Corrupted Machine Code

CPE310 Microprocessor Systems - Project

The Ohm Squad

Austin Driggs, Nate Kirby, Tucker Wilson, Ava Milano

Due: 2025–04-07

INTRODUCTION

ByteForge’s testing division has encountered a serious issue that could jeopardize the Titan-9 launch. A batch of compiled firmware has been corrupted due to a faulty EEPROM writer, causing bit flips in certain machine instructions. These errors could lead to unintended behavior in the Titan-9’s control system, and it is now your responsibility to analyze and correct the issue. Your task is to take the provided corrupted machine code, identify where the bit flips have occurred, and restore the original instructions.

The corrupted machine code can be found here.

To accomplish this, you will need to carefully inspect the binary representation of each instruction and compare it to valid opcode and register formats. By analyzing patterns and detecting anomalies in the bit sequences, you can determine which fields—such as opcodes, registers, or immediate values—have been altered. Once the errors have been identified, you must apply the necessary corrections to restore the proper machine instructions. After fixing the bit flips, you will run the corrected machine code through your disassembler to verify that it translates back into the expected assembly instructions.

PROMPT

Below is the machine code where there is a single bit flip in the following fields

```
00000001001010100101000000100000      ->      Rd field
00100001000010110000000000000101
00000001001010100110000000100100
00110001110011010000000000001111      ->      Rs field
00010001101000000000000000110100
00010101011011010000000000110100
00000001110010000000000000011010
00000000000000100111100000010000      ->      Rt field
00000000000000011000000000010010
00000001001010100000000000011000
00000001011011001100100000100001      ->      Funct field
00110111001100000000000011111111
00000001000010011000100000101010
00101001000100100000000000001010
00000001001010101001100000100010
00111100000101110100000000000000
10001110111101000000000000000000
10001110111101010000000000000100      ->      opcode field
00111100000101100001001000110100
```

SIMULATING

In order to troubleshoot the code, we decided to enter each line into our MIPS Translatron 3000 machine. This gave us the output shown below:

```
Enter Binary:
> 00000001001010100101000000100000
ADD $t2, $t1, $t2

Enter Binary:
> 00100001000010110000000000000101
ADDI $t3, $t0, #0x5

Enter Binary:
> 00000001001010100110000000100100
AND $t4, $t1, $t2

Enter Binary:
> 00110001110011010000000000001111
ANDI $t5, $t6, #0xF

Enter Binary:
> 00010001101000000000000000110100
BEQ $t5, $zero, #0x34

Enter Binary:
> 00010101101011010000000000110100
BNE $t5, $t3, #0x34

Enter Binary:
> 00000001110010000000000000011010
ERROR: The given instruction was not recognized

Enter Binary:
> 00000000000000100111100000010000
ERROR: The given instruction was not recognized

Enter Binary:
> 00000000000000001100000000010010
MFLO $t8
```

```
Enter Binary:
> 00000001001010100000000000011000
MULT $t1, $t2

Enter Binary:
> 00000010110110011001000000100001
ERROR: The given instruction was not recognized

Enter Binary:
> 00110111001100000000000011111111
ORI $s0, $t9, #0xFF

Enter Binary:
> 00000001000010011000100000101010
SLT $s1, $t0, $t1

Enter Binary:
> 00101001000100100000000000001010
SLTI $s2, $t0, #0xA

Enter Binary:
> 00000001001010101001100000100010
SUB $s3, $t1, $t2

Enter Binary:
> 00111100000101110100000000000000
LUI $s7, $zero, #0x4000

Enter Binary:
> 10001110111101000000000000000000
LW $s4, #0x0($s7)

Enter Binary:
> 10001110111101010000000000000100
LW $s5, #0x4($s7)

Enter Binary:
> 00111100000101100001001000110100
LUI $s6, $zero, #0x1234
```

DEBUGGING

Using these results and errors, we were able to debug the corrupted code to yield the following code with the corrected bits highlighted:

Line 1: Rd field
Enter Binary:
> 000000 01001 01010 01000 00000 100000
ADD $t0, $t1, $t2

Explanation: Because we were told a bit was flipped here in the Rd It could be any register.

Line 4: Rs field
Enter Binary:
> 001100 01010 01101 0000000000001111
ANDI $t5, $t2, #0xF

Explanation: One bit in the Rs field was flipped and seeing as any other bit didn't make sense the one highlighted was changed.

Line 7: Additional error found
Enter Binary:
> 000000 01110 01000 00000 00000 011000
MULT $t6, $t0

Explanation: The bit flip correctly identifies the function as MULT.

Line 8: Rt field
Enter Binary:
```
> 000000 00000 00000 01111 00000 010000
MFHI $t7
```

Explanation: Even though the error said that the given function was not recognized, we knew that the actual bitflip was in the Rt field, so we examined functions to try and match up the opcode (000000) and the function code (010000), and found that MFHI matched the most. The section of code with the bitflip is not used, and thus, should be all 0’s.

Line 11: Function field
Enter Binary:
```
> 000000 01011 01100 01001 00000100000
ADD $t9, $t3, $t4
```

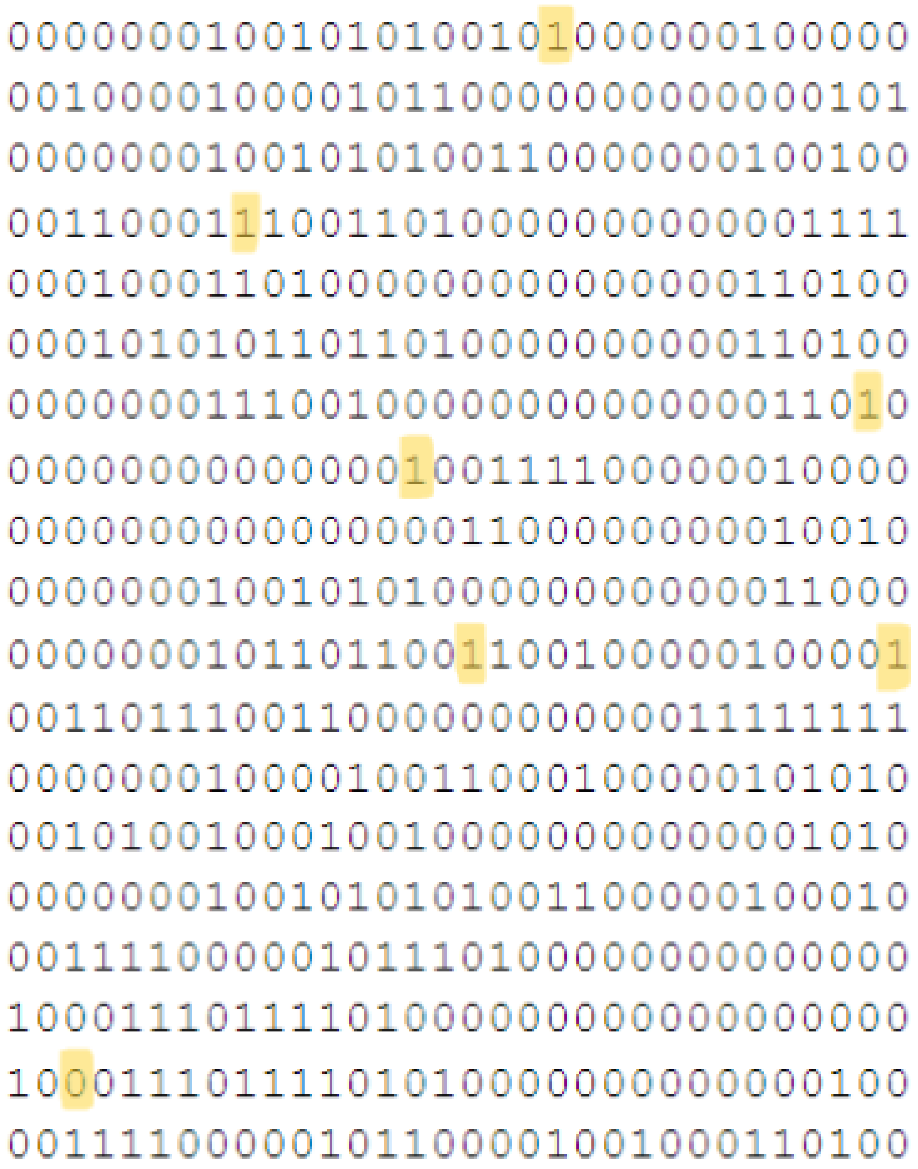
Explanation: The first bit flip is to identify that section of code as a register, and the second bitflip corrects the function code to be ADD.

Line 18: Opcode field
Enter Binary:
```
> 101011 10111 10101 0000000000000100
SW $s5, #0x4($s7)
```

Explanation: Since we know the bit flip is in the opcode, we know that it has to be that one because the only other option from LW is SW.

CONCLUSION

A summary of the results is shown below, with all of the incorrect bit flips highlighted:



```
00000001001010100101000000100000
00100001000010110000000000000101
00000001001010100110000000100100
00110001110011010000000000001111
00010001101000000000000000110100
00010101011011010000000000110100
00000001110010000000000000011010
00000000000000100111100000010000
00000000000000001100000000010010
00000001001010100000000000011000
00000001011011001100100000100001
00110111001100000000000011111111
00000001000010011000100000101010
00101001000100100000000000001010
00000001001010101001100000100010
00111100000101110100000000000000
10001110111101000000000000000000
10001110111101010000000000000100
00111100000101100001001000110100
```

The new, corrected binary and assembly code is shown below:

00000001001010100100000000100000	=	ADD $t0, $t1, $t2
00100001000010110000000000000101	=	ADDI $t3, $t0, #0x5
00000001001010100110000000100100	=	AND $t4, $t1, $t2
00110001010011010000000000001111	=	ANDI $t5, $t2, #0xF
00010001101000000000000000110100	=	BEQ $t5, $zero, #0x34
00010101101101100000000000110100	=	BNE $t5, $t3, #0x34
00000001110010000000000000011000	=	MULT $t6, $t0
00000000000000000011110000010000	=	MFHI $t7
00000000000000001100000000010010	=	MFLO $t8
00000001001010100000000000011000	=	MULT $t1, $t2
00000001011011000100100000100000	=	ADD $t9, $t3, $t4
00110111001100000000000011111111	=	ORI $s0, $t9, #0xFF
00000001000010011000100000101010	=	SLT $s1, $t0, $t1
00101001000100100000000000001010	=	SLTI $s2, $t0, #0xA
00000001001010101001100000100010	=	SUB $s3, $t1, $t2
00111100000101110100000000000000	=	LUI $s7, $zero, #0x4000
10001110111101000000000000000000	=	LW $s4, #0x0($s7)
10101110111101010000000000000100	=	SW $s5, #0x4($s7)
00111100000101100001001000110100	=	LUI $s6, $zero, #0x1234