Euclid’s Algorithm

**Java or C code:**

// Find m that is relatively prime to n.

int

relPrime(int n)

{

int m;

m = 2;

while (gcd(n, m) != 1) { // n is the input from the outside world

m = m + 1;

}

return m;

}

// The following method determines the Greatest Common Divisor of a and b

// using Euclid's algorithm.

int

gcd(int a, int b)

{

if (a == 0) {

return b;

}

while (b != 0) {

if (a > b) {

a = a - b;

} else {

b = b - a;

}

}

return a;

}

**Assembly Code:**

relPrime: # n is input in $m as an argument

swap $m, $t0 # store n

li 2 # m = 2

whileLoop:

sw $sp[0] # stores m at address ($sp + 0) in mem

copy $m, $t2 # copies m to $t2, the second argument reg

swap $m, $t0 # loads n

sw $sp[1] # stores n at address ($sp + 1) in mem

swap $m, $ra

sw $sp[2] # stores the return address in the stack at $sp + 2

swap $sp, $m

addi $sp, 3 # adds 3 to the stack pointer.

swap $m, $sp # saves the stack pointer, gets back $ra

swap $m, $ra # loads n as first argument.

jal GCD # jump to GCD ( call gcd(n[$m], m[$t2] )

copy $m, $s # saves GCD

li 1

copy $m, $t0 # compare to stored.

lw $sp[-2] # restores M

copy $m, $t2

lw $sp[-1] # restores N

copy $m $t1

lw $sp[0] # restore $ra

copy $m, $ra

copy $sp, $ra

addi -3 # reallocate

copy $m, $sp

copy $t0, $m # set main to GCD

beq $t0, whileLoop

jr $ra

GCD:

swap $m, $t0

li 0

swap $m, $t0

beq $t0, aEqZero # if a ($m) == 0, branch

swap $m, $t2 # load b

whileLoopGCD:

beq $t0, break # if $m = 0, break

slt $t2, $t1 # if (a > b) $t1 = 1, else $t1 = 0

swap $s, $m # store b

li 1 # set compare to

swap $s, $m # flag in $t1, compare to in $s

swap $m, $t1

bne $s else # if not, than else

swap $t1, $m # restore b

sub $t2, $t1 # a = a – b

copy $m, $t2 # store a

swap $t1, $tm # restore b

swap $m, $t2 # place b and a in correct regs.

j whileLoopGCD

else:

swap $t1, $m # restore b

sub $t1, $t2 # b = b - a

copy $m, $t2 # store b

swap $t2, $tm # restore a

j whileLoopGCD

break:

Jr $ra

aEqZero:

swap $m, $t2 # loads b (second arg, $t2)

jr $ra

Function Call

**Java or C code:**

/\*\*

Takes the given number, and adds or subtracts 1 to move the number closer to 10, then returns it. If N is already 10, returns 10.

\*\*/

IncrementToTen(int N){

If (N < 10) {

N++

} else if (N > 10) {

N—

}

Return N

**Assembly Program:**

IncrementToTen:

# procedure call, increments N closer to 10 by 1. (N is input and returned in the main accumulator

# register $m

copy $m, $t0 # saves N

li 10 # sets $m to 10

beq $t0, isTen # if N ($t0) equals 10 ($m) branch…

slt $t0, $t1 # sets a flag in $t1. If N >= 10, $t1 = 1, else $t1 = 0

li 1 # sets comparison value

beq $t1, greaterThan # if N is greater than 10

# if not equal, or greater than, must be less than

lessThan:

swap $m, $t0 # restores N to the accumulator

addi 1 # add one to raise N closer to 10

jr $ra # returns…

isTen:

swap $m, $t0 # restores N to the accumulator

jr $ra # returns…

greaterThan:

swap $m, $t0 # restores N to the accumulator

addi -1 # subtracts one to lower N closer to 10

jr $ra # returns…

Loop

**Java of C code:**

int i = 1;

for (int j = 0; i < 10; j++) {

i \*= 2;

}

**Assembly Program:**

li 1 # load immediate i = 1

copy $m, $t0 # store i on $t0

li 10 # load comparison value 10

copy $m, $t1 # store comparison value

li 0 # load immediate j = 0

Loop:

Beq $t1, End # if $m (j) = 10, it is no longer less than 10

Addi 1 # j++

sll $t0, 1 # i \*= 2

End:

jr $ra # return…