Université de Cergy-Pontoise

Architecture des Ordinateurs

Processeur 16 bits

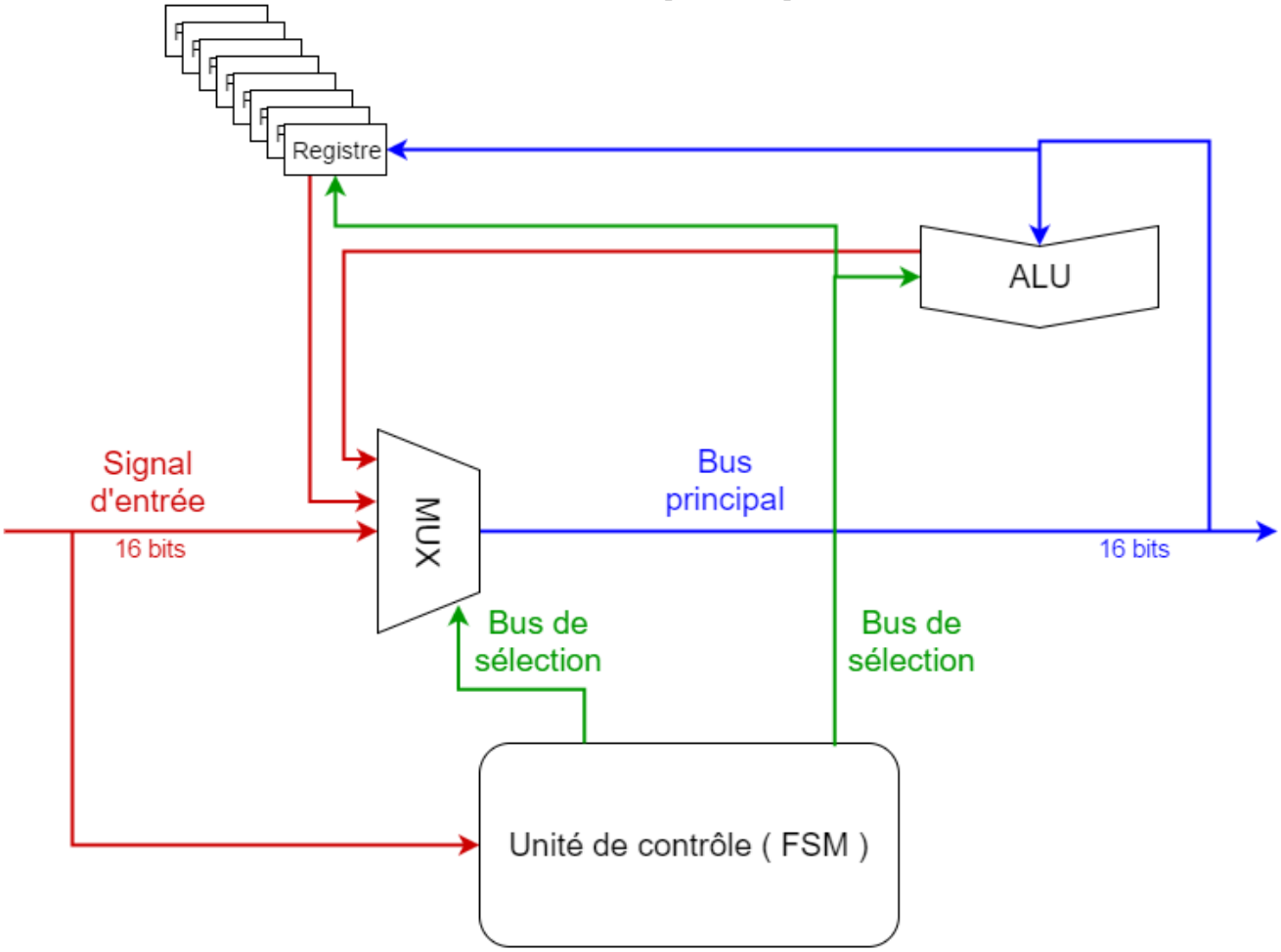
Lundi 04 Avril 2016

Auteurs :
Bastien LEPESANT
Lucas NICOSIA

Encadrant et Jury :
Laurent RODRIGUEZ
Alexandre MARCASTEL

1 Vue d'ensemble du processeur

FIGURE 1 – Schéma simplifié du processeur



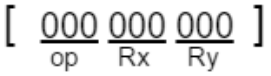
Le but de ce processeur est de pouvoir réaliser des opérations choisies (addition, soustraction, etc ..) sur des mots de 16 bits. Le schéma ci-dessus représente de manière simplifiée ses principaux composants ainsi que leurs connexions.

Composition Il est composée de deux parties principales : une ALU, qui réalise les opérations, et un banc de 8 registres où sont stockées données sur lesquelles seront effectuées ces opérations. L'utilisation sera donc la suivante : dans un premier temps, on écrit dans les registres les nombres/mots que l'on va traiter, puis on dit au processeur d'effectuer l'opération que l'on souhaite sur les registres choisis.

Fonctionnement L'utilisateur envoie un signal de16 bits au processeur contenant une instruction et/ou une valeur. Les 9 premiers bits de ce mot représentent l'instruction demandée, appelée "Code OP". Celle-ci est décomposée en 3 parties : les 3 premiers bits servent à indiquer l'opération à effectuer (op), les 3 suivants le premier registre à utiliser (Rx) et les 3 derniers le second (Ry). C'est donc une phrase du type "effectue une addition entre le mot contenu dans le registre 2 avec celui contenu dans le registre 5".

Le code OP est divisé comme suit :

FIGURE 2 – Division du code OP



Chaque opération correspond donc à un code bien précis :

Instruction	**code**
mv	000
mvi	001
add	010
sub	011
and	100
or	101
not	110
jump	111

Ces instructions correspondent à :
- mv (move) : copie le contenu du registre Rx dans le registre Ry
- mvi (move immediate) : écrit le prochain signal dans le registre Rx
- add : Opération d'addition (Voir ALU)
- sub : Opération de soustraction (Voir ALU)
- and : Opération logique "ET" (Voir ALU)
- or : Opération logique "OU" (Voir ALU)
- not : Opération logique inverse (Voir ALU)
- jump : Permet de "sauter" directement à une autre instruction

Chaque registre correspond lui aussi à un code bien précis :

Numéro de registre	**code**
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

2 ALU : Arithmetic and Logic Unit

L'ALU est le composant qui effectue les opérations sur les données circulant dans le processeur. A l'heure actuelle, nous y avons implémenté 5 opérations :

- Addition : nous pouvons additionner le contenu de deux registres au choix
- Soustraction : de même, nous pouvons soustraire le contenu de deux registres au choix
- Opération logique "ET" (AND) : nous pouvons réaliser un ET bit par bit entre deux registres au choix.
- Opération logique "OU" (OR) : nous pouvons réaliser un OU bit par bit entre deux registres au choix.
- Opération logique inverse (NOT) : nous pouvons réaliser un NON bit par bit sur un registre au choix.

3 Exemple de fonctionnement

Nous souhaitons additionner 3 et 7, nous allons donc envoyer les instructions suivantes :

- 001 000 000 0000000 : "Écris le prochain signal dans le registre R0"
- 000 000 000 0000011 : Le nombre 3 en binaire
- 001 001 000 0000000 : "Écris le prochain signal dans le registre R1"
- 000 000 000 0000111 : Le nombre 7 en binaire
- 010 000 001 0000000 : "Additionne le contenu du registre R0 à celui du registre R1"

4 Unité de contrôle

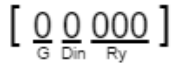
4.1 Données associées

Notre unité de contrôle est une machine à états finis (FSM = Final State Machine). C'est elle qui en fonction du Code OP va activer les bons composants du processeur. On peut dire qu'elle "exécute l'instruction". Elle réalise deux actions : choisir quel signal circule et activer les composants dans lequel le signal passera.

Son premier rôle est donc de choisir entre trois signaux (signal d'entrée, contenu d'un registre ou résultat de l'ALU) celui qui circulera dans le bus principal. Ceci se fait à l'aide d'un multiplexeur (MUX) qui prend comme entrée ces trois signaux et comme sélectionneur le code "set" envoyé par notre FSM.

Set :

FIGURE 3 – Division du "Set"

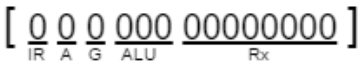


Il suffit de mettre à 1 le bit du signal qui sera choisi. G représente le résultat de l'ALU, Din le signal d'entrée et, une fois de plus, les 3 bits Rx permettent de choisir le registre désiré.

Son deuxième rôle est d'activer les composants du circuit principal (n'importe quel registre ou bien l'ALU). Nous utilisons le même principe que le signal "set" : un signal appelé "sel", relié à tous les composants du circuit principal et qui active ceux que nous voulons utiliser. Voici comment nous l'avons décomposé :

Sel :

FIGURE 4 – Division du "Sel"



Chaque composant du circuit reçoit donc ce signal et est activé lorsque le bit lui correspondant vaut 1 (sauf pour ALU qui est un cas particulier). IR, A et G sont trois registres dont nous n'avons pas encore parlé.

Champ IR (Instruction register) A chaque nouvelle instruction, le registre d'instruction contient les 9 bits de code OP du signal d'entrée. Lorsqu'il est activé, cela revient à demander l'instruction suivante.

Champ A Le fonctionnement de ce type de processeur pose problème si nous voulons faire une opération entre deux mots : nous avons un seul bus principal. L'ALU doit donc effectuer son opération entre le bus principal et autre chose. Nous sommes alors obligés d'ajouter un registre (A) juste avant. Elle effectuera l'opération entre le contenu de A et le bus principal quoi qu'il arrive. Avant chaque opération à deux opérandes, il faut donc d'abord stocker le contenu du premier registre dans A, envoyer le contenu du deuxième registre dans le bus principal et enfin activer l'ALU. Lorsque le bit correspondant à A vaut 1 le contenu du bus est y stocké.

Champ G Afin de connaître et de ne pas perdre le résultat d'une de ces opérations, nous devons l'écrire dans l'un des registres R. Or, à la fin d'une instruction, le contenu du bus principal disparaît. Nous sommes donc obligés d'ajouter un registre (G) juste après l'ALU dans lequel le résultat sera systématiquement stocké. Nous avons vu dans le signal "set" que l'on peut choisir de faire passer le contenu du registre G dans le bus principal. On peut donc sauvegarder le résultat d'une opération dans un des registres R. Lorsque le bit correspondant à G vaut 1 le contenu du bus sortant de l'ALU y est stocké.

Champ ALU C'est un cas particulier : il permet d'activer l'ALU sous différentes formes (addition, soustraction, etc ..), en fonction du code. Les trois bits "000" activent l'ALU sous les formes suivantes :

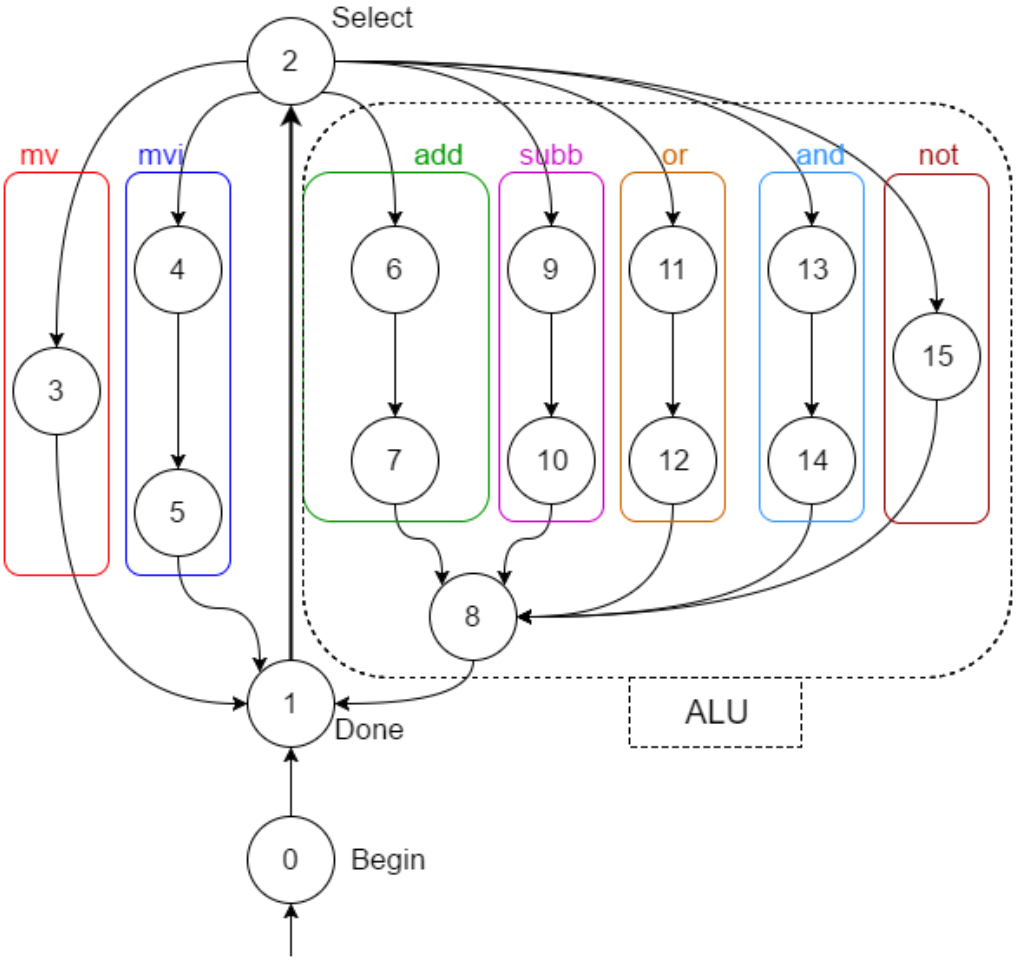
Champ ALU	**Opération demandée**
000	Soustraction
001	Addition
010	AND
011	OR
100	NOT

Champ Rx Il permet de choisir dans quel registre R nous allons sauvegarder le contenu du bus. Ici, nous n'utilisons pas 3 bits "000" afin de compter jusqu'à 8 mais 1 bit par registre "00000000" afin d'avoir une connexion par registre. Si le bit correspondant à un registre est activé (ex : 00100000 pour le registre 6), le bus y sera sauvegardé. Voici toute l'utilité des registres dans notre processeur : ils représentent notre espace de stockage.

4.2 États

Notre unité de contrôle est une machine à états finis, ce qui nous permet de réaliser une suite d'étapes. Voici une représentation des différents états, leurs liens et leurs descriptions :

FIGURE 5 – Schéma de la FSM



État	Description
0	État initial de la machine OU lors d'un reset
1	Demande l'instruction suivante
2	Sélectionne la suite d'états suivante en fonction de l'instruction
3	Copie le contenu de Rx dans Ry
4	Demande une nouvelle instruction
5	Écrit la nouvelle instruction dans Rx
6	Copie le contenu de Rx dans A
7	Additionne le contenu de A avec Ry et sauvegarde le résultat dans G
8	Copie le contenu de G dans Rx
9	Copie le contenu de Rx dans A
10	Soustrait le contenu de A avec Ry et sauvegarde le résultat dans G
11	Copie le contenu de Rx dans A
12	Réalise un OR logique entre chaque bit de A et Ry
13	Copie le contenu de Rx dans A
14	Réalise un AND logique entre chaque bit de A et Ry
15	Réalise un NOT pour chaque bit de Rx

5 Clock, Run & Reset

Le dernier point à propos du fonctionnement du processeur concerne trois signaux uniques : la Clock, le Run et Reset.

La Clock est l'horloge du processeur : à chaque nouvelle itération de l'horloge, une nouvelle instruction est demandée par l'unité de contrôle. Celle-ci est donc reliée à un bouton poussoir afin d'exécuter les instructions une à une lorsqu'on le désire.

Run est un signal qui doit obligatoirement être activé pour que le processeur fonctionne.

Enfin, le signal Reset permet comme son nom l'indique de recommencer l'exécution du programme, c'est à dire repartir à la première instruction.

6 ROM & Script

Notre processeur est composé d'une ROM, qui est une mémoire instantanée. On y écrit à la suite toutes les instructions que le processeur exécutera. Lorsqu'on lance le programme, il lit une nouvelle instruction contenue dans la ROM à chaque incrémention de l'horloge. Lorsqu'on reset, on recommence à la première instruction.

Cette ROM fonctionne via un fichier d'initialisation (en format "MIF") qui contient les valeurs des instructions en base décimale. Pour faciliter l'utilisation de notre processeur, nous avons conçu un langage de script et un compilateur associé qui construit un fichier "MIF" prêt à l'emploi. **Un script d'exemple est fourni en annexe**

Un script est composé de différents éléments associés à différentes opérations

Voici une liste des éléments :

Élément	**Construction**
Integer (Decimal)	nombre décimal entre 0 et 65535 inclus
Integer (Hexadecimal)	"0x" + nombre hexadécimal entre 0 et FFFF inclus
Registre	"R" + numéro du registre (entre 0 et 7 inclus)
Label ID	Le nom d'un label ID (composition de lettres minuscules et majuscules)

Et voici les opérations associées :

Opération	**Construction**
mv	Registre X '=' Registre Y
mvi	Registre X '=' Integer x
add	Registre X '+' Registre Y
sub	Registre X '-' Registre Y
and	Registre X '&' Registre Y
or	Registre X '|' Registre Y
not	'!' Registre X
jump	'label' Label ID => Permet de créer un label avec cet id à cette position 'goto' Label ID => Permet de se rendre au label avec cet id

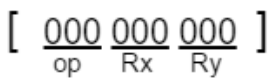
Manuel d'utilisation

Voici comment utiliser notre processeur. Il y a 2 différentes manières de lui envoyer des signaux :

- Manuellement : l'utilisateur écrit le signal d'entrée à l'aide des interrupteurs de la carte puis utilise le bouton d'horloge pour l'exécuter.
- En utilisant la ROM. On peut soit écrire à la main le fichier d'initialisation de la ROM soit utiliser un script.

Les instructions doivent être indiquées selon le code OP :

FIGURE 6 – Division du code OP



Et les opérations à effectuer selon le format suivant :

Instruction	**code**
mv	000
mvi	001
add	010
sub	011
and	100
or	101
not	110
jump	111

Voici l'assignement des différents boutons/visuels de la carte FPGA :

- Le switch S17 active Run lorsqu'il est à 1.
- Les switchs S0 à S15 permettent de configurer le signal d'entrée.
- Le switch S16 permet de choisir le mode d'utilisation : manuel s'il est à 1 ou ROM s'il est à 0.
- Les afficheurs HEX6 et HEX7 affichent le numéro de l'état actuel de la FSM.
- Les afficheurs HEX4 et HEX5 affichent le numéro de l'instruction actuelle.
- Les afficheurs HEX0 à HEX3 affichent le bus principal.
- Le bouton KEY0 incrémente Clock.
- Le bouton KEY1 permet de Reset.
- Les leds LEDR0 à LEDR7 s'allument lorsqu'on écrit dans un des registre de numéro correspondant.
- La led LEDR 16 s'allume lorsque Reset est utilisé.
- La led LEDR 17 s'allume lorsque Clock est utilisé.
- La led LEDR 15 s'allume lorsque Run est activé.
- La led LEDG 0 s'allume lorsque l'état Done est atteint.

Annexe

Voici un exemple de script :

```
1  //On definit le label "begin"
2  label begin;
3  //On mets la valeur 1 dans le registre R0
4  R0 = 1;
5  //On mets la valeur 2 dans le registre R1
6  R1 = 2;
7
8  //On additionne les registres R0 et R1 et on stocke le r sultat dans R0
9  R0 + R1;
10
11 //On mets la valeur 4 dans le registre R1
12 R1 = 4;
13 //On mets la valeur 5 dans le registre R2
14 R2 = 5;
15
16 //On additionne les registres R1 et R2 et on stocke le r sultat dans R1
17 R1 + R2;
18
19 //On soustrait les registres R1 et R0 et on stocke le r sultat dans R1
20 R1 - R0;
21
22 //On mets la valeur 107 dans le registre R2
23 R2 = 107;
24 //On mets la valeur 45 dans le registre R3
25 R3 = 45;
26
27 //On mets la valeur du registre R2 dans le registre R4
28 R4 = R2;
29 //On fait l'op ration logique "ET" entre les registres R4 et R3 et on stocke
       le r sultat dans R3
30 R4 & R3;
31
32 //On mets la valeur du registre R2 dans le registre R4
33 R4 = R2;
34 //On fait l'op ration logique "ET" entre les registres R4 et R3 et on stocke
       le r sultat dans R3
35 R4 | R3;
36
37 //On inverse logiquement le registre R3
38 !R3;
39
40 //On retourne au label "begin"
41 goto begin;
```