Architecture des microprocesseurs (CE313)

Examen Session 1 2022-23

Durée totale : 1 heure 30 / Tiers Temps = 2 heures

Toute communication (orale, téléphonique, par messagerie, etc.) avec les autres étudiants est interdite. Aucun document autorisé à l'exception du "mini-isa-riscv".

— Pour la partie QCM, plusieurs réponses peuvent être valides à chaque question, on souhaite avoir **toutes** les réponses valides. Chaque question admet au moins une réponse valide et au moins une réponse incorrecte. Le barème fait perdre des points à chaque erreur, et vous pouvez avoir des points négatifs s'il y a trop d'erreurs. Ne pas répondre à une question de QCM entraîne la note 0 à cette question.

— Les autres parties sont à **rédiger** sur (2) copies d'examen classique. Rédiger, c'est à dire justifier vos réponses.

— Il est probable que l'examen soit **trop long**. Nous adapterons le barème. D'ailleurs, le barème n'est donné qu'à titre indicatif.

---

1 Questions de cours et d'ordres de grandeur

Une feuille R/V contenant des questions à choix multiples ainsi que des questions à réponses brèves est **fournie à part**. Elle est à rendre dans la copie avec la partie **RISCV**.

Solution: L'énoncé de cette partie est à retrouvée à la fin du pdf.

Solution: Les 3 réponses QCM

i) Le cycle d'instructions est la séquence de tâches permettant le traitement d'une instruction par l'unité de calcul et de traitement.

ii) La mémoire d'un microprocesseur est organisée en niveaux avec des accès plus ou moins rapides

iii) Pour un jeu d'instructions donné, les modes d'adressage :

—

— définissent comment une instruction réalise un accès à ses opérandes.

— sont documentés dans le manuel de référence.

Solution: Éléments de réponse aux questions rapides : on peut décider de travailler en puissances de 2 ou de 10 indifféremment lorsqu'il s'agit de gros volumes. *Une réponse non justifiée n'a pas tous les points quand elle est juste, et je ne peux pas donner des points partiels quand elle est fausse.*

i) Puissances

i	1	2	3	4	5	6	7	8	9	10
2^i(dec)	2	4	8	16	32	64	128	256	512	1024
2^i(base 16)	$Ox2$	4	8	10	20	40	80	100	200	400

ii) 1 tera $= 10^{12} = 2^{40}$, et $128 = 2^7$ donc $128Tio = 2^7 \times 2^{40}o = 2^{47}o$. *développer les puissances de 2.*

iii) $4Gio = 4000 \times 10^3 Mio$, donc 128/4000 sec (1/32sec ou 32 ms environ).

iv) Dans le premier cas, on a $64.10^3 \simeq 64 * 2^{10} = 2^{16}$ adresses, codables sur 16 bits. Le deuxième nombre d'adresses est 4 fois moindre, donc 14 bits.

v) IR contient l'instruction courante (assemblée). PC contient l'adresse de la prochaine instruction (ou la courante, suivant où on est dans le cycle). *Attention à l'usage de "pointe vers".*

vi) Une case mémoire proche a plus de chance d'être accédée prochainement. *J'ai accepté des solutions partielles un peu moins précises, du type "la même donnée...", et donné 1/5ime des points pour la mention du cache.*

vii) `02a4fa13` (obtenu avec objdump car : la flemme). Il faut détailler les calculs pour avoir tous les points.

viii) `mv t1,s1` - ou `addi t1, s1,0` quand on désasemble. Attention à détailler les calculs.

2 Programmation RISC-V

On désire dans cet exercice résoudre le problème d'entrée d'un entier au clavier, sachant que l'utilisateur·rice n'entre qu'une série de caractères au clavier. Ainsi, la suite de caractères '1', '0', '2, '5' doit être interprétée en base 10 par le nombre 1025.

Pour simplifier, on suppose qu'une procédure externe a déjà stocké la chaîne de caractère en mémoire, à une adresse fixée. Il ne reste plus qu'à calculer le nombre associé, en lisant la chaîne "de gauche à droite" :

— On transforme chaque caractère chiffre en le chiffre correspondant.

— On applique la méthode de Horner pour calculer le nombre désiré.

Voici le code principal de notre programme (sur 2 colonnes).

```
1  section .text
2  .globl main
3  main:
4          addi    sp,sp,-16
5          sd      ra,8(sp)
6          la a0, .LC1
7          call    println_string
8          # appel à la routine convert
9          la a0 , .LC1
10         call convert
11         # affichage du résultat
12         la a0, .res
13         ld a0, 0(a0)
14         call println_int
15         ld      ra,8(sp)
16         addi    sp,sp,16
17         jr      ra
18         ret
19 ## Partie à compléter
20 # Section données.
21         .section        .data
22         .align  3
23 .LC1:
24         .string "1025"
25
26 .res :
27         .dword 0
28 ### fin du listing
```

La méthode de Horner consiste à utiliser la décomposition en base 10 :

$$1025 = ((1 \times 10 + 0) \times 10 + 2) \times 10 + 5$$

La première étape consiste donc à implémenter une routine `mul10`. On procède sans écrire de boucle, en remarquant que $10n = 2 \times n + 2 \times 2 \times 2 \times n$.

Solution: *Remarque générale de correction : je ne sanctionne pas la syntaxe approximativepour les instructions (par exemple un add avec 2 arguments et pas trois). En revanche, un mv ("move") n'est pas un load qui n'est pas un store, et là ce sont des erreurs de compréhension que je sanctionne.*

Question #1 (3 points)

Écrire la routine `mul10`, qui réalise l'opération $a_0 \leftarrow 10a_0$, en n'utilisant qu'un registre temporaire t_1, et la multiplication par 2 (à l'aide du shift : instruction `slli` "shift left logical immédiate").

Solution:

```
    mult10:
        addi sp,sp,-16
        sd  ra,8(sp)
        # arg is in a0
        slli a0, a0, 1  # 2n
        slli t1, a0, 1  # 4n
        slli t1, t1, 1 # 8n
        add a0, a0, t1 #10n
        ld      ra,8(sp)
        addi    sp,sp,16
        jr      ra
        ret

```

Je n'ai pas mis la totalité des points quand l'algo n'est pas justifié. La documentation du shift n'était pas mentionnée, mais un shift (cf CS351) est un décalage, cad une multiplication par un multiple de 2.

On fournit le pseudo-code de la routine convert :

```
routine convert(paramètre entree: a0){
  s1 = a0;  // copie de l'adresse de la chaîne
  t2 = 0;   // t2 calcule l'entier résultat
  s2 = nouveaucaractère() // à la bonne adresse
  tant que (s2 != '\0'){
       s2 = s2 - '0'; // <-- * à expliquer
       t2 = t2*10;
       t2 = t2+s2;
       s2 = nouveaucaractère()
  }
  memory[res] = t2 ; // stockage du résultat
}
```

Question #2 (2 points)

Dérouler les étapes de cette routine sur la chaîne "1025". Expliquer en particulier à quoi sert la ligne étoilée.

Solution: La ligne étoilée sert à transformer une valeur numérique entière qui est un code ascii (supposé coder un caractère entre '0' et '9') en la valeur numérique du chiffre correspondant. *attention la formulation "je transforme un caractère en entier" n'est pas suffisante*

- Première boucle, on lit '1' dans s2, quand on fait $'1' - '0'$ on récupère la valeur 1. $t2 = t2 * 10 = 0$, et finalement t1 vaut 1.
- Deuxième boucle, on lit '0', dont on récupère la valeur 0. $t2 = 10t_2 + 0 = 10$.
- Troisième boucle, on lit '2', ..., t2 vaut 102 à la fin de cette boucle.
- 4ième boucle, t2 prend la valeur 1020, puis 1025.
- Pas de 5ième boucle car le parcours est terminé.

Dérouler un programme ne consiste pas en une paraphrase des instructions, on donne une entrée précise, on demande de "dérouler", c'est à dire exécuter les étapes du programme sur cette entrée.

Question #3 (4 points)

En suivant les conventions du pseudo code pour le nom des registres, écrivez le code RISCV de la routine `convert`. Attention à l'appel à `mul10`. (On rappelle que le passage des paramètres et du résultat s'effectue via le registre `a0` en RISCV.)

Solution: Les commentaires sont dans le code. L'implémentation de la partie qui récupère un caractère était demandée (pas de call).

```
    convert:
        addi sp,sp,-16
        sd  ra,8(sp)
        # adresse de la chaîne dans a0
        mv s1, a0    # s1 : pointeur de chaîne
        li t2, 0     # t2 : registre de calcul
loop:
        lb s2, 0(s1)             # s2 = *s1  # LOAD BYTE
        beq s2, zero, end        # code 0 = fin de chaine
        addi s2, s2, -'0'        # '1' --> 1, '2' --> 2 etc
        # la suite est pour réaliser t2 = t2*10+s2
        mv a0, t2
        call mult10
        mv t2, a0
        add t2, t2, s2
        # fin de l'opération pour t2
        addi s1, s1, 1 # on passe au caractère suivant
        j loop
end:
        # stockage du résultat en mémoire
        la s1, .res
        sd t2, 0(s1)
        # restauration du contexte
        ld      ra,8(sp)
        addi    sp,sp,16
        jr      ra
        ret

```

La consigne demandait de suivre les conventions du pseudo-code, alors faites le. Un code non commenté ou expliqué est difficile à suivre, alors svp, rédigez !

Remarque à partir d'ici, rédiger les réponses sur une autre copie (partie Nicolas Barbot)

3 Partie ARM

Pour l'ensemble des exercices ci-dessous on donne les instructions ARM suivantes :

```
MOV Rd, Op2          MOVT Rd, Op2           MOVW Rd, Op2           ADD {Rd,} Rn, Op2
ADC {Rd,} Rn, Op2    SUB {Rd,} Rn, Op2      SBC {Rd,} Rn, Op2      RSB {Rd,} Rn, Op2
MUL {Rd,} Rn, Op2    MLA {Rd,} Rn, Rm, Ra   MLS {Rd,} Rn, Rm, Ra   SDIV {Rd,} Rn, Op2
UDIV {Rd,} Rn, Op2   AND {Rd,} Rn, Op2      OOR {Rd,} Rn, Op2      EOR {Rd,} Rn, Op2
ORN {Rd,} Rn, Op2    LSL {Rd,} Rn, Op2      LSR {Rd,} Rn, Op2      ROR {Rd,} Rn, Op2
ASR {Rd,} Rn, Op2    LDR Rd, [Rn {#n}]      LDRB Rd, [Rn {#n}]     LDRH Rd, [Rn {#n}]
STR Rd, [Rn {#n}]    STRB Rd, [Rn {#n}]     STRH Rd, [Rn {#n}]     CMP Rn, Op2
B label              BXX label
```

3.1 Questions de cours

Question #1 (1 point)
Donner le nombre et la taille des registres généraux disponibles sur une architecture Cortex-M.

Question #2 (1 point)
Donner le nom exact du micro-controlleur que vous avez programmé en TP.

Question #3 (1 point)
Donner la fréquence du processeur ARM que vous avez programmé en TP.

Question #4 (1 point)
Donner le type et la taille de la mémoire de ce micro-controlleur.

Question #5 (1 point)
Donner la taille de la mémoire cache du processeur ARM que vous avez programmé en TP.

Solution: TBD

3.2 Une histoire de modulo

Le modulo '%'est définit comme le reste de la division euclidienne de deux entiers non signés. Par exemple `5%2=1`, `8%5=3`, `2%5=2`... L'architecture ARMv7-M ne possède pas d'instruction modulo, à la place, le compilateur génère une suite d'instructions assembleur permettant de calcuer le résultat.

Question #1 (4 points)
Donner, en assembleur ARM, un programme permettant de calculer de résultat de l'opération modulo. On suppose les arguments d'entrée sont placés dans les registres `R0` et `R1` du processeur. Le résultat doit être placé dans le registre `R0` (à la fin du calcul).

Solution: TBD

3.3 Petite devinette

On donne le programme suivant :

```
main: LDR R0, =123456
      MOV R1, #0
      MOV R2, #10
loop: CMP R0, #0
      BEQ R0, stop
      SDIV R0, R0, R2
      ADD R1, R1, #1
      B loop
stop  B stop
```

Question #1 (1 point)
Donner le type de l'adressage utilisé dans la première pseudo-instruction du programme. Expliquer comment cette pseudo-instruction est exécutée.

Question #2 (3 points)
Donner la valeur de `R1` à la fin du programme.

Question #3 (1 point)
Donner la valeur de `R1` si `R2` est chargé avec 16.

Solution: TBD

3.4 Un peu de récursivité

La fonction factorielle est définie par :

$$n! = n \times (n-1) \times (n-2) \times \cdots \times 1 \tag{1}$$

Question #1 (1 point)
Donner, en C, la définition d'une fonction récursive permettant de calculer $n!$.

Question #2 (4 points)
Donner, en assembleur, la définition de la même fonction récursive permettant de calculer $n!$.

Question #3 (1 point)
Pour $n = 3$, dessinez l'état de la pile au moment de l'appel de `fact(1)`.

Solution: TBD

+1/1/60+

Architecture des microprocesseurs - Partie 1 (quick)

Consignes.

- Utiliser un **stylo à bille noir ou bleu foncé**.
- QCM : **noircir ou bleuir** la/les cases, sans dépasser ! (Ne pas cocher)
- Questions rapides : rédiger dans les cadres. La partie grisée sert à la notation.

REMPLIR ICI SVP : NOM (MAJUSCULE) et Prénom :

.....

1 QCM

Question 1 ♣ Le cycle d'instructions est :

- ☐ une suite d'instructions assembleur qui réalise une boucle de calcul
- ☐ un gestionnaire d'événements
- ☐ la séquence de tâches permettant le traitement d'une instruction par l'unité de calcul et de traitement

Question 2 ♣ La mémoire d'un microprocesseur :

- ☐ est organisée en niveaux avec des accès plus ou moins rapides
- ☐ doit être réalisée en assembleur
- ☐ permet de réaliser les opérations arithmétiques
- ☐ permet de réaliser les communications avec les périphériques

Question 3 ♣ Pour un jeu d'instructions donné, les modes d'adressage :

- ☐ définissent comment une instruction réalise un accès à ses opérandes.
- ☐ sont documentés dans le manuel de référence
- ☐ peuvent être modifiés par le.a programmeur·se
- ☐ définissent le nombre et la taille des registres

2 Questions à réponses brèves

Question 4 **(1 point)** Remplir le tableau suivant avec les puissances de 2 en décimal et en hexadécimal.

..... ☐0 ☐1 ☐2 ☐3 ☐4 ☐5 *Prof*

i	1	2	3	4	5	6	7	8	9	10
2^i(dec)	2									
2^i(base 16)	$Ox2$									

Question 5 **(1 point)** Quelle capacité mémoire, exprimée en nombre d'octets, représente 128 Tio. On rappelle que $2^{10} \simeq 10^3$

☐0 ☐1 ☐2 ☐3 ☐4 ☐5 *Prof*

.....

.....

Question 6 **(1 point)** Un bus système présente une bande passante de 4 Gio/s. Combien de temps faut-il pour que 128 Mio transitent sur ce bus ? ☐0 ☐1 ☐2 ☐3 ☐4 ☐5 *Prof*

.....

.....

Question 7 **(1 point)** On considère une mémoire de 64 kio. Sur combien de bits devront être codées (en binaire) les adresses si on suppose :

— que la taille de chaque case de la mémoire est de 1 octet ?

— que la taille de chaque case de la mémoire est de 32 bits ?

☐0 ☐1 ☐2 ☐3 ☐4 ☐5 *Prof*

.....

.....

Question 8 **(1 point)** Quels sont les rôles joués par le registre d'instruction IR et par le compteur de programme PC dans le modèle de Von Neumann ?

☐0 ☐1 ☐2 ☐3 ☐4 ☐5 *Prof*

.....

.....

Question 9 **(1 point)** Énoncer le principe de localité temporelle.

☐0 ☐1 ☐2 ☐3 ☐4 ☐5 *Prof*

.....

.....

Question 10 **(1 point)** Assembler l'instruction RISCV suivante : `ANDI, s4, s1, 42`

☐0 ☐1 ☐2 ☐3 ☐4 ☐5 *Prof*

.....

.....

Question 11 **(1 point)** Désassembler l'instruction RISCV codée en hexadécimal par `0x48313`. Les noms des registres seront fournis suivant la nomenclature standardisée de l'ABI.

☐0 ☐1 ☐2 ☐3 ☐4 ☐5 *Prof*

.....

.....