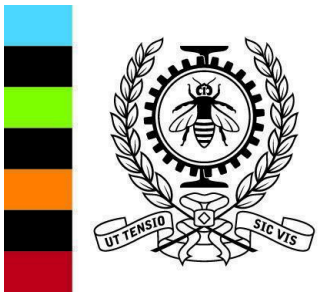

POLYTECHNIQUE
MONTRÉAL

INF1600
Architecture des micro-ordinateurs

Laboratoire 1
Section 4

Soumis par:
Tristan Cholette - 2261158
Moustafa, Hossam - 2279078

11 février 2024

Barème

TP1		**/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 Machine		/0.75
Q2 - C		/0.5

Section 2

Q1.0 : Indiquez votre valeur1 et de valeur2 ou il est marqué *ICI* dans le code.

valeur1 = 0 valeur2 = 3

Q1.1: Quelles sont les structures de contrôle utilisées au sein de ce programme ?

Les structures de contrôle du code se trouvent dans les sections de code associées aux étiquettes loop1(figure 1), end1(figure 2), et end2(figure 2). Le code sous l'étiquette loop1 est une boucle while, qui se répète par l'instruction "br loop1" à la fin, permettant de répéter l'exécution du code tant que la condition spécifiée n'est pas satisfaite (indiquée par "brz end1" ou "brz end2"). Ces instructions "brz" agissent comme des conditions dans la boucle while, dirigeant l'exécution vers end1 ou end2 en fonction des résultats de ces conditions. Les sections de code sous end1 et end2 servent à terminer le programme après avoir enregistré l'une des deux valeurs à l'adresse mémoire de la variable awnser. Ces sections peuvent être comparées à des blocs case ou des structures if, car elles définissent des chemins conditionnels en fonction des résultats des conditions sous l'étiquette loop1.

```
loop1:
ld temp_value1
sub one
brz end1
st temp_value1
ld temp_value2
sub one
brz end2
st temp_value2
br loop1
```

Figure 1 - Code de la loop 1

```
end1:
ld value1
st awnser
stop
end2:
ld value2
st awnser
stop
```

Figure 2 - Code de end1 et end2

Q1.2: 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 ?

Le contenu de la mémoire à l'adresse 0x0017 (figure 3) à la fin de l'exécution du programme est de 0x0003 ce qui équivaut à 3 en décimal.

Figure 3 - Mémoire à l'adresse 0x0017

Q1.3: 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.

Ce programme prend deux valeurs, assignées à value1 et value2, respectivement. Ces valeurs sont temporairement stockées dans temp_value1 et temp_value2. Ensuite, le programme entre dans la boucle loop1. Il décrémente temp_value1, et si le résultat est égal à zéro, le programme passe à la section end1, qui enregistre la valeur de value1 à l'adresse mémoire de la variable awnser, puis le programme s'arrête. Si le résultat n'est pas zéro, le programme décrémente temp_value2, et si le résultat est zéro, le programme passe à la section end2, qui enregistre la valeur de value2 à l'adresse mémoire de la variable awnser, et le programme s'arrête. Si aucune des valeurs temporaires n'atteint zéro, le code sous l'étiquette loop1 est réexécuté.

Il est à noter que si une des deux valeurs est initialement inférieure ou égale à zéro, cette valeur sera décrémentée à -1 avant d'être vérifiée pour égalité à zéro, et par conséquent, l'étiquette associée à cette valeur ne sera jamais atteinte. En fin de compte, ce code identifie la plus petite valeur positive et non nulle entre les deux valeurs initiales.

Q2.2: Identifier votre plage de possibilités et les raisons qui amènent cette étendue.

Nous savons que CodeMachine possède une capacité maximale d'éxecution de 511 cycles de types "fetch", "debug" et "execute". Nous savons aussi que notre code possède dans sa boucle "fibonacci" 12 lignes de commandes et 2 lignes de commandes sous l'étiquette de code "end". Chaque ligne de code nécessite une répetition de ces trois cycles pour se faire éxecuter. Sachant cela, nous pouvons établir que le nombre de cycle effectué pour trouver le nième nombre de fibonacci est égal à la formule suivante :

$$\text{Nombre total de cycles} = ((n-1)\times 12+4+2)\times 3$$

Le raisonnement derrière cette formule est que la dernière ligne (br fibonnaci) de la boucle "fibonnaci" n'est pas effectué à la dernière itération de cette boucle puisqu'à la dernière itération, on sort de cette boucle pour rentrer dans la boucle "end" grâce à la ligne "brz end" qui s'occupe de la gestion de la réponse. Nous pouvons donc affirmer grâçe à la formule ci-dessus:

$$n_{max} = (((511/3)-4-2)/12)+1 = 14.7$$

Ici, nmax représente le chiffre de la suite de fibonacci maximal que nous pouvons obtenir avant qu'on atteigne la capacité de cycles maximale de CodeMachine. Sachant cela, nous pouvons déduire que la plage de possibilités de notre code de suite de fibonacci peut aller du premier de la suite jusqu'au quatorzième chiffre de la suite (considérant que le premier chiffre de la suite quand n=1 est 0).

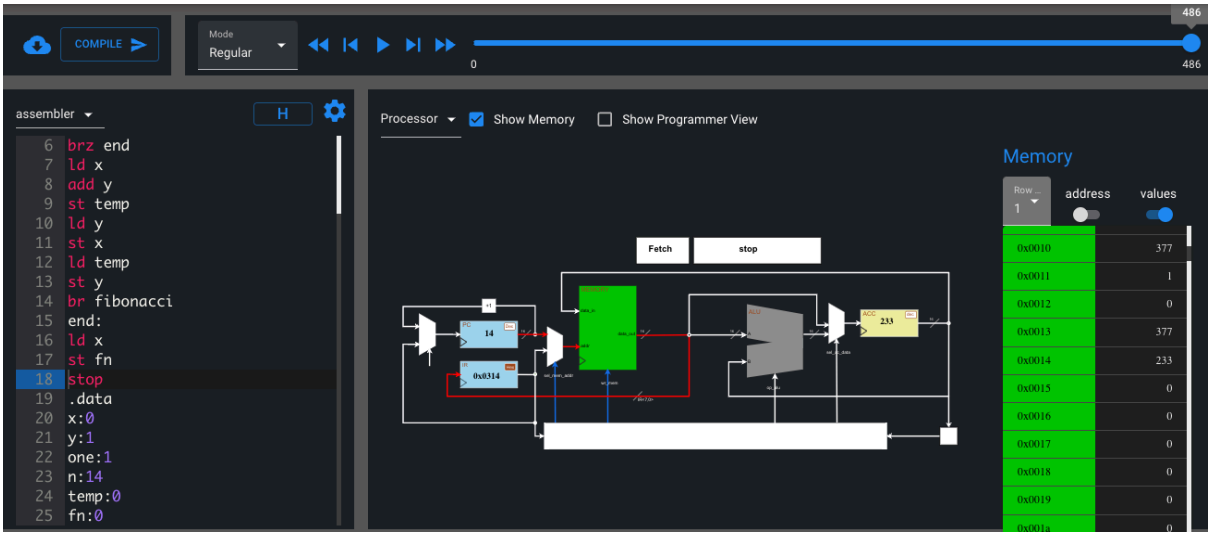

Figure 4 - Résultat au 14ième chiffre de la suite

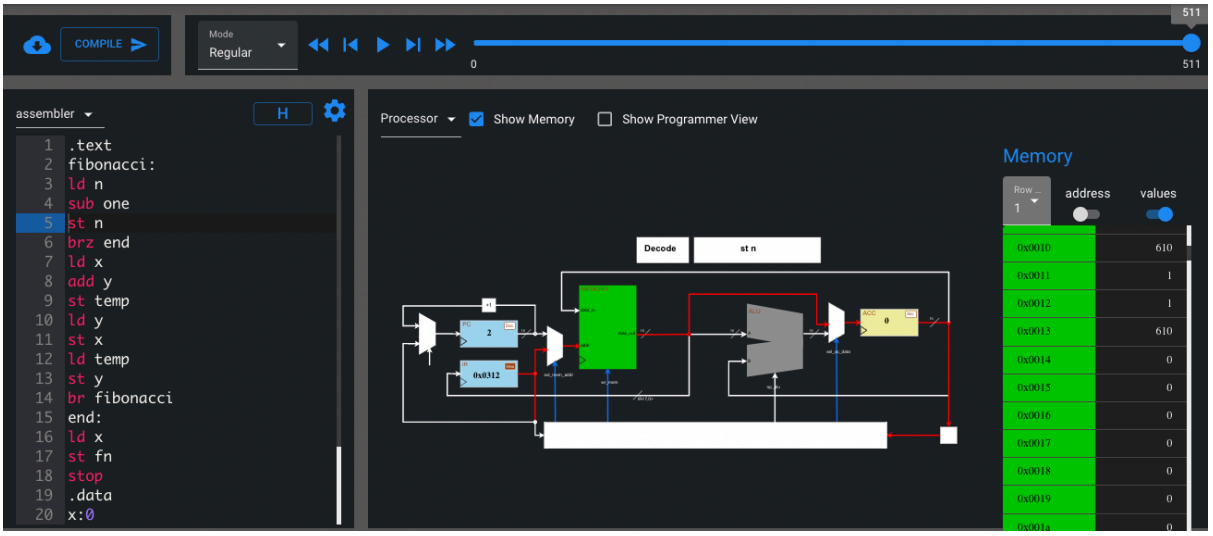

Figure 5 - Résultat au 15ième chiffre de la suite

On peut voir sur la figure 4 ci-dessus que le résultat enregistré à l'adresse 0x0014 pour le 14e chiffre de la suite est 233. On remarque que le nombre de cycles (486) est toujours inférieur à 511. Cependant, sur la figure 5, le résultat du 15e chiffre de la suite (377) n'est pas disponible à l'adresse du résultat (0x0014), qui est ici équivalente à 0. On remarque également que le nombre de cycles a atteint la limite de 511. On peut affirmer que ce résultat est dû au fait que nous avons atteint le nombre de cycles maximal de Code Machine, et que le code de la suite n'a pas pu terminer le calcul du quinzième chiffre. Notre formule de calcul de l'étendue est donc vérifiée.

Section 3

Q1.2: 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 ?

Notre programme utilise la valeur de la variable n pour enregistrer la progression de l'itération dans la boucle. La boucle principale loop1 itère à travers la valeur de n. À chaque itération, nous soustrayons 1 de n, multiplions la variable rTemp par la variable r, et stockons le résultat dans rTemp. Lorsque n atteint zéro, nous sortons de la boucle et multiplions la variable a1 par la valeur de rTemp. Cette valeur est ensuite enregistrée dans la variable an.

En ce qui concerne les limitations, notre code utilise 8 lignes de commandes dans la boucle loop1 et 3 lignes de commandes sous l'étiquette de code end. Chaque ligne de code nécessite une répétition des cycles de type "fetch", "debug" et "execute". Ainsi, le nombre total de cycles nécessaires pour calculer le nième terme de cette suite géométrique est donné par la formule :

$$\text{Nombre total de cycles} = ((n-1)\times 8+2+3)\times 3$$

Le raisonnement derrière cette formule est que la dernière ligne (br loop1) de la boucle loop1 n'est pas exécutée à la dernière itération, car on sort de cette boucle pour entrer dans la boucle end grâce à la ligne brz end, qui gère la sortie du programme.

Avec la contrainte de 511 instructions maximales de CodeMachine, nous pouvons calculer le n max tel que :

$$n_{max} = ((511/3) - 3 - 2)/8) + 1 = 22.08$$

Cela signifie que notre code de suite géométrique peut générer les termes de la suite jusqu'au vingt-deuxième terme avant d'atteindre la capacité maximale de cycles de CodeMachine. Sans la contrainte des 511 instructions maximales, nous pourrions définir la valeur de n (représentant le nième terme de la suite) à n'importe quelle valeur positive non nulle. Cependant, cela nécessiterait probablement une optimisation du code pour éviter les débordements de mémoire ou les erreurs de calcul pour des valeurs de n plus élevées. Il faudrait par exemple, allouer un plus grand espace mémoire pour pouvoir stocker les plus grandes valeurs tel que rtemp(valeur de r pour le nième chiffre de la suite) ou encore an(réponse) qui pourraient être de taille supérieure que l'espace que notre code leur alloue présentement.