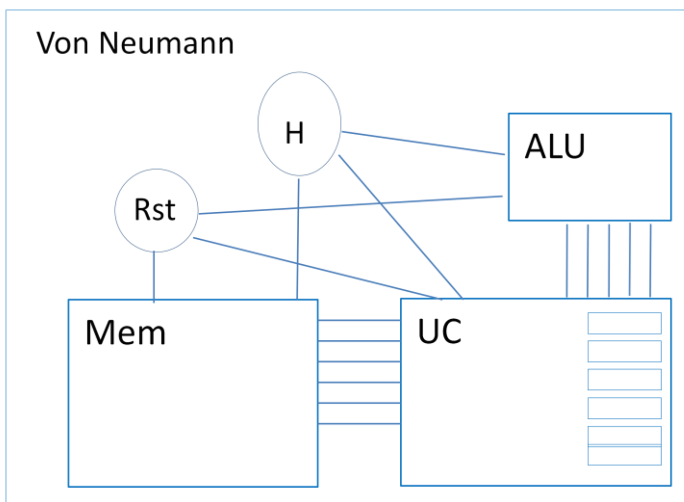
TP n°3 – Machine de Von Neumann
Partie1 – Fetch-Decode

Le but de ce TP est de construire en VHDL une machine de Von Neumann qui parcourt un programme chargé en mémoire principale et le décode. Nous nous intéresserons dans le TP suivant à l'exécution de ce programme. Une archive avec les composants de base est disponible sur AREL : **memory-student.tgz**

Architecture

Le TP reprend la machine de Von Neumann 4 bits présentée en cours (composant VHDL **von_neumann**) :



- Mémoire (composant VHDL **memory_with_cache**) de 2 Kmots de 4 bits avec temps de réponse variable (cache simulé). Le signal ready permet de savoir si le composant est prêt et/ou si le résultat est disponible.
- L'Unité Centrale (composant VHDL **cpu** à compléter) :
  - 4 registres 4 bits R_0 à R_3
  - 1 registre d'instruction RI 16 bits
  - 1 compteur ordinal PC 12 bits
  - 1 registre de donnée MDR 4 bits
  - 1 registre d'adresse MAR 12 bits
- L'ALU pour les calculs (cf TP suivant).

Langage Assembleur

Similaire à un petit Z80 avec 3 familles d'instructions :

1 – Les déplacements mémoires
- Mémoire vers registre
- Registre vers registre
- Regitre vers mémoire

2 – Les instructions de sauts

Utilisation des flags (C et Z) de l'ALU et sauts relatifs (sur 3 mots) du compteur ordinal (PC)

3 – Les opérations arithmétiques : utilisation des opérations correspondantes de l'ALU

La taille des instructions est variable de 1 à 4 mots : | CODE_OP | PARAMETRES |

- **CODE_OP** : code de l'instruction, taille variable 2 à 8 bits
- **PARAMETRES** : opérandes de l'instruction, taille variable 0, 2, 4 ou 12 bits

Liste des instructions :

Binaire	Mnémonique	Commentaire	
0000	**LD R_0 ($R_0R_1R_2$)**	MAR <- $R_0R_1R_2$; Lecture ; R_0 <- MDR Lecture du mot contenu à l'@ $R_0R_1R_2$ en mémoire puis rangé dans R_0	
$00j_1j_0$	**LD R_0** R_j	R_0 <- R_j, j= 1,2 ou 3	Copie du registre R_j vers R_0
0100 $v_3v_2v_1v_0$	**LD R_0 V**	R_0 <- V	Copie de la valeur V (4 bits) vers R_0
$01j_1j_0$	**LD R_j R_0**	R_j <- R_0	Copie du registre R_0 vers R_j
1111 1111	**LD ($R_0R_1R_2$) R_3**	MAR <- $R_0R_1R_2$; MDR <- R_3 ; Ecriture Ecriture du mot contenu dans R_3 dans la mémoire à l'@ $R_0R_1R_2$	
1000 $d_{11}...d_0$	**JRC d**	si FLAG_C = 1, PC <- PC + d	
1001 $d_{11}...d_0$	**JRNC d**	si FLAG_C = 0, PC <- PC + d	
1010 $d_{11}...d_0$	**JRZ d**	si FLAG_Z = 1, PC <- PC + d	
1011 $d_{11}...d_0$	**JRNZ d**	si FLAG_Z = 0, PC <- PC + d	
1100 $d_{11}...d_0$	**JR d**	PC <- PC + d Saut inconditionnel	Rq : les déplacements sont relatifs de -2^{11} à $2^{11}-1$
1101 $d_{11}...d_0$	**DJNZ d**	R_3--; MINUS R_3 0; JRNZ d décrémente R3 puis saut tant que R3 > 0	
1110 $00j_1j_0$	**ADD R_0 R_j**	R_0 <- R_0 + R_j	
1110 $01j_1j_0$	**ADC R_0 R_j**	R_0 <- R_0 + R_j + FLAG_C	
1110 $10j_1j_0$	**MINUS R_0 R_j**	R_0 <- R_0 - R_j	
1110 $11j_1j_0$	**MULT R_0 R_j**	R_0 <- R_0 * R_j	
1111 $00j_1j_0$	**DIV R_0 R_j**	R_0 <- R_0 / R_j	
1111 $01j_1j_0$	**MOD R_0 R_j**	R_0 <- R_0 % R_j	
1111 $10j_1j_0$	**INC R_j**	R_j++	

Rq : il reste 3 codes disponibles 1111 1100 à 1111 1110

CPU

On s'intéresse aux étapes FETCH et DECODE de la cpu. Le package **pack_cpu** vous donne les types et fonctions utiles au décodage des instructions au fur et à mesure de la lecture des mots du programme dans la mémoire.

L'algorithme est le suivant :

- Tant que vrai faire
  - lire le mot à l'@ PC et le mettre dans RI(15..12)
  - incrémenter PC
  - décoder ce 1er mot (fonction decode_1st_word)
  - si l'opcode ne dépasse pas un mot (fonction is_2word_operation)
    - alors
      - incrémenter PC du nombre de mots restant à lire de l'instruction (fonction nb_word_to_read1)
    - sinon
      - lire le 2ème mot à l'@PC et le mettre dans RI(11..8)
      - incrémenter PC
      - décoder l'instruction (fonction decode_2nd_word)
- Fin Tant Que

<u>**Travail à réaliser :**</u> compléter le composant **cpu** pour réaliser ces étapes de fetch et décode. Faire une capture du chronogramme obtenu ou vous mettrez en évidence les instructions décodées.

<u>NB :</u> la mémoire principale est chargée initialement avec un programme calculant la factorielle de 3.