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) mais aussi d'exploiter le registre MA pour faciliter les accès en mémoire.

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 et les fichiers sources des exercices.
- Format: un rapport PDF. Incluez une page titre où doivent figurer les noms et matricules des deux membres de l'équipe, votre groupe de laboratoire, 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, pensez à incluez les réponses aux questions et des captures d'écran si requis.
- Copiez les programmes en assembleur dans des fichiers `.<ext>` sous la forme `<matricule1>-<matricule2>-<tp1_section_X_partie_Y >.<ext>`
    - `X` vaut 2 ou 3 selon que le programme soit demandé à la section 2 ou 3
    - Y vaut 1, 2 ou 3 selon que le programme soit demandé à la partie 1, 2 ou 3 de la section `X`
    - `ext` vaut `py` si votre programme est en Pyton, `asm` si c'est de l'assembleur.

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.

1.2 Barème

Le 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 remettrez.

Le TP1 comprend une question bonus de 0.25. Il est donc possible d'obtenir 4.25 pour ce premier travail pratique. Notez cependant que la note totale des travaux pratiques ne peut pas dépasser 20/20.

TP 1			**/4,00**
Section 2			
Partie 1			/1,00
	Q1	/0,25	
	Q2	/0,25	
	Q2	/0,25	
	Q3	/0,25	
Partie 2			/1,00
	Q1	/0,25	
	Q2	/0,25	
	Q3	/0,25	
	Q4	/0,25	
Partie 3			/0,50
	Q1	/0,25	
	Q2	/0,25	
Section 3			
Partie 1			/0,75
	Q1	/0,25	
	Q2	/0,25	
	Q2	/0,25	
Partie 2			/0,75
	Q1 — *bon fonctionnement*	/0,25	
	Q1 — *facilement modifiable*	/0,50	
Bonus			/0,25

2 Utilisation des instructions de branchement

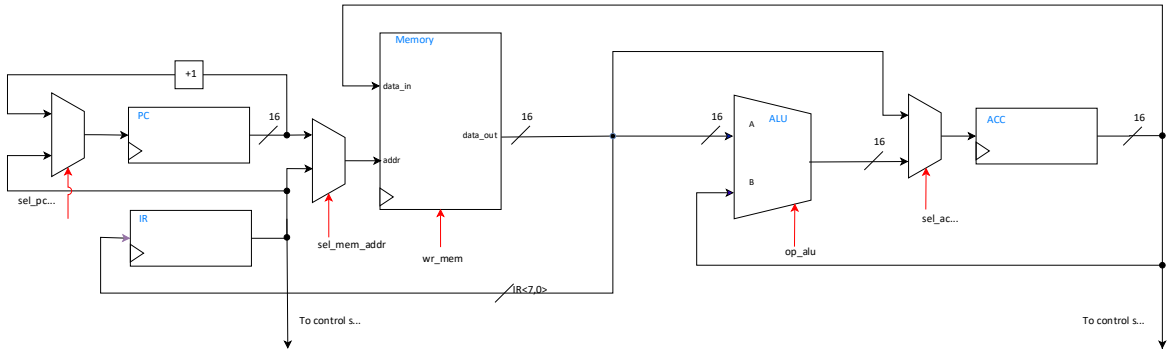

Figure 1 Architecture du processeur à accumulateur simple

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 ce code et collez le dans l'éditeur d'Assembleur de Code Machine. Suivez les instructions décrites en commentaires dans le programme pour initialiser les valeurs correspondantes à `valeur`, `seuil` et `constante`.

```
.text
main:
       ld  valeur
       sub seuil
       brz brancheA
       br  brancheB
brancheA:
       ld  valeur
       mul constante
       st  resultat
       br  fin
brancheB:
       ld  valeur
       sub constante
       st  resultat
fin:
       stop
.data
valeur:    0  // Remplacez 0 par la valeur de (MATR ETU 1 + MATR ETU 2) % 2021
seuil:     0  // Remplacez 0 par la valeur de (MATR ETU 1 + MATR ETU 2) % 2022
constante: 0  // Remplacez 0 par la valeur de MAX(MATR ETU 1, MATR ETU 2) % 10
resultat:  0
```

Q1/ 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.

Q2/ Quelles sont les structures de contrôle (`while, if/else, for`…) que ce programme fait intervenir ?

Q3/ Écrivez une version en **Python** du programme. Le code en Python devrait pouvoir être exécuté sans erreur et donner le même résultat que le code Assembleur.

Q4/ Quelle valeur se retrouve à l'emplacement mémoire `0x000F` à la fin de l'exécution de ce programme?

2.2 Partie 2 :

Copiez ce code et collez le dans l'éditeur d'Assembleur de Code Machine.

```
.text
l1:
       ld  valeur
       sub constante
       st  valeur
       ld  iteration
       sub constante
       st  iteration
       brz fin
       br  l1
fin:
       stop
.data
iteration:  8
valeur:    10
constante:  1
```

Q1/ 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.

Q2/ Quelles sont les structures de contrôle (`while, if/else, for`…) que ce programme fait intervenir ?

Q3/ Écrivez une version en **Python** du programme. Le code en Python devrait pouvoir être exécuté sans erreur et donner le même résultat que le code assembleur.

Q4/ Quelle valeur se retrouve à l'emplacement mémoire `0x000A` à la fin de l'exécution de ce programme ?

2.3 Partie 3 :

Q1/ À l'aide du jeu d'instructions étendu du processeur à accumulateur et des connaissances acquises grâce aux questions précédentes (structures de contrôle), écrivez un **programme assembleur** qui permet de calculer les 7 termes de la suite de Fibonacci : F_2, F_3, … F_8. Votre programme devrait être plus concis que celui écrit au TP0.

Q2/ Est-il possible de produire pour ce processeur un programme qui soit à la fois plus concis que celui du TP0 et qui permette d'écrire en mémoire chacun des termes calculés de la suite de Fibonacci ? Justifiez clairement votre réponse.

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:

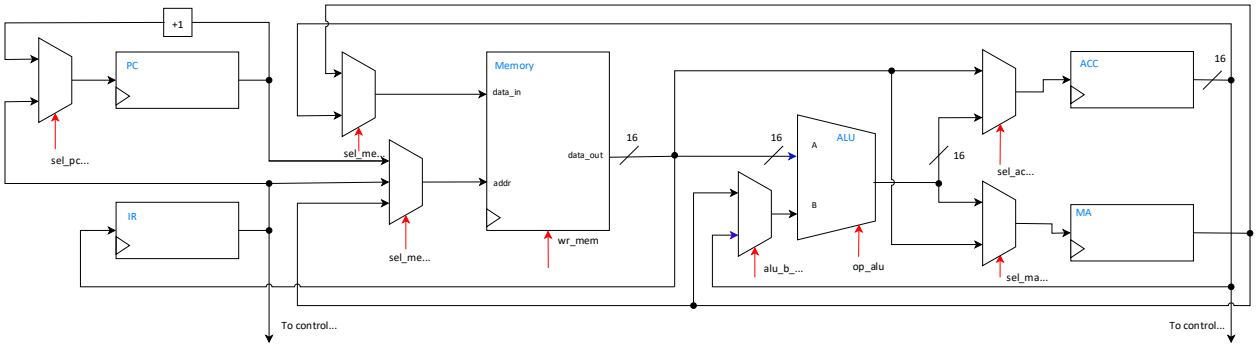

Figure 2 Processeur à accumulateur avec registre MA

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 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 espaces mémoires ne sont plus donnés en absolu dans les instructions, mais sont évalués 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 2** 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

Copiez ce code et collez le dans l'éditeur d'assembleur Code machine du processeur à accumulateur avec registre `MA`. Exécutez le code pour répondre aux questions suivantes.

```
.text
main:
       lda adr_tableau
       ldi
       add constante
       sti
l1:
       ldi
       adda constante
       add  constante
       sti
       ld iteration
       sub constante
       st iteration
       brnz l1
fin:
       stop
.data
constante: 1
iteration: 6
adr_tableau: tableau + 0
tableau: 0
```

Q1/ 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.

Q2/ Que se passe-t-il si on remplace `iteration` par `1000` ?

Q3/ Pourquoi est-il important que `tableau` soit déclaré en dernier dans ce programme ?

3.2 Partie 2

Q1/ À l'aide du jeu d'instructions étendu du processeur à accumulateur avec registre `MA` et des connaissances acquises grâce aux questions précédentes, écrivez un **programme assembleur** qui permet de calculer les 7 termes de la suite de Fibonacci : F_2, F_3, … F_8. Le programme doit être le plus court possible et il doit écrire en mémoire l'ensemble des termes calculés. Un 0.5 point est accordé à cette question si le code fourni peut être facilement modifié pour calculer les *M* termes de la suite de Fibonacci F_2 à F_M, et ce en changeant simplement la valeur d'une variable dans la partie `.data` du programme.

3.3 Partie 3 : Bonus

Soit la *suite de Syracuse* d'un nombre *N* est définie comme :

$$u_0 = N$$

$$u_{k+1} = \begin{cases} \dfrac{u_k}{2}, & \text{si } u_k \text{ est pair} \\ 3u_k + 1, & \text{si } u_k \text{ est impair} \end{cases}$$

La conjecture de Syracuse affirme que, pour tout entier $N > 0$, il existe un indice $k = K$ pour lequel $u_K = 1$.

À l'aide du jeu d'instruction étendu et des connaissances acquises grâce aux questions précédentes, écrivez un **programme assembleur** qui détermine la valeur de *K* de la *suite de Syracuse* du nombre *N* suivant :

N = `(MATR ETU 1 + MATR ETU 2) % 100`.

Dans votre rapport, indiquez les valeurs de *N* et de *K* trouvées.