INF1600

Travail pratique 0

Périphériques et architecture

Département de Génie Informatique et Génie Logiciel
Polytechnique Montréal

1 TABLE DES MATIERES

2 INTRODUCTION ET SOMMAIRE

Ce travail pratique a pour but de faciliter l'utilisation des outils employés dans le cadre du cours INF1600 pour les prochains travaux pratiques à venir. Il sera aussi question de travailler avec le langage C pour effectuer des manipulations de bits.

Ce travail pratique n'est pas à remettre. Nous vous invitons à demander aux chargés de valider vos réponses aux différents exercices. Les corrections de ces exercices seront disponibles la semaine de mise en ligne du travail pratique 1 pour que vous puissiez vérifier vos réponses.

3 UTILISATION DU DISCORD

Afin de faciliter les communications entre les étudiants, le professeur et les chargés de laboratoires, un Discord a été mis en place. Cliquez sur ce lien pour accéder à l'invitation. Nous vous demandons de modifier votre pseudo affiché sur le serveur en respectant le format suivant :

{Section lab} - {Prénom} {Nom}

Par exemple : 02 - Charles De Lafontaine

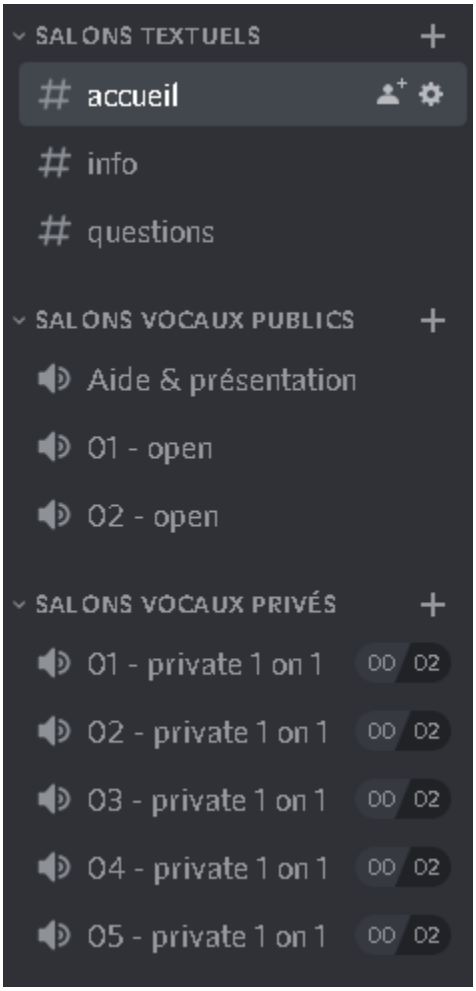

Figure 1. Affichage du Discord en INF1600 pour l'automne 2022

La **Figure 1** représente la structure générale du Discord. Les différents salons textuels ont une utilité bien définie et il est important de les respecter :

- **Accueil**: salon d'accueil des nouveaux membres.
- **Info** : différentes informations relatives aux travaux pratiques et aux cours seront partagées dans ce salon textuel. C'est au sein de ce salon que vous pouvez sélectionner votre groupe de laboratoire attitré.

Figure 2. Boutons permettant de sélectionner le groupe de laboratoire attitré

- **Questions** : posez toutes vos questions ici.

Nous vous demandons d'être courtois dans vos propos tenus sur le serveur Discord et de faire preuve d'un peu de patience. Les chargés se feront un plaisir de répondre à vos questions en temps voulu.
Nous vous encourageons fortement à discuter entre vous de la matière du cours, à vous entraider et à poser des questions pour toute incompréhension d'un énoncé. Cependant, faites attention au contenu que vous partagez afin d'éviter toute forme de plagiat.
N'hésitez pas à soumettre des suggestions d'amélioration du Discord auprès des chargés si vous trouvez que la structure du serveur manque de clarté.
Des salons vocaux sont à votre disposition pour vous retrouver en dehors des heures de laboratoires afin de travailler en groupe. Deux types de salons sont offerts : publics et privés.

4 UTILISATION DE L'OUTIL CODE MACHINE

Code Machine est un outil cherchant à faciliter la compréhension de concepts relatifs à l'architecture des ordinateurs et, plus généralement, des notions enseignées dans le cours INF1600. **Il est à noter que c'est un outil encore en phase de développement. Nous vous encourageons à donner des retours sur l'interface utilisateur et l'ergonomie générale de l'outil**.

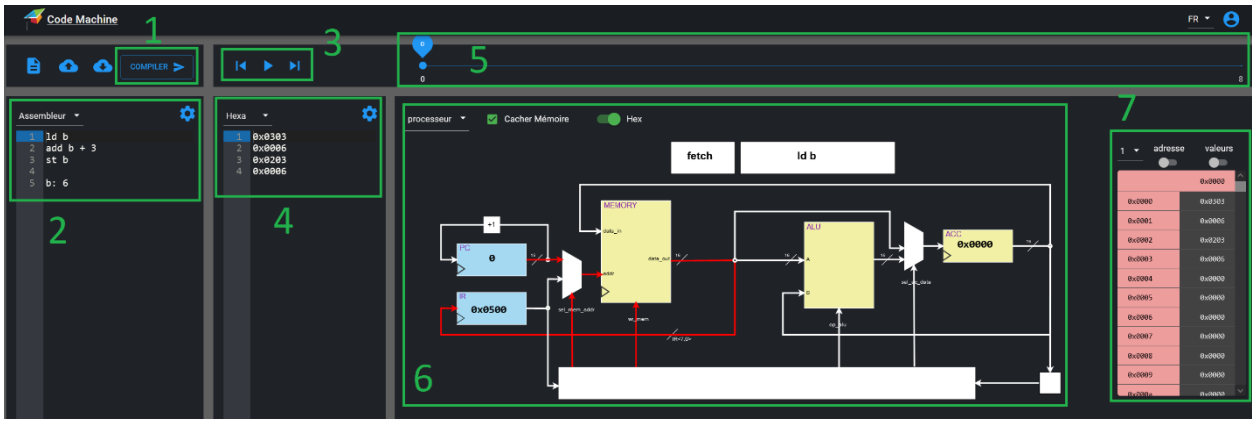

Figure 3. Affichage de Code Machine

Soit l'interface actuelle de Code Machine :

1. Le bouton de **compilation**. Votre code en pseudo-assembleur est envoyé à un serveur effectuant la compilation et la simulation. Après quelques secondes, le résultat est retourné à l'application.
2. **Éditeur** de code en assembleur. Nous vous demandons de bien respecter la syntaxe décrite plus bas afin d'éviter tout bug. **Évitez les espaces après vos instructions et les lignes vides.**
3. Boutons de **lecture** (respectivement **Reculer d'un cycle, Lecture/Pause, Avancer d'un cycle**).
4. Votre code **compilé** en hexadécimal.
5. La **frise chronologique** de la simulation où chaque étape correspond à un cycle d'exécution – vous pouvez saisir le curseur en bleu et vous déplacer librement dans la simulation.
6. Le **chemin des données** représentant le processeur accumulateur. Les signaux stimulés à l'instant *t* sont indiqués en rouge. Notez les différents blocs composant le processeur accumulateur.

7. Le **bloc mémoire** et son contenu. Vous pouvez modifier la disposition des cases mémoires et alterner entre un affichage décimal et hexadécimal.

Exemple d'utilisation :

- Rédaction d'un programme dans la partie (2) de l'interface.
- Compilation du programme en cliquant sur le bouton (1) de l'interface.
- Obtention du programme en hexadécimal, visible dans la partie (4) de l'interface.
- Manipulation de la frise chronologique (5).
- Observation des lignes stimulées dans la partie (6) de l'interface.
- Observation du contenu de la mémoire dans la partie (7) de l'interface.

En cas de doute sur une erreur au moment de la compilation, n'hésitez pas à communiquer avec un chargé pour qu'il puisse vous assister.

4.1 SYNTAXE DE PROGRAMMATION

Soit les instructions suivantes utilisables par le processeur accumulateur illustré sur Code Machine :

Instruction	Description
`add` **`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`

Tableau 1. Instructions en assembleur disponibles sur Code Machine et leurs descriptions

Le processeur accumulateur fonctionne de la façon suivante :

- Chargement (`ld`) dans le registre `ACC` d'une valeur contenue en mémoire à l'adresse `ADR`.
- Opérations (addition `add`, multiplication `mul`) entre la valeur courante dans le registre `ACC` et une valeur en mémoire à l'adresse `ADR`.
- Sauvegarde (`st`) de la valeur courante du registre `ACC` dans un espace mémoire à l'adresse `ADR`.

4.2 EXEMPLE DE PROGRAMME

```
1     ld   7
2     add  8
3     mul  7
4     mul  8
5     st   9
6     stop
7     3
8     5
9     0
```

Essayez de compiler le code ci-dessus et observez le comportement du processeur accumulateur tout au long de la simulation à l'aide de la frise chronologique et des boutons de lecture. Quel est le calcul mathématique (très simple) effectué par ce programme ? À quelle adresse mémoire le résultat de cette opération est-il sauvegardé ?

4.3 EXERCICE DE PROGRAMMATION : SUITE DE FIBONACCI ET ACCUMULATEUR

En vous appuyant sur l'exemple du programme précédent et sur les instructions disponibles pour le processeur à accumulateur, on vous demande de calculer les 7 termes ($F_2, F_3, F_4, F_5, F_6, F_7, F_8$) de la suite de Fibonacci.

Rappel :

$$F_0 = 0, F_1 = 1, F_n = F_{n-1} + F_{n-2}$$

On souhaite que les termes F_0 à F_8 soient stockés de façon consécutive (les uns à la suite des autres) en mémoire (peu importe l'emplacement de la mémoire de départ).

Conseil :

Réfléchissez aux termes de la suite devant figurer initialement en mémoire et faites attention à leur emplacement en mémoire (adresse). Que remarquez-vous dans la structure de votre programme ?

5 INSTALLATION D'UNE MACHINE VIRTUELLE LINUX

Linux est un système d'exploitation très utilisé en génie informatique et logiciel. Linux est disponible en différentes « saveurs » appelées distributions.
L'installation d'une machine virtuelle Linux vous permettra d'utiliser différents outils utiles dans le cadre des travaux pratiques du cours INF1600. Veuillez suivre les étapes décrites ci-dessous afin de mettre en place une machine virtuelle sur votre ordinateur :

- Téléchargez et installez *VMWare Workstation Player* à cette adresse, l'École dispose d'une entente pour la version pro de *VMWare Workstation*, cependant nous vous conseillons d'utiliser la version *Workstation Player* pour éviter tout souci de licence.
- Téléchargez un fichier ISO de votre distribution Linux préférée, par exemple Ubuntu (nous vous recommandons d'utiliser Ubuntu ou Fedora).
- Exécutez *VMWare Workstation Player* et sélectionnez « Create a New Virtual Machine » dans le menu de droite. Sélectionnez ensuite le fichier ISO de la distribution Linux à installer. Vous pouvez choisir les ressources allouées à la machine virtuelle (RAM, nombre de cœurs CPU), nous vous conseillons d'allouer *au moins* 2 Go de RAM à la machine virtuelle.
- Exécutez la machine virtuelle. Sur le bureau de la machine virtuelle, cliquez sur l'icône en forme de CD.
- Installez les dépendances suivantes qui vous seront utiles pour les prochains TPs de INF1600 en exécutant les commandes suivantes dans le terminal Linux :

```
$ sudo apt install make
$ sudo apt-get install libc6-dev-i386
```

6 QUELQUES NOTIONS DE PROGRAMMATION EN C

Le langage de programmation C est un langage proche de la machine, très utilisé dans le développement des systèmes embarqués (le premier projet intégrateur de GIGL utilise des notions de langage C pour le développement du *firmware* du robot autonome).

Nous souhaitons introduire quelques particularités du langage C au travers d'exercices impliquant **l'allocation dynamique**, la fonction **`printf`** et ses particularités, ainsi que la **manipulation de bits**.

Nous vous fournissons un programme de base pour les trois programmes différents. Dans le dossier `src`, vous trouverez un dossier respectif pour chaque question. Afin de compiler et d'exécuter un programme, déplacez-vous dans le dossier de la question (où se trouve un fichier `makefile`) à l'aide de la commande Linux `cd [chemin vers le dossier]` ou en effectuant un clic droit dans le dossier puis en sélectionnant « Open in Terminal » puis entrez les commandes suivantes :

```
$ make clean
$ make
$./exec
```

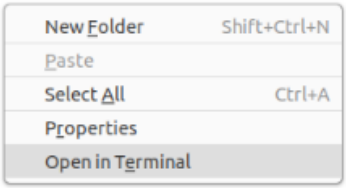

Figure 4. Option d'ouverture d'un terminal avec un clic droit

6.1 EXERCICE 1 EN C – UTILISATION DES FONCTIONS PRINTF ET SCANF

De façon très similaire à Python, on peut afficher des valeurs dans la console à l'aide de la fonction `printf` en C. Cependant, `printf` a une façon de fonctionner différente de `print` en Python. En effet, `printf` utilise un principe de buffer (tampon). Le contenu passé en paramètre de `printf` est envoyé dans un tampon, et le contenu n'est affiché que lorsqu'on atteint la terminaison du programme OU que l'on ajoute un caractère de fin de ligne comme `\n` à la fin du contenu à afficher, par exemple :

```
int main(){
   printf("Bonjour le monde!\n"); // Affiche le message suivi d'un saut
de ligne
   printf("A");
   printf("B");
   printf("C\n "); // Affiche ABC suivi d'un saut de ligne
   return 0;
}
```

On peut aussi afficher une valeur numérique (`int, float, unsigned, …`) à l'aide de la méthode `printf` ainsi qu'une « chaine de caractères ». Pour ce faire, il faut préciser dans la chaine de caractères passée en paramètre le type de données que l'on souhaite afficher, et passer la valeur à afficher comme second paramètre :

```
printf("Valeur : %d\n", 123456); // Affiche Valeur : 123456
```

Le préfixe % indique à `printf` qu'il reçoit des instructions de formatage. Par exemple, le caractère `d` précédé de `%` indique que nous souhaitons afficher un entier en décimal.

Voici une liste brève des paramètres de formatage qui nous serons utiles :

`%c`	`Caractère (char)`
`%s`	`Chaine de caractères (char *)`
`%d`	`Décimal (int/unsigned)`
`%x`	`Hexadécimal (int/unsigned)`
`%f, %g`	`Nombre en virgule flottante (float)`

Tableau 2. Divers paramètres de formatage à votre disposition

On peut afficher un grand nombre de valeurs, en utilisant `%` et en précisant les valeurs à afficher les unes à la suite des autres, séparées par des virgules.
Similairement à input en Python, on peut utiliser **scanf** en C pour lire une entrée utilisateur :

- `scanf("%type", &variable)` : on précise de façon similaire le type à interpréter, comme avec `printf`, et on passe l'adresse de la variable (avec le préfixe `&` devant le nom de la variable) qui doit recevoir l'entrée de l'utilisateur.

Exemple d'utilisation de `scanf` :

```
int main(){
   int valeur = 0;
   printf("Entrez une valeur \n");
   scanf("%d", &valeur); // %d pour lire un int, vers valeur
   return 0;
}
```

Question :

1) Écrivez un programme en C qui lit deux entrées d'utilisateur (`int`), `val1` et `val2`, et qui calcule le PGCD entre ces deux valeurs. Le PGCD est stocké dans une variable nommée `pgcd` et il est ensuite affiché dans la console.

6.2 EXERCICE 2 EN C – UTILISATION DE L'ALLOCATION DYNAMIQUE AVEC MALLOC, CALLOC, REALLOC ET FREE

L'allocation dynamique permet d'allouer dynamiquement de la mémoire pendant l'exécution du programme, sur le tas (`heap`). En langage C, on peut travailler avec des pointeurs, c'est-à-dire un objet qui pointe vers une donnée, en contenant l'adresse de cette donnée en mémoire.
Dans le langage C, on ne peut pas directement déclarer un tableau comme on le ferait en C++ ou en Python. En revanche, on peut déclarer un pointeur et associer à ce pointeur une allocation de mémoire correspondant au nombre d'éléments que l'on souhaite dans notre « tableau ».
Les fonctions `malloc` (*memory* allocation) et `calloc` (*contiguous* allocation) en C permettent d'allouer un certain espace en mémoire afin d'y faire contenir le nombre d'éléments voulus.

- `malloc(tailleEnBytes)` : on précise dans le paramètre unique le nombre de bytes (par convention, 1 byte = 8 bits) à allouer dynamiquement dans le tas. La fonction retourne un pointeur vers l'espace mémoire alloué.
- `calloc(nbrElem, tailleElem)` : on précise dans les deux paramètres respectivement le nombre d'éléments qui seront alloués et la taille d'un élément. Chaque élément est initialisé à 0. La fonction retourne un pointeur vers l'espace mémoire alloué.

Exemple d'utilisation de `malloc` et `calloc` :

```
int main(){
   int* pointeurMalloc; // int suivi * => pointeur vers int
   int* pointeurCalloc; // on pourrait aussi déclarer char* => pointeur
char

   pointeurMalloc = malloc(10*sizeof(int)); // sizeof retourne la taille
d'int
   pointeurCalloc = calloc(10, sizeof(int)); //noter la diff de
paramètres
   pointeurMalloc[0] = 5; // le premier élément du tableau est défini à
5
   pointeurCalloc[1] = 6; // le second élément du tableau est défini à
6
}
```

Le programme ci-dessus définit deux pointeurs, et utilise les deux méthodes `malloc` et `calloc` en précisant les paramètres respectifs de chaque méthode.
Il se peut que la taille du tableau ne soit pas suffisamment grande, on aimerait alors allouer à nouveau de l'espace sans perdre les éléments actuels du tableau. Pour cela, on va utiliser la fonction `realloc` telle que :

- `realloc(pointeur, nouvelleTaille)` : on précise le pointeur affecté par la réallocation, et la nouvelle taille de l'espace mémoire pointé par ce pointeur (en bytes)

Pour finir, il est très important de libérer l'espace mémoire alloué dynamiquement avant la terminaison du programme. Pour ce faire, on utilise la fonction `free` telle que :

- `free(pointeurALiberer)` : on précise le pointeur dont on veut libérer l'espace mémoire alloué

Lorsque vous employez l'allocation dynamique, n'oubliez pas de désallouer l'espace mémoire, c'est très important afin d'éviter les fuites de mémoire (espace mémoire non utilisé mais dont on ne dispose plus l'accès).

Questions :

1) Écrivez un programme en C, à l'aide de de `printf` et `scanf` vus précédemment, qui demande à l'utilisateur d'entrer le nombre d'éléments que doit contenir un tableau alloué dynamiquement. Vous êtes libre d'utiliser `malloc` ou `calloc`.
2) Complétez votre programme de telle sorte à ce que l'on demande à l'utilisateur les éléments à définir à l'intérieur du tableau après son allocation dynamique. L'utilisateur doit être en mesure d'entrer autant de valeurs qu'il le souhaite, la taille du tableau doit être AU MOINS 2 fois supérieure au nombre d'éléments actuellement présents dans le tableau. Lorsque l'utilisateur entre -1, on souhaite arrêter le remplissage et afficher TOUS les éléments du tableau.
3) Complétez votre programme pour libérer l'espace mémoire alloué dynamiquement à la fin de l'exécution.

6.3 EXERCICE 3 EN C – MANIPULATION DE BITS

En C, on peut effectuer des opérations logiques (ET, OU, OU EXCLUSIF…) à l'aide d'opérateurs associés, respectivement `&, |, ^.` On peut aussi effectuer des manipulations de bits, par exemple des décalages vers la droite et la gauche à l'aide de `>>` et `<<.`

En C, si l'on souhaite manipuler un ensemble de 8 bits, on peut utiliser un `char`. Pour manipuler 16 bits, on utilise un `short`, et pour une valeur de 32 bits on utilise un `int`. On peut initialiser une variable à l'aide de sa valeur sous forme de vecteur binaire ou forme hexadécimale :

```
char  val8bits    = 0b00001111;                          // 15
char  val8bitsHex = 0x0F;                                // 15
short val16bits   = 0b0000000011110000;                  // 240
int   val32bits   = 0b00000000000000000000000000000001;  // 1
```

Les entiers en C peuvent être traités comme des vecteurs de bits, sur lesquels il est possible d'appliquer des masques. Pour rappel, un masque permet d'isoler/sélectionner certains bits d'intérêt dans un ensemble de bits donné. Par exemple, si on souhaite isoler le bit le moins significatif (LSB), il est possible d'appliquer la valeur 1 comme masque :

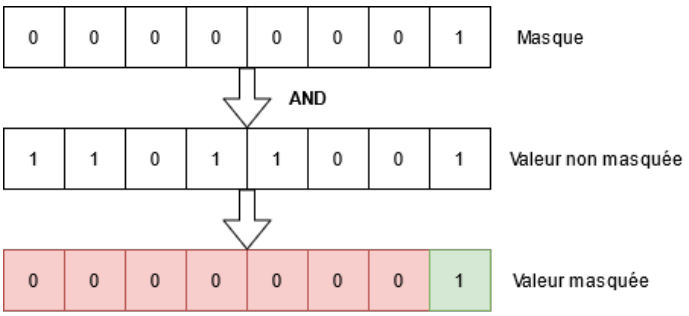

Figure 5. Exemple de masque

En effectuant un ET logique entre le masque `0b00000001` et la valeur `0b11011001`, on obtient la valeur masquée `0b00000001` : seul le bit le moins significatif est conservé. En langage C, cela donnerait :

```
int main(){
   char masque           = 0b00000001;
   char valeurNonMasquee = 0b11011001;
   char valeurMasquee    = masque & valeurNonMasquee; // 0b00000001
   return 0;
}
```

Pour effectuer des décalages vers la droite ou vers la gauche, on utilise les opérateurs >> et << tels que :

- `valeur << décalage` : `valeur` est décalée vers la gauche autant de fois que spécifié par `décalage`;
- `valeur >> décalage` : `valeur` est décalée vers la droite autant de fois que spécifié par `décalage`.

Pour illustrer le décalage vers la droite ou la gauche :

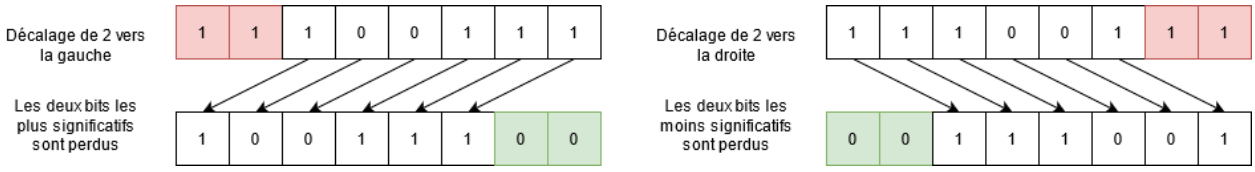

Figure 6. Illustration d'un décalage vers la droite et vers la gauche

Question :

1) Compléter la fonction `parité` prenant, en paramètre, une valeur de 8 bits (`char`), et qui détermine si le nombre de bits à 1 est pair ou impair. Dans le cas où le nombre est pair, affichez sur la console : « `nombre de bits à 1 pair` », sinon, affichez « `nombre de bits à 1 impair` ».
   Indice : utilisez l'opérateur ^ (OU EXCLUSIF). Vous devez utiliser les concepts abordés antérieurement.