I/ INTRODUCTION

II/ ASSEMBLEUR

III/ MACHINE VIRTUELLE

IV/ ANNEXES

MEMBRE: SOMASUNDRAM BARATHAN

TAILLEUR ALEXANDRE

SAM TONY

RAMALHO VALENTIN

I / INTRODUCTION

Avant d'entrer dans le vif du sujet, il me semble nécessaire de faire une petite introduction sur ce qu'est le Corewar, de quoi il est composé ainsi que le rôle de chacune de ces parties.

Sans perdre plus de temps passons tout de suite aux explications.

1- QU'EST-CE QUE LE COREWAR

Avant tout, il est normal de préciser cette étrange chose qu'est le Corewar.

Le Corewar est avant tout un jeu ayant pour but de faire combattre plusieurs programmes dans une sorte d'arène.

Le jeu se termine lorsqu'il ne reste qu'un seul programme encore « vivant » après un certain laps de temps.

Indirectement, le Corewar est aussi à l'origine des virus informatique justement sur le fait de créer des programmes qui doivent se battre pour survivre.

Les termes « programmes » et « arène », dit précédemment, vont sûrement vous sembler assez abstrait mais ne vous inquiétez pas, je vais y revenir.

Le Corewar est composé 3 éléments que l'on va analyser plus en détails : La Machine Virtuelle ; L'assembleur ; Les Champions.

2- LA MACHINE VIRTUELLE

La Machine Virtuelle (ou VM pour faire plus court) est en fait l'arène dans laquelle les programmes vont se battre.

Il s'agit d'un espace exclusivement réservé à l'exécution des commandes de chacun des programmes qui combat.

Elle est capable de gérer jusqu'à 4 programmes et a pour tâche de veiller au bon fonctionnement des programmes ainsi que de superviser l'avancé du combat en contrôlant qu'elles sont les programmes est encore en vie.

3- L'ASSEMBLEUR

L'assembleur (ou ASM pour faire plus court) est l'élément où l'on enverra les programmes évoqués précédemment pour en créer une version compréhensible par la VM (c'est-à-dire en binaire pour les plus curieux).

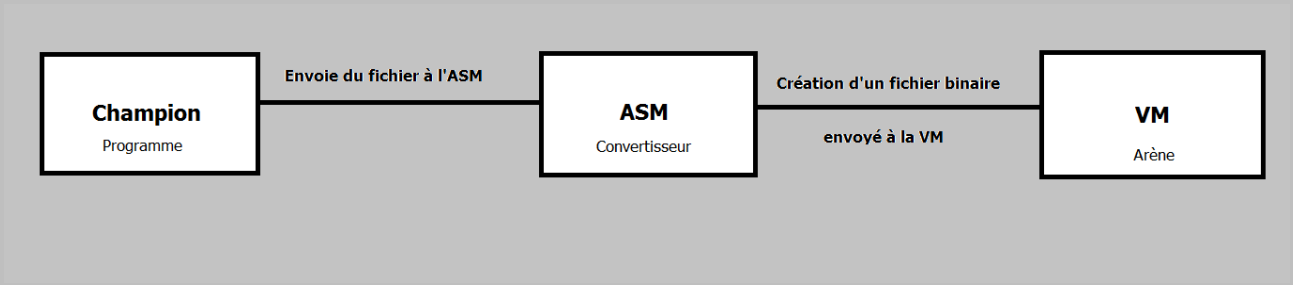
Il aura aussi pour but indirectement de vérifier si le programme est correct (voir p.5 pour plus de detail).

4- LES CHAMPIONS

Les Champions sont en fait les programmes énoncés précédemment dans l'ASM que l'on va convertir pour la VM (là les choses se clarifie n'est-ce pas ?).

Ces champions seront codé en langage assembleur propre à la VM utilisé dans ce Corewar (se seront donc des fichier « .s », ex : toto.s) .

De manière un peu plus imagé, cela donne ça :



Maintenant que les choses sont claires, on peut désormais entrer dans le vif du sujet, c'est-à-dire le fonctionnement de l'ASM ainsi que de la VM.

II/ L'ASSEMBLEUR

Bien que cette partie soit réservée pour l'ASM il est important d'apporter des précisions sur les Champions afin de mieux comprendre le fonctionnement de l'ASM.

I- LES CHAMPIONS

Que sait-on des Champions grâce à cette petite introduction ?

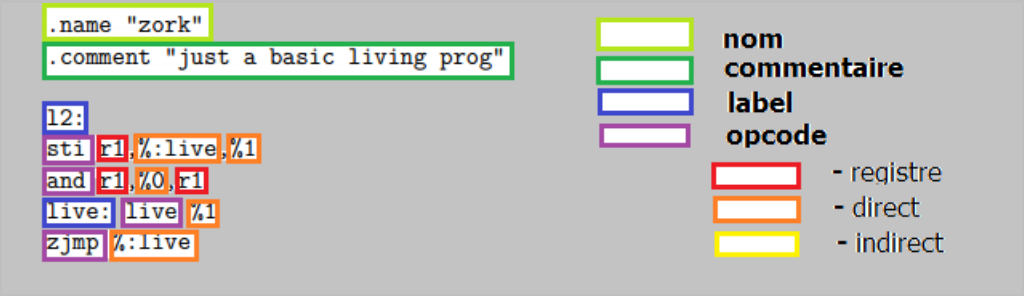
Qu'il s'agit des programmes qui vont combattre dans la VM et qu'ils sont écris en langage assembleur (.

Le langage assembleur est un langage bas niveau, c'est-à-dire qu'il se rapproche le plus du langage de la machine, le langage binaire. Le code sera donc découpé par ligne appelé « instruction » qui sont composé de 3 éléments : le label (optionnelle, on peut le définir comme le nom de l'instruction et est toujours suivi par « : ») ; le(s) code(s) d'instruction(s) (ou opcode, qui sont les types d'actions possibles, vous pouvez voir la liste des opcode en annexe p.7) ; le(s) paramètre(s) de l'instruction.

Il peut y avoir 3 type de paramètre d'instruction : des registres (noté « r# » avec # à replacer par un nombre entre 1 et 16), des Directs (valeur écrit avec un « % » devant, ex : %42) et des Indirects (écrit juste par un nombre, ex : 42).

Pour le champion, il est aussi possible d'ajouter un nom (.name) et un commentaire (.comment) afin de le distinguer des autres (cet ajout est optionnel).

Dans la forme voilà ce que donne un Champions :



Bien, maintenant qu'on en a fini avec les Champions, on passe à l'ASM.

2- COMPILATION DES CHAMPIONS

La compilation du Champion dans l'ASM se fait en 2 parties. La première étant la lecture, la vérification ainsi que le stockage du contenu du Champion (ex : toto.s) et la deuxième la création d'un fichier binaire (cette fois ci donc un fichier « .cor », ex : pour toto.s nous aurons un toto.cor) et l'écriture du contenu du fichier « .s » dans le fichier « .cor » sous une forme compréhensible par la VM.

Lors de lecture du fichier, l'ASM procède à une vérification ligne par ligne afin de détecter une erreur d'écriture, l'utilisation d'un opcode inexistant, etc. Si cette vérification est menée à bien l'ASM convertie la ligne lue, la stocke et passe à la ligne suivante.

Une fois la lecture terminée, l'ASM crée un nouveau fichier cette fois binaire (« .cor ») et écrit, tout ce qu'elle a lu dans le Champions et convertis, dans le fichier binaire.

Voilà un petit exemple avec un Champion appelé « zork.s » :

zork.s

```
l2:
sti r1,\%:live,\%1
and r1,\%0,r1
live: live \%1
zjmp \%:live
```

zork.cor

```
0x0b,0x68,0x01,0x00,0x0f,0x00,0x01
0x06,0x64,0x01,0x00,0x00,0x00,0x00,0x01
0x01,0x00,0x00,0x00,0x01
0x09,0xff,0xfb
```

Nous avons maintenant notre fichier binaire de notre Champion, maintenant il est de l'envoyer dans l'arène, place à la VM.

III/ MACHINE VIRTUELLE

Il est clair que nous allons du fonctionnement de la VM pour le Corewar mais nous allons aussi énumérer les différentes options qu'offre cette VM.

1- LES CHAMPIONS DANS LA MACHINE VIRTUELLE

Nous avons évoqués que le but ce jeu est qu'il ne reste qu'un seule programme « vivant », c'est à dire que le Champion confirme à la VM qu'il est toujours là. Pour ce faire, un des opcode pour le champion (voir Annexe p.7) est l'instruction « live » pour confirmer qu'il est toujours actif.

Tout le jeu repose sur un principe de prise de terrain. Cela semble abstrait donc je m'explique.

La VM réserve un espace dans la mémoire pour le combat. Une fois chose faite, les champions sont envoyé cette arène qu'est la VM et auront pour objectif de prendre de la mémoire mais aussi subtilisé la mémoire prise par les autre programmes. Pendant un certain laps de temps, les Champions devront confirmer leur présence par le biais de cette instruction « live » évoqué précédemment. Une fois ce laps de temps passé, la VM vérifie si les Champions ont envoyés leur instruction « live ». Si c'est le cas, le Champion continue à combattre, sinon, il est considéré comme éliminé et la partie se poursuit sans lui. Une fois qu'il ne reste plus qu'un Champion, la partie est terminée et le dernier Champion en vie gagne la partie. Oui, c'est un vrai « Battle royal ».

2- LES OPTIONS POUR LA MACHINE VIRTUELLE

La VM possèdent 3 options qu'il est possible d'ajouter la lors de son lancement :

- « -dump nbr_cycle » : Efface la mémoire après un certain laps de temps si la partie n'est pas encore finie.

- « -n prog_number » : Permet de fixer le nombre du prochain programme. Par défaut, cette option prendra le premier programme libre dans l'ordre dans l'ordre des programmes passés en paramètre de la VM.

- « -a load_adresse » : Permet de fixer l'adresse du prochain programme. Par défaut, si aucune n'est entré, on choisira les adresses de telle sorte que les programmes soit les plus éloignés les uns des autres.

IV/ ANNEXE

live	(1 arg) Instruction indiquant que le Champion est en vie.
ld	(2 arg) L'instruction charge (load) la valeur du premier paramètre dans un registre.
st	(2 arg) L'instruction range (store) la valeur du premier argument dans le second.
add	(3 arg) L'instruction additionne le contenu des 2 premiers arguments et le stock dans le troisième.
sub	(3 arg) Même chose que pour **add** mais en effectuant une soustraction
and	(3 arg) L'instruction effectue une porte logique « & » avec les 2 premiers paramètres et stock dans le troisième.
or	(3 arg) Même chose que pour **and** mais avec la porte logique « | ».
xor	(3 arg) Même chose que pour **and** mais avec la porte logique « ^ ».
zjump	(1 arg) L'instruction permet de faire un saut à l'index qui lui a était passée en paramètre.
ldi	(3 arg) L'instruction ajoute à l'adresse du premier paramètre, la valeur du deuxième paramètre et stock le résultat dans le troisième paramètre.
sti	(3 arg) L'instruction copie le premier paramètre à l'addition des adresses des 2 derniers paramètres (les 2 derniers paramètres sont des index).
fork	(1 arg) L'instruction prend un index en paramètre et créer un nouveau programme à partir de l'adresse du paramètre.
lld	(2 arg) Même chose que **ld** sans IDX_MOD.
lldi	(3 arg) Même chose que **ldi** sans IDX_MOD.
lfork	(1 arg) Même chose que **lfork** sans IDX_MOD.
aff	(1 arg) L'instruction prend comme paramètre un registre et affiche le caractère dont le code ascii est présent dans ce registre.

Répartitions des tâches :

- ASM : TAILLEUR Alexandre, SAM Tony, RAMALHO Valentin
- VM : SOMASUNDRAM Barathan, TAILLEUR Alexandre
- Documentation : RAMALHO Valentin