Project 4 - MIPS Machine Code - amfahe25 Aidan Fahey

collaboration.txt

```
I COLLABORATED WITH CHARLIE. I DID NOT PROVIDE ANY ASSISTANCE TO HIM AT ALL, HE HELPED ME
```

Part 1: example.txt (looping program, slightly modified)

```
example.txt
v3.0 hex words addressed
# Above line is needed by logisim, below is MIPS machine code and comments
00000: 24 0f b9 b1  # addiu r15, r0, -17999  // put constant -17999 (decimal) into r15
00004: 01 e0 38 21  # addu r7, r15, r0       // copy that same value to r7
00008: 00 00 10 21  # addu r2, r0, r0        // set r2 to zero
0000c: 30 e6 00 01  # andi r6, r7, 0x0001    // bitwise AND r7 with a constant, put result in
    r6
00010: 00 46 10 21  # addu r2, r2, r6        // compute r2+r6, put result in r2
00014: 00 07 38 42  # srl r7, r7, 1          // shift r7 right by one bit, put result in r7
00018: 08 00 00 03  # j c                    // jump to c
# 000010 00000000000000000000000011
```

Part 2: array.txt (another looping program, with memory access)

```
array.txt
v3.0 hex words addressed
000000: 34 04 00 20 # ori r4, r0, 0x0020    0011 0100 0000 0100
000004: 34 05 00 50 # ori r5, r0, 0x0050    0011 0100 0000 0101
000008: 34 06 00 05 # ori r6, r0, 5
00000c: 00 00 50 21 # addu r10, r0, r0      0000 0000 0000 0000 0101 0000 0010 0001

#000010: 8c ab 00 05 # lw r11, 0(r5)  // bne, below, lands here: this is the first instruction
    to be repeated 1000 1100 1010 1011
000010: 8c ab 00 00 # lw r11, 0(r5)  // bne, below, lands here: this is the first instruction
    to be repeated 1000 1100 1010 1011
# Note: kwalsh changed immediate from 0005 to 0000

000014: 01 4b 50 21 # addu r10, r10, r11    0000 0001 0100 1011 0101 0000 0010 0001
000018: ac 8b 00 00 # sw r11, 0(r4)         1010 1100 1000 1011
00001c: 24 84 00 04 # addiu r4, r4, 4       001001 00100 00100
000020: 24 a3 00 04 # addiu r5, r5, 4       001001 00101 00101
000024: 24 c6 ff ff # addiu r6, r6, -1      0010 0100 1100 0110
000028: 14 c0 ff f9 # bne r6, r0, -7  // this should (sometimes) skip back to repeat the last 7
     instructions 0001 0100 1100 0000
```

```
data.txt
v3.0 hex words addressed
00000: 00 00 00 00 00 00 00 00
00008: 00 00 00 00 00 00 00 00
00010: 00 00 00 00 00 00 00 00
00018: 00 00 00 00 00 00 00 00
00020: 33 33 33 33 00 00 00 07
00028: 00 00 00 25 ff ff ff ff
00030: 00 00 00 08 00 00 00 00
00038: 00 00 00 00 00 00 00 00
00040: 00 00 00 00 00 00 00 00
00048: 00 00 00 00 00 00 00 00
00050: 00 00 00 07 00 00 00 05
00058: 00 00 10 00 00 00 00 01
00060: ff ff ff ff 00 00 00 00
00068: 00 00 00 00 00 00 00 00
```

```
simulate -diffmem -nofuzz -regnums -logisim example.txt data.txt
---- begin execution log ----
Loading logisim-style memory image 'array.txt' ...
Loading logisim-style memory image '../2-array-data.txt' ...
Executing MIPS code...
MIPS program output will be in PLAIN TEXT.
Simulator messages will be in PLAIN TEXT as well.
Note: Each character of user input is shown in curly-braces, e.g. {H}{i}{!}{\n}
```

```
* 0x00000000: 34040020: ori r4, r0, 32  # hex 0x0020, ascii ' '
* 0x00000004: 34050050: ori r5, r0, 80  # hex 0x0050, ascii 'P'
* 0x00000008: 34060005: ori r6, r0, 5
* 0x0000000c: 00005021: addu r10, r0, r0
* 0x00000010: 8cab0000: lw r11, 0(r5)
* 0x00000014: 014b5021: addu r10, r10, r11
* 0x00000018: ac8b0000: sw r11, 0(r4)
* 0x0000001c: 24840004: addiu r4, r4, 4
* 0x00000020: 24a30004: addiu r3, r5, 4
* 0x00000024: 24c6ffff: addiu r6, r6, -1  # hex 0xffffffff
* 0x00000028: 14c0fff9: bne r6, r0, -7  # address 0x00000010
* 0x00000010: 8cab0000: lw r11, 0(r5)
* 0x00000014: 014b5021: addu r10, r10, r11
* 0x00000018: ac8b0000: sw r11, 0(r4)
* 0x0000001c: 24840004: addiu r4, r4, 4
* 0x00000020: 24a30004: addiu r3, r5, 4
* 0x00000024: 24c6ffff: addiu r6, r6, -1  # hex 0xffffffff
* 0x00000028: 14c0fff9: bne r6, r0, -7  # address 0x00000010
* 0x00000010: 8cab0000: lw r11, 0(r5)
* 0x00000014: 014b5021: addu r10, r10, r11
* 0x00000018: ac8b0000: sw r11, 0(r4)
* 0x0000001c: 24840004: addiu r4, r4, 4
* 0x00000020: 24a30004: addiu r3, r5, 4
* 0x00000024: 24c6ffff: addiu r6, r6, -1  # hex 0xffffffff
* 0x00000028: 14c0fff9: bne r6, r0, -7  # address 0x00000010
* 0x00000010: 8cab0000: lw r11, 0(r5)
* 0x00000014: 014b5021: addu r10, r10, r11
* 0x00000018: ac8b0000: sw r11, 0(r4)
* 0x0000001c: 24840004: addiu r4, r4, 4
* 0x00000020: 24a30004: addiu r3, r5, 4
* 0x00000024: 24c6ffff: addiu r6, r6, -1  # hex 0xffffffff
* 0x00000028: 14c0fff9: bne r6, r0, -7  # address 0x00000010
* 0x00000010: 8cab0000: lw r11, 0(r5)
* 0x00000014: 014b5021: addu r10, r10, r11
* 0x00000018: ac8b0000: sw r11, 0(r4)
* 0x0000001c: 24840004: addiu r4, r4, 4
* 0x00000020: 24a30004: addiu r3, r5, 4
* 0x00000024: 24c6ffff: addiu r6, r6, -1  # hex 0xffffffff
* 0x00000028: 14c0fff9: bne r6, r0, -7  # address 0x00000010
* 0x0000002c: 0000000d: break

MIPS code finishes with result $v0 = 0x00000000 (decimal 0, unsigned 0, char '\0').
MIPS processor executed approx. 40 instructions in 23708 nsec at 1.1687 MHz.
Final state of registers:
    r0 = 0x00000000     r8 = 0x00000000    r16 = 0x00000000    r24 = 0x00000000
    r1 = 0x00000000     r9 = 0x00000000    r17 = 0x00000000    r25 = 0x00000000
    r2 = 0x00000000    r10 = 0x00000023    r18 = 0x00000000    r26 = 0x00000000
    r3 = 0x00000054    r11 = 0x00000007    r19 = 0x00000000    r27 = 0x00000000
    r4 = 0x00000034    r12 = 0x00000000    r20 = 0x00000000    r28 = 0x00000000
    r5 = 0x00000050    r13 = 0x00000000    r21 = 0x00000000    r29 = 0x00000000
    r6 = 0x00000000    r14 = 0x00000000    r22 = 0x00000000    r30 = 0x00000000
    r7 = 0x00000000    r15 = 0x00000000    r23 = 0x00000000    r31 = 0xfffffffc
   $hi = 0x00000000    $pc = 0x0000002c
   $lo = 0x00000000   $npc = 0x00000030
All data memory locations written by program:
00000020: 0000 0007 0000 0007 0000 0007 0000 0007  ................
00000030: 0000 0007 .... .... .... .... .... ....  ....
----- end execution log -----
```

I collaborated with Charlie Youssef (He helped me, I provided zero assistance to him)

1.) Red is the register being altered, blue is the first argument, pink is the second argument, yellow means jump

2.) andi causes the ZI input to be selected, and it is 0x0001

3.) addiu , SI is -17999

4.) ZI is unsigned, SI is signed

5.) It adds the zero in the PC absolute, and JS adds the address into the register

6.) It is dividing the value in register 7 by 2

8.) The comparison happens in Branchcalc, sends -7*4 to the branch offset which gets added to current address + 4 and sends it to the register file

9.) It did 5 iterations of the loop, it stopped looping when r6 was 0. It is only based on r6 so nothing makes it have more or less iterations

10.) It is copying the data from lines 50-60 in the data memory to lines 20-30 in the data memory, and in the register file it is adding every four bites from 50-60 into r10