Blask
Jeu d'Instructions

Vahan Boghossian Kimbembe Malanda Benjamin Peter
Ilan Schemoul

22 janvier 2023

Chapitre 1

Introduction

Ce document contient toutes les informations sur le jeu d'instructions de l'architecture Blask.

Le **jeu d'instructions** définit toutes les **instructions supportées** par le **processeur**. L'architecture Blask étant assez basique et simple d'utilisation, celle-ci dispose seulement **d'opérations élémentaires** (Addition, Soustraction, ET/OU logique, etc...) qui pourront être exécutées par notre processeur.

Il contient également des informations sur les **registres accessibles et manipulables** par le programmeur. Les registres sont des **emplacements de mémoire** au plus proche du processeur, que l'utilisateur va pouvoir modifier afin de pouvoir interagir avec le processeur.

Chapitre 2

Registres

L'architecture Blask est composé de **16 registres**.

Un **registre** permet de contenir des informations **accessibles et manipulables** par le programmeur.

Chaque registre a une taille de **16 bits** (soit 1 byte sur cette architecture). Le programmeur dispose donc de 16 registres de 16 bits, permettant un total de 256 bits de données au plus proche du processeur.

Les registres sont définies par un **nom de registre** ainsi que leur **numéro** associé.

Register	*Numéro*
Z	0
A	1
B	2
C	3
D	4
E	5
F	6
G	7
H	8
I	9
J	10
K	11
L	12
M	13
N	14
S	15

Le registre **Z** est le registre 0, il a toujours la valeur 0 dans ce registre.

Chapitre 3

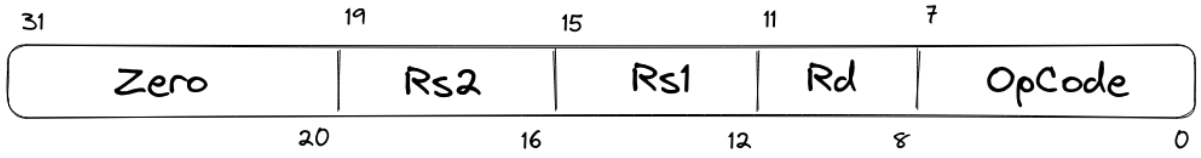
Instructions

Une **instruction** est une séquence de **32 bits**. Cette séquence est découpé en 1 **OpCode** ainsi qu'en plusieurs parties que l'on appelle **Opérande**. Chaque type d'instructions disposent de nombre d'opérandes différents.

L'architecture est séparé en plusieurs types d'instructions.

3.1 R-Type Instruction

Les **R-Type** Instructions sont des instructions qui vont effectuer des **opérations entre les registres**.



Opcode indique quel est le type précis de l'instruction.

OpCode	Valeur (en bits)
ADD	0000_0000
SUB	0001_0000
OR	0010_0000
AND	0011_0000
XOR	0100_0000
SLL	0110_0000
SRL	0111_0000

Rd, **Rs1** et **Rs2** sont des registres. Ceux-ci sont encodés sur 4 bits car on possède 16 registres disponibles sur notre architecture. Les valeurs possibles sont : $valeur \in [0, 15]$

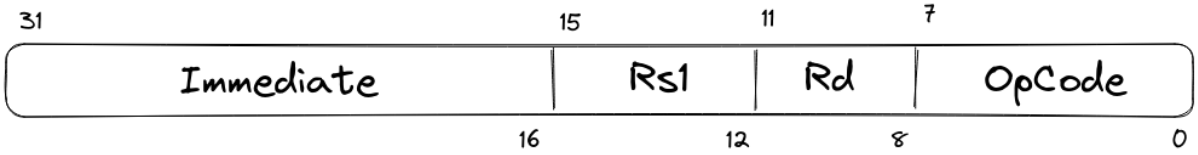
Rs1 (Registre Source 1) et **Rs2** (Registre Source 1) sont les 2 registres qui vont être utilisés pour l'opération. Le résultat sera alors stockés dans **Rd** (Regsitre Destination).

Zero indique que les bits de 20 à 31 sont tous à zéro.

On peut voir cette instruction comme : $Rd = Rs1\ OP\ Rs2$.

3.2 I-Type Instruction

Les **I-Type** Instructions sont des instructions qui vont effectuer des **opérations entre les registres et un immediat**.



Opcode indique quel est le type précis de l'instruction.

OpCode	Valeur (en bits)
ADDI	0000_0001
SUBI	0001_0001
ORI	0010_0001
ANDI	0011_0001
XORI	0100_0001
SLLI	0110_0001
SRLI	0111_0001

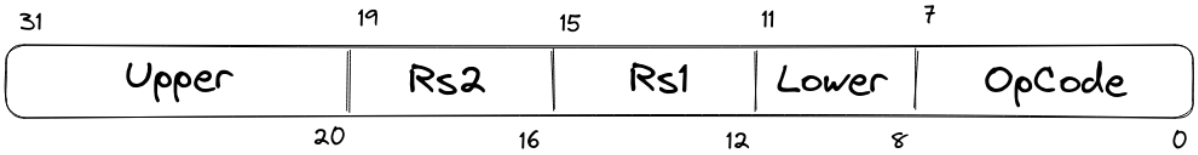
Immediate correspond à un entier signé sur 16 bits.

Rs1 et **Rd** ont toujours la même fonction. L'opération est effectué sur l'**immediat** ainsi que sur la valeur dans le registre **Rs1**, le résultat est stocké dans **Rd**.

On peut voir cette instruction comme : $Rd = Rs1\ OP\ Immediate$.

3.3 B-Type Instruction

Les **B-Type** Instructions sont des instructions qui vont effectuer des c**omparaisons entre des registres** et effectuer des **saut entre les instructions du programme** en fonction du résultat de la comparaison.



Opcode indique quel est le type précis de l'instruction.

OpCode	Valeur (en bits)
BE	0000_0010
BNE	0001_0010
BLT	0100_0010
BGE	0101_0010
BLTU	0110_0010
BGEU	0111_0010

Upper et **Lower** vont être combiné afin de former un entier signé sur 16 bits qui va nous indiquer la direction ainsi que le nombre d'instructions que l'on doit sauter lorsque

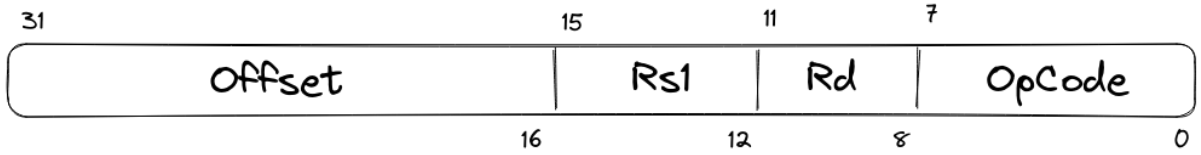
la comparaison est vraie.

Rs1 et **Rs2** sont utilisés pour la comparaison.

On peut voir cette instruction comme : $if\ Rs1\ OP\ Rs2\ alors\ jump\ addr\_courante + ((Upper << 4) + Lower)$.

3.4 Load/Store Instruction

Les **Load/Store** Instructions sont des instructions qui vont permettre d'**échanger des données entre la mémoire et les registres.**



Opcode indique quel est le type précis de l'instruction.

OpCode	**Valeur (en bits)**
LD	0000_0011
STR	0001_0011

L'instruction **Load** correspond à un chargement en registre de donnée en mémoire.
$mémoire[rd + offset] = rs1$

L'instruction **Store** correspond à un déplacement d'une valeur d'un registre dans la mémoire.
$rd = mémoire[rs1 + offset]$