**UE104 –** **Architecture des systèmes**

**HAUTE ÉCOLE DE NAMUR-LIÈGE-LUXEMBOURG**

**Bloc 1**

Atelier 4 : fonctionnement du CPU

Objectifs

* Utiliser CPU OS Simulator pour créer des instructions de base du CPU
* Utiliser le simulateur pour exécuter les instructions de base du CPU
* Utiliser le simulateur pour placer les données dans les registres, comparer les valeurs de registres, envoyer des données vers la pile, enlever des données de la pile, sauter d’adresses et additionner des valeurs mémorisées dans les registres
* Expliquer les fonctions des registres CPU spéciaux tels que les registres PC, SR et SP
* Créer des instructions conditionnelles (comparaison et saut)
* Créer des boucles
* Utiliser les modes d’adressage indirect pour accéder aux données

# Introduction

Cet atelier regroupe une série d’activités à réaliser avec CPU OS Simulator. Diverses instructions sont fournies afin d’identifier et de localiser les informations permettant de compléter ce document. N'oubliez pas de prendre des notes de vos observations.

Afin d’être capable de faire ces activités, vous devriez consulter les informations fournies dans les notes du cours de théorie associé. Le but est de vous permettre de saisir le fonctionnement d'un CPU, lisez donc attentivement et veillez à comprendre les instructions au lieu de faire des essais-erreurs qui ne vous amèneront pas nécessairement à la solution/réponse idéale.

# Premier programme

Tout d’abord vous devez placer des instructions dans la mémoire d'instruction (voir Figure 1), qui représente la RAM de la machine réelle, avant d’exécuter quelque instruction que ce soit.

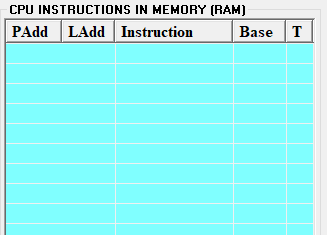
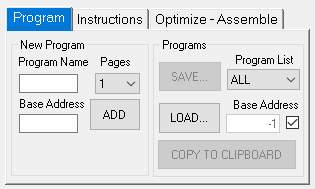


Figure 1 - Mémoire d'instructions

Pour ce faire, procédez comme suit :

1. Dans l'onglet **Program** (voirFigure 2**)**, entrez un nom de programme dans la zone située sous le libellé **Program Name (1)**, par exemple At5aEx1, puis entrez 100 comme adresse de base, dans la zone située sous le libellé **Base Address (2)** (cela peut être n’importe quel nombre, mais pour ce premier exercice, utilisez 100).



3

2

1

Figure 2 - Onglet Program

1. Ajouter le programme à la liste des programmes via le bouton **ADD (3)**. Un nouveau nom de programme apparaît dans cette liste (voir Figure 3).

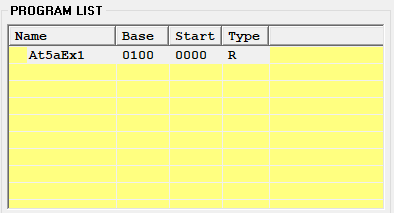
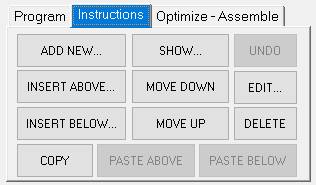


Figure 3 - Liste des programmes

Vous pouvez utiliser le bouton **SAVE…** pour enregistrer les instructions dans un fichier. L'extension associée au fichier contenant les instructions est sas. Vous pouvez également utiliser le bouton **LOAD** pour charger les instructions d’un fichier sas existant.

1. Vous êtes maintenant prêt à entrer des instructions dans le simulateur de CPU. Passez à l'onglet **Instructions** (voir Figure 4).



4

Figure 4 - Onglet Instructions

1. Ajouter une nouvelle instruction grâce au bouton **ADD NEW…** **(4)** Cela permet d’afficher la fenêtre listant toutes les instructions du jeu d'instructions (voir Figure 5).

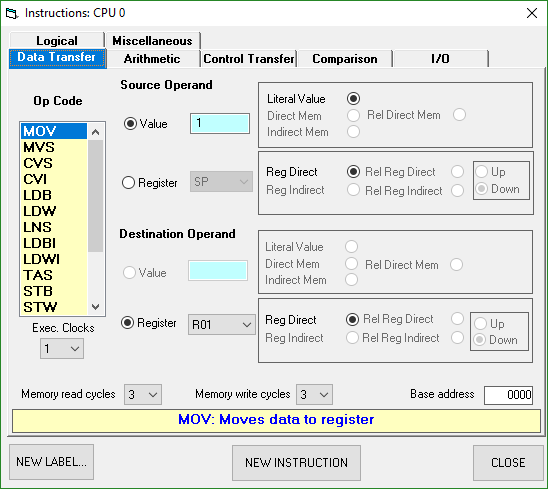


Figure 5 - Fenêtre Instructions : CPU 0

1. Utilisez cette fenêtre pour sélectionner et saisir les instructions CPU. La liste de la plupart des instructions est disponible dans les notes du cours de théorie. On y explique aussi comment les utiliser au moyen d'exemples.

**Remarque : enregistrer régulièrement votre code dans un fichier dans le cas où le simulateur se bloque. En effet, si c'est le cas, vous pourrez redémarrer le simulateur et recharger votre fichier.**

# Quelques instructions en assembleur...

Maintenant, à vous de chercher à comprendre en suivant les différents exercices proposés ci-après. Écrivez vos réponses dans les espaces laissés à cet effet !

1. Créez une instruction qui place le nombre 5 dans le registre R00.

MOV #5, R00

Le préfixe # est utilisé pour désigner une valeur littérale, qu'il distingue ainsi d'une valeur d’adresse qui ne s'écrit pas avec un #. R00 représente un registre arbitraire ; vous pouvez utiliser n'importe quel registre entre R00 à R31.

1. Exécuter l’instruction ci-dessus (pour faire cela simplement double-cliquez dessus dans la **mémoire d'instruction**). Observez le résultat dans les registres généraux (voir Figure 6).

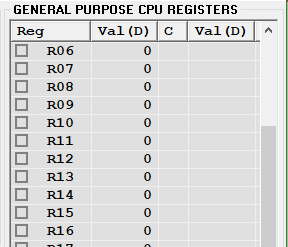


Figure 6 - Registres généraux

1. Créez une instruction qui place le nombre 8 dans le registre R01.

MOV #8, R01

1. Exécutez-la (pour cela double‐cliquez sur l'instruction).
2. Observez le contenu des registres R00 et R01 dans les registres généraux.
3. Créez une instruction qui additionne le contenu de R00 et R01.

ADD R00,R01

1. Exécutez-la.
2. Dans quel registre le résultat est-il mis ?

Le résultat est mis dans le registre 1, le 8 est écrasé et laisse place au résultat de l’addition

1. Créez une instruction qui pousse le résultat ci-dessus dans le haut de la **pile**, et puis exécuter-la.

PSH R01

1. Observer la valeur écrite dans le registre **SP** (voir Figure 8).
2. Créez une instruction qui pousse le nombre ‐2 sur le dessus de la **pile** et exécutez-la. Observer la valeur dans la pile du programme (voir Figure 7).

PSH #-2

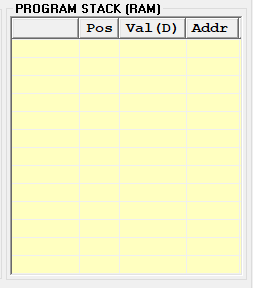


Figure 7 - Pile du programme

1. Observer la valeur écrite dans le registre **SP** (voir Figure 8). Chaque fois que vous empilez une valeur sur la pile du programme, le registre **SP** est mis à jour de manière à contenir l'adresse du sommet de la pile.

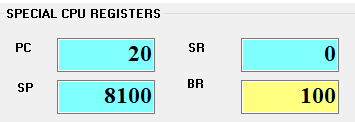


Figure 8 - Registres spéciaux

1. Créez une instruction pour comparer les valeurs des registres R00 et R01.

CMP R00, R01

1. Exécutez-la.
2. Quelle est la valeur du registre **SR** (voir Figure 8) ?

0

1. Observez les valeurs **OV/Z/N** du registre d’état (**SR Status flags**). Quelles cases sont cochées, et quelles sont celles qui ne le sont pas ? Qu'indiquent-elles ?

Aucunes cases n’est cochés et elles n’affichent rien.

1. Cochez et décochez alternativement les différentes *checkbox* du registre d'état (**SR Status flags**). Que pouvez-vous observer concernant la valeur affichée dans **SR** selon les cas ? À quoi cela vous fait-il penser (indice : pensez à l'atelier précédent) ?

Quand je coche la case OV, 4 s’affiche dans SR, quand je coche la case Z, 1 s’affiche dans SR et quand je coche la case N, 2 s’affiche dans SR

OV = overflow (grand nombre positif ou négatif)

Z = Zéro

N = négatif

1. Observez la valeur du registre **PC**. Quelle est sa valeur ?

24

1. Créez une instruction de saut inconditionnel vers la première instruction.

JMP 0000

1. Exécutez-la.
2. Observez la valeur du registre **PC**. Quelle est sa valeur ? Que pouvez-vous en déduire ?

La valeur du registre est 29, je peux en déduire que la valeur du registre pc prend comme valeur l’adresse logique de l’instruction.

1. Observez les valeurs dans les colonnes du **PAdd** et **LAdd**. Qu’indiquent ces valeurs ? Sont-elles différentes (indice : vérifiez la valeur de l’adresse de **Base**) ?

La valeur de Padd indique l’adresse physique et la valeur de LAdd indique l’adresse logique.

Padd = adresse physique vu par le matériel

LAdd = adresse logique vu par le programme

1. Quelle est la différence entre la valeur **LAdd** de la première instruction et la valeur **LAdd** de la seconde ? Qu’indique cette valeur (indice : pensez à la longueur de l’instruction en octets) ?

La valeur LAdd occupe 6 octet et commence à 0000 donc la prochaine est obligé d’être placé à l’adresse 0006

1. Positionnez-vous à nouveau sur la dernière instruction en cliquant dessus. Créez une instruction qui enlève la valeur située en haut de la **pile** du programme et la met dans le registre R02.

POP R02

1. Exécutez-la.
2. Observez la valeur du registre **SP**.

La valeur du registre SP est soustraite de la valeur contenu dans le top de la pile

1. Créez une instruction qui enlève la valeur située en haut de la **pile** du programme et la met dans le registre R03.

POP R03

1. Exécutez-la.
2. Observez la valeur du registre **SP**.

8098

1. Exécutez la dernière instruction à nouveau. Que s'est-il passé ? Expliquez.

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

1. Créez une instruction de comparaison, qui compare les valeurs des registres R04 et R05.

CMP R04,R05

1. Insérez manuellement deux valeurs égales dans les registres R04 et R05 via l’onglet *Registers* situé sous les registres généraux. (Voir Figure 9 - Registers)

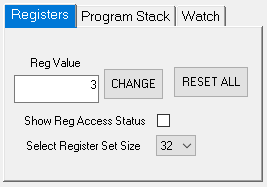


Figure - Registers

1. Exécutez l’instruction de comparaison de l’étape 31 ci-dessus.
2. Qu'en est-il des valeurs **OV/Z/N** du registre d’état (**SR Status flags**) ? Pourquoi ?

La case Z est coché car la valeur est null

1. Insérez manuellement, dans le registre R05, une valeur supérieure à celle du registre R04.
2. Exécutez à nouveau l’instruction de comparaison de l’étape 31 ci-dessus.
3. Qu'en est-il des valeurs **OV/Z/N** du registre d’état (**SR Status flags**) ? Pourquoi ?

Aucune case n’est coché car la valeur est positive et pas trop grande.

1. Insérez manuellement, dans le registre R04, une valeur supérieure à celle du registre R05.
2. Exécutez à nouveau l’instruction de comparaison de l’étape 31 ci-dessus.
3. Qu'en est-il des valeurs **OV/Z/N** du registre d’état (**SR Status flags**) ? Pourquoi ?

La case N est coché car la valeur est négative.

1. Créez une instruction qui saute à la première instruction si les valeurs dans les registres R04 et R05 sont égales.
2. Testez l'instruction ci-dessus en mettant manuellement des valeurs égales dans les registres R04 et R05. Que se passe-t-il ?
3. Exécutez d'abord l’instruction de comparaison suivie de l’instruction de saut (n’oubliez pas : vous exécutez une instruction en double‐cliquant dessus). Cela fonctionne-t-il ?
4. Enregistrez toutes ces instructions dans un fichier en cliquant sur le bouton **SAVE...** de l'onglet **Program**.
5. Sélectionnez le programme dans la liste des programmes et cliquez sur **REMOVE PROGRAM**. Ensuite cliquez sur **LOAD...** et choisissez le fichier At5aEx1.sas.

Vous savez maintenant utilisez le simulateur !

# Deux éléments complémentaires de CPU OS simulator

Plusieurs éléments de CPU OS Simulator ont déjà été présentés dans les notes du cours de théorie ainsi que dans l'Atelier 5a. En voici deux autres : la mémoire de données du programme et la console d'entrées/sorties.

En effet, en cliquant sur le bouton **SHOW PROGRAM DATA MEMORY...** (voir Figure 10), on accède à la fenêtre **DATA MEMORY** (voir Figure 11) qui permet de visualiser le contenu de la mémoire de données associée au programme en cours, de la modifier…

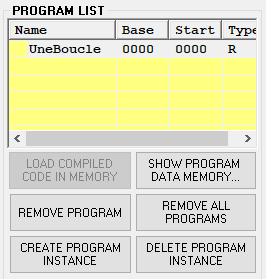


Figure 10 - Bouton SHOW PROGRAM DATA MEMORY...

Certaines instructions permettent d'accéder à la mémoire. Elles peuvent écrire ou lire ces données grâce au fait que celles-ci sont **adressées**. Ces données sont visibles à la Figure 11 ci-dessus.

La colonne **Ladd** affiche l’adresse logique de début de chaque ligne.

Dans cette représentation, chaque ligne représente 8 octets de données. Les colonnes B0 à B7 représentent les octets 0 à 7 de chaque ligne.

La colonne **Data** affiche les caractères affichables, correspondant aux 8 octets. Les octets correspondant à des caractères non‐affichables sont présentés sous la forme de points.

Les octets de données sont affichés au format hexadécimal seulement. Par exemple, à la Figure 11, il y a des octets de données non nuls aux adresses 19 et 37. Ces octets de données correspondent aux caractères affichables A et B.

Pour modifier la valeur d'un octet, commencez par sélectionner la ligne contenant cet octet. Puis, dans le cadre **Initialize Data**, utilisez le champ adéquat pour modifier la valeur des octets de la ligne sélectionnée, en fonction du type de la donnée (**Integer**, **Boolean** ou **String**). Cliquez sur le bouton **UPDATE** pour appliquer la modification.

Dans la partie **Debug control**, modifiez les valeurs selon la colonne et cliquez sur le bouton **UPDATE** pour appliquer la modification.

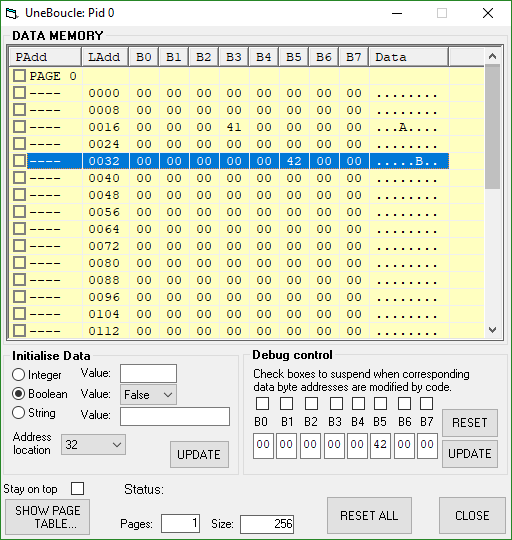


Figure 11 – Mémoire de données

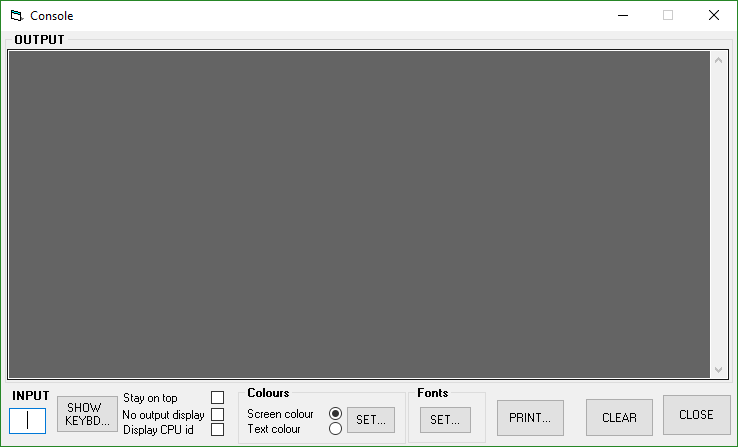


Figure 12 - Console d'entrées/sorties

La Figure 12 présente la Console d'entrées/sorties. Cette console est utilisée pour y afficher des messages et pour y lire les données. Elle peut être affichée en cliquant sur le bouton **INPUT OUTPUT…** visible sur la Figure 13.

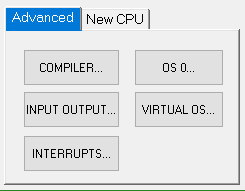


Figure 13 - Bouton INPUT OUTPUT...

Lorsque la console est ouverte, vous pouvez cliquer sur le bouton **SHOW KEYBD...** pour afficher le clavier qui permet d’entrer des données (voir Figure 14).

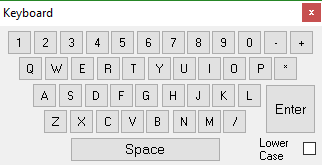


Figure 14 - Clavier

# Adressage direct et indirect, label et sauts conditionnels

Commencez par créer un nouveau programme comme vu au début de cet atelier avec comme un nom, par exemple At5aEx2, puis entrez 100 comme adresse de base.

1. Dans les slides du cours de théorie, localisez l’instruction qui est utilisée pour stocker un octet de données dans un emplacement de mémoire. Utilise-la pour stocker le nombre 6510 à l'emplacement mémoire dont l'adresse est 2010. Il s’agit d’un exemple d’**adressage direct**. Référez-vous à la Figure 11 pour savoir comment afficher le contenu de la mémoire de données. Notez ci-dessous l’instruction utilisée :
2. Créez une instruction pour placer le nombre décimal 22 dans le registre R01 et notez-la ci-dessous. Exécutez cette instruction et vérifiez le résultat dans R01.
3. Créez une instruction pour stocker le nombre décimal 51 dans l’emplacement mémoire dont l’adresse est actuellement stockée dans le registre R01. Il s’agit d’un exemple d’**adressage indirect**. Notez l’utilisation de « **@** » comme préfixe de R01 dans ce cas.
4. Prenez note de ce que vous voyez dans la mémoire de données, aux emplacements 20 et 22.
5. Maintenant, créez une boucle. Tout d’abord, entrez le code suivant.

MOV #0, R01

ADD #1, R01

CMP #5, R01

JNE 0

HLT

Le préfixe # est utilisé pour désigner une valeur littérale, qu'il distingue ainsi d'une valeur d’adresse qui ne s'écrit pas avec un #. R01 représente un registre arbitraire ; vous pouvez utiliser n'importe quel registre entre R00 à R31.

1. Le code ci-dessus n’est pas encore tout à fait terminé. L’instruction de branchement conditionnel **JNE** utilise une valeur numérique comme étant l’adresse à laquelle il faut sauter. Dans ce cas, c’est 0.

Mais, ce n'est pas toujours le cas, donc afin de rendre le code plus souple utilisez des **étiquettes/labels** pour représenter les adresses d’instruction. Le simulateur vous permet de le faire en suivant les instructions ci-dessous :

* Sélectionnez l'instruction **MOV**
* Cliquez sur le bouton **INSERT BELOW…**
* Dans la fenêtre permettant d'ajouter une instruction, cliquez sur **INSERT LABEL…**
* Écrivez le nom d'étiquette L0dans le champ
* Cliquez sur **Ok**

Le nouveau code devrait ressembler à ceci (modification en rouge) :

MOV #0, R01

L0:

ADD #1, R01

CMP #5, R01

JNE 0

HLT

* Ensuite, sélectionnez l'instruction **JNE**
* Cliquez sur **EDIT…**
* Dans la fenêtre permettant d'ajouter une instruction, dans la section **Source Operand**, sélectionnezl'étiquetteL0 dans la liste déroulante sous **Value**
* Cliquez sur **EDIT INSTRUCTION**

Le nouveau code devrait ressembler à ceci :

MOV #0, R01

L0:

ADD #1, R01

CMP #5, R01

JNE $L0

HLT

1. Comme vous pouvez le voir, l’étiquette L0 représente l’adresse de l’instruction immédiatement au-dessous, c'est-à-dire l’instruction **ADD**.   
   Maintenant l’instruction **JNE** peut utiliser L0 comme étant l’adresse à laquelle il faut sauter. Comme l’étiquette L0 peut représenter n’importe quelle adresse, ce code devrait fonctionner n’importe où dans la mémoire, ce qui le rend plus flexible. Le signe $ indique que L0est une étiquette.

Le code ci-dessus est maintenant prêt à fonctionner.

Pour exécuter ce programme, procédez comme suit :

* Cliquez sur le bouton **RESET** **PROGRAM** dans l'onglet **Program Control** de la fenêtre principale
* Sélectionnez l’instruction **MOV**, c'est-à-dire la première instruction du programme
* Régler la vitesse sur la valeur plus proche de 80
* Cliquez sur le bouton **RUN**
* Après un court instant, le programme devrait s'arrêter. S’il vous semble trop Iong, cliquez sur le bouton **STOP** et vérifier votre code. Rectifier si nécessaire et répéter les instructions ci-dessus.
* Quand le programme s'arrête, prenez note de la valeur de R01 dans le cadre ci-dessous

1. Afin de vous familiarisez avec les branchements conditionnels, vous allez faire une légère modification dans ce programme. Modifier le code du programme afin que la boucle du programme se répète tant que la valeur de R01 est inférieure ou égal à 3 (vous pouvez consulter les notes du cours de théorie pour cela) et testez-le. Quand vous estimez avoir la solution, prenez note de la valeur en R01 et copier le nouveau code dans le cadre ci-dessous.

Maintenant, rétablissez le programme comme il était avant ces modifications (vous pouvez utiliser le bouton **UNDO** pour cela).

# Sous-programme et pile du programme[[1]](#footnote-1)

Certains branchements sont utilisés pour fabriquer des sous-programmes. Les sous-programmes sont à l'assembleur ce que les fonctions sont au C.

Un programme contient souvent des suites d'instructions présentes en plusieurs exemplaires, qui servent à effectuer une tâche bien précise : calculer un résultat bien précis, communiquer avec un périphérique, écrire un fichier sur le disque dur... Sans utiliser de sous-programmes, ces suites d'instructions sont présentes en plusieurs exemplaires dans le programme. Cette répétition de code est souvent source d'erreur et empêche la bonne lisibilité du programme.

De plus, ces suites d'instructions sont présentes plusieurs fois dans le programme final, exécuté par l'ordinateur. Elles prennent donc de la place inutilement lors du chargement du programme en mémoire centrale !

Dans les langages de programmation modernes, il est possible de ne conserver qu'un seul exemplaire en mémoire et de l'utiliser au besoin. L'exemplaire en question est ce qu'on appelle un **sous-programme**, ou encore une **fonction**.

Pour exécuter un sous-programme, il suffit d'exécuter un branchement dont l'adresse de destination est celle du sous-programme : on dit qu'on appelle le sous-programme, d'où l'utilisation de l'instruction **CAL**.

Tout sous-programme se termine par une instruction de retour **RET**, qui permet au processeur de revenir là où il en était avant d'appeler le sous-programme. Certains processeurs disposent d'une instruction de retour dédiée, alors que d'autres l'émulent à partir d'autres instructions. L'instruction de retour a besoin de connaitre l'adresse de retour, l'adresse de la suite du programme. Celle-ci est sauvegardée soit par l'instruction d'appel de la fonction, soit par une instruction d'écriture spécialisée comme dans ce cas-ci **MSF**. Cette adresse de retour est sauvegardée au sommet de la pile du programme !

1. Vous allez maintenant créer un sous-programme simple à la suite des instructions précédentes.   
   Entrez le nouveau code suivant en respectant les consignes qui suivent.

L1:

OUT 24, 0

RET

Vous devez créer une nouvelle étiquette L1 au début du sous-programme. Cette étiquette représente l’adresse de début du sous-programme.   
Procédez comme au point (6). De plus, assurez-vous que vous sélectionnez l'option **Direct Mem** lorsque vous entrez la valeur de la première opérande 24 de l’instruction **OUT**.

1. Le code du sous-programme ci-dessus affiche simplement le texte commençant à l’emplacement de mémoire de données d'adresse 24 et quitte le sous-programme (**RET**). Pour que ce sous-programme fonctionne, il faut mettre du texte à partir de l'adresse 24. Vous pouvez le faire manuellement en suivant les étapes ci-dessous :

* Cliquez sur **SHOW PROGRAM DATA MEMORY…** (voir Figure 10)
* Dans la fenêtre qui s’ouvre, sélectionnez la ligne 0024 sous colonne **LAdd**
* Sous **Initialise Data**, sélectionnez l'option **String**
* Dans la zone de texte **Valeur**, ajouter du texte, par exemple "Mon nom est Besim"
* Cliquez sur le bouton **UPDATE**

1. Un sous-programme n'est d’aucune utilité en soi. Pour qu’il soit utile, votre programme doit y faire appel en utilisant les instructions **MSF** suivie de **CAL**.   
   Le **MSF** (**Mark Stack Frame**) permet de réserver une place pour l’adresse de retour sur la pile du programme. L’instruction **CAL** doit spécifier l’adresse de début du sous-programme appelé.   
   Modifiez votre code afin que lorsque le sous-programme est appelé, il affiche le texte à plusieurs reprises via une boucle. Par exemple, en utilisant le code ajouté aux points (6) et (9). Le programme modifié doit ressembler à ceci :

MOV #0, R01

L0:

ADD #1, R01

MSF

CAL $L1

CMP #5, R01

JNE $L0

HLT

L1:

OUT 24, 0

RET

1. Le code ci-dessus est maintenant prêt à fonctionner. Pour voir le texte qui s’affiche, vous devez afficher la console d'entrées/sorties, comme expliqué en début d'atelier. Pour exécuter ce programme, procédez comme suit :

* Cliquez sur le bouton **RESET PROGRAM**
* Sélectionnez l'instruction **MOV**, la première instruction du programme
* Ajuster la vitesse d'exécution pour être au plus proche de 80
* Cliquez sur le bouton **RUN**

1. Actuellement, l’instruction **OUT** utilise l’**adressage directe**, c'est-à-dire que l’adresse mémoire 24 fait partie de l’instruction.   
   Le but de cette étape est que vous utilisiez l’**adressage indirect** comme au point (48). Pour cela, vous devez placer l’adresse de mémoire 24 dans un registre (n'importe quel registre général). Ensuite vous devez faire en sorte que l’instruction **OUT** utilise la valeur de ce registre comme étant l'adresse du texte à afficher.   
   Exécutez le code pour tester votre modification. Prenez note de la partie modifiée du code programme dans le cadre ci-dessous.

Utilisez le bouton **UNDO** pour retrouver le sous-programme tel qu'il était avant ces modifications.

1. Un sous-programme peut appeler un sous-programme…

Convertissez la boucle de façon qu'elle soit dans un autre sous-programme et puis appelez-le.  
Maintenant vous avez deux sous-programmes, l'un appelant l’autre. Le code suivant représente ce changement. Notez que l’instruction **HLT** est remplacée par l’instruction **RET** et les instructions **MSF**, **CAL** et **HLT** sont ajoutées ainsi que l'étiquette L2.   
**CAL $L2** appelle le sous-programme avec la boucle et **CAL $L1** appelle le sous-programme qui affiche le texte.

MSF

CAL $L2

HLT

L2:

MOV #0, R01

L0:

ADD #1, R01

MSF

CAL $L1

CMP #5, R01

JNE $L0

RET

L1:

OUT 24, 0

RET

Ensuite, réinitialisez le programme, puis mettez en surbrillance la première instruction **MSF**.   
Exécutez le programme et vérifiez le résultat dans la fenêtre de la console comme avant.

1. Rendez ce programme plus flexible !  
   Le code ci-dessus fait la boucle 5 fois, et ce nombre est fixe. Pour plus de flexibilité, passez le nombre d'itérations en tant que paramètre au sous-programme (commençant à étiquette L2).

Pour cela, vous allez utiliser les instructions **PSH** et **POP** (voir **pile**). Modifiez votre code pour ressembler à celui ci-dessous et exécutez-le en observant l’affiche sur la console :

MSF

**PSH #8**

CAL $L2

HLT

L2:

**POP R02**

MOV #0, R01

L0:

ADD #1, R01

MSF

CAL $L1

**CMP R02, R01**

JNE $L0

RET

L1:

OUT 24, 0

RET

1. Examinez le code ci-dessus et expliquez brièvement comment fonctionne le passage de paramètres :
2. Enfin, comme défi final, modifiez le code ci-dessus afin qu'un second paramètre soit passé à la sous-programme (celle commençant à étiquette L2) de la même manière que le premier paramètre a été passé.   
   Le second paramètre est utilisé pour initialiser le registre R01 à la valeur de ce second paramètre. Copiez le code modifié après cette dernière modification dans la case ci-dessous :

# Instructions de comparaison et de branchement

1. Écrivez les instructions qui **si** R02 **est supérieur à** (>) R01 **alors** R03 est garni avec 8. Utilisez le R01 comme première opérande et R02 comme deuxième opérande.
2. Écrivez les instructions qui **si** R02 **est inférieur ou égal à** (≤) R01 **alors** R03 est garni avec -5. Utilisez le R01 comme première opérande et R02 comme deuxième opérande.
3. Écrivez les instructions qui **si** R01 **est égale à** (=) 0 alors R03 est mis à 5 **si non** R03 est mis à R01 plus 1.

# Quelques boucles : instructions de comparaison et de saut - RAPPEL

1. Écrivez une boucle qui se répète **5 fois** en incrémentant R02 de 2 à chaque itération.
2. Écrivez une boucle qui se répète **tant que** R04 **est** **supérieur à** (>) 0. Initialisez R04 à 8.
3. Écrivez une boucle qui se répète **jusqu'à ce que** R05 **soit supérieur à** (>) R09. Initialisez R05 à 0 et R09 à 12.

1. Inspiré de [https://fr.wikibooks.org/wiki/Fonctionnement\_d'un\_ordinateur/Langage\_machine\_et\_assembleur#Instructions\_d'appel\_et\_de\_retour\_de\_fonction](https://fr.wikibooks.org/wiki/Fonctionnement_d'un_ordinateur/Langage_machine_et_assembleur%23Instructions_d'appel_et_de_retour_de_fonction%20) [↑](#footnote-ref-1)