INF1600

Travail pratique 1

Périphériques et architecture

Département de Génie Informatique et Génie Logiciel
Polytechnique Montréal

1 Introduction et sommaire

Ce travail pratique a pour but de vous familiariser avec un processeur accumulateur disposant de fonctionnalités additionnelles. Les nouvelles instructions facilitent l'écriture de programmes concis et efficaces. Il sera question dans ce TP d'utiliser les instructions de branchements (permettant des boucles) et d'exploiter le registre `MA` pour faciliter les accès en mémoire. Enfin, vous allez analyser et développer divers programmes sur Code Machine à partir des connaissances acquises au TP0 (opérations binaires, écritures et lectures d'entrées en C, etc.).

1.1 Remise

Voici les détails concernant la remise de ce travail pratique :

- Méthode : sur Moodle, une **seule remise par équipe**, incluant un rapport **PDF**. **Seules des équipes de deux (2) étudiants sont tolérées**, sauf avis contraire.
- Format: un dossier compressé intitulé `<matricule1>-<matricule2>-<tp1_section_X>.ZIP`, incluant les sources de vos programmes en C et en assembleur, de même qu'un rapport en format `.PDF`. Incluez une **page titre** où figurent les noms et matricules des deux membres de l'équipe, votre groupe de laboratoires, le nom et le sigle du cours, la date de remise et le nom de l'École. Dans une seconde page, incluez le **barème** de la section 1.2. Finalement, diverses captures d'écran **pertinentes** doivent figurer au sein de votre rapport. Celles-ci ne peuvent expliquer votre travail d'elles-mêmes ; les captures d'écran servent de support complémentaire. Une justification écrite est de mise.
- **Attention :** L'équipe de deux que vous formez pour ce TP sera **définitive** jusqu'au TP5. Il **ne sera pas possible** de changer d'équipe au cours de la session (sauf avis contraire).

1.2 Barème

Les travaux pratiques 1 à 5 sont notés sur 4 points chacun, pour un total de 20/20. Le TP1 est noté selon le barème suivant. Reproduisez ce tableau dans le document PDF que vous allez remettre.

TP 1			**/4,00**
Section 2			
Partie 1			/0,75
	Q1	/0,25	
	Q2	/0,25	
	Q3	/0,25	
Partie 2			/0,75
	Q1	/0,25	
	Q2	/0,25	
	Q3	/0,25	
Section 3			
Partie 1			/1,25
	Q1	/0,75	
	Q2	/0,25	
	Q3	/0.25	
Partie 2			/1,25
	Q1 — *Code Codemachine*	/0,75	
	Q2 — *Code en C*	/0,5	

2 Utilisation des instructions de branchement

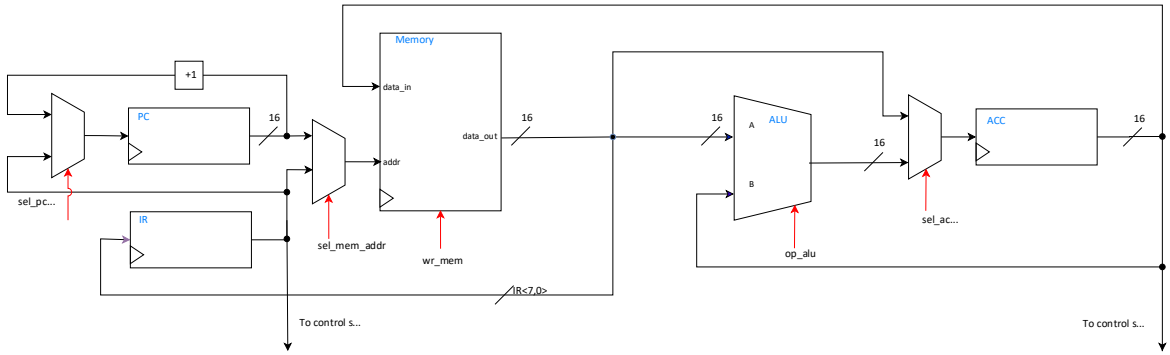

Figure 1 Architecture du processeur à accumulateur simple

Soit l'architecture du processeur accumulateur étudiée au TP0. Dans ce travail pratique, nous allons étendre le jeu d'instructions du processeur pour permettre des branchements avec et sans condition. Pour ce faire, le chemin de données entre le registre `IR` et le registre `PC` permet de modifier ce dernier au travers d'une adresse contenue dans l'adresse `IR.ADR`. Lorsqu'un branchement survient, `PC` prend la valeur précisée par `IR.ADR`.

Tableau 1 Jeu d'instructions du processeur à accumulateur

Instruction	Description
`add ADR`	`ACC ← ACC + Mémoire[ADR]`
`sub ADR`	`ACC ← ACC - Mémoire[ADR]`
`mul ADR`	`ACC ← ACC × Mémoire[ADR]`
`st ADR`	`Mémoire[ADR] ← ACC`
`ld ADR`	`ACC ← Mémoire[ADR]`
`stop`	`Arrêt du programme`
`br ADR`	`PC ← ADR`
`brz ADR`	`ACC = 0 ? PC ← ADR : PC ← PC + 1`
`brnz ADR`	`ACC != 0 ? PC ← ADR : PC ← PC + 1`

Le **Tableau 1** donne la liste des instructions acceptées par ce processeur. Les branchements conditionnels (`brz` et `brnz`) dépendent de la valeur courante dans l'accumulateur. Le branchement `br`, quant à lui, effectue un branchement inconditionnel, de sorte que `PC` prend toujours l'adresse contenue dans `IR` lors de l'exécution de cette instruction.

2.1 Partie 1 :

Copiez le code du fichier `partie-2-1.asm` et veuillez l'insérer dans l'éditeur de Code Machine du processeur à accumulateur **sans registre `MA`**. Répondez aux questions qui suivent après avoir remplacé la valeur de `value1` par

```
abs((MATRICULE_1 + MATRICULE_2) % 13)
```

et `value2` par :

```
abs(3 + ((MATRICULE_1 + MATRICULE_2) % 13))
```

Important : n'oubliez pas d'inclure des captures d'écran au sein de votre rapport lorsque nécessaire.

Q0/0,00 points Indiquez votre `valeur1` et de `valeur2` ou il est marqué ***ICI*** dans le code.

Q1/0,25 point Quelles sont les structures de contrôle utilisées au sein de ce programme ?

Q2/0,25 point Quel est le contenu en mémoire à l'adresse `0x0017` (qui diffère selon votre valeur de `data`) à la fin de l'exécution de ce programme ?

Q3/0,25 point Que fait ce programme ? Répondez à la question en décrivant le principe de fonctionnement du programme en évitant de référer à son contenu ligne par ligne.

2.2 Partie 2 :

Avec l'éditeur d'assembleur de Code Machine pour le processeur à accumulateur **sans registre `MA`**, écrivez un programme qui permet de trouver le **n**$^{\text{ième}}$ terme de la suite de Fibonacci. Vous devez impérativement utiliser les structures de contrôles. Un code n'utilisant pas ce principe aura la note de 0 à la question. La réponse doit être stockée dans une variable nommée **`answer`**.

Q1/0,25 point Réaliser la question sur code machine. Veuillez remettre la question en format ***.asm*** ou ***.txt***

Q2/0,25 point Identifier votre plage de possibilités et les raisons qui amènent cette étendue.

Q3/0,25 point Veuillez écrire la question en C. La question ne doit pas être écrite sous forme récursive. Référez-vous au fichier source donné.

3 Ajout du registre MA au processeur à accumulateur

Le processeur accumulateur a été enrichi d'un registre supplémentaire, le registre `MA`. Ce registre facilite les accès en mémoire. Il sert à garder l'adresse de l'espace en mémoire que le programme désire accéder.

Soit la nouvelle architecture du processeur à accumulateur avec registre `MA`:

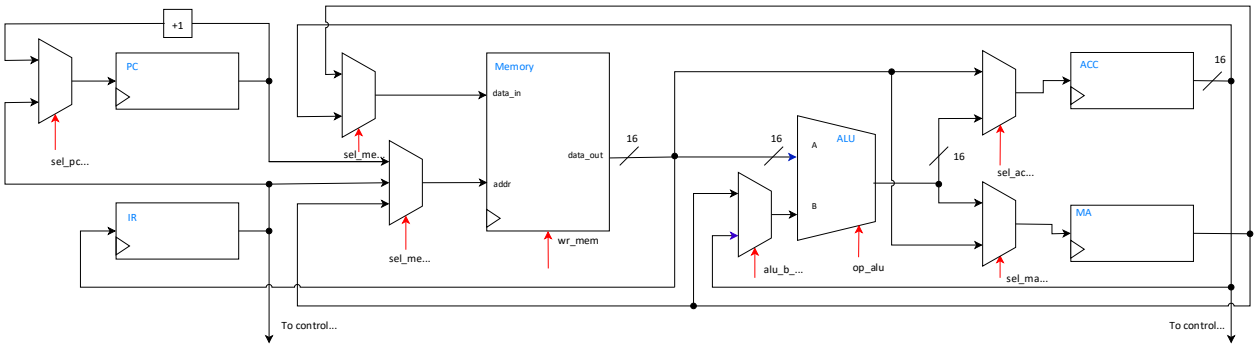

Figure 2 Processeur à accumulateur avec registre `MA`

La valeur du registre MA peut être enregistrée en mémoire, mais elle peut aussi être utilisée pour préciser une adresse en mémoire que le processeur peut y accéder en lecture ou en écriture. Le registre MA sert à implémenter l'adressage indirect puisqu'il permet d'écrire un programme où les adresses d'espaces mémoire ne sont plus données en absolu dans les instructions, mais sont évaluées dynamiquement durant l'exécution de celui-ci.

Le processeur à accumulateur avec registre `MA` est accessible sur Code Machine. Sur la fenêtre d'accueil de l'outil, sélectionnez l'option « Processor acc-ma » :

Le lien mène à la nouvelle architecture du processeur accumulateur, telle qu'illustrée à la **Figure 2**. La syntaxe de programmation de ce processeur est identique à celle du processeur à accumulateur simple et le jeu d'instructions présenté à la **Section 0** demeure valide pour cette architecture. Ainsi, ce processeur peut exécuter n'importe quel programme compatible avec la première version du processeur à accumulateur. De plus, le jeu d'instructions de ce processeur est étendu aux instructions données au **Tableau 2**. L'instruction `sub` permet de soustraire à la valeur présente dans l'accumulateur une valeur en mémoire. Les instructions `shl` et `shr` implémentent un décalage à gauche et à droite, respectivement. Finalement, le reste des instructions données au **Tableau 2** permettent de manipuler le registre `MA` de différentes façons.

Tableau 2 - Jeu d'instructions étendu du processeur à accumulateur

Instruction	Description
sub ADR	ACC ← ACC - Mémoire[ADR]
shl	ACC ← ACC << 1
shr	ACC ← ACC >> 1
adda ADR	MA ← MA + Mémoire[ADR]
suba ADR	MA ← MA - Mémoire[ADR]
addx	ACC ← ACC + Mémoire[MA]
subx	ACC ← ACC - Mémoire[MA]
lda ADR	MA ← Mémoire[ADR]
sta ADR	Mémoire[ADR] ← MA
ldi	ACC ← Mémoire[MA]
sti	Mémoire[MA] ← ACC

3.1 Partie 1 – Série Géométrique

Dans le cadre de cette troisième partie du travail pratique, nous allons procéder à une exploration détaillée des séries géométriques. Par définition, une série géométrique est une succession de nombres dans laquelle chaque terme, à l'exception du premier, est le produit du terme précédent par un coefficient constant dénommé « raison ». Ainsi, si le premier terme de notre série est désigné par a0, le n-ième terme an peut être exprimé par la relation :

$$a_n = a_0 \, r^{(n-1)}$$

où *r* représente la raison de la suite.

3.1.1 Objectifs et contraintes

L'exercice proposé vise à élaborer un algorithme permettant de calculer le n^e terme d'une suite géométrique, en s'appuyant sur un ensemble d'instructions prédéfini. Il est impératif de prendre en considération les contraintes de performance imposées, notamment la restriction à un maximum de 511 instructions pour l'exécution intégrale du programme.

3.1.2 Détails de l'exercice

Analyse algorithmique des suites géométriques (1.25 point)

Q1/0,75 point : Concevez un programme, en utilisant l'ensemble d'instructions fourni, destiné à implémenter la formule de calcul du Nième terme d'une suite géométrique, en partant d'un terme initial a0 et d'une raison r. Votre programme devra être optimisé pour réduire au minimum le nombre total d'instructions employées. Définissez et nommez ces 3 variables dans votre code.

Q2/0,25 point: Expliquez comment chaque terme de la suite est calculé et stocké dans votre programme, discutez ensuite des limitations de votre approche en termes de ressources et d'efficacité. Comment votre programme pourrait-il être amélioré si la contrainte de 511 instructions était levée ?

Q3/0,25 point: Traduisez votre algorithme dans le langage de programmation C.

3.2 Partie 2 – Syracuse

Pour ce premier exercice, il sera question d'implémenter une fonction permettant de calculer la [suite de Syracuse] d'un nombre entier $N > 0$. Cette suite est définie de la manière suivante :

$$u_0 = N$$

Et pour tout entier naturel n: $u_{n+1} = \begin{cases} \frac{u_n}{2} \; si \; u_n \; est \; pair \\ 3u_n + 1 \; si \; u_n \; est \; impair \end{cases}$

Suite de Syracuse pour *N* = 15

u_0	u_1	u_2	u_3	u_4	u_5	u_6	u_7	u_8	u_9	u_{10}	u_{11}	u_{12}	u_{13}	u_{14}	u_{15}	u_{16}	u_{17}	u_{18}	u_{19}	u_{20}	
15	46	23	70	35	106	53	**160**	80	40	**20**	**10**	5	16	8	4	2	**1**	4	2	**1**	…

Suite de Syracuse pour N = 15, Wikimedia Commons

La conjecture de Syracuse affirme que pour tout entier $N > 0$, il existe un indice n tel que $u_n = 1$.

Étant données les limitations sur code machine, votre code prend initialement en paramètre $u_0 = N$ et une variable *nombreDeTermes*. Par la suite, votre programme doit calculer le dernier terme et le stocker dans une variable nommée *awnser*.

Voici un exemple : $u_0 = 15$; *nombreDeTermes = 3 ; awnser = 0*

Première itération : 15 est impair, donc $u_1 = 46$

Deuxième itération : 46 est pair, donc $u_2 = 23$

Troisième itération : 23 est impair, donc $u_3 = 70$

On nous demande seulement 3 termes, donc nous nous arrêtons et stockons 70 dans la variable **`awnser`**.

Q1/0,75 point Réaliser la question sur code machine. Veuillez remettre la question en format ***.asm*** ou ***.txt***

Q2/0,50 point Réaliser la question en ***C***. Lisez bien les paramètres du code source donnée !

Votre programme en C doit conserver la même méthode de détection de parité que votre code sur Codemachine.