Ing1 – Examen d'Architecture des Ordinateurs

Durée : 2H – Aucun document autorisé – Calculatrice Interdite.
Documentation assembleur Z80 en annexe.

Exercice 1 – Espace adressable

On considère une machine avec une CPU gérant des adresses de 20 bits ($a_{19}…a_0$) et des mots de 32 bits ($d_{31}…d_0$). La machine intègre une ROM, 2 barrettes de RAM ainsi qu'une interface série et une interface parallèle pour les périphériques. On a les équations suivantes pour les circuits d'activation des composants adressables (actifs à niveau haut) :

- $CS_{ROM} = a_{19}\overline{a_{18}}a_{17}$
- $CS_{RAM\_1} = \overline{a_{19}}\overline{a_{18}}$
- $CS_{RAM\_2} = \overline{a_{19}}a_{18}$
- $CS_{I.SERIE} = a_{19}\overline{a_{18}}\overline{a_{17}}\overline{a_{16}}\overline{a_{15}}\overline{a_{14}}\overline{a_{13}}\overline{a_{12}}\overline{a_{11}}\overline{a_{10}}a_9a_8a_7a_6a_5$
- $CS_{I.PARALLELE} = a_{19}a_{18}\overline{a_{17}}\overline{a_{16}}\overline{a_{15}}\overline{a_{14}}\overline{a_{13}}\overline{a_{12}}\overline{a_{11}}\overline{a_{10}}\overline{a_9}a_8a_7a_6$

1.a) Quel est l'espace adressable maximal de cette machine (en mots) ?

1.b) Quelle est la plage d'adresses (en hexa) ?

1.c) Quelle sont les tailles maximales de la ROM et de la RAM (en octets) ? Utiliser l'unité adéquate.

1.d) Combien de registres séries peut-on adresser ?

1.e) Combien de registres parallèles peut-on adresser ?

1.f) Compléter le tableau suivant :

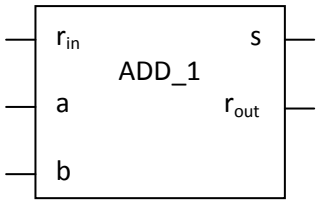
	Adresse minimale (hexa)	Adresse maximale (hexa)
ROM		
RAM_1		
RAM_2		
Interface série		
Interface parallèle		

1.e) Pour résumer, dessiner la carte mémoire du système (mapping des adresses).

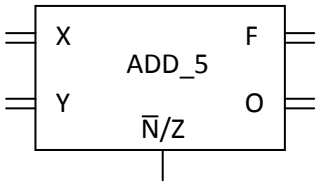
NB : rappel des unités de donnés : K, M, G, T, P, E, Z, Y

Exercice 2 – Calcul sur les entiers

Soit l'additionneur complet 1 bit classique ADD_1 :



On considère des entiers codés sur 5 bits et on souhaite faire des additions 5 bits sur des entiers naturels ou relatifs. On construit donc un additionneur 5 bits ADD_5 répondant à ces besoins à partir de 5 additionneurs complets ; le circuit additionne X ($X_4X_3X_2X_1X_0$) et Y ($Y_4Y_3Y_2Y_1Y_0$) et donne le résultat de l'addition sur les 5 bits de F ($F_4F_2F_2F_1F_0$) ainsi la validité du résultat avec le bit d'overflow O ; le signal $\overline{N}/Z$ permet de sélection l'addition sur des naturels (0) ou sur des relatifs (1).



2.a) Quel est l'intervalle des entiers naturels codables sur 5 bits ?

2.b) Quel est l'intervalle des entiers relatifs codables sur 5 bits en complément à 2 ?

2.c) Donner une condition d'overflow pour l'addition des nombres naturels.

2.d) Donner une condition d'overflow pour l'addition des nombres relatifs.

2.e) Donner le câblage interne de l'additionneur ADD_5 à partir de composants ADD_1 et de portes logiques de base (OR,AND,NOT,XOR,NOR,NAND,NXOR).

2.f) Donner le code 5 bits des entiers naturels : X=15 et Y=25.

2.g) Donner les détails de l'addition naturelle de X et Y par ADD_5. Interpréter le résultat (F, O).

2.h) Donner le code 5 bits en complément à 2 des entiers relatifs : 15 et -7.

2.i) Donner les détails de l'addition en relatif de X et Y par ADD_5. Interpréter le résultat (F, O).

Exercice 3 - Assembleur

On s'intéresse à la traduction en assembleur Z80 du programme Java suivant :

```
1. int v1;       // données sur 1024 bits à l'adresse 0x0100
2. int v2;       // données sur 1024 bits à l'adresse 0x0180
3. int somme;    // données sur 1024 bits à l'adresse 0x0200

4. somme = v1 + v2;
```

Donner la traduction du calcul de la somme (ligne 4) en assembleur en Z80.

Remarque : Chaque entier est codé sur 128 mots ; les poids faibles sont rangés avant les poids forts. Par exemple, si v1 contient l'entier 0xAF…..89, on a en mémoire

Adresse	Donnée
0x0100	0x9
0x0101	0x8
…	
0x0226	0xF
0x0227	0xA

Une fiche récapitulative du langage assembleur Z80 est donnée en annexe.

Exercice 4 – VHDL et Chronogramme

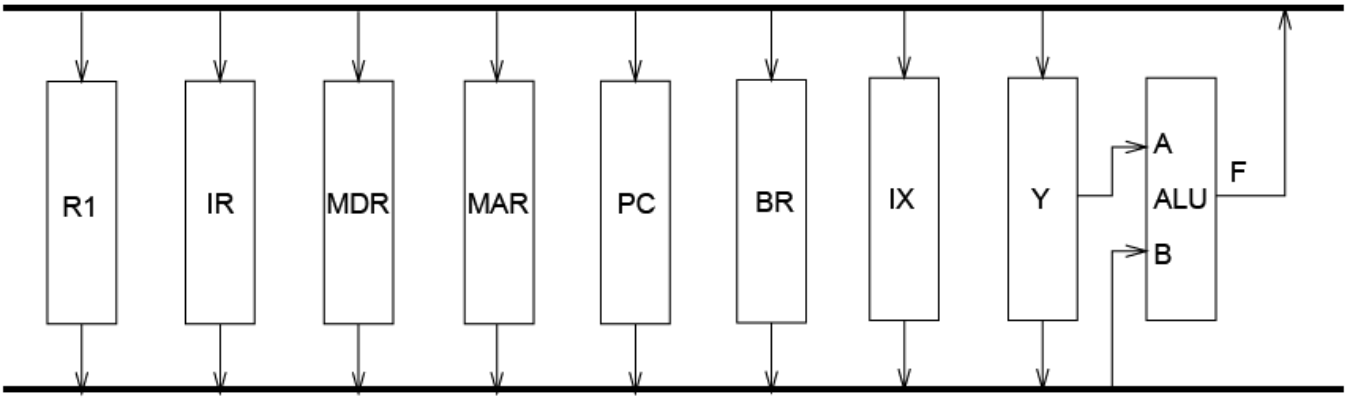
On considère l'extrait de programme VHDL suivant :

```
entity exo4rat is
end exo4rat;

architecture exo4rat_arch of exo4rat is
signal clock, reset : std_logic;
signal a, b, c, d : integer;
begin

d <= c;

process(clock,reset) -- process a, b, c synchronous management
begin
     if reset = '0' then
          a <= 1;
          b <= 2;
          c <= 3;
     elsif clock'event and clock = '1' then
          a <= a + 1;
          b <= a;
          c <= b;
     end if ;
end process;

process -- clock process
begin
     clock <= '0';
     wait for 10 ns;
     clock <= '1';
     wait for 10 ns;
end process;

process -- simulation process
begin
     reset <= '0';
     wait for 1 ns;
     reset <= '1';
     wait for 1000 ns;
end process;

end exo4rat_arch;
```

Dessiner le chronogramme de simulation sur 10 cycles d'horloge pour les signaux clock, reset, a, b, c et d.

Exercice 5 – CPU et µ-instructions

On considère une organisation de CPU à deux bus donnée par la figure suivante :



Registres :

Abréviation	Termes anglais	Termes francais
IR	Instruction Register	Registre d'Instruction
MDR	Memory Data Register	Registre de Données Mémoire
MAR	Memory Address Register	Registre d'Adresses Mémoire
PC	Program Counter	Compteur Ordinal
SP	Stack Pointer	Pointeur de Pile
ALU	Arithmetical and Logical Unit	Unité Arithmétique et Logique

µ-instructions :

`Reg out`	Sort le contenu du registre `Reg` sur le bus
`Reg in`	Met la valeur du bus dans le registre `Reg`
`Lecture`	Effectue une lecture de la mémoire à l'adresse `MAR` et met le résultat dans `MDR`
`Écriture`	Effectue une écriture dans la mémoire à l'adresse `MAR` de la valeur contenue dans `MDR`
`Attente`	Permet d'attendre la fin d'une lecture ou d'une écriture
`ADD`	Additionne les entrées `A` et `B` de l'ALU, met le résultat sur la sortie de l'ALU (`F=A+B`)
`INCRA`	Incrémente l'entrée `A` de l'ALU (`F=A+1`)
`DECRA`	Décrémente l'entrée `A` de l'ALU (`F=A-1`)
`INCRB`	Incrémente l'entrée `B` de l'ALU (`F=B+1`)
`DECRB`	Décrémente l'entrée `B` de l'ALU (`F=B-1`)
`REPA`	Reporte l'entrée `A` de l'ALU sur sa sortie (`F=A`)
`REPB`	Reporte son entrée `B` de l'ALU sur sa sortie (`F=B`)

On considère des instructions de chargement sur 2 mots en mémoire. On se place dans la phase d'exécution de ces instructions :

- **5.a)** LD R1 V (V est dans IR)
- **5.b)** LD R1 (V) (V est dans IR)
- **5.c)** LD R1 ((V)) (V est dans IR)
- **5.d)** LD R1 (R1)
- **5.e)** LD R1 (R1+2) (2 est dans IR)

Donner la suite de µ-instructions correspondant à l'exécution de chacune de ces 5 instructions.