POLYTECHNIQUE
MONTRÉAL

INF1600
Architecture des micro-ordinateurs

Laboratoire 5

Soumis par:
Ioana Daria Danciu - 2081744
Alexandre Gélinas - 2083465

Groupe de Laboratoire:
02

Le 25 avril 2020

1.2 Barème

TP5		**/4,00**
Q1	/1,00	
Q2	/1,00	
Q3	/1,00	
Q4	/1,00	
Q5 (bonus)	/1,00	

Q1:

INSTRUCTION/TYPE D'INSTRUCTION	CPI
OP_ALU	3
LDI	3
READ_MEM	4
WRITE_MEM	4

Q2:

Nous avons changé l'instruction suivant le decode pour fetch pour réduire le CPI à 2 (L3)

```
when decode =>
   case inst is
      when inst_alu       => state <= fetch;
      when inst_read_mem  => state <= read_mem;
      when inst_write_mem => state <= write_mem;
      when inst_loadi     => state <= ldi;
      when inst_branch    => state <= jump;
      when inst_stop      => state <= stop;
      when others         => state <= stop;
   end case;
```

Nous avons changé le process pour pouvoir faire une réaction lorsque notre s_op_ual change. De plus, nous avons modifié la condition pour faire les instructions de OP_ALU pour qu'il teste si la valeur du signal à changé pour '1'.

```
process( state, s_op_ual, rmem_confirmed ) is
begin
   wFLAG <= '0';
   if( s_op_ual = '1' ) then
      choixSource <= 0;
      wreg        <= '1';
      wFLAG       <= '1';
   elsif( rmem_confirmed = '1' ) then
      choixSource <= 1;
      wreg        <= '1';
   elsif( state = ldi ) then
      choixSource <= 2;
      wreg        <= '1';
   else
      choixSource <= 3;
      wreg        <= '0';
   end if;
end process;
```

Q3:

Afin de réduire le CPI de read_mem de 1 cycle d'horloge, nous avons retiré la condition ci-dessous, qui faisait en sorte que le programme passait à l'instruction fetch que lorsque l'instruction read_mem a été complétée. Ainsi, lorsqu'on est en train de lire dans la mémoire, on va également chercher la prochaine instruction, ce qui permet d'éviter de passer par un registre intermédiaire et de réduire le CPI de 1.

```
when read_mem =>
    --if( rmem_confirmed = '1' ) then
        state <= fetch;
    --end if;
```

Q4:

Afin de réduire le CPI de write_mem à deux cycles d'horloge, nous avons remplacé le changement d'état de write_mem à fetch.

```
when decode =>
    case inst is
        when inst_alu       => state <= fetch;
        when inst_read_mem  => state <= read_mem;
        when inst_write_mem => state <= fetch;
        when inst_loadi     => state <= ldi;
        when inst_branch    => state <= jump;
        when inst_stop      => state <= stop;
        when others         => state <= stop;
    end case;
when read_mem =>
```

Ensuite, nous avons ajouté un « elsif » , qui modifiait le moment où nous allions chercher la prochaine instruction. En effet, lorsque nous sommes dans un front descendant de l'horloge, lors du dernier état d'une instruction, nous rendons le flag wIR égal à 1, ce qui permet de régénérer le IR au début du « fetch » suivant.

```
                state <= start;
        end case;
    elsif clk'event then
        case state is
            when decode =>
                case inst is
                    when inst_alu       => wIR <= '1';
                    when inst_write_mem => wIR <= '1';
                    when inst_loadi     => wIR <= '1';
                    when others         => wIR <= '0';
                end case;
            when jump =>
                wIR <= '1';
            when read_mem =>
                wIR <= '1';
            when stop =>
                wIR <= '1';
            when others =>
                wIR <= '0';
        end case;
    end if;
end process;
```

Finalement, nous avons modifié le « elsif » ci-dessous afin qu'il s'effectue à chaque événement d'horloge, et non que lors des fronts montants. Ainsi, nous pouvons directement commencer à lire ou à écrire dans la mémoire, à partir d'un front descendant.

```
-- Ne pas modifier de preferenc
process (clk)
begin
    if rst = '1' then
        wmem <= '0';
        rmem <= '0';
    elsif clk'event then
        wmem <= s_op_wmem;
        rmem <= s_op_rmem;
    end if;
```

De plus, nous avons dû corriger l'instruction du jump pour que celle-ci puisse se faire au même rythme que la lecture des instructions. De ce fait, nous avons commencé le doBranch avant l'instruction du jump pour qu'il ait le temps de changer l'adresse de lecture du PC.

```
process( state, Z, N )is
begin
    if( state = fetch )then
        wPC      <= '1';
        doBranch <= '0';
    elsif( state = decode and inst = inst_branch ) then
        if ( jmp = br ) or              -- branchement sans condition
           ( jmp = brz  and Z = '1' ) or -- si = 0
           ( jmp = brnz and Z = '0' ) or -- si /= 0
           ( jmp = brs  and N = '1' ) or -- si < 0
           ( jmp = brns and N = '0' )    -- si >= 0
             then
            wPC      <= '1';
            doBranch <= '1';
        else
            wPC      <= '0';
```

Q5 (bonus):

Il est impossible de pouvoir faire un CPI de 2 pour le read_mem étant donné que l'opération de mettre A à MA prend un cycle autre que ceux de base. Ainsi, nous n'obtenons pas notre donnée assez rapidement pour pouvoir l'utiliser sans encombrer les prochaines instructions. De plus, même si nous arrivions à recevoir l'instruction plus tôt, nous ne pouvions pas écrire assez rapidement dans la mémoire des registres avant que la prochaine instructions change celle-ci. De ce fait, il est impossible de réduire à un CPI de 2 l'instruction du read_mem