TP 6
Construisons le LC-3 - partie 1

Fichiers fournis : `LC3_etu.circ`, `tp6_add_simple.mem`

6.1 Le circuit LC-3

Récupérez le fichier `LC3_etu.circ` et ouvrez-le avec LOGISIM. Ce fichier contient une implémentation partielle du LC-3, dans laquelle seules les instructions `ADD`, `AND`, `NOT` sont câblées, même si d'autres infrastructures sont déjà en place pour plus tard.

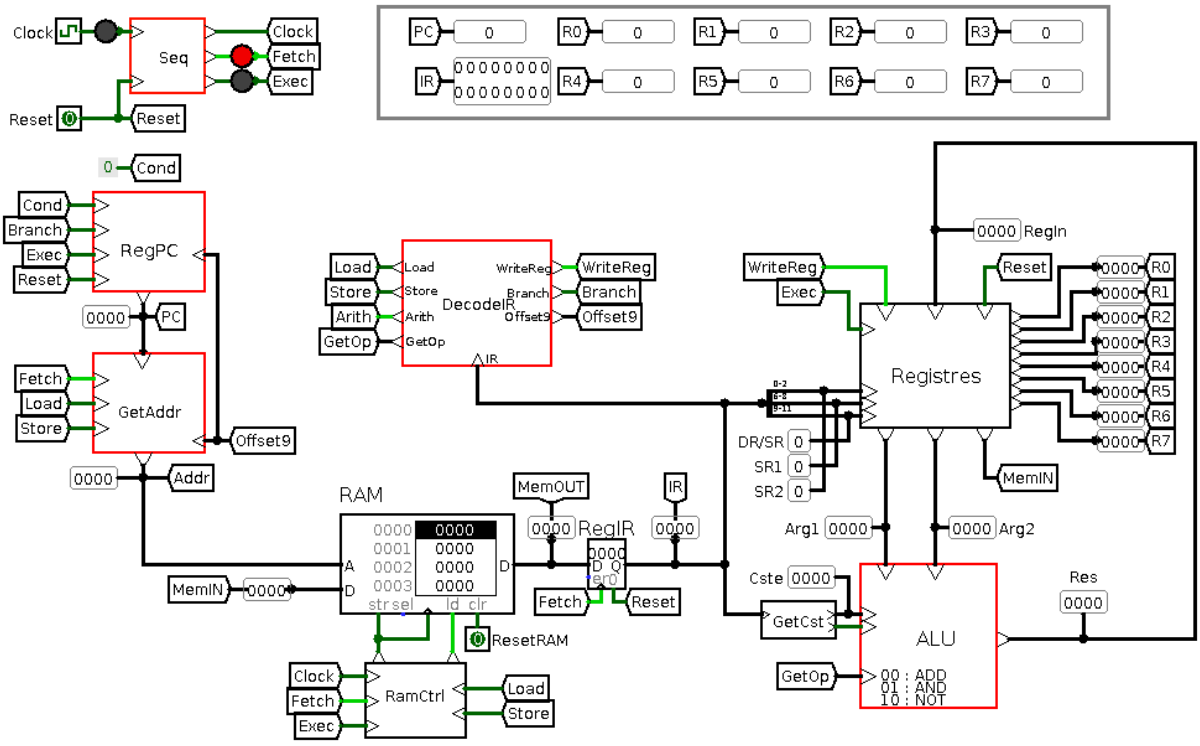

FIGURE 6.1 – Le circuit LC-3 fourni, onglet principal

Bref mode d'emploi. Pour simuler l'exécution de programmes LC-3 avec ce circuit :

- Entrez dans la RAM les octets d'un programme LC-3, d'une des trois manières suivantes :
  - En mode "Poke", on peut directement cliquer sur une case de la RAM et la remplir au clavier.
  - En faisant un clic-droit sur la RAM, on peut choisir "Edit Contents", ce qui ouvre un éditeur.
  - Le menu obtenu par un clic-droit sur la RAM propose aussi "Load Image" pour charger dans la RAM le contenu d'un fichier mémoire (voir ci-dessous).
- Pour faire avancer la simulation d'un programme, il faut changer l'état de l'horloge (`Clock`) du circuit (c'est-à-dire lui faire passer un front montant ou un front descendant) : vous pouvez soir cliquer sur le bouton d'horloge (en haut à gauche du circuit), soit envoyer un tic manuel *via* `Ctrl-T`.
- Si vous souhaitez relancer l'exécution du programme (remettre `PC` à 0, et vider le banc de registres), mettez l'horloge `Clock` à son niveau bas, puis faites passer l'entrée `Reset` à 1, puis de nouveau à 0.

Fichier mémoire. Un fichier mémoire chargeable dans une RAM LOGISIM est un simple fichier texte dont la première ligne est `v2.0 raw`. Viennent ensuite les différents octets de la mémoire en hexadécimal. Voir la section “Memory Components” de la documentation LOGISIM pour plus de détails. Voici par exemple le contenu du fichier `tp6_add_simple.mem` :

```
v2.0 raw
5020 1025 5260 1266 1440
```

Ce programme correspond au code en langage d’assemblage contenu dans le fichier `tp6_add_simple.mem` :

Listing 6.1 – tp6_add_simple.mem

```
1         .ORIG x0000                          ; code hexa
2         AND R0,R0,0                          ; 5020
3         ADD R0,R0,5       ; R0 <- 5          ; 1025
4         AND R1,R1,0                          ; 5260
5         ADD R1,R1,6       ; R1 <- 6          ; 1266
6         ADD R2,R1,R0      ; R2 <- R1 + R0    ; 1440
7         .END
```

EXERCICE 1 ▶ **Simulation d’un programme**
Ouvrez le circuit LC3_etu.circ, puis récupérez le programme `tp6_add_simple.mem` sur la page du cours. Chargez-le dans la RAM et lancez sa simulation. Observez en particulier l’évolution des contenu des registres (`PC`, `IR`, `R0`, ..., `R7`).

EXERCICE 2 ▶ **Cycle d’instruction**
Placez vous dans le sous-circuit `Seq` (Figure 6.2), et expérimentez. Représenter les signaux `Clock`, `Fetch` et `Exec` sur un chronogramme, en supposant que `ClockOrig` est un signal créneau périodique. Que se passe-t’il quand `Reset` est activé, puis relâché un peu plus tard ? Quel est le rôle joué par ce circuit dans cette implantation du LC-3 ?

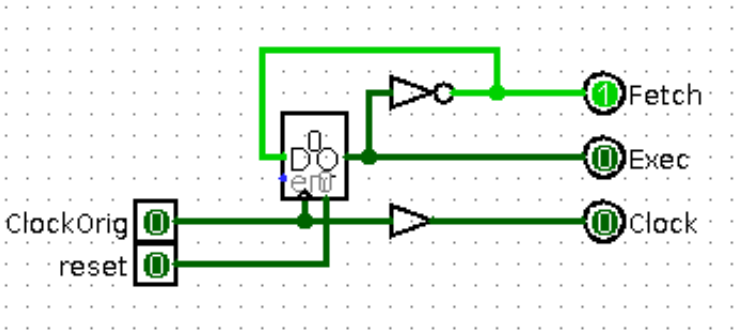

FIGURE 6.2 – Composant `Seq` du LC-3

6.2 L’unité arithmétique et logique

Placez-vous dans le module `ALU` (Figure 6.3), qui contient l’unité arithmétique et logique. Notez l’usage d’un multiplexeur, qui sélectionne parmi quatre entrées (dont une actellement non affectée) en fonction d’un fil de contrôle sur 2 bits.

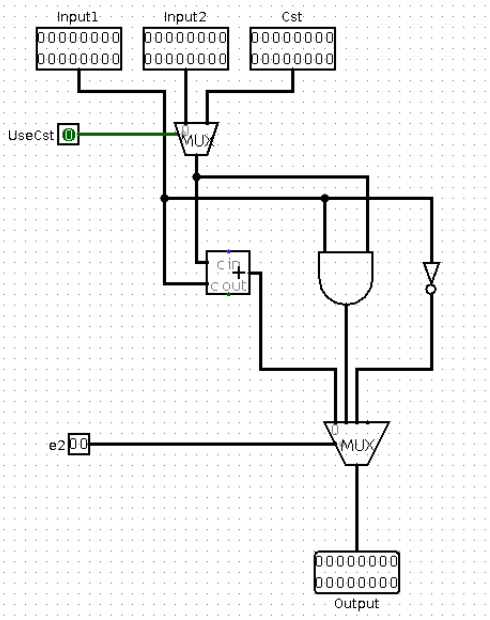

FIGURE 6.3 – Composant `ALU` du LC-3

EXERCICE 3 ▶ **Valeurs de contrôle**
Expérimentez différentes valeurs pour e_1 et e_2. Quelles valeurs donner à e_1 et e_2 pour :

— obtenir en sortie le ET bit à bit de `Input1` et `Input2` ?
— obtenir l'addition de `Input1` et `Cst`
— obtenir le NON bit à bit de `Input1` ?

EXERCICE 4 ▶ **Contrôle de l'ALU**
Reprenez le jeu d'instructions du LC-3 (cf cours). Les opérandes des instructions arithmétiques sont soit deux registres, soit un registre et une constante littérale.

— Comment sont différenciés les deux cas ?
— Quels bits différencient les opcodes des instructions arithmétiques `ADD, AND` et `NOT` ? Déduisez-en à quoi devront être branchés e_1 et les deux bits de e_2 dans le circuit complet.

6.3 Exécution des instructions arithmétiques

On suppose que vous avez ouvert le circuit `LC3_etu.circ`, puis chargé le programme `tp6_add_simple.mem` dans la RAM : servez vous de la simulation de ce programme pour répondre aux questions suivantes.

EXERCICE 5 ▶ **Phase** `Fetch`

1. En observant le module `RegPC`, expliquez comment est calculé `PC+1`. A quel instant `PC+1` remplace `PC` ?
2. Lors du cycle du chargement, comment une instruction est-elle chargée dans le registre `RegIR` ?
3. Pour l'instant, on n'exécute que des instructions arithmétiques. Expliquez les valeurs de sorties produites par `DecodeIR`.

EXERCICE 6 ▶ **Phase** `Exec`

1. Observez le fonctionnement du module `Registres` : pourquoi utiliser trois ports de lecture ? Dans quel cas sera utilisé le port de lecture `OUT (SR)` ?

2. En vous plaçant dans la phase exec de l'instruction `ADD R0,R0,5` (de code `x1025` en mémoire), expliquez comment est exécutée cette instruction. Comment les opérandes sources sont aménées à l'ALU ? Comment le résultat de l'opération est rangé dans le registre destination ? A quel instant le résultat est définitivement stocké ?

EXERCICE 7 ▶ **Chronogramme**
En supposant que l'horloge générale `Clock` du circuit est un signal créneau périodique, représentez l'évolution de l'état du circuit sur un chronogramme au cours des cycles des deux premières instructions du programme `tp6_add_simple.mem`. Vous représenterez les signaux suivants : `Clock`, `Fetch`, `Exec` ; les valeurs suivantes : `PC`, `IR`, `GetOp`, `R0`, `R1` sur la Figure 6.4.

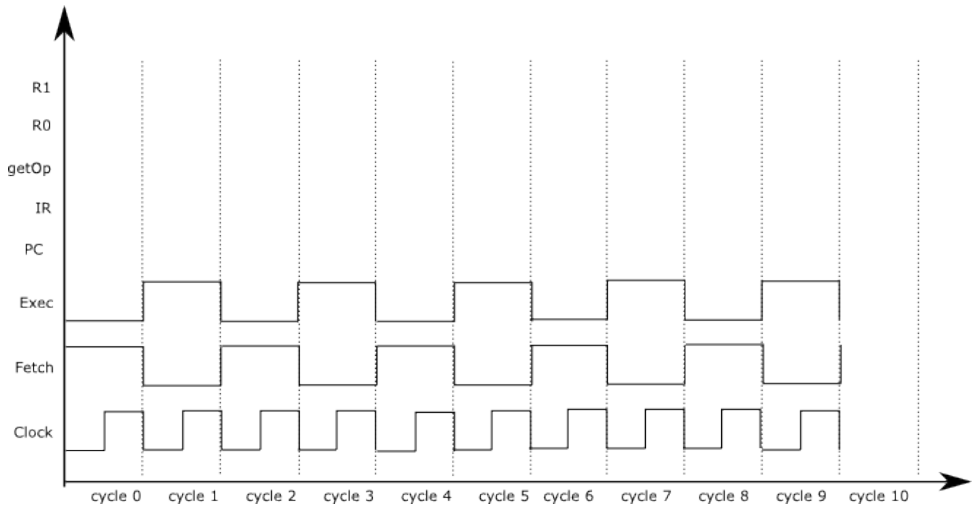

FIGURE 6.4 – Chronogramme pour `AddSimple`

6.4 Pour la suite...

EXERCICE 8 ▶ **Un programme**
Préparez un fichier `cst2007.mem` contenant les instructions en langage machine pour charger 2007 dans le registre R_1. Pour cela utilisez soit un éditeur de texte séparé, soit l'éditeur hexadécimal intégré puis l'entrée de menu "Save Image". Simulez ce programme : que constatez-vous ?

EXERCICE 9 ▶ **Opcode LC-3**
Construisez la table de vérité de toutes les instructions du LC-3 en fonction des 4 bits de l'opcode (cette table sera utile plus tard). Que pouvez-vous observer ? Comment caractériser les instructions qui peuvent modifier des registres `R0`, ..., `R7` ?