Partiel Architecture des Ordinateurs – 22 Octobre 2018

Tous documents autorisés. Calculatrices interdites. Durée 2h

Une justification concise sera donnée lorsque c'est nécessaire. Lorsqu'un résultat est demandé en notation héxadécimale, 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)
0x24 # 0x94				
0x80 # 0xFF				
0x7F # 0x01				
0x80 # 0x80				

Q2. Donner la représentation en complément à deux sur 8 bits des nombres suivants : -10, 200, -1, -128.

2. Représentation des réels, norme IEEE754 [3 pts]

On considère le format IEEE754 simple précision.

Q3. Coder -11,5 (représentation **hexadécimale**).

Q4. Donner la valeur (représentation décimale) des nombres réels suivants : 0x41200000, 0xBE000000.

Q5. Effectuer les additions suivantes : 1.25 + 65, $0.125+2^{30}$

3. Jeu d'instructions MIPS [6 pts]

Q6. Donner le codage des instructions :

a) ADDU r1, r10, r2

b) LW r1, 12(r5)

c) SLL r3, r2, 10

d) LUI r2, 0xABCD

Q7. L'état initial des registres est : R1 = 0x85463215 et R2=0x0F0FF0F0.

Donner le contenu du registre R3 après l'exécution des instructions

a) ADDI R3,R2,0xF0FF

b) ANDI R3, R2, 0x0F0F

c) SLT R3, R1, R2

d) SRA R3, R1, 8

Q8. La mémoire est organisée en Big Endian. L'état de la mémoire est donnée par la Table 1. Le registre R2 contient 0x10000000. Donner l'**état final** de la mémoire après exécution de la séquence Seq 1.

Seq 1
LW R1, 0(R2) SLL R1, R1, 4 SB R1, 0(R2) SLL R1, R1, 2 SB R1, 1(R2) SLL R1, R1, 2 SB R1, 2(R2) SLL R1, R1, 2 SB R1, 3(R2)

Table 1	
Adresse	Valeur
0x1000 0000	0x65
0x1000 0001	0xAB
0x1000 0002	0x12
0x1000 0003	0x55
0x1000 0004	0xFE
0x1000 0005	0x65
0x1000 0006	0x15
0x1000 0007	0x4C

4. Conditionnelles et boucles [4 pts]

Q7. On supposera que les tableaux x et y sont implémentés en mémoire aux adresses 0x1000 0000 et 0x2000 0000. La variable s à l'adresse 0x1001 0000. Ecrire en assembleur MIPS32 le code de la boucle suivante :

```
int x[100];
int y[100];
int s=0 ;
for(int i=0 ; i<100 ; i++) { s += x[i] + y[99-i] }
```

5. Procédures [3 pts]

On considère le fragment de code suivant :

```
Lb1 :
        ADDU R3,R0,R1
        SLT R4,R2,R1
        BEQ R4,R0,suite
        ADDU R3,R0,R2
suite :
        JR R31
main :
        LI R1, 0x80001000
        LI R2, 0x50101000
        JAL Lb1
        SUBU R3,R3,R1
```

Q8. Donner la valeur du registre R3 après l'exécution de ce programme.

Q9. Que fait la fonction « Lb1 » ?

Q10. Que faudrait-il rajouter dans le « main » pour pouvoir utiliser ce programme comme une procédure à part entière ?

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]