TP 3 : Définition d'un langage assembleur

1. Objectif

Dans ce TP vous allez apprendre à utiliser :

- La commande **PARASM** qui va vous permettre de compiler un fichier source **.ASM** écrit dans un certain langage assembleur et de générer un fichier objet **.OBJ**.
- La commande **OBJ2BIN** qui va vous permettre de convertir le fichier **.OBJ** généré par **PARASM** en un fichier **.BIN** exécutable dans **LOGISIM**.
- Le logiciel **MAG** qui va vous permettre de créer votre propre langage assembleur pour un microprocesseur donné.

2. Travail préparatoire

1. Sous moodle, dans la section «Séance 4» téléchargez le fichier **ARCHI.zip**.

2. Décompressez le fichier sur votre bureau windows. Vous devrez avoir sur votre bureau Windows, un dossier **ARCHI** qui contient lui-même 2 dossiers : **DOSBOX** et **MAG**

3. A l'aide du bloc-note ou de notepad++, éditez le fichier **start.ini** qui se trouve dans le dossier ARCHI et modifiez l'avant dernière ligne « mount m: C:\Users\monlogin\Desktop\ARCHI\MAG » de manière à indiquer le bon chemin menant vers le dossier MAG. Normalement, sous windows, vous avez juste à remplacer monlogin par votre login avec lequel vous vous êtes connecté.

4. Double cliquez sur le fichier **start.bat**. Une fenêtre terminal s'ouvre alors.

    a. Si vous obtenez l'affichage « Drive M does not exist ! You must mount it first.» c'est que le chemin indiqué dans le fichier start.ini n'est pas le bon. Dans ce cas, éditez de nouveau le fichier start.ini, indiquez le bon chemin et relancez start.bat.

    b. Si vous obtenez l'affichage « M:\> » à la dernière ligne, c'est que vous êtes prêt.
    Vous pouvez basculer du mode fenêtre au mode plein écran, et inversement, en appuyant sur **ALT-Entrée**.

    c. Si vous utilisez un écran 4K, la fenêtre risque d'être trop petite. Dans ce cas éditez le fichier start.ini et modifiez l'option windowresolution=original par windowresolution= 1920x1080.

3. Compilation d'un 1er programme assembleur

1. Ouvrir le bloc-notes ou notepad++ et saisissez le programme suivant puis sauvez le **dans le dossier MAG** sous le nom C2.ASM. Ce programme charge dans le registre A la valeur 5 puis la copie dans le registre B puis affiche à l'infini dans le registre B son complément à 2 (0005, FFFB, 0005, FFFB, …).

```
            LOAD_A #5
            LOAD_B_A
boucle      NOT_B
            INC_B
            JMP boucle
```

ATTENTION : toutes les lignes doivent commencer par au moins une tabulation sauf la ligne commençant par une étiquette (label boucle), sinon vous aurez des erreurs à la compilation. Mettre une tabulation après le label (boucle) et non pas le caractère « **:** ».

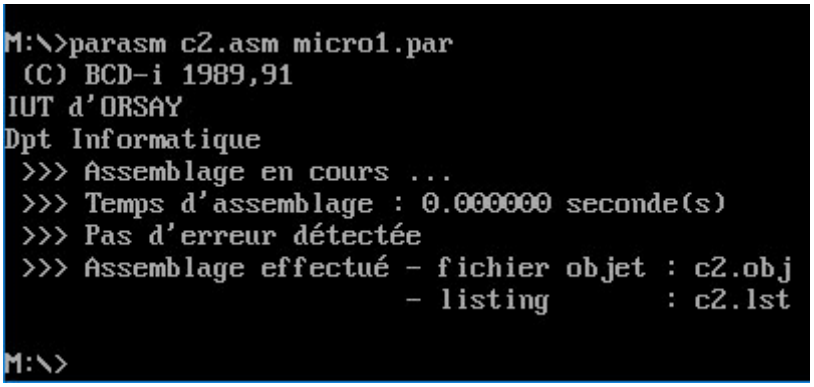
2. Après avoir sauvegardé le fichier C2.ASM, **appuyez sur CTRL-F4** dans la fenêtre terminal afin d'actualiser le contenu du dossier MAG. Cette manipulation devra être faite chaque fois qu'un **nouveau fichier est ajouté** au dossier MAG. Il n'est pas nécessaire de la faire s'il s'agit d'une modification d'un fichier existant.

3. Dans la fenêtre terminal, tapez la commande suivante :

```
parasm c2.asm micro1.par
```

Cette commande permet de compiler le fichier **c2.asm** et de générer le fichier **c2.obj** en utilisant le langage assembleur défini dans le fichier **micro1.par**. Le fichier **micro1.par** contient la définition des instructions assembleur pour le microprocesseur contenu dans le circuit **micro1.circ**.

Si vous avez correctement saisi, le programme vous devriez obtenir l'affichage suivant indiquant qu'aucune erreur n'a été trouvée et que le fichier **c2.obj** a bien été généré (dans le dossier MAG).



Notez qu'un second fichier est généré (à titre d'information). Il s'agit du fichier **c2.lst** qui contient pour chaque instruction de votre programme le code en binaire et en hexa correspondant. Vous pouvez l'ouvrir dans le bloc-notes ou notepad++ pour le visualiser.

4. Toujours dans la fenêtre terminal, tapez à présent la commande suivante :

```
obj2bin c2.obj
```

A partir du fichier **c2.obj**, cette commande va générer (dans le dossier MAG) un fichier **c2.bin** qui contiendra les instructions en langage machine et que nous pourrons ensuite charger directement sous LOGISIM dans la mémoire du microprocesseur **micro1.circ**. Si la génération s'est bien déroulée, vous devriez obtenir l'affichage suivant : « GENERATION DU FICHIER c2.bin REUSSIE !!! »

5. Lancer maintenant LOGISIM et chargez le circuit **micro1.circ.** Chargez dans la mémoire du circuit le fichier **c2.bin** : **clic droit** sur la mémoire puis « Clear Contents » puis de nouveau clic droit sur la mémoire puis « Load Image… » puis allez dans le dossier MAG et sélectionnez le fichier **c2.bin**.

6. Cliquez à plusieurs reprises sur l'horloge (ou CTRL-T) pour faire dérouler la simulation et vérifiez que le programme affiche bien à l'infini dans le registre B, 0005, FFFB, 0005, FFFB, … .

Remarques importantes concernant les fichiers .ASM

1. Le nom du fichier ne doit pas dépasser 8 caractères (sans compter l'extension .ASM).

2. Les commentaires doivent être précédés du point d'exclamation
3. Les commandes seront alignées verticalement et **doivent commencer par une tabulation** à gauche.
4. Dans le cas d'une instruction cible de saut , **le nom du label doit commencer complètement à gauche** sans tabulation.

4. Découverte du générateur de langage MAG

Dans la section précédente vous avez appris à :

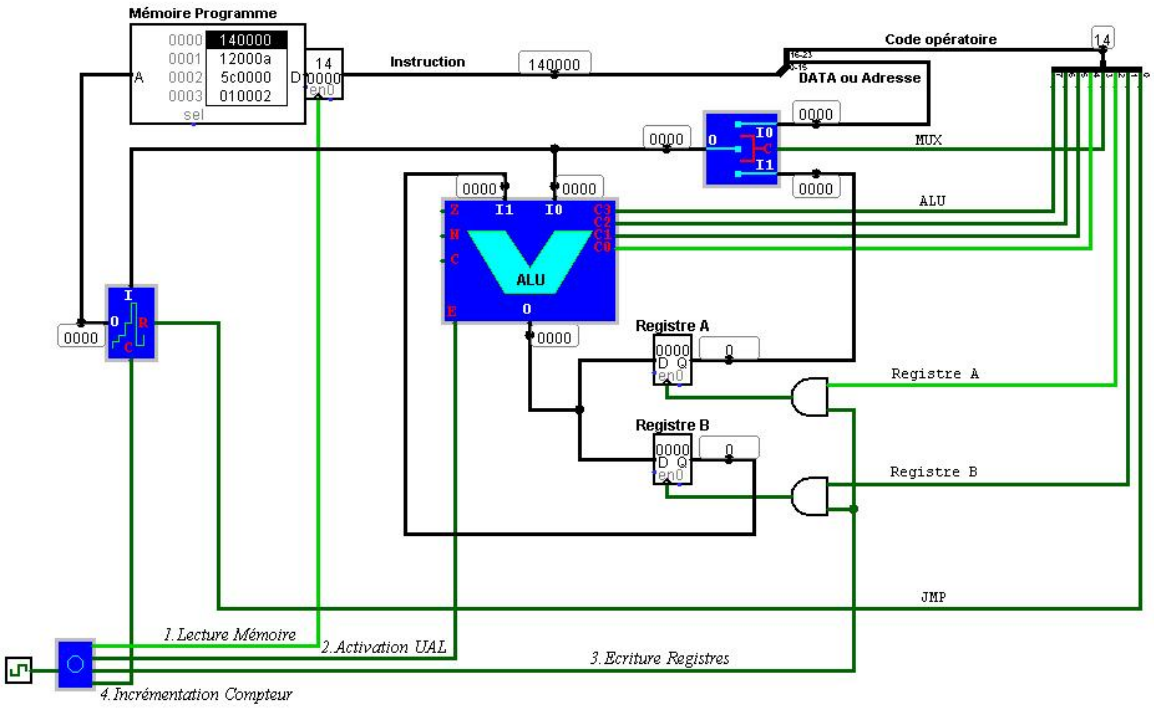
1. écrire un programme en assembleur,
2. à le compiler à l'aide de la commande **parasm**
3. à convertir le fichier **.obj** généré en fichier **.bin** exploitable sous LOGISIM à l'aide de la commande **obj2bin**
4. à exécuter le programme dans LOGISIM

Vous avez remarqué que la commande **parasm** nécessite 2 paramètres :

1. Le fichier source **.asm** contenant le programme écrit en assembleur.
2. Un fichier **.par** contenant la définition du langage assembleur (correspondance entre les mnémoniques assembleur et le code machine binaire à générer).

Dans cette section vous allez faire connaissance avec le logiciel **MAG** (**M**icro **A**ssembler **G**enerator) qui permet de définir un langage assembleur et de produire le fichier **.par** correspondant. Dans une première étape nous allons nous intéresser au langage créé pour le microprocesseur **micro1.circ** et défini dans le fichier **micro1.par** que nous avons utilisé dans la section précédente. Puis dans la section suivante, vous pourrez créer un nouveau langage pour un nouveau microprocesseur. En effet, une même instruction ne se code pas de la même façon en binaire avec deux microprocesseurs différents car le câblage n'est pas le même. Chaque microprocesseur possède donc son propre langage.

Dans un premier temps, nous allons nous intéresser au circuit micro1.circ, étudié lors du TP précédent.



Les instructions sont sur 24 bits. Le code d'une instruction est composé de 2 champs : le code opératoire (8 bits) et le champ DATA ou Adresse (16 bits). Le code opératoire est lui-même composé de plusieurs champs :

- ALU (4 bits)
- MUX (1 bit)
- REGISTRE_A (1 bit)
- REGISTRE_B (1 bit)
- JMP (1 bit)

Code opératoire					DATA ou Adresse
bits 23 à 20	**bit 19**	**bit 18**	**bit 17**	**bit 16**	**bits 15 à 0**
ALU	MUX	REG_A	REG_B	JMP	Donnée ou adresse de saut

Exemple : La première instruction du programme, que vous avez utilisée dans la section précédente, écrivait dans le registre A la valeur 5. Son mnémonique assembleur est **LOAD_A #5** et son codage binaire est le suivant:

Code opératoire					DATA ou Adresse
bits 23 à 20	**bit 19**	**bit 18**	**bit 17**	**bit 16**	**bits 15 à 0**
ALU	MUX	REG_A	REG_B	JMP	Donnée ou adresse de saut
0001	**0**	**1**	**0**	**0**	**0000 0000 0000 0101**

EXPLICATION D'UN EXEMPLE : LE FICHIER MICRO1.PAR

Lancez le programme MAG en tapant dans la fenêtre terminal : **MAG** (ou mag). La souris n'est pas utilisable dans MAG. **Utilisez les flèches du clavier** pour naviguer dans les items des menus et la touche Entrée pour choisir un item. Sélectionnez l'item « **Fichier** » puis « **Charge** ». Lorsque « *.MAD » apparaît appuyez sur Entrée et sélectionnez le fichier « MICRO1.MAD ».

REMARQUE

Le fichier **MICRO1.MAD** contient la définition du langage assembleur pour le circuit **micro1.circ**. Le logiciel MAG manipule en réalité 2 types de fichiers :

1. Les fichiers .MAD qui sont utilisés en interne par MAG pour sauver la définition du langage assembleur (peuvent apparaître parfois sous forme de raccourcis sous Windows).
2. Les fichiers .PAR utilisés par la commande PARASM et qui sont issus des fichiers .MAD.

Pour générer un fichier .PAR à partir d'un fichier .MAD il suffit de charger le fichier .MAD dans MAG puis de sélectionner « FICHIER » puis « COMPILE ».

CODE MACHINE

Le logiciel MAG supporte des instructions pouvant être codées sur un maximum de 128 bits !

Une fois le fichier MICRO1.MAD chargé, vous pouvez voir que les premiers 24 bits de l'encadré CODE ont pris la couleur blanche. Ceci indique que les instructions dans MICRO1.MAD sont codées sur 24 bits. Le bit blanc le plus à gauche correspond au bit de poids le plus fort du code de l'instruction (bit 23) alors que le bit blanc le plus à droite correspond au bit de poids le plus faible du code de l'instruction (bit 0).

LES CHAMPS

Allez dans le menu **Langage**, puis **Champs**, puis **Editer**. Ici sont créés les noms des différents bits (ou groupes de bits, c'est-à-dire les champs) de l'instruction. Vérifiez les informations suivantes :

	ALU	MUX	REGISTRE_A	REGISTRE_B	JMP	DATA
Taille	4	1	1	1	1	16
Adresse début	1	5	6	7	8	9
Valeur défaut	0000	0	0	0	0	0000 0000 0000 0000

Utiliser la touche ESC pour revenir en arrière.

LES INSTRUCTIONS

A l'aide de la touche Escape, remontez jusqu'au menu **Langage**, puis allez dans **Instructions**, puis **Editer**. Vous retrouvez l'ensemble des instructions qui ont été créées pour le microprocesseur **micro1.circ** .Vérifiez par exemple les informations pour les instructions ci-dessous. **Lorsque la valeur d'un champ n'est pas précisée, cela signifie que c'est la valeur par défaut du champ qui sera utilisée**.

Instructions	OpCode (champ)	OpCode (codage)	Opérandes (type)	Opérandes (champ)
LOAD_A	ALU	0001 R	VALEUR_IMMEDIATE	DATA
	MUX	0 R		
	REGISTRE_A	1 R		
ADD_B_A	ALU	0101 R		
	MUX	1 R		
	REGISTRE_B	1 R		
JMP	MUX	0 R	ADRESSE_DE_SAUT	DATA
	JMP	1 R		

Ainsi, pour l'instruction **LOAD_A**, le code binaire qui sera généré aura les bits de l'ALU à 0001, le bit MUX à 0 et le bit REGISTRE_A à 1. Les autres bits du code opératoire prendront la valeur par défaut indiquée lors de la définition des champs (0). Concernant les bits du champ DATA, le type VALEUR_IMMEDIATE pour l'opérande indique que les bits du champ DATA devront être déterminés en fonction d'une valeur qui suivra immédiatement l'instruction dans le code source : LOAD_A #valeur. La commande PARASM se chargera de générer les bonnes valeurs des bits du champ DATA en fonction de la valeur qui sera indiquée à la suite de l'instruction LOAD_A dans le code source.

L'instruction **ADD_B_A** n'a pas d'opérande. Ceci signifie que dans le code source il y aura juste ADD_B_A sans aucune valeur derrière.

Pour l'instruction **JMP**, il y a un opérande de type ADRESSE_DE_SAUT. Ceci signifie que l'instruction JMP sera suivie d'un label qui indiquera l'adresse de l'instruction vers laquelle il faut effectuer le saut. La commande PARASM se chargera de mettre la valeur de cette adresse dans le champ DATA.

La lettre R dans OpCode (codage) signifie que la valeur indiquée remplacera la valeur par défaut pour le champ correspondant.

5. Et si on créait un nouveau langage ?

Dans cette partie, votre travail va consister à créer un nouveau langage assembleur pour le microprocesseur **micro3.circ**. étudié lors du TP précèdent. Analysez bien ce microprocesseur car vous allez devoir lui créer vous-mêmes des instructions en suivant les étapes suivantes :

1. Lancer **LOGISIM** puis charger le circuit **micro3.circ.**
2. Tout en gardant **LOGISIM** ouvert, ouvrez 2 fenêtres terminal en lançant 2 fois le fichier **start.bat**. Dans la première fenêtre terminal, lancez **MAG**. La deuxième servira pour les commandes **PARASM** et **OBJ2BIN**.
3. Dans **MAG** choisissez *Fichier*, *Nouveau*, puis *Fichier*, *Sauve*, et tapez le nom **MICRO3.MAD** pour créer un nouveau fichier **MICRO3.MAD** vierge.
4. Allez ensuite dans *Langage*, *Champs*, *Ajouter* et créez les différents champs composant le code d'une instruction du microprocesseur **micro3.circ**. Pour y arriver, observez bien l'architecture du microprocesseur **micro3.circ.** **Appuyez sur F10** à la fin de la définition de chaque champ pour le sauver.
5. Quand vous avez fini de créer tous les champs (n'oubliez pas le champ DATA), allez dans *Langage*, *Instructions*, *Ajouter* et créez les instructions ci-dessous. Quand vous indiquez la valeur d'un champ, choisissez R (Remplace) pour la colonne Opération. **Appuyez sur F10** à la fin de la définition de chaque instruction pour la sauver.

Instruction	Signification
NOP	Pas d'opération
LOAD_A #valeur	Charge dans le registre A la valeur immédiate qui suit (ex : LOAD_A #5)
LOAD_B #valeur	Charge dans le registre B la valeur immédiate qui suit (ex : LOAD_B #5)
LOAD_A_B	Copie le registre B dans le registre A
LOAD_B_A	Copie le registre A dans le registre B
LOAD_B_C	Copie le registre C dans le registre B
LOAD_C_A	Copie le registre A dans le registre C
INC_B	Incrémente le registre B de 1 (B ← B + 1)
DEC_B	Décrémente le registre B de 1 (B ← B - 1)
AND_B_A	Fait le ET booléen entre les registres A et B et stocke le résultat dans le registre B
ADD_C_AB	Fait la somme des registres A et B et stocke le résultat dans le registre C
MUL_A_B	Fait le produit des registres A et B et stocke le résultat dans le registre A
DIV_B #valeur	Fait la division de B par valeur et stocke le quotient dans le registre B
JMP <label>	Effectue un saut inconditionnel à l'adresse indiqué par <label>
JMPZ <label>	Effectue un saut conditionnel à l'adresse indiqué par <label> si l'indicateur Z est à 1
JMPNZ <label>	Effectue un saut conditionnel à l'adresse indiqué par <label> si l'indicateur Z est à 0

6. Transformation du fichier de description **MICRO3.MAD** en fichier de paramètres **MICRO3.PAR :** une fois toutes les instructions créées, allez dans le menu *Fichier*, *Sauve* (pour sauvegarder MICRO3.MAD) puis *Fichier*, ***Compile*** (pour générer MICRO3.PAR).

6. Test du nouveau langage

Grâce au fichier **MICRO3.PAR** ainsi créé, vous allez maintenant pouvoir écrire des programmes dans votre langage assembleur, les compiler et les exécuter dans LOGISIM.

1. Ouvrez le bloc-notes ou notepad++ et écrivez le programme du TP précédent qui charge dans le registre A la valeur 0 et dans le registre B la valeur 1 puis calcule dans le registre C les valeurs de la suite de Fibonacci à l'infini. Sauvez votre programme dans le fichier **FIBO.ASM**, dans le dossier **MAG**.

2. Appuyez sur **CTRL-F4** dans la deuxième fenêtre terminal pour mettre à jour le contenu du disque avec les nouveaux fichiers créés dans la section précédente.

3. Tapez dans la deuxième fenêtre terminal la commande :**PARASM FIBO.ASM MICRO3.PAR**

4. S'il n'y a pas d'erreurs dans votre programme, un fichier FIBO.OBJ sera alors généré. Sinon éditez de nouveau le fichier FIBO.ASM pour corriger les erreurs et retapez ensuite la commande.

5. Tapez dans la deuxième fenêtre terminal la commande :**OBJ2BIN FIBO.OBJ**

6. S'il n'y a pas d'erreurs un fichier FIBO.BIN sera alors généré.

7. Sous LOGISIM, chargez le fichier FIBO.BIN dans la mémoire du circuit micro3.circ et lancez une simulation pour vérifier le bon fonctionnement de votre programme.

7. Conjecture de syracuse

La conjecture de syracuse est un énoncé mathématique qu'un enfant est capable de comprendre mais qu'aucun mathématicien au monde n'a encore réussi à démontrer. Cet énoncé dit la chose suivante :

- Prenez un nombre entier strictement positif quelconque
  - S'il est pair, divisez-le par 2
  - S'il est impair, multipliez-le par 3 et ajoutez 1
- Recommencez le même traitement avec le nouveau nombre obtenu

La conjecture de syracuse affirme que quel que soit le nombre de départ, la série aboutit toujours au cycle 4, 2, 1, 4, 2, 1, … Exemple : 14, 7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1, 4, 2, 1, … Ceci a été vérifié avec des nombres allant à plus de 2 puissance 60. Mais à ce jour il n'existe aucune démonstration formelle de cet énoncé.

En utilisant les instructions créées, écrire un programme **SYRACUSE.ASM** qui charge dans le registre A un nombre strictement positif quelconque et qui affiche dans le registre A à l'infini la suite de syracuse partant de ce nombre. Indice : le bit de rang 0 indique si le nombre est pair ou impair.

8. BONUS : Autres programmes

Reprendre les autres programmes du TP précédent et les tester à l'aide de votre nouveau langage. Attention à ne pas dépasser 8 lettres pour le nom du fichier source (.ASM).

1. Ecrire un programme qui charge dans le registre A la valeur 3 et dans le registre B la valeur 5 puis permute le contenu des 2 registres A et B. On utilisera le registre C comme variable temporaire pour effectuer la permutation.

2. Ecrire un programme qui charge dans le registre A la valeur 5 puis calcule sa factorielle. Vous donnerez 2 versions. La première version utilisera l'instruction JMPNZ. La deuxième version utilisera les instructions JMPZ et JMP.