SYSTEMES NUMERIQUES
& PROCESSEURS EMBARQUES
LU3EE100 — TD N°1

L'objectif de cette séance est de prendre en main l'outil de simulation EDA Playground et d'appréhender les différents niveaux de description du langage VHDL en travaillant sur des architectures combinatoires

Préparation :

- Lire sur Moodle le guide d'utilisation de l'outil EDA Playground, ou bien regarder la vidéo tutoriel.
- Créer votre compte EDA Playground avant le début du TD
- Donner les équations d'un additionneur complet (Full Adder) 1 bit avec trois entrées : A, B et Cin (retenue entrante) et deux sorties :S et Cout (retenue sortante)

Exercice 1 : Additionneur 4 bits

On souhaite décrire en VHDL un additionneur 4 bits de deux manières différentes

- D'une manière structurelle, en s'appuyant sur la description d'un additionneur 1 bit
- D'une manière comportementale

1. Aller au lien suivant: https://www.edaplayground.com/x/4Zb4 et recopier le projet sur votre compte EDA Playground
    - Décrire en VHDL flot de données un additionneur complet 1 bit.
        - On respectera bien les recommandations d'écriture indiquées au début du fichier design.vhd
    - Compiler puis lancer la simulation. A l'aide du chronogramme, vérifier le bon fonctionnement du modèle.

2. Aller au lien suivant: https://www.edaplayground.com/x/bcU et recopier le projet sur votre compte EDA Playground
   Ce projet comporte deux fichiers
   - FA.vhd : Décrit l'additionneur complet 1 bit de la question précédente
   - Design.vhd : Décrit un additionneur 4 bits par instanciation de 4 cellules 1 bit
   - Analyser le code de Design.vhd pour comprendre comment est décrite l'architecture. Vous pourrez simuler ce projet par vous-mêmes après le TD.

3. Aller au lien suivant: https://www.edaplayground.com/x/4cz3 et recopier le projet sur votre compte EDA Playground
   - Décrire en VHDL comportemental un additionneur 4 bits.
     - On respectera bien les recommandations d'écriture indiquées au début du fichier design.vhd
   - Compiler puis lancer la simulation. A l'aide du chronogramme, vérifier le bon fonctionnement du modèle.

Exercice 2 : Additionneur Binaire Codé Décimal

1. Donner l'architecture d'un additionneur BCD prenant en entrée deux chiffres BCD, chacun codés sur 4 bits.

2. Aller au lien suivant: https://www.edaplayground.com/x/bw2 et recopier le projet sur votre compte EDA Playground
   - Décrire l'additionneur BCD en VHDL comportemental.
     - Respecter les règles d'écriture précisées en tête du fichier design.vhd
     - On forcera pour le moment la sortie Error à 0.

3. Ajouter au module la description de la sortie 'Error' qui se met à '1' lorsque l'une des entrées est à une valeur incompatible avec le codage BCD.

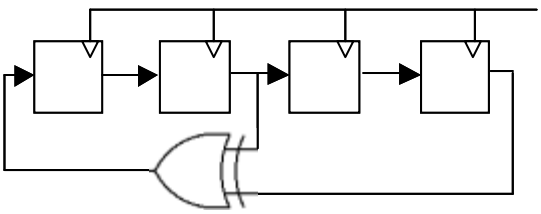
SORBONNE UNIVERSITÉ
CRÉATEURS DE FUTURS DEPUIS 1257

SYSTEMES NUMERIQUES & PROCESSEURS EMBARQUES LU3EE100 — TD N°2

L'objectif de cette séance est de décrire des architectures séquentielles à l'aide de process VHDL

Préparation : Etude d'un LFSR (Linear Feedback Shift Register)

Un LFSR est un registre à décalage dont l'entrée est une fonction linéaire de sa valeur précédente. Typiquement, cette fonction est un XOR logique. Un LFSR peut donc être représenté par le schéma suivant



Le LFSR permet ainsi de générer une séquence pseudo-aléatoire et périodique de valeurs (dans ce cas, des vecteurs de 4 bits).

1. On considère le LFSR du schéma ci-dessus. On initialise les bascules de son registre à la valeur 0100. Donner les valeurs successives du registre. Quelle est la période de la séquence générée ?

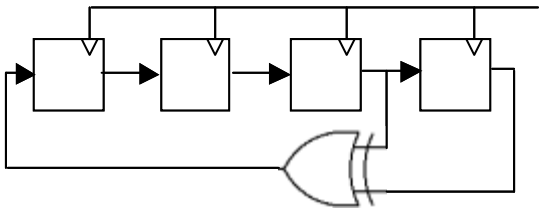
2. Par quelle valeur ne doit-on surtout pas initialiser les bascules du LFSR ?

3. En imaginant une fonction de rebouclage qui permettrait d'augmenter la période de la séquence du LFSR, préciser quelle est la période maximale atteignable par un LFSR de taille 4 bits.

Exercice 1 : Etude d'un LFSR (Linear Feedback Shift Register)

Il est possible de trouver des structures de LFSR maximisant la période des séquences.

1. Aller au lien suivant: https://www.edaplayground.com/x/4NGS et recopier le projet sur votre compte EDA Playground
    - Décrire en VHDL le modèle de LFSR ci-dessous. On pensera à ajouter un Reset asynchrone qui initialisera les bascules à une valeur appropriée.



    - Vérifier par simulation que le LFSR décrit ci-dessus propose bien une séquence de période maximale, du moment que la valeur d'initialisation ne soit pas celle trouvée lors de la question 2 de la préparation.

Exercice 2 : Registre universel

1. Donner l'architecture d'un registre de 8 bits pouvant effectuer soit un chargement parallèle, soit un maintien de sa valeur mémorisée

2. Modifier cette architecture pour intégrer une fonction remise à zéro synchrone et inversion

3. Aller au lien suivant: https://www.edaplayground.com/x/3fFj et recopier le projet sur votre compte EDA Playground
    - Modéliser en VHDL le registre de la question 2, en respectant les règles d'écriture indiquées dans Design.vhd.

Exercice 3 : Compteur

1. Donner l'architecture d'un compteur 8 bits pouvant effectuer les fonctions suivantes :
    - Comptage
    - Décomptage
    - Maintien de la valeur
    - Initialisation à une valeur fournie en entrée

   Il possède de plus une sortie Max qui passe à 1 quand le compteur atteint sa valeur maximale.

2. Aller au lien suivant: https://www.edaplayground.com/x/4DLf et recopier le projet sur votre compte EDA Playground
    - Modéliser en VHDL le compteur en respectant les règles d'écriture indiquées dans Design.vhd

SYSTEMES NUMERIQUES
& PROCESSEURS EMBARQUES
LU3EE100 — TD N°3

L'objectif principal de cette séance est d'écrire et d'utiliser des fichiers de simulation VHDL (testbenchs) en réutilisant les programmes des précédents TD

Exercice 1 : Ecriture de Testbenchs

On souhaite écrire un fichier testbench afin de simuler l'additionneur BCD décrit lors du TD1. Le code VHDL de cet additionneur est disponible sur le projet EDA Playground suivant : Aller au lien suivant: https://www.edaplayground.com/x/4R_x
Recopier le projet sur votre compte EDA Playground. Vous constaterez que le code du testbench est absent. Nous allons donc devoir lécrire.

1. Ecrire une première version du testbench qui va permettre de tester quelques exemples de calculs. Choisir un exemple où la somme est inférieure à 10, un exemple où la somme est supérieure à 10, et un exemple où la sortie Error de l'additionneur va être activée.

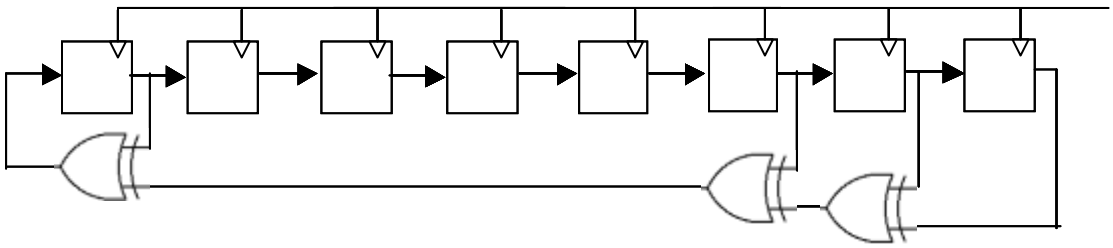
2. Faire une copie de ce projet (appelez la TD3 Projet 2 par exemple) Dans ce nouveau projet, modifier le testbench afin de générer toutes les combinaisons d'entrées possibles et ainsi effectuer tous les calculs possibles pour cette architecture.
   - NB1 : il n'est pas question ici d'écrire "à la main" toutes les valeurs des entrées…
   - NB2 : Pensez à ajuster la durée de la simulation afin de pouvoir visualiser toutes les combinaisons d'entrées.

3. Faire une copie de ce projet (appelez la TD3 Projet 3 par exemple) Dans ce nouveau projet, modifier à nouveau le testbench, afin cette fois-ci de ne générer que les combinaisons en entrée qui ne provoqueront pas la mise à '1' de la sortie 'Error' de l'additionneur.

Exercice 2 : Utilisation d'un LFSR pour le test de l'additionneur BCD (A FAIRE CHEZ VOUS)

L'une des applications d'un LFSR est de permettre « l'auto-test » des circuits (Built-In Self Test). Les séquences aléatoires générées par le LFSR sont placées en entrée du module à tester. Chaque sortie du module est ensuite vérifiée à l'aide d'un module d'analyse de signature. Nous allons adapter ce principe à l'additionneur BCD du TD1, uniquement pour la partie génération de vecteurs de test..

Le LFSR nécessaire pour le test de l'additionneur est de la forme suivante.



1. Aller au lien suivant : https://www.edaplayground.com/x/fQd et recopier le projet sur votre compte EDA Playground. Ce projet contient un fichier BCD.vhd décrivant l'additionneur BCD et un fichier design.vhd pour le moment incomplet, correspondant au top-level de l'architecture.
2. Ajouter dans la partie VHDL Design du projet un fichier LFSR.vhd décrivant le LFSR ci-dessus.
3. Compléter le fichier design.vhd pour y instancie le LFSR et l'additionneur BCD en connectant la sortie du premier module aux entrées du second.
4. Ecrire le testbench (en s'inspirant des éléments vus en cours pour décrire le comportement du signal d'horloge). Lancer la simulation et vérifier la génération pseudo-aléatoire de vecteurs d'entrées pour le test de l'additionneur BCD.

SYSTEMES NUMERIQUES
& PROCESSEURS EMBARQUES
LU3EE100 — TD N°4

L'objectif de cette séance est de comprendre, au travers d'exemples fournis, les conséquences de mauvaises descriptions VHDL sur les architectures générées, et leur simulation. Les programmes étudiés au cours du TD sont disponibles sur le site Moodle de l'UE, section TD.

Exercice 1 : Unité Arithmétique et Logique du Processeur LC-3

Aller au lien suivant : https://www.edaplayground.com/x/wnq et recopier le projet sur votre compte EDA Playground

Le programme ***design.vhd*** décrit l'architecture de l'unité arithmétique et logique (ALU) d'un processeur, c'est-à-dire l'organe en charge des calculs du processeur. Cette structure purement combinatoire peut effectuer les opérations suivantes : addition, ET logique, inversion de l'entrée A, et transfert en sortie de l'entrée A. L'ALU fournit également en sortie des indicateurs sur la nature du dernier résultat calculé : résultat négatif (N), égal à zéro (Z) ou positif (P).
Le testbench permet de simuler le comprtement du modèle de l'ALU. On précise que ce testbencj ne comporte pas d'erreurs.

1. Compiler le projet. Expliquer l'erreur indiquée par le compilateur et corriger le programme.

2. Recompiler le projet. A l'ouverture du chronogramme, afficher sur le chronogramme les signaux suivants dans cet ordre : ***inA, inB, opcode, result, s, nzp***. Au vu du chronogramme de simulation, quel(s) problème(s) pouvez-vous détecter ?

3. Y a-t-il dans le code de ***design.vhd*** des raisons pour expliquer ces erreurs de simulation. Quelles seraient les conséquences sur l'architecture matérielles générées ?

4. Corriger ces erreurs et refaire la simulation pour vérifier le bon fonctionnement de l'ALU.

Exercice 2 : Compteur d'impulsions

Aller au lien suivant : https://www.edaplayground.com/x/2anm et recopier le projet sur votre compte EDA Playground Le projet comporte plusieurs descriptions d'un compteur d'impulsions que nous allons examiner au fur et à mesure.

Le programme ***design.vhd*** décrit une 1ère version du système. On compte le nombre d'impulsions d'un signal d'entrée. Le compteur est incrémenté à chaque transition montante de top et est affiché en sortie.

1. Etudier le programme ***design.vhd*** pour comprendre le fonctionnement du système. Avec le testbench, simuler ce programme. Afficher dans cet ordre les signaux ***clk, top, top_last, test, increm, cpt, count***. Sachant que la simulation est censée durer plusieurs centaines de nanosecondes, que constatez-vous ?

2. Pour trouver la cause de cette erreur, regarder à quel moment précis s'arrête la simulation. Quelles sont les instructions concernées par cet événement ? Laquelle a provoqué l'erreur ?

3. Le programme ***version2.vhd*** tente de corriger le problème. Regarder le code, et modifier le testbench pour instancier l'entité correspondant à la version 2 du compteur d'impulsions. Simuler le programme et afficher les mêmes signaux dans le même ordre. Que constatez-vous ? Pourquoi ?

4. Le programme ***Version3.vhd*** résout le problème d'initialisation du compteur en ajoutant un reset asynchrone. Regarder le code, et modifier le testbench pour instancier l'entité correspondant à la version 3 du compteur d'impulsions, et mettre à jour la liste des entrées/sorties dans le port map. Simuler le programme. Que pouvez-vous dire du reset ? Modifier le code pour répondre au cahier des charges.

5. Toujours dans ***Version3.vjd***, déplacer l'instruction ***count <= cpt*** pour la placer à l'endroit indiqué par la marque - - ***. Simuler. Que constatez-vous ? Dessinez l'architecture matérielle correspondant à ce code VHDL (en fonction de la position de l'instruction ***count <= cpt*** dans le programme)

SORBONNE UNIVERSITÉ
CRÉATEURS DE FUTURS DEPUIS 1257

SYSTEMES NUMERIQUES & PROCESSEURS EMBARQUES LU3EE100 — TD N°5

L'objectif de cette séance est de comprendre la fonctionnalité d'un système à partir de son cahier des charges (description et/ou chronogramme), de reconstruire le chronogramme le cas échéant, puis d'en déduire sa machine à états.

Exercice 1 : Emission Série Asynchrone

On se propose de synthétiser un circuit d'émission de données en mode asynchrone type *start-stop*. Chaque message est défini par un start bit, 8 bits de données et un stop bit.
Le schéma de principe est donné par la figure 1.

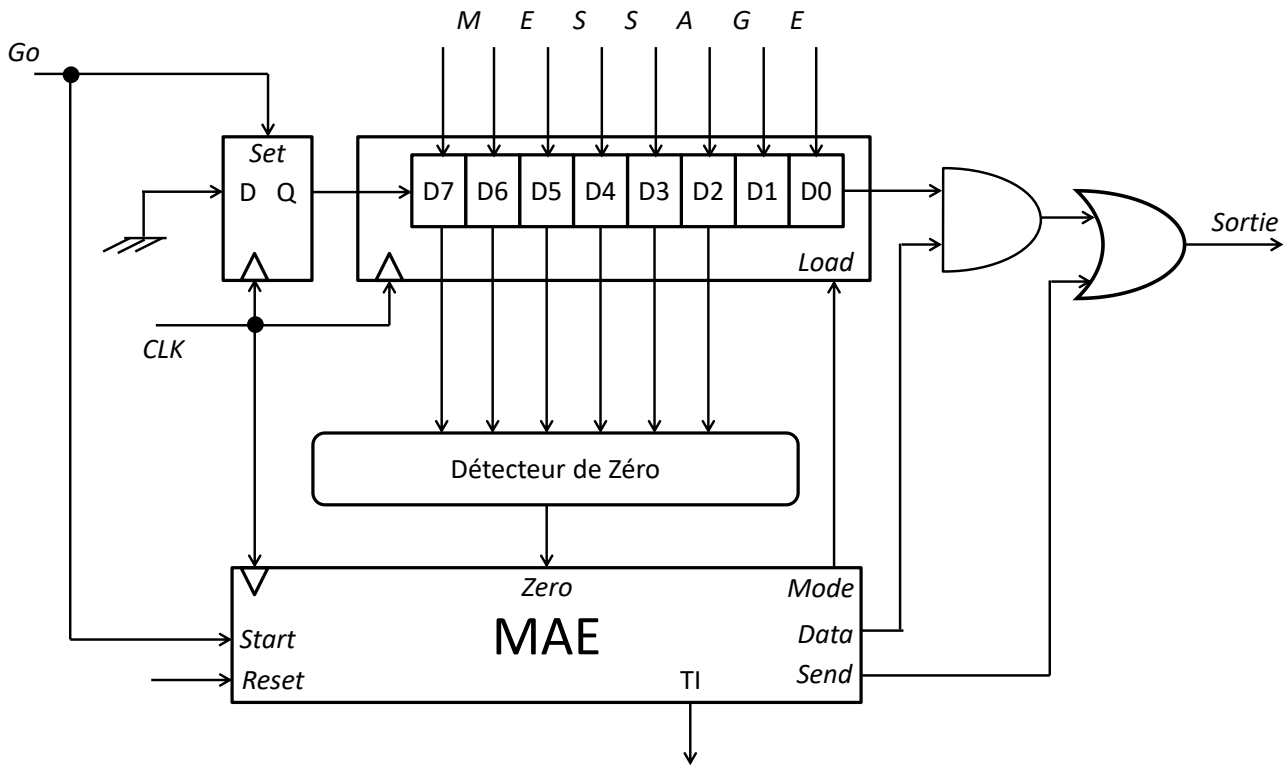


Figure 1: Système d'émission série

Les caractéristiques du circuit d'émission sont les suivantes :

- Les entrées d'horloges de tous les circuits sont actives sur un front descendant

- Le registre à décalage 8 bits charge en parallèle la donnée à émettre $D_7...D_0$. Il possède également une entrée de chargement série, connectée à la bascule D_7.

- Le registre possède un signal de contrôle LOAD qui fixe son mode de fonctionnement :
  - LOAD = 0 → décalage à droite
  - LOAD = 1 → chargement parallèle

- L'entrée Set de la bascule D est asynchrone, tout comme le Reset de la Machine à Etats (MAE).

- La commande Go est une impulsion asynchrone d'une durée équivalente à deux périodes d'horloge. La commande déclenche la transmission série de l'octet $D_7...D_0$

- La sortie Zéro du détecteur de zéro passe à '1' quand toutes ses entrées sont à '0'.

- Les commandes d'entrée de la MAE sont Start et Zero.

- Les sorties sont Mode, Send, Data et Ti (voir Table 1 pour leur représentation)

- La commande Send indique à quel moment le message est transmis.

Les chronogrammes de ces différents signaux, correspondant à l'émission d'un octet, sont tracés sur la figure 2.

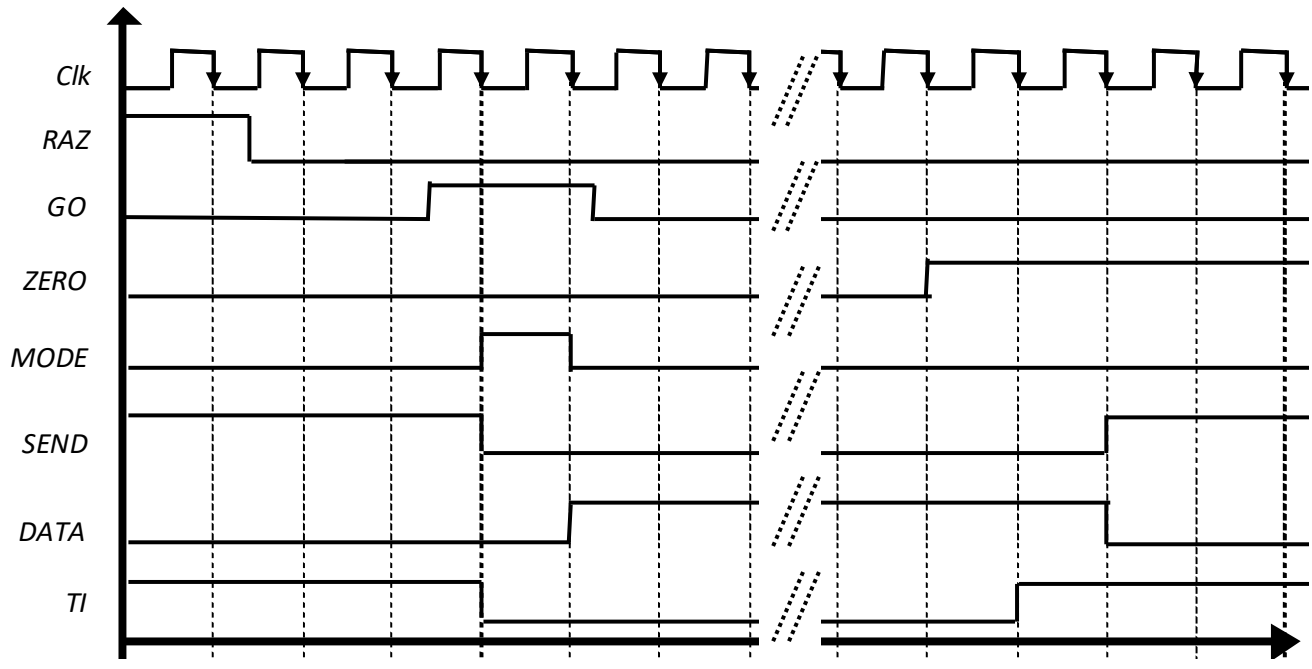


Figure 2: Chronogramme de fonctionnement

Vecteur de Sortie				
Mode	Send	Data	Ti	S
0	0	0	0	S0
0	0	0	1	S1
0	0	1	0	S2
0	0	1	1	S3
0	1	0	0	S4
0	1	0	1	S5
0	1	1	0	S6
0	1	1	1	S7
1	0	0	0	S8
1	0	0	1	S9
1	0	1	0	S10
1	0	1	1	S11
1	1	0	0	S12
1	1	0	1	S13
1	1	1	0	S14
1	1	1	1	S15

Table 1: Vecteur de sortie

Questions

1) Expliquer comment la sortie de la bascule D va passer au niveau haut, puis comment elle va passer au niveau bas.

2) Proposer un schéma simple du détecteur de zéro

3) A l'aide de l'annexe, tracer les chronogrammes des signaux Mode, Send, Data, TI, Sortie et Zero. On suppose qu'à t=0, toutes les bascules sont initialisées à 0. Le message à transmettre est A3 (en hexadécimal)

4) Expliquer comment sont obtenus le Start Bit et le Stop bit

5) Etablir le graphe d'état de la machine à état en mode Moore, les vecteurs de sorties seront ceux indiqués dans la Table 1. En déduire les équations des états futurs et des sorties de cette machine à états.

Exercice 2 : Synthèse des signaux d'horloge d'une barette CCD

Un CCD (Charge-Coupled Device) est un type de capteur d'image. Il repose sur une matrice de pixels réalisant l'acquisition de l'image (conversion photons → charge électrique) puis un transfert en série de chaque pixel de la matrice vers la sortie du capteur.

La lecture des 1024 pixels d'un capteur CCD nécessite l'application de signaux d'horloge particuliers, ces signaux se nomment FT et FP. On se propose de synthétiser ces deux signaux à l'aide d'une machine à états (MAE). Le montage utilisé est celui de la figure 3, les chronogrammes de ces signaux sont ceux de la figure 4.

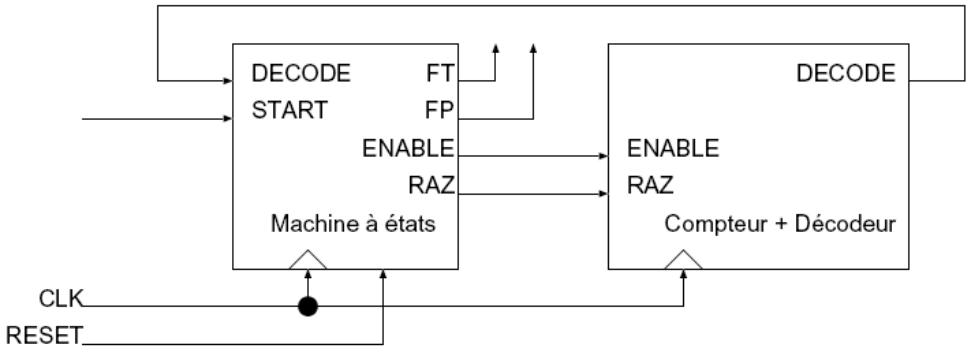


Figure 3: Circuit de lecture de la barette CCD

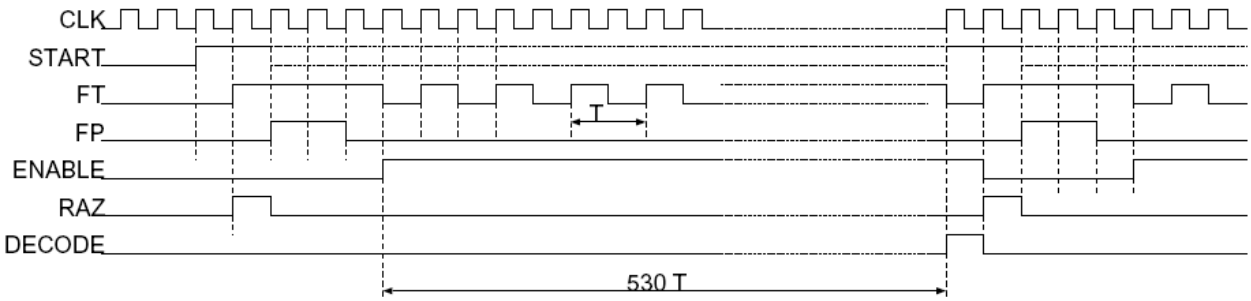


Figure 4: Chronogramme des signaux du circuit de lecture

Après initialisation de la machine à états, la détection du front de montée du signal START provoque la génération des signaux FP, FT, ENABLE et RAZ.

Lorsque le signal DECODE se produit, l'entrée START est testée :

- Si START = 0, toute les lignes de sortie restent à zéro.
  La machine se met en attente d'un front de montée du signal START
- Si START = 1, on génère une nouvelle trame des signaux FP, FT, ENABLE et RAZ

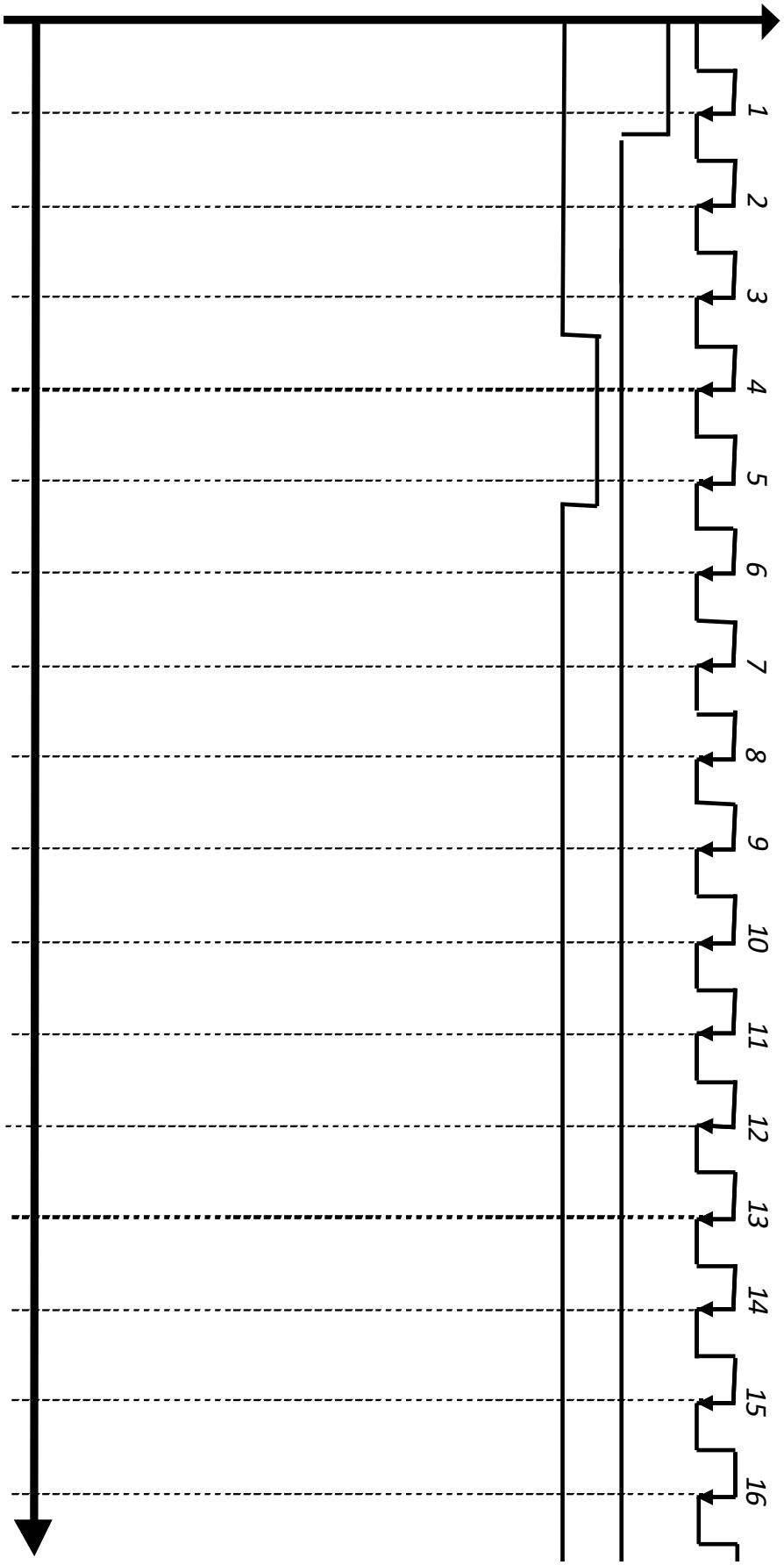
1) Expliquer qualitativement le fonctionnement du système "MAE + compteur/décodeur"

2) Préciser le nombre de bascules du compteur et proposer un schéma du décodeur

3) Etablir, en représentation de Moore, le graphe d'états de la machine à états.
   On indiquera sur le graphe les valeurs des signaux START et DECODE.
   On regroupera les 4 signaux de sortie en un vecteur S avec : S=[RAZ, ENABLE, FT, FP]

   Méthode : Identifier avec le chronogramme et la question 1) les différents états en partant du début et construire le graphe d'états au fur et à mesure.

ANNEXE


Clk
RAZ
GO
ZERO
MODE
SEND
DATA
TI
SORTIE
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16

SYSTEMES NUMERIQUES
& PROCESSEURS EMBARQUES
LU3EE100 — TD N°6

L'objectif de ce TD est de construire un système, à partir de son cahier des charges et de son schéma-bloc partie opérative/partie contrôle. On détaillera l'architecture des blocs de la partie opérative, et on explicitera le graphe d'état de la partie contrôle.

Exercice 1 : GUITAR HERO (Adapté d'un sujet d'examen)

GUITAR HERO
LICENCE ELEC EDITION

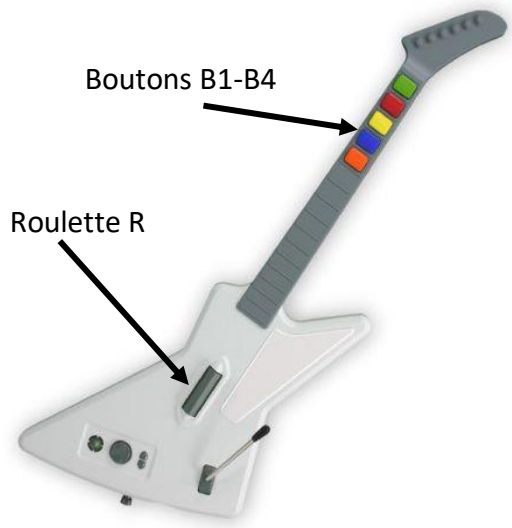
Le but de l'exercice est de concevoir à l'aide d'une machine à états le système de contrôle permettant de jouer à ce célèbre jeu vidéo.

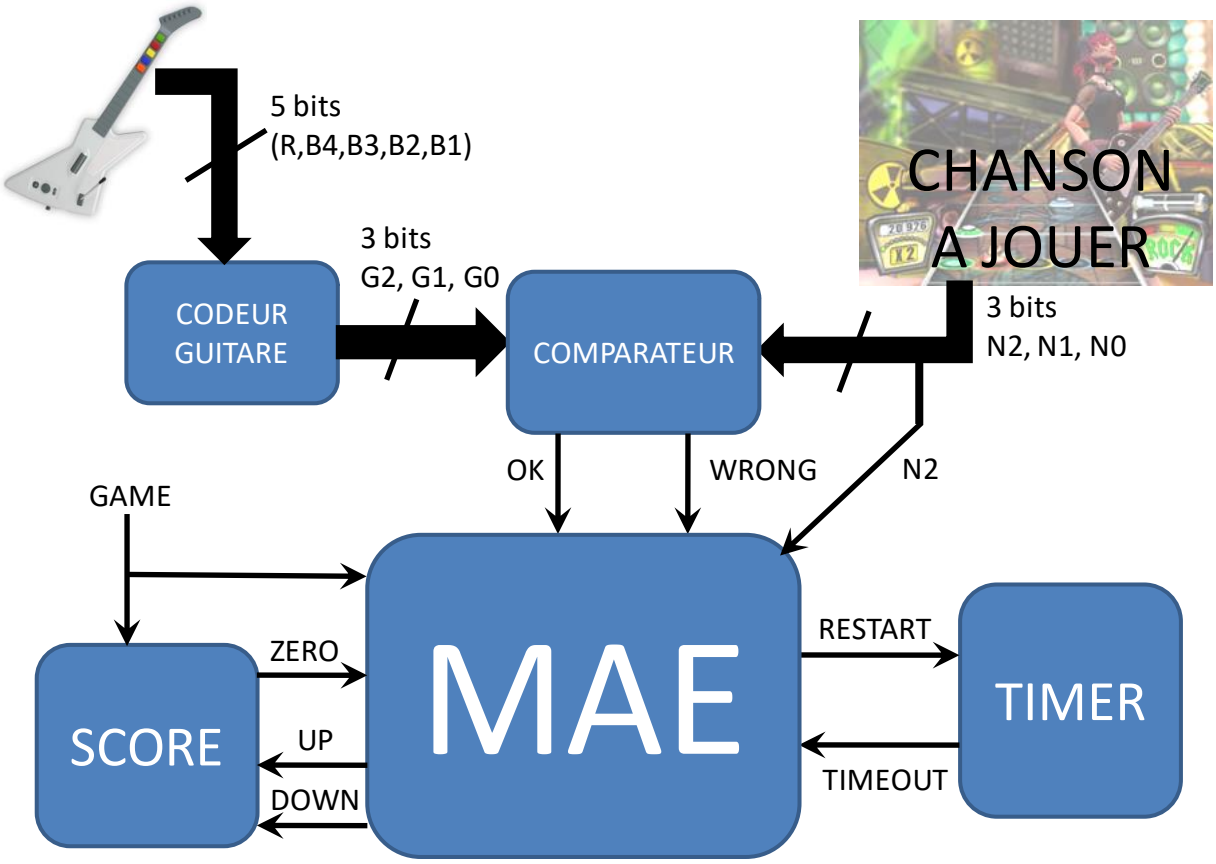
Pour mémoire, on rappelle que Guitar Hero consiste à « jouer » à l'aide de la guitare ci-dessous les plus grands standards du rock et,en fonction de votre score, devenir un dieu vivant de la musique, ou bien vous faire sortir de scène sous les huées du public…

La guitare du jeu (dans sa version Licence Elec) possède 4 boutons (B1, B2, B3 et B4) situés sur le manche et une roulette (R) au niveau de la caisse. Pour produire une note, il faut simultanément actionner la roulette (R=1) et appuyer sur l'un des boutons (Bi=1). On considère que l'action de la roulette dans appuyer sur un bouton produira une fausse note.

Le synoptique du système est présenté page suivante. Une partie commence lorsque le signal ***GAME*** est mis à 1.

Au cours de la partie, les commandes issues de la guitare (boutons + roulette) sont encodées pour produire un signal sur 3 bits. Ce code ***G2, G1, G0*** est comparé au signal ***N2, N1, N0*** généré par la chanson à jouer. La table des codes ***N*** et ***G*** est également donnée ci-dessous.





R	B4	B3	B2	B1	Code G2,G1,G0
0	X	X	X	X	0 0 0
1	0	0	0	0	0 0 1
1	X	X	X	1	1 0 0
1	X	X	1	0	1 0 1
1	X	1	0	0	1 1 0
1	1	0	0	0	1 1 1

N2	N1	N0	Note correspondante
0	X	X	Pas de note
1	0	0	Note R1+B1
1	0	1	Note R1+B2
1	1	0	Note R1+B3
1	1	1	Note R1+B4

Le bit ***N2*** est également envoyé à la ***MACHINE A ETATS (MAE)*** pour signifier qu'une note doit être jouée rapidement sous peine de faire baisser le score du joueur.

Le joueur dispose alors d'un certain temps pour jouer chaque note de la chanson. Ce temps est déterminé par un ***TIMER***, qui est initialisé par un signal ***RESTART*** et qui indique que le temps est écoulé grâce au signal ***TIMEOUT***.

Si pendant cet intervalle de temps les codes ***N*** et ***G*** indiquent la même note, le signal ***OK*** passe à 1. Si la note est différente, c'est le signal ***WRONG*** qui passe à 1.

Le module ***SCORE*** comptabilise les points du joueur. Au début de chaque partie, le score est initialisée à une valeur fixe supérieure à 0. Au cours du jeu, à chaque note correctement jouée, le signal ***UP*** est positionné à 1 pendant 1 cycle d'horloge pour incrémenter le score. En cas de fausse note ou si le joueur n'a pas joué la note à temps, la signal ***DOWN*** passe à 1 pendant un cycle d'horloge, décrémentant le score. Si le joueur a épuisé tous ses points, le signal ***ZERO*** est mis à 1 provoquant la fin du jeu.

Le jeu continue tant que le score du joueur est supérieur à 0. Dans la grande tradition du rock'n'roll, *"Play till you bleed"*

Questions

1) ***Identifier sur le schéma d'architecture les modules appartenant à la partie opérative et à la partie contrôle du système.***

2) ***Donner, sans passer par des tableaux de Karnaugh, les équations de sortie du module COMPARATEUR (OK, WRONG)***

3) ***Donner le schéma bloc (c'est-à-dire sans détailler toutes les portes logiques) du module SCORE en vous aidant de registres, multiplexeurs ou comparateurs.***

4) ***Donner le graphe d'états de la machine à états. On précisera notamment les valeurs de chaque sortie.***

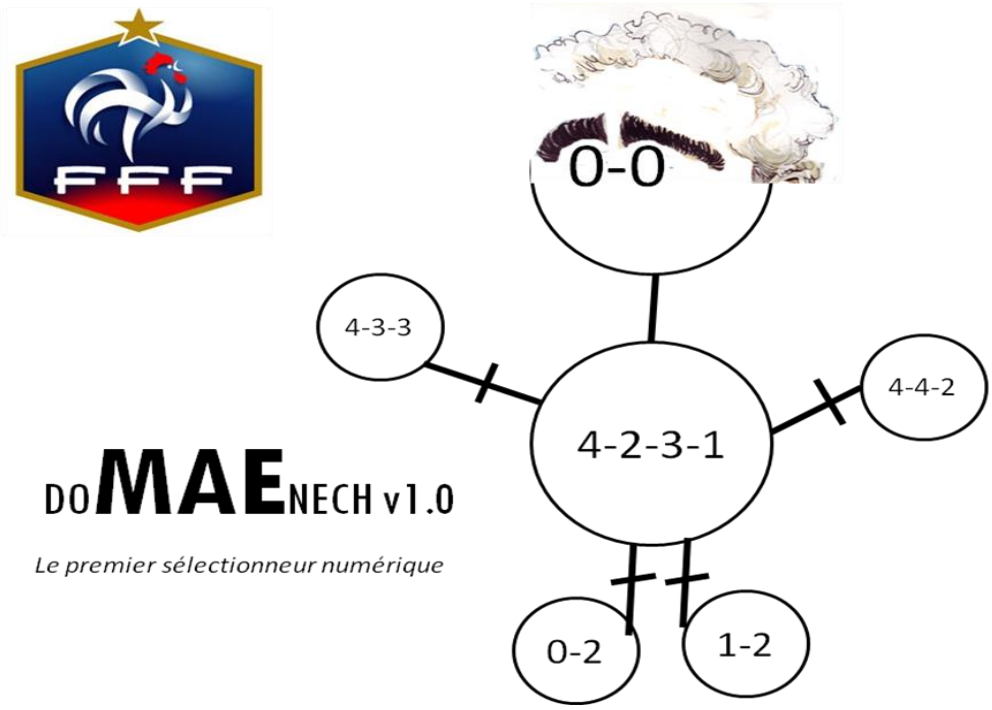
5) ***Proposer pour votre graphe d'états un encodage dans les modes binaire aléatoire, one-hot et total synchrone. En déduire pour chaque mode la taille du registre d'état.***

La guitare n'est pas fournie avec le sujet de TD…

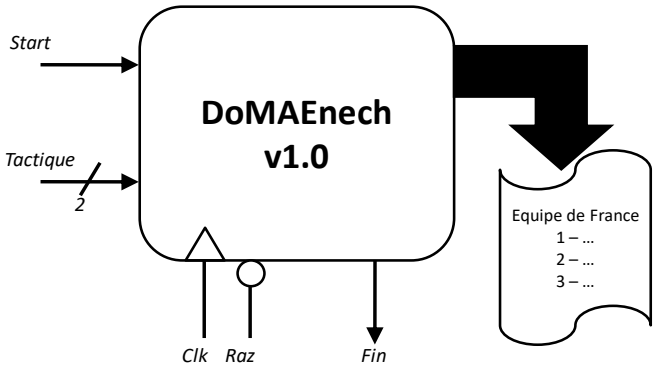
SYSTEMES NUMERIQUES
& PROCESSEURS EMBARQUES
LU3EE100 — TD N°7

L'objectif de cette séance est de construire un système numérique, à partir de son cahier des charges et de son schéma-bloc partie opérative/partie contrôle. On précisera les détails d'implémentation de la partie opérative, et on explicitera le graphe d'état de la partie contrôle.

Exercice 1 : DoMAEnech v6.0 (Euro 2020 Edition) (Adapté d'un sujet d'examen)



Depuis la Coupe du Monde 2010 et l'épisode du bus de Knysna, la Fédération Française de Football a décidé de mettre à la disposition de son entraîneur un outil de sélection automatique d'équipe de football. Ce système numérique, appelé ***DoMAEnech v6.0*** (en "hommage" à Raymond Domenech) permet de générer la liste des 11 joueurs titulaires pour le match, selon leurs caractéristiques techniques et d'une tactique de jeu prédéfinie.



Le principe du ***DoMAEnech v6.0*** est présenté ci-contre. On choisit d'abord une des 4 tactiques de jeu implémentées (le 4-3-3, le 4-4-2, le 5-4-1 ou le WM utilisé au début du XXème siècle).

Le processus de sélection du ***DoMAEnech v6.0*** démarre par la mise à 1 de la commande ***START***. le système effectue alors une recherche parmi sa ***BASE DE DONNEES*** de 26 joueurs pour sélectionner les 11 joueurs titulaires pour le match de l'Equipe de France.

L'équipe sera composée d'un ***GARDIEN*** de but, de ***DEFENSEURS*** centraux, de 2 arrières ***LATERAUX*** (un à gauche et un à droite), de ***MILIEUX*** de terrain, de 2 ***AILIERS*** (un à gauche et un à droite) et d'un ou plusieurs ***ATTAQUANTS.*** Le nombre de défenseurs, mileux et attaquants dépend de la tactique choisie (voir Annexe 2 pour un exemple de tactique).

La ***BASE DE DONNEES*** est donnée en Annexe 1: chaque joueur y est stocké par son ***NUMERO*** de maillot (équivalent à une adresse mémoire), son ***NOM*** de famille, son ***POSTE*** (gardien, défenseur, milieu, etc…), son ***PIED*** préféré (gauche, droit ou les deux) et si le joueur a déjà été sélectionné comme ***TITULAIRE*** sur un poste précédent.

PROCESSUS DE SELECTION

Le processus de sélection du ***DoMAEnech v6.0*** est le suivant

- Le processus est démarré par la mise à 1 de ***START***. A ce moment, la colonne ***TITULAIRE*** de la base de données est remise à 0 pour tous les joueurs. Tous les compteurs doivent avoir été initialisés à 0.

- La tactique étant connue, le système commence par sélectionner le joueur qui occupera le poste 1 sur le terrain (voir l'Annexe 2), puis le poste 2, le poste 3, et ainsi de suite jusqu'au poste 11.

- Pour cela, on parcourt la ***BASE DE DONNEES***, en commençant par la première ligne (LLORIS)

- Sur chaque poste, pour qu'un joueur soit sélectionné, on vérifiera dans cet ordre les contraintes suivantes :

    1) On vérifie d'abord que le ***POSTE*** du joueur correspond bien au poste demandé (par exemple un joueur ***ATTAQUANT*** ne pourra pas être sélectionné comme ***GARDIEN*** de but)
    2) On vérifie ensuite que le ***PIED*** du joueur est compatible avec le poste à affecter. Pour les ***LATERAUX*** et les ***AILIERS***, seul un gaucher (ou un gaucher-droitier) peut être sélectionné pour jouer à gauche, et seul un droitier (ou un gaucher-droitier) peut être sélectionné pour jouer à droite. Pour les autres postes, le pied n'a pas d'importance.
    3) Enfin, il faut vérifier que le joueur n'a pas déjà été sélectionné sur un autre poste (sa colonne ***TITULAIRE*** doit être égale à 0)

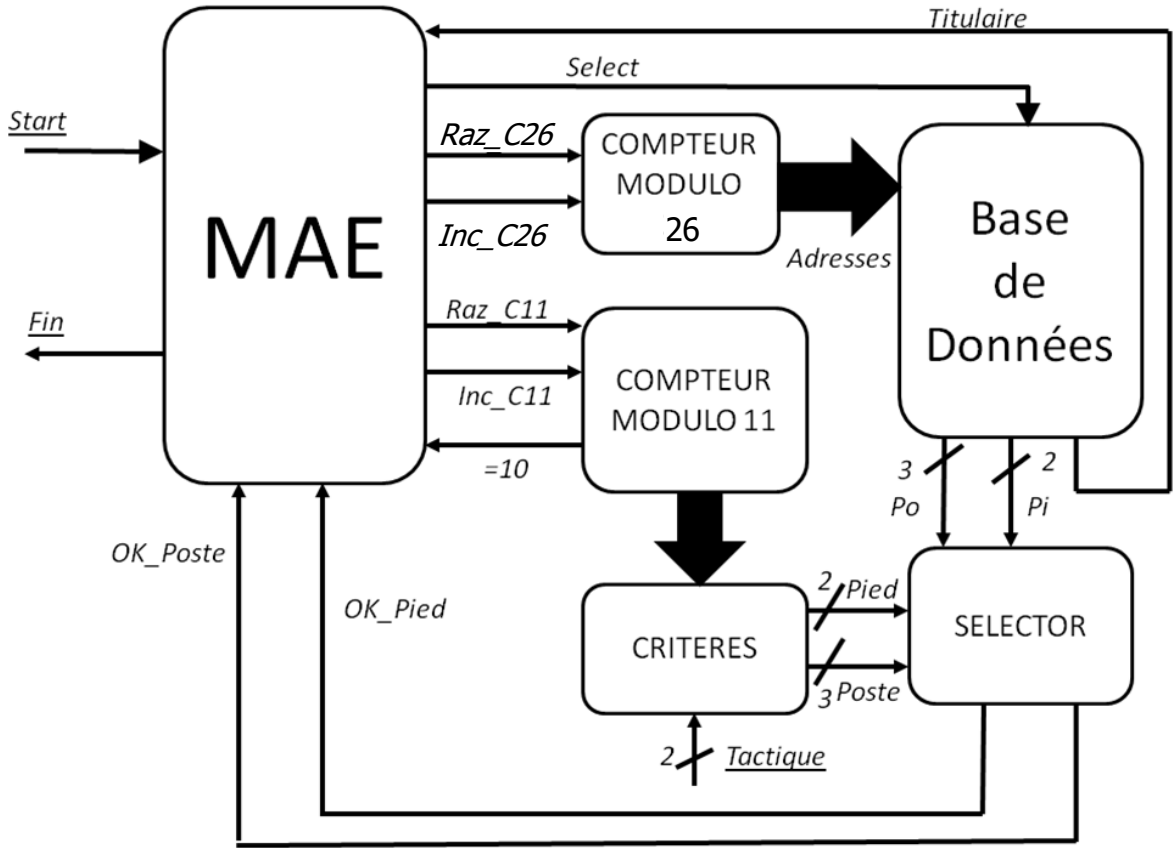
- Si le premier joueur de la base de données ne correspond pas à l'un de ces critères, on passe au joueur numéro 2, etc… Le premier joueur qui satisfait à tous les critères est automatiquement sélectionné comme titulaire. On active alors le signal ***SELECT*** pour que dans la ***BASE DE DONNEES***, la colonne ***TITULAIRE*** du joueur sélectionné passe à 1.

- On précise qu'il n'y a pas de remise à zéro du parcours de la base de données durant tout le processus de sélection. Ainsi, si BENZEMA est sélectionné pour jouer au poste 7, le premier joueur qui sera examiné pour le poste 8 sera COMAN.

- Lorsque les 11 joueurs titulaires ont été choisis, la sortie ***FIN*** passe à 1 et la liste des joueurs titulaires est automatiquement imprimée (nous n'aurons pas à gérer la partie impression)

ARCHITECTURE

Le schéma bloc de l'architecture du ***DoMAEnech v6.0*** est fourni ci-dessous. Les signaux d'entrée/sortie du système sont soulignés sur le schéma. L'horloge et le reset asynchrone n'ont pas été représentés pour plus de clarté.



NB :

1) La MAE et les compteurs ont également une horloge et un reset asynchrone commun.
2) Toutes les commandes sont actives au niveau haut.

La partie opérative du système est composée de :

- La ***BASE DE DONNEES*** des joueurs. Elle fournit en sortie le pied ***Pi,*** le poste ***Po*** et le statut de ***TITULAIRE*** du joueur. La base est adressée par le ***COMPTEUR MODULO 26***. La valeur de ce compteur correspond au numéro du joueur que l'on veut examiner. (Exemple : si le compteur est égal à 0, on aura en sortie les caractéristiques de LLORIS, si le compteur est égal à 8, ce sera celles de GIROUD)
- Le ***COMPTEUR MODULO 26*** qui permet donc de parcourir la ***BASE DE DONNEES***.
- Le ***COMPTEUR MODULO 11*** pour indiquer quel est le numéro de poste que l'on veut attribuer
- Le module ***CRITERES*** qui en fonction du numéro de poste et de la ***TACTIQUE***, indique par un code quels sont les critères de ***PIED*** et de ***POSTE*** requis pour chaque numéro. Par exemple, pour le poste numéro 7 (voir Annexe 2), les critères sont un joueur ***AILIER*** et ***DROITIER***).
- Le module ***SELECTOR*** qui va comparer les valeurs de ***POSTE*** avec ***Po*** et de ***PIED*** avec ***Pi.*** Si ces valeurs sont égales, les sorties ***OK_Poste*** et ***OK_Pied*** passeront à 1, pour indiquer que le joueur est compatible avec le poste demandé.

La partie contrôle est composée d'une ***MACHINE A ETATS***, qui va notamment commander les compteurs et indiquer à la ***BASE DE DONNEES***, en activant la commande ***SELECT***, qu'un joueur sera titulaire. Dans ce cas, la ***BASE DE DONNEES*** modifiera automatiquement la valeur de la case ***TITULAIRE*** du joueur concerné.

Questions :

PARTIE OPERATIVE

La signification du code de ***PIED*** donné par le module ***CRITERES*** est fourni dans le tableau suivant

PIED	
00	NON ATTRIBUE
01	Droitier obligatoire
10	Gaucher obligatoire
11	Gaucher ou droitier

Les infos sur le pied des joueurs (signal ***Pi*** fourni par la ***BASE DE DONNEES)*** sont données ci-dessous

PIED (Pi)	
00	Nul des deux pieds
01	Droitier seulement
10	Gaucher seulement
11	Gaucher ET Droitier

Les codes utilisés pour Poste et Po sont quant à eux identiques.

1) ***Donner les équations booléennes du module SELECTOR***

2) ***Donner le schéma bloc du module COMPTEUR MODULO 11***

PARTIE CONTROLE

3) ***Préciser les entrées et les sorties du module MAE (Machine à Etats).***

4) ***Donner le graphe d'états de la MAE du DoMAEnech v6.0. Pour chaque état on précisera quelles sorties sont actives (niveau haut).***

5) ***Compléter la feuille de match en Annexe en écrivant le nom des 11 joueurs titulaires de l'Equipe de France.***

Question subsidiaire :

Selon vous, si Didier Descjamps avait utilisé le DoMAEnech v6.0 pour faire son équipe, quel aurait été le résultat du match France – Suisse à l'Euro 2020 ?

ANNEXE 1

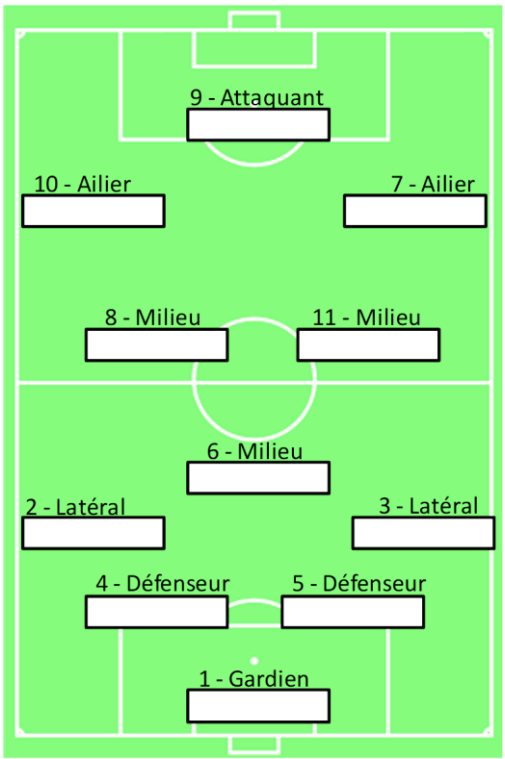
LISTE DE JOUEURS SELECTIONNABLES PAR LE

DoMAEnech v6.0

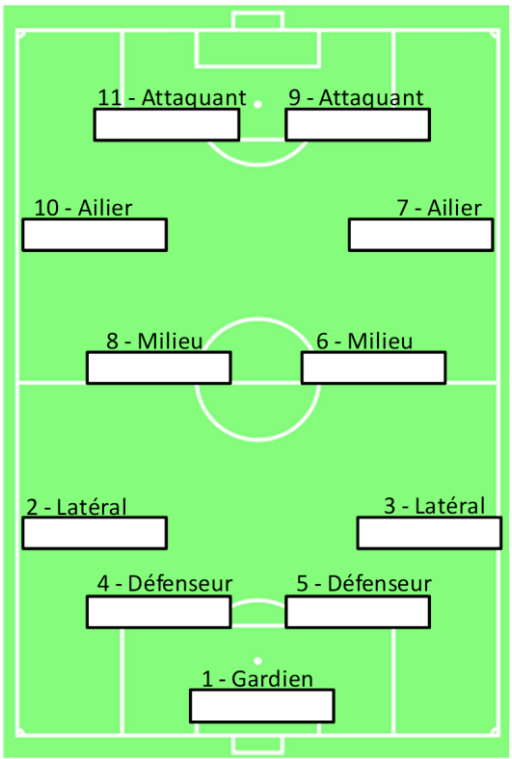
NUMERO	NOM	POSTE	PIED	TITULAIRE
1	LLORIS	Gardien	Gauche	0
2	PAVARD	Latéral	Droit	0
3	KIMPEMBE	Défenseur	Gauche	0
4	VARANE	Défenseur	Droit	0
5	LENGLET	Défenseur	Gauche	0
6	POGBA	Milieu	Droit	0
7	GRIEZMANN	Attaquant	Gauche	0
8	LEMAR	Ailier	Gauche	0
9	GIROUD	Attaquant	Gauche	0
10	MBAPPE	Ailier	Droit	0
11	DEMBELE	Ailier	Gauche ET Droit	0
12	TOLISSO	Milieu	Droit	0
13	KANTE	Milieu	Droit	0
14	RABIOT	Milieu	Gauche	0
15	ZOUMA	Défenseur	Droit	0
16	MANDANDA	Gardien	Droit	0
17	SISSOKO	Milieu	Droit	0
18	DIGNE	Latéral	Gauche	0
19	BENZEMA	Attaquant	Droit	0
20	COMAN	Ailier	Droit	0
21	HERNANDEZ	Latéral	Gauche	0
22	BEN YEDDER	Attaquant	Droit	0
23	MAIGNAN	Gardien	Droit	0
24	DUBOIS	Latéral	Droit	0
25	THURAM	Attaquant	Droit	0
26	KOUNDE	Défenseur	Droit	0

ANNEXE 2

TACTIQUE : 4-3-3

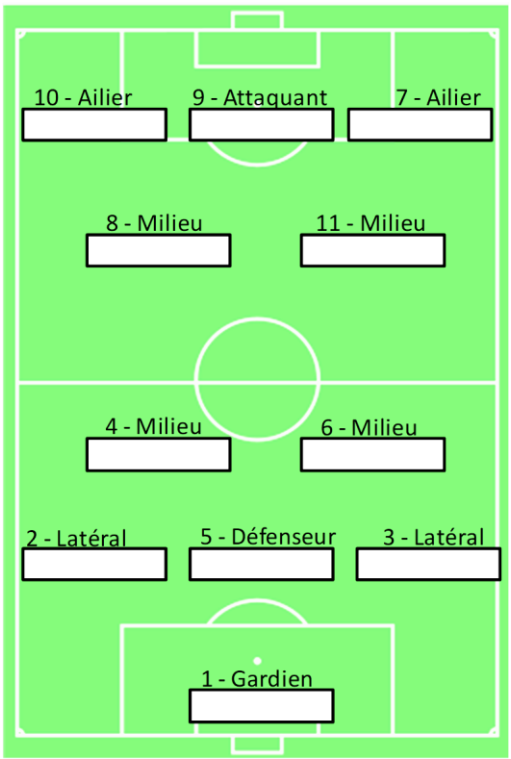


TACTIQUE : 4-4-2



TACTIQUE : 5-4-1

TACTIQUE : WM



SORBONNE UNIVERSITÉ
CRÉATEURS DE FUTURS DEPUIS 1257

SYSTEMES NUMERIQUES & PROCESSEURS EMBARQUES LU3EE100 — TD N°8

Exercice 1

On considère le tableau mémoire ci-dessous :

1	0	1	1	0	0	0	1	0	1	0	1	1	1	0	0	0	0	0	1	1	0	1	1	0	0	0	1	0	1	0	0
1	1	0	0	0	1	0	1	0	1	1	1	0	0	0	0	0	1	1	0	1	1	0	0	0	1	0	1	0	0	1	1
0	0	0	1	0	1	0	1	1	1	0	0	0	0	0	1	1	0	1	1	0	0	0	1	0	1	0	0	1	1	0	1
0	1	0	1	0	1	1	1	0	0	0	0	0	1	1	0	1	1	0	0	0	1	0	1	0	0	0	1	1	0	0	0
0	1	0	1	1	1	0	0	0	0	0	1	1	0	1	1	0	0	0	1	0	1	0	0	0	1	1	0	0	0	1	0
0	1	1	1	0	0	0	0	0	1	1	0	1	1	0	0	0	1	0	1	0	0	0	1	0	0	0	1	0	0	0	0
1	1	0	0	0	0	0	1	1	0	1	1	0	0	0	1	0	1	0	0	1	1	0	0	0	1	1	0	0	0	0	0
0	0	0	0	0	1	1	0	1	1	0	0	0	1	0	1	0	0	0	0	0	1	0	0	1	0	1	0	0	0	0	0
0	0	0	1	1	0	1	1	0	0	0	1	0	1	0	0	0	0	0	1	0	1	0	0	0	0	0	1	0	1	0	1
0	1	1	0	1	1	0	0	0	1	0	1	0	0	1	0	0	0	0	1	0	1	0	0	0	0	1	0	1	1	1	1
1	0	1	1	0	0	0	1	0	1	0	0	1	1	0	0	0	1	1	0	1	1	0	0	0	1	1	0	0	0	1	1
1	1	0	0	0	1	0	1	0	0	1	0	1	1	0	0	0	0	1	0	1	0	0	1	0	0	1	1	0	0	0	1
0	0	0	1	0	1	0	0	1	0	1	1	0	0	1	1	0	0	0	1	0	1	0	1	1	1	0	0	0	0	0	1
0	1	0	1	0	0	1	0	0	0	1	0	1	0	1	1	1	0	0	0	0	0	1	0	0	1	0	1	0	0	1	0
0	1	0	0	1	0	0	0	0	0	1	1	0	1	1	0	0	0	0	0	0	1	0	0	0	0	0	0	1	1	0	1
0	0	1	0	0	0	1	0	1	0	1	1	1	0	0	0	0	0	1	1	0	0	0	1	1	0	0	0	1	0	1	0
1	0	1	1	0	0	0	1	0	1	0	1	1	1	0	0	0	0	0	1	1	0	1	1	0	0	0	1	0	1	0	0
1	1	0	0	0	1	0	1	0	1	1	1	0	0	0	0	0	1	1	0	1	1	0	0	0	1	0	1	0	1	1	0
0	0	0	1	0	1	0	1	1	1	0	0	0	0	0	1	1	0	1	1	0	0	0	1	0	1	0	1	1	0	0	0
0	1	0	1	0	1	1	1	0	0	0	0	0	1	1	0	1	1	0	0	0	1	0	1	0	0	0	0	0	0	1	0
0	1	0	0	1	1	0	0	0	1	0	1	0	0	1	0	0	0	1	0	1	0	0	1	0	1	1	0	1	0	1	0
0	1	1	1	0	0	0	1	0	1	0	0	0	1	0	0	1	0	0	1	0	1	1	0	0	0	0	0	1	0	0	0
1	1	0	0	0	1	0	1	0	0	0	1	1	0	0	1	0	1	0	1	1	1	0	0	0	1	1	0	0	1	1	0
0	0	0	1	0	1	0	0	0	1	0	0	0	1	0	1	0	0	1	1	0	0	0	0	1	0	1	0	0	0	1	0
0	0	0	1	0	0	1	1	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0
0	1	1	0	0	0	0	1	0	0	1	0	1	0	1	0	0	0	0	0	0	1	1	0	0	0	1	0	1	0	1	0
1	0	1	0	0	1	0	1	0	0	0	0	0	1	0	0	0	1	0	1	1	0	1	0	0	1	1	1	0	1	1	0
1	1	0	0	0	1	0	1	0	0	0	0	1	0	0	0	0	0	1	0	1	1	0	1	0	0	1	1	0	1	0	0
0	0	0	1	0	1	0	0	1	0	1	1	0	0	0	1	1	1	0	0	0	0	0	1	1	1	0	0	1	0	1	0
0	1	0	1	0	0	1	0	0	0	1	0	1	0	1	1	0	0	0	0	0	1	1	0	1	1	0	0	0	0	0	1
0	1	0	0	1	0	0	0	0	0	1	1	0	1	0	0	0	0	0	1	1	0	1	1	0	0	0	0	0	0	1	0
0	0	1	0	0	0	1	0	1	0	1	1	1	0	0	0	0	1	1	0	1	1	0	0	0	0	0	0	1	0	1	0

Ce tableau mémorise des mots binaires organisés en octets.

1. Préciser la capacité en bits, en octets, et l'organisation du tableau mémoire.

2. Indiquer le nombre de lignes du bus de données, du bus d'adresse et précisant le nombre de lignes des bus en entrée et en sortie du décodeur adresse rangée et du décodeur adresse colonne. Numéroter les rangées et colonnes du tableau mémoire.

Ce tableau mémoire fait partie du bloc mémoire dont la structure est fournie en ***Figure n°1***. Cette mémoire possède comme entrées-sorties un bus d'adresse, un double bus de données Data-In et Data-Out, une ligne de commande R/W, et une ligne de validation.

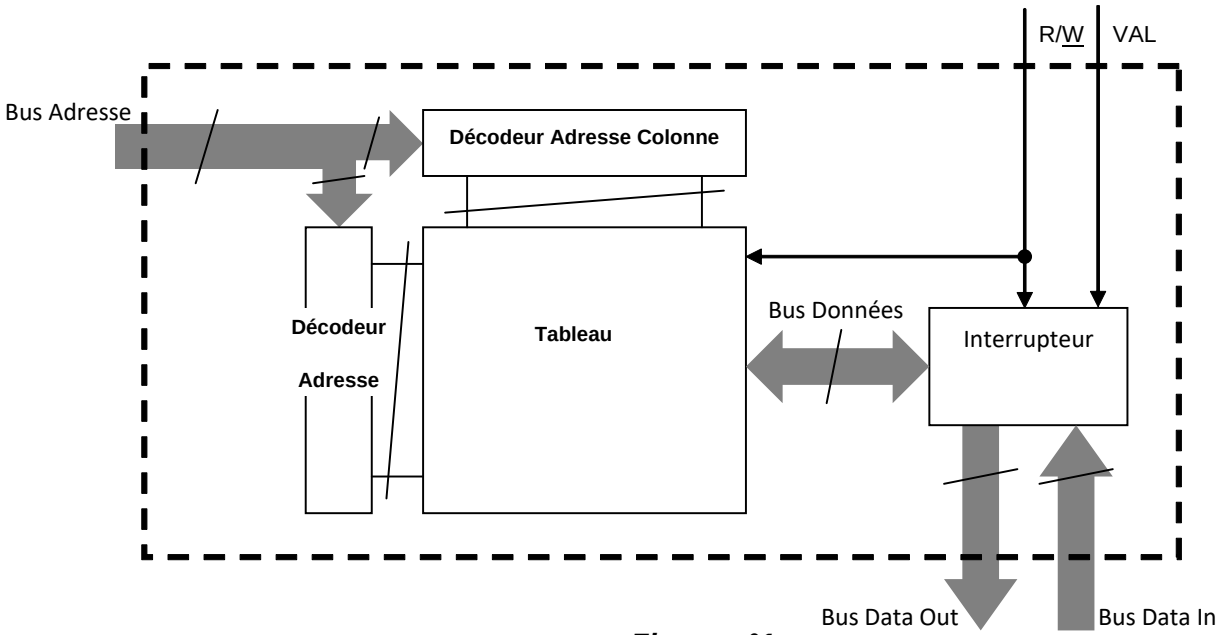
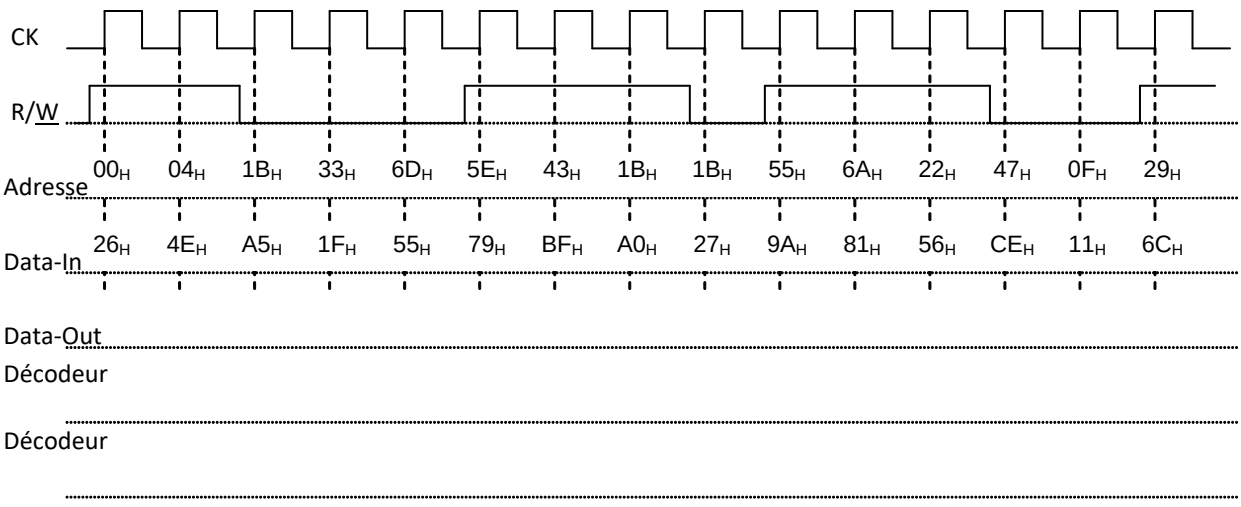


Figure n°1

3. A quelle condition va-t-on écrire une donnée dans le tableau mémoire ? A quelle condition va-t-on lire une donnée dans le tableau mémoire ? Comment fonctionne l'interrupteur commandé ?

On suppose que les opérations de lecture/écriture sont synchronisées sur front montant d'une horloge CK. Les chronogrammes de Adresse, Data-In et R/W sont fournis plus bas. Les bits de poids faible du bus d'adresse sont les bits du bus d'adresse rangée. L'entrée VAL est toujours à 1.



4. Déterminer les lignes de sortie des décodeurs qui sont activées ainsi que la valeur présente sur le bus Data-Out. On considère qu'en cas d'écriture Data-Out conserve la valeur qu'il possédait au moment de la dernière opération de lecture.

5. Mettre en évidence les cellules du tableau mémoire qui ont été modifiées en fin de cycle en donnant leurs nouvelles valeurs.

Exercice 2

On dispose de circuits intégrés de mémoire de capacité 256Kx1 et on désire construire avec ces composants un banc de mémoire 4Mx8.

1. Combien de lignes contiennent les bus de donnée et d'adresse de la capacité 256Kx1 ?
2. Combien de lignes contiennent les bus de donnée et d'adresse de la capacité 4Mx8 ?
3. Combien de mémoires 256Kx1 sont nécessaires pour réaliser la mémoire 4Mx8 ?
4. Proposer un montage pour réaliser la mémoire 4Mx8.

On dispose à présent 4 bancs mémoire reliés à un bus d'adresses commun de 24 bits, ainsi qu'à un bus de données commun (voir Figure 2). Sur les 4 bancs, 2 sont de type 4Mx8, les 2 autres sont de type 2Mx8.

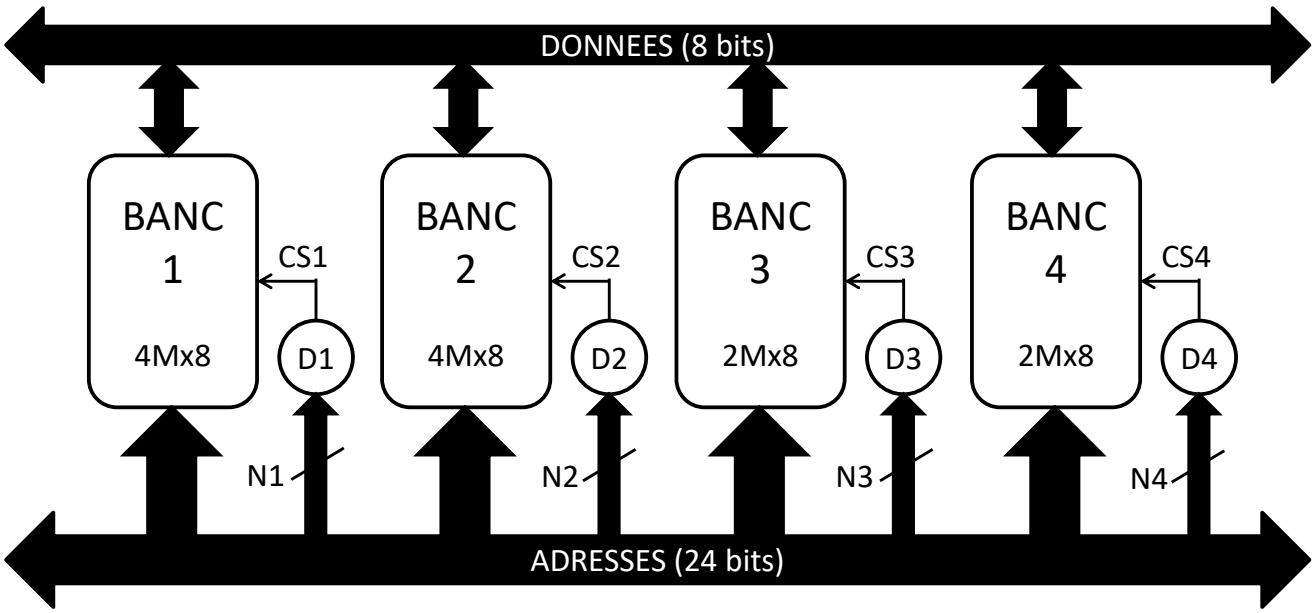


Figure n°2

5. Quel est l'espace d'adressage possible avec cette configuration.
6. Proposer un mapping mémoire des 4 bancs sur cet espace adressable.
7. En déduire les équations des décodeurs d'adresse D1, D2, D3, D4 activant les chip select de chaque banc, en précisant les nombres de nits N1, N2, N3 et N4..

Exercice 3 - (A faire chez soi via Moodle, ne sera pas traité en séance)

Une mémoire cache est utilisée pour faire l'interface avec la mémoire RAM d'un processeur. (***Figure n°3***), dans le but d'accroître ses performances.

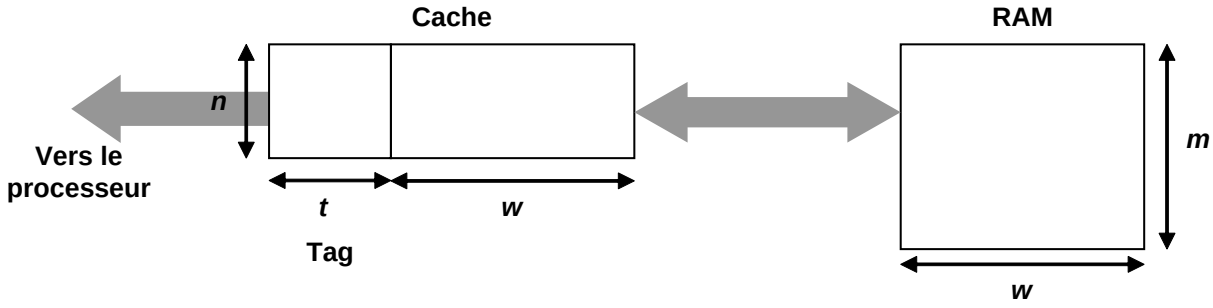


Figure n°3

Le cache est réalisé avec une technologie rapide mais gourmande en surface (comme la SRAM), il est donc trop coûteux d'implémenter toute la RAM de cette manière, on préfère ainsi une technologie plus compacte comme la DRAM, même si elle n'est pas aussi rapide.

L'accroissement des performances grâce à l'utilisation d'une mémoire cache est dû à deux principes qui ont été découverts dans des études sur le comportement des programmes informatiques:

- Principe de localité spatiale : Une demande d'accès à une instruction située à l'adresse X a de grandes chances d'être suivie d'une demande d'accès à une instruction très proche de X.
- Principe de localité temporelle : Une demande d'accès à une instruction située à l'adresse X a de grandes chances de se reproduire dans la suite immédiate du programme.

L'utilisation d'un cache rapide contenant les données récemment utilisées par le processeur permet ainsi d'améliorer ses performances.

1. Quelle est la différence fondamentale entre une mémoire cache et une mémoire tampon ?

Le principe de fonctionnement est le suivant:

- ❖ Le processeur demande une information contenue dans la RAM
- ❖ Le cache vérifie qu'il la possède :
  - ➢ Il la possède ("succès de cache")
  - ➢ Il ne la possède pas ("défaut de cache"): il la demande à la RAM. La RAM renvoie l'information au cache qui la recopie pour la stocker
- ❖ Le cache renvoie l'information au processeur

Une ligne du cache contient plusieurs mots. Généralement c'est une ligne complète qui est transmise de la RAM au cache et un mot du cache au processeur.

On définit le mapping comme étant la méthode utilisée pour définir à quel endroit du cache écrire une ligne de la RAM. Il y a trois types de mapping.

- Cache direct (Direct mapped cache) : Une ligne de la mémoire principale ne peut être écrite qu'à une seule adresse de la mémoire cache (voir ***Figure n°4***)
- Cache associatif (Fully associative cache) : Une ligne de la mémoire principale peut être écrite à n'importe quelle adresse de la mémoire cache.
- Cache associatif par ensemble N-ways (N-way set associative cache) : Une ligne de la mémoire principale ne peut être écrite qu'à N (puissance de 2) adresses de la mémoire cache.

En plus des données recopiées de la RAM, chaque ligne de la mémoire cache contient une étiquette (tag), c'est à dire une information qui précise de quelle adresse RAM est issue la donnée. Lorsque le processeur fait appel à une donnée, il appelle une adresse de la RAM. La première partie de l'adresse RAM est équivalente à une étiquette, la seconde partie à une adresse du cache (dans le cas du cache direct).

On considère une RAM dont le bus d'adresse est de 32bits et un cache dont la taille est 2Koctets. Les deux mémoires sont organisées en lignes contenant 16 mots. Les mots sont des octets.

2. Quelle est la taille de la RAM ? Préciser le nombre de lignes ***m*** (dont la numérotation va de 0 à m-1) de la RAM, ainsi que le nombre de lignes ***n*** du cache. Décomposer le bus d'adresse de la RAM en plusieurs blocs et préciser l'affectation de chaque bloc.

La méthode d'écriture des données dans le cache est le mapping direct.

3. Indiquer les lignes de la RAM pouvant être écrites sur la ligne 0 du cache.

4. Indiquer les lignes de la RAM pouvant être écrites sur la ligne 87 du cache.

Les mots d'une ligne sont numérotés de 0 à 15.

5. On considère l'octet qui correspond au mot 11 de la ligne 35 de la RAM. Donner l'adresse du mot. Préciser l'adresse des mots de la RAM qui occuperont le mot 11 de la ligne 35 du cache. Comparer les adresses. On considère que l'étiquette est constituée de la partie de ces adresses qui n'est pas commune. Préciser pour chaque mot l'étiquette et indiquer le nombre de bits de cette étiquette.

6. On suppose que au départ le cache recopie intégralement les ***n*** premières lignes de la RAM. Quelle est l'étiquette des données du cache?

7. Le processeur fait appel aux adresses successives ci-dessous. Indiquer pour chaque cas si la donnée est dans le cache ou non et dans le cas d'un défaut de cache, quel est le numéro de la ligne de la RAM qui est recopiée dans le cache.

$0000CE89_H$ 00000135_H 00000484_H $0001A13D_H$

8. Comment s'effectue le transfert de données vers le processeur en cas de cache associatif ? Quelle est la dimension de l'étiquette ?

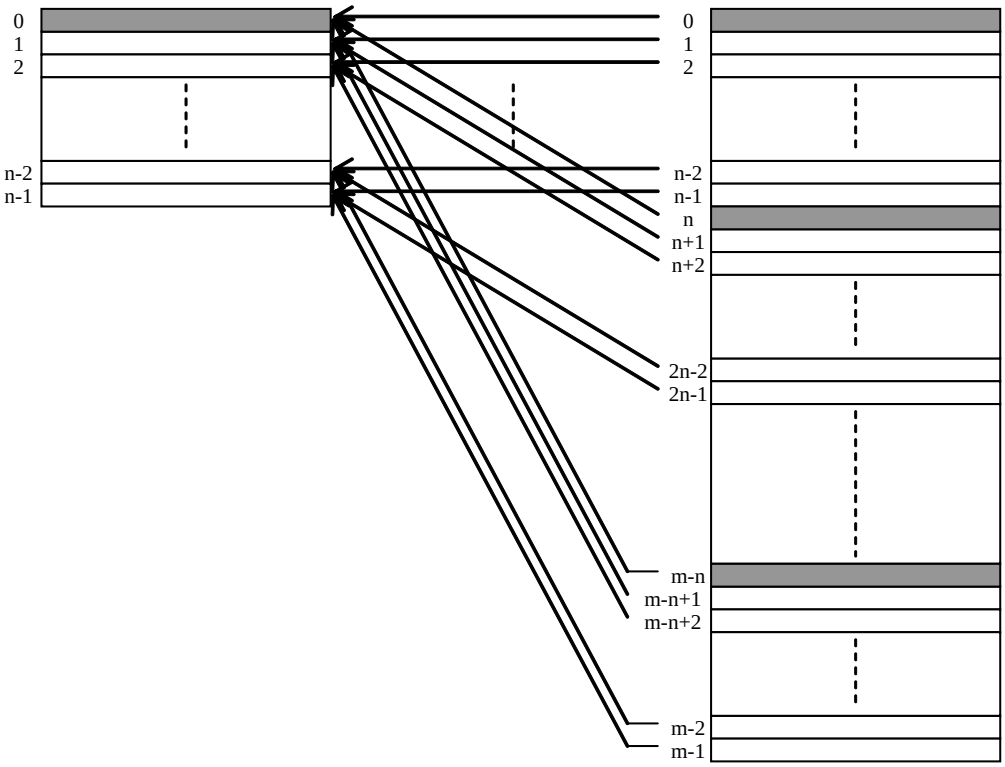


Figure n°4

SORBONNE UNIVERSITÉ
CRÉATEURS DE FUTURS DEPUIS 1257

SYSTEMES NUMERIQUES & PROCESSEURS EMBARQUES LU3EE100 — TD N°9

L'objectif de cette séance est d'appréhender l'architecture et le jeu d'instructions du processeur LC-3, à l'aide d'un simulateur graphique, avec un accent sur les différents modes d'adressages proposés par ce processeur.

Préparation

Lire cette section et vérifiez notamment AVANT LE TD que vous pouvez bien ouvrir le simulateur Java du LC-3. Si ce n'est pas le cas, il faut installer ou mettre à jour Java Runtime Environment (voir ci-dessous pour la procédure). Il faut également récupérer sur Moodle (dans la section TD) les codes sources utilisés dans l'exercice 2 du sujet.

a) Outils de simulation

- Pour ce TD, vous allez utiliser un éditeur/assembleur en ligne de code pour le processeur ***LC-3***, disponible à cette adresse : http://lc3tutor.org/
- Cet outil permet également de simuler les programmes, néanmoins, pour ce TD, pour bien analyser les mécanismes d'exécution des instructions des programmes, nous allons utiliser une application Java réalisant un ***simulateur graphique du LC-3***. L'application est téléchargeable depuis le site Moodle de l'UE, dans la section TD.
- Pour lancer le simulateur, il suffit juste de cliquer sur le fichier LC3_Simulator.jar. Si le simulateur ne se lance pas, alors il faut installer ou mettre à jour votre plate-forme Java Runtime Environment. Elle est téléchargeable ici : https://www.java.com/fr/download/

b) Editeur/Assembleur en ligne

- Même s'il est possible d'écrire directement du code dans l'outil en ligne, il est préférable d'écrire et de sauvegarder vos programmes avec un éditeur de texte

comme Notepad++. Vous trouverez sur Moodle une extension permettant d'ajouter une mise en forme de la syntaxe Assembleur LC-3 à Notepad++ (avec un mode d'emploi pour l'installer).

- Pour assembler un programme avec le simulateur, cliquer sur l'icône ***Assemble*** en bas de la fenêtre.
    - Faire un copier/coller depuis Notepad pour recopier votre programme dans la fenêtre qui s'ouvre dans le simulateur
    - Cliquer sur ***Assemble*** pour lancer l'assemblage.

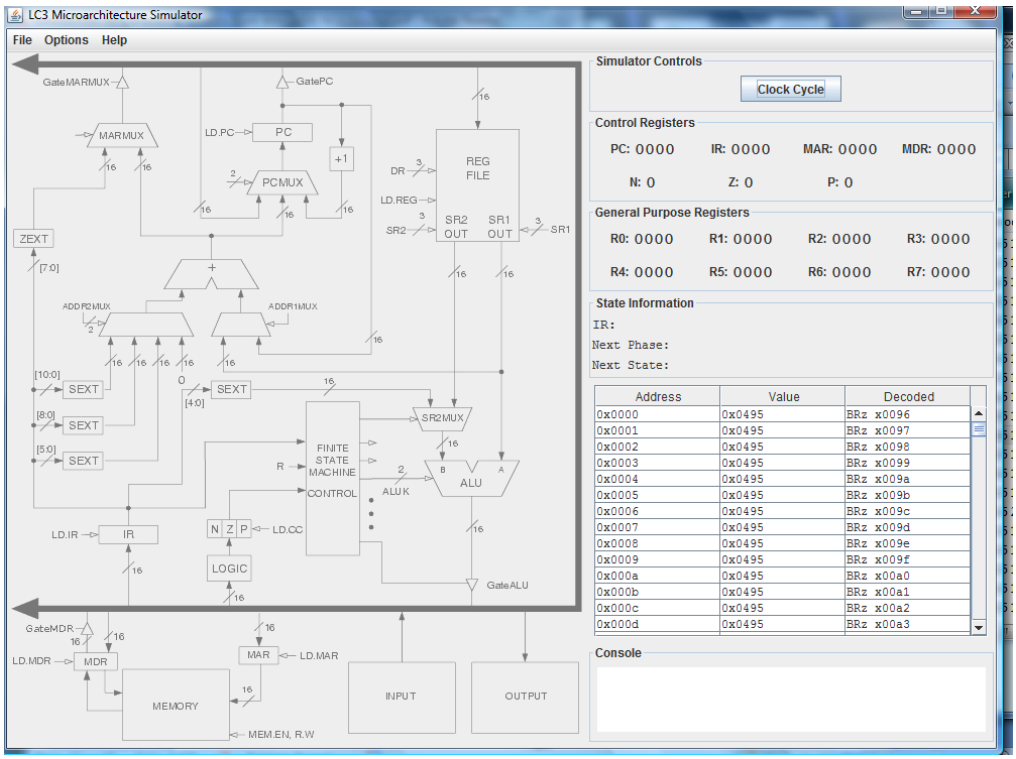
- En cas d'erreur, un bandeau rouge apparaît indiquant les erreurs avec leur numéro de ligne (retourner sur Notepad pour pouvoir profiter de l'affichage des numéros de ligne avec cet éditeur).

- Si le code est correct, un bandeau vert apparaît, vous pouvez alors télécharger le fichier objet (le code exécutable). Renommer le fichier téléchargé avec l'extension .obj pour le retrouver plus facilement au moment du chargement sur le simulateur.

c) ***Simulateur graphique***

- L'application Java ***LC3_simulator.jar*** est un simulateur graphique qui permet de suivre cycle après cycle l'exécution d'un programme et le cheminement des données dans le processeur.

- A l'ouverture, on trouve la fenêtre suivante



- La moitié gauche de l'écran est le schéma bloc du processeur. A droite, vous trouvez de haut en bas les sous-fenêtres suivantes
    - ***Simulator Controls*** : permet d'avancer la simulation d'un cycle d'horloge en appuyant sur le bouton ***Clock Cycle***
    - ***Control Registers*** : Indique les valeurs des registres PC, IR, MAR, MDR et l'état des drapeaux N,Z et P
    - ***General Purpose Registers*** : indique les valeurs des registres R0,…,R7
    - ***State Information*** : donne l'état de l'exécution de l'instruction courante.
    - ***Tableau mémoire*** : ses trois colonnes indiquent l'adresse de la case mémoire, son contenu (en hexadécimal) et sa signification (en considérant que la valeur en hexadécimal correspond à une instruction)
    - ***Console*** : Indique des messages.
- Dans le menu ***Options***, vous pouvez régler le contraste de la fenêtre et aussi la vitesse des animations.
- Enfin, pour charger un programme, il faut aller dans le menu ***File → Open*** puis sélectionner un fichier ***.obj***.

Exercice 1 – Prise en main

1) Ecrire un programme à partir de l'adresse ***x3000*** qui initialise deux constantes A à 6 et B à 4 et qui réalise l'opération ***A-B***.
    - Quelles instructions ou directives assembleur devez-vous utiliser pour cela ?
    - Assembler le code à l'aide du simulateur en ligne pour corriger d'éventuelles erreurs d'écriture. Renommer le fichier objet avec l'extension .obj après l'avoir téléchargé.

2) Charger le ***.obj*** dans le simulateur Java. Le tableau mémoire va pointer automatiquement à l'adresse ***x3000***.
    - Quelle est la valeur du registre ***PC*** ?
    - Appuyer une fois sur ***Clock Cycle***. Que s'est-il passé ?
    - Au bout de combien de cycles l'instruction est-elle chargée dans le ***LC-3*** ?
    - Combien de cycles sont nécessaires pour décoder l'instruction?
    - Combien de cycles sont nécessaires pour aller chercher l'opérande et exécuter l'instruction?
    - Observer la valeur des drapeaux ***N,Z,P***. Pouvez-vous l'expliquer ?
    - Exécuter le reste du programme en observant l'évolution des données dans l'architecture du ***LC-3***.
    - Que se passe-t-il à l'exécution de l'instruction ***TRAP x25*** ?

Exercice 2 – Modes d'adressage

1) Le fichier ***Exo2_1.asm*** effectue la somme des constantes ***A***, ***B*** et ***C*** et stocke le résultat dans une variable appelée ***SUM***.
   - Ouvrir le programme et comprendre son fonctionnement.
   - Assembler le programme. Pourquoi y a-t-il une erreur ? Corriger
   - Quelle instruction faudrait-il rajouter pour être sûr que la somme calculée dans ***R1*** soit bien correcte ?
   - Exécuter le code sur le simulateur et vérifier son bon fonctionnement
   - Quel mode d'adressage a été utilisé ?
   - A quelle adresse est stocké le résultat ?

2) Ecrire un nouveau programme qui réalise la même somme mais stocke le résultat à l'adresse ***x4000***.
   - Quel mode d'adressage doit-on alors utiliser ?

3) Ecrire un nouveau programme qui réalise la même somme mais stocke les sommes partielles puis le résultat final à partir de l'adresse ***x4000***.
   - Quel mode d'adressage doit-on alors utiliser ?

Exercice 3 – Multiplication (A faire à la maison)

1) Ecrire un programme qui réalise la multiplication de deux constantes A=6 et B=4 et qui stocke le résultat en mémoire.
   - On utilisera pour cela l'instruction ***BRnp Label***, qui charge le registre ***PC*** avec l'adresse du ***Label*** si le résultat de la dernière instruction est différent de 0

Exercice 4 – Suite de Fibonacci (A faire à la maison)

1) Ecrire un programme qui réalise le calcul des éléments d'une suite de Fibonacci.
   - ***F(n) = F(n-1) + F(n-2), avec F(0)=1 et F(2) = 2.***
   - On utilisera pour cela l'instruction ***BR Label*** qui charge le registre ***PC*** avec l'adresse du ***Label***
   - Exécuter ce programme sur le simulateur pour vérifier son bon fonctionnement.

SORBONNE UNIVERSITÉ
CRÉATEURS DE FUTURS DEPUIS 1257

SYSTEMES NUMERIQUES & PROCESSEURS EMBARQUES LU3EE100 — TD N°10

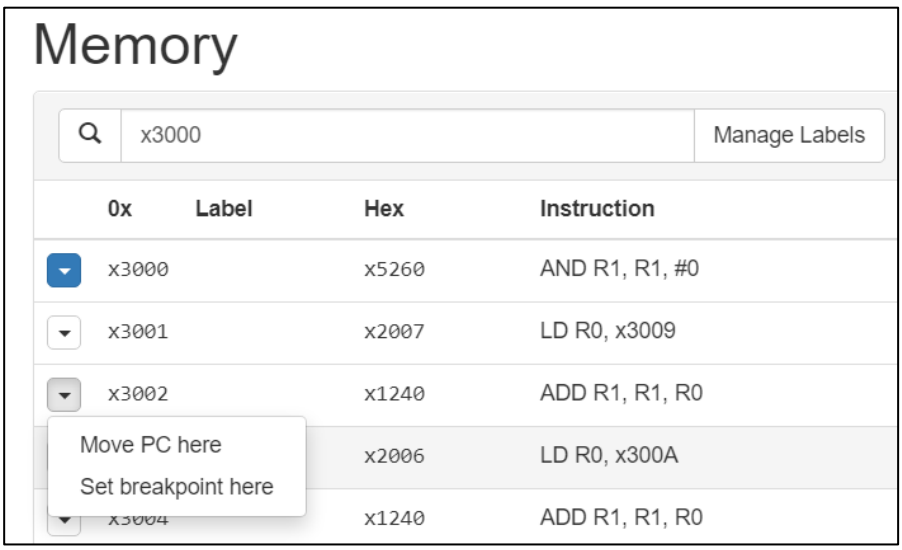
L'objectif de cette séance est d'appréhender la programmation en assembleur du LC-3 avec l'utilisation des instructions de branchement conditionnel et des appels système.

Préparation

Pour ce TD, nous allons utiliser, pour simuler les programmes, l'éditeur/assembleur en ligne du LC-3 que nous avons aperçu au TD précédent (http://lc3tutor.org/) Ce simulateur est en plus adapté aux exercices du TD. Lire le mode d'emploi de ce simulateur pour comprendre son fonctionnement. Les programmes assembleur utiles au TD sont également téléchargeables sur le site Moodle de l'UE, section TD.

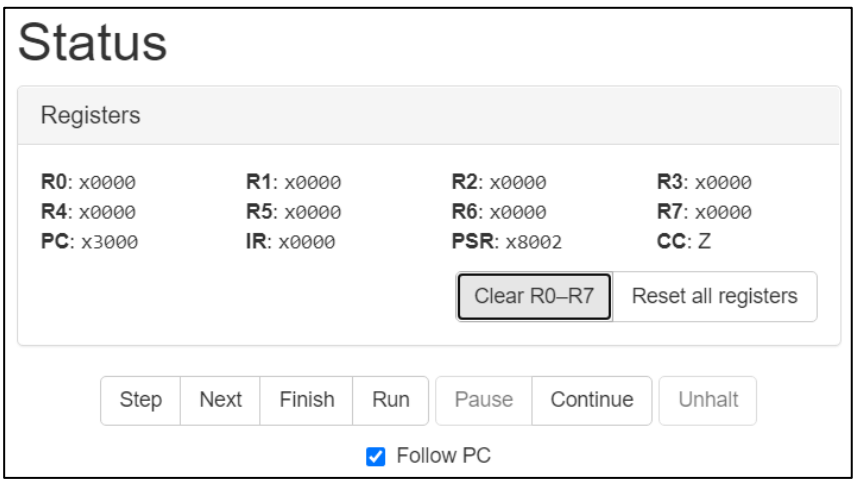
Editeur/Assembleur

- L'outil ***LC3Simulator*** est un outil similaire au simulateur Java du TD10, mais sans le côté graphique. Le programme est exécuté en une fois ou bien instruction par instruction (et non pas cycle d'horloge par cycle d'horloge). L'outil permet de voir l'état des registres, ainsi que de la mémoire. Une console permet d'écrire ou bien de lire des caractères et ainsi interagir avec le programme.



- Dans la section ***Memory***, vous trouvez le contenu de la mémoire du processeur. La zone du haut (avec la loupe) vous permet de vous déplacer à l'adresse que vous aurez inscrite.
- Le tableau vous donne pour chaque adresse la valeur en hexadécimal de la case mémoire, ainsi que son contenu désassemblé (en supposant que la case contienne une instruction)
- En cliquant à gauche de chaque adresse vous pouvez modifier la valeur du registre PC pour la mettre à l'adresse sélectionnée (***Move PC Here***) ou bien insérer un point d'arrêt (***Set Breakpoint Here***). Un point d'arrêt permet de suspendre automatiquement l'exécution d'un programme dès que l'on arrive à l'adresse marquée par ce breakpoint. Il et possible de retirer un point d'arrêt en cliquant à nouveau à gauche de l'adresse concernée.

- Les icônes du menu en bas de la section Memory permettent, de gauche à droite :
  - De naviguer dans le tableau mémoire vers le haut ou vers le bas
  - De recentrer la zone visible du tableau par rapport à la valeur du registre PC.
  - De charger un fichier objet issu d'un assemblage précédent. Pour cela, il faut glisser le fichier ***.obj*** dans la zone indiquée et cliquer sur ***Process Files***.
  - D'assembler un programme assembleur et ensuite de télécharger le fichier objet, ou bien pour le charger dans le LC-3 en cliquant sur ***Load into Simulator***.
  - De générer un fichier objet à partir d'un code écrit en binaire ou en hexadécimal. Nous n'utiliserons pas cette option.



- La section ***Status*** vous donne accès au contenu des registres du LC-3. Ce contenu est remis à jour après chaque instruction. L'état des indicateurs ***NZP*** est visible dans le registre ***CC***.

- Les icônes en bas de la section permettent :
  - ***Step*** : Exécuter une instruction (en rentrant éventuellement dans un sous-programme)
  - ***Next*** : Exécute une instruction du programme principal ou l'intégralité du sous-programme dans lec as d'une instruction JSR, JSRR ou TRAP.
  - ***Finish*** : Dans le cas, où l'on se trouve dans un sous-programme, exécute d'un bloc le reste du sous-programme.
  - ***Run*** : Exécute le programme d'un bloc, jusqu'à la rencontre d'un point d'arrêt ou de l'instruction TRAP x25 (HALT)
  - ***Pause*** : Met le programme en pause
  - ***Continue*** : Reprend l'exécution du programme après une pause ou un point d'arrêt.
  - ***Unhalt*** : Permet de redémarrer le processeur après un appel système HALT.

- La section ***Console*** comporte une fenêtre dans laquelle seront écrits les messages envoyés par les programmes exécutés par le LC-3
- Vous pouvez aussi écrire des informations en cliquant sur la fenêtre, et en tapant ensuite au clavier.
  - Attention, ce que vous écrivez ne sera pas affiché sur la console. Vous aurez en revanche une indication du nombre de caractères tapés avec le ***Input Buffer*** juste en dessous.
- Il est possible d'effacer le contenu de la console et de l'Input Buffer en cliquant sur les icônes correspondantes.

Exercice 1 – Multiplication

1) Le programme ***Exo1.asm*** effectue une multiplication de deux constantes ***A*** et ***B***. On souhaite modifier ce programme pour remplacer les constantes par deux chiffres entrés au clavier
   - Modifier le code assembleur. On pourra utiliser l'appel système ***IN*** pour l'acquisition des valeurs ***A*** et ***B***.
   - Charger le programme assemblé dans le simulateur et exécuter d'un bloc (***Run***). Que constatez-vous ?
   - Quel traitement doit-on appliquer aux données ***A*** et ***B*** avant de réaliser la multiplication ? Modifier le programme en conséquence et vérifier son bon fonctionnement.

Exercice 2 – Gestion des entrées/sorties

1) Ecrire un programme à partir de l'adresse ***x3000*** qui fait l'acquisition d'un caractère écrit au clavier, affiche ce même caractère à l'écran et enfin stocke le caractère dans une case mémoire. On utilisera pour cela les appels système ***GETC*** et ***OUT***
   - Exécuter le programme d'un bloc pour vérifier son fonctionnement. Ecrire un caractère dans la console et vérifier qu'il s'affiche bien en retour.

On recommence à présent l'exécution du programme, mais en mode ***Step.***

2) Appel système ***GETC***
   - A quelle adresse se trouve le sous-programme correspondant à ***GETC*** ?
   - A quoi servent les cases mémoire ***x406***, ***x407*** et ***x408*** utilisées par le code ?
   - Expliquer le mécanisme d'acquisition d'un caractère.

3) Appel système ***OUT***
   - A quelle adresse se trouve le sous-programme correspondant à ***OUT*** ?
   - A quoi servent les cases mémoire ***x438***, ***x439***, ***x43A*** et ***x43B*** ?
   - Expliquer le mécanisme d'affichage d'un caractère.

Exercice 3 – Boucle d'acquisition (A faire à la maison)

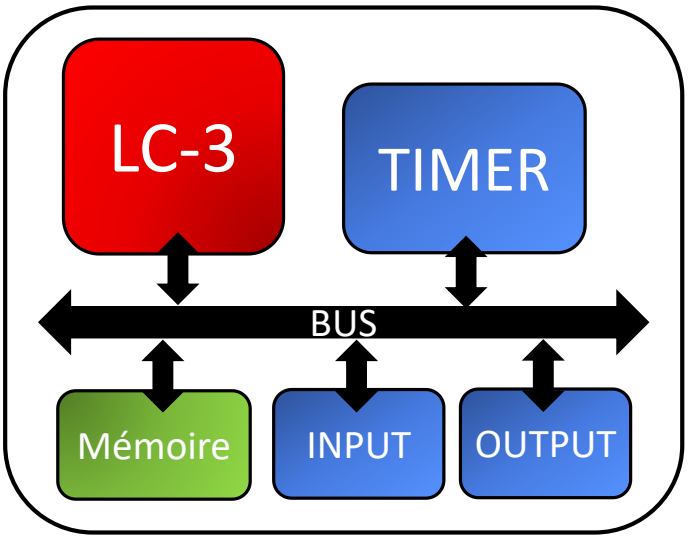
1) Ecrire un programme assembleur comportant une boucle (while ou for), et qui va :
   - Faire la somme de 4 chiffres dont on fait l'acquisition au clavier.
   - Afficher la lettre S si la somme est supérieure ou égale à 16
   - Afficher I sinon

SYSTEMES NUMERIQUES
& PROCESSEURS EMBARQUES
LU3EE100 — TD N°11

L'objectif de cette séance est d'utiliser un périphérique de type Timer que l'on va ajouter au LC-3, et de le programmer par l'intermédiaire de sous-programmes.

Exercice 1 – Sous-programmes

On imagine un système comprenant un processeur LC-3, une mémoire, les périphériques d'entrée et de sortie vus en cours, ainsi qu'un périphérique supplémentaire : un module Timer.



Ce module va permettre de mesurer un temps qui va nous servir à développer l'application suivante :

- On attend que l'utilisateur appuie sur une touche du clavier.
- A partir de ce moment, l'utilisateur a 5 secondes pour entrer un chiffre au clavier.
    - Si ce délai s'écoule sans qu'un chiffre ne soit tapé, l'application se termine
    - Si un chiffre est rentré, alors on va le comparer à un chiffre préalablement stocké en mémoire
        - Si les 2 chiffres sont identiques, on affiche un message « BRAVO ».
        - Sinon, on affiche un message « PERDU ».
- L'application se termine alors

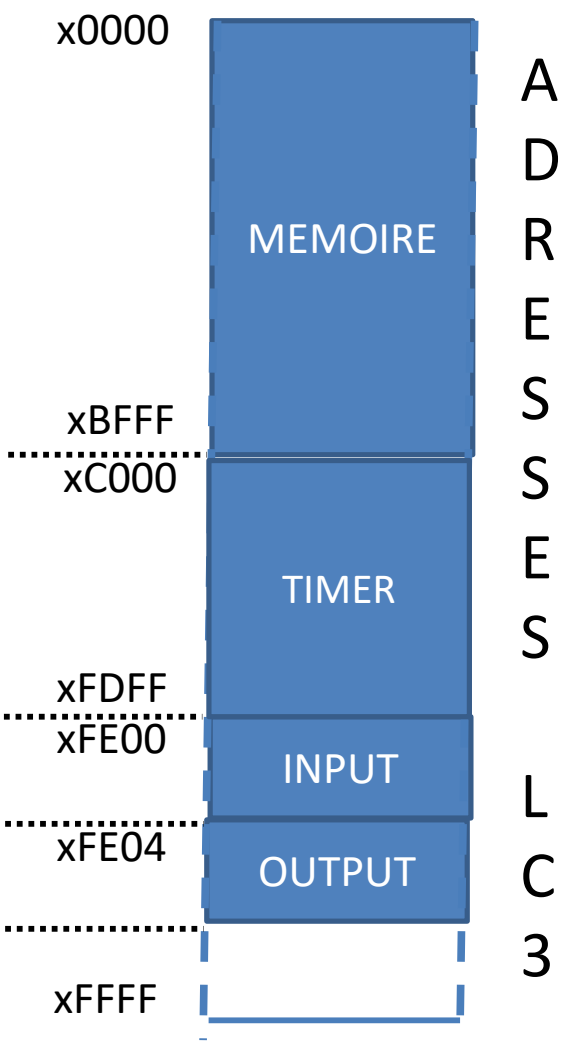
Le plan d'adressage du LC-3 est indiqué ci-dessous. La zone d'adressage du Timer est comprise entre les adresses xC000 et xFDFF.

Le délai de 5 secondes de l'application sera donc mesuré grâce à un Timer. Ce module est organisé autour d'un compteur 16 bits, cadencé par une horloge de fréquence 10 kHz. Lorsque le compteur atteint une valeur MATCH prédéfinie, alors un signal TIMEOUT est positionné, et le compteur s'arrête automatiquement.

L'évolution du Timer peut se faire de deux façons :

- *Incrémentation* : le compteur part de la valeur x0000 et s'incrémente jusqu'à la valeur MATCH.
- *Décrémentation* : le compteur part de la valeur xFFFF et se décrémente jusqu'à la valeur MATCH

La valeur du MATCH permet de définir le temps (le nombre de périodes d'horloges du compteur) que l'on veut mesurer.

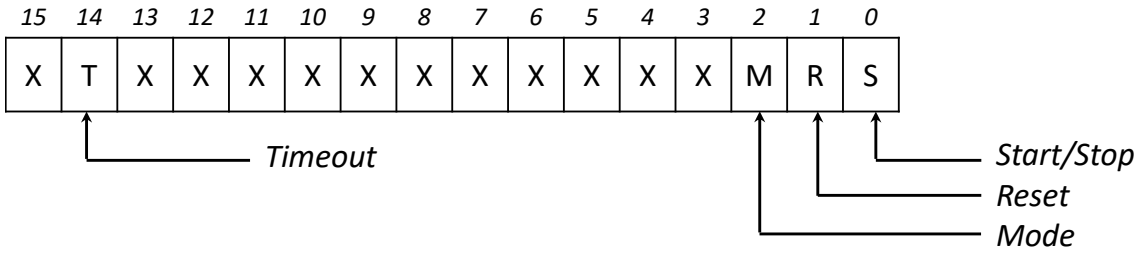


Pour que le LC-3 puisse interagir avec le Timer, celui-ci dispose de trois registres de configuration :

- ***Timer Value Register (TVR) – Adresse : xC000***
  - Ce registre, accessible en lecture seule, contient la valeur courante du compteur.

- ***Timer Match Register (TMR) – Adresse: xC002***
  - Ce registre, accessible en lecture/écriture, contient la valeur de MATCH du Timer.

- ***Timer Status Register (TSR) – Adresse: xC004***
    - Ce registre, accessible en lecture/écriture, permet de configurer le mode de fonctionnement du Timer, et permet de savoir si le TIMEOUT a été atteint.



        - **Bit 14 : Timeout**
            - Ce bit passe automatiquement à 1 lorsque le Timer atteint sa valeur de MATCH. Il repasse à 0 lorsque le Timer est réinitialisé.
        - **Bit 2 : Mode**
            - Permet de choisir si la Timer compte en s'incrémentant (bit à 1) ou en se décrémentant (Bit à 0)
        - **Bit 1 : Reset**
            - La mise à 1 de ce bit provoque un arrêt du compteur et sa remise à zéro. Le bit repasse alors automatiquement à 0.
        - **Bit 0 : Start/Stop**
            - La mise à 1 de ce bit provoque le démarrage du compteur. L'utilisateur peut arrêter le compteur en mettant le bit à 0. Le bit est automatiquement mis à 0 en cas de TIMEOUT ou de réinitialisation.

NB : On supposera que l'utilisateur ne tapera jamais sur une lettre pendant le délai de 5 secondes.

Le programme assembleur de l'application comprend un programme principal (voir ci-dessous), qui appelle plusieurs sous-programmes que nous allons écrire.

```
        .ORIG   x3000

MAIN    JSR INIT_TIMER  ; Sous-Programme de Configuration du Timer
        JSR START_APP   ; Démarrage de l'application
        JSR GET_CHIFFRE ; Acquisition du Chiffre
        JSR CHK_CHIFFRE ; Vérification du Chiffre

        HALT            ; Fin du Programme
```

1) Sous-programme INIT_TIMER

a) Quelle valeur écrire dans le registre TSR pour réinitialiser le Timer ?.
   - En déclarant cette valeur comme constante, écrire les instructions assembleur permettant de réinitialiser le Timer

b) Quelle valeur écrire dans le registre TSR pour choisir le mode « Incrémentation » ?
   - En déclarant cette valeur comme constante, écrire les instructions assembleur permettant de configurer le Timer en mode « incrémentation ».

c) Quelle valeur écrire dans le registre TMR pour que le Timer compte 5 secondes?

d) A l'aide des questions précédentes, écrire le sous-programme INIT_TIMER.

2) Sous-programme START_APP

Le code du sous programme START_APP est fourni ci-dessous.

```
; Sous-Programme du Début de l'Application
START_APP   LEA         R0,MSG1
            PUTS
            GETC
            LEA         R0,MSG2
            PUTS
            RET

MSG1        .STRINGZ    "Appuyer sur une touche..."
MSG2        .STRINGZ    "Choisir un Chiffre avant 5 secondes..."
```

a) Expliquer ce que fait le sous-programme.

b) Après exécution du code, on s'aperçoit que l'on ne revient jamais dans le programme principal. Expliquer pourquoi.

3) Sous-programme GET_CHIFFRE

a) Le sous-programme doit d'abord démarrer le Timer pour compter les 5 secondes. Ecrire les instructions permettant de réaliser cette action.

b) Comment vérifier que l'utilisateur a bien entré un chiffre au clavier (sans utiliser d'appel système) ?
Si tel est le cas, on veut aller à une instruction étiquetée par le label INPUT_OK.
Ecrire les instructions assembleur correspondantes.

c) Comment vérifier avec le LC-3 si le Timer est arrivé au TIMEOUT ?
Sachant que l'on souhaite aller à une instruction étiquetée par le label WAIT_LOOP si on n'est pas arrivé au TIMEOUT, écrire les instructions assembleur réalisant les opérations décrites ci-dessus.

d) Ecrire le sous-programme GET_CHIFFRE. On précise que si on atteint le TIMEOUT, il faudra arrêter le programme. Si au contraire on entre un chiffre avant le TIMEOUT, il faudra revenir au programme principal.

4) Sous-programme CHK_CHIFFRE

a) Le sous-programme CHK_CHIFFRE commence par récupérer le chiffre entré au clavier par l'utilisateur. Quelle instruction permet d'effectuer cette opération ?

b) Ce chiffre est codé en ASCII. Comment obtenir la valeur numérique à partir du code ASCII ? (Rappel des codes ASCII : '0' → x30, '1' → x31, … , '9' → x39). Ecrire le code assembleur correspondant.

c) Quelle instruction permet de récupérer le chiffre préalablement stocké en mémoire, sachant que celui-ci est stocké à l'adresse xA000 ?

d) Ecrire les instructions permettant de comparer ce chiffre à celui entré par l'utilisateur.

e) A l'aide des questions précédentes, écrire le sous-programme réalisant l'application demandée.