MMIX

Etudes et variations / Studies and variations

Benoît J. B. D. Auguet

Le jeudi 16 février 2017.
Version 0.3.0

Grand Public / General public

[fr] *"Pour que l'esprit gagne en sagacité, on doit lui donner de l'exercice en lui faisant chercher ce que les autres ont déjà trouvé, et en lui faisant examiner méthodiquement toutes les techniques humaines, même les plus insignifiantes, mais de préférence celles qui manifestent ou présupposent un ordre."*
Descartes - Règles pour la direction de l'esprit - Règles X - Vers 1628.

[en] *"For the mind to gaining wisdom, one must give it the exercise by seeking what others have already found, and making it methodically examine all human techniques, even the most insignificant, but preferably ones which manifest or presuppose an order."*
Descartes - Rules for the Direction of the Mind - Rules X - Toward 1628.

1. Intention / Intention

[fr] Ce document regroupe une partie des *études et variations* que je réalise à titre personnel sur `MMIX`. Comme j'y travaille sur mon temps libre, je le complète trés lentement au gré des sujets qui m'imperpellent. Pour ne rien arranger, d'autres domaines me passionnent...

Comme vous le constatez, ce travail est bi-langue. Il est rédigé en français, par attachement à ma belle langue, élégante et précise, ce qui facilite ma réflexion. Il est traduit en anglais pour échanger avec un plus large public.

J'espère que ce travail vous intéressera.

Envoyer, s'il vous plaît, vos commentaires, suggestions et caetera à benoit.auguet@verslinfini.com.

[en] This document gathers a part of the *studies and variations* that I realize personally on `MMIX`. As I work on my free time, I complete it very slowly at the whim of the subjects that inspire me. To make matters worse, other areas fascinate me ...

As you can see, this work is bi-language. It is written in French, by attachment to my beautiful language, elegant and precise, which facilitates my reflection. It is translated into English to exchange with a wider audience.

I hope this work will interest you.

Please send comments, suggestions et caetera to benoit.auguet@verslinfini.com.

2. Table des matières / Contents

3. Ressources et références / Resources and references

[1] MMIX Home Page - Documentation, Sources, Binaries, Links, Examples, Contributions.

[2] Donald Knuth's page - MMIX 2009, a RISC computer for the third millennium.

[3] Groupe d'auteurs. Préfixe numérique. Wikipedia, 2012.

[4] Donald E. Knuth. The Art of Computer Programming (TAOCP). Addison-Wesley.

[5] Donald E. Knuth. MMIXware, A RISC Computer for the Third Millennium. Springer, 1999.

[6] Donald E. Knuth. MMIX, A RISC Computer for the New Millennium, volume 1, fascicle 1 of The Art of Computer Programming. Addison-Wesley, 2005.

[7] Alexander A. Stepanov et Paul McJones. Elements of Programming. Addison-Wesley, 1ère edition, 2009. http://www.elementsofprogramming.com/book.html.

[8] Martin Ruckert. The MMIX Supplement, to *The Art of Computer Programming* volumes 1, 2, 3. Addison-Wesley.

4. Motivation / Motivation

[fr] Lorsque l'on se passionne pour la programmation de logiciels, on croise tôt ou tard l'œuvre de Donald E. Knuth, *"The Art of Computer Programming" (TAOCP) / "L'Art de la Programmation des Ordinateurs"*[4]. Cette rencontre est surprenante, voire déconcertante au début, tant par la rigueur et la densité du travail exposé que par l'approche de programmation utilisée par son auteur, résumé en quatre lettres MMIX. L'ensemble des algorithmes exposés dans le TAOCP sont écrits en langage machine, ou assembler, initialement en MIX, une architecture numérique inventée dans les années 1960 par l'auteur, aujourd'hui en MMIX[6], architecture RISC 64-bits moderne du même inventeur. La phrase d'introduction de Donald E. Knuth de son ouvrage MMIXware[5] nous donne la clé pour comprendre son approche, *"MMIX est un ordinateur destiné à illustrer au niveau de la machine les aspects de la programmation"*, et la couverture du fascicule 1 décrivant son architecture, nous livre comme une promesse découvertes, "MMIX, un ordinateur RISC pour le nouveau millénaire".

N'est-ce pas suffisant pour en attaquer l'étude ?

[en] When we has a passion for software programming, sooner or later we cross the work of Donald E. Knuth, *"The Art of Computer Programming" (TAOCP)*[4]. This meeting is surprising, even disconcerting at first, both for the rigor and the density of work exhibited, as the programming approach used by its author, summarized in four letters MMIX. All of the algorithms presented in the TAOCP are written in machine language, or assembly, initially in MIX, a digital architecture invented in the 1960s by the author, today in MMIX[6], modern 64-bits RISC architecture of the same inventor. The introductory phrase of Donald E. Knuth of his book MMIXware[5] gives us the key to understanding his approach: *"MMIX is a computer intended to illustrate machine-level aspects of programming"*, and the cover of the fascicule 1 describing its architecture, gives us like a promise of discoveries, "MMIX a RISC Computer for the New Millennium".

Is it not sufficient to tackle the study?

5. Architecture / Architecture

[fr] Un processeur est défini par son jeu d'instructions. Celui-ci contient les opérations élémentaires, ou instructions, à l'œuvre sur des données, contenue dans des cases de taille fixe, les registres ou la mémoire. Ces opérations sont soit d'ordre mathématique (et logique, ou logique, ... addition entière, soustractions entière, ...), soit fonctionnelles (déplacer un donnée d'une case A à une case B, interrompre un traitement suite à un événement particulier, ...).

Le jeu d'instructions reflète aussi l'architecture du coeur processeur, l'idée générale, le fonctionnement d'ensemble qui ont prévalu à sa définition. MMIX a la particularité d'avoir été créé avec un soucis de rigueur, de complétude et de régularité, afin de corriger des défauts subtiles que l'on peut rencontrer sur d'autres architectures processeur.

Le fascicule 1 du volume 1 de TAOCP, "MMIX, A RISC Computer for the New Millennium"[6], présente cette architecture.

[en] A processor is defined by its instruction set. This contains the basic operations or instructions to work on data, contained in fixed size boxes, registers or memory. These operations are either mathematical (logical and, laogical or... integer addition, integer subtraction, ...) or functional (move a given data from box A to box B, interrupt treatment after an particular event, ...).

The instruction set also reflects the architecture of the processor core, the general idea, the overall operation that prevailed in its definition. MMIX has the particularity to have been created with a rigor worries, completeness and consistency, to correct subtle defects that may be encountered on other processor architectures.

The fascicle 1 of the volume 1 of TAOCP, "MMIX, A RISC Computer for the New Millennium"[6], presents this architecture.

6. Installer les simulateurs et outils MMIX / Installing of the MMIX simulators and tools

[fr] Ce chapitre décrit l'installation des simulateurs et outils MMIX.

[en] This chapter describes the installation of the MMIX simulators and tools.

7. Télécharger et installer les binaires / Download and install binaries

[fr] La première approche consiste à télécharger et installer les binaires en suivant les instructions du site officiel. Personnellement, je préfère la suivante.

[en] The first approach is to download and install the binaries following the instructions on the official website. Personally, I prefer the next one.

8. Télécharger et installer les sources / Download and installing sources

[fr] Sur le site officiel, vous pourvez télécharger les sources des simulateurs et outils MMIX sous forme d'une archive. La version la plus récente, lors de l'écriture de ces notes est "mmix-20160804.tgz". L'archive est a plat. Je vous conseille donc de créer un répertoire ou mettre les sources et de désarchiver vers celui-ci.

[en] On the official website, you can download the sources of the MMIX simulators and tools as an archive. The most recent version when writing these notes is "mmix-20160804.tgz". The archive is flat. I therefore advise you to create a directory where to put the sources and to unarchive to this one.

```
$ mkdir ./src/mmix/mmix-20160804
$ tar xvf ./archives/mmix-20160804.tgz -C ./src/mmix/mmix-20160804
```

9. Compiler les sources / Compile sources

[fr] Les simulateurs et outils MMIX ont été réalisés en *programmation littéraire*. Le code source C et sa documentation sont exposés dans les fichier "*.w". Pour réaliser la compilation de la documentation et des binaires, vous devez disposer d'un compilateur C d'un distribution TEX ou LATEX et des outils CWEB, normalement inclus avec cette distribution.

[en] The MMIX simulators and tools have been produced in *literate programming*. The source code C and its documentation are exposed in the file "*.w". To compile documentation and binaries, you must have a C compiler, a TEX or LATEX distribution, and the CWEB tools that are normally included with this distribution.

[fr] Vérifiez simplement la présence de cc ou gcc.

[en] Simply check the presence of cc or gcc.

```
$ which cc
/usr/bin/cc
$ which gcc
/usr/local/bin/gcc
```

[fr] Vérifiez simplement la présence de cweave et ctangle outils de CWEB.

[en] Simply check for cweave and ctangle, tools of CWEB.

```
$ which cweave
/usr/local/bin/cweave
$ which ctangle
/usr/local/bin/ctangle
```

[fr] En suivant le fichier `README`, la commande `"make doc"` génère la documentation en postscript `"mmix-doc.ps"`, `"mmix-sim-intro.ps"` et `"mmixal-intro.ps"`. Vous pouvez les convertir en `PDF` avec la command `ps2pdf` et une petite boucle `for`. Vous pouvez ensuite copier la documentation dans un répertoire où vous la conserverez.

[en] By following the file `README`, the `"make doc"` generates the documentation in postscript `"mmix-doc.ps"`, `"mmix-sim-intro.ps"` and `"mmixal-intro.ps"`. You can convert them to `PDF` with command `ps2pdf` and a small loop `for`. You can then copy the documentation to a directory where you will keep it.

```
$ cd src/mmix/mmix-20160804
$ make doc
...
$ ls *.ps
mmix-doc.ps    mmix-sim-intro.ps    mmixal-intro.ps
$ for i in *.ps; do ps2pdf "$i" "${i%.*}.pdf"; done
$ ls *.pdf
mmix-doc.pdf   mmix-sim-intro.pdf   mmixal-intro.pdf
$ mkdir ../mmix-20160804-doc
$ cp *.ps ../mmix-20160804-doc
$ cp *.pdf ../mmix-20160804-doc
```

[fr] La commande `"make all"` génère les binaires suivants,

- `mmixal`, l'assembleur,
- `mmix`, le simulateur simple,
- `mmmix`, le méta-simulateur à file d'instructions,
- `mmotype`, le programme de conversion d'exécutables en un format lisible.

Pour compiler les binaires en version optimisé, vous devez éditer le fichier `Makefile` se trouvant dans le répertoire des sources et modifier l'option `"CFLAGS = -g"`, par `"CFLAGS = -O3"`.

[en] The command `"make all"` generates the following binaries,

- `mmixal`, the assembly program,
- `mmix`, the simple simulator,
- `mmmix`, the pipelined meta-simulator
- `mmotype`, the program to convert binary executables to readable format.

To compile the binaries in an optimized version, you must edit the file `Makefile` in the source directory and change the option `"CFLAGS = -g"` to `"CFLAGS = O3"`.

```
$ make all
```

[fr] Vous pouvez ensuite copier les binaires générés dans un répertoire où vous les conserverez.

[en] You can then copy the generated binaries to a directory where you will keep them.

```
$ mkdir ../mmix-20160804-bin
$ cp mmixal ../mmix-20160804-bin/
$ cp mmix ../mmix-20160804-bin/
$ cp mmmix ../mmix-20160804-bin/
$ cp mmotype ../mmix-20160804-bin/
```

[fr] Si vous le souhaitez installer les binaires au niveau du système, et que vous avez le privilège admistrateur, vous pouvez utilisez la commande suivante pour les copier dans, par exemple, `"/usr/local/bin/"`.

[en] If you want to install the binaries at the system level, and you have administrator privilege, you can use the following command to copy them to, for example, `"/usr/local/bin/"`.

```
$ sudo cp ../mmix-20160804-bin/* /usr/local/bin/
```

10. Notations, conventions et définitions

[fr] Ce chapitre présente les notations et conventions utilisées par l'auteur.

[en] This chapter presents the notations and conventions used by the author.

11. Bits, octets et mots / Bits, bytes and words

[fr] En informatique, un bit, contraction du terme anglais *binary digit*, est un chiffre binaire, ${}^{b}0$ ou ${}^{b}1$. C'est aussi l'information la plus élémentaire, appelons la quantum d'information, prenant l'une deux valeurs binaires ${}^{b}0$ ou ${}^{b}1$, ou encore le quantum ${}^{b}0$ et le quantum ${}^{b}1$. C'est encore l'unité de mesure élémentaire de la longueur d'une séquence finie de bits.

En électronique, un bit est un dispositif à deux états maintenant un quantum d'information, ce quantum pouvant être modifié ou ne pas l'être. Il exite plusieurs types de tels dispositifs dans différentes technologies.

[en] In computer science, a bit, contraction of the English word *binary digit*, is digit, ${}^{b}0$ or ${}^{b}1$. It is also the most elementary information, called the information quantum, taking one of two binary values ${}^{b}0$ or ${}^{b}1$, or the quantum ${}^{b}0$ and the quantum ${}^{b}1$. It is still the elementary unit of measurement of the length of a finite sequence of bits.

In electronics, a bit is a two-state device maintaining a quantum of information, this quantum can be modified or not. There are several types of such devices in different technologies.

[fr] Nos machines binaires manipulent de l'information sous forme de séquences finies de ${}^{b}0$ ou ${}^{b}1$. Nous appelons, une séquence finie de ${}^{b}0$ et de ${}^{b}1$, ou séquences de bits, ou séquences binaires, ou séquences de quanta d'information, un datum[1], au pluriel des data[2].

Nous écrirons en notation binaire les data de gauche à droite, le premier élément étant le plus à gauche. Nous les préfixerons par un bémol ^{b} pour signifier qu'il s'agit d'une séquence binaire. Voici différents data. Ils n'ont pas forcément la même longueur et l'on peut s'interroger sur ce qu'ils représentent.

[en] Our binary machines manipulate information in the form of finite sequences of ${}^{b}0$ or ${}^{b}1$. We call, a finite sequence of ${}^{b}0$ and ${}^{b}1$, or bit sequence, or binary sequence, or information quantuns sequences, a datum[3] in the plural the datums[4].

We write in binary notation the data from left to right, the first element being leftmost. We prefix them with a flat ^{b} to signify that it is a binary sequence. Here are different data. They do not necessarily have the same length and we can wonder about what they represent.

$$ {}^{b}0 \quad {}^{b}1 \quad {}^{b}0000 \quad {}^{b}0001 \quad {}^{b}1001 \quad {}^{b}1001001 \quad {}^{b}01001001 \quad {}^{b}10110110 \tag{0.1} $$

[fr] Les notions de quantum et de datum nous permettent d'introduire un notion d'information brute, indépendante de toute interprétation.

[en] The notions of quantum and datum allow us to introduce a notion of raw information, independent of any interpretation.

[1] [fr] Définition introduite par **Alexander A. Stepanov** et **Paul McJones**, dans leur ouvrage *Elements of Programming* [7]-§1.2-page 2 (2009).

[2] [fr] Latin : datum (datum, dati) nom neutre II déclinaison - 1. don, chose offerte - 2. (au pluriel) dépenses, sorties - nominatif singulier *datum*, pluriel *data*.

[3] [en] Definition introduced by **Alexander A. Stepanov** and **Paul McJones**, in their book *Elements of Programming* [7]-§1.2-page 2 (2009).

[4] [en] Latin: datum (datum, dati) neutral name II declension - 1. donation, thing offered - 2. (plural) expenditures - singular nominative *datum*, plural *data*.

[fr] Nos machines binaires manipulent des data de tailles prédéfinies, communément de 8, 16, 32, 64 voire 128 bits de longeur, d'autres longueurs pouvant être définies pour des besoins particuliers voire ponctuels.

En informatique, un mot est une séquence finie de bits contenant des quanta d'information d'une taille fixe l prenant comme valeur l'un des 2^l data possibles de longueur l.

De même que le terme *bit*, le terme *mot*, a un sens générale, abstrait et mathématique, lorsque nous abordons l'aspect théorique ou descriptif d'un sujet (comme ici), et, un sens particulier, concret et physique, lorsque nous parlons des dispositifs électonique de nos machines.

[en] Our binary machines manipulate data of predefined sizes, commonly 8, 16, 32, 64 or even 128 bits in length, other lengths can be defined for special needs or even punctual.

In computer science, a word is a finite sequence of bits containing information quanta of fixed size l taking as a value one of the possible 2^l datums of length l.

Just as the term *bit*, the word italic word, has a general, abstract and mathematical meaning when we approach the theoretical or descriptive aspect of a subject (as here), and, a particular meaning, concrete and physical, when we talk about the electonic devices of our machines.

[fr] Il est commode de nommer les longueurs ou quantités de bits communément utilisées. En français une quantité de 8 bits s'appelle un octet, en englais a byte. Suivons Donald E. Knuth[5] et définissons les quantités de 16, 32 et 64 bits. Knuth les définit par le terme anglais wyde, pour une quantité de 2 octets soit 16 bits, et les termes grecs, tetra et octa, pour les quantités de 4 et 8 octets. Les préfixes numériques tétra et octa, issus du grecs, sont aussi d'usage en français. Nous les conservons. Nous ajoutons le préfixe grec duo pour les quantités de 16 bits et hexa pour la quantité de 128 bits soit 16 octets.

[en] It is convenient to name the commonly used lengths or quantities of bits. In French a quantity of 8 bits is called a octet, in English a byte. Follow Donald E. Knuth[6] and define the quantities of 16, 32 and 64 bits. Knuth defines them by the English word wyde, for a quantity of 2 bytes or 16 bits, and the Greek terms tetra and octa 4 and 8 bytes. The numerical prefixes tetra and octa, derived from the Greek, are also used in French. We keep them. We add the Greek prefix duo in French for the quantities of 16 bits and hexa for the quantity of 128 bits or 16 octets.

Français / French	**Anglais / English**	**Quantité / Quantity** octet / byte	**Quantité / Quantity** bit / bit
bit	bit	1/8	1
octet	byte	1	8
duo	wyde	2	16
tétra - tetra	tetra	4	32
octa	octa	8	64
hexa	hexa	16	128

Table 1: Définitions des longueurs ou quantité de bits.

[fr] Maintenant, écrire le datum d'un octa comme une succession de 64 $^{b}0$ ou $^{b}1$ est fastidieux.

[5] [fr] Définitions introduites par Donald E. Knuth au début de la description de son architecture processeur MMIX dans TAOCP volume 1, fascicle 1, [6]-§1.3.1-page 4 (2005).

[6] [en] Definitions introduced by Donald E. Knuth at the beginning of the description of its MMIX processor in TAOCP volume 1, fascicle 1, [6]-§1.3.1-page 4 (2005).

[en] Now, writing the datum of an octa as a succession of 64 $^{b}0$ or $^{b}1$ is tedious.

$$^{b}0100110001100101001000000100110000100000001110100101111000101001 \tag{0.2}$$

[fr] En regroupant les bits par quatre, nous pouvons remplacer les chiffres binaires par les chiffres hexadécinaux définis par les symboles suivants.

[en] By grouping the bits by four, we can replace the binary digits with hexadecimal digits defined by the following symbols.

$$\begin{array}{llll} {}^{\#}0 = {}^{b}0000 & {}^{\#}4 = {}^{b}0100 & {}^{\#}8 = {}^{b}1000 & {}^{\#}\mathtt{c} = {}^{b}1100 \\ {}^{\#}1 = {}^{b}0001 & {}^{\#}5 = {}^{b}0101 & {}^{\#}9 = {}^{b}1001 & {}^{\#}\mathtt{d} = {}^{b}1101 \\ {}^{\#}2 = {}^{b}0010 & {}^{\#}6 = {}^{b}0110 & {}^{\#}\mathtt{a} = {}^{b}1010 & {}^{\#}\mathtt{e} = {}^{b}1110 \\ {}^{\#}3 = {}^{b}0011 & {}^{\#}7 = {}^{b}0111 & {}^{\#}\mathtt{b} = {}^{b}1011 & {}^{\#}\mathtt{f} = {}^{b}1111 \end{array} \tag{0.3}$$

[fr] Comme pour la notation binaire, nous écrirons en notation hexadécimale les data de gauche à droite, le premier élément étant le plus à gauche. Nous les préfixerons par un dièse # pour signifier qu'il s'agit d'une séquence hexadecimale, reprenant ainsi la notation de Knuth[7]. Le préfixe usuel en informatique est `0x`.

[en] As for the binary notation, we write in hexadecimal notation the datums from left to right, the first element being leftmost. We prefix them with a # sign to signify that it is a hexadecimal sequence, thus taking the notation of Knuth[8]. The usual prefix in computing is `0x`.

[fr] Le datum précédent se transforme de la sorte. [en] The preceding datum is transformed in this way.

$^{b}0100$	1100	0110	0101	0010	0000	0100	1100	0010	0000	0011	1010	0101	1110	0010	1001
$^{\#}4$	c	6	5	2	0	4	c	2	0	3	a	5	e	2	9

[fr] Donnant le résultat. [en] Giving the result.

$$^{\#}\mathtt{4c65204c203a5e29} \tag{0.4}$$

[fr] Les lettres minuscules des symboles peuvent être écrites en masjuscule. La notation $^{\#}$`4C65204C203A5E29` est équivalente à la précédente.

[en] The lowercase letters of the symbols can be written in uppercase. The $^{\#}$`4C65204C203A5E29` notation is equivalent to the previous one.

[7] Cette notation est introduite par Knuth dans TAOCP volume 1, fascicle 1, [?]-§1.3.1-page 4 (2005). Notez que la présentation des notions de ce chapitre reprend en partie celle du fascicule de Knuth, d'une très grande clareté et concision. Dès lors, il n'a été très difficile de m'en détacher.

[8] This notation is introduced by Knuth in TAOCP volume 1, fascicle 1, [6]-§1.3.1-page 4 (2005). Note that the presentation of the notions of this chapter partly reproduces that of the Knuth fascicle, very clear and concise. It was therefore very difficult to detach myself from it.

12. Simple programme bonjour / Simple hello program

[fr] Voici le plus simple programme que nous puissions écrire pour afficher la chaîne de caractères `"Bonjour Benoît :-) / Hello Benoît :-)"`. Il est écrit en `MMIXAL`, "`MMIX` Assembly Language", le language assembleur de `MMIX`. Comme vous le constatez, il s'agit d'un programme bi-langue, français-englais.

[en] Here is the simplest program we can write to display the string `"Bonjour Benoît :-) / Hello Benoît :-)"`. It is written in `MMIXAL`, the `MMIX` Assembly Language. As you can see, it is a bi-language program, French-English.

```
1           LOC   #100
2   Main    GETA  $255,Chaîne       $255 <- adresse de la chaîne / string address
3           TRAP  0,Fputs,StdOut    appelle 'Fputs' sur la sortie standard
4                                 / calls of 'Fputs' for standard output
5           SET   $255,0            $255 <- 0 : retour du programme / program return
6           TRAP  0,Halt,0          arrêt / halt
7   Chaîne  BYTE  "Bonjour Benoît :-) / Hello Benoît :-)",#a,0
```

Source 1: `bonjour.mms` – Simple programme bonjour / Simple hello program.

[fr] Détaillons ligne par ligne ce programme.

Ligne 1 – L'instruction `LOC` défini l'adresse de départ du programme, ici en héxadécimal, #`100`.

Ligne 2 – L'étiquette `Main` est le point d'entrée du programme ; si elle n'est pas présente, `MMIX` ne pourra pas lancer le programme. L'instruction `GETA`, pour "get adresse", charge l'adresse relative de l'étiquette `Chaîne` dans le registre `$255`. Notez que cette étiquette contient un accent. Le fichier source est au format UTF8 et les accents peuvent être utilisé pour les étiquettes sans poser de difficulté à l'assembleur mmixal.

Ligne 3 – L'instruction `TRAP` réalise un appel d'une routine système d'écriture d'une chaine de caractères d'un octet dans un fichier, `Fputs`. Le fichier en question est la sortie standard, `StdOut`, c'est à dire l'écran ou la console. Cette routine utilise l'adresse présente dans le registre `$255` et écrit chaque caractère vers la sortie jusqu'à atteindre un caractère nul. Elle retourne le nombre d'octets écrits dans le registre `$255`, ici 40 ('î' est un caractère unicode de 2 octets).

Lignes 5 et 6 – L'instruction `SET` met à zéro la valeur du registre `$255` (précédemment 40). La deuxième instruction `TRAP` réalise un appel d'une routine système d'arrêt de `MMIX` et retourne le contenu du registre `$255`. On sort ainsi de la simulation qui elle-même retourne la valeur de retour du programme. – Cette mise à zéro, ou à une valeur de retour correspondante à un besoin fonctionnel, à son importance. Par exemple, j'automatise systématiquement, la compilation et l'exécution d'un programme par un fichier `Makefile`. Le retour autre que zéro de l'appel du simulateur sur le programme bonjour provoque une erreur lors de l'exécution du `Makefile`.

Ligne 7 – Voici la chaîne de caractères. Notez que les accents dans le texte ne pause pas de problème. Nous pouvons donc utiliser des codes unicode UTF8 dans cette chaîne de caractères. Le code #`a` ajouté à la fin correspond à un saut de ligne. Le 0 final est le caractère nul de fin de chaîne.

[en] Let us detail this program line by line.

Line 1 – The instruction `LOC` set the starting address of the program here in hexadecimal, #100.

Line 2 – The label `Main` is the entry point of the program ; if it is not present, the `MMIX` will not be able to start the program. The `GETA` instruction, for "get address", loads the relative address of the label `Chaîne`, for "String", into the `$255` register. Note that this label contains an accent. The source file is in UTF8 format and the accents can be used for labels without asking the assembler mmixal.

Line 3 – The `TRAP` instruction performs a call to a system routine of writing a one-byte character string to a file, `Fputs`. The file in question is the standard output, `StdOut`, ie the screen or the console. This routine uses the address in the `$255` register and writes each character to output until it reaches a null character. It returns the number of bytes written to the `$255` register, here 40 ('î' is a 2-byte unicode character).

Line 5 and 6 - The `SET` instruction set to zero value of the `$255` register (previously 40). The second `TRAP` instruction makes a call for a stop to `MMIX` system routine. This thus leaves the simulation which in turn returns the return value of the program. – This setting to zero, or to a return value corresponding to a functional requirement, its importance. For example, I systematically automate the compilation and execution of a program by a file `Makefile`. The non-zero return of the simulator call to the program hello causes an error when running the `Makefile`.

Line 7 - Here is the character string. Note that the accents in the text do not pause any problem. We can therefore use UTF8 code in this string. The #`a` code added at the end is a line break. The final 0 is the null end of string.

[fr] Le fichier objet `boujour.mmo` s'obtient en utilisant le progamme assembleur `mmixal`. L'option `-l` servant à générer un fichier assembleur symbolique contenant l'adressage des instructions et la table des symboles (ci-dessous). La commande `mmix bonjour` lance `MMIX` avec le progarmme `boujour.mmo`.

[en] The file object `boujour.mmo` is obtained by using the assembly program `mmixal`. The `-l` option is used to generate a symbolic listing file containing the addressed instructions and the symbol table (below). The `mmix hello` command launches `MMIX` with the `boujour.mmo` program.

```
$ mmixal -l bonjour.lst bonjour.mms
$ mmix bonjour
Bonjour Benoît :-) / Hello Benoît :-)
$
```

```
1                                LOC   #100
2    ...100: f4ffxxxx  Main      GETA  $255,Chaîne       $255 <- adresse de la chaîne / string address
3    ...104: 00000701            TRAP  0,Fputs,StdOut    appelle 'Fputs' sur la sortie standard
4                                                      / calls of 'Fputs' for standard output
5    ...108: e3ff0000            SET   $255,0            $255 <- 0 : retour du programme / program return
6    ...10c: 00000000            TRAP  0,Halt,0          arrêt / halt
7    ...110: 426f6e6a  Chaîne    BYTE  "Bonjour Benoît :-) / Hello Benoît :-)",#a,0
8    ...114: 6f757220
9    ...118: 42656e6f
10   ...11c: c3ae7420
11   ...120: 3a2d2920
12   ...124: 2f204865
13   ...128: 6c6c6f20
14   ...12c: 42656e6f
15   ...130: c3ae7420
16   ...134: 3a2d290a
17   ...138: 00
18
19   Symbol table:
20    Chaîne = #0000000000000110 (2)
21    Main = #0000000000000100 (1)
```

Source 2: `bonjour.lst` – Fichier assembleur symbolique de Bonjour / Symbolic listing file of Hello.

13. Imprimer des entiers / Print interger

[fr] Cet chapitre traite de l'impression de nombres entiers en chaîne de caractères.

[en] This chapter discusses printing of interger numbers as a string.

14. Imprimer en notation décimal / Print in decimal notation

[fr] Nous souhaitons tout d'abord réaliser un programme `MMIX` imprimant à l'écran, en notation décimale, la valeur entière correspondant au contenu d'un registre. Un registre `$X` à une taille d'un octa soit 64 bits.

Nous considérons dans un premier temps que le registre contient un entier non signé ou entier naturel. Les data contenus dans un registre vont de #`0000000000000000` correspondant à la valeur décimale 0, à, #`ffffffffffffffff` correspondant à la valeur décimale $18\ 446\ 744\ 073\ 709\ 551\ 615$.

[en] First, we want to create a `MMIX` program that prints on the screen, in decimal notation, the integer value corresponding to the contents of a register. A register `$X` has a size of an octa or 64 bits.

We consider in at the first that the register contains an unsigned integer or natural integer. The data contained in a register goes from #`0000000000000000` corresponding to the decimal value 0, to, #`ffffffffffffffff` corresponding to the decimal value $18\ 446\ 744\ 073\ 709\ 551\ 615$.

15. La division euclidienne et `MMIX` / The Euclidean division and `MMIX`

[fr] La division euclidienne est définie par le théorème suivant.

[en] The Euclidean division is defined by the following theorem.

Théorème 1 (Division euclidienne / Euclidean division)

[fr] Soient a et b deux entiers naturels non nuls. Il exite deux uniques entiers naturels, le quotient q et le reste r, tel que l'on puisse écrire $a = b \times q + r$ avec $0 \leq r < b$. L'entier a est appelé le dividende et l'entier b le diviseur.

[en] Let a and b be two natural nonzero integers. There exist two unique natural integers, the quotient q and the remainder r, such that one can write $a = b \times q + r$ with $0 \leq r < b$. The integer a is called the dividend and the integer b the divisor.

[fr] L'instruction `DIVU` de `MMIX` réalise la division euclidienne d'entiers non signés. Son format est à 3 opérandes `DIVU $X,$Y,$Z|Z` et elle utilise deux registres spéciaux, le *registre dividende* `rD` étendant le registre `$Y`, le dividende `rD $Y` est donc de 128-bits, le *registre reste* `rR`, retournant le reste une fois l'instruction exécutée. Le registre `$Z`, ou l'octet `Z`, contient le diviseur. Le registre `$X` retourne le quotient une fois l'instruction exécutée. La définition formelle de l'opération est décrite ci-dessous et pour l'instant nous nous plaçons dans le cas $\mathrm{rD} = 0$.

[en] The instruction `DIVU` of `MMIX` performs Euclidean division of unsigned integers. Its format is 3 operands `DIVU $X,$Y,$Z|Z` and it uses two special registers, *dividend register* `rD` extending the `$Y`, the `rD $Y` is therefore 128-bit, the *remainder register* `rR`, returning the remainder after the instruction is executed. The register `$Z`, or the byte `Z`, contains the divisor. The `$X` register returns the quotient after the instruction is executed. The formal definition of the operation is described below and for the moment we put in the case $\mathrm{rD} = 0$.

- `DIVU $X,$Y,$Z|Z` (division non signé / divide unisgned) :
  cas générale/general case

  si/if $\mathrm{u}(\$\mathrm{Z}|\mathrm{Z}) > \mathrm{u}(\mathrm{rD})$
  $\mathrm{u}(\$\mathrm{X}) \leftarrow \lfloor \mathrm{u}(\mathrm{rD}\,\$\mathrm{Y})/\mathrm{u}(\$\mathrm{Z}|\mathrm{Z}) \rfloor,\ \mathrm{u}(\mathrm{rR}) \leftarrow \mathrm{u}(\mathrm{rD}\,\$\mathrm{Y}) \bmod \mathrm{u}(\$\mathrm{Z}|\mathrm{Z})$
  sinon/else
  $\$\mathrm{X} \leftarrow \mathrm{rD},\ \mathrm{rR} \leftarrow \$\mathrm{Y}.$

  cas/case $\mathrm{rD} = 0$

  si/if $\mathrm{u}(\$\mathrm{Z}|\mathrm{Z}) > 0$
  $\mathrm{u}(\$\mathrm{X}) \leftarrow \lfloor \mathrm{u}(\$\mathrm{Y})/\mathrm{u}(\$\mathrm{Z}|\mathrm{Z}) \rfloor,\ \mathrm{u}(\mathrm{rR}) \leftarrow \mathrm{u}(\$\mathrm{Y}) \bmod \mathrm{u}(\$\mathrm{Z}|\mathrm{Z})$
  sinon / else
  $\$\mathrm{X} \leftarrow 0,\ \mathrm{rR} \leftarrow \$\mathrm{Y}.$

[fr] Faisons le lien avec le théorème. Pour réaliser la divisions euclidienne de $a \neq 0$ par $b \neq 0$ nous devons définir l'entier a comme dividende, l'entier b comme diviseur, exécuter l'instruction `DIVU` et lire le quotient et le reste.

[en] Let us make the connection with the theorem. To realize the Euclidean division of $a \neq 0$ by $b \neq 0$ we must define the integer a as dividend, the integer b as divisor, execute the `DIVU` instruction and read the quotient and the rest.

$$\$\mathrm{Y} \leftarrow \mathrm{a}\ ;\ \$\mathrm{Z} \leftarrow \mathrm{b}\ ;\ \mathrm{DIVU}\ \$\mathrm{X}, \$\mathrm{Y}, \$\mathrm{Z}\ ;\ \mathrm{q} \leftarrow \$\mathrm{X}\ ;\ \mathrm{r} \leftarrow \mathrm{rR}$$

[fr] Une astuce pour faciliter l'utilisation de 'linstruction consiste à la noter `DIVU ($Q,[rR]),$A,$B|B` et dans le cas générale `DIVU ($Q,[rR]),[rD]$A,$B|B`.

[en] A tip to facilitate the use of the instruction is to note `DIVU ($Q,[rR]),$A,$B|B` and in the general case `DIVU ($Q,[rR]),[rD]$A,$B|B`.

16. Première algorithme / First algorithm

[fr] Partons par exemple de l'entier naturel 1728. Nous savons qu'il se décompose en somme de multiples de puissances de 10 : $1728 = 1 \times 1000 + 7 \times 100 + 2 \times 10 + 8 = 1 \times 10^3 + 7 \times 10^2 + 2 \times 10^1 + 8 \times 10^0$.

Si nous réalisons la division euclidienne du nombre 1728 par 1000, nous obtenons le quotient 1 et le reste 728 : $1728 = 1000 \times 1 + 728$. Maintenant en divisant le premier reste par le 100, nous obtenons le quotient 7 et le reste 28 : $728 = 100 \times 7 + 28$. Nous passons de 1000 à 100 en divisant par 10.

En répétant l'opération nous obtenons la séquence suivante, dont les quotients correspondent aux chiffres du nombre 1728 donnés dans l'ordre d'écriture. Nous arrêtons après que le diviseur vaut 1.

[en] Take for example the natural integrer 1728. We know it is broken down into sum of multiple of powers of 10 : $1728 = 1 \times 1000 + 7 \times 100 + 2 \times 10 + 8 = 1 \times 10^3 + 7 \times 10^2 + 2 \times 10^1 + 8 \times 10^0$.

If we realize the Euclidean division of 1728 per 1000, We get the quotient 1 and the remainder 728 : $1728 = 1000 \times 1 + 728$. Now dividing the first remainder by 100, we get the quotient 7 and the remainder 28 : $728 = 100 \times 7 + 28$.

By repeating the operation we obtain the following sequence, whose quotients correspond to the digits of the number 1728 given in the order of writing. We stop after the divisor is 1.

$$1728 \quad = \quad 1000 \times 1 + 728$$

$$\begin{aligned} 728 &= 100 \times 7 + 28 \\ 28 &= 10 \times 2 + 8 \\ 8 &= 1 \times 8 + 0 \end{aligned}$$

[fr] Nous pouvons en déduire un algorithme générale pour afficher l'entier naturel n contenu dans un octa.

[en] We can deduce a general algorithm to display the natural integer n contained in an octa.

A1	$\mathrm{a} \leftarrow \mathrm{n}$ $\mathrm{b} \leftarrow 10\,000\,000\,000\,000\,000\,000$	Initialisation / Initialization.
A2	$(\mathrm{q}, \mathrm{r}) \leftarrow \mathrm{a}/\mathrm{b}$	Division euclidienne / Euclidean division.
	Imprimer q à l'écran / Print q to screen.	Impression du caractère / Printing of character.
	$\mathrm{a} \leftarrow \mathrm{r}$ $\mathrm{b} \leftarrow \mathrm{b}/10$ Si $\mathrm{b} \neq 0$ aller en A2 / If $\mathrm{r} \neq 0$ go to A2.	Itération suivante ? / Next iteration?
A3	Imprimer un retour à la ligne / Print a newline to screen.	Terminaison / Terminaison. Fin / The end.

Table 2: Algorithme A - Imprimer en notation décimal / Print in decimal notation.

[fr] Voici notre première algorithme codé en `MMIXAL`. [en] Here is our first algorithm coded in `MMIXAL`.

```
1   n     IS    1844674407370955161 5    Le plus grand entier / The largest integer.
2   b0    IS    10000000000000000000     Le plus grand quotient / The largest quotient.
3   a     IS    $0                       Le dividende / The dividend.
4   b     IS    $1                       Le diviseur / The divisor.
5   q     IS    $2                       Le quotient / The quotient.
6   r     IS    $3                       Le reste / The remainder.
7   c     IS    $4                       Le caractère de sortie / The output character.
8   ch_   IS    $5                       L'adresse de la chaîne / The address of the string.
9   ch    BYTE  'c',0                    La chaîne de sortie / The output string.
10
11        LOC   #100                   * A1 - Initialisation / Initialization.
12  Main  SETH  a,(n>>48)&#ffff          a <- n
13        INCMH a,(n>>32)&#ffff
14        INCML a,(n>>16)&#ffff
15        INCL  a,n&#ffff
16        SETH  b,(b0>>48)&#ffff         b <- b0
17        INCMH b,(b0>>32)&#ffff
18        INCML b,(b0>>16)&#ffff
19        INCL  b,b0&#ffff
20        GETA  ch_,ch                  ch_ <- @ch, ainsi/so ch_,i <=> ch[i]
21
22                                      * A2 - Division euclidienne / Euclidean division.
23  0H    DIVU  q,a,b                     (q,r) <- a/b
24        GET   r,rR
25                                      * Imprimer q à l'écran / Print q to screen.
26        ADDU  c,q,'0'                   c <- '0'+q <=> 'q'
```

```
27          STBU    c,ch_,0                 ch[0] <- 'q' <=> ch = 'q',0
28          SET     $255,ch_                $255 <- ch_
29          TRAP    0,Fputs,StdOut
30                                        * Itération suivante ? / Next iteration?
31          SET     a,r                     a <- r
32          DIVU    b,b,10                  b <- b / 10
33          BNZ     b,0B                    si/if b != 0 => itérer / iterate.
34
35                                        * A3 - Terminaison / Terminaison.
36          SET     c,#a                    c <- '\n'
37          STBU    c,ch_,0                 ch[0] <- '\n' <=> ch = '\n',0
38          SET     $255,ch_                $255 <- ch[0]
39          TRAP    0,Fputs,StdOut
40                                        * Fin / The end.
41          SET     $255,0
42          TRAP    0,Halt,0                * Arrête MMIX / Halt MMIX.
```

Source 3: `imprimer_A01.mms` – Algorithme A01 - Imprimer en notation décimal / Print in decimal notation.

[fr] Voila le résultat de la compilation et de l'exécution. [en] Here is the result of compilation and execution.

```
$ mmixal -b 100 -l imprimer_A01.lst imprimer_A01.mms
$ mmix imprimer_A01
18446744073709551615
```

17. Ligne par ligne / Line by line

[fr] Lignes 1 et 2 – Le nom symbolique `IS` de `MMIXAL` permet de définir des valeurs décimales ou hexadécimales. Ici $\mathrm{n} = 18446744073709551615$ et $\mathrm{b0} = 10000000000000000000$.

[en] Lines 1 and 2 – The symbolic name `IS` of `MMIXAL` allows you to define decimal or hexadecimal values. Here $\mathrm{n} = 18446744073709551615$ and $\mathrm{b0} = 10000000000000000000$.

[fr] Lignes 3 à 7 – Le nom symbolique `IS` de `MMIXAL` permet d'associer des noms à des registres locaux.

[en] Lines 3 to 7 – The symbolic name `IS` of `MMIXAL` allows to associate names with local registers.

[fr] Lignes 8 et 9 – La fonction système '`Fputs`' prend l'adresse d'une chaîne de caractères en entrée. Nous devons donc avoir une chaîne de caractères `ch` se terminant par zéro, où écrire le caractère, et avoir son adresse `ch_`.

[en] Lines 8 and 9 – The system function '`Fputs`' takes the address of a string as input. We must therefore have a string `ch` ending with zero, where to write the character, and have its address `ch_`.

[fr] Lignes 12 à 19 – L'initialisation d'un registre à une valeur constante sur la totalité de l'octa peut se faire par 4 instructions successives définissant respectivement, les dys, haut (H), moyen-haut (MH), moyen-bas (ML) et bas (L). Les trois premiers dys peuvent être obtenus par décalage, respectivement de 48, 32, 16 bits et masquage de 16 bits, le quatrienne s'obtenant par un simple masquage de 16 bits. `MMIXAL` réalise le calcul des expressions `(x>>dec)&#ffff`. Les constantes `n` et `b0` sont donc implantées directement dans le code, ce qui est gênant pour la première. Nous verrons à changer ceci par la suite.

[en] Lines 12 to 19 – The initialization of a register to a constant value over the entire octa can be done by 4 successive instructions defining, respectively, the wydes, high (H), medium-high (MH), medium-low (ML) and low (L). The first three wydes can be obtained by shifting, respectively of 48, 32, 16 bits and masking of 16 bits, the fourth being obtained by a simple masking of 16 bits. `MMIXAL` performs the calculation of the expressions `(x>>dec)&#ffff`. The constants `n` and `b0` are thus implanted directly in the code, which is annoying for the first. We will see to change this afterwards.

[fr] Lignes 20, 27 et 37 – Nous initialisons `ch_` avec l'adresse de `ch`. Nous avons ici introduit une notation particulière. En effet lorsque nous voulons accéder au caractère de la chaîne en position `i`, nous le notons `ch[i]`. Or les instructions de lecture et d'écriture mémoire (lignes 27 et 37), additionnent les deux opérandes `$Y,$Z` pour former l'adresse. Pour `i` correctement calculé, l'écriture `'ch_,i'` se lit `ch[i]`. L'instruction d'écriture d'un octet en mémoire, lignes 27 et 37, `'STBU c,ch_,0'` se lit donc `'ch[0] <- c'`.

[en] Lines 20, 27 and 37 – We initialize `ch_` with the address of `ch`. We have here introduced a special notation. Indeed when we want to access the character of the string in position `i`, we denote it `ch[i]`. Or read instructions and memory write (lines 27 and 37), add the two operands `$Y,$Z` to form the address. For `i` correctly computed, the write `'ch_,i'` reads `ch[i]`. The write instruction of a byte in memory, lines 27 and 37, `'STBU c,ch_,0'` thus reads `'ch [0] <- c'`.

[fr] Lignes 22 à 33 – Nous retrouvons les étapes A2, de notre algorithme, la division euclienne, l'impression du caractère correspondant à `q`, la préparation de l'itération suivante.

Notez qu'il suffit d'additionner (ligne 26) la valeur de `q` au caractère '0' pour obtenir le caractère correspondant `'q'`, parce que les caractères '0', '1', ..., '9' ont des valeurs qui se suivent.

Pour itérer, nous utilisons l'instruction `BNZ`, en anglais *"branch if nonzero"* et en français *"bifurquer si non zéro"*. Si `b` est non nul, nous voulons itérer et retourner au début de A2. En début de ligne 23, l'étiquette `0H` signifie en anglais *"0 here"* et en français *"ici 0"*. Ligne 33, le synbole `0B` de l'instruction `BNZ` signifie en anglais *"0 backward"* et en français *"0 en arrière"*, ce que `MMIXAL` interprète comme l'étiquette `0H` précédant l'adresse courante où se trouve `0B`. L'instruction de bifucation s'écrit donc `BNZ b,0B` (ligne 33).

Il est aussi possible d'aller en avant vers l'étiquette `0H` suivant l'adresse courante avec le synbole `0F`, signifiant en anglais *"0 forward"* et en français *"0 en avant"*. Dix étiquettes `0H,1H, ..., 9H` peuvent être utilisées plusieurs fois puisque les symboles `0B,1B, ..., 9B` et `0F,1F, ..., 9F` se réfèrent toujours à l'étiquette la plus proche.

[en] Lines 22 to 33 - We find the steps A2, of our algorithm, the Euclidean division, the printing of the character corresponding to `q`, the preparation of the next iteration.

Note that it is sufficient to add (line 26) the value of `q` to character '0' to get the corresponding `'q'` character because the characters '0', '1' ..., '9' have values that follow one another.

To iterate, we use the instruction `BNZ`, *"branch if nonzero"*. If `b` is nonzero, we want to iterate and to return to the beginning of A2. Beginning of the line 23, the label `0H` means "0 here". Line 33, the synbole `0B` of the instruction `BNZ` means *"0 backward"*, which `MMIXAL` interprets as the `0H` label preceding the current address where `0B` is located. The branch instruction is written `BNZ b, 0B` (line 33).

It is also possible to go forward to the label `0H` according to the current address with the `0F` synbole, meaning *"0 forward"*. Ten labels `0H,1H, ..., 9H` can be used several times since the symbols `0B, 1B, ..., 9B` and `0F, 1F, ..., 9F` always refer to the closest label.

[fr] Lignes 35 à 39 – Une fois que `b` a atteint zéro, nous affichons une retour à la ligne et ainsi terminons l'algorithme.

[en] Lines 35 to 39 - Once `b` has reached zero, we display a newline and thus terminate the algorithm.

18. Erreurs et bogues commis / Errors and bugs committed

[fr] Lorsque vous suivez un cours, écoutez un raisonnement, lisez des équations, ou encore du code, vous avez l'inpression que le professeur ou la personne expérimentée, qui fait cet exposé, ne commet pas d'erreur. Et cela peut être décourageant pour vous d'en commettre. Vous ne vous sentez pas à la hauteur. Et bien sachez qu'ils en commettent mais qui'ils ne le disent pas.

Ce sont ces mêmes propos que le physicien Richard Phillips Feynman tenait à ces élèves en ajoutant que lui aussi se trompait lors de ses travaux. Et de lire cela dans ces cours de physique m'a aussi beaucoup aidé.

Ces une première motivation pour introduire ce chapitre et le répété par la suite.

[en] When you take a course, listen to a reasoning, read equations or code, you have the impression that the professor or the experienced person making this presentation does not make a mistake. And it can be discouraging for you to commit. You do not feel up to it. And well know that they do it but they do not say it.

These are the same words that physicist Richard Phillips Feynman held to these students, adding that he too was mistaken in his work. And reading this in these physics courses also helped me a lot.

These a first motivation to introduce this chapter and repeated it afterwards.

[fr] Une autre motivation nous est donnée par Donald E. Knuth que je cite directement *"Une autre bonne pratique de débogue est de maintenir un registre des erreurs commises. Même si ceci est probablement un peu embarrassant, de telles informations sont inestimables pour toute personne réalisant des recherches sur le problème du débogue, et vous aiderons aussi à apprendre comment faire face aux futures erreurs."* [6]-§1.4.1-p64.

[en] Another motivation is given by Donald E. Knuth that I quote directly *"Another good debbuging pratice is to keep a record of every mistake made. Even though this will probably be quite embarrassing, such information is invaluable to anyome doing research on the debbuging problem, and it will also help you learn how to cope with future errors."* [6]-§1.4.1-p64.

[fr] Ligne 25 – En écrivant le code j'ai remplacé q par r dans le calcul de la valeur du chiffre de sortie. L'affichage n'était pas un chiffre mais un caractère de grande valeur.

[en] Line 25 – In writing the code I replaced q with r in the calculation of the value of the output digit. The display was not a number but a high value character.

Bogue/bug 1: `ADDU c,r,'0'` au lieu de / instead of `ADDU c,q,'0'` – Faute de frappe / Typing error

[fr] Ligne 33 – En écrivant l'algorithme, au début, je me suis dit qu'il se terminait lorsque le reste r est égale à zéro. Ceci est vroi pour l'exemple 1728, ou encore 1008, dans ce cas le reste n'est jamais nul, mais est faux pour les multiples de 10, comme 1720. L'effet est dans ce cas de ne pas produire les derniers zéros, 1720 s'affiche comme 172 et 1000 comme 1. La condition de terminaison est bien que le diviseur b atteigne zéro.

[en] Line 33 – Writing the algorithm at the beginning, I told myself that it ended when the remainder r is equal to zero. This is true for example 1728, or 1008, in which case the remainder is never null, but is false for multiples of 10, like 1720. The effect is in this case not to produce the last zeros, 1720 displays as 172 and 1000 as 1. The termination condition is that the divisor b reaches zero.

Bogue/bug 2: `BNZ r,0B` au lieu de / instead of `BNZ b,0B` – Erreur de raisonnement / Reasoning error

[fr] Ligne 15 – J'ai tapé pour le filtrage sur 16 bits $^{\#}$`fff` au lieu de $^{\#}$`ffff`. La valeur 1728 était alors afficher 896. En effet, $17280 = {}^{\#}4380$, $^{\#}4380$ & $^{\#}$`fff` $= {}^{\#}380$ et $^{\#}380 = 896$.

[en] Line 15 – I typed for 16-bit filtering $^{\#}$`fff` instead of $^{\#}$`ffff`. The value 1728 then display 896. Indeed, $17280 = {}^{\#}4380$, $^{\#}4380$ & $^{\#}$`fff` $= {}^{\#}380$ and $^{\#}380 = 896$.

Bogue/bug 3: `INCL a,n&#fff` au lieu de / instead of `INCL a,n&#ffff` – Faute de frappe / Typing error

19. Comment déboguer ? / Howto debug?

[fr] Comment trouver les erreurs précédentes ? Comment déboguer ?

Les trois grands pratiques sont :

- Lire et comprendre le code.
- Utiliser des traces d'exécution.
- Tester différents cas.

[en] How to find previous errors? How to debug?

The three main practices are:

- Read and understand the code.
- Use execution traces.
- Test different cases.

20. Lire et comprendre le code / Read and understand the code

[fr] Pour trouver un bogue, la première chose à faire est de relire son code et de l'exécuter mentalement sur la valeur d'entrée qui pose un problème. Vous pouvez vous aider d'une feuille de papier où l'on écrit les intermédiaires de calcul. Lorsque vous lisez un code, le vôtre ou celui d'autrui, vous devez comprendre ce qu'il fait. Insistez sur les parties difficiles à comprendre, lisez plusieurs fois, calculez de tête, à la main ou à l'aide d'une calculatrice en mode informatique pour les grands nombres.

Dans le cas de l'assembleur, lisez la définition des instructions, le manuel de l'assembleur. Dans le cas d'une bibliothèque, lisez la définition des routes, le manuel de la bibliothèque (si disponible) ; ou lisez le code (si disponible).

Si le code reste obscure, trop complexe, il doit être simplifié et commenté pour l'assembleur, jusqu'à devenir clair.

[en] To find a bug, the first thing to do is to re-read its code and run it mentally on the input value that is causing a problem. You can use a sheet of paper where you write the calculation intermediates. When reading a code, yours or that of others, you must understand what he does. Insist on difficult parts to understand, read several times, calculate from the head, by hand or using a calculator in computer mode for large numbers.

In the case of assembler, read the definition of the instructions, the assembly manual. In the case of a library, read the road definition, the library manual (if available); or read the code (if available).

If the code remains obscure, too complex, it must be simplified and commented on for the assembler, until it becomes clear.

21. Utiliser des traces d'exécution / Use execution traces

[fr] Une pratique classique du débogue consiste à ajouter des traces s'affichant à l'écran ou sortant dans un fichier pour mettre en évidence étapes et valeurs intermédiaire ; l'ajout de `printf()` dans le code C par exemple. Cette approche n'est pas sans effet de bord dans différent cas et est à manipuler avec précaution.

Pour le moment, nous en somme à peine à essayer d'affichier un entier... Donc ce n'est pas la bonne approche dans notre cas.

[en] A typical debugging practice is to add traces that appear onscreen or out into a file to highlight steps and intermediate values; adding `printf()` in the C code, for example. This approach is not without edge effect in different cases and is to be handled with caution.

For now, we hardly have to try to display an integer... So this is not the right approach in our case.

[fr] Une autre pratique est d'utiliser un débogueur. Celui-ci permet d'exécuter le code pas à pas et de voir les valeurs intermédiaires.

Ne le faites surtout pas. Celà fait 20 ans que je travaille sans débogueur parce que, comme dirait l'amiral Ackbar, "c'est un piège !".

En effet vous allez passer un temps trés important à exécuter votre programme pas à pas sans réfléchir pour voir les valeurs. Pour un résultat plus efficace, lisez votre code et faites les opérations mentalement. Le débogueur doit être réserver en dernier recours ou pour des cas difficils à apréhender.

Heureusement, nous n'avons pas de débogueur et nous devons donc appliquer les méthodes plus efficace : lire et tester !

[en] Another practice is to use a debugger. This allows you to execute the step-by-step code and see the intermediate values.

Do especially not do it. I've been working without a debugger for 20 years because, as Admiral Ackbar would say, "it's a trap!"

Indeed you will spend a very important time to execute your program step by step without thinking to see the values. For a more effective result, read your code and do the operations mentally. The debugger should be reserved as a last resort or for difficult cases to apprehend.

Fortunately, we do not have a debugger and so we must apply the most effective methods: read and test!

[fr] Question : comment ai-je fait pour ce premier programme ?

Réponse : j'ai lu et rélu, et, j'ai utiliser le mode verbeux du simulateur `MMIX` qui fournit des traces d'exécution.

[en] Question: How did I do this first program?

Answer: I read and reread, and, I use the verbose mode of the `MMIX` simulator that provides traces of execution.

[fr] Voila le résultat de l'exécution en mode verbeux. [en] Here is the result of execution in verbose mode .

```
1  $ mmix -v imprimer_A01
2  ...
3  line 22:                                        * A2 - Division euclidienne / Eucl
4  line 23:    OH    DIVU   q,a,b                    (q,r) <- a/b
5           1. 0000000000000124: 1e020001 (DIVU) $2=l[2] = #0fffffffffffffff / #8ac7230489e80000 = #1, rR=#7538
6    10 instructions, 0 mems, 69 oops; 0 good guesses, 0 bad
7  line 24:          GET    r,rR
8           1. 0000000000000128: fe030006 (GET) $3=l[3] = rR = #7538dcfb7617ffff
9    11 instructions, 0 mems, 70 oops; 0 good guesses, 0 bad
```

```
10  line 25:                                        * Imprimer q à l'écran / Print q
11  line 26:          ADDU   c,q,'0'                  c <- '0'+q <=> 'q'
12           1. 000000000000012c: 23040230 (ADDUI) $4=l[4] = #1 + 48 = #31
13    12 instructions, 0 mems, 71 oops; 0 good guesses, 0 bad
14  line 27:          STBU   c,ch_,0                  ch[0] <- 'q' <=> ch = 'q',0
15           1. 0000000000000130: a3040500 (STBUI) M1[#0] = #31, M8[#0]=#3100000000000000
16    13 instructions, 1 mem, 72 oops; 0 good guesses, 0 bad
17  line 28:          SET    $255,ch_                 $255 <- ch[0]
18           1. 0000000000000134: c1ff0500 (ORI) $255=g[255] = 0 = #0
19    14 instructions, 1 mem, 73 oops; 0 good guesses, 0 bad
20  1line 29:          TRAP   0,Fputs,StdOut
21           1. 0000000000000138: 00000701 (TRAP) $255 = Fputs(StdOut,#0) = 1
22    15 instructions, 1 mem, 78 oops; 0 good guesses, 0 bad
23  ...
```

Source 4: `imprimer_A01.trc` – Algorithme A01 - Traces d'exécution / Traces of execution.

22. Tester différents cas / Test different cases

[fr] Pour vérifier le bon fonctionnement d'un algorithme ou d'un programme, il est nécessaire de le tester sur différents cas. Ces cas sont choisis de manière à stimuler le programme dans ses différents comportements. Nous sélectionnons aussi des cas pour lesquels nous sommes capables de définir leurs valeurs d'entrée et valeurs de sortie.

La démarche de test doit être organisée et démarrer dès le début de l'écriture du code. Je recommande d'écrire les cas de test avant d'écrire le code, dans l'approche de *développement mené par les tests*. Pour bien appréhender la démarche de test, nous commencerons par les exemples des tests de nos codes, puis je la développerai plus en détail dans un autre chapitre.

[en] To check the correct operation of an algorithm or a program, it is necessary to test it on different cases. These cases are chosen in such a way as to stimulate the program in its different behaviors. We also select cases for which we are able to define their input values and output values.

The test procedure must be organized and started as soon as the code is written. I recommend writing the test cases before writing the code in the *test driven development* approach. To understand the test procedure, we begin with the examples of the tests of our codes and then I will develop it in more detail in another chapter.

[fr] Revenons à notre algorithme. Constatez simplement que dans ce premier programme, il est nécessaire de changer la valeur de `n` et de le recompiler pour tester différents cas. Pour le faire dans un seul programme nous devons intégrer notre algorithme dans une souroutine ; objet du chapitre suivant.

Pour aller plus loin, nous allons devoir développer des routines nous simplifiant grandement l'écriture des tests. Tout un programme !

[en] Let's go back to our algorithm. Just note that in this first program it is necessary to change the value of `n` and recompile it to test different cases. To do this in a single program we must integrate our algorithm in subroutine; Subject of the next chapter.

To go further, we will have to develop routines that greatly simplify the writing of the tests. What a program!

23. Introduction d'une souroutine / Introduction of a subroutine

[fr] Une *souroutine*[9] est un fragment d'instructions placé en un unique endroit du code et répondant à une fonction[10] particulière, utile en plusieurs autres endroits. Grâce à quelques instructions supplémentaires, l'exécution peut être transférée entre un programme principale et une souroutine, ce que l'on appelle la *ligature de souroutines*[11].

Sur le sujet, vous pourez lire avec intérêt le chapitre [6]-§1.4.1'-"Subroutines".

[en] A *subroutine* is a fragment of instructions placed in a single location of the code and responding to a particular function[12], useful in several other places. With some additional instructions, execution can be transferred between a main program and subroutine, known as *subroutine linkage*.

On the subject, you can read with interest the chapter [6]-§1.4.1'-"Subroutines".

[fr] Le fichier suivant met en oeuvre l'algorithme A dans une deuxième version A02 introduisant deux souroutines '`:A02:impr_déc:`' et '`vli:sys:es:impr_car:`'. Le fichier a été séparé en trois parties commentées pour en faciliter la lecture, le *programme principale* et les deux *souroutines*.

[en] The following file implements the algorithm A in a second version A02 introducing ttwohe subroutines '`:A02:impr_déc:`' and '`vli:sys:es:impr_car:`'. The file was separated into tree commented parts for ease of reading, the *main program* and the two *subroutines*.

```
1  * ------------------------------------------------------------------------
2  * Programme principal / Main program.
3  * ------------------------------------------------------------------------
4            LOC    #100
5            PREFIX :
6  n_max     IS     18446744073709551615  Le plus grand entier / The largest integer.
7  _         IS     $0                    Haut de contexte / Top of context
8  n         IS     $1
9                                      * Appels multiples à :A02:impr_déc(n)
10 Main      SET    n,0                   n <- 0
11           PUSHJ  _,:A02:impr_déc:_sr_
12           SET    n,9                   n <- 9
13           PUSHJ  _,:A02:impr_déc:_sr_
14           SET    n,29                  n <- 29
15           PUSHJ  _,:A02:impr_déc:_sr_
16           SET    n,1728                n <- 1728
17           PUSHJ  _,:A02:impr_déc:_sr_
18           SET    n,17280               n <- 17280
19           PUSHJ  _,:A02:impr_déc:_sr_
20           SETH   n,(n_max>>48)&#ffff   n <- n_max
21           INCMH  n,(n_max>>32)&#ffff
22           INCML  n,(n_max>>16)&#ffff
23           INCL   n,n_max&#ffff
24           PUSHJ  _,:A02:impr_déc:_sr_
25           SET    $255,0              * Arrête MMIX / Halt MMIX.
26           TRAP   0,Halt,0
27 * ------------------------------------------------------------------------
28 * Souroutine / Subroutine   : :A02:impr_déc
29 * Algotithme / Algorithm    : A02
```

[9] [fr] Néologisme : j'introduis le préfixe *sou*, souroutine, au lieu du préfix *sous-*, *sous-routine*.
[10] [fr] Nous distinguons ici le rôle, la *fonction*, de son implantation, la *souroutine*.
[11] [fr] Dictionnaire Littré - ligature : en général, manière de lier un objet quelconque.
[12] [en] We distinguish here the role, the *function*, from its implantation, the *subroutine*.

```
30  * Arguments / Arguments     :
31  *      n -> L'entier naturel / The natural integer.
32  *
33  * Description / Description :
34  * [FR] Imprime à l'écran en notation décimal l'entier naturel n passé
35  *      en paramètre. Vingt chiffres sont écrits suivis d'un retour à la ligne.
36  *      Par exemple 1728 s'affiche "00000000000000001728\n"
37  *
38  * [EN] Prints to the screen in decimal notation the natural integer n passed
39  *      in parameter. Twenty digits are written followed by a line break.
40  *      For example, 1728 is displayed "00000000000000001728\n"
41  * ------------------------------------------------------------------------------
42            PREFIX :A02:impr_déc:
43  * Arguments / Arguments.
44  n         IS     $0                    n -> L'entier naturel / The natural integer.
45
46  * Internes / Internals.
47  b0        IS     10000000000000000000  Le plus grand quotient / The largest quotient.
48  a         IS     $1                    Le dividende / The dividend.
49  b         IS     $2                    Le diviseur / The divisor.
50  q         IS     $3                    Le quotient / The quotient.
51  r         IS     $4                    Le reste / The remainder.
52  _rJ       IS     $5                    Sauvegarde de rJ / Saving of rJ
53  _         IS     $6                    Haut de contexte / Top of context
54  c         IS     $7                    Le caractère de sortie / The output character.
55
56                                         * A1 - Initialisation / Initialization.
57  _sr_      GET    _rJ,:rJ               Sauvegarde rJ / Saves of rJ
58            SET    a,n                   a <- n
59            SETH   b,(b0>>48)&#ffff      b <- b0
60            INCMH  b,(b0>>32)&#ffff
61            INCML  b,(b0>>16)&#ffff
62            INCL   b,b0&#ffff
63                                         * A2 - Division euclidienne / Euclidean division.
64  0H        DIVU   q,a,b                 (q,r) <- a/b
65            GET    r,:rR
66                                         * Imprimer q à l'écran / Print q to screen.
67            ADDU   c,q,'0'               c <- '0'+q <=> 'q'
68            PUSHJ  _,:vli:sys:es:impr_car:_sr_
69                                         * Itération suivante ? / Next iteration?
70            SET    a,r                   a <- r
71            DIVU   b,b,10                b <- b / 10
72            BNZ    b,0B                  si/if b != 0 => itérer / iterate.
73
74                                         * A3 - Terminaison / Terminaison.
75            SET    c,#a                  c <- '\n'
76            PUSHJ  _,:vli:sys:es:impr_car:_sr_
77            PUT    :rJ,_rJ               Restaure rJ / Restore rJ
78            POP    0,0                   Retourne sans rien / Returns with nothing
79            PREFIX :                     * Fin / The end.
80  * ------------------------------------------------------------------------------
81  * Souroutine / Subroutine   : :vli:sys:es:impr_car
82  * Arguments / Arguments     :
83  *      c -> Le caractère / The character.
84  * Description / Description :
```

```
85  *      Imprime un carcatère à l'écran / Prints a char to the screen
86  * ------------------------------------------------------------------------
87            PREFIX :vli:sys:es:impr_car:
88  * Arguments / Arguments.
89  c         IS     $0                   c -> Le caractère / The character.
90
91  * Internes / Internals.
92  ch        BYTE   'c',0                La chaîne de sortie / The output string.
93  ch_       IS     $1                   L'adresse de la chaîne / The address of the string.
94
95  _sr_      GETA   ch_,ch               ch_ <- @ch, ainsi/so ch_,i <=> ch[i]
96            STBU   c,ch_,0              ch[0] <- c
97            SET    $255,ch_             $255  <- ch_
98            TRAP   0,:Fputs,:StdOut
99            POP    0,0                  Retourne sans rien / Returns with nothing
100           PREFIX :
101 * ------------------------------------------------------------------------
```

Source 5: `imprimer_A02.mms` – Algorithme A02 - Imprimer en notation décimal / Print in decimal notation.

[fr] Voila le résultat de la compilation et de l'exécution. [en] Here is the result of compilation and execution.

```
$ mmixal -b 100 -l imprimer_A02.lst imprimer_A02.mms
$ mmix imprimer_A02
0000000000000000000
0000000000000000009
0000000000000000029
0000000000000001728
0000000000000017280
18446744073709551615
```

24. Ligne par ligne / Line by line

[fr] Lignes 5, 42, 54, 79, 87, 89 et 100 – Le nom symbolique `PREFIX` de `MMIXAL` permet de définir un *espace de noms* ou une *lignée*. Les lignées permettent d'éviter les conflits de symboles sur un grand programme ([6]-§1.4.1'-p61-62-"Assembly language features"). Par exemple, la variable `n` est définie pour le programme principale ligne 8, sous le préfixe global '`PREFIX :`', ce qui correspond à '`:n`', et, pour la souroutine ligne 44, sous le préfixe '`PREFIX :A02:impr_déc:`', ce qui correspond à '`:A02:impr_déc:n`'.

Ainsi nous pouvons utiliser le même symbole `n` comme paramètre du programme principale lors de l'appel à la souroutine et comme argument de la souroutine. Il en va de même pour la souroutine '`:vli:sys:es:impr_car:`' ligne 87 et les registres `c` ligne 54 et 89.

Les lignes 79 et 100 '`PREFIX :`', à la fin de souroutine, permet de revenir à la *lignée globale*.

[en] Lines 5, 42, 79, 87 and 100 – The symbolic name `PREFIX` of `MMIXAL` allows you to define *name space*. The namespace can prevent the symbols conflicts on large program ([6]-§1.4.1'-p61-62-"Assembly language features"). For example, the variable `n` is defined for the main program line 8, under the global prefix '`PREFIX :`', which corresponds to `:n`, and for the subroutine line 44, under the prefix '`PREFIX :A02:impr_déc:`', which corresponds to `:A02:impr_déc:n`'.

Lines 79 and 100 '`PREFIX :`', at the end of souroutines, allow to return to the *global name space*.

Thus we can use the same symbol `n` as a parameter of the main program when calling to the souroutine and as argument of the suroutine. It is the same for subroutine '`:vli:sys:es:impr_car:`' line 87 and registers `c` lines 54 and 89.

[fr] Lignes 7-8, 44-54 et 89-93 – Nous introduisons un symbole particulier '_' que nous appelons le *registre chapeau*. Il s'agit du registre qui coiffe les registres locaux utilisées.

Ligne 7, nous n'en n'avons aucun dans le programme principale, donc le chapeau est à terre '`_ IS $0`'.

Lignes 44 à 54, nous avons 6 registres locaux, l'argument `n`, les variables `a`, `b`, `c`, `q`, `r` de notre algorithme et le registre de sauvegarde `_rJ` sur lequel nous reviendrons. Le chapeau vaut donc `$6`.

Lignes 89 à 93, la souroutine '`:vli:sys:es:impr_car:`' n'en appelle aucune autre. Elle n'a pas besoin de chapeau.

Martin Ruckert, auteur de *The MMIX Supplements*[8], utilise le symbole `t` pour désigner un registre temporaire et pour ce même rôle de chapeau [8]-"Syle guide"-§2-"Temporaries". Il indique que si le symbole `t` est utilisé par ailleurs, on peut utiliser le symbole `x`. Comme en physique nous utilisons très souvent `t` et `x`, j'en suis venu à ma demander quel symbole prendre et j'ai testé '_' qui a fonctionné. L'intérêt, de ce symbole '_' est qu'il introduit une séparation visuelle entre les registres locaux devant être sauvegardés et le registres passé en paramètre.

Le code devient plus clair, et nous n'avons pas de question à nous poser sur le symbole à utiliser lors de l'appel : '`PUSHJ _,:A02:impr_déc:_sr_`'.

[en] Lines 7-8, 44-54 and 89-93 – We introduce a particular symbol '_' which we call the *hat register*. This is the register that covers the local registers used.

Line 7, we have none in the main program, so the hat is on the ground '`_ IS $0`'.

Lines 44 to 54, we have 6 local registers, the `n` argument, the `a`, `b`, `c`, `q`, `r` variables of our algorithm and the safeguard register `_rJ` to which we shall return. The hat is therefore `$6`.

Lines 89 to 93, the subroutine '`:vli:sys:es:impr_car:`' does not call any other. She does not need a hat.

Martin Ruckert, author of *The MMIX Supplements*, uses the symbol `t` to designate a temporary register and for this same role of hat register. It indicates that if the symbol `t` is used elsewhere, the symbol `x` can be used. As in physics we often use `t` and `x`, I came to ask what symbol to take and I tested '_' which worked. The interest of this symbol '_' is that it introduces a visual separation between the local registers to be saved and the registers passed as a parameter.

The code becomes more clear, and we do not have to question us on the symbol to use when calling : '`PUSHJ _,:A02:impr_déc:_sr_`'.

[fr] Lignes 8, 10-24, 54, 67-68, 75-76 et 89 – Ces lignes réalisent les appels de souroutines grâce à l'instruction `PUSHJ`. Par exemple ligne 17, l'instruction '`SET n,1728`' donne la valeur 1728 à la variable `n`, et à la ligne 18 l'appel est réalisé par l'instruction '`PUSHJ _,:A02:impr_déc:_sr_`'.

L'instruction `PUSHJ` réalise plusieurs actions. Elle enregistre l'adresse de retour d'appel dans le registre spécial `rJ`. Elle sauvegarde sur la pile de registres les registres locaux situés sous le chapeau '_', et la valeur du chapeau, ici 0 pour '`$0`', 6 pour '`$6`'. Elle réalise un saut à l'adresse de la souroutine '`:A02:impr_déc:_sr_`'.

La sauvegarde des registres locaux provoque la renumérotation des registres se trouvant au dessus du chapeau en `$0`, `$1`, ..., `$N`. Ainsi la variable ligne 8 '`n IS $1`' de l'espace globale, devient après appel l'argument ligne 44 '`n IS $0`' de la souroutine '`:A02:impr_déc:`'.

De même, dans le cas de l'appel '`PUSHJ _,:vli:sys:es:impr_car:_sr_`', avec le chapeau '`_ IS $6`' ligne 53, la variable ligne 54 '`c IS $8`' de la souroutine '`:A02:impr_déc:`' devient après appel l'argument ligne 89 '`c IS $0`' de la souroutine '`:vli:sys:es:impr_car:`'.

[en] Lines 8, 10-24, 54, 67-68, 75-76 and 89 – These lines perform subroutine calls with the instruction `PUSHJ`. For example, line 17, the instruction '`SET n, 1728`' gives the value 1728 to the variable `n`, and to

the line 18 the call is made by the instruction 'PUSHJ _,:A02:impr_déc:_sr_'.

The PUSHJ instruction performs several actions. It registers the call return address in the special register rJ. It saves on the stack of registers the local registers under the hat '_', and the value of the hat, here 0 for '$0', 6 for '$6'. It performs a jump to the address of the address ':A02:impr_déc:_sr_'.

Saving local registers causes the registers above the hat to be renumbered as $0, $1, ..., $N. Thus the variable line 8 'n IS $1' of the global space, becomes after call the argument line 44 n IS $0 'of the subroutine ':A02:impr_déc:'.

Similarly, in the case of the call PUSHJ _,:vli:sys:es:impr_car:_sr_', with the hat '_ IS $6' line 53, the variable line 54 'c IS $8' of the subroutine ':A02:impr_déc:' becomes after the call the argument line 8 'c IS $0'9 of the subroutine ':A02:impr_déc:'.

[fr] Lignes 57 et 95 – Notez que j'utilise l'étiquette '_sr_' pour définir le point d'entrée de la souroutine. En fait, l'utilisation du nom de la souroutine comme étiquette provoquait un redondance pour l'appel ':A02:impr_déc:impr_déc' et la suppression de cette redondance, l'usage de 'PREFIX :A02:' au lieu de 'PREFIX :A02:impr_déc:', était potentiellement source de conflit de symbole pour les lignées contenant plusieurs souroutines. De plus, il est possible avec MMIX d'avoir des souroutines à point d'entrée multiples. Avec la notation '_sr_', je peux envisager '_sr0_', '_sr0_', ..., '_srN_' par exemple.

[en] Lines 57 and 95 – Note that I use the label '_sr_' to define the entry point of the subroutine. In fact, using the name of the subroutine as a label would cause redundancy for the call ':A02:impr_déc:impr_déc' and the removal of this redundancy, the use of PREFIX :A02:' instead 'PREFIX :A02:impr_déc:', was potentially a source of symbols conflict for lines containing several subroutines. In addition, it is possible with MMIX to have multiple entry-point subroutines. With the notation '_sr_', I can consider '_sr0_', '_sr0_', ..., '_srN_' for example.

[fr] Lignes 78 et 99 – Maintenant, puisque l'instruction PUSHJ réalise un saut à l'adresse de la souroutine, il est nécessaire à la fin de son exécution de revenir à l'instruction suivant l'appel en restaurant les registres locaux sauvegardés sur la pile de registres. C'est ce que réalise l'instruction POP 0,0 lignes 78 et 99. Le premier paramètre de cette instruction est le nombre de registre contenant des valeurs de retour ; dans notre cas zéro. Le second paramètre est un décalage par rapport de l'adresse de retour sauvegardé dans le registre global rJ.

L'instruction POP réalise les actions suivantes. Elle lit sur le registre de pile, virutellement en '$-1', la valeur du chapeau pour déterminer le nombre de regsitre à restaurer. Elle restore les registres locaux situés de ce fait sous le chapeau '_' de la routine appelante. Elle réalise un saut à l'adresse de retour d'appel contenue dans le registre spécial rJ additionnée du décalage.

[en] Lines 78 and 99 – Now, since the PUSHJ instruction makes a jump to the surreal address, it is necessary at the end of its execution to return to the instruction following the call by restoring the local registers saved on the stack of registers. This is done by the instruction POP 0,0 lines 78 and 99. The first parameter of this instruction is the number of registers containing return values; In our case zero. The second parameter is an offset relative to the return address saved in the global register rJ.

The POP statement performs the following actions. It reads on the stack register, virutelly in '$-1', the value of the hat to determine the number of regsitre to restore. It restores the local registers located under the '_' hat of the calling routine. It makes a jump to the call return address contained in the special register rJ added with the offset.

[fr] Lignes 52, 57 et 77 – Vous constatez que lorsqu'une souroutine appel une autre souroutine, le même registre spécial rJ est utilisé pour sauvegarder l'adresse de retour. De ce fait, tout souroutine en appelant d'autres doit sauvegarder ce registre avant tout appel et le restaurer après. Nous réalisons ceci en déclarant un registre local de sauvegarde juste en dessous du chapeau, ligne 52 '_rJ IS $5'. La sauvegarde est

faite par l'instruction de lecture d'un registre spécial, ligne 57 '`GET _rJ,:rJ`'. La restauration est faite par l'instruction d'écriture d'un registre spécial, ligne 77 '`PUT :rJ,_rJ `'.

[en] Lignes 52, 65 et 77 – You notice that when a subroutine calls another subroutine, the same special register `rJ` is used to save the return address. Therefore, any subroutine calling others must save this registry before any call and restore it afterwards. We do this by declaring a local backup register just below the hat, line 52 '`_rJ IS $ 6`'. The backup is done by the read instruction of a special register, line 57 '`GET _rJ,:rJ`'. The restoration is done by the write instruction of a special register, line 77 '`PUT :rJ,_rJ`'.

[fr] Voici l'illustration de ce que nous venons de décrire. Chaque triplet de colonnes correspond à l'état de la pile de registres et du registre `rJ` pour la ligne de code mentionnée. La première colonne donne le registre local ou spécial. La deusième colonne donne le symbole local. La troisième colonne précise les valeurs intéressantes ou une étoile pour les valeurs non significative pour notre explication.

La souroutine '`:A02:impr_déc:_sr_`' est nommée sr1. La souroutine '`:vli:sys:es:impr_car:_sr_`' est nommée sr2.

Il n'y a pas de registre à numérotation négative '`$-1`' à '`$-8`' en `MMIX`. Cependant en étendant la notion de numérotation des registres locaux, ceci permet d'indiquer leur présence et position dans la pile.

[en] Here is the illustration of what we have just described. Each triple of columns corresponds to the state of the stack of registers and the register `rJ` for the mentioned line of code. The first column gives the local or special register. The second column gives the local symbol. The third column specifies the interesting values or a star for the values not significant for our explanation.

The subroutine '`:A02:impr_déc:_sr_`' is manned sr1. The subroutine '`:vli:sys:es:impr_car:_sr_`' is manned sr2.

There is no negative numbering register '`$-1`' to '`$-8`' in `MMIX`. However, by extending the notion of numbering of the local registers, this makes it possible to indicate their presence and position in the stack.

Ligne 17			Ligne 58			Ligne 76			Ligne 96			Ligne 78			Ligne 18		
$10			$9			$9			$2			$9			$10		
$9			$8			$8			$1	ch_	@ch	$8			$9		
$8			$7	c	*	$7	c	#a	$0	c	#a	$7	c	#a	$8		
$7			$6	_	*	$6	_	*	$-1	_	6	$6	_	*	$7		
$6			$5	_rJ	sr1	$5	_rJ	sr1	$-2	_rJ	sr1	$5	_rJ	sr1	$6		
$5			$4	r	*	$4	r	*	$-3	r	*	$4	r	*	$5		
$4			$3	q	*	$3	q	*	$-4	q	*	$3	q	*	$4		
$3			$2	b	*	$2	b	*	$-5	b	*	$2	b	*	$3		
$2			$1	a	*	$1	a	*	$-6	a	*	$1	a	*	$2		
$1	n	1728	$0	n	1728	$0	n	1728	$-7	n	1728	$0	n	1728	$1	n	17280
$0	_	*	$-1	_	0	$-1	_	0	$-8	_	0	$-1	_	0	$0	_	*
rJ		*	rJ		sr1	rJ		sr1	rJ		sr2	rJ		sr1	rJ		sr1

Table 3: Algorithme A02 - Utilisation de la pile de registres / Use of stack registers.

25. Souroutine, pile de registres `MMIX` et paramètres de retour / Subroutine, register stack `MMIX` and return parameters

[fr] Suite au prochaine épisode. [en] Following the next episode.

26. Liste des codes source / List of source codes

27. Liste erreurs et bogues commis / List of errors and bugs committed

28. Licence Ouverte / Open Licence

Version 1.0.2 du dimanche 15 mai 2016.

[fr] La version française de la licence prévaut sur sa traduction en anglais.

Les présents documents et codes sont régis par le droit d'auteur français.

Les présents documents et codes sont distribués SANS AUCUNE GARANTIE, explicite ou implicite.

Autorisation vous est donnée de faire et de distribuer des copies conformes de ces documents et de ces codes à condition que la mention de droit d'auteur et l'avis d'autorisation soient conservés sur toutes les copies.

Autorisation vous est donnée d'utiliser ces documents et ces codes dans vos logiciels à condition de mentionner une telle utilisation et son avis d'autorisation, au sein des documents et des licences de vos logiciels.

Autorisation vous est donnée de copier et de distribuer des versions modifiées de l'ensemble ou d'une partie de ces documents et de ces codes à condition que le travail résultant, pour ce qui est de l'emprunt, bibliothèque ou programme ou partie, reçoive un nom différent et soit distribué selon des termes d'autorisation identiques à celui-ci.

En particulier, le nom et le logo *Vers l'Infini*, marque déposée en France par Benoît J. B. D. Auguet, ne doivent pas être utilisés et reproduits dans une version modifiée, à l'exception des références aux versions originales. L'acronyme *vli* du nom *Vers l'Infini*, utilisé pour l'espace des noms ou comme préfixe de noms des présentes bibliothèques ou des présents programmes, doit être changé dans une version modifiée.

Les droits d'auteur étant inaliénables, leurs mentions doivent être maintenues dans les versions modifiées et complétées de ceux des contributeurs. Un marquage peut être utilisée dans les versions modifiées pour faire apparaître les modifications par rapport à la version originale et les contributions de chacun.

Obligation vous est donnée de mentionner dans les versions modifiées, la version originale utilisée comme point de départ. En complément, la version originale peut être distribuée avec la version modifiée pour permettre aux utilisateurs d'avoir accès à l'historique.

Envoyer, s'il vous plaît, vos commentaires, suggestions et caetera à benoit.auguet@verslinfini.com.

Version 1.0.2 of Sunday, May 15, 2016.

[en] The French version of the license shall prevail over the English translation.

These documents and codes are governed by the French author's rights.

These documents and codes are distributed WITHOUT ANY WARRANTY, express or implied.

Permission is granted to make and distribute verbatim copies of these documents and these codes provided that the author's rights notice and the permission notice are preserved on all copies.

Permission is granted to use these documents and these codes in your software provided that you mention such use and its permission notice within documents and licenses of your software.

Permission is granted to copy and distribute modified versions of all or part of the documents and codes provided that the resulting work, regarding of the borrowing, library or program or part, is given a different name and distributed under the terms of a permission notice identical to this one.

In particular, the name and logo *Vers l'Infini*, registered trademark in France by Benoît J. B. D. Auguet, should not be used and reproduced in a modified version, with the exception of references to the original versions. The acronym *vli* of the name *Vers l'Infini*, used to for the namespace or as the prefix of names of these libraries or these programs must be changed in a modified version.

The author's rights being inalienable, their notices must be maintained in modified versions and complemented with those of the contributors. A marking may be used in the modified versions to show the changes compared to the original version and the contributions of each.

Obligation is given to state in modified versions, the original version used as a starting point. In addition, the original version can be distributed with the modified version to allow users to access them.

Please send comments, suggestions et caetera to benoit.auguet@verslinfini.com.

Index