Université de Bordeaux
Licence STS

ARCHITECTURE DES ORDINATEURS

TP : **09**

Construction du chemin de données d'un processeur (partie 2)

L'outil que vous allez utiliser est le simulateur de circuits SimcirJS, dont une copie adaptée au cours est installée sur :

`http://dept-info.labri.fr/ENSEIGNEMENT/archi/circuits/blank.html`

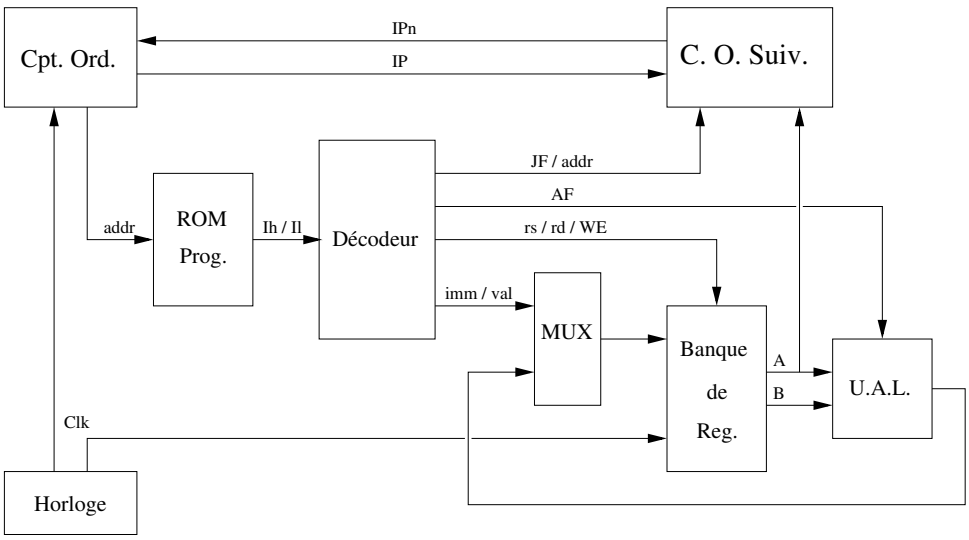
(ce lien est disponible depuis le site du cours).

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	$\texttt{addr}_{(3)}$		

L'architecture de l'ordinateur susceptible d'exécuter les instructions de ce langage est la suivante :



Exercice 1 : Passage à l'instruction suivante

En règle générale, la prochaine instruction à exécuter est l'instruction suivante, sauf lorsqu'un branchement doit être pris. Dans le cas du processeur décrit plus haut, il peut s'agir soit d'un branchement inconditionnel, soit d'un branchement si la valeur du registre source est nulle. Cette information est fournie par le circuit de décodage, à travers son signal `JF` (« *jump function* »).

- Sur le plan de travail, placez quatre `RotaryEncoderBus`, qui représentent respectivement la valeur courante du compteur ordinal (« `IP` », pour « *instruction pointer* »), l'adresse de destination du branchement éventuel `addr`, le contenu du registre éventuellement à tester `val`, et la fonction à appliquer `JF` (voir tableau ci-dessus).
- Câblez la fonction permettant de calculer la valeur correspondant à l'incrémentation de `IP`. Notez que comme les numéros d'instruction vont de 0 à 7 seulement, la valeur suivant 7 doit être 0.
- Câblez la fonction valant 1 lorsque la valeur `val` est égale à 0. Complétez-la pour que la fonction ne vale 1 que si également `JF` vaut 3.
- En vous appuyant sur la fonction précédente, câblez la fonction qui renvoie la valeur du nouveau compteur ordinal `IPn`. On peut voir cette fonction comme un multiplexeur qui, selon la valeur de `JF` (de 0 à 3), renvoie la nouvelle valeur de compteur ordinal correspondante. Le cas de l'instruction `jz` doit alors être traité comme un cas particulier : si `JF` vaut 3 et que `val` vaut zéro, alors la valeur du signal `JF` doit être changée en 2 en amont du multiplexeur, pour déclencher un branchement inconditionnel. Dans tous les autres cas, la valeur du signal restera identique. De fait, la valeur 3 non modifiée doit conduire, au niveau du multiplexeur, à ce que l'on passe à l'instruction suivante (branchement conditionnel non pris).

Toutes les fonctionnalités décrites ci-dessus sont encapsulées au sein du composant `CpuNextIp`, que vous pourrez utiliser par la suite.

Exercice 2 : Banque de registres

Une banque de registres est une catégorie de mémoire dite « *multi-ports* ». En effet, au cours d'une instruction, on doit pouvoir être capable de lire en même temps deux registres à la fois, identifiés par les signaux d'entrée $\texttt{r0}_{(2)}$ et $\texttt{r1}_{(2)}$, et éventuellement d'écrire dans un registre identifié par le signal $\texttt{rw}_{(2)}$ (pas au moment exact où on lira, mais un peu plus tard).

- Placez quatre `RegBus-E` au milieu de votre plan de travail, l'un au dessus de l'autre. Pour voir ce que contient chaque `RegBus-E`, vous pouvez les relier à des `4bit7segBus` (il sera sans doute utile de les supprimer plus tard, pour ne pas surcharger votre plan de travail).
- Pour pouvoir écrire dans le bon registre, utilisez un `Demux` permettant de router le signal d'écriture vers l'entrée `E` du bon `RegBus-E`. Le signal d'écriture sera pour le moment activé par un `PushOn`, et l'adresse de sélection du registre de destination par un `RotaryEncoderBus`. Utilisez un autre `RotaryEncoderBus` pour sélectionner la valeur à écrire, qui devra être propagée sur chaque entrée de registre.
- Pour pouvoir lire à partir d'un registre donné, sur un premier port, utilisez un `Muxbus` collectant les sorties de chacun des registres, et piloté par un `RotaryEncoderBus`.
- Pour pouvoir lire à partir d'un autre registre (ou le même), sur un second port, utilisez un second `Muxbus` collectant lui aussi les sorties de chacun des registres, et piloté par un second `RotaryEncoderBus`.

Toutes les fonctionnalités décrites ci-dessus sont encapsulées au sein du composant `CpuRegBank`, que vous pourrez utiliser par la suite.

Exercice 3 : Mémoire ROM

Une mémoire ROM peut être vue comme une fonction logique qui, en fonction de la valeur d'une adresse (donnée d'entrée), renvoie une valeur fixe, codée « en dur ». Chaque bit du mot renvoyé en

sortie du circuit mémoire peut donc être vu comme une fonction logique dépendant des variables codant chacun des bits de l'adresse. Plus simplement, on peut considérer une mémoire ROM comme un grand multiplexeur qui, pour chacune des valeurs d'adresse, renvoie une donnée fixe correspondant à l'une des voies d'entrée du multiplexeur.

Question 1

À base de `RotaryEncoderBus` et de `MuxBus`, construisez une mémoire ROM à quatre mots de quatre bits, contenant les quatre valeurs « `6` », « `A` », « `3` » et « `8` ».

Question 2

Étendez cette mémoire pour en faire une mémoire ROM à huit mots de quatre bits.

Le circuit `CpuRomMul`, que vous pourrez utiliser par la suite, code en ROM le programme vu au TD précédent. Deux entrées `d0l` et `d1l` permettent de positionner, par exemple avec des `RotaryEncoderBus`, les deux valeurs des deux premiers demi-octets de poids faible, correspondant aux valeurs x et y données dans le programme.

Exercice 4 : Unité arithmétique et logique

À quels numéros de fonction doivent correspondre les opérations définies dans le langage machine défini plus haut ?

Le circuit `CpuAlu`, que vous pourrez utiliser par la suite, est une unité arithmétique et logique compatible avec ce langage machine.

Exercice 5 : Ordinateur complet

Le composant `Cpu` de la barre d'outils est un ordinateur complet, mettant en œuvre le programme présenté ci-dessus. Double-cliquez sur ce composant pour en exposer le contenu.

Vous y retrouverez le schéma du processeur défini lors du TD précédent, qui intègre les différents composants déjà réalisés. Un certain nombre de « *latches* » ont été ajoutés, pour « casser » les boucles de calcul.

Plusieurs afficheurs permettent de visualiser l'état courant de la machine :

— « `IP` » affiche la valeur courante du registre de compteur ordinal, c'est-à-dire l'adresse de l'instruction courante ;

— « `IPn` » affiche l'adresse de l'instruction suivante ;

— « `x` » et « `y` » affichent les valeurs correspondant respectivement aux demi-octets de poids faible des adresses `0` et `1` de la ROM (qui n'en est donc pas vraiment une ici) ;

— « `Instr. H` » et « `Instr. L` » affichent les valeurs des deux demi-octets de l'instruction qui se trouvent en ROM à l'adresse donnée par le compteur ordinal `IP`. Ainsi, si `IP` est à `0`, tourner la molette « `x` » fait changer la valeur affichée par « `Instr. L` », puisque l'instruction à l'adresse `0` dans la ROM est l'instruction « `6`x » ;

— « `Val. Reg. Ar1` » affiche la valeur du registre d'adresse « `Ar1` » actuellement stockée dans la banque de registres. Du fait du codage de l'instruction `halt`, c'est le contenu du registre `r0` qui sera affiché à la fin du programme. On aura ainsi le résultat de la multiplication.

Pour faire fonctionner cet ordinateur :

1. Au niveau du panneau de configuration en bas à gauche :
   (a) S'il ne l'est pas déjà, désactivez le `Toggle` « `Run` », afin de découpler l'ordinateur de l'horloge ;

   (b) Appuyez sur le `PushOn` « `Init` », pour réinitialiser l'horloge (qui doit toujours démarrer avec un signal `Clk1` avant `Clk2`) et remettre à 0 le compteur ordinal (situé en haut à gauche du schéma) ;
2. Positionnez les deux molettes « `x` » et « `y` » aux deux valeurs que vous sohaitez faire multiplier (attention aux débordements sur un demi-octet !). Ces valeurs sont en fait insérées dans la ROM du programme ;
3. Pour choisir le mode d'exécution pas-à-pas, désactivez le bouton « `Clk` ». Dans ce cas, vous pourrez simuler chaque top d'horloge en appuyant sur le bouton « `Step` ».
4. À un moment où l'horloge est à la valeur 0, activez le bouton « `Run` », ce qui lancera les calculs. À tout moment, lorsque le signal d'horloge est à 0, vous pourrez désactiver ce bouton pour figer l'état de la machine, et le réactiver par la suite, toujours lorsque le signal d'horloge est à 0. Vous pourrez aussi agir sur le bouton « `Clk` » pour activer ou désactiver le mode pas-à-pas.

Une fois l'ordinateur pris en main :

— Lancez le programme, d'abord en mode pas à pas, puis en couplant l'ordinateur à l'oscillateur. En fonction des valeurs `x` et `x` que vous aurez fournies en ROM, observez l'exécution des instructions. Vous pouvez, à tout moment, arrêter l'exécution de l'horloge, et « instrumenter » le circuit avec des `4bit7segBus` et/ou des `LED` pour observer la valeur des différents signaux.

— Notez comment certains afficheurs peuvent, très rapidement, changer plusieurs fois de valeur avant stabilisation, à mesure que l'information se propage au sein des circuits. Quels valeurs cela concerne-t-il principalement ? Pourquoi ?