Here is the MIPS binary code you need to dis-assembly:

x"00400000" => x"20010001"

x"00400004" => x"20020002"

x"00400008" => x"20030003"

x"0040000C" => x"20040004"

x"00400010" => x"20050005"

x"00400014" => x"20060006"

x"00400018" => x"20070007"

x"0040001C" => x"20080008"

x"00400020" => x"20090009"

x"00400024" => x"200A000A"

x"00400028" => x"200B000B"

x"0040002C" => x"200C000C"

x"00400030" => x"200D000D"

x"00400034" => x"200E000E"

x"00400038" => x"200F000F"

x"0040003C" => x"20100010"

x"00400040" => x"20110001"

x"00400044" => x"20120002"

x"00400048" => x"20130004"

x"0040004C" => x"20140008"

x"00400050" => x"20150010"

x"00400054" => x"20160020"

x"00400058" => x"20170040"

x"0040005C" => x"20180080"

x"00400060" => x"20190100"

x"00400064" => x"201B0200"

x"00400068" => x"201C0400"

x"0040006C" => x"201D0800"

x"00400070" => x"201E1000"

x"00400074" => x"201F2000"

x"00400078" => x"02018020"

x"0040007C" => x"02028020"

x"00400080" => x"02038020"

x"00400084" => x"02048020"

x"00400088" => x"00000000"

x"0040008C" => x"00000000"

x"00400090" => x"00000000"

x"00400094" => x"00220820"

x"00400098" => x"00000000"

x"0040009C" => x"00000000"

x"004000A0" => x"00230820"

x"004000A4" => x"00000000"

x"004000A8" => x"00000000"

x"004000AC" => x"00810820"

x"004000B0" => x"00000000"

x"004000B4" => x"00000000"

x"004000B8" => x"00A10820"

x"004000BC" => x"00000000"

x"004000C0" => x"00000000"

x"004000C4" => x"00260820"

x"004000C8" => x"00000000"

x"004000CC" => x"00000000"

x"004000D0" => x"00E83820"

x"004000D4" => x"00000000"

x"004000D8" => x"00000000"

x"004000DC" => x"00E90820"

x"004000E0" => x"00000000"

x"004000E4" => x"00000000"

x"004000E8" => x"002A0820"

x"004000EC" => x"00000000"

x"004000F0" => x"00000000"

x"004000F4" => x"002B0820"

x"004000F8" => x"00000000"

x"004000FC" => x"00000000"

x"00400100" => x"002C0820"

x"00400104" => x"00000000"

x"00400108" => x"00000000"

x"0040010C" => x"002D0820"

x"00400110" => x"00000000"

x"00400114" => x"00000000"

x"00400118" => x"002E0820"

x"0040011C" => x"00000000"

x"00400120" => x"00000000"

x"00400124" => x"002F0820"

x"00400128" => x"00000000"

x"0040012C" => x"00000000"

x"00400130" => x"00008020"

x"00400134" => x"200F0003"

x"00400138" => x"00000000"

x"0040013C" => x"00000000"

x"00400140" => x"00000000"

x"00400144" => x"22100001"

x"00400148" = > x"00000000"

x"0040014C" => x"21EFFFFF"

x"00400150” = > x”11E00004"

x"00400154” = > x”00000000"

x"00400158” = > x”00000000"

x"0040015C” = > x”08100051"

x"00400160” = > x”00000000"

x"00400164” = > x”08100059"

x"00400168” = > x”00000000"

x"0040016C” = > x”00000000"

See questions below:

1. What does this code do?

The assembly program described above initializes a group of variables (instruction #1 to #16). Then a sequence of addition operations is made on the registers initialized in those commands.

Finally, the program compares a register (\*) to zero and performs a jump until reaching the end of the program.

Other than the technical side, this code actually perform several steps that utilize the different parts of the MIPS CPU, using the “state” of the CPU (for example, using the same register at several different cases, reset its value and reuse it).

An improper/invalid implementation of the CPU will fail to execute instructions as expected.

For a full disassembly – see the last section in the current document.

1. Explain how this code tests the GPR\_file and ALU parts of a MIPS CPU. What is not covered?

The GPR file and ALU are tested by performing R-type commands (**add** and **addi**) that require reading the data from registers and manipulating their values by adding the contents of either immediates (\*) or other regsiters. In addition, we check that the ALU does indeed perform addition and the comparison operation for the jump if equal zero. Other ALU operations are not checked, therefore we can not be certain they are operating correctly. we did not cover the logical and / or / xor / nand / sub /and SLT using the given program.

**Disassembly of the given program**

; Initialize the variables to 1 till 16

addi $at,$zero,1

addi $v0,$zero,2

addi $v1,$zero,3

addi $a0,$zero,4

addi $a1,$zero,5

addi $a2,$zero,6

addi $a3,$zero,7

addi $t0,$zero,8

addi $t1,$zero,9

addi $t2,$zero,10

addi $t3,$zero,11

addi $t4,$zero,12

addi $t5,$zero,13

addi $t6,$zero,14

addi $t7,$zero,15

addi $s0,$zero,16

; Initialize the rest of the registers to power of 2

addi $s1,$zero,1

addi $s2,$zero,2

addi $s3,$zero,4

addi $s4,$zero,8

addi $s5,$zero,16

addi $s6,$zero,32

addi $s7,$zero,64

addi $t8,$zero,128

addi $t9,$zero,256

addi $k1,$zero,512

addi $gp,$zero,1024

addi $sp,$zero,2048

addi $s8,$zero,4096

addi $ra,$zero,8192

add $s0,$s0,$at ; s0 = 16 + 1

add $s0,$s0,$v0 ; s0 = 17 + 4

add $s0,$s0,$v1 ; s0 = 21 + 3

add $s0,$s0,$a0 ; s0 = 24 + 4

add $at,$at,$v0 ; at = 1 + 4

add $at,$at,$v1 ; at = 5 + 3

add $at,$a0,$at ; at = 4 + 8

add $at,$a1,$at ; at = 5 + 12

add $at,$at,$a2 ; at = 17 + 6

add $a3,$a3,$t0 ; a3 = 7 + 8

add $at,$a3,$t1 ; at = 15 + 9

add $at,$at,$t2 ; at = 24 + 10

add $at,$at,$t3 ; at = 34 + 11

add $at,$at,$t4 ; at = 45 + 12

add $at,$at,$t5 ; at = 57 + 13

add $at,$at,$t6 ; at = 70 + 14

add $at,$at,$t7 ; at = 84 + 15

add $s0,$zero,$zero ; s0 = 0

addi $t7,$zero,3 ; t7 = 0 + 3

addi $s0,$s0,1 ; s0 = 0 + 1

nop ; no op

addi $t7,$t7,-1 ; t7 = 3 - 1

beqz $t7,0x400164 ; branch if equal zero - will not jump **(\*)**

nop ; no op

nop ; no op

j 0x400144 ; jump to "addi $s0, $s0, 1"

nop ; no op

j 0x400164 ; jump to self

nop ; no op

nop ; no op