Justino Almazan
663755021
ECE 366
Project 1

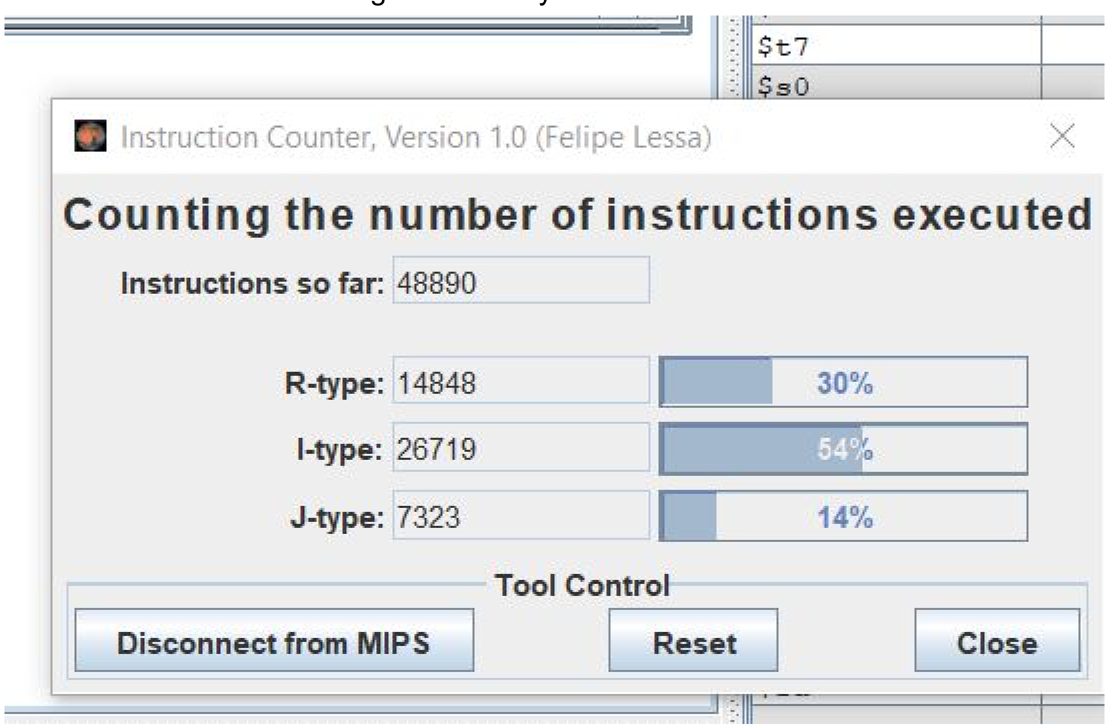
1. My program is able to earn 80%.
2.

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x00002000	0	6	36	216	1296	7776	7	6
0x00002020	36	216	1296	7776	46656	279936	1679616	16
0x00002040	6	36	216	1296	7776	46656	279936	1679616
0x00002060	10077696	60466176	362797056	5	0	0	0	0
0x00002080	0	0	0	0	0	0	0	0
0x000020a0	0	0	0	0	0	0	0	0
0x000020c0	0	0	0	0	0	0	0	0
0x000020e0	0	0	0	0	0	0	0	0
0x00002100	0	0	0	0	0	0	0	0
0x00002120	0	0	0	0	0	0	0	0

0x00002000 (.data) ☑ Hexadecimal Addresses ☐ Hexadecimal Values ☐ ASCII

In this data segment you can see all my values for power.
In segment 0x2014 you see the result of 6^5 = 7776, in segment 0x2018 you see the mod 17 of that result as 7. In segment 2038 you the result of 6^8 = 1679616 and in segment 0x203C you mod 17 of 6^8 is = 16. In segment 2068 you can see result of 6^11 = 362797056 and in segment 206C you'll see mod 17 of that result as 5.



This is the total instruction count for p=5, 8, 11

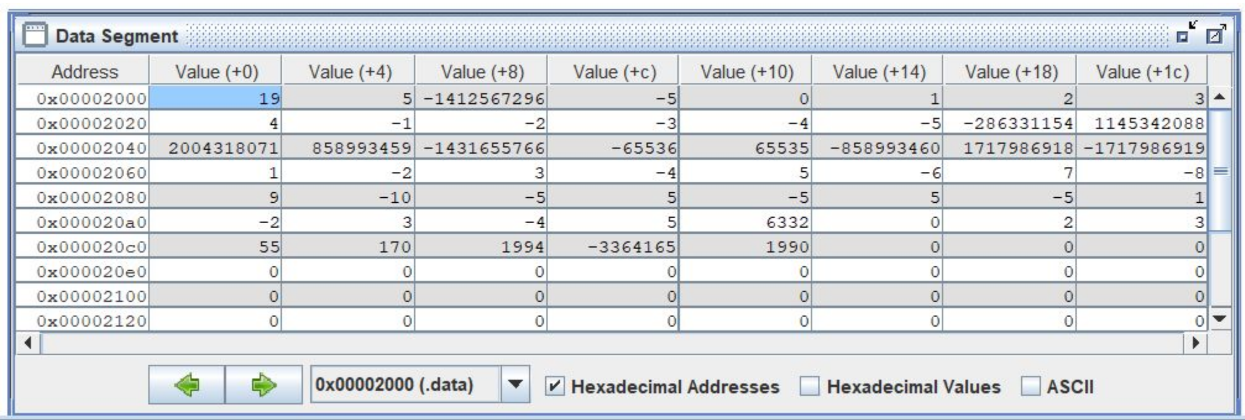
3. A couple weeks ago we took a quiz in class where the number one added itself to 3 and then once looping began, 3 would add again to itself becoming 9 then 27. I noticed this was 3^p. I was inspired by this to solve 6^p in a similar method. For mod 17, I decided to add 1020(a multiple of 17) to itself until finally the number was just one iteration above 6^p. From this point on whatever sum resulted from continuously adding 1020 would be subtracted by 17 many times until this new number was under 6^p. Immediately I subtracted 6^p by this new number and received mod 17 of 6^p.

4.
Addi
Jal
Add
Sw
Slt
Beq
J
Jr
Subi
Sub

$t1, $0, $t6, $t2, $t5, $t0, $t4, $t3, $ra

5. Mine achieves level 2

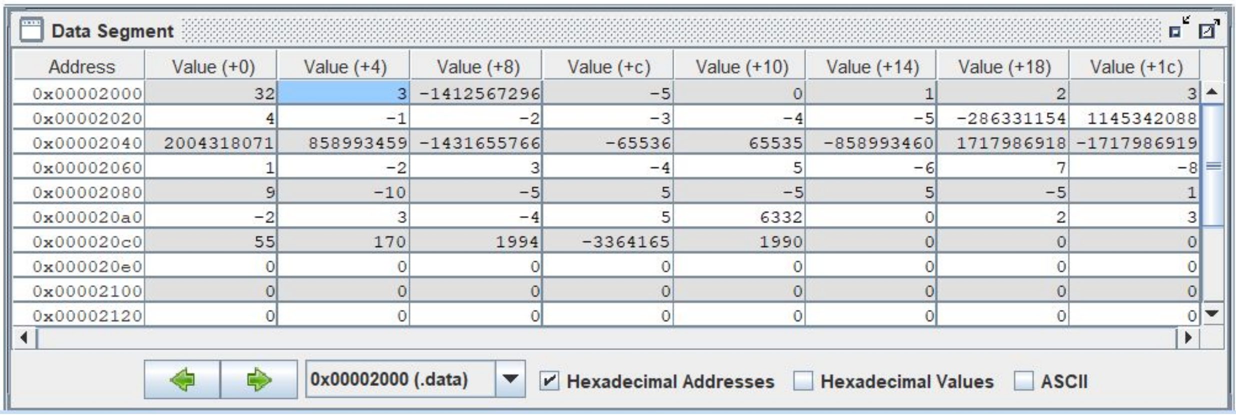
6.



Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x00002000	19	5	-1412567296	-5	0	1	2	3
0x00002020	4	-1	-2	-3	-4	-5	-286331154	1145342088
0x00002040	2004318071	858993459	-1431655766	-65536	65535	-858993460	1717986918	-1717986919
0x00002060	1	-2	3	-4	5	-6	7	-8
0x00002080	9	-10	-5	5	-5	5	-5	1
0x000020a0	-2	3	-4	5	6332	0	2	3
0x000020c0	55	170	1994	-3364165	1990	0	0	0
0x000020e0	0	0	0	0	0	0	0	0
0x00002100	0	0	0	0	0	0	0	0
0x00002120	0	0	0	0	0	0	0	0

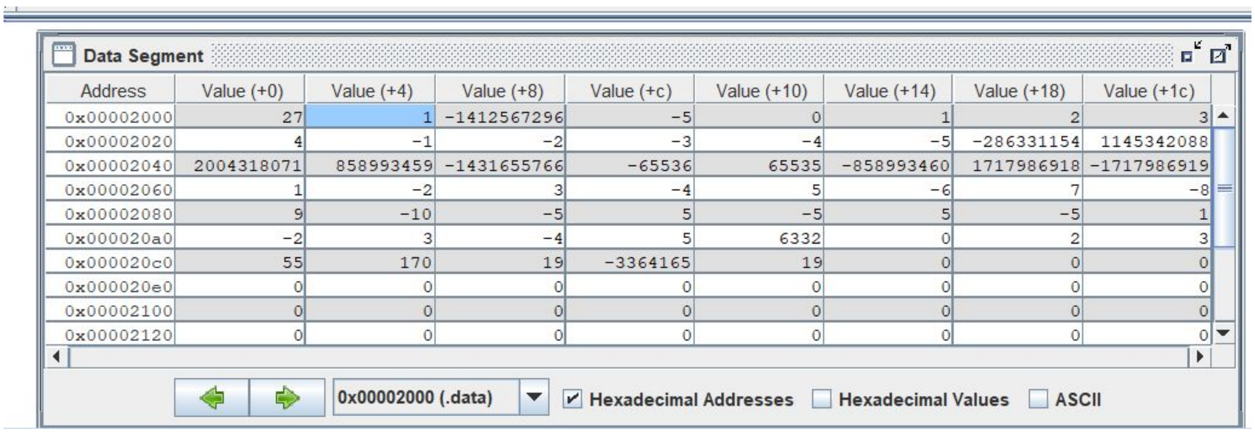
0x00002000 (.data) Hexadecimal Addresses Hexadecimal Values ASCII



Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x00002000	32	3	-1412567296	-5	0	1	2	3
0x00002020	4	-1	-2	-3	-4	-5	-286331154	1145342088
0x00002040	2004318071	858993459	-1431655766	-65536	65535	-858993460	1717986918	-1717986919
0x00002060	1	-2	3	-4	5	-6	7	-8
0x00002080	9	-10	-5	5	-5	5	-5	1
0x000020a0	-2	3	-4	5	6332	0	2	3
0x000020c0	55	170	1994	-3364165	1990	0	0	0
0x000020e0	0	0	0	0	0	0	0	0
0x00002100	0	0	0	0	0	0	0	0
0x00002120	0	0	0	0	0	0	0	0

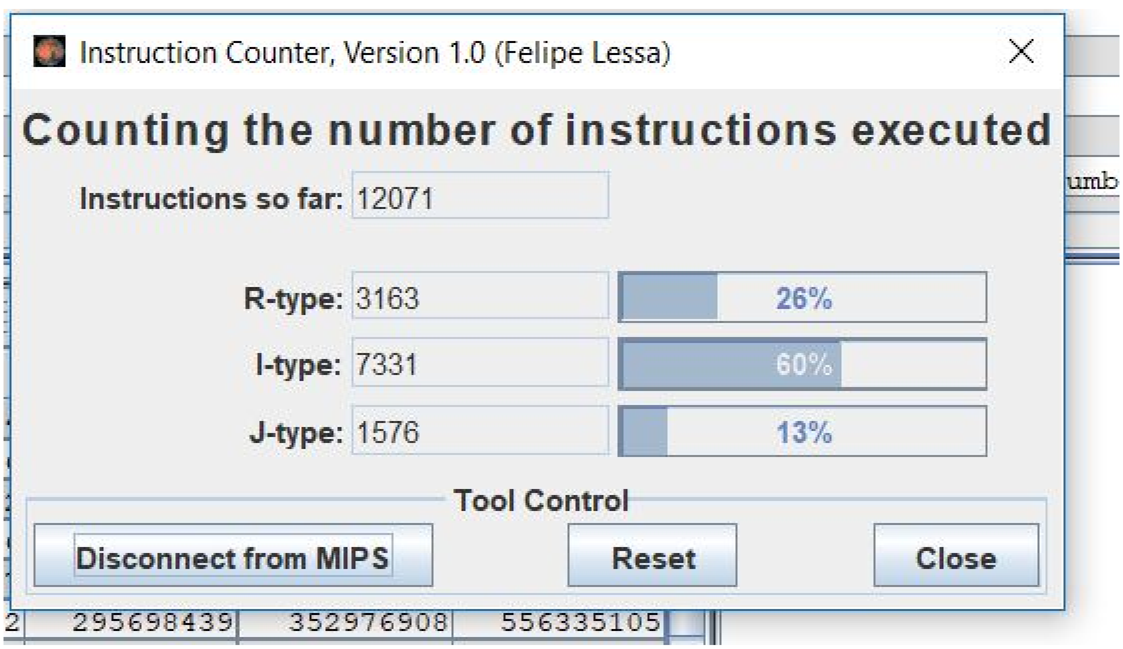
0x00002000 (.data) ☑ Hexadecimal Addresses ☐ Hexadecimal Values ☐ ASCII



Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x00002000	27	1	-1412567296	-5	0	1	2	3
0x00002020	4	-1	-2	-3	-4	-5	-286331154	1145342088
0x00002040	2004318071	858993459	-1431655766	-65536	65535	-858993460	1717986918	-1717986919
0x00002060	1	-2	3	-4	5	-6	7	-8
0x00002080	9	-10	-5	5	-5	5	-5	1
0x000020a0	-2	3	-4	5	6332	0	2	3
0x000020c0	55	170	19	-3364165	19	0	0	0
0x000020e0	0	0	0	0	0	0	0	0
0x00002100	0	0	0	0	0	0	0	0
0x00002120	0	0	0	0	0	0	0	0

0x00002000 (.data) ☑ Hexadecimal Addresses ☐ Hexadecimal Values ☐ ASCII



7. My program will xor two registers and then begin counting the 0's using an AND command as well as a shift right logic feature. When finding the best comparison it adds one to a counter for each 0, also it keeps track of how many matched pairs have this 'best' comparison with another counter.

8.
Add
Addi
J
Sw
Lw
Slt
Beq
And
Bne
Slti
Srl

$t0, $0, $t1, $t7, $s0, $s2, $t6, $t4, $s3, $s4, $ra, $s1, $t3, $t5, $t2

9. Mod 17 took forever, I spent nearly two weekends working on it, I would say I spent close to 70 hours working on this project maybe closer to 100, MIPS is not my strong point.

10. I probably would have attempted 100% because last week I got swamped with homework and an ECE 333 java project. As I'm typing this report minutes before due date, I barely trial and error corrected part 2 of project one

PART 1:

```
.data
my_array: .word -1, -2, -3, -4, -5, -6, -7, -8, -9, -10, -11, -12, -13, -14, -15, -16, -17, 18,
-19, -20, -21, -22, -23, -24  #this is to store the values of 6^p and mod 17 of 6^p

.text
main:
addi $t1, $0, 1020     #multiple to be used for mod 17
addi $t6, $t6, 0 #counter
addi $t2, $0, 1        #small variable to be added repeatedly
addi $t0, $0, 0x2018   #power will be 5, 8 11 and jump to a function to perform a nested
loop
```

```
addi $t5, $0, 0x2000
jal dapowa

addi $t0, $0, 0x201C   #setting up addresses for modulation
addi $t5, $0, 0x2018
add $t2, $0, $0
add $t6, $0, $0
jal mods

addi $t2, $0, 1
addi $t0, $0, 0x203C
addi $t5, $0, 0x2018
jal dapowa

addi $t0, $0, 0x2040
addi $t5, $0, 0x203C
add $t2, $0, $0
jal mods

addi $t2, $0, 1
addi $t0, $0, 0x206C
addi $t5, $0, 0x203C
jal dapowa

addi $t0, $0, 0x2070
addi $t5, $0, 0x206C
add $t2, $0, $0
jal mods

j end

dapowa:  #this applies the power to 6
sw $t4, ($t5)         #save $t4 into $s0
addi $t5, $t5, 4
slt $t3, $t5, $t0     #save address is less than the limiting address for the power
beq $t3, $0, donebruhh
add $t4, $t2, $t2     # 1--->2    6--->12
add $t2, $t4, $t2     # 2--->3    12-->18
add $t4, $t2, $t2     # 3--->6    18-->36    6^3    6^4     6^5.....     6^11
add $t2, $0, $t4      #After trial and error it was easier to save $t4 meanwhile swapping it
with $t2
                      #before looping again
j dapowa
```

```
donebruhh:
jr $ra

mods:    # here we take care of the modulo part of program
slt $t3, $t2, $t4           #$t2 < $t4
beq $t3, $0, sub17     #when $t4 is greater than $t2
add $t2, $t2, $t1         #add 1020 (a num divisible by 17) over and over again
addi $t6, $t6, 1          #if this method takes too long we will 2x1020 then 3x1020...so on
and so forth
beq $t6, 50, mul1020#jump to perform the above comment
j mods                         #repeat loop until this number is just above 6^power
sub17:
slt $t3, $t4, $t2          #6^p is less than the number divisible by 17
beq $t3, $0, done
subi $t2, $t2, 17       #keep subtracting 17 until we are under 6^p
j sub17
done:
sub $t4, $t4, $t2     #subtract 6^p by $t2 which gives us 6^p mod17
sw $t4, ($t5)
jr $ra

mul1020:
addi $t1, $t1, 1020  #double the number divisible by 17
add $t6, $0, $0       #reset counter that tells you when the adding of $t2 is taking too long
j mods


end:
```

PART 2

```
.data
best_matching_score:  .word -1   #best score
best_matching_count:  .word -1   #total number of patterns achieving best score
match:                          .word 0xABCDEF00
match1:                        .word -5
Pattern_Array:             .word 0, 1, 2, 3, 4, -1, -2, -3, -4, -5, 0xEEEEEEEE, 0x44448888,
0x77777777, 0x33333333, 0xAAAAAAAA, 0xFFFF0000, 0xFFFF, 0xCCCCCCCC,
0x66666666, 0x99999999
Pattern_Array1:            .word 1, -2, 3, -4, 5, -6, 7, -8, 9, -10, -5, 5, -5, 5, -5, 1, -2, 3, -4, 5
mymatch:                      .word 0x18BC
myarr:                          .word 0, 2, 3, 55, 0xAA, 19, 0XFFCCAABB, 19
```

```
.text
add $t0, $t0, $0        #counter for zeroes
add $t1, $0, $0         #counter for best matching zeroes
addi $t7, $0, 0x205C
addi $s0, $0, 0x2008            #$s0 will be compared to by $s1
addi $s1, $s0, 8
j main

next:
addi $s0, $0, 0x200C         #$s0 will be compared to by $s1
addi $s1, $0, 0x2060
addi $t7, $0, 0x20AC
sw $0, ($t6)                         #reset values in best score and best total numbers
sw $0, ($t4)
j main

mystuff:
addi $s0, $0, 0x20B0
addi $s1, $0, 0x20B4
addi $t7, $0, 0x20D0
sw  $0, ($t6)
sw  $0, ($t4)
j main

main:
lw $s3, ($s0)              #load match
lw $t3, 0($s1)            #load pattern array
addi $s1, $s1, 4         #move in pattern array
xor $t4, $s3, $t3          #xor the two values together to find exact same bits
j count0                   #begin counting zeroes
cont:
slt $t5, $s1, $t7          #will stop this loop if we reach the end of $s1 aka pattern array
beq $t5, $0, out
j main                          #loop again until we've gone through entire pattern array
out:
beq $s0, 0x200C, mystuff
beq $s0, 0x20B0, end
j next

count0:
addi $t5, $0, 1             #$t5 =1
and $t5, $t4, $t5             #$t5 and with lsb in $t4
bne $t5, $0, loopcount      #if current xored value is not zero loop again
```

```
slti $t3, $s2, 32          #check to see if $t4 has counted all xored values
beq $t3, $0, bigone    #branch to compare
addi $t0, $t0, 1           #add 1 to 0 counter for zeroes
loopcount:
addi $s2, $s2, 1          #count number of bits already scanned
srl $t4, $t4, 1              #shift in xoredvalue
j count0                      #loop


bigone:
add $s2, $0, $0            #reset bit counter for 32 bits
addi $t6, $0, 0x2000     #open memory of best matching score
lw $t2, ($t6)                          #load score $t2 for comparison
beq $t0, $t2, add1                   #if it equals current score we will add one to best
matching count
slt $t5, $t0, $t2                      #count should be less than count from previous count
from register load
beq $t5, $0, reset                   #if not, then reset by saving new register count in best
matching score
add1:
bne $t0, $t2, edit
addi $t1, $t1, 1
addi $s4, $0, 0x2004
sw $t1, ($s4)
add $t0, $0, $0
add $t2, $t2, $0
why:
j cont

reset:
sw $t0, ($t6)
add $t1, $0, $0                    #also reset the best matching score count
addi $t1, $0, 1                     #add one to the best matching score count immediately
sw $t1, ($s4)
add $t0, $0, $0
j cont

edit:
add $t0, $0, $0
j cont
end:
```