

---

# Circuits & Architecture

## **Diagrammes temporels d'exécution**

«LC-3»

---

**JACQUETTE Pierrick & STASYSZYN Romain**  
**21305551 & 21305734**

## Table des matières

Introduction.....	3
Instruction Arithmétique.....	4
1.NOT.....	4
2.AND.....	4
3.ADD.....	4
4.SET.....	4
5.RESET.....	4
Instruction de chargement.....	6
1.LEA.....	6
2.LD.....	7
3.LDR.....	8
4.LDI.....	9
Instruction de rangement.....	10
1.ST.....	10
2.STR.....	11
3.STI.....	12
Instruction de branchement.....	13
1.BR.....	13
2.JMP.....	14
3.JSR.....	15
4.JSRR.....	16

# Introduction

Ce projet est réalisé au sein de l'enseignement Circuits et Architecture. Notre projet est de construire avec logisim un processeur qui implante certaines instructions du LC3.

Commencer le projet par les diagrammes nous permettra d'avoir une représentation claire et structurée de notre projet. Une exécution est découpé en quatre temps : Fetch (1 et 2) et Exec (1 et 2), correspond respectivement aux fronts montants et descendants.

Dans une première partie, nous présenterons les instructions arithmétiques.

Ensuite, nous vous montrerons les instructions de chargement.

Puis, nous verrons les instructions de rangement.

Enfin, pour terminer nous parlerons des instructions de branchement.

Sur les diagrammes, des fils n'ont pas été coloriés car ce sont des fils secondaires à Exec2. Certains sous-circuits ont des entrées actives mais pas de sorties actives parce qu'ils n'ont pas d'utilité première dans l'illustration des instructions.

# Instruction Arithmétique

## 1. **NOT**

Cette instruction permet de faire le "non logique" bit à bit d'un registre source (SR) à un registre destination (DR).  $DR \leftarrow \text{not } SR$ .

Sa syntaxe est NOT DR,SR. Son codage est 1001 DR SR 111111, modifie NZP.

## 2. **AND**

Cette instruction permet de faire le "et logique" bit à bit d'un registre source(SR) à un registre destination(DR). Deux formes sont possibles avec des registres sources (SR1 et SR2) ou avec une constante (étendue sur 16 bits).

$DR \leftarrow SR1 \text{ and } SR2$  ou  $DR \leftarrow SR1 \text{ and } \text{SEXT}(\text{Imm5})$ .

Sa syntaxe est AND DR,SR1,SR2/Imm5. Son codage est 0101 DR SR1 0 00 SR2 ou 0101 DR SR1 1 00 Imm5, modifie NZP.

## 3. **ADD**

Cette instruction permet de faire l'addition de deux termes en complément à deux. Deux formes sont possibles avec des registres sources (SR1 et SR2) ou avec une constante (étendue sur 16 bits).  $DR \leftarrow SR1 + SR2$  ou  $DR \leftarrow SR1 + \text{SEXT}(\text{Imm5})$ .

Sa syntaxe est ADD DR,SR1,SR2/Imm5. Son codage est 0001 DR SR1 0 00 SR2 ou 0001 DR SR1 1 Imm5, modifie NZP.

## 4. **SET**

Cette instruction permet de mettre un bit d'un registre à 1. Deux formes sont possibles avec des registre sources (SR1 et SR2) ou avec une constante (étendue sur 16 bits).  $DR \leftarrow \text{swap}(SR1, \text{bit}_i\_SR2)$  ou  $DR \leftarrow \text{swap}(SR1, \text{bit}_i\_SEXT(\text{Imm4}))$  avec swap permettant d'échanger le bit i. Ce dernier est choisi en récupérant les 4 bits de poids faible de SR2.

Sa syntaxe est SETB DR,SR1,SR2/Imm4 . Son codage est 1101 DR SR1 010 SR2 ou 1101 DR SR1 1 1 Imm4, modifie NZP.

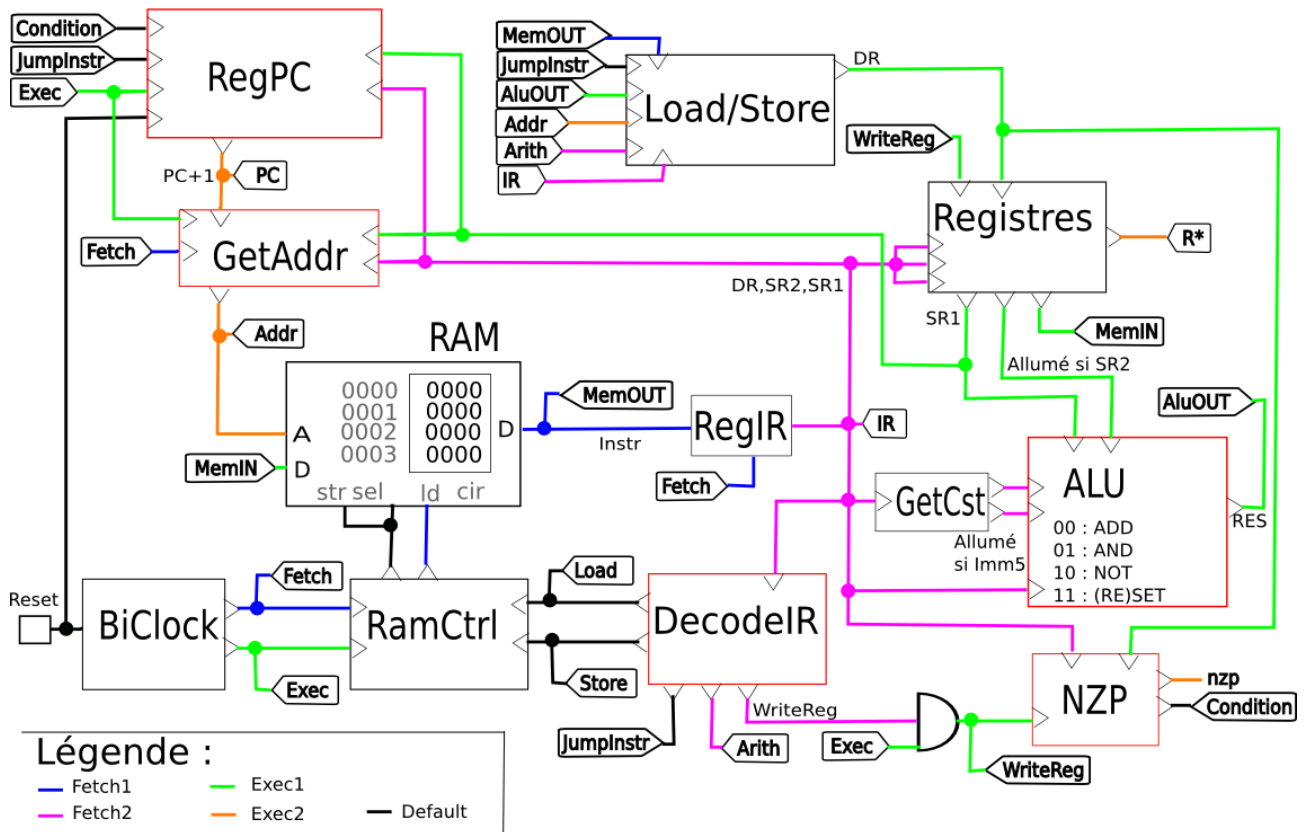
## 5. **RESET**

Cette instruction permet de mettre un bit d'un registre à 0. Deux formes sont possibles

avec des registre sources (SR1 et SR2) ou avec une constante (étendue sur 16 bits).  
 $DR \leftarrow \text{swap}(\text{SR1}, \text{bit}_i \text{ SR2})$  ou  $DR \leftarrow \text{swap}(\text{SR1}, \text{bit}_i \text{ SEXT}(\text{Imm4}))$  avec swap permettant d'échanger le bit i. Ce dernier est choisi en récupérant les 4 bits de poids faible de SR2.

Sa syntaxe est RSTB DR,SR1,SR2/Imm4. Son codage est 1101 DR SR1 000 SR2 ou 1101 DR SR1 1 0 Imm4 ,modifie NZP.

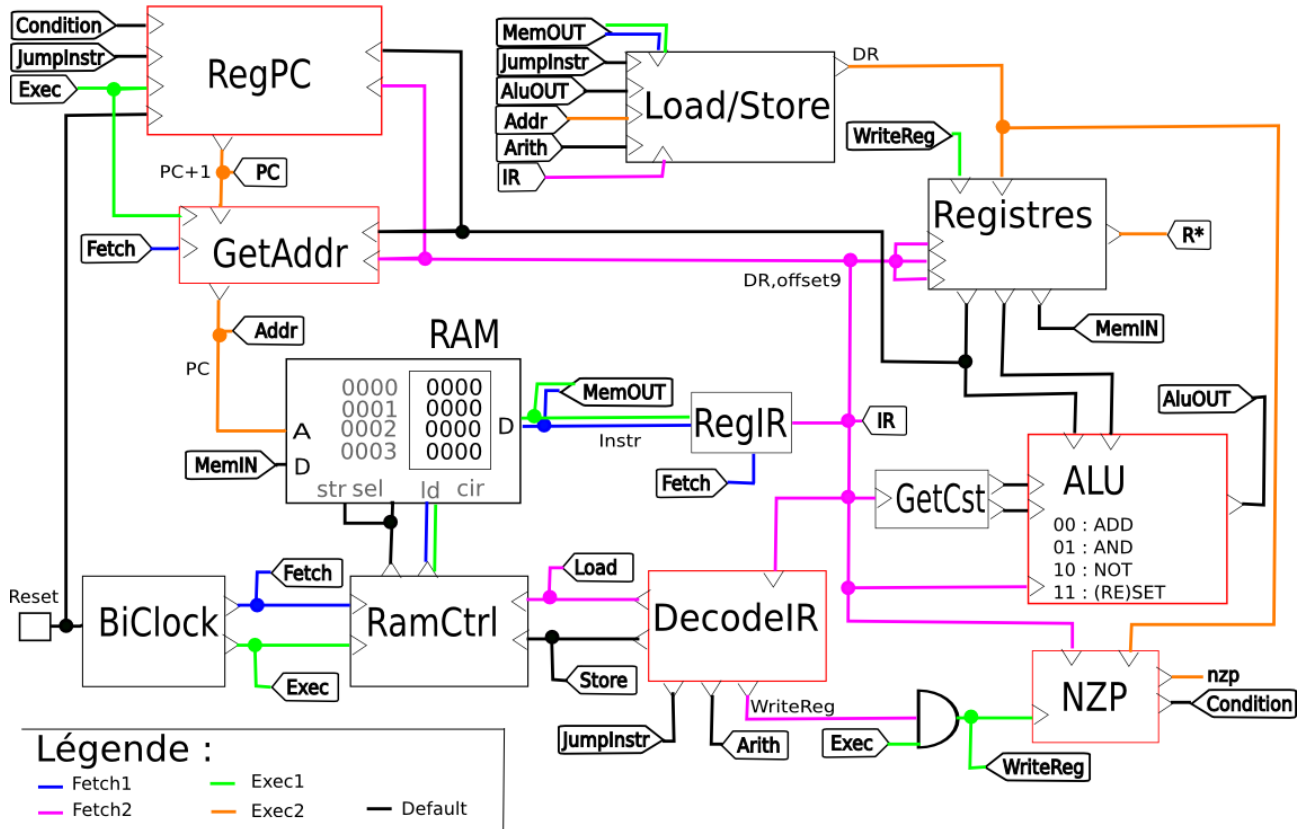
Diagramme temporel d'exécution des instructions arithmétiques



# Instruction de chargement

## 1. LEA

Diagramme temporel d'exécution de LEA

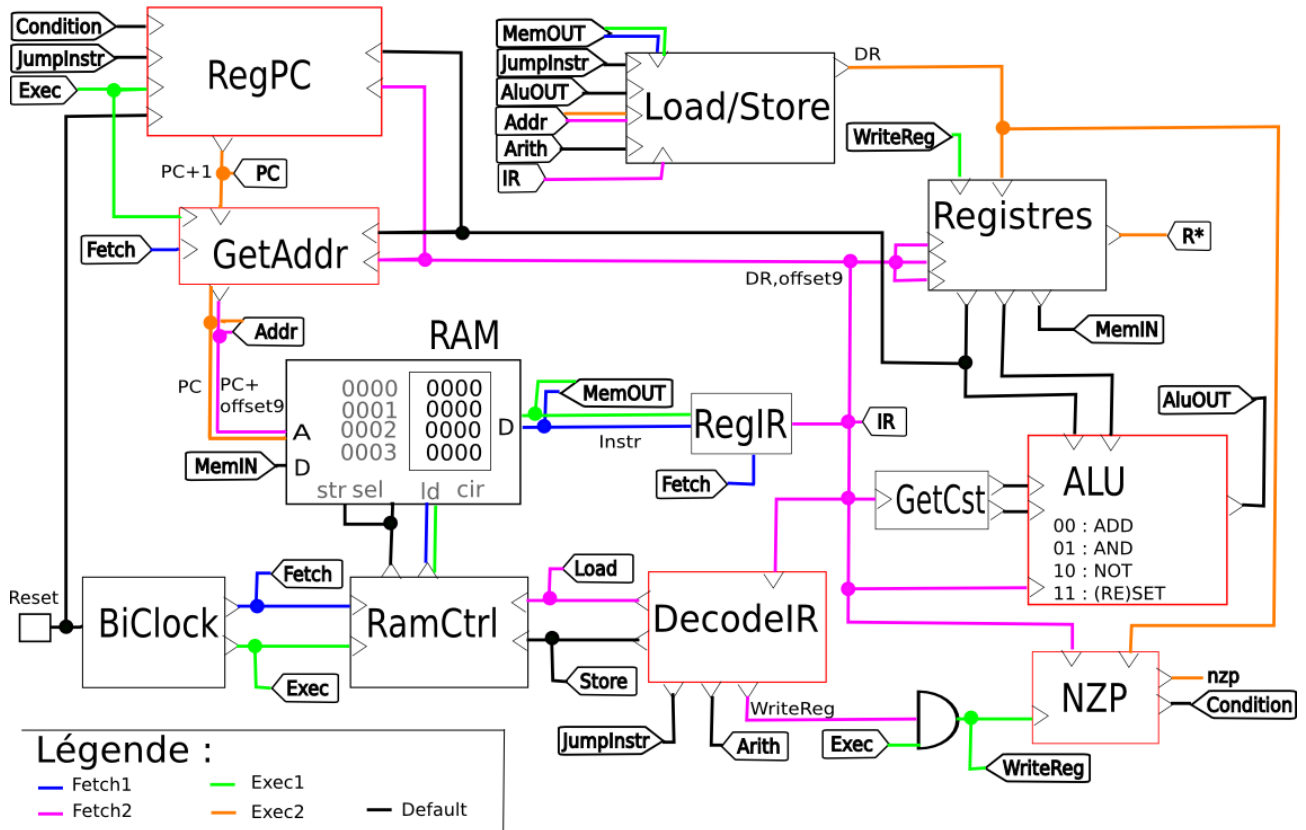


Cette instruction permet de charger une adresse dans un registre. Ceci est un adressage absolu (=immédiat), aucun accès à la mémoire n'est nécessaire.  
 $DR \leftarrow PC + \text{SEXT}(\text{PCoffset9})$ .

Sa syntaxe est `LEA DR,label`. Son codage est `1110 DR PCoffset9`, modifie NZZ. Les fils secondaires sont les sorties des REGISTRES.

## 2. LD

Diagramme temporel d'exécution de LD



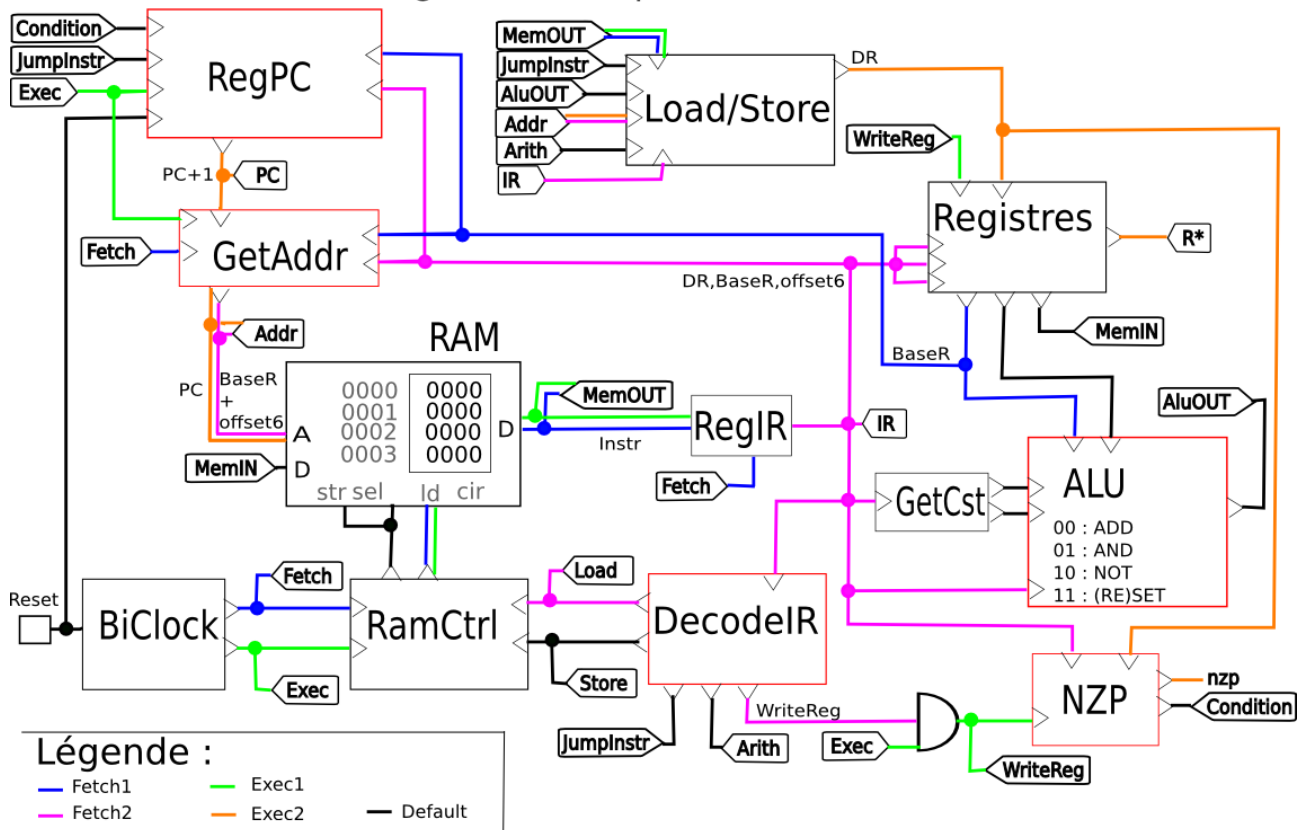
Cette instruction permet de charger un mot mémoire à l'adresse de PC+offset de 9 bits. Ceci est un adressage direct, un accès à la mémoire.

$DR \leftarrow \text{mem}[\text{PC} + \text{SEXT}(\text{PCoffset9})]$ .

Sa syntaxe est LD DR,label. Son codage est 0010 DR PCoffset9, modifie N/ZP. Les fils secondaires sont les sorties des REGISTRES.

### 3. LDR

Diagramme temporel d'exécution de LDR



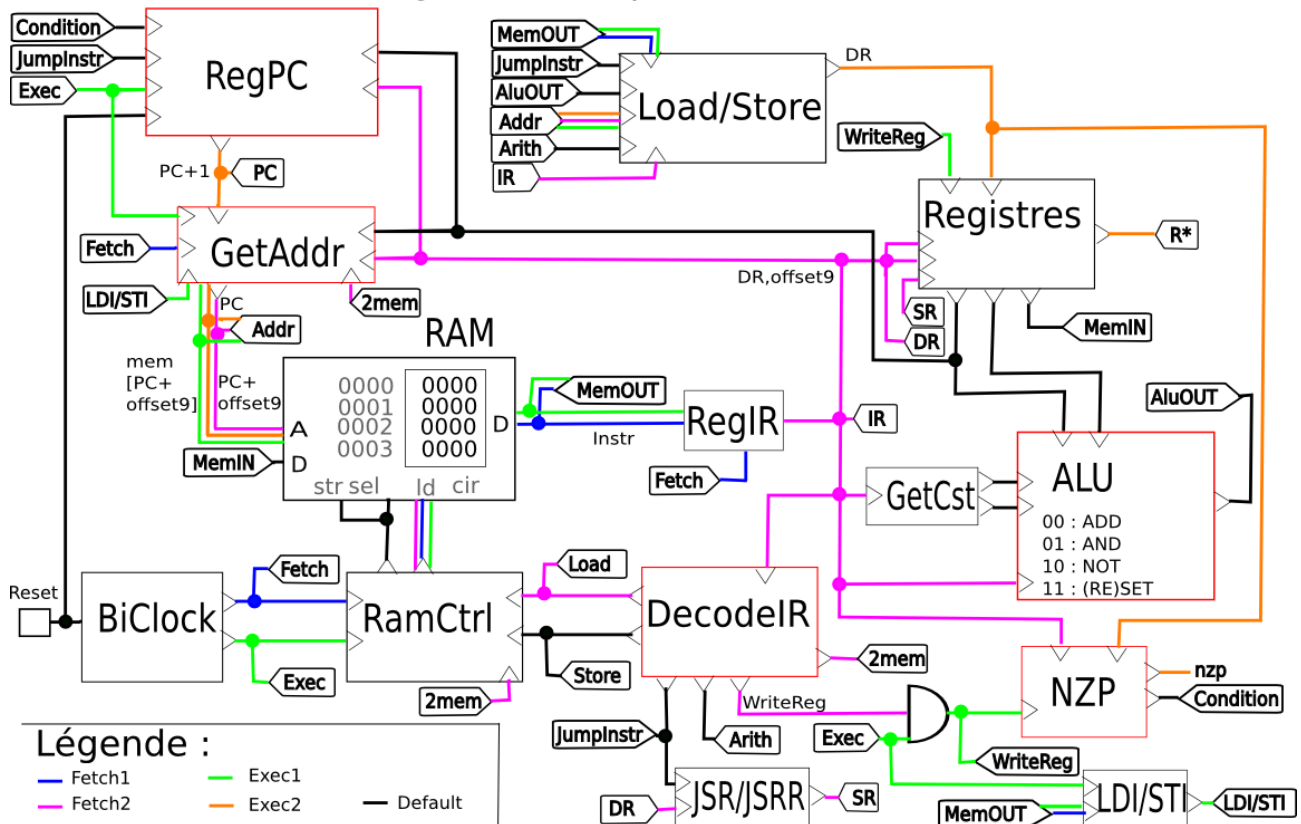
Cette instruction permet de charger un mot mémoire à l'adresse d'un registre de base + offset de 6 bits. Ceci est un adressage relatif (=basé), un seul accès mémoire.  
 $DR \leftarrow \text{mem}[\text{BaseR} + \text{SEXT}(\text{Offset6})]$ .

Sa syntaxe est LDR DR,BaseR,Offset6. Son codage est 0110 DR BaseR Offset6, modifie N/ZP. Les fils secondaires sont les sorties des REGISTRES MemIn et SR2 à Fetch1 et toutes les sorties à Exec2.



#### 4. LDI

Diagramme temporel d'exécution de LDI



Cette instruction permet de charger un mot mémoire à l'adresse de PC+offset de 9 bits puis utilise cette adresse pour charger le registre. Ceci est un adressage indirect, deux accès à la mémoire sont nécessaires.

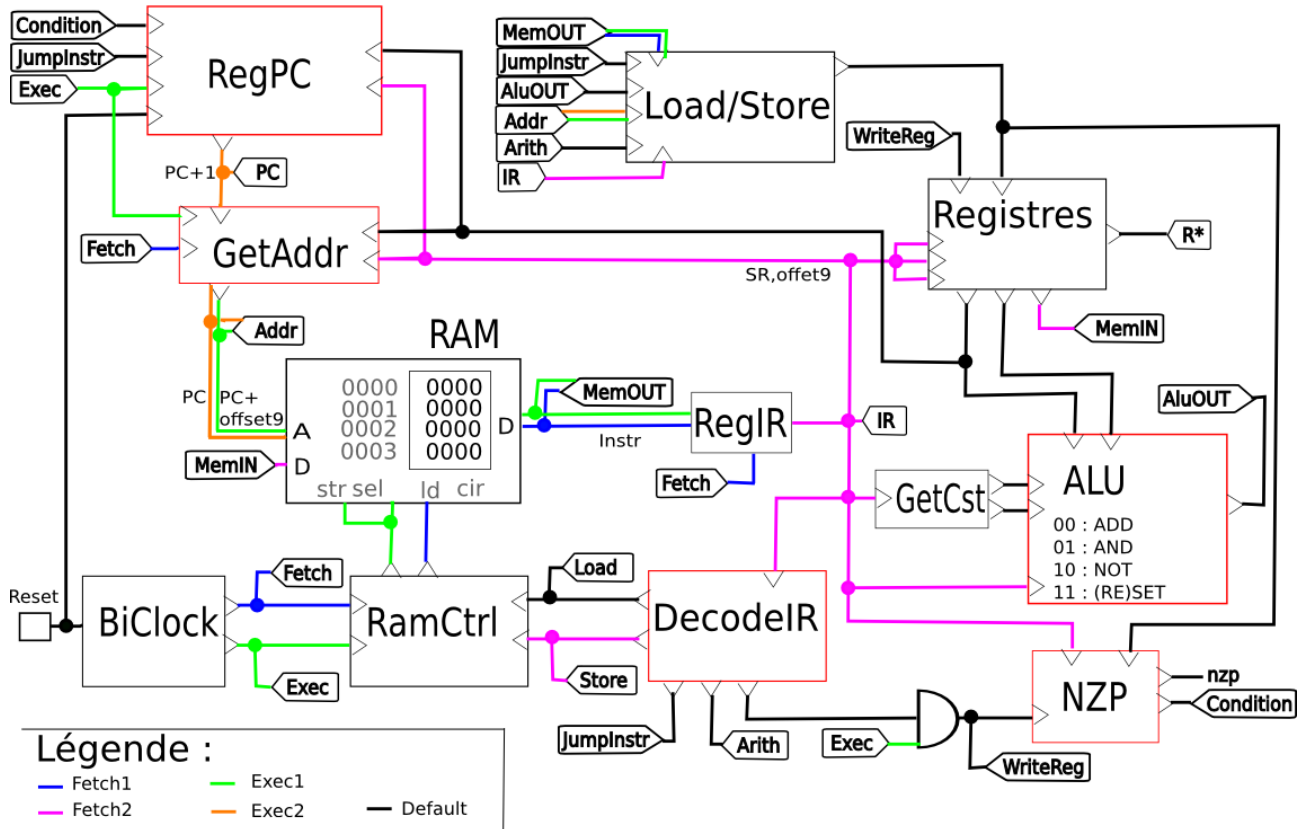
$DR \leftarrow \text{mem}[\text{mem}[\text{PC} + \text{SEXT}(\text{PCoffset9})]]$ .

Sa syntaxe est LDI DR,label. Son codage est 1010 DR PCoffset9, modifie NZP. Les fils secondaires sont les sorties des REGISTRES.

# Instruction de rangement

## 1. ST

Diagramme temporel d'exécution de ST

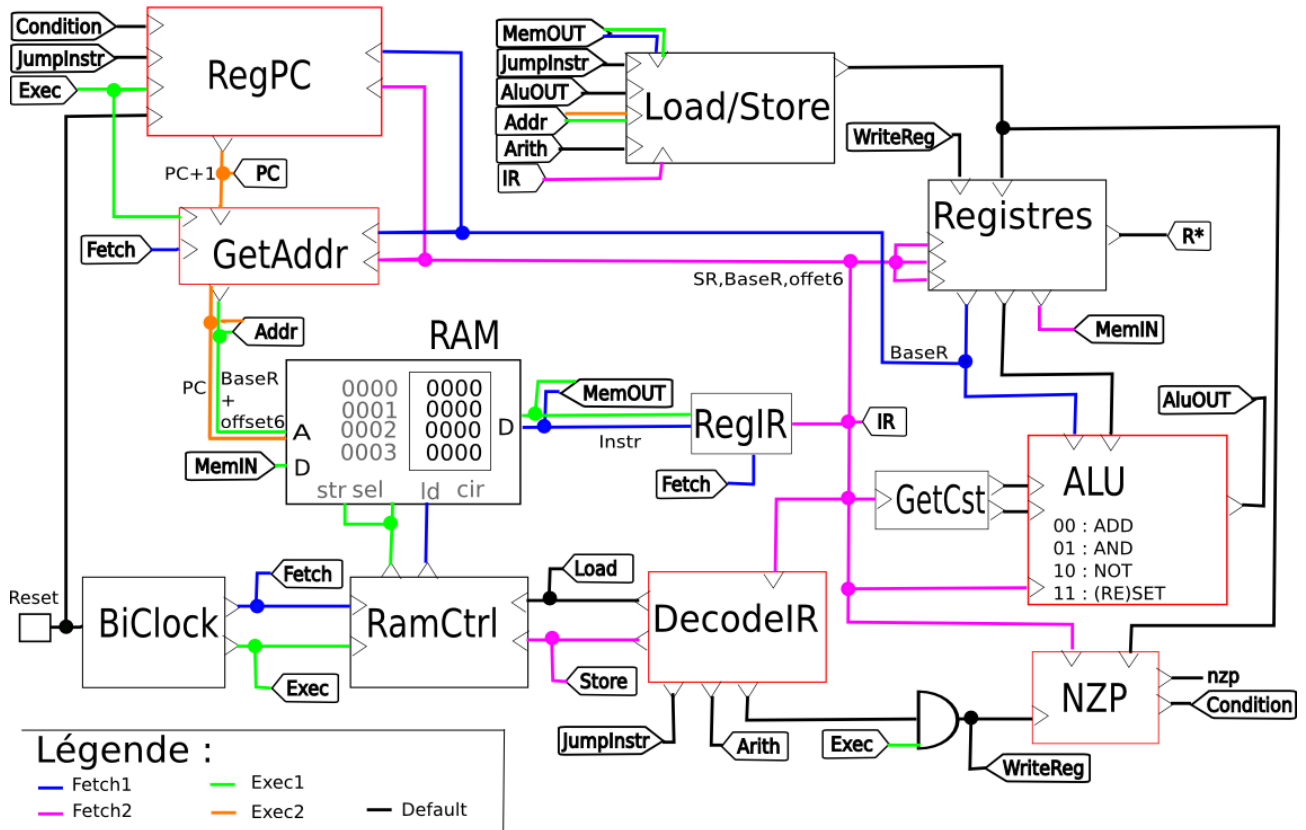


Cette instruction permet de ranger un mot mémoire depuis l'adresse de PC+offset de 9 bits. Ceci est un adressage direct, un accès à la mémoire.  
 $\text{mem}[\text{PC} + \text{SEXT}(\text{PCoffset9})] \leftarrow \text{SR}$ .

Sa syntaxe est ST SR,label. Son codage est 0011 SR PCoffset9. Les fils secondaires sont les sorties SR1 et SR2 des REGISTRES à Fetch2.

## 2. STR

Diagramme temporel d'exécution de STR

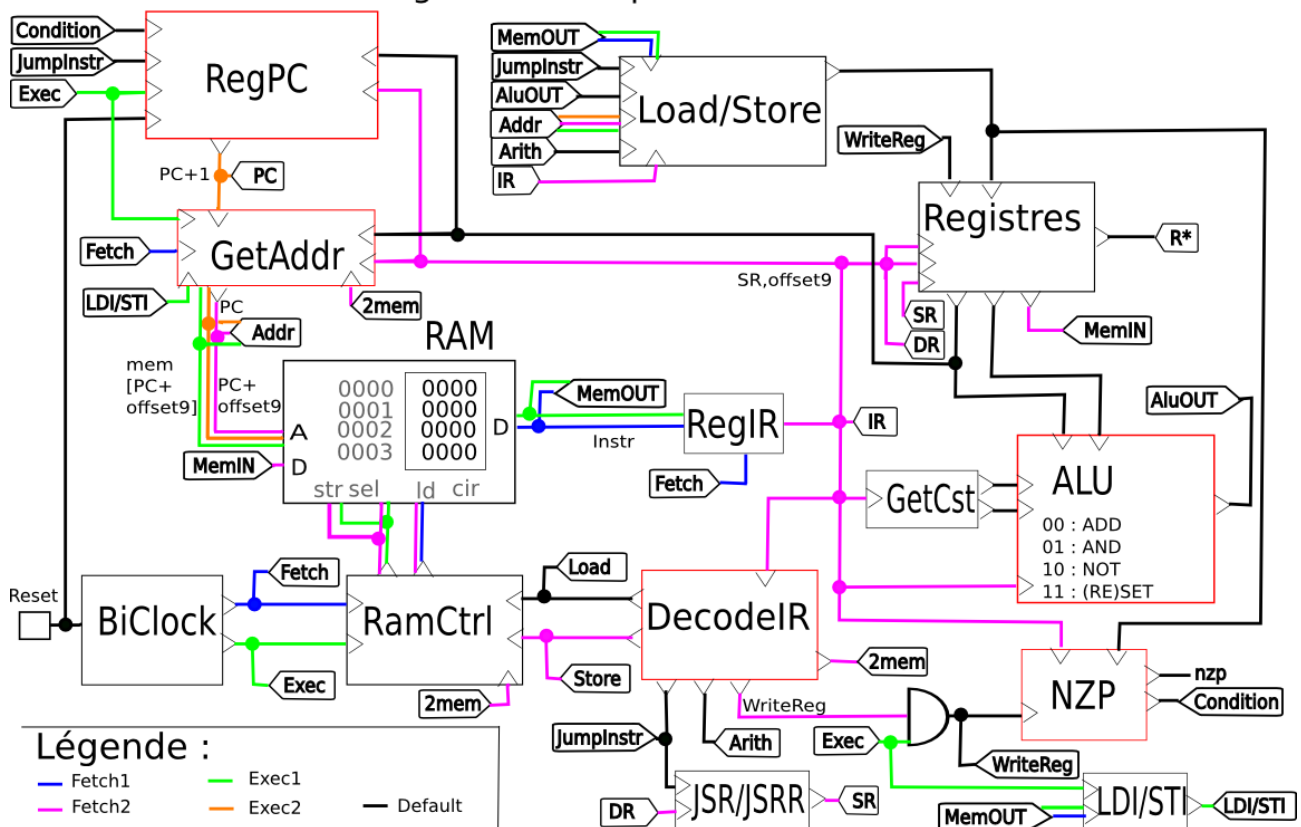


Cette instruction permet de ranger un mot mémoire depuis l'adresse d'un registre de base + offset de 6 bits. Ceci est un adressage relatif (=basé), un seul accès mémoire.  $\text{mem}[\text{BaseR} + \text{SEXT}(\text{Offset6})] \leftarrow \text{SR}$ .

Sa syntaxe est STR SR,BaseR,Offset6. Son codage est 0111 SR BaseR Offset6. Les fils secondaires sont les sorties SR1 et SR2 des REGISTRES à Fetch2.

### 3. STI

Diagramme temporel d'exécution de STI



Cette instruction permet de ranger un mot mémoire depuis l'adresse de PC+offset de 9 bits puis utilise cette adresse pour ranger le mot mémoire. Ceci est un adressage indirect, deux accès à la mémoire sont nécessaires.

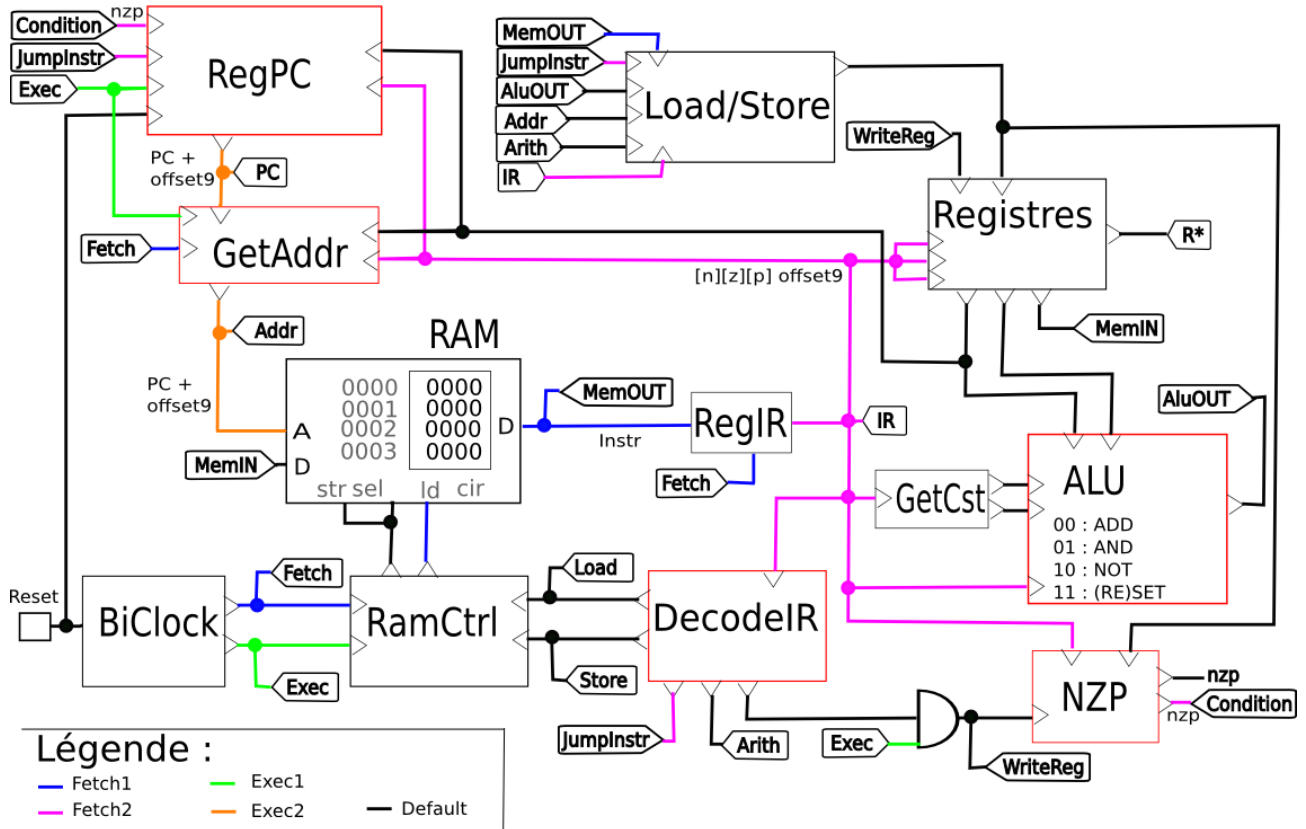
$\text{mem}[\text{mem}[\text{PC} + \text{SEXT}(\text{PCoffset9})]] \leftarrow \text{SR}$ .

Sa syntaxe est STI SR,label. Son codage est 1011 SR PCoffset9. Les fils secondaires sont les sorties SR1 et SR2 des REGISTRES à Fetch2.

# Instruction de branchement

## 1. BR

Diagramme temporel d'exécution de BR

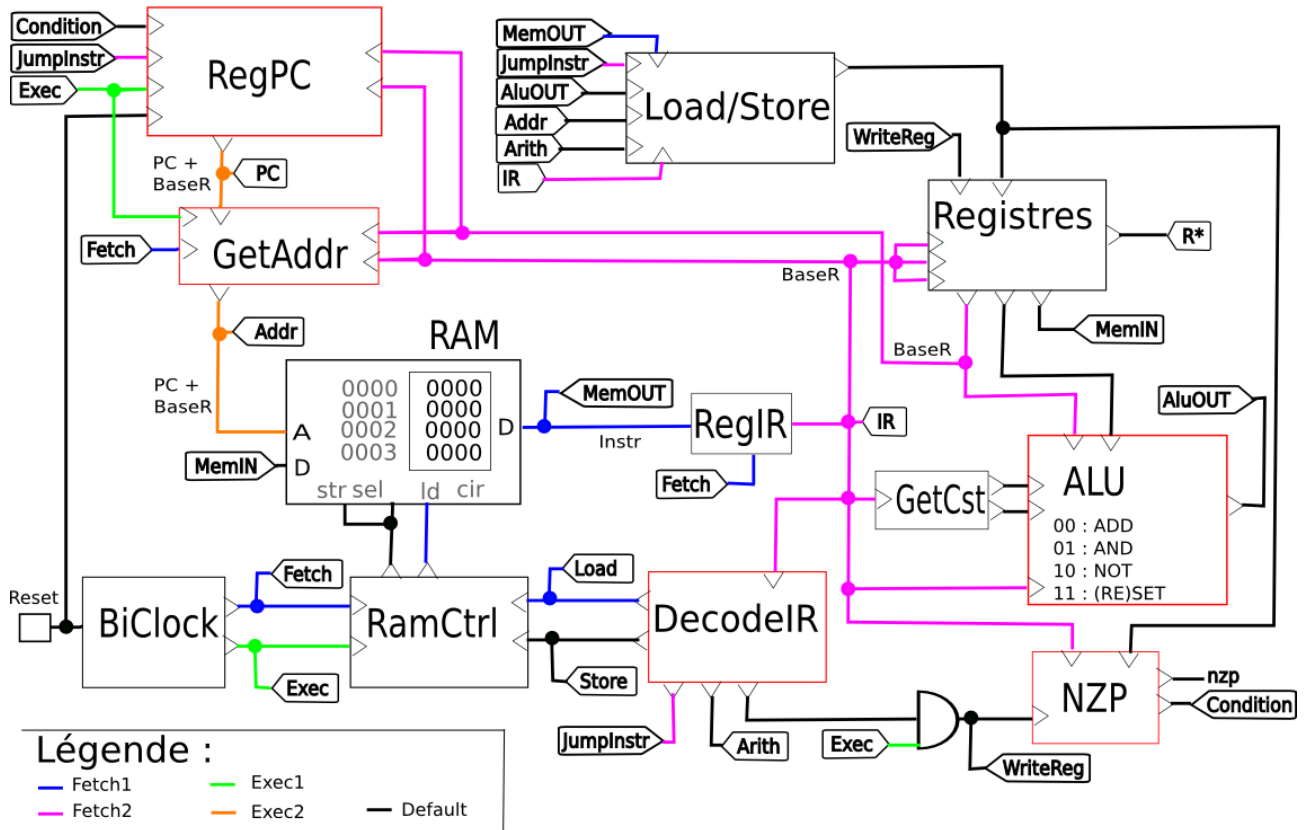


Cette instruction permet de modifier le déroulement normal des instructions en modifiant la valeur de PC. PC est calculé en additionnant PC avec un offset de 9 bits. Si la condition n'est pas respectée, on passe à l'instruction suivante. Si (cond)  $PC \leftarrow PC + \text{SEXT}(\text{PCoffset9})$ .

Sa syntaxe est `BR[n][z][p] label`. Son codage est `0000 nzp PCoffset9`.

## 2. JMP

Diagramme temporel d'exécution de JMP

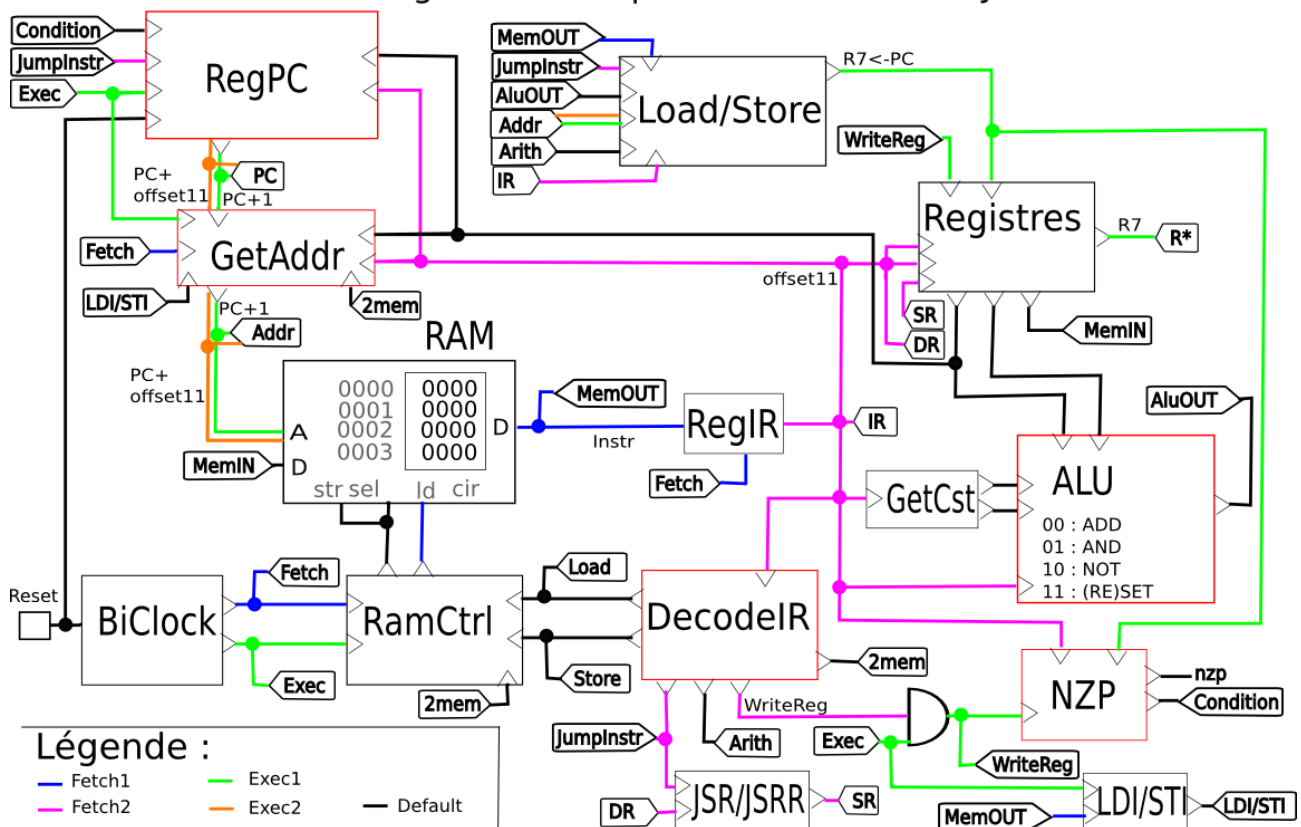


Cette instruction permet de charger PC avec le contenu d'un registre. Elle permet de compléter BR (saut à  $\pm 256$  mots mémoire), pour les pointeurs, les sous-routines.  
 $PC \leftarrow BaseR$ .

Sa syntaxe est `JMP BaseR`. Son codage est `1100 000 BaseR 000000`. Les fils secondaires sont les sorties SR2 et MemIn des REGISTRES à Fetch2.

### 3. JSR

Diagramme temporel d'exécution de JSR

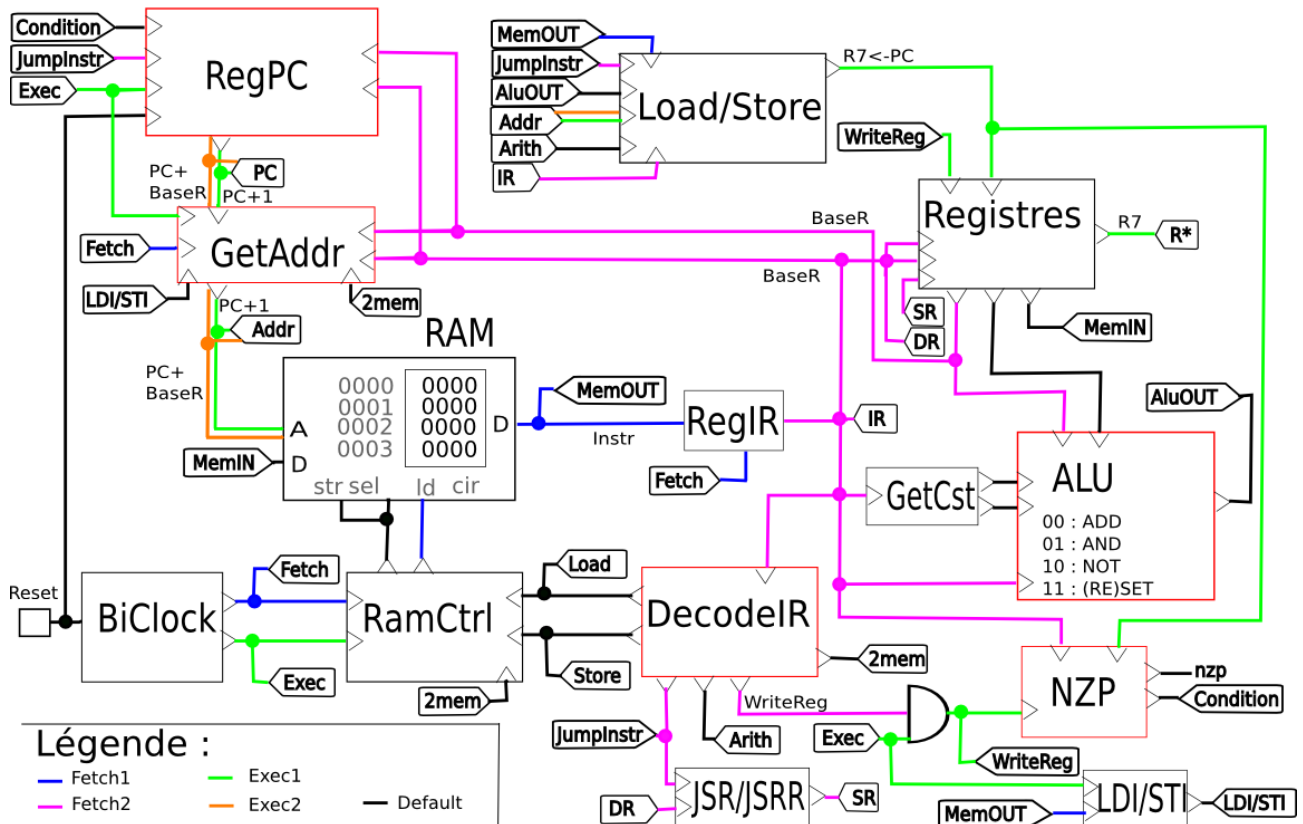


Cette instruction permet d'appeler une sous-routine. Il faut mémoriser l'adresse à laquelle on doit revenir, elle est donc stockée dans R7. Puis on va à l'instruction donnée par l'offset.  $R7 \leftarrow PC$  ;  $PC \leftarrow PC + \text{SEXT}(\text{PCoffset11})$ .

Sa syntaxe est `JSR label`. Son codage est `0100 1 PCoffset11`. Les fils secondaires sont les sorties des REGISTRES.

#### 4. JSRR

Diagramme temporel d'exécution de JSRR



Cette instruction permet d'appeler une sous-routine. Il faut mémoriser l'adresse à laquelle on doit revenir, elle est donc stocker dans R7. Puis on va à l'instruction donnée par BaseR.  $R7 \leftarrow PC$  ;  $PC \leftarrow PC + BaseR$ .

Sa syntaxe est JSR label. Son codage est 0100 0 00 BaseR 000000. Les fils secondaires sont les sorties SR2 et MemIn des REGISTRES à Fetch2 et Exec1 et le fils SR1 à Exec1.