# MINI-PROJET VHDL 2012-2013

## Description d'un calculateur de Von Neumann

Ecole polytechnique de l'université de Nice-Sophia Antipolis

## *Avant-propos*

*Le mini-projet qui vous est proposé de réaliser vise comme objectif essentiel de faire la synthèse de toutes vos connaissances accumulées dans le module d'enseignement VHDL. Il permet également de parfaire des connaissances dispensées dans d'autres modules en proposant une description d'une architecture dont les principes ont pu être étudiés dans ces modules. Enfin, ce mini-projet est destiné à vous permettre de développer une première expérience pratique de la modélisation VHDL, dans des conditions proches de celles d'un fonctionnement en entreprise.*

# 1. Introduction

Les circuits non programmables déterminent les opérations à exécuter sur la base uniquement de ces entrées de contrôle et de bits d'états du chemin de données. Un tel circuit n'exécute donc qu'une seule opération définie (ex : multiplication par additions et décalages, affichage d'une donnée décimale, etc). Par opposition, un microprocesseur est responsable de l'extraction d'instructions depuis la mémoire et du séquencement de l'exécution de ces instructions (calcul de l'adresse suivante de l'instruction à exécuter). Dans ce type d'architecture, le microprocesseur est composé d'un chemin de données dont les opérations sont pilotées par une unité de contrôle. La tâche de cette unité de contrôle consiste à extraire, décoder et exécuter les instructions. Un calculateur bâtit sur le modèle de **Von Neuman** possède la faculté de traiter l'ensemble de ces opérations en un ou plusieurs cycle d'horloge élémentaires. Contrairement à une architecture de Harvard, les données et instructions peuvent être stockées dans un composant mémoire unique. L'interconnexion des différents composants au travers de signaux permettant de propager les signaux de données ainsi que signaux de contrôle définit le modèle interne du microprocesseur. L'objectif du projet est de décrire le comportement des différents composants de ce modèle interne, modèle qui sera vérifié par simulation VHDL sous ModelSim.

# 2. Modèle de programmation du calculateur

Le programmeur d'un calculateur doit spécifier les instructions à exécuter à l'aide d'un *programme*. Un programme peut donc se définir comme une liste d'instructions spécifiant les opérations (code opération) et leurs opérandes. Il peut également s'agir d'instructions de contrôle du déroulement du programme indiquant des branchements conditionnels ou inconditionnels à effectuer. Le registre compteur ordinal (PC) permet de conserver en mémoire l'adresse de la prochaine instruction à exécuter.

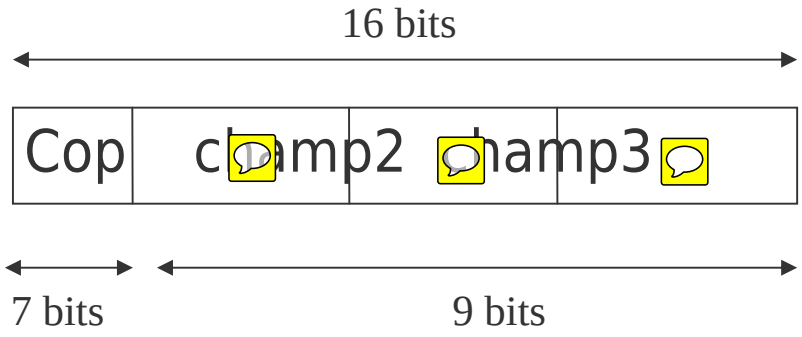


**Figure 1: Format générale d'une instruction**

Les instructions sont stockées dans une mémoire d'instructions tandis que les opérandes peuvent être stockées dans différents emplacements :

- un champ de l'instruction
- un registre dans un fichier de registres
- un emplacement mémoire de données

Par conséquent, outre la nature de l'opération à exécuter, le code opération doit préciser les modalités d'exécution de l'instruction et préciser l'emplacement des opérandes à utiliser. On parle de format d'instruction ou encore de méthode d'adressage (Figure 1). Le format des instructions qui nous intéresse ici est un format fixe (toutes les instructions sont codées sur un

nombre de bits identique). Le format utilisé dans le cadre de notre calculateur fait apparaître l'utilisation de 3 champs adresse/opérande. On distingue par ailleurs 3 formats d'instruction différents:

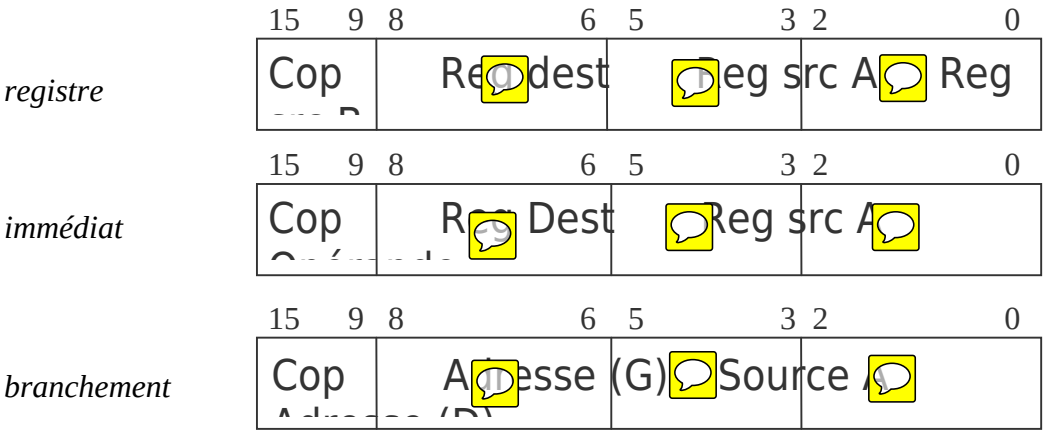


**Figure 2: Les différents formats d'instructions**

## A. Format registre

Dans le cas d'un format registre, les 3 champs fournissent un n° de registre à utiliser comme emplacement des opérandes source A,B ou comme emplacement destinataire du résultat. Les champs étant codés sur 3 bits, on peut désigné jusqu'à 8 registres ($2^3$). Un des champs source peut s'avérer non significatif lorsque l'instruction n'a besoin que d'un seul opérande. Par exemple, l'instruction codée

```
0001011 011 101 XXX        code machine
  NOT    R3  R5            assembleur
```

définit une instruction de complément à 1 (NOT) du registre n°5 (Reg src A), résultat dans le registre n°3 (destination). La valeur des bits du champ B n'est pas significative. On notera que certaines opérations utilisent le champ B au lieu du champ A. Le champ registre destinataire peut également ne pas s'avérer significatif lorsqu'il s'agit d'un transfert vers la mémoire, un des registres stockant l'adresse mémoire pour le transfert. Par exemple, l'instruction codée

```
0100000 XXX 100 101      code machine
  ST          R4,R5      assembleur
```

définit une demande de transfert de la valeur de R5 à l'emplacement mémoire d'adresse contenue dans R4.

## B. Format immédiat

Dans le cas d'un format immédiat, un des opérandes est implicitement stocké dans l'instruction, dans le champ de droite en l'occurrence ici. Il s'agit donc d'une constante dont la valeur se situera dans l'intervalle 0 à 7. Par exemple,

```
1000010 010 111 011      code machine
  ADI     R2  R7   3     assembleur
```

spécifie une opération d'addition de R7 avec la constante 3, avec résultat de l'opération dans R2.

## C. Format branchement

Par opposition aux deux formats précédents, le dernier format d'instruction n'engendre aucune modification de registre ou de mémoire de données. L'instruction est destinée à modifier (éventuellement sous condition) l'ordre d'extraction des instructions de la mémoire, séquentiel par défaut On parle encore d'instructions de rupture de séquence. Pour cela, le champ d'adresse est séparé en deux parties , avec les 3 bits de poids forts situés dans le champ *adresse(G)* et les trois bits de poids faibles situés dans le champ *adresse (D)*. La méthode d'adressage est appelée *adressage relatif au PC*, les 6 bits d'adresses formant le déplacement à ajouter au PC. Ce déplacement est considéré comme signé (représentation en complément à 2). Cela permet de se déplacer dans le sens croissant ou décroissant des adresses. Par exemple,

```
1100000 101 110 100
  BRZ         R6,-20
```

spécifie un branchement à l'adresse PC – 20(101100), si R6=0.

| Instruction | Opcode | Mnemo | Format | Description | Indic. |
|---|---|---|---|---|---|
| Move A | 0000000 | MOVA | RD,RA | R[DR]← R[SA] | N,Z |
| Incrément | 0000001 | INC | RD,RA | R[Dr]← R[SA]+1 | N,Z |
| Add | 0000010 | ADD | RD,RA,RB | R[DR]← R[SA]+R[SB] | N,Z |
| Substract | 0000101 | SUB | RD,RA,RB | R[DR]← R[SA]-R[SB] | N,Z |
| Decrement | 0000110 | DEC | RD,RA | R[DR]← R[SA]-1 | N,Z |
| AND | 0001000 | AND | RD,RA,RB | R[DR]← R[SA] ^ R[SB] | N,Z |
| OR | 0001001 | OR | RD,RA,RB | R[DR] ← R[SA] ∨ R[SB] | N,Z |
| Exclusive OR | 0001010 | XOR | RD,RA,RB | R[DR] ← R[SA]⊕R[SB] | N,Z |
| NOT | 0001011 | NOT | RD,RA | R[DR] ← ¬R[SA] | N,Z |
| Move B | 0001100 | MOVB | RD,RB | R[DR] ← R[SB] | |
| Shift Right | 0001101 | SHR | RD,RB | R[DR] ← sr R[SB] | |
| Shift Left | 0001110 | SHL | RD,RB | R[DR] ← sl R(SB] | |
| Load Immediat | 1001100 | LDI | RD,OP | R[DR] ← Zf OP | |
| Add Immediat | 1000010 | ADI | RD,RA,OP | R[DR] ← R[SA]+zf OP | |
| Load | 0010000 | LD | RD,RA | R[DR] ← M[SA] | |
| Store | 0100000 | ST | RA,RB | M[SA] ← R[SB] | |
| Branch on zero | 1100000 | BRZ | RA,AD | R[SA]=0 : PC ← PC +se AD | |
| Branch on neg | 1100001 | BRN | RA,AD | R[SA]<0 : PC ← PC +se AD | |
| Jump | 1110000 | JMP | RA | PC ← R[SA] | |

**Figure 3: Le jeu d'instructions du calculateur**

La table de la Figure 3 liste les différentes catégories d'instruction. Une description de type transfert de registre explique le mode opératoire de chacune des instructions. La colonne de droite précise si les indicateurs N (pour négatif) et Z (pour zéro) sont affectés par les instructions.

# 3. Architecture de Von Neumann

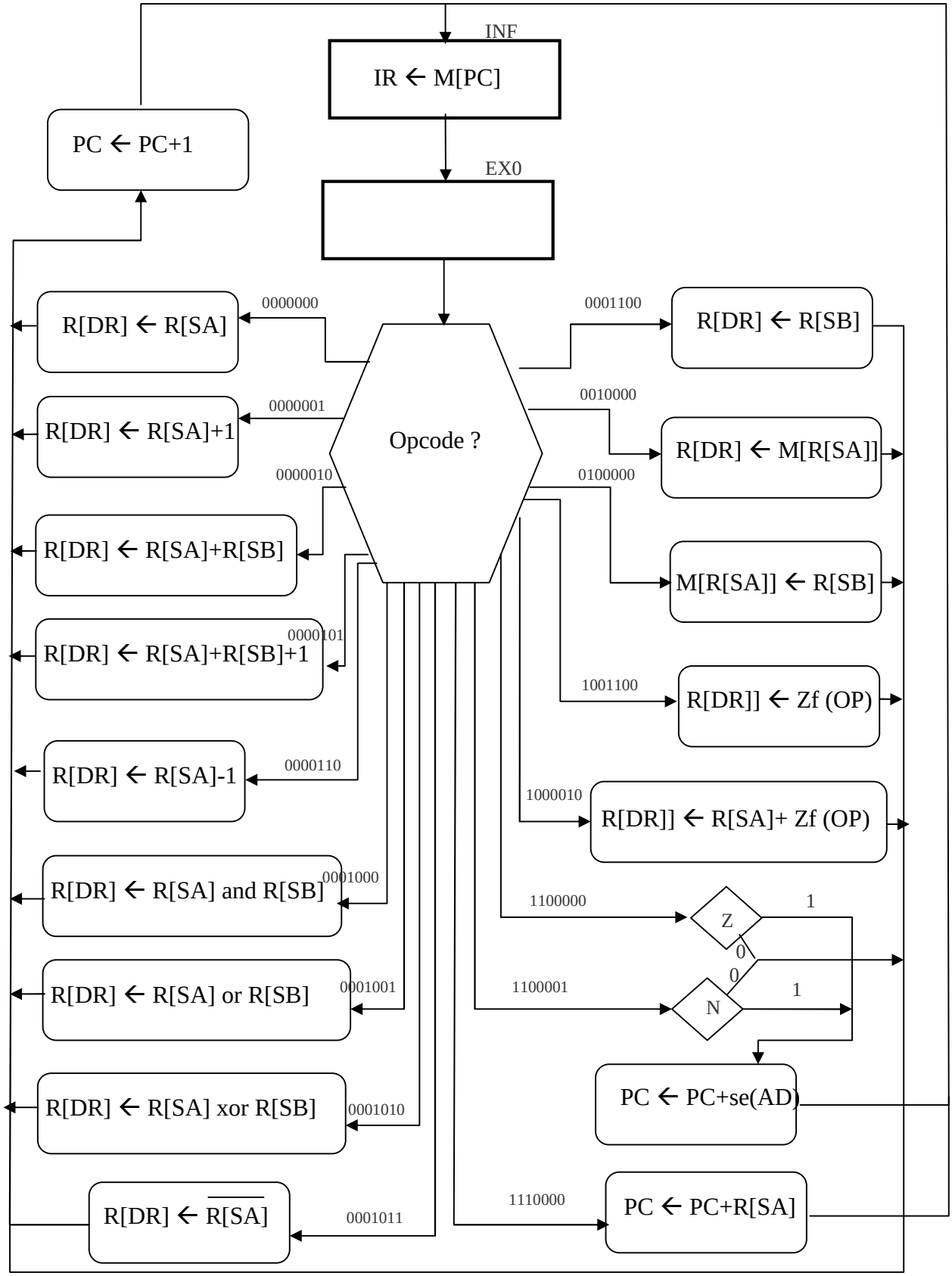


**Figure 4: Algorithmic Stete Machine du calculateur**

On s'intéresse à présent à l'architecture du calculateur permettant l'exécution des instructions définit précédemment. L'Algorithmic State Machine ci-dessous permet de caractériser le traitement des instructions effectué par le calculateur en termes de transferts de registres. Pour les instructions nécessitant un accès mémoire de même que pour l'instruction elle-même, au moins 2 cycles d'horloge sont nécessaires. Ces 2 cycles sont classiquement définis comme les étapes de :

Instruction Fetch : Extraction de l'instruction depuis la mémoire de données
Instruction execution : Exécution de cette instruction sur 1 ou plusieurs cycles d'horloge.

L'extraction de l'instruction se déroule dans le cycle d'horloge correspondant à l'état INF. Dans l'état EX0, l'instruction est décodée (test sur le code d'opération),les micro-opératoins permettant d'exécutant tout ou partie de l'instruction apparaissant d'une boite d'action conditionnelle à ce test. Lorsque l'instruction ne nécessite qu'un cycle d'horloge pour son traitement, INF est l'état suivant. Lorsque l'instruction ne nécessite pas de modification du contenu du PC (ce qui est le cas de toutes les instructions qui ne sont pas des instructions de branchement), celui-ci est simplement incrémenté pour pointer sur l'instruction suivante. Pour des instructions nécessitant plus d'un cycle d'horloge pour leur traitement (instructions qui seront détaillées plus tard), d'autres états peuvent être nécessaires (EX1, etc). pour certaines instructions des test supplémentaires sont effectués sur les indicateurs d'état (Z, N dans la figure) pour définir la ou les micro-opérations à effectuer.

Par exemple, le code opératoire 0000000 correspondant à l'instruction MOVA implique un simple transfert du contenu du registre d'adresse SA dans le registre d'adresse DR. Le code opératoire 1100001 correspondant à l'instruction BRN provoque le branchement ou non du compteur ordinal sur l'adresse étendue résultant de la concaténation des champs DR et SB, selon le signe (indicateur N) de la valeur contenu dans le registre désigné par SA.

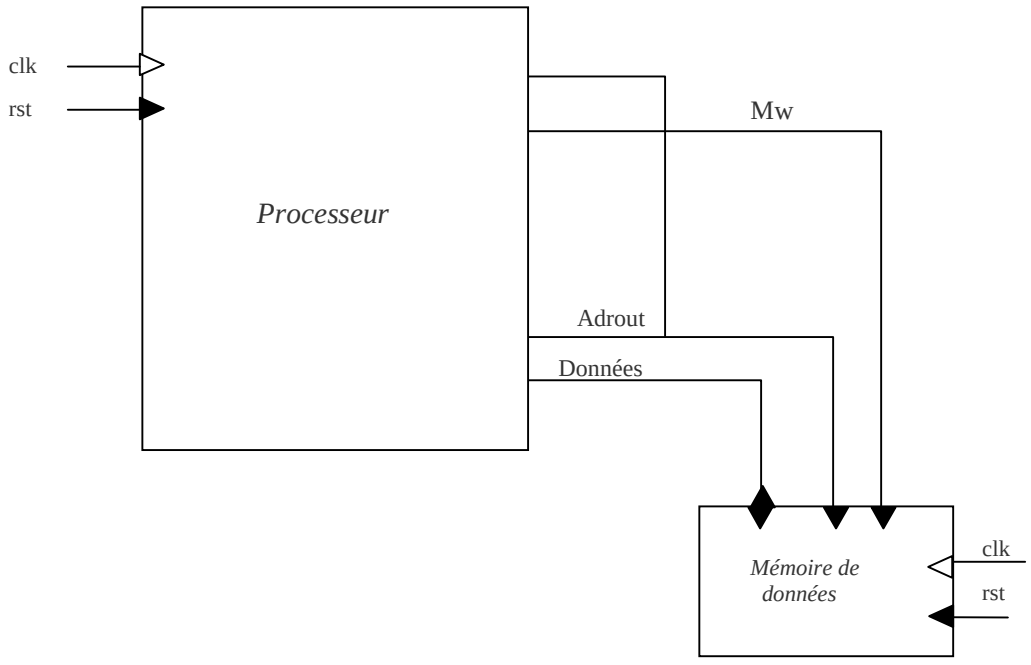


**Figure 5: Structure générale du calculateur de Von Neuman**

Pour réduire la complexité du modèle, on se propose de décomposer le calculateur en 2 entités principales:

- la mémoire centrale
- Le processeur

Ces 2 entités sont couplées par un bus de données, un bus d'adresse et un signal de contrôle *Memory write* autorisant l'opération d'écriture. On notera que théoriquement le bus de données de la mémoire centrale est bidirectionnel. Il n'est donc pas possible de lire et d'écrire simultanément et la mémoire est dite simple port. Ceci est une particularité de l'architecture de von neuman par rapport à celle de Harvard qui est monocycle. Dans le cadre du projet, et pour simplifier la description, nous distinguerons la donnée en entrée de la donnée en sortie.

## **3.1.** La mémoire centrale

La mémoire de données est une mémoire de $2^{16}$ éléments (adresses) de 16 bits chacun. Le bus d'adresse correspond à l'adresse du mot en lecture ou en écriture. Comme déjà mentionné, l'opération d'écriture est autorisée par le signal de commande *MW* (pour memory write) sur niveau haut. Dans le cas d'une lecture, la valeur du mot de 16 bits correspondant à l'adresse est accessible sur le bus de données.

## **3.2** Le processeur

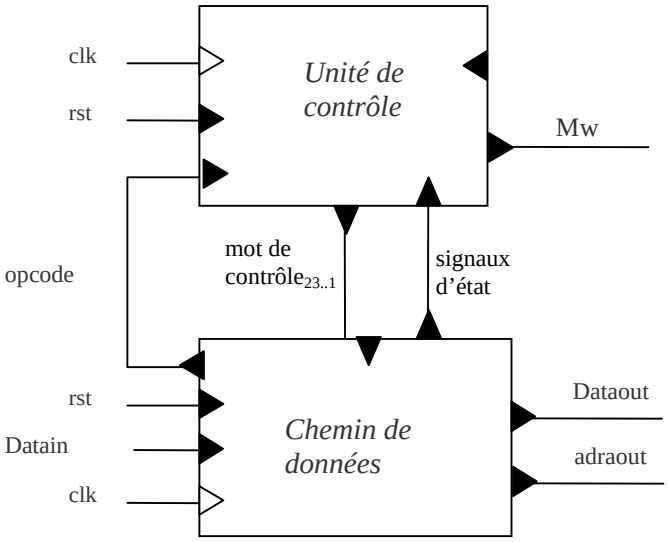


**Figure 6: Raffinement de l'architecture du processeur**

L'entité processeur se raffine en 2 sous blocs :

- l'unité de contrôle
- le chemin de données

Comme le montre la Figure 5, l'unité de contrôle pilote le chemin de données grâce à un mot de contrôle. Ce mot de contrôle est formé d'un ensemble de signaux de contrôle permettant de positionner le chemin de données en fonction de l'instruction à exécuter. Une constante

(Opérande immédiat) permet également la récupération de valeur de l'opérande depuis la mémoire d'instructions dans le cas de l'adressage immédiat.

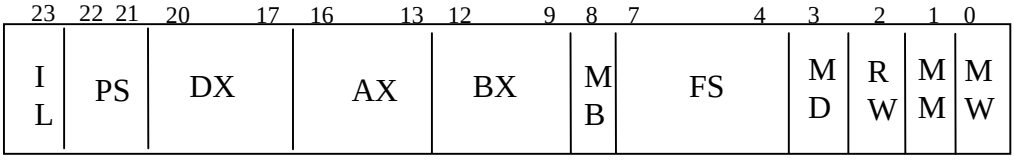


**Figure 7: Le format du mot de contrôle**

Le chemin de donnée fournit en sens inverse le code opération de l'instruction en cours d'exécution et des indicateurs d'état (N,V,Z,C) qui permettront des ruptures de séquences (branchements conditionnels) dans les programmes.

## **3.2.1.** Le chemin de données

Le bloc chemin de données peut être spécifié à partir de la table d'opération ci-dessous. Cette table liste les transferts de registre (ou micro-opérations) en fonction des signaux du mot de contrôle, pour les poids 1 à 22 de ce mot(cf Figure 7). Pour chaque micro-opération, DX,AX,BX précisent l'adresse éventuelle des registres destinataires, source1 et source2 de l'opération. Le signal MB est positionnée en mode registre ou constante. Dans le premier cas, l'opérande utilisé par l'unité fonctionnelle provient du fichier de registre.. Dans le second cas, l'opérande provient de la mémoire d'instructions de l'unité de contrôle et passe de 3 à 16 bits par ajout de 13 bits à zéro en tête. Le signal MD permet de préciser si la donnée à inscrire dans un fichier du registre provient du le résultat d'une opération ou d'une donnée de la mémoire (data in). Les 4 bits FS définissent la nature de l'opération à effectuer le cas échéant. Finalement, RW précise si un registre doit être modifié (1) ou non (0).

| DX AX BX | | MB | | FS | | MD | | RW | |
|---|---|---|---|---|---|---|---|---|---|
| fn | code | fn | code | fn | code | fn | code | fn | code |
| R[Dr] R[Sa] R[Sb] | 0xxx | Registre | 0 | F = A | 0000 | UF | 0 | non | 0 |
| R[code] R[code] R[code] | 1xxx | cste | 1 | F= A+1 | 0001 | datain | 1 | ecriture | 1 |
| | | | | F = A+B | 0010 | | | | |
| | | | | inutilisé | 0011 | | | | |
| | | | | inutilisé | 0100 | | | | |
| | | | | F= A+B+1 | 0101 | | | | |
| | | | | F=A-1 | 0110 | | | | |
| | | | | inutilisé | 0111 | | | | |
| | | | | F= A et B | 1000 | | | | |
| | | | | F= A ou B | 1001 | | | | |
| | | | | F= A ⊕ B | 1010 | | | | |
| | | | | F= $\overline{A}$ | 1011 | | | | |
| | | | | F = B | 1100 | | | | |
| | | | | F= sr B | 1101 | | | | |
| | | | | F= sl B | 1110 | | | | |
| | | | | inutilisé | 1111 | | | | |

**Figure 8: Spécification du chemin de données**

L'implémentation du chemin de données à partir de cette table d'opérations nécessite différents composants combinatoires et séquentiels (Figure 9). On peut distinguer:

- un banc de registres, mémoire rapide interne au processeur composé de 16 registres de 16 bits chacun.

- une unité fonctionnelle, circuit combinatoire réalisant des opérations arithmétiques, logiques, décalages ou simples transferts.
- quelques multiplexeurs permettant de configurer le chemin de données en fonction du mot de contrôle.
- Un registre d'instruction permettant de stocker l'instruction en cours d'exécution
- Un registre compteur ordinal permettant de pointer en mémoire centrale sur la prochaine instruction à exécuter.
- Un bloc *zero fill* permettant de compléter la constante sur 16 bits, constante issue de l'instruction dans le cas d'un adressage immédiat
- Un bloc *register adress logic* permettant de fixer l'adresse des opérandes sources et du registre destinataire
- Un bloc *address extend* permettant d'étendre le signe du déplacement à effectuer dans le cas d'un branchement relatif.

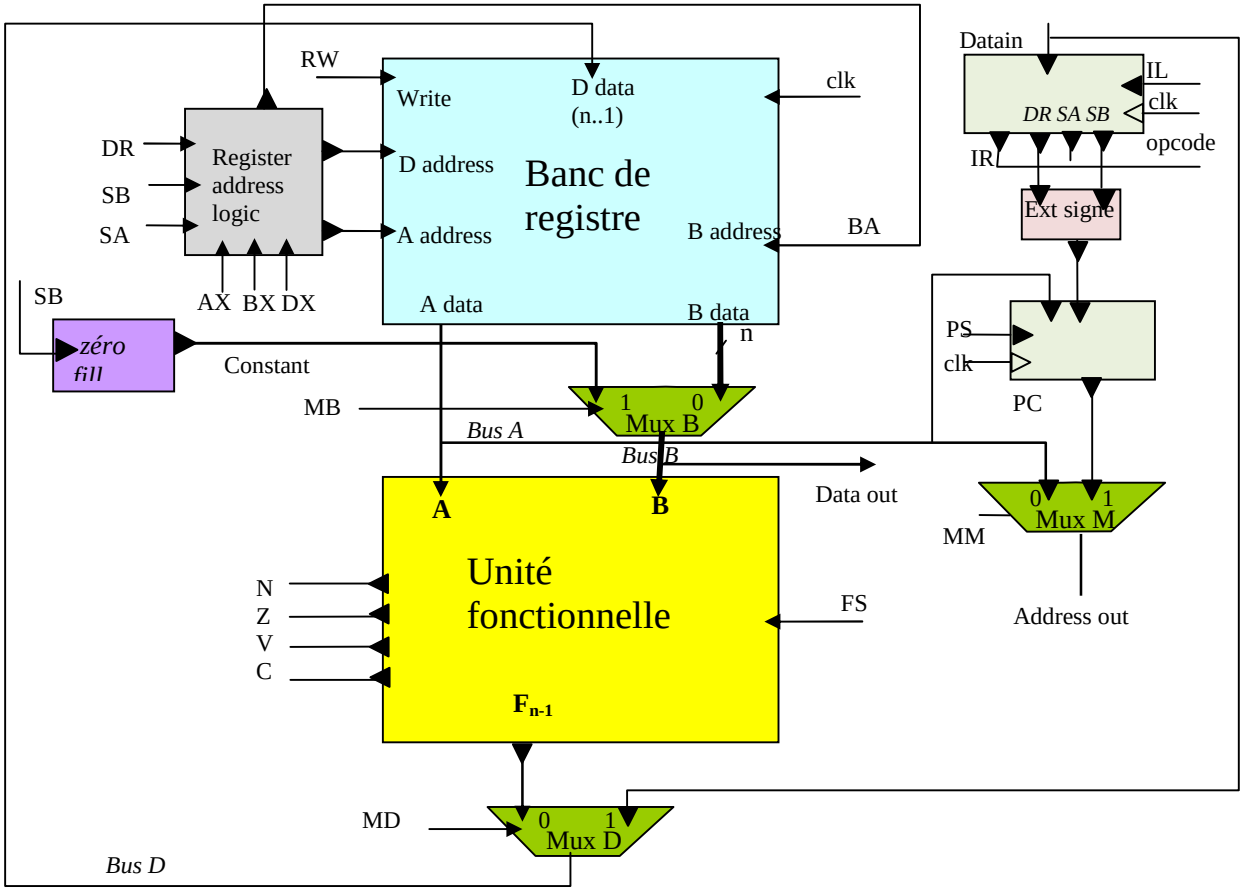


**Figure 9: Architecture du chemin de données partagé**

### *A Le fichier de registres*

Comme pour la mémoire de données, le banc de registre enregistre les données à l'adresse DR sur un niveau haut de RW. La lecture des données s'effectue sur deux bus distincts, le bus A et le bus B. La présence des données en sortie est permanente et fonction des adresses respectives DA et DB. On notera que les écritures sont synchrones à l'horloge et sont effectuées au début du cycle d'horloge suivant la fin du traitement de l'instruction courante. Comme indiqué précédemment, l'architecture de l'unité de contrôle est une architecture permettant le traitement d'instructions sur plusieurs cycles d'horloge. On parle de registre de travail. Ce type d'exécution nécessite de pouvoir stocker des données temporairement depuis

leur instant de génération jusqu'à l'instant de leur utilisation. Dans notre architecture, il s'agit des registres 8 à 15. Ces registres ne sont pas visibles du programmeur et ne sont pas considérés comme ressources de stockage du modèle de programmation.. Lors de l'étude de l'unité de contrôle, nous verrons plus précisément leur exploitation.

***B.*** *Le bloc Register Address Logic*

.

L'adressage des 16 registres du *register File* peut être réalisé à partir de l'instruction et de l'unité de contrôle lorsqu'il s'agit d'un registre visible du programmeur (R0 à R7) et uniquement par l'unité de contrôle lorsqu'il s'agit des registres R8 à R15. Le rôle de ce bloc est de multiplexer ces adresses en fonction du bit de poids fort de AX,BX et DX. Lorsque ce bit vaut 1, l'adresse appliquée à l'entrée du *register file* est directement l'adresse 4 bits issue de AX,BX ou DX selon. Dans le cas où ce bit vaut 0, on concatène un bit de poids fort à 0 à SA,SB ou DR issu de l'instruction pour former l'adresse sur 4 bits.

***C*** *L'unité fonctionnelle*

| $FS_{3..0}$ | Fonction |
|---|---|
| 0 0 0 0 | F = A |
| 0 0 0 1 | F= A+1 |
| 0 0 1 0 | F = A+B |
| 0 0 1 1 | Inutilisé |
| 0 1 0 0 | Inutilisé |
| 0 1 0 1 | F= A+$\overline{B}$+1 |
| 0 1 1 0 | F=A-1 |
| 0 1 1 1 | Inutilisé |
| 1 0 0 0 | F= A et B |
| 1 0 0 1 | F= A ou B |
| 1 0 1 0 | F= $\overline{A} \oplus$ B |
| 1 0 1 1 | F= $\overline{A}$ |
| 1 1 0 0 | F = B |
| 1 1 0 1 | F= sr B;F(n)=0 |
| 1 1 1 0 | F= sl B;F(1)=0 |
| 1 1 1 1 | Inutilisé |

**Figure 10: Spécification du comportement de l'unité fonctionnelle**

L'unité fonctionnelle est capable de réaliser différentes opérations s'étendant de la simple opération de transfert pour les mouvements de données internes au microprocesseur ou les transferts impliquant la mémoire externe de données. La table de vérité complète du composant vous est fournie Figure 10.

***D.*** *Le bloc zero fill*

Comme mentionné précédemment, le microprocesseur peut utiliser une donnée implicitement contenu dans l'instruction elle-même. L'UF utilisant des données de 16 bits, 13 bits à zéros sont ajoutés en tête de cette donnée de 3 bits de poids faible de IR (SB) pour respecter cette contrainte. Par exemple,

110 devient 0000000000000110

***E.*** *Le registre instruction*

Le registre instruction permet de mémoriser l'instruction en cours d'exécution. Son fonctionnement est contrôlé par le bit *IL* du mot de contrôle qui, s'il vaut 1, autorise le chargement de la sortie de la mémoire au front d'horloge.

***F.*** *Le compteur ordinal*

Le compteur ordinal permet de pointer sur la prochaine instruction à exécuter. Le comportement du bloc au front d'horloge est défini par les 2 bits *PS* du mot de contrôle :

00 : contenu inchangé
01 : incrémentation
10 : ajout à la valeur courante de DR concaténé à SB après extension du signe
11 : chargement du contenu de SA

Les deux dernières opérations permettent d'effectuer des ruptures de séquences dans l'enchainement des instructions, la rupture pouvant être conditionnelle ou conditionnelle respectivement.

***G.*** *L'extension de signe*

Lors d'un branchement relatif,(cf 2.C), l'adresse concaténée doit être étendue de 6 à 16 bits. Qui est la taille du PC. C'est le rôle de ce bloc de copier le bit de signe de cette adresse de 6 bits sur les 10 bits de poids fort. Par exemple, si DR=101 et SB=100, on a un déplacement relatif négatif de $101100_2$ soit $-20_{10}$. L'extension de signe fournit alors la valeur 1111111111101100. Si DR=001 et SB=100, on a un déplacement positif de $001100_2$ soit $+12_{10}$. L'extension de signe fournit alors la valeur 0000000000001100.

***H.*** *Micro-opérations*

| Micro-opération | DA | AA | BA | MB | FS | MD | RW | |
|---|---|---|---|---|---|---|---|---|
| $R_1 \leftarrow R_2 - R_3$ | $R_1$ | $R_2$ | $R_3$ | Register | $F = A+\overline{B}+1$ | function | write | 0001001000110010101X |
| $R_4 \leftarrow Sl\ R_6$ | $R_4$ | -- | $R_6$ | Register | F = Sl B | function | write | 0100XXXX01100111001X |
| $R_7 \leftarrow R_7+1$ | $R_7$ | $R_7$ | -- | Register | F = A+1 | function | write | 01110111XXXX0000101X |
| $R_1 \leftarrow R_0+2$ | $R_1$ | $R_0$ | -- | Constant | F = A+B | function | write | 00010000XXXX1001001X |
| Dataout $\leftarrow R_3$ | -- | -- | $R_3$ | Register | --- | --- | no write | XXXXXXXX00110XXXXX0X |
| $R_4 \leftarrow$ Datain | $R_4$ | -- | -- | --- | --- | datain | write | 0100XXXXXXXXXXXX11X |
| $R_5 \leftarrow 0$ | $R_5$ | $R_0$ | $R_0$ | Register | F=A⊕ B | function | write | 0101000000000101001X |

**Figure 11: Quelques micro-opérations du chemin de données et leur mot de contrôle**

Grâce à ces composants, le chemin de données est capable d'effectuer un certain nombre de transferts de registres ou micro-opérations. Les signaux de contrôle définit au travers du mot de contrôle détermine le type de transfert de registre effectué. A titre d'illustration, la table ci-dessus montre quelques exemples de micro-opérations réalisées par le chemin de données selon l'état des 21 bits de poids faible du mot de contrôle.

### 3.2.2 L'unité de contrôle

L'unité de contrôle est de type séquentiel (contrairement à l'unité de contrôle des processeurs Harvard qui est mono-cycle et donc combinatoire). Elle est donc plus complexe et son comportement se déduit de l'algorithme State machine (Figure 4) et de la table d'opération (Figure 8) du chemin de données, en remplaçant les micro-opérations de l'ASM par les signaux de contrôle du mot de contrôle commandant les transferts. Ce comportement est décrit par la table d'états ci-dessous

| Etat | Opcode | VCNZ | NS | IL | PS | DX | AX | BX | MB | FS | MD | RW | MM | MW |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| INF | xxxxxxx | xxxx | EX0 | 1 | 00 | XXXX | XXXX | XXXX | X | XXXX | X | 0 | 1 | 0 |
| EX0 | 0000000 | xxxx | INF | 0 | 01 | 0XXX | 0XXX | XXXX | X | 0000 | | 0 | 1 | X | 0 |
| EX0 | 0000001 | xxxx | INF | 0 | 01 | 0XXX | 0XXX | XXXX | X | 0001 | | 0 | 1 | X | 0 |
| EX0 | 0000010 | xxxx | INF | 0 | 01 | 0XXX | 0XXX | 0XXX | X | 0010 | | 0 | 1 | X | 0 |
| EX0 | 0000101 | xxxx | INF | 0 | 01 | 0XXX | 0XXX | 0XXX | 0 | 0101 | | 0 | 1 | X | 0 |
| EX0 | 0000110 | xxxx | INF | 0 | 01 | 0XXX | 0XXX | XXXX | X | 0110 | | 0 | 1 | X | 0 |
| EX0 | 0001000 | xxxx | INF | 0 | 01 | 0XXX | 0XXX | 0XXX | 0 | 1000 | | 0 | 1 | X | 0 |
| EX0 | 0001001 | xxxx | INF | 0 | 01 | 0XXX | 0XXX | 0XXX | X | 1001 | | 0 | 1 | X | 0 |
| EX0 | 0001010 | xxxx | INF | 0 | 01 | 0XXX | 0XXX | 0XXX | X | 1010 | | 0 | 1 | X | 0 |
| EX0 | 0001011 | xxxx | INF | 0 | 01 | 0XXX | 0XXX | XXXX | X | 1011 | | 0 | 1 | X | 0 |
| EX0 | 0001100 | xxxx | INF | 0 | 01 | 0XXX | XXXX | 0XXX | 0 | 1100 | | 0 | 1 | X | 0 |
| EX0 | 0010000 | xxxx | INF | 0 | 01 | 0XXX | 0XXX | XXXX | X | xxxx | | 1 | 1 | 0 | 0 |
| EX0 | 0100000 | xxxx | INF | 0 | 01 | XXXX | 0XXX | 0XXX | 0 | xxxx | | X | 0 | 0 | 1 |
| EX0 | 1001100 | xxxx | INF | 0 | 01 | 0XXX | XXXX | XXXX | 1 | 1100 | | 0 | 1 | 0 | 0 |
| EX0 | 1000010 | xxxx | INF | 0 | 01 | 0XXX | 0XXX | XXXX | 1 | 0010 | | 0 | 1 | 0 | 0 |
| EX0 | 1100000 | xxx1 | INF | 0 | 10 | XXXX | 0XXX | XXXX | X | 0000 | | X | 0 | 0 | 0 |
| EX0 | 1100000 | xxx0 | INF | 0 | 01 | XXXX | 0XXX | XXXX | X | 0000 | | X | 0 | 0 | 0 |
| EX0 | 1100001 | xx1x | INF | 0 | 10 | XXXX | 0XXX | XXXX | X | 0000 | | X | 0 | 0 | 0 |
| EX0 | 1100001 | xx0x | INF | 0 | 01 | XXXX | 0XXX | XXXX | X | 0000 | | X | 0 | 0 | 0 |
| EX0 | 1110000 | xxxx | INF | 0 | 11 | XXXX | 0XXX | XXXX | X | 0000 | | X | 0 | 0 | 0 |

**Figure 12: Table d'états de l'unité de contrôle**

Les variables booléennes formant le vecteur d'entrée de la machine à état sont des valeurs binaires issues du code opération de l'instruction et d'indicateurs d'état du chemin de données. En fonction de l'état interne et de ce vecteur d'entrée, l'unité de contrôle définit le vecteur de sortie (mot de contrôle) et l'état suivant (*NS* pour Next State).

*Remarque :*Des instructions plus complexes peuvent être développées à partir de cette architecture comme des décalages multiples, des instruction à adressage indirect, etc. Ces instructions nécessiteraient typiquement plus de 2 cycles d'horloges et des registres de travail l(8 à 15) pour leur exécution. Ce type d'instruction ne sera pas étudié dans le cadre de ce projet..

# 4. Description du calculateur en VHDL

## *4.1 Définition du Package*

2 composants du calculateur, mémoire centrale et fichier de registres vont nécessiter le chargement de données initiales. Pour cela plusieurs solutions sont possibles au moment de l'initialisation :

- initialisation dans le programme VHDL au niveau de la déclaration : cette solution est la plus simple mais elle suppose une recompilation de la description pour tout changement de contenu.
- déclaration d'un paramètre générique avec initialisation du contenu sur un signal *rst*. Cette solution ne nécessite plus que la re-compilation de l'entité test en cas de modification du contenu. Mais, il vous faudra définir un package pour déclarer le type de ce paramètre.
- Lecture des données initiales lues à partir d'un fichier : c'est la solution la plus souple car elle ne nécessite aucune recompilation. C'est également la solution la plus proche de la réalité. Vous pouvez définir une fonction *load_program,* fonction réalisant le chargement de la mémoire au *rst*.

Pour assurer le maximum de flexibilité, le chargement devra s'opérer à partir d'un fichier stockant ces données de manière externe. Cette solution, permettra d'éviter de recompiler l'architecture de l'entité suite une modification des données de la mémoire interne. Pour cela, on définira une fonction de prototype suivant :

```
Impure function init_mem(fich :string ; nbard:natural) return defmem;
```

Ou *fich* est le nom du fichier des données à charger en mémoire et *nbadr* est le nombre d'adresses de la mémoire. Cette fonction retournera un tableau non contraint de type *defmem* définit comme suit

```
subtype word is signed(15 downto 0);
type defmem is array(natural range <>) of word;
```

en l'initialisant avec les données issues du fichier de nom *fich*. *Nbbits* fixera le nombre de bits d'adresse. Les données du fichier sont présentées sous la forme <adresse> <valeur>. Par exemple, le contenu du fichier ci-dessous

```
248 0000000000000010
249 0000000000001010
```

permettra le chargement de la valeur 2 à l'adresse 248 et la valeur 10 à l'adresse 249. Le reste des données sera laissé non initialisé. Pour assurer l'accès de cette fonction par les futurs composants, le prototype de la fonction sera déclaré dans l'entête d'un package tandis que le corps de cette fonction sera défini dans le corps de ce package  (package body On n'oubliera pas bien entendu de rendre visible ce package lorsque nécessaire par la suite.

## *4.2. La mémoire centrale*

Le nom du fichier stockant les données à charger au moment du rst sera fournit comme paramètre générique. Le bus d'adresse sera de type *unsigned,* tandis que les bus de données *datazin* et *dataout* seront de type *signed* .

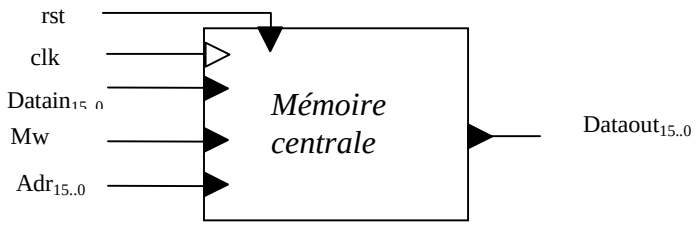


**Figure 13: symbole de la mémoire**

Ces types sont accessibles dans le package *numeric_bit* de la bibliothèque *ieee*. Le comportement de la mémoire en écriture sera définit par un *process* manipulant une variable de type tableau, process synchrone à l'horloge clk. La lecture sera quand à elle asynchrone et sera définie par une instruction concurrente. La mémoire sera initialisée à partir du fichier sur *rst*. La fonction de conversion *to_integer* vous permettra de convertir le type *unsigned* de l'adresse sous forme d'entier, de manière à pouvoir indicer correctement le tableau. On prendra 4 ns comme délai d'opérations en lecture ou écriture.

### *4.3 Le chemin de données partagé*

*Ctrlword*, *adrout et opcode* seront de type unsigned, *dataout* et *datain* de type signed. Le nom du fichier stockant les valeurs initiales des registres sera fournit comme paramètre du composant. La description des différents blocs du chemin de données s'effectuera à l'aide d'instructions d'affectations concurrentes pour les différents signaux (pas d'entités) : bus D, dataout, ardout, busB, etc. On n'oubliera pas de décrire le comportement des indicateurs d'états en sortie de l'Unité Fonctionnelle avec comme convention:

- Z =1 si le résultat de l'opération est nul
- N=1 si le résultat de l'opération est négatif.

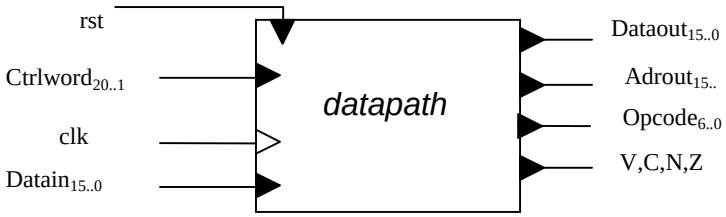


**Figure 14: Symbole du chemin de données**

La description des indicateurs C(carry) et V(overflow) sera facultative. Le signal *rst* permettra de charger les registres dans un état initial souhaité depuis un fichier, si nécessaire. On retiendra en autres les temps de propagation suivants :

- Lecture ou écriture d'un registre du fichier de registre : 3 ns
- Opérations sur PC ou IR : 2 ns
- Multiplexeurs (MD, MB ou MM) et indicateurs: 1 ns
- Unité fonctionnelle : 4 ns

Les délais de l'extension et du zero fill seront négligé. On notera que des conversions unsigned-signed peuvent être nécessaire lors, par exemple, de la récupération d'une adresse à partir du bus de données seront effectuées par un simple « cast ». Pour faciliter l'écriture et la mise au point, on définira des alias pour les différents champs du registre d'instruction. Par

exemple, $IR_{2..0}$ aura SB comme alias, $IR_{5..3}$ aura SA comme alias, etc. Même chose pour le mot de contrôle pour lequel, par exemple, il sera possible de désigner les bits 20..17 par l'alias *DX*.

### *4.3 L'unité de contrôle*

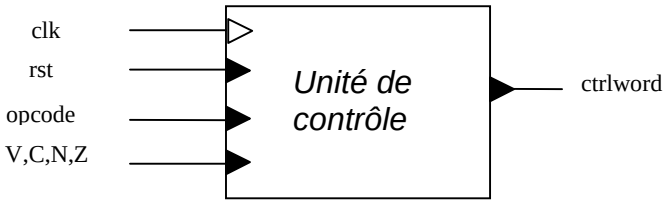


**Figure 15: symbole de l'unité de contrôle**

La description de l'automate s'effectuera classiquement avec 2 process :

- Un process pour l'enregistrement du nouvel état
- Un process combinatoire pour le calcul de l'état suivant et des sorties

Les valeurs non significatives (X) du mot de contrôle seront forcées à 0. On retiendra les temps de propagation suivants :

- Mémorisation du vecteur d'état : 3 ns
- Calcul des sorties et de l'état suivant: 2 ns

### *4.4 Le calculateur complet*

Le calculateur complet (dont les entrées/sorties se limitent aux entrées *clk* et *rst*) se décrira en deux niveaux hiérarchiques : la calculateur formé des composants *mémoire* et *CPU*, entité elle-même formée des composants *datapath* et *controle*. Les noms des fichiers instructions/données en mémoire centrale et valeurs initiales des registres seront fournis sous forme de paramètres.

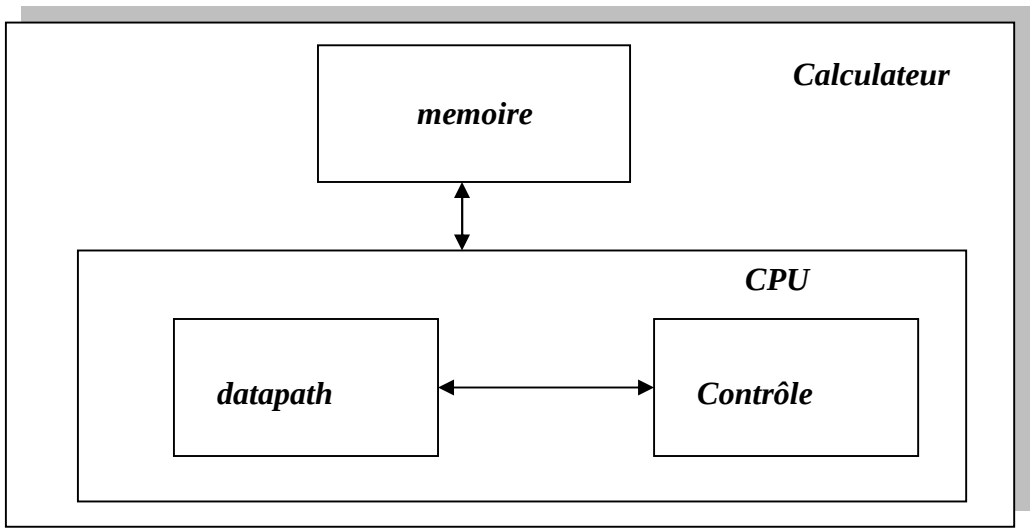


**Figure 16: Structure hiérarchique du calculateur**

Le cycle d'horloge devra être réglé par rapport au chemin critique en cumulant les temps de propagation des composants traversés lors du parcours de ce chemin. Un script de simulation devra être disponible pour les différentes commandes menant à la simulation du projet.

## *5. Démarche et conseils de mise au point*

On vous recommande de suivre une démarche très progressive en accompagnant l'évolution de la description de nombreux tests et validations intermédiaires. Conservez également certaines de ces versions comme points de retour arrières possibles en cas de problème sur votre nouvelle version. Les 3 entités doivent être testées (testbench) et vérifiées séparément avant de procéder à leur assemblage, en commençant de préférence par la mémoire de données, suivi du chemin de données et de l'unité de contrôle. La description complète devra être validée sur l'exemple de programme fourni en annexe. Un test sur un programme plus complexe mettant en œuvre les instructions de branchement (multiplication de deux nombres par la méthode des additions répétitives, PGCD, etc par exemple) devra être proposé.

Pour chaque partie du calculateur, il vous est demandé de fournir (par courrier électronique) le programme VHDL et une copie d'écran de la simulation) en respectant impérativement les dates suivantes (tout retard étant sujet à pénalisation) :

Mémoire centrale : 29 mars
Chemin de données : 19 avril
Unité de contrôle : 10 mai
Top level ; 31 mai

Le compte rendu du mini projet doit fournir tous les éléments nécessaires et utiles à une évaluation correcte du travail. Le respect des consignes sera un élément très important de l'appréciation. Au minimum, ce rapport présentera :

- Un sommaire
- Le cahier des charges
- Le descriptif du travail
- Une conclusion
- Les listings VHDL et les chronogrammes complets en annexe.

Le rapport doit être tapé sur informatique, relié et remis à l'accueil le vendredi 7 juin 2013, 16h30 au plus tard (le tampon de l'accueil faisant foi). Aucun envoi par mail du rapport ne sera accepté et tout retard dans la remise du projet sera également pénalisé.

## Annexe : Un petit programme d'essai

Le programme suivant est un programme permettant d'effectuer indéfiniment la soustraction de la valeur de l'emplacement mémoire pointé par R3 ajouté à 3 à la valeur de l'emplacement R3+1. Le résultat est stocké à l'emplacement d'adresse R3+2.

```
LD    R1,R3    charge R1 avec le contenu mémoire dont l'adresse est dans R3
ADI   R1,R1,3  additionne 3 à R1
NOT   R1,R1    complémente à 1 la valeur dans R1
INC   R1,R1    incrémente la valeur dans R1
INC   R3,R3    incrémente la valeur dans R3
LD    R2,R3    charge R2 avec le contenu mémoire dont l'adresse est dans R3
ADD   R2,R2,R1 additionne le contenu de R1 au contenu de R2
INC   R3,R3    incrémente le contenu de R3
ST    R3,R2    stocke la valeur de R2 à l'adresse contenue dans R3
JMP   R5       on reboucle (saut inconditionnel) sur l'adresse 0
```

ce qui correspond au code machine (aux formats binaire et hexa) suivant :

```
0010000 001 011 000          2058
1000010 001 001 011          844B
0001011 001 001 000          1648
0000001 001 001 000          0248
0000001 011 011 000          02D8
0010000 010 011 000          2098
0000010 010 010 001          0491
0000001 011 011 000          02D8
0100000 000 011 010          401A
1110000 000 101 000           E028
```

Le programme est stocké sur 10 adresses (de 0 à 9) dans la mémoire d'instructions. Si R3 et R1 contiennent initialement 248 et 5, respectivement, et qu'aux adresses 248 et 249 de la mémoire de données se trouvent, respectivement, les valeurs 2 et 78, le programme stockera la valeur 73 (78-5) à l'adresse 250. Le programme reboucle indéfiniment sur lui-même avec un saut inconditionnel à l'adresse contenue dans R5 (qu'on suppose initialement pré chargé avec la valeur 0). La Figure 17 donne un chronogramme obtenu lors de l'exécution de ce programme.

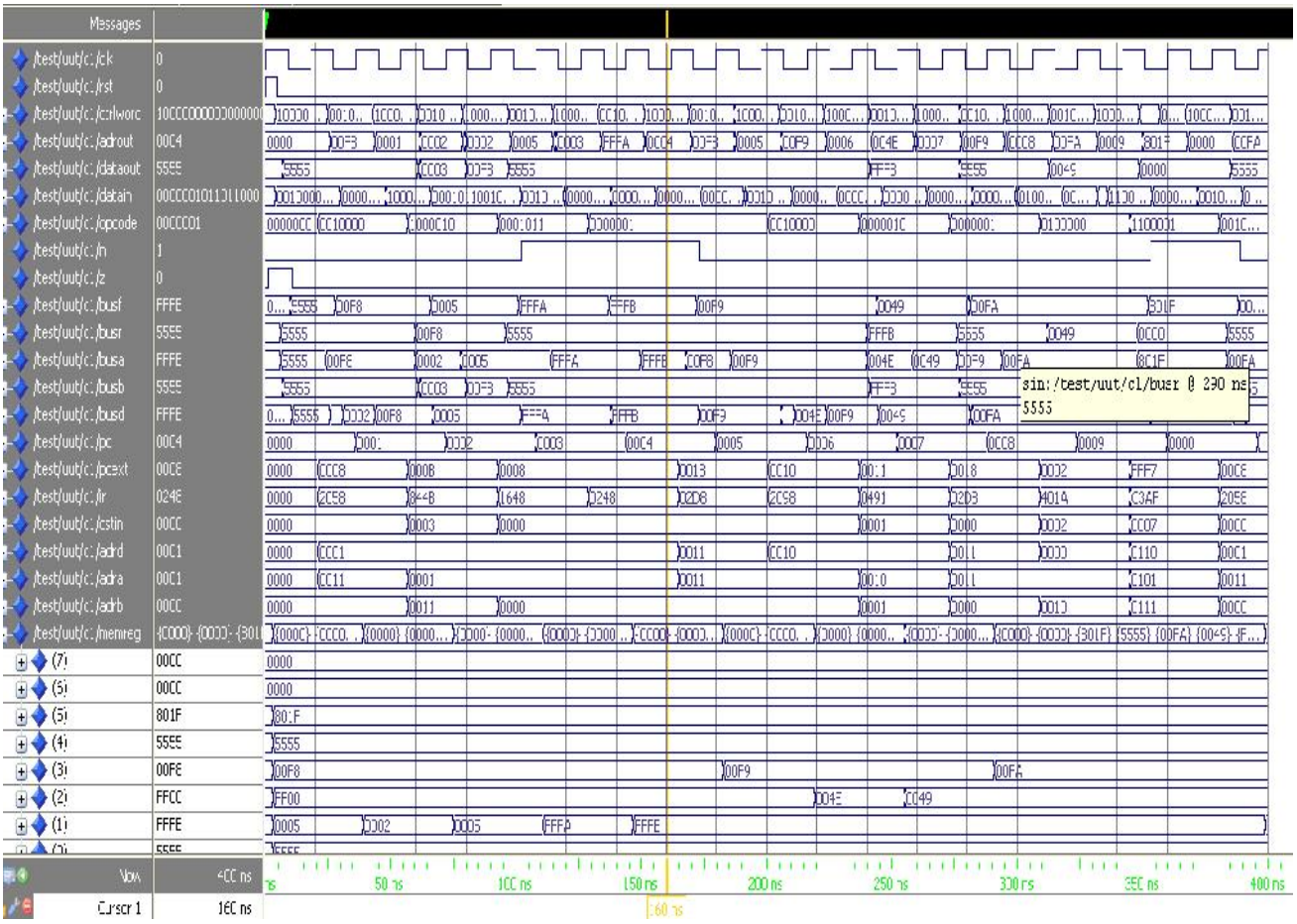


**Figure 17: Chronogramme résultat de l'exécution du programme**