Université Sétif 1
Faculté des Sciences
Département d'informatique

2-ème année Inforamtique (année 2018)
Module Architecture des Ordinateurs
Enseignant Kara Abdelaziz

Mini-Projet

Contenants du projet

1. Préambule

Le mini-projet comme pour l'année précédente consiste à réaliser une architecture 8 bits minimaliste de *Von Neumann* qui contient tous les composants de base nécessaires pour la conception d'une architecture hardware complète programmable avec son propre langage machine, l'architecture tourne au tour de l'élément principal qui est le processeur, soutenu d'une mémoire RAM et de quelques éléments d'entrées/sorties basiques. L'architecture en question est décrite dans la section suivante (Description de l'architecture), elle représente une simplification inspiré du processeur Mic-1 du très célèbre livre de *T.A. Tanenbaum* **Structured Computer organisation**, cette version simplifier est décrite par *D.C. Schuurman* dans sont papier **Step-by-Step Design and Simulation of a simple CPU Architecture**, L'architecture représentée ici est une version modifier que celle présentée dans le papier. L'implémentation est à faire sur le Simulateur Logisim en suivant étape par étape la construction des différents composants, et tous rassembler à la fin. Quoique avant de pouvoir commencer la construction, l'étudiant a besoin de maîtriser les concepts de base de la micro-architecture (*computer organisation* en anglais) en suivant le tutoriel de *Ben Eater* utilisé l'année passée pour construire le processeur SAP-1, néanmoins le processeur pour cette année comme vous allez le voir est d'un degré plus évolué que celui de l'année passée. Les concepts et les méthodes pour les maîtriser la micro-architecture sont décrits dans la section 3 (Méthodes de construction). La section 4 est un guide sous forme de conseils et directives pour aider l'étudiant lors la réalisation du projet. La section 5 définie les clauses et le conditions pour l'accéptation du mini-projet.

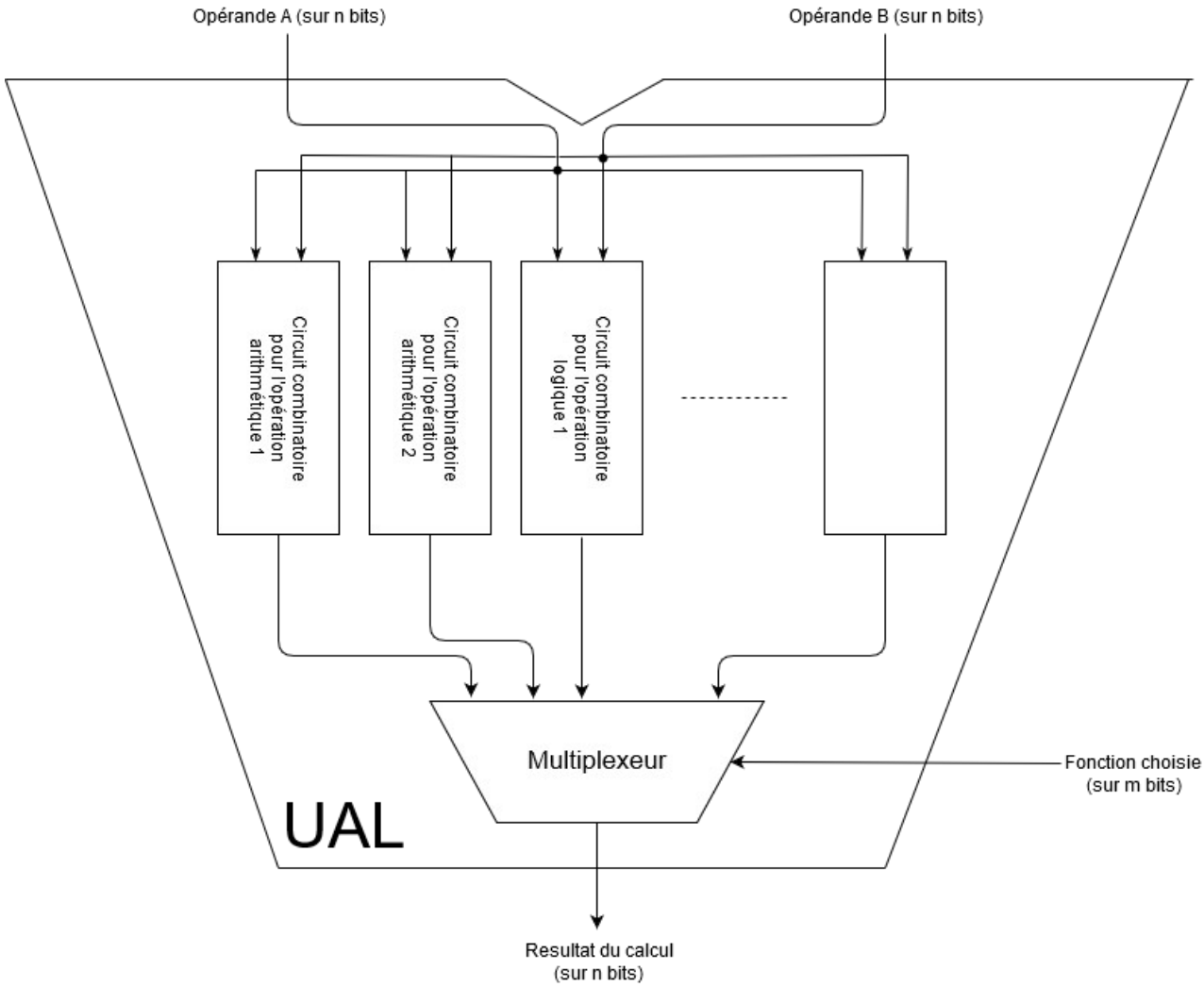
2. Description de l'architecture

La construction de l'architecture se déroule sur trois étapes, la première est celle de UAL (Unité Arithmétique et Logique), la deuxième est celle du datapath (Le chemin de l'instruction), et la troisième c'est UCC (Unité de Commande et de Contrôle). La conception d'un processeur (parce qu'il représente la quasi-totalité de l'architecture) toujours suit ces trois étapes, l'UAL détermine les opération arithmétique et logique que le processeur peut effectuer, le datapath c'est un réseau de chemins dans lequel chaque instruction prend son propre chemin, c'est la métaphore d'un réseau de chemin de fer dans lequel les trains changent le chemin par des aiguilleurs, c'est les Multiplexeurs qui vont faire ce travail d'aiguillage des chemins dans le processeur. L'UCC son rôle est de reconnaître l'instruction lors de sont entrée dans processeur (on appelle ça le décodage de l'instruction) puis de lui tracer le chemin approprié dans le datapath en manœuvrant les Multiplexeurs et les contrôles des différents composants sur le chemin.

2.1. L'UAL

Pour cette année l'UAL est beaucoup plus aboutie que l'année passé, elle va comporter 8 fonctions (au lieu de 2 pour l'année passée) qui sont résumé dans le tableau d'en bas. Pour la réalisation de l'UAL il faut se référer à la figure d'en bas. concrètement l'UAL se présente sous forme de plusieurs circuits combinatoires réalisant pour chacun une opération logique ou arithmétique unique, tous les circuit reçoivent les 2 opérandes A et B sur 8 bits (n=8 pour notre cas) et font tous leurs calcules en parallèle et retournent leurs résultats au Multiplexeur, un seul résultat est choisi par le Multiplexeur parmi les 8 proposés, le choix du Multiplexeur pour laisser passer le résultat d'un circuit précis est pris en rapport avec le nombre encodé en binaire sur m bits (dans notre cas m=3 puisque on a 8 opérations) fourni sur l'entré *Fonctions* du Multiplexeur. Le tableau en dessous résume l'encodage binaire sur 3 bits de l'entrée *Fonctions* de l'UAL. Il est aussi à noter le symbole du Multiplexeur dans le diagramme qui est un losange, et celui de l'UAL qui un losange avec une encoche en haut pour faire la séparation entre les 2 entrées A et B, ces 2 symboles sont souvent utilisés dans la littérature des micro-architectures.

F	F2	F1	F0	Sortie de l'UAL
0	0	0	0	A
1	0	0	1	B
2	0	1	0	B+1
3	0	1	1	A+B
4	1	0	0	A-B
5	1	0	1	not A
6	1	1	0	A and B
7	1	1	1	A or B

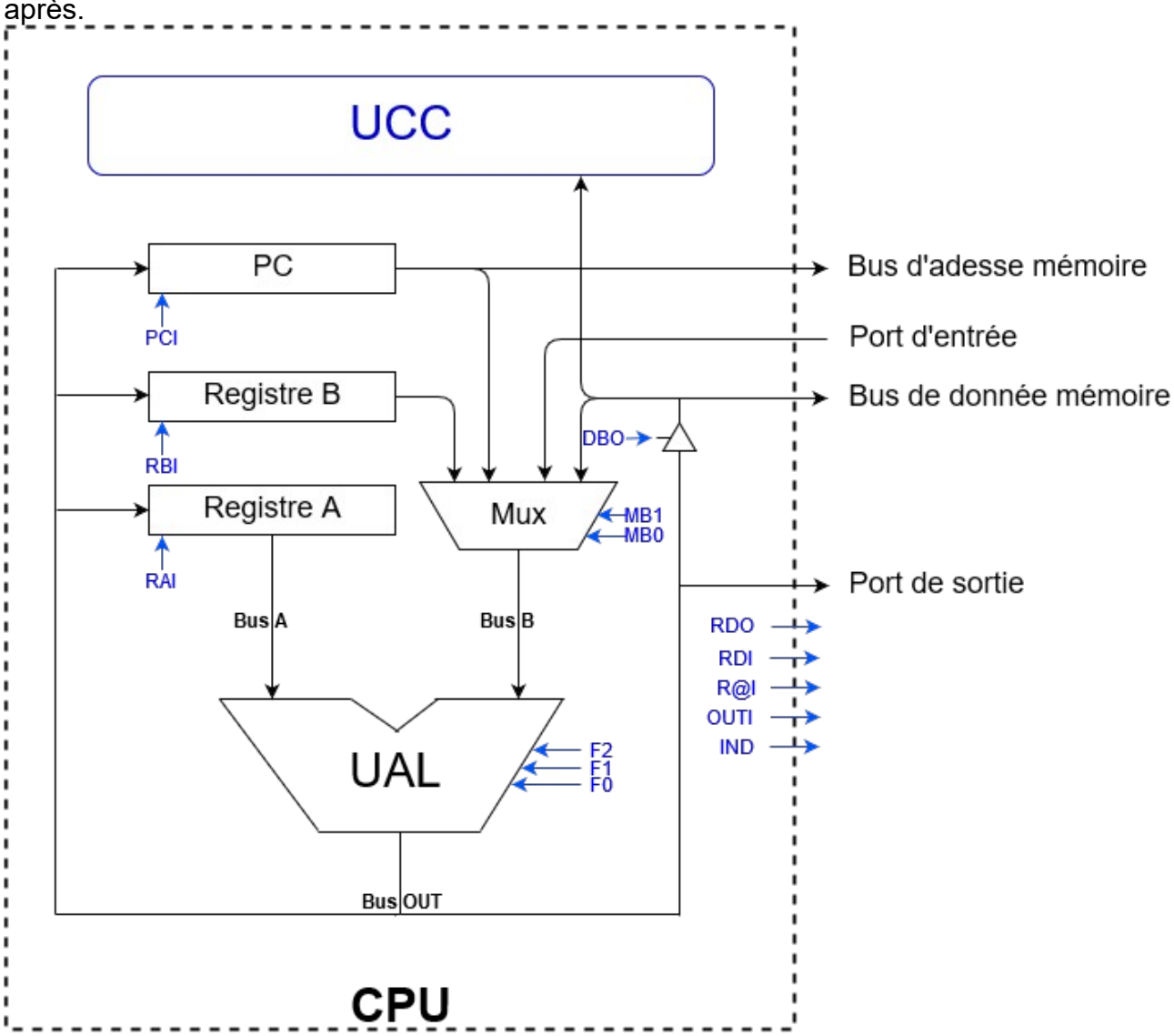


Bien sur il est possible de faire de l'optimisation pour réduire le nombre de portes, Ce n'est pas obligatoire mais vous pouvez vous inspirer de l'UAL du SAP-1 de l'année dernière ou de l'UAL du livre *Digital Design and Computer Architecture* (section 5.2.4 page 248) pour faire de l'optimisation.

2.2. Le Datapath

Comme signalé auparavant le datapath est un réseau interconnecté de chemins, chaque instruction parcours son propres chemin dans ce réseau. Le plus souvent sur un chemin on trouve de la RAM, des registres, des circuits combinatoires (entre autres l'UAL), des Multiplexeurs, des tristate buffers...etc. Le chemin de notre processeur est décrit dans la figure d'en bas. On peut observer que le datapath pour cette année est clairement plus évolué que l'année passé, dans le fait qu'il est plus grand et se compose de plusieurs bus reliant les composants, contrairement au SAP-1 qui se basait principalement sur un unique bus commun pour tous les composants en raison des contraintes évidentes d'implémentation hardware (plusieurs bus aurait exploser sa taille sur les BreadBoards). Sur le diagramme on peut noter le *bus A* et le *bus B* qui se trouvent sur les 2 entrées de l'UAL, le *bus OUT* à la sortie de l'UAL, et plusieurs d'autres qui ne sont pas étiquetés. L'avantage de l'utilisation de plusieurs bus c'est de ne pas être obligé de gérer les sorties des composants avec des tristates buffers pour éviter la contention (la contention c'est lorsque 2 soties ou plus partages le même bus avec 2 signaux différents 0 et 1, ça produit un cours circuit), ça concerne principalement dans ce projet les registres.

Le nombre de registres utilisés à l'intérieur du CPU est 3, Le registre A, le registre B et le registre PC, Le registre A est appelé accumulateur en raison de la façon dont il fonctionne, c'est un registre impliqué dans toutes les opérations arithmétiques/logiques du processeur, il est toujours relié au premier opérande de l'UAL, contrairement au 2-ième opérande qui peut prendre selon l'instruction ; le registre B, le registre PC, le port d'entré ou bien le bus mémoire, de cette façon toutes les instructions arithmétiques/logiques ne mentionnent pas le registre A, pour eux il est implicite, en plus le résultat d'un calcul arithmétique/logique est toujours retourné et sauvegardé dans ce registre d'où le nom *Accumulateur*. Plusieurs processeurs commerciaux utilisent le premier registre comme accumulateur dont intel x86, 6502, motorola 6800, Zilog Z80...etc. Le 2-ième registre est le registre B qui est un registre générique, il peut pour certaines instructions devenir le 2-ième opérande de l'UAL. Le registre PC (contrairement ici au SAP-1 n'est pas un conteur, mais un Registre) contient l'adresse de l'instruction dans la mémoire à exécuter, c'est un pointeur vers l'instruction en cours. En remarque ici le travail d'aiguillage et de sélection du Multiplexeur pour choisir le chemin de l'instruction, les processeurs plus complexe généralement en possèdent plusieurs. L'architecture en entier contient plusieurs autres registres qu'on a pas mentionnés, les registres qu'on viens de voir sont spécifique au CPU, d'autres registres dans la RAM et dans les unités d'Entrées/Sorties serons évoqués après.



Le processeur pour communiquer avec les autres éléments de l'architecture, entre autres la RAM et les unités d'Entrées/Sorties utilise 4 ports. Un port de sortie sur le Bus OUT à destination de l'unité de Sortie, qui est un Afficheur sur 3 chiffres qui sera décrit après. Un port d'entré redirigé vers le Multiplexeur, relié avec l'unité d'Entrée qui sera représentée après par un clavier binaire. Pour la RAM le CPU utilise 2 ports; le ports de sortie pour le bus d'adresse mémoire, et le port à double sens d'entrée/sortie pour le bus de données mémoire. La communication de données entre le CPU et la RAM se fait dans les 2 sens, c'est utilisé comme entrée pour la lecture de la mémoire, et comme sortie pour l'écriture sur la mémoire, c'est un bus Half-Duplex (un sens de communication à la fois). Pour ce bus la gestion du partage des fils pour éviter la contention est obligatoire, que ça soit par le CPU, ou du coté de la RAM, pour le CPU c'est un tristate buffer avec la commande DBO sur le schéma entre le Bus OUT et le bus de donnée de mémoire qui est responsable de la gestion de contention en sortie, pour l'entrée elle est gérer par le Multiplexeur, de même à l'intérieur de l'UCC c'est un autre Multiplexeur qui la gère. Le CPU se fait contrôlé par l'UCC, toutes les flèches de couleur bleu représentent des signaux de contrôle sur différents composants, toutes émisent de l'UCC. 5 signaux (ou fils) sont émis vers l'extérieure, 3 sont destinés pour le contrôle de la RAM, un pour l'unité d'Entrée et un pour l'unité de Sortie. Il est à noter aussi qu'un 6-ième signal appelé signal HLT (Halt : arrêt en anglais) pour arrêter l'horloge qui n'est pas décrit ici, mais qu'il est à prendre en considération pour la réalisation de l'architecture. Les codes binaires de contrôle du chois du Multiplexeur sont décrit dans la table en dessous, les autres signaux serons expliqués dans la section suivante.

Valeurs de sélection du Multiplexeur de l'opérande B	MB1 (bit 1)	MB0 (bit 0)	Les entrées du Mutiplexeur de l'opérande B
0	0	0	Registre B
1	0	1	Registre PC
2	1	0	Port de l'unité d'Entrée
3	1	1	Bus de données Mémoire

2.3. L'Unité de Commande et de Contrôle (UCC)

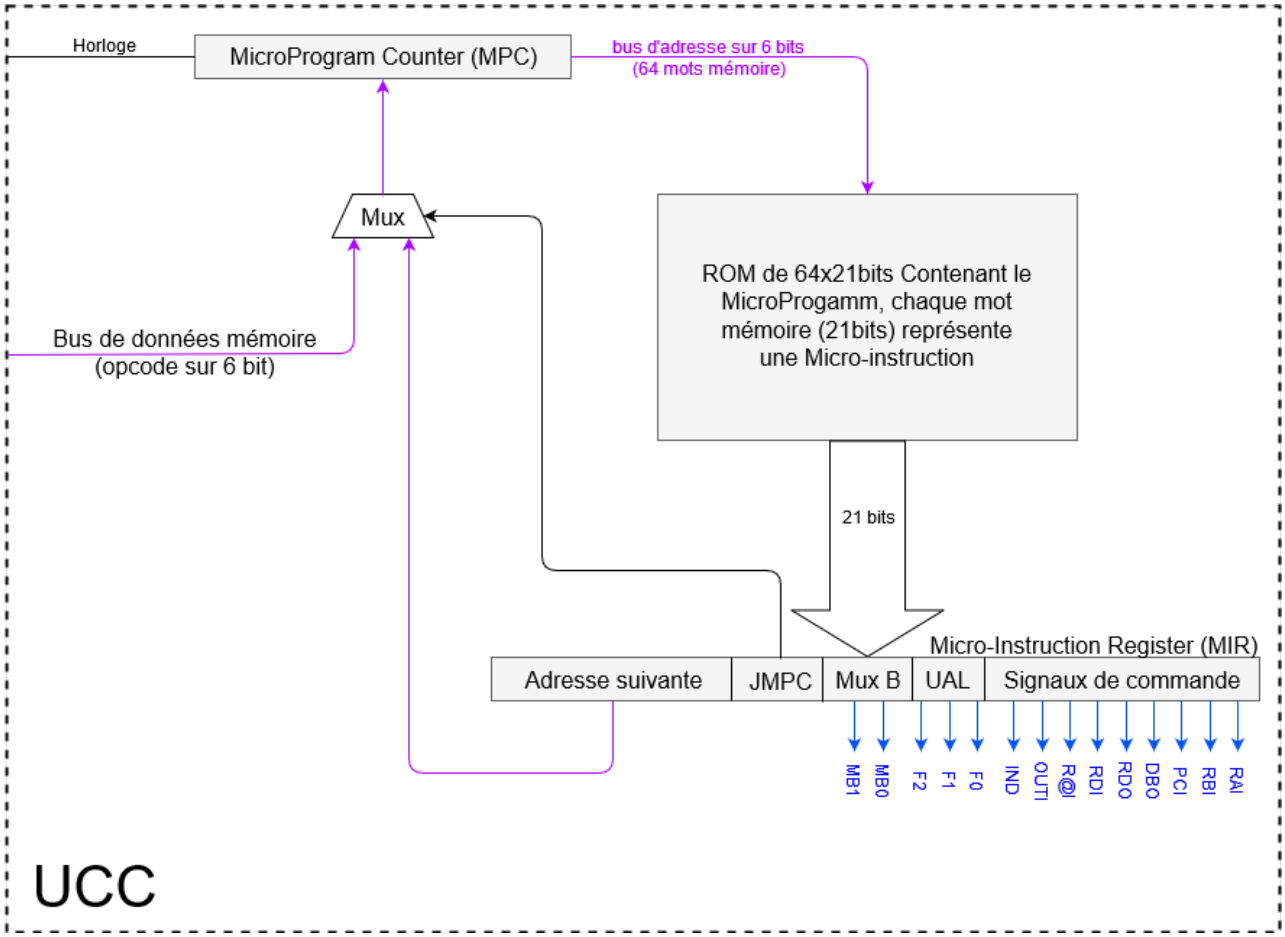
Pour l'UCC encore ici c'est un circuit bien plus sophistiqué que celui du SAP-1, objectivement parlant la technique de la microprogrammation utilisée dans le Mic-1 est grossièrement la même techniques utilisées sur les processeurs réels commerciaux, on peut observer sont utilisation historiquement lors des processeurs de la première génération des micro-ordinateurs, à la fin des années 70 début des années 80 sur des processeurs comme le 6502, le Zilog Z80 et Motorola 6800. On peut aussi l'observer sur les processeurs modernes, ou plutôt ça version dans la section informations de processeur sur les Bios des machines actuelles, généralement sous l'appellation micro-code. Le Tic-1 utilise une technique plus flexible pour modifier ou maître à jour les instructions du processeur, la technique consiste à utiliser l'UCC comme un séquenceur flexible programmable, en sachant que l'UCC sont rôle c'est de décoder l'instruction (précisement la partie opcode de l'instruction dans le langage machine) puis d'orchestrer une séquence de combinaison de signaux de commandes (les flèches bleu sur le diagramme précédent) pour faire avancer l'instruction étape par étape sur son chemin dans le datapath, jusqu'à ce qu'elle soit terminée.

Pour bien comprendre le fonctionnement de l'UCC prenons un exemple d'évolution sur le datapath de l'instruction move A,B qui copie la valeur du registre B dans le registre A. Pour la première étape on a besoin de récupérer l'instruction de la mémoire, ainsi sur la première période de l'horloge la valeur dans PC (Programme Counter contient l'adresse de l'instruction à exécuter dans la RAM) est transmise sur le bus d'adresse mémoire à destination de la RAM, l'UCC entre temps ordonne la RAM de sauvegarder l'adresse dans son registre avec le signal R@I. Lors de l'étape suivante (chaque étape est accomplie sur une seule période d'horloge) L'UCC ordonne à la RAM de faire sortir l'instruction sur le bus de données mémoire en utilisant le signal RDO, ainsi le code machine de l'instruction arrive sur le bus de données mémoire à l'entrée de l'UCC, et ce dernier récupère la partie opcode sur 6 bits de l'instruction. Dans la période suivante l'UCC fait le décodage et la reconnaissance de l'instruction (il reconnaît dans notre cas que c'est l'instruction Move) il envoie ainsi pour la période suivante des commandes au Multiplexeur sur MB1 et MB0 pour laisser passer le registre B, et à l'UAL sur les signaux de F2,F1,F0 pour choisir l'opération 1 qui fait passer l'opérande B sans aucune modification, ainsi à la fin de la période la valeur de B se trouve sur le bus OUT donc à l'entrée du registre A, lors du front montant de l'horloge pour la dernière étape l'UCC envoie la commande au Registre A de lire et de sauvegardé l'information à son entrée par le signal RAI. De cette manière plusieurs étapes successives contribuent à l'exécution de l'instruction. En commençant par l'envoie de l'adresse de l'instruction de PC vers la RAM, de récupérer de la RAM l'opcode de l'instruction par l'UCC, le décodage de l'instruction et les commandes pour faire traverser la valeur de B jusqu'à l'entrée de A, et finalement de sauvegarder A. La traversée d'une partie du chemin par l'instruction est parfois appelée étage processeur, pour chaque période d'horloge le CPU exécute une micro-instruction, l'instruction toute entière prend plusieurs cloques d'horloge, les instructions varies dans le nombre de cloque selon la complexité des uns et des autres, ce type de processeur est appelé processeur multi-cycles. La liste des signaux émis par l'UCC est présentée sur le tableau d'en bas (le tableau ne contient pas les signaux du Multiplexeur ni de l'UAL, qui ont été déjà décrit au préalable).

Signal	Acronyme du signal	Description du signal
RAI	Register A In	Écriture sur le registre A.
RBI	Register B In	Écriture sur le registre B.
PCI	Register PC In	Écriture sur le Program Counter
DBO	Data Bus Out	Sortir l'information sur le bus de données mémoire
RDO	RAM Data Out	Lire l'information de la RAM.
RDI	RAM Data In	Écrire l'information sur la RAM.
R@I	RAM Address In	Écrire sur le registre d'adresses de la RAM.
OUTI	OUTput In	Écrire sur le registre de l'unité de Sortie.
IND	INput Demande	Requête de demande d'informations de l'unité d'Entrée.

L'UCC n'est réellement qu'un simple séquenceur programmable, à chaque cloque d'horloge il exécute une Micro-instruction responsable uniquement de produire une combinaison de signaux de commandes au différents composants du datapath qui fait avancer l'instruction sur une portion de son chemin, la micro-instruction suivante avance

encore plus l'instruction jusqu'à ce que l'instruction se termine. Le schéma du circuit du séquenceur programmable est représenté sur la figure d'en bas, ça se compose de 2 registres, un multiplexeur, et d'une ROM.



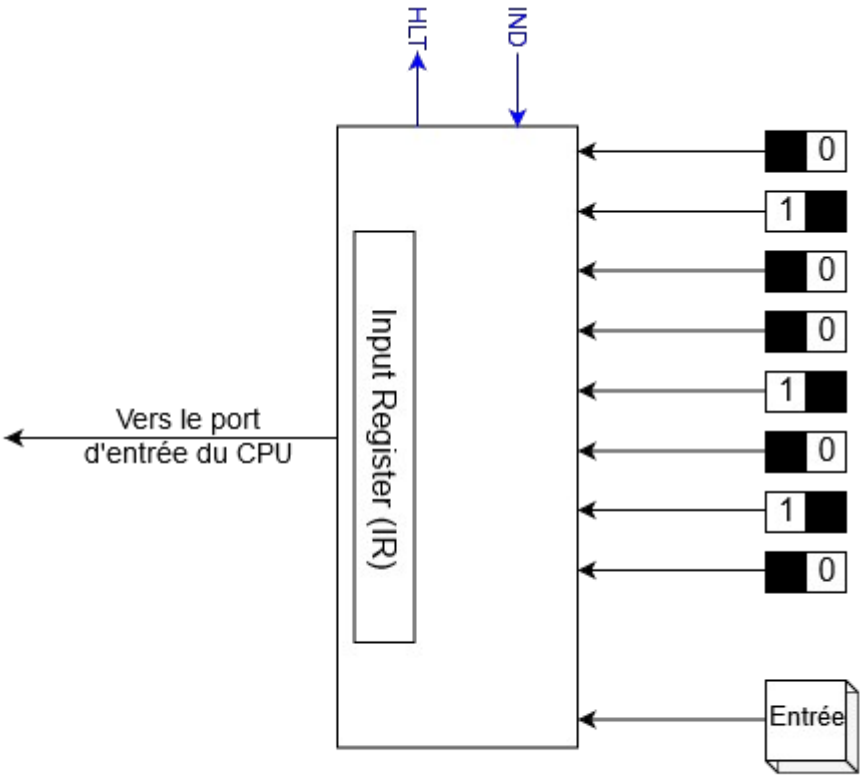
La ROM de taille 64 cellules de 21 bits contient l'image binaire du micro-programme, chaque micro-instruction est retenue dans une seule cellule (ou mot) mémoire, donc pour créer le jeu d'instruction (liste d'instructions) que peut exécuter le processeur il faut transcrire successivement les Micro-instructions pour chaque instruction et former le Micro-code de l'instruction, et de sauvegardé successivement les Micro-codes de toutes les instructions dans la ROM. Le MicroProgram Counter (MPC) est un registre de 6 bits (malgré son nom ce n'est pas un Conteur) qui contient l'adresse de la Micro-instruction actuelle à exécuter, le MPC peut adresser toutes les 64 cellules mémoires de la ROM. La valeur contenue dans MPC est transmise sur le bus violet (tous les bus de couleur violette sont des bus de 6 bits) au port d'Adresses de la ROM, ce qui se traduit par le renvoi par la ROM de la Micro-instruction sur le bus de 21 bits vers le registre Micro-Instruction Register (MIR). Si on observe bien le schéma on voit que la Micro-instruction se compose de 2 principaux parties, les champs; *Mux B*, *UAL* et *Signaux de commandes* sont tous les signaux pour avancer d'un pas l'instruction, alors que la 2-ème partie qui contient les champs *Adresse suivante* et *JMPC* sont responsable du déroulement de l'instruction suivante. le champs *Adresse suivante* sur 6 bits est responsable de fournir l'adresse suivante à exécuter (on peut déduire qu'il n'est pas obligatoire de maîtres les micro-instructions d'une manière successive). Le bit *JMPC* (Jump MicroProgram Counter) est le bit responsable de choisir si l'instruction suivante doit être prise du champ *Adresse*

suivante de MIR ou bien de l'extérieur de l'UCC, donc de l'opcode de l'instruction. Pour réaliser ce mécanisme de choix, le JMPC est directement branché dans le pin de sélection du Multiplexeur. Une importante remarque a dire aussi, c'est que les signaux de commandes sont déphasés d'une demi période d'horloge par rapport (comme pour le SIP-1) à l'horloge des composants du datapath, de cette manière on s'assure qu'ils sont stable lors du front montant de l'horloge pour les composant du datapath. Pour réaliser cette opération en sauvegarde la Micro-instruction dans MIR lors du front descendant de l'horloge.

On peut déduire par observation que le séquenceur est un circuit séquentiel particulier, ou plutôt spécialisé dans le séquencement des micro-instructions, dans lequel le MPC représente le registre d'état, La ROM le circuit combinatoire de sortie, Le Multiplexeur est le circuit combinatoire de l'état suivant, et la boucle avec le retour du champ *Adresse suivante,* et l'entrée extérieur opcode.

2.4. Les unités d'Entrées/Sorties et la Mémoire

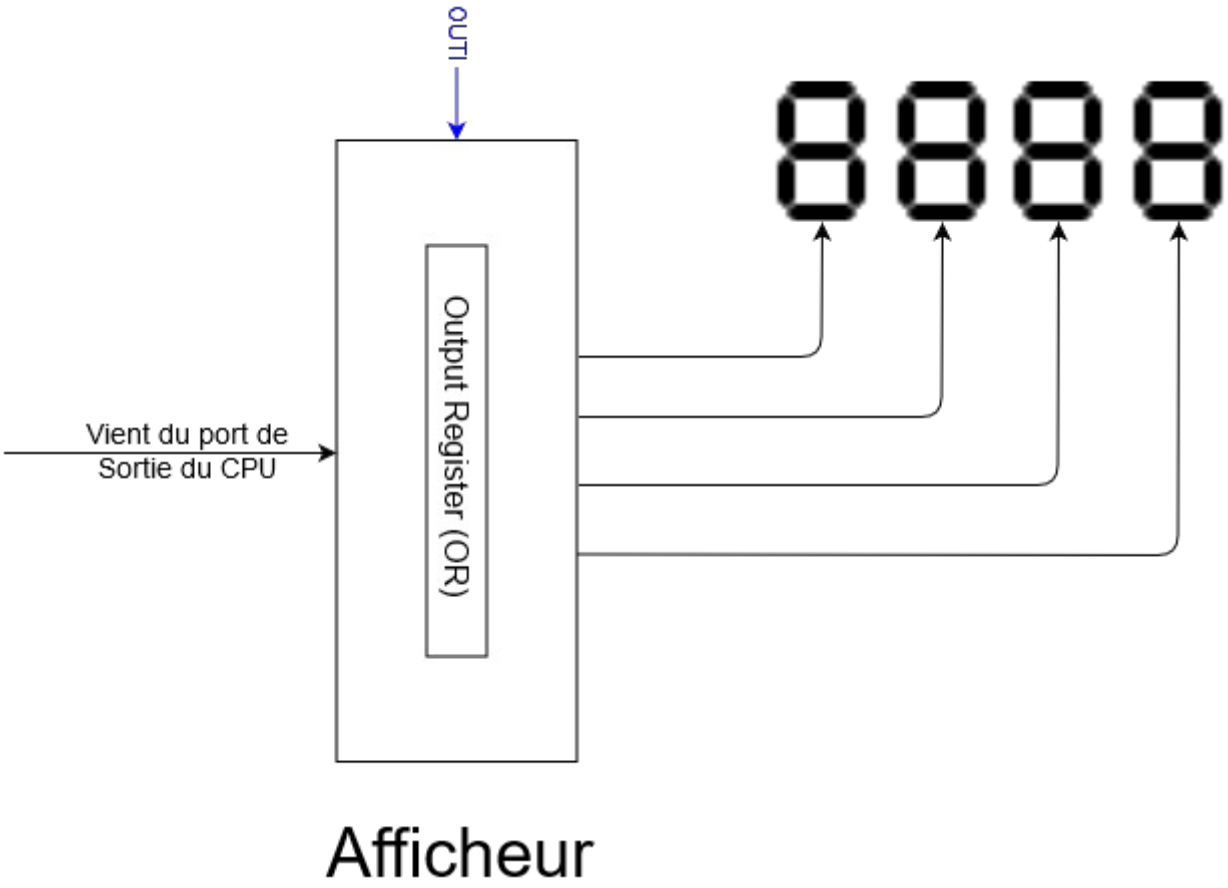
Les unités d'Entrées/Sorties sont relativement basiques, l'unités d'Entrées représentée dans le schéma d'en bas sous forme de clavier binaire, qui avec 8 switch (interrupteurs) peut représenter n'importe quelle valeur sur 8 bits, même les valeurs négatifs, en sachant que notre architecture fonctionne à base de valeurs entières signées complément à 2. Le bouton poussoir *entrée* du clavier permet de valider la valeur, la validation mémorise la valeur des switch dans le registre Input Register (IR). Lorsque le processeur envoie une demande de lecture sur la commende IND, la valeur sera transmise du registre IR sur un bus qui relie le CPU et le clavier vers le port d'entrée du processeur. Après que la lecture du registre soit faite, le registre doit être considéré comme étant vide, dans ce cas l'utilisation d'une flipflop par exemple peut mémoriser le suivi de l'état du registre. Dans le cas où le processeur tente de lire une nouvelle valeur alors que IR est encore vide, le clavier doit arrêter le processeur avec la commende HLT qui devrait être envoyée vers l'horloge pour la faire arrêter (voir le projet SAP-1) le temps que l'utilisateur modifie les switch et valide la nouvelle valeur en appuyant sur la touche entrée de nouveau. Le clavier peut de nouveau relancer l'horloge par la commende HLT et ainsi le processeur pour qu'il termine l'opération de l'écriture. Ce mécanisme qui permet d'arrêter le processeur le temps que la nouvelle valeur soit prête est dans les processeurs réels géré par le mécanisme des *Interruptions*, ces derniers sont assez complexe pour être implémentés dans un mini-projet, donc on utilise cette astuce pour pouvoir faire correctement la lecture de plusieurs valeurs au processeur. C'est au étudiants de réfléchir à une solution pour trouver le moyen de réaliser ce mécanisme.



Clavier Binaire

L'unité de sortie est similaire à celle de l'année passée, elle reçoit la valeur à afficher venant du processeur par un bus reliant le port de Sortie du CPU et l'entrée de *l'afficheur*, comme illustré sur le schéma d'en bas. Sur le schéma on aperçoit aussi le registre Output Register (OR) qui est chargé de sauvegarder la valeur à afficher, l'écriture sur le registre est assurée par le signal OUTI. L'afficheur doit afficher les valeurs positives comme les valeurs négatives encodées sur 8 bits en complément à 2.

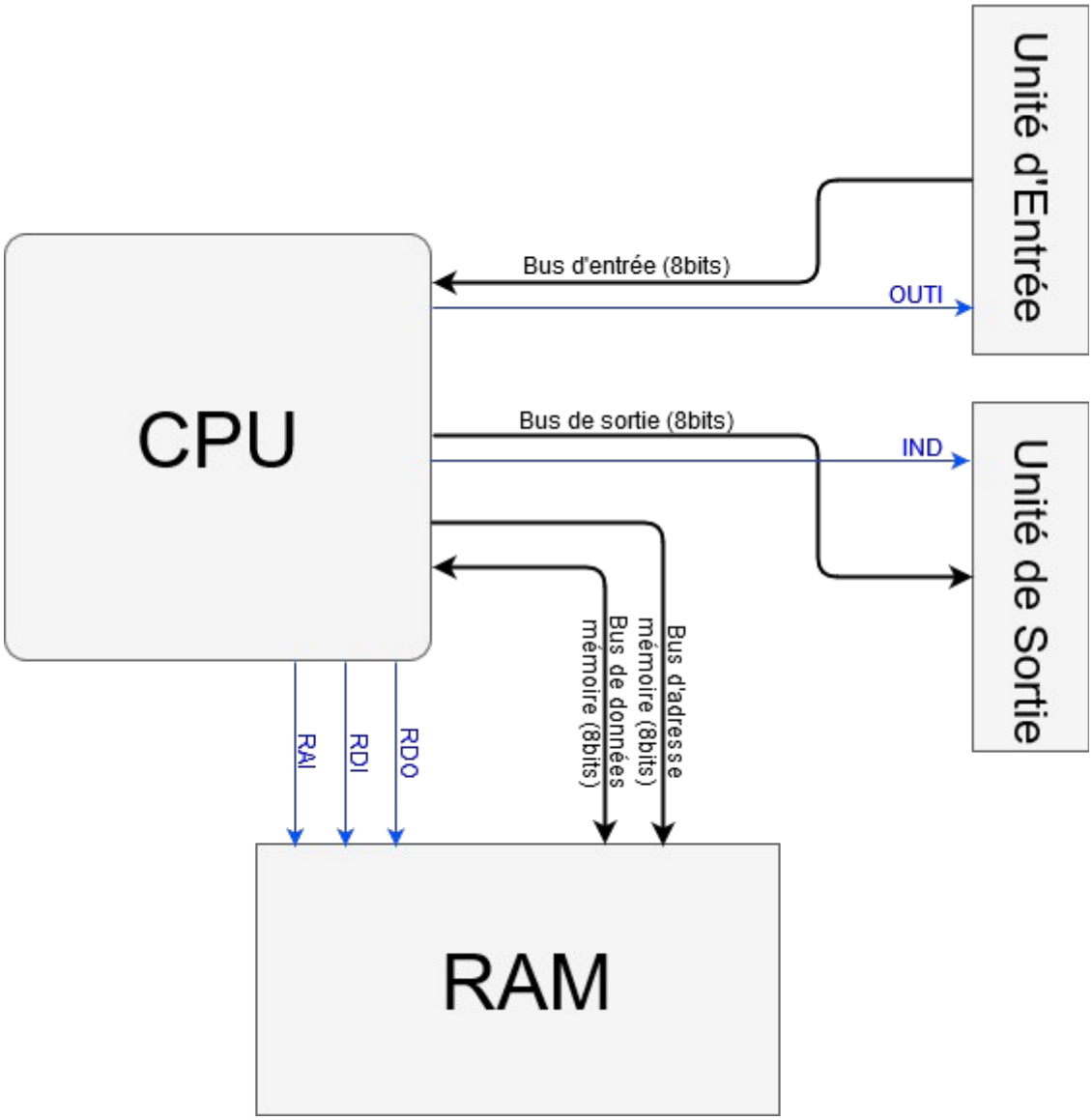
Le circuit de la mémoire est quasiment le même que celui de l'année passée, sauf peut être pour la taille, pour cette année ça sera une RAM de 256 Octets puisque on a un bus mémoire de 8 bits de taille. Le circuit de la mémoire contient une RAM brute, un registre d'adresse pour sauvegarder l'adresse, c'est le *Memory Address Register* (MAR), de la circuitrie pour gérer la contention du bus de données étant un bus bidirectionnel avec le signal DBO, et une circuitrie pour le débogage, c'est la circuitrie qui permet de choisir d'entrer des valeurs à la RAM d'une façon manuelle. Pour faire une opération de lecture ou d'écriture sur la mémoire, le processeur a besoin de 2 cycle d'horloge, le 1-ier fixe l'adresse dans le registre MAR, et le second fait la lecture ou l'écriture à partir de la ARM.



Afficheur

2.5. L'assemblage de tous les éléments de l'architecture

L'assemblage de tous les éléments de l'architecture Mic-1 est présenté sur la figure d'en bas. Les 3 éléments de l'architecture de Von Neumann son représenter ; Le CPU, La RAM et les unités d'Entres/Sorties. C'est 3 éléments sont liés par des bus de données, des bus d'adresses et des bus de contrôles.



2.6. La micro-programmation des instructions

Pour pouvoir faire la micro-Programmation des instructions dans la ROM du séquenceur, il faut tout d'abord comprendre le format et l'organisation des instructions du processeur en langage machine (au format binaire). On a déjà vu dans l'implémentation de l'UCC que l'*opcode* de l'instruction est sur 6 bits, l'opcode (pour *operation code*) réellement représente un encodage binaire pour chaque instructions, ça veut dire que chaque instruction à son propre code, avec 6 bits le nombre maximum d'instructions dans le jeu d'instruction que le processeur peut supporter est 64 (réellement c'est moins, il faut compter les micro-instructions). Néanmoins notre architecture est une architecture sur 8 bits, il est plus simple de prendre 1 octet pour une instruction et de négliger ces 2 bits dans le mot mémoire, il est aussi possible d'utiliser ces 2 bits pour un adressage avec segmentation de mémoire, ça nous permettra d'adresser une RAM de 4 fois plus grande que la RAM actuelle de 256 octets, en l'occurrence une RAM de 1 Ko subdivisée en 4 segments. En plus de l'opcode plusieurs instructions ont besoin d'une opérande en plus pour fonctionner, pour notre architecture quelques instructions aurons un octets en plus après l'opcode, ainsi dans le jeu d'instruction on aura des instructions sur 1 octet et

d'autres sur 2 (opcode + opérande). Dans le domaine de l'architecture, 2 types d'opérandes sont impérativement utilisées; l'immédiate et l'adresse, lorsque l'opérande est une immédiate le processeur utilise directement la valeur retenue dans l'octet de l'immédiate (d'où le nom immédiate), et lorsque l'opérande est une adresse le processeur doit chercher la valeur dans la mémoire pointée par cette adresse.

Comme il a été mentionné auparavant, l'utilisation du séquenceur nous permet avec flexibilité de formuler les instructions sans toucher au datapath et aux composants de l'architecture. Pour cette architecture, vous devez implémenter une liste (la liste d'en bas) d'instructions rudimentaires et fondamentales obligatoires pour pouvoir implémenter des programmes sur l'architecture, d'autres peuvent être rajoutées mais qui ne seront pas obligatoires au bon fonctionnement du processeur, et qu'on pourrait toujours les refaire à partir des instructions fondamentales. Par exemple l'instruction JMP <imm> qui est très utiliser, pourrait être formulée indirectement par LDI <imm> suivie de JMP. Cette manier de programmer peut aussi être faite par le logiciel Assembleur, il peut proposer des instructions qui ne sont pas réellement dans le jeu d'instructions mais sont combinées par d'autres instructions, c'est le cas par exemple de MARS dans le quel ce genres d'instructions est appelé pseudo-instructions. C'est à vous de voir si vous voulez rajouter d'autres instructions non fondamentales au processeur ou non.

Nom	**Forme**	**Description de l'instruction**	**Taille** (Octet)
NOP	NOP	Ne fait rien, aucune opération	1
HALT	HALT	Arrête l'exécution du programme	1
INPUT	INPUT	Lire du clavier et sauvegarder dans A	1
OUTPUT	OUTPUT	Écrire la valeur de A sur l'afficheur	1
LOAD	LOAD <addr>	Charger la valeur de l'adresse de la RAM dans A	2
LDI	LDI <imm>	Charger l'immédiate dans A	2
STR	STR <addr>	Stocker A dans la RAM à l'adresse *addr*	2
JMP	JMP	Sauter vers l'instruction avec l'adresse dans A	1
MOVE	MOVE B	A ← B	1
MOVE	MOVE A	B ← A	1
ADD	ADD	A ← A+B	1
SUB	SUB	A ← A-B	1
AND	AND	A ← A&B (et binaire)	1
OR	OR	A ← A|B (ou binaire)	1
NOT	NOT	A ← ~A (non binaire)	1
INC	INC	A++	1

La micro-programmation d'une instruction ce fait par l'implémentation successive de chaque micro-instruction de l'instruction sur une partie de la ROM du séquenceur. pas obligatoirement d'une manière successivement, puisque le champ Adresse suivante de la micro-instruction peut faire un saut vers n'importe quelle adresse dans la ROM, cette caractéristique peut être utilisée pour une optimisation en sachant que vous pouvez

trouver des instructions qui partage les même parties du micro-codes. Le nombre maximum que peut supporter la ROM est de 64 micro-instructions, c'est la limite aussi pour le nombre d'instructions possible en sachant que chaque instruction contient plusieurs micro-instructions. La ROM doit commencer par ce qu'on appelle *main loop* (la boucle principale) comme illustré sur la table d'en bas. Le rôle de la boucle principale est commun à toutes les instructions, c'est l'étape du *Fetch* (en français : aller chercher) de chaque instruction, son rôle est de récupérer l'instruction de la RAM et faire parvenir son opcode à l'UCC. Chaque ligne dans le tableau représente une micro-instruction, par exemple la main loop comporte 3 micro-instructions, le NOP une micro-instruction, LOAD 5 micro-instruction...etc. Les valeurs des 21 bits de MIR sont indiqués sur les colonnes; *Adresse Suivante*, *JMPC*, *Mux B*, *UAL* et *Signaux de Commande*. La colonne *Adresse Actuelle* représente l'adresse de la cellule mémoire où devrait être sauvegardée la micro-instruction dans la ROM. La colonne *Description* donne une de bref explication du comportement de la micro-instruction.

Instr	Description	Adr Actu	Adr Suiv	JM PC	Mux B		UAL			Signaux de commande								
					MB1	MB0	F2	F1	F0	IND	OUTI	R@I	RDI	RDO	DBO	PCI	RBI	RAI
Fetch (main loop)	MAR←PC	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
	PC++	1	2	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0
	PC←OUT MPC←opcode	2	X	1	0	0	0	0	0	0	0	0	0	1	0	1	0	0
NOP	-,MPC←0	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LOAD	R@←PC	5	6	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
	PC++	6	7	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0
	PC←OUT	7	8	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
	OUT← mem	8	9	0	1	1	0	0	1	0	0	0	0	1	0	0	0	0
	A ← OUT MPC ← 0	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.

Si on fait par exemple l'analyse de l'exécution de l'instruction *main loop.* On aperçoit sur la 1-ière ligne du tableau, la 1-ière micro-instruction, sur laquelle seul le signal R@I est active, il permet au registre MAR de la mémoire centrale de sauvegarder la valeur se trouvant à son entrée, en occurrence le bus d'adresse mémoire qui contient selon le schéma du CPU l'adresse de l'instruction dans PC. En d'autre mots il fait transférer l'adresse se trouvant dans PC vers le registre MAR dans la mémoire centrale. Le champs JMPC contient la valeur 0, si on suit l'exécution sur l'UCC cette valeur indique au Multiplexeur que l'adresse de la micro-instruction suivante à choisir est celle du champs *Adresse Suivante* et non de l'extérieur, donc l'adresse 1 qui est celle de la 2-ème micro-instruction sur le tableau. L'adresse est transférée lors du front montant de l'horloge au registre MPC, qui à son tour pour la période suivante envoie l'adresse à la ROM qui fait

sortir le code de la 2-ième micro-instruction à l'entrée du registre MIR, cette étape est l'étape du décodage de la micro-instruction. Comme signalé au part avant lors du front descendant, MIR sauvegarde la micro-instruction, donc avec un déphasage d'un demi-cycle d'horloge par rapport aux composants de l'architecture pour donner le temps nécessaire aux commandes pour se propager, et d'être présent pour les composants lors du front montant. Les signaux de la 2-ième micro-instruction indique que le Multiplexeur B va choisir de laisser passer le chemin venant du PC (Mux B = 01), et la fonction de l'UAL (010) est B+1 (B c'est le 2-ième opérande de l'UAL et pas le registre B), donc l'adresse dans PC va s'incrémenter de 1 et pointer sur l'instruction suivante dans la RAM, cette adresse est désormais transmise sur le Bus OUT à fin la de cette période. De la même manière indiquée précédemment l'UCC va faire le décodage de la 3-ème micro-instruction dans MIR puisque JMPC dans la 2-ième micro-instruction contient aussi la valeur 0. La 3-ième micro-instruction active les signaux PCI et RDO, le premier permet au PC de sauvegarder la nouvelle adresse incrémentée par la micro-instruction précédente, et le 2-ième indique au circuit de la mémoire de laisser sortir le code de l'instruction sur le bus de données de la mémoire (on rappel que cette instruction est pointée par MAR dans la RAM), ainsi l'opcode sur 6 bits sera à la porté de l'entrée extérieur de l'UCC. En même temps la valeur de JMPC est cette fois-ci 1 pour indiquer à l'UCC de sauter vers la micro-instruction avec comme adresse l'opcode de l'instruction à l'extérieur, et de la même manière quelque soit l'instruction elle sera exécutée de la même façon par l'UCC.

Il est a noter que le champs Adresse Suivante de la 3-ième micro-instruction dans le tableau est marqué par X pour indiquer que sa valeur est sans importance parce qu'elle ne sera jamais prise (JMPC = 1). il est a noter aussi que la dernière micro-instruction de n'importe quelle instruction pointe avec le champs Adresse Suivante sur l'adresse 0 (le début de main loop) comme par exemple pour l'instruction NOP et LOAD sur le tableau, la raison est que le début de n'importe quelle instruction doit toujours commencer avec l'étape du Fetch (ramener l'instruction de la RAM). Le reste du tableau contient le micro-code pour toutes les instruction fondamentales listées auparavant, et l'enrichir à volonté de quelques instructions non fondamentales jusqu'à ce qu'il ne reste plus de mémoire dans la ROM. Il est à noter aussi qu'il y a moyen d'optimisation dans la ROM, en sachant que plusieurs instructions partage le même chemin dans le data-path, C'est surtout le cas des instructions arithmétiques/logiques par exemple.

2.7. Les instructions de branchement conditionnel

Sur le listing des instructions fondamentales du processeur il n'existe pas d'instructions de branchement conditionnel, le branchement conditionnel est un saut (jump) par rapport à une condition, si la condition est satisfaite il y aurait un jump, si non l'exécution continue avec l'instruction suivante. L'ajout du mécanisme du branchement conditionnel augmente grandement les compétences du processeur, ça lui permettrais de faire l'équivalent des branchements (si, sinon, switch) et les boucles(pour, tant-que, répétez) dans un langage évolué, malheureusement l'ajout du micro-code dans la ROM de l'UCC n'est pas suffisant pour implémenter ce genres d'instructions, des modifications dans le data-path sont nécessaires pour pouvoir réaliser les branchements. Pour notre processeur on a besoin d'avoir 2 instructions de branchement fondamentales, qui sont BEQ<imm> (Branch If EQual) qui fait un branchement vers l'instruction avec l'adresse de l'immédiate *imm* si le registre A est égal au registre B, sinon ça continue vers l'instruction suivante, et l'instruction BLT<imm> (Branch if Less Than) qui fait un branchement vers *imm* si A<B. C'est deux instructions puisque ils sont fondamentales, en composition avec

d'autres instructions nous permettent de faire n'importe quelle type de branchement conditionnel (réellement seule BLT est suffisante). Vous avez à implémenter ces 2 instructions.

Pour implémenter ces 2 instructions on a besoin de modifier l'UAL pour qu'elle rajoute une 2-ième sortie sur un bit pour indiquer si le résultat de l'opération est égal à zéro ou non, cette information est primordiale pour l'instruction BEQ. Il faut aussi extraire une sortie du bit de signe à partir du Bus OUT, qui va nous permettre de déduire que si on fait une soustraction entre A et B et que le résultat est négatif, ça implique que A<B, donc cette information sera utilisée par l'instruction BLT. Il faut aussi rajouter de la circuitrie dans le data-path et du micro-code en conséquence dans la ROM de l'UCC. L'UCC ne doit pas être modifier. Vous pouvez vous inspirer de l'implémentation de l'instruction de branchement conditionnel dans le chapitre 7 du livre *Digital Design and Computer architecture* (page 381).

3. Méthodes de construction

Pour pouvoir construire une architecture il faut maîtriser les concepts de la micro-architecture (Computer Organisation en englais). Pour se faire on va s'appuyer sur le projet de l'année passée qui se base sur un tutoriel vidéo proposant la construction d'une architecture présentée sur le site web ou la chaine Youtube de l'académicien nommé Ben Eater (vous pouvez le trouver en cherchant sur google). L'architecture réalisée par Ben Eater est inspirée de l'architecture SAP-1 (Simple As Possible 1) du livre **Digital Computer Electronic** d'*Albert Paul Malvino*. Sur la chaine Youtube, l'architecture et sa construction sont expliquées a travers une playlist de 42 vidéos qui détaillent pas à pas les différentes étapes de construction en commençant par la construction des modules de l'architecture d'une façon isolé et indépendante de l'architecture globale, ensuite de faire l'assemblage de ces modules et former une architecture globale et complète, ensuite à la fin de faire de la programmation sur celle-ci en écrivant des programmes basiques pour tester l'architecture et son langage machine. Le tutoriel vise à faire une construction physique électronique de l'architecture, notre but de faire une simulation. L'année passé une solution de l'architecture sur Logisim a été proposée sur le depot github, que vous pouvez utiliser comme source d'inspiration, ou même de prendre des éléments et les réutiliser pour le projet de cette année.

Néanmoins l'utilisation du tutoriel de Ben Eater demande quelques explications et remarques à prendre en considération. On peut les liste comme suite :

1. Il faut savoir que l'explication de l'architecture touche 2 aspects de l'électronique, l'électronique analogique et l'électronique numérique. L'électronique numérique représente la majeure partie du projet et vous avez le devoir de bien maîtriser cette partie qui entre dans votre apprentissage. Par contre l'électronique analogique, vous avez déjà le niveau académique de savoir c'est quoi le voltage, l'ampérage, une alimentation, une résistance, un condensateur et leurs fonctionnement, par contre vous ignorer le fonctionnement d'un transistor en mode switch, le comparateur, et le circuit RC, ainsi que les résistances pull-up et pull-down, dans ce cas vous n'êtes pas obligé de tout comprendre, quoique avec un peu d'effort et de recherche il vous serait assez facile de bien suivre et tout comprendre.

2. Lors de la réalisation sur simulateur, il n'est souvent pas question d'implémenter les parties analogiques, il suffit juste de comprendre le fonctionnement et de trouver une méthode pour la faire fonctionner sur le simulateur, par exemple le circuit 555 générateur d'horloge contient plusieurs éléments analogiques que vous pouvez simplement supplanter par une simple horloge dans le simulateur, malgré qu'il est très bénéfique de comprendre son fonctionnement.

3. Le transistor en mode switch dans la vidéo 3 joue le rôle d'un simple interrupteur, qui est activé ou désactivé (ouvert ou fermé) par un signal sur le troisième Pin, pour comprendre plus son fonctionnement vous pouvez faire une simulation sur Logisim, quoique il n a pas d'utilisation réel dans la simulation de notre projet.

4. Le circuit RC (pour Résistance Condensateur) que vous allez le retrouver sur la vidéo 8 et 31 est un circuit qui permet de générer une impulsion électrique de très bref délai (de quelque milliseconde), ça peut être bénéfique pour certaines situations. À noter qu'il n'est pas implémentable sur le simulateur Logisim.

5. Les résistances pull-down et pull-up doivent être implémentées sur le simulateur (vous les trouverez dans le répertoire Wiring). Elles sont utilisées tout au long du projet mais elles sont bien décrites dans la vidéo 34. En bref ces résistances sont des résistances directement branchées sur le GND = 0V (pour la pull-down) ou +5V (pour la pull-up), leurs rôle est de fournir un 0 logique faible ou un 1 logique faible. Ils sont souvent utilisés avec des bus, ou des fils susceptibles de porter la valeur logique Z (valeur flottante), et c'est justement le cas des bus lorsqu'ils sont libres et pas utilisés. La raison d'utiliser des 0 et 1 faibles au lieu de Z c'est qu'il y a des composants qui ne supportent pas une valeur flottante en entrée et présentent un comportement imprévisible ou erroné. Dans ce cas la présence d'une résistance pull-down ou pull-up pousse a superposé la valeur flottante par la valeur logique 0 ou 1 faible, et lorsque sur un fil la valeur est faible elle ne présente pas de danger de conflit (contention) avec une valeur logique normale, car elle laisse place à cette valeur dans le cas où ça devrait y arriver.

4. Conseils et directives

Cette section présente quelques conseils et directives à suivre tout au long du déroulement du projet, qui peuvent vous éclairer, vous aider et vous faciliter le parcours de ce genre de projet :

1. La première chose à prendre en compte, c'est que ce projet par nature est complexe et relativement difficile, qui devrait exiger de l'étudiant beaucoup d'investissement dans son temps et son énergie. C'est pour cette raison que le projet n'est pas obligatoire, et que même seulement les 10 premiers étudiants à le rendre serons acceptés, pour ne pas faire sentir à tout le monde la nécessiter de devoir le faire. Si vous pensez vouloir se lancer dans le projet, il faut prendre en considération l'aspect du temps et de l'énergie que vous devriez investir.

2. Malgré le côté long et fastidieux du projet, le bénéfice acquis en apprentissage est énorme, ça vaudrait le coup de le tenter. même si le simple fait de visionner les vidéos et tester la simulation de l'année derrière et de cette année (qui sera publiée à la fin du délai) reste pédagogiquement très bénéfique.

3. Pour la maîtrise et le contrôle de la complexité de l'architecture, la réalisation sur simulateur demande une utilisation judicieuse de la modularité, ainsi l'utilisation des mécanismes de bibliothèque ou des subcircuits (voir la documentation de Logisim) permettent de ne pas créer plusieurs fois le même composant, en plus de rendre le projet plus organisé et plus facile à construire. Vous pouvez vous inspirer de la réalisation de l'année dernière, et même d'en utiliser des parties s'il conviennent au projet.

4. L'utilisation des composants du simulateur tel que les Splitters et les Tunnels (qui se trouvent dans le répertoire Wiring) permettent de diminuer encore plus la complexité du projet. La première permet de rassembler plusieurs fils sur un seul fil (de couleur noire). La deuxième permet de créer une liaison filaire entre deux points distants sans pour autant dessiner le fil, qui serait dans le cas contraire une opération fastidieuse.

5. Le projet a une vocation de recherche académique personnelle, les principaux références pour le projet sont; les 42 vidéos de Ben Eater, le livre *Digital Design and Computer Architecture 2nd edition* de *David Harris*, le papier *Step-by-Step Design and Simulation of a simple CPU Architecture* de *D.C. Schuurman*, Le guide utilisateur de Logisim (menu Help >User's guide), youtube, et le moteur google. Si vous avez fait le tour complet sur une question ou un point qui vous pose problème, vous pouvez venir me voir et demander conseil, et c'est aussi valable pour la partie électronique analogique du projet.

6. Le projet sur simulateur est relativement complexe et il existe plusieurs façons de l'implémenter, c'est pour cette raison qu'il est pratiquement impossible d'avoir des solutions strictement identiques. Le projet est destiné a être fait individuellement, si d'après jugement personnel je remarque un degré de similitude entre 2 ou plusieurs projets la note sera amputé en rapport avec le degré de similitude de ces projets. Le projet aussi doit suivre la description donnée dans la section 2 et les directives dans cette section, une pénalité aussi applicable sur les réalisation qui dérive de l'énoncé du mini-projet.

7. Il est impératif d'implémenter les circuits combinatoires de l'additionneur, du multiplexeur et du décodeur (le multiplexeur et le décodeur sont expliqués dans le livre *Digital Design and Computer Architecture* au chapitre II section 2.8.1 et 2.8.2 respectivement) à partir de portes logiques et de ne pas les utiliser directement à partir des bibliothèques standard du simulateur. Il est de même pour les registres et les compteurs où il faut les construire à partir de flipflops.

8. Par contre les RAMs et les ROMs il est préférable d'utiliser ceux fournies par le simulateur que de les créer soi-même en raison de leurs complexités. Les concepts de RAM et ROM sont bien expliqués dans le livre *Digital Design and Computer Architecture* au chapitre 5 section 5.5.

9. Dans le projet et comme expliqué dans la vidéo 31 la ROM est utilisée comme une méthode alternative pour implémenter les circuits combinatoires, cette méthode est basée sur le fait que les entrées dans la table de vérité d'un circuit combinatoire représentent la porte l'adresse dans la ROM et les sorties représentent l'information à la sortie de la ROM, en résumé la table de vérité est l'adresse/information à

l'intérieur d'une ROM. Pour savoir comment utiliser la RAM et la ROM et aussi les Splitteurs et les Tunnels, il faut s'appuyer sur le guide utilisateur de Logisim.

10. Sur la vidéo 33 l'afficheur est implémenté en utilisant une ROM (ou EEPROM) avec une deuxième horloge à fréquence très élevée pour afficher 3 chiffres en même temps. Cette méthode est impossible a faire avec le simulateur parce qu'il n'accepte qu'une seule fréquence d'horloge, de ce fait vous avez a implémenter la première méthode suggérée dans la vidéo, c'est à dire d'utiliser 3 ROM différentes, une pour chaque chiffre, plus une ROM dans notre cas pour afficher le signe moins des nombres négatifs.

4. Clauses et conditions

Le mini-projet doit suivre les causes et conditions suivante pour qu'il soit accepté :

1. L'implémentation du projet doit se faire en monôme, sauf autres indications de votre enseignant de TP.

2. La réalisation du mini-projet vous octroie une note bonus de 15 points additionnels à votre note TP, sauf autres indications de votre enseignant de TP.

3. La date butoir pour la remise des mini-projets est le 31 décembre à 23 heurs 59 minutes.

4. Seuls les 10 premières solutions correctes serons acceptées.

5. La condition pour que le mini-projet soit accepté est qu'il puisse exécuter le programme factoriel et le programme de la suite de Fibonacci avec des valeurs entrées par l'utilisateur. Les 2 programmes doivent être inclus avec les fichiers du projet.

6. La solution doit contenir tous les fichiers .circ de votre projet, plus les fichiers des mémoires ROM, plus les 2 programme en langage machine (les images mémoire de la RAM), plus les 2 programme écrit en assembleur dans un fichier texte.

7. Tous les fichiers doivent être regroupés sur un même fichier compressé et envoyés par émail en fichier joint à l'adresse *ado.mini.projet@gmail.com*.

وفقكم الله