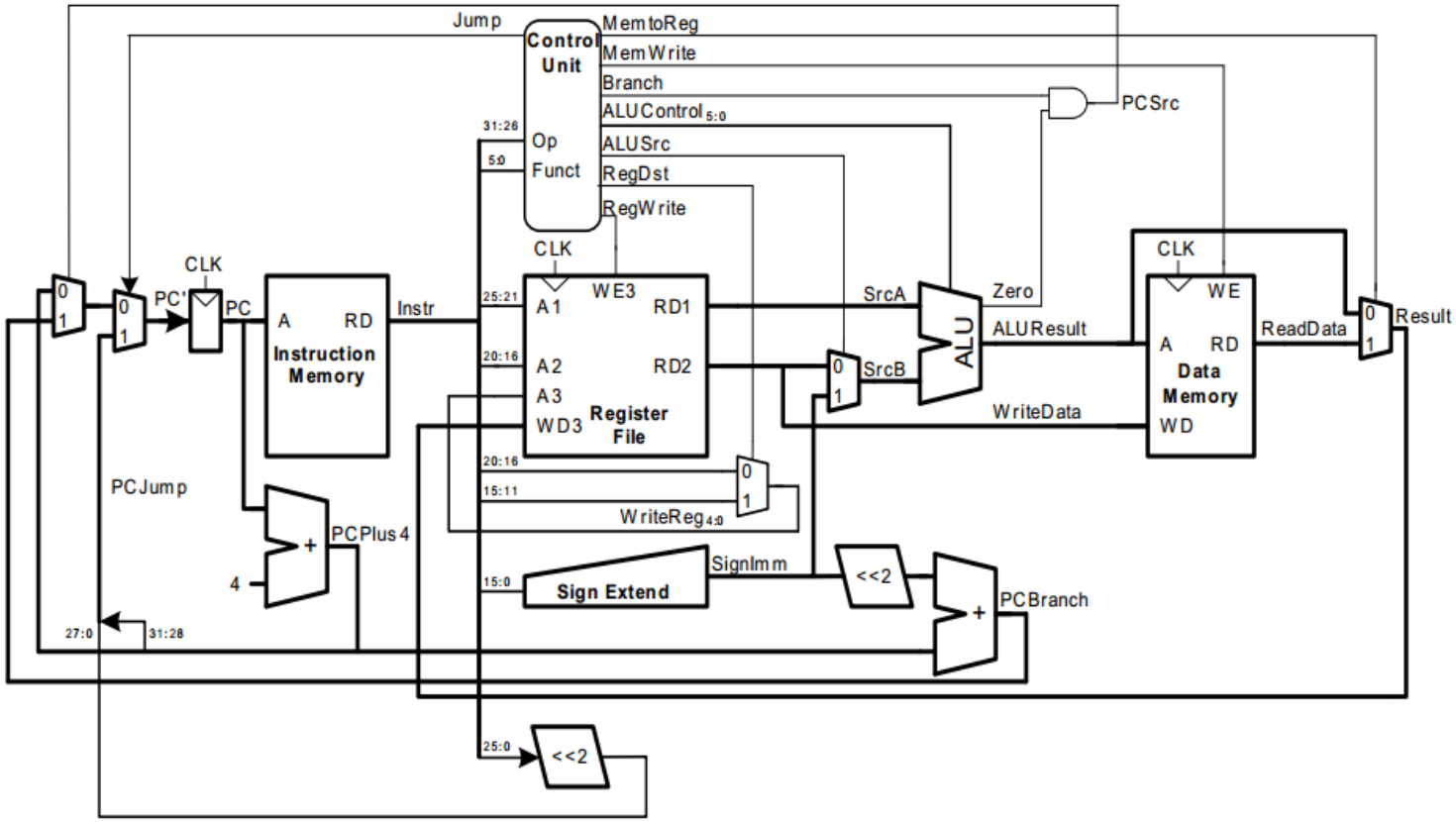


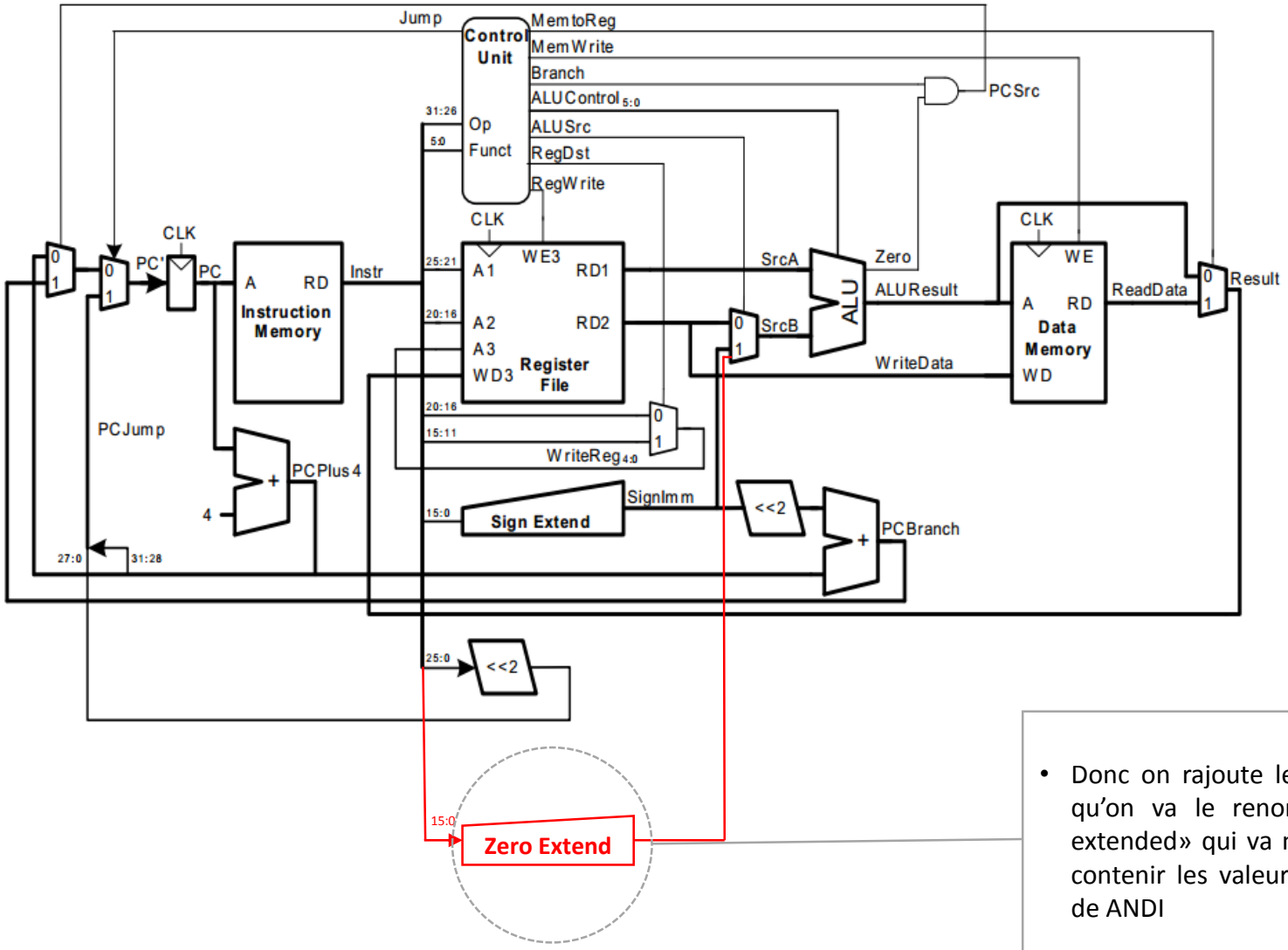
- Tout d'abord on doit comprendre l'opération ANDI.
  `andi St0, St0, 0x07`

  Andi va exécuter le « and » entre la valeur $t0 et la valeur immediate 0x07 (ici en hexa) et enregistre le résultat dans $t0

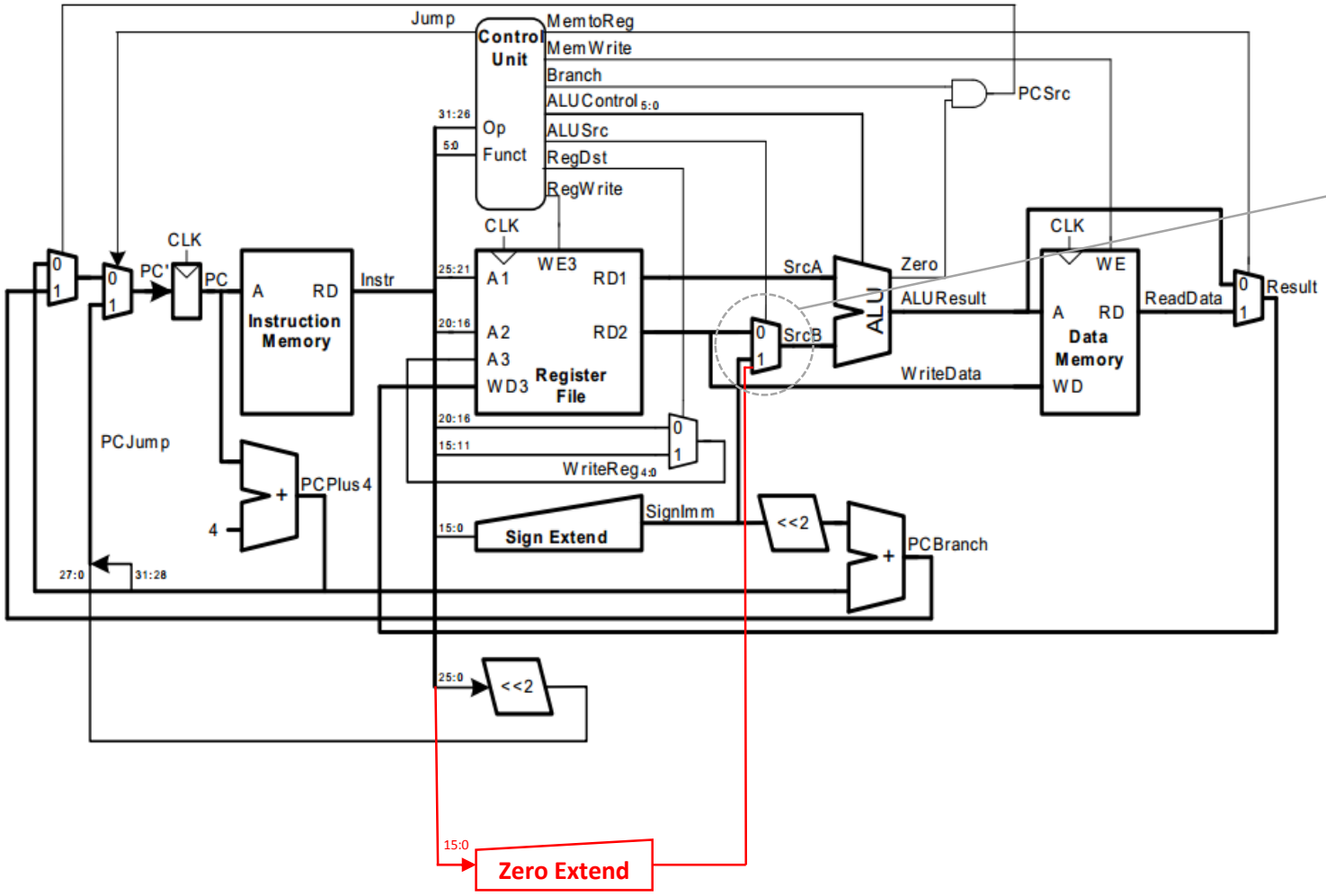
- Puis, et tant que ALU est le responsable des opérations logiques, on voit s'il a toutes les entrées nécessaires pour exécuter ANDI.

  ANDI n'utilise pas des valeur immediate signé, elle utilise plutôt des valeur de sorte que

  `0x07` devient `0x00000007`

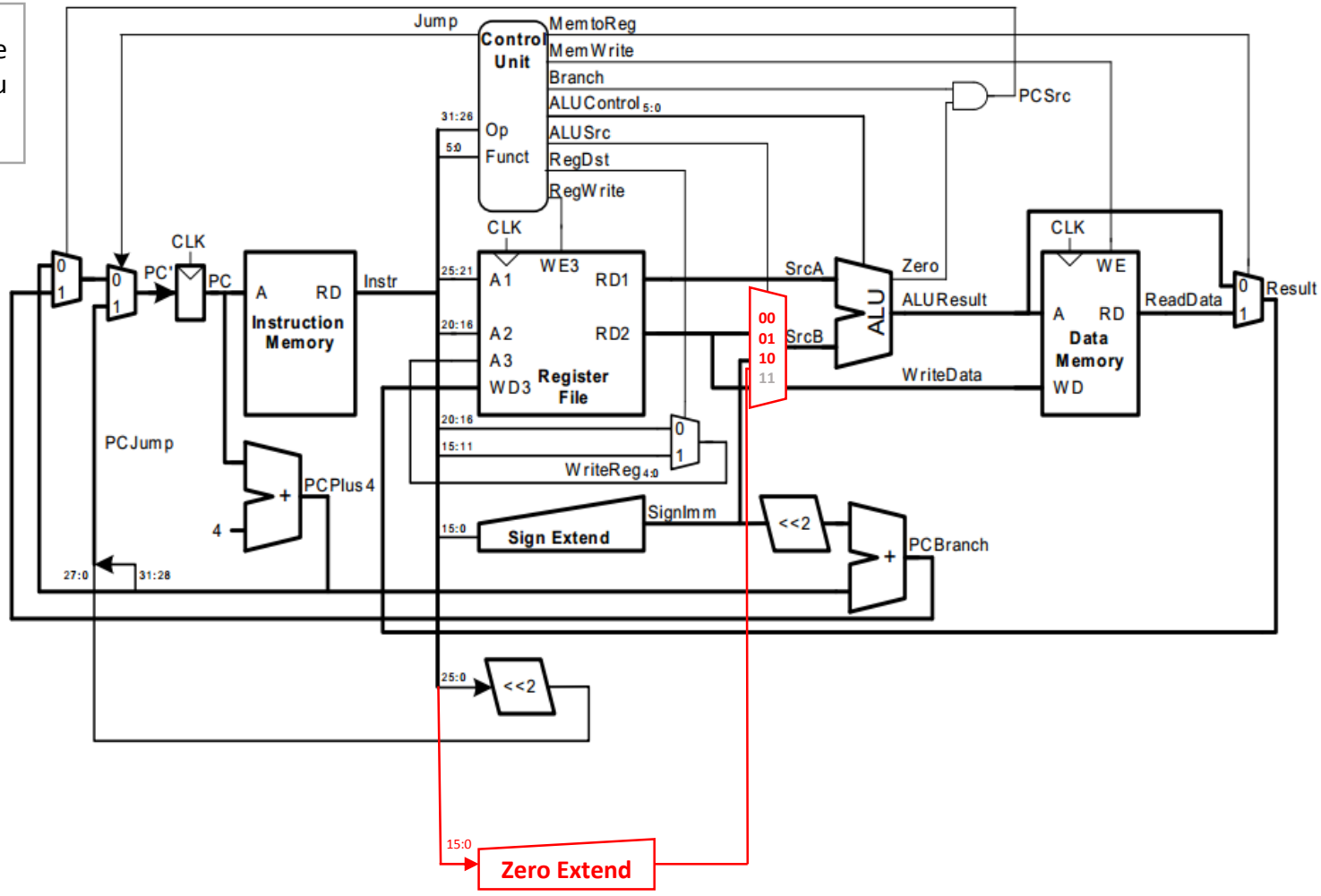


- Donc on rajoute le composant qu'on va le renommer «zero extended» qui va nous servir a contenir les valeurs immediate de ANDI



- On remarque que le multiplexeur ne couvre pas toutes les entrées

- Et voila, il nous reste maintenant la mise a jour du tableau des opérations



Instruction	$Op_{5:0}$	RegWrite	RegDst	AluSrc	Branch	MemWrite	MemtoReg	$F_{5:0}$	Jump
R-type	000000	1	1	**00**	0	0	0	100000 a+b 100010 a-b 100100 a and b 100101 a or b 100110 a xor b 100111 not(a or b) 101010 slt	0
lw	100011	1	0	**01**	0	0	1	100000	0
sw	101011	0	X	**01**	0	1	X	100000	0
beq	000100	0	X	**00**	1	0	X	100010	0
j	**000100**	**0**	**X**	**XX**	**X**	**0**	**X**	**XX**	**1**
addi	**001000**	**1**	**0**	**01**	**0**	**0**	**0**	**100000**	**0**
andi	**001100(12)**	**1**	**0**	**10**	**0**	**0**	**0**	**100100**	**0**

Valeurs selon le nouveau multiplexeur

La valeur de AND