Université Assane Seck de Ziguinchor

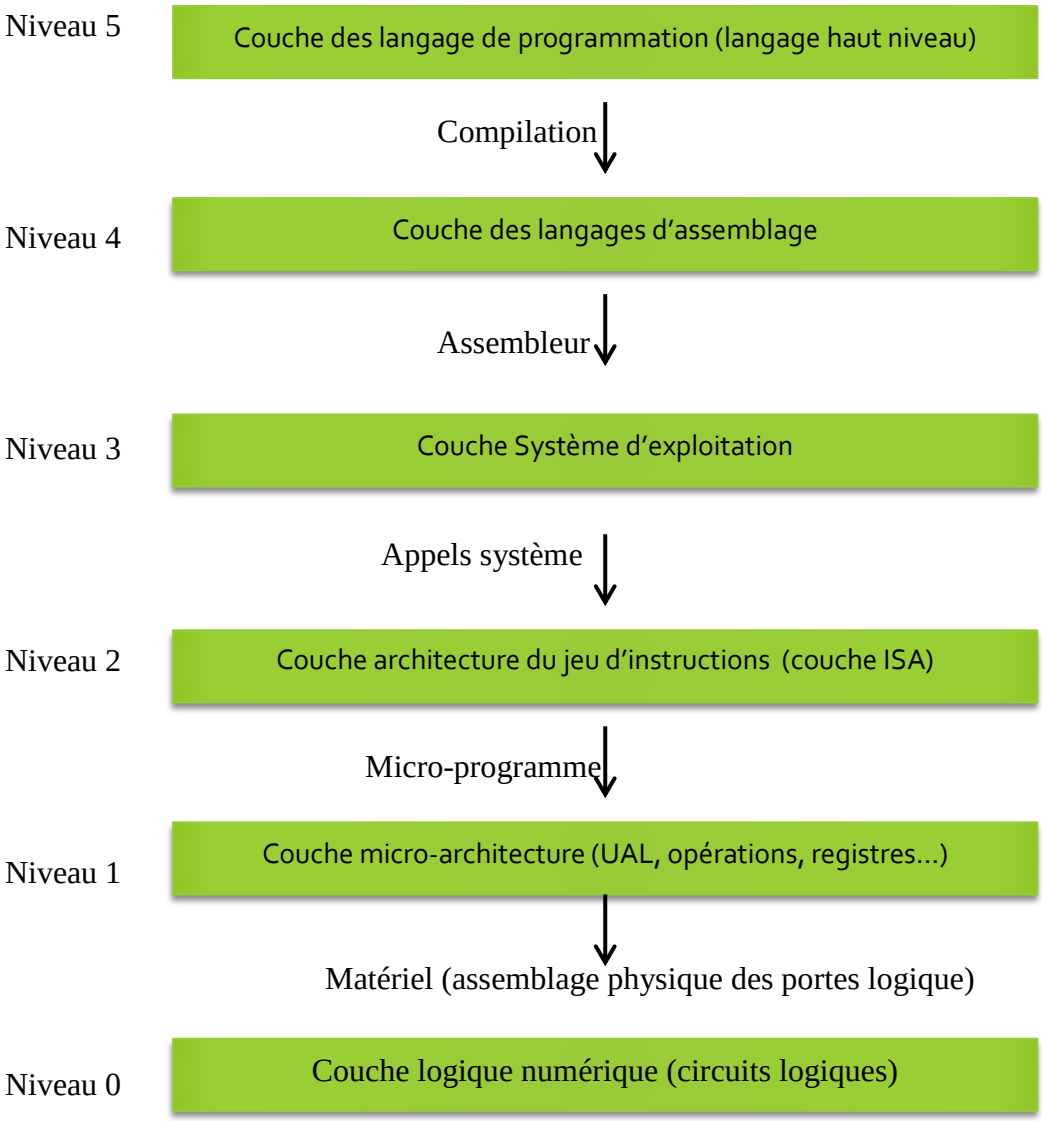
Système d'Exploitation

CHEMIN DES DONNEES

SOMMAIRE

I. INTRODUCTION

Le choix d'organisation de données forment ce que l'ont appelle la **microarchitecture** de processeur. Le contrôle de l'exécution d'une instruction se fait par une sequence de commandes appellees micro-commandes ou micro-instructions. La couche **architecture** ou **ISA** (Instruction Set Architecture) s'occupe du jeu d'instruction alors que la couche microarchitecture s'occupe de l'UAL, des registres, des données manipulées, des modes d'adressage, du modèle d'organisation de la mémoire, du contrôleur etc… On peut distinguer une architecture en six couches ou niveaux:



Il faut noter que la couche ISA n'a pas de vision sur l'implémentation des instruction.

I. TYPE D'ARCHITECTURE SELON LES REGISTRE

La classification des architectures est base sur le jeu d'instructions:

- Architecture à pile: les operands sont sur le haut de la pile;
- Architecture à accumulation: une opérande est dans l'accumulateur
- Architecture à registres:
    - Architecture mémoire(registre)-mémoire: une operands est en mémoire
    - Architecture à registre-registre: toutes les operandes sont dans les registres (excepté pour le load/store)

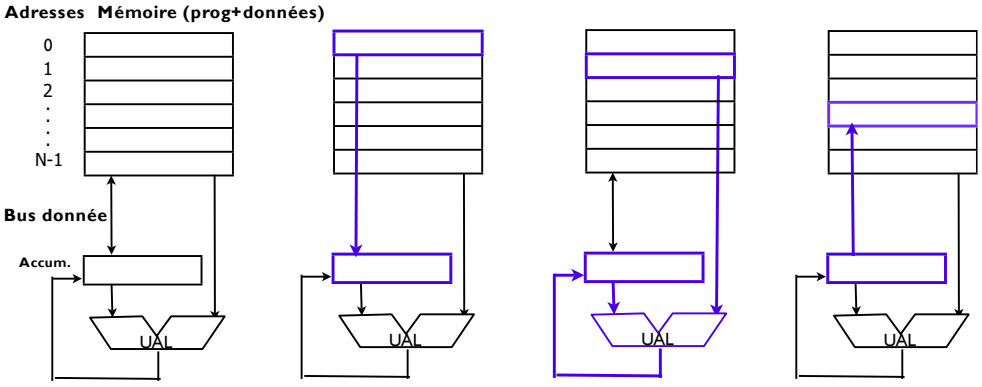
Toute architecture (ou modèle d'exécution) est caractérisé par un couple (n, m) où $m \leq n$; avec n= nombre total d'opérandes **spécifiés** par une instruction UAL ; et m = nombre d'opérandes **mémoire** autorisés dans une instruction UAL.

Illustration de l'architecture à pile ou mémoire–pile : l'instruction UAL ne spécifie aucun opérande, et fait implicitement référence au sommet d'une pile. C'est l'architecture typique des premiers microprocesseur 8 bits (Intel 8080 ou Motorola 6800)

-Chargement du premier opérande dans l'accumulutateur

-Addition du deuxème opérande avec l'accumulateur dans lequel est range le résultat.

-rangement du résultat en mémoire.



C=A+B

LOAD @A
ADD @B
STORE @C

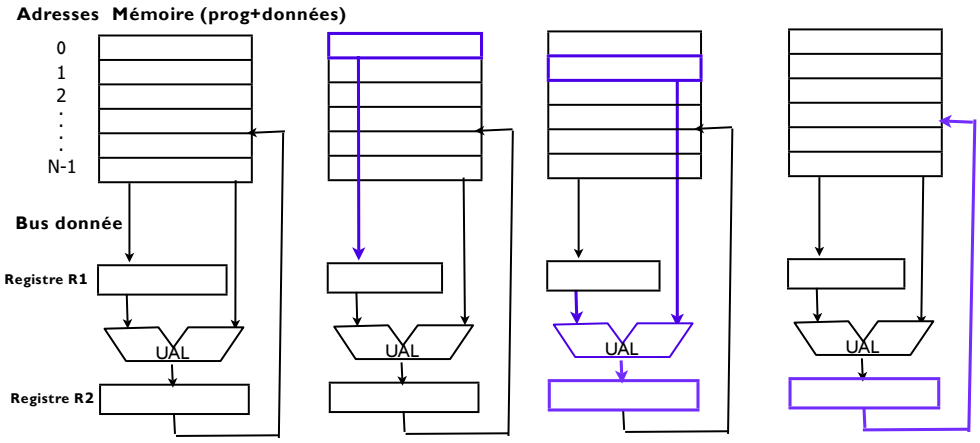
LOAD @A
ADD @B
STORE @C

LOAD @A
ADD @B
STORE @C

LOAD @A
ADD @B
STORE @C

Illustration de l'architecture à accumuluteur ou mémoire–accumulateur : l'instruction UAL ne spécifie qu'un opérande ; les autres sont implicites, contenus dans l'accumulateur.

Illustration de l'architecture registres- mémoire (2,1) : l'accumulateur est remplacé par un ensemble de régistres généraux.



C=A+B

LOAD R1, @A;
ADD R1, @B;
STORE R1, @C;

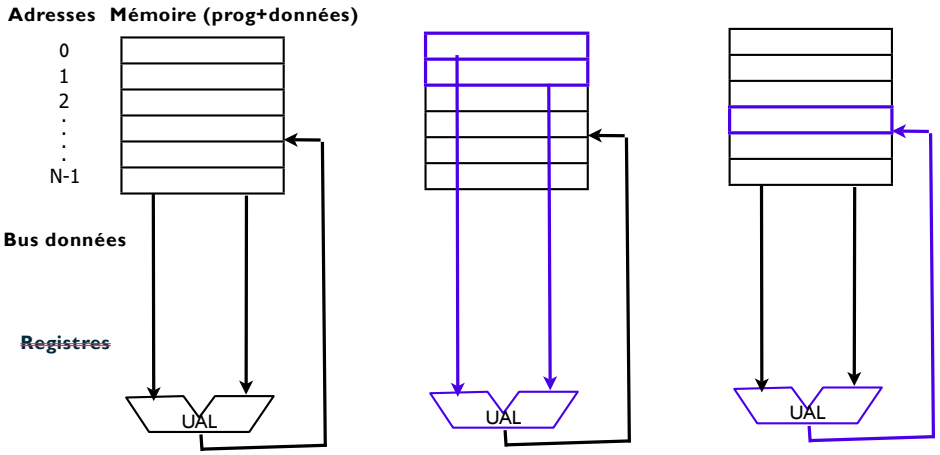
LOAD R1 @A
ADD R2 R1 @B
STORE R2, @C

LOAD R1 @A
ADD R2 R1 @B
STORE R2, @C

LOAD R1 @A
ADD R2 R1 @B
STORE R2, @C

Il existe aussi une variante où (2,0) où les deux opérandes sources sont dans les registres et le résultat dans un des deux registres sources.

Illustration de l'architecture mémoire–mémoire (3,3): une instruction peut spécifier les deux operandes sources et le résultat, et chaque opérande peut être situé en mémoire. Il est caractéristique du VAX 11 de Digital.



C=A+B

ADD @C @A @B

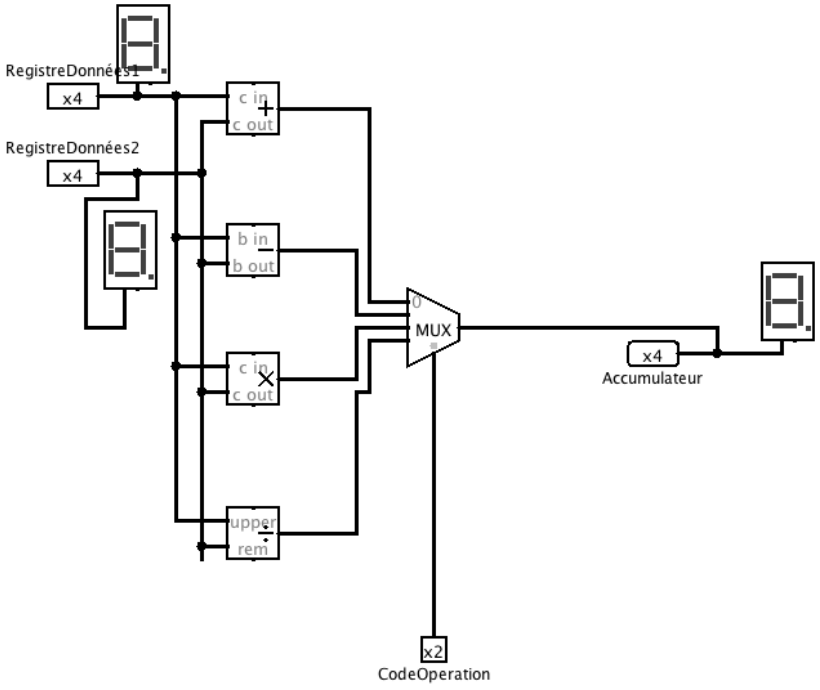
ADD @C @A @B

ADD @C @A @B

Illustration de l'architecture registre-régistre: l'instruction UAL ne spécifie aucun opérande en mémoire, et ne fait référence qu'à des registres.

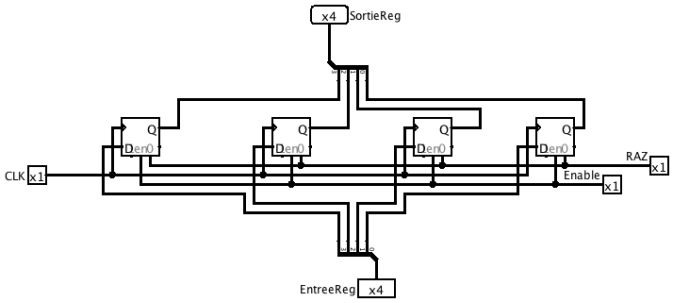
II. UAL

L'UAL comme son nom l'indique permet d'exécuter les opérations arithmétiques et logiques. Notre microprocesseur pourra réaliser les quatre opérations arithmétiques de base (addition, soustraction, multiplication et division qui seront implémentées à partir de circuits logiques. On peut utiliser un multiplexeur simple avec 4 bits de données et 2 bits d'adresse (ou de sélection) qui permettront de sélectionner l'entrée de données à rediriger vers la sortie du MUX. Les sorties des circuits logiques feront office d'entrées du multiplexeur et les entrées d'adresse du multiplexeur permettront de dire l'opération à effectuer (00=Addition, 01=Soustraction, 10=Multiplication et 11=Division)



III. LE BANC DE REGISTRE

Un banc de registre est composé de registres. Dans notre cas, nous utilisons un banc de 4 registres de 4 bits. A l'aide du schéma suivant, on réalise facilement un registre de 4 bits avec des bascules D.

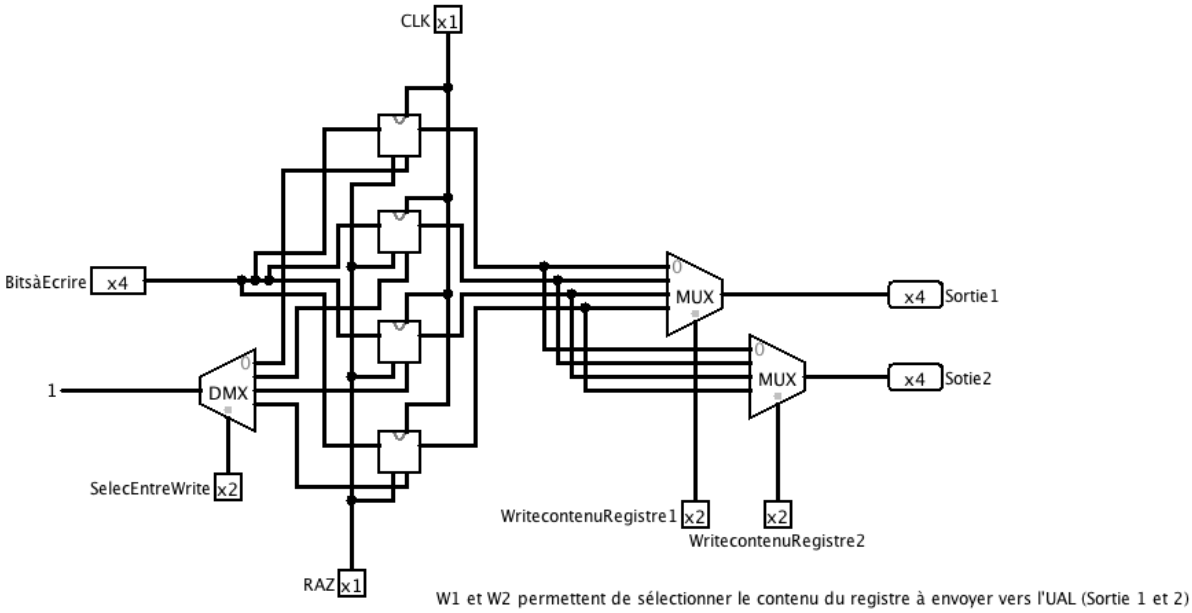


L'entrée RAZ à 1, remet les sorties des bascules à 0

L'entrée Enable si elle existe doit être à 1 pour que les entrés soit enregistrées sur les sorties sous l'effet de l'horloge.

L'entrée EntrReg permet de charger le régistre, chaque bit passera vers l'entrée d'une bascule.

Notre banc de registre est composé de quatre registres de 4 bits chacun comme le montre le schéma suivant.



_Toutes les entrées EntrReg des bascule sont connectése à une unique entrée BitsàEcrire, et à l'aide d'un DEMUX dont les sorties sont branchées sur les entrées Enable des regitres (aussi des bascules au sein de chaque registre), on peut facilement charger l'entree BitàEcire et choisir (à partir SelecEntreWrite) sur quel registre **écrire : c'est le principe d'écriture.**

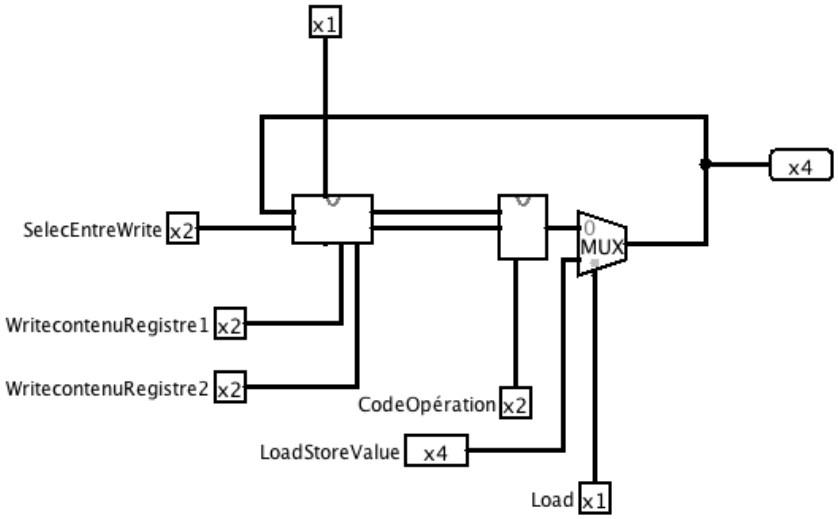
Le principe de lecture est simple, chaque registre diffuse sa valeur à chaque coup d'horloge, cependant, seule la valeur souhaitée sort du banc de registre au moyen d'un multiplexeur. Ici, on exploite deux multiplexeurs pour sélectionner deux registres.

Ainsi les 2 bits WriteContenuRegistre1 et ceux de WriteContenuRegistre2 peuvent servir de

codes dans nos instructions machine pour écrire dans les registres.

III. L'UAL ET BANC DE REGISTRE INTERCONNECTES

Une interconnexion de l'UAL et du banc de registres comme étape intermédiaire permettra de comprendre plus facilement la composition de l'UCT.



-SelectEntréWrite permet de sélectionner dans quel registre écrire la valeur "SortieUAL/LoadStoreValue"

-WritecontenuRegostre1 permet de sélectionner la première opérande en entrée de l'UAL (sortie 1 du banc de registre)

--WritecontenuRegostre1 permet de sélectionner la deuxième opérande en entrée de l'UAL (sortie 2 du banc de registre).

-CodeOpréation determine l'opération à faire (+, -, x, ÷), il doit être partie intégrante de nos instructions (partie code instruction).

-LoadStoreValue: valeur à charger, elle permettra de charger des valeurs et pourra figurer dans nos instruction comme opérande

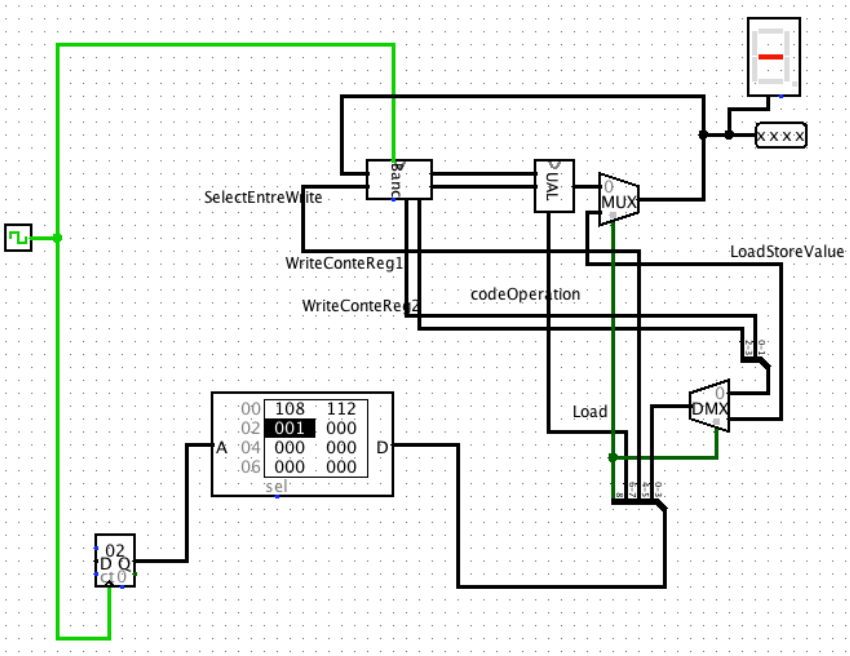
-Load à 0: permet de laisser passer le résultat issu de l'opération exécutée par l'UAL dans SortieUAL/LoadStoreValue".

-Load à 1: permet de laisser passer la valeur à charger LoadStoreValu dans dans SortieUAL/LoadStoreValue"

-Le MUX permet de choisir entre le result de l'UAL et LoadStoreValue, laquelle à charger dans SortieUAL/LoadStoreValue

III. L'UCT

Notre UCT est compsée de l'UAL, du banc de registres et de la mémoire que contient le programme à executer comme on peut le vois sur la figure suivante.



La mémoire est adressée selon la taille des instructions.

Format des instructions

Nous distinguons 4 instructions arithmétiques (+, -, x, ÷) et une instruction de chargement LDR (Load).

- Load sur 1 bit et opcode (2 bits), ainsi on a comme code instruction:
- Addition 00 et Load doit être à 0 = 000
- Soustraction 01 et Load doit être à 0 = 001
- Multiplication 10 et Load doit être à 0 = 010

- Division 11 et Load doit être à 0 = 110
- On a une instruction de chargement LDR où Load doit être à 1=100
- Le registre résultat qui stocke les résultats sur 2 bits parce que le banc est compose de 4 registre, donc 2 bits pour coder les 4.
- Une constants sur 4 bits à charger ce qui nécessite load à 1 pour être traité).
- Exemple d'instruction en binaire pour additionner les contenus des régistres 1 et 2, et d'écrire le résultat dans le registre 1.

  En "Langage Assembeur": ADD R1 R1 R2.// **ADD** (additionner) **R1**(ranger le résultat dans R1 désigné ici par le code de R1) **R1**(contenu de R1 comme opérande 1 désigné ici par le code de R1) **R2** (contenu de R2 comme opérande 2, désigné ici par le code de R2)

  En Langage machine: 000 00 00 01.

  Il faut noter ici le code operation sur 3 bits, dont le bit à gauche (qui est à 0) associé au code de l'opération permet d'inhiber le chargement (pour une operation +, -, x, ÷) et de l'activiter pour une operation de chargement (LDR) quand il est à 1.

  En hexadécimal: 001_{16}

  Avant de pouvoir effectuer des operations arithmétiques sur les contenus des régistres, il faut les charger comme le montre l'exemple ci-après du chargement de R1.

  Exemple d'instruction de chargement en **"Langage Assembeur": LDR R1 4** // **LDR** (charger) **R1** (ranger la valeur à charger dans R1 désigné ici par le code de R1) **4** (valeur à charger dans R1 comme la seule opérande code sur 4 bits. Les 4bits viennent du fait qu'avec deux operandes on devait les coder sur 4 bits).

  En Langage machine: 100 00 0100.

  Il faut noter ici le code operation sur 3 bits, dont le bit à gauche (qui est à 1) associé au code de l'opération permet d'activer le chargement pour l'operation LDR et de l'inhiber pour les operations (+, -, x, ÷) quand il est à 0.

  En hexadécimal: 104_{16}

  Ainsi les étapes pour faire une opération arithmétique (+, -, x, ÷) entre deux valeurs (8 et 2 par exemple) sont:

  1-Charger l'opérande1(8) dans le régitre R1

  2-Charger l'opérande2 (2) dans le Régistre R2

  3- Additionner, soustraire, multiplier ou diviser les opérande 1 et opérande 2.

  En "Assembleur":

  1-LDR R1 8_{10} //R1 =8_{10}

2-LDR R2 2_{10} //R2=2_{10}

3-ADD R1 R1 R2 //R1=10_{10} (result addition qui écrase le contenu initial 2); R2=2

En langage machine:

1-100 00 1000

2-100 01 0010

3-000 00 00 01

En hexadécimal:

1-108_{16}

2-112_{16}

3-001_{16}

Puisque le chargement en mémoire se fait en hexadicimale, l'image à charger en mémoire est: 108 112 001.

Il faut note ici que les instruction sont codes sur 9 bits, ce qui nous a permis de choisir un adressage sur 9 bits (i.e un ensemble de 9 bits possède une seule adresse).