Programmation élémentaire

Championnat Corewar

Responsable du module
younes2.serraj@epitech.eu
Dernière modification
01/02/2016_14h30

Table des matières

Détails administratifs

- Le projet est à rendre sur le site web https ://www.codingame.com/hackathon/corewar2020
- Votre champion devra se nommer de la maniére suivante : [ville][promo]login
  - Ville :
    Bordeaux -> BDX
    Lille -> LIL
    Lyon -> LYN
    Marseille -> MAR
    Montpellier -> MPL
    Nancy -> NCY
    Nantes -> NAN
    Nice -> NCE
    Paris -> PAR
    Rennes -> REN
    Strasbourg -> STG
    Toulouse -> TLS
  - promo : La promotion à laquelle vous appartenez (ex : 2020 pour les tek1 actuels)
  - login : Votre login Epitech
  - Vous pouvez retrouver le classement et le mode “tv” à l’adresse suivante : https ://www.codingame.com/leaderboards/global/challenge/corewar2020

Configuration de la VM en ligne

- La VM (Virtual Machine) disponible en ligne est configurée de la façon suivante :
  - `#define CYCLE_TO_DIE 1536`
  - `#define CYCLE_DELTA 4`
  - `#define NBR_LIVE 2048`

Le Corewar : Qu'est ce que c'est que ce truc ?

- Le corewar est un jeu, un jeu très particulier. Il consiste à faire combattre de petits programmes (appelés champions) dans une machine virtuelle (appelée VM).
- Le but du jeu pour les champions est d'être le dernier à survivre.
  Par survivre on entend exécuter une instruction spéciale : "live", qui veut dire "je suis en vie".
- Les champions s'exécutant simultanément dans la VM, ils écrivent tous sur son espace mémoire, et peuvent donc s'écraser les uns sur les autres (d'où l'intérêt en C de la fonction Malloc qui permet d'éviter cela). Il est donc possible (et conseillé) d'empêcher les autres champions de fonctionner correctement, et ce, par tous les moyens disponibles.

Comment ça marche ?

Durant ce premier projet, il ne s'agit que d'écrire des champions.
Vos champions utiliseront un langage assembleur décrit ci-dessous. Ces champions seront ensuite compilés sur notre plate-forme. Vous pourrez alors affronter et vaincre les champions de vos adversaires.

La machine virtuelle (VM) : Qu'est ce que c'est que ce truc ?

- La VM est une machine multi-programmes.
  - Elle dispose d'une mémoire circulaire de 8 Ko, décomposée en registres (petits éléments de mémoire).
  - Elle peut faire tourner de 2 à 4 programmes.
  - Elle fonctionne en cycles d'horloge.
- Le rôle de la VM est d'exécuter les champions qui lui sont donnés en paramètres.
- Elle vérifie que chaque champion exécute l'instruction "live" tous les CYCLE_TO_DIE (fixé à 1500 cycles au départ) (si ce n'est pas le cas, le champion meurt).
- Tous les 30 appels à la fonction "live", CYCLE_TO_DIE est décrémenté de CYCLE_DELTA (fixé à 1) unités si aucun champion n'est mort.
- S'il reste un seul champion vivant, ce dernier a gagné.
- S'il reste au moins deux champions vivants lorsque CYCLE_TO_DIE atteint 100, tous les champions encore vivants finissent à égalité.

Scheduling

- La VM est supposée émuler une machine parfaitement parallèle.
- Pour des raisons d'implémentation, on supposera que chaque instruction s'exécute entièrement à la fin de son dernier cycle et attend durant toute sa durée préalable. Les instructions commençant à un même cycle s'exécutent dans l'ordre croissant des numéros de programme *(voir figure)*.
- Exemple :
  Considérons trois programmes (P1, P2, P3) respectivement constitués des instructions 1.1 à 1.7 , 2.1 à 2.7 et 3.1 à 3.7. Les nombres de cycles de chaque instruction sont donnés entre parenthèses dans le tableau suivant :

P1	1.1(4)	1.2(5)	1.3(8)	1.4(2)	1.5(1)	1.6(3)	1.7(2)
P2	2.1(2)	2.2(7)	2.3(9)	2.4(2)	2.5(1)	2.6(1)	2.7(3)
P3	3.1(2)	3.2(9)	3.3(7)	3.4(1)	3.5(1)	3.6(4)	3.7(9)

La VM exécutera les instructions dans l'ordre suivant :

Cycles	1	2	3	4	5	6	7	8	9	10	11	12	13
Instructions	1,1				1,2					1,3			
Instructions	2,1		2,2							2,3			
Instructions	3,1		3,2									3,3	

Cycles	14	15	16	17	18	19	20	21	22	23	24	25
Instructions					1,4		1,5	1,6			1,7	
Instructions						2,4		2,5	2,6	2,7		
Instructions						3,4	3,5	3,6				3,7

Indices `au cycle 21 la machine exécute 1.6, puis l'instruction 2.5, puis 3.6.`

Le langage

La machine exécute du code machine. Mais pour écrire les programmes, on utilisera un langage simple nommé assembleur. Il est composé d'une instruction par ligne. Les instructions sont composées de trois éléments :

- Un label optionnel suivi du caractère " :"
- Un code d'instruction (opocode).(Les instructions que la machine connaît sont définies dans le tableau op_tab déclaré dans op.c.)
- Les paramètres de l'instruction.

Une instruction peut avoir de 0 à MAX_ARGS_NUMBER paramètres séparés par des virgules. Chaque paramètre peut être de trois types :

- Registre : r1 <–> rx avec x = 16
  Exemple : ld r1,r2. (load r1 dans r2)
- Direct : Le caractère DIRECT_CHAR suivi d'une valeur ou d'un label (précédé de " :"). ce qui représente la valeur directe.
  Exemple : ld %4,r5 (load 4 dans r5)
  Exemple : ld % :label, r7 (load label dans r7)
- Indirect : Une valeur ou un label (précède de LABEL_CHARS) qui représente la valeur qui se trouve à l'adresse du paramètre relativement au PC.
  Exemple : ld 4,r5 (load les 4 octets se trouvant a l'adresse (4+PC) dans r5).

Les instructions

- Chaque instruction nécessite un certain nombre de cycles pour s'exécuter.
- Chaque instruction dispose en mémoire de 16 registres de 4 octets chacun.
- Chaque champion dispose :
  - d'un pc (compteur de programme)
    C'est un registre spécial qui contient l'adresse de la prochaine instruction à décoder et exécuter. Très pratique si on veux savoir où on se trouve et pour écrire des choses en mémoire.
  - d'un carry
    C'est un flag qui vaut 1 si la dernière opération a renvoyé 0.
- La machine doit reconnaître les instructions suivantes :

Mnémonique	Cycles	Effet
live	10	suivie de 4 octets qui représente le numéro du joueur. Cette instruction indique que ce joueur est en vie.
ld	5	2 paramètres (le deuxième est forcément un registre autre que le PC). Elle charge la valeur du premier paramètre dans le registre. Cette opération modifie le carry. *ld 34,r3 charge les 4 octets à partir de l'adresse* $PC + 34\%IDX\_MOD$ *dans le registre r3*
lld	10	même principe que *ld* sans le $\%IDX\_MOD$.
ldi	25	3 paramètres (les 2 premiers sont des index). Cette opération modifie le carry. *ldi 3,%4,r1 lit IND_SIZE octets à l'adresse* $PC + 3\%IDX\_MOD$*, ajoute 4 à cette valeur. On nommera S cette somme. On lit REG_SIZE octet à l'adresse* $PC + S\%IDX\_MOD$ *qu'on copie dans r1*
lldi	50	même principe que *ldi* sans le $\%IDX\_MOD$.

st	8	2 paramètres (le premier est forcément un registre). Elle stocke (4 octets) la valeur du premier argument dans le second. *st r4,34 store la valeur de r4 à l'adresse* $PC + 34\%IDX\_MOD$ *st r3,r8 copie r3 dans r8*
add	10	3 paramètres (nécessairement des registres). Additionne le contenu des 2 premiers et stocke le résultat dans le troisième. Cette opération modifie le carry.
sub	10	même principe que *add* mais soustrait
and	6	3 paramètres (le troisième est toujours un registre). Effectue un "et logique" des 2 premiers et met le résultat dans le troisième. Cette opération modifie le carry. *and r2, %0,r3 met r2 & 0 dans r3*
or	6	même principe que *and* mais effectue un "ou logique".
xor	6	même principe que *and* mais effectue un "ou exclusif".
zjmp	20	1 paramètre (un index) Fait un saut au premier argument si le carry est à 1. Ne fait rien sinon mais consomme le même temps. *zjmp %23 si carry=1 met* $PC + 23\%IDX\_MOD$ *dans le PC*
sti	25 cycles	3 paramètres Les paramètres 2 et 3 sont des index. Si ce sont des registres, on utilisera leur contenu comme un index. *sti r2,%4,%5 copie 4 octets de r2 à l'adresse 4+5*
fork	800	1 paramètre (un index). Elle prend toujours un index et crée un nouveau programme qui s'exécute a partir de l'adresse $PC + paramètre\%IDX\_MOD$. *fork %34 créé un nouveau processus qui hérite des différents états du père.*
lfork	1000	même principe que *fork* sans le $\%IDX\_MOD$.
aff	2	1 paramètre (un registre). Affiche sur la sortie standard le caractère dont le code ascii est présent dans le second argument (un modulo 256 est appliqué au code ascii). *ld %52,r3 puis aff r3 affiche '*' sur la sortie standard*

- Tous les adressages sont relatifs au PC, IDX_MOD sauf "lld", "lldi" et "lfork".
- Tous les autres codes n'ont aucune action à part passer au suivant et perdre un cycle.

Exemple de champion

```
#
# ex.s for corewar
#
.name "zork"
.comment "just a basic living prog"

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

Modus operandi d'un champion

- Un objectif : il ne peut en rester qu'un.
- Au démarrage du jeu et donc de la machine virtuelle, chaque champion va trouver dans son registre perso le numéro qui lui est attribué. Il devra s'en servir pour ses instructions "live".
- Si un champion fait un live avec un numéro autre que le sien, pas de bol, au pire c'est pas perdu pour tout le monde.
- A ce sujet-là, jetez un œil au code donné en exemple pour la partie codage des instructions, il est très utile.
- Toutes les instructions sont utiles, toutes les réactions de la machine qui sont décrites dans ce sujet sont exploitables pour donner de la vie a vos champions.
- Par exemple, quand la machine tombe sur un opcode inconnu, que fait-elle ? Comment alors en détourner l'utilisation ?
- Notez bien que la machine dispose d'une instruction "fork", elle est très utile pour submerger l'adversaire, mais elle prends du temps et peut devenir fatale si la fin du CYCLE_TO_DIE arrive sans qu'elle ait pu se terminer et permettre de faire un "live" derrière !

Le championnat

Le championnat se déroule intégralement au travers du site web corewar-www.epitech.eu.
On se connecte au site web avec son login Unix.
Chaque joueur peut créer jusqu'à 42 champions.

Phase 1 : entraînement

- Explorez le site web et découvrez comment uploader vos champions.
- Votre champion est directement compilé par l'assembleur. Si la structure de votre champion n'est pas bonne, vous ne pourrez pas l'uploader. Seuls les champions fonctionnels seront donc stockés.
- Cette phase dure deux semaines, qui sont les deux semaines du projet.
- Durant cette phase, seuls les matches d'entraînement sont accessibles. Les paramètres de la VM proposés par défaut seront ceux du championnat. N'hésitez pas à tester avec d'autres valeurs pour voir ce que donnent vos champions dans d'autres conditions.

Phase 2 : Le championnat

- Cette phase durera 1 semaine et aura lieu la semaine qui suivra la fin du projet.
- Chaque nuit auront lieu les "Daily Melee".
  - Chaque champion fera 4 matches par nuit.
  - Chaque match nocturne fait s'affronter 4 champions.
  - A chaque mort, tous les vivants marquent 1 point.
    Par exemple, si J1 puis J2 meurent et que J3 et J4 restent vivants, J1 marque 0, J2 marque 1, J3 et J4 marquent 2 points.
  - Les matches restent stockés 2 jours et sont visionnables en différé.
- A tout moment les joueurs peuvent lancer manuellement des Skirmish.
  - Il s'agit d'un duel contre 1 autre champion ayant un classement équivalent.
  - Les points marqués en duel sont doublés par rapport aux combats nocturnes.
  - Profitez de la journée pour défier les autres et grimper dans le classement !

Conclusion

- Pour le reste, réfléchissez, creusez, et dans le cas où il y a une ambiguïté, posez des instructions précises à vos assistants.
- Bon travail et n'oubliez pas : le Corewar est un jeu !