
Travaux pratiques d'architecture des ordinateurs

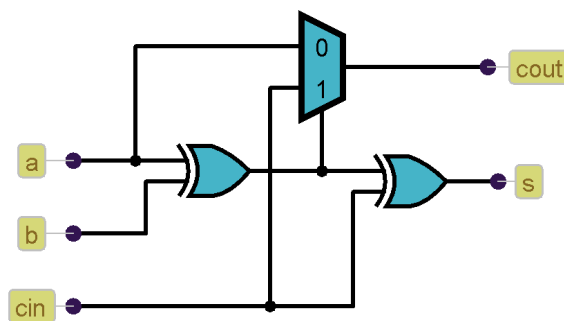
Processeur CRAPS : Unité Arithmétique et Logique (4h)

1. Additionneur complet 1 bit

Concevoir l'additionneur complet 1-bit d'interface :

```
module fulladder(a, b, cin : s, cout)
```

On utilisera la méthode déjà vue :



Tester le module avec les entrées-sorties à distance.

2. Additionneur 32 bits

Chaîner en 'ripple-carry' 32 modules fulladder pour former un additionneur 32 bits d'interface :

```
module adder32(a[31..0], b[31..0], cin : s[31..0], cout)
```

On pourra utiliser des module intermédiaires adder4, adder8, etc. Tester le module avec les entrées-sorties à distance.

3. Additionneur-soustracteur 32 bits

En réutilisant le module adder32 développé précédemment, concevoir un module additionneur soustracteur 32 bits ayant l'interface suivant :

```
module addsub32(a[31..0], b[31..0], sub: s[31..0],V,C)
```

Lorsque `sub = 0` le module calcule la somme `a[31..0] + b[31..0]` sur `s[31..0]` ; lorsque `sub = 1`, il calcule la différence. Les bits `V` et `C` ont le sens habituel des bits de débordement et de retenue-emprunt pour les opérations d'addition et de soustraction.

Tester le module avec les entrées-sorties à distance, en essayant les deux opérations avec des opérandes de signes variés. Vérifier que `C` et `V` ont les valeurs attendues.

4. Décodeurs

Concevoir de façon modulaire les décodeurs, d'interfaces :

```
module decoder2to4(e[1..0] : s[3..0])
module decoder3to8(e[2..0] : s[7..0])
module decoder4to16(e[3..0] : s[15..0])
module decoder5to32(e[4..0] : s[31..0])
module decoder6to64(e[5..0] : s[63..0])
```

Une seule sortie d'un décodeur vaut 1 : celle dont l'indice est le nombre donné en entrée.

Tester `decoder6to64` avec les entrées-sorties à distance.

5. Unité arithmétique et logique

Créer une unité arithmétique et logique ayant l'interface suivant :

```
module ual(a[31..0], b[31..0], cmd[5..0] :
    s[31..0], enN, enZ, enVC, dN, dZ, dV, dC)
```

L'opération est spécifiée par la commande `cmd[5..0]` suivante :

cmd	opération	flags modifiés
010000 (16)	ADDCC, addition	N, Z, V, C
010100 (20)	SUBCC, soustraction	N, Z, V, C
011010 (26)	UMULCC, multiplication non signée	Z
010001 (17)	ANDCC, et logique bit à bit	N, Z
010010 (18)	ORCC, ou logique bit à bit	N, Z
010011 (19)	XORCC, xor logique bit à bit	N, Z
000000 (0)	ADD, addition	aucun
000100 (4)	SUB, soustraction	aucun
000001 (1)	AND, et logique bit à bit	aucun
000010 (2)	OR, ou logique bit à bit	aucun
000011 (3)	XOR, xor logique bit à bit	aucun
001101 (13)	Décalage à droite	aucun

001110 (14)	Décalage à gauche	aucun
100000 (32)	SIGNEXT13, extension de signe bus A, 13 bit → 32 bits	aucun
100001 (33)	SIGNEXT25, extension de signe bus A, 25 bits → 32 bits	aucun
100011 (35)	SETHI, forçage des 24 bits de poids forts	aucun
101000 (40)	NOPB, no operation bus B	aucun

- un multiplieur déjà câblé dans le FPGA sera recruté par utilisation du module prédéfini d'interface :

```
umult16x16(a[15..0], b[15..0]: s[31..0])
```

- les opérations de décalage à droite et à gauche utiliseront le module `barrelshifter32` donné dans `lib` ; la valeur à décaler est prise sur `a[31..0]` et le nombre de bits de décalage est pris sur les 5 bits de poids faibles `b[4..0]`
- l'opération SETHI force les 24 bits de poids fort de `s[31..0]` avec les 24 bits de poids faible de `a[31..0]` et force à 0 tous les 8 bits de poids faible restants
- les opérations d'extension de signe transforment un nombre signé dans la taille de départ en un nombre signé de même valeur dans la taille d'arrivée
- l'opération NOPB transfère simplement la valeur de `b[31..0]` vers `s[31..0]`, sans changement

Les sorties `dN`, `dZ`, `dV` et `dC` indiquent si le résultat de l'opération est négatif (resp. nul, provoque un overflow, une retenue) et seront mémorisés dans des bascules delay `N`, `Z`, `V`, `C` situées dans la micromachine pour former les flags du microprocesseur. Les signaux `enN`, `enZ` et `enVC` indiquent quels flags sont affectés par l'opération demandée ; ils serviront de signal 'enable' pour les bascules `N`, `Z`, `V`, `C`.