Les composantes du processeur à 1 cycle

Ce document doit être lu en visualisant le circuit le plus complet du processeur à 1 cycle présenté en classe afin de bien comprendre les explications comprises dans ce document.

Les composantes principales

Le registre PC

Le registre PC, pour *Program Counter*, contient l'adresse de la prochaine instruction à exécuter. Lors d'un cycle, nous exécutons l'instruction stockée à l'adresse dans le registre PC et nous changerons ensuite la valeur de PC pour qu'elle corresponde à l'adresse de la prochaine instruction que nous exécuterons.

La mémoire d'instruction (*Instruction memory*)

Ce module accède à la mémoire pour aller chercher l'instruction stockée à l'adresse reçue du registre PC par l'entrée *Read address*. L'instruction, encodée sur 32 bits, est ensuite disponible pour son exécution. La sortie *Instruction* fournie les 32 bits de l'instruction.

Le contrôleur (*Control*)

Le contrôleur reçoit en entrée l'opcode de l'instruction et calcule les bits de contrôle à propager à travers le circuit. Les bits de contrôle vont permettre d'activer et de désactiver certaines parties du processeur.

Le banc de registres (*Registers*)

Le banc de registres contient les 32 registres programmables de MIPS. L'entrée *Read register 1* reçoit un signal sur 5 bits correspondant au numéro du registre à lire. La donnée lue dans le registre en question sera disponible à travers la sortie *Read data 1*. C'est toujours le champ *rs* (type R ou I) qui est envoyé dans l'entrée *Read register 1*. L'entrée *Read register 2* reçoit également un signal sur 5 bits correspondant au numéro du registre à lire. La donnée lue dans ce registre sera disponible à travers la sortie *Read data 2*. C'est toujours le champ *rt* (type R ou I) qui est envoyé dans l'entrée *Read register 2*. L'entrée *Write register* reçoit un signal sur 5 bits correspondant au numéro du registre où nous voulons sauvegarder une donnée sur 32 bits. Cette donnée sur 32 bits est fournie au banc de registres par l'entrée *Write Data*.

L'unité arithmétique et logique – UAL (*ALU*)

L'UAL prend en entrée deux données sur 32 bits chacune. Avec ces données, elle effectuera un calcul mathématique et le résultat, toujours sur 32 bits, sera disponible à travers la sortie *ALU result*. Lorsque *ALU Result* vaut 0, le bit *zero* vaut 1, dans tous les autres cas, le bit *zero* vaut 0. Le bit *zero* sert donc à indiquer que le résultat du calcul vaut bien la valeur 0. Ceci nous sera utile dans le cas de l'instruction *beq* pour déterminer si nous allons effectuer le branchement ou non.

La mémoire de données (Data memory)

La mémoire de données possède deux fonctions : (1) lire une donnée en mémoire; (2) écrire une donnée en mémoire. Que nous fassions une lecture ou une écriture en mémoire, il est indispensable de fournir l'adresse mémoire où nous devons faire la lecture ou l'écriture. Cette adresse mémoire est fournie à la composante par l'entrée *Address*. Dans le cas d'une lecture, la donnée lue à l'adresse en question sera disponible à travers la sortie *Read data*. Dans le cas d'une écriture, la donnée qui doit être écrite à l'adresse en question sera fournie à la composante à travers l'entrée *Write data*. Lors de l'exécution d'une instruction, il n'est possible que de faire un seul accès à la mémoire avec cette composante. Nous sommes donc limité à une lecture ou une écriture par instruction, il n'est pas possible de faire les deux en même temps.

Les composantes auxiliaires

Additionneur de la prochaine instruction (*Add* le plus à gauche)

Cet additionneur est une version simplifiée d'une UAL ne servant qu'à faire une addition. Elle prend en entrée l'adresse de l'instruction courante provenant du registre PC et la valeur 4. En additionnant l'adresse de l'instruction courante avec la valeur 4 (car les instructions sont encodées sur 4 octets), nous obtenons l'adresse de la prochaine instruction à exécuter. Cette nouvelle adresse sera sauvegardée dans le registre PC dans tous les cas sauf lors d'une instruction *j* (jump) ou lors d'une instruction *beq* si les valeurs des registres sont égaux.

Extension de signe (*Sign-extend*)

Ce module effectue une extension de signe, il prend un signal de 16 bits et le transforme en signal de 32 bits. La donnée qui est étendue sur 32 bits est la valeur immédiate encodée sur 16 bits directement dans l'instruction. Dans le cas de l'instruction *addi*, c'est la valeur à additionner qui sera encodée dans ce champ. Dans le cas de l'instruction *lw*, c'est l'indice à additionner à l'adresse de base qui sera encodée dans ce champ. Dans le cas d'un *beq*, c'est le nombre d'instructions à partir de PC+4 que nous voulons sauter en cas de branchement qui sera encodée dans ce champ.

Décallage à gauche de 2 bits (*Shift left 2*) du calcul du *Branch Target*

Le shift est situé devant l'additionneur du Branch Target. Ce shift n'est utile que pour l'instruction *beq*. Comme la valeur immédiate encodée dans l'instruction beq correspond à un nombre d'instructions à sauter, et comme les instructions sont encodées sur 4 octets, il est nécessaire de multiplier la valeur immédiate par 4 pour obtenir un nombre d'adresse mémoire à sauter pour atteindre l'instruction désirée par le branchement. Cette multiplication est faite en décallant les bits de la valeur immédiate à gauche de 2 bits.

Additionneur du *Branch Target* (*Add* le plus à droite)

Cet additionneur n'est réellement utile que lorsqu'on exécute une instruction beq. C'est l'UAL qui déterminera si le branchement sera effectué ou non (grâce au bit zero) mais pendant que l'UAL effectue ce calcul, nous optimisons le temps d'exécution en précalculant l'adresse du branchement potentiel. Cet additionneur prend en entrée l'adresse de la prochaine instruction et le nombre d'octets à sauter en mémoire pour atteindre l'instruction désirée en cas de branchement. En additionnant ces deux valeurs, nous obtenons l'adresse mémoire de l'instruction où nous désirons brancher.

Contrôleur de l'UAL (*ALU Control*)

Le contrôleur de l'UAL reçoit en entrée des bits de contrôle provenant du contrôleur. Lorsque l'opcode de l'instruction vaut 0x00 (instruction de type R), le contrôleur du processeur envoit un signal au contrôleur de l'UAL pour l'aviser qu'il doit consulter le champ *funct* de l'instruction pour déterminer quelle opération il doit faire faire à l'UAL, sinon les derniers bits de l'instruction sont ignorées par le contrôleur de l'UAL.

Les bits de contrôle

RegDst

Ce bit de contrôle est relié à un multiplexeur permettant de choisir le signal qui sera envoyé dans l'entrée *Write register* du banc de registres. Pour certaines instructions de type I, c'est le champ *rt* qu'on envoit dans *Write register*. Pour toutes les instructions de type R, c'est le champ *rd* qu'on envoit dans *Write register*.

Branch

Ce bit de contrôle n'est activé que pour l'instruction *beq*. Ce bit est destiné à être envoyé dans une porte ET avec le bit *zero* de l'UAL. Le résultat de la porte ET va être vrai uniquement si l'on doit sauvegarder le *Branch Target* dans le registre PC. D'ailleurs, ce signal (le résultat de la porte ET) contrôle un multiplexeur permettant de diriger vers le registre PC l'adresse de l'instruction suivante ou l'adresse du *Branch Target*.

MemRead

Indique à la mémoire de données que l'on doit faire une lecture en mémoire.

MemToReg

Ce bit de contrôle est relié à un multiplexeur permettant de choisir la provenance de la donnée qui sera envoyé dans le *Write data* du banc de registres. Dans le cas d'opérations arithmétiques, c'est la valeur calculée par l'UAL que l'on veut sauvegarder dans un registre. Dans le cas d'une opération de chargement, c'est la donnée lue en mémoire que l'on veut sauvegarder dans un registre.

ALUOp

Ces bits de contrôle sont envoyés au contrôleur de l'UAL. Entre autres, c'est avec ces bits de contrôle que le contrôleur de l'UAL va être en mesure de déterminer s'il doit considérer le champ *funct* ou non.

MemWrite

Indique à la mémoire de données que l'on doit faire une écriture en mémoire.

ALUSrc

Ce bit de contrôle est relié à un multiplexeur qui permet de choisir la deuxième opérande envoyée dans l'UAL. Dans certains cas, nous voulons envoyer dans l'UAL la valeur d'un registre. Dans d'autres cas, nous voulons envoyer dans l'UAL une valeur immédiate étendue sur 32 bits.

RegWrite

Ce bit de contrôle indique au banc de registres s'il doit effectuer une écriture dans un registre.

Jump

Ce bit de contrôle est relié à un multiplexeur qui permet de choisir la valeur finale qui sera envoyée dans le registre PC. Ce bit n'est activé que lorsqu'on exécute une instruction *j*. Le multiplexeur permet donc de choisir entre l'adresse calculée du jump ou l'autre adresse calculée de la prochaine instruction à exécuter (qui peut être PC+4 ou le *Branch Target*).

Le calcul de l'adresse d'un jump

L'encodage de type J ne permet d'avoir que 26 bits pour stocker l'adresse du saut à effectuer. Comme nous devons toujours sauter à une adresse d'instruction et comme toutes les instructions en MIPS sont encodées sur 4 octets, toutes les adresses d'instructions vont toujours avoir les mêmes deux bits les moins significatifs, c'est-à-dire 00. Comme ces bits sont toujours les mêmes, ils sont donc omis lors de l'encodage de l'instruction.

Lorsque le circuit veut reconstruire l'adresse d'origine à partir des 26 bits dans l'instruction, il doit ajouter les deux bits qui ont été omis au départ. Ceci est fait à l'aide d'un décallage à gauche de 2 bits (le *Shift left 2* le plus à gauche du circuit). Nous nous retrouvons ainsi avec les 28 bits les moins significatifs de l'adresse du saut. Il nous manque les 4 bits les plus significatifs pour avoir l'adresse complète du saut. Ces 4 bits sont pris de l'adresse calculée de PC+4.