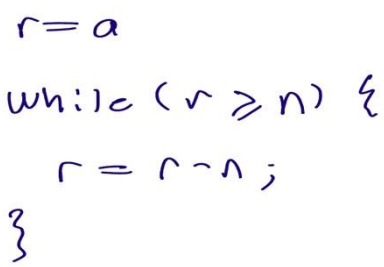
Arhan UNAY
Utku GEMICIOGLU

02.04.2025
INFO3 Groupe 2

TP : Réalisation d'une machine algorithmique

5.3 Modification de l'algorithme

1.



Pour calculer le modulo d'un entier en n'utilisant que des soustractions, l'idée est de retirer répétitivement la valeur n de l'entier a jusqu'à ce que le résultat soit strictement inférieur à n. Autrement dit, si l'on souhaite calculer r = a mod n, on peut procéder de la manière suivante :

Cette méthode garantit que le résultat final r satisfait 0 ≤ r < n.

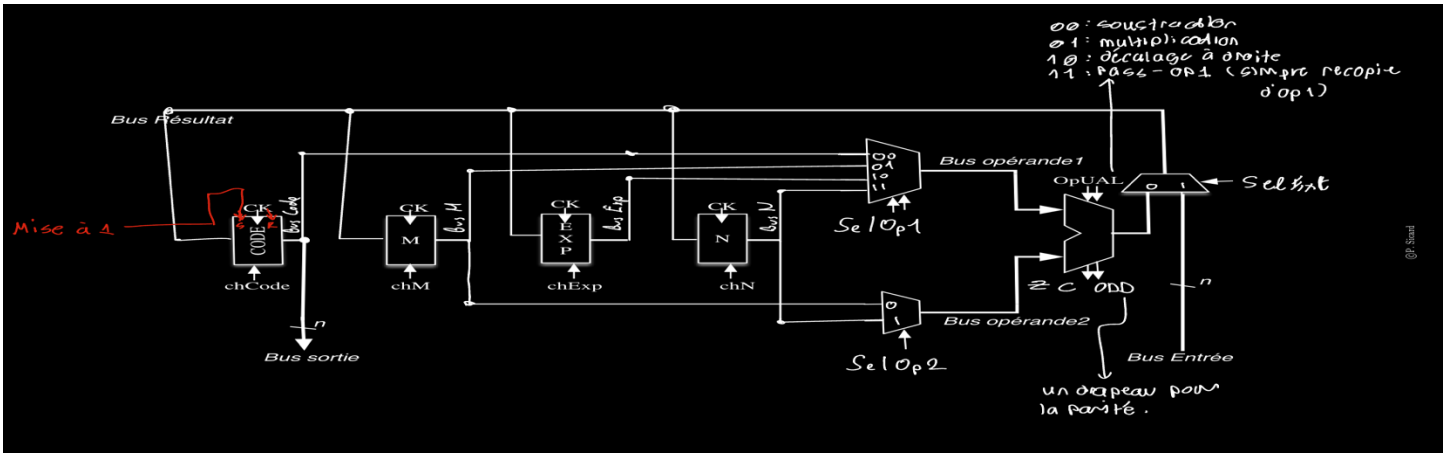
2.

```c
#include <stdio.h>
#include <stdlib.h>

void main(int argc, char * argv[])
{
    unsigned long int m, exp, n;
    unsigned long int code, temp;

    if (argc != 4) exit(0);
    sscanf(argv[1], "%ld", &m);
    sscanf(argv[2], "%ld", &exp);
    sscanf(argv[3], "%ld", &n);

    code = 1;
    while (exp > 0) // exp != 0 donc flag (not Z)
    {
        if ((exp & 1) > 0)  //(LSB == 1) Last Significant Bit
        {
            // code = (code * m) % n, calculé par soustractions répétées
            code = code * m;
            while (code >= n) //while C
                code = code - n;

        }
        exp = exp >> 1;  // màj flag Z
        // m = (m * m) % n, calculé par soustractions répétées
        m = m * m;
        while (m >= n) //while C
            m = m - n;

    }
    printf("Resultat chiffrement %ld\n", code);
}
```

Arhan UNAY
Utku GEMICIOGLU

5.4 La partie opérative

2.



3.

```
-- Instruction      |Op 1      |Op 2    |Opération      |Résultat chargé dans
-- code ← code * m  |CODE      |M       |MUL            |CODE
-- m ← m * m        |M         |M       |MUL            |M
-- code ← code - n  |CODE      |N       |SUB            |CODE
-- exp ← exp >> 1   |EXP       |-       |SHR1           |EXP

-- Il nous faut donc quatre traitements dans l'UAL : PASS OP1, SUB, MUL, SHR1 (shift logique d'un bit vers la droite).

-- Codage des commandes opUAL
-- On garde un codage compact sur 2 bits :

-- opUAL[1:0]|Signification                    |Drapeaux générés
-- 00        |   SUB (Op 1 - Op 2)             |  Z, C (emprunt), N
-- 01        |   MUL (Op 1 x Op 2)             |  Z
-- 10        |   SHR1 (décalage à droite de Op 1)|  Z, C (bit perdu)
-- 11        |   PASS-OP1 (simple recopie d'Op 1)|  Z (=0 ?),Odd

-- ------------------------------------------------------------------------
```

Arhan UNAY
Utku GEMICIOGLU

02.04.2025
INFO3 Groupe 2

```
-- Définition d'un soustracteur n bits
include "../multv6base2.lus"
include "../entierv6.lus"
include "../UALCryptoV6.lus"

function subnbits<<const n: int>>(A, B: bool^n) returns (S: bool^n; Borrow: bool);
var
    B_out: bool^n;
let
    -- Complément à 2 de B pour la soustraction (A - B = A + (~B + 1))
    B_out = map<<not, n>>(B); -- Complément à 1 en inversant les bits
    (S, Borrow) = add_n_bits<<n>>(A, B_out, true); -- +1 pour le complément à 2
tel;


-- Notre définition de l'UAL
function UAL<<const n: int>>(
    OpUAL: bool^2;       -- Commande de l'UAL (2 bits)
    Op1, Op2: bool^n;    -- Opérandes (n bits)
)
returns (
    C, Z, Odd: bool;  -- Drapeaux de statut
    Result: bool^n    -- Résultat de l'UAL (n bits)
);
var
    Mult_Result: bool^(2*n);  -- Résultat de la multiplication (2*n bits)
    Sub_Result: bool^n;       -- Résultat de la soustraction
    Borrow: bool;             -- Retenue de la soustraction
    Shift_Result: bool^n;     -- Résultat du décalage
    Exp_Is_Odd: bool;         -- EXP & 1 = 1 ?
let
    -- Multiplication (code * m)
    --- On évite de faire le calcul selon OpUAL
    Mult_Result = multnbits<<n>>(Op1, Op2);
    -- Soustraction (pour le modulo)
    (Sub_Result, Borrow) = subnbits<<n>>(Op1, Op2);
    -- Décalage à droite (exp >> 1)
    Shift_Result = shiftl<<n>>(Op1, false) ;

    -- Résultat final (selon OpUAL)
    Result = map <<mux4, n>>(
        OpUAL[1]^n, OpUAL[0]^n,
        Sub_Result, Mult_Result[0..n-1], Shift_Result, Op1
     );
    -- Calcul des flags
    Z = (Result = false^n);  -- Z: EXP = 0 ?
    Exp_Is_Odd = Result[0];            -- EXP & 1 = 1 ?

    C = not Borrow;             -- Emprunt de la soustraction
    Odd = Exp_Is_Odd;             -- EXP & 1 = 1 ?
```

Arhan UNAY
Utku GEMICIOGLU


```
tel

-- Exemple d'utilisation de l'UAL comme l'exemple donné dans le UALcryptoV6
function ualaff_2<<const n:int>>(  A , B : bool^n ; OpUAL : bool^2 )
returns ( C,Z,Odd : bool  ; T : bool^n;  AE,BE,R:int );
let
(C,Z,Odd,T )=UAL<<n>>( OpUAL, A , B );

R=entiernat<<n>>(T);
AE=entiernat<<n>>(A);
BE=entiernat<<n>>(B);

tel;
node ualinst_2=ualaff<<a>>;
```

4.

```
-- Version sans CK
--------------------------------------------------------------------------
include "../memoire.lus"

node RegN <<const n : int>>
  (ent    : bool^n;    -- bus de données à charger
   char   : bool;      -- signal de chargement commun
   reset  : bool;      -- reset synchrone (met tous les bits à 0)
   set    : bool)      -- set asynchrone (met tous les bits à 1)
  returns
  (sort      : bool^n);   -- bus de sortie (état du registre)
let
  -- map applique n fois le noeud bascule,
  -- en dupliquant char, reset et set en vecteurs de taille n
  sort = map <<bascule; n>>
         ( ent,         -- ent[i]
           char^n,      -- char pour chaque bit
           reset^n,     -- reset pour chaque bit
           set^n        -- set pour chaque bit
         );
tel
-- Pour juste 4 bits on instancie :
node Reg4 = RegN<<4>>;


-- Version avec CK
--------------------------------------------------------------------------

-- Registre n bits avec chargement conditionnel
node registre_n_bits<<const n: int>>(
    CK: bool;        -- Horloge globale du circuit --On n'utilise pas dans le code juste pour la clarté
    chM: bool;       -- Signal de chargement (connecté à CHAR des bascules)
    reset: bool;     -- Reset global (actif à 1)
    set: bool;       -- Set global (actif à 1)
    D: bool^n        -- Donnée d'entrée
)
returns (
    Q: bool^n        -- Sortie du registre
);
let
    -- Génération des n bascules en parallèle
    Q = map<<bascule; n>>(D, chM^n, reset^n, set^n);

    -- Connexion implicite de l'horloge :
    -- Le signal CK est implicite dans l'opérateur -> et pre de la bascule
tel;

--exemple
-- Registre M de 8 bits
node registre_M(
    Clock: bool;     -- Horloge --On n'utilise pas dans le code juste pour la clarté
    load_M: bool;    -- Commande de chargement
    rst: bool;       -- Reset
    data_in: bool^8 -- Bus d'entrée
)
returns (
    M_out: bool^8    -- Sortie
);
let
    M_out = registre_n_bits<<8>>(Clock, load_M, rst, false, data_in);
    -- set mis à false car on n'initialise pas le registre M avec 1
tel;
```

Arhan UNAY
Utku GEMICIOGLU

02.04.2025
INFO3 Groupe 2

5.

```
-- 1) On inclut d'abord tout ce qu'il faut :
--    • Les bascules les registres et UAL à n bits
include "../memoire.lus"
include "3.lus"
include "4.lus"
include "../entierv6.lus"
include "../multv6base2.lus"
include "../UALCryptoV6.lus"


-------------------------------------------------------------------
-- 2) Node PO générique à n bits ----------------------------------
-------------------------------------------------------------------
node PO <<const n:int>>(
  Clock    : bool;      -- horloge
  Reset    : bool;      -- reset synchrone =1
  selExt   : bool;      -- 1 = charger depuis busIn (initialisation), 0 = recirculer UAL
  busIn    : bool^n;    -- bus d'entrée externe (M, EXP ou N)
  chCode   : bool;      -- load CODE
  chM      : bool;      -- load M
  chExp    : bool;      -- load EXP
  chN      : bool;      -- load N
  selOp1   : bool^2;    -- code 00..11 pour choisir CODE/M/EXP/N en op1
  selOp2   : bool;      -- code 0 1 pour choisir M/N en op2
  opUAL    : bool^2 ;    -- commande UAL (00=sub, 01=mul, 10=shr1, 11=pass)
  mise_1    : bool      -- mise à 1 de CODE (pour le premier tour)
)
returns (
  busOut   : bool^n;    -- seule sortie visible : CODE
  Z, C, Odd     : bool;     -- flags Zero et Carry (emprunt)
  CodeInt, MInt, NInt, ExpInt : int  -- pour le debug
);
var
  CODE, M, EXP, Nreg : bool^n;  -- registres internes
  op1, op2          : bool^n;  -- bus opérandes vers l'UAL
  resUAL            : bool^n;  -- résultat de l'UAL
  Borrow, Zflag : bool;     -- emprunt, zero renvoyés par l'UAL
  setVecCode: bool^n;  -- vecteur de set pour CODE
let

  -------------------------------------------------------------------
  setVecCode = [true] | false^(n-1); -- vecteur à 1 seulement en position 0
  -- CODE = if mise_1 then [true] | registre_n_bits<<n>>(Clock, chCode, Reset, false,
  --            if selExt then busIn else resUAL)
  --            else registre_n_bits<<n>>(Clock, chCode, Reset, false,
  --            if selExt then busIn else resUAL);

    -- Instanciation "à la main" de CODE via map<<bascule;n>> pour passer setVecCode
    CODE = map <<bascule; n>>(
          -- ent : si phase init (selExt) on prend busIn, sinon resUAL
          if selExt then busIn else resUAL,
          -- char  : chCode répliqué
          chCode^n,
          -- reset : Reset répliqué
          Reset^n,
          -- set   : notre setVecCode
          if mise_1 then setVecCode else false^n
        );



  -- On charge CODE en premier, puis les autres registres
  --peut etre false au lieu de  Reset and not mise_1
  M    = registre_n_bits<<n>>(Clock, chM, Reset and not mise_1, false,
```

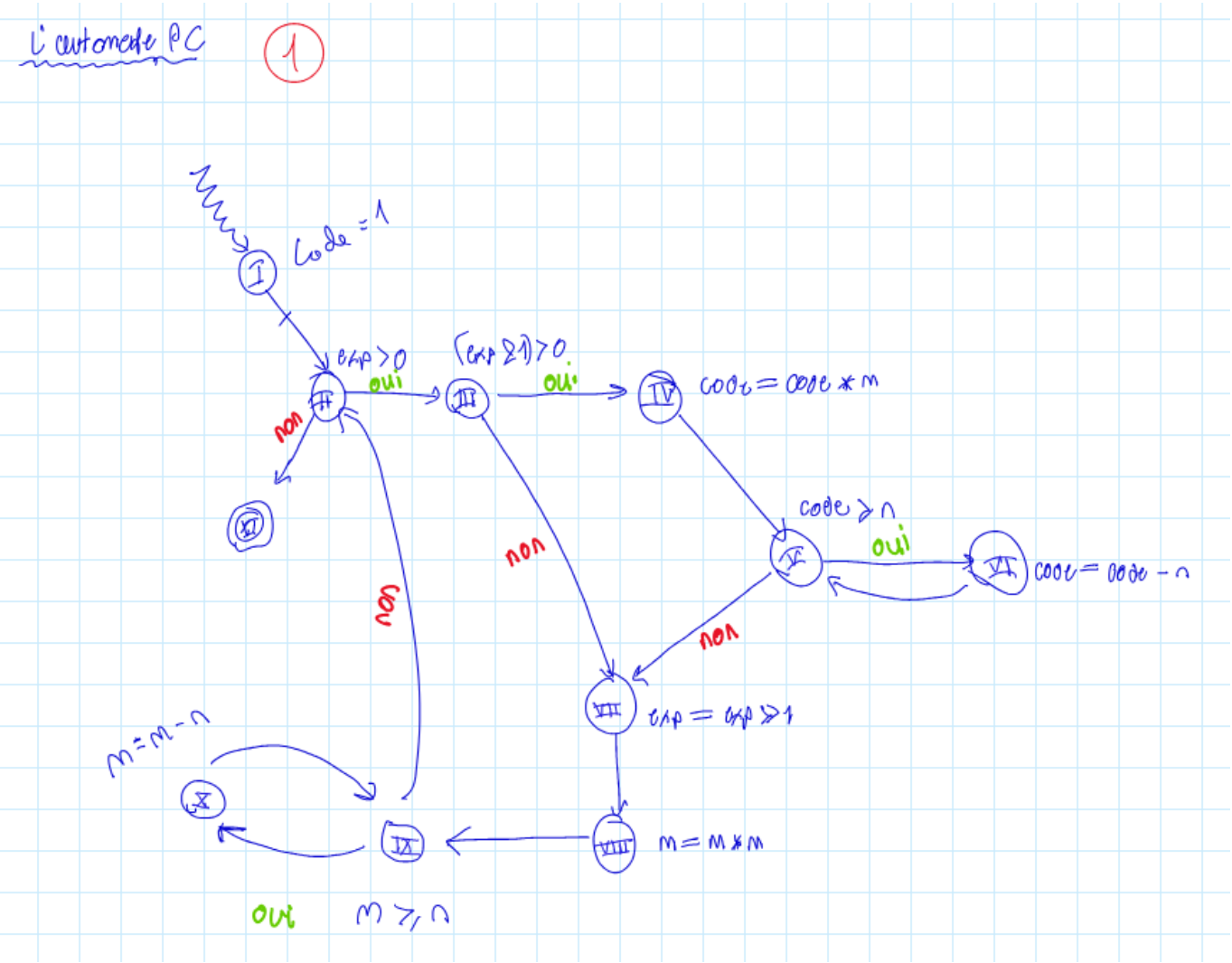
```
  -- On charge CODE en premier, puis les autres registres
  --peut etre false au lieu de  Reset and not mise_1
  M    = registre_n_bits<<n>>(Clock, chM, Reset and not mise_1, false,
                              if selExt then busIn else resUAL);
  EXP  = registre_n_bits<<n>>(Clock, chExp, Reset and not mise_1, false,
                              if selExt then busIn else resUAL);
  Nreg = registre_n_bits<<n>>(Clock, chN, Reset and not mise_1, false,
                              if selExt then busIn else resUAL);

  ------------------------------------------------------------------
  -- 4) Multiplexeurs 4:1 et 2:1 pour sélectionner les deux opérandes
  op1 = map <<mux4; n>>(
              selOp1[1]^n, selOp1[0]^n,
              CODE, M, EXP, Nreg);

  op2 = map <<mux1; n>>(
              selOp2^n,
              Nreg, M);

  ------------------------------------------------------------------
  -- 5) Appel de l'UAL (issue de 3.lus) :
  --    UAL(opUAL, op1, op2) returns (C, Z, Odd, Result)
  (Borrow, Zflag, Odd, resUAL) = UAL<<n>>(opUAL, op1, op2);

  ------------------------------------------------------------------
  -- 6) Bus de sortie et drapeaux
  busOut = CODE;     -- on n'exporte que CODE
  C      = not Borrow;  -- emprunt de la soustraction
  Z      = Zflag;   -- résultat nul ?

  --pour le voir mieux --debug
  CodeInt = entiernat<<n>>(CODE);
  MInt = entiernat<<n>>(M);
  NInt = entiernat<<n>>(Nreg);
  ExpInt = entiernat<<n>>(EXP);

tel

-- Instanciation pour n=8 bits, par exemple
node PO8 = PO<<8>>;
```

5.5 La partie contrôle

1.

Arhan UNAY
Utku GEMICIOGLU

02.04.2025
INFO3 Groupe 2

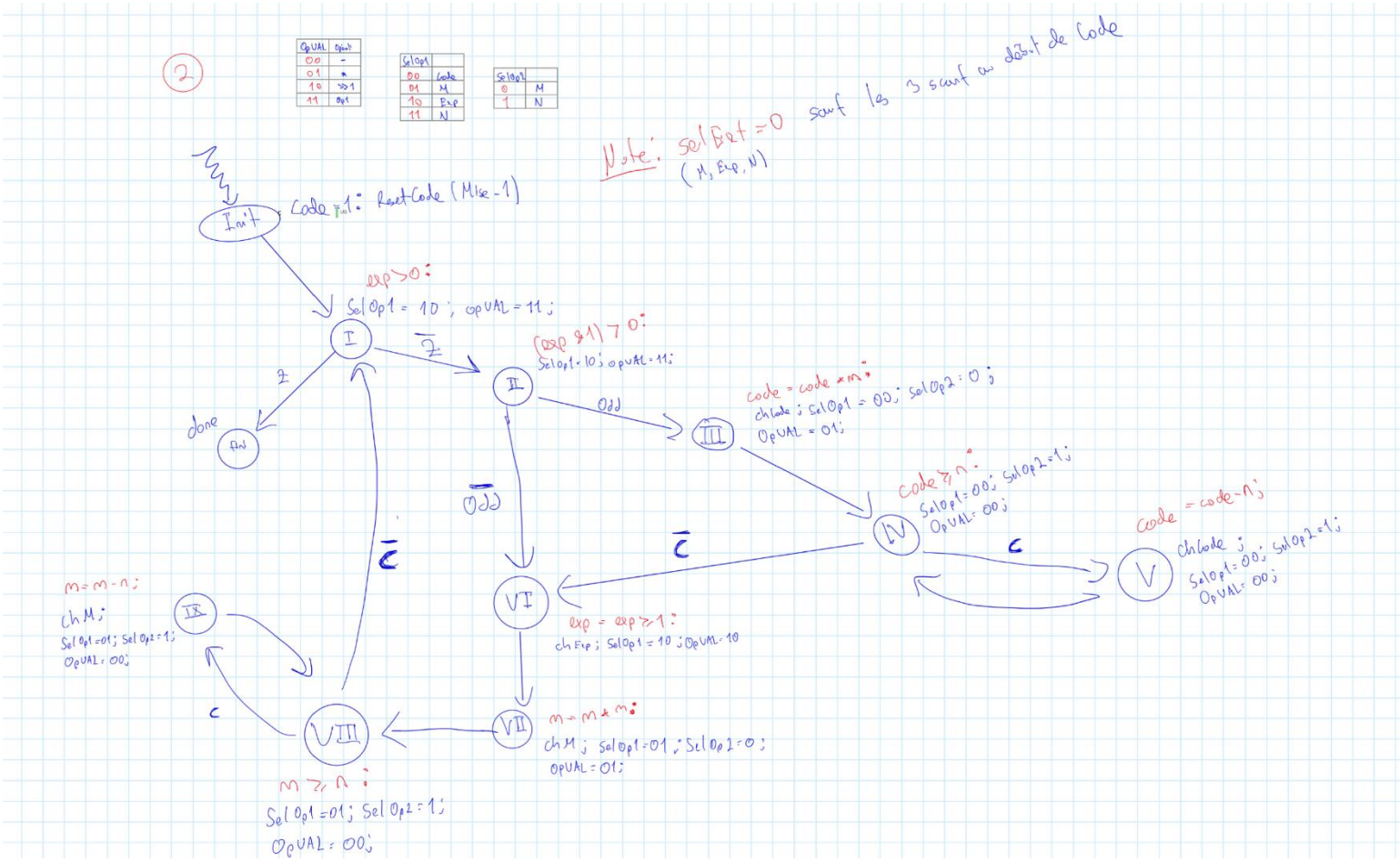


2.

N.B : On n'indique pas explicitement les valeurs qui ne sont pas utilisées dans l'état en question.

Arhan UNAY
Utku GEMICIOGLU

02.04.2025
INFO3 Groupe 2



Arhan UNAY
Utku GEMICIOGLU

02.04.2025
INFO3 Groupe 2

3.

```
--------------------------------------------------------------------------
include "../memoire.lus"

node PC(
  CLK    : bool;       -- Horloge --ce n'est pas utilisé dans le code , juste pour la simulation
  RESET  : bool;       -- Reset synchrone (actif haut)
  Zexp   : bool;       -- Flag EXP = 0
  C      : bool;       -- Flag Carry (pour modulo)
  Odd    : bool        -- Flag EXP impair (LSB)
)
returns (
  -- Signaux de chargement
  chExp, chCode, chN, chM : bool;
  -- Sélection entrée
  selExt : bool;
  -- Contrôle multiplexeurs
  selOp1 : bool^2;
  selOp2 : bool;
  -- Commande UAL
  opUAL  : bool^2;
  -- Indicateur de fin
  done   : bool;
  --signale de mise à 1
  mise_1 : bool;
  -- Signaux de débogage
  etat_courant : bool^11

);
var
  st, st_next : bool^11;   -- vecteurs un-among-11

let
  st[0] = bascule(st_next[0], true , false, RESET); --initial;
  st[1..10] = map <<bascule;10>>(
    st_next[1..10], -- Entrée D
    true^10,        -- CHAR toujours actif
    RESET^10,       -- Reset synchrone
    false^10        -- Set initial
  );

  --------------------------------------------------------------------------
  -- 3) Machine à états (version optimisée)
  st_next = [

   -- État 0 (initCode) → État 1 (loopTest)
    RESET,

    -- Etat 1 : exp>0
    st[0] or (st[8] and not C),

    -- État 2 : (exp & 1) > 0
    st[1] and not Zexp ,

    -- État 3: code = code * m
    st[2] and Odd,

    -- État 4 : code >= n
    st[3] or st[5],

    -- État 5: cpde = code - n;
    st[4] and C,

    -- État 6: exp = exp >> 1
    (st[2] and not Odd) or (st[4] and not C),
```

```
    -- État 6: exp = exp >> 1
    (st[2] and not Odd) or (st[4] and not C),

    -- État 7 : m = m*n
    st[6],

    -- État 8 : m>= n
    st[7] or st[9],

    -- État 9 : m = m - n
    st[8] and C,

    -- État 10 : m = m >> 1
    st[1] and Zexp
  ];

  ------------------------------------------------------------------------
  -- 4) Génération des signaux de contrôle (version clarifiée)
  mise_1 = st[0];
  -- Chargements initiaux
  chCode = st[0] or st[3] or st[5] ;
  chExp  = st[6];
  chM = st[7] or st[9];

  -- On n'utilise pas selExt, chN  comme nous ne faisons que les affectations au debut de trois scanf
  selExt = st[0];
  chN  = false;

  -- Sélection entrée externe pendant l'init
  --selExt = st[0];

  -- Contrôle des multiplexeurs (table de vérité optimisée)
  selOp1 = [
    -- Bit 0: 1 pour M
    st[7] or st[8] or st[9],

    -- Bit 1: 1 pour  Exp
    st[1] or st[2] or st[6]

  ];

  -- selOp2 = 1 pour N dans multiply(5)/shift(6)/square(7)
  selOp2 = st[4] or st[5] or st[8] or st[9];

  -- Commande UAL (nouveau codage plus lisible)
  opUAL = [
    st[1] or st[2] or st[3] or st[7],
    st[1] or st[2] or st[6]

  ];

  -- Signal de fin
  done = st[10];

  -- Nouveau: visualisation de l'état courant pour débogage
  etat_courant = st;
tel
```

4.

Arhan UNAY
Utku GEMICIOGLU

L'initialisation du circuit repose sur un **reset synchrone actif-haut** partagé par la partie contrôle (PC) et la partie opérative (PO). Au premier front montant de l'horloge après assertion de **RESET=1**, la bascule st[0] reçoit sa ligne **set** et passe à 1, tandis que toutes les autres bascules st[1..10] sont simultanément remises à 0 grâce à leur ligne **reset**. Cet unique "1" dans le vecteur d'état identifie l'**état 0** de la machine de contrôle et, via le décodage de cet état, active automatiquement les signaux mise_1 = 1, selExt = 1, chCode = 1 (ainsi que, si nécessaire, chM, chExp et chN).

Dans la PO, **mise_1=1** alimente la ligne **set** du registre CODE avec un vecteur [1,0,...,0], ce qui lui donne la valeur 1, tandis que **selExt=1** et **chCode=1** ordonnent à tous les registres (CODE, M, EXP, N) de charger depuis le bus d'entrée leurs valeurs initiales. Dès que **RESET** retombe à 0, la machine conserve ce vecteur d'état "1 000...0" un cycle puis passe à l'état 1, lançant ainsi l'exécution normale de l'algorithme de chiffrement.

5. La réunion de deux parties PC et PO : top.lus

```
-- top.lus
-- dépendances : PC.lus, PO.lus (votre 5.lus renommé), multv6base2.lus, UALCryptoV6.lus
include "PC.lus"
include "../PO/5.lus"


node top_debug <<const n:int>>(
  begin    : bool;     -- checkbox "begin"
  chM      : bool;     -- checkbox "chM"
  chEXP    : bool;     -- checkbox "chEXP"
  chN      : bool;     -- checkbox "chN"
  input    : int       -- slider "input" (valeur entière à charger)
)
returns (
  m        : int;   -- affichage registre M
  exp      : int;   -- affichage registre EXP
  nreg     : int;   -- affichage registre N
  end_     : bool;     -- flag "end_"
  q        : int;    -- copie de "code" si vous voulez un label supplémentaire
  -- Signaux de débogage
  etat_courant : bool^11;

  Sig_chExp : bool;
  Sig_chCode : bool;
  Sig_chN: bool;
  Sig_chM : bool;

  mise_1 : bool;

  selOp1 : bool^2;     -- commande UAL (issue de PC)
  selOp2 : bool;
  opUAL : bool^2;     -- commande UAL (issue de PC)

  Sig_C : bool;       -- flag Carry
  Sig_Z : bool;       -- flag Zero
  Sig_Odd : bool;     -- flag Odd


);
var
  Z, C, Odd        : bool;       -- flags retour UAL
  --selOp1           : bool^2;     -- commande UAL (issue de PC)
  --selOp2           : bool;
  --opUAL            : bool^2;
  -- vecteur de bits issu du slider
  busIn            : bool^n;
  -- bus interne de sortie PO
  code_bits        : bool^n;

  reset : bool;       -- reset PO


  code : int;  -- code de sortie
  --signale de mise à 1

  Sig_chM_interne : bool;
  Sig_chExp_interne : bool;
  Sig_chN_interne : bool;
  Sig_chCode_interne : bool;

  -- vecteurs de bits issus des registres
  selExt : bool;  -- 1 = charger depuis busIn (initialisation), 0 = recirculer UAL
let
```

Arhan UNAY
Utku GEMICIOGLU

02.04.2025
INFO3 Groupe 2

```
let
  --------------------------------------------------------------
  -- conversion de l'entier 'input' en bus de n bits (LSB à l'indice 0)
  busIn = natier16(input);

  --------------------------------------------------------------
  -- Partie opérative (PO) : on charge manuellement via 'begin',chM,chEXP,chN
  ( code_bits, Z, C, Odd ,code, m, nreg, exp ) =
    PO<<n>>(false,
            reset,      -- reset PO = false (on ne reset pas ici)
            not begin,      -- selExt = !begin
            busIn,      -- bus d'entrée
            Sig_chCode_interne,      -- chCode = begin (on force reset à 1 pour initialiser CODE=1)
            Sig_chM_interne,        -- chM     = checkbox chM
            Sig_chExp_interne,      -- chExp   = checkbox chEXP
            Sig_chN_interne,        -- chN     = checkbox chN
            selOp1,     -- commande alu issue de PC
            selOp2,
            opUAL,
            mise_1
    );

    Sig_chExp_interne = mux1(begin, Sig_chExp, chEXP);
    Sig_chN_interne = mux1(begin, Sig_chN, chN);
    Sig_chM_interne = mux1(begin, Sig_chM, chM);
    Sig_chCode_interne = mux1(begin, Sig_chCode, false);

    reset = (not pre begin) and begin;


  --------------------------------------------------------------
  -- Partie contrôle (PC) : on reset PC quand begin=0→1, on récupère
  -- uniquement les commandes sorties; on ignore ici selExt/chCode/ch*
  ( Sig_chExp, Sig_chCode, Sig_chN, Sig_chM,
    selExt, selOp1, selOp2,
    opUAL, end_, mise_1,etat_courant ) =
    PC(false,
      not begin,  -- RESET_PC = 1 quand begin passe à 0→1 (synchronous)
      Z, C, Odd);

  --------------------------------------------------------------
  -- sorties finales
  q    = entiernat<<n>>(code_bits);  -- alias si besoin
  Sig_C = C;
  Sig_Z = Z;
  Sig_Odd = Odd;

tel;

------
--utilisation top_debug
-- inst_top16 lus

-- On fixe n = 16
node top16(
  begin   : bool;
  chM     : bool;
  chEXP   : bool;
  chN     : bool;
  input   : int
)
returns (
  m       : int;
```

Arhan UNAY
Utku GEMICIOGLU


```
    input   : int
  )
  returns (
    m      : int;
    exp    : int;
    n      : int;
    end_   : bool;
    q      : int;
    -- Signaux de débogage
    etat_courant : bool^11;
    Sig_chExp : bool;
    Sig_chCode : bool;
    Sig_chN: bool;
    Sig_chM : bool;
    mise_1 : bool;
    selOp1 : bool^2;    -- commande UAL (issue de PC)
    selOp2 : bool;
    opUAL : bool^2;     -- commande UAL (issue de PC)
    Sig_C : bool;       -- flag Carry
    Sig_Z : bool;       -- flag Zero
    Sig_Odd : bool;     -- flag Odd
  );
  let
    -- On déplie top_debug avec n=8
    (m, exp, n, end_, q, etat_courant,Sig_chExp,Sig_chCode,Sig_chN,Sig_chM, mise_1, selOp1, selOp2, opUAL, Sig_C, Sig_Z, Sig_Odd) =
     top_debug<<16>>(begin, chM, chEXP, chN, input);
  tel;



  -- natier :

  -- natier8.lus
  -- Convertit un entier i en vecteur de 8 bits (LSB en position 0)
  function natier8(i:int) returns (b: bool^8);
  let
    b[0] = (i mod 2) = 1;
    b[1] = ((i / 2)   mod 2) = 1;
    b[2] = ((i / 4)   mod 2) = 1;
    b[3] = ((i / 8)   mod 2) = 1;
    b[4] = ((i / 16)  mod 2) = 1;
    b[5] = ((i / 32)  mod 2) = 1;
    b[6] = ((i / 64)  mod 2) = 1;
    b[7] = ((i / 128) mod 2) = 1;
  tel;

  -- Convertit un entier i en vecteur de 16 bits (LSB en position 0)
  function natier16(i:int) returns (b: bool^16);
  let
    b[0]  = (i mod 2) = 1;
    b[1]  = ((i / 2)   mod 2) = 1;
    b[2]  = ((i / 4)   mod 2) = 1;
    b[3]  = ((i / 8)   mod 2) = 1;
    b[4]  = ((i / 16)  mod 2) = 1;
    b[5]  = ((i / 32)  mod 2) = 1;
    b[6]  = ((i / 64)  mod 2) = 1;
    b[7]  = ((i / 128) mod 2) = 1;

    b[8]  = ((i / 256) mod 2) = 1;
    b[9]  = ((i / 512) mod 2) = 1;
    b[10] = ((i / 1024) mod 2) = 1;
    b[11] = ((i / 2048) mod 2) = 1;
    b[12] = ((i / 4096) mod 2) = 1;
    b[13] = ((i / 8192) mod 2) = 1;
    b[14] = ((i / 16384) mod 2) = 1;
    b[15] = ((i / 32768) mod 2) = 1;

  tel;
```

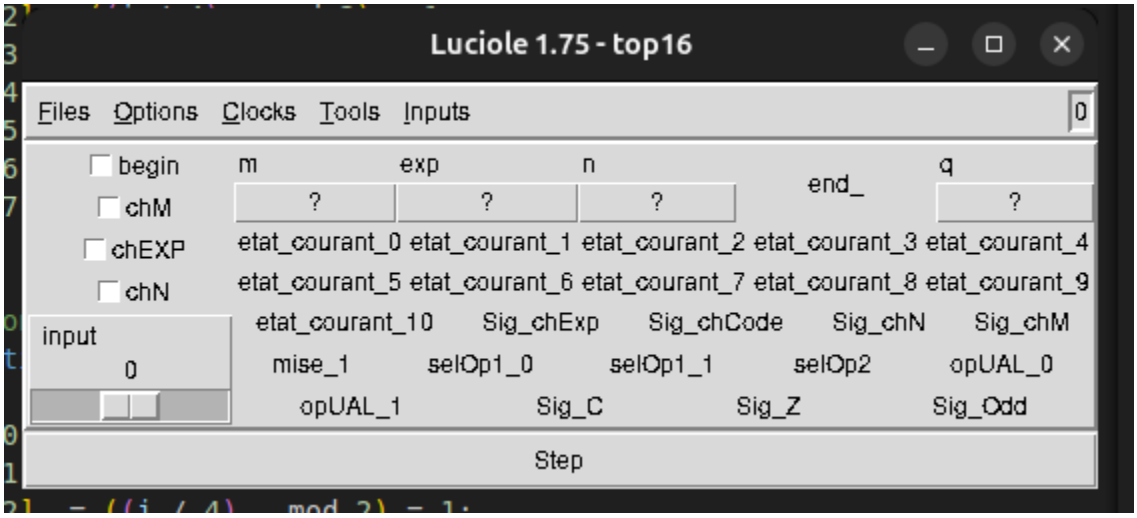
Arhan UNAY
Utku GEMICIOGLU

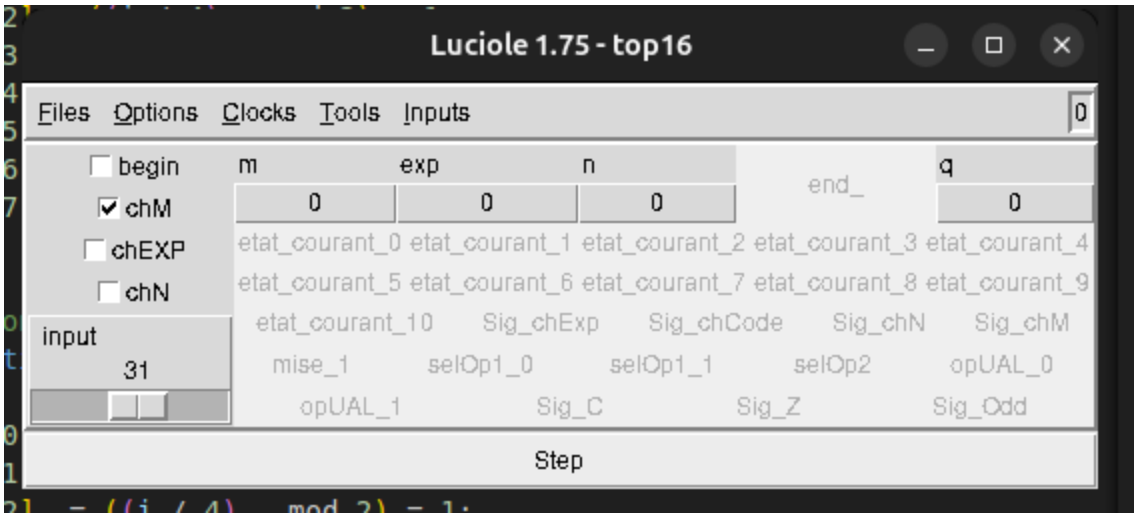
02.04.2025
INFO3 Groupe 2

Jeux de Tests

Avec p=3 et q=11: (13,33), (17,33) et m =31:

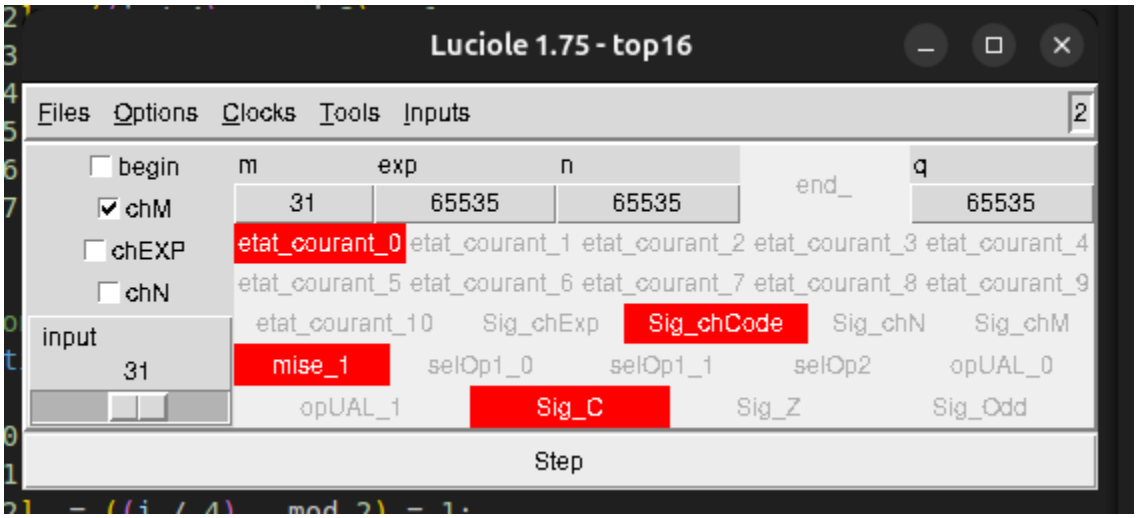
```
[~/Desktop/repertoire/ArchMat/Real_mach_alg]
base  arhis  arhan-master -  ./chiffrement 31 13 33
Resultat chiffrement 25

[~/Desktop/repertoire/ArchMat/Real_mach_alg]
base  arhis  arhan-master -
0 0  Live Share
```

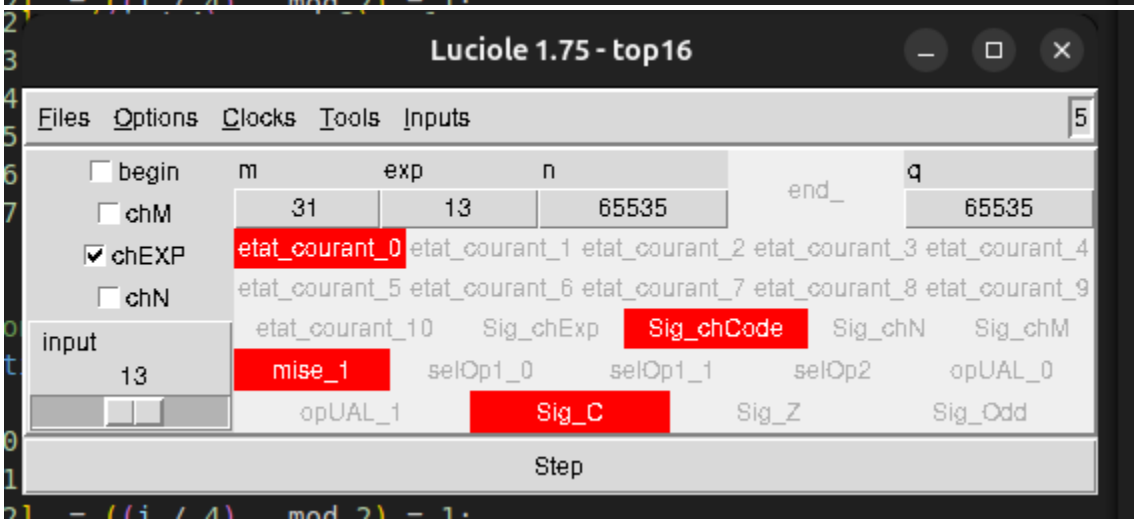


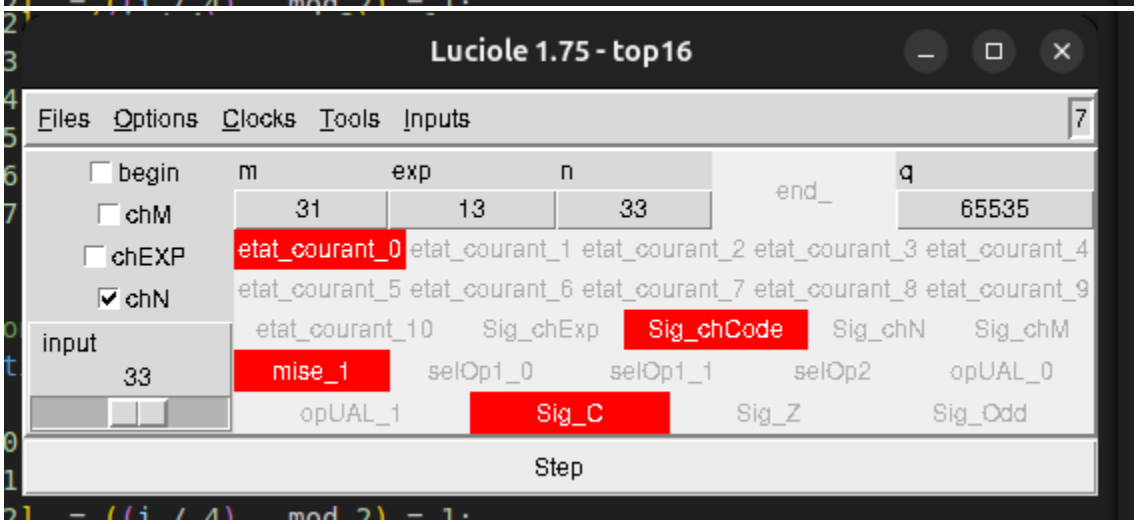


Arhan UNAY
Utku GEMICIOGLU

02.04.2025
INFO3 Groupe 2

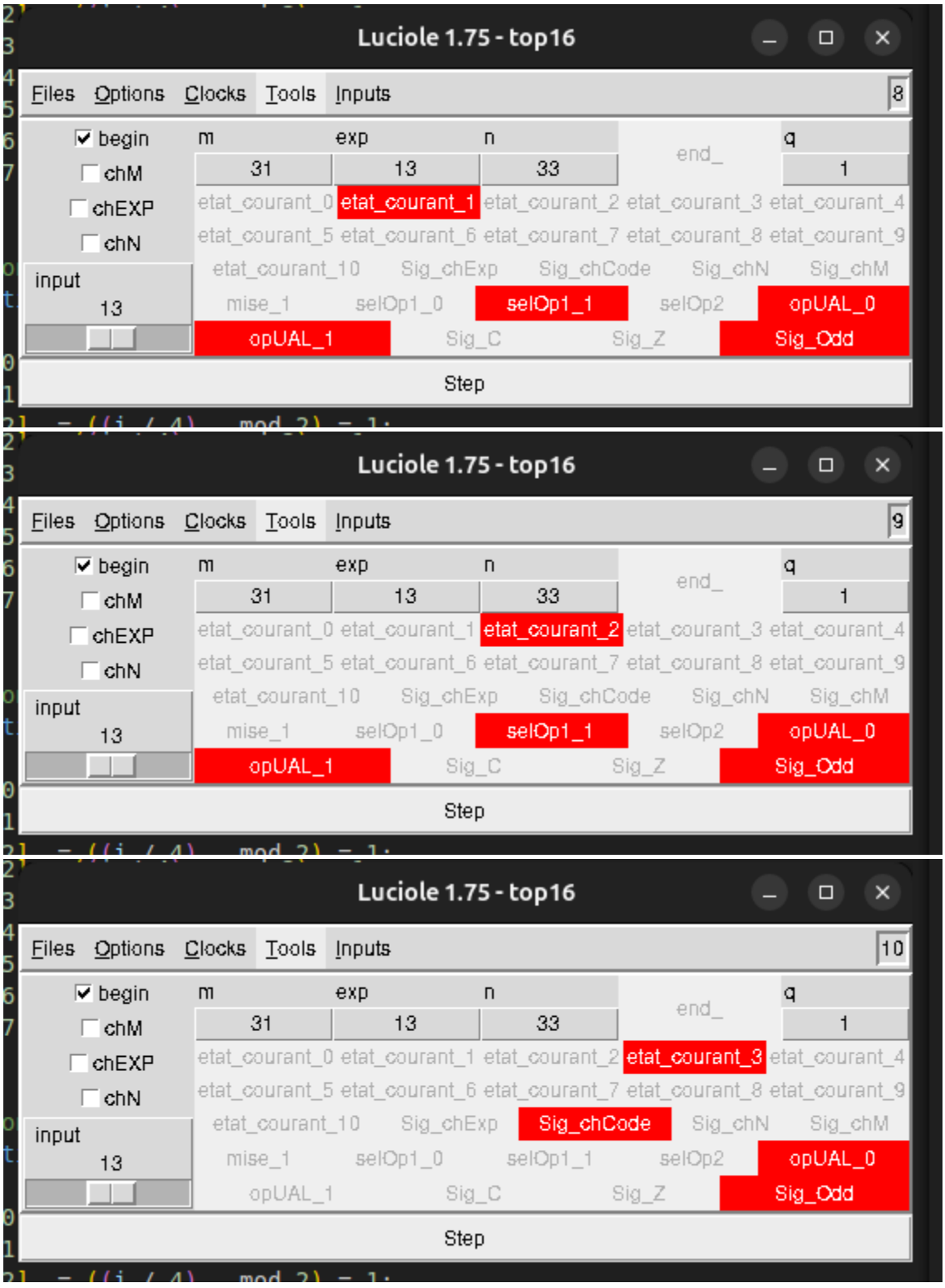






Arhan UNAY
Utku GEMICIOGLU

02.04.2025
INFO3 Groupe 2

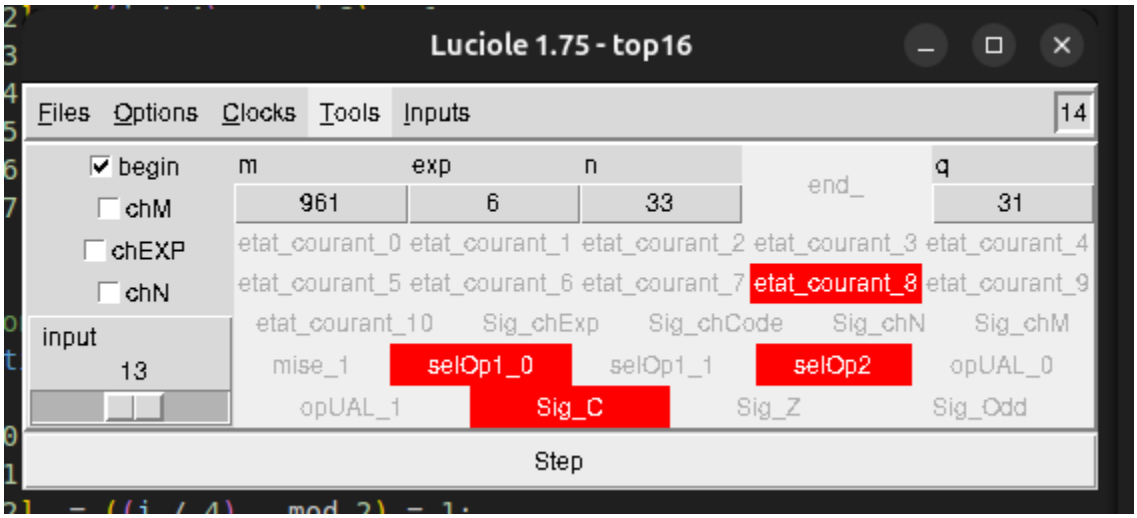


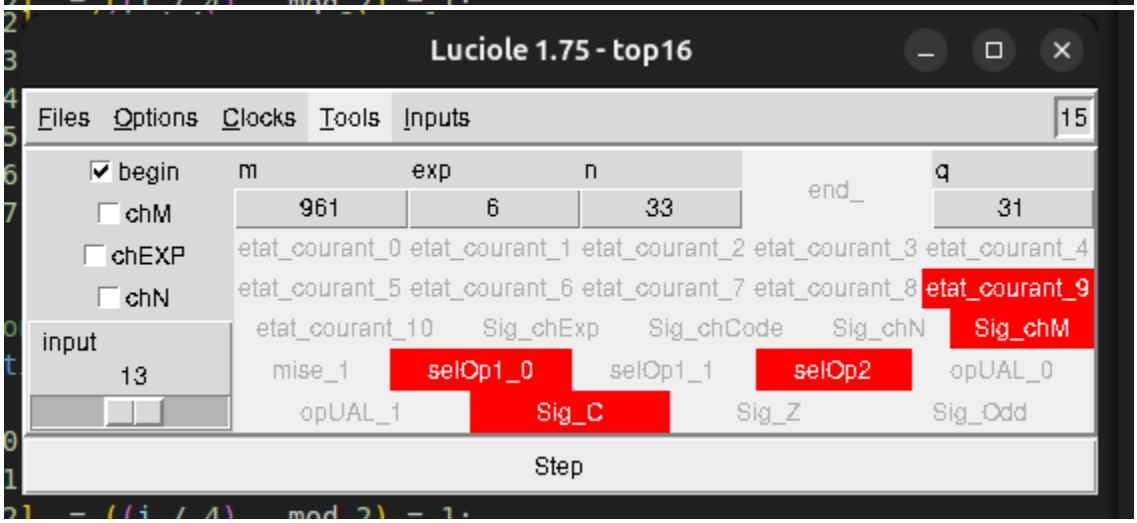
Arhan UNAY
Utku GEMICIOGLU

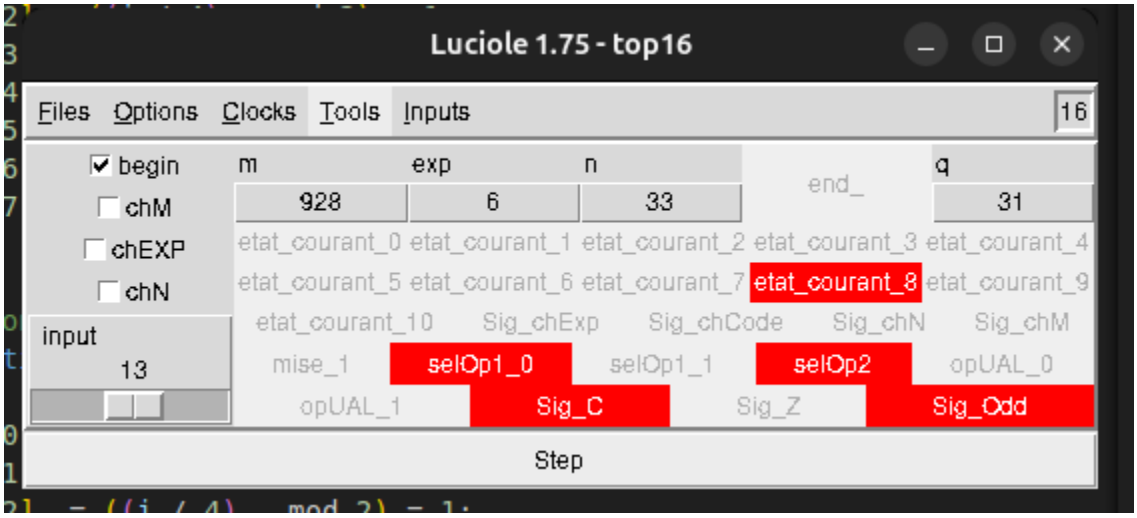
02.04.2025
INFO3 Groupe 2

Arhan UNAY
Utku GEMICIOGLU

02.04.2025
INFO3 Groupe 2







Arhan UNAY
Utku GEMICIOGLU

02.04.2025
INFO3 Groupe 2

Arhan UNAY
Utku GEMICIOGLU

02.04.2025
INFO3 Groupe 2

Arhan UNAY
Utku GEMICIOGLU

02.04.2025
INFO3 Groupe 2

Arhan UNAY
Utku GEMICIOGLU

02.04.2025
INFO3 Groupe 2

Arhan UNAY
Utku GEMICIOGLU

02.04.2025
INFO3 Groupe 2

Arhan UNAY
Utku GEMICIOGLU

02.04.2025
INFO3 Groupe 2

Arhan UNAY
Utku GEMICIOGLU

02.04.2025
INFO3 Groupe 2

Arhan UNAY
Utku GEMICIOGLU

02.04.2025
INFO3 Groupe 2

Arhan UNAY
Utku GEMICIOGLU

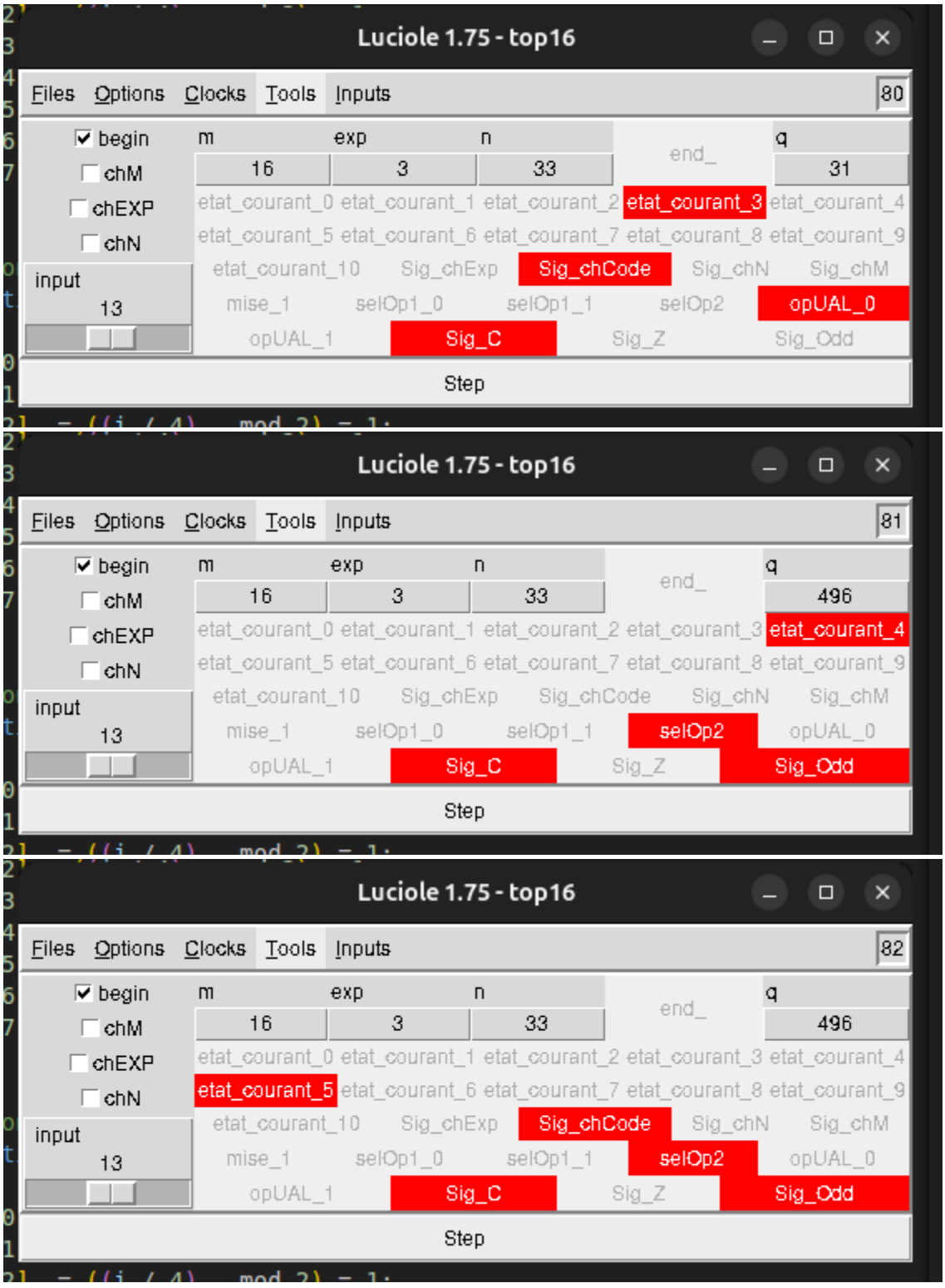
02.04.2025
INFO3 Groupe 2

Arhan UNAY
Utku GEMICIOGLU

02.04.2025
INFO3 Groupe 2

Arhan UNAY
Utku GEMICIOGLU

02.04.2025
INFO3 Groupe 2

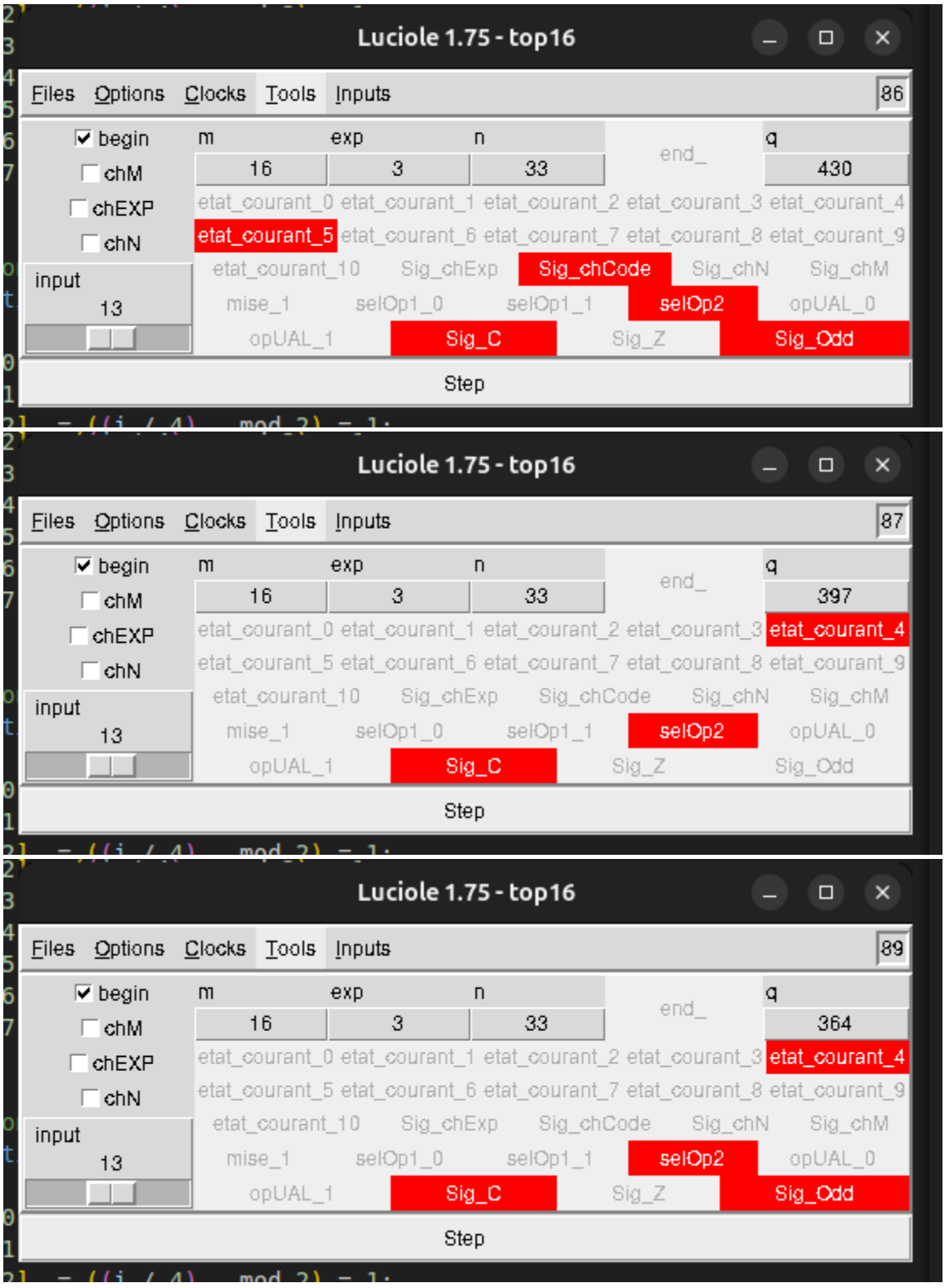


Arhan UNAY
Utku GEMICIOGLU

02.04.2025
INFO3 Groupe 2

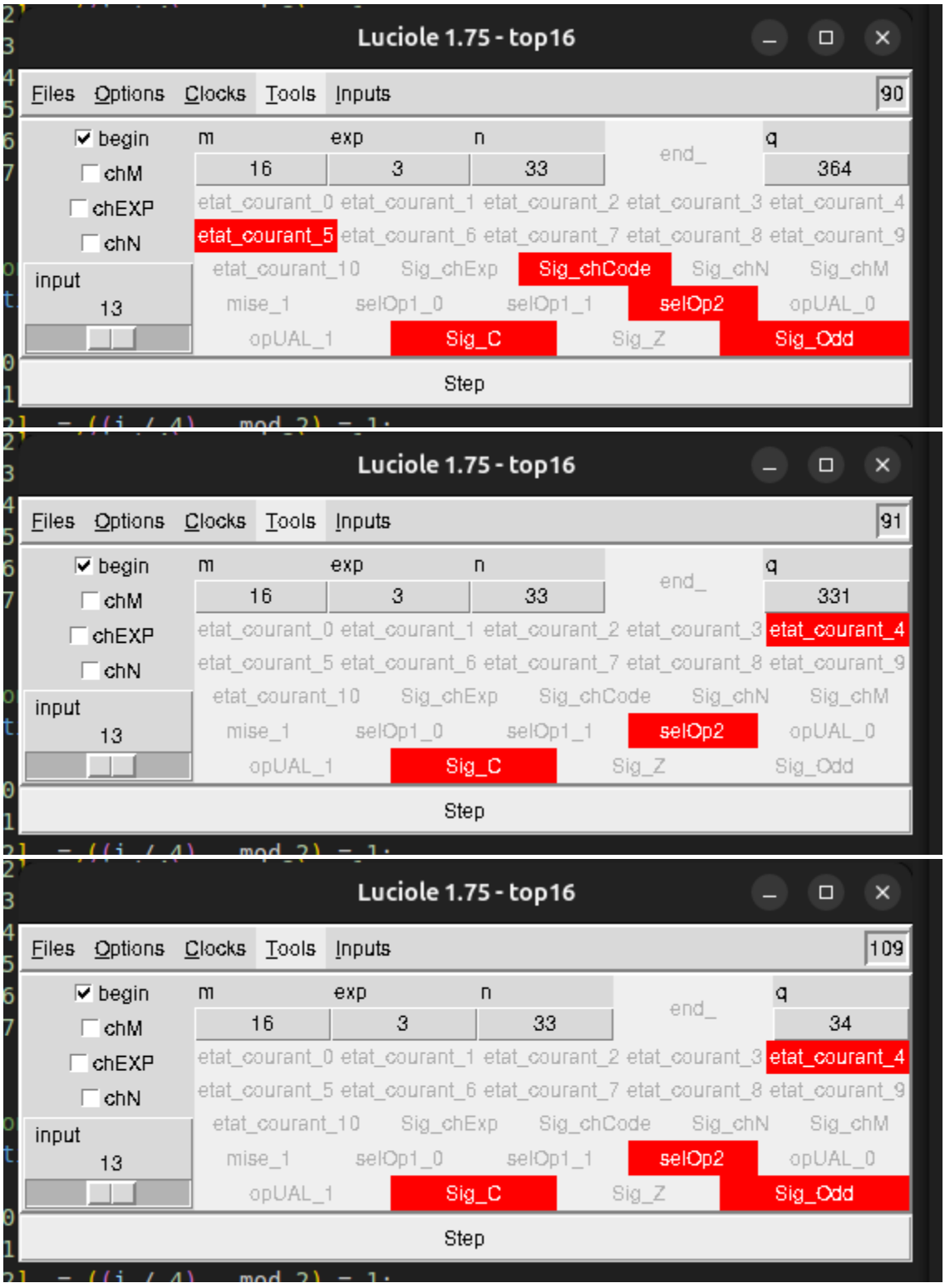
Arhan UNAY
Utku GEMICIOGLU

02.04.2025
INFO3 Groupe 2



Arhan UNAY
Utku GEMICIOGLU

02.04.2025
INFO3 Groupe 2



Arhan UNAY
Utku GEMICIOGLU

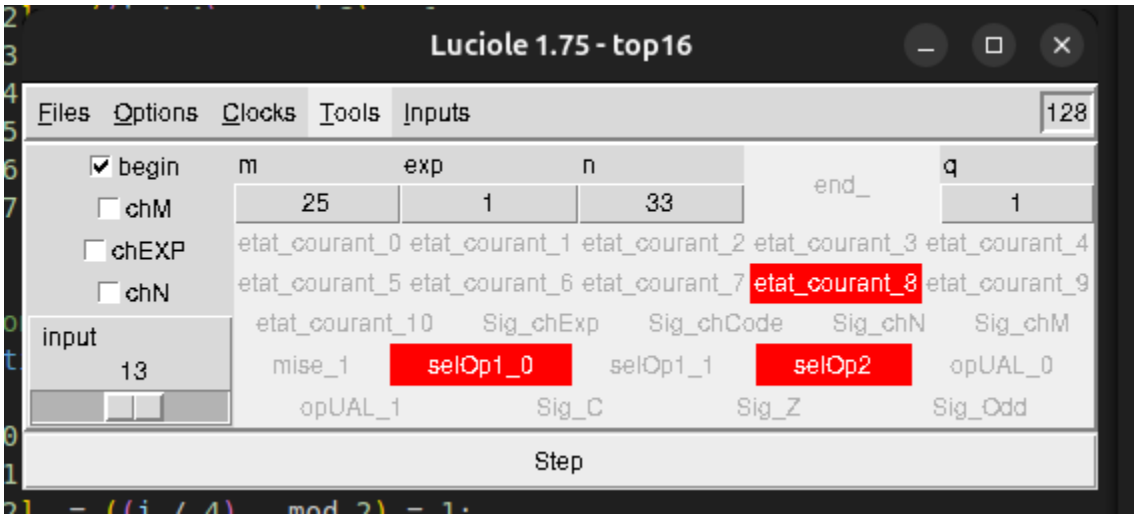
02.04.2025
INFO3 Groupe 2

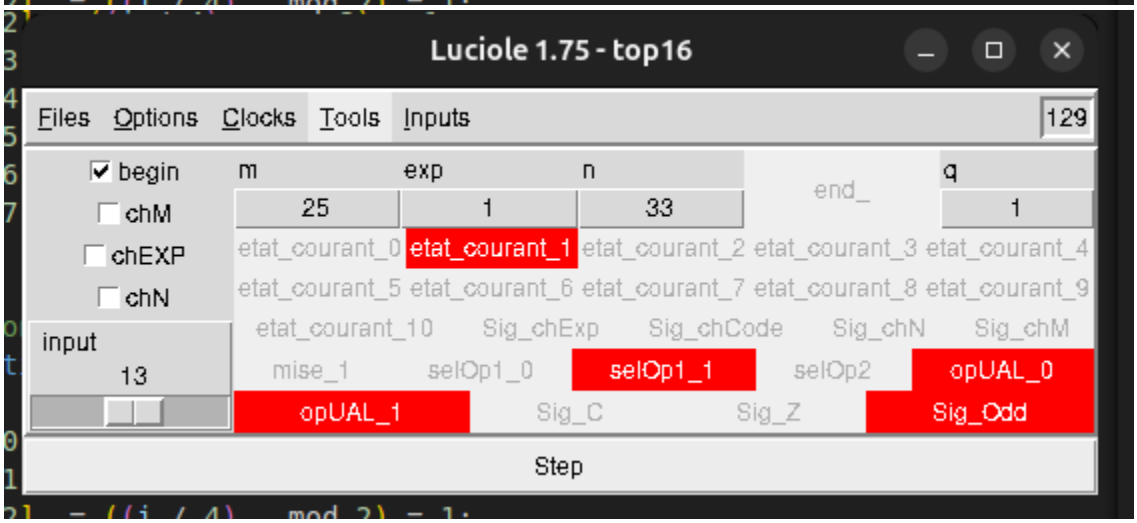
Arhan UNAY
Utku GEMICIOGLU

02.04.2025
INFO3 Groupe 2

Arhan UNAY
Utku GEMICIOGLU

02.04.2025
INFO3 Groupe 2





Arhan UNAY
Utku GEMICIOGLU

02.04.2025
INFO3 Groupe 2

Arhan UNAY
Utku GEMICIOGLU

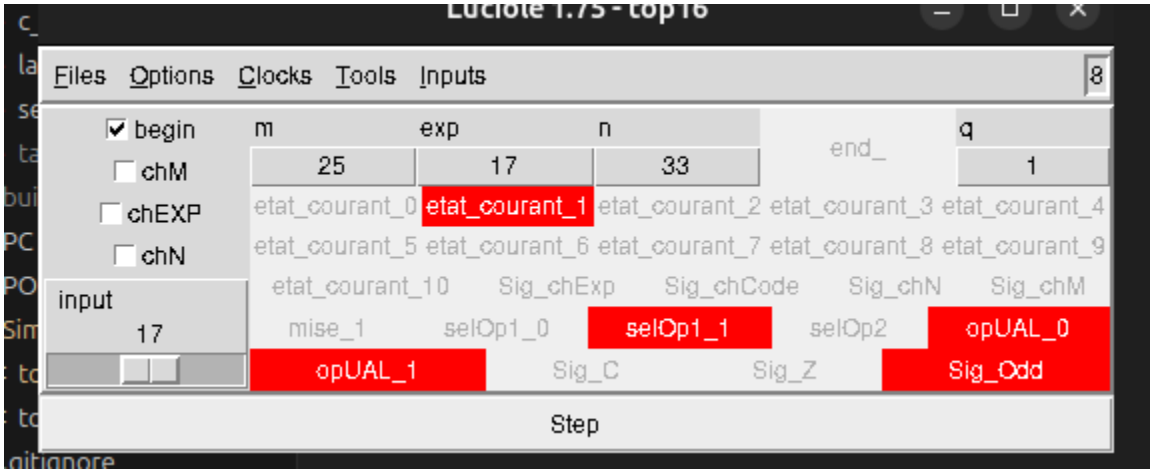
02.04.2025
INFO3 Groupe 2

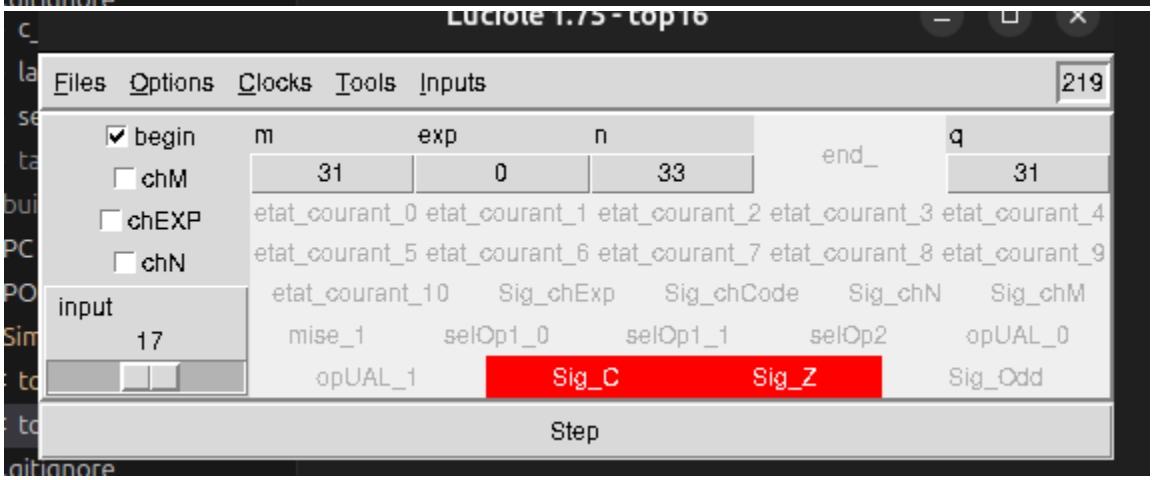
Pour les tests suivants, seuls les résultats et les états initiaux vous seront présentés.

Arhan UNAY
Utku GEMICIOGLU

02.04.2025
INFO3 Groupe 2

```
[~/Desktop/repertoire/ArchMat/Real_mach_alg]
base arhis arhan-master - ./chiffrement 31 13 33
Resultat chiffrement 25

[~/Desktop/repertoire/ArchMat/Real_mach_alg]
base arhis arhan-master - ./chiffrement 25 17 33
Resultat chiffrement 31
```

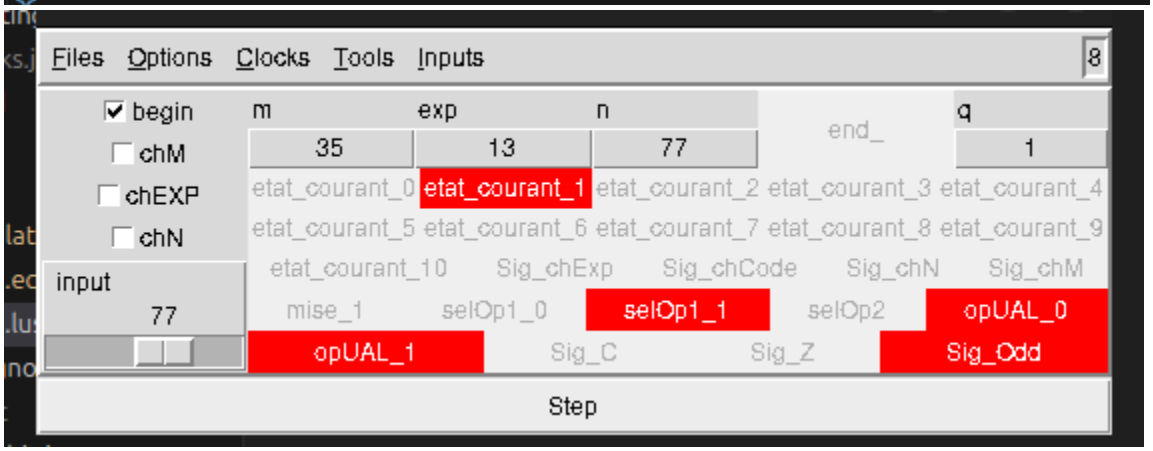


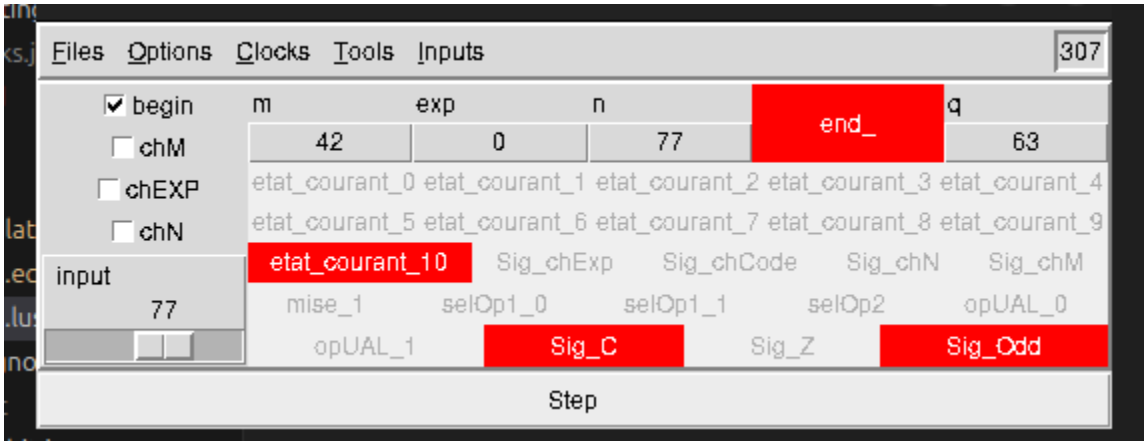


Arhan UNAY
Utku GEMICIOGLU

02.04.2025
INFO3 Groupe 2

```
[~/Desktop/repertoire/ArchMat/Real_mach_alg]
base arhis arhan-master - ./chiffrement 35 13 77
Resultat chiffrement 63

[~/Desktop/repertoire/ArchMat/Real_mach_alg]
base arhis arhan-master - ./chiffrement 63 37 77
Resultat chiffrement 35
```





Arhan UNAY
Utku GEMICIOGLU

02.04.2025
INFO3 Groupe 2