Partiel Architecture des Ordinateurs – 25 Octobre 2019

Tous documents autorisés. Calculatrices interdites. Durée 2h

Une justification concise sera donnée lorsque c'est nécessaire. Lorsqu'un résultat est demandé en notation hexadécimale, la notation binaire ne sera pas acceptée.

1. Représentation des entiers sur 8 bits [4 pts]

Q1. On note # l'opération effectuée par un additionneur sur 8 bits. Pour les opérations suivantes, donner le résultat en notation **hexadécimale**, donner la retenue et indiquer si le résultat est égal à celui de l'opération arithmétique d'addition lorsque les opérandes sont interprétées en naturels ou en relatifs (compléments à deux), suivant le format de la table.

Opération	Résultat	Retenue (0/1)	Correct en naturel(Oui/Non)	Correct en relatifs (Oui/Non)
0x19 # 0x33				
0x57 # 0xC3				
0xFF # 0xFF				
0x11 # 0xAA				

Q2. Donner la représentation en **complément à deux** sur 8 bits (**représentation hexadécimale**) des nombres suivants : -8, -128, 20, 70.

2. Représentation des réels, norme IEEE754 [4 pts]

On considère le format IEEE754 simple précision.

Q3. Coder 40,25 (représentation **hexadécimale**).

Q4. En déduire le code de 40,25 (**représentation hexadécimale**) pour les flottants double précision.

Q5. Donner la valeur (**représentation décimale**) des nombres réels suivants : 0x41000000, 0xBFC00000.

Q6. Sans effectuer de conversion au format IEEE754, donner le résultat de l'addition suivante dans les flottants simple précision : $2^{20} + 1{,}5.2^{-10}$

3. Jeu d'instructions MIPS [6 pts]

Q7. Donner le codage (**représentation hexadécimale**) des instructions :

a) ANDI R2, R4, 0xF0F0

b) OR R2, R2, R3

c) JR R31

Q8. L'état initial des registres est : R1 = 0x73459876 et R2=0xFF0000FF.

Donner le contenu du registre R3 (**représentation hexadécimale**) après l'exécution des instructions

a) ADDIU R3,R2,0x700F

b) OR R3, R1, R2

c) SRA R3, R2, 31

La mémoire est organisée en Big Endian. L'état de la mémoire est donnée par la Table 1. On considère que les variables de type « int » x,y,z sont rangées en mémoire à la suite à partir de l'adresse 0x00001000 :

Adresse	Contenu
0x00001000 - 0x00001003	x
0x00001004 - 0x00001007	y
0x00001008 - 0x0000100B	z

Q9. Donner le code assembleur du Code 1

Code 1
y = 2*x ; z = x+y+z ;

Q10. Donner le contenu du registre R3 (**représentation hexadécimale**) après la séquence suivante

Séquence 1

LI R1, 0x00001000
LI R2, 0XFF00FF00
SW R2, 4(R1)
LB R3,6(R1)
SW R3, 8(R1)

Q11. Données l'état des variables x,y et z (**représentation hexadécimale**) à la suite de cette séquence.

4. Conditionnelles et boucles [4 pts]

Q12. On supposera que les tableaux x, y et z sont implémentés en mémoire aux adresses 0x00001000, 0x00002000 et 0x00003000. La variable s est implémentée à l'adresse 0x1000 0000. Écrire en assembleur MIPS32 le code de la boucle suivante :

```
int x[100];
int y[100];
int z[100] ;
int s=0 ;
for(int i=0 ; i<100 ; i++) {
   z[i] = x[i] +2*y[i]
   s += x[i]
}
```

5. Procédure [3 pts]

On considère le fragment de code suivant :

```
Lb1:
        SLT R3,R1,R0
        BNE R3, R0, p2
        NOT R2,R1
        ADDIU R2, R2, 1
        JR R31
p2:
        ADDIU R2, R1, -1
        NOT R2, R2
        JR R31
main :
        LI R1, 0x7000FFFE
        JAL Lb1
        JR R31
```

Q13. Donner la valeur du registre R2 (**représentation hexadécimale**) après l'exécution de ce programme.

Q14. Si on modifie la première ligne en LI R1, 0xFFFFFFFF, quelle est la valeur de R2 (**représentation hexadécimale**) après exécution ?

Q15 Que fait la « fonction » Lb1 ?

MIPS Reference Sheet

31–26	25–21	20–16	15–11	10–6	5–0	
opcode	rs	rt	rd	shamt	funct	R-type
opcode	rs	rt	immediate			I-type
opcode	target					J-type

R[$x] indicates the register with address x
BMEM indicates a byte aligned access to memory
HMEM indicates a half word aligned access to memory
WMEM indicates a word aligned access to memory

Load and Store Instructions

				Name	Mnemonic	RTL Description
100000	base	dest	signed offset	Load Byte	LB rt,offset(rs)	R[$rt] = SignEXT(BMEM[(R[$rs]+ SignEXT(imm))[31:0]])
100001	base	dest	signed offset	Load Halfword	LH rt,offset(rs)	R[$rt] = SignEXT(HMEM[(R[$rs]+ SignEXT(imm))[31:1]])
100011	base	dest	signed offset	Load Word	LW rt,offset(rs)	R[$rt] = WMEM[(R[$rs]+ SignEXT(imm))[31:2]]
100100	base	dest	signed offset	Load Byte Unsigned	LBU rt,offset(rs)	R[$rt] = ZeroEXT(BMEM[(R[$rs]+ SignEXT(imm))[31:0]])
100101	base	dest	signed offset	Load Halfword Unsigned	LHU rt,offset(rs)	R[$rt] = ZeroEXT(HMEM[(R[$rs]+ SignEXT(imm))[31:1]])
101000	base	dest	signed offset	Store Byte	SB rt,offset(rs)	BMEM[(R[$rs]+SignEXT(imm))[31:0]] = R[$rt][7:0]
101001	base	dest	signed offset	Store Halfword	SH rt,offset(rs)	HMEM[(R[$rs]+SignEXT(imm))[31:1]] = R[$rt][15:0]
101011	base	dest	signed offset	Store Word	SW rt,offset(rs)	WMEM[(R[$rs]+SignEXT(imm))[31:2]] = R[$rt]

R-Type Computational Instructions

						Name	Mnemonic	RTL Description
000000	00000	src	dest	shamt	000000	Shift Left Logical	SLL rd, rt, shamt	R[$rd] = R[$rt] << shamt
000000	00000	src	dest	shamt	000010	Shift Right Logical	SRL rd, rt, shamt	R[$rd] = R[$rt] >> shamt
000000	00000	src	dest	shamt	000011	Shift Right Arithmetic	SRA rd, rt, shamt	R[$rd] = R[$rt] >>> shamt
000000	src1	src2	dest	00000	100000	Add (with overflow)	ADD rd, rs, rt	R[$rd] = R[$rs] + R[$rt]
000000	src1	src2	dest	00000	100001	Add Unsig. (no overflow)	ADDU rd, rs, rt	R[$rd] = R[$rs] + R[$rt]
000000	src1	src2	dest	00000	100010	Subtract	SUB rd, rs, rt	R[$rd] = R[$rs] - R[$rt]
000000	src1	src2	dest	00000	100011	Subtract Unsigned	SUBU rd, rs, rt	R[$rd] = R[$rs] - R[$rt]
000000	src1	src2	dest	00000	100100	And	AND rd, rs, rt	R[$rd] = R[$rs] & R[$rt]
000000	src1	src2	dest	00000	100101	Or	OR rd, rs, rt	R[$rd] = R[$rs] | R[$rt]
000000	src1	src2	dest	00000	100110	Xor	XOR rd, rs, rt	R[$rd] = R[$rs] ^ R[$rt]
000000	src1	src2	dest	00000	100111	Nor	NOR rd, rs, rt	R[$rd] = ~R[$rs] & ~R[$rt]
000000	src1	src2	dest	00000	101010	Set Less Than	SLT rd, rs, rt	R[$rd] = R[$rs] < R[$rt]
000000	src1	src2	dest	00000	101011	Set Less Than Unsig.	SLTU rd, rs, rt	R[$rd] = R[$rs] < R[$rt]

MIPS Reference Sheet

31–26	25–21	20–16	15–11	10–6	5–0	
opcode	rs	rt	rd	shamt	funct	R-type
opcode	rs	rt	immediate			I-type
opcode	target					J-type

R[$x] indicates the register with address x
BMEM indicates a byte aligned access to memory
HMEM indicates a half word aligned access to memory
WMEM indicates a word aligned access to memory

I-Type Computational Instructions

				Name	Mnemonic	RTL$Description
001001	src	dest	signed immediate	Add Imm. Unsigned	ADDIU rt, rs, signed-imm.	R[$rt] = R[$rs] + SignEXT(imm)
001010	src	dest	signed immediate	Set Less Than Imm.	SLTI rt, rs, signed-imm.	R[$rt] = R[$rs] < SignEXT(imm)
001011	src	dest	signed immediate	Set Less Than Imm. Unsig.	SLTIU rt, rs, signed-imm.	R[$rt] = R[$rs] < SignEXT(imm)
001100	src	dest	zero-ext. immediate	And Immediate	ANDI rt, rs, zero-ext-imm.	R[$rt] = R[$rs] & ZeroEXT(imm)
001101	src	dest	zero-ext. immediate	Or Immediate	ORI rt, rs, zero-ext-imm.	R[$rt] = R[$rs] | ZeroEXT(imm)
001110	src	dest	zero-ext. immediate	Xor Immediate	XORI rt, rs, zero-ext-imm.	R[$rt] = R[$rs] ^ ZeroEXT(imm)
001111	00000	dest	zero-ext. immediate	Load Upper Imm.	LUI rt, zero-ext-imm.	R[$rt] = {imm, 0x0000}

Jump and Branch Instructions

						Name	Mnemonic	RTL$Description
000010	target					Jump	J target	PC = {PC[31:28], target, 00}
000011	target					Jump and Link	JAL target	R[31] = PC + 8; PC = {PC[31:28], target, 00}
000000	src	00000	00000	00000	001000	Jump Register	JR rs	PC = R[$rs]
000000	src	00000	dest	00000	001001	Jump and Link Register	JALR rd, rs	R[$rd] = PC + 8; PC = R[$rs]
000100	src1	src2	signed offset			Branch On Equal	BEQ rs, rt, offset	PC = PC + 4 + (R[$rs] == R[$rt] ? SignEXT(imm) << 2 : 0)
000101	src1	src2	signed offset			Branch On Not Equal	BNE rs, rt, offset	PC = PC + 4 + (R[$rs] != R[$rt] ? SignEXT(imm) << 2 : 0)
000110	src	00000	signed offset			Branch On Less Than or Equal to Zero	BLEZ rs, offset	PC = PC + 4 + (R[$rs] <= 0 ? SignEXT(imm) << 2 : 0)
000111	src	00000	signed offset			Branch on Greater than Zero	BGTZ rs, offset	PC = PC + 4 + (R[$rs] > 0 ? SignEXT(imm) << 2 : 0)
000001	src	00000	signed offset			Branch on Less Than Zero	BLTZ rs, offset	PC = PC + 4 + (R[$rs] < 0 ? SignEXT(imm) << 2 : 0)
000001	src	00001	signed offset			Branch on Greater than or Equal to Zero	BGEZ rs, offset	PC = PC + 4 + (R[$rs] >= 0 ? SignEXT(imm) << 2 : 0)

Pseudoinstruction Set

These are simple assembly language instructions that do not have a direct machine language equivalent. During assembly, the assembler translates each pseudo-instruction into one or more machine language instructions.

Name	Mnemonic	RTL$Description
Branch Less Than	BLT rs, rt, label	if(R[$rs]<R[$rt]) PC = Label
Branch Greater Than	BGT rs, rt, label	if(R[$rs]>R[$rt]) PC = Label
Branch Less Than or Equal	BLE rs, rt, label	if(R[$rs]<=R[$rt]) PC = Label
Branch Greater Than or Equal	BGE rs, rt, label	if(R[$rs]>=R[$rt]) PC = Label
Load Immediate	li rd, immediate	R[$rd] = immediate
Move	move rd, rs	R[$rd] = R[$rs]