**ECE 366 Group 17 Project 3**

Part A) Our ISA is called "Ultimately Specific". It is designed based on the number of different instructions we have. We have a total of 43 different instructions represented by their decimal value from 0-42.The philosophy behind this idea is create the simplest and easiest ISA creatable to our knowledge. The only way to decipher the instruction is to find its decimal value and figure it out from the table. We have a total of 128 possible different instructions (2^7) so reusing instructions within each program is encouraged. This design is based off our previous project ISA that encouraged as little code lines as possible. This would increase efficiency within the code and decrease computation time. The cost of such design is the complexity of our hardware implementation. For now, we will not arrange instruction alphabetically to further increase the simplicity of the machine code and Python. In other words, as the PC instruction value increases, so do the Decimal/Binary values unless branching to a previous instruction. The positive side is that we use only 6 bits for both programs (limited with static Q = 17). ISA is without the parity bit but our machine code will include parity bit.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Dec. Val | | Binary/Machine code | | Instr. | | Description | |
| 0 | | 0000000 | | lw $7, P | | # $7 = P | |
| 1 | | 0000001 | | addi $9, $0, 1 | | # $9 = 1 | |
| 2 | | 0000010 | | add $10, $0, $0 | | # $10 = 0 = counter N = exponent | |
| 3 | | 0000011 | | andi $10, $7, 1 | | # $10 = $7 and 1 (0/1) | |
| 4 | | 0000100 | | srl $7, $7, 1 | | # $7 >> 1 | |
| 5 | | 0000101 | | beq $10, $0, Mult2 | | # if zero, go to Mult2 | |
| 6 | | 0000110 | | add $10, $9, $9 | | # temp $10 = $9 + $9 = $ 10 = 2 | |
| 7 | | 0000111 | | add $9, $9, $9 | | # $9 = $9 + $9 = $9 = 2 | |
| 8 | | 0001000 | | add $9, $10, $9 | | # $9 = $10 + $9 = $9 = 6 | |
| 9 | | 0001001 | | beq $10, $0, Mult4 | | # if zero, go to Mult2, check to multiply next bit by 4, else multiply by 2 | |
| 10 | | 0001010 | | beq $10, $0, Mult16 | | # if zero, go to Mult2, check to multiply next bit by 16, else Mod | |
| 11 | | 0001011 | | beq $10, $0, Mod | | # if zero, go to Mult2, check to multiply next bit by 4, else multiply by 2 | |
| 12 | | 0001100 | | slti $10, $9, 17 | | # if $9 < $8, $10 = 1, else $10 = 0, these next 3 lines will subtract by 17 until the is only the remainder | |
| 13 | | 0001101 | | bne $10, $0, End | | # if($10 =1) Save2, else cont | |
| 14 | | 0001110 | | subi $9, $9, 17 | | # $9 = $9 -17 = 36 -17 = 19 | |
| 15 | | 0001111 | | j Mod | | # Will loop back to Mod to get remainder | |
| 16 | | 0010000 | | sw $9, R | | #Stores the remainder value into R | |
| 17 | | 0010001 | | addi $5 $0, 0 | | # Counter i | |
| 18 | | 0010010 | | addi $7 $0, 0 | | # Counter J | |
| 19 | | 0010011 | | addi $6 $0, 400 | | # stop value of array | |
| 20 | | 0010100 | | beq $5, $6, Exit | | # if $5 = 400, Exit program | |
| 21 | | 0010101 | | lw $1, 0xC($0) | | # $1 = T = Value to compare to / first Anded value | |
| 22 | | 0010110 | | lw $2, 0x2020($5) | | # $2 = mem(0x2020 + $5) | |
| 23 | | 0010111 | | addi $5, $5, 4 | | # Increment $5 for next array index | |
| 24 | | 0011000 | | and $1, $1, $2 | | # $1 = $1 and $2 = T and Array(i) = Anded value/rid of T to save on registers | |
| 25 | | 0011001 | | addi $8 $0, 16 | | # stop value for bit # | |
| 26 | | 0011010 | | beq $7, $8, Save | | # Once $7 equals 16, go to next array index | |
| 27 | | 0011011 | | addi $7, $7, 1 | | # Increment $7 by 1 | |
| 28 | | 0011100 | | andi $2, $1, 1 | | # $2 = $1 and 1 = Anded value and 1, $2 is temp value | |
| 29 | | 0011101 | | add $3, $3, $2 | | # counts # of 1s in $1 (# of matching bits) | |
| 30 | | 0011110 | | srl $1, $1, 1 | | # $1 = $1 >> 1, reduces it by 1 bit till = 0 | |
| 31 | | 0011111 | | j Compare | | # now Compare with old values | |
| 32 | | 0100000 | | lw $1, 0x2010($0) | | # reuse $1 = S | |
| 33 | | 0100001 | | lw $2, 0x2014($0) | | # reuse $2 = C | |
| 34 | | 0100010 | | slt $6, $1, $3 | | # $6 = 1 if( $1(S) < $3), else $6 = 0 | |
| 35 | | 0100011 | | beq $6, $0, Check1 | | # if (s>= $3 then $6 = 0) => if ($6 = 0) go to check 1, else continue | |
| 36 | | 0100100 | | add $1, $0, $3 | | # $1 = $3, $1 will now take highest value | |
| 37 | | 0100101 | | addi $2, $0, 1 | | # $2(T) will be set to one since its first highest value | |
| 38 | | 0100110 | | sw $1, 0x2010($0) | | # mem(0x2010) = $1 = S | |
| 39 | | 0100111 | | sw $2, 0x2014($0) | | # mem(0x2014) = $2 = T | |
| 40 | | 0101000 | | j Start | | # start over/chech next array | |
| 41 | | 0101001 | | beq $1, $3, Check2 | | # if ($1 = $3) go to check 2, else continue | |
| 42 | | 0101010 | | addi $2, $2, 1 | | # Increment 2(T) by 1 since matching count was found | |

Part B) 1. The most significant advantage of our ISA is that it is extremely simplistic and easy to program new set of instructions the individual wants to add. The biggest limitation in our ISA is that a certain person cannot add more than 127 set of instructions as this is the limit. The main compromises we have done are reducing the amount of registers used to cut back on the number of instructions needed.

2. We have significantly shortened the amount of computation time need to run however we increased the complexity of the hardware as each instruction had its own machine code. A different approach we of taken is to simplify the hardware and assigning a set of instructions with set registers for each line of machine code. However I would assume that this would increase the computation time of the program.

3. A) I learned a whole new way of creating an extremely simple ISA that is efficient to its certain limitations yet very useful. The best part was being able to come up with less and less instructions as we continued to simplify the instructions needed to run program one and two. The worst part was trying to figure out how to create the simulation in python as we needed to conduct some trial and error to receive the correct results.

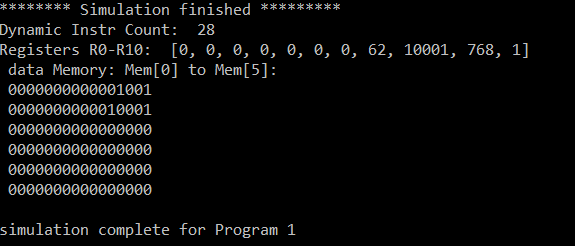
B) Some advice for a student taking this class in a future semester is to continuously work on the project day to day and set goals for oneself as to not neglect the deadline day and keeping up with the work of the project.

C) The value of this project is immense it allows us to communicate and work with individuals from different character and knowledge backgrounds, each bringing their own idea to the table. This allows for one to learn from each other as well as showing what it takes to successfully work as a team for a multilayered project.

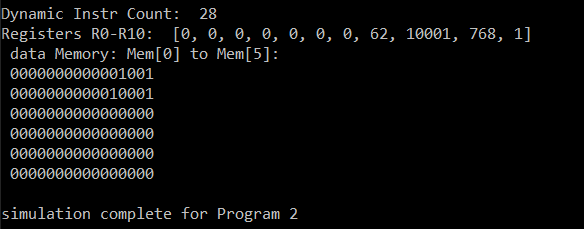
Part C)

1.Pattern Version A

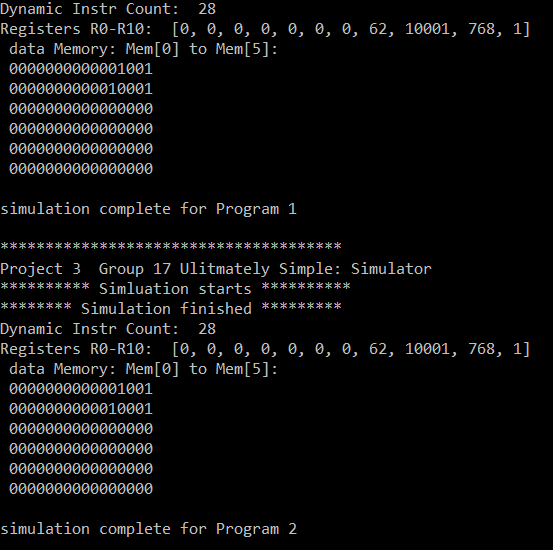
P1:



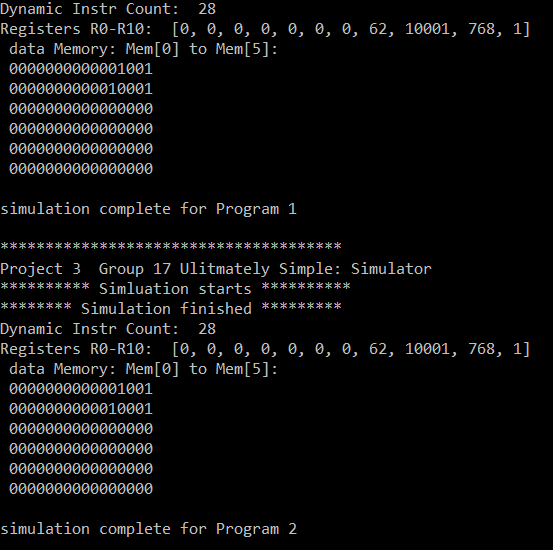
P2:



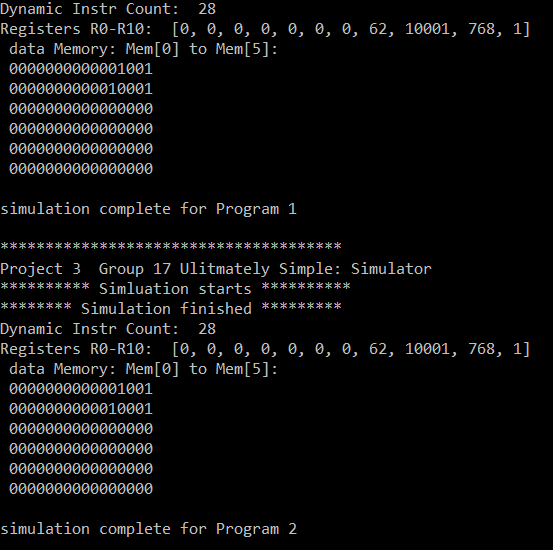
Pattern Version B



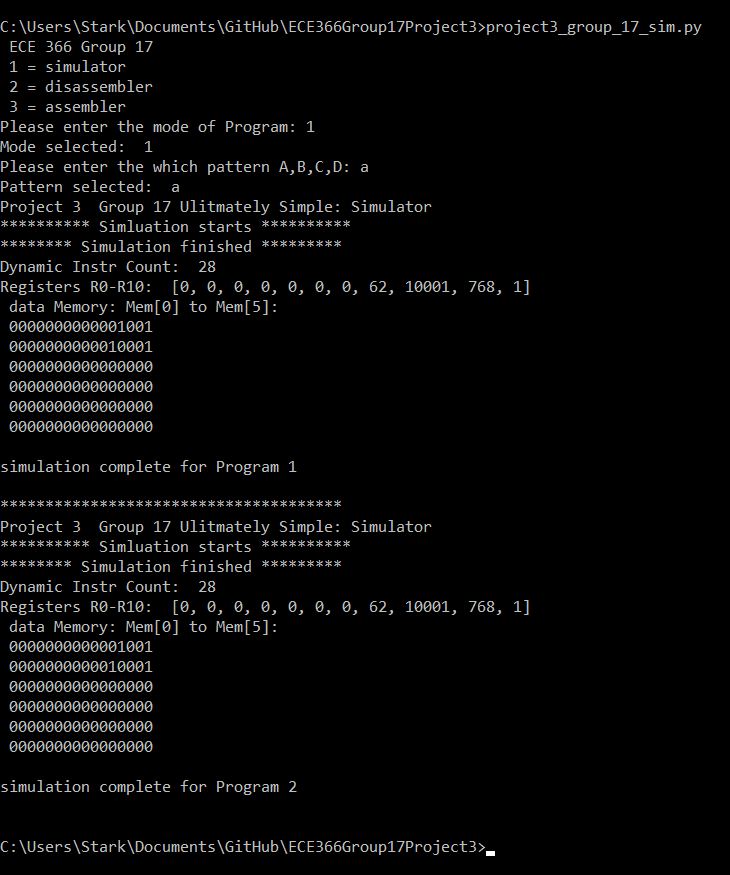
Pattern version C



Pattern version D

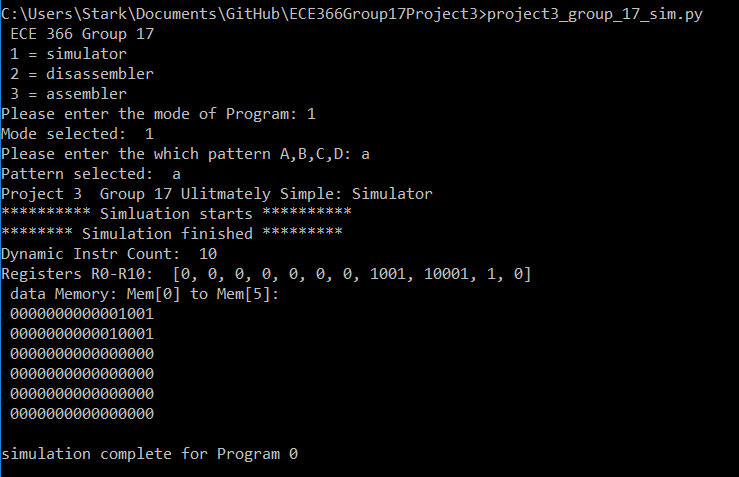


2.



3.

P0 Pattern Version A



Part D)

1. Algorithms

Assembly for Program 1

.data

P: .word 15

Q: .word 17

R: .word -1 #r will be stored here

.text

lw $7, P

lw $8, Q

addi $9, $0, 1 # $9 = 1

add $10, $0, $0 # $10 = 0 = counter N = exponent

Base6: andi $10, $7, 1 # $10 = $7 and 1 (0/1)

srl $7, $7, 1 # $7 >> 1

beq $10, $0, Mult2 # if zero, go to Mult2

add $10, $9, $9 # temp $10 = $9 + $9 = $ 10 = 2

add $9, $9, $9 # $9 = $9 + $9 = $9 = 2

add $9, $9, $9 # $9 = $9 + $9 = $9 = 4

add $9, $10, $9 # $9 = $10 + $9 = $9 = 6

Mult2: andi $10, $7, 1 # $10 = $7 and 1 (0/1)

srl $7, $7, 1 # $7 >> 1

beq $10, $0, Mult4 # if zero, go to Mult4

add $9, $9, $9 # $9 = $9 + $9 = 2x$9 10

Mult4: andi $10, $7, 1 # $10 = $7 and 1 (0/1)

srl $7, $7, 1 # $7 >> 1

beq $10, $0, Mult16 # if zero, go to Mult16

add $9, $9, $9 # $9 = $9 + $9 = 2x$9

add $9, $9, $9 # $9 = $9 + $9 = 4x$9

Mult16: andi $10, $7, 1 # $10 = $7 and 1 (0/1)

srl $7, $7, 1 # $7 >> 1

beq $10, $0, Mod # if zero, go Mod

add $9, $9, $9 # $9 = $9 + $9

add $9, $9, $9 # $9 = $9 + $9

add $9, $9, $9 # $9 = $9 + $9

add $9, $9, $9 # $9 = $9 + $9

Mod: slt $10, $9, $8 # if $9 < $8, $10 = 1, else $10 = 0

bne $10, $0, End # if($10 =1) Save2, else cont

sub $9, $9, $8 # $16 = $16 -17 = 36 -17 = 19

j Mod

End: sw $9, R

Assembly for Program 2

addi $5 $0, 0 # Counter for Array

addi $7 $0, 0 # Counter for bit #

Start: addi $6 $0, 400 # stop value of array

beq $5, $6, Exit # if $5 = 400, Exit program

lw $1, 0xC($0) # $1 = T = Value to compare to / first Anded value

lw $2, 0x2020($5) # $2 = mem(0x2020 + $5)

addi $5, $5, 4 # Increment $5 for next array index

and $1, $1, $2 # $1 = $1 and $2 = T and Array(i) = Anded value , gets rid

of T during this loop to save on registers

Compare: addi $8 $0, 16 # stop value for bit #

beq $7, $8, Save # Once $7 equals 16, go to next array index

addi $7, $7, 1 # Increment $7 by 1

andi $2, $1, 1 # $2 = $1 and 1 = Anded value and 1, $2 is temp value

add $3, $3, $2 # counts # of 1s in $1

srl $1, $1, 1

j Compare

Save: lw $1, 0x2010($0)

lw $2, 0x2014($0)

slt $6, $1, $3

beq $6, $0, Check1

add $1, $0, $3

addi $2, $0, 1

sw $1, 0x2010($0)

sw $2, 0x2014($0)

j Start

Check1: beq $1, $3, Check2

j Start

Check2: addi $2, $2, 1

sw $2, 0x2014

j Start

Exit:

2) Machine Code for Program 1

00000000

00000011

00000101

00000110

00001001

00001010

00001100

00001111

00010001

00010010

00010100

00010111

00011000

00011011

00011101

00011110

00100001

Machine Code for Program 2

00100010

00100100

00100111

00101000

00101011

00101101

00101110

00110000

00110011

00110101

00110110

00111001

00111010

00111100

00111111

01000001

01000010

01000100

01000111

01001000

01001011

01001101

01001110

01010000

01010011

01010101

Machine P0

00000000

00000011

00000011

00000011

00000011

00000011

00000011

00000011

00000011

00000011

00000011

00000011

00000011

00000011

00000011

00000011

00000011

00000011

00000011

00000011

00000011

00000011

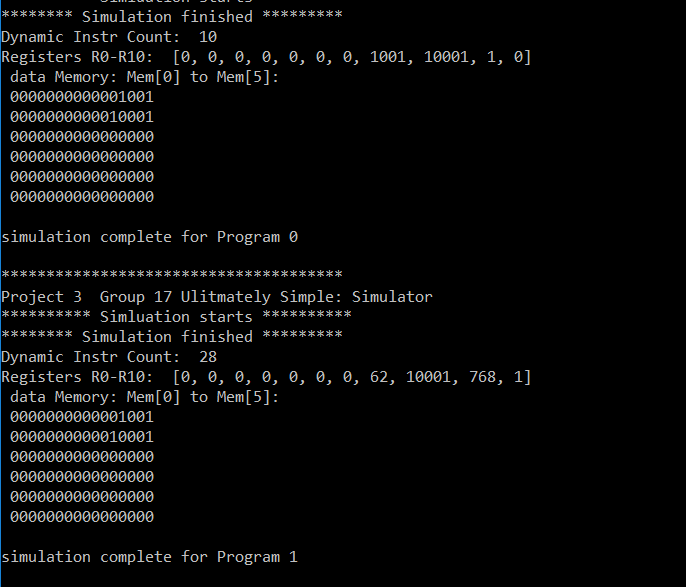
00000011

00000011

00000011

00000101

00011011

3) Data mem for Pattern C and D 

4) Python Code

|  |
| --- |
|  |

def disassembler(I, lines, mode):

print("ECE 366 Group 17 Disassembler")

print("----------------")

if(mode == 1):

output\_file = open("project3\_group\_17\_p1\_asm.txt", "w")

elif(mode == 2):

output\_file = open("project3\_group\_17\_p2\_asm.txt", "w")

else:

print("Invalid Mode")

# output\_file.write("This is a test before loop")

print( lines)

#write to output file

for i in range(lines):

line =I[i]

#output\_file.write("This is a Test in loop")

if(line[0:8] == "00000000"):

output\_file.write("lw $7, P #$7 = P \n")

elif(line[0:8] == "00000011"):

output\_file.write("add $9, $0, 1 #$9 = 1\n")

elif(line[0:8] == "00000101"):

output\_file.write("addi $10, $0, $0 # $10 =0 = counter N =exponent\n")

elif(line[0:8] == "00000110"):

output\_file.write("andi $10, $7, 1 # $10-$7 and 1(0/1) \n")

elif(line[0:8] == "00001001"):

output\_file.write("srl $7, $7, 1 # $7 >>1 \n")

elif(line[0:8] == "00001010"):

output\_file.write("beq $10, $0, Mult2 # if zero go to Mult2\n")

elif(line[0:8] == "00001100"):

output\_file.write("add $10, $9, $9 # temp $10 = $9 + $9 + $10 =2\n")

elif(line[0:8] == "00001111"):

output\_file.write("add $9, $9, $9 # $9 = $9 + $9 =$9 =2 $0\n")

elif(line[0:8] == "00010001"):

output\_file.write("add $9, $10, $9 # $9 = $10 +$9 = $9 = 6\n")

elif(line[0:8] == "00010010"):

output\_file.write("beq $10, $0, Mult4 # $if zero, go Mult2, check to multiply next bit by 4, else multiply by 2\n")

elif(line[0:8] == "00010100"):

output\_file.write("beq $10, $0, Mult16 # $if zero, go Mult2, check to multiply next bit by 16, else MOD\n")

elif(line[0:8] == "00010111"):

output\_file.write("beq $10, $0, Mod # $if zero, go Mult2, check yo multiply next bit by 4, else multiply by 2\n")

elif(line[0:8] == "00011000"):

output\_file.write("slti $10, $9, 17 # if $9< $8, $10 = 1, else $10 = 0, these next 3 lines will subract by 17 until the is only the remainder\n")

elif(line[0:8] == "00011011"):

output\_file.write("bne $10, $0, End # if($10 = 1) Save2, else cont\n")

elif(line[0:8] == "00011101"):

output\_file.write("subi $9, $9, 17 # $9 = $9 -17 = 36 -17 = 19\n")

elif(line[0:8] == "00011110"):

output\_file.write("j Mod # Will loop back to Mod to get remainder \n")

elif(line[0:8] == "00100001"):

output\_file.write("sw $9, R #Stores the remainder value into R\n")

elif(line[0:8] == "00100010"):

output\_file.write("addi $5 $0, 0 # Counter i \n")

elif(line[0:8] == "00100100"):

output\_file.write("addi $7 $0, 0 # Counter J\n")

elif(line[0:8] == "00100111"):

output\_file.write("addi $6 $0, 400 # stop value of array\n")

elif(line[0:8] == "00101000"):

output\_file.write("beq $5, $6, Exit # if $5 = 400, Exit program\n")

elif(line[0:8] == "00101011"):

output\_file.write("lw $1, 0xC($0) # $1 = T = Value to compare to / first Anded value\n")

elif(line[0:8] == "00101101"):

output\_file.write("lw $2, 0x2020($5) # $2 = mem(0x2020 + $5)\n")

elif(line[0:8] == "00101110"):

output\_file.write("addi $5, $5, 4 # Increment $5 for next array index\n")

elif(line[0:8] == "00110000"):

output\_file.write("and $1, $1, $2 # $1 = $1 and $2 = T and Array(i) = Anded value/rid of T to save on registers\n")

elif(line[0:8] == "00110011"):

output\_file.write("addi $8 $0, 16 # stop value for bit \n")

elif(line[0:8] == "00110101"):

output\_file.write("beq $7, $8, Save # Once $7 equals 16, go to next array index\n")

elif(line[0:8] == "00110110"):

output\_file.write("addi $7, $7, 1 # Increment $7 by 1\n")

elif(line[0:8] == "00111001"):

output\_file.write("andi $2, $1, 1 # $2 = $1 and 1 = Anded value and 1, $2 is temp value\n")

elif(line[0:8] == "00111010"):

output\_file.write("add $3, $3, $2 # counts # of 1s in $1 (# of matching bits)\n")

elif(line[0:8] == "00111100"):

output\_file.write("srl $1, $1, 1 # $1 = $1 >> 1, reduces it by 1 bit till = 0\n")

elif(line[0:8] == "00111111"):

output\_file.write("j Compare # now Compare with old values\n")

elif(line[0:8] == "01000001"):

output\_file.write("lw $1, 0x2010($0) # reuse $1 = S\n")

elif(line[0:8] == "01000010"):

output\_file.write("lw $2, 0x2014($0) # reuse $2 = C\n")

elif(line[0:8] == "01000100"):

output\_file.write("slt $6, $1, $3 # $6 = 1 if( $1(S) < $3), else $6 = 0\n")

elif(line[0:8] == "01000111"):

output\_file.write("beq $6, $0, Check1 # if (s>= $3 then $6 = 0) => if ($6 = 0) go to check 1, else continue\n")

elif(line[0:8] == "01001000"):

output\_file.write("add $1, $0, $3 # $1 = $3, $1 will now take highest value\n")

elif(line[0:8] == "01001011"):

output\_file.write("addi $2, $0, 1 # $2(T) will be set to one since it's first highest value\n")

elif(line[0:8] == "01001101"):

output\_file.write("sw $1, 0x2010($0) # mem(0x2010) = $1 = S\n")

elif(line[0:8] == "01001110"):

output\_file.write("sw $2, 0x2014($0) # mem(0x2014) = $2 = T\n")

elif(line[0:8] == "01010000"):

output\_file.write("j Start # start over/chech next array\n")

elif(line[0:8] == "01010011"):

output\_file.write("beq $1, $3, Check2 # if ($1 = $3) go to check 2, else continue\n")

elif(line[0:8] == "01010101"):

output\_file.write("addi $2, $2, 1 # Increment 2(T) by 1 since matching count was found\n")

elif(line[0:8] == "01010110"):

output\_file.write("slt $10, $9, $8 # if $9 < $8, $10 = 1, else $10 = 0\n")

elif(line[0:8] == "01011001"):

output\_file.write("sub $9, $9, $8 # $16 = $16 -17 = 36 -17 = 19\n")

else:

output\_file.write("Instructions not supported\n")

def simulate(I,Memory,Nlines,program):

print("Project 3 Group 17 Ulitmately Simple: Simulator")

PC = 0 #program counter

DIC= 0 #dynamic instruction counter

Reg = [0,0,0,0,0,0,0,0,0,0,0] # intializing all ten registers to zero, reg[0] is $0

Mem = 8; # Where the memory for the arrays starts

# Memory = [0 for i in range(10)] # data memory,

print("\*\*\*\*\*\*\*\*\*\* Simluation starts \*\*\*\*\*\*\*\*\*\*")

# finished = False

#while( not(finished)):

# fetch = Instructions[PC]

# DIC += 1

# print(fetch)

if (program == 1):

max = 31 #When it will stop running the program for P1

else:

max = 32 # When it will stop running the program for P2

while(PC < max): # Based on the program length its the max pc can run

line=I[PC]

if(line[0:8] == "00000000"):

#output\_file.write("lw $7, P #$7 = P \n")

Reg[7] = int(Memory[0])

Reg[8] = int(Memory[1]) # loads Q even thou its fixed at 17

PC += 1

elif(line[0:8] == "00000011"):

#output\_file.write("add $9, $0, 1 #$9 = 1\n")

Reg[9] = Reg[0] + 1

PC += 1

elif(line[0:8] == "00000101"):

#output\_file.write("addi $10, $0, $0 # $10 =0 = counter N =exponent\n")

Reg[10] = 0

PC += 1

elif(line[0:8] == "00000110"):

#output\_file.write("andi $10, $7, 1 # $10-$7 and 1(0/1) \n")

Reg[10] = Reg[7] + 1

PC += 1

elif(line[0:8] == "00001001"):

#output\_file.write("srl $7, $7, 1 # $7 >>1 \n")

Reg[7] = Reg[7] >> 1

PC +=1

elif(line[0:8] == "00001010"):

#output\_file.write("beq $10, $0, Mult2 # if zero go to Mult2")

if(Reg[10] == Reg[0]):

PC = 12 #PC 11 should be Mult2

else:

PC += 1

elif(line[0:8] == "00001100"):

#output\_file.write("add $10, $9, $9 # temp $10 = $9 + $9 + $10 =2\n")

Reg[10] = Reg[9] +Reg[9]

PC += 1

elif(line[0:8] == "00001111"):

#output\_file.write("add $9, $9, $9 # $9 = $9 + $9 =$9 =2 $0\n")

Reg[9] = Reg[9]+ Reg[9]

PC += 1

elif(line[0:8] == "00010001"):

#output\_file.write("add $9, $10, $9 # $9 = $10 +$9 = $9 = 6\n")

Reg[9] = Reg[10]+ Reg[9]

PC += 1

elif(line[0:8] == "00010010"):

#output\_file.write("beq $10, $0, Mult4 # $if zero, go Mult2, check to multiply next bit by 4, else multiply by 2\n")

if(Reg[10] == Reg[0]):

PC = 16 # Mult4 should be PC 15

else:

PC += 1

elif(line[0:8] == "00010100"):

#output\_file.write("beq $10, $0, Mult16 # $if zero, go Mult2, check to multiply next bit by 16, else MOD")

if(Reg[10] == Reg[0]):

PC = 21 #PC Mult16

else:

PC += 1

elif(line[0:8] == "00010111"):

#output\_file.write("beq $10, $0, Mod # $if zero, go Mult2, check to multiply next bit by 4, else multiply by 2\n")

if(Reg[10] == Reg[0]):

PC = 28 #PC for MOD

else:

PC += 1

elif(line[0:8] == "00011000"):

#output\_file.write("slti $10, $9, 17 # if $9< $8, $10 = 1, else $10 = 0, these next 3 lines will subract by 17 until the is only the remainder\n")

if( Reg[9] < Reg[8]):

Reg[10] = 1

else:

Reg[10]= 0

PC += 1

elif(line[0:8] == "00011011"): #PC 28

#output\_file.write("bne $10, $0, End # if($10 = 1) Save2, else cont\n")

if( Reg[10] == 1): # or 10 != 0

#PC = What ever "Save2"s

PC = 32 # should excute instruction with PC 31 then end

else:

PC += 1

elif(line[0:8] == "00011101"):

#output\_file.write("subi $9, $9, 17 # $9 = $9 -17 = 36 -17 = 19\n")

Reg[9] = Reg[9] - 17

PC += 1

elif(line[0:8] == "00011110"):

#output\_file.write("j Mod # Will loop back to Mod to get remainder \n")

PC= 28 # PC for MOD

elif(line[0:8] == "00100001"):

#output\_file.write("sw $9, R #Stores the remainder value into R\n")

Memory[2] = str(Reg[9]) #Memory[2] is R (result) !!Memory is a string not a int

PC += 1

elif(line[0:8] == "00100010"):

#output\_file.write("addi $5 $0, 0 # Counter i \n")

Reg[5] = Reg[0] + 0

PC += 1

elif(line[0:8] == "00100100"):

#output\_file.write("addi $7 $0, 0 # Counter J\n")

Reg[7] = Reg[0]+ 0

PC += 1

elif(line[0:8] == "00100111"):

#output\_file.write("addi $6 $0, 400 # stop value of array\n")

Reg[6] = Reg[0]+ 400

PC += 1

elif(line[0:8] == "00101000"):

#output\_file.write("beq $5, $6, Exit # if $5 = 400, Exit program\n")

if (Reg[5] == 400):

PC = 31

else:

PC += 1

elif(line[0:8] == "00101011"):

#output\_file.write("lw $1, 0xC($0) # $1 = T = Value to compare to / first Anded value\n")

Reg[1] = int(Memory[3])

PC += 1

elif(line[0:8] == "00101101"):

#output\_file.write("lw $2, 0x2020($5) # $2 = mem(0x2020 + $5)\n")

Reg[2] = int(Memory[mem +Reg[5]])

PC +=1

elif(line[0:8] == "00101110"):

#output\_file.write("addi $5, $5, 4 # Increment $5 for next array index\n")

Reg[5] = Reg[5]+ 1 #array index is only by one using python

PC += 1

elif(line[0:8] == "00110000"):

#output\_file.write("and $1, $1, $2 # $1 = $1 and $2 = T and Array(i) = Anded value/rid of T to save on registers\n")

Reg[1] = Reg[1]+ Reg[2]

PC += 1

elif(line[0:8] == "00110011"):

#output\_file.write("addi $8 $0, 16 # stop value for bit \n")

Reg[8] = Reg[0] + 16

PC += 1

elif(line[0:8] == "00110101"):

#output\_file.write("beq $7, $8, Save # Once $7 equals 16, go to next array index\n")

if( Reg[7] == Reg[8]):

#PC = what ever "Save" is

PC= 17

else:

PC += 1

elif(line[0:8] == "00110110"):

#output\_file.write("addi $7, $7, 1 # Increment $7 by 1\n")

Reg[7] = Reg[7]+ 1

PC += 1

elif(line[0:8] == "00111001"):

#output\_file.write("andi $2, $1, 1 # $2 = $1 and 1 = Anded value and 1, $2 is temp value\n")

Reg[2] = Reg[1]+ 1

PC += 1

elif(line[0:8] == "00111010"):

#output\_file.write("add $3, $3, $2 # counts # of 1s in $1 (# of matching bits)\n")

Reg[3] = Reg[3]+ Reg[2]

PC += 1

elif(line[0:8] == "00111100"):

#output\_file.write("srl $1, $1, 1 # $1 = $1 >> 1, reduces it by 1 bit till = 0\n")

Reg[1] = Reg[1] >> 1

PC +=1

elif(line[0:8] == "00111111"):

#output\_file.write("j Compare # now Compare with old values\n")

PC = 9

elif(line[0:8] == "01000001"):

#output\_file.write("lw $1, 0x2010($0) # reuse $1 = S\n")

Reg[1] = int(Memory[4])

PC += 1

elif(line[0:8] == "01000010"):

#output\_file.write("lw $2, 0x2014($0) # reuse $2 = C\n")

Reg[2] = int(Memory[5])

PC += 1

elif(line[0:8] == "01000100"):

#output\_file.write("slt $6, $1, $3 # $6 = 1 if( $1(S) < $3), else $6 = 0\n")

if (Reg[1] < Reg[3]):

Reg[6] = 1

else:

Reg[6] = 0

PC +=1

elif(line[0:8] == "01000111"):

#output\_file.write("beq $6, $0, Check1 # if (s>= $3 then $6 = 0) => if ($6 = 0) go to check 1, else continue\n")

if( Reg[1] >= Reg[3]):

Reg[6] = 0

PC +=1

else:

PC = 28

elif(line[0:8] == "01001000"):

#output\_file.write("add $1, $0, $3 # $1 = $3, $1 will now take highest value\n")

Reg[1] = Reg[0]+ Reg[3]

PC += 1

elif(line[0:8] == "01001011"):

#output\_file.write("addi $2, $0, 1 # $2(T) will be set to one since it's first highest value\n")

Reg[2] = Reg[0]+ 1

PC += 1

elif(line[0:8] == "01001101"):

#output\_file.write("sw $1, 0x2010($0) # mem(0x2010) = $1 = S\n")

Memory[4] = str(Reg[1])

PC += 1

elif(line[0:8] == "01001110"):

#output\_file.write("sw $2, 0x2014($0) # mem(0x2014) = $2 = T\n")

Memory[5] = str(Reg[2])

PC += 1

elif(line[0:8] == "01010000"):

#output\_file.write("j Start # start over/chech next array\n")

PC = 3

elif(line[0:8] == "01010011"):

#output\_file.write("beq $1, $3, Check2 # if ($1 = $3) go to check 2, else continue\n")

if (Reg[1] ==Reg[3]):

PC = 28

else:

PC += 1

elif(line[0:8] == "01010101"):

#output\_file.write("addi $2, $2, 1 # Increment 2(T) by 1 since matching count was found\n")

Reg[2] = Reg[2]+ 1

PC += 1

elif(line[0:8] == "01010110"):

#output\_file.write("slt $10, $9, $8 \n")

if (Reg[9] < Reg[8]):

Reg[10] = 1

else:

Reg[10] = 0

PC += 1

elif(line[0:8] == "01011001"): #PC 29

#output\_file.write("sub $9, $9, $8 \n")

Reg[9] = Reg[9] - Reg[8]

PC += 1

else:

#output\_file.write("Instructions not supported/n")

print("instruction not supported")

DIC+=1

print("PC value: ", PC)

#print("Instruction[", PC, "]: ", I[PC-1])

print("\*\*\*\*\*\*\*\* Simulation finished \*\*\*\*\*\*\*\*\*")

print("Dynamic Instr Count: ",DIC)

print("Registers R0-R10: ",Reg)

def main():

data\_file = open("project3\_group\_17\_p1\_bin.txt" ,"r")

data\_file2 = open("project3\_group\_17\_p2\_bin.txt", "r")

data\_fileA = open("patternA.txt", "r")

data\_fileB = open("patternB.txt", "r")

data\_fileC = open("patternC.txt", "r")

data\_fileD = open("patternD.txt", "r")

#we need a file for the data set

#Nsteps = 3 #How many cycles to run before output

Nlines = 0 #How may instrs total in input.txt for P1

Mlines = 0 # How many instrs total in input.txt for P2

MemLines = 0

Instructions = [] #all instructions will be stored here for P1

Instructions2 = [] #all instructions will be stored here for P2

Memory = [] # where the data is being stored

print( " ECE 366 Group 8")

print( " 1 = simulator")

print( " 2 = disassembler")

print( " 3 = assembler")

mode= int(input( "Please enter the mode of Program: "))

print( "Mode selected: ", mode)

#modedis= int(input( "Please enter the which program 1 or 2: "))

#print( "Mode selected: ", end=" ")

pattern = input( "Please enter the which pattern A,B,C,D: ")

print( "Pattern selected: ", pattern)

#if (modedis == 1):

#P = int(input( "Program 1 was selected, What is the value of P: ")

#print( "P selected: ", end=" ")

if (pattern == 'A'or'a'):

for line in data\_fileA:

if(line == "\n" or line[0] =='#'):

continue

Memory.append(line)

MemLines+=1

elif(pattern == 'B'or'b'):

for line in data\_fileB:

if(line == "\n" or line[0] =='#'):

continue

Memory.append(line)

MemLines+=1

if (Memory[1] != '0000000000010001'):

Memory[1] = '0000000000010001'

print(" Program 1 has fixed Q = 17 ")

elif(pattern == 'C'or'c'):

for line in data\_fileC:

if(line == "\n" or line[0] =='#'):

continue

Memory.append(line)

MemLines+=1

if (Memory[1] != '0000000000010001'):

Memory[1] = '0000000000010001'

print(" Program 1 has fixed Q = 17 ")

elif(pattern == 'D'or'd'):

for line in data\_fileD:

if(line == "\n" or line[0] =='#'):

continue

Memory.append(line)

MemLines+=1

if (Memory[1] != '0000000000010001'):

Memory[1] = '0000000000010001'

print(" Program 1 has fixed Q = 17 ")

else:

print(" Incorrect pattern input ")

for line in data\_file: # Read in data P1

if(line== "\n" or line[0] =='#'):

continue

Instructions.append(line)

Nlines+=1

for line in data\_file2: # Read in data from P2

if(line== "\n" or line[0] =='#'):

continue

Instructions2.append(line)

Mlines+=1

if(mode == 1): #Check whether to use disassembler, assembler or simulator

simulate(Instructions,Memory,Nlines,1)

print("simulation complete for Program 1 \n")

# needs memory information printed here

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

simulate(Instructions,Memory,Mlines,2)

print("simulation complete for Program 2 \n")

# needs memory information printed here

elif(mode== 2):

disassembler(Instructions,Nlines,1)

print("disassembler is done for P1")

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

disassembler(Instructions2,Mlines,2)

print("disassembler is done for P2")

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

elif(mode== 3):

#assembler(Instructions,Nlines,1)

#assembler(Instructions2,Mlines,2)

print("assembler is being done")

else:

print("Error. Unrecognized mode. Exiting")

exit()

data\_file2.close()

data\_file.close()

data\_fileA.close()

data\_fileB.close()

data\_fileC.close()

data\_fileD.close()

if \_\_name\_\_ == "\_\_main\_\_":

main()

5) 