Système Digital

Microprocesseur

Implémentation d'une horloge digitale

Auteurs :
Nicolas ASSOUAD

13 décembre 2015

Table des matières

Introduction 5
1 Le Microprocesseur 7
1.1 Le concept 7
1.2 Les instructions matérielles 8
ADD(i) 8
SUB(i) 9
MOVE(i) 9
AND 9
OR 9
XOR 9
NOT 10
LSE 10
RSE 10
EQZ(i) 10
LTZ(i) 11
MTZ(i) 11
1.3 Les macros 12
Conclusion 13

Introduction

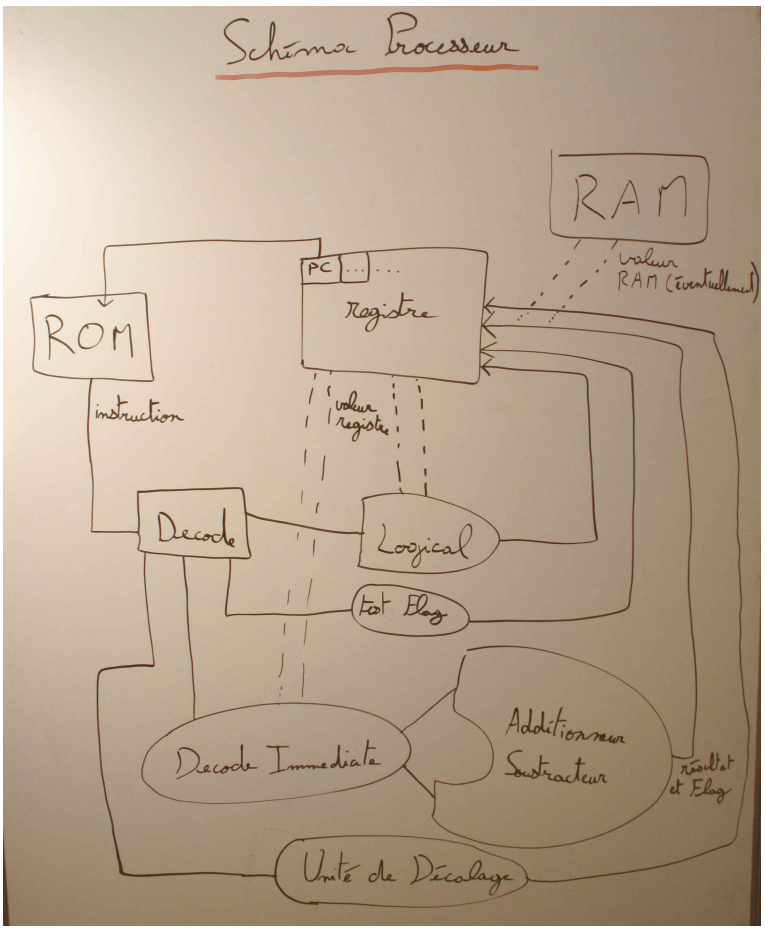
Dans le cadre du cours de Système Digital, nous sommes amenés à concevoir un microprocesseur qui sera capable notamment d'exécuter un programme de montre électronique. Nous allons d'abord présenter les choix de conception de notre microprocesseur puis nous discuterons du programme de l'horloge digitale en lui même.

1 Le Microprocesseur

1.1 Le concept

Notre ligne directrice de conception de notre microprocesseur a été de n'implémenter que les opérations de bases en matériel, et de laisser le soin au programmeur ou à un éventuel compilateur de les combiner pour produire des opérations plus évoluées. Néanmoins, le jeu d'instruction pourrait être encore plus réduit, mais cela ne jouerait pas en faveur des performances.

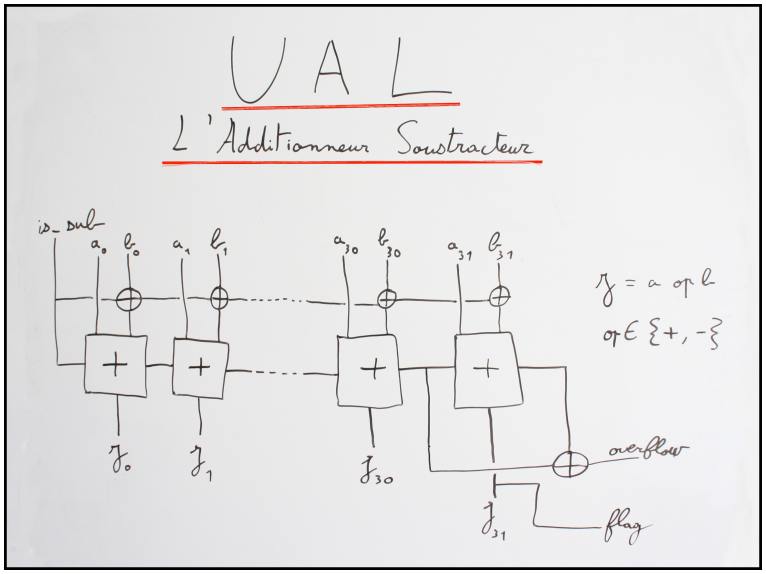
Le schéma global du microprocesseur suit ce schéma de principe (qui se précisera quand la Netlist sera bien finalisé en Minijazz) :



Le microprocesseur possède 32 registres différents dont certain spéciaux comme le pointeur d'instructions PC. Il sont numérotés de 1 à 30, le 31^{ieme} registre étant le registre contenant la valeur du "flag" (il prendra d'ailleurs la dénomination flag dans le code), et le 32^{ieme} registre étant lui le registre du compteur d'instructions (qui sera noté PC dans le code).

Les mots mémoires sont des mots de 32 bits, si bien que toutes les opérations matérielles se font sur des entiers signés de 32 bits (à cela, il faut noter l'exception

des calculs avec des valeurs immédiates dans les instructions qui, elles, sont représentées sur 16 bits). Cela donne une implémentation de l'additionneur/soustracteur que le peut représenter de la manière suivante par un schéma :



1.2 Les instructions matérielles

Les instructions du microprocesseur sont codé sur 27 bits. Les 4 premiers bits codent l'instruction voulue. Le reste du codage l'instruction dépend de sa nature. Si la description du codage d'une instruction ne va pas jusqu'à 27 bits (notamment quand il n'y a pas d'argument entier), il est sous entendu que le code de l'instruction est complété par des zéros à la fin.

Le microprocesseur supporte les 12 instructions suivantes.

ADD(i)

L'instruction ADD implémente l'addition. Elle s'emploie de la façon suivante : ADD a b. Cette instruction calcule la somme des valeurs contenues dans les mémoires a et b, et place le résultat dans la mémoire b.
L'instruction supporte que sont premier paramètre soit directement un nombre, l'addition s'effectue alors directement avec celui-ci.

Le code de l'instruction prend la forme suivante :

0000	0	premier registre (a)	second registre (b)

Dans le cas ou le premier argument est directement un entier (instruction ADDi), le code de l'instruction prend cette forme :

0000	1	entier (sur 16 bits)	registre (b)

SUB(i)

L'instruction SUB implémente la soustraction. Elle s'emploie de la façon suivante : SUB a b. Cette instruction calcule la différence des valeurs contenues dans les mémoires a et b (b - a), et place le résultat dans la mémoire b.
L'instruction supporte que sont premier paramètre soit directement un nombre, la soustraction s'effectue alors directement avec celui-ci.

Le code de l'instruction prend la forme suivante :

0001	0	premier registre (a)	second registre (b)

Dans le cas ou le premier argument est directement un entier (instruction SUBi), le code de l'instruction prend cette forme :

0001	1	entier (sur 16 bits)	registre (b)

MOVE(i)

L'instruction MOVE implémente la copie d'octet

AND

L'instruction AND implémente un "et" logique bit à bit. Elle s'emploie de la façon suivante : ADD a b. Cette instruction calcule le "et" logique entre les bits de la mémoire de a et b, et place le résultat dans la mémoire b.

Le code de l'instruction prend la forme suivante :

0011	0	premier registre (a)	second registre (b)

OR

L'instruction OR implémente un "ou" logique bit à bit. Elle s'emploie de la façon suivante : OR a b. Cette instruction calcule le "ou" logique entre les bits de la mémoire de a et b, et place le résultat dans la mémoire b.

Le code de l'instruction prend la forme suivante :

0100	0	premier registre (a)	second registre (b)

XOR

L'instruction XOR implémente un "ou exclusif" logique bit à bit. Elle s'emploie de la façon suivante : XOR a b. Cette instruction calcule le "ou exclusif" logique entre les bits de la mémoire de a et b, et place le résultat dans la mémoire b.

Le code de l'instruction prend la forme suivante :

0101	0	premier registre (a)	second registre (b)

NOT

L'instruction NOT implémente un "non" logique bit à bit. Elle s'emploie de la façon suivante : NOT a b. Cette instruction calcule le "non" logique des bits de la mémoire de a, et place le résultat dans la mémoire b.

Le code de l'instruction prend la forme suivante :

0110	0	premier registre (a)	second registre (b)

LSE

L'instruction LSE implémente un décalage de un bit à gauche. Elle s'emploie de la façon suivante : LSE a. Cette instruction décale les bits à gauche de la mémoire de a, et place le résultat dans la mémoire a.

Le code de l'instruction prend la forme suivante :

0111	0	registre (a)

RSE

L'instruction RSE implémente un décalage de un bit à droite. Elle s'emploie de la façon suivante : RSE a. Cette instruction décale les bits de la mémoire à droite de a, et place le résultat dans la mémoire a.

Le code de l'instruction prend la forme suivante :

1000	0	registre (a)

EQZ(i)

L'instruction EQZ implémente un saut conditionnel si le flag indique une égalité à zéro. Elle s'emploie de la façon suivante : EQZ a. Si le flag indique une égalité à zéro, cette instruction incrémente le pointeur d'instruction de la valeur contenue dans la mémoire a.
L'instruction supporte que sont premier paramètre soit directement un nombre, la saut s'effectue alors directement avec celui-ci.

Le code de l'instruction prend la forme suivante :

1001	0	registre (a)

Dans le cas ou le premier argument est directement un entier (instruction EQZi), le code de l'instruction prend cette forme :

1001	1	entier (sur 16 bits)

LTZ(i)

L'instruction LTZ implémente un saut conditionnel si le flag indique une infériorité à zéro. Elle s'emploie de la façon suivante : LTZ a. Si le flag indique une infériorité à zéro, cette instruction incrémente le pointeur d'instruction de la valeur contenue dans la mémoire a.
L'instruction supporte que sont premier paramètre soit directement un nombre, la saut s'effectue alors directement avec celui-ci.

Le code de l'instruction prend la forme suivante :

1010	0	registre (a)

Dans le cas ou le premier argument est directement un entier (instruction LTZi), le code de l'instruction prend cette forme :

1010	1	entier (sur 16 bits)

MTZ(i)

L'instruction MTZ implémente un saut conditionnel si le flag indique une supériorité à zéro. Elle s'emploie de la façon suivante : MTZ a. Si le flag indique une supériorité à zéro, cette instruction incrémente le pointeur d'instruction de la valeur contenue dans la mémoire a.
L'instruction supporte que sont premier paramètre soit directement un nombre, la saut s'effectue alors directement avec celui-ci.

Le code de l'instruction prend la forme suivante :

1011	0	registre (a)

Dans le cas ou le premier argument est directement un entier (instruction MTZi), le code de l'instruction prend cette forme :

1011	1	entier (sur 16 bits)

1.3 Les macros

De ces instructions de bases implémentées en matériel, on pourra écrire des macros pour des fonctions de plus haut niveau. Le multiplier peut par exemple se réaliser à coup d'additions et de sauts conditionnels (on suppose ici que les deux facteurs se trouvent dans les registres r2 et r3, le résultat sera stocké dans r1) :

```
1 XOR r1 r1 // Mise a zero du registre de retour
2 ADD 0 r3  // Initialise le flag qui va permettre de savoir si r3 = 0
3 EQZ 4     // Si r3 est egal a 3, on ne fait rien, on saute
4 ADD r2 r1 // Sinon on additionne et on decremente
5 SUB 1 r3
6 EQZ -2    // On teste l'egalite a 0, s'il n'est pas encore a 0 on
      recommence !
```

Conclusion