Le contenu des TD change : après avoir vu tout ce qui concerne le codage des nombres entiers, nous allons apprendre peu à peu à programmer en langage machine, avec un assembleur. Cela vous permettra de comprendre le travail d'un compilateur qui est de traduire un programme en langage de haut niveau vers un programme exécutable en langage machine.

Nous allons utiliser un simulateur d'unité centrale appelé CimPU. Cette UC est très simplifiée par rapport à un véritable processeur, car son objectif est pédagogique avant tout. Ce simulateur fonctionne en principe dans n'importe quel navigateur moderne (signaler tout bug constaté).

1. Découverte de CimPU

👉 Ouvrez l'URL https://perso.univ-rennes1.fr/pierre.nerzic/IntroArchi/CimPU/.

Agrandissez la fenêtre (ou zoomez) pour travailler confortablement.

1.1. Parties de CimPU

La page web est composée de plusieurs parties.

1.1.1. Introduction

En haut, il y a une présentation générale et dans le paragraphe *Niveau de CPU*, on peut choisir le « niveau de complexité ». Pour le premier TD, vous resterez sur le niveau 1. Chaque niveau rajoute des fonctionnalités de plus en plus complexes : nouvelles instructions, nouveaux registres, nouveaux mécanismes...

👉 Passez (ou restez) en niveau 1.

1.1.2. Périphériques

Ensuite, il y a un paragraphe sur les périphériques. Dans CimPU, ils sont très simples : DSKY = display + keyboard, juste des affichages et des zones de saise très simples. Utilisez la boîte déroulante pour voir chacun. À chaque choix que vous faites, le dessin change dans l'Unité Centrale plus bas.

👉 Mettez le DSKY en mode 0.

1.1.3. Unité centrale (schéma de CimPU)

En dessous, il y a un grand schéma de l'unité centrale comprenant aussi le DSKY et la RAM, mais dans un ordinateur réel, ces deux éléments sont extérieurs au processeur. Vous pouvez passer la souris au dessus des éléments pour avoir une bulle d'information.

Les fils rouges, vert et bleu sont appelés bus. Ils transportent des informations codées en base 2.

- Les fils rouges constituent le bus de données. Il transporte des octets qui sont traités par l'unité centrale (UC).
- Les fils verts constituent le bus d'adresse. Il indique l'adresse de l'octet en cours de transfert entre l'UC et la mémoire ou le DSKY.
- Les fils bleus sont le bus de commande ou de contrôle. Normalement ce bus est omniprésent. Il relie tous les circuits entre eux. Ici il n'a été dessiné qu'entre le décodeur et l'Unité

Arithmétique et Logique (UAL). C'est le bus de commande qui active des sortes de portes entre les éléments, comme les portes d'une écluse, qui permettent de transférer des octets d'un endroit à l'autre.

En bas du schéma, il y a un panneau de boutons.

- Le bouton `Reset` met tous les registres de l'UC à zéro, `IP`, `R0` et `R1`. On appuie sur ce bouton avant toute nouvelle exécution, pour repartir sur des bases saines.
- Le bouton déroulant `Animations` permet de choisir la vitesse des animations lors de l'exécution. Ça va servir seulement dans ce TD.
- Les boutons `STEP` et `NEXT` permettent d'exécuter le programme une instruction à la fois. L'exécution s'interrompt à la fin de chaque instruction.
- Le bouton `RUN` lance l'exécution du programme à pleine vitesse (entre 20 et 60 par seconde, selon votre PC).
- Le bouton `STOP` arrête l'exécution.

1.1.4. Messages

En dessous du schéma, vous verrez une fenêtre de messages. Ils indiquent comment chaque instruction est exécutée, avec plus ou moins de détails. Le bouton de menu permet de choisir le niveau de détail des messages.

1.1.5. Mémoire centrale

En dessous des messages ou à côté de l'Unité Centrale (selon la taille de l'écran), vous verrez le contenu de la mémoire. Il y a deux modes d'affichage. Ce sont les onglets `Vue en tableau` et `Contexte d'exécution`.

👉 Passez (ou restez) en mode `Vue en tableau` pour la suite

La mémoire de CimPU contient 256 octets. Ils peuvent être arrangés, pour l'affichage, en 16 lignes de 16 octets ou 32 lignes de 8 (menu Affichage).

👉 Passez en mode 16 octets par ligne. Dézoomez s'il y a un problème d'affichage.

Vous devez comprendre comment les octets sont rangés. Vous avez des lignes numérotées `00`, `10`, `20`..., `F0`. Ce sont les « dizaines » en base 16. Et voyez les colonnes numérotées `00`, `01`, `02`..., `0F`. Ce sont les unités en base 16. Pour avoir l'adresse d'une case, il faut juste additionner ces deux nombres. En mode 8 octets par ligne, c'est un peu plus compliqué.

👉 Quelle est l'adresse de l'octet situé 3e ligne, et 5e colonne ? Où est situé l'octet d'adresse 56 en hexadécimal : ligne ? et colonne ?

Le tableau affiche la valeur de chaque octet en hexadécimal et aussi en ASCII dans la colonne `texte`. Ces valeurs peuvent être éditées à tout moment. Dans la partie texte ASCII, les points à mi-hauteur · signifient que le code ne correspond pas à un caractère ASCII, ou qu'il n'est pas affichable.

👉 Cliquez sur la ligne `50`, colonne `03` et saisissez `45`. Vous voyez apparaître un `E` côté texte ASCII. C'est parce que le code ASCII de `E` est 45 en hexadécimal.

La table ASCII est disponible sur internet, par exemple ASCII Chart. Ce tableau donne la signification des codes. Par exemple, le code `0A` (hexadécimal) : *line feed*, c'est à dire changement

de ligne.

☛ Cliquez en dessous du `E` dans la partie texte ASCII et tapez le mot « `Bonjour !` » (sans vous soucier du nombre de caractères de la ligne). Vous verrez les codes de ces caractères dans la partie gauche. Quel est le code du caractère `j` ? Quel est le code du caractère espace ?

Vous comprenez comment les textes ASCII sont écrits dans la mémoire d'un ordinateur ou un fichier : il y a un octet par caractère. Tous les signes de ponctuation et aussi les sauts de lignes sont codés selon cette norme.

C'est un peu le même principe pour des textes Unicode en UTF8. La norme Unicode regroupe plus d'une centaine de milliers de caractères. Voir cette liste et les pages suivantes. Le mot `U+nnnn`, ex: `U+0042` pour `B`, s'appelle le « point de code ». C'est l'identifiant du caractère.

La norme UTF8 indique comment les caractères Unicode sont codés sous forme d'octets. Les caractères occidentaux de base (*latin-1*) sont codés sur 1 octet comme en ASCII et les caractères spécifiques comme les lettres accentuées demandent deux à quatre octets, selon leur rareté. Par exemple un `é` est codé `C3` suivi de `A9`.

2. Premier programme

2.1. Saisie de codes d'instruction

On va faire une expérience qu'ont vécue les tout premiers informaticiens. À l'époque, les programmes devaient être entièrement conçus sur papier et saisis code par code dans l'ordinateur, avant l'exécution.

☛ Mettez-vous devant la RAM et saisissez les valeurs suivantes à l'adresse `00` (au tout début de la mémoire).

```
02 B5 72 00 5F
```

NB: si vous saisissez trop de valeurs pour une ligne, les excédentaires à droite seront enlevées. Et si vous en saisissez trop peu, les anciennes valeurs de droite seront préservées.

C'était comme ça qu'on faisait avec les premières calculatrices programmables (TI-58, HP-41...). Il y avait des magazines qui donnaient des listes de codes à saisir pour obtenir un programme sur la calculatrice.

Les codes que vous avez tapés sont des nombres en hexadécimal qui représentent des instructions :

- `02` est le code d'une instruction appelée `LD R0, constante`. Ces mots signifient : *load* càd charger (affecter) le registre `R0` avec la constante. Cette constante est dans l'octet suivant : `B5`. Ce nombre est appelé « opérande » de l'instruction (`R0` est aussi un opérande, mais il est indiqué dans le code de l'instruction).
- `72` est le code de l'instruction `OUT R0, port`. Le port est dans l'octet suivant, `00`. Donc ici, on demande à l'UC d'envoyer la valeur de `R0` sur le « port 0 ». Les ports sont des sortes d'adresses pour les périphériques. Dans CimPU, le port 0 est celui des leds.
- `5F` est le code de l'instruction `HLT`, sur un seul octet. Elle demande à l'UC de s'arrêter.

Il ne faut surtout pas apprendre ces codes, mais seulement comprendre que chaque octet représente une instruction différente. C'est comme ça que le processeur reconnaît les actions qu'il doit faire.

2.2. Exécution à pleine vitesse

Si vous n'avez pas fait d'erreur de saisie, revenez devant le schéma de CimPU.

☛ Dans le menu `Animations`, choisissez `Aucune`.

☛ Cliquez sur le bouton `Reset` pour réinitialiser l'UC (mettre `IP` à zéro).

☛ Lancez l'exécution avec le bouton `RUN`. Si le DSKY est bien en mode 0, vous verrez s'allumer des leds.

☛ Passez la souris au dessus de `R0` dans la partie *Registres*. La bulle affiche les valeurs dans différents codages : hexa, binaire, décimal non signé, signé et ASCII.

☛ Cliquez sur `R0` et changez sa valeur, mettez par exemple `AF`. Ensuite, changez la valeur de `IP` et mettez `02` (pas autre chose !). Puis cliquez sur le bouton `RUN`. Les leds allumées sont celles qui correspondent à l'écriture binaire de `AF`.

Le fait de changer `IP` indique où il faut commencer l'exécution quand on clique sur `RUN`. L'instruction en `02` est le `OUT R0,0`. Évidemment, si vous mettez une valeur fantaisiste dans `IP`, l'exécution partira de là, et il arrivera ce qui arrivera : rien car la mémoire est vide, mais c'est une chance. Sur un PC Windows, changer IP pour n'importe quoi peut entraîner un écran bleu, parce que la mémoire n'est pas vide et tout n'est pas exécutable. Sur Linux, le système vous dira qu'il y a un incident ou une violation de segmentation.

2.3. Exécution étape par étape

On va s'intéresser à l'exécution d'une instruction. C'est fait en plusieurs étapes.

1. La valeur de `IP` est l'adresse du prochain code d'instruction à récupérer. Ce code est placé dans le décodeur d'instruction. Il y a un registre appelé `IR` (*instruction register*). Il est affiché en binaire à gauche.
2. `IP` est incrémenté après avoir été chercher le code de l'instruction.
3. L'instruction est décodée. C'est à dire que son code est identifié : c'est ceci, ou c'est cela, d'après le poids fort du code. Le type des opérandes est également identifié : c'est tel registre, c'est avec une constante...
4. Comme il y a un opérande « immédiat », c'est à dire une constante dans l'octet suivant, le séquenceur va chercher cet octet pour le placer dans `R0`.
5. `IP` est incrémenté après avoir été chercher l'opérande.

Parfois, les opérandes sont stockées temporairement dans un registre `OpA`. C'est le cas de l'instruction `OUT`. `OpA` mémorise des adresses citées dans les instructions.

☛ Dans le menu `Animations`, choisissez `Lente` (au début, puis `Normale` ou `Rapide` quand vous avez compris).

☛ Dans le menu `Messages`, choisissez `Niveau 3`.

☛ Cliquez sur le bouton `Reset` puis cliquez sur le bouton `STEP` pour voir l'exécution de chaque étape d'une instruction. Regardez les messages qui arrivent pour expliquer chaque étape.

Notez que `IP` est toujours en avance d'un octet. Il est après le code d'instruction, puis après l'opérande. Il est toujours incrémenté juste après avoir consulté la mémoire. Ainsi, à la fin d'une instruction, il est devant la suivante.

👉 Cliquez sur le bouton `Reset` puis cliquez sur le bouton `NEXT` pour voir chaque instruction.

👉 Désactivez les animations et les messages d'information. L'exécution de chaque instruction sera maintenant à pleine vitesse sans messages.

3. Deuxième programme

On voudrait écrire un programme capable de demander deux nombres, les additionner et afficher le résultat. On va le créer en plusieurs étapes, et on va utiliser l'« assembleur ». C'est un outil qui permet de créer des programmes sans connaître les codes des instructions.

3.1. Version 1

La première version consiste à simplement demander un nombre à l'utilisateur et à le lui réafficher.

On va utiliser le DSKY en mode 10. Dans ce mode, il y a 4 lignes (fond gris clair) pour afficher des valeurs en base 10, 16… et une ligne (fond blanc) pour saisir une valeur. Le programme écrit une valeur par un `OUT reg, 10` et il peut lire une valeur par `IN reg, 10`. La lecture est bloquante tant que l'utilisateur n'a pas tapé quelque chose de correct.

👉 Mettez le DSKY en mode 10.

👉 Copiez le programme suivant en cliquant sur ce bouton noir/bleu → 

```
        IN  R0, 10          ; lecture d'un octet sur le terminal 10
        OUT R0, 10          ; affichage d'un octet sur le terminal 10
        HLT                 ; fin du programme, arrêt du CPU
```

👉 Collez ce source dans l'assembleur (CTRL-A, CTRL-V).

👉 Menu `Source/Effacer la mémoire et assembler`. Le début de la mémoire centrale s'est rempli avec de nouveaux codes. La couleur orange clair indique que ce sont des instructions.

👉 Cliquez sur les boutons `Reset` puis `RUN`. Le cadre du DSKY s'allume en rouge, et la ligne de saisie en rose. Entrez une valeur, comme 12 ou 223. Elle s'affichera en haut du terminal (`?` pour vos saisies, `>` pour les affichages).

NB: dans ce terminal, les valeurs sont saisies avec la même syntaxe que dans le source en assembleur : juste des chiffres pour une valeur en base 10, ou `$` suivi de chiffres hexa, ou `%` suivi de chiffres binaires, ou un caractère entre `'...'` ou `"..."`. On peut tous les faire précéder d'un signe moins pour un entier signé en convention $C2^8$.

3.2. Version 2

On rajoute une petite complication (copiez-collez ce source dans l'assembleur) : 

```
        IN  R0, 10          ; lecture d'un octet
        INC R0              ; incrémentation de R0
        OUT R0, 10          ; affichage d'un octet
        HLT                 ; fin du programme
```

L'instruction `INC R0` incrémente la valeur de `R0`.

👉 Assemblez, puis cliquez sur les boutons `Reset` puis `RUN`. Entrez une valeur...

NB: le menu `Source/Effacer la mémoire et assembler` est à utiliser quand vous changez beaucoup de choses et qu'il y a un risque que la mémoire soit mélangée avec des restes de programmes précédents.

👉 Il y a d'autres instructions dans le genre de `INC`. Essayez :

- `DEC R0` : décrémente `R0`.
- `NEG R0` : change le signe de `R0`, considéré en convention $C2^8$. Essayez de saisir 0, et d'autres valeurs positives et négatives.
- `NOT R0` : inverse les bits de `R0`. Pour le voir, saisissez plutôt des valeurs en binaire, avec un `%` devant.

Un prochain TD proposera d'étudier les instructions logiques.

3.3. Version 3

C'est le programme voulu au début :

```
        IN  R0, 10          ; lecture d'un octet dans R0
        IN  R1, 10          ; lecture d'un octet dans R1
        ADD R0, R1          ; addition de R0 avec R1, résultat dans R0
        OUT R0, 10          ; affichage de la somme qui est dans R0
        HLT                 ; fin du programme
```

👉 Assemblez, puis cliquez sur les boutons `Reset` puis `RUN`.

L'instruction `ADD reg1, reg2` réalise : reg1 = reg1 + reg2. Vous constaterez qu'elle additionne aussi bien des entiers signés que des non-signés. Vous comprenez que c'est à la conception du programme de savoir quelle est la convention utilisée, en particulier quand on affiche un octet. Le terminal de CimPU affiche toutes les interprétations d'un même octet : en tant que nombre entier non signé, entier signé, hexadécimal, binaire, code ASCII...

👉 Voici une séquence pour voir comment se passe un calcul :

1. Cliquez sur le bouton `Reset`,
2. Dès maintenant saisissez une valeur (clic dans le terminal, saisie de chiffres, touche entrée),
3. Appuyez sur `NEXT` (explication: l'instruction `IN` est bloquante tant qu'une valeur n'a pas été saisie, donc le bouton `NEXT` ferait du sur-place si on n'avait pas saisi une valeur avant).
4. Vérifiez que 1) R0 contient la valeur saisie, et 2) IP vaut `02`.
5. Mettez les animations en vitesse `Normale`,
6. Cliquez plusieurs fois sur le bouton `STEP` pour voir comment l'instruction `ADD R0,R1` est exécutée. Vous allez voir que la valeur de `R0` est transférée dans l'UAL, l'ordre de calcul est envoyé par le décodeur, puis le résultat est remis dans `R0`.

👉 Remettez les animations en mode `Aucune`.

👉 Juste pour essayer la soustraction, remplacez l'instruction `ADD R0,R1` par `SUB R0,R1`, et testez différents nombres.

CimPU est extrêmement lent, environ 50 calculs par seconde. Un processeur actuel est extrêmement rapide. Un GPU, processeur de carte graphique, fait plusieurs milliards de calculs par seconde (nVidia Hopper : 60.10^{12} calculs/s). Voir cette page sur le nombre d'opérations par seconde.

4. Bilan du TD

Vous devez comprendre :

- que les instructions et les données sont codées sous la forme d'octets dans la mémoire centrale.
- L'unité centrale utilise des « registres » pour les calculs. Ce sont comme les « mémoires » d'une calculatrice simple.
- Un programme machine est composé d'instructions assez rudimentaires : affectation d'une valeur à un registre, envoi et réception d'un registre sur un périphérique, calculs simples.
- Il y a un registre appelé `IP` qui indique quelle est l'instruction à exécuter. Il est incrémenté à chaque instruction.

Le prochain TD montrera :

- qu'on peut transférer des valeurs entre mémoire centrale et registres,
- faire des calculs complexes malgré la simplicité des instructions.

5. Partie optionnelle

5.1. Désassemblage de codes

☛ Effacez la mémoire centrale : menu Fichier, Effacer tout.

☛ Placez ces codes à l'adresse 0

```
02 B5 72 00 5F
```

☛ Mettez-vous devant l'assembleur et utilisez le menu *Désassembler la mémoire*.

Le source se remplit avec des lignes générées d'après ce qu'il y a dans la mémoire.

- `ORG n` est une *directive* qui dit où placer les instructions qui suivent, à partir de l'adresse n. Les programmes doivent toujours être mis en 0, mais il peut y avoir des données à ajouter quelque part ailleurs.
- Les mots `L00:`, `L02:` sont appelés des *labels*. Ce sont des symboles qui représentent les adresses des instructions. Dans ce TD, ils ne sont pas utiles.

Vous devez reconnaître les constantes, `B5` et `00`. La syntaxe `$...` signale que le nombre est en hexadécimal. S'il y a un `%` devant, alors il est en binaire.

☛ Dans la ligne `L00:  LD R0, $B5`, changez le `B5` en `%01010100`. Ensuite utilisez le menu `Source/Assembler en mémoire`.

Vous allez voir que les codes ont changé dans la RAM. Également, ils sont colorés en orange. CimPU a reconnu que c'étaient des instructions. Avant, il ne savait pas ce que c'était.

☛ Relancez le programme avec `Reset` puis `RUN`. Constatez que les leds allumées ont changé, selon le nouvel opérande. Refaites la manip avec d'autres valeurs.

☛ Changez carrément la valeur de l'octet d'adresse `01` dans la RAM. Il vaut actuellement `54` ($(01010100)_2$) et mettez ce que vous voulez. Les leds qui s'allumeront correspondent aux bits qui sont à 1 dans son écriture en base 2.

Le programme se termine par l'instruction `NOP` (*no operation*, càd « ne rien faire »). Son code est `00` et elle permet de savoir où se finit le programme. Attention, si vous mettez des données quelconques après, ces données seront désassemblées aussi, comme c'étaient des instructions, mais ça ne voudra rien dire du tout.

5.2. Pour en savoir plus

Les codes de CimPU sont décrits dans cette documentation, à partir de la page 11. C'est un peu difficile à comprendre parce que chaque instruction a de multiples variantes, selon le type des opérandes. Par exemple, `LD R0, constante` est codée par $(0000rsss)_2$) avec $r =$ `0` puisque c'est `R0`, et $sss =$ `010` parce que c'est une constante (appelé mode immédiat), donc ça donne $(00000010)_2$), `02` en hexa, suivi du nombre à mettre dans `R0`. Si c'était le registre `R1`, alors le code serait $(00001010)_2$), c'est à dire `0A` en hexa.

Dans un vrai microprocesseur, c'est exactement la même chose, sauf que les codes et les types d'opérandes sont très différents. Voir cette documentation de l'ARM7 ; vous verrez qu'il y a à peu près les mêmes instructions, par contre les opcodes sont absents car trop complexes.

Regardez plutôt le codage des instructions dans le 80x86, dans cette documentation ; cherchez par exemple l'instruction `ADD` (à cet endroit). Les registres 8 bits sont nommés `AL`, `BL`... et par exemple le codage de `ADD AL,nb` est `04` suivi du nombre *nb*. Dans cette documentation, *nb* est appelé *ib*, c'est à dire *immediate byte*.

Le codage des instructions est devenu beaucoup trop complexe pour un humain. On doit utiliser un assembleur, comme dans la suite du TD.