J3eA, Journal sur l'enseignement des sciences et technologies de l'information et des systèmes
J3eA 24 (2025) 8
https://doi.org/10.1051/j3ea/20250008
https://www.j3ea.org

ArchiLaSimu : Un émulateur d'Architecture Von Neumann pour La Simulation*

Mikhaël Myara[1,3*], Arnaud Virazel[2,3]

[1] Université de Montpellier - CNRS, IES (UMR 5214), Montpellier, France
[2] Université de Montpellier - CNRS, LIRMM (UMR 5506), Montpellier, France
[3] Université de Montpellier, Département EEA, Montpellier, France

*Auteur de correspondance : mikhael.myara@umontpellier.fr

Résumé : *Dans les années 1940, le mathématicien et physicien John Von Neumann a formulé les principes fondamentaux des architectures d'ordinateurs, des concepts qui restent toujours pertinents aujourd'hui. Ce modèle d'architecture constitue un environnement pédagogique idéal pour l'enseignement de l'informatique industrielle. Dans ce contexte, nous avons développé un émulateur visant à rendre l'architecture informatique accessible de manière simple et pédagogique. L'émulateur ArchiLaSimu se présente sous la forme d'une interface web et permet aux utilisateurs de coder des instructions en langage assembleur utilisant une mnémonique simplifiée, de réaliser la micro-programmation d'un jeu d'instructions, tout en manipulant des données au format décimal.*

Mots clés : architecture Von Neumann, simulateur, programmation, micro-programmation, micro-code.

Abstract: **A Von Neumann Architecture Emulator for Simulation**
In the 1940s, mathematician and physicist John Von Neumann formulated the fundamental principles of computer architectures—concepts that remain relevant to this day. This architectural model provides an ideal educational environment for teaching industrial computing. In this context, we have developed an emulator designed to make computer architecture accessible in a simple and educational way. The ArchiLaSimu emulator is presented as a web interface that allows users to write instructions in assembly language using simplified mnemonics, to perform

*****

microprogramming of an instruction set, and to manipulate data in decimal format.

Keywords: Von Neumann architecture, simulator, programming, microprogramming, microcode.

1. Introduction

L'architecture de Von Neumann, du nom de John Von Neumann, un mathématicien et physicien américano-hongrois, a été formulée dans les années 1940. Elle repose sur trois éléments fondamentaux : le processeur, la mémoire et les périphériques d'entrée/sortie. Ce modèle est considéré comme la base des architectures d'ordinateurs modernes [1], [2].

Les principes de l'architecture de Von Neumann sont essentiels dans l'enseignement de l'informatique industrielle, notamment pour la mise en œuvre des compétences liées à la logique combinatoire et séquentielle ainsi qu'à la programmation. En effet, cette architecture offre un cadre pédagogique complet pour explorer la gestion de la mémoire, la structure des processeurs et l'interaction avec les périphériques d'entrée/sortie, qui sont des concepts cruciaux dans la conception des systèmes informatiques.

Dans ce contexte, l'état de l'art est riche en solutions allant des plus simples aux plus complexes. On peut retrouver des simulateurs d'architecture de type Von Neumann, tels que celui développé par Peter L. Higginson [3], jusqu'à des solutions d'émulation d'architectures réelles de processeurs, comme QEMU [4]. Ces simulateurs varient considérablement en termes de fonctionnalités et de complexité, en fonction des objectifs visés. Certains, comme celui de Higginson, se concentrent sur l'enseignement des principes fondamentaux de l'architecture des ordinateurs, tandis que d'autres, comme QEMU, offrent une émulation plus avancée permettant de simuler des architectures matérielles modernes.

Notre objectif est de proposer un outil de simulation intuitif et accessible, permettant d'aborder l'architecture informatique de manière simple et pédagogique.

Contrairement aux solutions existantes, notre simulateur n'a pas vocation à simuler un processeur existant. Il repose sur une mnémonique assembleur simplifiée, offrant ainsi une prise en main plus rapide et intuitive pour les étudiants. De plus, il n'affiche que des données en base décimale, une approche qui améliore la clarté et la compréhension visuelle de l'exécution des instructions. Enfin, il offre la possibilité pour l'étudiant de micro coder le jeu d'instructions du processeur, permettant ainsi une compréhension plus profonde dans la logique interne de l'architecture et une meilleure compréhension du décodage des instructions.

Une première version de l'émulateur d'architecture *ArchiLaSimu*, a été développée en 2010 dans le cadre de projets étudiants en Licence EEA à l'Université de Montpellier. Cette première version, réalisée en Visual Basic, comportait plusieurs limitations. Parmi celles-ci, un processus d'installation complexe nécessitant l'ajout de fichiers DLL spécifiques, ainsi que l'absence de gestion des incohérences potentielles dans les demandes de programmation et de micro-programmation des étudiants. Cela entraînait fréquemment des boucles infinies dans l'outil d'émulation, avec pour conséquence la perte de tout travail en cours. La nouvelle version d'*ArchiLaSimu* présentée dans cet article offre une utilisation plus stable dans un environnement web plus convivial et multiplateforme (Windows, MacOS, Linux)[5]. Elle intègre également des processus de vérification améliorés, permettant de prévenir les plantages pendant l'exécution.

La suite de cet article est organisée de la manière suivante. La Section 2 présente l'architecture utilisée ainsi que son fonctionnement. La Section 3 détaille l'implémentation informatique de l'émulateur *ArchiLaSimu*. Enfin, la Section 4 illustre des exemples d'application et d'utilisation de l'outil dans le cadre de travaux pratiques au sein d'unités d'enseignement relevant de l'informatique industrielle.

2. Architecture utilisée

Dans cette section, nous détaillons l'organisation de l'architecture implémentée dans *ArchiLaSimu*, ainsi que des exemples d'instructions.

2.1. Organisation de l'architecture

L'architecture utilisée s'inspire des principes de Von Neumann tout en restant aussi simplifiée que possible. La Figure 1 illustre cette architecture.

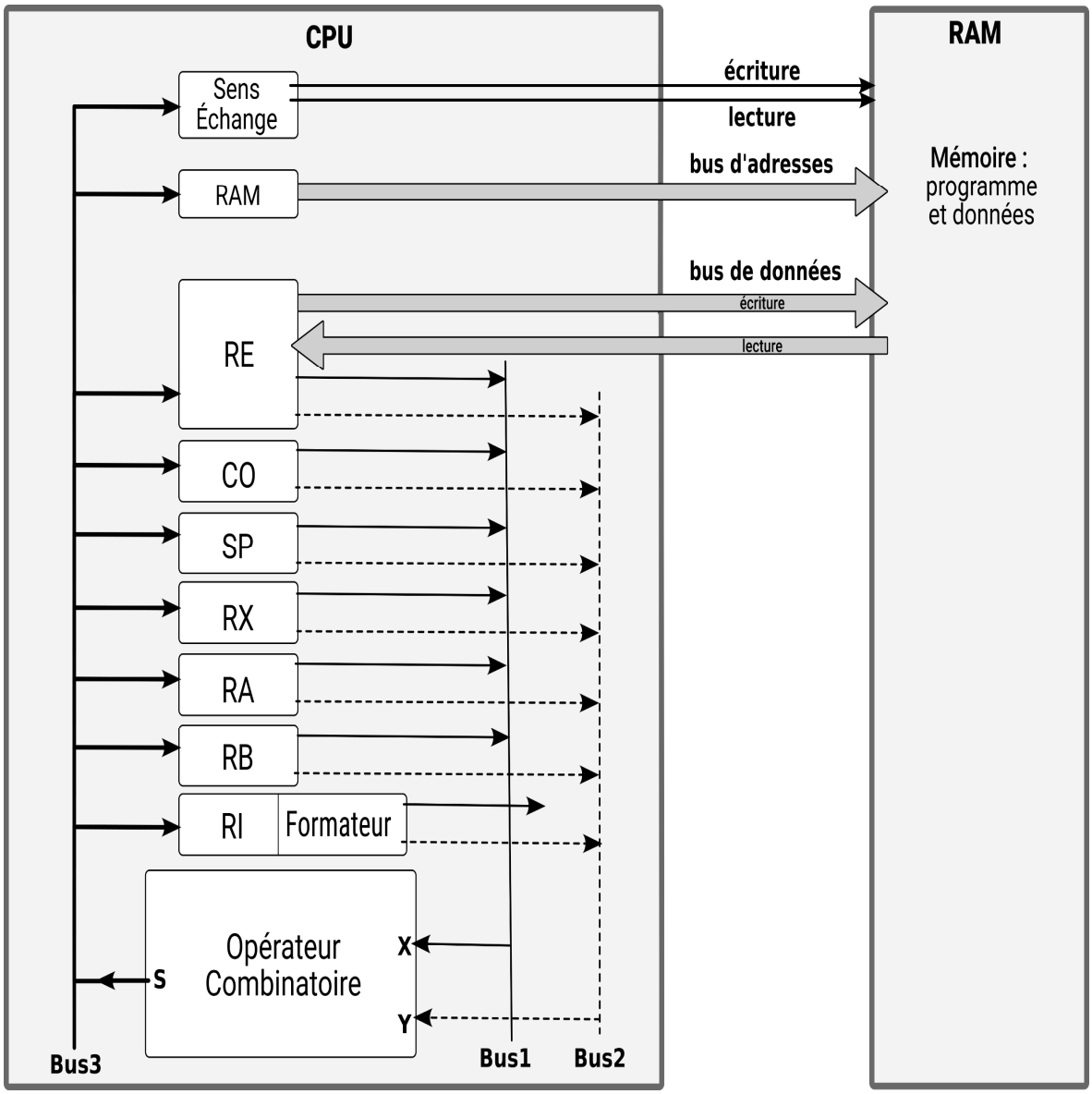

Figure 1. Organisation de l'architecture considérée

Dans la zone CPU, une série de registres est utilisée, avec les rôles suivants :

- **RI** : Registre d'Instruction, qui stocke l'instruction en cours d'exécution.
- **CO** : Compteur Ordinal, qui conserve l'adresse mémoire de l'instruction en cours d'exécution.
- **RA** et **RB** : Registres A et B manipulables par les instructions.
- **RAM** et **RE** : Le Registre d'Adresse Mémoire et le Registre d'Echange, permettent d'effectuer des opérations de lecture ou d'écriture.
- **RX** : Registre indeX, utilisé pour les opérations indexées (adressage des tableaux).
- **SP** : Registre de Pile, qui gère l'adresse de la pile en mémoire.

Un **opérateur combinatoire** complète l'architecture et permet l'exécution de calculs entre un ou deux registres via les bus Bus1 et Bus2. La partie RAM de la Fig. 1 représente la mémoire externe, à la fois pour les programmes et les données. Il est important de noter que la taille de ces éléments n'a pas besoin d'être définie précisément. Enfin, pour faciliter la compréhension, toutes les données et adresses manipulées sont représentées en décimal plutôt qu'en binaire/hexadecimal. Afin d'orchestrer l'ensemble de l'architecture, un décodeur d'instructions est utilisé. Nous avons opté pour un format d'instruction structuré en trois éléments COP, MA et RA définis comme suit :

- **COP** (Code Opération) : Il définit l'objectif principal de l'instruction, c'est-à-dire l'opération à réaliser (par exemple, LOAD, STORE, JUMP, etc.).
- **MA** (Mode d'Adressage) : Il spécifie le type d'accès à la mémoire, en précisant si l'adresse est immédiate, directe, indirecte, relative ou indexée.
- **RA** (Référence Adresse) : Il indique la valeur ou l'adresse à manipuler, en fonction du mode d'adressage spécifié par le Mode d'Adressage.

Afin de décoder l'instruction, nous avons adopté le principe de décodage CISC (Complex Instruction Set Computer), dans lequel le comportement du décodeur est défini par le contenu d'une mémoire [6]. Ce modèle est basé sur la microprogrammation, où l'ensemble des phases ou micro-codes décrivant le déroulement séquentiel d'une instruction est stocké dans une mémoire dédiée. Cette mémoire contient le jeu d'instructions de l'architecture.

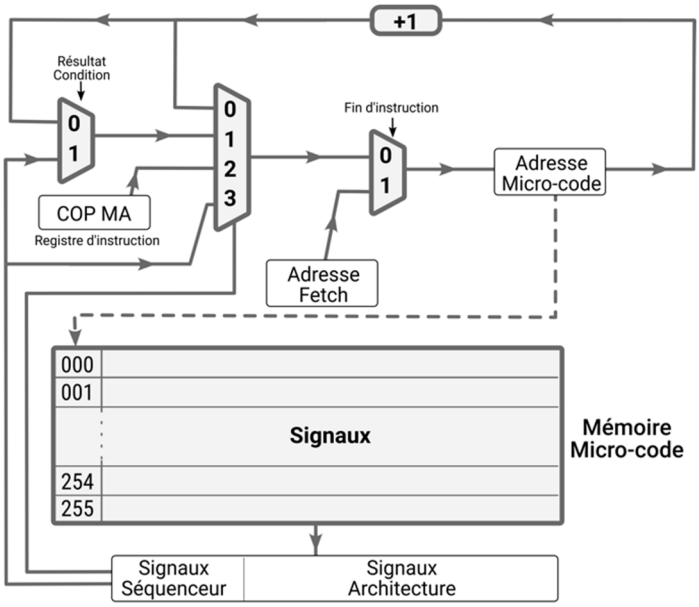

Figure 2. Principes du décodage d'instructions utilisé

La Figure 2 présente le schéma de principe du décodeur d'instruction utilisé. La mémoire contient les micro-codes associés au jeu d'instructions. Le registre Adresse Micro-Code est responsable de la sélection, pour chaque phase d'exécution de l'instruction, de l'adresse du micro-code à appliquer à l'architecture. Un premier multiplexeur, situé en amont de ce registre, permet de sélectionner l'adresse correspondant à l'étape de Fetch si l'instruction précédente est terminée. Dans le cas contraire, un second multiplexeur choisit l'adresse du micro-code suivant à appliquer. Il est important de noter que le registre RI (COP MA) sert à déterminer l'adresse du premier micro-code lié à l'instruction en cours. Enfin, un troisième multiplexeur, dont la commande dépend du résultat d'un test conditionnel (Résultat Condition), génère l'adresse du micro-code à appliquer en fonction de la validité de la condition sélectionnée. Le registre de sortie applique le micro-code (*Signaux Architecture*) à l'architecture et commande les multiplexeurs (*Signaux Séquenceur*) pour la recherche du micro-code suivant.

2.2. Cycles de fonctionnement

Chaque instruction se déroule en trois étapes :

- Étape 1 : Le cycle de recherche d'instruction (ou cycle Fetch) permettant de charger l'instruction dans le registre RI.
- Étape 2 : L'exécution des opérations associées à l'instruction.
- Étape 3 : L'incrémentation du registre CO pour pointer vers l'instruction suivante en mémoire.

Sur la base des signaux détaillés dans l'architecture présentée à la Figure 1, le cycle Fetch correspond à l'application successive des micro-codes suivants :

```
Cycle Fetch
COB1 XS eRAM            Charger RAM avec CO
sM                      Lire la mémoire à l'adresse CO
REB1 XS eRI             Déplacer l'instruction dans RI
```

À titre d'exemple, nous détaillons ci-dessous les micro-codes de quelques instructions :

```
LOAD A, Immédiat, RA (Registre A dans RA)
RIB1 XS eRA             Charger RA dans le registre A
COB1 XP1 eCO            Incrémenter CO
STORE B, Direct, RA (Registre B → Adresse RA)
RIB1 XS eRAM            Charger RA dans RAM
RB1 XS eRE              Charger le registre B dans RE
eM                      Ecrire RE à l'adresse RAM
COB1 XP1 eCO            Incrémenter CO
JUMP Direct, RA (Prochaine instruction à l'adresse RA)
RIB1 XS eCO             Charger RA dans CO
JUMP A>0, Relatif, RA (Prochaine instruction à l'adresse CO+RA si A>0,
sinon prochaine instruction à l'adresse CO+1)
si A>0
COB1 RIB2 ADDX eCO      Charger CO avec CO + RA
sinon
COB1 XP1 eCO            Incrémenter CO
```

3. Implantation informatique

3.1. Présentation de l'outil

Le logiciel a deux objectifs principaux :

- d'une part de faire travailler les étudiants sur la compréhension de l'architecture, notamment la fonction remplie par chaque registre, ainsi le séquençage des signaux pour aboutir à la description d'une instruction,
- d'autre part comment cela est utilisé pour écrire du code au niveau assembleur et langage machine.

Pour cela, les étudiants auront typiquement deux étapes à réaliser :

- Remplir la table de micro-codes, c'est à dire décrire la séquence des signaux qui permettent de remplir la fonctionnalité attendue, et ce pour quelques instructions assembleur,
- Écrire un programme assembleur qui exploite les instructions implémentées à l'étape 1

Le logiciel a été écrit pour répondre à cette démarche pédagogique. La Figure 3 présente l'interface principale qui concentre à elle seule l'essentiel des besoins.

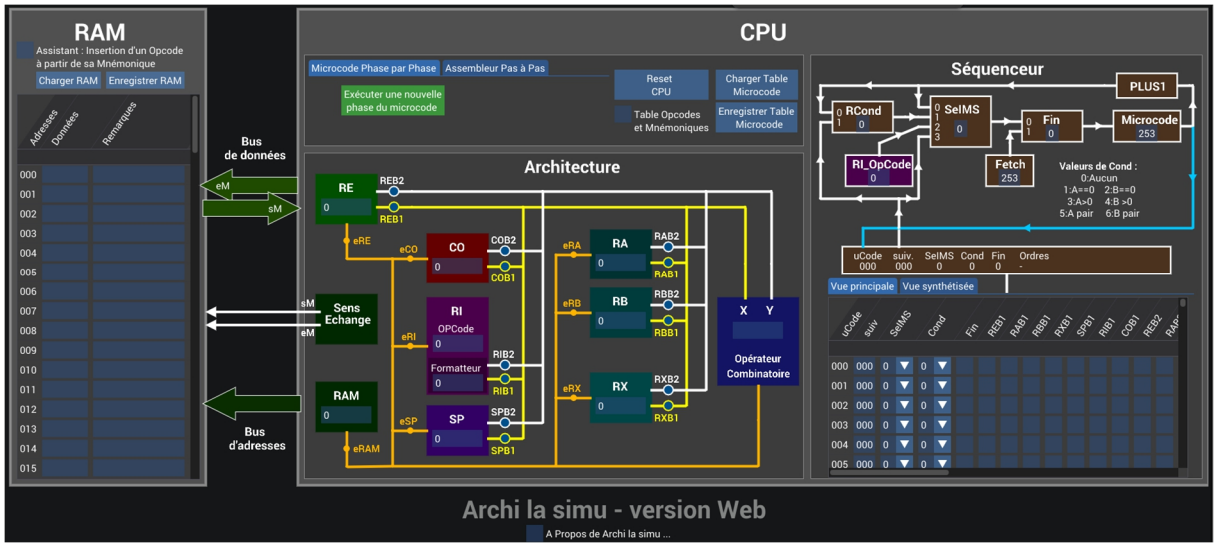

Figure 3. Copie d'écran du logiciel

Cette interface comporte deux blocs principaux. D'abord, **le bloc « RAM »** : c'est un tableau éditable de 128 adresses, chaque adresse pouvant contenir une valeur, associée à un commentaire. Ce commentaire n'est pas utilisé par le simulateur, il est juste là à des fins de documentation, et peut être modifié librement. Chaque adresse peut être remplie soit en indiquant directement la valeur, soit en utilisant un assistant (voir Figure 4) capable de générer le code opération d'une instruction à partir de son écriture en mnémonique. Lorsque l'assistant génère un code-opération dans une adresse de la RAM, le commentaire associé est remplacé pour contenir l'instruction assembleur qui correspond.

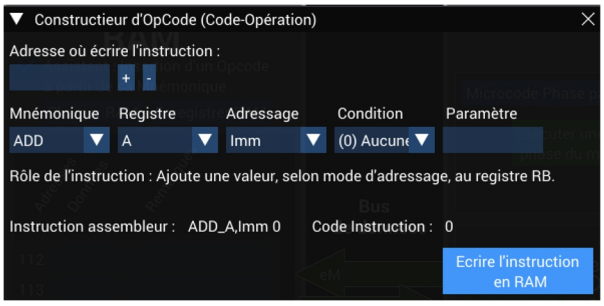

Figure 4. Constructeur de code d'opération

Le bloc « CPU » : il permet d'interagir avec les éléments constitutifs du CPU. Il est composé de 3 blocs principaux :

- « Architecture » : ce bloc sert à donner une représentation visuelle ainsi que l'état de l'architecture simulée. Pour cela, il contient une représentation de chaque registre avec sa valeur, ainsi qu'un opérateur combinatoire déjà fonctionnel et les noms des différents signaux que l'on va pouvoir stimuler lors de l'écriture du microcode.
- « Séquenceur » : ce bloc contient la table de micro-codes elle-même, ainsi qu'un ensemble d'éléments qui permettent de gérer le séquencement des phases de la table de microcode.
- Un dernier bloc permet l'exécution pas à pas, avec 2 modes de fonctionnement : soit avec des pas de la taille d'une instruction dans la table de microcode, soit avec des pas de la taille d'une instruction assembleur. Lors de l'exécution pas à pas, des animations permettent de visualiser quels signaux sont stimulés et où ils aboutissent.

Au-delà de l'aspect visuel et de l'exécution pas à pas, le simulateur permet à tout moment de modifier les valeurs des éléments (registres, multiplexeurs, ...) sans repasser par de l'exécution de micro-code. Ce nouvel état servira de base à la prochaine exécution pas à pas. Cela permet de réaliser des tests rapidement, et de corriger localement un bug "pour voir", sans avoir à forcément repartir de la situation initiale.

En complément, le logiciel comporte quelques utilitaires de base, tels que :

- Un tableau qui donne, pour chaque instruction assembleur et son mode d'adressage, son point d'entrée dans la table de microcode
- Une documentation succincte,
- De quoi enregistrer/recharger l'état de la RAM et/ou l'état de la table du microcode.

3.2. Choix liés au développement du logiciel

Le logiciel ne nécessite aucune installation : c'est une application web fonctionnant en ligne, afin de rendre le logiciel facilement disponible pour tout étudiant. Il doit idéalement fonctionner sur toutes les plateformes existantes. Le logiciel nécessite de nombreuses interactions avec l'utilisateur, et compte tenu de ce cahier des charges on pense naturellement à Javascript comme langage pour ce développement. Mais, bien que très utilisé en développement Web, Javascript est connu pour être un langage problématique, notamment à cause de sa gestion des types et de leur coercition[7], qui présente une grande source de bugs. Parallèlement, C/C++ sont des langages assez naturels pour les enseignants en EEA, pour des raison "métier" (développement sur microcontrôleurs).

Or depuis 2017 il est possible de développer en C/C++ des applications web multiplateformes, grâce un système type « machine virtuelle » embarqué dans les navigateurs web. Pour cela, un langage assembleur virtuel a été créé, nommé « webassembly »[8]. Ce système présente de très nombreux avantages : robustesse et outils puissants de C/C++, et grande vitesse d'exécution du code si cela s'avérait nécessaire. Au-delà du langage, il fallait une bibliothèque graphique fonctionnant correctement avec Webassembly dans le contexte du navigateur. Il existe à ce jour une bibliothèque multiplateforme très répandue, qui sert surtout à réaliser des interfaces pour des jeux video ou plug-in sur ordinateur. Il s'agit de « Dear ImGUI »[9], téléchargeable sur github. Elle est simple à prendre en main, réactive, dispose de beaucoup d'exemples, et dispose de très nombreux éléments d'interface graphique très souples. L'approche est en revanche un peu différente d'un développement habituel d'interface graphique sur ordinateur : l'aspect « événements » est moins marqué, il s'agit surtout de générer à chaque étape la prochaine image qui devra être affichée à l'écran, avec un certain « framerate », comme on le ferait pour un jeu vidéo. L'inconvénient est que cela prend un peu plus de ressources que pour une application graphique classique (mais rien d'énorme), l'avantage est une interface très réactive aux actions de l'utilisateur.

Le logiciel fonctionne sur toutes les plateformes "ordinateur" (PC/Windows, MacOs, Linux), a été testé sur de nombreux navigateurs (Firefox, Chrome, Safari ….) sans aucun problème et sans qu'aucune adaptation n'ait été nécessaire. Il se lance sur téléphone mobile et tablette mais ne permet pas d'interagir, les « claviers virtuels » de ces appareils n'étant pas gérés en l'état.

Il est utilisable par n'importe qui, sans login ou mot de passe, qui se connecterait à l'adresse[5] : https://dl.eea-fds.umontpellier.fr/ArchiLaSimu/. Il est également disponible en code source (licence GPLv3) et en binaire sur https://github.com [12] pour quiconque voudrait le modifier et/ou l'héberger via ses propres serveurs.

Le logiciel s'exécute, via le navigateur, sur la machine locale et n'utilise aucune ressource de la part d'un serveur (excepté le téléchargement du logiciel au lancement, qui fait 1.1 Mo).

4. Mise en application

Cette nouvelle version web d'*ArchLaSimu* est utilisée depuis l'année universitaire 2024/2025 dans le cadre d'unités d'enseignement en L1 Informatique et L3 EEA.

En première année de Licence informatique, l'objectif principal est de sensibiliser les étudiants aux concepts fondamentaux de bas niveau, essentiels pour développer leur capacité à écrire du code de qualité, notamment dans des langages comme le C. La méthode d'enseignement des travaux pratiques repose sur deux séances introductives consacrées à la logique combinatoire et séquentielle, dans lesquelles les étudiants utilisent l'outil *Digital*[10]. Lors de ces séances, un ensemble d'éléments est implémenté, éléments qui sont nécessaire pour la mise en œuvre d'*ArchiLaSimu* :

- un additionneur 4 bits qui sensibilise au fonctionnement de l'opérateur combinatoire ;
- un système bus d'adresses/bus de données qui sert à écrire dans des registres ;
- un système d'acquisition et d'arbitrage de signaux d'interruption.

Suite à cela, les étudiants utilisent *ArchiLaSimu* au cours des deux séances de travaux pratiques suivantes. Dans un premier temps, ils se familiarisent avec l'outil en réalisant quelques instructions en implantant les micro-codes correspondant. Ensuite, ils codent un petit algorithme accompagné de son jeu d'instructions associé. Cette approche les pousse à repenser la manière de concevoir les programmes, c'est à dire sans utiliser de structures classiques telles que *for, if, ou while*, mais uniquement à travers des branchements conditionnels.

En troisième année de Licence EEA, les étudiants disposent déjà des connaissances nécessaires en logique combinatoire et séquentielle. L'enseignement se concentre sur l'architecture des systèmes et les différentes optimisations possibles. L'outil ArchiLaSimu est utilisé lors de deux séances de travaux pratiques pour initier les étudiants au fonctionnement d'une architecture simple. Ces séances sont suivies de travaux pratiques sur l'environnement *CubeIDE* [11], où les étudiants abordent la programmation en assembleur sur cartes STM32.

De manière générale, *ArchiLaSimu* offre aux étudiants une interface conviviale et facilement accessible, leur permettant de travailler de manière autonome. Cet outil leur fournit un environnement simple pour développer et tester des micro-codes, tout en leur offrant la possibilité de vérifier en temps réel les codes produits. Cette approche favorise l'apprentissage pratique et permet aux étudiants d'analyser et de corriger leurs erreurs de manière plus efficace, renforçant ainsi leur compréhension des concepts liés à l'architecture et à la programmation bas niveau.

5. Conclusion et perspectives

ArchiLaSimu est un émulateur d'architecture de type Von Neumann, conçu pour être facile à utiliser et permettant de soutenir l'enseignement des concepts généraux liés à l'architecture, ainsi que le micro-codage des instructions. Accessible sous la forme d'une page web, il offre une interface conviviale et multiplateforme, facilitant ainsi la prise en main par les étudiants, tant lors des travaux pratiques que pour le travail personnel en autonomie. Il est également disponible en code source (licence GPLv3) et en binaire sur https://github.com [12] pour quiconque voudrait le modifier et/ou l'héberger via ses propres serveurs.

Des évolutions futures sont envisagées, notamment l'intégration d'un module de gestion et d'émulation d'un système d'interruption. Cela permettrait de mettre en œuvre l'utilisation de la pile et d'approfondir davantage les notions liées à la gestion des interruptions.

Références

[1] J. Von Neumann, *"First Draft of a Report on the EDVAC," Moore School of Electrical Engineering, University of Pennsylvania*, 1945.

[2] J. L. Hennessy, D. A. Patterson, *"Computer Architecture: A Quantitative Approach," Morgan Kaufmann*, 2019.

[3] P. L. Higginson, *"Von Neumann Architecture Simulator," Proceedings of the 3rd Annual Conference on Teaching Computer Science*, 145-148, 2000.

[4] F. Bellard, *"QEMU, a Fast and Portable Dynamic Translator," Proceedings of the Annual Conference on Computer Systems*, 41-46, 2005.

[5] https://dl.eea-fds.umontpellier.fr/ArchiLaSimu/

[6] A. Smith, *"The CISC Processor Design: A Review of Memory-Centric Architectures," Journal of Computer Architecture*, Vol. 10 N°2, 112-124, 1982.

[7] D. Thomas, A. Hunt, "*The Pragmatic Programmer: Your Journey to Mastery," Addisson-Wesley,* ISBN-13 : 978-0135957059, 2019.

[8] G. Gallant, *"WebAssembly in Action," Manning Publications*, ISBN : 978-1617294824, 2020.

[9] https://github.com/ocornut/imgui

[10] https://github.com/hneemann/Digital

[11] https://www.st.com/en/development-tools/stm32cubeide.html

[12] https://github.com/mikamyara/archiLaSimu_2

Biographie des auteurs

Mikhaël Myara : Maître de conférences à l'Université de Montpellier depuis 2005, il consacre depuis 20 ans une grande part de ses enseignements à l'informatique et à ses applications (Python, Langage C, microcontrôleurs, programmation objet, traitement numérique du signal, ...) à destination des électroniciens, en Licence et Master EEA. En recherche, ses intérêts se portent sur l'optoélectronique (physique, métrologie et applications des lasers) et plus récemment sur la photonique (optique guidée).

Arnaud Virazel : Il a obtenu son doctorat en Microélectronique à l'Université de Montpellier, France, en 2001. Il est actuellement Professeur à l'Université de Montpellier – LIRMM (Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier), où il est responsable de l'équipe TEST (« Test et fiabilité des systèmes microélectroniques intégrés »). Il est l'auteur de 9 livres ou chapitres de livres, de 50 articles dans des revues scientifiques, ainsi que de plus de 170 communications dans des conférences et symposiums couvrant divers domaines tels que la conception pour le test (DfT), la fiabilité, les tests à faible consommation d'énergie, et le test des mémoires. Il a encadré plus de 30 thèses de doctorat dans ces domaines. Il est également responsable du département de génie électrique (environ 450 étudiants en licence et master) à l'Université de Montpellier. Ses enseignements portent principalement sur la conception des circuits numériques, le test et la fiabilité.