Architecture Avancée
SLE/ISI
Examen

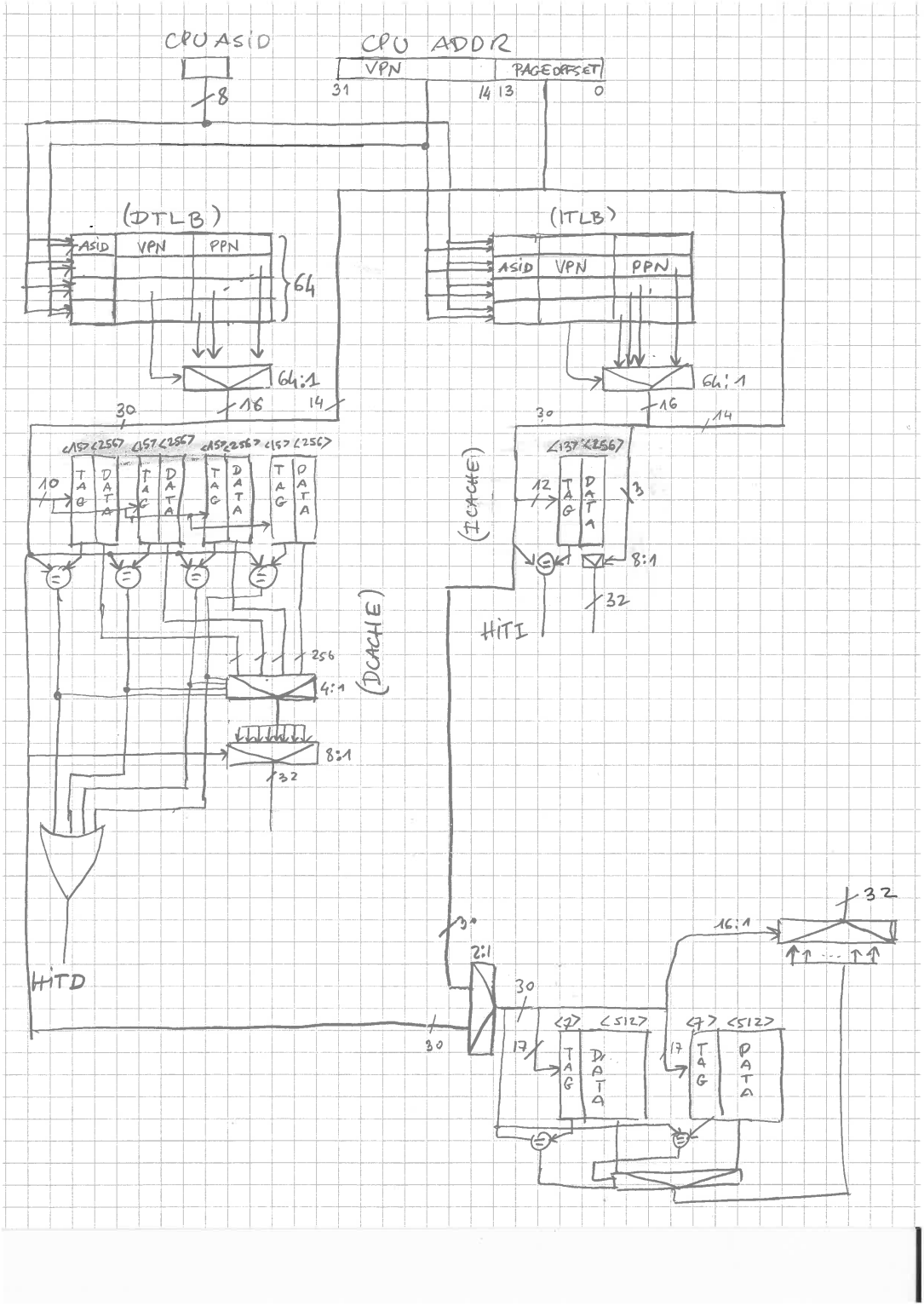
Ensimag 2A

Année scolaire 2016–2017

— Durée : 3h. Tous documents et calculatrices autorisés ;
— Le barème est donné à titre indicatif ;
— Les exercices sont indépendants et peuvent être traités dans le désordre, et certaines questions au sein du même exercice peuvent aussi être traitées indépendamment ;
— Chaque exercice nécessite une lecture attentive et complète avant d'être commencé. Certains exercices sont longs à lire et contiennent beaucoup d'information, mais aucun ne nécessite la rédaction de longues réponses.
— Il n'y a pas de question piège, et une réponse peut tenir en une ligne sans que ce soit une erreur !

Ex. 1 : Cache et mémoire virtuelle (4 pts)

La figure ci-dessous présente une architecture de hiérarchie mémoire.



L'objectif de l'exercice est d'analyser cette structure pour en déterminer les caractéristiques.

Question 1 Donnez la taille des adresses virtuelles et des adresses physiques, ainsi que la taille des pages.

VA : 32 bits, PA : 30 bits, Pages : 16 KB (2^{14}).

Question 2 Précisez les caractéristiques de la géométrique (nombre d'octets dans le cache, associativité, taille du TAG, de l'INDEX, de l'OFFSET) du cache L1 de données, et sa relation à la TLB (physically indexed/physically tagged, virtually indexed/virtually tagged, ou virtually indexed/physically tagged) Justifiez en quelques lignes.

nombre d'octets dans le cache : 128Ko, associativité : 4-way, taille du TAG : 15 bits, de l'INDEX : 10 bits, de l'OFFSET : 3 bits
Physically Indexed/Physically tagged (ce sont les addresses en sortie de la DTLB qui attaquent le cache)

Question 3 Précisez les caractéristiques de la géométrique (nombre d'octets dans le cache, associativité, taille du TAG, de l'INDEX, de l'OFFSET) du cache L1 d'instructions, et sa relation à la TLB (physically indexed/physically tagged, virtually indexed/virtually tagged, ou virtually indexed/physically tagged) Justifiez en quelques lignes.

nombre d'octets dans le cache : 128Ko, associativité : direct-map, taille du TAG : 13 bits, de l'INDEX : 12 bits, de l'OFFSET : 3 bits
Physically Indexed/Physically tagged (ce sont les addresses en sortie de la DTLB qui attaquent le cache)

Question 4 Précisez les caractéristiques de la géométrique (nombre d'octets dans le cache, taille du TAG, de l'INDEX, de l'OFFSET) du cache L2.

nombre d'octets dans le cache : 16Mo, associativité : 2-way, taille du TAG : 7 bits, de l'INDEX : 17 bits, de l'OFFSET : 4 bits

Ex. 2 : Cohérence mémoire par espionnage (7 pts)

Soit p processeurs, tous connectés grâce à un unique PI-Bus à la mémoire centrale partagée, par l'intermédiaire d'un cache. Les caches sont tous identiques, à correspondance directe (*direct mapping*), à écriture systématique (*write through*), structurés en 128 lignes de 4 mots de 32 bits. On fait également l'hypothèse que les seuls transferts qui ont lieu sont des lectures ou des écritures de *mots de 32 bits*.

Rappels et préliminaires

On rappelle ici les signaux du PI-Bus :

Nom	emis	reçu	commentaires
RESETN	environnement	tous	remise à zéro.
CLK	environnement	tous	horloge système, active sur front montant.
REQ$_x$	maître x	BCU	Requête du maître x pour le bus.
GNT$_x$	BCU	maître x	Donne le bus au maître x.
READ	maître	esclave	Lecture si 1, écriture si 0.
LOCK	maître	BCU	Garde la propriété du bus.
OPC[3 :0]	maître	esclave, BCU	Type de transfert.
A[31 :2]	maître	esclave, BCU	Adresse du transfert.
D[31 :0]	maître, esclave	esclave, maître	Donnée.
ACK[2 :0]	esclave, BCU	maître	Réponse au transfert.
SEL$_y$	BCU	esclave y	Sélection de l'esclave y.

Les signaux sont tous actifs à l'état haut, hormis le **RESETN** qui est actif à l'état bas. Nous rappelons que les adresses sont sur 30 bits, les 2 bits de poids faibles étant définis par les opcodes spécifiant le transfert, et que les octets significatifs sont alignés à droite. Les signaux **READ**, **OPC**, **LOCK** et **A** sont positionnés au cycle t, et les signaux **ACK** et **D** sont positionnés au cycle $t + 1$.

Le signal **ACK** peut prendre les valeurs suivantes :

symbole	description
WAT	transfert retardé.
RDY	transfert effectué.

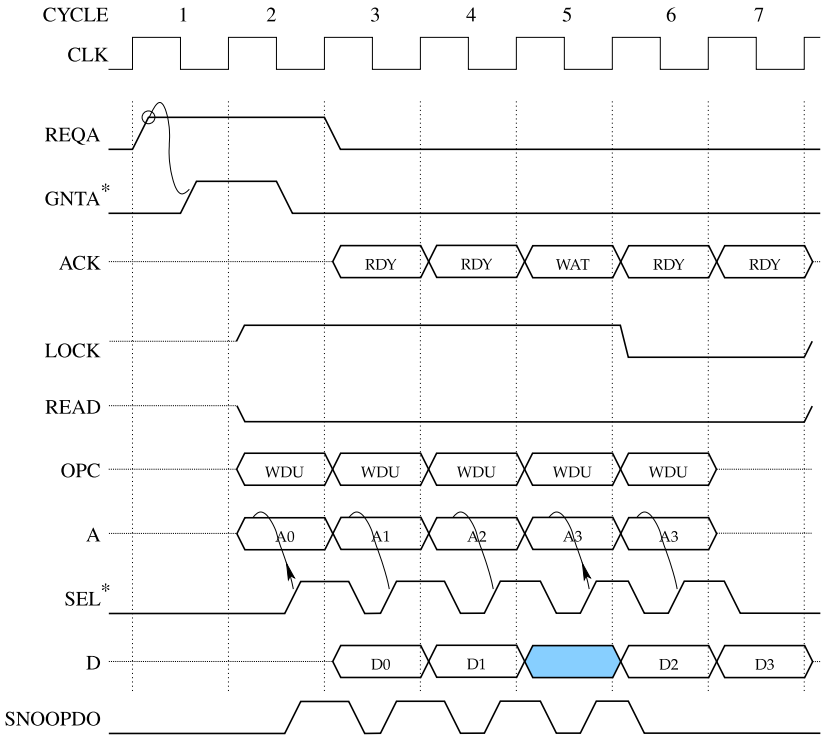


FIGURE 1 – Chronogramme d'écritures successives en mémoire.

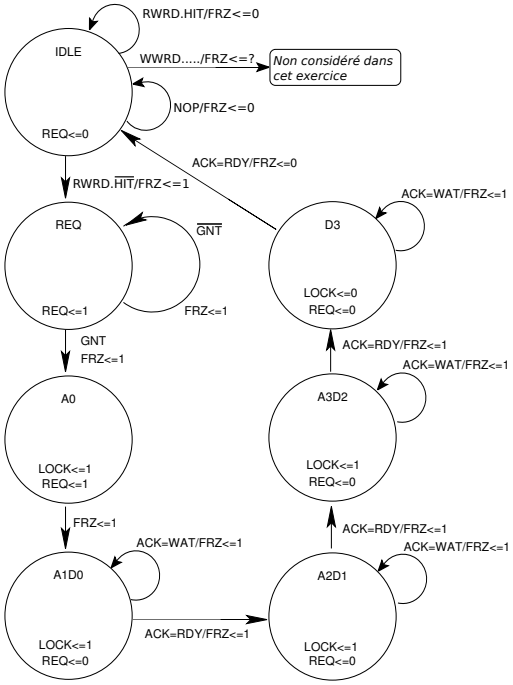


FIGURE 2 – Automate de chargement d'une ligne de cache (sans la génération des sorties).

Le signal **OPC** ne peut prendre que la valeur **WDU** dans notre exercice (uniquement transferts de mots de 32 bits).

Un exemple de chronogramme est donné dans la figure 1.

La mémoire utilisée pour implanter le cache est une mémoire simple accès, les lectures et les écritures ne peuvent donc avoir lieu au même cycle.

L'interface du processeur vers le cache de données dont nous avons l'usage est la suivante :

- **TYPE**. Émis par le processeur, indiquant le type de transfert à effectuer. Dans un but de simplification, seules 3 valeurs sont possibles : NOP (pas de requête), RWD (requête de lecture d'un mot), WWD (requête de d'écriture d'un mot).
- **ADR[31 :0]**. Émis par le processeur, contenant l'adresse de la requête courante. Pour notre exercice **ADR[1 :0]** = 00 ;
- **FRZ**. Reçu par le processeur. Si ce signal vaut 1, le processeur est gelé (combinatoirement, c.-à-d. que si **DFRZ** passe à 1 dans un cycle, les registres du processeur ne sont pas modifiés à la fin du cycle).

Exercice proprement dit

On se propose dans cet exercice de concevoir un mécanisme implanté dans le cache qui espionne les transferts de données en écriture et *invalide* la ligne de cache si une écriture a lieu à une adresse correspondant à une donnée qui se trouve dans le cache.

On doit pour se faire ajouter un signal supplémentaire, **SNOOPDO**, émis par le contrôleur de bus à destination des maîtres, qui est à 1 la première fois qu'une adresse est émise sur le bus par un maître. On voit sur le chronogramme de la figure 1 que le signal **SNOOPDO** est à 1 chaque fois qu'une *nouvelle* adresse est présentée sur le bus, mais qu'en cas de ACK=WAT, qui implique que le maître remet l'adresse suivante, le signal n'est pas réactivé.

Question 1

Le cache est-il maître ou esclave sur le bus ? Justifiez en moins de 15 mots.

C'est un maître, car il est à l'initiative des transferts.

Question 2

Donnez la « structure » de l'adresse pour ce cache, c'est-à-dire le nombre de bits de TAG, d'INDEX et d'OFFSET.

Question 3
Du point de vue du protocole de bus (et indépendamment du comportement attendu du cache qui sera l'objet de la suite), qu'est-il absolument nécessaire de faire au niveau des caches lorsque le signal **SNOOPDO** est à 1 ?

Échantillonner l'adresse !

Dessinez le petit circuit permettant de faire cette action. (Il n'y a pas de piège, la réponse est simple.)

Un registre avec @ sur data et snoopdo sur we

On change la sémantique du signal **HIT**. Ce signal indique à présent soit que la donnée requise par le processeur est cachée, *soit qu'il y a un* hit *externe*, c'est à dire que la donnée écrite circulant sur le bus se trouve dans le cache.

Question 4
Que se passe-t-il lorsque le processeur et le bus réalisent des requêtes simultanément (écriture pour le bus) ? Proposez une solution à faible coût matériel (typiquement sans dupliquer des choses existantes) qui garantisse le bon fonctionnement dans ce cas de figure. Faites le schéma de la partie opérative correspondante, sachant que le contrôle sera déterminé dans les 2 questions suivantes.

Un mux sur les fils d'adresse représentant le tag et l'index multiplexant soit l'adresse venant du processeur, soit celle échantillonnée dans le registre.

Question 5
En considérant les instants d'arrivée possibles des requêtes de lecture du processeur et des requêtes d'écriture circulant sur le bus, décrivez le problème qui se pose sur le cache et proposez une solution qui garantisse un fonctionnement correct. (On notera ici que le cache qui espionne ne peut influencer de quelque manière que ce soit les transferts en cours). Vous pouvez vous aider d'une table énumérant les 4 situations possibles pour expliquer votre raisonnement.

Proc	Bus	Problème ?
0	0	Pas de problèmes
1	0	Pas de problèmes
0	1	Pas de problèmes
1	1	Geler le processeur et donner la priorité au bus

Le chargement d'une ligne de cache pour un système mono-processeur se passe comme indiqué grossièrement (on n'indique pas ici les sorties de l'automate pilotant le chemin de données) par l'automate de la figure 2, déjà vu en cours.

On veut à présent prendre en compte le signal **SNOOPDO** dans les transitions de l'automate.

Question 6

— à quelle condition y a-t-il un *hit* externe ? Donnez l'équation logique permettant de la calculer.

xhit = hit.snoopdo d'après ce qui a été dit avant

— dans quels états de l'automate est-on certain de ne jamais avoir de *hit* externe ? Justifiez votre réponse.

Dans les états durant lequel le processeur est propriétaire du bus : A0, A1D0, A2D1, A3D2, D2

On fera l'hypothèse que l'on va dans un état INVAL pour invalider la ligne visée (écriture d'un 0 dans le bit de validité).

Question 7
Ajoutez un ou des états INVAL là où cela est nécessaire dans l'automate. Modifiez les conditions de transitions des états qui peuvent être sensibles à un *hit* externe pour prendre en compte le signal **SNOOPDO** et passer ainsi dans cet état. Donnez également les transitions permettant de sortir de l'état INVAL.

Question bonus : inutile de s'y attaquer si vous n'avez pas fini le reste !

Au lieu d'invalider la ligne, on désire la mettre à jour, ce qui nécessite 2 états : UPDTR pour lire la ligne et UPDTW pour la ré-écrire modifiée dans le cache, comme vu en cours.

Question 8
Quel est le problème si un *hit* externe à lieu (sur une autre ligne) lorsque l'automate est dans l'un de ces 2 états ?

On est en train d'invalider et on a besoin de l'adresse

Quelle est alors la seule solution pour assurer la cohérence des données ?

Il faut invalider totalement le cache, car on a peut-être raté qqchose, ...

Ex. 3 : Pipeline RISC (9 pts)

On se propose dans cet exercice d'étudier des instructions permettant l'accès « non-aligné » à des *mots de 32 bits* en mémoire. Pour mémoire (ha, ha, ha !), les instructions *load word* et *store word* du MIPS R3000 requièrent que l'adresse utilisée soit un multiple de 4 (deux bits de poids faible à zéro), sinon il y a levée d'une exception.

Pour permettre des lectures non-alignés (on ne traitera pas les écritures), le jeu d'instruction du MIPS est enrichi de deux instructions [1] : `lwl rt, imm(rs)`, *load word left*, et `lwr rt, imm(rs)`, *load word right*.

On suppose que la mémoire contient la valeur `I J K L`, `I` étant l'octet de poids le plus significatif, `L` l'octet de poids le moins significatif, et que le registre destination contient initialement `e f g h`, `e` étant l'octet de poids le plus significatif et `h` l'octet de poids le moins significatif. Le comportement de ces instructions est précisé ci-dessous :

	`lwl`	`lwr`
bits de poids faible de l'adresse	registre après l'instruction	registre après l'instruction
00	`L f g h`	`I J K L`
01	`K L g h`	`e I J K`
10	`J K L h`	`e f I J`
11	`I J K L`	`e f g I`

Une partie du registre destination reste inchangée (lettres minuscules, sauf dans le cas 11 pour `lwl` et 00 pour `lwr`), et une partie du contenu de la mémoire est injectée soit sur les poids forts du registre (`lwl`), soit sur ses poids faibles (`lwr`) en fonction des deux bits de poids faible de l'adresse.

Le pipeline du MIPS est donné en annexe figure 3. Une partie des questions porte sur des ajouts à ce schéma, aussi **n'oubliez pas** d'y apposer votre nom et de le rendre à la fin de l'examen !

Question 1 Sur le schéma en annexe ajoutez, sur les fils et dans les registres du pipeline des annotations identifiant :
- les numéros des registres rs, rt, et rd que l'on notera respectivement nrs, nrt, et nrd. Dans le cas ou le registre destination est rt, on le notera néanmoins nrd, en précisant sur le dessin ou est fait ce choix ;
- la valeur immédiate issue de l'instruction ;
- les valeurs des registres rs, rt, et rd.que l'on notera respectivement vrs, vrt, et vrd. Dans le cas ou le registre destination est rt, on notera néanmoins vrd le registre destination ;
- les ports din, $dout$, et $addr$ des mémoires caches.

XXX

Question 2 Que représentent les signaux HITI et HITD ? Comment vous proposez-vous de les utiliser dans le pipeline ?

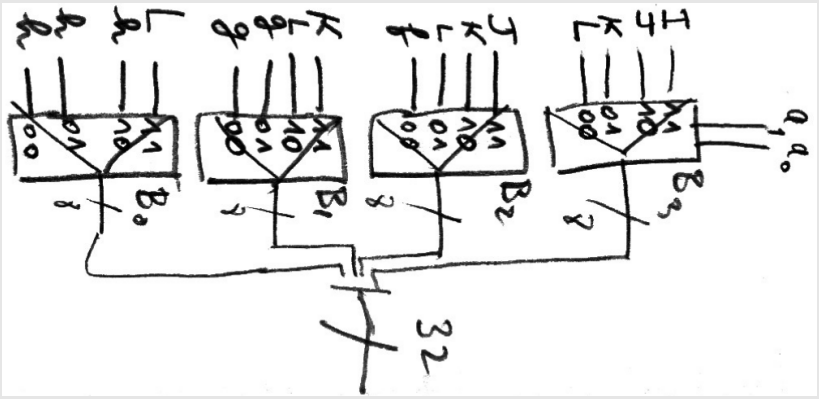
1. Pour information, ces deux instructions ont été brevetés par MIPS, ce qui a permis à la société de tuer au moins une compagnie ayant fait un clone de MIPS. Le brevet est tombé dans le domaine public en 2006.

HITI et HITD informent de la présence effective de l'instruction ou de la donnée dans le cache correspondant. Si l'un d'eux vaut 0, tous les WE sont à zéro.

On veut à présent réaliser l'instruction `lwl` (on ne fera pas `lwr` qui est basée sur les mêmes principes, et on notera `lwx` l'une ou l'autre des instructions lorsque l'opération « effective » n'est pas importante). On cherche tout d'abord à implanter son comportement indépendamment du pipeline.

Question 3 Cette opération de « fusion » de deux valeurs est elle séquentielle ou combinatoire ? Proposez un schéma qui permette de la réaliser. On ne cherchera pas à optimiser

C'est une opération combinatoire qui se réalise à l'aide de mux. On remarque que l'octet de poids fort du résultat peut prendre les valeurs I, J, K, L, que l'octet qui suit peut prendre les valeurs J, K, L, f, celui qui suit, les valeurs g, L et K, et l'octet de poids faible h et L. On a donc besoin de deux mux 1 parmi 4, 1 mux 1 parmi 3 et un mux 1 parmi 2. Néanmoins, pour faire simple, on peut prendre 4 mux 4 vers 1, car il est ainsi plus aisé d'exprimer la fonction de choix :



On veut ensuite insérer ce matériel dans le pipeline.

Question 4 Quels sont les étages susceptibles d'accueillir ce matériel ? En faisant l'hypothèse que les étages les plus contraints en temps sont les étages qui accèdent aux caches, dans quel étage proposeriez-vous de placer le matériel de la question précédente ?

Seuls les étages MEM et WB sont à même de réaliser cette opération de fusion, car on ne dispose pas de la valeur issue de la mémoire avant. Avec la contrainte de temps, l'étage WB est le plus approprié.

Question 5 Précisez les informations qu'il est nécessaire d'avoir ou d'ajouter (s'il y en a) dans le registre du pipeline que vous avez choisi.

Il faut les 2 bits de poids faibles de l'adresse (la valeur effective de l'adresse, qu'on aura sans doute noté vrd, issue de EX/MEM doit donc être en partie recopié dans MEM/WB), vrt et l'information que l'instruction est un `lwx`, car il faut maitenant sélectionner le résultat de la fusion et non ce qui sort de la mémoire (on peut aussi mettre les deux bits d'adresse à 1 lorsque l'on détecte un `lw`, ce qui est sans doute plus économique).

Question 6 [(difficile)] Mr MIPS propose la macro-instruction `ulw rt, imm(rs)` pour réaliser un accès non aligné, qui en réalité est implantée comme suit :

```
lwl rt, 3(rs)
lwr rt, 0(rs)
```

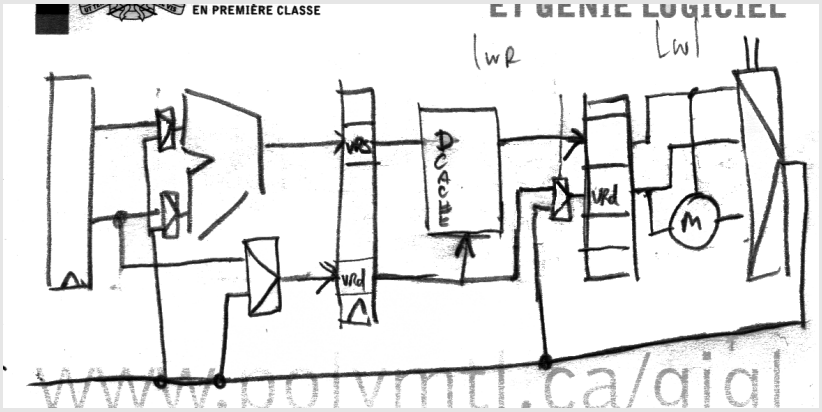
Listez les dépendances possibles entre ces deux instructions. On ne s'intéresse dans la suite qu'à la dépendance entre les rt. Sachant que le compilateur ou l'assembleur peuvent insérer des instructions entre ces deux instructions, étudiez les cas ou 0, 1 et 2 instructions sont insérées.

Ajoutez les bypass pour conserver le débit d'une instruction par cycle, sans ajouter de multiplexeur sur la sortie de la mémoire. Vous pouvez le faire sur l'annexe ou redessiner la partie concernée sur votre feuille (inutile dans ce cas d'ajouter les détails qui ne sont pas utiles pour la gestion des dépendances).

Précisez les commandes des mux si vous en ajoutez.

La dépendance entre le rs du lwr et le rt du lwl est déjà gérée par la détection des aléas des lw standards, par insertion de bulles, même si cela n'a pas grand sens d'un point de vue programmation.

Par contre, vu que l'opération de fusion utilise la précédente valeur de rt, il y a une dépendance entre le rt du lwr et celui du lwl. Pour cela, on doit ajouter un bypass entre WB et MEM, et entre WB et EX. Pour le cas ou il y a 2 instructions, les mux en sortie du banc de registres rendent le service. Cela donne :



- commande du mux dans WB : entrée du bas si MEM/WB.opc == lwx, sinon entrée du haut si opc == lx, sinon entrée du milieu
- commande du mux dans MEM : entrée du bas si EX/MEM.opc == lwx et MEM/WB.opc == lwx et EX/MEM.rt == MEM/WB.rt
- commande du mux dans EXE : entrée du bas si ID/EX.opc == lwx et EX/MEM.opc == lwx et ID/EX.rt == EX/MEM.rt

Annexe à rendre

Votre nom :

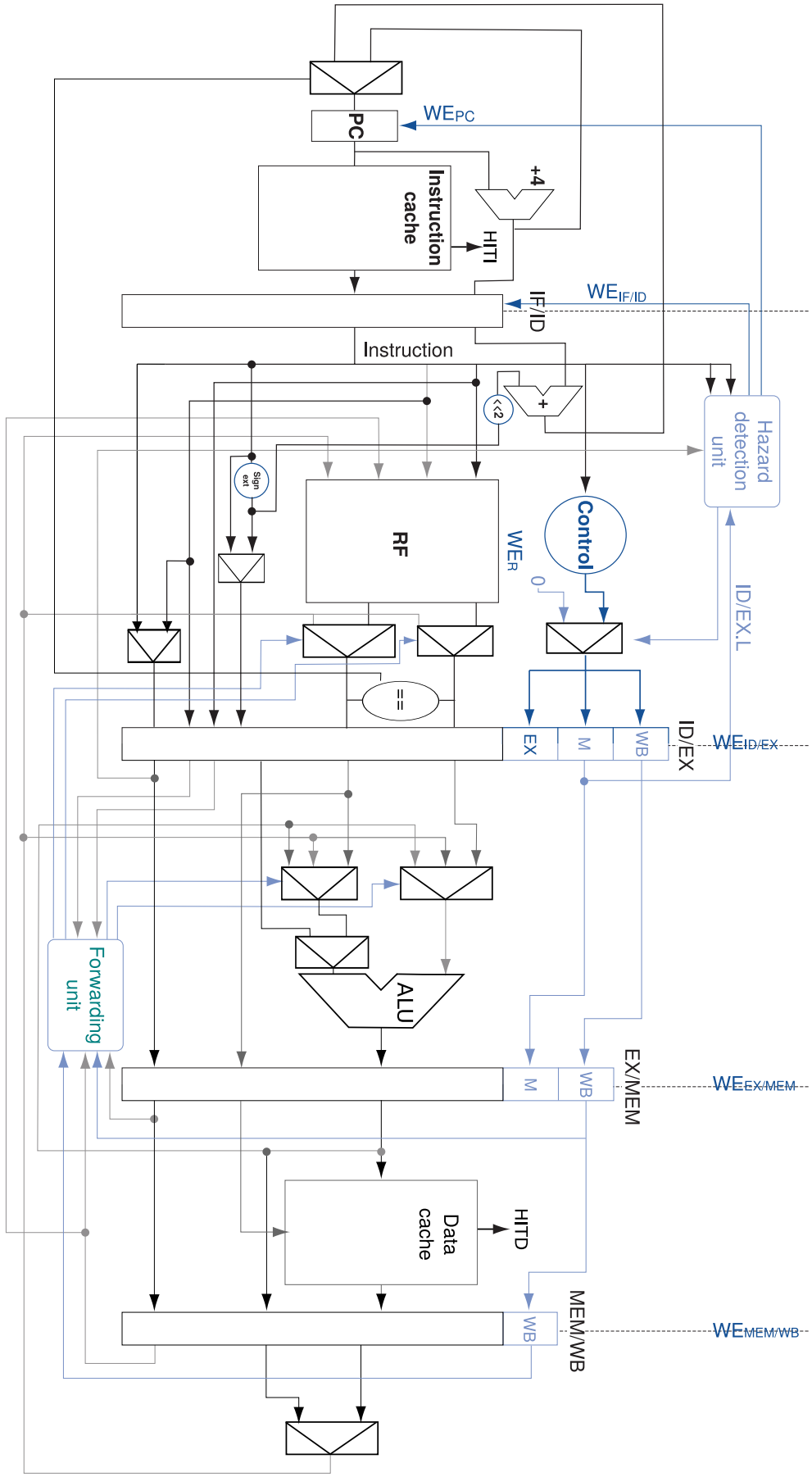


FIGURE 3 – Pipeline du MIPS