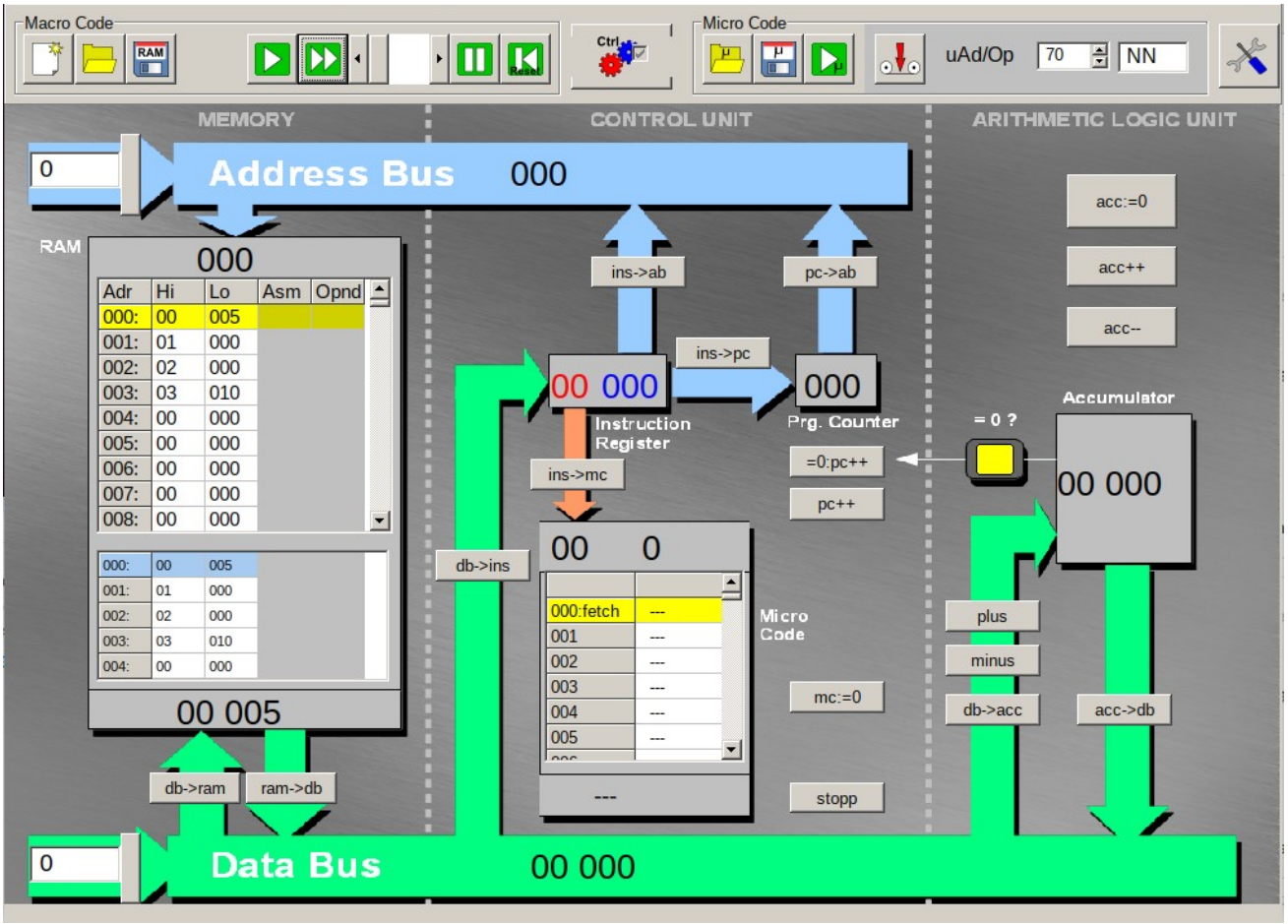
Compte Rendu: TP Architecture des ordinateurs

Séance 1.1 : Système minimum Johnny

Après avoir parcouru la documentation du logiciel Jhonny afin de connaître les codes permettant de faire les différentes instructions, et de comprendre le fonctionnement de l'instruction TST, j'ai assemblé les instructions me permettant de faire l'addition.

J'ai ensuite réalisé la multiplication par 2 (image ci dessous).



J’ai enfin réalisé un programme permettant de calculer un nieme carré d’un nombre (positionné à l’adresse 017) .

Adr	Hi	Lo	Asm
000	09	019	NULL
001	01	017	TAKE
002	04	019	SAVE
003	01	019	TAKE
004	02	019	ADD
005	04	019	SAVE
006	08	018	DEC
007	06	018	TST
008	05	003	JMP

Pour cela j’utilise l’instruction TST qui me permet, avec l’aide de l’instruction JMP, de pouvoir enchainer plusieurs séquences de code (ici la même) afin de terminer la boucle quand le nombre d’incrémentation de puissance est terminé.

Little man computer :
Le mode de fonctionnement du LMC est le même que celui de Jhonny, à part qu’une input est requise à l’utilisateur. Cependant les animations permettent demieux comprendre le fonctionnement de l’architecture.

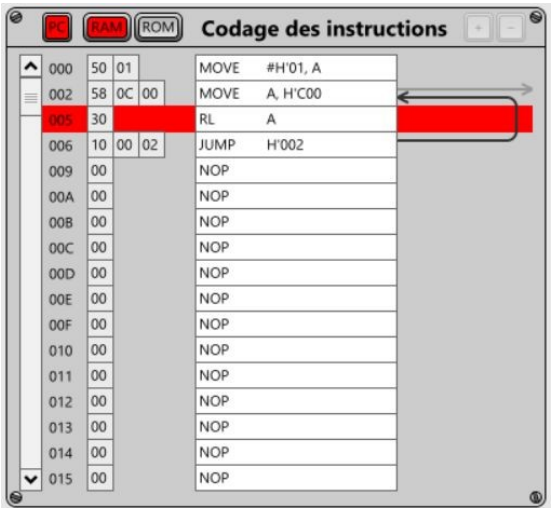
Après avoir assembler et implanter le programme ci contre en mémoire, je me suis aperçu qu’il nous renvoie le code en décimal des différentes instructions , ainsi que le nombre d’opérateurs restant à traduire (au tout début).

```
LOAD LDA 0
OUT
SUB ONE
BRZ ONE
LDA LOAD
ADD ONE
STA LOAD
BRA LOAD
ONE  DAT 1 // directive d’ALIAS
```

Séance 2.1 Simulateur Dauphin

L'assembleur permet de s'éviter de devoir assembler à la main, et par conséquent permet un gain de temps lors du développement car le code s'écrit à l'aide de commande à base de mots, et non plus de codes, ce qui le rend plus compréhensible.
On peut comparer cela dans les images ci dessous :

```
        MOVE  #1, A
LOOP:   MOVE  A, _DIGIT0
        RL    A
        JUMP  LOOP
```



Utilisation de la pile :
Voici un programme simple appelant un sous-programme simple :

```
MOV #6,A
CALL DOUBLE
DOUBLE:
ADD A,A
RET
```

on observe l'utilisation de la pile par le micro-processeur, lorsque la commande « CALL DOUBLE » fait appel au sous-programme DOUBLE.
On observe que SP, le pointeur de pile, pointe dans un premier temps vers l'adresse du premier programme. Après la commande « CALL », le pointeur de pile pointe vers l'adresse où se trouve notre sous-programme DOUBLE.

J'ai codé le programme récursif suivant :

```
MOVE #1,A
MOVE #8,B
CALL DOUBLE
HALT
DOUBLE:
ADD A,A
COMP B,A
JUMP EQ DONE
CALL DOUBLE
DONE :
RET
```

Après la commande « CALL », le pointeur de pile pointe vers l'adresse où se trouve notre sous-programme DOUBLE. Celui-ci étant récursif, on se rend compte qu'à chaque appel, le pointeur de pile décrémente,de manière à ce chaque appel de DOUBLE est traité comme un sous programme différent.

Séance 3.1 : Processeur ARMv4

Simulateur de Chronomètre :

A partir du code expliqué fourni pour la simulation du chronomètre, j'ai compris qe l'on stock sur 4 adresses le temps dans différentes unités (millisecondes, secondes, minutes et heures), et que lorsque la quantité d'une unité atteint le seuil de passer à une unité supérieure (1000ms passe en 1sec par exemple) alors on incrémente la dite unité.

Je me suis inspiré de cela pour essayer de faire un compteur à la vitesse de l'interruption, j'ai cependant eu du mal à aboutir.

```
SECTION INTVEC

B main
NOP
NOP
NOP
NOP
NOP
NOP
NOP
LDR PC, =ajout
first ASSIGN32 0
second ASSIGN32 0

SECTION CODE

ajout

ADD R10, R10, #1
CMP R10, #1000
STM R9, {R9}
SUBS PC, LR, #4

main

ADD R9, R9, #1
Bmain
STM R9, {R9}

SECTION DATA
```

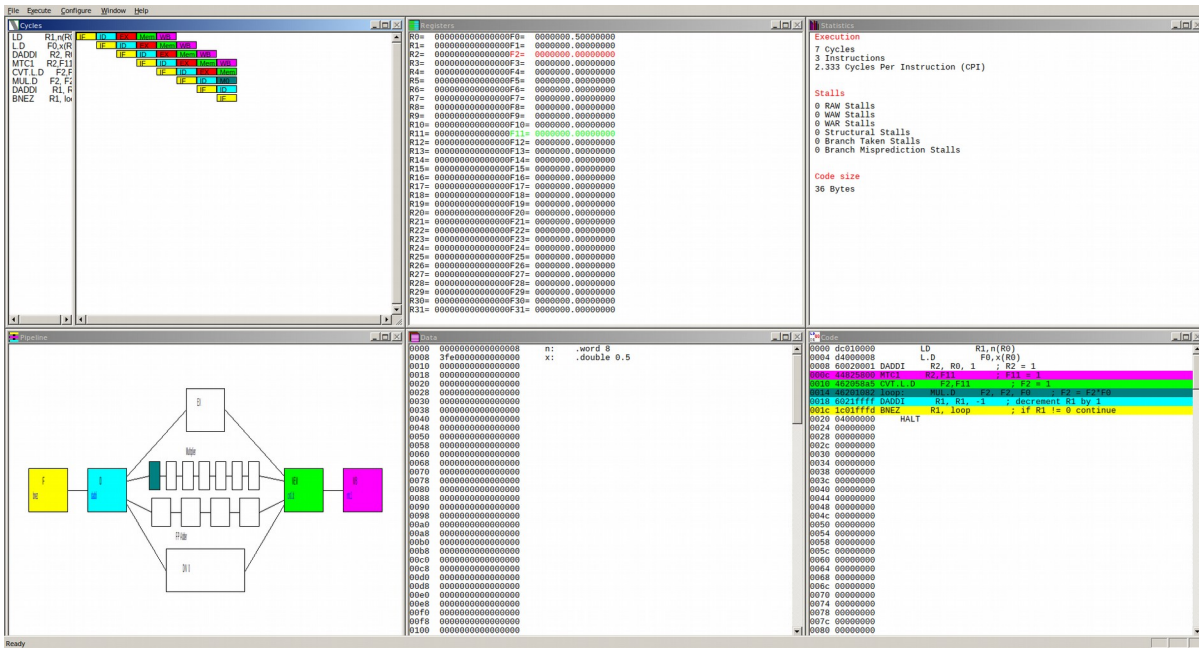
Séance 3.2 : Processeur 8085, séquenceur interne et interface :

Observation des chronogrammes de l'interface:

on choisit le code « 8 bit decimal substraction ».
Les Différentes épates sont :
-Le stockage de la valeur de l'accumulateur dans un registre.
- on soustrait la valeur de l'accumulateur à celle de la mémoire.
-on affecte la valeur en mémoire à celle de l'accumulateur.
-on incrémente l'accumulateur.
-on ajoute la valeur du registre L à celle de la mémoire.
-enfin on décrémente le registre L.

Séance 3.4 : Simulation d'un pipeline

On créé un fichier et on copie colle la fonction de x^n.



On va décrémenter le double de l'adresse R1, jusqu'à ce qu'il soit nul.
A chaque fois qu'on le décrémente, on multipliera F2 par F0, F2 étant la variable qui stocke le résultat temporaire et qui à la fin de la boucle contient le résultat final.