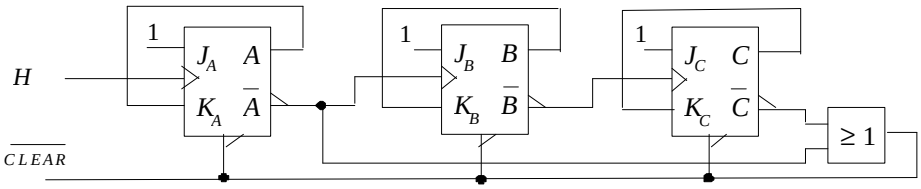
EXAMEN D'ELECTRONIQUE NUMERIQUE INTEGREE – Janvier 2012 *Note* ____

Durée 2h30 - Tous documents et calculatrice autorisés sauf Ordinateur - **Répondre exclusivement sur ces feuilles à rendre**
Toute réponse non commentée ne sera pas prise en compte

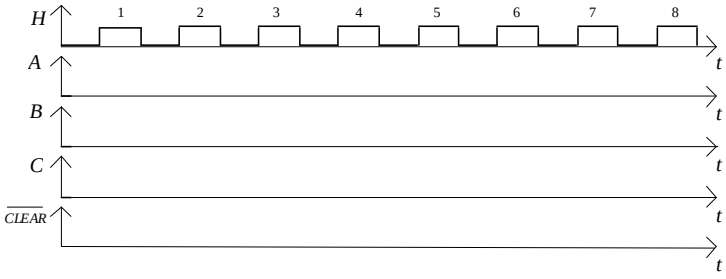
NOM - Prénom : ____ **Groupe :** ____

1. Compteur asynchrone avec Remise à Zéro (RAZ) (2.5 points)



1. Les bascules sont remises à zéro quand le signal $\overline{CLEAR}$ est à 0. Compléter le chronogramme avec l'état initial CBA = 000 en prenant en compte les temps de propagation.

Commentaires :



1

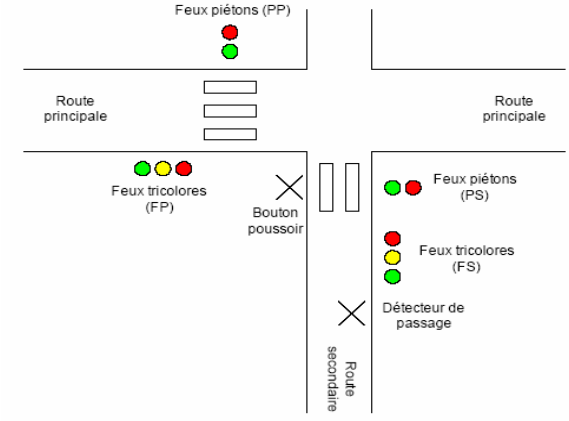
2. Si on lit les valeurs des états en fin de cycle (on laisse le temps au circuit de se stabiliser), compléter la séquence CBA des états du compteur (ne pas prendre en compte les brefs états transitoires) :

0.5

3. Le compteur est-il auto-correcteur ? **Auto-correcteur** *OUI / NON* 1

2. Analyse de système logique (2.5 points)

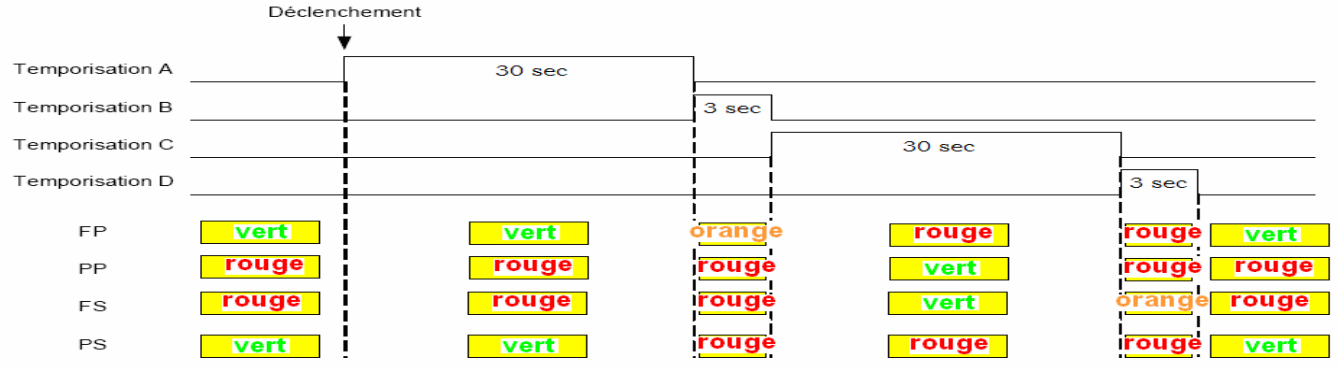
On considère un système de feux de signalisation lumineux à l'intersection de deux routes, l'une principale et l'autre secondaire :



Feux tricolores (pour véhicules) :
Vert : autorisation de passer
Orange : annonce le feu rouge
Rouge : interdiction de passer

Feux pour piétons (2 couleurs) :
Vert : autorisation de traverser
Rouge : interdiction de traverser

Par défaut, FP et PS sont au vert, FS et PP au rouge. Dès que le détecteur de passage (ou le bouton poussoir du passage piéton) devient actif, une série de 4 temporisations est déclenchée (niveau haut = 1, niveau bas = 0) :



1. Donner les équations logiques **les plus simples** des feux tricolores de la route secondaire et des feux piétons de la route principale, en fonction des variables *A, B, C et D.*

Commentaires :

FSvert = 0.5

FSrouge = 0.5

FSorange = 0.5

PPvert = 0.5

PProuge = 0.5

Note :

FPvert = 1 : feu tricolore principal vert allumé (0 éteint)
FPorange = 1 : feu tricolore principal orange allumé (0 éteint)
FProuge = 1 : feu tricolore principal rouge allumé (0 éteint)
PPvert = 1 : feu piéton vert principal allumé (0 éteint)
PProuge = 1 : feu piéton rouge principal allumé (0 éteint)

FSvert = 1 : feu tricolore secondaire vert allumé (0 éteint)
FSorange = 1 : feu tricolore secondaire orange allumé (0 éteint)
FSrouge = 1 : feu tricolore secondaire rouge allumé (0 éteint)
PSvert = 1 : feu piéton vert secondaire allumé (0 éteint)
PSrouge = 1 : feu piéton rouge secondaire allumé (0 éteint)

3. Synthèse de circuit logique (6.5 points)

1. Donner l'équation logique **la plus simple** du système logique qui reçoit en entrée *ABC* un pays appartenant au groupe de discussion et de partenariat économique *G8* (codé en binaire naturel BCD), et fournit en sortie *Z* un état logique haut (bit 1) si le pays n'appartient pas à l'Europe, un état bas (bit 0) sinon :

Commentaires

Pays	***ABC***	***Z***
0 Etats-Unis	000	1
1 Japon	001	1
2 Allemagne	010	0
3 France	011	0
4 Canada	100	1
5 Russie	101	1
6 Royaume-Uni	110	0
7 Italie	111	0

$Z =$ [ ] 1

2. Concevoir un circuit synchrone, de réalisation **la plus simple**, cadencé par une horloge *CLK* et constitué de 3 bascules *JK positive edge triggered*, de sorties Q_0, Q_1, Q_2 et d'entrées respectives J_0K_0, J_1K_1, et J_2K_2 permettant de produire la séquence d'entrée *ABC* suivante :

***ABC* = $Q_0Q_1Q_2$ = 000 → 001 → 011 → 111 → 110 → 100 → 000 → 001 …**

J_0	Q_1Q_2 = 00	01	11	10
Q_0 = 0				
Q_0 = 1				

} 0.5

J_1	Q_1Q_2 = 00	01	11	10
Q_0 = 0				
Q_0 = 1				

} 0.5

J_2	Q_1Q_2 = 00	01	11	10
Q_0 = 0				
Q_0 = 1				

} 0.5

K_0	Q_1Q_2 = 00	01	11	10
Q_0 = 0				
Q_0 = 1				

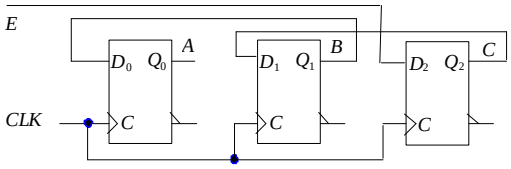
K_1	Q_1Q_2 = 00	01	11	10
Q_0 = 0				
Q_0 = 1				

K_2	Q_1Q_2 = 00	01	11	10
Q_0 = 0				
Q_0 = 1				

$J_0 =$ [ ] 0.5 $J_1 =$ [ ] 0.5 $J_2 =$ [ ] 0.5

$K_0 =$ [ ] 0.5 $K_1 =$ [ ] 0.5 $K_2 =$ [ ] 0.5

3. Pour engendrer cette séquence, une autre solution est envisagée, à l'aide de bascules D câblées en registre à décalage :



Indiquer la valeur la plus simple à donner au bit E pour engendrer la séquence désirée.

$E =$ [ ] 0.5 Le compteur est-il auto-correcteur ? [*OUI / NON*] 0.5

Commentaires

4. Représentation des nombres en machine : somme et produit en flottant (3 points)

1. Donner (*en Hexadécimal*) le codage IEEE754 simple précision des nombres flottants X = $10,5_{10}$ et Y = 13_{10}.

X (*HEXA*) 0.5 Y (*HEXA*) 0.5

Commentaires

2. Donner (*en Hexadécimal*) le codage IEEE754 simple précision de la somme S = X + Y en flottant. Lors du calcul, y-a-t-il normalisation ?

S = X + Y (*HEXA*) 0.5 Normalisation : *OUI / NON* 0.5

Commentaires

3. Donner (*en Hexadécimal*) le codage IEEE754 simple précision de la multiplication M = XY en flottant. Lors du calcul, y-a-t-il normalisation ?

M = XY (*HEXA*) 0.5 Normalisation : *OUI / NON* 0.5

Commentaires

5. Pipelining (1 point)

On considère le pipeline ci-dessous dédié aux instructions en virgule flottante (pour données FP *Floating Point*) :

Etage	*pipeline P*
1	**lit cache d'instructions**
2	**decode instruction**
3	**lit registres FP**
4	**execute FP / acces cache de donnees**
5	**execute FP / acces cache de donnees**
6	**execute FP / acces cache de donnees**
7	**ecrit registre FP**

Le débit théorique maximum de ce pipeline est de 1 instruction/cycle *(1 cycle représente le temps nécessaire pour exécuter 1 étage du pipeline).* On suppose un prédicteur de branchements parfait, les instructions déjà dans les caches et le pipeline déjà rempli à l'instant où les extraits de programme considérés sont exécutés.

Les instructions en virgule flottante (FP) sont pipelinées sur 3 cycles :
2 bulles sont insérées si une instruction utilise en opérande le résultat de l'instruction immédiatement avant.

On considère un extrait de programme destiné à calculer le discriminant d'une équation du 2nd degré : $b^2 - 4ac$.
Les registres sont ainsi initialisés : F0 = *b* F1 = -4.0 F2 = *a* F3 = *c*

Méthode 1 :

```
1: F5 = F1 FMUL F2     // F5 = -4a
2: F4 = F0 FMUL F0     // F4 = b^2
3: F5 = F5 FMUL F3     // F5 = -4ac
4: F5 = F4 FADD F5     // F5 = b^2 - 4ac
```

Méthode 2 :

```
1: F4 = F0 FMUL F0     // F4 = b^2
2: F5 = F1 FMUL F2     // F5 = -4a
3: F5 = F5 FMUL F3     // F5 = -4ac
4: F5 = F4 FADD F5     // F5 = b^2 - 4ac
```

1. Indiquer le coût temps calcul, exprimé *en nombre de cycles*, du programme de la méthode 1 :

[ ] *cycles* 0.5

Commentaires

2. Indiquer le coût temps calcul, exprimé *en nombre de cycles*, du programme de la méthode 2 :

[ ] *cycles* 0.5

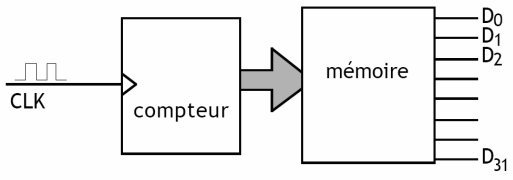
Commentaires

6. Mémoires programmables (2.5 points)

Les mémoires mortes sont couramment utilisées pour générer des signaux. Le circuit suivant représente une mémoire dont les entrées d'adresse sont reliées à un compteur binaire cadencé par une horloge CLK.

On veut que ce circuit génère, via les données (bits D_0 à D_{31}), le code binaire IEEE754 simple précision associé à une séquence croissante de nombres : $\sqrt{1}$, $\sqrt{2}$, $\sqrt{3}$ ... $\sqrt{255}$, $\sqrt{256}$, $\sqrt{1}$, $\sqrt{2}$, $\sqrt{3}$ ... etc…

Le modulo du compteur est tel que toutes les adresses de la mémoire sont parcourues.



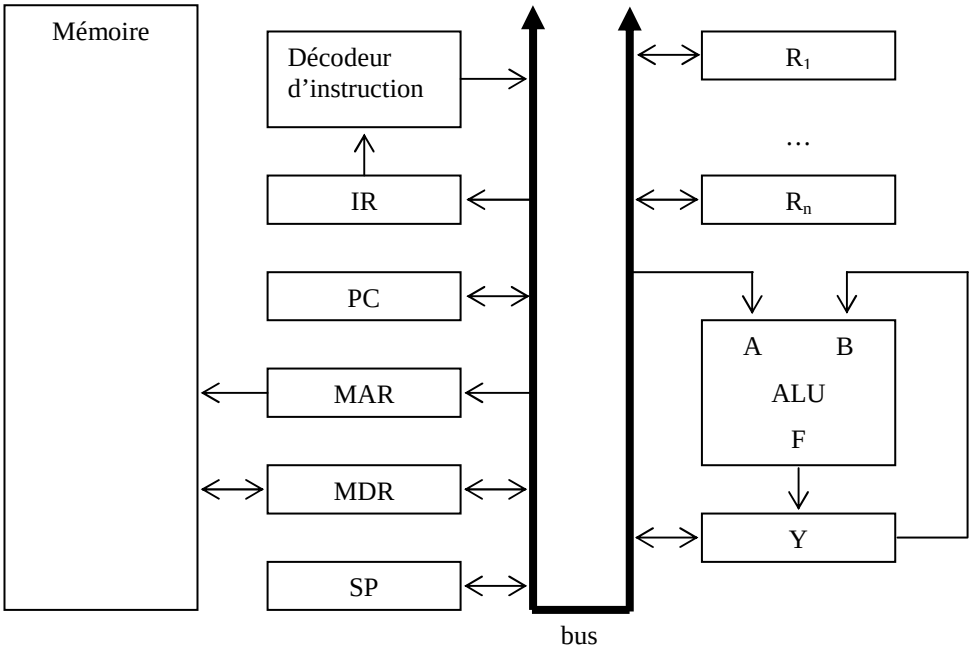
1. Indiquer le nombre de bits d'adresse [ ] 0.5 ainsi que la capacité (en *octets*) de la mémoire [ ] 0.5

2. Quel doit être le modulo du compteur ? [ ] 0.5

3. Indiquer la donnée (exprimée en *Hexadécimal*) contenue à la dernière adresse de la mémoire [ ] 1

7. Microprogrammation (2 points)

On considère une organisation de CPU à 1 bus de données/adresses représentée par la figure suivante :



Registres et ALU :

- R1 à Rn : registres de données
- IR : registre d'instruction
- MDR : registre de donnée d'échange avec la mémoire
- MAR : registre d'adresse d'échange avec la mémoire
- PC : compteur ordinal
- SP : pointeur de pile
- Y : registre accumulateur de l'ALU à la fois pour l'opérande B et le résultat F
- ALU : Unité Arithmétique et Logique

Micro-instructions (étapes possibles du traitement d'une instruction machine) :

- 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
- Ecriture : effectue une écriture dans la mémoire à l'adresse MAR de la valeur contenue dans MDR
- 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 l'entrée B de l'ALU sur sa sortie (F=B)

Pour l'ALU, l'entrée A est lue sur le bus, l'entrée B sur la sortie du registre Y (aucune opération à effectuer par rapport au registre Y) ; la sortie F est inscrite écrite dans le registre Y quand une micro-instruction concernant l'ALU est déclenchée.

On considère une machine de Von Neumann manipulant des **données sur 8 bits et des adresses sur 8 bits**. On s'intéresse à l'exécution de l'instruction suivante :

LD R_1 (R_2+*d*)	Chargement (*load*) de R_1 avec le mot en mémoire à l'adresse R_2+*d*.

L'instruction est sur 3 mots de 8 bits : 0x **DD 7E XX**. Les deux premiers mots (0xDD7E) représentent le code de l'opération et les noms des deux registres utilisés ; le troisième mot (0xXX) code le déplacement relatif *d* en complément à 2 sur 8 bits (X chiffre hexadécimal quelconque). Le déplacement *d* est ajouté au contenu du registre R_2 pour obtenir l'adresse de la donnée à lire. Le mot lu de la mémoire est alors rangé dans le registre R_1. Le compteur ordinal est systématiquement **incrémenté d'une unité** après chaque lecture de mot (8 bits) de l'instruction en mémoire.

(Rappel : le préfixe 0x indique une valeur exprimée en *Hexadécimal*)

Mémoire

Adresse(@)	*Donnée*
0x00	
…	…
0x06	0xDD
0x07	0x7E
0x08	0xFD
…	…
0x6C	0x27
0x6D	0xBD
0x6E	0x11
0x6F	0xBA
0x70	0xAB
0x71	0x44
0x72	0x1F
…	…
0x7E	0xD0
…	…
0xFD	0xC6
…	…

On s'intéresse à l'évolution des données dans cette machine lors du cycle de Chargement-Décodage-Exécution (*Fetch-Decode-Execute*) de cette instruction.

On prend le programme à l'adresse 0x06 avec l'instruction *load* 0xDD 7E FD sur 3 mots.

On donne l'état de la machine suite aux 2 premières phases de traitement (*Fetch+Decode*). Les deux premiers mots de l'instruction sont lus et décodés. La machine est prête à lire le 3ème mot toujours en mémoire.

Les opérations élémentaires à réaliser (dans cet ordre) sont donc les suivantes :

1. Aller chercher en mémoire le prochain mot (le 3ème octet) de l'instruction à exécuter.
2. Incrémenter le compteur ordinal.
3. Ajouter le déplacement *d* au registre R_2.
4. Aller chercher en mémoire la donnée à l'adresse R_2+*d*.
5. Ranger la donnée lue dans le registre R_1.

1. Donner la valeur du déplacement relatif *d* (exprimée en valeur algébrique décimale) : _____ 0.5

2. Compléter le tableau suivant pour la phase d'exécution (*Execute*) du *load* en ne remplissant que les cases dont **la valeur change**. Chaque case représente le contenu d'un registre (ou du bus) après exécution de la micro-instruction correspondante : 1

	Micro-instruction	**R1**	**R2**	**IR**	**MDR**	**MAR**	**PC**	**Y**	**SP**	**Bus**
	État initial	indéfini	0x6F	0x7E	0x7E	0x07	0x08	0x08	indéfini	0x7E
1										
2										
3										
4										
5										
6										
7										
8										
9										
10										
11										
12										
13										
14										
15										
16										
17										
18										
19										
20										

Execute

3. Quelle donnée (exprimée en *Hexadécimal*) a été écrite dans le registre R_1 : _____ 0.5