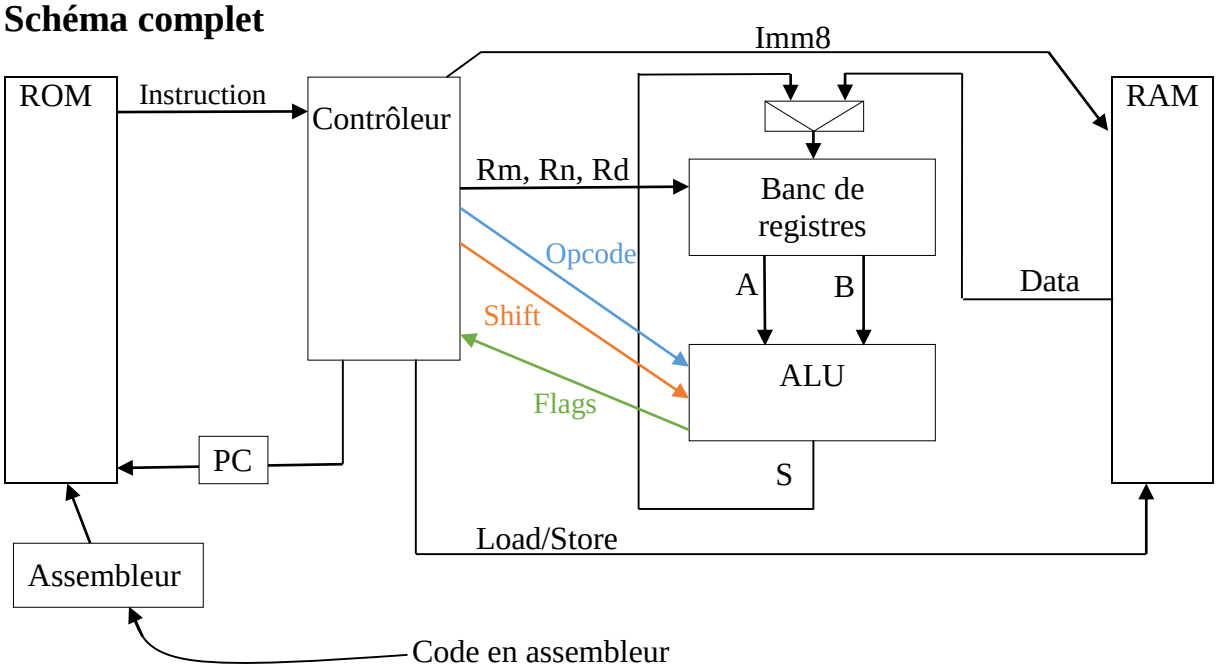
Révision : Processeur ARM Cortex-M

Schéma complet



Taches de réalisation du projet

1. ALU : Réaliser les blocs d'opérateurs arithmétiques et logiques, et générer les flags.
2. Contrôleur : Lire les instructions depuis la mémoire de programme (ROM) et décoder les instructions, puis générer les signaux de commande du chemin de données. Calcule également l'adresse de la prochaine instruction.
3. Cheminement de données : Mouvement de registre à registre, lecture et écriture en mémoire de données, envoie d'adresse pour les lectures/écritures.
4. Assembleur : Parser un fichier assembleur, générer le fichier binaire à charger dans la mémoire d'instruction de logisim.
5. FPGA : Tester le déploiement du processeur sur FPGA.

Cycle d'exécution machine

1. Charger l'instruction dans le contrôleur depuis la ROM
2. Charger les données à l'aide du banc de registres
3. Faire un traitement sur ces données en fonction de l'opération voulue (à l'aide de l'ALU sauf pour « Load » et « Store »).
4. Ranger les résultats du traitement dans le banc de registres
5. Désigner la prochaine instruction en incrémentant le PC.

Boucle

Instructions ARM Thum 16-bit type

LDR Rt, Imm8	**Load Register Immediate :** Permet de charger (load) une valeur immédiate (contenue à l'adresse imm8 dans la RAM) vers le registre Rt.
EOR Rdn, Rm	**Exclusive Or Register :** Effectue un « ou exclusif » entre la valeur contenue dans le registre Rn (Rdn) et la valeur du registre Rm, et stocke le résultat dans le registre Rd (Rdn).
ADD Rd, Rn, Rm	**Add Register :** Permet d'ajouter la valeur contenue dans le registre Rn et le registre Rm, et de stocker le résultat dans le registre Rd.
B<c><label>	**Conditional branch :** Permet de « sauter » vers le *label* si la condition *c* est valide (par rapport à la ligne qui précède le branch).

Découpage des instructions

LDR			Total
10011	Rt	Imm8	**16 bits**
CODOP, 5 bits	**Sur 3 bits**	**Sur 8 bits**	**16 bits**

EOR				Total
010000	0001	Rm	Rt	**16 bits**
CODOP, 6 bits	**Code ALU, 4 bits**	**Sur 3 bits**	**Sur 3 bits**	**16 bits**

ADD					Total
00xxxx	0 (1 pour sub)	Rm	Rn	Rd	**16 bits**
CODOP, 6 bits	**1 bits**	**Sur 3 bits**	**Sur 3 bits**	**Sur 3 bits**	**16 bits**

B			Total
1101	c	label	**16 bits**
CODOP, 4 bits	**Condition, 4 bits**	**Sur 8 bits**	**16 bits**

Exécuter un code assembleur

```
.global _start
                    .global main
_start:             b       main

main:

    LDR R0, cnt
    LDR R1, cnt
    LDR R2, var1
    LDR R3, var2

loop :
    ADC R1, R0
    SUB R3, R3, R2
    Bne loop
    STR R1, cnt
    STR R3, rem

// Allocation et contenu des variables
var2:   .word 0x00186A0
var1:   .word 0x000000A
cnt :   .word 0x0000000
rem :   .word 0x0000000

.end
```

Symbole	Adresse mémoire en décimal
Main	0
Loop	8
Var2	120
Var1	128
cnt	132
rem	136

LDR R0, cnt	132 = 10000100 (binaire) 10011 000 10000100 = **9884** (hexa)
LDR R1, cnt	10011 001 10000100 = **9984**
LDR R2, var1	128 = 10000000 10011 010 10000000 = **9A80**
LDR R3, var2	120 = 1111000 10011 011 01111000 = **9B78**

ADC R1, R0	010000 0101 000 001 = **4141**
SUB R3, R3, R2	000000 1 010 011 011 = **29B**
Bne loop	1101 0001 00001000 = **D108**
STR R1, cnt	10010 001 10000100 = **9184**
STR R3, rem	136 = 10001000 10010 011 10001000= **9388**

Fonctionnement additionneur

Où C_i est la retenu :

A	**B**	**C_i**	**S_i**	**S**
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

$$S_i = \overline{A}BC_i + A\overline{B}C_i + AB\overline{C_i} + ABC_i$$
$$= C_i(\overline{A}B + A\overline{B}) + ABC_i + AB\overline{C_i}$$
$$= C_i(A \oplus B) + AB(\overline{C_i} + C_i)$$
$$= C_i(A \oplus B) + AB$$

$$S = \overline{A}\overline{B}C_i + \overline{A}B\overline{C_i} + A\overline{B}\overline{C_i} + ABC_i$$
$$= C_i(\overline{A}\overline{B} + AB) + \overline{C_i}(\overline{A}B + A\overline{B})$$
$$= C_i(\overline{A \oplus B}) + \overline{C_i}(A \oplus B)$$
$$= C_i \oplus A \oplus B$$

Dans l'exemple précédent le registre R0 n'est jamais mis à jour (et vaut 0), le « ADC R1, R0 » va donc ajouter dans R1, R1 + R0 + 1 (la retenue), soit l'opération suivante R1 = R1 + 0 +1. Il s'agit d'une incrémentation équivalente à R1++ en java.

Le programme précédent a d'ailleurs pour effet de boucler 10 000 fois dans loop (avec R3 = 100 000 (en décimal), R2 =10 (en décimal)). *cnt* vaut 10 000 à la fin alors que *rem* vaut 0. *rem* a pour intérêt de vérifier que la boucle s'est bien terminée entièrement (doit valoir 0 à la fin), correspond aux itérations restantes.
Le programme se traduirait de cette manière :

```
R0 = 0;
R1=0; R2=10;
R3=100 000;
TANTQUE (R3=R3-R2) != 0
        R1++;
FINTANTQUE
cnt = R1; //Soit 10 000
rem = R3; //Soit 0
```