The 0th Crew Member
The computer technology needed to get us to Mars.

Tim Williamson
“Life On Mars?”
Human Biology 01
5/28/02
The Problems

When most people think about a manned mission to Mars, they marvel at the obstacles which must be overcome. Massive rockets must be built. Self-sufficient environments must be made and stockpiled with food. Exceptional people must be found who can handle the pressures of space travel. Very few people, however, consider the difficulties computer technology faces in space. Without sophisticated circuitry and programming, the trip would be impossible. A manned mission to Mars needs fast, reliable, resilient machines. What follows is a look into the problems of processing speed, radiation exposure, and software bugs that we must overcome before humans can walk on Mars. By looking back at the primitive integrated circuits of the Apollo Program and the more sophisticated computer systems of the International Space Station, it shall be shown that the technology exists today to produce powerful, radiation-resistant chips capable of getting us to the red planet. However, the success or failure of this technology — and therefore the lives of the astronauts on board — ultimately still depends on the humans who program it.

Processing Power: How Much Is Enough?

“The way of progress is neither swift nor easy.”

-- Marie Curie

The explosion of integrated circuits, the heart of all our present day technology, was sparked by NASA’s development of the Apollo Guidance Computer. In 1963, NASA’s prototyping needs accounted for 60% of all integrated circuits used in the US ("Computers" 2.4). What exactly is an integrated circuit? An IC layers transistors, resistors, wires, and capacitors on a single silicon wafer that does what these parts used to do as separate components, thereby reducing size and power requirements and increasing speed.

Smaller, faster, and more efficient electronics sound ideal for space applications. However, the first IC was created in 1959 — just three years before NASA began
considering their use ("Computers" 2.4). This led to a debate still relevant to today’s missions: how does one balance the advantages of new technology against the safety of proven technology?

Implicit in the question is the assumption that we need the new technology in the first place. Without computers, the Apollo astronauts would have had to entrust their lives to pen and pencil calculations. Obviously, more reliable mechanical solutions are vastly preferable. Unlike in low Earth orbit, Apollo astronauts couldn’t receive the necessary results over radio, thereby leaving the bulky machines on Earth: it takes 1.5 seconds for a round-trip signal between the Earth and the Moon. Landing on the Moon requires much quicker calculations ("Computers" 2.1). A trip to Mars, with its 40 minute one-way transit time, further underscores just how self-sufficient interplanetary spacecraft must be.

In the end, NASA decided the risks of using ICs were outweighed by the benefits in size, speed, and power consumption. The final chip consisted only of transistors and resistors which combined to form some 5,000 NOR gates\(^1\). It had a clock speed of 1 Mhz (Allday 177), 32 KB of fixed memory and 12 KB of erasable memory ("Computers" 2.5). Physically, the machine was about the size of a small suitcase, weighed slightly over 70 pounds, and consumed 70 watts of power (Allday 177, "Computers" 2.5). It also worked remarkably well.

In the years since, these numbers have improved considerably. In order to cut down on costs, NASA uses off-the-shelf notebook computers on the International Space Station. The ISS has several Thinkpad 760ED laptops with Pentium processors running at 133 Mhz, with 48 MB of RAM for main memory, and up to 2.4 GB of hard disk space (Uri). Each machine weighs less than 10 pounds ("TP"). That’s 133 times the clock speed at 1/7th the weight.

Of course, this is nowhere near the speed and capacity available today. Current high end processors run at greater than 2 Ghz, 512 MB of DRAM sells for less than a hundred dollars, and even 160 GB hard drives cost less than two hundred dollars. One

\(^1\) All logical statements can be represented by combinations of NOR operations.
might well wonder why top of the line machines are not used. Certainly NASA has the funds, even with recent budget cutbacks, to spend a couple thousand dollars on each machine.

NASA does not push for cutting-edge processors for the same reason the space shuttle’s antiquated circuitry has not been replaced: “if it ain’t broke, don’t fix it.” It has been decades since the creation of the shuttle’s avionics computers. Indeed, in the words of one Boeing engineer, “the avionics computers are basically obsolete by PC standards” (Lewis). However, the cost of using new machines is much more than just the cost of new hardware. Any change has to be thoroughly tested, reviewed, stressed, and run through the wringer until it can be guaranteed to function within given specifications. In fact, it would have been simpler to program the Apollo’s computer had they used a mixture of logic gates, but NASA engineers went with using only NOR gates because it increased reliability by decreasing variability. (Allday 177) The question therefore becomes not what is possible, but simply what is needed. Older hardware with its longer track record and simpler circuitry is preferable to newer hardware. If we go to Mars within the next two decades, we will likely go with 133 Mhz Pentiums along with other ICs already used in space today.

Radiation: Will The Chips Fry?

“The changing of bodies into light, and light into bodies, is very conformable to the course of Nature, which seems delighted with transmutations.”

-- Sir Isaac Newton

Imagine the house of a mad plumber, full of showers, sinks, and pipes strewn about everywhere. Surprisingly enough, this house is similar to a computer chip in many ways. Instead of electrons flowing, we have water. Instead of wires and resistors, we have large and small pipes. Instead of capacitors (which hold charge for a small amount of time), we have sinks (which slowly drain over time). Instead of transistors (which can be turned on and off and only allow current to flow one way) we
have hundreds of showers (which have a knob that can be turned on and off and only allow water to flow from the shower head to the drain, but not vice versa). Now that we’ve got this crazy house, imagine it is haunted by unpredictable and destructive poltergeists. This is the situation a computer chip faces when exposed to the radiation of space.

There are two primary radiation sources (Carson “Intro”). The Sun produces electromagnetic radiation in the form of particles such as X-rays and gamma rays. Galactic Cosmic Rays from other stars in the universe produce ionizing radiation primarily in the form of protons and occasionally as atomic nuclei, electrons, and gamma radiation (Draganic 145). Additionally, solar winds (the product of bursts of activity on the sun) can produce large streams of protons similar to GCR. (Draganic 147) These two forms of radiation in turn can cause three different types of damage to electronic components: Single Event Effects, or SEEs; Total Ionizing Dose effects, or TIDs; and Displacement Damage Dose effects, or DDDs (LaBel).

DDDs occur when a high energy particle smacks into an electronic component. Microchips embed their circuitry in a lattice work of silicon atoms. When the particle smacks into one of these atoms, it can displace or bump it. This creates a gradual degradation of the chip over time (Carson “Effects”). To use the plumber’s house analogy, DDDs are like ghosts which twist the walls and foundation of the house. Given enough time, the warpage causes pipes to burst, allowing water to run in unpredictable and unintended ways.

TIDs are very similar to DDDs. They slowly ionize the chip, which is to say they change the balance of energy, making some switches easier to turn on and others harder to turn off (Carson “Effects”). TIDs are like poltergeists who tighten or loosen the shower knobs of the house. This makes the water flow with even the slightest touch or makes it very difficult turn the knob on at all. Unless the plumber allows for such variability, the result is a house where nobody knows what is on and what is off.

SEEs happen when the collision of a radiation particle with a chip produces
enough energy to cause a change in a component near the impact site. This can cause simple “soft errors” such as a “0” becoming a “1,” or can create latch-ups or burn-outs. A latch-up is like a persistent error; a “0” permanently becomes a “1,” regardless of whatever input the chip receives. A burnout, as the name implies, is the literal burning or melting of all or part of the chip due to an intense amount of energy (Carson “Effects”). In our plumber’s house, SEEs are like ghosts who play with the water pressure. Sometimes they turn a tap on when we’re not expecting it. Other times, they make sure the water is always running in certain pipes. Worst of all, they occasionally create a massive pressure surge, which breaks the pipes.

The Apollo’s computer did not have to worry about such ghosts. The size of the on-chip components were huge by today’s standard. Like the difference between a house’s plumbing and a city’s sewer system, this alone makes the IC much less susceptible to all but the most serious radiation events. Additionally, the trip to the Moon does not take very long, making the chip’s exposure time relatively small.

In contrast, satellites and machines aboard the ISS stay exposed for extended periods\(^2\). Most SEEs can be accounted for in software using error correction algorithms. The basic idea behind this strategy is that for every piece of information stored in memory, some additional data is also stored that allows the program to verify the value is still correct. For example, the simplest such algorithm consists of making a copy of every bit of information. If the original and the copy are not the same, then you know an error has occurred. Error correction is an old problem in computer science: efficient and robust algorithms have been developed to deal with errors in hard disks, modem transmissions, and supercomputing clusters, all of which can be directly transferred to space applications. While this approach works well for events which affect memory, events affecting the computational elements of the chip remain a problem. Fortunately, manufacturers have created a variety of ways to “radiation harden” equipment.

\(^2\) It should be noted that the ISS is still within part of Earth’s atmosphere, which acts as a partial shield against radiation. Geosynchronous satellites, however, are well beyond any such protection.
The easiest is a trick long used in processor fabrication: stress the chips and throw away the ones that fail. Despite the best efforts of manufacturers, each chip is minutely different from every other chip. The tiny differences can make it more or less resistant to radiation. Simply by exposing the chips to radiation, the ones which continue to work can be separated from the ones that don’t. This is often more cost-effective than creating special radiation-resistant chips, since off-the-shelf parts can be used (LaBel).

Of slightly greater difficulty is a method in which the insulation between layers of the chip can be made thinner. This leaves less material that can be ionized, thus reducing the risk of TIDs. Unfortunately, this decreases the acceptable temperatures and number of imperfections that can occur during the manufacture of the chip, making it much more difficult to produce (Perrine).

The most complex way to harden a chip involves restructuring components to increase their isolation from one another. This can be done by repositioning various component junctions or by using silicon-on-sapphire or silicon-on-insulator technologies (Perrine). Increasing the separation between circuit elements reduces the likelihood that the energy of a radiation strike in one part of the chip will affect enough of the circuit to be a problem. Additionally, on-chip wires can be made thicker, allowing for greater current to flow through should a high-energy particle strike (Perrine).

These last methods further underscore how, in terms of radiation tolerance, older chips are generally preferable over newer chips. The connections of today’s chips are so small that IBM has started to use silicon on insulator technology just to keep the chip from interfering with itself (“Silicon”). If the circuitry needs such techniques simply to function under normal conditions, they cannot be used as a means to increase radiation tolerance.

For the International Space Station, NASA decided to use off-the-shelf-parts. They went with the simplest method of radiation hardening: testing the chips and throwing out the bad ones. In the end, this approach costs less money than designing
entirely new chips (Carson “Hardening”).

As was the case with processing speed, we see that the primary technology needed for prolonged exposure to radiation matured many years ago. As far as integrated circuits go, manned missions to Mars will probably not use any technology developed within the last 5 years. Older chips are simply more resistant to radiation than newer ones — and they cost less.

Software: Have We Caught All The Bugs?

"Beware of bugs in the above code; I have only proved it correct, not tried it."

-- Donald Knuth

In many ways, the question of whether we have rugged enough computer hardware to send humans to Mars is a moot point. Pathfinder went to Mars, landed on Mars, and even traveled about the Martian surface without any mechanical or electrical failures. However, the same cannot be said of its software.

Errors in software are human errors. They cannot be blamed on cosmic rays or meteor strikes. They cannot be blamed on corroded wires or loose connections. When computer code is written, the environment in which it will run — the computer chip — is known, and the results it must produce are known. Therefore, the programmer is accountable for any failures to generate these results.

All too often, simple transcription errors cost billions of dollars. Mariner 1, the US’s first attempt at an interplanetary spacecraft, blew up just seconds after liftoff due to such an error. An unexpected problem caused a primary control system to fail during liftoff. The craft was designed to handle the situation by switching over to a backup system which used a different means to determine the craft’s position and velocity. This backup received data from two external systems whose physical location produced a difference in timing of 43 milliseconds between them. In order to compensate, the backup code needed to average data it received about the rate of change of a radius in one of the control equations. The mathematicians writing the
equations out long hand should have used a symbol which looked like $\bar{X}_n$. (The bar
stands for average.) Instead, they left out the bar while working, and forgot to put it
back in. Therefore, the final code did not take the average of the data. This made it
seem like the rocket’s ascent was erratic when in reality it was not, causing the program
to “correct” for the nonexistent abnormalities. These incorrect corrections caused the
rocket to actually veer off course until ground control had to destroy it for safety
reasons. Ironically, had the backup system not kicked in, the rocket may have launched
successfully on its own (Neuman).

An even more heartbreaking story involves Russia’s first attempt to reach Mars. In
the sending of 20 to 30 pages worth of instructions to the Phobos I spacecraft, a
single character was omitted. By an “unbelievably small chance,” the resulting
instructions order the craft to “commit suicide” (Fiske).

One may think that any modern mission would be immune to such simple
transcription errors. On December 3, 1999, however, NASA made an even more
blatant mistake programming the Mars Polar Lander, causing the spacecraft to crash
into the planet. According to the Mars Climate Orbiter Mission Failure Investigation
Board, the “‘root cause’ of the loss of the spacecraft was the failed translation of English
units into metric units in a segment of ground-based, navigation-related mission
software” (“Mars”).

Many detractors of a manned Mars mission point to the “human factor” as a
reason against such a project. They are referring to the supposed unpredictability of, or
possibility of error by, astronauts on board the spacecraft. Incidents such as those
above suggest we may have just as much — if not more — to fear from the “human
factor” of those back on Earth. Any Mars mission must have extreme policies and
procedures to ensure that every piece of information that makes it into the computer
system is accurate and correct.

Unfortunately, transcription errors such as these are the “easy” ones catch. Far
more difficult are problems like those of the ARIANE 5 rocket. Costing 7 billion dollars to develop, the rocket blew up on its first launch, vaporizing another 500 million dollars worth of rocket and payload in the process (Arnold). Responsibility fell on an incorrectly programmed conversion from a 64-bit floating point number to a 16-bit signed integer value. The code did not correctly handle cases when the value in the 64-bit number was greater than the largest value that the 16-bit value could be (Lions 2.1). Amidst thousands of lines of correct code and data, such small, subtle, and relatively infrequent mistakes simply do not stand out.

All the preceding examples are child’s play, however, compared to what happened on the Mars Pathfinder mission. While the Sojourner rover worked flawlessly on the surface, the Pathfinder base began experiencing unexpected system resets not long after the landing. While not presenting a significant danger to the mission, these resets did cause scientific data to be lost, so NASA engineers began to try to debug the mystery. Fortunately, they had a replica system back on Earth, which could be configured to record detailed records of everything the processor did. Since they had no idea what was causing the problem, the scientists spent hours running the simulation with various environmental conditions. Finally, late at night, the single remaining engineer managed to recreate the reset (Jones).

Understanding the problem is almost as hard as finding a solution. Pathfinder’s software used multithreading. Multithreading is similar to how most people use their personal computers: several different applications are all running “at once,” and the processor shares time between them. A non-multithreaded system allows only one program to run at a time. Some programs have a higher priority than others. For example, an MP3 playing program might always get to use the processor before a word processing program. This keeps the music from skipping and is unnoticeable to the user, who cannot detect a few millisecond delay. This works well in most situations. However, there are some system resources which only one program can use at a time.

3 Multithreading actually applies to threads of execution within a single program, but the concept is very similar to separate programs running under a single operating system.
If our word processing program needed to beep at the user, it would have to ask the system to give it exclusive access to the speakers. This would momentarily interrupt the MP3 playing program, which would have to wait to regain control of the speakers.

Keeping this in mind, we can now return to the Pathfinder error. By tracing through the situation on their special debugging setup, NASA engineers discovered that the resets were caused by a classic multithreading bug known as “priority inversion.” A low-priority task responsible for gathering meteorological data occasionally had to ask for exclusive access to the chip’s data bus in order to “publish” the information it had gathered to the rest of the system. Occasionally, a higher priority task that needed to use the bus to publish its own information would interrupt the meteorological task (Jones). Fortunately, the system was smart enough to realize that the higher priority thread needed to use the same resource that the lower priority task was using. It therefore told the high priority task to wait. The problem occurred when a medium priority communications task just happened to run at the precise moment when the high priority task was waiting for the low priority task. Since the communications task had a higher priority than the meteorological one and did not need access to the data bus, the system told the meteorological task to wait while the communications task executed. This in turn caused the high priority task to wait for longer than it normally would. An error correction system noticed that the high priority task had apparently stalled, decided that something had gone terribly wrong, and reset the chip (Jones). That was the cause of the mysterious resets.

Calling this problem confusing does not do it justice. It underscores a fundamental rule of all software development: the more complex a program is, the more bugs it contains. It also shows the importance of failing safe. The International Space Station contains over 3.5 millions of lines of code created by several different countries (Gross). Given such a mass of code, it is hardly surprising that it has experienced “glitches.” Although these problems appear to have been caused by failures of two hard drives (Carreau), the inability of the system to recover leads one to
question: why doesn’t some of that code handle such conditions? The ISS is fortunate enough to be close to Earth, where NASA engineers can easily monitor the system, make adjustments, or even send up replacement parts. In a Martian mission, the time-lag of communications puts a much greater strain on the spacecraft’s software.

The Final Analysis

The hardware needed to go to Mars is available to us today. Software poses the greatest difficulty. After building our massive rockets, stocking our self-sufficient habitats, and choosing our exceptional crew, we can have faith the computer technology on board will be fast enough and resilient enough to process our programs. Years of experience have given us the knowledge needed to make powerful and capable processors able to withstand the harsh radiation of space. In the end, however, the success or failure of the mission may depend on the actions of a single individual, sitting alone in a cubicle years before the launch, typing instructions on his computer, and making an accidental mistake.
Sources Cited


<http://www.ima.umn.edu/~arnold/disasters/ariane.html>


<http://www.mse.vt.edu/faculty/hendricks/mse4206/projects97/group02/radharm.htm>

<http://www.hq.nasa.gov/office/pao/History/computers/Part1.html>


<http://catless.ncl.ac.uk/Risks/7.53.html#subj2.1>

<http://catless.ncl.ac.uk/Risks/21.37.html#subj5>

<http://catless.ncl.ac.uk/Risks/19.49.html#subj1.1>


<http://spaceflight.nasa.gov/history/shuttle-mir/science/iss/sc-iss-tpcs.htm>