VM :

- **RAM (arène) :**
  MEM_SIZE -> 4*1024 = 4096
  La VM devra allouer 4096 octets d'espace pour le combat
- **Processus :**
  REG_NUMBER -> 16 registres de 4 octets chacun
- **PC : lier au processus**
  Registre de référence de chaque processus (tete de lecture)
- **Carry : lier au processus**
  Flag (genre un int)

IDX_MOD = 512

Lexique :

- **Registre :** 1 octet
  Emplacement mémoire du processeur (ou processus)
- **Index :** 2 octets
  Emplacement mémoire dans la RAM (arène)
- **Direct :** 2 ou 4 octets
  Valeur numérique

- ❏ Direct : '%' indique une valeur numérique (pas une adresse) 4 octets
- ❏ Indirect : ':' indique une adresse qui contient un adresse qui donnera l'élément 2 octets
- ❏ Relatif : indique l'adresse pointe qui donnera l'élément 1 octet

Operation :

Legende : OCP, paramètre #1, paramètre #2, paramètre #3

- **name : opcode (Hexa)**

- **live : 1 (0x01)**
  - **direct (nb entier sur 4 octets)**
  - **déclare le processus vivant et donc le Champion**
  - **Champion indiqué grâce au nb donné par les 4 octets**
  - **ex: 00 00 00 01 -> Champion 1**
  - **ex: 00 00 00 0b -> Champion 11**
- **ld : 2 (0x02) Carry**
  - **quelconque, registre**
  - **Copie la valeur du paramètre quelconque dans le registre**
  - **quelconque : index ou direct (valeur)**
  - **Si index sera calculée avec PC + (valeur % 512)**
- **st : 3 (0x03)**
  - **OCP, registre, (registre ou indirect)**
  - **Copie la valeur du registre dans le registre ou l'index**
  - **Si index (ex : 42 fera | PC + (42 % 512) soit PC + 42 |)**
- **add : 4 (0x04) Carry**
  - **OCP, registre, registre, registre**
  - **Additionne les 2 premiers registre et met le résultat dans le 3eme**
- **sub : 5 (0x05) Carry**
  - **OCP, registre, registre, registre**
  - **Idem a add mais en soustraction**
- **and : 6 (0x06) Carry**
  - **OCP, registre, registre, registre**
  - **Idem a add mais en effectuant un & binaire**
- **or : 7 (0x07) Carry**
  - **OCP, registre, registre, registre**
  - **Idem a add mais en effectuant un | binaire**
- **xor : 8 (0x08) Carry**
  - **OCP, registre, registre, registre**
  - **Idem a add mais en effectuant un ^ binaire**
- **zjmp : 9 (0x09)**
  - **index**
  - **Si le Carry est à 1 va à l'adresse indiquée par l'index**
  - **Ex : zjmp %:live (l'asm aura mis la valeur pour atteindre le label live)**
  - **Ex : zjmp %5 (va aller 5 octet plus loin)**
  - **Sera calculé avec PC + (index % 512)**
- **ldi : 10 (0x0a)**
  - **(OCP), index, index, registre**

    - **Comme le ld mais va chercher la valeur grâce aux 2 premiers paramètres**
    - **Les 2 paramètres seront calculés avec PC + (index % 512)**
    - **Additionner les 2 premiers paramètres pour avoir l'adresse ou prendre la valeur**
    - **Puis la range dans le registre indique en 3eme paramètre**
- **sti : 11 (0x0b)**
    - **OCP , registre, index ,index**
    - **Comme st mais va additionner les 2 derniers paramètres pour avoir l'adresse ou ranger la valeur du registre donne en 1er paramètre**
    - **Les 2 paramètres seront calculés avec PC + (index % 512)**
- **fork : 12 (0x0c)**
    - **index**
    - **Créer un nouveau processus qui hérite de son père (le processus qui a atteint le fork), hérite de : tous ses registres et son carry. Le PC lui est déterminée grâce au l'index**
    - **Ex: index = 42 | PC + (42 % 512) | soit PC + 42**
- **lld : 13 (0x0d) Carry**
    - **OCP, quelconque, registre**
    - **Copie la valeur du paramètre quelconque dans le registre**
    - **quelconque : index ou direct (valeur)**
    - **Si index n'utilisera pas % 512 et sera calculé telquel**
- **lldi : 14 (0x0e) Carry**
    - **(OCP), index, index, registre**
    - **Comme ldi mais les 2 index n'utiliseront pas %512, seront calculés telquel**
- **lfork : 15 (0x0f)**
    - **index**
    - **Comme le fork mais l'index n'utilisera pas %512**
- **aff : 16 (0x10)**
    - **OCP, registre**
    - **Prend la valeur du registre passer en paramètre**
    - **Applique un % 256 dessus**
    - **Puis affiche le caractère ASCII qui en résulte sur la sortie standard**

OP_TAB :

{"live", 1, {T_DIR}, 1, 10, "alive", 0, 0},
{"ld", 2, {T_DIR | T_IND, T_REG}, 2, 5, "load", 1, 0},
{"st", 2, {T_REG, T_IND | T_REG}, 3, 5, "store", 1, 0},
{"add", 3, {T_REG, T_REG, T_REG}, 4, 10, "addition", 1, 0},
{"sub", 3, {T_REG, T_REG, T_REG}, 5, 10, "soustraction", 1, 0},
{"and", 3, {T_REG | T_DIR | T_IND, T_REG | T_IND | T_DIR, T_REG}, 6, 6, "et(and r1, r2, r3   r1&r2 -> r3", 1, 0},
{"or", 3, {T_REG | T_IND | T_DIR, T_REG | T_IND | T_DIR, T_REG}, 7, 6, "ou (or   r1, r2, r3   r1 | r2 -> r3", 1, 0},
{"xor", 3, {T_REG | T_IND | T_DIR, T_REG | T_IND | T_DIR, T_REG}, 8, 6, "ou (xor r1, r2, r3   r1^r2 -> r3", 1, 0},
{"zjmp", 1, {T_DIR}, 9, 20, "jump if zero", 0, 1},
{"ldi", 3, {T_REG | T_DIR | T_IND, T_DIR | T_REG, T_REG}, 10, 25, "load index", 1, 1},
{"sti", 3, {T_REG, T_REG | T_DIR | T_IND, T_DIR | T_REG}, 11, 25, "store index", 1, 1},
{"fork", 1, {T_DIR}, 12, 800, "fork", 0, 1},
{"lld", 2, {T_DIR | T_IND, T_REG}, 13, 10, "long load", 1, 0},
{"lldi", 3, {T_REG | T_DIR | T_IND, T_DIR | T_REG, T_REG}, 14, 50, "long load index", 1, 1},
{"lfork", 1, {T_DIR}, 15, 1000, "long fork", 0, 1},
{"aff", 1, {T_REG}, 16, 2, "aff", 1, 0},
{0, 0, {0}, 0, 0, 0, 0, 0}

{“name”, nb de paramètre, {type de paramètre possible}, opcode, nb de cycle, “nom complet”, modif carry, taille direct (0 = 4 octets | 1 = 2 octets)}

Longueur opération :

- live : 5 octets | 01 00 00 00 10
- ld : 7 octets | 02 90 00 00 00 01 03 | :D0 -> 5 octets 90 -> 7 octets
- st : 4-5 octets | 03 70 01 00 06 | 50 -> 4 octets 70 -> 5 octets
- add : 5 octets | 04 54 02 03 02
- sub : 5 octets | 05 54 0a 0b 0a
- and : 5-11 octets | 06 64 01 00 00 00 00 01 | 54 -> 5 D4/74 -> 6 F4 -> 7 94/64 -> 8 B4/E4 -> 9 A4 -> 11
- or : 5-11 octets | 07 f4 00 01 00 0f 02 | 54 -> 5 D4/74 -> 6 F4 -> 7 94/64 -> 8 B4/ E4 -> 9 A4 -> 11
- xor : 5-11 octets | 08 54 02 05 0f | 54 -> 5 D4/74 -> 6 F4 -> 7 94/64 -> 8 B4/E4 -> 9 A4 -> 11
- zjmp : 3 octets | 09 ff a7
- ldi : 5-7 octets | 0a d4 00 01 03 04 | 54 -> 5 D4/94/64 -> 6 E4/A4 -> 7
- sti : 5-7 octets | 0b 68 01 00 34 00 01 | 54 -> 5 74/64/58 -> 6 78/68 -> 7
- fork : 3 octets | 0c 00 90
- lld : 5-7 octets | 0d d0 00 05 02 | D0 -> 5 octets 90 -> 7 octets
- lldi : 5-7 octets | 0e d4 00 01 03 04 | 54 -> 5 D4/94/64 -> 6 E4/A4 -> 7
- lfork : 3 octets | 0f 00 90
- aff : 3 octets | 10 40 02

OCP :

40 = R
50 = R R
54 = R R R
58 = R R D
64 = R D R
68 = R D D
70 = R I
74 = R I R
78 = R I D
90 = D R
94 = D R R
A4 = D D R
B4 = D I R
D0 = I R
D4 = I R R
E4 = I R D
F4 = I I R

VM :

<table>
<tr><td rowspan="11">RAM (arène)
4096 octets</td><td colspan="2">Processus : X</td></tr>
<tr><td></td><td>16 registre (4 octets chacun)</td></tr>
<tr><td></td><td>1 registre spé (PC = tete de lecture)</td></tr>
<tr><td></td><td>1 flag (Carry = int)</td></tr>
<tr><td colspan="2">Processus : X+1</td></tr>
<tr><td></td><td>16 registre (4 octets chacun)</td></tr>
<tr><td></td><td>1 registre spé (PC = tete de lecture)</td></tr>
<tr><td></td><td>1 flag (Carry = int)</td></tr>
<tr><td colspan="2">Processus : X+2</td></tr>
<tr><td></td><td>16 registre (4 octets chacun)</td></tr>
<tr><td></td><td>1 registre spé (PC = tete de lecture)</td></tr>
<tr><td></td><td>1 flag (Carry = int)</td></tr>
</table>

Analyse de Zork :

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

live:   live %1
        zjmp %:live
```

L2 et Live sont des labels

sti : va écrire la valeur contenu dans le registre 1, à l'adresse cumuler de la valeur du label live par rapport au PC (10 actuellement) + la valeur direct de 1 (donc si r1 vaut 42, live %1 deviendra live %42)

and : va effectuer {0bxxxxxxxx & 00000000} et renger le résultat dans r1 (donc peut importe la valeur du Champion, va remettre le registre 1 a 0) et cette action va permettre de mettre le carry à 1 pour le zjmp qui suit

live : qui a été modifié, avec le numéro du Champion donné par la VM dans le r1 au lancement du processus, va déclarer le Champion comme vivant

zjmp : va sauter à l'adresse du label live (la valeur hexa est FF FB ce qui vaut 65 531 ou 0b 1111 1111 1111 1011 en unsigned mais en signe cela vaut -5, et donc remonter de 5 octets et être sur l'opcode du live)

Adressage :

Ex :

A17C:022E = 2709258798
A17C * 16 + 022E = 661998 (A19EE)
A * 16 + 19EE = 6798 (1A8E)

2709258798 -> 661998 -> 6798

Fonction (a faire) :

ft_move_pc -> grâce à la section longueur opération, le pc sera sur un op, un fois l'action faite avance le pc du nombre d'octets adéquat pour aller à l'opération suivante

ASM	Parthing	Recup le fichier dans les av et faire un open	
		lire le fichier avec GNL et le stocker (liste chaine)	
	Traduction		
VM	Initialisation		
	Parthing		
	Execution		

VOIR LE POWERPOINT

Comportement VM :

STI : si