TD1

Processeur « Craps » : architecture minimale

0- Rappels

Le rôle d'un processeur est d'exécuter des programmes : un ensemble d'instructions machine. Chaque instruction machine est représentée par un code binaire, de 32 bits sur craps. Mais comme il n'est pas aisé pour un humain de travailler en binaire, on utilise un langage intermédiaire appelé « assembleur » que l'on découvrira progressivement. Mais on commence dans un premier temps d'exprimer nos instructions en langage algorithmique.

Pour illustrer cela, on s'intéresse à l'exemple suivant : additionner 4 valeurs différentes que l'on peut exprimer en langage algorithmique sous la forme suivante : S ← V1 + V2 + V3 + V4.
C'est l'équivalent d'une instruction en langage évolué.

Mais notre processeur ne dispose pas d'instruction de ce type : il ne peut exécuter que des instructions élémentaires du type : S <- V1 + V2. Ce qui nous conduit à écrire notre programme comme suit : S ← V1 + V2
S ← S + V3
S ← S + V4
C'est ce que fait le compilateur en traduisant une instruction en langage évolué en un ensemble d'instructions en « assembleur », puis en langage machine.

Une étape importante consiste à choisir les récipients dans lesquels on stocke nos opérandes :

- Un opérande peut être une constante que l'on met directement dans le code de l'instruction (sera étudié plus tard),
- Un opérande peut être mis dans un registre : emplacement de mémoire interne, à accès rapide, et référencé par son nom, en réalité son numéro (r0, r1, …)
- Un opérande peut être en mémoire : mode non utilisé dans craps

Dans notre exemple, on choisira des registres pour stocker nos opérandes S, V1, V2, V3, et l'on pourra écrire :

	code binaire	**code hexadécimal**
R5 ← R1 + R2	1000 1010 0000 0000 0100 0000 0000 0010	8a004002
R5 ← R5 + R3	1000 1010 0000 0001 0100 0000 0000 0011	8a014003
R5 ← R5 + R4	1000 1010 0000 0001 0100 0000 0000 0100	8a014004

Comme nous le verrons plus tard, le code binaire permet d'indiquer au processeur les différents éléments constitutifs de l'instruction. Par exemple, pour la première instruction les champs grisés codent dans l'ordre, le registre R5, l'addition, le registre R1 et le registre R2 :
10 00101 000000 00001 00 0000 000 00010

Pour exécuter ce programme, nous avons besoin :

- D'une RAM pour stocker les codes des instructions
- De registres pour contenir les opérandes
- D'une unité de calcul pour effectuer les opérations

1- Architecture générale

La figure ci-dessous présente l'architecture générale du processeur « craps » que l'on propose d'étudier, de réaliser et de tester durant les 3 séances de TD et 6 séances de TP de ce module.

Ce processeur se compose principalement de 3 modules :
A- L'UAL, Unité Arithmétique et Logique qui sera fournie en TP, a pour rôle de réaliser les opérations arithmétiques (addition, soustraction) et logiques (ET, OU, …) :

module ual (a[31..0], b[31..0], cmd[5..0] : s[31..0], enN, enZ, enVC, N, Z, V, C)

- Les entrées « a » et « b » représentent les opérandes
- L'entrée « cmd » indique le code l'opération à réaliser (2^6=64 opérations possibles) :
  - Addition : ADD (''000000 ''), ADDCC (''010000 '')
  - Soustraction : SUB (''000100 ''), et SUBCC (''010100 '')
  - etc.
- La sortie « s » fournit le résultat de l'opération commandée
- Les indicateurs N, Z, V et C indiquent l'état du résultat :
  - N = 1 lorsque le résultat « s » est négatif
  - Z = 1 lorsque le résultat « s » est égal à 0
  - V = 1 lorsque une addition ou une soustraction engendre un débordement sur le bit de signe (voir addsub32 réalisé en semestre 5)
  - C = 1 lorsque l'addition engendre une retenue, et lorsque la soustraction engendre un emprunt (voir addsub32 réalisé en semestre 5)
- Seules certaines opérations demandent la prise en compte des indicateurs en mettant à 1 les sorties enN, enZ, et enVC : ADDCC, SUBCC, …
- Les sorties enN, enZ, et enVC, lorsqu'elles sont à 1, permettent de mémoriser, respectivement, N, Z, V et C dans des bascules.

B- Le module « Registres » : peut contenir jusqu'à 32 registres numérotés de 0 à 31 :

- Des registres banalisés, que l'on appellera r0, r1, …, r5, destinés à contenir les opérandes
- Deux registres constants (non modifiables) : R0 = 0, R20 = 1
- Un registre tmp (R21) à usage interne(non utilisable par le programmeur)
- Des registres spécialisés, destinés à un usage précis tels que :
  - PC (Programm Counter, r30) : contient l'adresse de l'instruction courante
  - IR (Instruction Register, r31) : contient le code de l'instruction courante
  - etc.

Ecrire le code shdl du module reg32 qui permet de mémoriser une entrée « e » codée sur 32 bits, et donner en sortie le contenu du registre « reg ».

Le module registres (qui sera fourni en TP), possède l'interface suivante :
registres(rst,clk, areg[4..0],breg[4..0],dreg[4..0], dbus[31..0] : abus[31..0], bbus[31..0], ir[31..0])

- l'entrée areg indique le numéro du registre dont on veut lire le contenu sur la sortie abus
- l'entrée breg indique le numéro du registre dont on veut lire le contenu sur la sortie bbus
- l'entrée dreg indique le numéro du registre dans lequel on souhaite enregistrer la valeur fournie sur l'entrée dbus.

Ecrire le début du code shdl du module registres (implantation de r1)

C- La RAM (Random Access Memory) : mémoire dans laquelle seront stockées les instructions et les données du programme. Ici, elle a la forme d'un tableau de 512 mots de 32 bits chacun. C'est un module prédéfini ayant l'interface suivante :

$ram_aread_swrite(wrclk, write, addr[8..0], data_in[31..0] : data_out[31..0])

- l'entrée « addr » indique l'adresse dont on veut lire ou modifier le contenu
- l'entrée « data_in » indique la valeur que l'on souhaite écrire à l'adresse courante (addr)
- la sortie « data_out » fournit le contenu de l'adresse courante (« addr »)
- la lecture est asynchrone : le contenu de l'adresse « addr » est disponible immédiatement sur la sortie « data_out »
- l'écriture est synchrone : l'entrée « data_in » est enregistrée dans l'adresse « addr » sur le front montant de l'horloge « wrclk » lorsque l'entrée « write » est à 1 (ordre d'écriture).

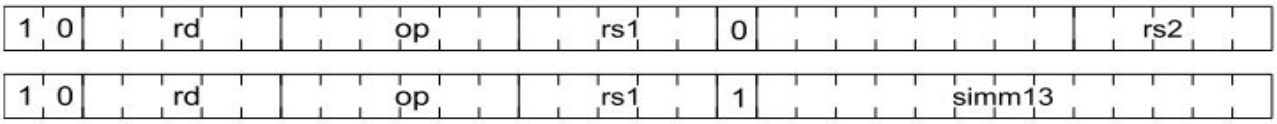
Les 3 modules précédents sont reliés entre eux par des bus :
- **Les sorties abus et bbus** du module registres sont connectées aux entrées « a » et « b » de l'ual, et permettent d'y acheminer les opérandes
- **La sortie abus (9 bits de faible poids)** du module registres est connectée à l'entrée « addr » de la RAM, et permet d'indiquer l'adresse dans laquelle on veut lire ou écrire
- **dbus** (bus de données) sert à acheminer :
  - la sortie de l'ual (dbus_ual) vers le module registre ou la mémoire, pour enregistrer une donnée
  - la sortie de la ram vers le module registres pour enregistrer une donnée lue en mémoire dans un registre
  - on ne s'intéresse pas dans un premier temps aux entrées sorties (sw, led et moniteur)
  - l'entrée sur dbus doit être exclusive (sinon court-circuit) : les signaux (triangles blancs) oe[1], oe[2], oe[3] servent à assurer cette exclusivité et sont générés par un décodeur à partir de l'entrée oe_num[1..0]
  - les signaux cs (chip select) sont générés par un décodeur à partir des 4 bits de fort poids de l'adresse, et permettent de distinguer les adresses des différents composants :
    - RAM : adresses commençant par 0x0…
    - Switches : adresses commençant par 0x9…
    - Leds : adresses commençant par 0xB…

2- Codage des instructions

Craps gère différents groupes d'instructions, codées chacune sur 32 bits. Ces 32 bits doivent permettre de coder les instructions et leurs arguments.

- On choisit de différencier ces groupes d'instructions par l'intermédiaire des 2 bits de fort poids : la configuration ''10'' sera réservée aux instructions arithmétiques et logiques. Ces dernières possèdent deux formats algorithmiques :
  1. Rd ← Rs1 op Rs2 //R : registre, d : destination, op : opération, s : source
  2. Rd ← Rs1 op Constante
- 1 bit sera utilisé pour différencier le format 1 (0) du format 2 (1)
- Ayant un jeu de 32 registres maximum, le numéro du registre doit être codé sur 5 bits : cela consomme 15 bits pour le format 1 (rd, rs1 et rs2), et 10 pour le format 2 (rd et rs1)
- Le code de chaque instruction (op) consomme 6 bits : cela fait au total, 24 bits pour coder le format 1 (2+1+5*3+6) et 19 bits pour le format 2 sans la constante (2+1+2*5+6). Ce qui laisse 13 bits pour code la constante du format 2 (32-19).

On choisit de coder ces instructions selon les formats suivants :



3- Schéma d'exécution d'un programme

L'exécution d'un programme est gérée par l'intermédiaire des deux registres :

- PC (Program Counter, r30) : contient l'adresse de l'instruction courante
- IR (Instruction Register, r31) : contient le code de l'instruction courante

Le cycle d'exécution d'une instruction arithmétique ou logique de format 1 prend la forme suivante :

1. Lecture du code de l'instruction depuis la mémoire (à l'adresse contenue dans PC) et son enregistrement dans le registre IR. On notera : IR ← [PC]
2. L'exécution de l'opération dans l'ual et l'enregistrement du résultat dans le registre destination
3. L'incrémentation du PC pour passer à l'instruction suivante

L'exécution de chacune de ces étapes est commandée par les 5 microcommandes suivantes :

- areg pour indiquer le numéro du registre lu sur abus
- breg pour indiquer le numéro du registre lu sur bbus
- cmd_ual pour indiquer l'opération que l'on veut exécuter dans l'ual
- oe-num pour indiquer la source de la donnée entrée sur dbus (UAL, RAM, …)
- dreg pour indiquer le numéro du registre dans lequel on souhaite enregistrer le résultat

La sixième microcommande « write » sera toujours mise à 0 sauf pour l'écriture en mémoire, qui sera traitée plus tard.

Dessiner le graphe d'états qui permet de représenter ce cycle d'exécution, en indiquant pour chaque transition, les conditions et l'opération réalisée (parmi les 3 dessus). L'état de départ sera appelé « fetch » et l'état suivant appelé « decode ». Lors de la transition de « fetch » à « decode » (sans condition), l'opération IR ← [PC] sera effectuée en mettant :

- 30 dans areg (adresse de l'instruction dans PC),
- 0 dans breg (non utilisé),
- 31 dans dreg (résultat enregistré dans IR),
- 0 dans cmd_ual (non utilisée),
- 2 dans oe_num pour faire entrer la sortie de la ram dans dbus.

Compléter le tableau suivant :

Transition	Condition	areg	breg	dreg	cmd_ual	oe_num
Fetch → decode	1	30	0	31	0	2

Implanter ce séquencement en utilisant une bascule D pour chaque état ("one-hot encoding").