Part 1

a. A1

```
local Generate Display X in
fun {Generate T}
fun {$}
        (T#{Generate (T+1)})
        end
end
proc {Display X T} fun {K Z T1}
   if (T1 == 0) then nil else
   (V#F) = {Z} in
   (V|{K F (T1-1)})
   end
end
local L in
  L = {K X T}
  skip Browse L
  end
end

//Testing
X = {Generate 1} {Display X 10}
end
```

Output -> L: [1,2,3,4,5,6,7,8,9,10]

A2:

```
local A B Generate Display T Times in
        fun {Generate N}
        fun {$} (N#{Generate (N+1)}) end

end

fun {Times X B}
  fun {$}
  (V#F) = {X} in
  ((V*B)#{Times F B}) end
End

proc {Display X N} fun {T Z N1}
  if (N1 == 0) then nil else
  (V#F) = {Z} in
   (V|{T F (N1-1)}) end
end
local L in
L = {T X N}
skip Browse L end
end

//Testing
A = {Generate 1}
B = {Times A 10} {Display B 10}
End
```

Output -> L : [10,20,30,40,50,60,70,80,90,100]

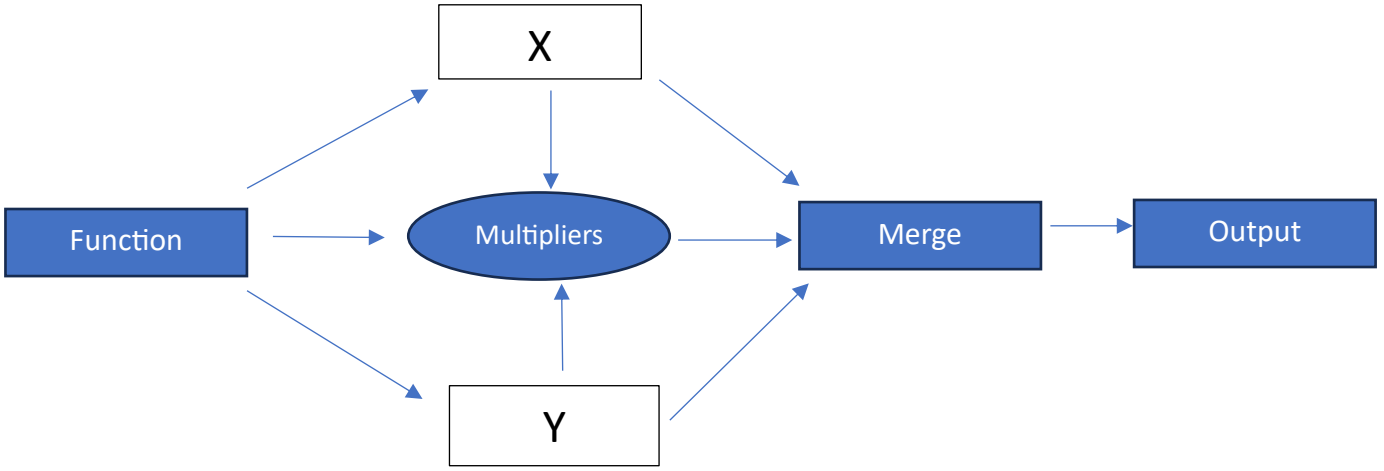
A3:

```
local X Y Generate Display H Merge Times G in
        fun {Generate N}
        fun {$} (N#{Generate (N+1)}) end
end
fun {Times X Y} fun {$}
(V#F) = {X} in
((V*Y)#{Times F Y}) end
End
```

```
Merge = fun {$ X Y}
   fun {$}
   (V#F) = {X}
   (U#G) = {Y} in
   if (V < U) then
  (V#{Merge F Y})  end
  end
  else
  if (V > U) then
  (U#{Merge X G})
  else (V#{Merge F G}) end
  end
  G = fun {$}
  (1#{Merge {Times G 2} {Merge {Times G 3} {Times G 5}}}) end
proc {Display X N}
  fun {H Z Num}
  if (Num == 0) then nil
  else
  (V#F) = {Z} in
  (V|{H F (Num-1)}) end
  end
  local L in
  L = {H X N} skip Browse L end
End

//Testing
X = {Generate 3}
Y = {Generate 5}
{Display X 5}
{Display Y 5}
{Display {Merge X Y} 5}
Emd
```

Output->
L: [3,4,5,6,7]
L: [5,6,7,8,9]
L: [3,4,5,6,7]



b.

```
ModG :: Gen Int -> Int -> Gen Int
modG x n = let G(v,f)= x in
  G ((v mod n), (modG f n))
```

Testing:

```
*Main> G (v,f) = modG (gen 0) 2
*Main> G (v1,f1) = f
*Main> G (v2, f2) = f1
*Main> v
0
*Main> v1
1
*Main> v2 0
```

Merge

```
interleave :: [Gen Int] -> Gen Int
interleave (x:xs) = let G(v1,f1) = x in
      G(v1, (interleave (xs ++ [f1])))
```

Testing:

```
*Main> G (v1,f) = interleave [gen 3, gen 7, gen 13]
*Main> G (v1,f1) = f
*Main> G (v2,f2) = f1
*Main> G (v3,f3) = f2
*Main> v
0
```

```
*Main> v1
 7
*Main> v2
 13
*Main> v5
 6
```

Part 2:

DigitalLogic.txt (Without any edit)

```
local GateMaker AndG OrG NotG A B S IntToNeed Out MulPlex in
fun {GateMaker F}
fun {$ Xs Ys} GateLoop T in
fun {GateLoop Xs Ys}
case Xs of nil then nil
[] '|'(1:X 2:Xr) then
case Ys of nil then nil
[] '|'(1:Y 2:Yr) then
({F X Y}|{GateLoop Xr Yr})
end
end
end
T = thread {GateLoop Xs Ys} end // thread isn't (yet) a
returnable expression
T
end
end
fun {NotG Xs} NotLoop T in
fun {NotLoop Xs}
case Xs of nil then nil
[] '|'(1:X 2:Xr) then ((1-X)|{NotLoop Xr})
end
end
T = thread {NotLoop Xs} end // thread isn't (yet) a
returnable expression
T
end
```

```
AndG = ... // Use GateMaker
OrG = ... // Use GateMaker
fun {IntToNeed L}
...
end
fun {MulPlex A B S}
...
end
A = {IntToNeed [0 1 1 0 0 1]}
B = {IntToNeed [1 1 1 0 1 0]}
S = [1 0 1 0 1 1]
Out = {MulPlex A B S}
// run a loop so the MulPlex threads can finish before displaying Out
local Loop in
proc {Loop X}
if (X == 0) then skip Basic
else {Loop (X-1)} end
end
{Loop 1000}
end
skip Browse Out
end
```

a.

```
fun {IntToNeed L}
  case L of nil then nil
   [] '|'(1:X 2:Ar) then
   T W in byNeed fun {$} X end W
   T = {IntToNeed Ar}
   (W|T)
   end
end

fun {MulPlex A B S} R Z T W in R = {NotG S}
Z = {AndG R A} T = {AndG S B} W = {OrG Z T} W
End


A = {IntToNeed [0 1 1 0 0 1]}
B = {IntToNeed [1 1 1 0 1 0]}
S = [1 0 1 0 1 1]
Out = {MulPlex A B S}
```

```
// run a loop so the MulPlex threads can finish before displaying Out
local Loop in
  proc {Loop X}
    if (X == 0) then skip Basic
    else {Loop (X-1)} end
end
{Loop 1000}
end
skip Browse Out
skip Full end
fun {IntToNeed L}
case L of nil then nil
[] '|'(1:X 2:Xr) then T W in byNeed fun {$} X end W T = {IntToNeed Xr} (W|T)
End
End
```

b.

```
AndG = {GateMaker fun {$ A B} if (A == 0) then 0 else (A*B) end end}
OrG = {GateMaker fun {$ A B} if (A == 1) then 1 else (A+B) end end}
```

c.

```
Fun {MulPlex A B S} {OrG {AndG S B} {AndG {NotG S} A}}  end
End
```

d.

d.1:

To determine which values of A and B will not be needed, we can analyze the values of S (the selection bits)

For S=0, B bit is not needed

For S=1, A bit will not be needed

d.2:

After uncommenting the line, the trace statement "Needed: "++show id2 will display the values of id2, which corresponds to the store locations of the variables that are needed. They match with the resultant output of d1