

Principe d'exécution des programmes

Examen – B. Miramond – Polytech Nice

On commence par dire : cela est impossible pour se dispenser de le tenter, et cela devient impossible, en effet, parce qu'on ne le tente pas.

Charles Fourier [1772-1837]

Exercice 1 (5 points)

- Refaire à la règle le schéma complet du processeur réalisé en projet et correspondant au jeu d'instruction ARM Cortex-M. Vous ne ferez apparaître que les composants et les signaux principaux.
- Rappeler les 5 tâches de réalisation du projet (une phrase par tâche).

Exercice 2 (3 points)

- Rappelez les différentes étapes du cycle d'exécution machine étudié en cours.

Exercice 3 (4 points)

- On suppose s'intéresser au jeu d'instructions ARM Thumb 16-bit étudié en cours. Préciser le rôle des instructions ARM suivantes (nom complet de l'instruction, préciser le rôle des opérandes + une explication d'une phrase)
 - LDR Rt, [Imm8] *charge une valeur dans la variable Rt*
 - EOR Rdn, Rm *$Rdn = Rdn \oplus Rm$*
 - ADD Rd, Rn, Rm *$Rd = Rn + Rm$*
 - B<c> <label>
- Indiquer le format de codage sur 16 bits de ces 4 types d'instructions sous la forme de schémas suivant le modèle suivant. Quelques indications sont fournies en annexe.

Instruction Inst :

Champs 1 - taille	Champs 2 - taille	Champs 3 - taille	...	Champs n - taille
-------------------	-------------------	-------------------	-----	-------------------

Exercice 4 (5 points)

On suppose maintenant que le processeur exécute le programme assembleur suivant.

```
.global _start
_start:
    .global main
    b      main

main:

    LDR R0, cnt = 0
    LDR R1, cnt = 0
    LDR R2, var1 = 10
    LDR R3, var2 = 1392

loop:
    ADC R1, R0
    SUB R3, R3, R2
    Bne loop
    STR R1, cnt
    STR R3, rem

// Allocation et contenu des variables
var2: .word 0x00186A0
var1: .word 0x000000A
cnt : .word 0x0000000
rem : .word 0x0000000

.end
```

Handwritten notes:
 R1++
 R3 = R3 - R2
 for/while
 for (i=0; i < ?; i++) {
 faire qqch
 }
 while (cont) { faire qqch }

Convertir ces instructions en hexadécimal grâce aux tables de codage de la documentation ARM fournie en annexe. (On ne prendra pas en compte les macros du type .global - symbole global-, .word -allocation 32 bits-, .end -fin de section .text-)

On suppose pour cette conversion que l'assembleur a fait le placement suivant en mémoire.

Symbole	Adresse mémoire en décimal
Main	0
Loop	8
Var2	120
Var1	128
cnt	132
rem	136

Préciser les différentes étapes de codage ou de changement de base.

Exercice 5 (2 points)

Rappeler comment fonctionne l'additionneur complet (avec retenue) réalisant ces calculs au sein du processeur : table de vérité, équation de somme et retenue

En notant que le registre R0 n'est jamais mis à jour, en déduire l'opération réalisée par l'instruction **ADC R1, R0** dans le code précédent.

Cin

Exercice 6 (2 points)

Que fait ce code ? Quelle est la valeur du compteur nommé 'cnt' à la fin de l'exécution ? A quoi correspond la variable 'rem' ? Expliquer en raisonnant en base 10.

Annexe – Codage des instructions Thumb-16bits

CODOP	Catégories d'instructions
00xxxx	Shift, add, sub, move and compare
010000	Data processing
10010	Store immediate
10011	Load immediate
1101	Conditional Branch
<C>	Quelques codes condition de branchement
0000	EQ, Z==1
0001	NE, Z==0
0010	CS, C==1
0011	CC, C==0
0100	MI, N==1
0101	PL, N==0
0110	VS, V==1
0111	VC, V==0,
1000	HI, C==1 and Z==0, Higher, greater than

Code ALU	Opération de l'ALU
0000	Logical And $A \cdot B$
0001	Exclusive logical Or $A \oplus B$
0010	Logical Shift Left
0011	Logical Shift Right
0100	Arithmetic Shift Right
0101	Add with Carry $A+B+Car$
0110	Subtract with Carry $A-B-Car$
0111	Rotate Right
1000	Set flags on bitwise AND
1001	Reverse Subtract from 0 $0-B$
1010	Compare Registers
1011	Compare Negative
1100	Logical OR $A+B$
1101	Multiply Two Registers $A \cdot B$
1110	Bit Clear
1111	Bitwise NOT