TP compilation : VM

1 BTC : bytecode

On manipule dans les TP une architecture RISC virtuelle 32 bits, donc avec des mots de 4 octets et des instructions codées uniformément sur 4 octets. On suppose disposer de 16 registres (identifiés par les nombres de 0 à 15) et d'une mémoire de 2^{16} mots, dans laquelle les adresses prennent comme unité de base le mot (les architectures réelles ciblent en général plus précisément les octets).

Les 4 octets codant une instruction sont utilisés comme suit.

— Le premier octet contient le code d'opération, qui sera en pratique un entier entre 0 et 31.

— Les trois octets suivants peuvent être utilisés pour désigner les éventuels paramètres de l'instruction. Plus précisément :

  — Le deuxième octet désigne un registre de destination où placer le résultat (ou alternativement dans le cas de l'instruction `WRITE`, le registre contenant l'adresse où écrire le résultat). On note r_d ce paramètre dans le tableau de référence.

  — Le troisième octet désigne le registre contenant le premier opérande (ou alternativement dans le cas de l'instruction `READ`, le registre contenant l'adresse à lire). On le note r_1.

  — Le quatrième octet désigne le registre contenant le deuxième opérande. On le note r_2.

  — Alternativement les deux derniers octets peuvent, ensemble, contenir un nombre entier compris entre 0 et $2^{16} - 1$. On le note i.

Les instructions sont stockées en mémoire à partir de l'adresse 0, et dans l'ordre des adresses croissantes. On se donne les instructions de la figure 1. Par convention, le programme termine par une instruction `EXIT`. Il est suivi par une séquence d'entiers donnant les valeurs initiales des variables globales, qui sont allouées statiquement.

Un fichier de bytecode est un fichier binaire dont les mots de 4 octets consécutifs représentent les instructions et données du programme. Un tel fichier est chargé et exécuté par la machine virtuelle `./VM`.

opcode	nom	paramètres	description
0	`NOP`		pas d'opération
1	`EXIT`		termine l'exécution
2	`PRINT`	r_1	affiche le caractère de code ASCII r_1
3	`CONST`	r_d, i	r_d reçoit i (nombre entier non signé sur 16 bits)
4	`READ`	r_d, r_1	r_d reçoit la valeur à l'adresse r_1
5	`WRITE`	r_d, r_1	écrit r_1 à l'adresse r_d
6	`DIRECTREAD`	r_d, i	r_d reçoit la valeur à l'adresse i
7	`DIRECTWRITE`	r_d, i	écrit r_d à l'adresse i
8	`JUMP`	r_1	saute à l'adresse r_1
9	`JUMPWHEN`	r_1, r_2	saute à l'adresse r_1 si $r_2 \neq 0$
12	`MOVE`	r_d, r_1	r_d reçoit r_1
13	`MINUS`	r_d, r_1	r_d reçoit $-r_1$
14	`NEG`	r_d, r_1	r_d reçoit $\neg r_1$
16	`ADD`	r_d, r_1, r_2	r_d reçoit $r_1 + r_2$
17	`SUB`	r_d, r_1, r_2	r_d reçoit $r_1 - r_2$
18	`MULT`	r_d, r_1, r_2	r_d reçoit $r_1 * r_2$
19	`DIV`	r_d, r_1, r_2	r_d reçoit r_1 / r_2
20	`REM`	r_d, r_1, r_2	r_d reçoit $r_1 \% r_2$
21	`EQ`	r_d, r_1, r_2	r_d reçoit $r_1 = r_2$
22	`NEQ`	r_d, r_1, r_2	r_d reçoit $r_1 \neq r_2$
23	`LT`	r_d, r_1, r_2	r_d reçoit $r_1 < r_2$
24	`LE`	r_d, r_1, r_2	r_d reçoit $r_1 \leq r_2$
25	`GT`	r_d, r_1, r_2	r_d reçoit $r_1 > r_2$
26	`GE`	r_d, r_1, r_2	r_d reçoit $r_1 \geq r_2$
27	`AND`	r_d, r_1, r_2	r_d reçoit $r_1 \wedge r_2$
28	`OR`	r_d, r_1, r_2	r_d reçoit $r_1 \vee r_2$
30	`INCR`	r_d, i	r_d reçoit $r_d + i$
31	`DECR`	r_d, i	r_d reçoit $r_d - i$

FIGURE 1 – Jeu d'instruction de la machine virtuelle