Part 1:

a)

```
local  A B Gen Out Op Merge Times Hamming in
        fun {Gen N}
                fun {$} (N#{Gen (N+1)}) end
        end

        fun {Times A B}
                fun{$}
                        (J#K) = {A} in
                        ((J*B)#{Times K B})
                end
        end

        Merge = fun {$ A B}
                fun {$}
                        (J#K) = {A}
                        (H#I) = {B} in
                        if (J < H) then
                                (J#{Merge K B})
                        else
                        if (J > H) then
                                (H#{Merge A I})
                        else
                                (J#{Merge K I})
                        end
                        end
                end
        end

        Hamming = fun {$}
                (1#{Merge{Times Hamming 2}{Merge {Times Hamming 3}{Times
Hamming 5}}})
                end

        proc {Out A N}
                fun{Op Z Num}
                if (Num == 0) then
                        nil
                else
                        (J#K) = {Z} in
                        (J|{Op K (Num-1)})
                end
        end
```

```
            local List in
                List = {Op A N}
                skip Browse List
                end
            end

            //Edit third parameter to gen first N of Hamming
            {Out Hamming 12}
    end
```

b)

```
    data Gen a = G(() -> (a, Gen a))

    generate :: Int -> Gen Int
    generate n = G(\_-> (n, generate(n+1)))

    gen_take:: Int -> Gen a -> [a]
    gen_take 0 _ = []
    gen_take n (G f) = let (a, g) = f() in a : gen_take (n-1) g

    times :: Int -> Gen Int -> Gen Int
    times n (G f) = let (x,g) = f() in G(\_->((n*x), times n g))

    merge :: Gen Int -> Gen Int -> Gen Int
    merge (G x) (G y) = let (v, g) = x() in let (a, b) = y() in
       if v < a then G(\_->(v, merge g(G y)))
       else if v > a then G(\_-> (a, merge b (G x)))
       else G(\_-> (v, merge g b))

    hamming :: Gen Int -> Gen Int
    hamming (G f) = let (x,g) = f() in G(\_->(1,merge (times 2 (hamming g)) (merge (times 3
    (hamming g)) (times 5 (hamming g))) ))

    --ghci>: gen_take 10 (generate 1)
    --[1,2,3,4,5,6,7,8,9,10]
```

Part 2:

a)

```
fun {IntToNeed L}
                    case L of nil then nil
                    [] '|'(1:X 2:Xr) then O P in
                byNeed fun{$} X end P
                    O = {IntToNeed xr}
                    (P|O)
                    end
 end
```

b)

```
AndG ={GateMaker fun {$ X Y} if (X==0) then 0 else (X*Y) end end}
OrG ={GateMaker fun {$ X Y} if (X==1) then 1 else ((X+Y)-(X*Y)) end end}
```

c)

```
 fun {MulPlex A B S} E F G H in
    E = {NotG S}
        F = {AndG E A}
        G = {AndG S B}
        H = {OrG F G}
        H
 End
```

d) 2

E = 0 1 0 1 0 0

F =
0 <0> = 0
1 1 = 1
0 <1> = 0
1 0 = 0
0 <0> = 0
0 <1> = 0

G=
1 1 = 1
0 <1> =0
1 1 = 1
0 <0> = 0
1 1 = 1
1 0 = 0

H =
0 1 = 1
1 <0> =1
0 1 =1
0 0 =0
0 1 = 0
0 0 = 0

The numbers in < > are not needed.