Institut
Mines-Télécom

Pipelining,

parallélisme et dépendances

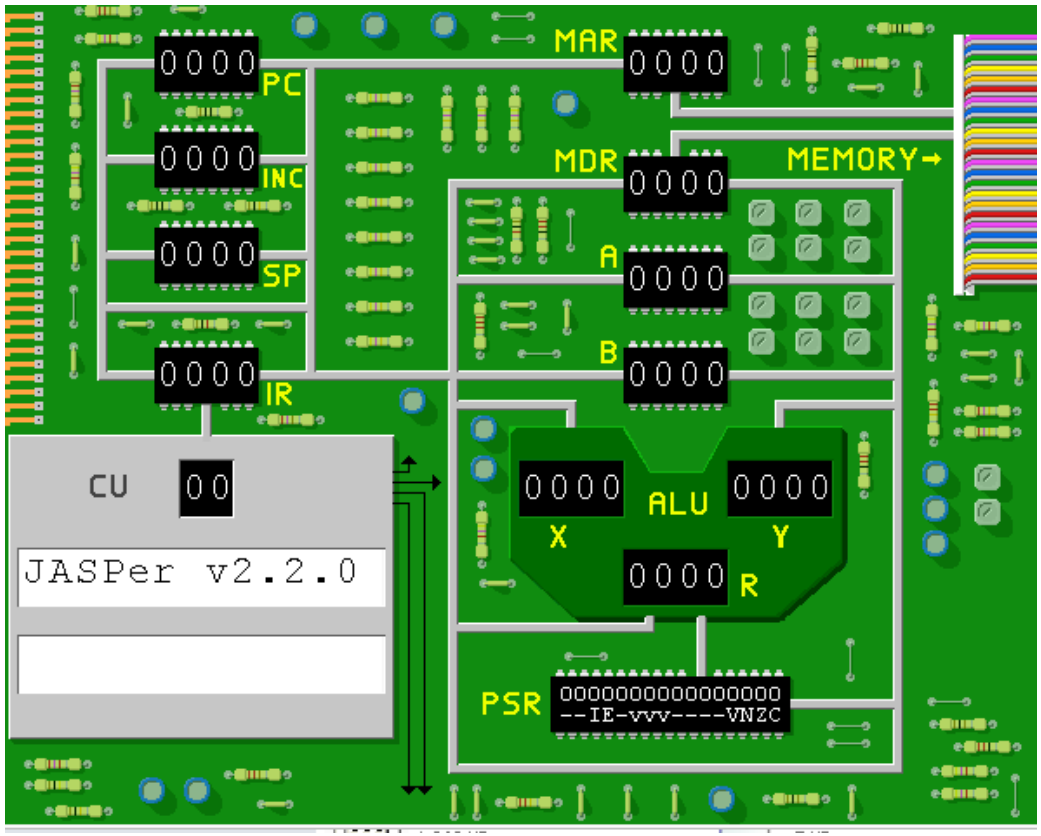
Intervenant : Florian Brandner

prénom.nom@telecom-paristech.fr

Le principe du pipeline

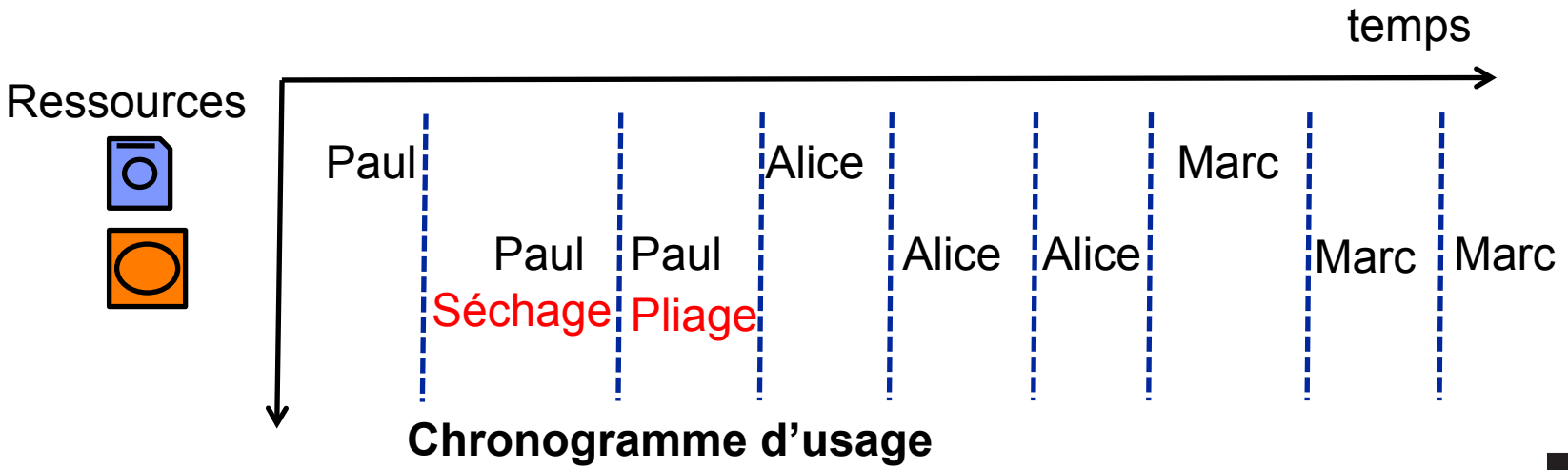
Mirco-Architecture et rendement

- Exécution d'un programme == traitement séquentiel de chaque instruction
- A un instant t une seule instruction en cours d'exécution
- 1 cycle = 1 transfert
- Pb : durée d'exécution d'un programme
- Instruction add 4,A => 10 cycles (cf TP)



Principe du pipelining dans une laverie (1)

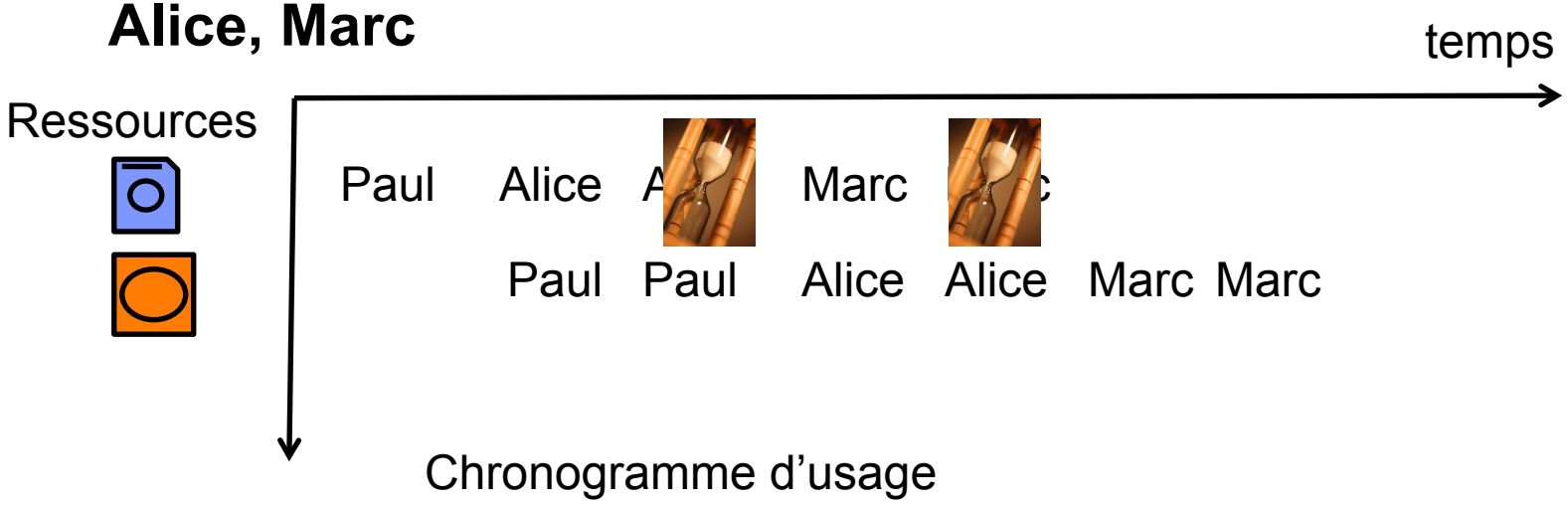
- **Structure 1 séchoir, 1 machine, 1 durée unique d'usage de 30 minutes, et il faut 30 minutes pour plier le linge.**
- **Cas naïf, la pièce est trop petite pour contenir + d'une personne.**
- **Les clients font la queue, 1 seule personne dans la pièce, et personne ne double : Paul, Alice, Marc**



Chronogramme d'usage

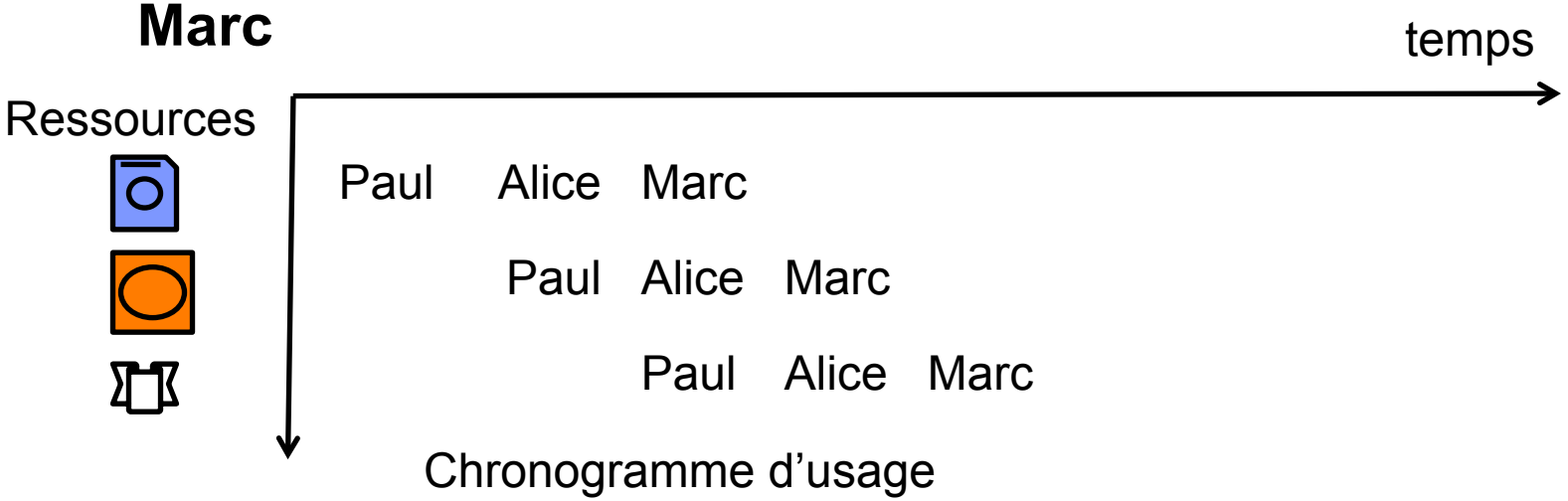
Principe du pipelining dans une laverie (2)

- **Structure 1 séchoir, 1 machine, 1 durée unique d'usage de 30 minutes**
- **Cas avancé n°1 , la pièce est assez grande pour 3 personnes, on suppose le temps de transfert de la machine au séchoir nul, le pliage se fait depuis le séchoir**
- **Les clients font la queue, et personne ne double : Paul, Alice, Marc**



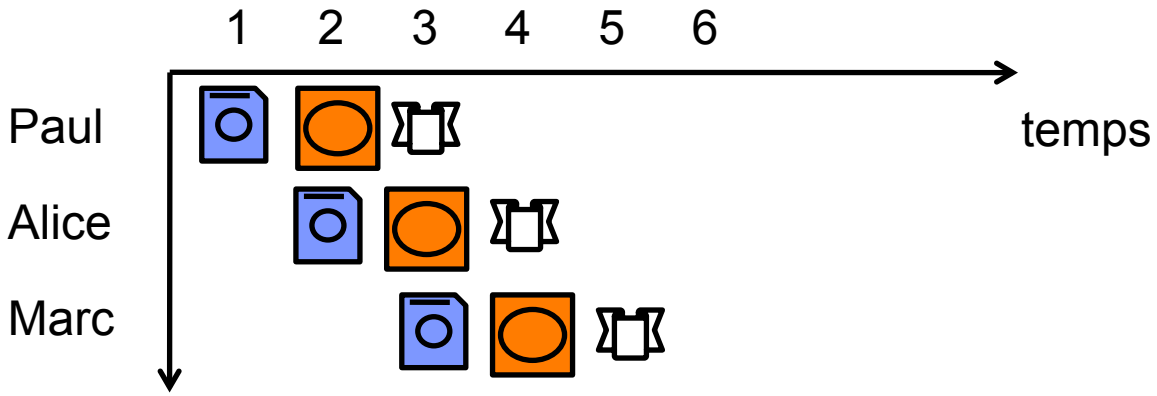
Principe du pipelining dans une laverie (3)

- **Structure 1 séchoir, 1 machine, 1 durée unique d'usage de 30 minutes**
- **Cas avancé n°2 , la pièce est assez grande pour 3 personnes, on suppose le temps de transfert de la machine au séchoir nul, le pliage se fait depuis une panière**
- **Les clients font la queue, personne ne double : Paul, Alice, Marc**



La représentation alternative utilisée en TD / examen

- **Structure 1 séchoir, 1 machine, 1 table de pliage**
- **1 durée unique de 30 minutes**
- **Amélioration 1 : la pièce est assez grande pour 3, séchoir + table**



Chronogramme d'exécution

En un « cycle » (1 colonne), chaque ressource de la salle est utilisée au plus 1 fois

Ce qu'il faut retenir ici

- Les ressources sont utilisées dans un certain ordre.
- Si une étape prend plus longtemps qu'une autre, elle bloque toutes les ressources antérieures (car on a pas l'équivalent de la corbeille …)
- On peut imaginer d'autre sources de délais : si l'utilisateur s'éclipse, il faut en plus aller le chercher dehors pour qu'il vide sa machine ou son séchoir …

Et le rapport avec les processeurs ?

- **Les ressources :**
  - Dans la laverie : machine à laver, séchoir, volume de la pièce, emplacement pour stocker le linge durant le pliage
  - Dans un processeur : l'ALU, l'interface avec la mémoire (e.g. MAR/MDR), l'unité de contrôle, les registres de travail,
- **Le cycle de fonctionnement :**
  - Dans la laverie : la durée d'un programme == plus petite durée commune d'usage des ressources (ici identique pour toutes)
  - Dans le processeur **: à discuter mais durée fixe**
- **Le diagramme d'étapes == enchaînement des activités couplées avec leur usage des ressources**
  - Dans la laverie : 3 étapes lavage (machine), séchage (séchoir), pliage (zone de stockage du linge :: séchoir ou panière)
  - Dans le processeur : **à définir…**

Et sur un processeur alors ?

- **Ressources == ALU, Interface avec la mémoire, CU, registres de travail…**
- **Séquence pour ADD 4, A**

1. MAR<- [PC]
2. PC<-[PC]+1
3. MDR<- [M[MAR]]
4. IR<- MDR

Récupération de l'instruction
==
FETCH

5. CU<- IR(opcode)
6. ALUy<-[IR(operand)]

Séparation opérandes/
opcode affectation CU
DECODE

7. ALUx<-[A]
8. ALUr=[ALUx]+[ALUy]
9. A<-[ALUr]

Exécution de l'instruction
(toutes les données sont
correctement positionnées
dans les registres

Activités et usage des ressources

instr	S1	S2	S3	S4	S5	S6	S7	S8	S9							
Add 3,A	1	2	3	4	5	6	7	8	9							
Add 4,A				1	2	3	4	4	4	5	6	7	8	9		

Observations :

- L'étape 1 modifie MAR
- L'étape 3 reçoit dans MDR => MAR doit rester fixe jusqu'à la fin de la transaction
- L'étape 4 altère IR
- Les étapes 5 et 6 lisent IR
- CU détermine où la sortie de l'ALU va, donc CU doit rester constant durant 7, 8, 9.

Leçon à en tirer ?

Parallélisme possible sur l'exécution des étapes de différentes instructions

si les étapes n'entrent pas en conflit sur l'usage des registres [1,3] /[4,x]

Abstraction de l'activité en étapes (ou étages)

Etapes de ADD 4, A

1. MAR<- [PC]
2. PC<-[PC]+1
3. MDR<- [M[MAR]]
4. IR<- MDR
5. CU<- IR(opcode)
6. ALUy<-[IR(operand)]
7. ALUx<-[A]
8. ALUr=[ALUx]+[ALUy]
9. A<-[ALUr]

PB : les transferts sont trop détaillées …
Pourtant il y a des similitudes … (couleurs)

Etapes de SUB B+addr,A

1. MAR<- [PC]
2. PC<-[PC]+1
3. MDR<- [M[MAR]]
4. IR<- MDR
5. CU<- IR(opcode)
6. ALUy<-[IR(operand)]
7. ALUx<-[B]
8. ALUr=[ALUx]+[ALUy]
9. MAR<-[ALUr]
10. MDR<-[M[MAR]]
11. ALUy<-[MDR]
12. ALUx<-[A]
13. ALUr=[ALUx]-[ALUy]
14. A<-[ALUr]

Les étapes clés du pipeline

- **A retenir : ce ne sont pas des noms officiels, ils peuvent légèrement changer (mais l'idée reste la même).**
- **Usuellement**
  - Instruction Fetch (F) : récupération d'une instruction depuis la mémoire
  - Instruction Decode (D) : décomposition de l'instruction entre opcode, opérandes et placement des données dans les registre adéquats
  - Instruction Execute (E) : exécution à proprement parler (l'op code est connu et les opérandes a priori en place
- **Lorsque l'accès à la mémoire est beaucoup plus long qu'un cycle une étape lui est dédié**
  - Memory access (M) : pour load/store et puis d'autre transferts plus exotiques
- **Le transfert final vers un registre de travail**
  - Write back : placement du résultat de l'opération dans le registre prévu par l'instruction.

Temps d'exécution et pipeline

- **Question : A quelle fréquence transitent les données via les étages ?**
- **Observation : + simple si les données circulent à la même vitesse dans le pipeline**
- **Durée d'un étage = temps de traversée des registres d'entrée vers les registres de sortie**
- **Pas si simple : repensez à l'exemple de Jasper**
  - Durée de la phase execute de Sub addr+(A), B
  - ⇒Si 1 unité temps par étape =>6 étapes
  - Durée de la phase execute d'un add #data, A
  - ⇒Si 1 unité temps par étape =>2 étapes
- **Définition cycle de fonctionnement du pipeline**
  - Calcul le délai de propagation entrées/sorties pour chaque étage
  - Faire progresser les données dans le pipeline à une vitesse inférieure.

Pipeline et cycles d'horloges

- **Entre chaque étape : des registres de stockage permettant de mémoriser les données propre à l'étage**
- **Le séquenceur pilote et planifie le déplacement des données d'étages en étages à une certaine fréquence**
- **Rappel : cycle == durée fixe caractéristique de l'évolution de l'état du processeur.**
- **Cas simple :**
  - chaque étape requiert 1 seul cycle pour s'exécuter
  - Interprétation naïve du pipeline avec N étages :
  - 1 instruction => N cycles
  - P instructions => P* N cycles sans le pipeline

    => N+P-1 avec ….

REDUCTION DU TEMPS D ' EXECUTION

Cas plus réaliste : étage de durée variable

- Idée : E ou M possèdent une durée variable fonction de l'OpCode
- Décomposition de E en E1, E2 …. En (cf TD)

- Pour le moment seul la durée varie, toue les étages restent obligatoires et « personne » ne double

- Pour un comportement plus riche cf Superscalaires.

Les bons et mauvais pipeline : se rappeler que c'est compliqué !

Propagation et duplication des registres dans les blocs

- **Soit R un registre utilisé par de nombreux étages (e.g. RI)**
- **Principe : si lors de l'exécution d'une instruction le registre interne R sert dans plusieurs étages (par exemple RI)**
  - on duplique le registre
  - chaque version aura une valeur différente car utilisée pour traiter une instruction différente
  - Lorsque l'on propage les données dans le pipeline on transfert le contenu de ce registre à l'étage suivant

Notation : chaque copie est désignée par NomEtage.NomRegistre : F.RI, E.RI

Ainsi à t=3, F.RI=i3, D.RI=i2, E.RI=i1;
i1..3 trois instructions consécutives

La concurrence dans le pipeline (I)

- **N étages s'exécutant en parallèle et accèdent à :**
  - des copies locales de registres clés (e.g. RI)
  - des registres partagés (e.g. registre de travail, où accès à la mémoire)

- **Objectif :**
  exécution du code C depuis l' état S0, donne la même séquence d'instructions et le même état final que dans le cas non-pipeliné.

La concurrence dans le pipeline (Ii)

- **Risques :**
  - de situations de « data race » sur un registre partagé R
  - De non respect de précédence « lecture/ écriture » sur R
- **Vocabulaire : situation à risque == hazard**

- **Donnée stockée dans R ≠ attendu, les conséquences :**
  - Cas 1: séquence d'instruction identique mais résultat incorrect == défaillance en valeur
  - Cas 2: le contenu de R entraine un changement de la séquence d'instruction exécutée attendue

Exécution du pipeline, le cas idéal … et le reste

- **Le cas parfait : les étages n'ont aucun problèmes de synchronisation, mais *que se passe-t-il en vrai ?***

instructions	T=1	2	3	4	5	6	7	8	9	10	11	12
instr1	F	D	E	M	W							
instr2		F	D	E	M	W						
instr3			F	D	E	M	W					
instr4				F	D	E	M	W				
instr5					F	D						

- Conflit structurel : requête sur le même bus mémoire (1 seul registre d'adresse, accès exclusif => 1 perdant forcément …)
- Dépendance de donnée : *Décode* attend la mise à jour d'un registre
- Problème du aux branchement : le résultat de la phase E rend caduc les fetch/decode déjà réalisés

Conflit structurel d'accès à la mémoire

- **Solution 1 : Duplication des bus (cf TD mémoire)**
- **Solution 2 : insertion de pauses (stall)**
  - Un dispositif est mis en place pour détecter ces conflits
  - Lors de la détection d'un conflit, la priorité est donnée aux actions permettant de terminer des instructions
  - L'étage en conflit le plus en amont du pipeline conserve son état précédant
  - Tous les étages en amont de ce dernier font de même.
  - Les étages en aval poursuivent leur exécution
- **Conséquence :**
  - L'étage vient de retarder le traitement conflictuel
    - Soit la situation se reproduit (et on réinjecte une bulle)
    - Soit le conflit structurel ne se reproduit pas et l'étage réalise l'opération

Effet bulle d'air en pratique (cas du conflit structurel)

instructions	T=1	2	3	4	5	6	7	8	9	10	11	12
instr1	F	D	E	M	W							
Instr2		F	D	E	M	W						
Instr3				F	F	D	E	M	W			
instr4					(rien)	F	D	E	M	W		

- **Conflit structurel en 4 : l'étage F est mis en pause**
- **Une pause s'appelle une pénalité**
- **Lorsque un étage est mis en pause souligné sa « lettre »**
- **Attention à ne pas confondre (rien) F (cf cycle 5)**

Analyse des dépendances de données identification des problèmes

- Ordre des écritures sur un registre incorrect (violation du Write after Write, WAW)
- Une lecture réalisée sur une vielle valeur d'un registre (violation du Read after Write, RAW)
- Une écriture réalisée sur une valeur déjà écrasée (violation du Write after Read,WAR)
- Astuce : un conflit XAY signifie que la contrainte de précédence X avant Y n'est pas respectée.
- A quoi cela correspond il concrètement ?

Le cas Read after Write (si non résolu)

- **Hypothèse :**
  - R1 et R2 sont des registres de travail contenant 2 et 3 respectivement
  - Format : opcode op1 op2 = op2<- op1 #opération# op2
  - 2 registres intermédiaires sont en entrée de Execute pour stocker les opérandes transmises par l'étage Decode
  - 1 étage Memory pour les accès mémoires et 1 étage WB pour placer le résultat de l'instruction dans les registres de travail

instructions	T=1	2	3	4	5	6	7	8	9	10	11	12
mul R1,R2	F	D	E	M								
mul 4, R2		F	D	E	M							

  - Résultat attendu : 24, obtenu 12 ….

Résolution de dépendance de données par l'ajout de retards

- Les dépendances de données peuvent toute être résolues en ajoutant des pauses à l'exécution des instructions en amont dans le pipeline
- Cas du RAW sur le DLX (cf TD)
  (le transfert W→R→D peut se faire en 1 cycle)

instructions	T=1	2	3	4	5	6	7	8	9	10	11	12
mul R1,R2	F	D	E	M	W							
mul 4, R2		F	D	D	D	E	M	W				

Résolution conflit de données : Anticipation ou Bypass (par Forwarding)

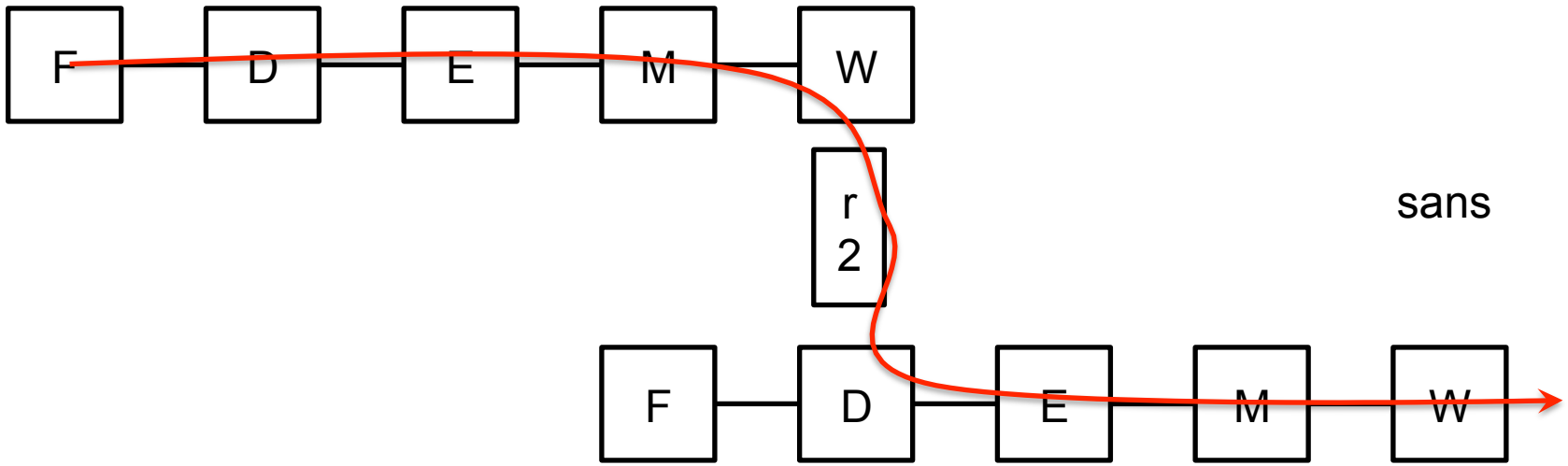
- **Pour exécuter :**

  I1:add r1, r2

  I2:mul 4 , r2

instructions	T=1	2	3	4	5	6	7	8
i1	F	D	E	M	W			
i2		F	D	D	D	E	M	W

- **Idée: utiliser un raccourcis entre les sorties de E ou M, et les entrées de E pour l'opérande de I2**



Résolution conflit de données : Anticipation ou Bypass (par Forwarding)

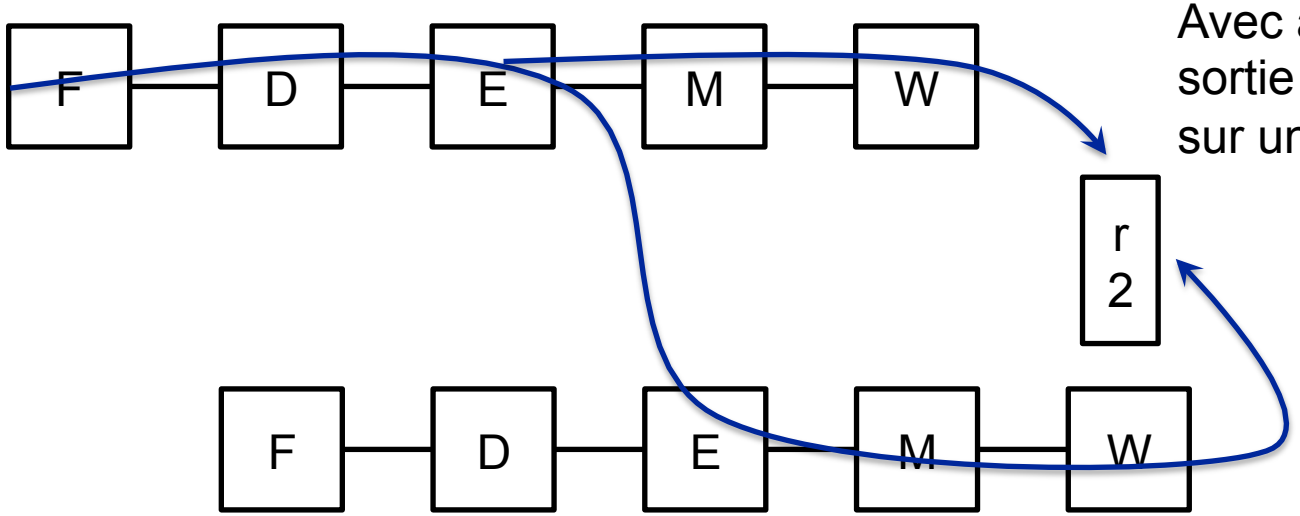
- **Pour exécuter :**

  I1:add r1, r2

  I2:mul 4 , r2

instructions	T=1	2	3	4	5	6	7	8
i1	F	D	E	M	W			
i2		F	D	E	M	W		

- **Idée: utiliser un raccourcis entre les sorties de E ou M, et les entrées de E pour l'opérande de I2**



Avec anticipation:
sortie de E envoyée
sur une de ses entrées

Branchement : problème

- Instruction de branchement inconditionnel : jmp @3
- Code en mémoire :

@	instructions	Description
@3	bmi @2+3	Va à @2+3 si flag négatif vrai
@3+1	sub 4,R2	
@2+2	jmp @3	
@2+3	mul R2, R2	Met le flag N à faux

@ + contexte	instructions	T=1	2	3	4	5	6	7
@3+1 et R2= 2	sub R1,R2	F	D	E	M	W		
@3+2	jmp @3		F	D	E	M	W	
@3+3	mul R2,R2			F	D	⦸		
@3+4					F	⦸		
@3	bmi @2+3					F		

- PB : @ destination branchement prise en compte dans E=> ne pas exécuter @3+3 et @3+4
- Encore + dur avec les branchements conditionnels

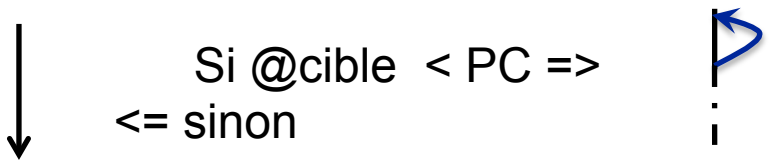
Détails des problèmes & Solutions

- Pb 1: éviter de compléter l'exécution des instructions parasites
- R: vider le contenu des étages en amont de E
- Pb 2: éviter les pénalités dues aux retards sur la définition de PC.
- R: prédire l'instruction suivant le branchement et la Fetcher dès que possible
- Pb 3 : éviter les effets de bord des instructions parasites (e.g. effet sur Flags si étage E exécuté)

Limiter les pénalités de branchement :
Prédiction de branchement

- **Idée :**
  - Solution 1 : faire un pronostic sur le futur (prédiction) et exécuter la branche sélectionnée
  - Solution 2 : Laisser s'exécuter ce qui est rentré dans le pipeline
- **Solution 1 très populaire :**
  - Prédiction statique (indépendant de l'exécution)

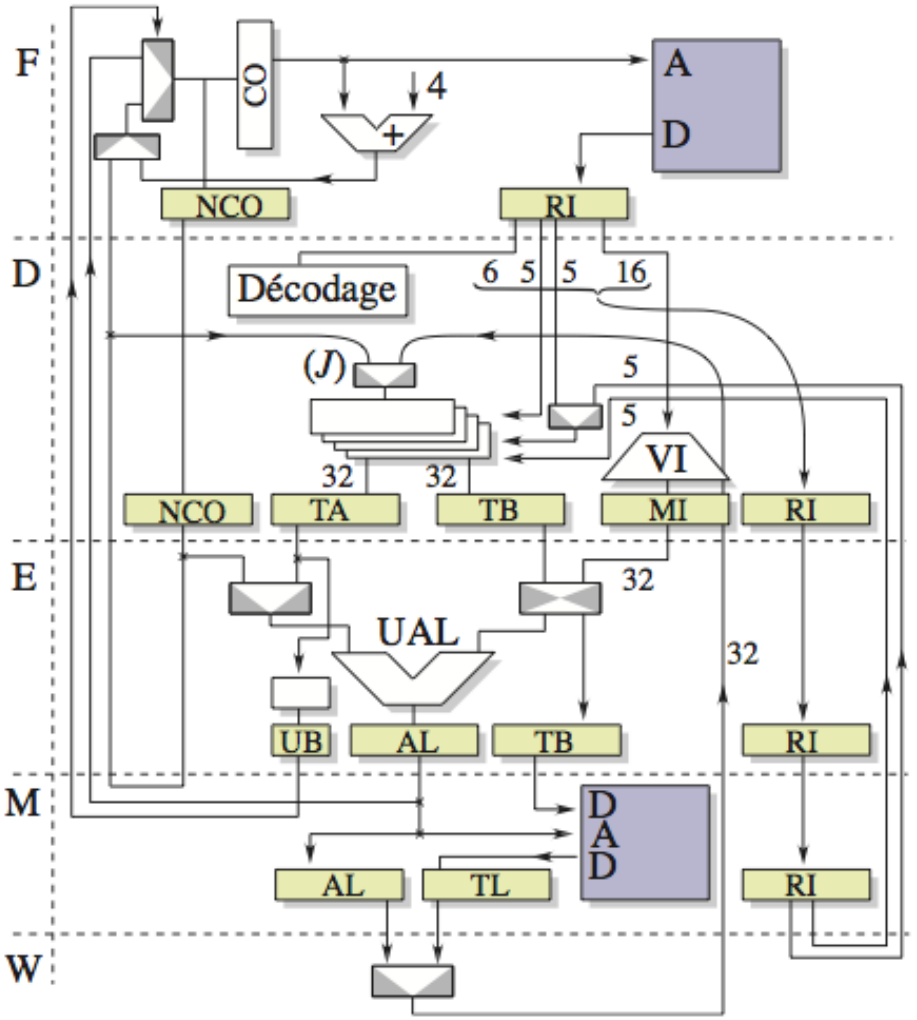
  - Prédiction dynamique (dépend de l'exécution) : 1 exemple



Étude de cas DLX

La micro-architecture

- **Processeur RISC**
- **Séparation accès mém/opérations**
- **Pipeline d'exécution séquentiel en 5 étapes**
  - Fetch
  - Decode
  - Execute
  - Memory
  - WriteBack



Les relations de transferts à chaque étage

Etage	Types d'instructions		
	Toutes les instructions		
F	F.RI ← MEM(CO) si (opcode(E.RI)=branch et UB) alors E.AL sinon CO+4		
D	TA ← R(F.RI(6:10)) D.TB ← R(F.RI(11:15)) D.NCO ← F.NCO D.RI ← F.RI D.MI ← F.RI(16:31) avec extension de signe	2)	
	Instruction UAL	Load /Store	Branch
E	E.RI ← D.RI E.AL ← (TA op D.TB) ou (TA op MI) UB ← 0	E.RI ← D.RI E.AL ← TA+MI UB ← 0 E.TB ← D.TB	E.AL ← D.NCO+MI UB ← (D.TA op 0)
M	M.RI ← E.RI M.AL ← E.AL	M.RI ← E.RI TL ← MEM(E.AL) ou MEM(E.AL) ← E.TB M.RI ← E.RI	
W	$R(M.RI_{16:20}) \leftarrow M.AL$ ou $R(M.RI_{11:15}) \leftarrow M.AL$	$R(M.RI_{11:15}) \leftarrow TL$ (seulement pour load)	1)

Détail du branchement / Calcul pour anticipation

- **Execute:**

(a) Référence mémoire : AL:=TA+MI; (calcul d'adresse)
(b) Opération registre à registre : AL:=TA Op TB;
(c) Opération registre, valeur immédiate : AL:=TA Op MI;
(d) Branchement : AL:=NCO+MI; COND:= TA Op 0;

- **4. Memory:**

(a) Référence mémoire : TL:=Mem[AL];
ou Mem[AL]:=TB;
(b) Branchement : Si COND M.CO:=AL sinon M.CO:=NCO;

Anticipation : ne pas attendre le Write pour répercuter les valeur en sortie de E ou M vers les entrées de E.