Architecture avancée : TD5

Exercice 1 : Principes de base

Sujet

Considérons un processeur fictif constitué des éléments suivants :

- machine à mots de 16 bits, adresses sur 12 bits
- 1 accumulateur 16 bits
- compteur ordinal 12 bits
  - jeu de 13 instructions sur 16 bits
  - opérations arithmétiques : addition et soustraction 16 bits, complément à 2
  - chargement et rangement directs et indirects
- saut conditionnel et inconditionnel, appel de sous-programme

Les instructions que ce processeur peut recevoir sont codées sur 16 bits, avec un format unique :

- code opération sur 4 bits (poids forts)
- opérande sur les 12 bits restants

Question 1 : Que représente le mot `0x3064` ?

	Mnémonique	Description	Action	Cp =
0	`loadi` *imm12*	chargement immediat	`Acc = ext(imm12)`	`Cp + 1`
1	`load` *adr12*	chargement direct	`Acc = M[adr12]`	`Cp + 1`
2	`loadx` *adr12*	chargement indirect	`Acc = M[M[adr12]]`	`Cp + 1`
3	`store` *adr12*	rangement direct	`M[adr12] = Acc`	`Cp + 1`
4	`storex` *adr12*	rangement indirect	`M[M[adr12]] = Acc`	`Cp + 1`
5	`add` *adr12*	addition	`Acc += M[adr12]`	`Cp + 1`
6	`sub` *adr12*	soustraction	`Acc -= M[adr12]`	`Cp + 1`
7	`jmp` *adr12*	saut inconditionnel		`adr12`
8	`jneg` *adr12*	saut si négatif		si `Acc < 0` alors `adr12` sinon `Cp+1`
9	`jzero` *adr12*	saut si zero		si `Acc==0` alors `adr12` sinon `Cp+1`
A	`jmpx` *adr12*	saut indirect		`M[adr12]`
B	`call` *adr12*	appel	`M[adr12] = Cp+1`	`M[adr12]+1`
C	`halt 0`	arrêt		

FIGURE 1 – Jeu d'instructions.

- `0x3064` $\longrightarrow$ `store 064` $\Leftrightarrow$ `M[064] <- Acc`
- `0x3064` $\longrightarrow +12\,388$ en complément à $2^{(16)}$

Question 2 : Associer chaque instruction du jeu fourni en Fig. 1 à l'une des 4 classes suivantes :

Transferts	Arithmétique	Branchements	Divers
$[0;4]$	$[5;6]$	$[7;B]$	C

Question 3 : Que signifie "charger un programme" ?

Charger un programme c'est récupérer le code assembleur de ce programme et le copier en mémoire. Il suffit que la première instruction soit bien calée pour que tout se déroule bien ensuite.

Question 4 : Décodez et exécutez le programme suivant : `0009 5005 6006 3007 C000 0005 0003 0000`

Assembleur	Instruction	Traduction
0009	loadi 009	Acc <- ext(9) = 9
5005	add 005	Acc += M[5]
6006	sub 006	Acc -= M[6]
3007	store 007	M[007] <- Acc
C000	halt 0	Arrêt
0005	loadi 005	Acc <- ext(5) = 5 (pas exécuté car arrêt)
0003	loadi 003	Acc <- ext(3) = 3 (pas exécuté car arrêt)
0000	loadi 000	Acc <- ext(0) = 0 (pas exécuté car arrêt)

Question 5 : Que fait le programme suivant ?

```
            loadi       0           ; Acc <- ext(0)     ==> Acc = 0
            store       S           ; S <- Acc          ==> S = 0
            loadi       1           ; Acc <- ext(1)     ==> Acc = 1
            store       K           ; K <- Acc          ==> K = 0
BOUCLE
            load        N           ; Acc <- N          ==> (1) Acc = 5
            sub         K           ; Acc -= K          ==> (1) Acc = 5 - 0 = 5
            jneg        SUITE       ; Saut vers SUITE si Acc < 0
            load        S           ; Acc <- S          ==> (1) Acc = 0
            add         K           ; Acc += K          ==> (1) Acc = 0 + 0 = 0
            store       S           ; S <- Acc          ==> (1) S = 0
            loadi       1           ; Acc <- ext(1)     ==> (1) Acc = 1
            add         K           ; Acc += K          ==> (1) Acc = 1 + 0 = 1
            store       K           ; K <- Acc          ==> (1) K = 1
            jmp         BOUCLE      ; Saut vers BOUCLE
SUITE
            halt        0           ; Arrêt
```

```
# variables

N           word        5           ; N = 5
K           word        0           ; K = 0
S           word        0           ; S = 0
```

Ce programme fait la somme des `N` entiers consécutifs (`S`).

Exercice 2 : Assembleur : ARM

Sujet

On considère que les registres d'un processeur contiennent les huit chiffres hexadécimaux donnés en Fig. 2.

R0	4444 AAAA
R1	8765 4321
R2	FFFF 0000
R3	DBCA 1234
R4	FFFF FFFF
R5	1234 5678
R15	F000 0000

FIGURE 2 – Etat des registres.

Question 1 : Donner le contenu des registres indiqués (sous forme de huit chiffres hexadécimaux) après exécution des instructions suivantes :

Instruction	Résultat
`ADD R6, R1, R0`	`R6 <- CBA9 EDCB`
`SUB R7, R1, R5`	`R7 <- 7530 DCA9`
`ADD R8, R1,R0 LSL #8`	`R8 <- DC0F ED21`
`RSB R9, R1, R2 ASR #4`	`R9 <- 789A ACDF`

Instruction	Résultat
`ORR R10, R3,R5`	`R10 <- DBFE 567C`
`AND R11, R3,R5`	`R11 <- 1200 1230`
`EOR R12,R3,R5`	`R12 <- C9FE 444C`

Attention à la retenue pour `RSB`