Exercice 1 (8 points)

On veut construire un processeur spécialisé pour exécuter le programme suivant :
A ← inport
B ← inport
C ← A + B
C ← C + A
C ← C + B
C ← C >> 1
outport ← C

Question 0 : Quel est le résultat (la valeur envoyé sur OUTPORT) du programme?

Question 1 : De quels éléments (bascules pour le stockage et circuits logiques pour le traitement) a-t-on besoin dans l'unité de traitement de ce processeur ?
L'adressage des registres se fait par l'envoi d'un numéro binaire, donc le nombre de registres devrait être un multiple d'une puissance de 2.

Question 2 : Combien de registres devrons-nous mettre dans cette unité de traitement ?

On utilise une « register file « qui permet à chaque clock :

- Deux lectures simultanées de deux registres, en donnant leur adresse (RAA et RAB) et produisent deux data A et B en sortie.
- Une écriture sur un registre, on peut écrire une data en entrée A' sur le registre d'adresse WA, en validant le signal enable correspondant (WEN=1). La valeur lue sur ce registre à cette période est la valeur avant écriture. La valeur écrite sera la valeur lue à la prochaine période.

Cette unité de traitement possédera deux signaux additionnels, en entrée, Reset qui remet à zéro tous les registres et bien sur CLK.

Question 3 : Schématiser l'entité et construire cette register file. Les signaux en entrées sont A', WA, WEN, clk, Reset, RAA, RAB et les signaux de sortie sont : A et B. (on utilisera des bascules de 8 bits de largeur FDR8 avec reset synchrone et des multiplexeurs/démultiplexeurs de largeur 8 bits)

Pour les opérations sur les données, nous utiliseront une ALU suivie d'un Shifter. L'ALU et le shifter doivent doit être capables de réaliser le minimum d'opérations possible.
Question 4 : Schématiser l'entité composée de l'ALU et du Shifter en montrant les signaux d'entrée sortie.

Question 5 : Donner un codage binaire des opérations de l'ALU et du Shifter (en utilisant les signaux d'entrée de l'ALU et du shifter)

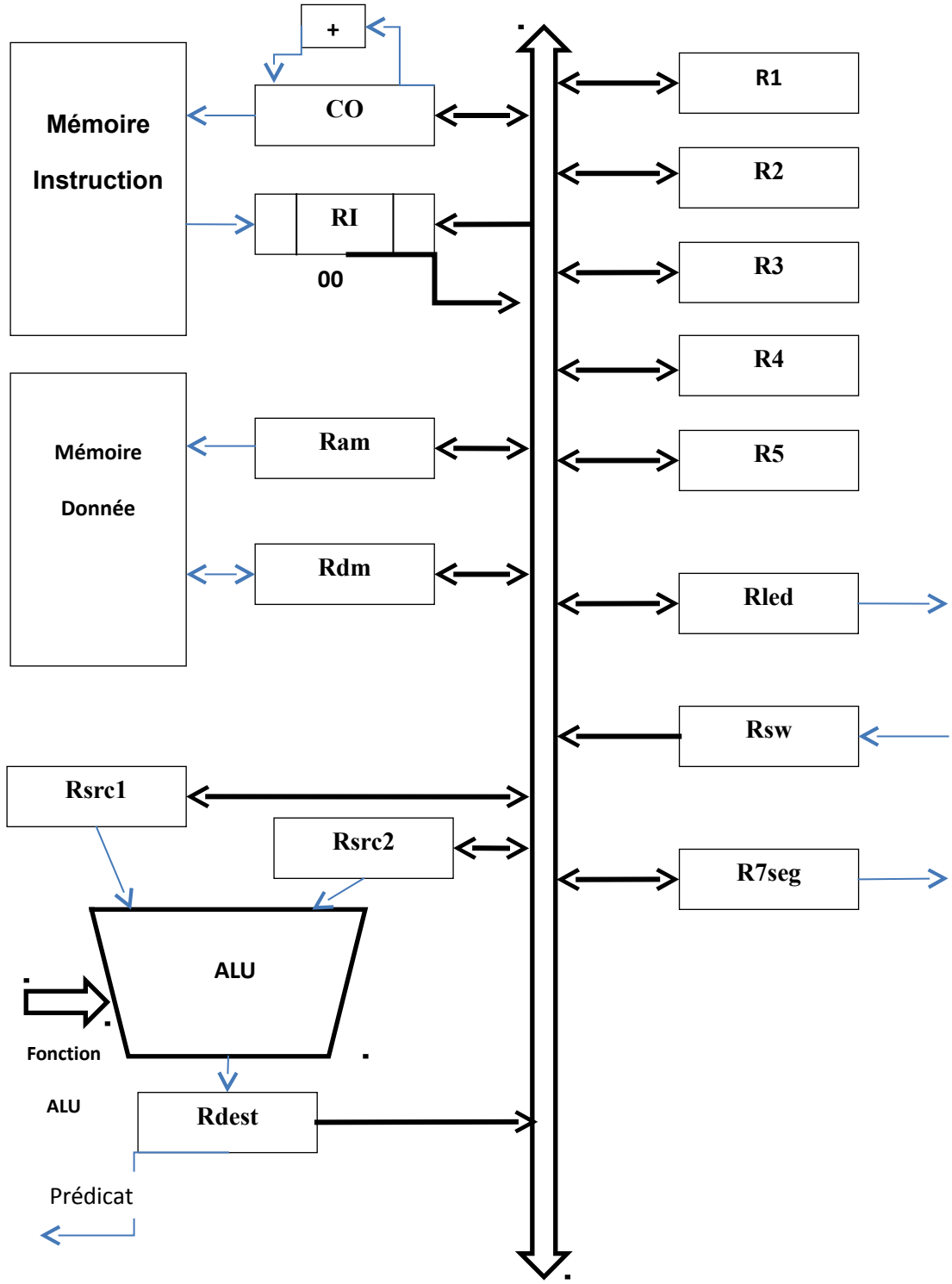
Question 6 : Proposer un schéma de l'unité de traitement (ALU + Shifter + Register file) réalisant l'algorithme (vous utiliserez un comparateur 8 bits pour comparer à chaque itération la sortie du shifter et zéro).

Question 7 : Combien de bits sont nécessaires au contrôle de cette unité ? Proposez une

codification par champs

Question 8: Proposez une FSM contrôlant cette unité de traitement avec un nombre minimal d'états

Exercice 2 (10 points)



Le processeur S3 est un processeur 16 bits autant pour les données que pour les adresses. Le jeu d'instructions est réduit et se situe au niveau du transfert de registre. Les instructions sont décodées puis exécutées. S3 anticipe le chargement de l'instruction suivante pendant l'exécution du chargement en cours : on parle de pipeline à deux étages. Les flèches épaisses correspondent aux microcommandes qui permettent de déclencher des actions depuis la FSM. La flèche particulière en sortie de *RI* transfert les 8 bits de *RI(11 :4)* concaténés à gauche avec x00. Cela construit un mot de

16 bits à partir d'une partie de RI. La flèche *prédicat* est utilisée par la FSM, elle correspond à '0' si tous les bits de Rdest sont égaux à '0' et '1' sinon. L'ALU peut exécuter 16 fonctions différentes, certaines vous seront données par la suite. Le registre Rled est toujours connecté aux leds de la carte Nexys2 en sortie, R7seg à l'afficheur 4 digits 7 segments et Rsw aux switches en entrée les 8 bits de poids forts sont alors forcés à x00. L'exercice consiste à écrire des programmes assembleurs pour le S3. Voici la liste des instructions.

MOV Rs Rd : transfert de registres Rs : tous les registres qui ont une flèche vers le bus
Rd : tous les regsitres qui ont une flèche depuis le bus

ADD : Rdest = Rscr1 + Rscr2
SUB : Rdest = Rscr1 - Rscr2
ADDC : Rdest = Rscr1 + Rscr2 + 1
SUBC : Rdest = Rscr1 - Rscr2 – 1
INV : Rdest = NOT Rsrc1
AND : Rdest = Rsrc1 AND Rsrc2
OR : Rdest = Rsrc1 OR Rsrc2
INC : Rdest = Rsrc1 + 1
COMP2 : Rdest = NOT Rsrc1 + 1
CONCAT : Rdest = Rsrc2(7 :0) & Rsrc1(7 :0)
MVI xdd Rd : transfert de x00dd vers Rd
MZ Rs Rd : idem que MOV mais seulement si prédicat = '0'
MNZ Rs Rd : idem que MOV mais seulement si prédicat <> '0'
MVIZ xdd Rd: idem que MVI mais seulement si prédicat = '0'
MVINZ xdd Rd: idem que MVI mais seulement si prédicat <> '0'
PAUSE : bloque l'exécution jusque la pression du bouton Btn0 de la carte.

Les transferts de registres vers le CO ont pour effet de changer le déroulement séquentiel du programme vers une nouvelle adresse : celle qui sera transférée dans CO. Comme le processeur est un pipeline de 2 étages, quand le changement du CO est effectif, l'instruction suivante a déjà été chargée, elle sera quand même exécutée avant le branchement : on parle de branchement retardé.

Exemple
```
00 MVI 04 CO
01 ADD
02 …..
03 …..
04 PAUSE
```
L'instruction ADD est exécutée puis on passe à l'adresse 04 donc sur le PAUSE.

On rajoute à ce processeur les deux instructions READ et WRITE.

- READ met dans le registre RDM la donnée se trouvant dans la mémoire de données à l'adresse indiquée dans RAM.
-WRITE écrit dans la mémoire de données à l'adresse se trouvant dans RAM la donnée présente dans RDM.

On supposera aussi que l'ALU sait réaliser les deux opérations indiquées par les instructions suivantes :

- LEFT : transfert les 8 bits de poids fort de Rsrc1 vers Rdest.
- right: transfert les 8 bits de poids faible de Rsrc1 vers Rdest.

Question 1 :

Le triangle de Pascal se construit de la façon suivante :

Sous forme triangulaire, *i* étant l'indice de ligne et *j* l'indice de colonne :

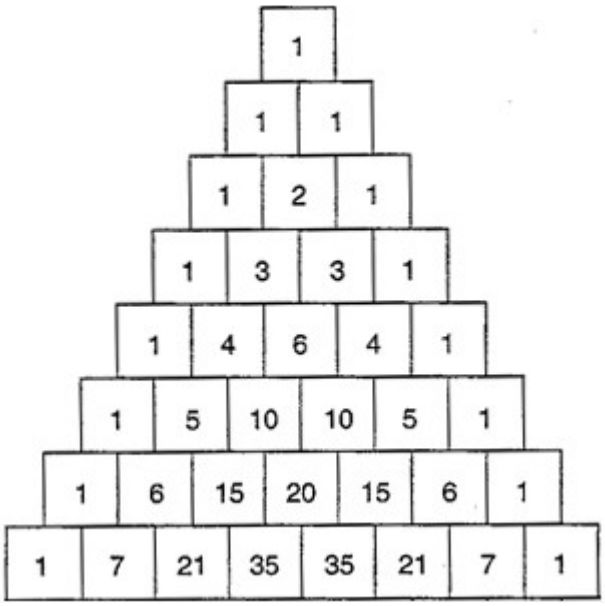
- placer dans la colonne 0 des 1 à chaque ligne, et des 1 à chaque entrée de la diagonale,
- en partant du haut et en descendant, compléter le triangle en additionnant deux coefficients adjacents d'une ligne, pour produire le coefficient de la ligne inférieure, en dessous du

coefficient de droite.

Sous forme triangulaire, *i* étant l'indice de ligne et *j* l'indice de colonne :

- placer dans la colonne 0 des 1 à chaque ligne, et des 1 à chaque entrée de la diagonale,
- en partant du haut et en descendant, compléter le triangle en ajoutant deux coefficients adjacents d'une ligne, pour produire le coefficient de la ligne inférieure, en dessous du coefficient de droite.

Les premières lignes du triangle de pascal sont données dans la figure ci-dessous :



Question 1.1. : Donner le programme assembleur permettant de lire un entier ***i*** (supérieur à 1) se trouvant en mémoire à l'adresse 0100h, et qui l'affiche sur les LEDs.

Question 1.2. : Donner le programme permettant de calculer la ***i*** eme ligne du triangle à partir de la ligne ***i*** – 1.

Les éléments de la ligne ***i*** – 1 se trouvent déjà en mémoire à partir de l'adresse 1000h, et ceux de la ligne ***i*** doivent être stockés en mémoire à partir de l'adresse 5000h

Question 1.3. : Donner le programme permettant d'afficher les éléments de la ***i*** eme ligne du triangle (***i*** se trouvant en mémoire à l'adresse 0100h). On affichera un élément de la ligne à la fois, suivi d'une pression sur un bouton.

Remarques : *Dans toute la question 1 on considère que :*
- un entier est représenté sur 16 bits
- le nombre i est inférieur ou égal à 10

Question 2 :

Un palindrome de lettres, est une figure de style désignant un texte ou un mot dont l'ordre des lettres reste le même qu'on le lise de gauche à droite ou de droite à gauche, comme dans la phrase « Esope reste ici et se repose »

Soit une chaîne de caractères se terminant par le caractère 00h (00h ne peut pas se trouver ailleurs qu'à la fin de la chaîne) et stockée en mémoire à partir de l'adresse 1000h.

Sachant qu'**un caractère est représenté sur 8 bits**, donner le programme affichant 1 sur le 7-seg si la chaîne est un palindrome et 0 sinon.