CpE 313: Microprocessor Systems Design

Handout 06
Pipelining - Hazards

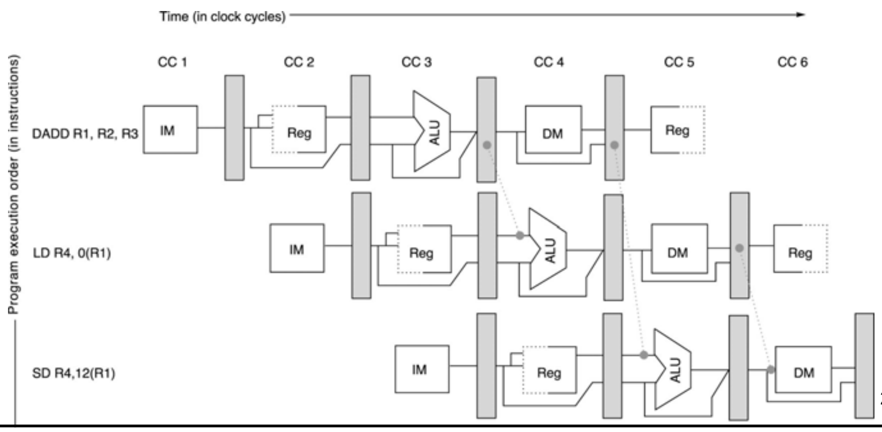
September 16, 2004
Shoukat Ali

shoukat@umr.edu

UMR UNIVERSITY OF MISSOURI-ROLLA
The Name. The Degree. The Difference.

Another Forwarding Example

- dadd R1,R2,R3
  ld R4,0(R1)
  sd R4, 12(R1)
- list all dependencies



Data Hazards: What We Have Done So Far?

- data hazard
  - an instruction *x* wants to read a register that an *earlier* instruction yet has to write

- software solution
  - compiler inserts enough no-ops BEFORE instruction *x* to avoid prevent *x* from reading the old data
    - correctness is achieved, but at the cost of performance
- hardware solution
  - hardware detects data hazard and forwards/bypasses the missing item from internal register (instead of waiting to get the item from register file)
    - ***how does the hardware detect data hazard?***

Data Hazard Detection

```
sub     $2, $1, $3
and     $12, $2, $5
or      $13, $6, $2
add     $14, $2, $2
sw      $15, 100($2)
```

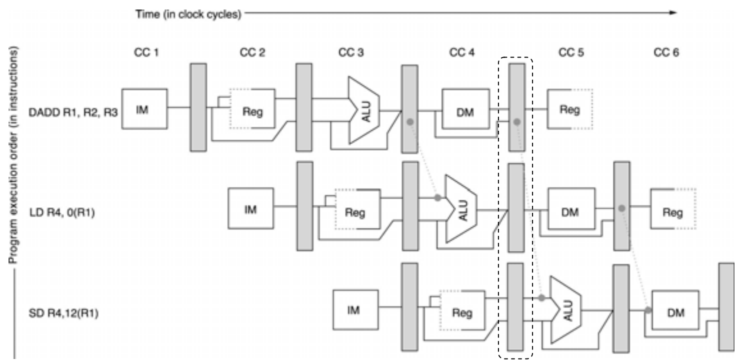
- instruction *x* wants to read a register that an *earlier* instruction *y* yet has to write
- for data hazard, how many instructions earlier *y* could be and still pose a data hazard?
  -
- *every instruction **x**, before **reading its source registers**, tries to determine if either of the two earlier instructions wants to write to one of **x**'s source registers*
  - if yes, there is a data hazard
  - if no, the instruction ***x*** can go ahead and read its registers
- that is, data hazards will exist if
  - (y+1)'s source register = y's dest register
  - (y+2)'s source register = y's dest register

Detecting Data Hazards: Specifics

- Is y’s dest register = (y+1)’s source register?
  - Is EX/MEM.RegisterRd = ID/EX.RegisterRs?
    Is EX/MEM.RegisterRd = ID/EX.RegisterRt?

{ register addresses, not contents!

- Is y’s dest register = (y+2)’s source register?
  - Is MEM/WB.RegisterRd = ID/EX.RegisterRs?
    Is MEM/WB.RegisterRd = ID/EX.RegisterRt?

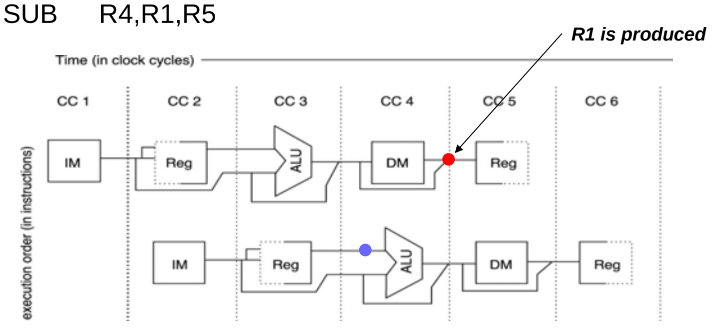


Hardware Needed For Forwarding

- a 32-bit path from EX/MEM to ID/EX
- a 32-bit path from MEM/WB to ID/EX
- additional control unit functionality to check for two conditions on previous slide

Hazards That Cannot Use Forwarding

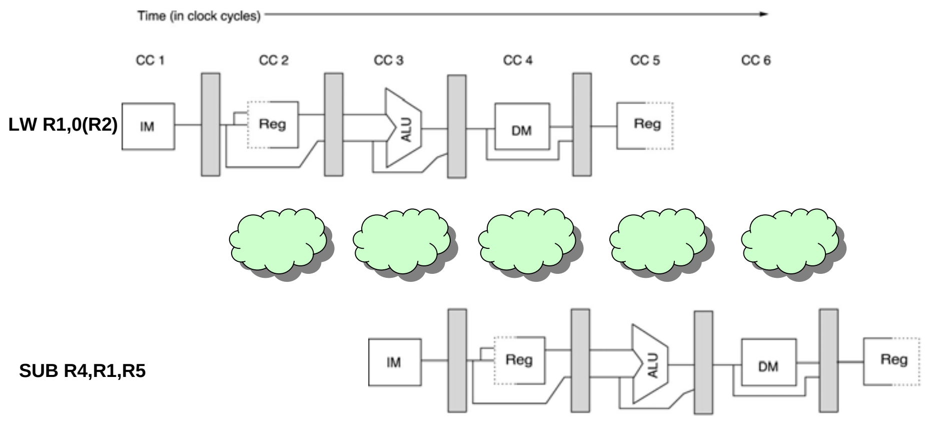
LW R1,0(R2)
SUB R4,R1,R5



- only solution: must *stall* sub instruction, i.e., must start it one cycle later
- "pipeline interlocking"
  - hardware that detects a hazard and stalls the pipeline until the hazard is clear

7

Pipeline "Bubble"



after the lw, we stall the pipeline for one clock cycle unless ....

8

Slide 12, Lecture 3: Register-Register Is Flexible!

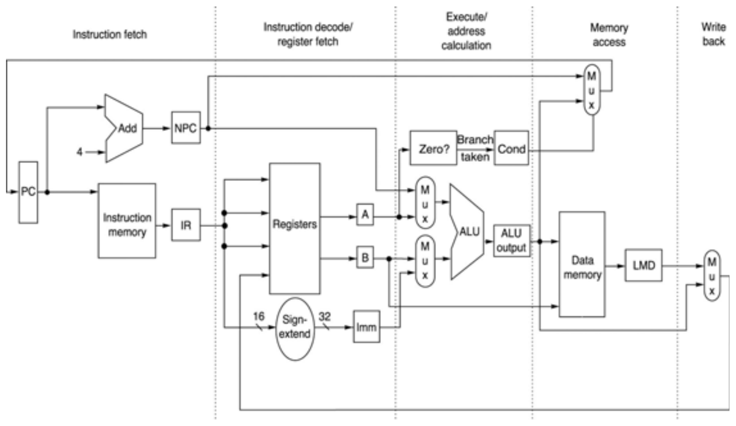
stack	accumulator	register-register	register-register	register-register
push B	load C	load R1, B	load R2, C	load R2, C
push C	mult D	load R2, C	load R1, B	load R1, B
push D	add B	load R3, D	load R3, D	load R3, D
mult	store A	mult R4,R2,R3	mult R4,R2,R3	sub R6, R1,R2
add	load B	add R5, R4,R1	add R5, R4,R1	add R7, R6,R3
pop A	sub C	store R5, A	store R5, A	store R7, E
push D	add D	sub R6, R1,R2	sub R6, R1,R2	mult R4,R2,R3
push C	store E	add R7, R6,R3	add R7, R6,R3	add R5, R4,R1
push B		store R7, E	store R7, E	store R5, A
sub				
add				
pop E				

for load/store arch, some instructions can be re-ordered by compiler. This is important for caches and pipelining! We will see later.

9

Control Hazard

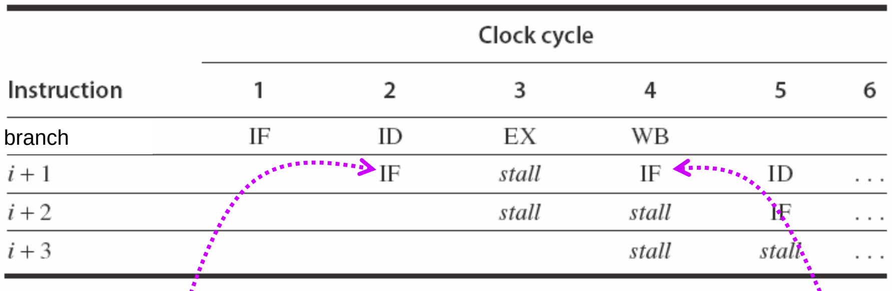
- pipeline must be fed a new instruction every cycle for full performance
- what instruction should be fed in pipeline after a branch?
  -
  -



10

Control Hazard - Stalls

- can not issue next instruction until branch decision is clear
  - will cause a *two-cycle stall*
    - CPI_{branch} = 3 ……………..can we do better?



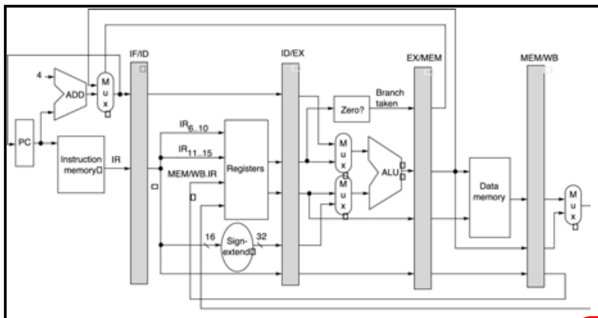
Instruction	Clock cycle 1	2	3	4	5	6
branch	IF	ID	EX	WB		
i + 1		IF	*stall*	IF	ID	. . .
i + 2			*stall*	*stall*	IF	. . .
i + 3				*stall*	*stall*	. . .

Figure S.44 Effects of a taken conditional branch on the pipeline.

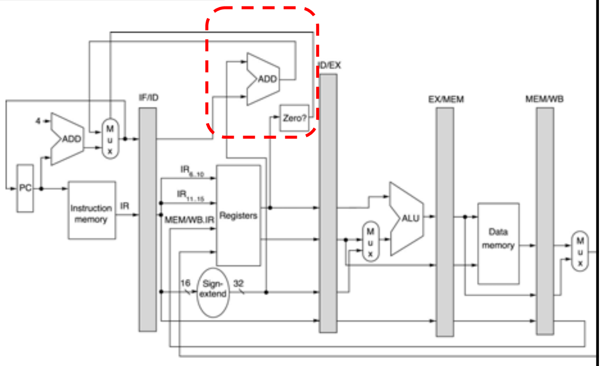
PC+4 (“not-taken” path) fetched automatically.

After EX of branch, both the target and branch direction are known. At that time, fetch is restarted to get branch target. One cycle of “work” is wasted + 1 stall.

11



Modified Pipeline: Move Branch Target Back

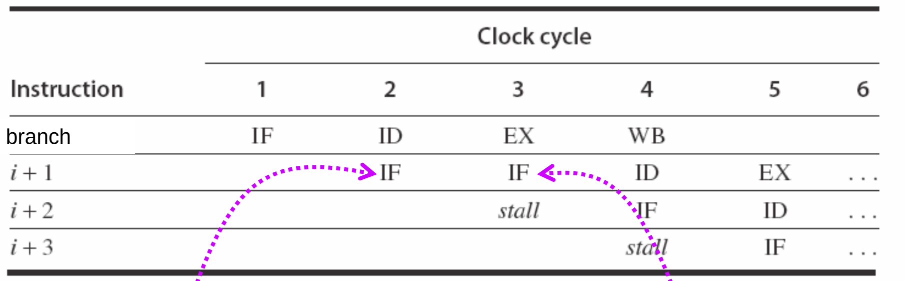


Extra adder, for address computation + moving back of ZERO unit eliminates 1 cycle, but we still must stall by 1 cycle

12

Reducing Branch Stalls: Summary Scheme 1

- move up decision to 2nd stage by adding more hardware
- improvement: one-cycle stall per branch instruction instead of two-cycle stall as before
  - $\text{CPI}_{\text{branch}} = 2$



	Clock cycle					
Instruction	1	2	3	4	5	6
branch	IF	ID	EX	WB		
$i+1$		IF	IF	ID	EX	...
$i+2$			*stall*	IF	ID	...
$i+3$				*stall*	IF	...

Figure S.43 Effects of a jump or call Instruction on the pipeline.

PC+4 (“not-taken” path) fetched automatically.

After ID of branch, real target is known so fetch is restarted. One cycle of “work” is wasted.

Impact of Stalls

$$\text{speedup} = \frac{\text{unpipelined time}}{\text{pipelined time}} = \frac{\text{CPI}_{\text{unpipe}} \times \text{CT}_{\text{unpipe}}}{\text{CPI}_{\text{pipe}} \times \text{CT}_{\text{pipe}}}$$

$$\text{CPI}_{\text{pipe}} = \text{CPI}_{\text{no-stall-pipe}} + \text{stall cycles per inst}$$

$$\text{speedup} = \frac{\text{CPI}_{\text{unpipe}} \times \text{CT}_{\text{unpipe}}}{\text{CPI}_{\text{pipe}} \times \text{CT}_{\text{pipe}}} = \frac{\text{CPI}_{\text{unpipe}}}{1 + \text{stall cycles per inst}}$$

$$= \frac{n}{1 + \text{stall cycles per inst}}$$

$$= \frac{n}{1 + \text{inst freq} \times \text{inst stall cycles}}$$

Example

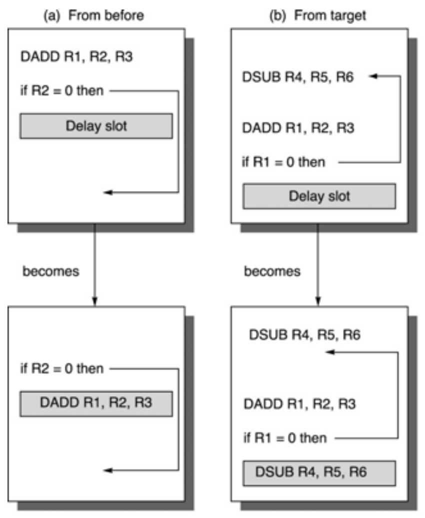
- determine speedup for a program where 20% of the instructions are branches
  - normal pipeline
  - move-target-back pipeline

- with no scheme, branch penalty or stall = 2
- with scheme 1, branch penalty =1

Reducing Branch Stalls: Scheme 2

- while waiting for branch decision, execute some other “safe” instruction
  - such a branch called a “delayed branch”
  - as opposed to a “predicted branch”
    - will see later
- what to fill in “branch delay slot”?

- instruction in delay slot ALWAYS gets executed regardless of branch outcome



Reducing Branch Stalls: Scheme 3

- improvement:
  - 1-cycle stall per branch instruction if compiler can find one “safe” instruction to put in “delay slot”
    - CPI_{branch} = 2
  - 0-cycle stall if can find two “safe” instructions
    - CPI_{branch} = 1

- typically can fill:
  - 1 slot 50% of time
  - 2 slots about 25% of time
  - >2 slots almost never

17

Other Types of Data Hazards

- *RAW (Read after Write)*
  - *later instruction, B, tries to read source register before it is written by an earlier instruction, A*
- WAR (Write after Read)
  - B tries to write an operand before it is read by A
  - Occurs if pipeline allows late register reads
  - Only happens in some pipeline architectures
    - Ex: Skip over MEM stage if not using memory

```
                                                        Reads R2 during MEM1
SW   0(R1),R2    IF       ID       EX       MEM1     MEM2     WB
ADD R2, R3,R4             IF       ID       EX       WB
                                                    writes R2 during WB
```

  - Note: WAR only happens because R2 is being reused to hold a second value
    - If registers are never reused, WAR doesn’t happen

18

Types of Data Hazards (cont.)

- WAW (Write after Write)
  - B tries to write to the same register as A
  - Result: Later instructions see wrong value in the register
  - Occurs if instructions can write register file out-of-order
  - This also doesn't happen in MIPS, but happens in other pipelines

```
                                                    WB writes 1st version of R1
LW   R1,0(R2)    IF        ID        EX        MEM1     MEM2    WB
ADD R1, R2,R3              IF        ID        EX       WB
                                                    WB writes 2nd version of R1
```

the above is NOT the MIPS pipeline!

Types of Data Dependencies

- True dependence (pure-dependence, flow-dependence)
  - ADD R1, R2,R3
  - SUB R4, R5,R1
  - ***May*** cause RAW hazards
- Anti-Dependence
  - ADD R3, R2,R1
  - SUB R1, R4,R5
  - May cause WAR hazards
  - Due to reuse: Removed by using another register
- Output-Dependence
  - ADD R1, R2,R3
  - SUB R1, R4,R5
  - May cause WAW hazards
  - Due to reuse: Removed by using another register