Université de Bordeaux
Licence STS

ARCHITECTURE DES ORDINATEURS

TD : **08**

# Chemin de données d'un processeur (partie 1)

## Introduction

On veut construire un processeur sur 4 bits, à l'image du processeur historique « 4004 » d'Intel. Le processeur à construire possède les caractéristiques suivantes :

- Il dispose de 4 registres, numérotés de 0 à 3, c'est-à-dire adressés sur 2 bits ;
- Le compteur ordinal est sur 3 bits, permettant l'existence de $2^3 = 8$ instructions dans la mémoire de programme (oui, c'est peu !) ;
- Chaque instruction est formatée sur un octet, à savoir 2 mots de 4 bits. Le tableau suivant recense les instructions constituant ce langage machine, ainsi que leur codage :

| Instruction | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| `halt` | 0 | 0 | — | | | | | |
| `li` $\texttt{val}_{(4)}$,$\texttt{rd}_{(2)}$ | 0 | 1 | $\texttt{rd}_{(2)}$ | | $\texttt{val}_{(4)}$ | | | |
| `add` $\texttt{rs}_{(2)}$,$\texttt{rd}_{(2)}$ | 1 | 0 | 0 | 0 | $\texttt{rs}_{(2)}$ | | $\texttt{rd}_{(2)}$ | |
| `dec` $\texttt{rd}_{(2)}$ | 1 | 0 | 1 | 1 | — | | $\texttt{rd}_{(2)}$ | |
| `jmp` $\texttt{addr}_{(3)}$ | 1 | 1 | — | | 0 | $\texttt{addr}_{(3)}$ | | |
| `jz` $\texttt{rs}_{(2)}$,$\texttt{addr}_{(3)}$ | 1 | 1 | $\texttt{rs}_{(2)}$ | | 1 | $\texttt{addr}_{(3)}$ | | |

## Exercice 1 : Codage d'un programme

### Question 1

Quelle est la fonction du programme codé ci-dessous ?

| Adresse | Code |
|---|---|
| 0 | 6 $x$ |
| 1 | 5 $y$ |
| 2 | 4 0 |
| 3 | E F |
| 4 | 8 4 |
| 5 | B 2 |
| 6 | C 3 |
| 7 | 0 0 |

### Question 2

Proposez le chemin de données de l'ordinateur susceptible d'exécuter les instructions de ce langage. Pour cela, déterminez :

- les unités fonctionnelles nécessaires ;
- leurs liaisons.

## **Exercice 2 :** Décodage des instructions

Pour alimenter les différentes unités fonctionnelles du processeur, il est nécessaire de décoder les instructions afin d'en extraire les données pertinentes et de les fournir aux autres unités fonctionnelles selon un format uniforme. Soit $\texttt{I} = i_7 i_6 i_5 i_4 i_3 i_2 i_1 i_0$ un octet d'instruction.

### Question 1

Donnez les fonctions logiques permettant d'obtenir la valeur de l'adresse de destination $\texttt{addr} = a_2 a_1 a_0$ sur trois bits, en fonction de la valeur des bits de l'instruction `I`, pour les instructions qui l'utilisent.

On n'a pas à se soucier de la valeur de `addr` dans le cas des autres instructions, puisqu'alors on ne l'utilisera pas, quelle qu'elle soit. Il est donc inutile de la mettre à une valeur particulière (`0` ou autre), ce qui complexifierait inutilement les fonctions.

### Question 2

Soit `rs` (pour « *register number (source)* ») la fonction donnant la valeur de `rs` contenue dans chacune des instructions qui la possèdent. Donnez les fonctions logiques des valeurs $\texttt{rs}_1$ et $\texttt{rs}_0$, en fonction des bits de `I`.

Notez que, si les instructions `add` et `jz` possèdent un champ de registre source `rs`, celui-ci n'est pas placé au même endroit dans chacune de ces instructions. Comme précédemment, il est inutile de se soucier de la valeur de `rs` pour les instructions qui ne l'utilisent pas.

### Question 3

Câblez la fonction `WE` (« `write enable` »), qui renvoie 1 lorsque l'instruction doit effectivement modifier la valeur du registre de numéro `rd` dans la banque de registres. Cette fonction doit donc identifier les instructions qui possèdent un champ `rd`.

### Question 4

Câblez la fonction `JF` (« *jump function* »), qui indique les modalités de passage à l'instruction suivante :

| Valeur | Signification |
|---|---|
| 0 | Même instruction (cas de `halt`) |
| 1 | Passage à l'instruction suivante |
| 2 | Branchement inconditionnel |
| 3 | Branchement conditionnel |

Pour coder cette fonction, considérez chacun des deux bits de la valeur à retourner comme une fonction indépendante, calculée à partir des bits 7, 6 et 3 de l'instruction. Ces deux bits pourront être agrégés au sein d'un bus.

## **Exercice 3 :** Horloges

Pour faire fonctionner un ordinateur, une unique base de temps n'est pas suffisante. En effet, considérons par exemple le cas de l'addition du contenu d'un registre avec celui d'un autre. Si l'on réécrit le contenu du registre de destination avec la valeur issue de l'UAL, au même moment qu'on lit la valeur de ce même registre pour l'envoyer dans l'UAL, alors on se trouve face à une boucle de calcul qui peut donner des résultats imprévisibles !

Il faut donc isoler les phases de calcul et de réécriture dans les registres. On procède donc en deux temps :

1. Dans un premier temps, une fois les valeurs d'entrée stabilisées à partir des registres du processeur, et donc le résultat du calcul sur ces valeurs également stabilisé, on stocke le résultat dans un registre intermédiaire ;

2. Dans un deuxième temps, les registres définitifs sont mis à jour grâce aux valeurs stockées dans les registres intermédiaires.

Ces registres intermédiaires jouent le rôle de « *verrous* » (en anglais : « *latch* ») pour « casser » les boucles de calcul.

## Question 1

Quelles sont les boucles de calcul présentes lors de l'exécution d'une instruction, et qui doivent être « cassées » ?

## Question 2

Câblez un circuit d'horloge, alimenté par une entrée d'horloge `Clk`, qui fournit deux signaux d'horloge de sortie, `Clk1` et `Clk2`, tels que, sur quatre périodes de temps consécutives, on ait `Clk1` et `Clk2` alternativement à `1` une fois sur deux, lorsque l'entrée d'horloge `Clk` est à `1`.
*N.B. : on ne veut pas de dérive de l'horloge.*