Circuits et Architecture – 2015/2016

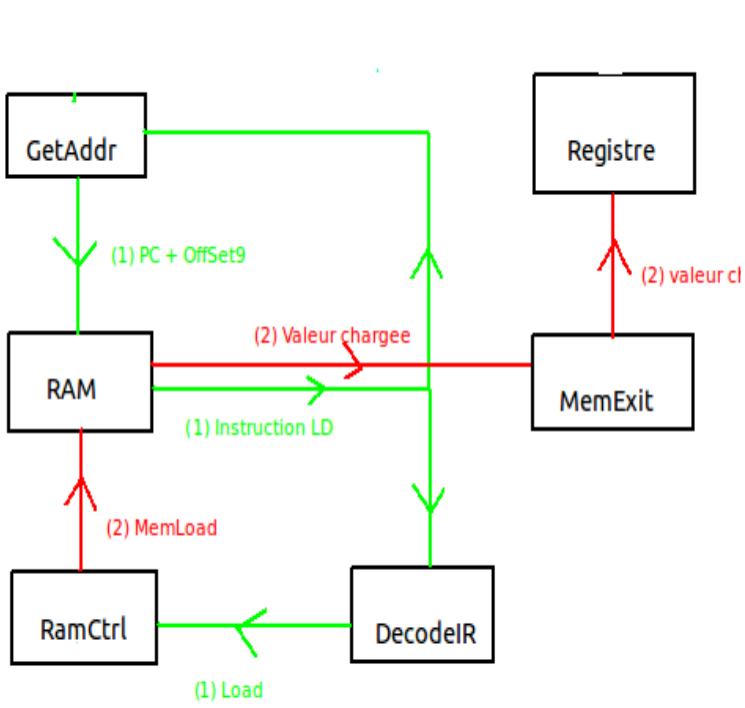
Projet : LC3

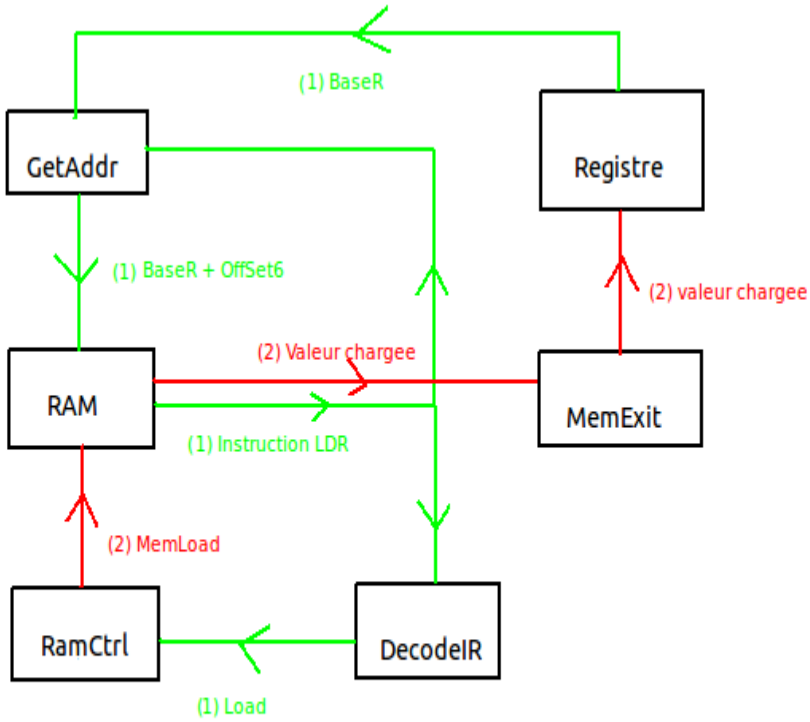
Instruction Arithmétique :

-XOR
Pour l'instruction XOR, nous avons remplacer la porte NOT du module ALU par une porte XOR.
Comme pour le ADD et le AND, le bit 5 indique si il faut faire l'opération avec un constante ou un registre.
Les programmes qui utilise l'instruction NOT continue de fonctionner, car le codage de l'instruction correspond a celle d'un XOR avec la constante (-1) ce qui donne le même résultat.

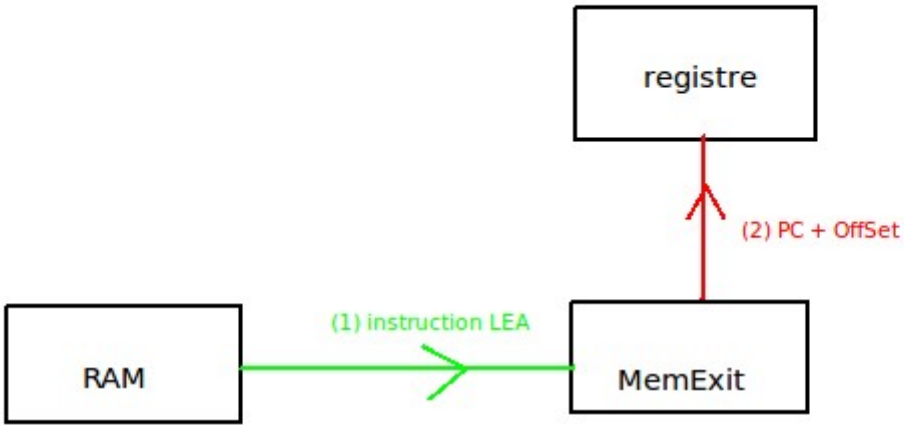
Instructions de chargement :
Les instructions de chargement effectuent des actions dans plusieurs modules :
Dans GetAddr, on va calculer la valeur de l'emplacement de la mémoire à charger.
Dans Memory Exit, on donne le mot mémoire à écrire.

-LD
GetAddr va calculer la somme du PC + l'offset, envoyer cette adresse pendant l'exécution du Load, Memory Exit va récupérer le mot mémoire correspondant et l'écrire dans DR.
-LDR
GetAddr va calculer la somme de la valeur du registre BaseR et de l'offset, envoyer cette adresse pendant l'exécution du Load et Memroy Exit enverra la valeur mot mémoire correspondant.

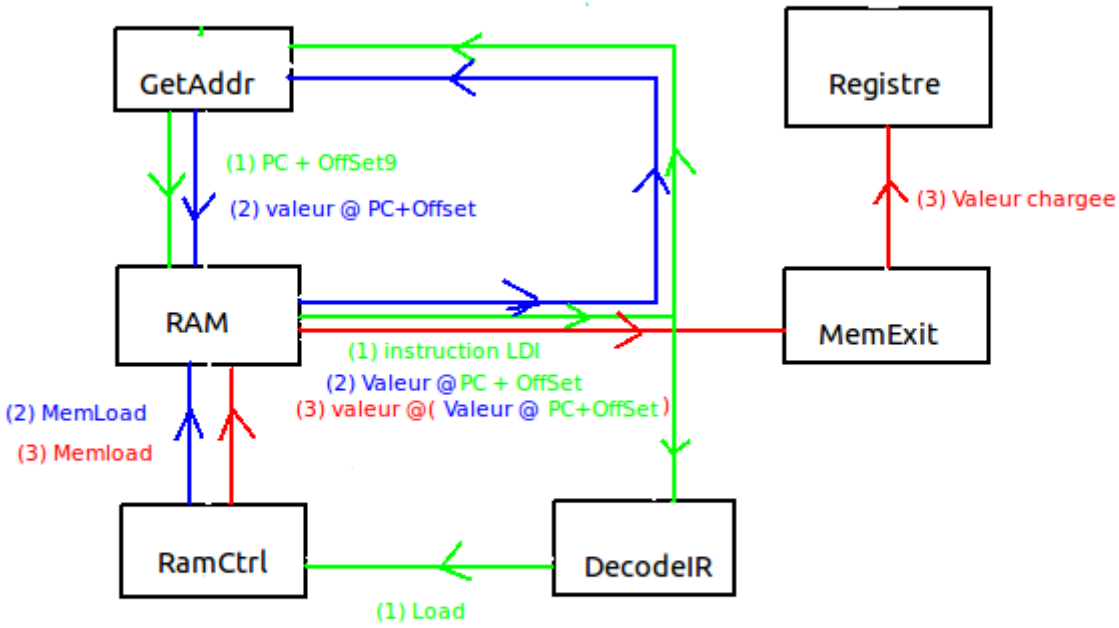




-LEA
Pour LEA Memory Exit va calculer la somme du PC plus de l'offset et cette donner cette valeur a écrire en registre.

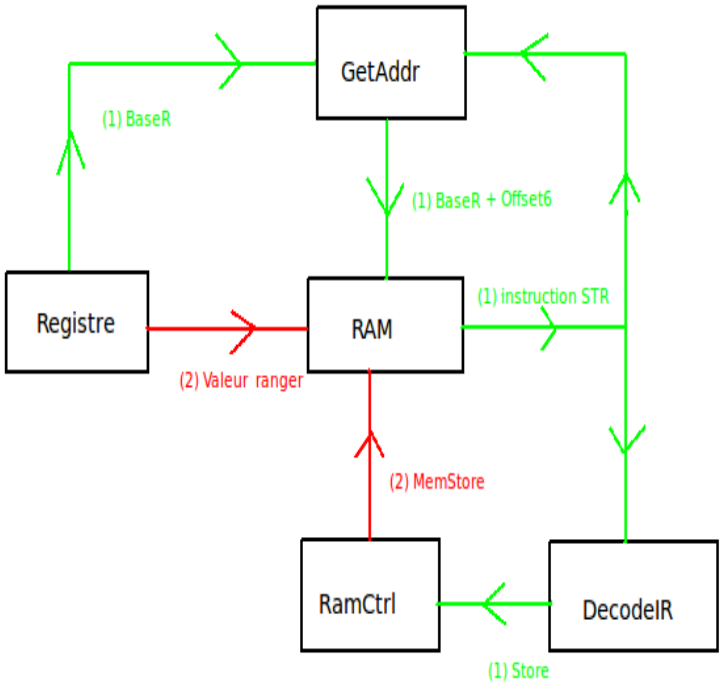


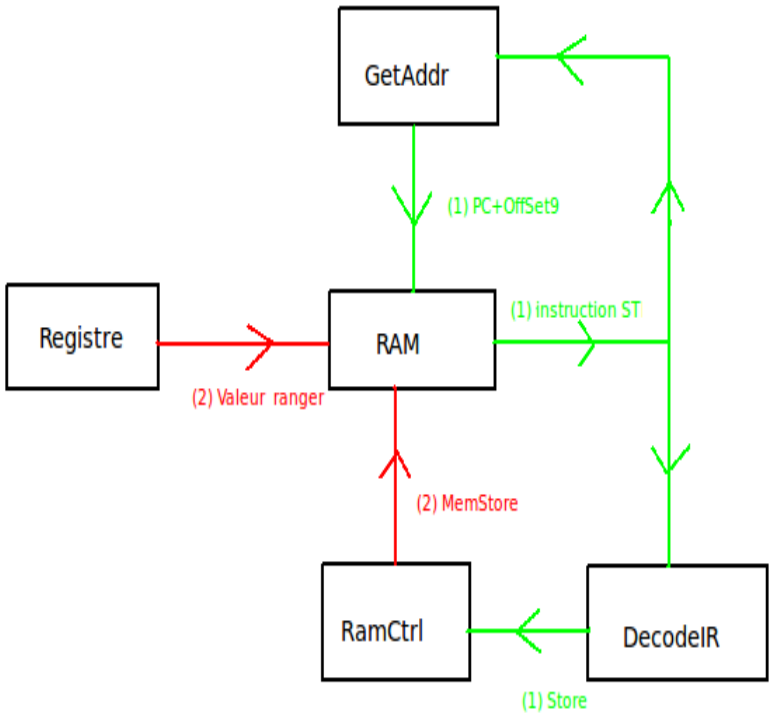
-LDI
Dans un premier temps GetAddr va être egale au PC+Offset, on va charger la mémoire à cette endroit et prendre la valeur comme nouvelle adresse de chargement



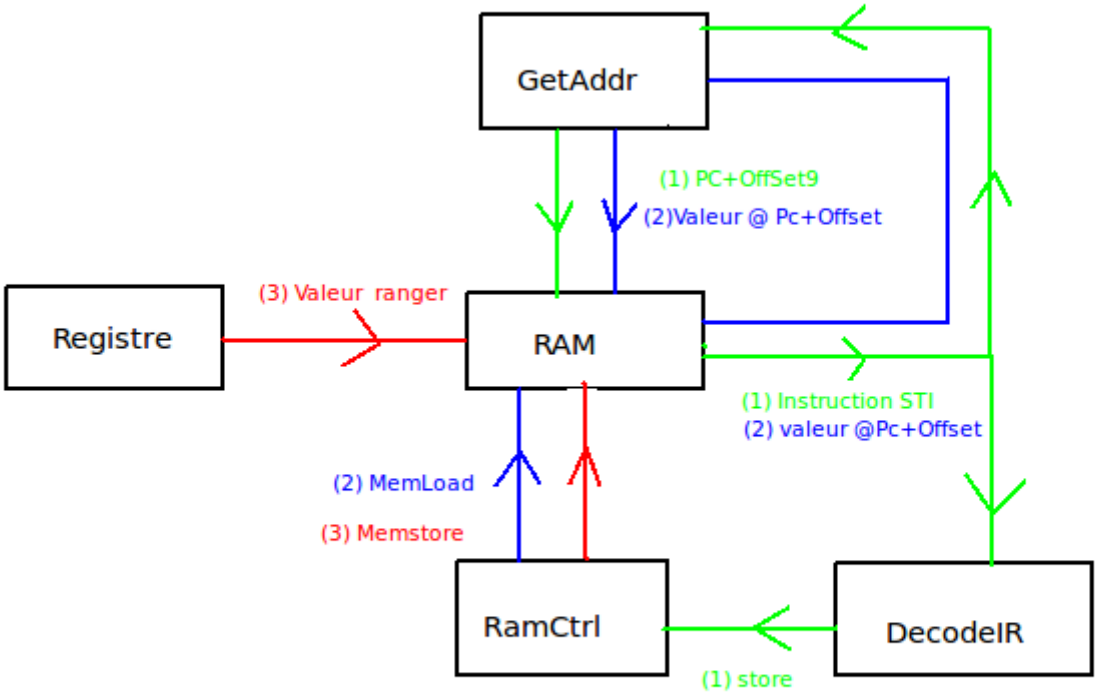
Instructions de rangement :
Comme pour Load, GetAddr calculer l'emplacement mémoire à modifier.

-ST
GetAddr calculer la somme du PC + de l'offset, on écrit ensuite la valeur de SR a cette emplacement.
-STR
GetAddr calcule cette fois la somme de la valeur du registre BaseR et de l'offset.



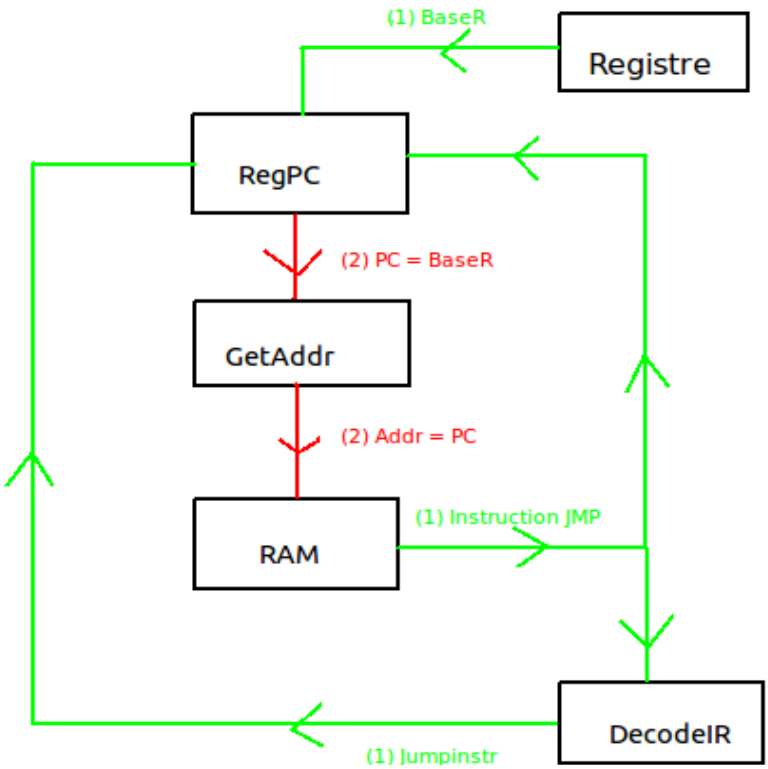


-STI
Comme pour LDI, on va charger la mémoire a l'emplacement PC+OffSet et utiliser cette valeur comme adresse pour y ranger notre valeur.

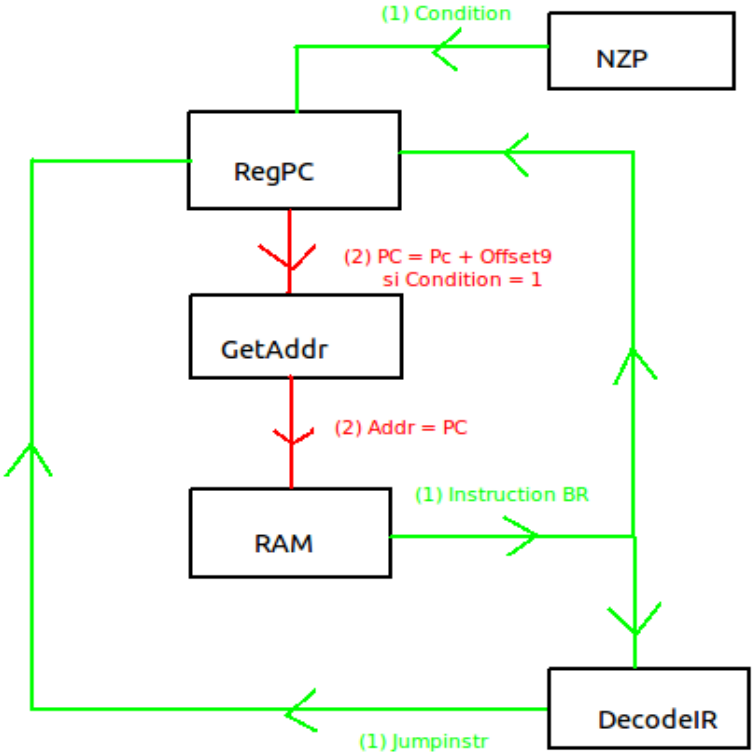


Instruction de branchement :
Les calcules se font cette fois dans RegPC, si il s'agit d'une instruction de saut, PC prend la valeur calculé, sinon on incrémente seulement.

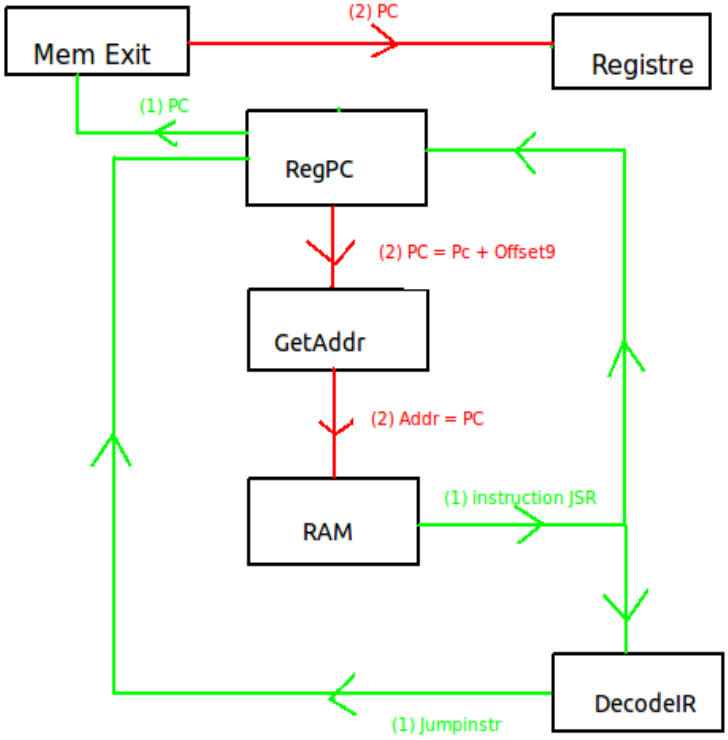
-BR
Si la condition NZP est respecter, PC est egal à la somme du PC + de l'offset.
-JMP
PC est égale à la valeur contenue dans BaseR

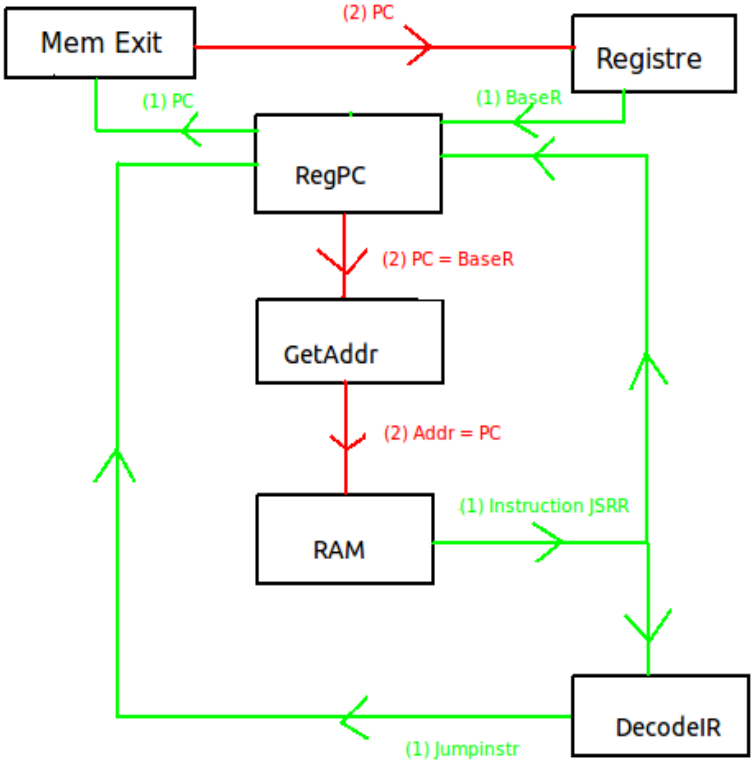


-



JSR & JSSR
RegPC va calculer la bonne valeur du PC en fonction de l'offset ou de la valeur du registre et va aussi sortir l'information comme quoi il s'agit d'une instruction JSR/JSRR.
Cette information sera réutilisé pour pouvoir écrire l'ancienne valeur du PC dans R7.





Autre module modifié :

-DecodeIR
On regarde la valeur des bit 13 et 12 pour savoir de quel type d'instruction il s'agit.
00 pour Jump, 01 pour Artith, 10 pour Load et 11 pour Store.
Si la parité des bits 13 et 12 = 1 (01 ou 10), alors c'est une instruction de chargement ou arithmétique, du coup WriteReg = 1.

-NZP
Si WriteReg est à 0, NZP de change pas.
Sinon on compare RES à zéro avec un comparator, on récupère le bit > comme bit 0, le bit = comme bit 1 et le bit < comme bit 2.
pour TestNZP, pour P il faut que le bit 0 de NZP et quel le bit 9 de IR soient égale à 1
pour Z, il faut que le bit 1 de NZP et le 10 de IR soient égale à 1,
enfin pour N il faut que le bit 2 de NZP et le bit 11 de IR soient égale à 1, si une de ces condition est respecter, TestNZP vaut 1.

```
        ;; Programme qui prend un tableau et un entier, ajoute l'entier a tout les element, et remplace l'entier par la somme des element du tableau après addition
        .ORIG x3000
        AND R4,R4,0         ;R4 <- 0
        LEA R0,entier       ;R0 <- @entier
        ST R0,addr          ;mem[addr] <- @ntier
        LDI R0,addr         ;R0 <- entier
        LEA R1,tab          ;R1 <- @T[O]
        LD R2,len           ;R2 <- len(T)
        BRnzp 1                 ;ingore la prochaine instruction
loop:   ADD R1,R1,1         ;R1++
        JSR addtab          ;exectute la sous-routine addtab
        ADD R2,R2,-1                ;R2--
        BRp loop            ;tant que R2>0
        STI R4,addr         ;entier = somme du tableau après addition
        BRnzp fini          ;fin
addr:   .FILL 0
len:    .FILL 3
tab:    .FILL 5
        .FILL 2
        .FILL 7
        ;; init : R0 = s1, R1 = @s2 | fin : @s2 <- s1+s2, s4 <- s4+(s1+s2), utilise R3
addtab:LDR R3,R1,0          ;R3 <- T[i]
        ADD R3,R3,R0                ;R3 <- R3 + R0
        STR R3,R1,0         ;t[i] <- R3
        ADD R4,R4,R3                ;R4 <- R4 + R3
        RET
entier: .FILL 3
fini:   .END
```

Ce programme test partiquement toute les instruction du LC3, les instructions de chargement LD, LDR, LEA et LDI, les instruction de rangement ST, STR et STI, les instructions de saut BR, JUMP (RET) et JSR (manque seulement JSRR).
Situation Initiale : T = [5, 2, 7] et entier = 3 // T[0] = mem[F], T[1] = mem[10], T[2] = mem[11]
Situation finale : T = [8, 5, 10] et entier = (17)16 = 23 // entier = mem[17] (en base 16)