COREWAR
Un ASM
Une VM

Une explication simple

La VM est une machine virtuel, c'est à dire un ordinateur avec ses propres règles qui s'exécute dans un ordinateur. L'ASM est un assembleur c'est a dire un langage qui se veut l'un des plus bas possible pour une machine donnée, en général si l'on veut écrire des instruction plus bas qu'en assembleur il faudrait écrire en binaire...

Le principe du projet est simple on fait une VM et un ASM, la VM va pouvoir exécuté le code généré par l'ASM.

L'ASM

Le but est de lire un fichier assembleur et d'interpréter les instructions pour les retranscrites en binaire.
Par exemple ld est une instruction classique, son numéro est 2. Elle prends divers paramètres, j'en parlerais plus tard. Donc pour écrire une instruction comme ld, il suffit d'écrire dans le binaire un octet pour son numéro (opcode) son octet de codage des paramètres (code qui indique de quel types sont les argument que ld prends en paramètre cet octet est donc variable pour une même instruction) et ses paramètres (ce sont juste des nombres je détaillerais plus loin),

Par pur exemple : ld %5, r1

	ld	Octet de codage des paramètres	Premier octet de%5	Deuxième octet de%5	Troixième octet de%5	Quatrième octet de%5	r1
binaire	00000010	10010000	00000000	00000000	00000000	00001001	00000001
hexa	02	90	00	00	00	05	1

Le big endian

Vous aurez peut être remarquer, j'ai tout écris en big endian. Pour expliquer simplement la différences entre le little et le big. C'est simple chaque variable a son nombre d'octet propre et l'ordre dans lequels on considèrent ces octets est inversés. C'est à dire en big on écrit les octet en partant de la gauche alors qu'en little on écrit les octets en partant de la droite. Pareil pour lire en big on lit à partir de la gauche et en little de la droite, Pourquoi je parle de ça parce que le corewar est une VM qui fonctionne en big endian donc sachez le si vous êtes en little (99.99 % de chances) vous devez inverser l'ordre des octets d'une variable.

Little ==> ABCD
Big ==> DCBA

Le nom et le commentaire du binaire

Ils existent deux règles spéciales '.name' et '.comment', tout simple le nom du programme et le commentaire n'hésiter pas a décrire précisément le but du programme dans le commentaire il y a de la place...

Le carry

Mais pourquoi carry ? C'est un booléen qui vaut 1 si la dernière opération a renvoyé 0. Logique ? oui car if (carry) then jump.

Les cycles

Chaque instruction met un certain nombre de cycles pour s'exécuter dans la VM.

Le compteur de programme

Comme son nom ne l'indique pas le PC contient l'adresse de la prochaine instruction à exécuter. Cela permet à la VM de savoir quel instruction elle doit exécuter, c'est elle qui modifie le PC mais on peut lui demander de le bouger la où on le souhaite (zjmp).

L'octet de codage

Après chaque instruction*, il y a un octet qui décrit les paramètre de l'instruction. Pour l'instant chaque instruction peut avoir au max 3 instruction. Donc cet octet sera remplit de la façon suivante. Pour chaque argument il y a un code sur 2 bit :

-registre : 01
-indirect : 11
-direct et label : 10

exemple : xor 5, %6, r1 va donner 11 10 01 00

Bizarrement il y a des règles spéciaux sur tout sauf sur l'octet de codage, juste pour que la VM ait ses propres difficulté de parsing ? Non je pense pas que ça soit ça...;)

*sauf chez certaine instructions, live, zjmp, fork, lfork. Pourquoi ? J'ai bien une réponse mais elle pose une autre question « et pouqruoi pas add sub et aff aussi eux n'ont pas besoin d'octet de codage »

Les types d'arguments

Il y a 3 types d'arguments le registre (T_REG), l'index (T_IND || T_LAB) et le direct (T_DIR).

Le registre est contenu dans la VM, chaque instance d'un programme a ses propres registre il y en a 16 dans le précédent exemple (ld %5, r1) je charge le nombre 5 dans le registre numéro 1.

l'index est une adresse relatif au PC,
il y a l'indirect c'est en gros 5 = PC + 5 c'est simple à comprendre ld 5, r1 va mettre la valeur de la mémoire a PC + 5 dans r1.
Il y a le label, c'est un index automatique c'est à dire que c'est asm qui va trouver la bonne valeur a mettre par exemple.

```
        sti r1,%:label,%1
label:  live %42
```

%:label va être remplacé par la valeur qui correspond a l'adresse de l'instruction live

Le label peut être utilisé si l'argument est du type T_DIR ou T_IND logique ? Non... car l'octet de codage code les labels comme des directs alors ça devrait être que T_DIR, vous la sentez venir la prise de tête ?

le direct est simple (lol) %5 = 5... c'est la valeur qui est écrit en général, mais pas toujours par exemple ld %5, r1 met 5 dans r1 alors que sti r1,%5, r2 va mettre r1 dans PC + 5 + r2.

Description des instructions

Op code	Nom	Description	Paramètres	Nombres de cycles
1	live	Dit live pour le champion mis en premier paramètres... conseil votre numéro de champion est aléatoire donc pensez à changer le paramètre de live avec un st...	T_DIR	10
2	ld	Charge le premier paramètre dans le deuxième obligatoirement un registre (lit 4 octet)	T_DIR | T_IND, T_REG	5
3	st	Stock le premier paramètres obligatoirement un registre dans le deuxième (toujours 4 octet)	T_REG, T_IND | T_REG	5
4	add	Additionne les deux premiers argument et met le résultat dans le troisième	T_REG, T_REG, T_REG	10
5	sub	Soustrait les deux premiers argument et met le résultat dans le troisième	T_REG, T_REG, T_REG	10
6	and	Fais un et (&) binaire entre les deux premiers arguments et stocke le résultat dans le troisième	T_REG | T_DIR | T_IND, T_REG | T_IND | T_DIR, T_REG	6
7	or	Fais un ou (|) binaire entre les deux premiers arguments et stocke le résultat dans le troisième	T_REG | T_DIR | T_IND, T_REG | T_IND | T_DIR, T_REG	6
8	xor	Fais un ou exclusif (^) binaire entre les deux premiers arguments et stocke le résultat dans le troisième	T_REG | T_DIR | T_IND, T_REG | T_IND | T_DIR, T_REG	6
9	zjmp	Additionne le premier argument au pc si le carry vaut 1 sinon elle ne fais rien, Attention le paramètre 1 de cette instruction est un index	T_DIR	20
10	ldi	Charge le premier paramètre puis l'additionne au second et utilise cette somme comme index pour charger la valeur a cette index pour le mettre dans le troisième obligatoirement un registre, Attention les paramètres 1 et 2 de cette instruction sont des index	T_REG | T_DIR | T_IND, T_DIR | T_REG, T_REG	25
11	sti	Stock le premier paramètres obligatoirement un registre a l'adresse obtenue en faisant PC + valeur du deuxième argument + valeur du troisième argument, Attention les paramètres 2	T_REG, T_REG | T_DIR | T_IND, T_DIR | T_REG	25

		et 3 de cette instruction sont des index		
12	fork	Créer une nouvelle instance du programme qui qui aura comme PC l'actuel plus la valeur du premier paramètre, Attention le paramètre 1 de cette instruction est un index	T_DIR	800
13	lld	Charge le premier paramètre dans le deuxième obligatoirement un registre (lit 4 octet)	T_DIR | T_IND, T_REG	10
14	lldi	Charge l'addition du premier paramètre et du second dans le troisième obligatoirement un registre, Attention les paramètres 1 et 2 de cette instruction sont des index	T_REG | T_DIR | T_IND, T_DIR | T_REG, T_REG	50
15	lfork	Créer une nouvelle instance du programme qui qui aura comme PC l'actuel plus la valeur du premier paramètre, Attention le paramètre 1 de cette instruction est un index	T_DIR	1000
16	aff	Affiche le caractère contenus dans le premier paramètre donc modulo 256 pour ASCII étendu	T_REG	2

Tout les index auront une porter maximum grâce a un modulo sauf lld, lldi et lfork. Donc st r1, 5 ==> st r1, 42 % 512,
==> sti r1, 4, 2 ==> (4 + 2) % 512
==> ldi 42,%42, r1 ==> ((42 % 512) + 42) % 512

Les instruction suivantes modifie le carry : ld, add, sub, and, or, xor, ldi, peut être forl, lld, lldi et lfork.

La VM

Son but est d’exécuter des champions, il y a des règles pour gagner le championnat, un combat a mort entre champions.