LOBO TEIXEIRA André
MONTOVANELLI BOLDRINI Murillo

TP4 et TP5 : ELECTRONIQUE NUMERIQUE

Réalisation d'un microprocesseur

1) Les instructions du programme sont : 0210, 0011, 0112, 0212 et 0308.
Nous chargerons le contenu de la mémoire 10 qui vaut AAAA dans l'accumulateur, après nous ferons la somme avec le contenu de la mémoire 11 qui vaut 5555, nous stockerons la valeur du résultat qui est à l'accumulateur dans la mémoire 12, ensuite, pour vérifier, nous la chargerons et nous entrerons dans un boucle infini.

2) La sortie program_counter_out reçoit les valeurs du signal program_counter. Le program_counter a comme rôle determiner la sequénce des instructions qui seront processées pour le processeur et est incrementé à chaque cycle. Sa taille est de 8 bits.

La Sortie register_AC_out reçoit les valeur du signal register_AC. Le register_AC a comme rôle mantenir un valeur pour faire une operation ensuite avec un autre valeur et aussi acumuler le résultat. Sa taille est de 16 bits.

La sortie memory_data_register_out reçoit les valeurs du signal memory_data_register. Le memory_data_register a comme rôle donner une instruction ou une valeur au register_AC. Sa taille est de 16 bits.

La sortie memory_adress_register_out reçoit les valeurs du signal memory_adress_register. Le memory_adress_register a comme rôle reçoit un adress du program_counter ou du memory_data_register pour indiquer quel adress memoire sera lu et executé. Sa taille est de 8 bits.

La sortie instruction_register_out reçoit les valeurs du signal instruction_register. L' instruction_register récupère les instructions lues grâce au pointeur program_counter.

L'intérêt d'ajouter le signal memory_write_out est de permettre la visualisation d'une écriture memoire, c'est à dire, pour savoir quand une fonction d'stockage est utilisée.

3)

3.1) Le rôle de cette partie du code est actualiser la valeur du MAR par chaque status qui est executé. En plus, il y a une modification du MW à la valeur 1 quand l'instruction store est executé et il vaut 0 par les autres instructions. Le instruction_register est mis à jour dans l'instruction fetch.

3.2) Le Memory_data_register est mis à jour à chaque clock et en fonction de la valeur du memory_adress_register. Le contenu lorsque le state vaut Reset_PC est égal à la valeur de l'adresse 00, lorsqu'il vaut fetch il reçoit la valeur du contenu à l'adresse du program_couter. Si le state vaut decode, store ou execute-add, le memory_adress_register prend la valeur des bit de poids faible de l'instruction_register et ensuite le memory_data_register reçoit la valeur du contenu de cette adresse.

3.3) Non, le code est dehors du processus, donc il sera exécuté en parallèle. Il n'y a pas de différence s'il est placé avant ou après le processus.

4) Le Reset est asynchrone, malgré qu'il dépend du prochain clock pour arriver, après qu'il

est appuyé le reset sera l'instruction à être exécutée. Le tableau qui concerne à cette question est en pièce jointé.

5) Les bonnes fonctionnement des instructions peuvent être vues d'après les archives .MIF en pièce jointé.

6) Nous proposons un fichier <<PROGRAMFINAL.MIF>> qui montrera toutes instructions créées. Le tableau ci-dessous montre les instructions et ses respectifs Opcodes et descrptions.

Instruction	Opcode	Description
ADD	00	AC <= AC + contenu de la mémoire
STORE	01	contenu de la mémoire <= AC
LOAD	02	AC <= contenu de la mémoire
JUMP	03	PC <= addresse
JZ	10	Branchement si AC = 0000
JNZ	11	Branchement si AC != 0000
OR	20	AC or contenu de la mémoire
AND	21	AC and contenu de la mémoire
XOR	22	AC xor contenu de la mémoire
LDI	30	AC <= addresse
ADDI	31	AC <= AC + addresse
LSL	40	décalage logique à gauche du AC
LSR	41	décalage logique à droite du AC
ROL	42	rotation à gauche du AC
ROR	43	rotation à droite du AC
IN	50	AC <= port d'entrée
OUT	51	port de sortie <= AC
TEST_IN	52	AC <= port d'entrée et si le addresse eme bit et 1 une variable stocke 1
TEST_RES	53	Branchement si la valeur stocké en TEST_IN vaut 1

Nous avons fait un .bdf pour illustrer le fonctionnement du programme. Dans l'Image 1 il y a la partie de sélection du clock et dans l'Image 2 il y a la partie du processeur avec ses sorties et aussi les afficheurs.

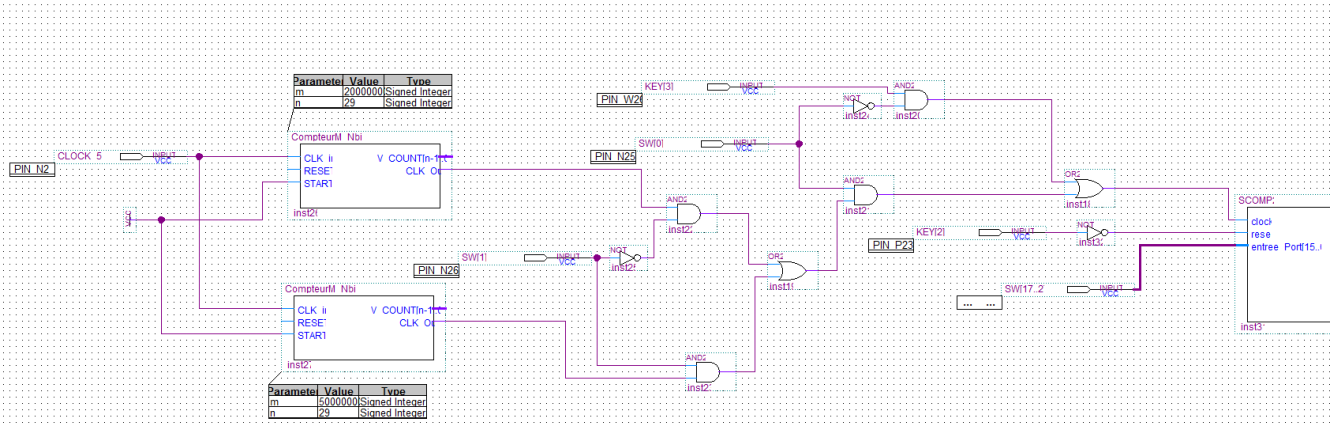

Image 1. La sélection du clock.

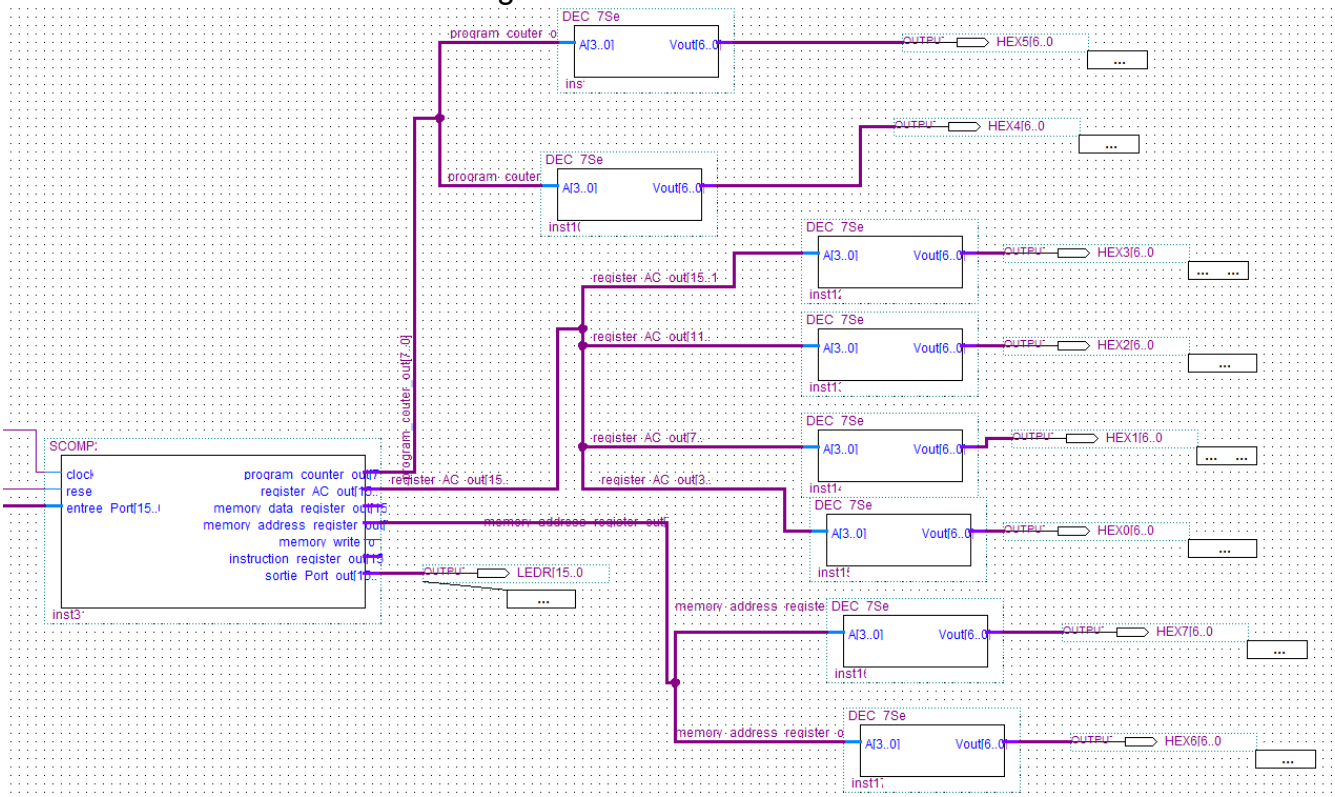

Image 2. Le processeur, ses sorties et les afficheurs.

Les premières instructions du programme sont : 023D, 003E, 013F, 023F et 0308. Nous chargerons le contenu de la mémoire 3D qui vaut AAAA dans l'accumulateur, après nous ferons la somme avec le contenu de la mémoire 3E qui vaut 5555, nous stockerons la valeur du résultat qui est à l'accumulateur dans la mémoire 3F, ensuite, pour vérifier, nous la chargerons et nous ferons un saute à l'addresse 08, qui aura la prochaine instruction.

Les instructions suivantes sont 100A, 1110 et 030A. Nous savons que nous avons chargé la valeur FFFF de l'adresse 3F et maintenant elle est dans l'accumulateur. Donc l'instruction 100A sera faux est nous ne sauterons pas à l'adresse 0A. Par contre, L'instruction 1110 sera vrai et le programme compteur sera 10. Le PC n'arrive pas à 030A, qui mettrait le programme dans un boucle infini, car l'instruction 030A a l'adresse 0A.

Ensuite nous avons 0245, 2045, 1014, 0313, 0246, 2147, 1118 et 0317. Nous ferons un OR entre la valeur 0000, qui est dans l'adresse 45, et elle même. Nous testons si le résultat est correct, donc 0000, et si oui le PC vaut 14 et le programme n'entre pas dans un boucle en 13. Après nous ferons un AND entre les valeurs 1111 et 1111 qui sont, respectivement, dans 46 et 47 et nous testons si le résultat n'est pas nul avec 1118. Comme il n'est pas, le PC vaut 18 et pas 17.

Ensuite : 024E, 224F, 1120 et 031B. Nous ferons un XOR entre 1010 et 0101 et comme le résultat ne vaut pas nul le PC vaut 20.

Les instructions suivantes sont 3000, 1023, 0322, 31FF, 31FF et 0328. Nous ferons un chargement immédiat du 0000 à l'accumulateur et après nous testons si AC vaut 0000. Comme c'est vrai, nous sautons le boucle 0322, nous ferons deux additions du FF avec l'instructions 31FF et après nous mettons le PC à 28 pour aller à l'instruction suivante.

Pour les instructions des lignes 028, 030, 038 et 040, qu'on peut voir dans l'image 3, nous testerons, respectivement, le décalage logique à gauche, à droite, la rotation à gauche et à droite.

Comme le décalage à gauche d'un seule bit de la valeur 0101 donnera le même résultat de la rotation à gauche de 0101, nous utiliserons cette valeur(laquelle dans l'adresse 56) et la valeur espérée(laquelle dans l'adresse 57) deux fois. Par le décalage à droite la valeur utilisée et la valeur espérée sont, respectivement, à l'adresse 5E et 5F. Par la rotation à droite la valeur utilisée et la valeur espérée sont, respectivement, à l'adresse 66 et 67.

Donc, premièrement nous chargeons la valeur, nous faisons l'opération, soit décalage ou soit rotation, après nous faisons un XOR avec la valeur espérée qui sera à une position suivante à l'adresse décalée(ou tournée) et, à la fin, nous testons si le résultat est nul. S'il est nul, le PC reçoit l'adresse de l'instruction de la ligne suivante. Sinon, le programme entrera dans un boucle infini. Alors, si les instructions marchent bien, le programme n'entrera pas aux boucles car l'opération XOR avec deux valeur égales est toujours nul.

028	0256	4001	2257	1030	032C	0000	0000	0000
030	025E	4101	225F	1038	0334	0000	0000	0000
038	0256	4201	2257	1040	033C	AAAA	5555	0000
040	025E	4301	2267	1068	0344	0000	1111	1111
048	0000	0000	0000	0000	0000	0000	1010	0101
050	0000	0000	0000	0000	0000	0000	0101	0202
058	0000	0000	0000	0000	0000	0000	0101	0080
060	0000	0000	0000	0000	0000	0000	0000	8080

Image 3. Les instructions du décalage et rotation.

Les dernières instructions sont 5000, 5100, 5055, 5177, 5201, 5370, 0368 et 0370. Nous affichons deux fois qu'est-ce que il y a aux portes d'entrée aux portes de sortie en utilisant les instructions 50 et 51. Nous affichons deux fois pour montrer que l'opérateur n'influence pas dans ces instructions. Après nous utilisons l'instruction 5201 et ensuite 5370. Si le bit moins faible à l'entrée vaut 1 le programme entrera dans un boucle infini à l'adresse 70, sinon il retournera à première instruction de cette partie, l'instruction 5000. Donc, le programme s'arrête quand on met 1 au bit moins faible de l'entrée.