

Projet

Modalités de rendu

- Par qui : groupes de 3 ou 4.
- Quand : avant le 07/12 à 23h59.
- Quoi : une archive `Nom1Nom2Nom3(Nom4).zip` contenant le fichier Logisim, un assembleur, et tout autre fichier que vous jugerez nécessaire (ex : un rapport, un mode d'emploi, des programmes asm de test pour les questions 3, 9, 13, 16,...).
- Comment : par mail `theo.pierron@univ-lyon1.fr`.
- Pas de soutenance.

0 Présentation

Le but du projet est d'implémenter un circuit modélisant un processeur, qui pourra effectuer les instructions suivantes :

- ADD, SUB, AND, OR, XOR, SL, SR (et leurs variantes immédiates)
- ST, LD
- JMP, JEQU, JNEQ, JSUP, JINF
- CALL, RET

L'architecture générale du processeur est déjà partiellement construite dans le fichier https://perso.liris.cnrs.fr/tpierron/archi2025/projet_etu.circ. Votre objectif consiste à compléter chacun des sous-circuits rouges pour obtenir un processeur fonctionnel. Ceci nécessitera parfois de modifier le circuit global.

Le processeur a les caractéristiques suivantes :

- Mémoire 16×32 , i.e. qui stocke 2^{16} mots de 32 bits.
- 8 registres 16 bits de travail R_0, \dots, R_7 .
- Deux registres spéciaux : IR (32 bits) et PC (16 bits).
- Chaque cycle d'instruction est divisé en deux phases : Fetch et Exec (cf. cours). Chacune de ces phases dure un cycle d'horloge.

1 Composant Registres

Le composant **Registres** prend en entrée :

- Write (1 bit) : active l'écriture dans les registres
- IN (16 bits) : l'information à stocker dans un registre lorsque Write est à 1
- SR1, SR2, DR (3 bits chacun) : les adresses des registres source et destination
- Clock
- Reset

et a pour sortie le contenu des registres source et destination (16 bits chacun). Si Write est à 1, l'entrée IN doit en plus écraser le contenu du registre de destination.

2 Composant ALU

Le module UAL vise à effectuer une opération UAL. Il prend en entrée le code `op` d'une opération sur 3 bits, une entrée `Imm` valant 1 lors des opérations immédiates, et trois arguments `Input1`, `Input2` et `Cst` sur 16 bits. Il retourne en sortie le résultat de l'opération `op` entre `Input1` et `Input2` si `Imm` = 0, et entre `Input1` et `Cst` sinon.

Votre UAL doit implémenter les 7 opérations arithmétiques et logiques :

- addition et soustraction (ADD, SUB)
- et, ou, xor bit à bit (AND, OR, XOR)
- décalage à gauche/à droite (SL, SR)
- une huitième opération (intéressante) de votre choix

3 Decode IR

Les instructions sont encodées sur 32 bits selon un modèle similaire à celui proposé en cours :

- Bits 0-1 : 00 pour une instruction UAL, 01 pour MEM, 11 pour CTRL.
- Bits 2-4 : code de l'opération
- Bit 5 : 1 si opération immédiate
- Bits 6-8 : adresse du registre de destination
- Bits 9-11 : adresse du premier registre source
- Bits 12-14 : adresse du deuxième registre source
- Bits 16-31 : constante (sur 16 bits) dans le cas d'une instruction immédiate.

Le composant `DecodeIR` permet de décoder l'instruction stockée dans `RegIR`, en activant différentes sorties selon le type d'instruction. En parallèle de l'implémentation de ce composant, écrire un assembleur dans le langage de votre choix, c'est-à-dire un programme permettant de traduire un code assembleur (avec la syntaxe du cours) en un code machine (le contenu de la mémoire).

Attention : Dans Logisim, la mémoire stocke les instructions avec le bit 0 à droite (c'est-à-dire le symétrique de l'encodage vu en cours). Il faudra donc faire attention en remplaçant la mémoire avec vos programmes.

1. Quel comportement doivent avoir les composants `RegPC` et `GetAddr` dans le cas d'un programme sans instruction mémoire ni de contrôle ?
2. Implémenter ces composants dans ce cas.
3. Tester votre processeur avec des programmes qui n'utilisent pas encore les instructions mémoire ou de contrôle, afin de vérifier que l'UAL et `Registres` fonctionnent.

4 Instructions MEM

4. Quel est le rôle des composants `RegPC` et `GetAddr` ?
5. Pour implémenter les instructions MEM, il faut adapter le composant `GetAddr`. Que doit valoir sa sortie lors des phases `Fetch` et `Exec` ?
6. Proposer un encodage des instructions LD et ST. Implémenter `DecodeIR` en conséquence.

7. Implémenter l'instruction LD en modifiant le composant `GetAddr` (et éventuellement le reste du circuit principal si besoin).
8. Traiter ensuite le cas ST.
9. Tester avec un programme qui n'utilise pas d'instruction de contrôle.

5 Sauts

10. Choisir (intelligemment¹) un encodage et actualiser `DecodeIR` et votre assembleur en conséquence.
11. Pour commencer, on considère uniquement l'instruction JMP. Implémenter le traitement de cette instruction dans `RegPC`.
12. Gérer ensuite les instructions JEQ, JNEQ, JSUP et JINF. On pourra ajouter un sous-circuit permettant de tester les conditions recherchées.
13. Tester avec un programme sans CALL ni RET.

6 Appel de fonction

Il reste à implémenter les instructions CALL et RET.

14. Créer un composant `Pile` contenant une mémoire sur laquelle on peut empiler ou dépiler des informations. La sortie de ce composant sera toujours le contenu en haut de la pile.
15. En utilisant `Pile`, modifier `RegPC` pour implémenter CALL et RET.
16. Tester

1. Réfléchissez à comment vous allez implémenter chaque opération !