# Ébauche d'un processeur

# Plan

# Le but du jeu

Notre but est de montrer comment ébaucher la micro-architecture d'un processeur suivant le modèle de Von Neumann :

- une *mémoire centrale* contient le programme et les données ;
- le *processeur* exécute le programme contenu en mémoire centrale.

De façon assez abstraite, on a décomposé le processeur en deux parties :

- l'*unité de contrôle*, chargée de commander l'exécution des instructions ;
- l'*unité arithmétique et logique*, qui effectue les opérations de traitement.

Ici, il faut entrer dans plus de détails, et mettre au point tout le *chemin de données* du processeur : le chemin de données est l'ensemble des circuits pour le traitement des instructions et des données, hormis toute la partie contrôle.

Dans la suite, nous allons proposer une implantation d'une partie de l'architecture du LC3 décrite précédemment :

- on va commencer à réfléchir aux instructions arithmétiques et à LD/ST,
- vous ajouterez une instruction de branchement (BR) en TP.

1. Il va falloir proposer un chemin de données adapté pour l'architecture.
2. Lister les actions algorithmiques requises pour chaque instruction.
3. Mettre au point l'unité de contrôle du processeur.

Lors de la mise au point d'un processeur réel, il faut bien entendu des allers et retours nombreux entre ces différentes phases...

# Méthodologie de synchronisation

On va concevoir un circuit dans lequel certains signaux logiques doivent être lus, d'autres écrits à des instants précis : si on se retrouvait dans une situation où un signal doit être lu en même temps qu'il est écrit, cela poserait problème. . .

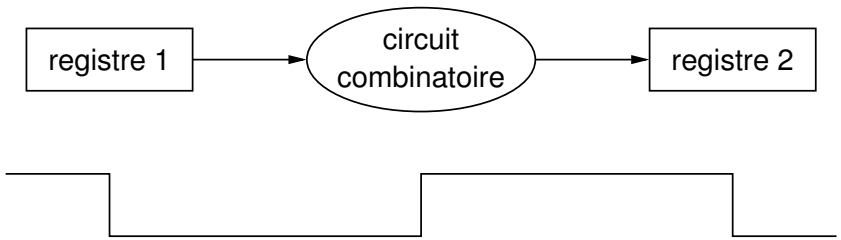
On doit donc choisir une *méthodologie de synchronisation* pour définir quand les signaux seront lus ou écrits.

On choisit une *méthodologie de synchronisation basée sur le front descendant de l'horloge* : cela signifie que les valeurs contenues les registres du circuit seront mises à jour seulement en fin de cycle, sur un front descendant de l'horloge.

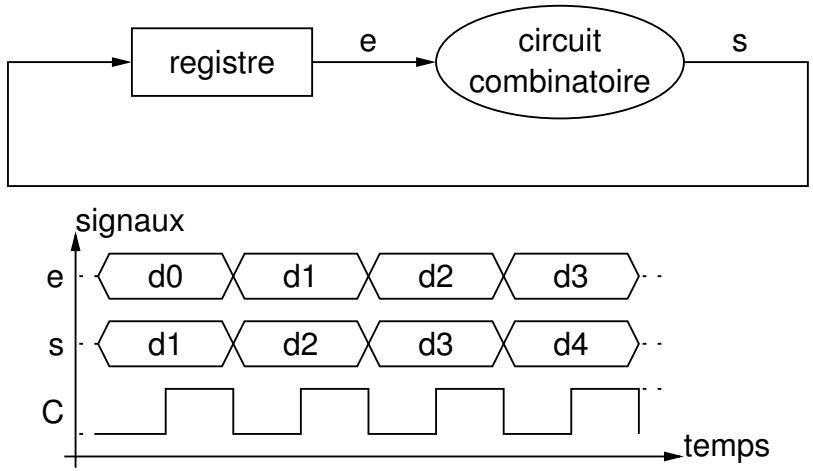
Seuls les registres permettent de stocker une donnée, et de la maintenir sur leur sortie ; donc tout « sous-circuit » combinatoire doit :

- avoir ses entrées connectées aux sorties d'un registre,
- avoir ses sorties connectées aux entrées d'un registre.

Les entrées du « sous-circuit » ont été écrites dans un registre au cycle précédent, alors que ses sorties seront utilisables au cycle suivant.




La sortie d'un registre peut aussi être lue et envoyée au travers d'un circuit combinatoire vers l'entrée du même registre :



# Plan

# Les principaux circuits requis

Une bonne manière de débuter la conception du chemin de données est d'examiner les différents composants requis par chaque classe d'instructions.

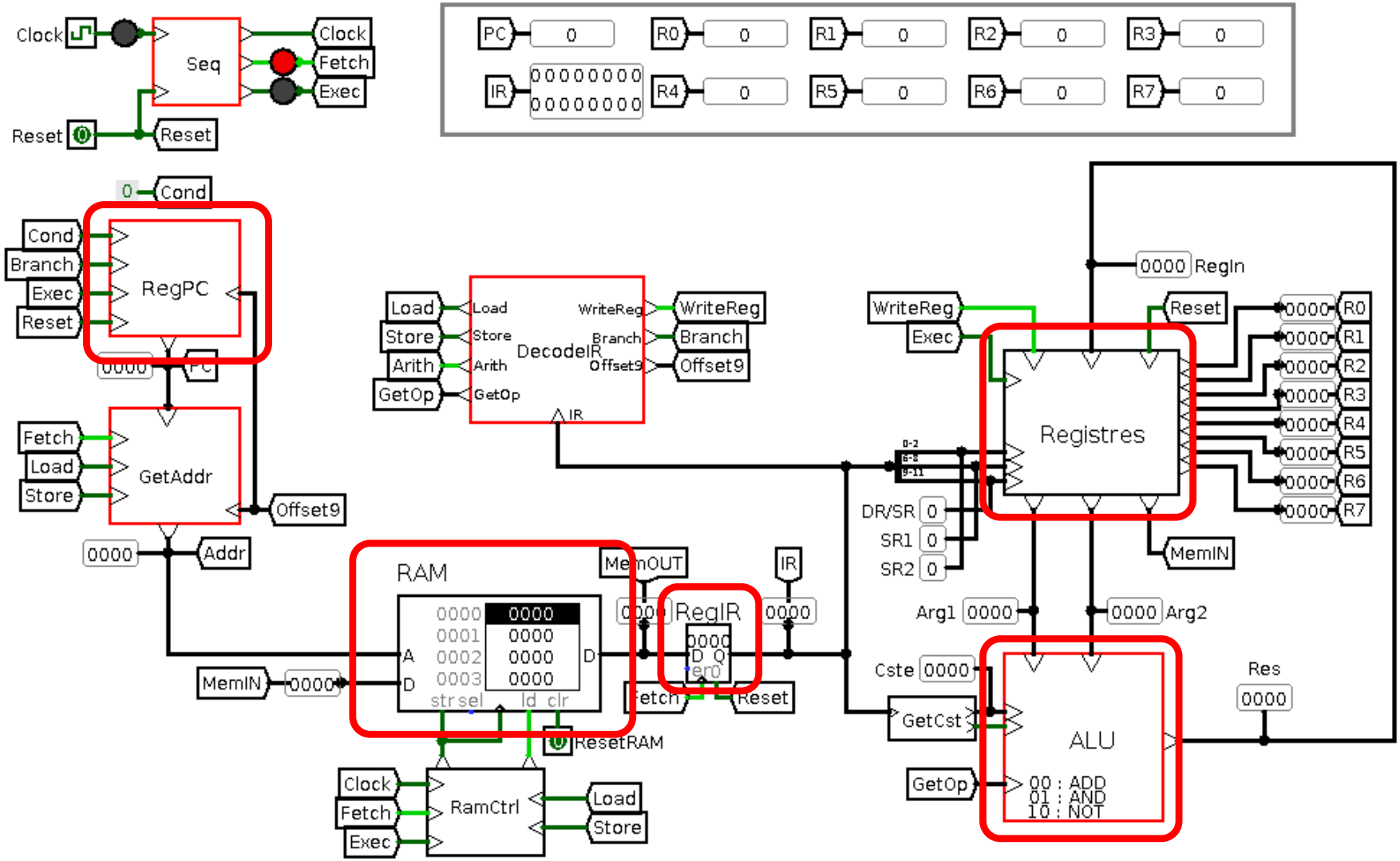
On a 3 classes : instructions arithmétiques, accès mémoire, branchement.

| **syntaxe** | **action** | **NZP** | **codage** | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | opcode | | | | arguments | | | | | | | | | | | |
| | | | F | E | D | C | B | A | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| NOT DR,SR | DR <- not SR | * | 1 0 0 1 | | | | DR | | | SR | | | 1 1 1 1 1 1 | | | | | |
| ADD DR,SR1,SR2 | DR <- SR1 + SR2 | * | 0 0 0 1 | | | | DR | | | SR1 | | | 0 | 0 0 | | SR2 | | |
| ADD DR,SR1,Imm5 | DR <- SR1 + SEXT(Imm5) | * | 0 0 0 1 | | | | DR | | | SR1 | | | 1 | Imm5 | | | | |
| AND DR,SR1,SR2 | DR <- SR1 and SR2 | * | 0 1 0 1 | | | | DR | | | SR1 | | | 0 | 0 0 | | SR2 | | |
| AND DR,SR1,Imm5 | DR <- SR1 and SEXT(Imm5) | * | 0 1 0 1 | | | | DR | | | SR1 | | | 1 | Imm5 | | | | |
| LD DR,label | DR <- mem[PC + SEXT(PCoffset9)] | * | 0 0 1 0 | | | | DR | | | PCoffset9 | | | | | | | | |
| ST SR,label | mem[PC + SEXT(PCoffset9)] <- SR | | 0 0 1 1 | | | | SR | | | PCoffset9 | | | | | | | | |
| BR[n][z][p] label | Si (cond) PC <- PC + SEXT(PCoffset9) | | 0 0 0 0 | | | | n | | z | p | PCoffset9 | | | | | | | |

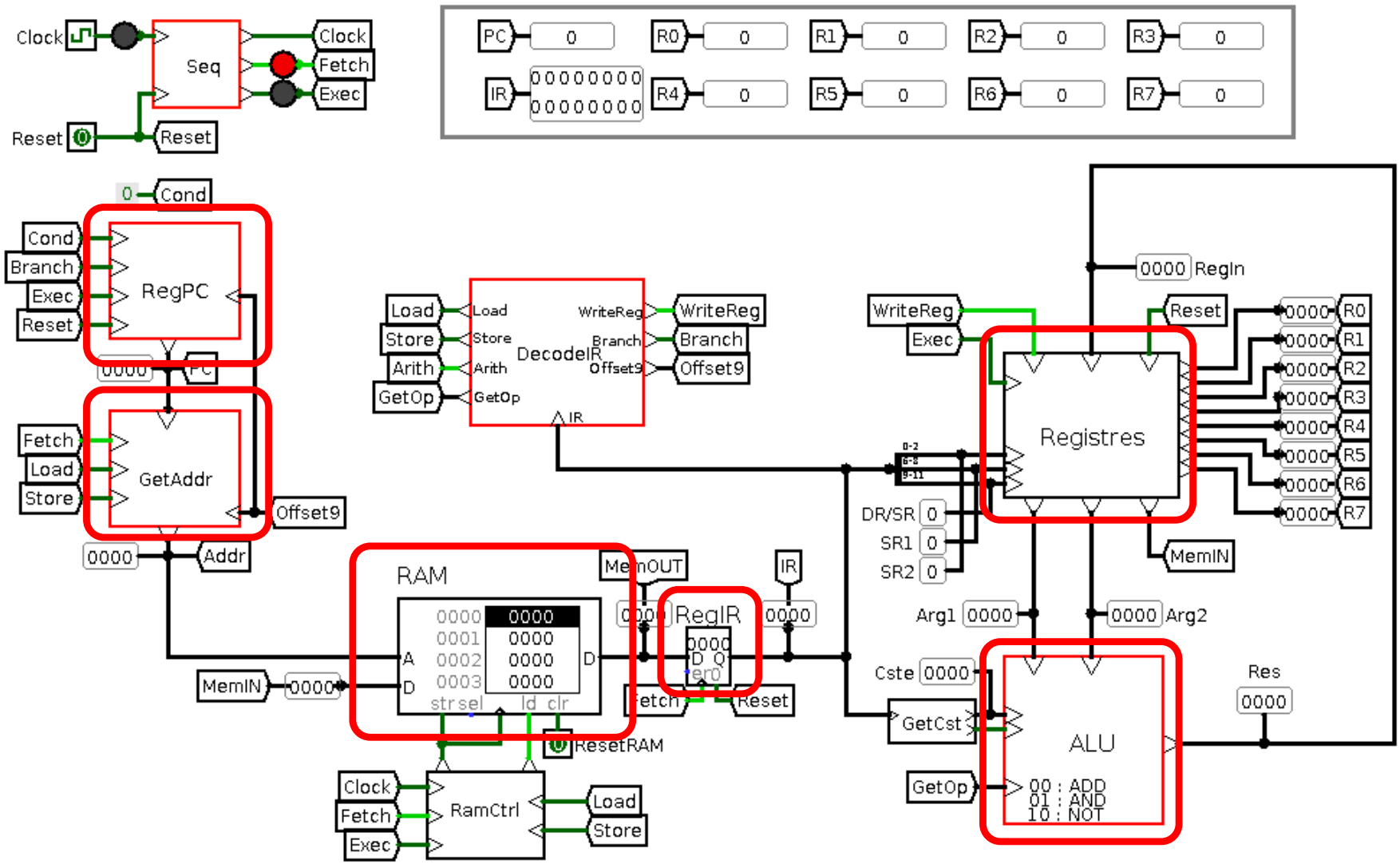
On utilisera au moins les éléments suivants :

- un compteur de programme PC, et un registre d'instruction IR,
- une mémoire RAM,
- un banc de registres ;
- une ALU.

# Principaux circuits du chemin de données :



Principaux circuits du chemin de données :




On ajoute un circuit GetAddr, qui servira pour le calcul des adresses pour les instructions d'accès à la mémoire et le branchement. Pour l'instant, Addr = PC.

# Le cycle d'instruction

Le cycle d'instruction peut être découpé en plusieurs cycles d'horloge : les résultats obtenus à un cycle d'horloge, et devant être utilisés au cycle suivant, doivent être stockés dans un registre ou la mémoire. Quel découpage effectuer ?

On passe en revue les classes d'instructions :

- Instructions arithmétiques : opérandes lus dans le banc de registres ou dans l'instruction, et le résultat écrit en fin de cycle d'instruction.
- Instructions d'accès à la mémoires : échange entre le banc de registres et la mémoire, avec écriture a priori en fin de cycle d'instruction.
- Branchement : calcul de l'adresse de destination à partir de l'instruction, et écriture en fin de cycle d'instruction du PC.

$\hookrightarrow$ a priori, on pourrait avoir un cycle d'instruction par cycle d'horloge...

Le point est que l'on a choisi d'utiliser un registre d'instruction IR. Il faut :

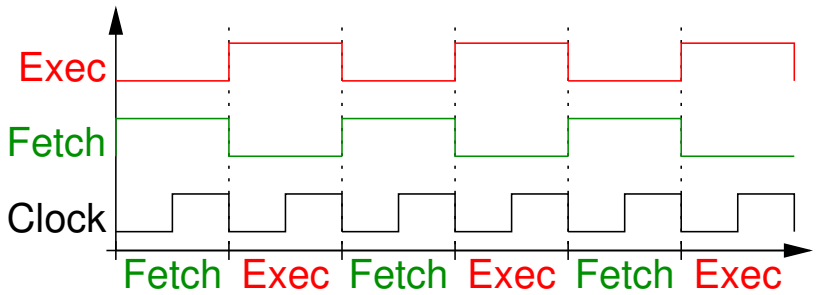
- un cycle d'horloge pour charger une instruction de la RAM vers IR,
- un autre pour produire un résultat en fonction de l'instruction chargée.

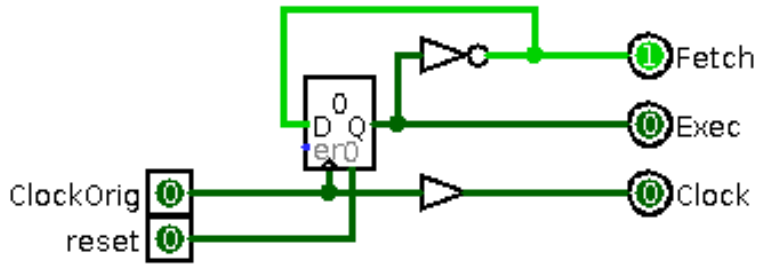
On va donc découper le cycle d'instruction en deux phases :

- Fetch : chargement d'une instruction dans IR,
- Exec : exécution de l'instruction.

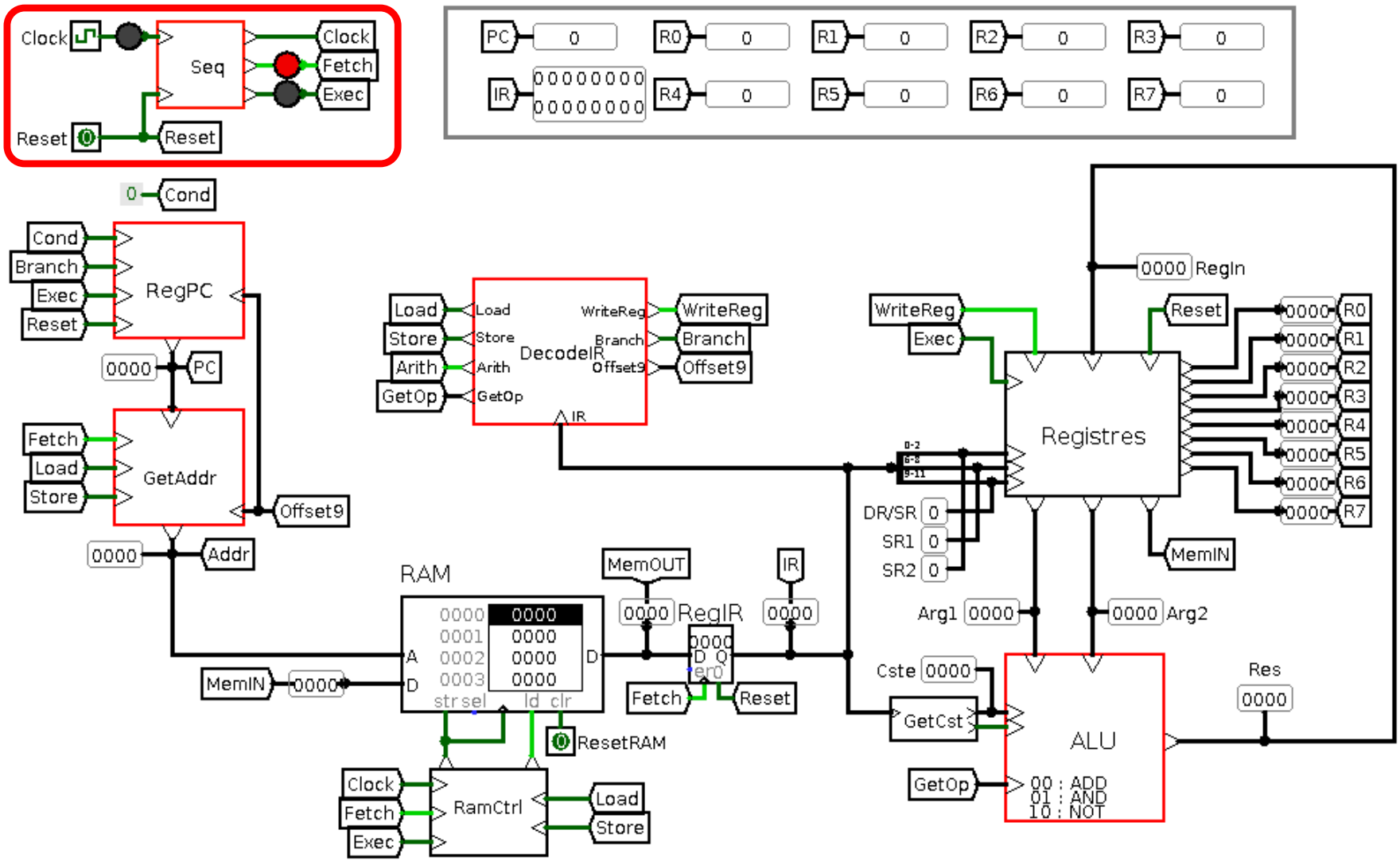
A chaque phase correspond un cycle d'horloge.

On utilise pour cela un circuit séquenceur appelé Seq :



# Le circuit Seq produit les signaux Clock, Fetch et Exec :



# Autres circuits de contrôle

Le circuit RamCtrl contrôle la mémoire RAM :

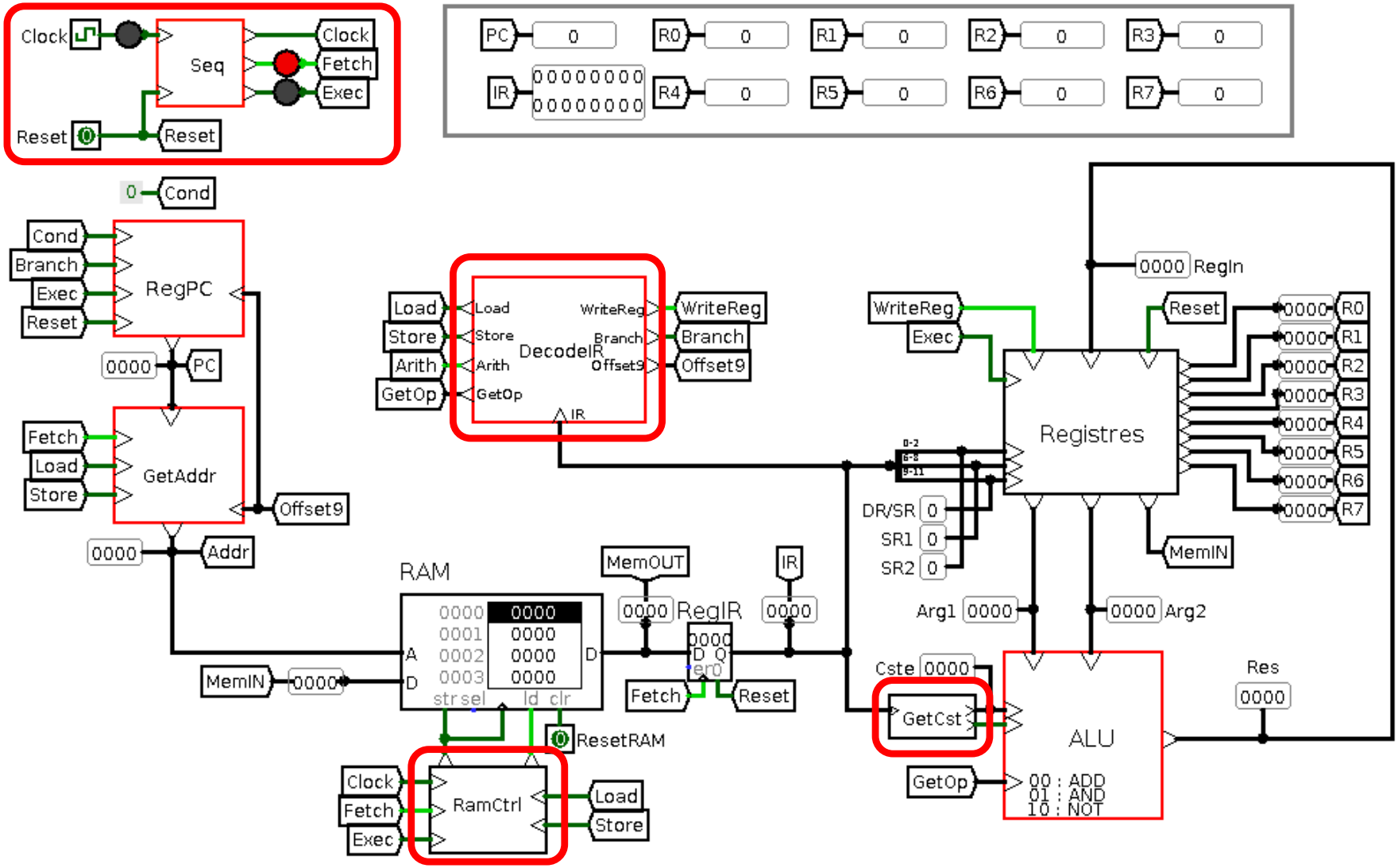
- il la met en mode lecture pendant la phase Fetch ou pour l'exécution de LD,
- il la place en mode écriture dans le cas de l'exécution de ST.

Le circuit DecodeIR se charge de décoder l'instruction pendant la phase Exec :

- il active les signaux de contrôle dans le chemin de données,
- en plus, il isole le décalage Offset9 pour LD/ST ou BR.

Le circuit GetCst permet d'extraire une constante immédiate de IR, pour les instructions arithmétiques qui le nécessitent.

# Circuits utilisés pour le contrôle du chemin de données :



# Plan

# Le banc de registres

Considérons à nouveau notre jeu d'instructions :

| syntaxe | action | NZP | codage | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | opcode | | | | arguments | | | | | | | | | | | |
| | | | F | E | D | C | B | A | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| NOT DR,SR | DR <- not SR | * | 1 0 0 1 | | | | DR | | | SR | | | 1 1 1 1 1 1 | | | | | |
| ADD DR,SR1,SR2 | DR <- SR1 + SR2 | * | 0 0 0 1 | | | | DR | | | SR1 | | | 0 | 0 0 | | SR2 | | |
| ADD DR,SR1,Imm5 | DR <- SR1 + SEXT(Imm5) | * | 0 0 0 1 | | | | DR | | | SR1 | | | 1 | Imm5 | | | | |
| AND DR,SR1,SR2 | DR <- SR1 and SR2 | * | 0 1 0 1 | | | | DR | | | SR1 | | | 0 | 0 0 | | SR2 | | |
| AND DR,SR1,Imm5 | DR <- SR1 and SEXT(Imm5) | * | 0 1 0 1 | | | | DR | | | SR1 | | | 1 | Imm5 | | | | |
| LD DR,label | DR <- mem[PC + SEXT(PCoffset9)] | * | 0 0 1 0 | | | | DR | | | PCoffset9 | | | | | | | | |
| ST SR,label | mem[PC + SEXT(PCoffset9)] <- SR | | 0 0 1 1 | | | | SR | | | PCoffset9 | | | | | | | | |
| BR[n][z][p] label | Si (cond) PC <- PC + SEXT(PCoffset9) | | 0 0 0 0 | | | | n | | z | p | PCoffset9 | | | | | | | |

Notons que :

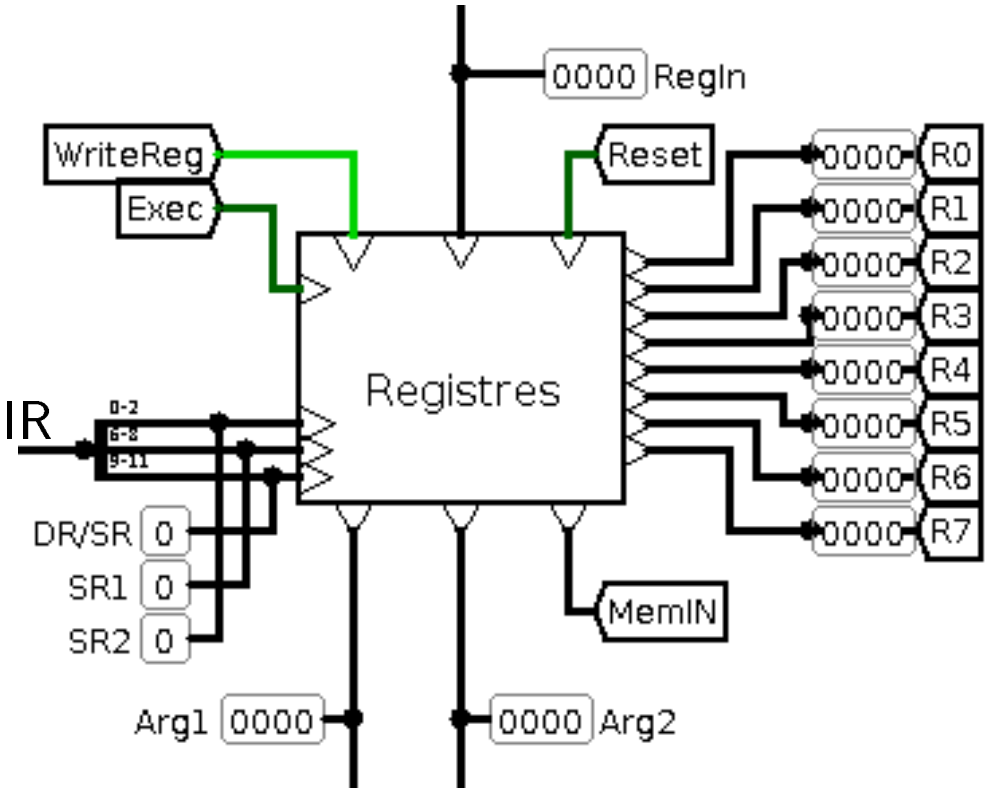
- BR n'utilise pas le banc de registres,
- les ins. arithmétiques nécessitent 2 ports de lecture et 1 port d'écriture.

Si le processeur a chargé une instruction autre qu'un BR dans IR :

- IR[11-9] est toujours un indice de registre (à lire ou à écrire),
- IR[8-6] est un indice de registre source pour les ins. arithmétiques,
- IR[2-0] est un indice de registre source pour certaines ins. arithmétiques.

Le banc de registres utilisé présente :

- 3 ports de lectures : 2 pour les ins. arith, 1 pour le LD [1],
- 1 port d'écriture.




Il est contrôlé par les signaux WriteReg, Exec et Reset.

1. C'est du gaspillage de ressources : comment faire plus simple ?

# L'unité arithmétique et logique

Les instructions arithmétiques prennent comme deuxième argument un registre source SR1, et comme troisième argument soit :

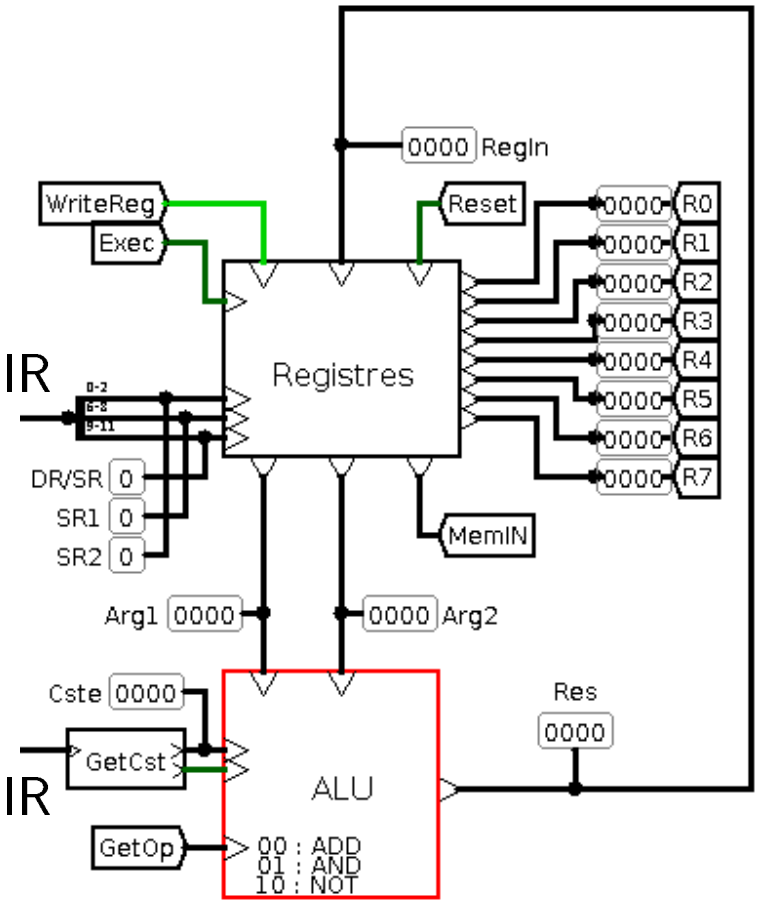
- un registre source SR2 (ADD DR, SR1, SR2),
- une constante immédiate sur 5 bits (ADD DR, SR1, Imm5),
- rien du tout (NOT DR, SR1).

| **syntaxe** | **action** | **NZP** | **codage** | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | opcode | | | | arguments | | | | | | | | | | | |
| | | | F | E | D | C | B | A | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| NOT DR,SR1 | DR <- not SR1 | * | 1 0 0 1 | | | | DR | | | SR1 | | | 1 1 1 1 1 1 | | | | | |
| ADD DR,SR1,SR2 | DR <- SR1 + SR2 | * | 0 0 0 1 | | | | DR | | | SR1 | | | 0 | 0 0 | | SR2 | | |
| ADD DR,SR1,Imm5 | DR <- SR1 + SEXT(Imm5) | * | 0 0 0 1 | | | | DR | | | SR1 | | | 1 | Imm5 | | | | |
| AND DR,SR1,SR2 | DR <- SR1 and SR2 | * | 0 1 0 1 | | | | DR | | | SR1 | | | 0 | 0 0 | | SR2 | | |
| AND DR,SR1,Imm5 | DR <- SR1 and SEXT(Imm5) | * | 0 1 0 1 | | | | DR | | | SR1 | | | 1 | Imm5 | | | | |

On met donc en place une unité arithmétique qui prend en entrée :

- forcément, une premier opérande venant du banc de registres,
- un second opérande venant soit du banc de registres, soit de GetCst.

Le résultat calculé retourne sur le port d'écriture du banc.




Notons que GetOp contrôle l'opération qui doit être effectuée par l'ALU.

# La mémoire

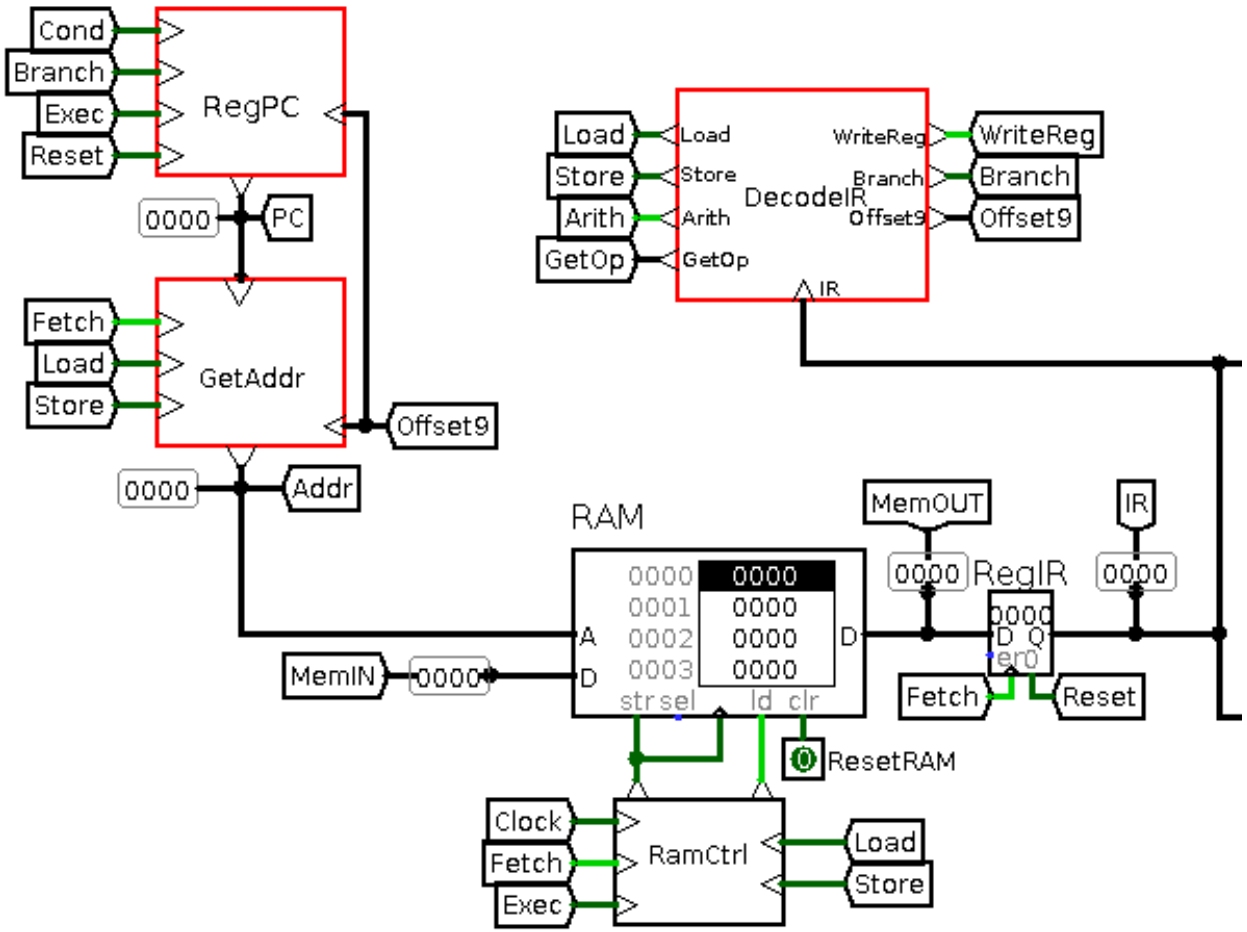
Ce que l'on veut pouvoir faire avec la mémoire :

- Pendant la phase de Fetch, on veut que l'instruction dont l'adresse est donnée par le PC soit chargée dans IR.
- Pendant la phase Exec d'un LD ou d'un ST, on veut soit lire, soit écrire une donnée en mémoire.

| **syntaxe** | **action** | **NZP** | **codage** | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | opcode | | | | arguments | | | | | | | | | | | |
| | | | F | E | D | C | B | A | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| LD DR,label | DR <- mem[PC + SEXT(PCoffset9)] | * | 0 0 1 0 | | | | DR | | PCoffset9 | | | | | | | | | |
| ST SR,label | mem[PC + SEXT(PCoffset9)] <- SR | | 0 0 1 1 | | | | SR | | PCoffset9 | | | | | | | | | |

L'adresse de lecture ou d'écriture de la RAM est calculée par GetAddr ; ce sera :

- PC dans pendant la phase de Fetch,
- PC + SEXT(PCoffset9) pendant la phase Exec d'un LD ou d'un ST.




Signaux de contrôle : Load, Store, Fetch, Exec et Clock.

Fonction de RamCtrl :

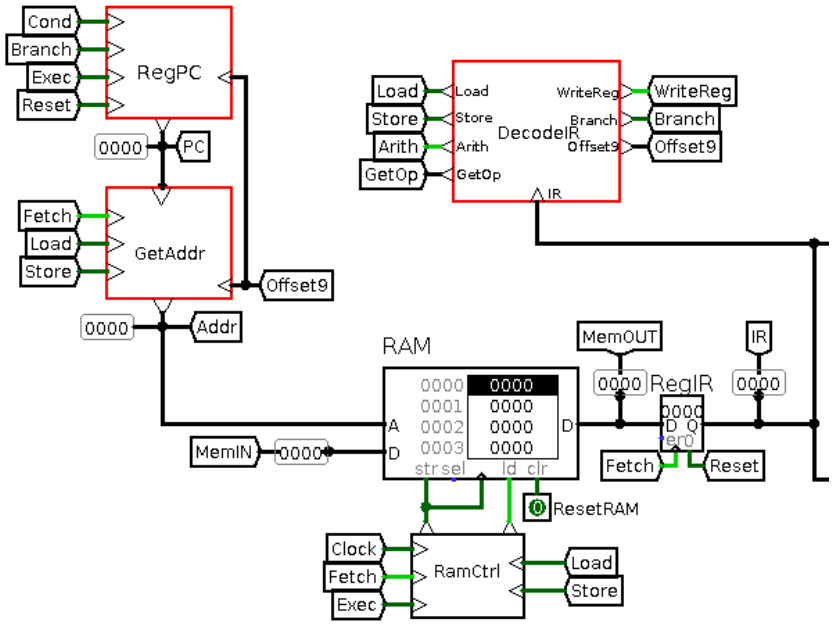
- lecture si $\text{Fetch} + \text{Exec} \cdot \text{Load}$,
- écriture si $\text{Exec} \cdot \text{Store} \cdot \text{Clock}$.

# Plan

# Phase de Fetch

Au cours de la phase de Fetch le processeur doit réaliser IR ← Mem[PC] :

- Le séquenceur est en mode Fetch, donc la mémoire est en lecture,
- PC est disponible directement en sortie de RegPC,
- RegIR reçoit donc l'instruction à charger sur son entrée.




A la fin de la phase de Fetch, le signal Fetch retombe à 0, donc RegIR reçoit l'instruction courant, et la maintient sur sa sortie pendant toute la phase Exec.

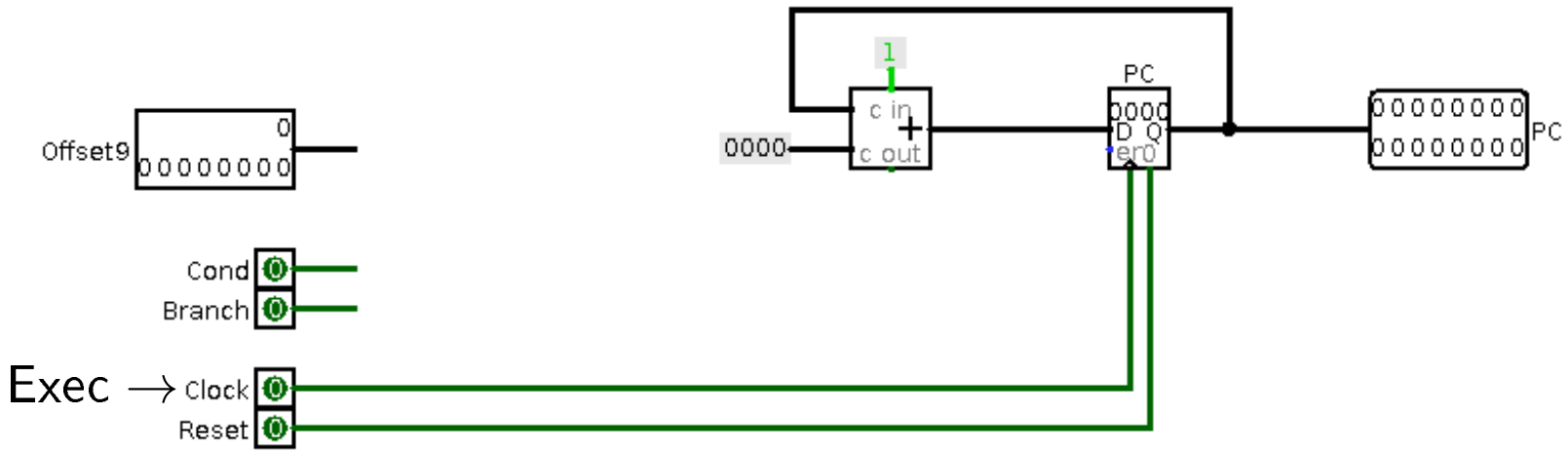
# Phase Exec

Au cours de la phase de Exec, le processeur doit :

- mettre à jour PC en vue du cycle d'instruction suivant,
- effectuer les actions requises pour exécuter l'instruction dans IR,
- en fin de cycle, le résultat doit être écrit en mémoire ou dans un registre.

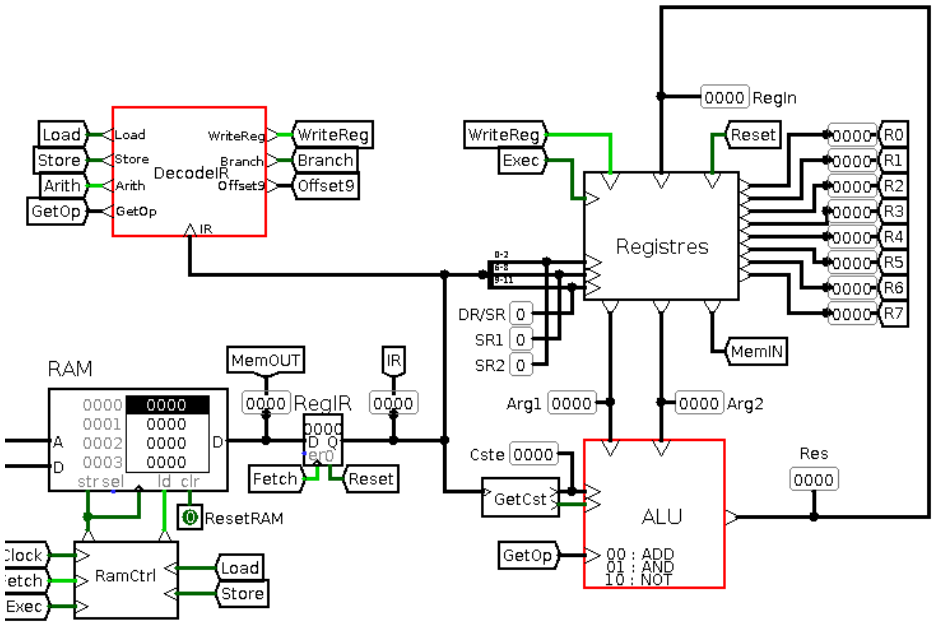
Comme on ne se préoccupe pas de BR pour l'instant, la mise à jour de PC est

$$PC \leftarrow PC + 1.$$



# Phase Exec d'une instruction arithmétique

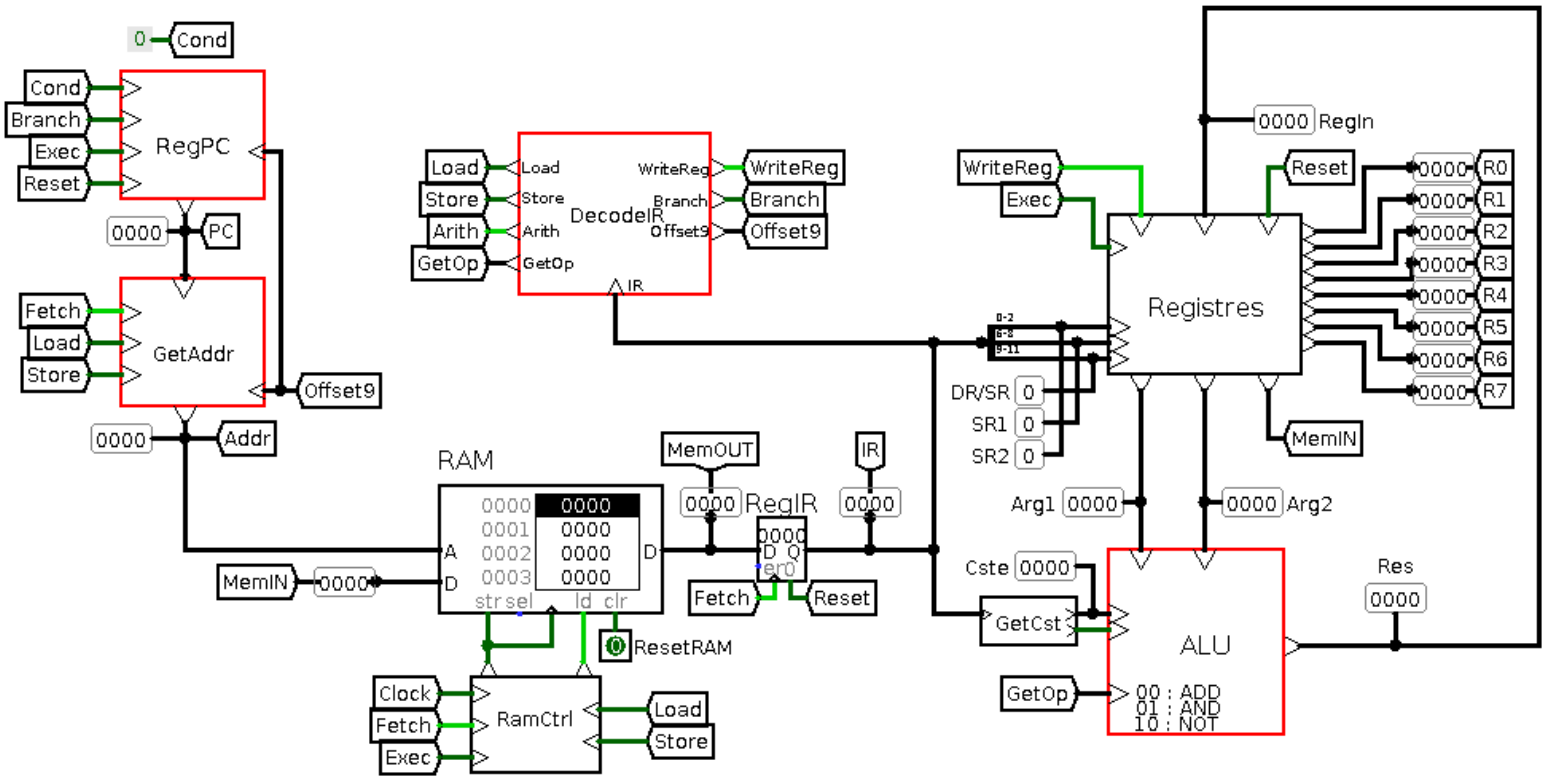
Considérons ADD DR, SR1, SR2, pour laquelle l'action est DR <- SR1 + SR2.




- Le banc reçoit SR1, SR2 comme indices de registres à lire, donc l'ALU reçoit les bons opérandes. Le banc reçoit aussi l'indice de registre DR.
- GetOp doit être correctement activé pour que l'ALU effectue une addition.
- WriteReg doit être activé pour l'écriture du résultat en fin de cycle.

# Phase Exec d'une instruction d'accès à la mémoire

Pour ST SR, label, l'action est mem[PC + SEXT(PCoffset9)] <- SR .




- Le banc reçoit SR comme indice de registre à lire, donc on a le bon contenu sur MemIN, qui est redirigé vers l'entrée de la mémoire.
- Store doit être activé pour que GetAddr calcule l'adresse de rangement en fonction de PC et de Offset9. De plus, cela met la mémoire en écriture.

# Circuit DecodeIR

En passant en revu chacune des instructions du jeu d'instruction, on peut déterminer la valeur que doivent prendre les signaux de contrôle (Load, Store, WriteReg, GetOp Offset9) pour chaque instruction.

| | | | opcode | | | | arguments | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | F | E | D | C | B | A | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| NOT DR,SR | DR <- not SR | * | 1 | 0 | 0 | 1 | DR | | | SR | | | 1 | 1 | 1 | 1 | 1 | 1 |
| ADD DR,SR1,SR2 | DR <- SR1 + SR2 | * | 0 | 0 | 0 | 1 | DR | | | SR1 | | | 0 | 0 | 0 | SR2 | | |
| ADD DR,SR1,Imm5 | DR <- SR1 + SEXT(Imm5) | * | 0 | 0 | 0 | 1 | DR | | | SR1 | | | 1 | Imm5 | | | | |
| AND DR,SR1,SR2 | DR <- SR1 and SR2 | * | 0 | 1 | 0 | 1 | DR | | | SR1 | | | 0 | 0 | 0 | SR2 | | |
| AND DR,SR1,Imm5 | DR <- SR1 and SEXT(Imm5) | * | 0 | 1 | 0 | 1 | DR | | | SR1 | | | 1 | Imm5 | | | | |
| LD DR,label | DR <- mem[PC + SEXT(PCoffset9)] | * | 0 | 0 | 1 | 0 | DR | | | PCoffset9 | | | | | | | | |
| ST SR,label | mem[PC + SEXT(PCoffset9)] <- SR | | 0 | 0 | 1 | 1 | SR | | | PCoffset9 | | | | | | | | |
| BR[n][z][p] label | Si (cond) PC <- PC + SEXT(PCoffset9) | | 0 | 0 | 0 | 0 | n | | z | p | PCoffset9 | | | | | | | |

Il faut ensuite exprimer chacun de ces signaux en fonction du code de l'instruction contenu dans IR. On peut noter par exemple que :

- GetOp $=$ IR[15-14]
- Load $=$ IR[13] $\cdot \overline{\text{IR[12]}}$
- Store $=$ IR[13] $\cdot$ IR[12]
- WriteReg $= \overline{\text{IR[13]}} \cdot$ IR[12] $+$ *Load*
- . . . le reste sera à faire en TP !

# Plan

# Conclusion

Pour construire cette ébauche, nous sommes partis de la description (restreinte) d'une architecture au niveau du jeu d'instruction :

- nous avons mis en place un chemin de données pour cette architecture,
- le cycle d'une instruction peut prendre deux cycles d'horloge,
- nous avons commencé la partie contrôle du processeur.

En TP, vous compléterez ce processeur :

- en étudiant en détails l'exécution des instructions arithmétiques,
- en mettant en place le contrôle pour les instructions LD/ST,
- en complétant le chemin de données et le contrôle pour BR.