



**POLYTECHNIQUE  
MONTRÉAL**

# INF1600

## Architecture des micro-ordinateurs

TP5

Groupe 02 (B2)

Soumis par:


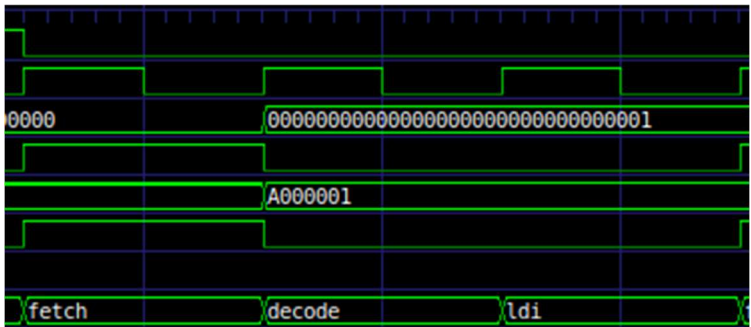

Charles de Lafontaine - **2076524**  
Geneviève Pelletier-Mc Duff - **2088742**

Le 25 avril 2021

## Barème de correction

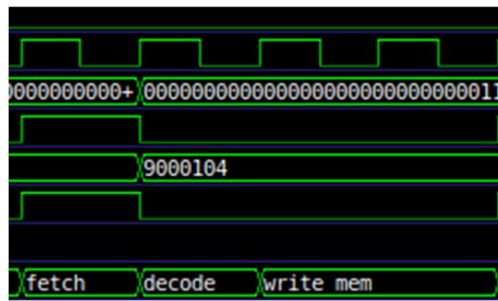
<b>TP 5</b>		<b>/4,00</b>
Q1	/1,00	
Q2	/1,00	
Q3	/1,00	
Q4	/1,00	
Q5 (bonus)	/1,00	

**Q1 : Exécutez le programme fourni et complétez la table suivante :**

INSTRUCTION/TYPE D'INSTRUCTION	CPI
OP_ALU	<p><b>CPI = 3</b></p>  <p><b>Figure 1.</b> Capture d'écran de la simulation sur <i>GTKWave</i> permettant de montrer le nombre de cycles d'horloge pour effectuer l'instruction <i>op_alu</i>.</p>
LDI	<p><b>CPI = 3</b></p>  <p><b>Figure 2.</b> Capture d'écran de la simulation sur <i>GTKWave</i> permettant de montrer le nombre de cycles d'horloge pour effectuer l'instruction <i>ldi</i>.</p>
READ_MEM	<p><b>CPI = 4</b></p>  <p><b>Figure 3.</b> Capture d'écran de la simulation sur <i>GTKWave</i> permettant de montrer le nombre de cycles d'horloge pour effectuer l'instruction <i>read_mem</i>.</p>

**WRITE\_MEM**

**CPI = 4**



**Figure 4.** Capture d’écran de la simulation sur *GTKWave* permettant de montrer le nombre de cycles d’horloge pour effectuer l’instruction *write\_mem*.

**Q2 : Le fichier `simple_risc_cu.vhd` contient l'unité de contrôle du processeur. Proposez une modification de l'unité de contrôle qui garantisse un CPI de 2 pour toute instruction de type `op_alu`. Décrivez dans votre rapport la modification apportée.**

Afin de décrémente le CPI de l'instruction `op_alu` de 1, soit pour passer d'un CPI de 3 à un CPI de 2, nous avons modifié les lignes 60, 106 et 109 du fichier `simple_risc_cu.vhd` (veuillez vous référer au fichier `2088742-2076524-tp5q2.vhd`) (voir le **Tableau I**).

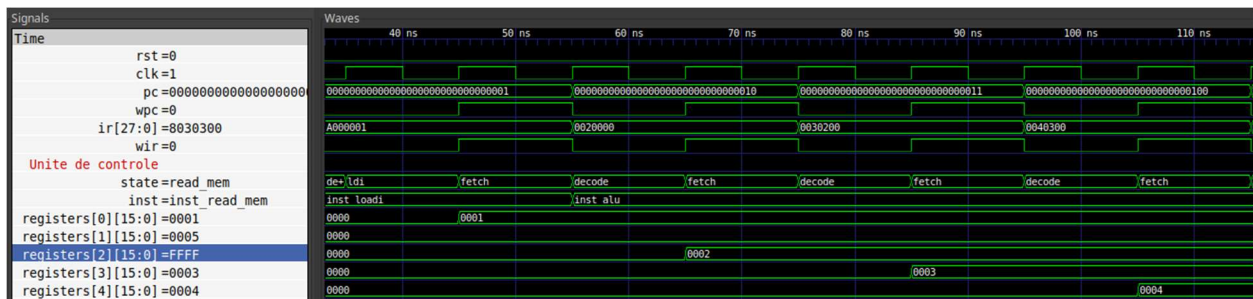
La modification apportée à la ligne 60 nous permet de mettre à jour l'état du programme pour qu'il puisse rechercher la prochaine instruction dès qu'il est à `inst_alu`. Par ailleurs, la modification apportée à la ligne 106 permet de prendre en paramètre le signal de contrôle `s_op_alu` dans le `process` au lieu d'utiliser l'état (`state`).

Puis, nous avons modifié la ligne 109 afin de prendre en compte le changement du signal `s_op_ual` au lieu de l'état `op_alu`. Ainsi, lorsque `s_op_ual` est égal à 1, nous modifions les signaux de contrôle `choixSource` à 0, `wreg` et `wFlag` à 1, afin de pouvoir écrire directement dans le registre de destination le résultat de l'opération arithmétique.

**Tableau I.** Comparaison des modifications apportées aux lignes 60, 106 et 109 du fichier `simple_risc_cu.vhd` pour permettre de réduire le CPI à 2 de l'instruction `op_alu`.

#	Lignes d'origine	Lignes modifiées
60	<code>60            when inst_alu       =&gt; state &lt;= op_alu;</code>	<code>60            when inst_alu       =&gt; state &lt;= fetch;</code>
106	<code>103       process( state, rmem_confirmed ) is</code>	<code>106       process( s_op_ual, state, rmem_confirmed ) is</code>
109	<code>106       if( state = op_alu ) then</code>	<code>109       if( s_op_ual = '1' ) then</code>

Ces modifications permettent ainsi de réduire le CPI de l'instruction `op_alu` à 2 comme il est possible de le constater à la **Figure 5** (`fetch`, `decode`, sans `op_alu`).



**Figure 5.** Capture d'écran de la simulation sur *GTKWave* qui montre que le nombre de cycles d'horloge de l'instruction `op_alu` est diminué de un suite à aux modifications apportées.

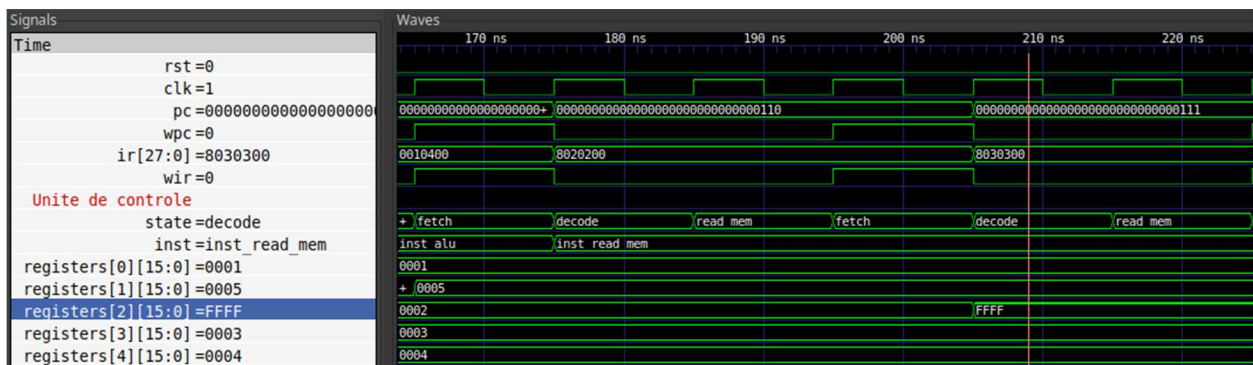
**Q3 : Proposez une modification de l'unité de contrôle qui réduise le CPI d'une instruction *read\_mem* de 1. Décrivez dans votre rapport la modification apportée.**

Le CPI d'origine de l'instruction *read\_mem* était de 4, l'objectif était ainsi d'obtenir un CPI de 3 suite à nos modifications. Pour ce faire, nous avons modifié les lignes 68 à 71 du fichier *simple\_risc\_cu.vhd* (veuillez vous référer au fichier *2088742-2076524-tp5q3.vhd*) (voir le **Tableau II**).

Avant les changements, lorsque le processeur était à l'état *read\_mem*, il effectuait un cycle d'horloge supplémentaire pour attendre que le signal *rmem\_confirmed* soit égal à 1. Ainsi, lors de la phase d'exécution de l'instruction *read\_mem*, il fallait nécessairement 2 cycles d'horloge pour l'effectuer. Afin de réduire le CPI de 1, nous avons retiré la condition qui confirmait la lecture de la mémoire (*rmem\_confirmed*). Ainsi, dans notre nouveau programme, nous avons une seule étape pour exécuter, soit lire en mémoire, puis nous passons directement au *fetch* de la prochaine instruction. Il est possible de constater le succès de nos modifications à la **Figure 6**, puisque l'instruction *read\_mem* est d'une durée de seulement 3 cycles d'horloges. De plus, nous remarquons, à la **Figure 6**, que l'écriture dans le registre de destination R[r2] s'est bel et bien déroulée adéquatement, puisque *0xFFFF* est maintenant indiqué.

**Tableau II.** Comparaison des modifications apportées aux lignes 68 à 71 du fichier *simple\_risc\_cu.vhd* pour permettre de réduire le CPI à 3 de l'instruction *read\_mem*.

#	Lignes d'origine	Lignes modifiées
68-71	<pre> 68      when read_mem =&gt; 69          if( rmem_confirmed = '1' ) then 70              state &lt;= fetch; 71          end if; </pre>	<pre> 68      when read_mem =&gt; state &lt;= fetch; </pre>



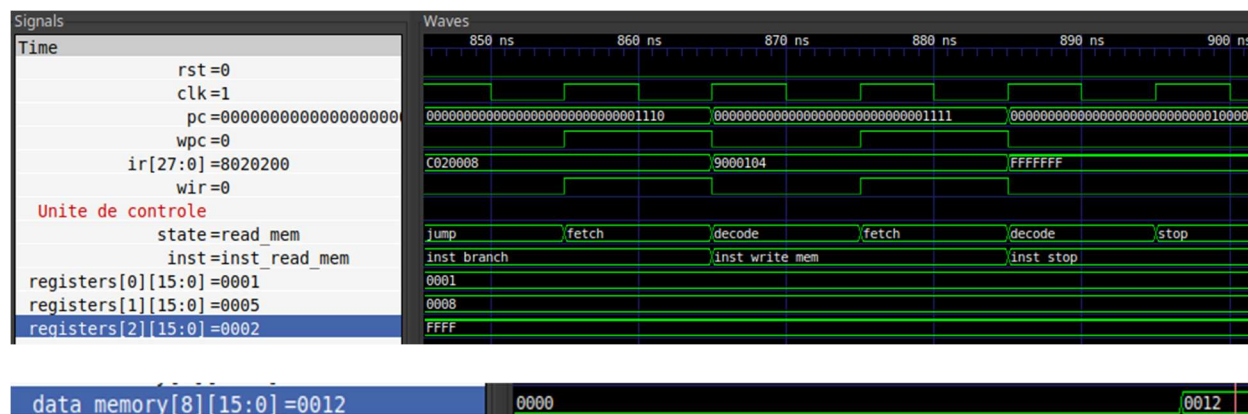
**Figure 6.** Capture d'écran de la simulation sur *GTKWave* qui montre que le nombre de cycles d'horloge de l'instruction *read\_mem* est diminué de un suite à aux modifications apportées.

**Q4 : Proposez une modification de l'unité de contrôle qui garantisse un CPI de 2 pour l'instruction *write\_mem*. Seule la moitié des points de cette question sont accordés si vous réussissez à réduire le CPI de 1 par rapport au chiffre de la question Q1.**

Afin d'obtenir un CPI de 2 pour l'instruction *write\_mem*, nous avons éliminé les deux cycles liés à l'exécution de l'instruction. Pour ce faire, dans la section *decode* du programme, lorsque nous avons une instruction de type *write\_mem*, nous avons modifié la ligne 62 afin d'aller directement à la prochaine instruction *fetch*, au lieu d'atteindre l'état *write\_mem* (veuillez vous référer au fichier *2088742-2076524-tp5q4.vhd*) (voir **Tableau III**). De plus, nous avons retiré les lignes 72 à 75 du programme puisque nous avons éliminé l'état *write\_mem* en allant directement au *fetch*. En exécutant la simulation, nous avons confirmé que cette modification au programme nous permettait bel et bien d'écrire en mémoire des données en deux cycles (voir **Figure 7**). En effet, nous obtenons la valeur 12 dans la case 8 de la mémoire des données suite à l'instruction *write\_mem*, comme il était attendu. Ainsi, nous n'avons pas changé le comportement de la simulation et simplement réduit le temps d'exécution total.

**Tableau III.** Comparaison des modifications apportées aux lignes 62, 72 à 75 du fichier *simple\_risc\_cu.vhd* pour permettre de réduire le CPI à 2 de l'instruction *write\_mem*.

#	Ligne d'origine	Ligne modifiée
62	<pre> 62      when inst_write_mem =&gt; state &lt;= write_mem; 72      when write_mem =&gt; 73          if( wmem_confirmed = '1' ) then 74              state &lt;= fetch; 75          end if; </pre>	<pre> 62      when inst_write_mem =&gt; state &lt;= fetch; </pre>
72-75		



**Figures 7 et 8.** Capture d'écran de la simulation sur *GTKWave* confirmant l'exécution en deux cycles d'horloge de l'instruction *write\_mem*. La valeur résultante 12 est bel et bien inscrite en mémoire des données à la case 8 suite à cette dernière.

**Q5 (bonus) : Peut-on réduire modifier l'unité de contrôle de sorte que *read\_mem* ait un CPI de 2. Si vous répondez par l'affirmative, expliquez comment faire. Dans le cas contraire, expliquer pourquoi ce n'est pas possible.**

Il est impossible de réduire le CPI de 2 pour l'instruction *read\_mem*. En effet, si nous réduisons le nombre de cycles d'horloge, nous introduisons la notion de pipelinage puisque nous allons faire deux cycles pour les étapes de *fetch* et *decode*. Les deux autres cycles de lecture en mémoire s'exécutent simultanément aux étapes de *fetch* et *decode* de l'instruction suivante. Ainsi, lorsqu'il est temps d'inscrire la valeur contenue dans la mémoire des données au registre de destination, la simulation est à l'étape du *decode* de l'instruction suivante. Cela a de graves conséquences, puisque le *decode* va permettre d'inscrire une nouvelle adresse d'instruction dans le registre IR, ce qui va modifier le registre de destination. Donc, l'instruction va inscrire le résultat du *read\_mem* au registre de l'instruction suivante. Plus précisément, le **Tableau IV** présente les RTN abstraits initiaux (à gauche du tableau) et ceux réellement exécutés (à droite du tableau). Ce comportement est visible lors de la simulation (voir **Figure 9**), étant donné que toutes les instructions *read\_mem* écrivent leurs données lues dans les registres de destination de l'instruction suivante. Le **Tableau V** présente les valeurs inscrites dans les registres R[2] à R[5] dans la version originale du programme et avec la réduction du CPI à 2 à la suite des quatre instructions *read\_mem*.

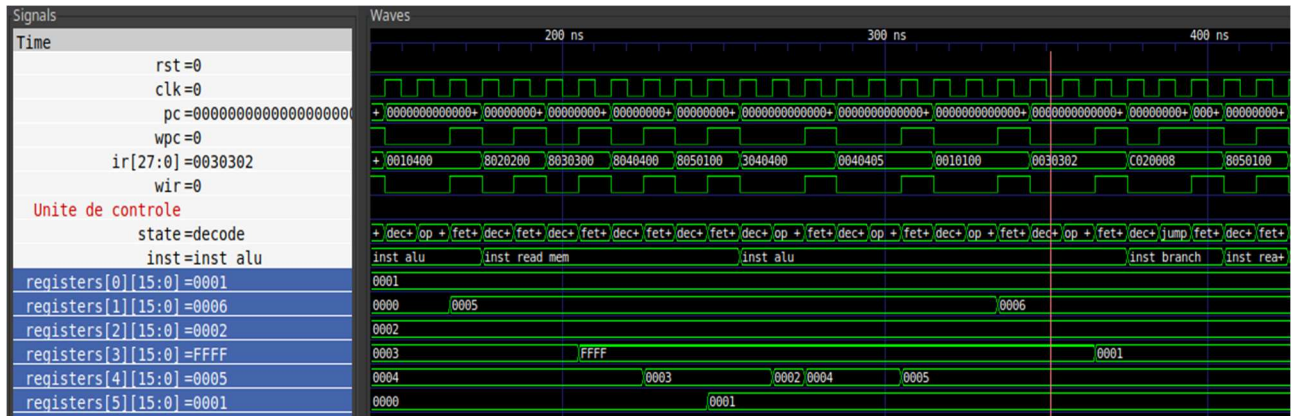
**Tableau IV.** Comparaison des RTN abstraits inscrits dans le fichier *simple\_risc\_programs.vhd* (à gauche) versus ce qui est réellement exécuté en réduisant le CPI à 2 pour l'instruction *read\_mem*.

Instructions initiales (en RTN abstrait)	Instructions exécutées en changeant le CPI à deux de l'instruction <i>read_mem</i>
r2 <- M[r2] // addr x05 r3 <- M[r3] // addr x06 r4 <- M[r4] // addr x07 r5 <- M[r1] // addr x08 r4 <- r4 << 1 // addr x09	r3 <- M[r2] // addr x05 r4 <- M[r3] // addr x06 r5 <- M[r4] // addr x07 r4 <- M[r1] // addr x08 r4 <- r4 << 1 // addr x09



**Tableau V.** Comparaisons du contenu des registres R[2] à R[5] suite aux quatre instructions *read\_mem* dans la version originale du programme versus suite aux modifications permettant la réduction du CPI à 2 pour l’instruction *read\_mem*.

Registres	Avant les modifications	Après les modifications	
	Après l’instruction à l’adresse 0x08	Après l’instruction à l’adresse 0x06	Après l’instruction à l’adresse 0x08
R[2]	0xFFFF	0x0002	
R[3]	0x0003	0xFFFF	
R[4]	0x0001	0x0003	0x 0002
R[5]	0x0002	0x0001	



**Figure 9.** Capture d’écran de la simulation sur *GTKWave* suite aux modifications pour réduire le CPI à 2 de l’instruction *read\_mem*.