Description d'un Microprocesseur en VHDL

On se propose de réaliser un micro processeur simple. Celui-ci est principalement constitué d'une unité arithmétique et logique (ALU), d'une FSM et d'une mémoire, comme décrit sur la figure 1. Dans une première partie, nous décririons les différents blocs de cette architecture, puis, une fois assemblés, nous écrirons un programme que nous exécuterons sur ce microprocesseur.

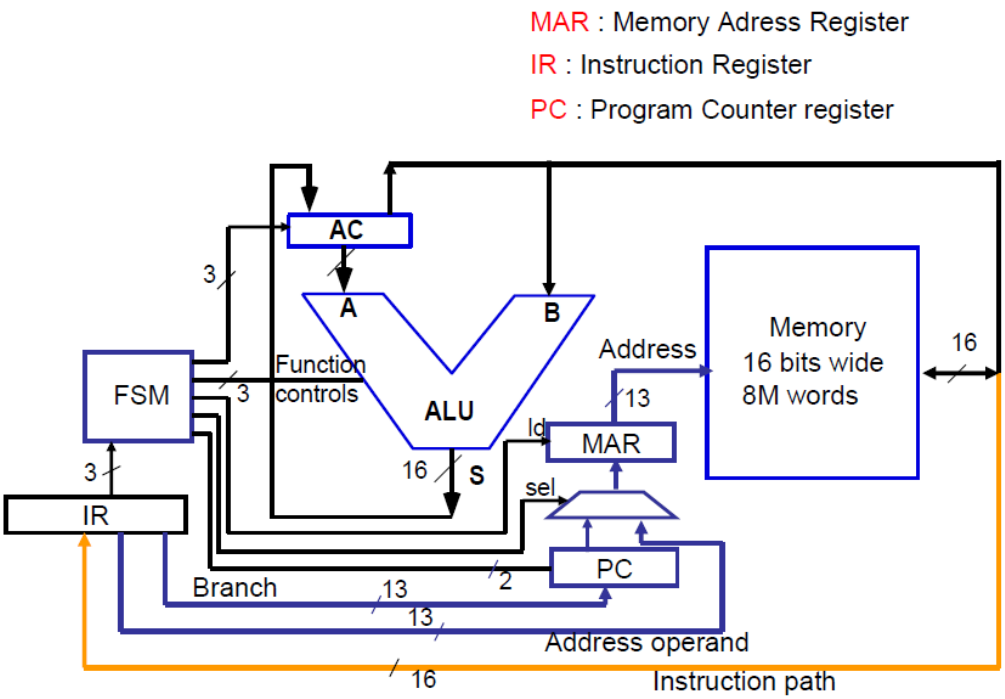

Figure 1 : description structurelle de l'architecture du microprocesseur

ALU

Pour décrire cette architecture, nous allons procéder de façon hiérarchique. Tout d'abord, nous réaliserons l'ALU représentée figure 2. Les 2 vecteurs A et B d'entrée auront une taille de 16 bits. La sortie de l'ALU sera donnée sur 16 bits (les 16 bits de poids faible). Celle-ci pourra, dans un premier temps, réaliser 3 opérations différentes : addition, soustraction et multiplication.

La fonction réalisée par cette ALU est déterminée par un vecteur d'entrée *fc_ctrl* de 3 bits. On utilisera le codage suivant :

- 000 pour une addition
- 001 pour une soustraction
- 010 pour une multiplication

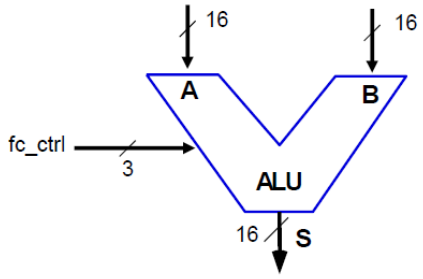

Figure 2 : schéma de l'unité arithmétique et logique

ALU + registre AC

La valeur de l'entrée A sera fournie par un registre d'accumulation AC (figure 3) de 16 bits. Ce registre permettra de stocker la valeur en sortie de l'ALU grâce à un signal d'écriture ld_AC. On pourra remettre ce registre à zéro grâce à un signal RAZ. Il comptera 2 sorties sur 16 bits. L'une d'entre elle sera directement reliée sur l'entrée A de l'ALU, la seconde sera commandée pas un signal store qui permettra soit de positionner la valeur stockée dans le registre sur cette sortie soit de maintenir cette sortie à l'état haute impédance 'Z'.

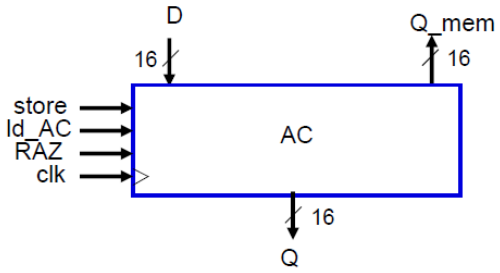

Figure 3 : schéma du registre d'accumultaion

Ecrivez le vhdl comportemental de ce registre, assemblez le avec l'ALU puis créez un testbench pour vérifier le fonctionnement de cet ensemble.

PC

Le registre PC (figure 4) comporte une entrée incr qui permet de l'incrémenter, une entrée branch sur 13 bits qui permet de faire des sauts dans la mémoire. L'entrée branch est stockée dans le registre lorsqu'on a un signal de chargement ld = '1' sur un front montant de l'horloge clk.

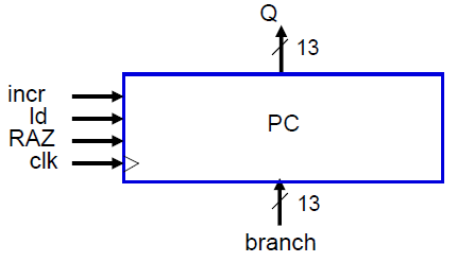

Figure 4 :schema du registre PC

Décrivez ce registre.

MAR

Ce registre (figure 5) prend un vecteur D de 13 bits en entrée. Le signal ld permet de transférer l'information en sortie du multiplexeur dans le registre MAR.

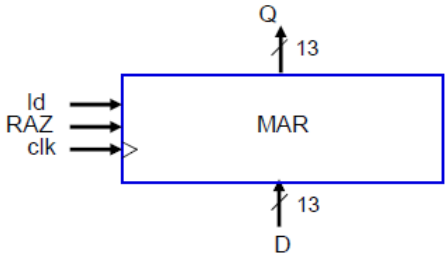

Figure 5 : schéma dur registre MAR

La commande *sel* en entrée du multiplexeur permet de choisir l'origine de la valeur en sortie du multiplexeur : soit du PC, soit du registre IR.

Décrivez l'ensemble registre + multiplexeur et créez un testbench pour l'évaluer.

IR

Le registre d'instruction (figure 6) reçoit des informations directement de la mémoire sur 16 bits. Celles-ci arrivent par l'instruction path sur l'entrée *D* de ce registre. Cette valeur est échantillonnée sur front d'horloge lorsque le signal *ld* vaut 1. Elle est ensuite redirigée (figure 7) sur les différentes sorties : les 3 bits de poids fort sur la sortie *Op_code*, les 13 bits de poids faibles sur les sorties *branch* et *Addr_op*.

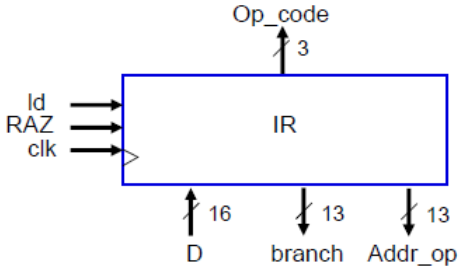

Figure 6 : schema du registre d'instruction

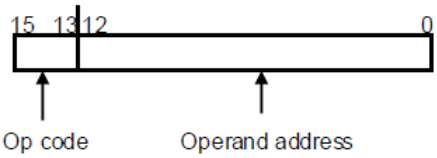

Figure 7 : format de l'information stockée dans IR

Décrivez le comportement de ce registre IR.

FSM

Afin de gérer le fonctionnement de ce microprocesseur, nous utiliserons une FSM.

Le premier état à définir est l'état d'initialisation. Cet état permettra de remettre le microprocesseur dans son état initial : RAZ des différents registres. Cette initialisation sera commandée par un signal *n_rst* actif bas. Une fois le signal *n_rst* à '1', le microprocesseur commencera à fonctionner.

Le premier état après l'initialisation sera un état de lecture d'instruction. Transfert de la valeur de PC dans MAR, récupération par IR de la valeur en sortie de la RAM, décodage dans IR et positionnement des signaux en sortie d'IR.

Le signal *op_code* de 3 bits en provenance de l'IR et à destination de la FSM déterminera la suite des évènements. Si on détaille les différents scénarios possibles, ils sont pour l'instant au nombre de 6, suivant la valeur d'*op_code* :

- 000 pour une addition
- 001 pour une soustraction
- 010 pour une multiplication
- 101 pour un load
- 110 pour un store
- 111 pour un branch

On aura donc une machine d'états de la forme de celle représentée figure 8

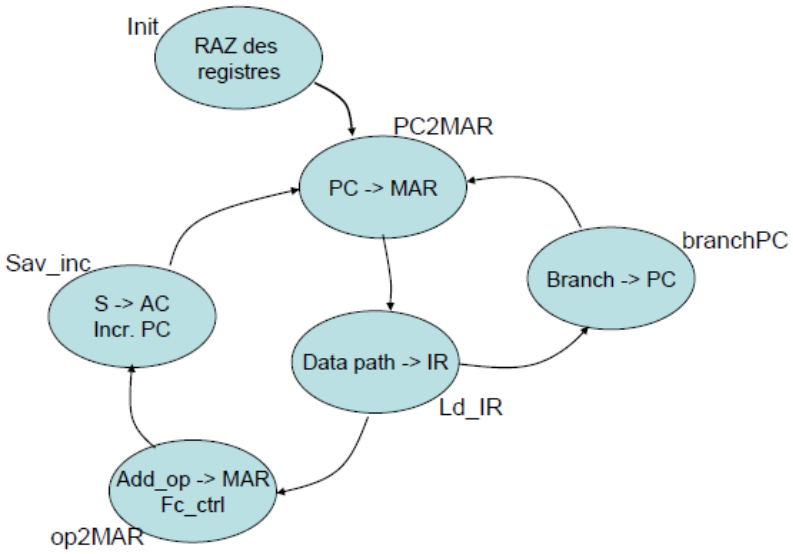

Figure 8 : description de la FSM

Cette FSM supporte les opérations d'addition, soustraction, multiplication et branch. A vous de la compléter si nécessaire pour implémenter des fonctions telles que load et store.

Un exemple de fonctionnement :

Le signal *rst_n* permet de positionner la FSM dans l'état initial. Les registres sont tous mis à zéro. Une fois le signal *rst_n* relaché, le fonctionnement commence. On passe dans l'état PC2MAR. La valeur stockée dans le PC est transmise dans le MAR. De manière asynchrone, la mémoire va, en fonction de l'adresse disponible en sortie du MAR positionner la valeur correspondante à cette adresse en sortie.
On passe ensuite dans l'état *ld_IR* au cours duquel la valeur en sortie de la mémoire est échantillonnée dans le registre IR et donc sur les sorties *branch*, *Addr_op* et *op_code*.

On détaille ici le fonctionnement d'une addition. La FSM passera donc dans l'état op2MAR au cours duquel la valeur de l'*op_code* positionnera l'ALU dans la configuration d'additioneur grâce au signal *fc_ctrl*. Simultanément, la valeur présente sur la sortie *Addr_op* de l'IR sera échantillonnée dans le MAR. De manière asynchrone, la mémoire positionnera sur sa sortie (et donc sur l'entrée *B* de l'ALU) la valeur correspondant à l'adresse stockée dans le MAR. L'addition est donc en cours puisque les données sont disponibles sur les entrées de l'ALU et elle est positionnée en mode additionneur.

On passera au coup d'horloge suivant dans l'état *Sav_inc*. Durant cet état, la valeur en sortie de l'ALU sera stockée dans l'AC, le PC sera incrémenté puisque l'opération d'addition est terminée. On retournera dans l'état PC2MAR pour aller lire une nouvelle instruction.

Procédez par étapes.
Dans un premier temps, décrivez la FSM pour qu'elle supporte les opérations, ne vous préoccupez pas du branch. Créez un testbench pour vérifier qu'elle fonctionne selon le cahier des charges. Une fois que tous les éléments fonctionnent séparément, assemblez-les et testez l'ensemble de l'architecture en utilisant le fichier ROM.vhd in https://seafile.lirmm.fr/d/8da14d7af28540948675/ → */Final_project_files/* à la place de la RAM. Attention, ce fichier ne vous permettra que de tester l'addition. L'écriture dans cette ROM est impossible en cours de fonctionnement. Vous pouvez éventuellement modifier ce fichier pour valider la soustraction la multiplication… etc… mais vous ne pouvez pas l'utiliser pour stocker des valeurs.
Implémentez l'opération branch dans votre FSM et testez-la.

Mémoire

Vous trouverez dans le répertoire */Final_project_files/* in https://seafile.lirmm.fr/d/8da14d7af28540948675/ un fichier RAMchip.vhd contenant le code d'une RAM. Téléchargez ce fichier dans votre répertoire de travail. Etudiez son fonctionnement et déterminez les entrées et les sorties de ce composant.

Vous trouverez aussi un fichier load_RAM.vhd et un fichier ram.dat dans le même répertoire. Ces fichiers vont vous permettre de charger la RAM à partir des données contenues dans le fichier subrt.txt, de sorte que la RAM ne soit pas vide lorsqu'on démarre le microprocesseur.

Créez un testbench permettant de tester ce composant puis intégrez-le à votre architecture.