

TDs - Architecture des ordinateurs Semestre

6

7 Février, 2024

Louis Thevenet

Table des matières

1. TD1	2
1.1. Module registre	2
1.2. Module UAL (Unité arithmétique et logique)	2
1.3. Les instructions et le séquenceur	2
2. TD2	2
3. TD3	2
3.1. Entrées / Sorties	2

1. TD1

Exemple :

```
1 resultat := variable1 + variable2;
```

%r	
%r2	38
%r3	39
%r4	37
%r5	12
%r6	27
%r7	39

```
1 set variable1 %r2
2 set variable2 %r2
3 set resultat %r3
4
5 load(%r2), %r5
6 load(%r3), %r6
7 add %r5, %r6, %r7
8
9 store %r7, [%r4]
```

1.1. Module registre

Définition 1.1.1:

```
1 module registre(rst, clk, areg[31..0], breg[31..0], dreg[31..0], dataIn[31..0] : a[31..0], b[31..0], ir[31..0])
```

areg numéro du registre qu'on souhaite lire sur la sortie **a**
breg numéro du registre qu'on souhaite lire sur la sortie **b**
dreg numéro du registre dans lequel on souhaite écrire l'entrée **dataIn**
IR servira pour accéder directement au code de l'instruction courante sans passer par **a** ou **b**

Exercice 1.1.1:

```
1 module registres(rst, clk, areg[31..0], breg[31..0], dreg[31..0], datain[31..0] : a[31..0], b[31..0], pc[31..0], ir[31..0])
2
3 // constantes
4 r0[31..0] = "00000000000000000000000000000000"
5 r1[31..0] = "00000000000000000000000000000001"
6
7 // où on va écrire
8 decoder4to16(dreg[31..0] : dsel[15..0])
9
10 // écriture
11 reg32_D(rst, clk, dsel[2], datain[31..0] : r2[31..0])
12 reg32_D(rst, clk, dsel[3], datain[31..0] : r3[31..0])
13 reg32_D(rst, clk, dsel[4], datain[31..0] : r4[31..0])
14 reg32_D(rst, clk, dsel[5], datain[31..0] : r5[31..0])
15 reg32_D(rst, clk, dsel[6], datain[31..0] : r6[31..0])
16 reg32_D(rst, clk, dsel[7], datain[31..0] : r7[31..0])
17
18 reg32_D(rst, clk, dsel[12], datain[31..0] : r12[31..0])
19 reg32_D(rst, clk, dsel[13], datain[31..0] : r13[31..0])
20 reg32_D(rst, clk, dsel[14], datain[31..0] : r14[31..0])
21 reg32_D(rst, clk, dsel[15], datain[31..0] : r15[31..0])
22
23
24 // qu'est-ce qu'on met dans a ?
25 decoder4to16(areg[31..0] : asel[15..0])
26 a[31..0] = r0[31..0] * asel[0]
27           + r1[31..0] * asel[1]
28           + r2[31..0] * asel[2]
29           + r3[31..0] * asel[3]
30           + r4[31..0] * asel[4]
31           + r5[31..0] * asel[5]
32           + r6[31..0] * asel[6]
33           + r7[31..0] * asel[7]
34
35           + r12[31..0] * asel[12]
36           + r13[31..0] * asel[13]
37           + r14[31..0] * asel[14]
38           + r15[31..0] * asel[15]
39
40 decoder4to16(breg[31..0] : bsel[15..0])
41 b[31..0] = r0[31..0] * bsel[0]
42           + r1[31..0] * bsel[1]
43           + r2[31..0] * bsel[2]
44           + r3[31..0] * bsel[3]
45           + r4[31..0] * bsel[4]
46           + r5[31..0] * bsel[5]
47           + r6[31..0] * bsel[6]
48           + r7[31..0] * bsel[7]
49
50           + r12[31..0] * bsel[12]
51           + r13[31..0] * bsel[13]
52           + r14[31..0] * bsel[14]
53           + r15[31..0] * bsel[15]
54
55
56 pc[31..0] = r14[31..0]
57 ir[31..0] = r15[31..0]
58 end module
```

1.2. Module UAL (Unité arithmétique et logique)

Exercice 1.2.1:

Opérations :

sigext 1100 (signe extension)

add 0000

sub 0001

```
1 module ual(a[31..0], b[31..0], cmd[31..0] : s[31..0], N, Z, V, C)
2
3 addsub32(a[31..0], b[31..0], cmd[0] : saddsub[31..0], V, C)
4
5 sext[23..0] = a[23..0]
6 sext[31..24] = a[23] * "11111111"
7
8 s[31..0] = saddsub[31..0] * /cmd[3] * /cmd[2] * /cmd[1]
9           + sext[31..0] * cmd[3] * cmd[2] * /cmd[1] * /cmd[0]
10
11 Z = "tous les bits à 0"
12 N = /s[31]
13 end module
```

1.3. Les instructions et le séquenceur

```
1 module sequenceur(rst, clk, ir[31..0], N, Z, V, C : fetch, decode, pcplus1, areg[31..0], breg[31..0], dreg[31..0], ualcmd[31..0], dbusin[1..0], write, setflags)
2
3 areg[31..0] = fetch * "1110"
4               + decode2pcplus1 * ir[23..20]
5               + pcplus1 * "1110" // l'adresse de PC
6
7 breg[31..0] = fetch * "0000"
8               + decode2pcplus1 * ir[19..16]
9               + pcplus1 * "0001" // l'adresse de 1
10
11 dreg[31..0] = fetch * "1111"
12               + decode2pcplus1 * ir[27..24]
13               + pcplus1 * "1110" // l'adresse de PC pour l'addition
14
15 ualcmd[31..0] = fetch * "0000"
16               + decode2pcplus1 * ir[31..28]
17               + pcplus1 * "0000"
18
19 dbusin[1..0] = fetch * "10"
20               + decode2pcplus1 * "01"
21               + pcplus1 * "01"
22
23 write = fetch * "0"
24               + decode2pcplus1 * "0"
25               + pcplus1 * "0"
26
27 setflags = decode2pcplus1
28 end module
```

2. TD2

Exercice 2.1:

- $TQ \ A \neq B$
 - Si $A > B$ Alors
 - $A \leftarrow A - B$
 - Sinon
 - $B \leftarrow B - A$
 - FinSi- FinTQ

```
1 set A, %r1 # r1 vaut l'adresse de A
2 ld [%r1], %r1 # r1 vaut maintenant la valeur de A
3
4 set B, %r2
5 ld [%r2], %r2
6
7 TantQue : cmp %r1 %r2
8           beq FinTantQue
9
10           cmp %r1 %r2
11           blu AsupB
12
13 AinfEqB : sub %r1, %r2, %r1
14           ba TantQue
15
16 AsupB : sub %r2, %r1, %r2
17         ba TantQue
18
19 FinTantQue : Stop
20
21 A : .word 21
22 B : .word 35
```

Exercice 2.2:

- $Fact \leftarrow 1$
- $i \leftarrow N$
- TQ $i > 1$
 - $Fact \leftarrow Fact \times i$
 - $i \leftarrow i - 1$
- FinTQ

```
1 Debut : set 1, %r1 # r1 = Fact
2         set N, %r2 # r2 = i
3
4 TQ : cmp %r2, 1
5       bleu finTQ
6
7       umulcc %r1, %r2, %r1
8       sub %r2, 1, %r2
9       ba TQ
10
11 finTQ : set fact, %r3
12        st %r1, [%r3]
13 Stop : ba stop
14
15
16 N : .word 5
17 fact : .word 0
```

Exercice 2.3:

- $somme \leftarrow 0$
- pour i de 0 à $N - 1$ faire
 - $somme \leftarrow somme + tab[i]$
- finpour

\Leftrightarrow

- $somme \leftarrow 0$
- $i \leftarrow 0$
- tq $i < N$ faire
 - $somme \leftarrow somme + tab[i]$
 - $i \leftarrow i + 1$
- fintq

```
1 Debut : clr %r12 # somme <- 0
2         set Tab, %r2 # r2 <- @debut
3         clr %r3 # i <- 0
4
5 tq : cmp %r3, N
6       bgeu fintq
7       ld [%r2 + %r3], %r4 # %r4 <- tab[i]
8       add %r1, %r4, %r1 # somme <- somme + tab[i]
9       add %r3, 1, %r3
10      ba tq
11
12 fintq : set somme, %r4
13        st %r1, [%r4]
14 Stop : ba stop
15
16 Tab : .word 1,5,3,6,5,8,10,2,8,5
17 somme : .word 0
```

3. TD3

3.1. Entrées / Sorties

Exercice 3.1.1:

```
1 LEDS = 0xB0000000
2
3 Afficher_LEDS_7_0:
4   push %r1
5   push %r20
6
7   set LEDS, %r20
8
9   and %r1, 0xFF, %r1
10
11  st %r1, [%r20]
12  pop %r20
13  pop %r1
14  ret
```

```
1 LEDS = 0xB0000000
2
3 Afficher_LEDS_15_8:
4   push %r1
5   push %r20
6
7   set LEDS, %r20
8
9   and %r1, 0xFF, %r1
10  sll %r1, 8, %r1 # décalage de 8 bits (shift left logical)
11  st %r1, [%r20]
12  pop %r20
13  pop %r1
14  ret
```