  - ALU/FPU,
  - Combinational logic
- Logic units are sensitive to single-event transients (SETs) in combinational circuitry, and single-event upsets (SEUs) in the pipelining registers
- SETs are the basis of all other SEEs
  - SETs have to be separated from SEUs by frequency
  - Also the microprocessor has to be operating to measure SETs at all

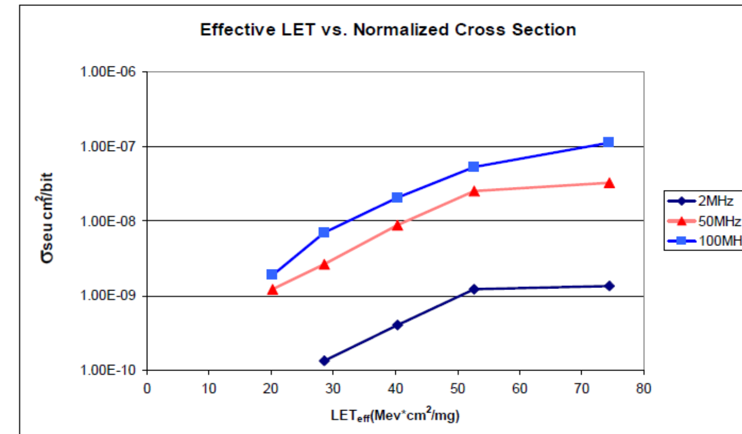
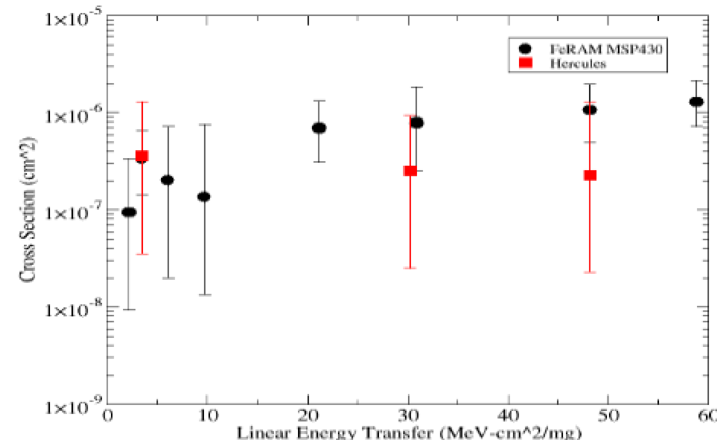


Figure 7: Comparison Cross Sections with respect to a spectrum on LET Values for Several Frequencies



https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=7004596&tag=1

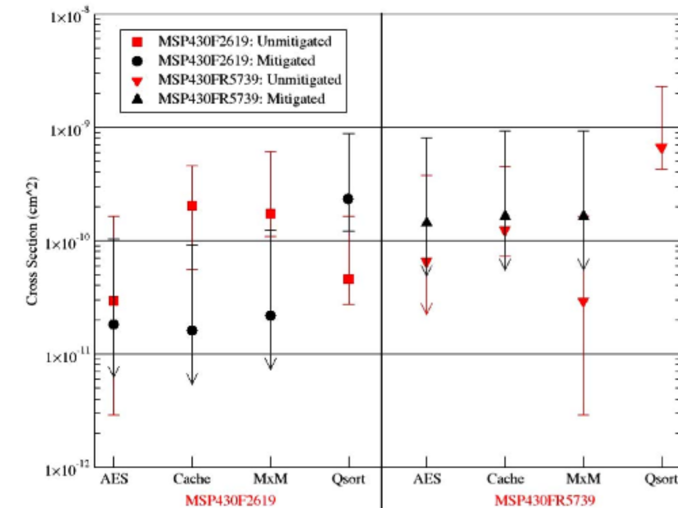
Single-Event Upsets (SEUs)

- An SEU is essentially an SET that is immediately latched
- The most common design for static memory is a cross-coupled set of inverters:
  - An SET across one inverter only needs to last long enough for the second inverter to change its output value
  - Once the second inverter changes, the SET will self-refresh until overwritten

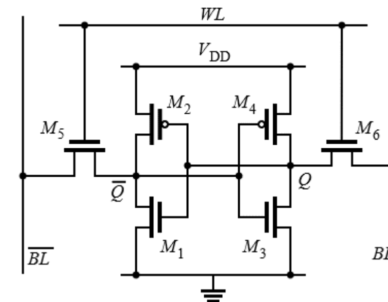
https://en.wikipedia.org/wiki/Static_random-access_memory#/media/File:SRAM_Cell_(6_Transistors).svg

Potential Radiation-induced Failures in the Memory/Datapath

- Microprocessor memory units abound:
  - Managed caches or static memory,
  - Register file,
  - Pipeline registers,
  - Buffers, and
  - Flip flops.
- Memory is sensitive to SEUs
  - An SEU is essentially an SET that is immediately latched by the memory structure
  - An SET across one inverter only needs to last long enough for the second inverter to change its output value
  - Once the second inverter changes, the SET will self-refresh until overwritten
- The total memory in a general-purpose processor (GPP) or graphics processing unit (GPU) is immense
  - SEUs in memory dominant error rate
  - Failure modes from SEUs are not only confined to changes in data variables



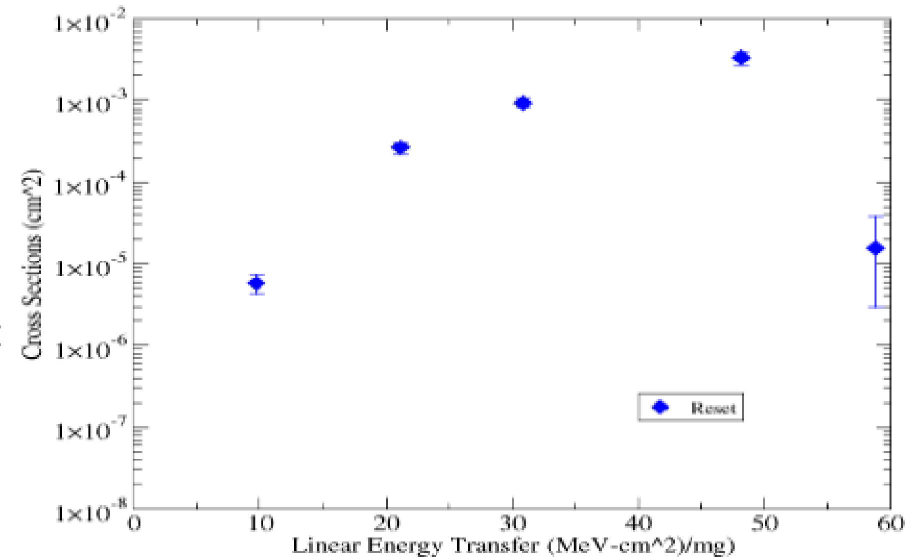
https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=7348804



https://en.wikipedia.org/wiki/Static_random-access_memory#/media/File:SRAM_Cell_(6_Transistors).svg

Potential Radiation-induced Failures in Control Units

- The control units are small, but the effect is immense, because they provide both timing and control signals to the microprocessor
  - Precise timing of registers being loaded and stored;
  - Instructions being reordered and executed;
  - Transferring data into and out caches;
  - Determining whether a branch is taken or not; and
  - Clearing out the pipeline registers when a branch is taken.
- All of the faults in these circuits are generically categorized as single-event functional interrupts (SEFIs).
  - SEFIs are caused by SEUs or SETs in control circuitry of a component
  - Components are not "self-aware" and usually need external components to detect and correct SEFI states
- Problems
  - Very little documentation on control logic
  - Very difficult to observe failures in these circuits

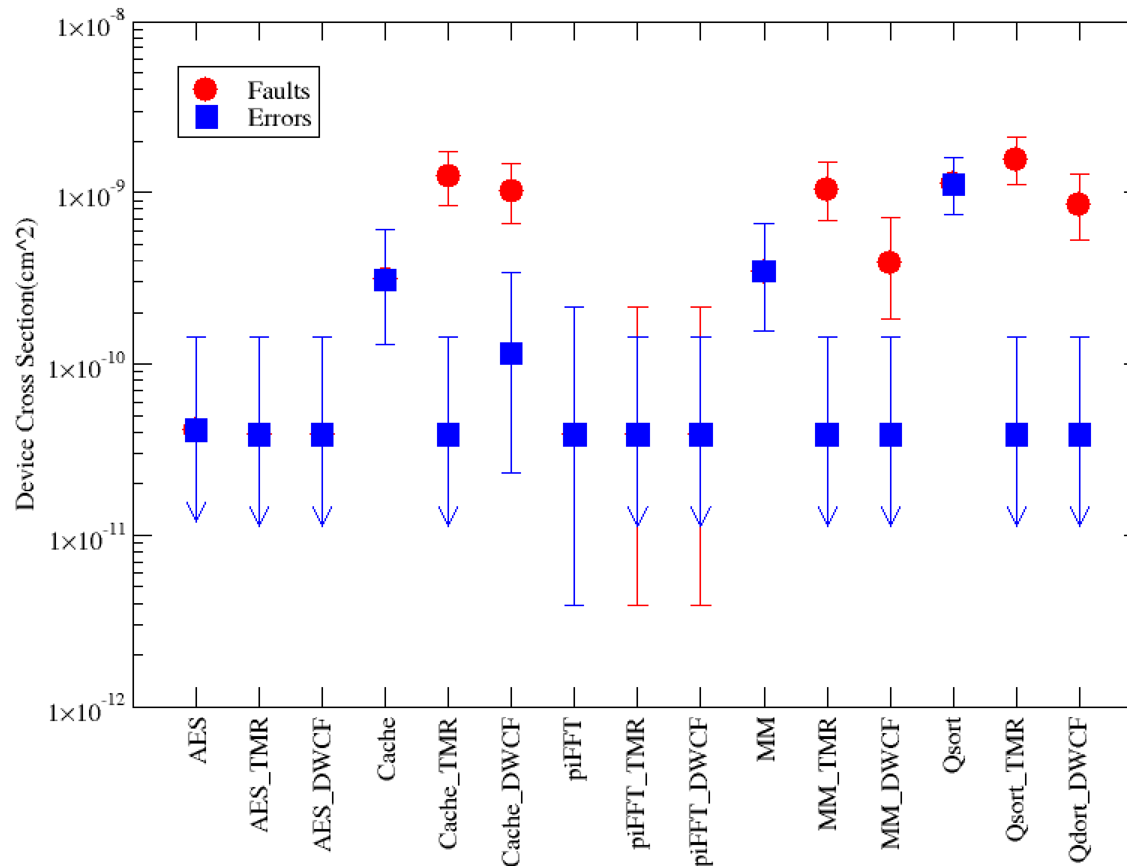


https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=7004596&tag=1

Original Focus: Creating Benchmarks Tests for Mitigation Studies

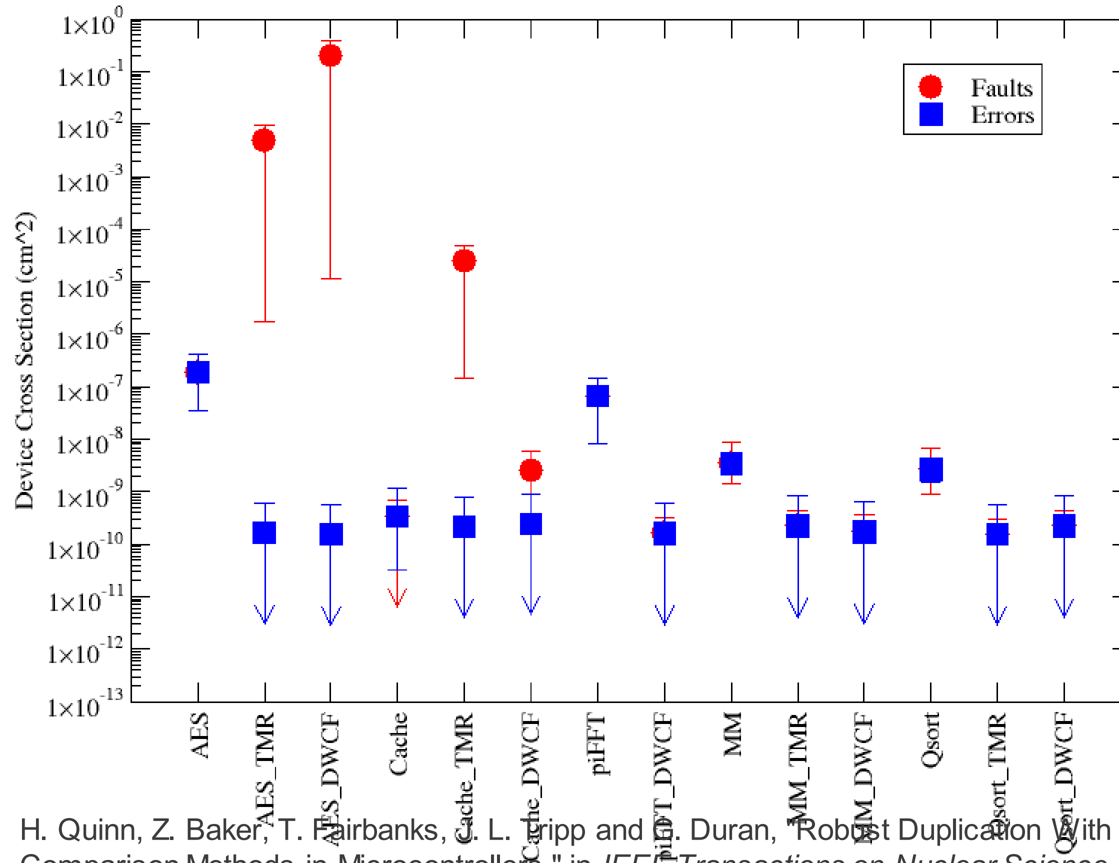
- The number of mitigation methods for FPGAs and microprocessors has been increasing rapidly since 2000
  - Need a method for comparing all of the mitigation methods
  - Need metrics to determine differences between mitigation methods
- FPGA benchmark: ITC'99
- Microprocessor benchmark:
  - AES
  - Coremark
  - LANL Cache Test
  - piFFT
  - Matrix Multiply
  - Quicksort

MSP430F2619 Radiation Test Results



H. Quinn, Z. Baker, T. Fairbanks, J. L. Tripp and G. Duran, "Robust Duplication With Comparison Methods in Microcontrollers," in *IEEE Transactions on Nuclear Science*, vol. 64, no. 1, pp. 338-345, Jan. 2017.

Zynq ARM Radiation Test Results



H. Quinn, Z. Baker, T. Fairbanks, J. L. Tripp and G. Duran, "Robust Duplication With Comparison Methods in Microcontrollers," in *IEEE Transactions on Nuclear Science*, vol. 64, no. 1, pp. 338-345, Jan. 2017.

Remaining Issue for the Original Benchmark: Inputs

- It is clear that some of the codes are very sensitive to input variations
- Matrix Multiply can mask SEUs in the inputs by multiplying by zeros – you can make Matrix Multiply appear “harder” by multiplying by lots of zeros
  - We’re testing with all zeros, all ones, and random numbers to assess how much the cross section changes based on inputs
- Qsort can mask problems with the sorting code by sorting arrays with a widely varying set of values:
  - For example, an input array of (0000, 1000, 2000, 3000, 4000) all SEUs in the lowest three nibbles will not affect the sort
  - Sorting random numbers often creates an array of numbers with too much variation
  - Sorting the input array of (0, 1, 2, 3, 4) is unrealistic
  - We’re trying a combination of both sequential and random numbers

Issues with the Microprocessor Benchmark

- We have achieved our goals:
  - Good for comparing results across architectures
  - Good for comparing results across mitigation techniques
  - Good for measuring results on common algorithms
- Besides the LANL Cache test, it was not designed for determining the basic characteristics of the microprocessor
  - It is, in fact, a bad benchmark for characterizing microprocessors
  - It cannot predict the cross section other codes
  - In fact, we specifically avoided this originally
  - Most benchmark codes are also very memory heavy, making it only good for predicting memory cross sections
- There is pressure inside and outside of the benchmark group to add codes for characterizing the microprocessor to predict how other codes behave in radiation environments

Pros and Cons of These Ideas

Operations

- Pros:
  - Closer to what we are doing now
- Cons:
  - What we are doing right now is not helpful
  - Might not be able to instrument all of operations
  - Likely a lot of assembly
  - Hard to disambiguate SEUs in cache
  - Might not be predictive for more complex systems

Machine Learning

- **Pros:**
  - Not the same method we have been trying for a decade
  - Might be more predictive for complex systems
- **Cons:**
  - Not completely certain how this will work
  - Can we use the data we already have? Do not know.
  - Will we need to test a lot of codes? Do not know.

What Is the Plan?

- Operations
- Heather is designing micro-benchmarks that focus on reducing memory access and narrowing the focus to single operations:
  - Multiply
  - Add
  - Divide
- Looking to see if we can port the LANL Cache test into a TLB Test

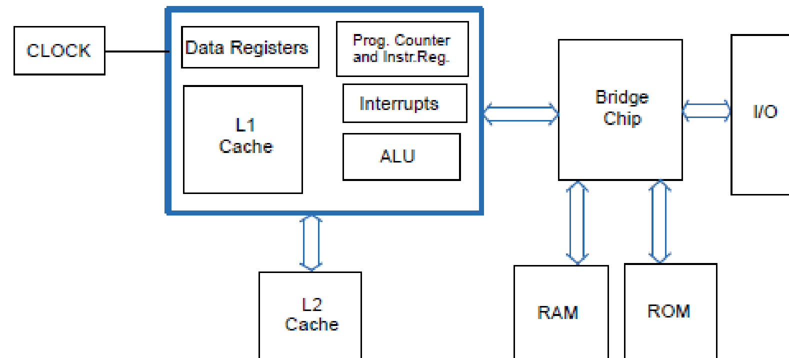
Machine Learning

- **Paolo and Steve are collecting information on a set of applications on a specific architecture**
- **The test data are used to create a model to estimate the FIT and SDC propagation behavior for untested applications**

The Basics of Characterizing Microprocessors

Characterizing Microprocessors

- Characterizing microprocessors often focuses on testing:
  - Input/output data,
  - Multiple levels of memory,
  - Multiple processing cores,
  - Multiple processing modes, and
  - Peripherals.
- To fully characterize a microprocessor it is necessary to test all of these different circuits
- Testing focuses on hardware, but software and operating systems affect how faults present in the system, and determine how faults turn into errors



F. Irom, "Guideline for Ground Radiation Testing of Microprocessors in the Space Radiation Environment," 2008.

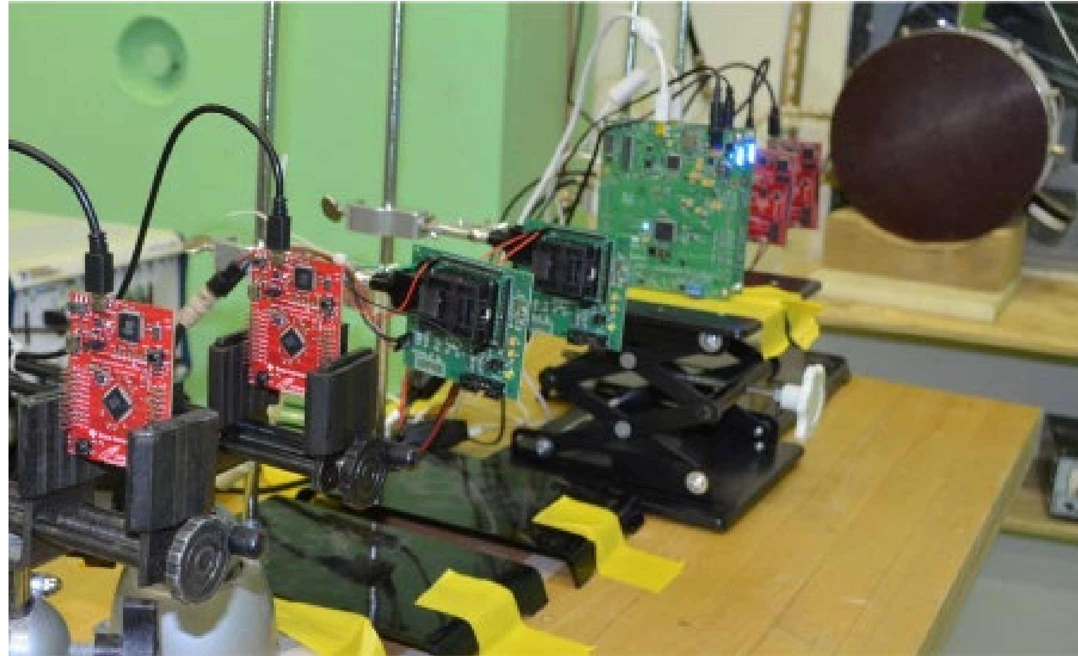
Microprocessor Test Standards

- The Jet Propulsion Laboratory (JPL) Microprocessor Test Guideline recommends testing:
  - Registers
  - Cache
  - Flight software
- The JPL System-on-a-chip (SOC) Test Guideline recommends testing:
  - Peripherals
  - Fault-tolerance circuitry
  - Radiation-hardened circuitry

[1] https://nepp.nasa.gov/DocUploads/C288941E-C4DF-486A-9ADD317D00A26BC3/07-118%20Irom_JPL%20Guideline%20for%20Ground%20Rad%20test.pdf
[2] S. M. Guertin, B. Wie, M. K. Plante, A. Berkley, L. S. Walling, and M. Cabanas-Holmen, "SEE Test Results for Maestro Microprocessor," in *RADECS*, 2012.

Hardware Test Setups

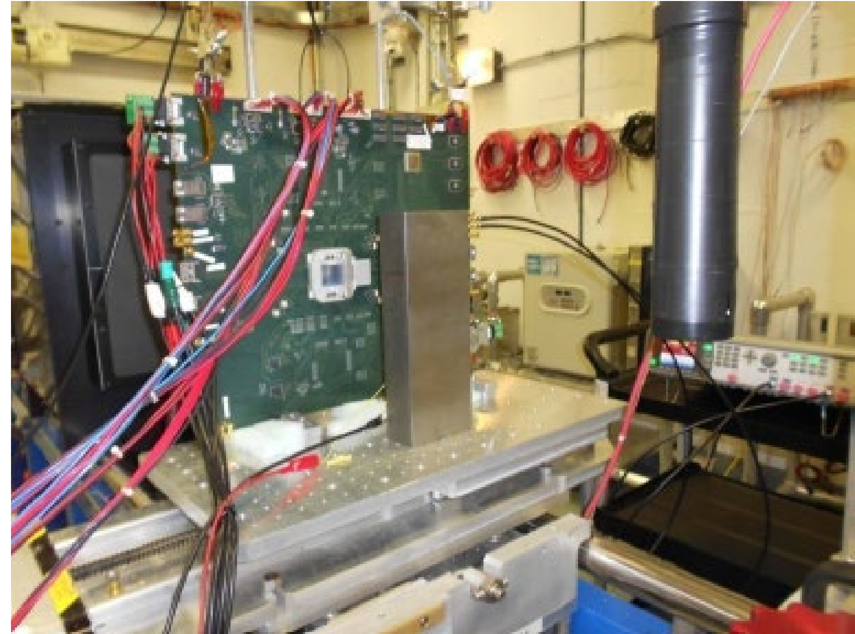
- Basic parts:
  - Test boards
  - Monitoring internal state
  - Monitoring functionality
  - Monitoring test conditions



https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=7004596&tag=1

Test Boards

- There are a wide range of test boards used to test microprocessors, including:
  - Evaluation boards,
  - Application boards,
  - Custom test boards, and
  - Desktop/laptop computers.
- Every option has advantages and disadvantages:
  - Evaluation boards: inexpensive, but limited interfaces and capabilities
  - Application boards: full stack, but complicated
  - Desktop/laptop: availability and full stack, but hard to test and complicated
  - Custom: ideal, but expensive.



S. M. Guertin, B. Wie, M. K. Plante, A. Berkley, L. S. Walling, and M. Cabanas-Holmen, "SEE Test Results for Maestro Microprocessor," in *RADECS*, 2012.

Monitoring Internal State

- Determines where faults are in the system
- Most commonly done through boundary scan:
  - Most components have a boundary scan port
  - Boundary scan provides access to some or all of the internal memory
- Joint Test Action Group (JTAG) standard is the most common boundary scan
  - JTAG implementations vary widely
- Some ARM components have Serial Wire Debug (SWD) capabilities
  - The data rate is much faster than JTAG
- Can transfer the microprocessor's state to a secondary computer during the test: possibly need debugger hardware and/or software

Monitoring Functionality

- Determines whether the output is erroneous
  - Software is outputting errors
  - Microprocessor is hung or crashed
- In theory, a secondary computer is monitoring for faulty operation
- In practice, detecting errors in real time is not simple
  - Detection process has to be fast
  - Detecting errors in random inputs is challenging
- Simplification of error detection:
  - BIBO systems
  - Self-checking test codes
  - Analyze the data offline to find the errors after the test (last resort)


```
---
hw: TMS570LS1224
test: qsort_no_ecc
mit: none
printing: 1
Array size: 2000
ver: 0.1
fac: LANSCE Aug 2018
start_time: 2018 Sep 04 06:38:35
end_time: 2018 Sep 04 06:48:35
count: 43884
LANSCE_conversion_factor: 21601
distance: 35.0
d:
# 0, 0
- i: 134231096
   E: {9141: 949,}
- i: 134231088
   E: {768: 512,}
# 10000, 2
- i: 134231096
   E: {1592: 1593,1593: 1594,1594: 1595,1595: 1596,…
- i: 134231096
   E: {1741: 9933,}
# 20000, 67
```

Monitoring Functionality and Internal State Simultaneously

- Ideally want to correlate errors to faults simultaneously
- Right now, independent researchers have not been able to fully correlate all of the faults with errors
- Full functionality monitoring is impractical for three reasons:
  - Often the boundary scan ports have been disabled
  - Transferring all the internal state in real-time to the test control computer might not be possible, making real-time monitoring difficult
  - Most microprocessor have hidden state, so a full accounting of the internal state is not possible
- Most microprocessor characterization monitors the functionality only

Test Control and Monitoring

- Test control and monitoring make the test.
- Test control provides the ability to interact with the test fixture to change the test conditions:
  - Resetting the test board when the test crashes or hangs,
  - Loading, reloading, and changing test software,
  - Changing inputs, and
  - Changing power supply conditions.
- Test monitoring provides the ability to collect results from the test fixture:
  - Collecting outputs,
  - Logging outputs to the hard drive,
  - Determining functional errors,
  - Monitoring power supply conditions, and
  - Plotting results in real-time.

Handling SEFIs

- Most microprocessors crash when exposed to radiation, which complicates test control
- The test can be resurrected by:
  - Reloading a program into the microprocessor,
  - Issuing a reset to a power-on-reset pin,
  - Power cycling, or
  - A combination of these options.
- Good test control allows these commands to be issued easily (or automatically), and quickly during the test.

Test Methodologies

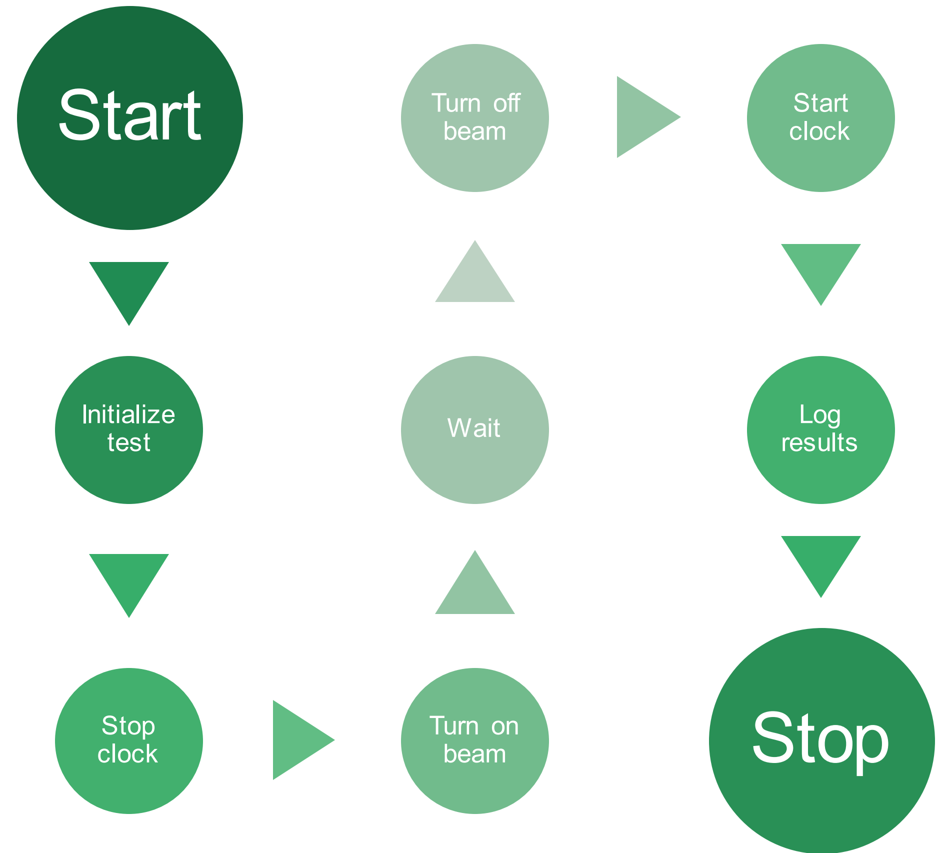
Test Methodologies

- There are three basic test methods:
  - Static
  - Semi-static
  - Dynamic
- The differences depend on:
  - Whether the part is clocked or not clocked while the beam is on,
  - Whether the microprocessor is executing instructions or not in the beam, and
  - If the microprocessor is executing instructions, whether the software being executed is meant to provide full or characteristic coverage.

Clock
Instr
Test
Protocol
Off
Limited
Op Codes
Application
Static
Semi-
Static
Dynamic
Beam On
Low Flux Beam
Beam Off

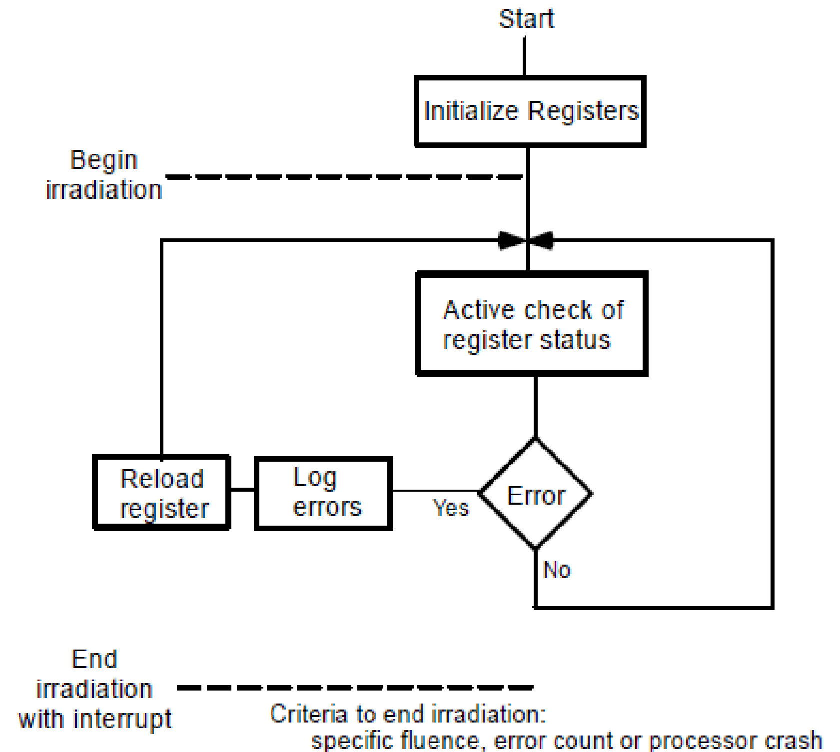
Static Testing

- Commonly used for cache/register tests in the last decade
  - Pro: Simple, great for measuring SEUs in memory, should be able to get end-to-end memory coverage, no interference from software or OS
  - Con: bad for measuring SETs, unlike normal operating conditions, unable to discriminate multiple-cell upsets (MCUs) from single-bit upsets, might need boundary scan to be implemented for the memory



Semi-static Testing

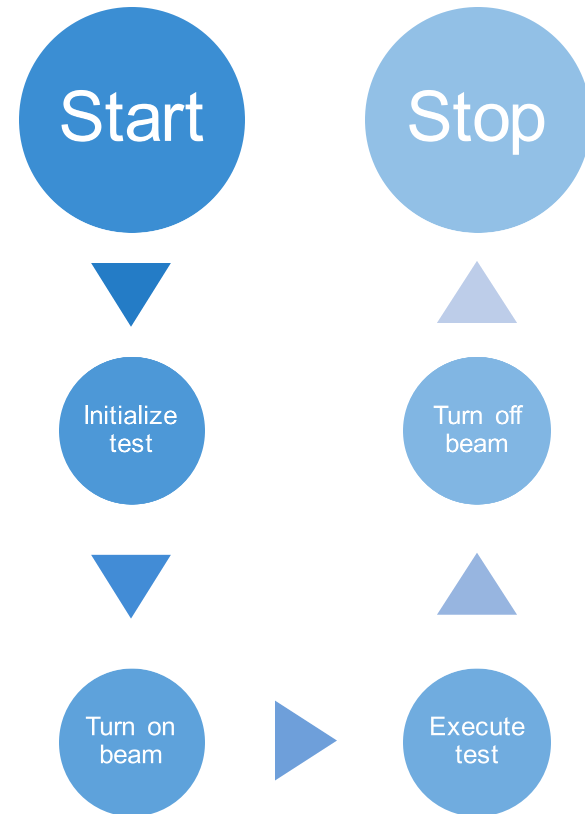
- Forms of semi-static testing have become fairly standard in recent years
  - Pro: Simple, good for SEUs in memory, can be done in assembly, might not need JTAG, more accurate accounting of behavior, able to handle MCUs, beam will be on the entire test
  - Con: Might still not get SETs, needs to be done in assembly for registers, susceptible to crashes
- Irom notes for this particular register test: “Note that this test method assumes that the processor works properly nearly all of the time during the test. It will not work effectively unless the error rate is relatively low and dominated by register errors.”
  - Likely works better as a cache test in modern architectures



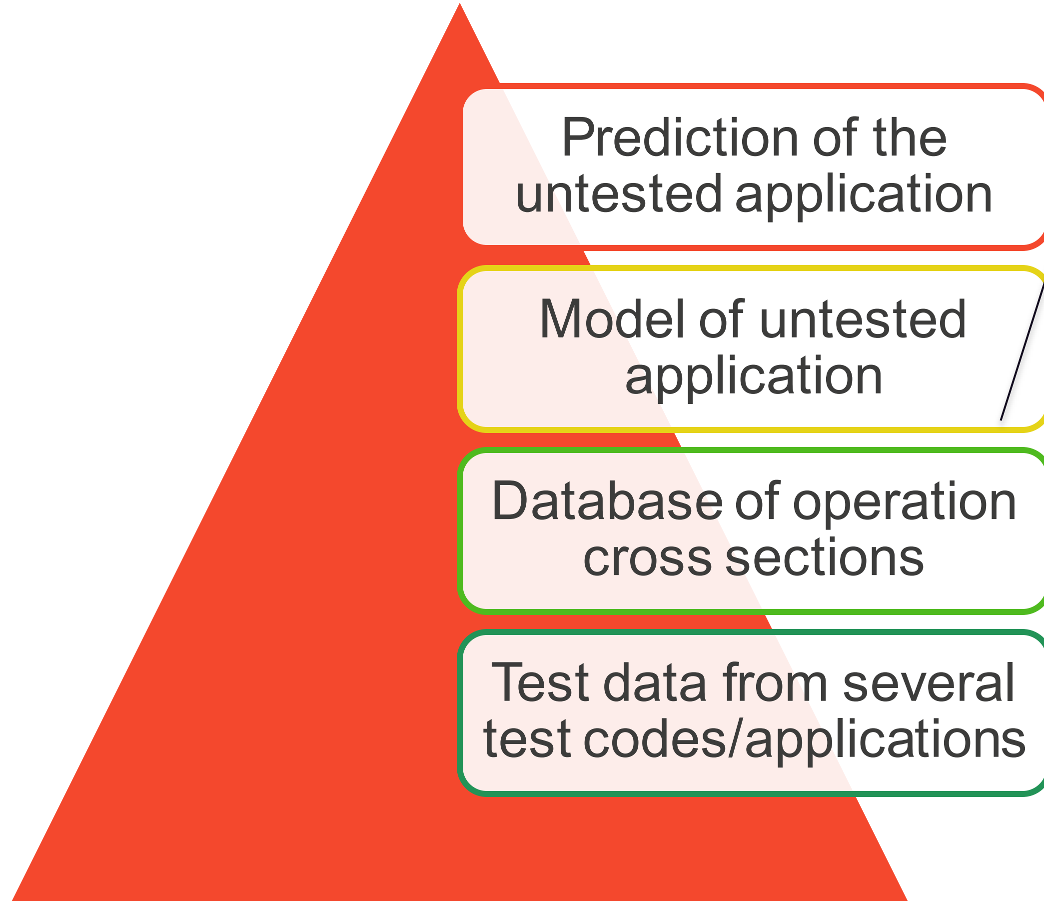
F. Irom, "Guideline for Ground Radiation Testing of Microprocessors in the Space Radiation Environment," 2008.

Dynamic Testing

- Pros: capture full behavior of the system, beam on during entire test, measure complex operations, best option for measuring SETs
- Cons: hard to distinguish/categorize failure modes, crashes, full interference from software and OS
- Dynamic testing is necessary.
  - Determine the on-orbit behavior for the microprocessor system
  - Even if the flight software is not completed in time to test, testing the operating system with some reasonable software analog to the flight software will prepare the designers for satellite operations



The Devil Is in the Details



This piece is largely not well defined at this point. The sensitivities of the operations is not enough. We also need to understand the timing of operations in the application and faults.

Testing Issues: Fault Simulation and Emulation

Fault Simulation and Emulation

- The part of this talk that is missing is a robust discussion of fault emulation and simulation techniques
  - Fault simulation methodologies model the fault's behavior in the hardware,
  - Fault emulation methodologies mimic the fault's behavior in the hardware,
  - Fault injection: where fault emulation and fault simulation can be used interchangeably.
- Fault emulation has been enormously useful for FPGA testing:
  - Better prepared for the actual radiation test
  - Can test faults uniformly
  - Can test on the bench
- So why not do more fault injection on microprocessors?

Timing, Timing, Timing!

- Microprocessors have very complex operations
  - Not all sub-components are active at all times
  - Resources like the caches and TLB are shared causing values and faults to be overwritten, causing masking of faults
  - All of these sub-components are working at MHz-GHz speeds
- The radiation effects community needs tools that mimic the radiation environment, but this can be difficult
  - Challenging to inject faults that are not on clock boundaries
  - Challenging to inject faults into sub-components that are not well understood or visible
  - Challenging to validate tools

With These Challenges Should We Give Up?

- No.

- The community needs methods for:
  - Slowing down tests so that we can see the propagation of a fault into an error
  - Determining how timing affects masking
  - Providing non-radiation methods for exploring faults
  - Predicting the behavior of untested applications. Even if we have to test them later, at least you should be able to test the best version of the application

- Let’s cover the current state of the art

Architectural Vulnerability Factors

- AVF is a simulation method for determining what part of the architecture is important or not
  - It is built on top of a model of the architecture
  - It is heavily used by the manufacturers
- AVF can determine at the architectural level the timing in which a fault is important
- After all, it is not just a matter of what you are doing but when you are doing it
  - An SEU after the last read has no consequence
  - An overwritten SEU has no consequence
- AVF culls the part of the architecture not used from the cross section

S. S. Mukherjee, C. Weaver, J. Emer, S. K. Reinhardt, and T. Austin, "A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor," in *Int. Symp. Microarchitecture (MICRO-36)*, 2003, pp. 29–40.

Program Vulnerability Factors

- PVF is a simulation method for determining what part of the program is important or not
  - It is essentially an improvement of AVF that focuses on the program
- PVF is an important improvement, because the program determines a less conservative estimate of what is being used in the architecture
- It also takes into account all of the issues with timing

- It could be that some combination of AVF and PVF can help us model the untested applications so that we can translate from the test codes to the untested application

V. Sridharan and D. R. Kaeli, “Eliminating microarchitectural dependency from architectural vulnerability,” in *Proc. 15th Int. Symp. High-Perform. Comput. Architect. (HPCA-15)*, 2009, pp. 117–128.

Fault Emulation Through Code Modification

- The most common method of inserting faults is by modifying the code to insert SEUs directly into the code or microprocessor
  - This process has the advantages of changing areas of the computation that are likely to trigger errors
  - This process cannot usually get into the non-user areas of the microprocessor, which makes it hard to determine how to scale the results
- Only of the early examples of this technique is the Code Emulated Upsets (CEU)
- CEU inserts the fault through this process:
  - Interrupting the software
  - Saving the context
  - Injects the fault directly into memory by XORing the SEU value
  - Restoring the context
  - Restarting the computation
- Most current code modification tools essentially follow the same process

R. Velazco, S. Rezgui, and R. Ecoffet, “Predicting error rate for microprocessor-based digital architectures through CEU (code emulating upsets) injection,” *IEEE Trans. Nucl. Sci.*, vol. 47, pp. 2405–2411, Dec. 2000.

Fault Emulation Through Boundary Scan

- Boundary scan ports, including the JTAG interface or the SWD debug port, have also been used to insert faults directly into microprocessors
  - This technique has the advantage of being able to insert faults through a secondary mechanism that does not affect current computation
  - This technique has the disadvantage of inserting all faults on a clock edge, which does not mimic all radiation faults, and still might not be able to inject into non-user areas
- The process is straightforward:
  - Stop the microprocessor clock
  - Using the boundary scan clock shift in the SEU
  - Restart the microprocessor clock
- As a test of the test fixture, you can often do a "poor person" version of this fault emulation technique using the debugger:
  - Stop at a breakpoint
  - Insert the fault through the memory viewer into a variable

Mitigating Microprocessors

Increasing Software Resilience

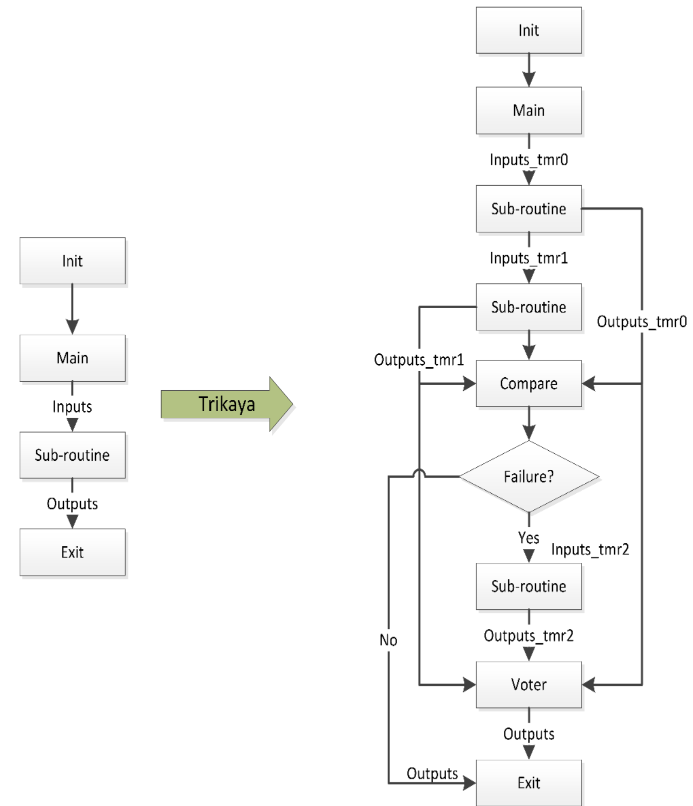
- Several years ago LANL was able to determine that modifications of the software could mask the errors caused by SEUs and SETs
  - Redundancy and majority voters are added to the program so that errors can be detected and corrected
- Programs could operate functionally through the SEU without causing computational errors or possibly crashing

DWCF Technique

- The Trikaya technique is based on spatial and temporal redundancy.
  - Spatial redundancy: replicating the sub-routine's input and output variables
  - Temporal redundancy: replicating the execution of the mitigated sub-routine
- For microprocessors that are dominated by SEUs in the data variables or SETs in the calculation, the redundancy should mask SEUs and SETs
- If there are no faults, the replicated sub-routine is executed twice with two independent data replicas
  - Comparison between outputs detects errors (duplication with compare)
  - If the two outputs do not match, then the sub-routine is executed a third time with the third replica of the data variables and a voter corrects the error (failover to TMR)
- Peripheral scrubbing is also added as part of the process to reduce issues with SEUs in the peripherals.

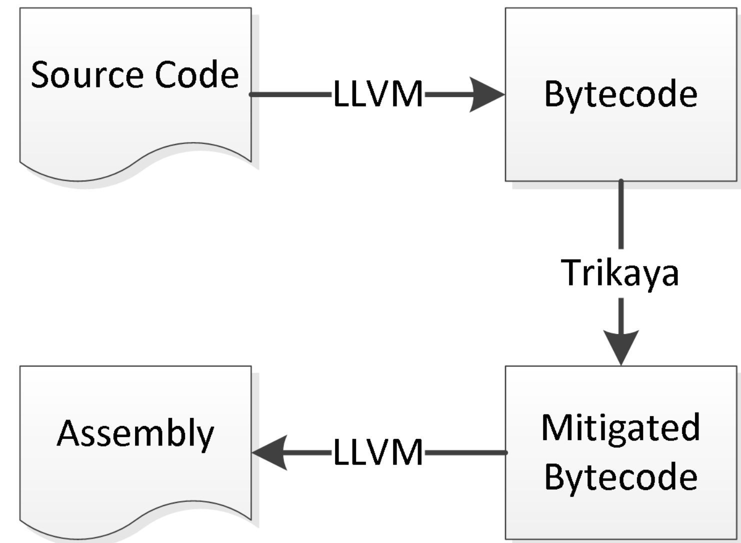
Software Modifications

- We are currently focused on sub-routine mitigation, including these insertions:
  - Replicated input and output variables;
  - Comparison code;
  - Majority voter code;
  - Peripheral scrubbing code; and
  - Code to trigger the DWCF algorithm and peripheral scrubbing
- Issues with code structure are not addressed when mitigating full sub-routines



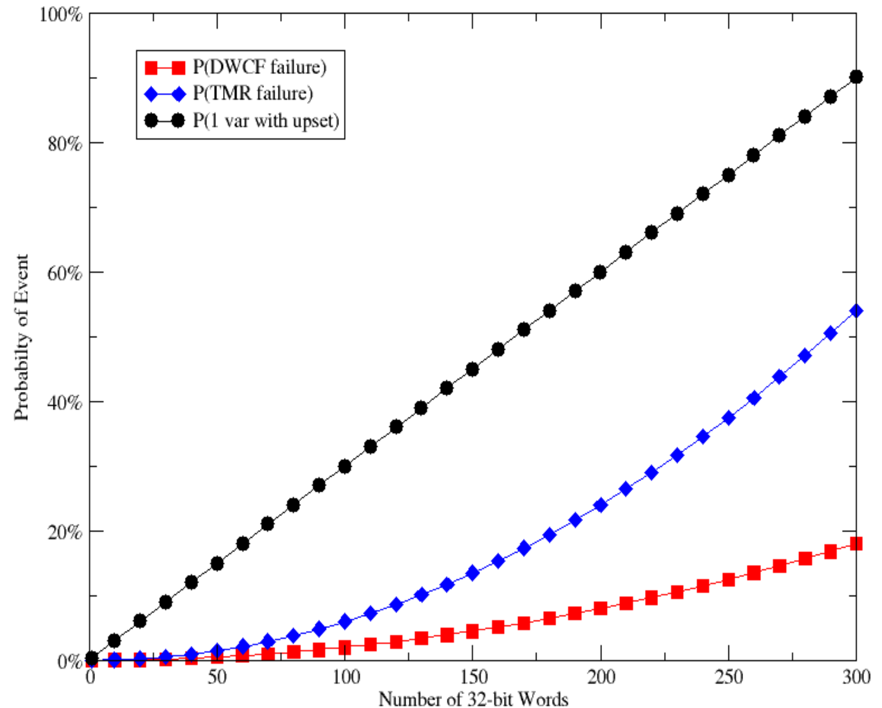
Automating Software Redundancy and Voter Insertion

- Automating the insertion of the redundant code and variables; and the voters is ideal
  - Can insert the extra code after the software has been parsed and optimized
  - Can avoid the inserted code being removed during optimization
- The LLVM compiler supports these types of modifications
  - There this a bytecode representation of the code that has been used extensively for compiler extensions
  - Can represent the bytecode as a data and control flow graph
- LLVM supports nearly all modern languages, so the automated tool can support several languages



Limitations to the DWCF technique

- Latent faults:
  - On average a third of all SEUs in the replicated data variables will affect the third replica
- MIUs
  - Two upsets in multiple copies of the same variable accumulate before the first upset is corrected



Root Causes for Failures

- Reporting shows the variety of faults possible from the software codes
- Many faults are less extensive than previously thought
- Quicksort allow us to measure whether sorting a sorted array would have a different probability of failure than sorting an unsorted array
  - The algorithm does two forward sorts followed by two reverse sorts
  - All four sorts have approximately the same number of errors
  - The location of the SEU within the affected word (MSB, LSB) determines the effect
  - MSB: could affect entire array
  - LSB: could affect a few values

Root Causes for Failures (2)

- Matrix multiply: similar results
  - Each SEU could cause the resultant matrix to have an entire row or column of faults, but only happens about half of the time
  - Rest of SEUs occur in the resultant matrix :
  - Resultant matrix is the same size as the two input matrices combined
  - SEUs are equally likely in the output than the input
  - Some cases where the SEU occurs during the calculation, causing partial failures of a column or row in the resultant matrix
- The piFFT code:
  - Malloc failures when the input variables are being instantiated on the stack causes the code to crash
  - The code is an iterative code: a number of tests did not converge