# Modélisation et implémentation d'un processeur RISC 16 bits

## Objectif :

*Le but de ce projet est de mettre en application de maintes notions théoriques vues en cours : architecture avancée des processeurs, modélisation VHDL, implémentation sur FPGA.*

Nous allons donc réaliser un vrai processeur RISC 16 bits. Pour ce faire, nous allons le décrire intégralement en VHDL, puis nous l'implanterons sur un FPGA (*Field-Programmable Gate Array*), un circuit intégré complètement programmable capable d'implanter n'importe quel circuit.

### 1. Processeur RISC 16 bits

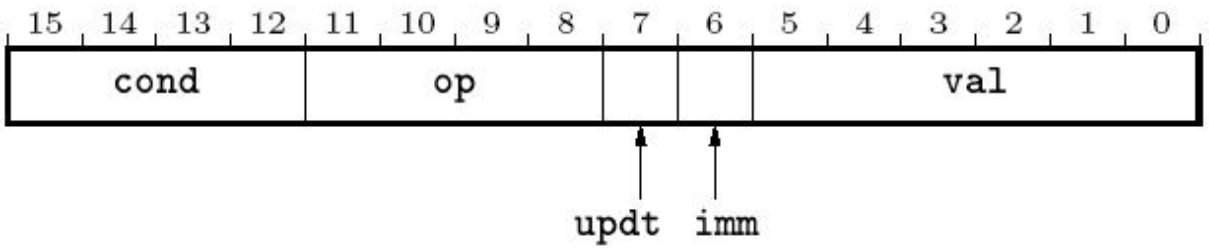
Le processeur considéré est à une adresse et compte 64 registres de 16 bits. Il dispose donc d'un accumulateur Acc correspondant au registre r0, ainsi que d'un compteur ordinal (PC, pour *program counter*) correspondant à r63. On a aussi un compteur ordinal de retour, dans lequel est stocké l'adresse de retour après un appel de fonction; il est noté RPC et fixé à r62.

| Nom du registre | numéro |
|---|---|
| accumulateur | R0 |
| PC | R63 |
| RPC | R62 |

#### 1.1 Mot d'instruction

Le mot d'instruction est découpé de la manière suivante :

- bits 15 à 12, `cond` : prédicat conditionnant l'exécution de l'instruction en fonction des bits de statut ;
- bits 11 à 8, `op` : opcode de l'instruction ;
- bit 7, `updt` : drapeau déterminant si les bits de statut doivent être mis à jour par cette instruction;
- bit 6, `imm` : drapeau déterminant si le champ suivant correspond à un numéro de registre ou à un immédiat ;
- bits 5 à 0, `val` : numéro de registre rX si `imm` est à 0, ou valeur immédiate X si `imm` est à 1.




#### 1.2 Jeu d'instructions

Le jeu d'instructions est très simple, puisqu'il n'y a que 16 instructions possibles décrites dans le tableau-ci-dessous.

| Opcode | Codage | Action |
|---|---|---|
| AND | 0000 | Acc ← Acc AND rX/X |
| OR | 0001 | Acc ← Acc OR rX/X |
| XOR | 0010 | Acc ← Acc XOR rX/X |
| NOT | 0011 | Acc ← NOT rX/X |
| ADD | 0100 | Acc ← Acc + rX/X |
| SUB | 0101 | Acc ← Acc − rX/X |
| LSL | 0110 | Acc ← Acc ≪ rX/X |
| LSR | 0111 | Acc ← Acc ≫ rX/X |

| Opcode | Codage | Action |
|---|---|---|
| LDA | 1000 | Acc ← [rX/X] |
| STA | 1001 | [rX/X] ← Acc |
| MTA | 1010 | Acc ← rX/X |
| MTR | 1011 | rX ← Acc |
| JRP | 1100 | PC ← PC + rX/X |
| JRN | 1101 | PC ← PC − rX/X |
| JPR | 1110 | PC ← rX/X |
| CAL | 1111 | RPC ← PC + 1<br>PC ← rX/X |

À noter :
- MTA, *Move To Accumulator* ;
- MTR, *Move To Register* : le bit imm ne doit pas être à 1 (ceci est en théorie assuré par le compilateur assembleur) ;
- JRP, *Jump Relative Positive* ;
- JRN, *Jump Relative Negative* ;
- JPR, *JumP Register* ;
- CAL, *CALl* : sauvegarde la valeur incrémentée du PC dans RPC (*return program counter*, fixé à r62), puis effectue un JPR. Pratique pour un appel de fonction.

**Syntax:**

- **<Operation>{<cond>}{S} Operand**

**Exemple :**

- **Instruction arithmétique et logique:**

  **a. adressage registre :**

  ANDT r1 --> Toujours accumulateur = accumulateur and r1 et ne pas mettre à jours les flag

  **b. adressage immédiat :**

  SUBPS 0x23 --> Toujours accumulteur = accumulteur - 0x23 et mettre à jours les flags

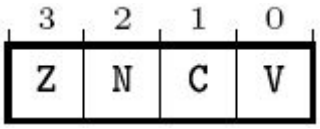
- **Instruction de transfert :**

  Move to accumulteur MTAP r3 --> si Z=0 et N=0 alors accumulteur = r3
  MTAN 0xAC --> si N=1 alors accumulteur = 0X00AC

**1.3 Statut et conditions**

On se donne les 4 drapeaux de statut suivants :

- Z, *Zero* : 1 si le résultat d'une opération est nul, et 0 sinon ;
- N, *Negative* : 1 si le résultat est strictement négatif, et 0 sinon ;
- C, *Carry* : 1 s'il y a dépassement de capacité sur 16 bits, et 0 sinon.
- V, *oVerflow* : 1 s'il y a dépassement de capacité sur 15 bits (dépassement signé), et 0 sinon.




Les prédicats conditionnant l'exécution des instructions sont alors codés de la manière suivante :

| Condition | Codage | Expression |
|---|---|---|
| F | 0000 | 0 |
| T | 0001 | 1 |
| Z | 0010 | Z |
| NZ | 0011 | ¬Z |
| P | 0100 | $(\neg Z) \wedge (\neg N)$ |
| NP | 0101 | $Z \vee N$ |
| N | 0110 | N |
| NN | 0111 | ¬N |

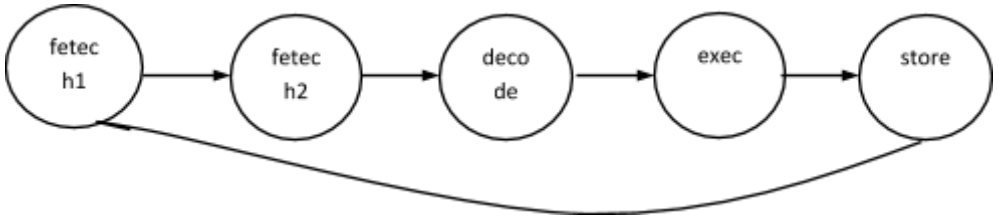
| Condition | Codage | Expression |
|---|---|---|
| C | 1000 | C |
| NC | 1001 | ¬C |
| V | 1010 | V |
| NV | 1011 | ¬V |
| – | 1100 | – |
| – | 1101 | – |
| – | 1110 | – |
| – | 1111 | – |

l'exécution de toutes instructions est conditionnelle, en effet une instruction n'est exécutée que si la condition est vérifiée. En général une condition est rajouté au mnémonique de l'instruction
Exemple

- si le condition est 0010, l'instruction ADDZ ne s'exécute que si le bit Z = sataus(3)=1
- si la condition est 0100, l'instruction ANDP ne s'exécute que si le bit Z=0 **et** le bit N=0
- si la condition est 001, l'instruction MTAT s'exécute toujours

### 1.4 Cycle d'exécution

Le cycle d'exécution d'une instruction sur ce processeur se déroule en 5 étapes, chacune d'entre elles durant un cycle d'horloge :




–
- fetch1: la valeur de PC est mise sur le bus d'adresse vers la mémoire.
- fetch2: le mot d'instruction lu en mémoire est stocké dans le registre d'instruction.
- decode: les valeurs des opérandes de l'instruction sont stockées dans les registres op1 et op2.

- exec: l'UAL calcule le résultat de l'opération ; l'accès mémoire (en lecture ou en écriture) est aussi initié; le PC est incrémenté, puis stocké dans PC ou dans RPC, selon qu'il s'agisse ou non d'un CAL.
- store: le résultat de l'ALU ou de la lecture en mémoire est mémorisé dans le registre adéquat.

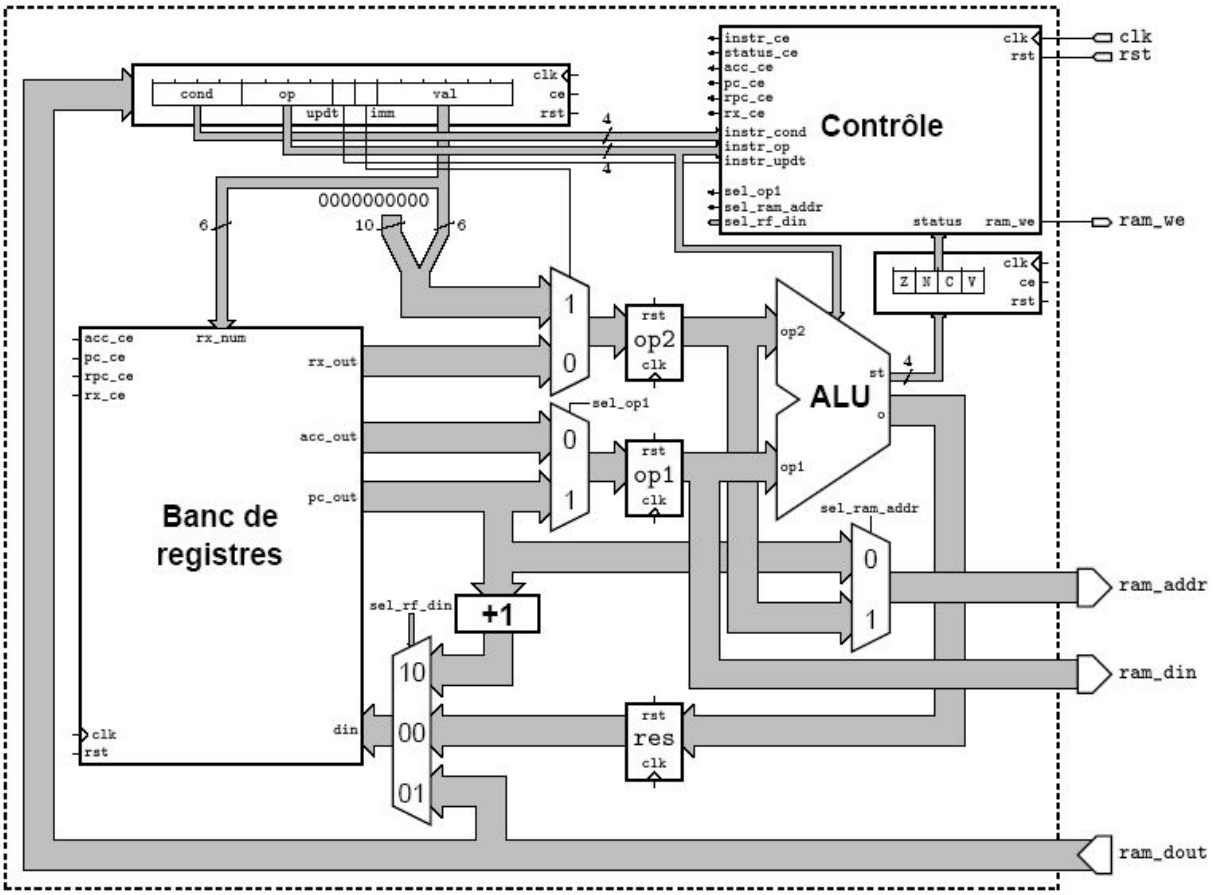



FIG. 1 – Architecture générale du processeur.

## 2. Réalisation

Pour réaliser ce processeur, vous allez utiliser Modelsim.

1.Créer un projet nommé processeur et y rajouter des fichiers.vhd pour décrire les différents composants de ce processeur. Commencer par décrire:

- les registres (instruction, statut et banc de registres),
- l'ALU,
- l'unité de contrôle,
- l'architecture globale du processeur sur laquelle viendront se greffer les composants précédents.

Un dossier vous est fournit servant d'ébauche de modèles VHDL ainsi que des test benchs pour valider vos composants. Il faut donc déterminer la description comportementale au niveau RTL

(Register Transfer Level) écrite en langage VHDL de votre composant. *Simuler* et vérifier son bon fonctionnement en utilisant l'outil **`ModelSim`**.

### 2.1 Registre de statut

Même topologie que pour le registre d'instruction, mis à part que celui-ci ne fait que 4 bits. Les drapeaux de statut arrivent de l'ALU par le port i, puis sont stockées dans le registre lorsque le signal de *clock enable* **`ce`** est à 1.

### 2.2 Banc de registres

Le banc de registre contient donc 64 registres de 16 bits. Tous ces registres sont accessibles grâce au port rx_num. Ainsi, le contenu du registre sélectionné par rx_num se retrouve sur rx_out grâce à un multiplexeur.

L'accumulateur étant r0 et le PC étant r63, ces deux registres proposent aussi directement leur contenu en sortie sur les ports acc_out et pc_out respectivement --> **la lecture est alors asynchrone**

Enfin, l'écriture dans le banc de registre s'effectue par le port din. Le registre dans lequel cette valeur sera stockée est indiqué par les différents signaux de *clock enable* :

- acc_e s'il s'agit de l'accumulateur,
- pc_ce s'il s'agit du PC,
- rpc_ce s'il s'agit du RPC (r62),
- et rx_ce s'il s'agit du registre sélectionné par rx_num (nécessite donc un démultiplexeur).

*Remarque* : Lors d'un reset, le PC doit être réinitialisé à 0x00,

### 2.3 Unité arithmétique et logique

Vous pouvez ici aussi vous répartir les tâches entre les différentes opérations si vous le souhaitez, mais une approche unifiée pour minimiser la logique serait préférable.

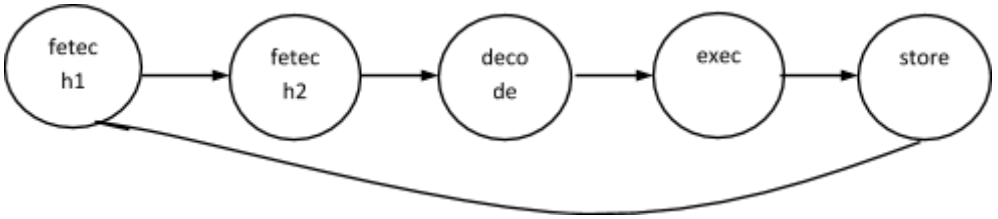
Il faut donc que votre ALU réalise les opérations suivantes :

- AND, OR, XOR et NOT bit à bit : vous pouvez utiliser pour cela les versions vectorielles des instructions and, or, xor et not.
- Addition et soustraction : de même, pour vous simplifier la vie, utilisez les opérateurs + et -. Attention aux dépassements de capacité !
- Décalages à gauche et à droite : vous pouvez utiliser aussi les opérateurs shl et shr.
- Encore une fois, attention aux dépassements de capacité !
- Recopies (pour les instructions comme MTA ou MTR) : notez que les déplacements de registre à registre passent par l'ALU pour pouvoir mettre les drapeaux de statut à jour.

Pensez aussi à mettre correctement à jour les drapeaux de statut en fonction de la valeur du résultat.

### 2.4 Unité de contrôle

L'unité de contrôle est composée d'un automate tout simple qui enchaîne de manière cyclique les états fetch1, fetch2, decode, exec et store.




En fonction de l'état courant stocké dans le registre state_r ainsi que d'autres paramètres

tels que l'instruction ou le statut, cette unité doit générer les signaux de commande appropriés pour contrôler les écritures dans les registres (*_ce) ou en mémoire (ram_we) ainsi que les multiplexeurs (sel_*). Ces signaux de commandes sont:

- instr_ce,
- acc_ce,
- pc_ce,
- rpc_ce ,
- rx_ce ,
- ram_we ,
- sel_ram_addr,
- sel_op1,
- sel_rf_din

1. Décrire tout d'abord la machine d'états (FSM) de l'unité de commande en précisant pour chaque état les valeurs des sorties ( la µinstruction à envoyer au chemin de données)

2. Procéder à sa description en VHDL niveau RTL ( voir le chapitre 2) .

**3.4 Architecture globale du processeur**

Il s'agit ici de relier tous les composants (registres, ALU, contrôle) entre eux, en insérant multiplexeurs et registres correctement contrôlés aux endroits nécessaires. Il suffit en gros de suivre le schéma global du processeur donné précédemment.

un fichier .vhd sert alors de décrire l'architecture globale d'une façon structurelle en instanciant tous les composants déja décrit.