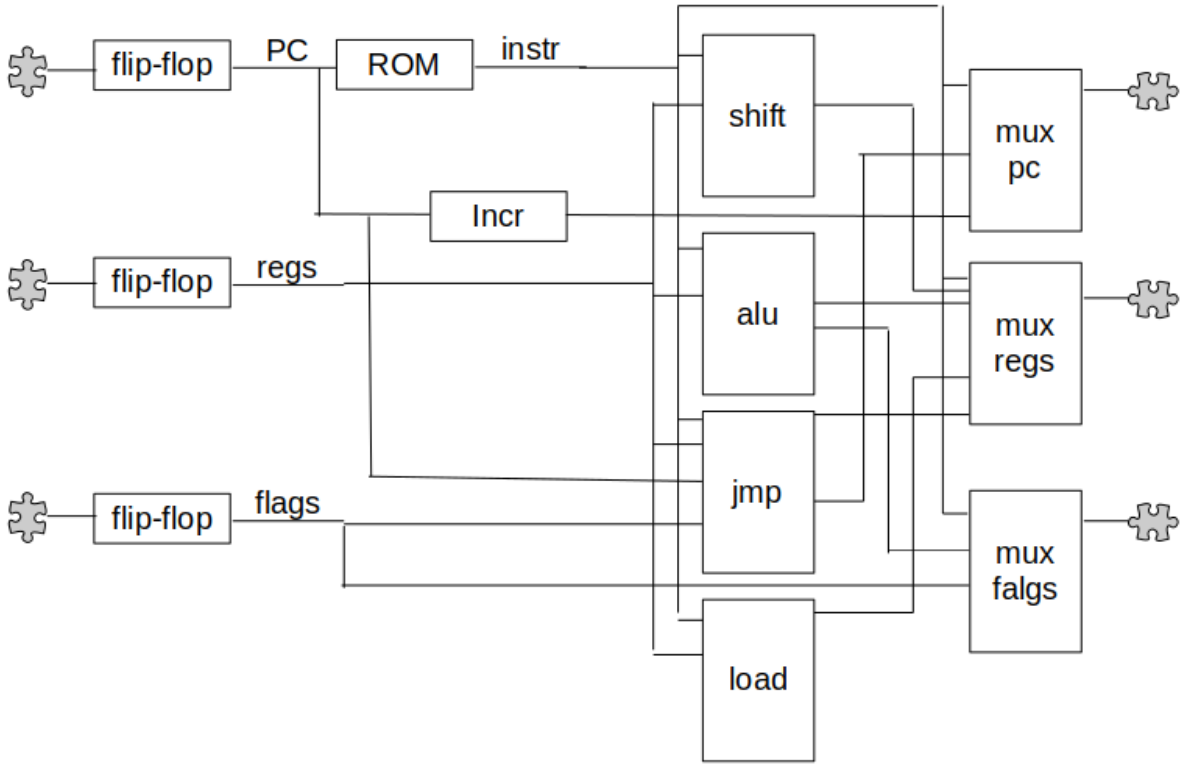
TODO : Rendre l'affichage des fonctions plus beaux, j'ai créé une commande pour que ce soit facilement modifiable.

Fonctionnement du main

Le fichier main.py s'occupe de définir le pc, les registres, les flags, et la ROM. À chaque cycle, il récupère toutes les anciennes valeurs grâce à des REG, fait appel aux différents modules et sélectionne la valeur souhaitée parmi les résultats renvoyés. Les différents modules appelés sont :

— l'alu
— les décalages de bits
— les opérations sur la RAM
— les sauts dans le programme

Le circuit logique simplifié correspondant au main est donc :



L'ALU

Le coeur du CPUlm.

Fonctions auxiliaires

Cette partie utilise les fonctions auxilliaires suivantes (toutes dans 'alu.ply') :

—

big_adder :Entrée: $(b_i)_{i<=n}$

Sortie: (s, c) correspondant à $\sum_{i=1}^{n} b_i$

Notes:

—

mul :Entrée: ab deux bus de même taille
Sortie: (s, c) correspondant au produit
Notes: Peut se désactiver avec le flag WITH_MUL pour plus de performances.

—

or_n_bits :Entrée: $(b_i)_{i<=n}$
Sortie: $\vee_{i=1}^{n} b_i$
Notes:

—

div :Entrée: $(a_i)_{i<=n}$ dividende,$(b_i)_{i<=n}$ diviseur
Sortie: q quotient
Notes:

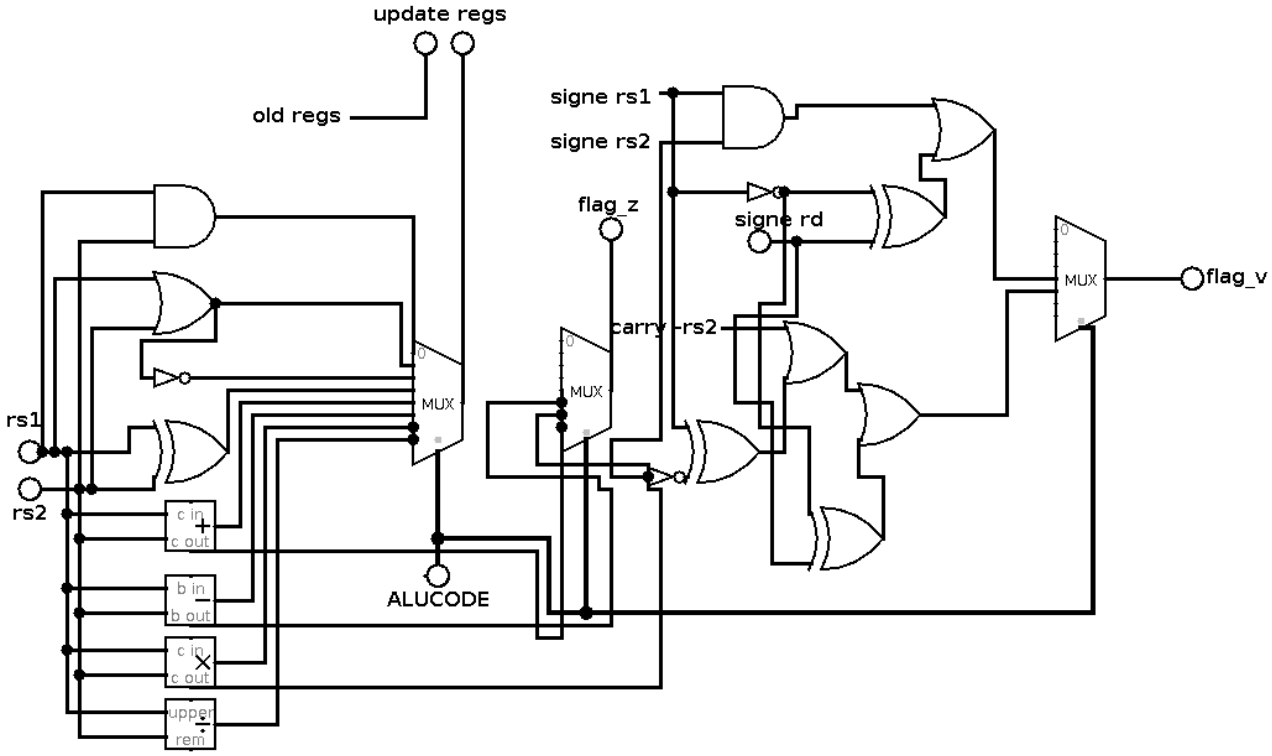
Fonctionnement bref

L'ALU prend deux arguments, l'instruction et les registres du cycle précédent.
L'instruction est passée selon cette syntaxe :

0 :4	4 :9	9 :14	14 :19	19 :24
OP CODE	RD	RS1	RS2	ALUCODE

RD,RS1 et RS2 sont utilisés par les différentes instructions disponibles, les détails sont sur la documentation du processeur. Le code ALU est unique pour chaque instruction et est aussi disponible sur la doc du processeur. L'ALUCODE est utilisé dans un mux qui prend les différents calculs possibles en argument. rd (regsitre de destination) est modifié par un appel à update_regs (cf tools.py). Enfin les flags sont mis à jour et la fonction renvoie les anciens flags et les flags mis à jour (c'est le main qui s'occupe de mettre à jour les regs).

Circuit de l'ALU



Les Jumps

Todo

Les Shifts

Todo