Understanding the Bull GAMMA 3 first generation computer through emulation

PONSARD Christophe
NAM-IP Computer Museum
CETIC Researcher, DW Champion

FOSDEM 23 – Emulation – February 5

Context – NAM-IP Computer Museum

Computer Museum NAM-IP

www.nam-ip.be

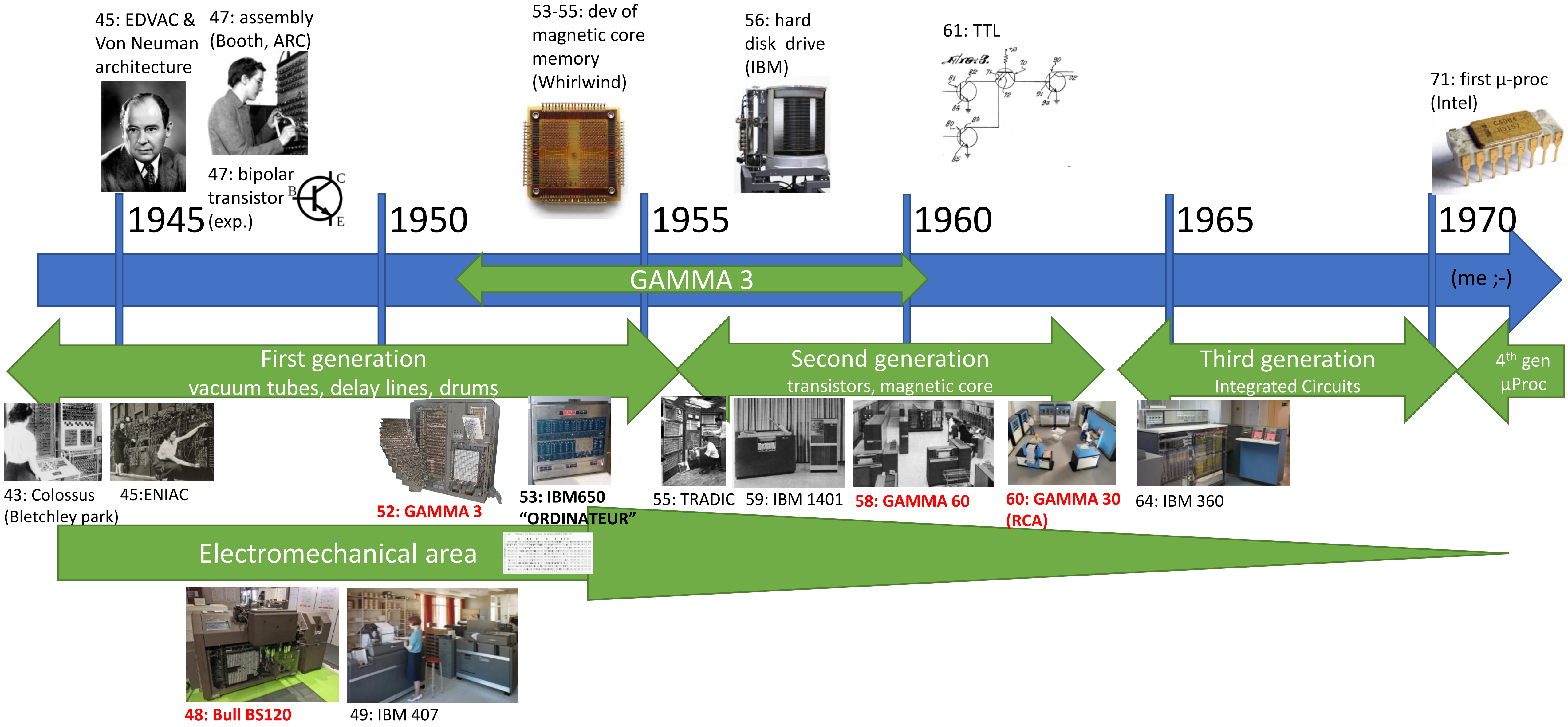
- Located in Namur/Belgium - 30’ from Brussels
  - worth a visit if you are staying a few days I Belgium after FOSDEM)
  - also: Pixel Museum (in BXL) & HomeComputerMuseum (in Eindhoven/NL not so far)
- Missions:
  - Preservation: safeguarding digital heritage, focus on local pioneers
  - Acquisition of artefacts, enriching collections: Bull, Burroughs/Unysis, I&B,…
  - Exhibitions: for all, specific animation, permanent/temporary
  - **Research: about machines, software, communities ➔ here BULL GAMMA 3**
- “Container design”, an historical parallel

Outlook

- Historical context : a long time ago…
- Discovering the machine
- A look at existing emulators
- Our JAVA emulator
- Some lessons learned

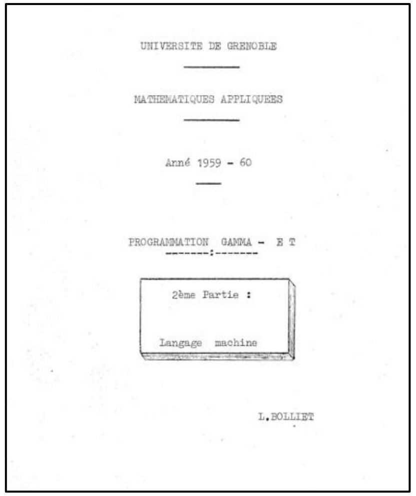
Back to Early Electronic Machines

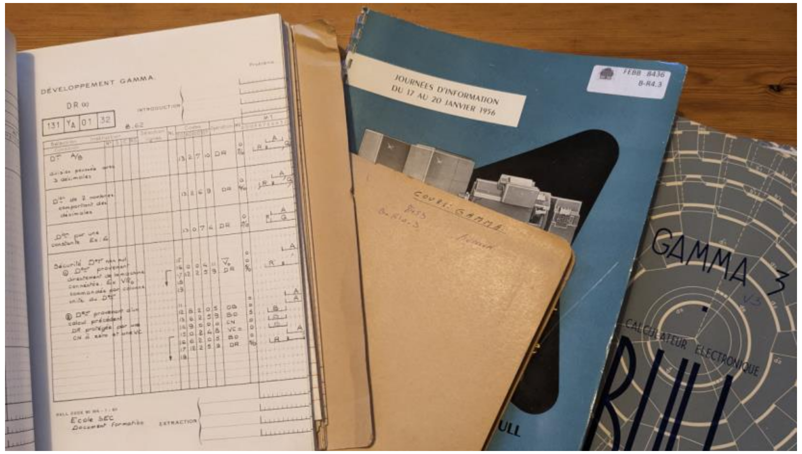
with transition from electromechanical ➔ electronic machines



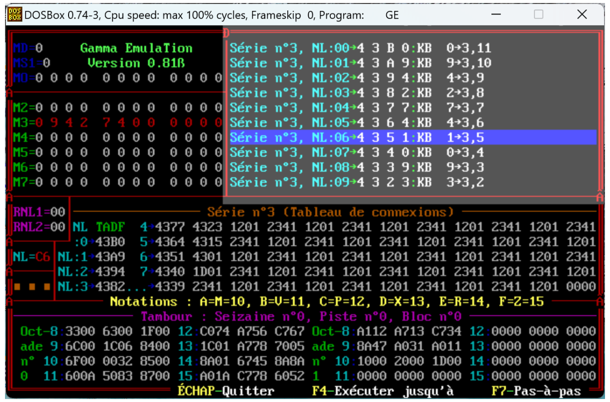
Inside the machine: sources ?

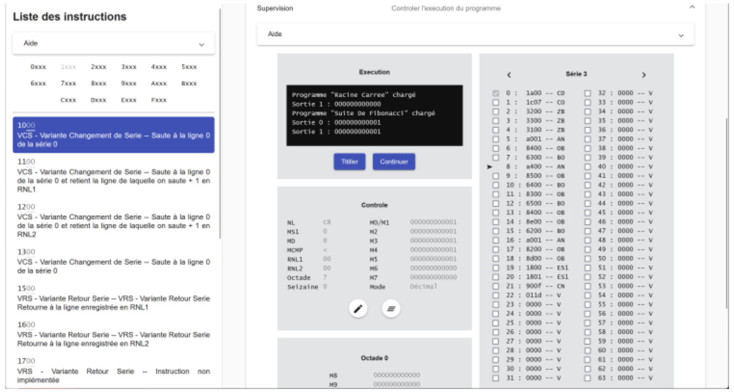
- Only a few left (Anger, Grenoble, Frankfurt, some pieces in Namur)
- Documentation: ACONIS/Bull Belgium



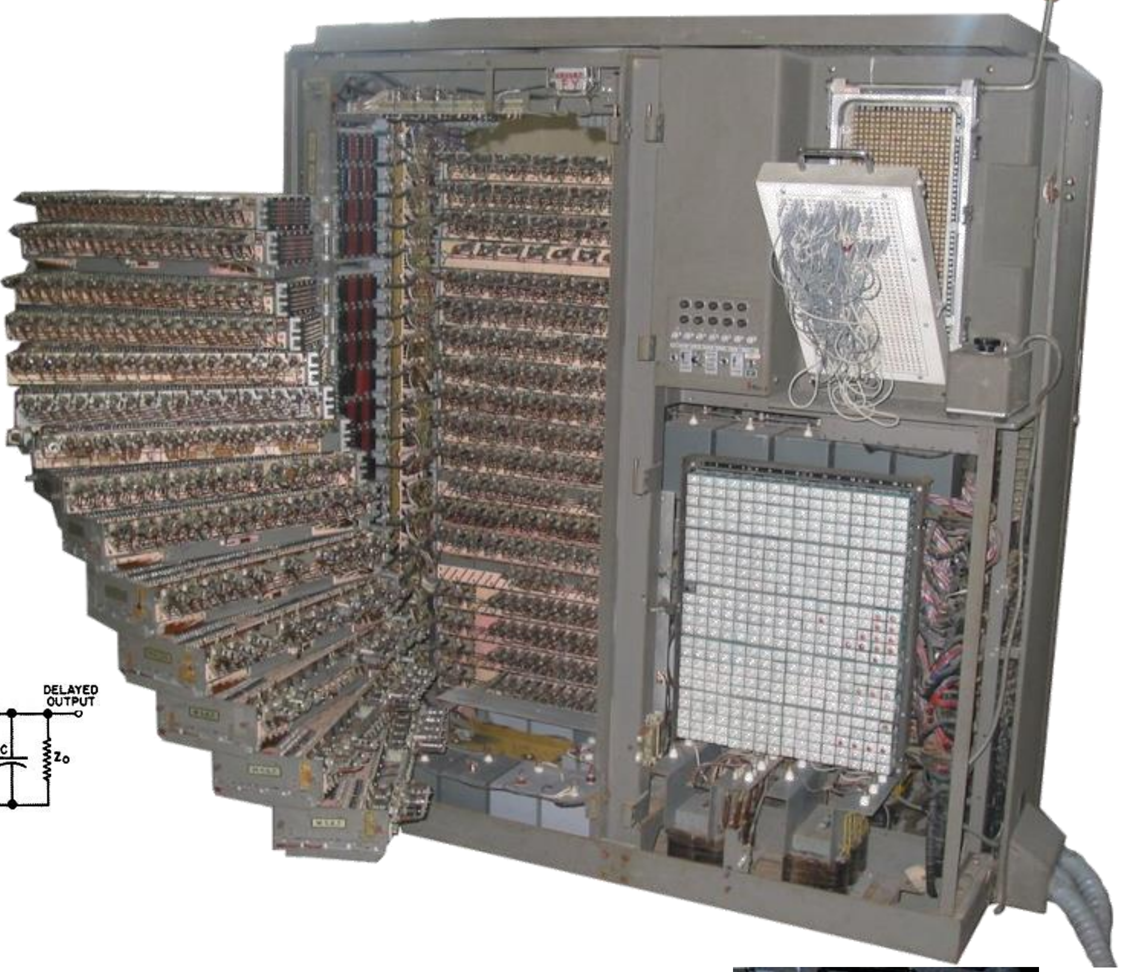


- Emulators !





Hardware (initial version)



First generation computer:

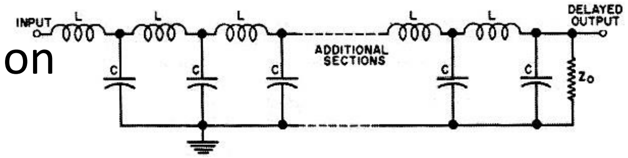
- vacum tubes, delay lines

Code stored in a connection panel (64 instructions)

- So not Von Neumann (pgm need to be in memory)
- although somehow “memory mapped” on “serie 3”

Main memory: 7 registers

- Delay lines
- Vacuum tube for regeneration
- 8.5 kg to store… 6 bytes

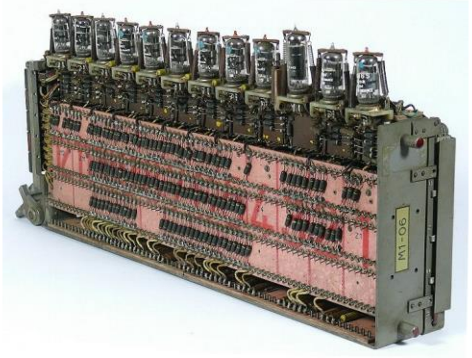


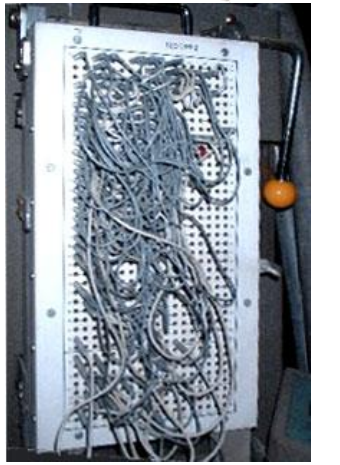
ALU:

- vacuum tubes (low reliability) ➔ germanium diodes
- + - ➔ * % performed internally by iterative + -

Clock: 2.5 Hz… because synchronised with motor
of motor of the connected electromechanical equipment !

Nice “drawer” design easing maintenance !





GAMMA 3 : Computer and/or Calculator ?

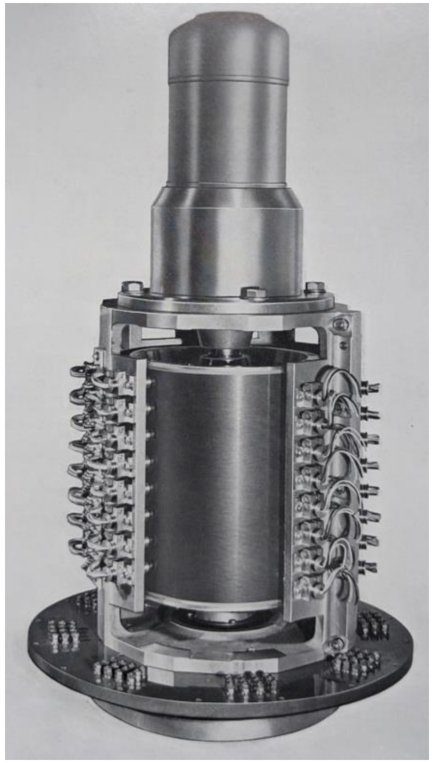
- Initially: designed as calculator « slave » for tabulating machine

  *"In its current configuration, **the computer acts as an extension unit for the punched card machine it is connected to**. The cards are read in the reader station which transmits data to the computer. The computer in turn performs all necessary calculations and transmits the results back to the punched card machine which will print or punch these values. Regardless of the task, **the computer is so fast that there is no visible delay** caused by the calculations"*
  (Pierre Letort, 1953)

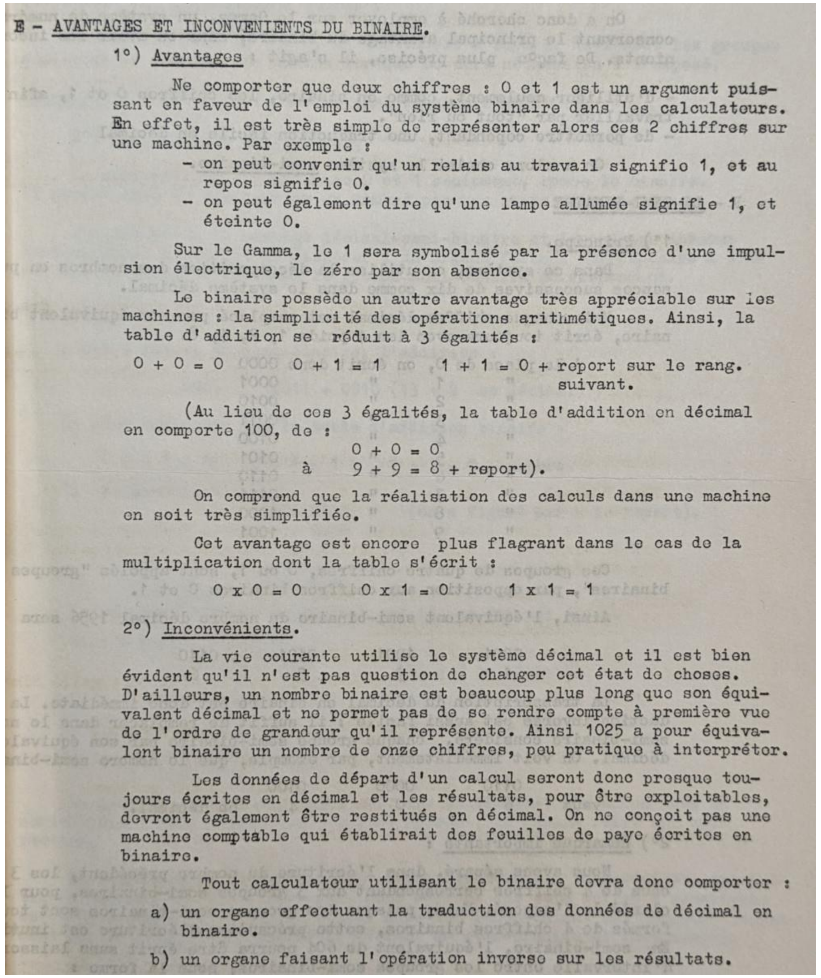
  So programs = specific calculation routines called by tabulating machine

- Evolution:
  - 3M: scientific version with floating point support
    (otherwise need subroutines)
  - ET (1957): drum extension (about 100Kb) ➔ program NOW truly in memory

- Finally: GAMMA 3 : first French computer
  ➔ GAMMA 3 becomes the central unit
  and tabulating machine becomes the peripheral device (= transition !)



Should a Computer Operate in Decimal or Binary ?



E - AVANTAGES ET INCONVENIENTS DU BINAIRE.

1°) Avantages

Ne comporter que deux chiffres : 0 et 1 est un argument puissant en faveur de l'emploi du système binaire dans les calculateurs. En effet, il est très simple de représenter alors ces 2 chiffres sur une machine. Par exemple :

- on peut convenir qu'un relais au travail signifie 1, et au repos signifie 0.
- on peut également dire qu'une lampe allumée signifie 1, et éteinte 0.

Sur le Gamma, le 1 sera symbolisé par la présence d'une impulsion électrique, le zéro par son absence.

Le binaire possède un autre avantage très appréciable sur les machines : la simplicité des opérations arithmétiques. Ainsi, la table d'addition se réduit à 3 égalités :

0 + 0 = 0   0 + 1 = 1   1 + 1 = 0 + report sur le rang suivant.

(Au lieu de ces 3 égalités, la table d'addition en décimal en comporte 100, de :

0 + 0 = 0
à 9 + 9 = 8 + report).

On comprend que la réalisation des calculs dans une machine en soit très simplifiée.

Cet avantage est encore plus flagrant dans le cas de la multiplication dont la table s'écrit :

0 x 0 = 0   0 x 1 = 0   1 x 1 = 1

2°) Inconvénients.

La vie courante utilise le système décimal et il est bien évident qu'il n'est pas question de changer cet état de choses. D'ailleurs, un nombre binaire est beaucoup plus long que son équivalent décimal et ne permet pas de se rendre compte à première vue de l'ordre de grandeur qu'il représente. Ainsi 1025 a pour équivalent binaire un nombre de onze chiffres, peu pratique à interpréter.

Les données de départ d'un calcul seront donc presque toujours écrites en décimal et les résultats, pour être exploitables, devront également être restitués en décimal. On ne conçoit pas une machine comptable qui établirait des feuilles de paye écrites en binaire.

Tout calculateur utilisant le binaire devra donc comporter :

a) un organe effectuant la traduction des données de décimal en binaire.

b) un organe faisant l'opération inverse sur les résultats.

Pro:

- Only two figure: powerful
- Maps easily to relays !

Cons:

- Longuer
- Needs to translate back & forth with decimal (for humans

Conclusion ?

- Use « semi-decimal » ➔ binary-coded decimal !
- Char = 4 bit to code 1 BCD (or 1 HEX)

Note: later extension (ET) supported full binary mode

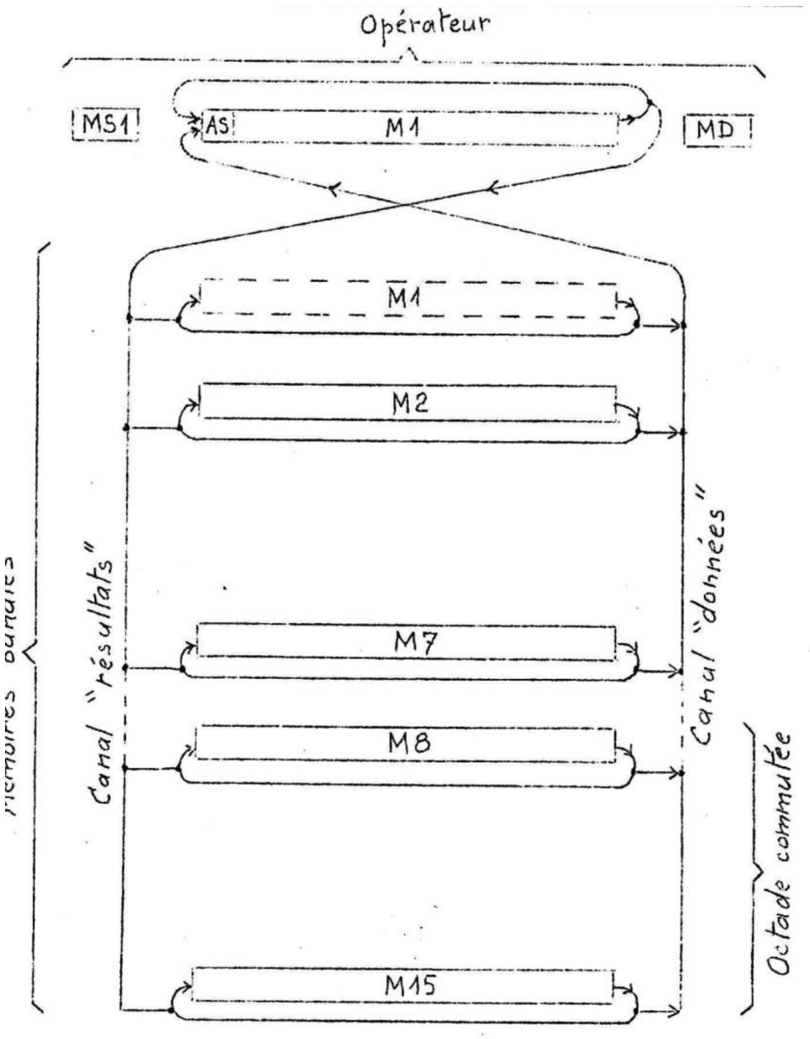
1958 GAMMA3 course

GAMMA 3 – Core “banale” memory : a few registers

- M0=M1: “accumulator” IN/OUT
- M2..M7: only IN
- M8..M15: “extra memory”
  (switched “octade” see extension)
- Technology: delay line
  - 1 register = 1 word
  - 1 word of 12 chars
    of 4 bits (BCD or HEX)

  ➔1 word = 6 bytes (note: longer than 32bit integer)

  ➔M1..M7 = 42 bytes of “main” memory

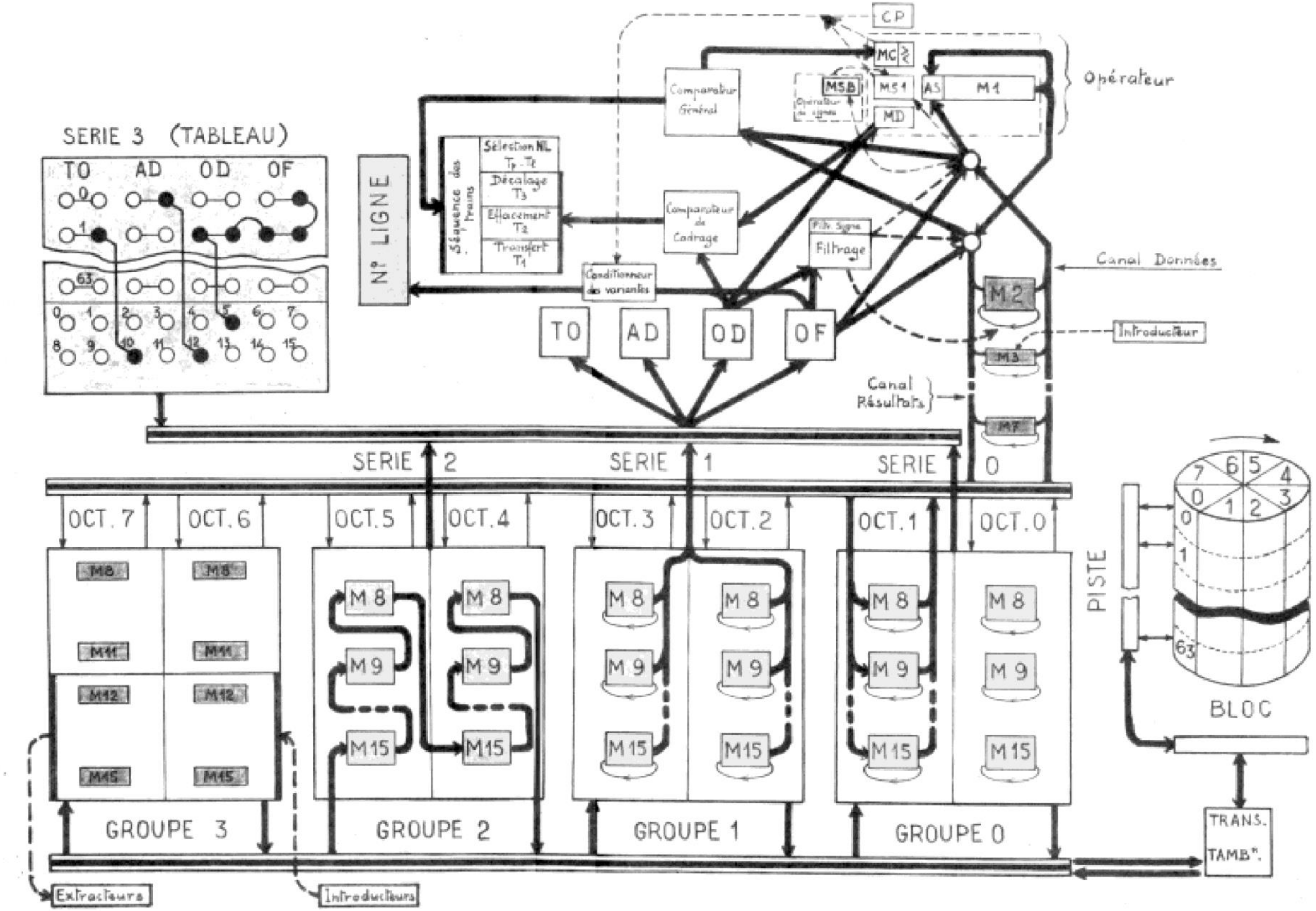


GAMMA 3 Complete Architecture

Note also:

PC, CMP, shift…

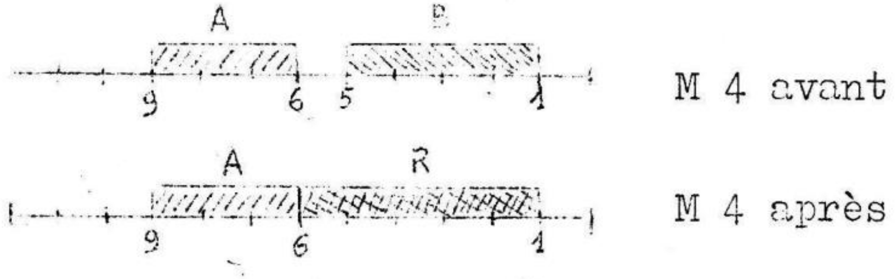
Used in instructions

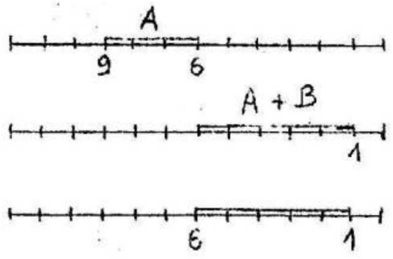


GAMMA 3 – Structure of Instruction Set

Structure: 4 chars

- TO: Type of Operation
  - E.g. AN (add), SN (subtract), MR (reduced mult.), MC (complete mult.),...
- AD: ADdress
  - Memory register involved in operation
- OD: Ordre Début (i.e. start position)
- OF: Ordre Fin (i.e. end position)
  - Because operation can consider a subrange of a Word !

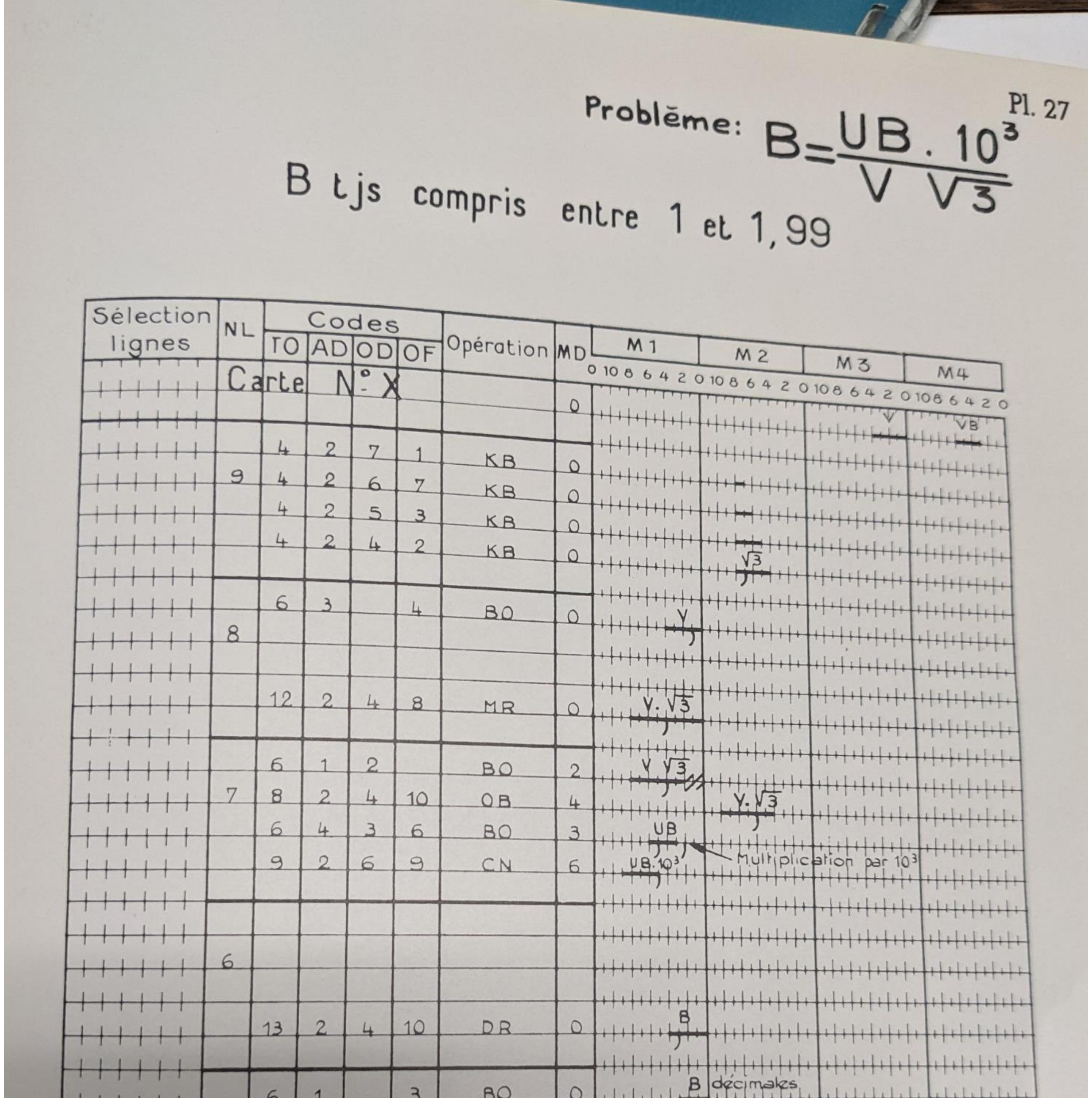




TO	AD	OD	OF		MD
6	4	6	9	BO	6
10	4	1	5	AN	1
8	4	1	6	OB	1

1 Word = 12 char
➔ so possible to store 3 instructions/word

Coding sheets


Pl. 27

Problème: $B = \frac{UB}{V} \cdot \frac{10^3}{\sqrt{3}}$

B tjs compris entre 1 et 1,99

Sélection lignes	NL	Codes TO	AD	OD	OF	Opération	MD	M1	M2	M3	M4
	Carte N° X						0			V	VB
		4	2	7	1	KB	0				
	9	4	2	6	7	KB	0				
		4	2	5	3	KB	0				
		4	2	4	2	KB	0		√3		
	8	6	3		4	BO	0	V			
		12	2	4	8	MR	0	V.√3			
		6	1	2		BO	2	V √3			
	7	8	2	4	10	OB	4		V.√3		
		6	4	3	6	BO	3	UB			
		9	2	6	9	CN	6	UB.10^3 ← Multiplication par 10^3			
	6										
		13	2	4	10	DR	0	B			
		6	1		3	BO	0	B décimales			

Exploring the Instruction Set (excerpt)

0XXX – V variant
- 0000 NOP ;-)
- 01xx JUMP if > flag set
- 02xx JUMP if = flag set
- 03xx JUMP if >= flag set
- 04xx JUMP if neg flag set

1XXX
- 11/12/13xx: VCS: serie switch
- 15/16/17xx: VRS: serie return
- 18/19xx: out to connected machine
- 1Axx: decimal mode (CD)
- 1Fxx: binary mode (CB)
- …

2XXX – drum transfer (BT)
…

3XXX – setting memory to zero

4XXX – constant transfer to memory
- AD: mem
- OD: position
- OF: value

5XX – copy between octads

6XX – transfer to operative mem. (BO)
- AD: mem
- OD & OF: range

7XX
- 71/72xx: shift memory
- 73/74xx: logical AND (what about OR ?)

8XX – transfer from operative mem. (OB)
- AD: mem
- OD & OF: range

9XXX – comparison operations
…

AXXX – addition (AN)

BXXX – subtraction (SN)

CXXX – reduced multiplication (RM)

DXXX – reduced division (RD)

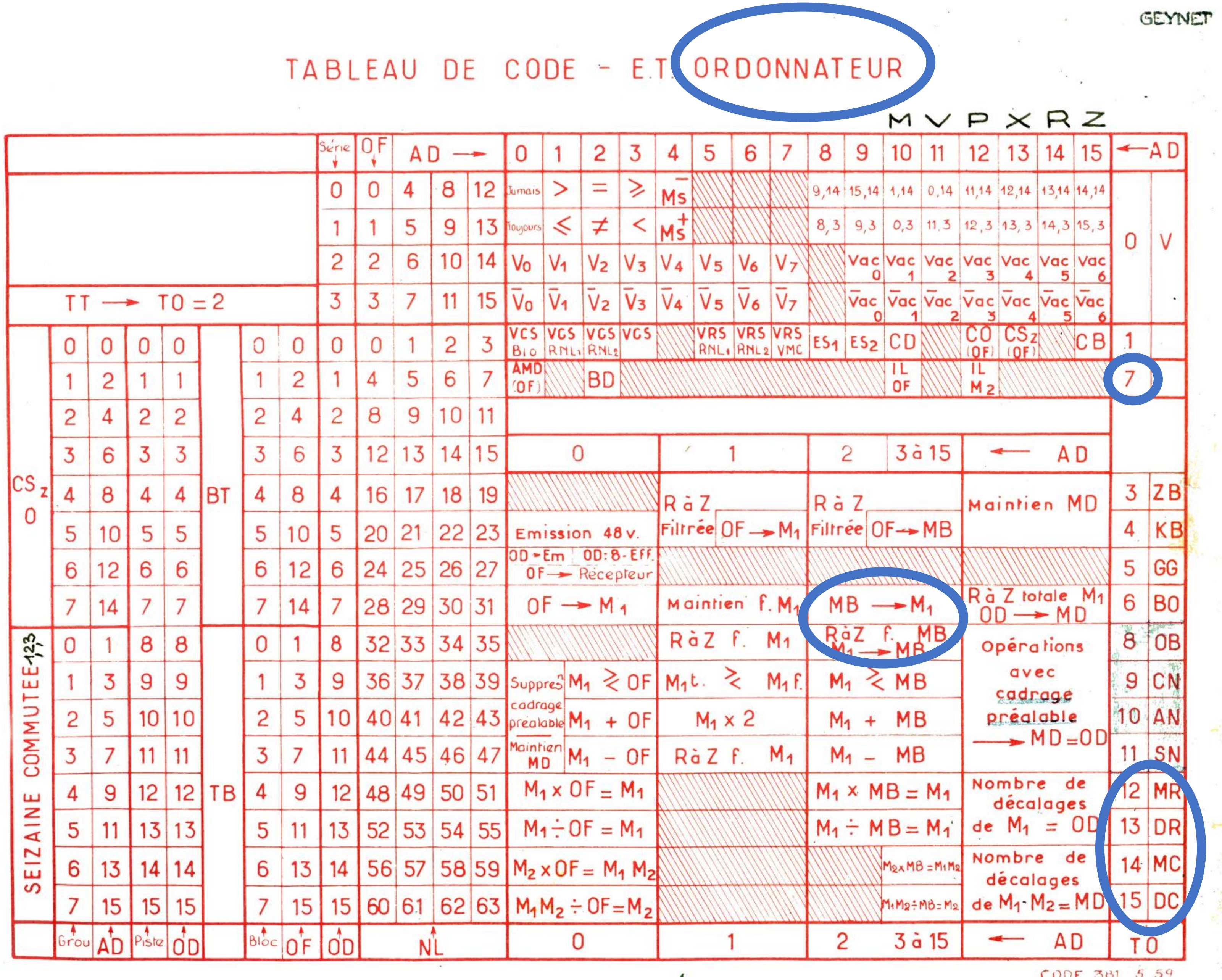
EXXX – full multiplication (MC)

FXXX – full division (FD)

Note:
- “reduced” * / for “small numbers” ➔range need to be manager
- “complete” * / operation using M1+M2 as extended storage

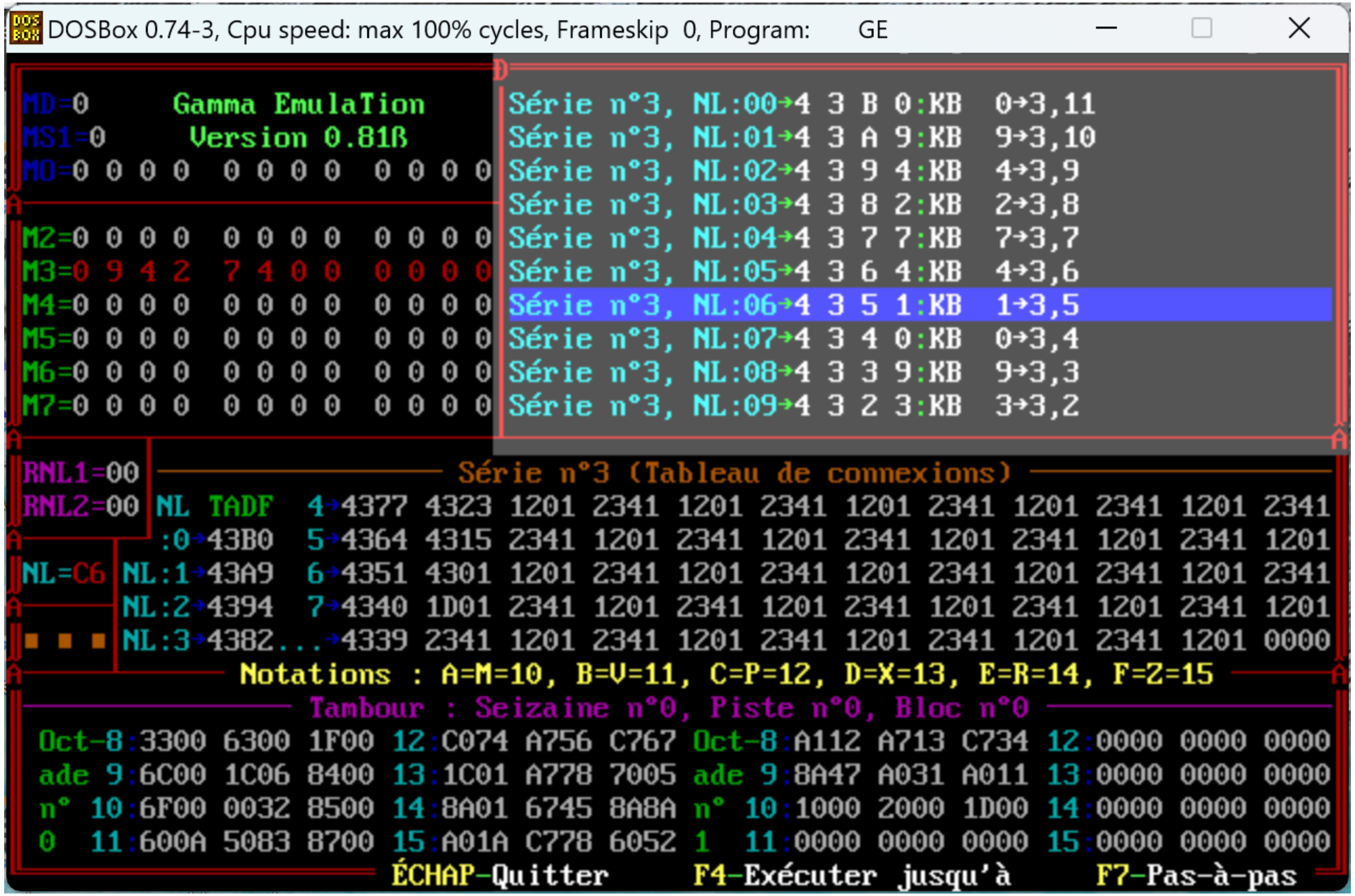
A look at the code card

- “ordonnateur” “ordinateur” in French was coined I year later by J. Perret for IBM “god puts world in order”
- Not so easy to decode…



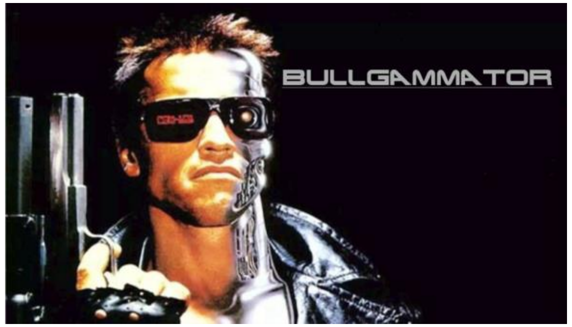
Emulator by V. Joguin (1990’s)

- Written in x86 assembly code by Vincent Joguin (1995)
- Now need an emulator (thanks DOSBox !)

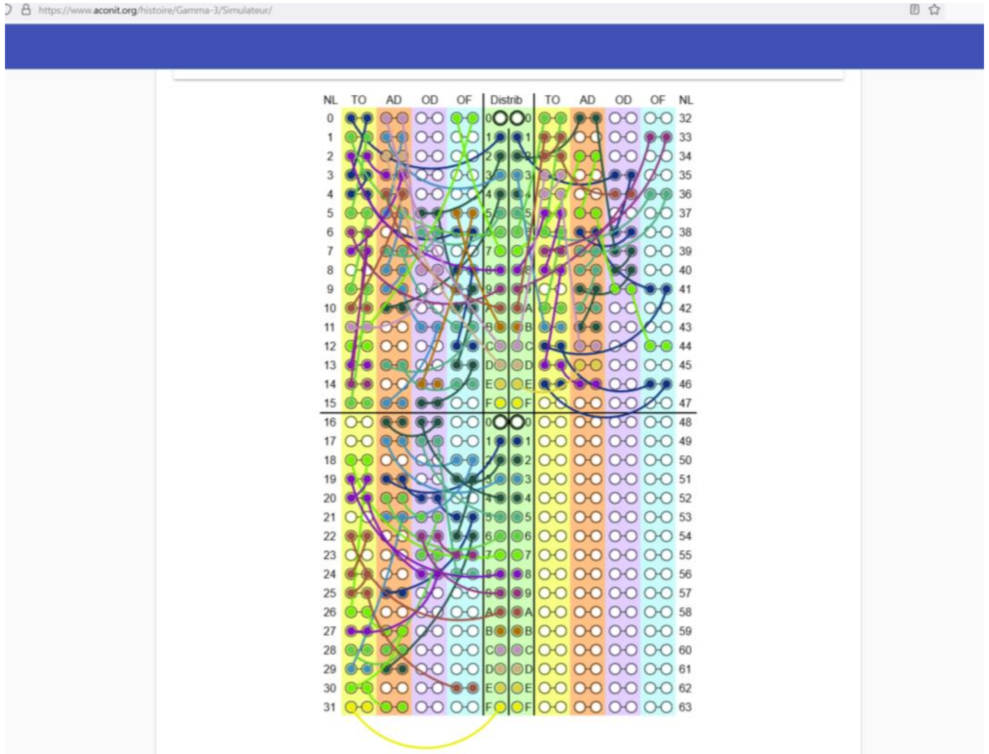


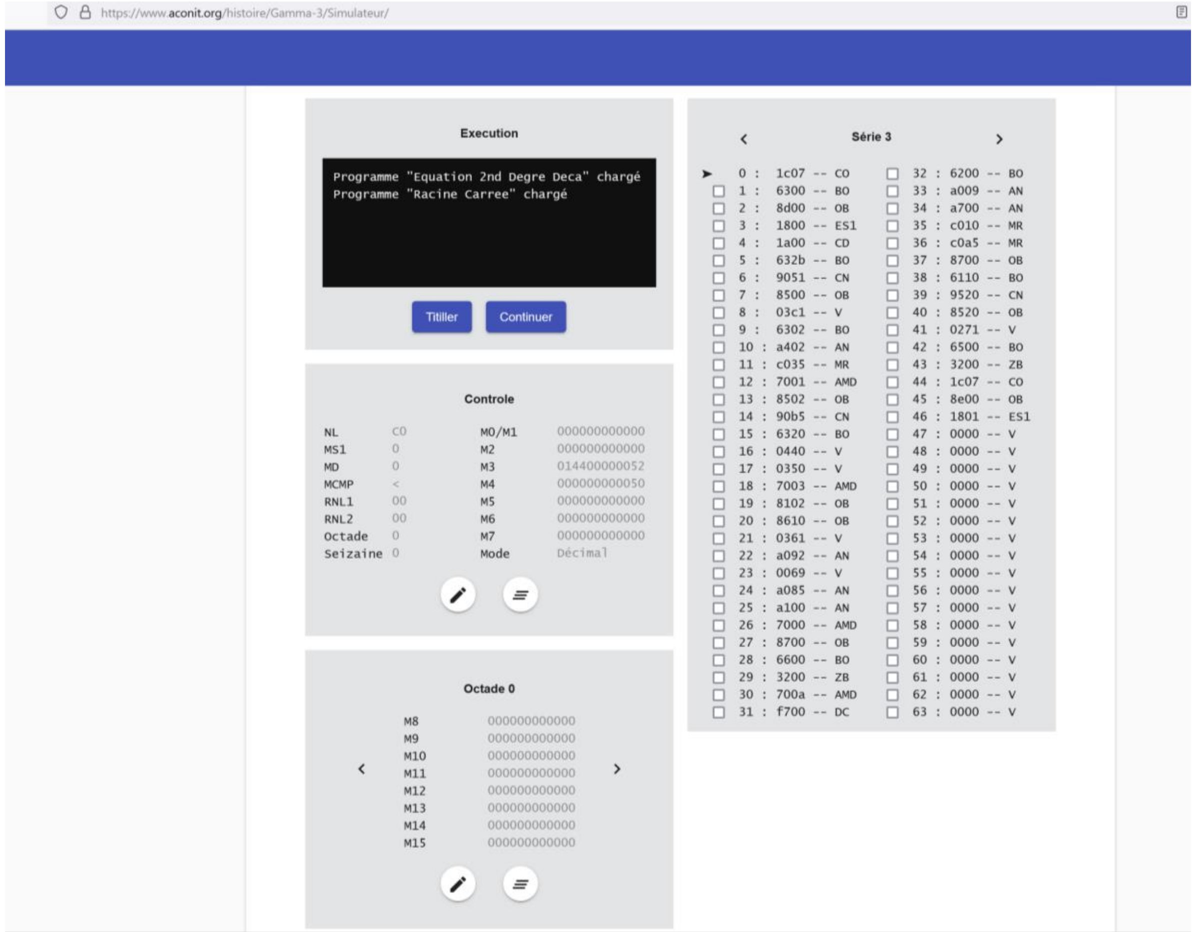
http://vincent.joguin.com/GAMMAET.ZIP

BullGammator (ACONIT, 2020)

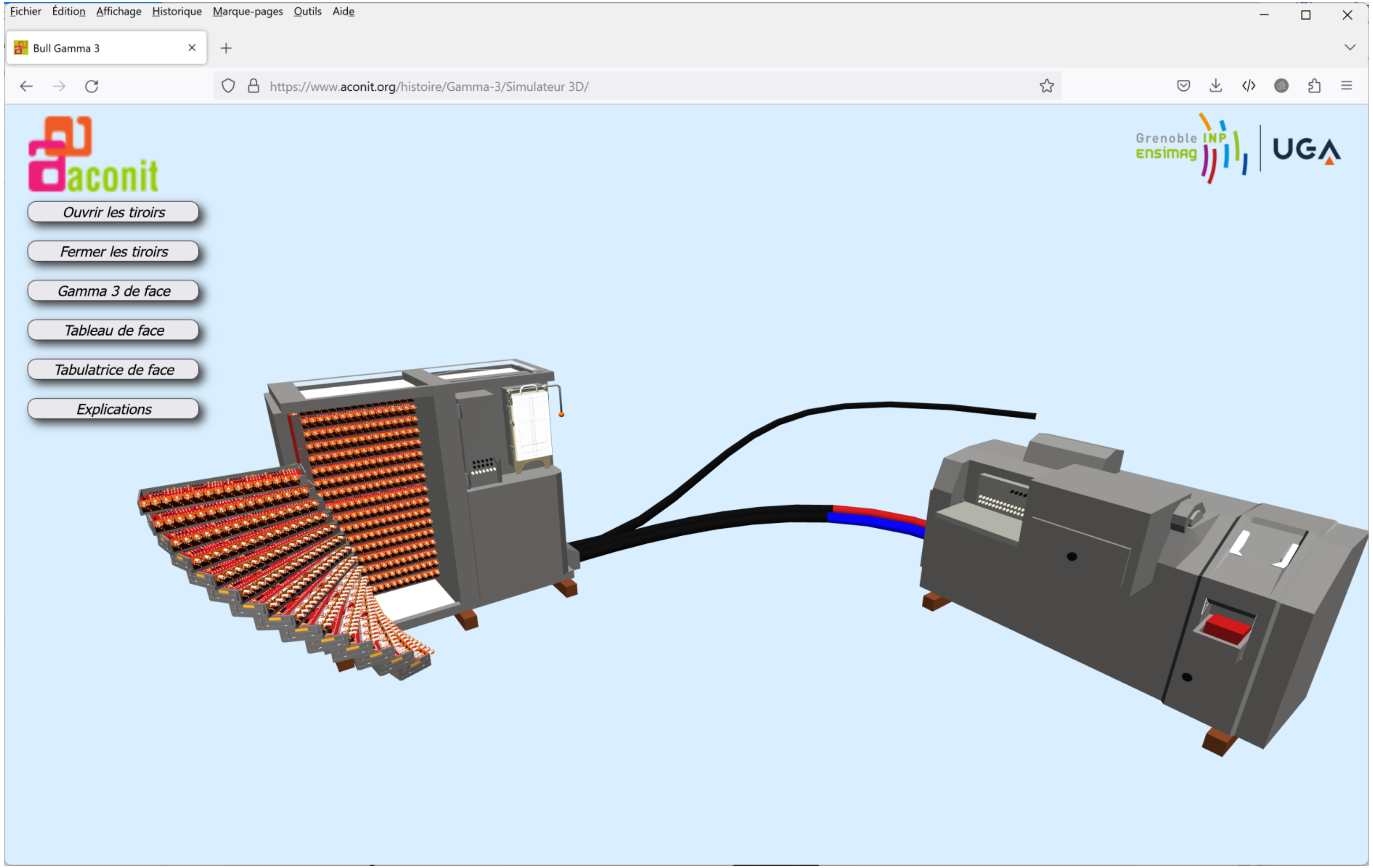


- Open Source : https://github.com/lutrampal/bullgammator
- Web-base written in Javascript (nodeJS), very well documented
- Include panel interface, library, console (with stepping mode “titiller”)
- Nice code base for emulator core with unit tests
  ➔ analysed and mainly transposed in JAVA



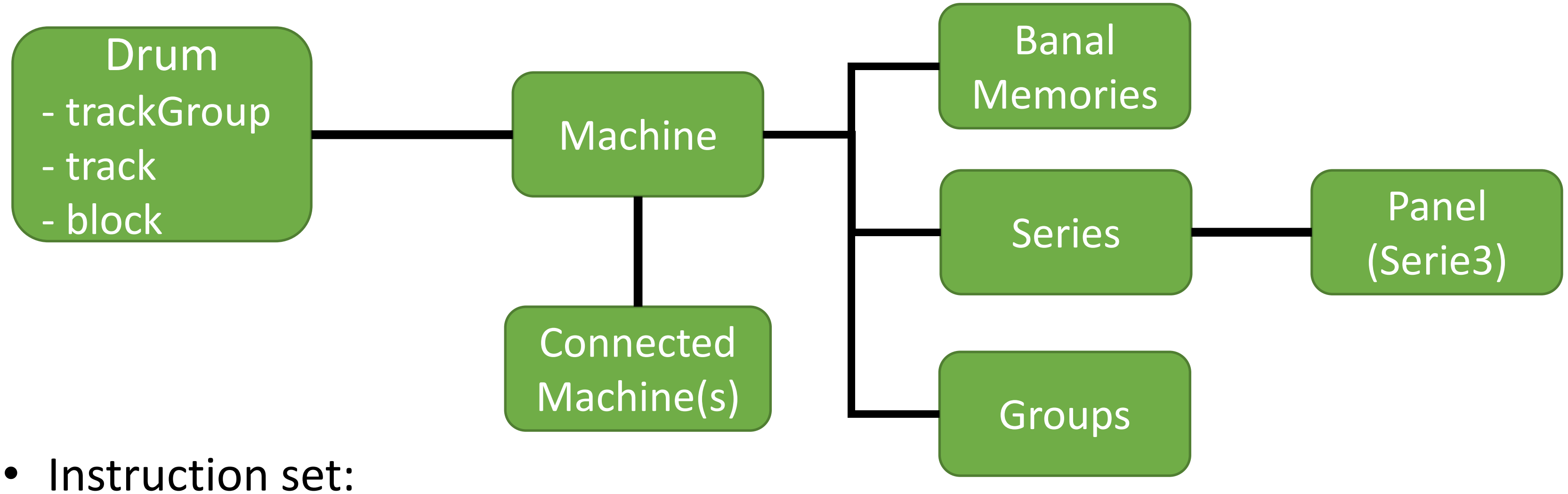


3D Simulator (by Aconit)

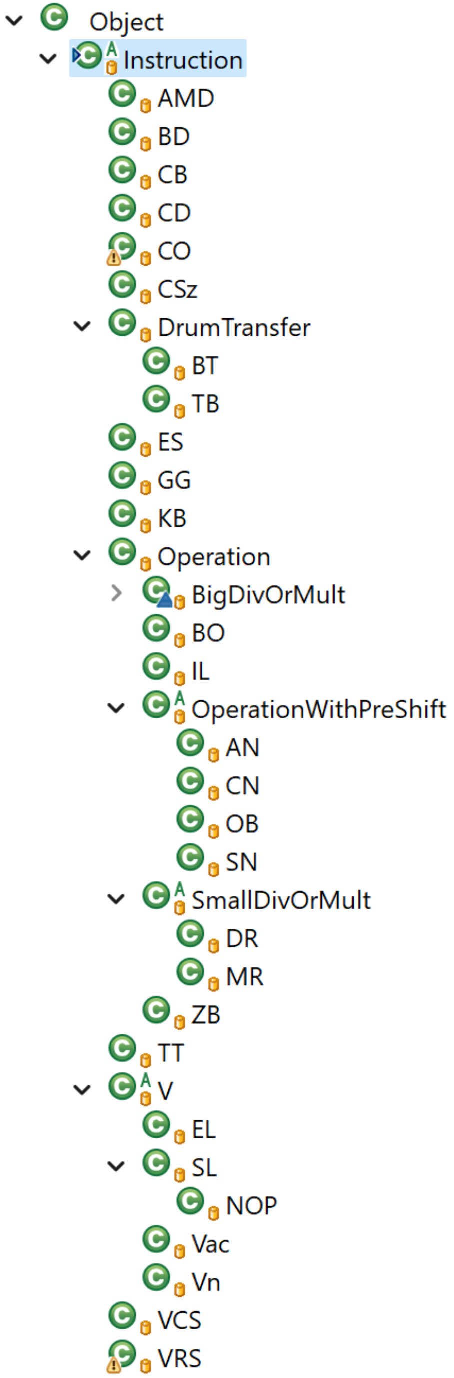


Java(script) Emulator Structure

- Machine description (see structure)
  - BullGamma ➔ memory, series, drum, connected machines,…



- Instruction set:
  - Execute method taking instruction and machine (context) as parameter
  - By type, using a object hierarchy
- Execution management, UI (basic)
- Tests (w.r.t. spec)
- Code: https://github.com/NAMIP-Computer-Museum/gamma3



A look at the addition – carry algo (dec/bin fine)

```
/**
add the given memory to this one - this works both in decimal and binary mode
@param other the memory that should be added
@param from index of the block from which the addition should start
@param to index of the block to which the addition should end (excluded)
@param overriding_carry if true, at the end of the addition, the resulting carry out will override the next
memory block if it is not null. Otherwise it will be added to the next memory block.
   */
public void add(Memory other, byte from, byte to, boolean overriding_carry) {
         assert (from >= 0) : "from should not be negative";
         assert (from < to):  "from should be inferior to "+ to;
         assert (to <= this.blocks.length) : "to should be inferior to the number of blocks per memory";
         int carry = 0; // implementation simulates the real work across memories
         for (int i = from; (i < to) || ((carry == 1) && !overriding_carry); i++) { // TODO check precedence
               int other_val = i < to ? other.blocks[i] : 0;
               int res = this.blocks[i%this.blocks.length + (this.blocks.length - NB_CHRS_PER_WORD)] + other_val + carry;
               if (res >= this.getMode().base) {
                     carry = 1;
                     res -= this.getMode().base;
               } else {
                     carry = 0;
               }
               this.blocks[i%this.blocks.length + (this.blocks.length - NB_CHRS_PER_WORD)] = (byte)res;
         }
         if (overriding_carry && (carry!=0)) {
               this.blocks[to%this.blocks.length] += carry;
         }
}
```

A look at the subtraction ➔ Java substract !

```java
/**
subtract the given memory to this one
@param other the memory that should be subtracted
@param from index of the block from which the subtraction should start
@param to index of the block to which the subtraction should end (excluded)
  */
public void subtract(Memory other, byte from, byte to, byte this_from, byte this_to) {
	assert (from >= 0) : "from should not be negative";
	assert (from < to) : "from should be inferior to to";
	assert (to <= this.blocks.length) : "to should be inferior to the number of blocks
	per memory";
	long valM1 = this.getDecimalValue(this_from, this_to) - other.getDecimalValue(from,
	to); // implem is done through translation to java long (int not enough !)
	this.setDecimalValue(Math.abs(valM1), this_from, this_to);
	if (valM1 < 0 && this.getMode() == MemoryMode.DECIMAL) {
		this.bullGamma.ms1 = 10;
	}
}
```

A look at the division

```
/**
divide the given memory to this one
@param other the memory that should be divided
@param from index of the block from which the division should start
@param to index of the block to which the division should end (excluded)
   */
public void divide(Memory other, byte from, byte to) {
        long vmb = other.getDecimalValue(from, to);
        if (vmb == 0) {
              throw new Error("Division by 0.");
        }
        while (this.bullGamma.md > 0) {
              while (this.getDecimalValue((byte)(from + this.blocks.length - NB_CHRS_PER_WORD), (byte)this.blocks.length) <
              vmb && this.bullGamma.md > 0) {
                    this.shiftLeft();
                    this.bullGamma.md--;
              }
              while (this.getDecimalValue((byte)(from + this.blocks.length - NB_CHRS_PER_WORD), (byte)this.blocks.length)
              >= vmb) {
                    this.blocks[0]++;
                    this.subtract(other, from, to, (byte)(from + this.blocks.length - NB_CHRS_PER_WORD),
                    (byte)this.blocks.length);
              }
        }
}
```

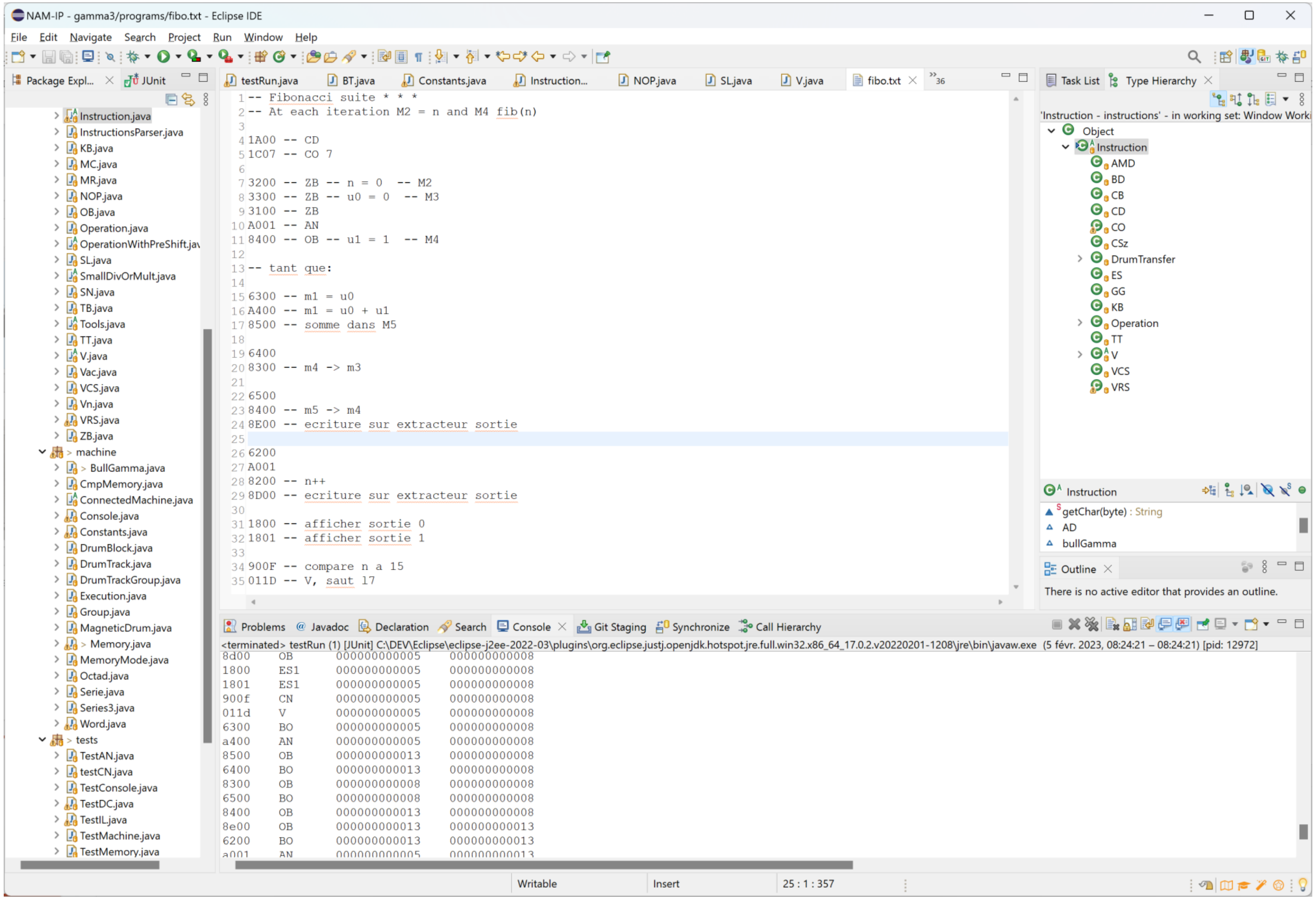
Current Experimentation Environment

IDE (Eclipse)

Fibonnacci suite

Utilities:

- sqrt
- 2^{nd} degree



Some available FP subroutines

	AVANT L'OPÉRATION							APRÈS L'OPÉRATION				
Opération	Durée opération	Variante	M_1	M_4	M_5	M_6	M_7	M_1	M_4	M_5	M_6	M_7
AB	1 pt	0 0 2 5			A	B		AB	inchangée	A	AB	inchangée
AB — C	2 pt	0 0 2 9			A	B	C	AB — C	inchangée	A	AB	AB — C
AB + C	2 pt	0 0 2 13			A	B	C	AB + C	inchangée	A	AB	AB + C
A + B	1 pt	0 0 7 13				A	B	A + B	inchangée	inchangée	A	A + B
A — B	1 pt	0 0 7 1				A	B	A — B	inchangée	inchangée	A	A — B
A / B	1 pt	0 0 15 13			x	A	B	A / B	inchangée	inchangée	A	A / B
$Ax_2 + Bx + C$ (CD av. 4 col. 6)	4 pt	0 0 2 13		C		A	B	$Ax^2 + Bx + C$	0	x	0	0
sin. et cos. x ..	4 cartes	en ligne 11			inchangée	x			chang.	inchangée	sin. x	cos. x
arctg N/D	4 cartes				inchangée	N	D	arctg	chang.	inchangée	chang.	arctg

Fig. 197

N.B. — Les sous-programmes Sin. cos. x et arctg N/D altèrent la mémoire 3.

Lessons Learned…

- Exploring early machines quite a strange journey:
  - Emerging concepts, still being explored
  - Gone technologies

  ➔need to look at larger historical context

- Not so complex to code
  but many details
  and limited original reference to test against

- Still a lot to explore (e.g. floating point), implement (UI…), gather/experiment with “code”

- In summary, an very rewarding experience both technically and culturally !

Questions ?

Computer Museum NAM-IP

Some reference & credits

- Fédération des Equipes Bull for access to their GAMMA3 documentation at NAM-IP museum or online: http://www.feb-patrimoine.com/projet/gamma3/gamma3.htm
- Vincent Joguin for its DOS-based emulator
    - vidéo (french): https://www.youtube.com/watch?v=X_ermLbQYLI
    - executable: http://vincent.joguin.com/GAMMAET.ZIP
    - José Maillard and Lucas Trampal for the Open Source javascript emulator (coordinated by ACONIT) https://github.com/lutrampal/bullgammator/
- ACONIT for on-line documentation and running emulator and keeping the GAMMA3 memory alive with students
    - documentation: https://www.aconit.org/histoire/Gamma-3
    - online emulator: https://www.aconit.org/histoire/Gamma-3/Simulateur
- Code: https://github.com/NAMIP-Computer-Museum/gamma3

Visit us: www.nam-ip.be Twitter @ComputerMuseumB

Contact me: christophe.ponsard@cetic.be @cponsard @cponsard@ludosphere.fr