Architecture des Ordinateurs

Processeur MIPS en Pipeline et Aléas d'Exécution

2ème Année Licence Informatique

Travaux Dirigés #6

1. Contrôle préalable

Cette partie est conçue comme une vérification pour vous permettre de déterminer si vous comprenez les concepts abordés en cours ou non. Veuillez répondre par « Vrai » ou « Faux » aux questions suivantes et inclure une explication :

1.1. L'introduction du pipeline au chemin de données du processeur entraîne l'exécution d'instructions avec une latence et un débit plus élevés.
Vrai. La latence est le temps nécessaire pour qu'une instruction se termine, tandis que le débit est le nombre d'instructions traitées par unité de temps. L'introduction du pipeline dans un processeur se traduit par un débit plus élevé car plus d'instructions sont exécutées à la fois. La latence est également plus élevée car chaque instruction individuelle prend plus de temps du début à la fin (rappelons que chaque cycle doit durer aussi longtemps que le cycle le plus long dans le pipeline). De plus, des aléas d'exécutions peuvent surgir.

1.2. Les « aléas de données » se traduiront généralement par trois suspensions (*anglais.* stalls) du pipeline si on n'utilise pas l'unité de transfert.
Vrai. L'instruction suivante doit attendre que l'instruction précédente termine les étapes EX, MEM et WB avant de pouvoir utiliser l'étage EX du pipeline.

1.3. Tous les « aléas de données » peuvent être résolus en utilisant l'unité de transfert.
Faux. Les aléas qui peuvent surgir à la suite d'une instruction de chargement (`lw`, …) ne peuvent pas être entièrement résolus avec l'unité de transfert car la donnée n'est connue qu'à l'étage MEM. De ce fait, une suspension du pipeline est nécessaire.

1.4. La suspension du pipeline est le seul moyen de résoudre les « aléas de branchement ».
Faux. La suspension du pipeline est une façon de résoudre les aléas de branchement. D'autres techniques plus avancées tentent de prédire le chemin emprunté par l'instruction après un branchement et purge le pipeline si la prédiction s'avère erronée.

2. Registres du pipeline

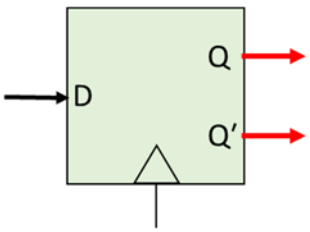
2.1. Afin de convertir un processeur en cycle-unique en un processeur en pipeline, des registres sont introduits entre les différentes étapes du chemin de données. Quel est le rôle de ces registres ?

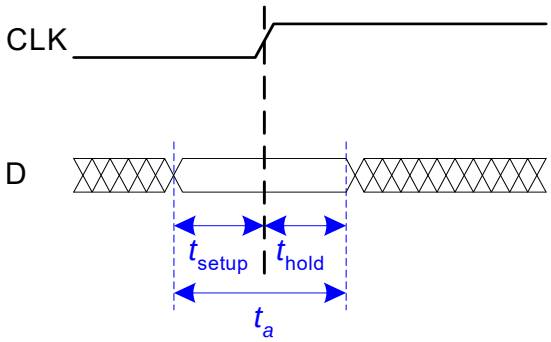
Dans un chemin de données en pipeline, les valeurs de chaque étage doivent être transmises à chaque cycle d'horloge. Chaque étage du pipeline ne fonctionne que sur un petit ensemble de valeurs, mais ces valeurs doivent être correctes par rapport à l'instruction en cours de traitement. Prenons l'instruction de chargement « `lw` » comme exemple : si l'instruction est à l'étape EX, alors les valeurs de cette étape devraient ressembler aux valeurs de l'étape EX dans un chemin de données à cycle unique. Les valeurs des registres `rs1`, `rs2`, de l'immédiat « `imm` » et du `PC` doivent être comme si l'instruction « `lw` » était la seule instruction dans le chemin complet. Cela inclut également les signaux de commande : comme l'instruction est transmise d'un étage à l'autre, les signaux de commande appropriés pour chaque étage sont générés pendant le décodage de l'instruction, puis transmis dans le pipeline afin d'assurer que chaque étage ait les signaux corrects pour son exécution.

2.2. Pourquoi avons-nous besoin de sauvegarder les signaux de commande plusieurs fois dans le pipeline ?
Nous devons sauvegarder les signaux de commande plusieurs fois car chaque étage de pipeline doit recevoir les bons signaux de commande pour l'instruction actuellement à cet étage.

3. Analyse de Performance

Avant de continuer avec les questions de ce TD, faisons un petit rappel sur les contraintes de temps associées aux bascules et circuits combinatoires.



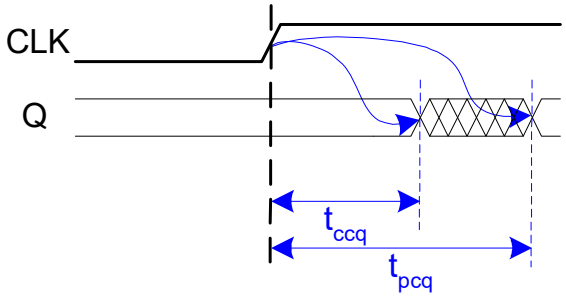


Contraintes en entrée :

- Le "Setup Time" t_{setup} est le temps avant le front montant de CLK auquel l'entrée D doit être stable (c.-à-d. ne change pas).
- Le "Hold Time" t_{hold} est le temps après le front montant de CLK auquel l'entrée D ne doit pas changer.
- t_a est la fenêtre de temps auquel l'entrée D doit être stable ($t_a = t_{setup} + t_{hold}$).

Contraintes en sortie :

- Le "min CLK to Q delay" t_{ccq} est le temps après le front montant CLK auquel la sortie Q serait instable (c.-à-d. pourrait changer).
- Le "max CLK to Q delay" t_{pcq} est le temps après le front montant CLK

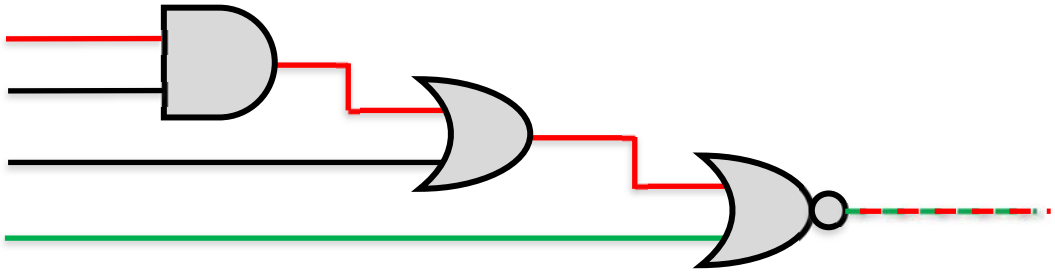


auquel la sortie Q est garantie d'être stable.

Circuits combinatoires :

Les circuits combinatoires possèdent également des contraintes de temps sur les ports logiques utilisés. Le "min delay" t_{cd} pour un circuit (en vert) est le temps minimum entre le moment où une entrée du circuit change et le moment où la sortie de ce circuit commence à changer.

Le "max delay" t_{pd} (en rouge) est le temps maximum entre le moment où une entrée du circuit change et jusqu'à ce que la sortie atteigne sa valeur finale.

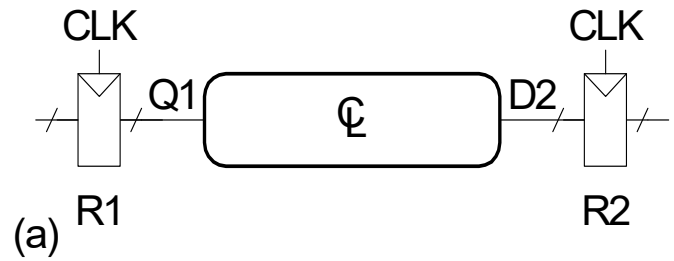


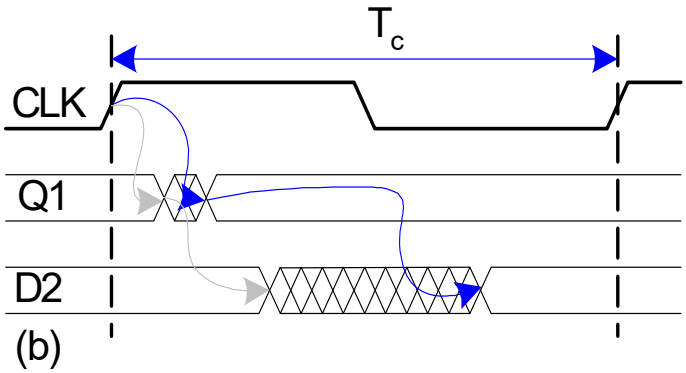
Registres et circuits combinatoires :

Le schéma suivant montre un circuit constitué de deux registres et une logique combinatoire quelconque entre ces deux registres ;

Dans le schéma à droite, l'entrée d'une bascule provient de la sortie d'une autre bascule, via un circuit combinatoire. C-à-d. l'entrée `D2` de la bascule `R2` provient de la sortie `Q1` de la bascule `R1` après avoir traversé le circuit logique CL.

Ainsi, il faut que la sortie du circuit R1_CL (c.-à-d. D2) se stabilise pour une durée minimale combinée supérieure ou égale au temps de setup t_{setup} de la bascule R2 si l'on veut que cette dernière « observe » la nouvelle valeur D2. Il faut aussi que cette valeur reste stable pendant une durée t_{hold} après le front montant de l'horloge pour que la bascule R2 ait le temps de faire une copie.



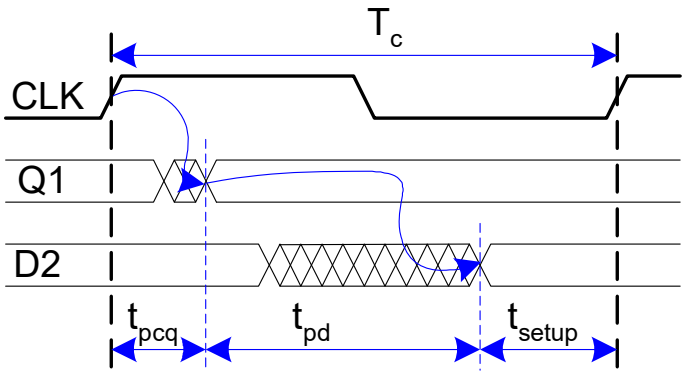


Pour respecter la contrainte sur t_{hold}, nous avons :

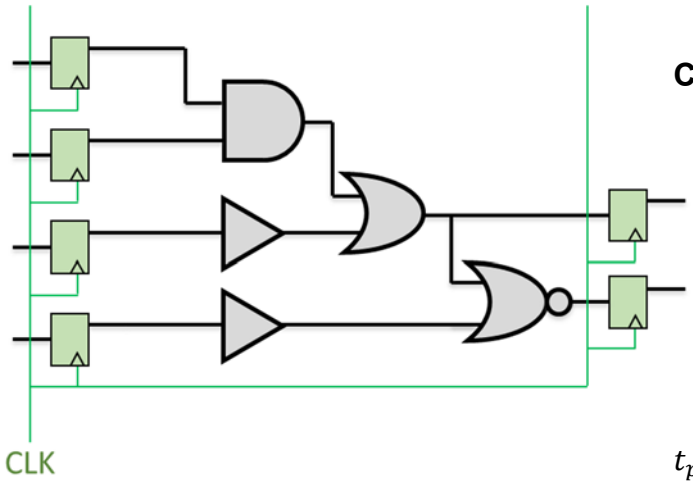
$$t_{hold} < min\_delai(bascule) + min\_delai(CL)$$
$$< t_{ccq} + t_{cd}$$

Et pour respecter la contrainte sur t_{setup}, nous avons :

$$T_c \geq max\_delai(bascule) + max\_delai(CL) + t_{setup}$$
$$\geq t_{pcq} + t_{pd} + t_{setup}$$



Exemple :



Caractéristiques temporelles

t_{ccq} = 30 ps, t_{pcq} = 50 ps

t_{setup} = 60 ps, t_{hold} = 70 ps

t_{pd} = 35 ps, t_{cd} = 50 ps

t_{pd} circuit = $3 \times 35ps = 105ps$

t_{cd} circuit = 2 x 25ps = 50ps

Contrainte sur t_{setup}

$T_c \geq (50 + 105 + 60)ps = 215ps$
$f_c = 1/T_c = 4.65\ GHz$

Contrainte sur t_{hold}

$t_{ccq} + t_{cd} > t_{hold}$
$(30 + 50)ps > 70ps$ Vérifiée !

Retour maintenant à notre circuit de processeur en pipeline. Afin de réaliser une analyse temporelle des performances, supposons les durées de temps suivantes :

t_{cpq} registre :	30 ps	,	t_{setup} registre :	20 ps
t_{pd} Mux :	25 ps	,	t_{pd} UAL :	200 ps
t_{cpq} Mem :	250 ps	,	t_{setup} MEM :	200 ps
t_{cpq} Banc Regs. :	150 ps	,	t_{setup} Banc Regs. :	20 ps

3.1. Avec les délais indiqués ci-dessus pour chacun des composants du chemin de données, quel serait le temps d'horloge le plus rapide possible pour un chemin de données à cycle unique ?

$$T_c \geq t_{cpq}(PC) + t_{cpq}(IMEM) + t_{cpq}(BR) + t_{pd}(Mux) + t_{pd}(UAL) + t_{cpq}(DMEM) + t_{pd}(Mux) + t_{setup}(BR)$$
$$T_c \geq 950ps$$
$$f_c \leq \frac{1}{T_c} \cong 1.05\ GHz$$

Notons que le délai du Mux avant le « Banc de registres » est omis ici car dans le cas d'une instruction de chargement (*i.e.* notre chemin critique), la sortie de ce composant est requise beaucoup plus tard dans le chemin de donnée (après WB). Ce qui laisse largement le temps à la sortie de ce Mux de se stabiliser pour l'entrée « Write register » du « Banc de registres ».

3.2. Quelle serait la fréquence d'horloge la plus rapide possible pour un chemin de données en pipeline ?

IF : $t_{cpq}(PC) + t_{cpq}(IMEM) + t_{setup}(Regs\_IF{:}ID) = 300\ ps$

ID : $t_{cpq}(Regs\_IF{:}ID) + t_{cpq}(BR) + t_{setup}(Regs\_ID{:}EX) = 200\ ps$

EX : $t_{cpq}(Regs\_ID{:}EX) + t_{pd}(Mux) + t_{pd}(UAL) + t_{setup}(Regs\_EX{:}MEM) = 275\ ps$

MEM : $t_{cpq}(Regs\_EX{:}MEM) + t_{cpq}(DMEM) + t_{setup}(Regs\_MEM{:}WB) = 300\ ps$

WB : $t_{cpq}(Regs\_MEM{:}WB) + t_{pd}(Mux) + t_{setup}(BR) = 75\ ps$

$max(IF, ID, EX, MEM, WB) = 300\ ps \quad \Rightarrow \quad f_c = \frac{1}{300\ ps} \cong 3.33\ GHz$

3.3. Quelle est l'accélération obtenue après conversion du chemin de données à cycle unique vers un chemin de données en pipeline ? Pourquoi l'accélération est-elle inférieure à 5 ?

Une accélération de $\frac{950\ ps}{300\ ps} \cong 3.17$ fois. L'accélération est inférieure à 5 parce que les registres ajoutés pour le pipeline introduisent des délais supplémentaires (t_{cpq} et t_{setup}) au chemin de données. D'autre part, il est nécessaire de régler l'horloge sur le délai maximal requis parmi les cinq étapes.

Remarque : Dans la réalité, des aléas d'exécutions surgissent dans un chemin de données en pipeline. Ce genre de problèmes nécessitent des circuits additionnels pour être résolus ce qui impacterait davantage l'accélération.

4. Aléas d'exécution

La conversion d'un chemin de données à cycle unique en une version en pipeline introduit trois types d'aléas d'exécution : aléas structurels, aléas de données et aléas de branchement.

Aléas structurels

Ils surviennent lorsque plusieurs instructions doivent utiliser en même temps la même ressource dans le chemin de données. Il existe deux causes principales d'aléas structurels :

Banc de registres : Le « banc de registres » est accédé à la fois pendant l'étape ID, lorsqu'il est lu, et pendant l'étape WB, lorsqu'il est modifié (écrit). Ce problème est résolu en ayant des ports de lecture et d'écriture séparés. Dans le cas de lecture et d'écriture simultanées dans

le même registre, le processeur écrit dans le registre pendant la première moitié du cycle d'horloge et lit à partir de celui-ci pendant la seconde moitié. Ceci est également connu sous le nom de « double pompage ».

Accès Mémoire : La mémoire héberge au même temps les instructions du programme à exécuter et les données à traiter. Disposer d'une mémoire d'instructions séparée (IMEM) et d'une mémoire de données (DMEM) résout ce problème. Cela est implémenté en utilisant des mémoires caches séparées pour les instructions et les données (voir cours sur les caches mémoire).

Les aléas structurels peuvent toujours être résolus en ajoutant plus de matériel.

Aléas de données

Ces aléas sont causés par les dépendances sur les données entre les instructions. En particulier, Un aléa de données survient lorsqu'une instruction lit depuis un registre avant qu'une instruction précédente n'ait fini d'écrire dans ce registre.

Unité de transfert

La plupart des aléas de données peuvent être résolus par transfert, c.-à-d. lorsque la sortie de l'étage EX ou l'étage MEM est transférée directement à l'étage EX de l'instruction ultérieure.

4.1. Signalez les aléas de données dans le code ci-dessous et indiquez comment le transfert do données pourrait être utilisé pour les résoudre.

Cycles / Instructions	C1	C2	C3	C4	C5	C6	C7
`1. addi $t0, $a0, -1`	IF	ID	EX	MEM	WB		
`2. and $s2, $t0, $a0`		IF	ID	EX	MEM	WB	
`3. sltiu $a0, $t0, 5`			IF	ID	EX	MEM	WB

Il y a deux aléas de données : Entre les instructions 1 et 2 et entre les instructions 1 et 3. Le premier aléas pourrait être résolu en transférant le résultat de l'étape EX en C3 à l'entrée de l'étape EX en C4. Le second aléa pourrait être résolu en transférant le résultat de l'étape EX en C3 au début de l'étape EX en C5.

4.2. Combien d'instructions après une instruction `addi` pourraient être affectées par des aléas de données liés à cette instruction ?

Trois instructions. Par exemple, en considérant l'instruction 1 dans la question 4.1, toute instruction ultérieure qui utiliserait le registre $t0 dans son étape ID pendant les cycles d'horloges C3, C4 ou C5 n'aura pas la bonne valeur de $t0 parce que l'instruction `addi` le mettra seulement à jour dans le cycle C5. Si, cependant, nous sommes autorisés à supposer la possibilité d'écriture-puis-lecture dans un registre en un seul cycle d'horloge, alors il y'aura que deux instructions qui seront affectées parce que l'étape ID de l'instruction 4 pourrait être exécutée au même temps que l'étape WB de l'instruction 1.

Suspension de pipeline

4.3. Identifiez les aléas de données dans le code ci-dessous. Pourquoi l'un de ces aléas ne peut pas être résolu avec l'unité de transfert ? Que pouvons-nous faire pour résoudre ce problème ?

Cycles / Instructions	C1	C2	C3	C4	C5	C6	C7	C8
`1. addi $s0, $s0, 1`	IF	ID	EX	MEM	WB			
`2. addi $t0, $t0, 4`		IF	ID	EX	MEM	WB		
`3. lw $t1, 0($t0)`			IF	ID	EX	MEM	WB	
`4. add $t2, $t1, $0`				IF	ID	EX	MEM	WB

Il existe deux aléas de données dans le code. Le premier problème se situe entre les instructions 2 et 3 (registre $t0), et le second entre les instructions 3 et 4 (registre $t1). Le problème entre les instructions 2 et 3 peut être résolu avec l'unité de transfert, mais celui des instructions 3 et 4 ne peut pas être résolu avec cette unité (seulement). En effet, même si on utilisait l'unité de transfert, l'instruction 4 a besoin du résultat de l'instruction 3 au début du cycle C6, mais la valeur de $t1 ne sera disponible qu'à la fin de ce cycle.

Nous pouvons résoudre ce problème en insérant une bulle dans le pipeline entre les instructions 3 et 4 (c-à-d. une instruction **n**o-**op**eration ou **nop**).

4.4. Dans une équipe d'ingénieurs, vous travaillez sur la conception d'un compilateur pour un processeur MIPS (rappelez-vous, c'est le compilateur qui produit des fichiers en langage assembleur). Comment ce compilateur pourrait réorganiser les instructions du code de l'exercice 4.1 afin de minimiser les aléas de données tout en garantissant le même résultat.
Réorganiser les instructions dans l'ordre 2–3–1–4 parce que l'instruction 1 n'a pas de dépendances.

Détection des aléas de données

Supposons que nous ayons les signaux `rs`, `rt`, `RegWEn` et `rd` pour deux instructions aux instants t et $t + 1$. Nous souhaitons déterminer si un aléa de données existe entre ces instructions. En ce sens, nous pouvons vérifier si le `rd` de l'instruction à l'instant t correspond à `rs` ou `rt` de l'instruction à l'instant $t + 1$, indiquant, de ce fait, un aléa de données. Nous pourrions alors utiliser cette détection pour déterminer quels chemins de transfert à utiliser ou la période de suspension (le cas échéant) à appliquer pour assurer une exécution correcte des instructions. En pseudo-code, cela donne :

```
if (rs(n + 1) == rd(n) || rt(n + 1) == rd(n) && RegWEn(n) == 1) {
    /* transférer sortie de l'UAL de l'instruction n */
}
```

Aléas de branchement

Les aléas de branchement sont causés par les instructions de saut et de branchement. En effet, dans une exécution séquentielle sans branchement et sans saut l'instruction suivante se trouve à l'adresse spécifiée par PC + 4. Dans le cas des instructions de saut et pour certaines instructions de branchement, le PC suivant n'est pas PC + 4, mais sera le résultat du calcul effectué à l'étape EX. Une solution possible à ce problème est de suspendre le pipeline lors d'un aléa de branchement, mais cela impactera négativement les performances.

4.5. Outre la suspension du pipeline, que pouvons-nous faire pour résoudre les aléas de branchement ?
Nous pouvons essayer de prédire dans quelle direction les branchements iront, et lorsque cette prédiction est incorrecte, nous « vidons » le pipeline et continuons avec l'instruction

correcte. Une méthode simple (et naïve) de prédiction consiste à considérer que les sauts dans les instructions de branchement ne sont jamais effectués.

4.6. Combien d'aléas d'exécution y aurait-il dans le code MIPS ci-dessous s'il est exécuté dans un processeur en pipeline **sans** unité de transfert ? Quel est le type de chaque aléa d'exécution ? (Examinez toutes les paires d'instructions possibles pour des aléas éventuels).

Combien de fois faut-il suspendre le pipeline pour éliminer des aléas de données éventuels ? Quid des aléas de branchement ?

Cycles / Instructions	C1	C2	C3	C4	C5	C6	C7	C8	C9
`1. sub $t1, $s0, 1`	IF	ID	EX	MEM	WB				
`2. or $s0, $t0, $t1`		IF	ID	EX	MEM	WB			
`3. sw $s1, 100($s0)`			IF	ID	EX	MEM	WB		
`4. beq $s0, $s2, 1`				IF	ID	EX	MEM	WB	
`5. add $t2, $0, $0`					IF	ID	EX	MEM	WB

Nous avons quatre aléas d'exécution : entre les instructions 1 et 2 (aléa de donnée sur $t1), entre les instructions 2 et 3 (aléa de donnée sur $s0), entre les instruction 2 et 4 (aléa de donnée sur $s0), et entre les instructions 4 et 5 (aléa de branchement).

En supposant que nous pouvons lire et écrire dans le « banc de registres » pendant le même cycle d'horloge, deux suspensions sont nécessaires entre les instructions 1 et 2, et deux autres suspensions entre les instructions 2 et 3. Aucune suspension n'est nécessaire pour l'aléa de branchement, parce qu'il peut être géré avec la prédiction de branche / vidage du pipeline.