

# Projet LC-3

Lefranc Joaquim, Skoda Jérôme



## Tableaux de Karnaugh

*AddUnit*

IR [13, 12] \ IR [15, 14]	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	1	0	0	1
10	0	0	0	0

*Double indirection*

IR [13, 12] \ IR [15, 14]	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	0	0	0	0
10	0	0	1	1

*Jump*

IR [13, 12] \ IR [15, 14]	00	01	11	10
00	1	0	0	0
01	1	0	0	0
11	1	0	1	0
10	1	0	0	0

*Load*

IR [13, 12] \ IR [15, 14]	00	01	11	10
00	0	0	0	1
01	0	0	0	1
11	0	0	1	0
10	0	0	1	1

## LoadMem

IR [13, 12]					
IR [15, 14]		00	01	11	10
		00	01	11	10
00	0	0	0	1	1
01	0	0	0	1	1
11	0	0	0	1	0
10	0	0	0	1	1

## Store

IR [13, 12]					
IR [15, 14]		00	01	11	10
		00	01	11	10
00	0	0	0	1	0
01	0	0	0	1	0
11	0	0	0	0	0
10	0	0	0	1	0

## WriteR7

IR [13, 12]					
IR [15, 14]		00	01	11	10
		00	01	11	10
00	0	0	0	0	0
01	1	0	0	0	0
11	0	0	0	1	0
10	0	0	0	0	0

## WriteReg

IR [13, 12]					
IR [15, 14]		00	01	11	10
		00	01	11	10
00	0	1	0	1	
01	1	1	0	1	
11	0	1	1	1	
10	-	1	0	1	

## Arg1

IR [13, 12]		5			
IR [15, 14]		00	01	11	10
		00	01	11	10
00		PC	-	PC	PC
01	JSR: PC		-	BaseR	BaseR
	JSRR: BaseR				
11		BaseR	-	Vide	PC
10		-	-	PC	PC

## Arg2

IR [13, 12]		9			
IR [15, 14]		00	01	11	10
		00	01	11	10
00		9	-	9	9
01	JSR: 11	-		6	6
	JSRR: 0	-			
11		0	-	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 écrit 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= $\overline{12.13}$  (AND, ADD, NOT, RETB SETB)

**WriteReg:** Indique que l'on ecrit dans le registre  
WriteReg= $\overline{12.13} + \overline{12.13} + \overline{12.14.15} + \overline{13.14.15}$

**Jump:** Indique au RegPC que l'on jump  
Jump= $\overline{12.13} + \overline{12.13.14.15}$

**UseCondition:** Indique au RegPC que l'on utilise les condition nzp  
UseCondition= $\overline{12.13.14.15}$  (juste BR)

**Load:** Indique au RAMCtrl (uniquement) que l'on va lire une valeur  
Load= $\overline{12.13.15} + \overline{12.13.14.15} + \overline{13.14.15}$

**Store:** Indique au RAMCtrl (uniquement) que l'on va stoquer une valeur  
Store= $\overline{12.13.15} + \overline{12.13.14.15}$

**DoubleIndirection:** Indique au RAMCtrl ET au GetAddr que l'on va faire une double indirection  
DoubleIndirection= $\overline{13.14.15}$

## 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 . UseCondition

PC <- Resultat add Unit

**Sinon**

PC <- PC+1

**GetAddr :**

Choisi l'adresse de la RAM à lire ou écrire

Valeur de sorti:

	<b>Si DoubleIndirection</b>	<b>Sinon</b>
<b>Exec</b>	regAddrIndirection	resultat addUnit
<b>Post Exec</b>	PC	PC
<b>Fetch</b>	PC	PC
<b>Post Fetch</b>	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'écriture ou la lecture de la RAM

	<b>Lecture</b>	<b>Ecriture</b>
<b>Exec</b>	Si load	Si Store
<b>Post Exec</b>	0	0
<b>Fetch</b>	1	0
<b>Post Fetch</b>	Si DoubleIndirection	0

**NZP :**

Calcule du NZP:

**N:**  $\text{RES}[15] \cdot \overline{Z}$

**Z:**  $\overline{\text{Or}(\text{RES}[15..0])}$

**P:**  $\text{RES}[15] \cdot \overline{Z}$

Test de saut:

**testN**=  $\text{IR}[11] \cdot N$

**testZ**=  $\text{IR}[10] \cdot Z$

**testP**=  $\text{IR}[9] \cdot P$

**testNZP**= testN + testZ + testP

**Set/Reset bit :**

**Si set:**

Out = DataIn +  $(1 \ll \text{bit } n)$





















**Sinon (si reset)**

Out = DataIn .  $\overline{(1 \ll \text{bit } n)}$



## Fonctionnalités implémentées

---

 <i>BR</i>	 <i>ADD</i>	 <i>LD</i>	 <i>NOP</i>
 <i>JSR</i>	 <i>AND</i>	 <i>LDR</i>	 <i>RTI</i>
 <i>JSRR</i>	 <i>RETB</i>	 <i>LDI</i>	
 <i>JMP</i>	 <i>SETB</i>	 <i>ST</i>	
 <i>RET</i>	 <i>NOT</i>	 <i>STR</i>	
 <i>TRAP</i>	 <i>LEA</i>	 <i>STI</i>	