UNIVERSITÉ GRENOBLE ALPES
Licence de Sciences et Technologie - Parcours INM, MIN, Min-Int.

Contrôle Continu UE INF401 : Architectures des ordinateurs

Mars 2023, durée 1 h 30
Document : 1 A4 R/V personnel manuscrit autorisé ; calculettes et téléphones portables interdits.
La plupart des questions sont indépendantes, si vous avez du mal avec l'une, passez à la suivante.
Tant que possible, indiquez bien tous les détails et justifiez vos réponses.
Le barème est donné à titre indicatif.

1 Numération, opération en base 2 et en complément à 2 (6 points)

Rappel important : en machine, les entiers relatifs ($\mathbb{Z}$) sont représentés selon la méthode dite du complément à 2 (C2), il ne faut pas confondre cette méthode de représentation avec l'opération dite de complémentation.

(a) Donner la représentation en base 2 sur 16 bits de l'entier naturel 3202 et la représentation binaire sur 8 bits en complément à 2 de l'entier relatif -75. **(2 points)**

Réponse : 0b0000 1100 1000 0010 (0C82 en hexa), 0b01011 0101 (B5 en hexa)

(b) Donner la représentation binaire et la valeur décimale des 2 entiers relatifs suivants codés en complément à 2 sous forme hexadécimale : $(322)_{16}$ et $(FE75)_{16}$. **(2 points)**

Réponse : 802, -395

(c) Effectuer, en écrivant toutes les retenues, les deux opérations suivantes sur 1 octet : $(1100\,1011)_2 + (0110\,0011)_2$ et $(1100\,1011)_2 - (0110\,0011)_2$.
Pour l'addition, donner la valeur des indicateurs (Z, N, C et V). Rappeler le sens de ces indicateurs et indiquer ce qu'ils signifient pour cette addition.
Pour la soustraction, effectuer l'opération par addition du complémentaire. **(2 points)**

Réponse :

```
563d084d> ./add 8 0b11001011 0b01100011

          Bit numbers
          7654 3210                              if natural     if signed

          1100 1011 left   :  0x      cb -->        203 or         -53
        + 0110 0011 right  :  0x      63 -->         99 or         +99
   C=1 == 1000 0110 < c0=0   (in carries)
   V=0  ^ ---- ----
 Z=0 N=0->0010 1110  =     :  0x      2e -->         46 or         +46

563d084d> ./subc2 8 0b1100011 0b01100011
```

```
         Bit numbers
         7654 3210                              if natural     if signed

         1100 1011 left   :  0x      cb -->        203 or          -53
       + 1001 1100 right  :  0x      9c -->        156 or         -100
  C=1 != 0011 1111 < c0=1   (in carries)
  V=1  ^ ---- ----
 Z=0 N=0->0110 1000  =     :  0x      68 -->        104 or         +104
```

2 Petit dessin ASCII (10 points)

Cet exercice a pour objectif de faire un petit dessin ASCII avec des points et des dièses.

```
 1. Tant que tortue<10 faire
 2.    Pour i allant de 0 à 20 faire
 3.       Si i<tortue alors
 4.          EcrCar(Point)
 5.       sinon
 6.          Si i<lievre alors EcrCar(Diese)
 7.          sinon EcrCar(Point) finsi
 8.       finsi
 9.    finpour
10.    ALaLigne
11.    tortue=tortue+1
12.    lievre=lievre+4
13. fintantque
```

Dans l'algorithme, les variables `tortue` et `lievre` sont des entiers relatifs sur 1 mot, placés en mémoire aux étiquettes `tortue` et `lievre` et pré-intialisés à 0 ; la variable `i` est un entiers relatif sur 1 mot que vous placerez dans le registre 5 ; les fonctions `EcrCar` et `ALaLigne` correspondent aux fonctions du fichier `es.s` étudiées en cours et en TP. Le fonctionnement des fonctions d'`es.s` est rappelé en annexe du sujet.

Vous allez maintenant traduire l'algorithme en assembleur `ARM` en vous basant sur le squelette de code donné ci-dessous.

```
  .data
point:  .byte '.'
diese:  .byte '#'
  .balign 4
tortue: .word 0
lievre: .word 0

  .text
  .global main
main:
  push {lr}
```

```
        @ partie manquante

  pop {lr}
  bx lr

LD_point:  .word point
LD_diese:  .word diese
LD_tortue: .word tortue
LD_lievre: .word lievre
```

Questions :

(d) Peut-on enlever le `.balign` de la zone data (sans changer le reste du code) ? Expliquer pourquoi on peut l'enlever ou pourquoi il faut le mettre. **(0,5 point)**.

Réponse : il faut le laisser sinon l'adresse de tortue n'est plus alignée (sur une adresse multiple de 4)

(e) Donner le code `ARM` pour l'affichage à la ligne 4 (affichage point) **(1 point)**.

Réponse :

```
ldr r1, LD_point
ldrb r1, [r1]
bl EcrCar
```

(f) Donner le code `ARM` pour l'affichage à la ligne 10 (passage à la ligne) **(0.5 point)**.

Réponse :

```
bl ALaLigne
```

(g) Donner le code `ARM` pour les affectations lignes 11 et 12 (faire avancer la tortue et le lièvre) **(1 point)**.

Réponse :

```
ldr r0, LD_tortue
ldr r1, [r0]
add r1, r1, #1
str r1, [r0]
ldr r0, LD_lievre
ldr r1, [r0]
add r1, r1, #4
str r1, [r0]
```

(h) Donner le code `ARM` pour la petite conditionnelle lignes 6 et 7 **(2 points)**.

Réponse :

```
    ldr r0, LD_lievre
    ldr r1, [r0]
    cmp r5, r1
    bge sinon
alors:
    ldr r1, LD_diese
    ldrb r1, [r1]
    bl EcrCar
   b fsi
sinon:
    ldr r1, LD_point
    ldrb r1, [r1]
    bl EcrCar
fsi:
```

(i) Donner le code ARM pour la grande conditionnelle lignes 3 – 8 (pour les lignes 6 et 7, **recopier** le code de la question précédente) **(2 points)**.

Réponse :

```
    ldr r2, LD_tortue
    ldr r3, [r2]
    cmp r5, r3
    bge sinonG
alorsG:
    ldr r1, LD_point
    ldrb r1, [r1]
    bl EcrCar
   b fsiG
sinonG:
    ldr r0, LD_lievre
    ldr r1, [r0]
    cmp r5, r1
    bge sinon
alors:
    ldr r1, LD_diese
    ldrb r1, [r1]
    bl EcrCar
   b fsi
sinon:
    ldr r1, LD_point
    ldrb r1, [r1]
    bl EcrCar
fsi:
fsiG:
```

(j) Donner le code `ARM` complet du programme. L'essentiel étant déjà fait, il s'agit surtout d'ajouter le code des deux boucles "tant que" et "pour". Pour le corps de ces boucles, recopier les réponses aux questions précédentes ou indiquer par un commentaire *@ici le code de la question (x), lignes (m) à (n)* sans recopier ce code **(3 points)**.

Réponse :

```
tq: ldr r2, LD_tortue
    ldr r3, [r2]
    cmp  r3, #10
    bge finTQ
    mov r5, #0
pour: cmp r5, #20
   bgt finPour
   @lignes 3 à12 des questions préccédentes
   add r5, r5, #1
   b pour
finPour:
    b tq
finTQ:
```

3 Détection et correction d'erreur (4 points)

En théorie des codes, le Code de Hamming (7,4) est un code correcteur linéaire binaire de la famille des codes de Hamming. À travers un message de sept bits, il transfère quatre bits de données et trois bits de parité. Il permet la correction d'un bit erroné. Autrement dit, si, sur les sept bits transmis, l'un d'eux au plus est altéré (un « zéro » devient un « un » ou l'inverse), alors il existe un algorithme permettant de corriger l'erreur.

Il fut introduit par Richard Hamming (1915-1998) en 1950 dans le cadre de son travail pour les laboratoires Bell. Source : Wikipedia, Code de Hamming (7,4).

3.1 Détection d'erreur

Dans cette partie, un octet sera noté $[PN_6N_5N_4\ N_3N_2N_1N_0]$. Le bit de poids fort sera appelé bit de parité de l'octet et sera noté P. L'information représentée par cet octet sera le nombre binaire sur 7 bits $[N_6N_5N_4\ N_3N_2N_1N_0]$ (sans le bit de parité).

Les octets ayant un nombre pair de bits à 1 seront appelés des octets valides, les autres seront dits invalides. Pour une information donnée (7 bits), il y a toujours un choix judicieux de P possible pour avoir un octet valide : si l'information comporte un nombre pair de bits à 1 il faut prendre P = 0, sinon prendre P=1. À la fin, dans les deux cas, il y a un nombre pair de bits à 1, l'octet est valide.

Après transfert, sauvegarde ou restitution d'un octet valide, si l'octet obtenu n'est plus valide, on dit qu'une erreur est détectée.

Questions (2 points) :

(k) (le sujet initial avait un décalage dans le numéro des dernières questions, pour conserver les correspondances entre questions et réponses, cette pseudo-question est ajoutée dans ce sujet post-édité sans réponse associée.)

(l) Des octets valides ont été transmis, les octets reçus l_1, l_2, l_3 sont les suivants, l_1 : [0110 1101], l_2 : [1000 1001] et l_3 : [1101 0010]. Dire pour chacun des octets reçus s'il est valide ou pas et quand il n'y a pas d'erreur détectée, indiquer l'information transmise (en binaire).

Réponse : l1 : invalide, L2 : invalide, l3 : valide transmet 101 0010

(m) Donner l'octet valide permettant de représenter l'entier 22 (m_1).
Donner l'octet valide (m_2) le plus grand possible (interprétation C2 sur 8 bits de l'octet valide complet) et sa valeur apparente (interprétation C2 des 7 bits d'information).

Réponse : m1 : 1001 0110, m2 : 0111 1110, valeur -2

3.2 Correction d'erreur

Dans cette partie, un octet sera noté $[N_3N_2N_1P_3\ N_0P_2P_1-]$. Le bit de poids faible $(-)$ ne sera pas utilisé. Les bits P_3, P_2 et P_1 seront des bits de parité. Respectivement, P_3 sera le bit de parité du quartet $[P_3N_3N_2N_1]$, P_2 sera le bit de parité du quartet $[P_2N_3N_2N_0]$ et P_1 sera le bit de parité du quartet $[P_1N_3N_1N_0]$. L'octet sera dit valide si les 3 quartets suivants sont valides : $[N_3N_2N_1P_3\ .\ .\ .-]$, $[N_3N_2\ .\ .\ N_0P_2\ .-]$ et $[N_3\ .\ N_1\ .\ N_0\ .\ P_1\ -]$.

Après transfert, sauvegarde ou restitution d'un octet valide, si l'octet obtenu n'est plus valide, on dit qu'une erreur est détectée. Par déduction et en prenant l'hypothèse qu'il n'y a eu qu'un seul bit falsifié lors d'un transfert (d'une sauvegarde ou d'une restitution), si un octet n'est pas valide, en déterminant les quartets non valides, on trouve l'erreur et on peut la corriger.

Pour un octet valide (même après correction), l'information représentée est le quartet $[N_3N_2N_1N_0]$.

Exemple, pour l'octet [0101 001−], le quartets $[N_3N_2N_1P_3\ .\ .\ .-] = [0101\ .\ .\ .-]$ est valide, les quartets $[N_3N_2\ .\ .\ N_0P_2\ .-] = [01\ .\ .\ 00\ .-]$ et $[N_3\ .\ N_1\ .\ N_0\ .\ P_1\ -] = [0\ .\ 0\ .\ 0\ .\ 1-]$ sont invalides. Le bit erroné est à l'intersection des quartets invalides, hors de l'union des quartets valides, ici, c'est N_0, l'octet corrigé est[0101 101−], c'est un octet valide.

Questions (2 points) :

(n) Des octets valides ont été transmis, les octets reçus n_1, n_2 sont les suivants, n_1 : [0011 001−] et n_2 : [1111 100−]. Dire pour chacun des octets reçus s'il est valide ou pas.
Quand il n'y a pas d'erreur détectée, indiquer l'information transmise en l'interprétant comme un nombre relatif représenté en complément à 2 sur 4 bits.
Si l'octet reçu est invalide, en prenant l'hypothèse qu'il n'y a eu qu'un seul bit falsifié, corriger l'octet et indiquer l'information transmise en l'interprétant comme un nombre relatif représenté en complément à 2 sur 4 bits.

Réponse : n1 : valide, transmet 0010, c'est à dire 2, n2 : invalide, correction 1111 000- donc transmet 1110, valeur -2

(o) Donner l'octet valide (o_1) permettant de représenter +6 donnée en complément à 2 sur 4 bits.

Réponse : 6 : 0110, soit 0110 011-

ANNEXES

Fonctions d'entrée/sortie

Nous rappelons les principales fonctions d'affichages du fichier `es.s` :

- `bl EcrCar` affiche le caractère dont le code ascii est dans `r1`.
- `bl ALaLigne` provoque un passage à la ligne dans l'affichage.

Principales instructions du processeur ARM

Code	Nom	Explication du nom	Opération	Remarque
0000	`AND`	AND	et bit à bit	
0010	`SUB`	SUBstract	soustraction	
0100	`ADD`	ADDition	addition	
1000	`TST`	TeST	et bit à bit	pas `rd`
1010	`CMP`	CoMPare	soustraction	pas `rd`
1100	`ORR`	OR	ou bit à bit	
1101	`MOV`	MOVe	copie	pas `rn`
	`Bxx`	Branch	branchement conditionnel	xx = condition
	`LDR`	LoaD Register	lecture mémoire	
	`STR`	STore Register	écriture mémoire	

L'opérande source d'une instruction `MOV` peut être une valeur immédiate notée `#5` ou un registre noté `Ri`, `i` désignant le numéro du registre. Il peut aussi être le contenu d'un registre sur lequel on applique un décalage de `k` bits ; on note `Ri, DEC #k`, avec `DEC` $\in$ {`LSL`, `LSR`, `ASR`, `ROR`}.

Principaux codes conditions du processeur ARM

cond	mnémonique	signification	condition testée
0000	EQ	égal	Z
0001	NE	non égal	$\overline{Z}$
1010	GE	$\geq$ dans Z	$(N \wedge V) \vee (\overline{N} \wedge \overline{V})$
1011	LT	$<$ dans Z	$(N \wedge \overline{V}) \vee (\overline{N} \wedge V)$
1100	GT	$>$ dans Z	$\overline{Z} \wedge ((N \wedge V) \vee (\overline{N} \wedge \overline{V}))$
1101	LE	$\leq$ dans Z	$Z \vee (N \wedge \overline{V}) \vee (\overline{N} \wedge V)$