PROJET D'ARCHITECTURE DE L2 MIE
Simulation d'une machine virtuelle

1. Description

Le but de ce projet est de simuler une machine fictive, appelée MIE-L2, composée d'une mémoire et d'un microprocesseur. Vous trouverez ci-dessous la description du processeur. Il y a deux programmes à écrire (ou un seul offrant les deux fonctionnalités). Le premier est un assembleur qui va transformer un programme écrit en assembleur (dans un fichier) en un programme écrit en langage machine (dans un autre fichier). Le deuxième programme est le simulateur proprement dit qui récupère le programme écrit en langage machine et l'exécute instruction après instruction.

La machine MIE-L2

Notre machine comprend une mémoire de 64 kilo-octets, adressée de 0 à 65535. Dans cette mémoire sont stockés le programme, à partir de l'adresse 0, ainsi que les données de ce programme.
Le processeur comprend les registres suivants :

• PC : dans ce registre se trouve l'adresse de la prochaine instruction à exécuter ;

• R0 à R31 : 32 registres généraux de 32 bits chacun. R0 est toujours égal à 0, même après une instruction le modifiant ;

• un registre d'état : ce registre comprend 3 bits intéressants, les bits Z, C et N. Ils sont mis à jour après l'exécution d'une instruction, suivant le résultat de l'instruction. Z est mis à 1 si le résultat est nul, à 0 sinon ; C est mis à 1 s'il y a une retenue, à 0 sinon et N est la recopie du bit de poids fort du résultat.
Aucune instruction ne travaille directement sur ce registre, on pourra donc stocker ces trois bits séparément.

Le langage machine

Chaque instruction est codée sur 32 bits. Voici le format général.

5 bits	5 bits	5 bits	1 bit	16 bits
code op.	Dest (Rd)	Src 1 (Rn)	Imm.	Src 2 (Rm ou #N)

Les 5 premiers bits correspondent au code opératoire (première colonne dans les tables d'instructions).
Les 5 bits suivant donnent le numéro du registre destination (R*d* dans l'assembleur)
Les 5 bits suivant donnent le numéro du premier registre source (R*n* dans l'assembleur).
Le bit suivant indique, quand il est à 1, que la deuxième opérande est une valeur immédiate, codée dans les 16 bits suivant ; s'il est à 0, l'opérande est un registre (*Rm* dans l'assembleur) dont le numéro *m* est codé dans les 5 bits de poids faible du champ Src 2.

i) instructions arithmétiques et logiques

0	`OR  Rd, Rn, S`	Rd ← Rn | S
1	`XOR  Rd, Rn, S`	Rd ← Rn ^ S
2	`AND  Rd, Rn, S`	Rd ← Rn & S
3	`ADD  Rd, Rn, S`	Rd ← Rn + S
4	`SUB  Rd, Rn, S`	Rd ← Rn - S
5	`MUL  Rd, Rn, S`	Rd ← Rn × S (On ne prend que les 16 bits de poids faible de Rn et S)
6	`DIV  Rd, Rn, S`	Rd ← Rn / S (division entière)
7	`SHR  Rd, Rn, S`	Rd ← Rn décalé S fois à droite (ou à gauche si S est négatif), on remplace par 0 les nouveaux bits.

Après l'exécution de ces instructions, les bits d'état sont mis à jour suivant le résultat du calcul. Attention, il s'agit bien du résultat du calcul, même si le registre destination est R0, qui vaut toujours zéro. Dans le cas du décalage (SHR), le bit C reçoit le dernier bit qui disparaît.

S peut valoir *Rm*, le bit Imm est alors à 0 et les 5 bits de poids faible de l'instruction porte la valeur *m* ; soit S vaut une valeur numérique immédiate sur 16 bits, écrite comme #N, avec N en décimal par défaut (ou #hN si elle est en hexadécimal) et elle est codée directement dans les 16 bits de poids faible de l'instruction.

Toutes ces instructions travaillent sur les 32 bits des registres sauf la multiplication qui ne prend que les 16 bits de poids faible des deux opérandes pour que le résultat, sur 32 bits, rentre dans R*d*. Si une valeur immédiate est donnée, elle est étendue sur 32 bits (c'est-à-dire que le bit de signe est répété dans les 16 bits de poids fort).

ii) instructions de transfert

8	`LDB Rd, (Rn)S`	Rd ← contenu de l'adresse (Rn+S), sur 1 octet
9	`LDH Rd, (Rn)S`	Rd ← contenu de l'adresse (Rn+S), sur 2 octets
10	`LDW Rd, (Rn)S`	Rd ← contenu de l'adresse (Rn+S), sur 4 octets
11	`STB (Rd)S, Rn`	l'adresse (Rd+S) reçoit le contenu de Rn, sur 1 octet
12	`STH (Rd)S, Rn`	l'adresse (Rd+S) reçoit le contenu de Rn, sur 2 octets
13	`STW (Rd)S, Rn`	l'adresse (Rd+S) reçoit le contenu de Rn, sur 4 octets

Ces instructions affectent le registre d'état.

LD* charge l'octet de l'adresse précisée dans les 8 bits de poids faible de *Rd*, l'octet suivant dans les 8 bits suivant et ainsi de suite. Si le chargement est sur un demi-mot (LDH) ou un octet (LDB), la valeur est étendue à 32 bits dans *Rd* (c'est-à-dire que le bit de signe est répété dans les bits de poids fort).

ST* sauvegarde en mémoire dans le même ordre.

S peut être *Rm*, l'adresse de l'opérande est alors la somme du contenu des deux registres (et Imm vaut 0), ou une valeur numérique décimale immédiate (ou hexadécimale si elle est précédée d'un h) et l'adresse est alors la somme du contenu du registre et de la valeur (et Imm vaut 1).

iii) instructions de sauts

20	`JMP S`	PC ← S
21	`JZS S`	PC ← S, si Z=1 *(Jump if Z Set)*
22	`JZC S`	PC ← S, si Z=0 *(Jump if Z Clear)*
23	`JCS S`	PC ← S, si C=1 *(Jump if C Set)*
24	`JCC S`	PC ← S, si C=0 *(Jump if C Clear)*
25	`JNS S`	PC ← S, si N=1 *(Jump if N Set)*
26	`JNC S`	PC ← S, si N=0 *(Jump if N Clear)*

S peut être *Rm*, l'adresse de saut est alors égale au contenu du registre (et Imm vaut 0), ou une valeur numérique décimale immédiate (ou hexadécimale si elle est précédée d'un h) et l'adresse est alors directement cette valeur (et Imm vaut 1).

Les 5 bits du champ Rd et les 5 bits du champ Src 1 ne sont pas significatifs. Ces instructions n'affectent pas le registre d'état.

L'exécution se poursuit à l'adresse indiquée si le saut est pris, à l'instruction suivante si ce n'est pas le cas.

iv) instructions d'entrée-sortie

28	`IN Rd`	met dans Rd une valeur entrée par l'utilisateur au clavier
29	`OUT Rd`	affiche à l'écran en décimal le contenu du registre Rd

Ces instructions affectent le registre d'état.
Elles travaillent sur les 32 bits du registre. Les valeurs sont signées en complément à 2.
Les champs Src 1, Imm et Src 2 sont sans signification.

v) instructions diverses

30	`RND Rd, Rn, S`	met dans Rd un nombre aléatoire entier entre Rn et S-1 inclus
31	`HLT`	termine l'exécution du programme

2. Projet

Nous vous demandons d'écrire les deux programmes.

Le premier récupère un fichier texte dans lequel est écrit un programme en assembleur (une instruction par ligne) et génère un fichier texte où est stocké le programme en langage machine (une instruction, soit quatre octets écrits en hexadécimal, par ligne ; on pourra ou non séparer les octets par des espaces). S'il y a des erreurs de syntaxe (mauvaise orthographe, mauvais nombre de paramètres) dans le fichier source, il ne faudra pas générer un fichier code machine mais signaler l'erreur en indiquant la ligne erronée.

Attention, toutes les valeurs numériques peuvent être négatives.

On adoptera une gestion assez libérale des espaces dans le fichier source : il n'y a pas forcément d'espace avant une instruction, il peut y avoir plusieurs espaces entre instruction et opérandes, ceux-ci sont séparés par des virgules avec ou non des espaces…

Une instruction peut avoir une étiquette (c'est un identifiant classique, composé de lettres et éventuellement de chiffres), représentée par « `etiq:` » avant l'instruction. Dans ce cas, un saut « `JMP` (ou autre) `etiq` » pourra s'écrire « `JMP` *adr* » où *adr* est l'adresse calculée de l'étiquette.

La première instruction sera toujours placée à l'adresse 0, la deuxième à l'adresse 4 (puisque chaque instruction fait 32 bits, soit 4 octets), ce qui permet de calculer l'adresse des sauts si ceux-ci sont donnés par une étiquette.

i) exemple

Si le programme à gauche (qui affiche l'inverse d'un nombre tant que l'utilisateur ne rentre pas la valeur 0) est passé à l'assembleur, ce dernier devra générer le fichier de droite :

```
ici:    IN R1                   E0 40 00 00
        JZS fin                 A8 01 00 14
        SUB R1, R0, R1          20 40 00 01
        OUT R1                  E8 40 00 00
        JMP ici                 A0 01 00 00
fin:    HLT                     F8 00 00 00
```

Le deuxième programme simule le fonctionnement de la machine à partir du fichier « code machine », dont les instructions auront été chargées en « mémoire », l'exécution commençant à l'adresse 0.

ii) rendu

Le projet est à faire en binôme. Vous rendrez un rapport avec un listing du programme, un mode d'emploi, une explication sur l'implémentation et un exemple d'exécution donnant, à chaque instruction, le contenu des registres et des zones mémoires utilisées.

Date de remise : 5 février 2018.
Une soutenance devant votre machine sera faite juste après la remise du rapport.