Lab 5 Report

Maheedhar Mandapati
Joanna John

12/7/2019

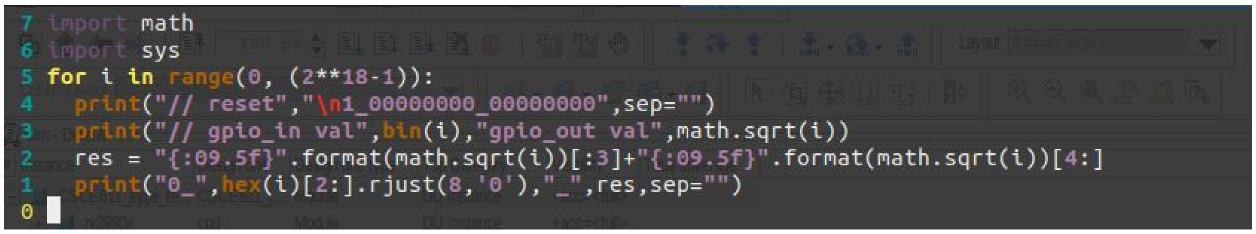
Testing Strategy

For our testing strategy, we took the given MIPS code in lecture that tests all the new instruction we implemented in lab 5 and ran it with our CPU. This was in order to test our implemented MIPS instructions in the CPU. Our test vectors are 68 bits long, the highest 4 bits were used to flag the reset; the lowest 32 bits was used for the expected output from our CPU; and the rest of the 32 bits in the middle are the GPIO in. Then the input test vectors were generated from the python code shown below in the Justification section. With this strategy, our program can have any number [0,2^18) as input. We tested all possible numbers in this range.

When we designed the test bench, we did it to take in our vectors and separate them into parts: reset, GPIO in, and GPIO expected output. If reset is flagged we pass in that flag to the CPU and wait 10 nanoseconds. This 10 nanosecond wait is to give the CPU time to process the flag and complete the reset command. If reset is not flagged then we move on to the next 32 bits, the GPIO in. We pass that in and wait 4500 nanoseconds. This 4500 nanosecond wait is because that is the amount of time our MIPS square root program takes to run. We found this run time by running one test vector and recording the time. Since each test vector is running the same code we determined that this time frame can be used as the limit for all test vectors that we run . Then the test bench checks if the CPU output is equivalent to the GPIO in expected output. If the CPU output and expected output were not equal we print out the CPU output, expected output and that there was an error resulting in an unexpected output. Along with this we print which vector resulted in giving the error.

For the CPU design, it takes in the clock and reset signals as well as the GPIO in and returns GPIO out. The CPU then initializes the ALU module as well as a register file module. We implemented the control unit inside the CPU. IT then loads the given MIPS code into the CPU memory to be compared to the CPU output later. Starting at the top, the MIPS code is executed one instruction at a time. The total of these instructions are run for 4500 nanoseconds and the resulting output is compared with the expected output.

Justification

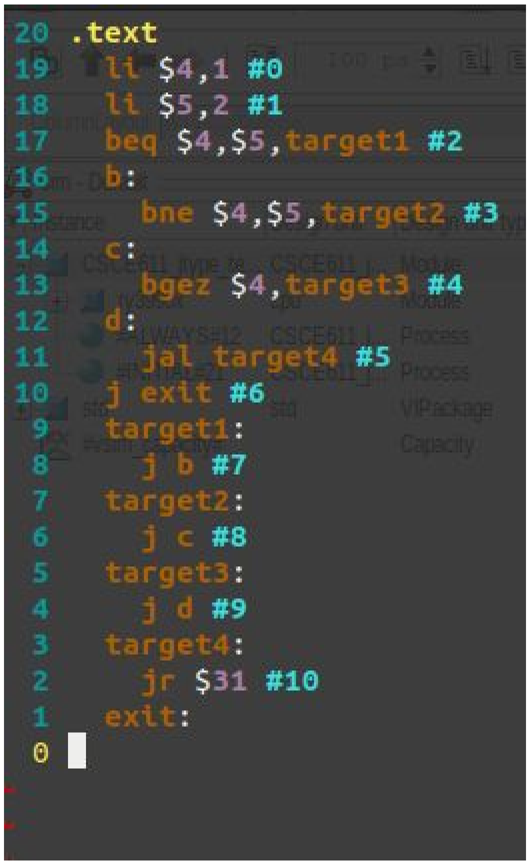


```python
import math
import sys
for i in range(0, (2**18-1)):
  print("// reset","\n1_00000000_00000000",sep="")
  print("// gpio_in val",bin(i),"gpio_out val",math.sqrt(i))
  res = "{:09.5f}".format(math.sqrt(i))[:3]+"{:09.5f}".format(math.sqrt(i))[4:]
  print("0_",hex(i)[2:].rjust(8,'0'),"_",res,sep="")
```

Lab 5 Report

Maheedhar Mandapati
Joanna John

12/7/2019



```
.text
  li $4,1 #0
  li $5,2 #1
  beq $4,$5,target1 #2
  b:
    bne $4,$5,target2 #3
  c:
    bgez $4,target3 #4
  d:
    jal target4 #5
  j exit #6
  target1:
    j b #7
  target2:
    j c #8
  target3:
    j d #9
  target4:
    jr $31 #10
  exit:
```

MIPS test file used to test out instruction implementation of the branch and jump statements. File name: test.asm Hex file: test.dat

Analysis

When we ran out design with the tester MIPS file we receive desired outputs for all the branch and jump statements. Our CPU such as it is now can run all the necessary MIPS instruction without any hitches. We also ran the required square root MIPS code on our design and got matching results to our outputs in MARS. However the way we have chosen to calculate the square root only give us a precision of ~7x10^-5 instead of the required 1x10^-5 we have spent hours trying to fix this to no avail. Our square root program also does not procure the desired results for any input value above 0x20000. We think this is due to some bit shifting issue.

Test Vectors

```
// reset
1_00000000_00000000
```

Lab 5 Report

Maheedhar Mandapati
Joanna John

12/7/2019

```
// gpio_in val 0b0 gpio_out val 0.0
0_00000000_00000000
// reset
1_00000000_00000000
// gpio_in val 0b1 gpio_out val 1.0
0_00000001_00100000
// reset
1_00000000_00000000
// gpio_in val 0b10 gpio_out val 1.4142135623730951
0_00000002_00141421
// reset
1_00000000_00000000
// gpio_in val 0b11 gpio_out val 1.7320508075688772
0_00000003_00173205
// reset
1_00000000_00000000
// gpio_in val 0b100 gpio_out val 2.0
0_00000004_00200000
// reset
1_00000000_00000000
// gpio_in val 0b101 gpio_out val 2.23606797749979
0_00000005_00223607
// reset
1_00000000_00000000
// gpio_in val 0b110 gpio_out val 2.449489742783178
0_00000006_00244949
// reset
1_00000000_00000000
// gpio_in val 0b111 gpio_out val 2.6457513110645907
0_00000007_00264575
// reset
1_00000000_00000000
// gpio_in val 0b1000 gpio_out val 2.8284271247461903
0_00000008_00282843
// reset
1_00000000_00000000
// gpio_in val 0b1001 gpio_out val 3.0
0_00000009_00300000
// reset
```

Lab 5 Report

Maheedhar Mandapati
Joanna John

12/7/2019

```
1_00000000_00000000
// gpio_in val  gpio_out val 3.1622776601683795
0b10100_0000000a_00316228
// reset
1_00000000_00000000
// gpio_in val 0b1011 gpio_out val 3.3166247903554
0_0000000b_00331662
// reset
1_00000000_00000000
// gpio_in val 0b1100 gpio_out val 3.4641016151377544
0_0000000c_00346410
// reset
1_00000000_00000000
// gpio_in val 0b1101 gpio_out val 3.605551275463989
0_0000000d_00360555
// reset
1_00000000_00000000
// gpio_in val 0b1110 gpio_out val 3.7416573867739413
0_0000000e_00374166
// reset
1_00000000_00000000
// gpio_in val 0b1111 gpio_out val 3.872983346207417
0_0000000f_00387298
// reset
1_00000000_00000000
// gpio_in val 0b10000 gpio_out val 4.0
0_00000010_00400000
// reset
1_00000000_00000000
// gpio_in val 0b10001 gpio_out val 4.123105625617661
0_00000011_00412311
// reset
1_00000000_00000000
// gpio_in val 0b10010 gpio_out val 4.242640687119285
0_00000012_00424264
// reset
1_00000000_00000000
// gpio_in val 0b10011 gpio_out val 4.358898943540674
0_00000013_00435890
```

Lab 5 Report

Maheedhar Mandapati
Joanna John

12/7/2019

```
// reset
1_00000000_00000000
// gpio_in val 0b10100 gpio_out val 4.47213595499958
0_00000014_00447214
// reset
1_00000000_00000000
// gpio_in val 0b10101 gpio_out val 4.58257569495584
0_00000015_00458258
// reset
1_00000000_00000000
// gpio_in val 0b10110 gpio_out val 4.69041575982343
0_00000016_00469042
// reset
1_00000000_00000000
// gpio_in val 0b10111 gpio_out val 4.795831523312719
0_00000017_00479583
// reset
1_00000000_00000000
// gpio_in val 0b11000 gpio_out val 4.898979485566356
0_00000018_00489898
// reset
1_00000000_00000000
// gpio_in val 0b11001 gpio_out val 5.0
0_00000019_00500000
// reset
1_00000000_00000000
// gpio_in val 0b11010 gpio_out val 5.0990195135927845
0_0000001a_00509902
// reset
1_00000000_00000000
// gpio_in val 0b11011 gpio_out val 5.196152422706632
0_0000001b_00519615
// reset
1_00000000_00000000
// gpio_in val 0b11100 gpio_out val 5.291502622129181
0_0000001c_00529150
// reset
1_00000000_00000000
// gpio_in val 0b11101 gpio_out val 5.385164807134504
```

Lab 5 Report

Maheedhar Mandapati
Joanna John

12/7/2019

```
0_0000001d_00538516
// reset
1_00000000_00000000
// gpio_in val 0b11110 gpio_out val 5.477225575051661
0_0000001e_00547723
// reset
1_00000000_00000000
// gpio_in val 0b11111 gpio_out val 5.5677643628300215
0_0000001f_00556776
// reset
1_00000000_00000000
// gpio_in val 0b100000 gpio_out val 5.656854249492381
0_00000020_00565685
// reset
1_00000000_00000000
// gpio_in val 0b100001 gpio_out val 5.744562646538029
0_00000021_00574456
// reset
1_00000000_00000000
// gpio_in val 0b100010 gpio_out val 5.830951894845301
0_00000022_00583095
// reset
1_00000000_00000000
// gpio_in val 0b100011 gpio_out val 5.916079783099616
0_00000023_00591608
// reset
1_00000000_00000000
// gpio_in val 0b100100 gpio_out val 6.0
0_00000024_00600000
// reset
1_00000000_00000000
// gpio_in val 0b100101 gpio_out val 6.082762530298219
0_00000025_00608276
// reset
1_00000000_00000000
// gpio_in val 0b100110 gpio_out val 6.164414002968976
0_00000026_00616441
// reset
1_00000000_00000000
```

Lab 5 Report

Maheedhar Mandapati
Joanna John

12/7/2019

```
// gpio_in val 0b100111 gpio_out val 6.244997998398398
0_00000027_00624500
// reset
1_00000000_00000000
// gpio_in val 0b101000 gpio_out val 6.324555320336759
0_00000028_00632456
// reset
1_00000000_00000000
// gpio_in val 0b101001 gpio_out val 6.4031242374328485
0_00000029_00640312
// reset
1_00000000_00000000
// gpio_in val 0b101010 gpio_out val 6.48074069840786
0_0000002a_00648074
// reset
1_00000000_00000000
// gpio_in val 0b101011 gpio_out val 6.557438524302
0_0000002b_00655744
// reset
1_00000000_00000000
// gpio_in val 0b101100 gpio_out val 6.6332495807108
0_0000002c_00663325
// reset
1_00000000_00000000
// gpio_in val 0b101101 gpio_out val 6.708203932499369
0_0000002d_00670820
// reset
1_00000000_00000000
// gpio_in val 0b101110 gpio_out val 6.782329983125268
0_0000002e_00678233
// reset
1_00000000_00000000
// gpio_in val 0b101111 gpio_out val 6.855654600401044
0_0000002f_00685565
// reset
1_00000000_00000000
// gpio_in val 0b110000 gpio_out val 6.928203230275509
0_00000030_00692820
// reset
```

Lab 5 Report

Maheedhar Mandapati
Joanna John

12/7/2019

```
1_00000000_00000000
// gpio_in val 0b110001 gpio_out val 7.0
0_00000031_00700000
```