Examen 1h30
Programmation RISCV
Performance et synchronisation
Documents et calculatrice autorisés

L'examen est composé de 2 exercices chacun noté sur 10 points, **à rédiger sur des feuilles différentes**.

1 Etude d'un programme en langage d'assemblage RISC-V

Vous venez de récupérer le bout de programme en langage d'assemblage RISC-V suivant :

```
800000e0 <inconnu>:
800000e0:     ff410113      addi   sp,sp,-12
800000e4:     00112423      sw     ra,8(sp)
800000e8:     00a12223      sw     a0,4(sp)
800000ec:     00b12023      sw     a1,0(sp)
800000f0:     00058c63      beq    a1, <fin1>
800000f4:     ????????      addi   a1,a1,-1
800000f8:     ????????      jal    ra, <inconnu>
800000fc:     ????????      lw     a2,4(sp)
80000100:     02c50533      mul    a0,a0,a2
80000104:     0080006f      jal    zero, <fin2>

80000108 <fin1>:
80000108:     00100513      addi   a0,zero,1

8000010c <fin2>:
8000010c:     00812083      lw     ra,8(sp)
80000110:     00012583      lw     a1,0(sp)
80000114:     00c10113      addi   sp,sp,12
80000118:     00008067      jalr   zero,0(ra)
```

Il s'agit d'un fichier desassemblé, qui fait apparaître sur une ligne soit l'adresse et l'étiquette, soit l'adresse, le codage binaire de l'instruction et l'instruction désassemblée. Par exemple, à l'adresse 0x800000e0, l'instruction addi sp,sp,-12 est codée en binaire par 0xff410113. L'adresse 0x800000e0 correspond à l'étiquette <inconnu>, nom de la fonction à découvrir.

On cherche en effet à retrouver l'équivalent en C de cette fonction, et sa fonctionnalité !

Question 1 *Vous avez noté la série de ???????? qui indique que le codage binaire de 3 instructions a disparu ! En utilisant en annexe le codage du jeu d'instructions et les numéros de registres, donnez le codage binaire de ces 3 instructions.*

Dans le *main*, on trouve l'appel à la fonction <inconnu> à l'adresse 0x80000080.

```
80000078 <main>:
80000078:     00300513      addi   a0,zero,3
8000007c:     00400593      addi   a1,zero,4
80000080:     060000ef      jal    ra,<inconnu>
...
```

Question 2 *A l'entrée dans la fonction <inconnu>, on observe que sp = 0x80110290. Indiquer les adresses mémoire et le contenu de ces adresses après l'exécution des 4 premières instructions de la fonction <inconnu>.*

Question 3 *Vous avez certainement noté que la fonction <inconnu> contient un appel à elle même. Il s'agit donc d'une fonction récursive. Retrouvez la condition d'arrêt de la fonction, c'est à dire la condition qui stoppe les appels récursifs. Comment l'écririez vous en langage C ?*

Question 4 *On souhaite dessiner la pile qui correspond à l'exécution de la fonction <inconnu>, en commençant à son appel depuis le main et jusqu'à la condition d'arrêt. Indiquer les valeurs et les adresses de façon précise. Indiquer ind quand le contenu de la pile n'est pas précisé.*

Question 5 *Retrouvez l'équivalent en langage C de la fonction <inconnu>. Pour cela, on pourra retrouver le calcul effectué après le retour de l'appel récursif.*

Question 6 *Quelle est la fonctionnalité de la fonction ?*

2 Structure de cache et programmation

Le nouveau processeur P650 de la société SiFive est un multiprocesseur RISCV contenant plusieurs cœurs RISCV 64 bits. Il contient 16 cœurs, 16 Mo de cache L3, 256 Ko de cache L2 et 64 Ko de cache L1 divisé en 32 Ko de cache instructions et 32 Ko de cache données. Comme sur beaucoup de processeurs les adresses utilisées indiquent une adresse d'octet.

Données du problème :

— Dans la suite de cet exercice on s'intéressera uniquement au cache L1 données de 32 Ko. On sait que ce cache est associatif à 4 voies.

— On supposera que les valeurs des adresses réellement utilisées sont sur 44 bits.

— On supposera que les lignes de caches contiennent 8 mots de 64 bits.

Question 7 *Une adresse mémoire arrivant vers le cache sera divisée en plusieurs champs indiqués dans la figure précédente. Quelles sont les caractéristiques (longueur et valeur) des champs* ***Align*** *et* ***Partie haute****. Pourquoi ?*

Question 8 *Le constructeur indique que la partie donnée du cache L1 fait 32 Ko associatif à 4 voies. Les lignes de caches contiennent 8 mots de 64 bits. Quelle est la taille (en octet) d'une ligne ? Quelles sont les largeurs des différents champs* ***Etiquette****,* ***Index*** *et* ***Mot*** *? Indiquez votre raisonnement !*

Question 9 *Quelle est la taille totale du cache en bits ? (la taille des données, des tags, bit de validité et des différentes voies)*

Question 10 *On charge une donnée de 64 bits dont l'adresse est 0x00000E935B87E298. Quel est le numéro de la ligne de cache dans laquelle cette donnée sera stockée ?*

Question 11 *Soit un programme, en C, utilisant un tableau à une dimension contenant 20 mots de 64 bits. Le programme doit faire la somme des 20 valeurs. En supposant qu'un accès à une donnée permet de remplir une ligne entière de cache, combien de défaut de cache ce programme va provoquer ? Pourquoi ?*

RV32I Base Instruction Set

imm[31:12]				rd	0110111	LUI
imm[31:12]				rd	0010111	AUIPC
imm[20|10:1|11|19:12]				rd	1101111	JAL
imm[11:0]		rs1	000	rd	1100111	JALR
imm[12|10:5]	rs2	rs1	000	imm[4:1|11]	1100011	BEQ
imm[12|10:5]	rs2	rs1	001	imm[4:1|11]	1100011	BNE
imm[12|10:5]	rs2	rs1	100	imm[4:1|11]	1100011	BLT
imm[12|10:5]	rs2	rs1	101	imm[4:1|11]	1100011	BGE
imm[12|10:5]	rs2	rs1	110	imm[4:1|11]	1100011	BLTU
imm[12|10:5]	rs2	rs1	111	imm[4:1|11]	1100011	BGEU
imm[11:0]		rs1	000	rd	0000011	LB
imm[11:0]		rs1	001	rd	0000011	LH
imm[11:0]		rs1	010	rd	0000011	LW
imm[11:0]		rs1	100	rd	0000011	LBU
imm[11:0]		rs1	101	rd	0000011	LHU
imm[11:5]	rs2	rs1	000	imm[4:0]	0100011	SB
imm[11:5]	rs2	rs1	001	imm[4:0]	0100011	SH
imm[11:5]	rs2	rs1	010	imm[4:0]	0100011	SW
imm[11:0]		rs1	000	rd	0010011	ADDI
imm[11:0]		rs1	010	rd	0010011	SLTI
imm[11:0]		rs1	011	rd	0010011	SLTIU
imm[11:0]		rs1	100	rd	0010011	XORI
imm[11:0]		rs1	110	rd	0010011	ORI
imm[11:0]		rs1	111	rd	0010011	ANDI
0000000	shamt	rs1	001	rd	0010011	SLLI
0000000	shamt	rs1	101	rd	0010011	SRLI
0100000	shamt	rs1	101	rd	0010011	SRAI
0000000	rs2	rs1	000	rd	0110011	ADD
0100000	rs2	rs1	000	rd	0110011	SUB
0000000	rs2	rs1	001	rd	0110011	SLL
0000000	rs2	rs1	010	rd	0110011	SLT
0000000	rs2	rs1	011	rd	0110011	SLTU
0000000	rs2	rs1	100	rd	0110011	XOR
0000000	rs2	rs1	101	rd	0110011	SRL
0100000	rs2	rs1	101	rd	0110011	SRA
0000000	rs2	rs1	110	rd	0110011	OR
0000000	rs2	rs1	111	rd	0110011	AND
fm | pred | succ		rs1	000	rd	0001111	FENCE
000000000000		00000	000	00000	1110011	ECALL
000000000001		00000	000	00000	1110011	EBREAK

Register	ABI Name	Description	Saver
x0	zero	Hard-wired zero	—
x1	ra	Return address	Caller
x2	sp	Stack pointer	Callee
x3	gp	Global pointer	—
x4	tp	Thread pointer	—
x5	t0	Temporary/alternate link register	Caller
x6–7	t1–2	Temporaries	Caller
x8	s0/fp	Saved register/frame pointer	Callee
x9	s1	Saved register	Callee
x10–11	a0–1	Function arguments/return values	Caller
x12–17	a2–7	Function arguments	Caller
x18–27	s2–11	Saved registers	Callee
x28–31	t3–6	Temporaries	Caller