Manuel d'utilisation du simulateur ARM epater
Révision 2

Marc-André Gardner
Yannick Hold-Geoffroy
Jean-François Lalonde

28 février 2017

Table des matières

Introduction 3
Prise en main 3
Accès au simulateur 3
Enregistrement et chargement 3
Utilisation du simulateur 4
Éditeur de code 5
Contrôles du simulateur 7
Vue des registres 8
Vue des drapeaux 9
Vue de la mémoire 10
Description de l'instruction courante 12
Contrôles d'enregistrement et de chargement 13
Configuration 14
Instructions ARM supportées 15
Suffixes conditionnels 15
Instructions de manipulation de données 16
Instructions de comparaison 16
Instructions de décalage 16
Instructions de branchement 17
Instructions d'accès mémoire 17
Instructions d'accès mémoire multiples 17
Instructions d'accès au registre de contrôle et de statut 17
Instructions de déclenchement d'une interruption logicielle 17
Remerciements 17

Introduction

Ce manuel présente l'utilisation du simulateur **épater**, qui émule un processeur à architecture ARMv4 (coeur ARM7), tel qu'étudié dans le cours *GIF-1001 Ordinateurs : Structure et Applications*. Ce manuel se concentre uniquement sur l'utilisation du simulateur à proprement parler. Les informations concernant l'architecture ARM et la programmation en langage assembleur peuvent être obtenues dans les notes de cours ou dans les manuels de référence cités dans le plan de cours.

Prise en main

Accès au simulateur

Le simulateur est disponible en ligne, à l'adresse `http://gif1001-sim.gel.ulaval.ca/`. Aucune installation n'est nécessaire et l'accès ne requiert pas d'authentification. Le simulateur a été testé avec les navigateurs suivants :

— Google Chrome version 50 et plus, sur Windows, MacOS et Linux

— Mozilla Firefox version 44 et plus, sur Windows, MacOS et Linux

— Microsoft Edge, sur Windows 10

— Apple Safari, sur MacOS

D'autres navigateurs peuvent également être compatibles avec l'interface du simulateur, mais cette compatibilité n'est pas garantie. Si vous rencontrez des problèmes avec un navigateur alternatif, installez un des navigateurs énumérés plus haut.

Enregistrement et chargement

Note importante : **Le serveur ne réalise *aucune* sauvegarde de votre code ou de vos données. Il est de *votre* responsabilité de télécharger le fichier contenant votre code à chaque fois que vous quittez la session de simulation ou fermez votre navigateur.**

Utilisation du simulateur

La page d'accueil du simulateur est présentée à la figure 1. Elle contient, sur la gauche, un menu permettant de choisir le type d'activité (démonstrations, exercices, travaux pratiques ou codage libre) et, dans sa section principale, une liste des programmes disponibles.

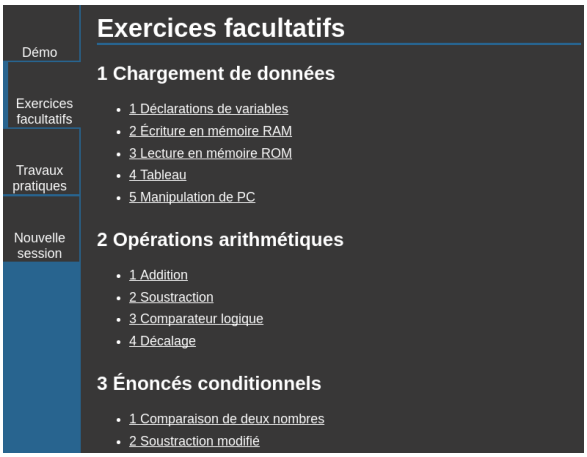


FIGURE 1: Page d'accueil du simulateur

Cliquer sur l'un de ces programmes ouvre l'interface de simulation à proprement parler. La figure 2 présente un exemple typique de cette interface, annotée. Les sous-sections suivantes présentent chaque zone de l'interface.

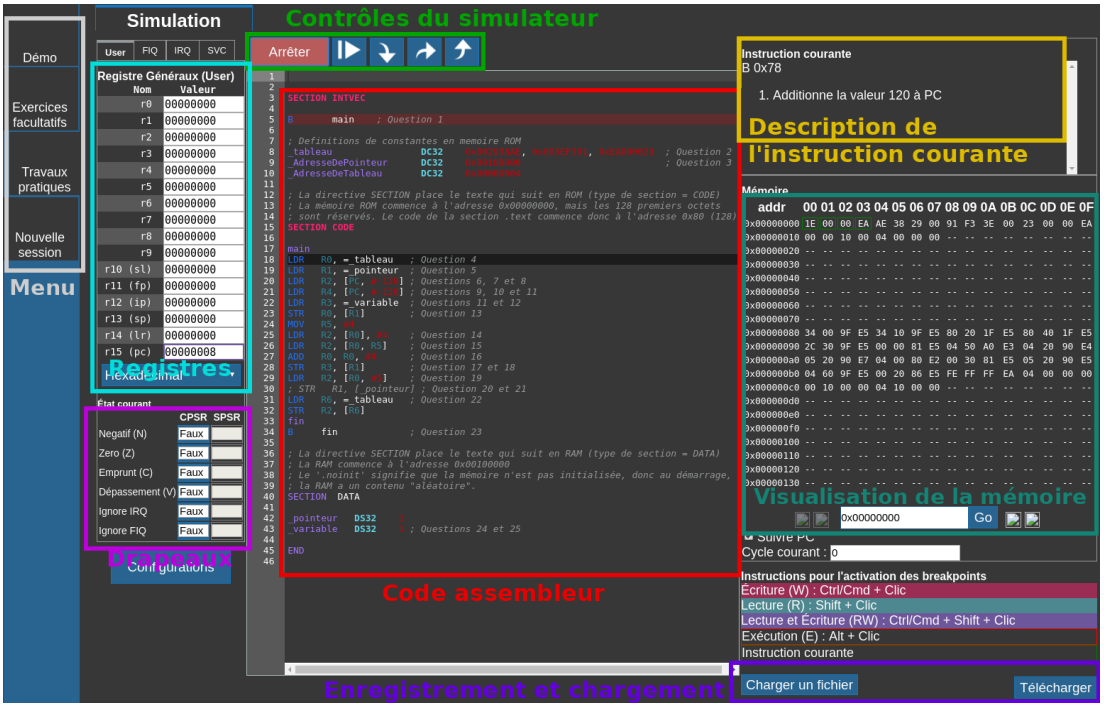


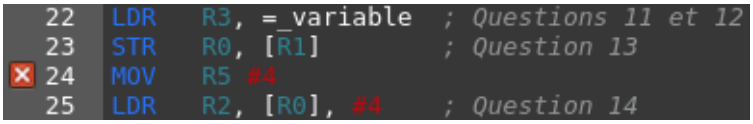
FIGURE 2: Vue d'ensemble de l'interface du simulateur.

Éditeur de code

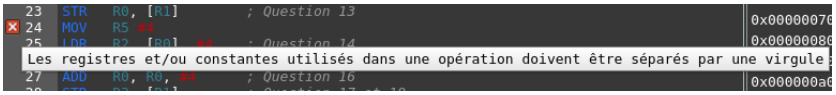
L'éditeur de code constitue la zone principale du simulateur. C'est dans cet éditeur que vous pouvez créer votre programme. Lors de l'exécution, *l'éditeur passe en mode lecture seulement*, mais présente certaines informations comme la ligne en cours d'exécution.

Cet éditeur possède une *coloration syntaxique*, c'est-à-dire qu'il est capable d'analyser le code qu'il convient pour colorier différemment les diverses parties d'une instructions. Cela permet de faciliter la lecture et la détection des erreurs.

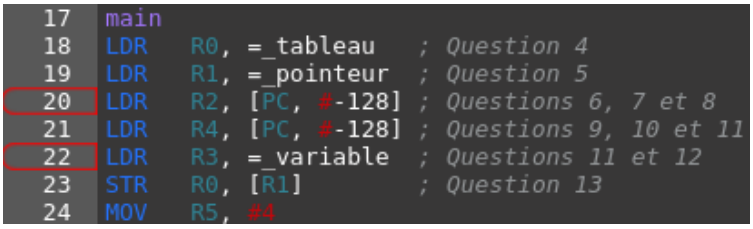
La colonne de gauche contient le numéro de ligne. Lorsqu'une instruction est erronée, un X blanc sur fond rouge y apparaît pour signaler le problème :



Passer la souris sur ce X fait apparaître une infobulle contenant un texte explicatif de l'erreur :



Une fois le programme *assemblé* (le bouton "Démarrer" pressé et le code assemblé sans erreur), il est possible de mettre en place des points d'arrêt (*breakpoints*) en cliquant sur le numéro d'une ligne. Un point d'arrêt force le simulateur à s'arrêter lorsqu'il l'atteint. Un point d'arrêt peut être désactivé en recliquant sur le même numéro de ligne. Par exemple, dans la figure suivante, les lignes 20 et 22 sont des points d'arrêt :



Note importante : un point d'arrêt ne peut être placé que sur une ligne *contenant une instruction*. Si l'utilisateur clique sur une ligne ne contenant pas d'instructions, le point d'arrêt sera placé sur la

FIGURE 3: Vue typique de l'éditeur

FIGURE 4: Exemple d'instruction erronée. Le X blanc sur fond rouge signale la ligne contenant l'instruction invalide.

FIGURE 5: Le message expliquant l'erreur peut être consulté en passant la souris sur le X.

FIGURE 6: Un point d'arrêt (*breakpoint*) peut être mis en place en cliquant sur le numéro de ligne, une fois le programme assemblé.

prochaine ligne contenant une instruction. Par exemple, à la figure 6, cliquer sur la ligne 17 placera un point d'arrêt à la ligne *18*.

Lors de l'exécution, l'éditeur passe en mode lecture seule. Il affiche cependant certaines informations. La ligne en cours d'exécution est surlignée en rouge bourgogne :

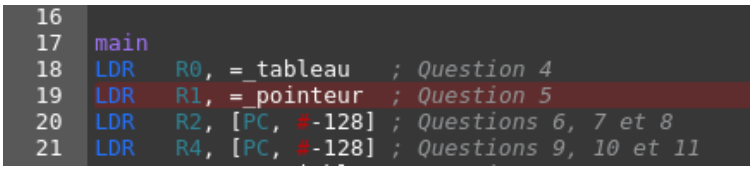


FIGURE 7: L'instruction qui va être exécutée au prochain pas de temps est surlignée en rouge dans l'éditeur. Si aucune instruction n'est surlignée, c'est que le *Program Counter* (PC) ne se situe pas dans la mémoire d'instructions.

De même, lorsque l'instruction courante est un branchement ou un appel de fonction, l'éditeur affiche la destination du branchement en surlignant en noir la prochaine instruction. Par exemple, dans ce cas-ci, l'instruction en cours d'exécution (surlignée en rouge) est le *B main*, et la prochaine instruction sera celle de la ligne 18 (surlignée en noir) :

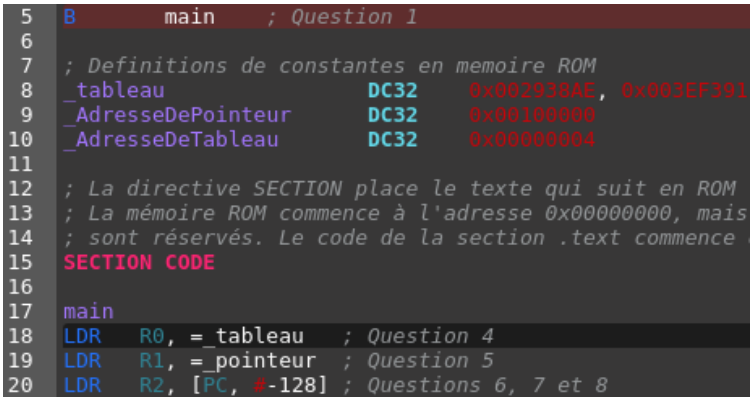


FIGURE 8: La prochaine instruction à être exécutée après un branchement est surlignée en noir profond. Si le branchement est conditionnel, cet affichage en tient compte et affiche la prochaine instruction correspondant à la branche prise.

Contrôles du simulateur

Les boutons de contrôle du simulateur permettent d'agir sur la simulation. Cette interface diffère selon le mode actuel. Dans le mode *édition* (figure 9), seul le bouton *Démarrer* est actif. Dans le mode *exécution*, que l'on peut accéder en pressant sur *Démarrer*, tous les boutons sont actifs, et le bouton *Démarrer* devient *Arrêter*.

FIGURE 9: Barre de contrôle en mode *édition*

FIGURE 10: Barre de contrôle en mode *exécution*

Lorsque le bouton *Démarrer* est pressé, le simulateur tente d'abord d'assembler le code. Si aucune erreur n'est détectée, il passe alors en mode exécution. Dans ce mode, tous les boutons sont actifs et sont, respectivement, de gauche à droite :

1. **Arrêt** : interrompt la simulation et revient en mode édition.
2. **Réinitialisation** : effectue l'équivalent d'une interruption *reset* sur le microprocesseur. La valeur de PC est mise à 0. Notez que cela n'affecte *pas* la valeur des autres registres et des drapeaux.
3. **Exécution en continu** : le simulateur exécute les instructions suivantes sans arrêt, jusqu'à ce qu'il rencontre une erreur ou un point d'arrêt. Notez que le simulateur est volontairement limité quant au nombre d'instructions qu'il peut exécuter d'un seul coup. Lorsque ce nombre est atteint, le simulateur arrête comme s'il avait rencontré un point d'arrêt. Il suffit d'appuyer à nouveau sur le bouton d'exécution en continu pour poursuivre.
4. **Exécuter instruction courante** : exécute l'instruction courante (celle qui est surlignée dans l'éditeur) et passe à la suivante puis arrête.
5. **Exécuter ligne courante** : dans le cas où la ligne courante est une instruction autre que BL, cette action a le même effet que la précédente. Toutefois, dans le cas d'un appel de fonction (BL), cette action exécute l'entièreté de la fonction jusqu'à son retour.
6. **Exécuter jusqu'à la sortie** : dans le cas où la ligne courante est dans une fonction, cette action exécute sans arrêt les instructions jusqu'à sortir de la fonction (appel de BX). Si la ligne courante n'est pas dans une fonction, cette action a le même effet qu'une exécution en continu.

Vue des registres

La section de gauche de l'interface présente les valeurs contenues dans les registres du processeur. Les 16 registres que contient un processeur ARM (R0 à R15) sont affichés.

Par défaut, leur valeur est présentée en hexadécimal, mais il est possible de choisir différents modes d'affichage en cliquant sur le mode actuel pour faire apparaître le menu de sélection. Le mode décimal signé correspond à une interprétation complément-2 de la valeur du registre, alors que le mode non-signé correspond à une simple conversion vers le système décimal.

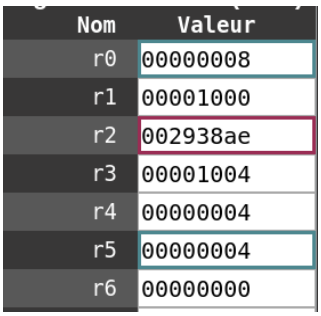
Les onglets au-dessus des registres permettent de choisir la *banque* de registre à visualiser. La plupart des banques de registre d'un processeur ARM sont présentes, incluant la banque IRQ (interruption), FIQ (interruption rapide) et SVC (interruption logicielle).

La valeur d'un registre peut être modifiée en changeant sa valeur dans l'interface. Attention toutefois à respecter le type d'affichage sélectionné (par exemple, il est illégal d'écrire autre chose que 0 ou 1 en mode binaire).

Il est possible d'affecter un point d'arrêt à un registre, en mode lecture ou écriture. Un point d'arrêt lié à un registre met la simulation en pause lorsque le registre est écrit ou lu. Par exemple, dans la figure suivante, les registres R3 et R5 ont un point d'arrêt en écriture et le registre R2 un point d'arrêt en lecture :

Un point d'arrêt en écriture peut être ajouté en cliquant sur le nom du registre tout en tenant la touche Ctrl ou Cmd enfoncée, respectivement sur Windows et MacOS. De même, un point d'arrêt en lecture peut être ajouté en cliquant tout en tenant la touche Maj (Shift) enfoncée. Un point d'arrêt précédemment créé peut être retiré en répétant la même opération. Il est possible de mettre à la fois un point d'arrêt en lecture et en écriture sur le même registre.

Lorsque l'instruction courante lit ou écrit un registre, sa valeur est entourée d'un rectangle turquoise (lecture) ou rouge (écriture). Par exemple, dans l'image suivante, l'instruction courante lit les valeurs de R0 et R5 et écrit dans R2 :



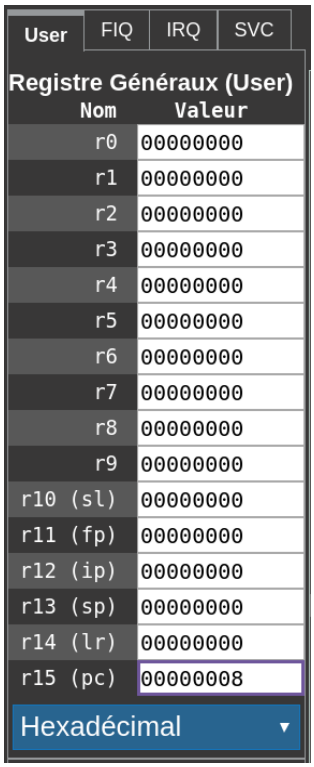


FIGURE 11: Vue des registres généraux

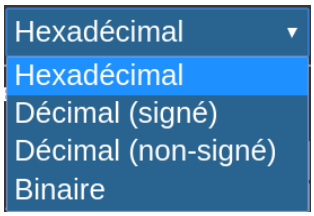


FIGURE 12: Menu de sélection du mode d'affichage

FIGURE 13: Onglets de sélection de la banque de registres à visualiser

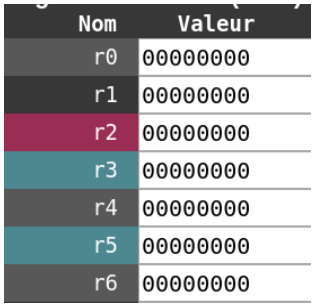


FIGURE 14: Points d'arrêt sur les registres R2, R3 et R5.

Vue des drapeaux

Les drapeaux de l'ALU sont présentés juste en dessous de la vue des registres.

Un drapeau ne peut prendre que deux valeurs : vrai ou faux (0 ou 1 en binaire). Il est possible de changer la valeur d'un drapeau en cliquant sur sa valeur. L'interface présente à la fois la vue pour le registre de statut courant (CPSR) et le registre de statut sauvegardé (SPSR), si applicable.

Le registre de statut sauvegardé (SPSR) est propre à chaque banque de registres, hormis la banque *User* qui n'en possède pas. Il permet de conserver le registre de statut du programme principal pendant une interruption. Dans la majorité des cas (exécution en mode *User*), les valeurs du SPSR ne sont pas définies ou modifiables.

Tout comme pour les registres, les drapeaux lus ou écrits par une instructions voient leur contour coloré de manière différente.

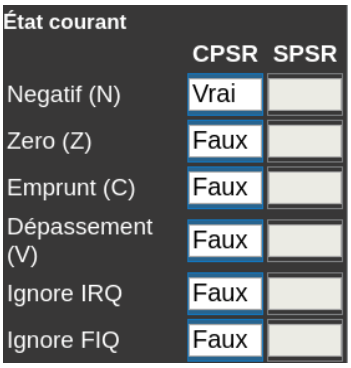


FIGURE 15: Vue des drapeaux de l'ALU

Vue de la mémoire

La vue de la mémoire présente le contenu de la mémoire d'instructions et de données. Cette vue est arrangée en tableau, où chaque ligne fait 16 octets de largeur. Par exemple, pour retrouver la valeur à l'adresse 0x9A, il suffit d'aller au croisement de la ligne 0x90 et de la colonne 0xA. Les espaces mémoire non déclarés (qui ne se rapportent ni à une instruction, ni à une variable) sont indiqués par des tirets. L'affichage du contenu de la mémoire se fait en hexadécimal.

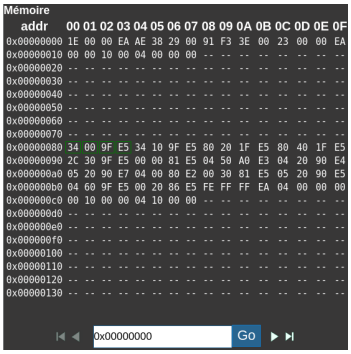


FIGURE 16: Vue de la mémoire

Tout comme pour les registres et les drapeaux, il est possible de modifier une valeur initialisée de la mémoire en cliquant. Les zones non déclarées (dont la valeur est identifiée par des tirets) ne peuvent être modifiées. La valeur doit être écrite en hexadécimal et ne peut excéder la capacité d'un octet (255, soit 0xFF).

Lors de l'exécution, les octets composant l'instruction courante (celle surlignée en rouge dans l'éditeur) sont entourés de vert, comme dans l'exemple suivant :

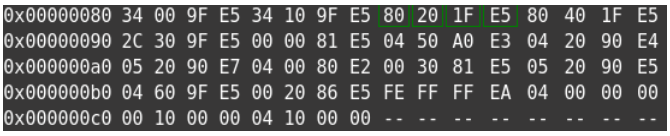


FIGURE 17: Affichage des octets composant l'instruction courante

Lorsque l'instruction est une opération agissant en mémoire, les cases mémoires lues ou écrites voient leurs valeurs colorées de manière différente (vert dans le cas d'une lecture, rouge dans le cas d'une écriture). Par exemple, dans l'image suivante, l'instruction courante lit les valeurs 0x10 à 0x13 inclusivement :

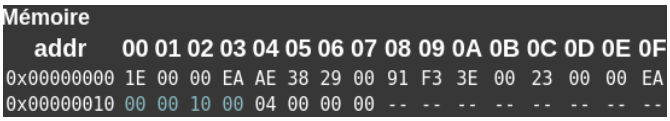


FIGURE 18: Exemple du changement de couleur des valeurs d'une case mémoire lors d'un accès

Il est possible de placer des points d'arrêt en mémoire. Ceux-ci peuvent être de trois types :

— **Lecture** : le simulateur s'arrête lorsqu'un accès en lecture (par exemple via l'instruction LDR) est effectué sur cette case mémoire

— **Écriture** : le simulateur s'arrête lorsqu'un accès en écriture (par exemple via l'instruction STR) est effectué sur cette case mémoire

— **Exécution** : le simulateur s'arrête lorsque le *Program Counter* (PC) atteint cette valeur

Notons que ces trois types de points d'arrêt peuvent être combinés. Par exemple, dans l'image suivante, un point d'arrêt en lecture est présent pour les adresses 0x04 à 0x07, un point d'arrêt en écriture est

actif aux adresses 0xoC à 0xoF et un point d'arrêt en exécution est présent à l'adresse 0x8C :

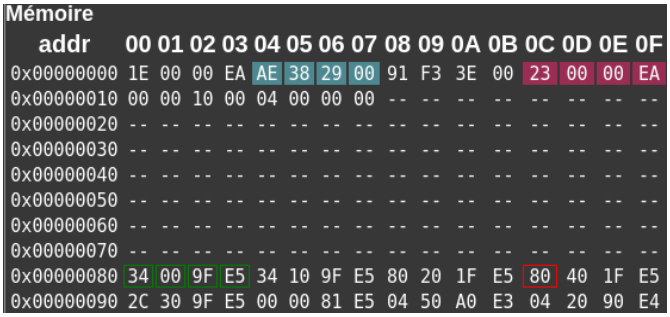


FIGURE 19: Exemple de points d'arrêt en mémoire. Un point d'arrêt en lecture peut être ajouté en cliquant sur une case mémoire tout en pressant la touche Control (Ctrl). Un point d'arrêt en écriture peut être ajouté en cliquant et pressant la touche Maj (Shift). Un point d'arrêt en écriture peut être ajouté en cliquant et pressant la touche Alt.

Afin de faciliter l'utilisation du simulateur, un aide-mémoire succinct reprenant les informations présentées ici est disponible en bas de la vue mémoire :

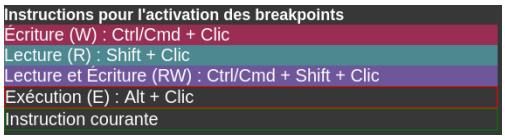


FIGURE 20: Description des différents types de points d'arrêt en mémoire dans l'interface du simulateur

Finalement, en bas du visualisateur, on retrouve une interface permettant de naviguer dans la mémoire :

FIGURE 21: Contrôle de la position dans la mémoire

Les flèches permettent d'aller vers des adresses mémoire plus hautes ou plus basses. Le champ de texte central permet de spécifier directement une adresse, à laquelle on peut par la suite se rendre en pressant le bouton *Go*.

Finalement, notons que cette vue indique, par un soulignement blanc, la position en mémoire de la ligne couramment *sélectionnée* dans l'éditeur (celle sur laquelle se trouve le curseur). Si cette ligne est une instruction, elle couvrira 4 cases (soit 4 octets). S'il s'agit plutôt d'une constante ou d'une variable, elle indiquera toutes les cases mémoire appartenant à de celle-ci.

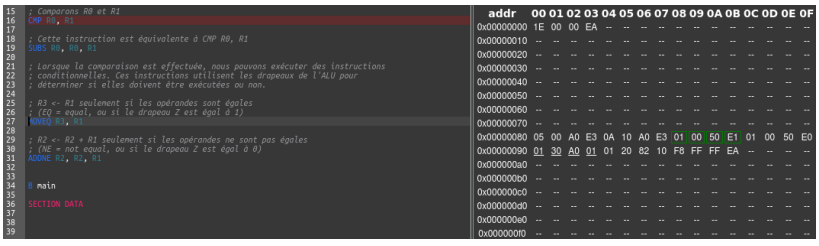


FIGURE 22: Vue de la position de la ligne couramment sélectionnée. Dans ce cas-ci, la ligne 27 est sélectionnée ; la vue mémoire souligne en blanc les cases mémoires qui y correspondent, soit les adresses 0x90 à 0x93 inclusivement. La ligne en cours d'exécution est quant à elle surlignée en rouge et sa position en mémoire indiquée par des carrés verts autour des cases concernées.

Description de l'instruction courante

Cette section présente des informations textuelles sur l'instruction courante. La première ligne présente l'instruction *désassemblée* : elle provient non pas de l'éditeur, mais du *bytecode* lui-même. Cela lui permet d'apporter de l'information supplémentaire dans deux circonstances :

1. Les étiquettes y sont remplacées par les décalages effectifs requis pour atteindre la case mémoire demandée
2. Dans le cas où l'on exécute des données présentes en mémoire, auxquelles aucun code assembleur n'est lié, cet affichage reste fonctionnel puisqu'il ne se base que sur la représentation binaire des instructions.

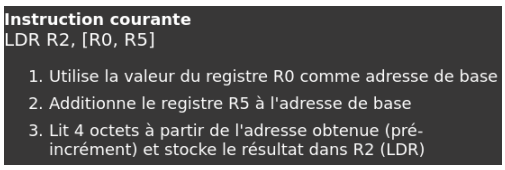


FIGURE 23: Affichage de l'instruction désassemblée et de sa description. En fonction de l'instruction, la description peut être plus ou moins longue.

Les lignes suivantes sont une description textuelle de l'instruction, générée automatiquement. Elles indiquent, dans l'ordre, les opérations effectuées pour obtenir le résultat final et leurs effets de bord s'il y a lieu. Dans le cas d'une instruction indéfinie, cette zone reste vide.

Contrôles d'enregistrement et de chargement

Ces contrôles, situés au bas de l'écran, vous permettent de télécharger le code présent dans l'éditeur et de charger un code présent sur votre ordinateur. Ces deux actions sont analogues à l'ouverture et l'enregistrement dans une application traditionnelle.

FIGURE 24: Interface d'ouverture et d'enregistrement

N'importe quel type de fichier texte brut peut être lu par le simulateur. Nous vous recommandons toutefois d'utiliser l'extension "txt" afin d'éviter la confusion avec d'autres formats de fichier. C'est d'ailleurs cette extension que comportent les fichiers téléchargés depuis le simulateur.

Note importante : ces contrôles constituent la *seule* manière de sauvegarder et réouvrir votre code. Il n'y a *aucun* système de sauvegarde automatique et le serveur ne conserve *aucune* trace du code exécuté. **Si vous ne téléchargez pas votre code avant de fermer votre navigateur ou de changer de page, les modifications que vous y avez apportées seront perdues et irrécupérables.** Un message d'avertissement vous est présenté lorsque vous tentez de fermer la fenêtre afin de vous rappelez de télécharger le fichier.

Configuration

La fenêtre de configuration peut être ouverte en pressant le bouton *Configurations* situé sous la vue des drapeaux. Une fois ouverte, la fenêtre ressemble à celle-ci :

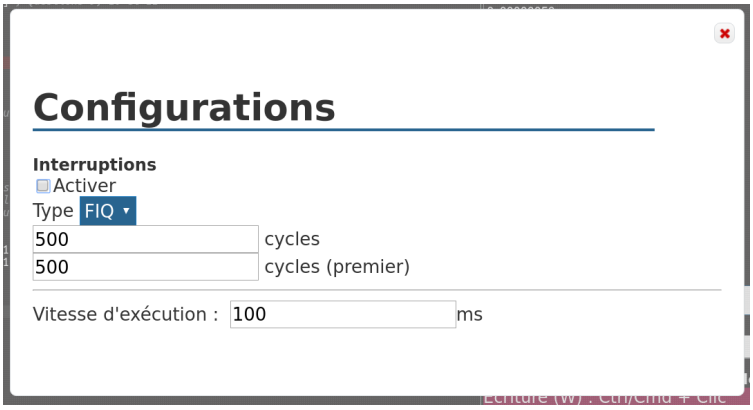


FIGURE 25: Interface de configuration des interruptions et du simulateur

Deux éléments principaux peuvent être configurés dans cette interface :

1. Une interruption temporelle peut être activée, en mode FIQ ou IRQ. Le nombre de cycles avant la première interruption, de même que le délai de répétition (encore là en nombre de cycles processeur) peuvent être configurés.
2. La vitesse d'exécution, qui correspond au délai minimum entre l'exécution de deux instructions consécutives. Normalement, un délai de 100ms est suffisant pour exécuter les différents travaux pratiques et exercices du cours. Toutefois, dans certains cas spécifiques, une vitesse plus rapide peut être bénéfique. Dans tous les cas, n'oubliez pas que le simulateur ne peut exécuter plus qu'un certain nombre d'instructions à la suite, avant de se mettre en pause et d'attendre une nouvelle commande.

Instructions ARM supportées

Le simulateur supporte les instructions suivantes. La syntaxe à utiliser est celle présentée dans les spécifications et documentations techniques d'ARM, sauf pour les spécificités suivantes :

1. Toutes les mnémoniques (MOV, LDR, BL, etc.) doivent être écrites en **majuscules**
2. Tous les noms de registres doivent être écrits en **majuscules**
3. Les étiquettes peuvent être indifféremment en majuscules ou minuscules, mais ne peuvent être directement une mnémonique. Elles peuvent également contenir des chiffres, mais ne peuvent pas *commencer* par un chiffre.
4. L'indentation n'a pas d'importance. De même, l'espacement entre les différents composants d'une instruction n'est pas pris en compte, hormis pour l'espace ou la tabulation après la mnémonique, qui est obligatoire.
5. Une *section* se déclare uniquement par son nom. Les attributs de section (par exemple *.noinit*) ne sont pas supportés.

Référez-vous aux notes de cours ou à la documentation d'ARM pour plus de détails sur chacune de celles-ci.

Suffixes conditionnels

Toutes les instructions listées ci-dessous peuvent être exécutées *conditionnellement*, en utilisant un des codes à 2 lettres suivants :

— **EQ** teste l'égalité ($A == B$)

— **NE** teste l'inégalité ($A! = B$)

— **CS** teste si un premier nombre non-signé est plus grand ou égal à un second ($A >= B$)

— **CC** teste si un premier nombre non-signé est strictement plus petit qu'un second ($A < B$)

— **MI** teste si le résultat est négatif ($A < 0$)

— **PL** teste si le résultat est positif ($A >= 0$)

— **VS** teste si la précédente opération a généré un *débordement*

— **VC** teste si la précédente opération n'a PAS généré un *débordement*

— **HI** teste si un premier nombre non-signé est plus grand à un second ($A > B$)

— **LS** teste si un premier nombre non-signé est plus petit ou égal à un second ($A <= B$)

— **GE** teste si un premier nombre est plus grand ou égal à un second ($A >= B$)

— **LT** teste si un premier nombre est strictement plus petit qu'un second ($A < B$)

— **GT** teste si un premier nombre est strictement plus grand qu'un second ($A > B$)

— **LE** teste si un premier nombre est plus petit ou égal à un second ($A > B$)

— **AL** exécute systématiquement l'instruction

Instructions de manipulation de données

— MOV

— MVN

— ADD

— SUB

— MUL

— MLA

— AND

— EOR

— ORR

— ADC

— RSB

— SBC

— RSC

— BIC

— NOP

Instructions de comparaison

— CMP

— CMN

— TST

— TEQ

Instructions de décalage

Note : ces instructions sont des pseudo-instructions. Elles sont transformées par le programme assembleur en une instruction MOV intégrant le décalage.

— LSL

— LSR
— ASR
— ROR
— RRX

Instructions de branchement

— B
— BL
— BX

Instructions d'accès mémoire

— LDR
— STR
— LDRB
— STRB

Instructions d'accès mémoire multiples

— LDM
— STM
— PUSH
— POP

Instructions d'accès au registre de contrôle et de statut

— MSR
— MRS

Instructions de déclenchement d'une interruption logicielle

— SWI
— SVC

Remerciements

Les auteurs tiennent à remercier les personnes suivantes pour l'aide qu'ils ont apporté à la réalisation de ce simulateur :

— Jessica Déziel, pour le design de l'interface graphique
— Jonathan Gilbert, pour les tests préliminaires du simulateur et la réalisation des exercices