Corrigé TD 6 – Architecture logicielle

Instructions mémoire et instructions arithmétiques

Dans tout le TD, on considère le jeu d'instructions NIOS – II. Le jeu d'instructions est défini en annexe. Les explications sur le jeu d'instructions seront fournies au fur et à mesure.
Dans ce TD, on considérera essentiellement les instructions mémoire et les instructions arithmétiques.

Format des instructions

Donner le codage hexadécimal des instructions suivantes

a) ADD r3,r1,r2

Format R – A=1 – B= 2 - C=3 - OPX=0x31 - OP = 0x3A –-
00001 00010 00011 00000110001 111010 = $08860C71_H$

b) ADDI r2,r1, -1

Format I – A=1 – B=2 – IMM16 = -1 – OP = 04
00001 00010 1111111111111111 001000 = $08BFFFC8_H$

c) SLLI r2,r1,4

Format R – A=1 – B=0 C=2 – OPX= 12 - IMM5 = 4 – OP = 3A

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
A					0					C					0x12						IMM5					0x3a						

00001 00000 00010 010010 00100 111010 = $0804913A_H$

d) SRLI r2,r1,12

Format R – A=1 – B=0 C=2 – OPX= 1A - IMM5 = 4 – OP = 3A
00001 00000 00010 011010 01100 111010 = $0804D33A_H$

e) SRAI r2,r1,8

Format R – A=1 – B=0 C=2 – OPX= 3A - IMM5 = 4 – OP = 3A
00001 00000 00010 111010 01000 111010 = $0805D243A_H$

f) ORHI r2,r1,0xABCD

Format I – A=1 – B=2 – IMM16 = ABCD – OP = 0x34
00001 00010 1010101111001101 110100 = $08AAF374_H$

Quel est l'intervalle des entiers représentables par le champ immédiat d'une instruction en format I ?
16 bits en complément à 2 soit de -2^{15} à $+2^{15} - 1$

Etude des instructions

Initialement, le registre r1 contient 0x95842103 et le registre r2 contient 0x00001000
En repartant à chaque fois de l'état initial, donner l'état des registres modifiés après exécution des instructions

a) ADD r3,r1,r2
r3 = 95843103

b) ADDI r3, r1, -1
r3=95842102
c) SLLI r2,r1,4
r2 = 58431030
d) SRLI r2,r1,8
r2 = 00958421
e) SRAI r2,r1,12
r2= FF958421
f) ORHI r2,r1, 0xABCD
r2 = BFCD2103
g) ORI r2,r1,0xABCD
r2 = 0x9584ABCF

Suite d'instructions

- Quel est le résultat des deux séquences d'instructions suivantes :

1) Séquence 1

- ORHI r1,r0, 0x9876 r1 = 98760000
- ADDI r1,r1, 0x8432 …...r1 = 98758432

2) Séquence 2

- ORHI r1,r0, 0x9876……..r1 = 98760000
- ORI r1,r1,0x8432 r1 = 98768432

- Ecrire une séquence d'instructions qui positionne le registre r2 à 0x9876

ORI r2,r0, 9876

- Ecrire une séquence d'instructions qui positionne le registre r2 à 0x12348765

ORHI r2,r0,1234
ORI r2,r2 ,8765

- Ecrire un programme qui multiplie par 65 le contenu du registre r1 interprété en non signé, avec résultat dans r2 (sans considérer le problème du dépassement).

- SLLI r2,r1,8
- ADD r2,r2,r1

Programmation

Exercice 1

Donner la suite d'instructions pour exécuter le code C suivant, où les variables A, B et C sont en mémoire à partir de l'adresse 0x1000 0000.

```
int A, B, C ;
A=B+C ;

ORHI r1,r0,1000
LDW r2,4(r1)
LDW r3,8(r1)
ADD r3,r3,r2
STW r3, 0(r1)
```

Exercice 2

Donner la suite d'instructions pour exécuter le code C suivant, où les variables D, E et F sont en mémoire à partir de l'adresse 0x1000 0010.

```
char D,E,F ;
D=E+F ;
```

```
ORHI r1,r0,1000
ORI r1,r1,0010
LDB r2,1(r1)
LDB r3,2(r1)
ADD r3,r3,r2
STB r3, 0(r1)
```

Exercice 3

Donner la suite d'instructions pour exécuter le code C suivant, où les variables H,J,K sont en mémoire à partir de l'adresse 0x1000 0020.

```
unsigned char H,J,K ;
H=J+K ;

ORHI r1,r0,1000
ORI r1,r1,0020
LDBU r2,1(r1)
LDB r3,2(r1)
ADD r3,r3,r2
STB r3, 0(r1)
```

Exercice 4

Donner la suite d'instructions pour exécuter le code C suivant, où les variables L,M,N sont en mémoire à partir de l'adresse 0x1000 0030.

```
short L,M,N ;
L=M+N ;

ORHI r1,r0,1000
ORI r1,r1,0030
LDH r2,2(r1)
LDH r3,4(r1)
ADD r3,r3,r2
STH r3, 0(r1)
```

Jeu d'instructions NIOS II

Formats d'instructions

Le processeur NIOS II a un jeu d'instructions de type RISC. Il possède 32 registres de 32 bits, notés r0 à r31, avec r0 ≡ 0.
Les instructions sont de longueur fixe (32 bits). Il y a trois formats d'instructions.

1. Le format I est indiqué en Figure 1. Le code opération est sur 6 bits. Les champs A et B sur 5 bits contiennent un numéro de registre. Dans la plupart des cas, A indique le numéro du registre source et B le numéro du registre destination. IMM16 est un immédiat sur 16 bits, signé, sauf pour les opérations logiques et pour les comparaisons non signées

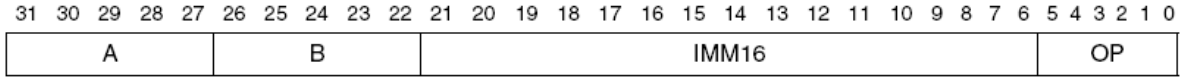

Figure 1 : Instructions de type I

2. Le format R est donné en Figure 2. Le code opération est sur 6 bits. Une extension du code opération est codée sur 11 bits. A et B contiennent les numéros des registres source et C contient le numéro du registre destination.

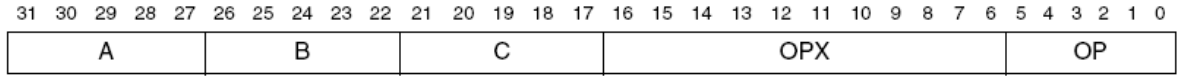

Figure 2 : Instructions de type R

3. Le format J est donné en Figure 3. Le code opération est sur 6 bits et IMMED26 est un immédiat sur 26 bits. Seule l'instruction CALL utilise ce format.

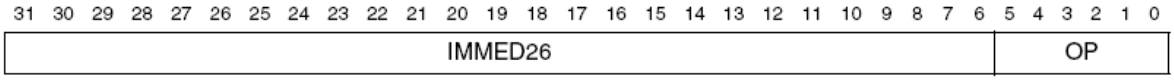

Figure 3 : Instructions de type J

Les différents types d'instructions

Les instructions mémoire

Les instructions load (chargement) et store (rangement) transfèrent des données entre la mémoire (ou les entrées/sorties) et les registres du processeur. Elles utilisent le format de type I.
La mémoire est adressable par octet. Les adresses mémoire sont calculées comme la somme du contenu d'un registre (rA) et d'un immédiat signé de 16 bits (IMM16).
Les instructions sont données dans la table. Les instructions load et store ont des versions spéciales pour les composants d'entrée/sortie. La différence entre les versions mémoire et les versions entrées/sorties, repérées par l'ajout de io, est que ces dernières contournent le cache de données lorsque celui-ci est présent.

ldw / ldwio	ldw rB, imm16(rA)	rB ← Mem32 (rA+simm16)
ldb /ldbio	ldb rB, imm16(rA)	rB ← ES | Mem8 (rA+simm16)
ldbu /ldbuio	ldbu rB, imm16(rA)	rB ← Zero | Mem8 (rA+simm16)
ldh /ldhio	ldh rB, imm16(rA)	rB ← ES | Mem16 (rA+simm16)
ldbu /ldbuio	ldhu rB, imm16(rA)	rB ← Zero | Mem16 (rA+simm16)
stw /stwio	stw rB, imm16(rA)	Mem32 (rA+simm16) ← rB
stb /stbio	stb rB, imm16(rA)	Mem8 (rA+simm16) ← rB
sth /sthio	sth rB, imm16(rA)	Mem16 (rA+simm16) ← rB

Les instructions arithmétiques

Les instructions arithmétiques opèrent sur des opérandes contenus dans des registres (format R) ou un opérande situé dans un registre et un immédiat (format I). Les immédiats sont étendus sur 32 bits (extension de signe).
A noter que selon les versions du processeur NIOS II, les instructions de multiplication et de division sont implantées soit par logiciel, soit par matériel.

add	add rC,rA,rB	rC ← rA + rB
addi	addi rB,rA,imm16	rB ← rA + simm16
sub	sub rC,rA,rB	rC ← rA + rB
subi	subi rB,rA,imm16	rB ← rA - simm16 (pseudo : addi rB, rA, -imm16)
mul	mul rC,rA,rB	rC ← rA * rB (32 bits poids faible du résultat)
muli	muli rB,rA,imm16	rB ← rA * simm16 (32 bits poids faible du résultat)
mulxss	mulxss rC,rA,rB	rC ← rA * rB (32 bits poids fort) rA et rB signés
mulxuu	mulxuu rC,rA,rB	rC ← rA * rB (32 bits poids fort) rA et rB non signés
div	div rC,rA,rB	rC ← rA / rB (nombres signés)
divu	divu rC,rA,rB	rC ← rA / rB (nombres non signés)

Les instructions logiques

Les instructions logiques opèrent sur des opérandes contenus dans des registres (format R) ou un opérande situé dans un registre et un immédiat (format I). Les immédiats sont étendus sur 32 bits (extension de zéros).
Les instructions andhi, orhi, xorhi utilisent un opérande immédiat sur 32 bits en utilisant imm16 comme bits de poids forts et 16 zéros comme bits de poids faible.

and	and rC,rA,rB	rC ← rA et rB (ET logique bit à bit)
andi	andi rB,rA,imm16	rB ← rA et zimm16
or	or rC,rA,rB	rC ← rA ou rB (OU logique bit à bit)
ori	ori rB,rA,imm16	rB ← rA ou zimm16
xor	xor rC,rA,rB	rC ← rA xor rB (OU exclusif bit à bit)
xori	xori rB,rA,imm16	rB ← rA xor zimm16
nor	nor rC,rA,rB	rC ← rA nor rB (NOR bit à bit)
nori	nori rC,rA,rB	rB ← rA nor zimm16
andhi	andhi rB,rA,imm16	rB ← rA et imm16|0000000000000000
orhi	orhi rB,rA,imm16	rB ← rA et imm16|0000000000000000
xorhi	xorhi rB,rA,imm16	rB ← rA et imm16|0000000000000000

Les instructions de transfert

Les instructions de transfert transfèrent le contenu d'un registre ou un immédiat étendu dans un autre registre. Elles sont implantées à l'aide de pseudo-instructions

mov	mov rC,rA	rC ← rA (add rC,rA,r0)
movi	movi rB,imm16	rB ← simm16 (addi rB, r0, imm16)
movui	movui rB,imm16	rB ← zimm16 (ori rB, r0, imm16)

Le transfert d'une adresse de 32 bits dans un registre est implanté par la pseudo-instruction movia rB, ETIQ où ETIQ est une adresse sur 32 bits. Elle est implantée par les deux instructions

```
orhi rB, ro, %hi (ETIQ)      // %hi(ETIQ) correspond aux 16 bits de poids fort
or rB,r0, %lo(ETIQ)          //%lo(ETIQ) correspond aux 16 bits de poids faible
```

Instructions de comparaison

Les instructions de comparaison comparent le contenu de deux registres (format R) ou le contenu d'un registre et d'un immédiat (étendu sur 32 bits) et écrit un 1 (vrai) ou un 0 (faux) dans le registre résultat. Dans le tableau ci-dessous, les instructions en italique sont des pseudo-instructions, implantées par l'assembleur à l'aide d'autres instructions de comparaison et en permutant le contenu des registres pour les instructions de type R, et en modifiant l'immédiat pour les instructions de type I. Dans la table ci-dessous, les instructions en italique sont des pseudo-instructions

cmplt	cmplt rC,rA, rB	rC ← 1 si rA < rB (signés) et ← 0 sinon
cmpltu	cmpltu rC,rA, rB	rC ← 1 si rA < rB (non signés) et ← 0 sinon
cmpeq	cmpeq rC,rA, rB	rC ← 1 si rA == rB (signés) et ← 0 sinon
cmpne	cmpne rC,rA, rB	rC ← 1 si rA != rB (signés) et ← 0 sinon
cmpge	cmpge rC,rA, rB	rC ← 1 si rA >= rB (signés) et ← 0 sinon
cmpgeu	cmpgeu rC,rA, rB	rC ← 1 si rA >= rB (non signés) et ← 0 sinon
cmpgt	*cmpgt rC,rA, rB*	*rC ← 1 si rA > rB (signés) et ← 0 sinon*
cmpgtu	*cmpgtu rC,rA, rB*	*rC ← 1 si rA > rB (non signés) et ← 0 sinon*
cmple	*cmple rC,rA, rB*	*rC ← 1 si rA <= rB (signés) et ← 0 sinon*
cmpleu	*cmpleu rC,rA, rB*	*rC ← 1 si rA <= rB (non signés) et ← 0 sinon*
cmplti	cmplti rA,rA,IMM16	rC ← 1 si rA <simm16 (signés) et ← 0 sinon
cmpltui	cmpltui rA,rA,IMM16	rC ← 1 si rA < zimm16 (non signés) et ← 0 sinon
cmpeqi	cmpeqi rA,rA,IMM16	rC ← 1 si rA == simm16 (signés) et ← 0 sinon
cmpnei	cmpnei rA,rA,IMM16	rC ← 1 si rA != simm16 (signés) et ← 0 sinon
cmpgei	cmpgei rA,rA,IMM16	rC ← 1 si rA >= simm16 (signés) et ← 0 sinon
cmpgeui	cmpgeui rA,rA,IMM16	rC ← 1 si rA >= zimm16 (non signés) et ← 0 sinon
cmpgti	*cmpgti rA,rA,IMM16*	*rC ← 1 si rA > simm16 (signés) et ← 0 sinon*
cmpgtui	*cmpgtiu rA,rA,IMM16*	*rC ← 1 si rA > zimm16 (non signés) et ← 0 sinon*
cmplei	*cmplei rA,rA,IMM16*	*rC ← 1 si rA <= simm16 (signés) et ← 0 sinon*
cmpleui	*cmpleui rA,rA,IMM16*	*rC ← 1 si rA <= zimm16 (non signés) et ← 0 sinon*

Instructions de décalage

Les instructions de décalage décalent le contenu d'un registre vers la droite ou vers la gauche et placent le résultat dans un autre registre. Dans le cas du décalage à droite, le décalage

logique introduit des zéros à gauche alors que le décalage arithmétique réintroduit le bit de signe. Le décalage à gauche introduit des zéros à droite.
Toutes les instructions de décalage sont de type R.
Le nombre de décalages est spécifié soit par les 5 bits de poids faible du registre rB, soit par un immédiat sur 5 bits contenu dans l'extension du code opération du format R.

srl	srl rC,rA, rB	rC ← rA >> (nb spécifié dans rB) - Logique
srli	srli rC,rA, imm5	rC ← rA >> (imm5) - Logique
sra	srl rC,rA, rB	rC ← rA >> (nb spécifié dans rB) - Arithmétique
srai	srai rC,rA, imm5	rC ← rA >> (imm5) - Arithmétique
sll	srl rC,rA, rB	rC ← rA << (nb spécifié dans rB)
slli	srli rC,rA, imm5	rC ← rA << (imm5) - Logique

Instructions de rotation

Il y a trois instructions de rotation, qui utilisent le format R. Pour ces instructions, une rotation à droite décale à droite les bits d'un registre en réintroduisant à gauche les bits sortant à droite et place le résultat dans un autre registre. Le nombre de rotations est spécifié comme le nombre de décalages par les cinq bits de poids faible de rB ou par un immédiat de 5 bits contenu dans l'extension du code opération de l'instruction.

ror	ror rC,rA, rB	rC ← rotation droite de rA (nb spécifié dans rB)
rol	rol rC,rA, rB	rC ← rotation gauche de rA (nb spécifié dans rB)
roli	rol rC,rA, imm5	rC ← rotation gauche de rA (imm5)

Instructions de saut et de branchement.

L'instruction de saut jmp rA effectue un saut à l'adresse dans rA.
L'instruction de branchement inconditionnel br ETIQ branche à l'adresse ETIQ. C'est une instruction de type I. La valeur imm16 correspond au déplacement signé entre l'adresse de l'instruction qui suit l'instruction br et l'adresse cible du branchement (ETIQ).
Les instructions de branchement conditionnel comparent le contenu de deux registres et effectuent le branchement si la condition est vrai ou exécutent l'instruction suivante si la condition est fausse.
Dans la table ci-dessous

- NPC est l'adresse de l'instruction suivant le branchement, soit Adresse de l'instruction de branchement + 4
- simm16 = Adresse cible (ETIQ) – NPC
- Les instructions en italique sont des pseudo-instructions

blt	blt rA,rB, ETIQ	PC ← NPC + simm16 si rA<rB (signés) ; PC ← NPC sinon
bltu	bltu rA,rB, ETIQ	PC ← NPC + simm16 si rA<rB (non signés) ; PC ← NPC sinon
beq	beq rA,rB, ETIQ	PC ← NPC + simm16 si rA==rB (signés) ; PC ← NPC sinon
bne	bne rA,rB, ETIQ	PC ← NPC + simm16 si rA!=rB (signés) ; PC ← NPC sinon
bge	bge rA,rB, ETIQ	PC ← NPC + simm16 si rA>=rB (signés) ; PC ← NPC sinon
bgeu	bgeu rA,rB, ETIQ	PC ← NPC + simm16 si rA>=rB (non signés) ; PC ← NPC sinon
bgt	*bgt rA,rB, ETIQ*	*PC ← NPC + simm16 si rA>rB (signés) ; PC ← NPC sinon*
bgtu	*bgtu rA,rB, ETIQ*	*PC ← NPC + simm16 si rA>rB (non signés) ; PC ← NPC sinon*
ble	*ble rA,rB, ETIQ*	*PC ← NPC + simm16 si rA<=rB (signés) ; PC ← NPC sinon*

bleu	*bleu rA,rB, ETIQ*	*PC ← NPC + simm16 si rA<=rB (non signés) ; PC ← NPC sinon*

Instructions d'appel et retour de fonctions

Il y a deux instructions d'appel de fonctions (sous programmes).
L'instruction call ETIQ est de type J. Elle contient un immédiat sur 26 bits. L'instruction call effectue les actions suivantes :

- Elle range l'adresse de retour (NPC) dans le registre r31.
- Elle saute à l'adresse ETIQ, qui est obtenue par concaténation des 4 bits de poids fort de l'adresse de CALL, des 26 bits de l'immédiat et de 00 (Les deux zéros garantissent que les adresses sont alignées sur des frontières de mots de 32 bits).

L'instruction callr rA effectue les actions suivantes

- Elle range l'adresse de retour (NPC) dans le registre r31.
- Elle saute à l'adresse contenue dans rA.

Le retour de fonction (sous programme) est réalisé par l'instruction ret. C'est une pseudo-instruction qui correspond à jmp r31.

Utilisation des registres

Le processeur NIOS II a 32 registres de 32 bits. Certains registres ont un rôle particulier, et ont un nom spécial reconnu par l'assembleur.

- Le registre r0 contient la constante 0. On ne peut écrire dans ce registre. Il est aussi appelé `zero`
- Le registre r1 est utilisé par l'assembleur comme registre temporaire. Il ne doit pas être utilisé dans les programmes utilisateur.
- Les registres r24 et r29 sont utilisés pour le traitement des exceptions. Ils ne sont pas disponibles en mode utilisateur
- Les registres r25 et r30 sont utilisés exclusivement par le module de débogage JTAG.
- Les registres r27 et r28 sont utilisés pour contrôler la pile
- Le registre r31 est utilisé pour les adresses de retour des fonctions/sous programmes.

Registre	Nom	Fonction
R0	Zero	0x00000000
R1	at	Temporaire pour l'assembleur
R2		
.		
R23		
R24	et	Temporaire pour le traitement des exceptions
R25	bt	Temporaire pour les points d'arrêt
R26	gp	Pointeur global
R27	sp	Pointeur de pile (stack pointer)
R28	fp	Pointeur de trame (Frame pointer)
R29	ea	Adresse de retour des exceptions
R30	ba	Adresse de retour des points d'arrêt
R31	ra	Adresse de retour

Figure 4 : utilisation et nom des registres

Les codes opération

Table 8–1. OP Encodings

OP	Instruction	OP	Instruction	OP	Instruction	OP	Instruction
0x00	call	0x10	cmplti	0x20	cmpeqi	0x30	cmpltui
0x01	jmpi	0x11		0x21		0x31	
0x02		0x12		0x22		0x32	custom
0x03	ldbu	0x13	initda	0x23	ldbuio	0x33	initd
0x04	addi	0x14	ori	0x24	muli	0x34	orhi
0x05	stb	0x15	stw	0x25	stbio	0x35	stwio
0x06	br	0x16	blt	0x26	beq	0x36	bltu
0x07	ldb	0x17	ldw	0x27	ldbio	0x37	ldwio
0x08	cmpgei	0x18	cmpnei	0x28	cmpgeui	0x38	rdprs
0x09		0x19		0x29		0x39	
0x0A		0x1A		0x2A		0x3A	R-type
0x0B	ldhu	0x1B	flushda	0x2B	ldhuio	0x3B	flushd
0x0C	andi	0x1C	xori	0x2C	andhi	0x3C	xorhi
0x0D	sth	0x1D		0x2D	sthio	0x3D	
0x0E	bge	0x1E	bne	0x2E	bgeu	0x3E	
0x0F	ldh	0x1F		0x2F	ldhio	0x3F	

Table 8–2. OPX Encodings for R-Type Instructions (Part 1 of 2)

OPX	Instruction	OPX	Instruction	OPX	Instruction	OPX	Instruction
0x00		0x10	cmplt	0x20	cmpeq	0x30	cmpltu
0x01	eret	0x11		0x21		0x31	add
0x02	roli	0x12	slli	0x22		0x32	
0x03	rol	0x13	sll	0x23		0x33	
0x04	flushp	0x14	wrprs	0x24	divu	0x34	break
0x05	ret	0x15		0x25	div	0x35	
0x06	nor	0x16	or	0x26	rdctl	0x36	sync
0x07	mulxuu	0x17	mulxsu	0x27	mul	0x37	
0x08	cmpge	0x18	cmpne	0x28	cmpgeu	0x38	
0x09	bret	0x19		0x29	initi	0x39	sub
0x0A		0x1A	srli	0x2A		0x3A	srai
0x0B	ror	0x1B	srl	0x2B		0x3B	sra
0x0C	flushi	0x1C	nextpc	0x2C		0x3C	
0x0D	jmp	0x1D	callr	0x2D	trap	0x3D	

Table 8–2. OPX Encodings for R-Type Instructions (Part 2 of 2)

OPX	Instruction	OPX	Instruction	OPX	Instruction	OPX	Instruction
0x0E	and	0x1E	xor	0x2E	wrctl	0x3E	
0x0F		0x1F	mulxss	0x2F		0x3F	