Manuel d'utilisation du simulateur ARM

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

30 janvier 2017

Introduction

Ce manuel présente l'utilisation du simulateur d'un processus à architecture ARM, 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 trouvées 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. Le simulateur est disponible 24 heures sur 24, sans interruption. 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.

L'accès au simulateur peut se faire sur le campus ou en dehors de celui-ci et ne requiert pas d'authentification.

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 en cours, exercices, travaux pratiques ou codage libre) et, dans sa section principale, une liste des programmes disponibles.

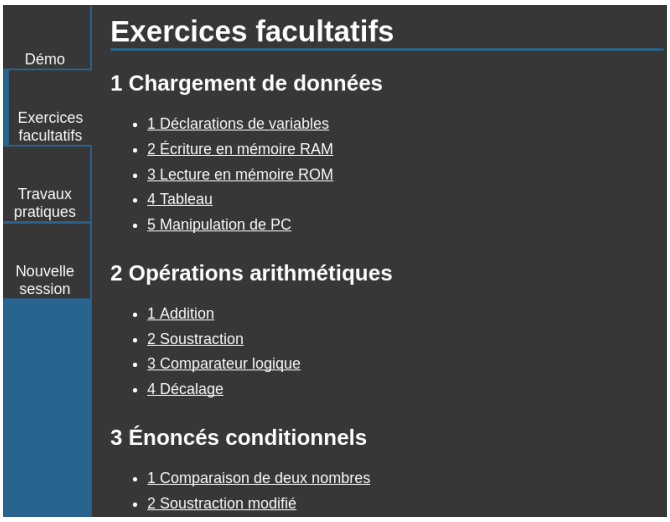


FIGURE 1: Page d'accueil du simulateur

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.

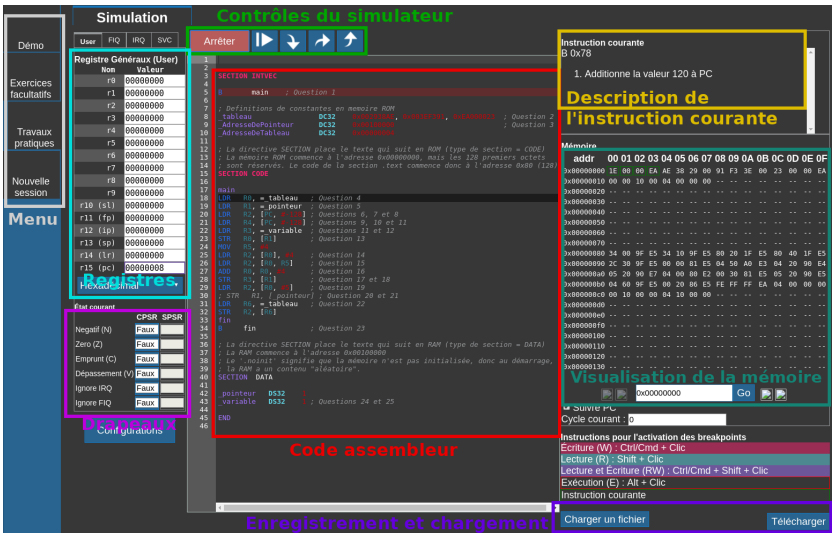


FIGURE 2: Vue d'ensemble de l'interface du simulateur. La section centrale constitue l'éditeur de code. La section de droite présente l'instruction courante et une vue de la mémoire. La section de gauche indique les valeurs présentes dans les registres et les drapeaux de l'ALU. Au-dessus de l'éditeur de code, on retrouve les boutons permettant de contrôler l'avancement de la simulation. Finalement, en bas de la colonne de droite se trouve les boutons permettant de charger ou d'enregistrer le code de l'éditeur sur votre ordinateur.

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

FIGURE 3: Vue typique de l'éditeur

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 :

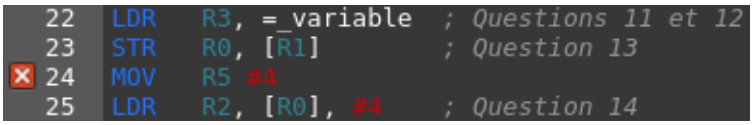


FIGURE 4: Exemple d'instruction erronée. Le X blanc sur fond rouge signale la ligne contenant l'instruction invalide.

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

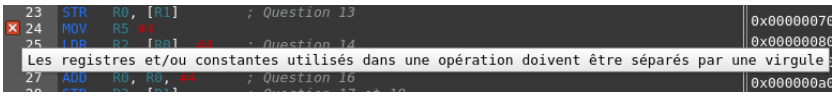


FIGURE 5: Le message expliquant l'erreur peut être consulté en passant la souris sur le X.

Une fois le programme *assemblé* (le bouton "Démarré" 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 :

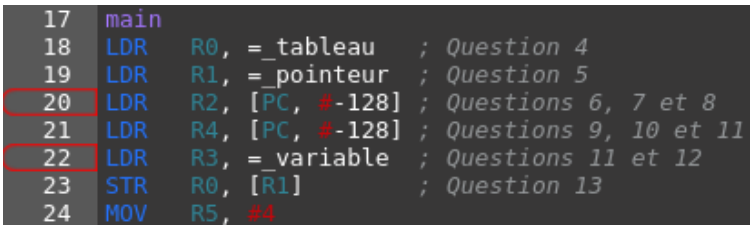


FIGURE 6: Un point d'arrêt (*breakpoint*) peut être mis en place en cliquant sur le numéro de ligne, une fois le programme assemblé.

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 prochaine ligne contenant une instruction. Par exemple, dans 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 :

De même, lorsque l'instruction courante est un branchement ou un appel de fonction, l'éditeur affiche la destination du branchement

```
16
17  main
18  LDR   R0, =_tableau   ; Question 4
19  LDR   R1, =_pointeur  ; Question 5
20  LDR   R2, [PC, #-128] ; Questions 6, 7 et 8
21  LDR   R4, [PC, #-128] ; Questions 9, 10 et 11
```

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.

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) :

```
5   B       main    ; Question 1
6
7   ; Definitions de constantes en memoire ROM
8   _tableau              DC32    0x002938AE, 0x003EF391
9   _AdresseDePointeur    DC32    0x00100000
10  _AdresseDeTableau     DC32    0x00000004
11
12  ; La directive SECTION place le texte qui suit en ROM
13  ; La mémoire ROM commence à l'adresse 0x00000000, mais
14  ; sont réservés. Le code de la section .text commence
15  SECTION CODE
16
17  main
18  LDR   R0, =_tableau   ; Question 4
19  LDR   R1, =_pointeur  ; Question 5
20  LDR   R2, [PC, #-128] ; Questions 6, 7 et 8
```

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

FIGURE 9: Les deux modes de la barre de contrôle.

Les boutons de contrôle du simulateur permettent d'agir sur la simulation.

Cette interface diffère selon le mode actuel. Dans le mode *d'édition* (image du haut de la figure), seul le bouton *Démarrer* est actif. Dans le mode *d'exécution*, tous les boutons sont actifs, et le bouton *Démarrer* devient *Arrêter*.

En mode édition, le bouton *Démarrer* lance l'assemblage du code présent dans l'éditeur. Si aucune erreur n'est détectée, le simulateur passe alors en mode d'exécution. Dans ce mode, le bouton *Arrêter* stoppe la simulation et revient en mode édition. Notez qu'une modification à l'éditeur a le même effet. Les quatre autres boutons sont, respectivement, de gauche à droite :

1. **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. Lorsque ce nombre est atteint, le simulateur arrête comme s'il avait rencontré un point d'arrêt.

2. **Exécuter instruction courante** : exécute l'instruction courante (celle qui est surlignée dans l'éditeur) et passe à la suivante puis arrête.
3. **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.
4. **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 (BX). Si la ligne courante n'est pas dans une fonction, cette action a le même ffet qu'une exécution en continu.

Vue des registres

La section de gauche présente les valeurs contenues dans les registres du processeur. Les 16 registres que contient un processeur ARM (Ro 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. Le mode binaire permet de voir chacun des 32 zéros et uns composant la valeur contenue dans le registre.

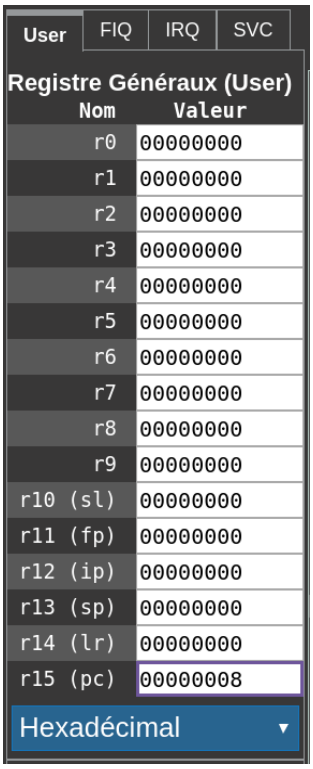


FIGURE 10: Vue des registres généraux

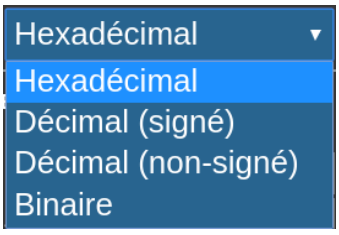


FIGURE 11: Menu de sélection du mode d'affichage

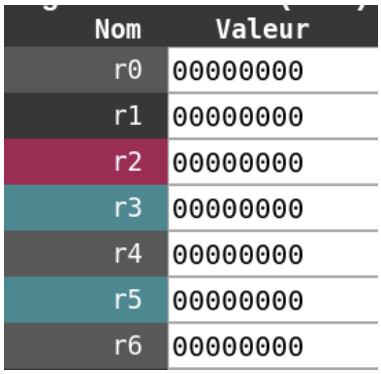
Les onglets au-dessus des registres permettent de choisir la *banque* de registre à visualiser. La plupart des banques de registre d'un processeur ARM sont présentes, incluant la banque IRQ (interruption), FIQ (interruption rapide) et SVC (interruption logicielle).

FIGURE 12: Onglets de sélection de la banque de registres à visualiser

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 :

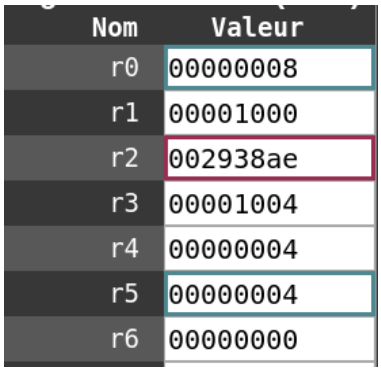


Nom	Valeur
r0	00000000
r1	00000000
r2	00000000
r3	00000000
r4	00000000
r5	00000000
r6	00000000

FIGURE 13: Points d'arrêt sur les registres R2, R3 et R5.

Un point d'arrêt en lecture peut être ajouté en cliquant sur le nom du registre tout en tenant la touche Control (Ctrl) enfoncée. De même, un point d'arrêt en écriture 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 (dans le cas d'une lecture) ou rouge (dans le cas d'une écriture). Par exemple, dans l'image suivante, l'instruction courante lit les valeurs de R0 et R5 et écrit dans R2 :



Nom	Valeur
r0	00000008
r1	00001000
r2	002938ae
r3	00001004
r4	00000004
r5	00000004
r6	00000000

FIGURE 14: Utilisation des registres par l'instruction courante.

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.

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

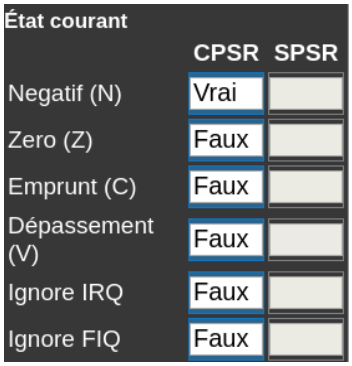


FIGURE 15: Vue des drapeaux de l'ALU

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.

Tout comme pour les registres et les drapeaux, il est possible de modifier une valeur initialisée de la mémoire en cliquant. La valeur doit être écrite en hexadécimal et ne peut excéder 0xFF (255).

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 :

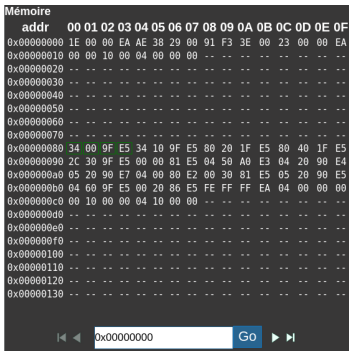


FIGURE 16: Vue de la mémoire

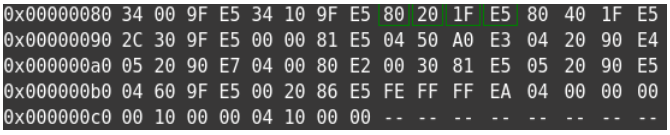


FIGURE 17: Affichage des octets composant l'instruction courante

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 :

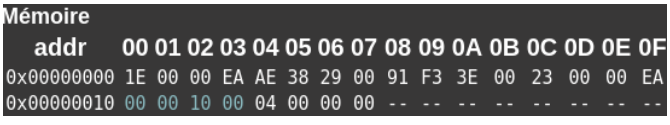


FIGURE 18: Exemple du changement de couleur des valeurs d'une case mémoire lors d'un accès

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 0x0C à 0x0F et un point d'arrêt en exécution est présent à l'adresse 0x8C :

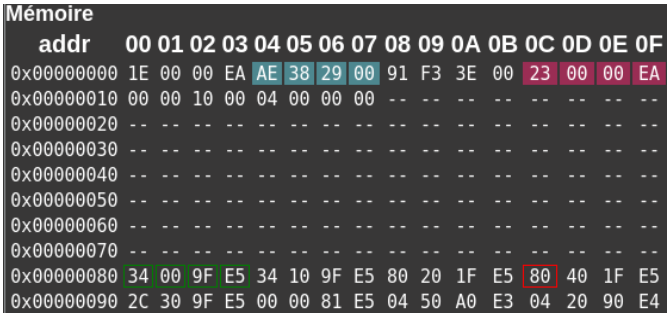


FIGURE 19: Exemple de points d'arrêt en mémoire. Un point d'arrêt en lecture peut être ajouté en cliquant sur une case mémoire tout en pressant la touche Control (Ctrl). Un point d'arrêt en écriture peut être ajouté en cliquant et pressant la touche Maj (Shift). Un point d'arrêt en écriture peut être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

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

FIGURE 20: 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*.

Description de l'instruction courante

Contrôles d'enregistrement et de chargement

Configuration

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
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.

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

Utilisation du simulateur