

# Circuits & Architecture Diagrammes temporels d'exécution

«LC-3»

JACQUETTE Pierrick & STASYSZYN Romain 21305551 & 21305734

# **Table des matières**

Introduction	3
Instruction Arithmétique	
1.NOT	4
2.AND	4
3.ADD	4
4.SET	
5.RESET	4
Instruction de chargement	6
1.LEA	
2.LD	
3.LDR	
4.LDI	
Instruction de rangement.	
1.ST	
2.STR	11
3.STI	
Instruction de branchement	
1.BR	
2.JMP	
3.JSR	
4.ISRR	

# 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.

Nous avons rajoutés des sous-circuits, des fils. Dans le circuit main, le fil allant du tunnel Exec vers le circuit GetAddr: il nous permet de calculer les adresses de lancer cette étape. Ajout du circuit Load/Store pour gérér les instructions de rangement et chargement. Modification de fil allant de la sortie de l'ALU vers les registres par conséquent. Ajout d'un fil entre RegIr et ALU permettant de récupérer le 4ième bit pour faire la distinction entre l'instruction SET et RESET. Pour traiter ultérieurement les instructions JSR, JSRR, LDI et STI, ajout respectivement des circuits JSR/JSRR et LDI/STI. Nous avons également ajoutés les fils nécessaires pour leurs intégrations dans le circuit actuel.

# **Instruction Arithmétique**

### NOT

Cette instruction permet de faire le "non logique" bit à bit d'un registre source (SR) à un registre destination (DR). DR  $\leftarrow$  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 ← SR1 and SR2 ou DR ← SR1 and SEXT(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 + SEXT(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 ← swap(SR1,bit\_i\_SR2) ou DR ← swap(SR1,bit\_i\_SEXT(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).  $DR \leftarrow swap(SR1,bit i SR2)$  ou  $DR \leftarrow swap(SR1,bit i SEXT(Imm4))$  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.

#### Condition MemOUT DR Jumpinstr Jumplnstr RegPC Exec AluOUT Load/Store Addr WriteRea Arith PC IR Registres R\* Fetch GetAddr DR,SR2,SR1 SR1 Addr **RAM** MemIN Allumé si SR2 0000 0000 MemOUT AluOUT 0001 0000 **IR** RegIR 0000 D Instr 0003 0000 MemIN ALU Fetch cir GetCst RES 00 : ADD Allumé si Imm5 01: AND 4ième bit Load 10: NOT Fetch 11 : (RE)SET 14-15ième bits DecodeIR BiClock RamCtrl Store nzp Exec NZP WriteReg Condition Jumpinstr Exec Légende : Arith WriteReg Fetch1 Exec1 Fetch2 Default Exec2

Diagramme temporel d'exécution des instructions arithmétiques

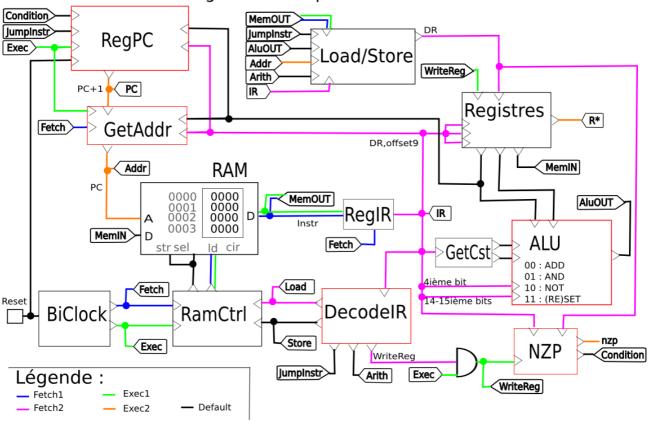
Les fils secondaires sont les sorties des REGISTRES à exec2. Beaucoup de choses se déroulent pendant fetch2. Voici l'enchaînement, au début la commande est distribuée à tout les circuits (ALU, Registres, Load/Store, DecodeIR). Les registres vont être consultés pour connaître SR1 (optionnellement SR2 ou une constante est récupéré). Ces données vont être transmis à l'ALU qui produit la valeur à mettre dans le registre destination.

Pendant ce temps, DecodeIR va autorisé l'écriture dans un fichier pour plus tard, signalé au sous-circuit Load/Store que l'on est en présence d'une instruction arithmétique et écrit sur sa sortie la valeur de l'entrée AluOUT (donc le fils vers les Registres).

# 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 ← PC + SEXT(PCoffset9).

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

# 2. LD

#### Diagramme temporel d'exécution de LD Condition MemOUT Jumpinstr Jumpinstr RegPC Exec AluOUT Load/Store Addr WriteReg Arith **PC** IR Registres R\* GetAddr Fetch DR,offset9 MemIN Addr **RAM** 0000 0000 0000 D PC PC+ 0000 MemOUT AluOUT offset9 0001 RegIR (IR) 0003 0000 MemIN D ALU Fetch str sel cir GetCst 00 : ADD 01: AND 4ième bit Load Fetch 10: NOT 11: (RE)SET 14-15ième bits Reset DecodelR RamCtrl **BiClock** nzp Store Exec NZP Condition WriteReg Jumplnstr Exec Arith Légende : WriteReg Fetch1 Exec1 Fetch2 Exec2 Default

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.

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

Sa syntaxe est LD DR,label. Son codage est 0010 DR PCoffset9, modifie NZP. Les fils secondaires sont les sorties des REGISTRES à fetch2 puis à exec2.

# 3. LDR

#### Diagramme temporel d'exécution de LDR Condition MemOUT Jumpinstr Jumpinstr RegPC Exec AluOUT Load/Store Addr WriteReg Arith < PC Registres R\* Fetch GetAddr DR,BaseR,offset6 MemIN Addr RAM BaseR BaseR 0000 0000 0000 D PC 0000 MemOUT AluOUT 0001 RegIR **IR** Instr 0003 0000 MemIN D **ALU** Fetch str sel cir GetCst 00 : ADD 01: AND 4ième bit Load Fetch 10: NOT 11: (RE)SET 14-15ième bits Reset DecodelR RamCtrl **BiClock** nzp Store Exec NZP Condition WriteReg Jumplnstr Exec Arith Légende : WriteReg Fetch1 Exec1 Fetch2 Exec2 Default

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 ← mem[BaseR + SEXT(Offset6)].

Sa syntaxe est LDR DR,BaseR,Offset6. Son codage est 0110 DR BaseR Offset6, modifie NZP. 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 Condition DR Jumpinstr Jumpinstr RegPC Exec AluOUT Load/Store Addr WriteReg Arith PC IR Registres R\* Fetch GetAddr DR,offset9 LDI/STI 2mem SR MemIN Addr **RAM** DR mem [PC+ PC+ 0000 0000 MemOUT AluOUT offset9 0001 0002 0003 0000 0000 D offset91 RegIR IR Instr 0000 MemIN **ALU** Fetch cir str sel GetCst 00 : ADD 01 : AND 4ième bit Load Fetch 10: NOT 14-15ième bits 11: (RE)SET Reset DecodeIR **BiClock** RamCtrl 2mem nzp Store Exec NZP 2mem WriteReg Condition Jumpinstr Arith Exec Légende : WriteReg Fetch1 Exec1 }JSR/JSRR 1,DI/STI⊳ (SR (LDI/STI Fetch2 Exec2 Default DR MemOUT

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.

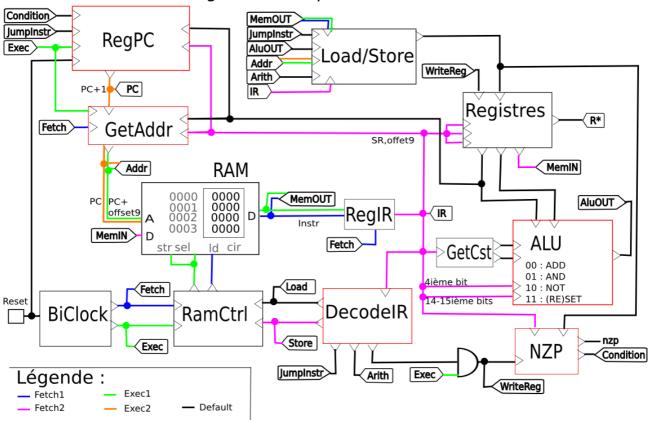
 $DR \leftarrow mem[mem[PC + SEXT(PCoffset9)]].$ 

Sa syntaxe est LDI DR,label. Son codage est 1010 DR PCoffset9, modifie NZP. Les fils secondaires sont les sorties des REGISTRES à fetch2 et à exec2.

# 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.  $mem[PC + SEXT(PCoffset9)] \leftarrow SR$ .

Sa syntaxe est ST SR,label. Son codage est 0011 SR PCoffset9. Les fils secondaires sont les sorties SR2 et MemIn des REGISTRES à fetch2 puis les trois sorties à exec2.

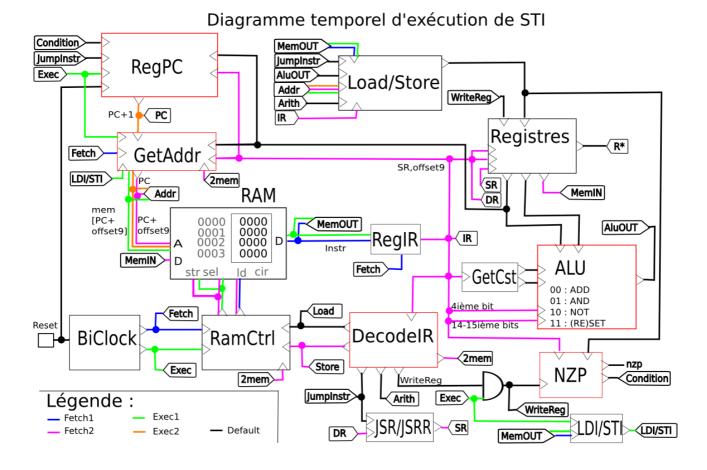
# 2. STR

#### Diagramme temporel d'exécution de STR Condition MemOUT Jumpinstr Jumpinstr RegPC Exec AluOUT Load/Store Addr WriteReg Arith) < PC IR Registres R\* Fetch GetAddr SR,BaseR,offet6 MemIN Addr RAM BaseR BaseR 0000 0000 0000 D PC 0000 MemOUT AluOUT 0001 RegIR **IR** Instr 0003 0000 MemIN D **ALU** Fetch str sel ld cir GetCst 00 : ADD 01: AND 4ième bit Load Fetch 10: NOT 11: (RE)SET 14-15ième bits Reset DecodelR RamCtrl **BiClock** nzp Store Exec NZP Condition Jumpinstr Arith Exec Légende : WriteReg Fetch1 Exec1 Fetch2 Exec2 Default

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. mem[BaseR + SEXT(Offset6)]  $\leftarrow$  SR.

Sa syntaxe est STR SR,BaseR,Offset6. Son codage est 0111 SR BaseR Offset6. Le fil secondaire est la sortie SR2 des REGISTRES à fetch2.

# 3. 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.

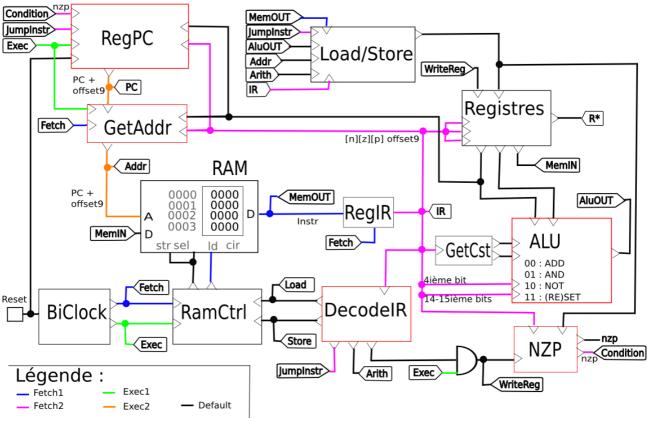
 $mem[mem[PC + SEXT(PCoffset9)]] \leftarrow SR.$ 

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 + SEXT(PCoffset9).

Sa syntaxe est BR[n][z][p] label. Son codage est 0000 nzp PCoffset9. Les fils secondaires sont les sorties des REGISTRES à fetch2.

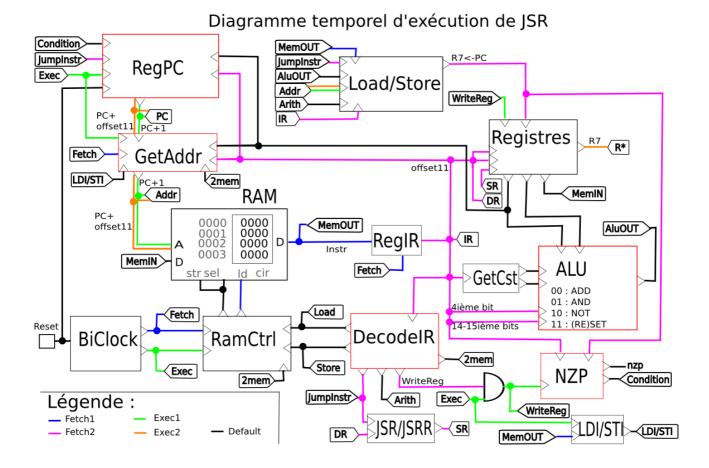
# 2. JMP

#### Diagramme temporel d'exécution de JMP Condition MemOUT Jumpinstr Jumpinstr RegPC Exec AluOUT Load/Store Addr WriteReg Arith) PC + BaseR PC IR Registres R\* GetAddr Fetch BaseR MemIN Addr **RAM** BaseR PC + 0000 0000 0000 D 0000 MemOUT AluOUT BaseR 0001 RegIR (IR 0003 0000 MemIN D **ALU** Fetch str sel ld cir GetCst 00 : ADD 01: AND 4ième bit Load Fetch 10: NOT 11: (RE)SET 14-15ième bits Reset DecodelR RamCtrl **BiClock** nzp Store Exec NZP Condition Jumpinstr Arith Exec Légende : WriteReg Fetch1 Exec1 Fetch2 Exec2 Default

Cette instruction permet de charger PC avec le contenu d'un registre. Elle permet de compléter BR (saut à -/+256 mots mémoire), pour les pointeurs, les sous-routines. PC ← 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



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 l'offset. R7  $\leftarrow$  PC ; PC  $\leftarrow$  PC + SEXT(PCoffset11).

Sa syntaxe est JSR label. Son codage est 0100 1 PCoffset11. Les fils secondaires sont les sorties des REGISTRES à fetch2 et à exec2.

# 4. JSRR

#### Diagramme temporel d'exécution de JSRR Condition MemOUT R7<-PC Jumpinstr Jumpinstr RegPC Exec AluOUT Load/Store Addr WriteReg Arith PC+ PC BaseR PC+1 IR Registres R7 **R\*** BaseR Fetch GetAddr BaseR 2mem LDI/STI PC+1 SR MemIN Addr **RAM** DR PC+ 0000 0000 MemOUT BaseR AluOUT 0000 D 0001 0002 0003 RegIR IR Instr 0000 MemIN D **ALU** Fetch str sel ld cir GetCst 00 : ADD 01 : AND 4ième bit Load 10: NOT Fetch 14-15ième bits 11: (RE)SET Reset DecodelR **BiClock** RamCtrl 2mem nzp Store Exec NZP 2mem WriteReg Condition Jumpinstr Arith Exec Légende : WriteReg Fetch1 Exec1 JSR/JSRR TDI/STI TDI/STI (SR Fetch2 Exec2 Default DR MemOUT

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 exec2 et le fils SR1 à exec2.