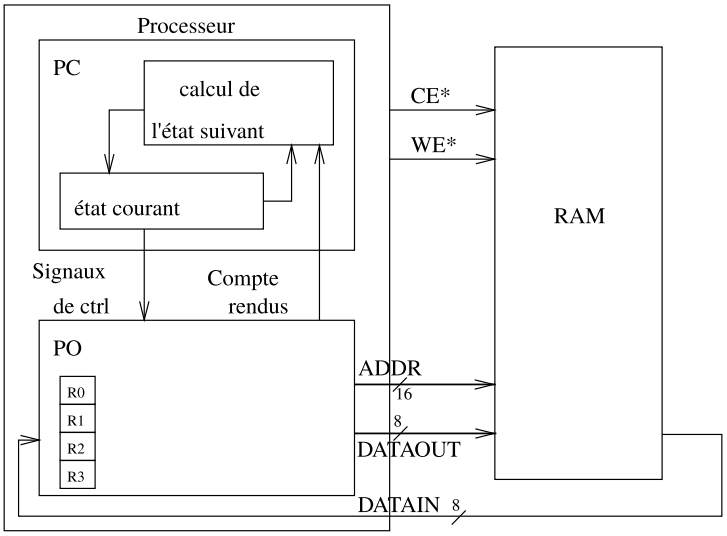
TD 8 - 9
Conception d'un processeur

L'objectif de ces deux TD est de construire un processeur capable d'effectuer des instructions très simples. Le processeur construit sera utilisé dans les deux séances de TD suivantes. On suppose que ce processeur est équipé de 4 registres internes de 8 bits chacun (nommés `r0`, `r1`, `r2` et `r3`), d'un bus adresse sur 16 bits et de 2 bus donnée (lecture et écriture) sur 8 bits. On rappelle l'architecture globale du processeur et sa liaison avec la mémoire asynchrone :



Le jeu d'instructions de ce processeur est le suivant :

- `st ri, @mem` stocke le contenu du registre `ri` dans l'octet situé à l'adresse `@mem` ;
- `ld @mem, ri` stocke le contenu de l'octet situé à l'adresse `@mem` dans le registre `ri` ;
- `op rs, rd` stocke dans le registre `rd` le résultat de l'opération `rd op rs`.

Les opérations à deux opérandes supportées par ce processeur sont l'addition (instruction `add`), la soustraction (instruction `sub`), la conjonction (instruction `and`), la disjonction (instruction `or`) et la disjonction exclusive (instruction `xor`).

Les opérations unaires supportées (`rd := op rd`) sont la négation (instruction `not`), le décalage d'un bit vers la gauche (instruction `shl`) et le décalage arithmétique d'un bit vers la droite (instruction `shr`).

On donne également l'algorithme exécuté par le processeur pour récupérer, décoder et exécuter les instructions d'un programme situé à l'adresse $0x4000$:

```
1  PC ← 0x4000;
2  while True do
3  |  IR ← MEM(PC) ;
4  |  PC ← PC + 1 ;
5  |  switch IR do
6  |  |  case add : do  rd ← rd + rs ;                    // rd, rs ∈ {r0, r1, r2, r3}
7  |  |  case sub : do  rd ← rd − rs;
8  |  |  case and : do  rd ← rd AND rs;
9  |  |  case or : do  rd ← rd OR rs;
10 |  |  case xor : do  rd ← rd XOR rs;
11 |  |  case not : do  rd ← NOT rd;
12 |  |  case shl : do  rd ← rd ≪ 1;
13 |  |  case shr : do  rd ← rd ≫ 1;
14 |  |  case st : do
15 |  |  |  AD(15 : 8) ← MEM(PC) ;
16 |  |  |  PC ← PC + 1;
17 |  |  |  AD(7 : 0) ← MEM(PC);
18 |  |  |  PC ← PC + 1;
19 |  |  |  MEM(AD) ← ri  ;                              // ri ∈ {r0, r1, r2, r3}
20 |  |  case ld : do
21 |  |  |  AD(15 : 8) ← MEM(PC) ;
22 |  |  |  PC ← PC + 1;
23 |  |  |  AD(7 : 0) ← MEM(PC);
24 |  |  |  PC ← PC + 1;
25 |  |  |  ri ← MEM(AD) ;
26 |  |  end case
27 |  end switch
28 end while
```

On remarquera que l'algorithme suppose que les instructions d'accès à la mémoire sont sur 3 octets et les autres instructions sur un seul octets.

Il s'agit donc de concevoir le processeur sur le modèle PC-PO.

Partie opérative

Question 1 Identifier et lister les registres dans cet algorithme. Préciser leur taille, puis expliquer leur rôle et leur fonctionnement (en particulier, qui les lit et les écrit).

Question 2 Définir l'ensemble des opérations à réaliser. Proposer des opérateurs pour réaliser ces actions, en cherchant à utiliser le moins d'opérateurs possible.

Question 3 Proposer une partie opérative interconnectant les unités fonctionnelles identifiées (utiliser le composant décrit en annexe). Indiquer clairement les signaux de commandes (sélecteurs

de mux, signaux de chargement de registres, etc) et les comptes-rendus.

Question 4 Repérer les actions de l'algorithme pouvant être réalisées en parallèle sur votre PO. Réécrire l'algorithme en utilisant la notion d'écriture concurrente.

Partie contrôle

Automate de contrôle

Question 5 Pour piloter la PO, proposer une machine d'état correspondant au nouvel algorithme en précisant uniquement les opérations de transfert de registre à registre [1]. Ne préciser ni les signaux de contrôle ni les conditions sur les transitions pour l'instant.

Question 6 Simplifier la PC de manière à réduire le nombre d'états successeurs de l'état « decode » (correspondant au test du switch).

On va chercher maintenant à écrire les valeurs des conditions sur les transitions de la partie contrôle. Ces conditions de transitions dépendent du codage retenu pour chacune des instructions. Comme ce codage n'a pas encore été fixé, nous allons pour l'instant utiliser des fonctions booléennes f(IR) qui représentent ce lien entre le contenu du registre IR et l'information que l'on cherche à extraire :

- *AccesMem(IR)* : fonction exprimant si le codop de l'instruction stockée dans IR est un accès à la mémoire ;
- *AccesEcr(IR)* : fonction exprimant si le codop de l'instruction stockée dans IR est un accès à la mémoire en écriture.

Question 7
Compléter les conditions manquantes sur les transitions de votre automate.

Codage des instructions

L'objectif est maintenant de compléter la partie contrôle du processeur par la génération des signaux de contrôle vers la partie opérative, en choisissant un codage astucieux des instructions.

Question 8 Comme à la question précédente, on se donne des fonctions booléennes qui représentent le lien entre le contenu du registre IR et les informations qu'on cherche à extraire de l'instruction courante. On cherchera par la suite à trouver un codage des instructions qui minimise l'expression de ces fonctions. Lorsque la fonction n'a pas de sens (par exemple, l'instruction `st` n'a pas de registre destination Rd), sa valeur est arbitraire (ϕ-valeur). On considère 2 types de fonctions :

- *selRs(IR)*, *selRd(IR)*, *selRi(IR)* : Fonctions prenant les 8 bits de IR en entrée et retournant les 2 bits déterminant le numéro de registre Rs, Rd ou Ri de l'instruction décodée.
- *selUAL(IR)* : Fonction retournant l'opération réalisée par l'instruction en la codant sur 3 bits selon le codage de l'UAL fournit en annexe.

Exprimer, sous forme de tableau, pour chaque état de la PC la valeur des signaux de contrôle en fonction de *selRs(IR)*, *selRd(IR)*, *selRi(IR)*, *selUAL(IR)*. Lorsque la valeur dépend d'un numéro de registre, on pourra par exemple écrire « *selRd(IR) = 10* » pour dire que la valeur est

1. C'est à dire les opérations réalisées dans l'algorithme décrivant le processeur.

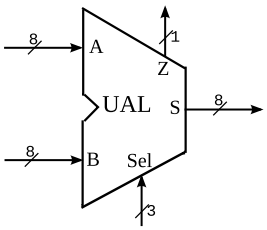
1 lorsque *selRd(IR)* vaut “10” (2 écrit en binaire) et 0 sinon.

Question 9 Proposer un codage astucieux de chaque instruction en essayant de minimiser les fonctions utilisées précédemment.

Question 10 Substituer dans le graphe d'états et dans le tableau des signaux de commandes les fonctions booléennes que nous avions définies de façon abstraite par des fonctions des bits du registre `IR`.

Pour aller plus loin...
Question 11 Proposer une réalisation de la partie contrôle à partir de bascules D et de portes, en codage 1 parmi n.

Annexe : UAL à utiliser dans ce TD



Sel	S
000	A or B
001	A xor B
010	A and B
011	not A
100	A + B
101	A - B
110	A $\ll$ 1
111	A $\gg$ 1

Z=1 ssi S=0

Pour aller plus loin...
Question 12 Construire l'UAL décrite ci-dessus en utilisant l'UAL vu au TD 5, dont le fonctionnement est rappelé ci-dessous, et deux multiplexeurs 8 bits 2 vers 1. Donner les fonctions numériques dépendant des bits de **sel** permettant de contrôler l'UAL (via CI, M, f0, f1, f2 et f3) et les multiplexeurs.

Opération	Mode M	f0 f1 f2 f3	Résultat
add	1	1010	A + B + CI
sub	1	0101	A - B + 1 - CI
notA	0	1111	not(A)
xor	0	1010	A XOR B
or	0	0010	A OR B
and	0	0100	A AND B