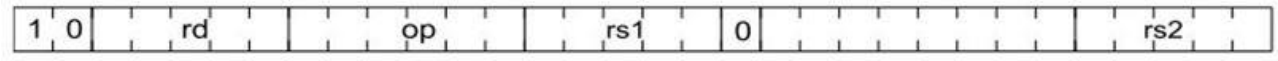
Architecture des ordinateurs - TP1

Processeur « Craps » : architecture minimale

Un module craps (version minimale) est fourni sur moodle. Il permet d'exécuter des instructions arithmétiques et logiques de type Rd ← Rs1 op Rs2 (Rd, Rs1 et Rs2 sont des registres, op peut être une addiction, une soustraction, un ET logique, etc.). Le format du code de ces instructions est le suivant (32 bits) :



1- Lire le code shdl du module craps et le comprendre (étudié en TD). Cette étape est très importante pour la suite, car ce code sera complété durant les TP suivants.

2- Construire craps sur la plateforme shdl en y implantant les modules fournis.

3- Enregistrer le code suivant dans un fichier « additions.txt » sous la forme suivante (chaque ligne contient l'adresse et le code de l'instruction qui y est enregistré) :

```
[
        {
                "000000000": "10001010000000000100000000000010",
                "000000001": "10001010000000010100000000000011",
                "000000010": "10001010000000010100000000000100"
        }
]
```

A- Décoder ces 3 instructions et les écrire sous forme algorithmique.

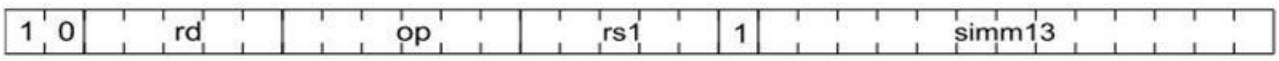
B- Sur le simulateur, importer ce fichier dans la mémoire (bouton jaune en haut à droite). Puis enregistrer les valeurs 1, 2, 3 et 4 respectivement dans les registres r1, r2, r3 et r4.

- mon_mode=1 : craps est en mode moniteur
    - écriture dans un registre : mon_reg=numéro du registre, monitor=valeur sur clk
    - lecture d'un registre : mon_reg=numéro du registre, registre affiché sur abus

C- Passer en mode exécution (mon_mode=0) et cliquer sur l'horloge (clk) autant de fois que nécessaire, pour exécuter ces instructions. Vérifier le passage dans les différents états. Vérifier à la fin de chaque instruction, le résultat dans le registre destination (mon_mode=1, mon_reg=numéro du registre, abus=contenu du registre).

3- Ecrire le code binaire du programme qui calcule la somme des 3 premiers entiers, en utilisant R1 pour stocker la valeur initiale 1, et R2 pour contenir le résultat (penser à utiliser les registres constants R0=0 et R20=1). Tester ce programme.

4- Les instructions arithmétiques et logiques possèdent un second format :



Il permet d'exécuter des instructions de type de type Rd ← Rs1 op constante13 (constante13 ou simm13 représente une valeur comprise entre -4096 et +4095).

A- compléter la partie « séquenceur » de craps en implantant la gestion de ce format d'instructions. Penser à regarder dans le module ual (descriptif dessous) pour choisir l'instruction dont vous aurez besoin (extraction de la constante depuis le registre IR).

B- transformer le programme du point 3 en n'utilisant qu'un seul registre pour stocker le résultat. Tester ce programme.

UAL : Unité Arithmétique et Logique

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

L'ual permet de réaliser les opérations arithmétiques et logiques nécessaires au fonctionnement du processeur CRAPS.

- Les entrées a et b représentent les opérandes
- L'entrée cmd indique l'opération à réaliser : addition, soustraction, …
- La sortie s présente le résultat
- Les sorties N, Z, V et C indiquent l'état du résultat : négatif, égal à 0, V et C générés par l'addition (retenue finale) et la soustraction (emprunt final)
- Les sorties enN et enZ sont activées par les instructions ADDCC, SUBCC, ANDCC, ORCC et XORCC
- La sortie enVC est activée par les instructions ADDCC et SUBCC
- Certaines instructions se trouvent en deux versions : par exemple, l'instruction ADDCC effectue la même opération que l'instruction ADD, mais active en plus enN, enZ, et enVC pour indiquer qu'elle souhaite que les flags N, Z, V et C soient pris en compte (mémorisés à l'extérieur de l'UA). Même chose pour SUBCC, ANDCC, …
- L'affectation 3 états est utilisée pour la sortie s :
    - On écrit : s = s1 output enabled when c1
s = s2 output enabled when c2
au lieu de : s = s1 * c1 + s2 * c2
    - Cela facilite la lisibilité et améliore le contrôle de l'exclusivité entre les mintermes
    - Mais introduit un 3ème état (indéterminé) pour s

cmd[5..0]	**Opération**	**Flags validés**
000000 (0)	ADD, addition	aucun
000100 (4)	SUB, soustraction	aucun
000001 (1)	AND, et logique bit à bit	aucun
000010 (2)	OR, ou logique bit à bit	aucun
000011 (3)	XOR, xor logique bit à bit	aucun
010000 (16)	ADDCC, addition	N, Z, V, C
010100 (20)	SUBCC, soustraction	N, Z, V, C
010001 (17)	ANDCC, et logique bit à bit	N, Z
010010 (18)	ORCC, ou logique bit à bit	N, Z
010011 (19)	XORCC, xor logique bit à bit	N, Z
100000 (32)	SIGNEXT13, extension signée de l'entrée a[12..0] vers s[31..0] : s[31..0]<- a[12..0] étendu à 32 bits	aucun
100001 (33)	SIGNEXT25, extension signée de l'entrée a[24..0] vers s[31..0] : s[31..0]<- a[24..0] étendu à 32 bits	aucun
100011 (35)	SETHI, s[31..8]<- a[23..0], , s[7..0] <- 0	aucun
101000 (40)	NOPB, No OPeration sur l'entrée B, s[31..0]<- b[31..0]	aucun