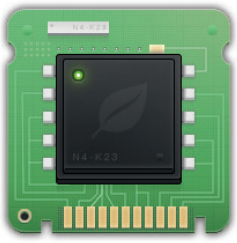

Projet LC-3

Lefranc Joaquim, Skoda Jérôme

Tableaux de Karnaugh

Table AddUnit

	00	01	11	10
00	No	No	No	No
01	No	No	No	No
11	Yes	No	No	Yes
10	No	No	No	No

Table DOUBLE INTERUPT

	00	01	11	10
00	No	No	No	No
01	No	No	No	No
11	No	No	No	No
10	No	No	Yes	Yes

Table JUMP

	00	01	11	10
00	Yes	No	No	No
01	Yes	No	No	No
11	Yes	No	Yes	No
10	Yes	No	No	No

Table LOAD

	00	01	11	10
00	No	No	No	Yes
01	No	No	No	Yes
11	No	No	Yes	No
10	No	No	Yes	Yes

Table LoadMem

	00	01	11	10
00	No	No	Yes	Yes
01	No	No	Yes	Yes
11	No	No	Yes	No
10	No	No	Yes	Yes

Table STORE

	00	01	11	10
00	No	No	Yes	No
01	No	No	Yes	No
11	No	No	No	No
10	No	No	Yes	No

Table WriteR7

	00	01	11	10
00	No	No	No	No
01	Yes	No	No	No
11	No	No	Yes	No
10	No	No	No	No

Table WriteReg

	00	01	11	10
00	No	Yes	No	Yes
01	Yes	Yes	No	Yes
11	No	Yes	Yes	Yes
10	-	Yes	No	Yes

Table ARG1

13, 12 / 15, 14	00	01	11	10
00	PC	-	PC	PC
01	JSR: PC JSRR: BaseR	-	BaseR	BaseR
11	BaseR	-	Vide	PC
10	-	-	PC	PC

Table ARG2

13, 12 / 15, 14	00	01	11	10
00	9	-	9	9
01	JSR: 11 JSRR: Vide	-	6	6
11	Vide	-	9 ou 11	9
10	-	-	9	9

AddUnit :

*Fait toutes les additions d'**offset/PC/BaseR** en fonction de ce qu'a besoin l'instruction en cour.*

Exemple:
***JSR:** PC + Offset11*
***STR:** BaseR + Offset6*

Il est composé d'un additionneur prenant deux arguments.
Les équations en logique combinatoire ont été faite à partir de ce tableau (Voir tableau).

L'ALU n'utilise pas l'addUnit donc nous avons choisi les argument qui nous arrange le plus pour les équations logique:

Arg1:
Offset11 = JSR
Offset9= $\overline{14}$ + 13.15
Offset6= 13.14.$\overline{15}$
0= $\overline{13}$.14.$\overline{JSR}$

Arg2:
PC= $\overline{14}$+$\overline{12}$.13.15+JSR
BaseR=$\overline{13}$.15+14.$\overline{15}$.$\overline{JSR}$
0=12.14.15

DecodeIR :

Choisi les actions à effectuer en fonction de l'instruction en cour.

***LoadMem:** Selectionne MemOUT dans le writeSelector*
LoadMem=12.13+13.$\overline{15}$+13.$\overline{14}$

***LoadAddUnit:** Selectionne le resultat calculé par le addUnit dans le writeSelector*
LoadAddUnit=$\overline{12}$.14.15

***WriteR7:** Indique l'on ecrit dans le Registre n°7 (prioritaire sur BaseR) et selectionne PC dans le writeSelector*
WriteR7=12.13.14.15+$\overline{12}$.$\overline{13}$.14.$\overline{15}$

Arith: *Selectionne le resultat de l'ALU dans le writeSelector*
Arith=12.$\overline{13}$ (AND, ADD, NOT, RETB SETB)

WriteReg: *Indique que l'on ecrit dans le registe*
WriteReg=12.$\overline{13}$+$\overline{12}$.13+12.14.15+$\overline{13}$.14.$\overline{15}$

Jump: *Indique au RegPC que l'on jump*
Jump=$\overline{12}$.$\overline{13}$+12.13.14.15

UseCondition: *Indique au RegPC que l'on utilise les condition nzp*
UseCondition=$\overline{12}$.$\overline{13}$.$\overline{14}$.$\overline{15}$ (juste BR)

Load: *Indique au RAMCtrl (uniquement) que l'on va lire une valeur*
Load=$\overline{12}$.13.$\overline{15}$+12.$\overline{13}$.$\overline{14}$.$\overline{15}$+13.$\overline{14}$.15

Store: *Indique au RAMCtrl (uniquement) que l'on va stoquer une valeur*
Store=12.13.$\overline{15}$+12.13.$\overline{14}$.15

DoubleIndirection: *Indique au RAMCtrl ET au GetAddr que l'on va faire une double indirection*
DoubleIndirection=13.$\overline{14}$.15

WriteSelector :

Permet de choisir ce que l'on ecrit dans le registre.

Valeur possible:
- Resultat de l'alu
- PC
- Memout
- Resultat de l'addUnit

RegPC :

Gestion du PC

A chaque front descendant de exec:
Si *jump . UseCondition . TestNZP + jump . $\overline{\text{UseCondition}}$*
PC <- Resultat add Unit
Sinon
PC <- PC+1

GetAddr :
Choisi l'adresse de la RAM à lire ou ecrire

Valeur de sorti:

	Si DoubleIndirection	***Sinon***
Exec	*regAddrIndirection*	*resultat addUnit*
Post Exec	*PC*	*PC*
Fetch	*PC*	*PC*
Post Fetch	*resultat addUnit*	*resultat addUnit*

Pour effectuer la double indirection, un registre est utilisé contenant l'addr.

Au front descendant de post-Fetch . DoubleIndirection:
regAddrIndirection <- MemOUT

RamCtrl :

Active / Desactive l'ecriture ou la lecture de la RAM

	Lecture	***Ecriture***
Exec	*Si load*	*Si Store*
Post Exec	*0*	*0*
Fetch	*1*	*0*
Post Fetch	*Si DoubleIndirection*	*0*

NZP :

Calcule du NZP:
N: $RES[15] \cdot \overline{Z}$
Z: $Or(RES[15..0])$
P: $\overline{RES[15]} \cdot \overline{Z}$

Test de saut:
testN$= IR[11] \cdot N$
testZ$= IR[10] \cdot Z$
testP$= IR[9] \cdot P$
testNZP$= testN + testZ + testP$

Set/Reset bit :

Si set:
$Out = DataIn + (1 << bit\ n)$
Sinon *(si reset)*
$Out = DataIn \cdot \overline{(1 << bit\ n)}$