Programmation des systèmes
TP1 – février 2020
Date rendu: 11 mars 2020

Le but de ce TP est d'émuler le fonctionnement d'un processeur. Le programme peut être écrit dans le langage de votre choix avec préférence pour le C qui est le langage le plus utilisé pour la programmation de 'bas' niveau.

Le processeur est constitué de:

- 4 registres généraux notés **r0**, **r1**, **r2**, **r3** (on utilisera R pour n'importe lequel).
- 1 registre particulier de contrôle appelé **pc** (program counter).
- 2 bits d'états C = Carry et N = Negative.

Les registres (rx et pc) sont utilisés pour stocker des informations sur **8 bits** (type *char* en C). Les bits **C** et **N** peuvent prendre deux valeurs 0/1 (true/false).

Le processeur a accès à de la mémoire, un tableau de données sur 8 bits (en C *char mem[32]*).

Etats du processeur/système

A un moment donnée l'état du processeur est déterminé par les valeurs des registres et des bits de contrôle.

(rx, pc, C, N)

On peut voir le processeur comme un automate fini à (2^8)^4 x 2^8 x 2 x 2 états.

L' **état du système** est déterminé par l'état du processeur et les valeurs stockées en mémoire.

(rx, pc, C, N, mem)

On peut voir le système comme un automate fini à (2^8)^4 x 2^8 x2 x 2 x 2^(8*sizeMem).

Les transitions correspondent à l'exécution des instructions.

Boucle principale

Le processeur effectue toujours le même type d'opération.

```
while(true){
        lire l'instruction à exécuter
        décoder l'instruction
        exécuter l'instruction
}
```

L'instruction à exécuter se trouve en mémoire à l'adresse pointée par le **pc, program counter**.

Pour lire l'instruction il faut utiliser un registre interne **ir** et la lecture revient à exécuter ir=mem[pc]. Ensuite, le programme étant séquentiel on veut exécuter l'instruction suivante, pour cela on fait **pc=pc+1**.

Boucle principale

```
pc=0
while(true){
        ir=mem[pc]
        pc=pc+1
        ir'=decode(ir)
        execute(ir')
}
```

Instructions

Les instructions vont provoquer des changements d'états du système, les transitions de l'automate.

On a 9 instructions représentées par des **mnémoniques**

LDM, LDI, STR, ACC, B, BC, BN, MOV, CMP

En général les instructions agissent sur des **opérandes** qui peuvent être soit des registres soit des valeurs (immédiates).

LDM: Load Memory

Syntaxe: LDM r0,adresse

Où adresse est un valeur immédiate codée sur 5 bits, par exemple LDM r0, 17.

Action: r0=mem[adresse]

Instructions

LDI: Load Immediate
Syntaxe: LDI r0, valeur
Valeur est une valeur numérique codée sur 5 bits.
Action: r0=valeur

STR: Store (memory)
Syntaxe: STR r0, adresse
Action: mem[adresse]=r0

ACC: Accumulate
Syntaxe: ACC R1, R2
Action: R1 = R1 + R2
if (R1+R2>=2^8) C=1 **else** C=0
if (R1+R2<0) N=1 **else N**=0

R1,R2 un registre parmi r0,r1,r2,r3

Instructions

B: Branch

Syntaxe: B adresse

Action: pc=adresse

BC: Branch if Carry

Syntaxe: BC adresse

Action: **if** (C=1) pc=adresse **else** nop (nop = no operation)

BN: Branch if Negative

Syntaxe: BN adresse

Action: **if** (N=1) pc=adresse **else** nop (nop = no opertion)

CLR: Clear

Syntaxe: CLR R

Action: R=0

Instructions

CMP: Compare

Syntaxe: CMP R1, R2

Action: **if** (R1-R2<0) N=1 **else** N=0

if (R1+R2>=2^8) C=1 **else** C=0

Les instructions sous la forme de mnémoniques sont utilisées par le programmeur, sous cette forme (lisible) non parle de **langage assembleur** (assembly, assembler language).

Pour le processeur, il faut le convertir en un **langage machine** interprétable par le processeur. C'est l'assembleur qui réalise cette tâche. Les instructions en langage machine sont codés en binaire, souvent on a deux champs pour coder une instruction

code opération (opcode)	code opérandes

Codage des instructions

	7	6	5	4	3	2	1	0
LDM	0	0	0	adresse sur 5 bits				
LDI	0	0	1	valeur sur 5 bits				
STR	0	1	0	adresse sur 5 bits				
ACC	0	1	1	0	R1		R2	
B	1	0	0	adresse sur 5 bits				
BC	1	0	1	adresse sur 5 bits				
BN	1	1	0	adresse sur 5 bits				
CMP	1	1	1	x	R1		R2	
CLR	0	1	1	1	xx		R1	

Remarques/questions

1. Ecrivez un programme qui fait une boucle sans fin.
2. Ecrivez un programme qui calcule la 3+5 et stocke le résultat en mémoire à l'adresse 20.
3. Ecrivez un programme qui fait R1=R2, comme si on avait une instruction MOV R1,R2.
4. Ecrivez un programme qui implémente une instruction SWAP R1,R2.
5. Ecrivez un programme qui calcule 1+2+3+4+5.
6. Ecrivez un programme qui calcule la somme de deux registres et qui tient compte des dépassements de capacité, i.e. la somme de deux nombres sur 8 bits peut ne pas être codable sur 8 bits.
7. Décrivez la structure générale en assembleur d'une structure **if … then … else**.
8. Les instructions sont **toutes** codées sur 8 bits. Expliquez comment il faudrait modifier la boucle principale (diapositive 3) si ce n'était pas le cas.
9. Que faudrait-il modifier si on ne voulait pas limiter les opérandes adresse et valeur à 5 bits?