ECSE 425 - Tutorial 4

Hazards

Hazards

- Structural hazards: multiple stages can't run at the same time because they share a resource
- Data hazards: adjacent instructions use results not yet produced/saved
- Branch hazards: you need to jump in the instruction flow but only figure out some number of cycles later

Hazards

- Structural hazards: multiple stages can't run at the same time because they share a resource

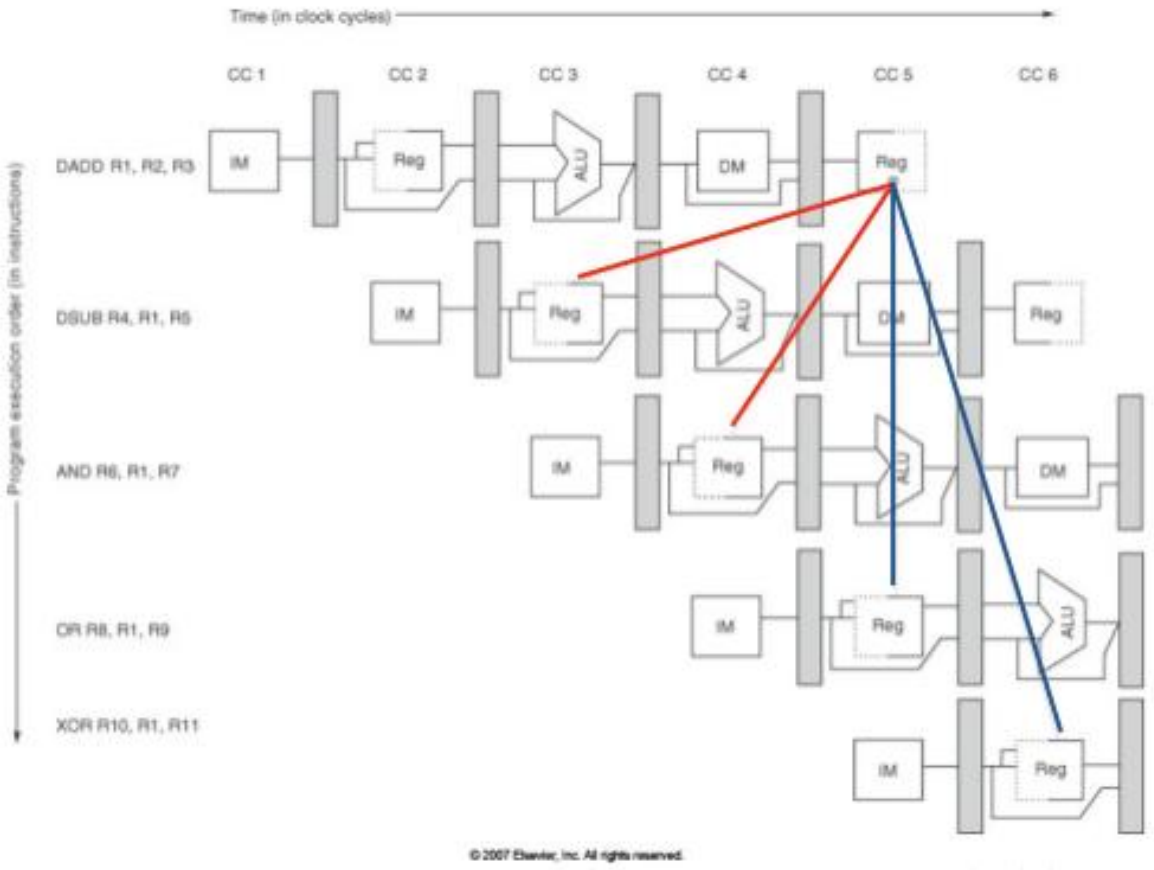
CC#	1	2	3	4	5	6	7	8
i1	IF	ID	EX	MEM	WB			
	…							
i3			IF	ID	EX	MEM	WB	
i4				stall	IF			

MEM : retrieve/store data from/to **memory**
IF : retrieve instruction from **memory**

Requires a memory capable of two accesses per clock cycle -> expensive!
Solution? Two separate memories

Hazards

- Data hazards: adjacent instructions use results not yet produced/saved

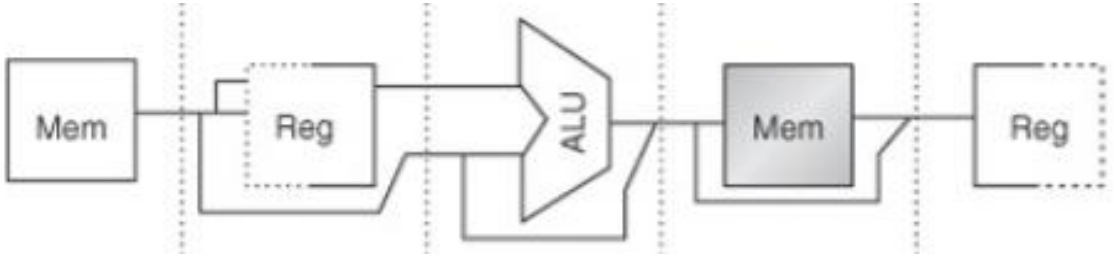


DADD R1, R2, R3
DSUB R4, **R1**, R5
AND R6, **R1**, R7
OR R8, **R1**, R9
XOR R10, **R1**, R11

Classic 5-stage Pipeline

When is data actually produced or needed?

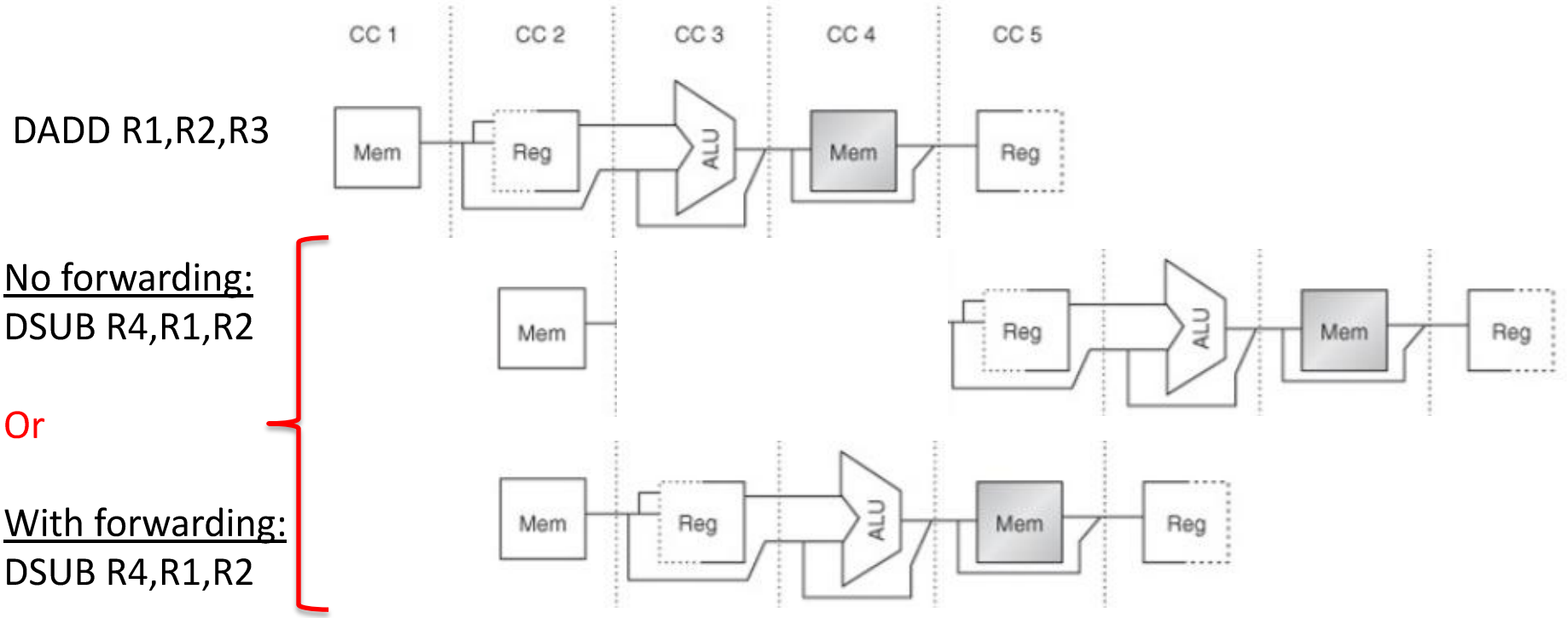
Instruction	Data needed	Data produced
Arithmetic/ Logical	**Before the ALU stage** e.g.: DADD R1, **R2**, **R3**	**After the ALU stage** e.g.: DADD **R1**, R2, R3
Load/ Store	**Before the ALU stage** e.g.: SD **R1**, 0(**R2**) e.g.: LD **R1**, 0(**R2**)	**After the MEM stage** e.g.: LD **R1**, 0(R2)
Branch	**Before the ID stage** e.g.: BNEZ **R1**, loop	N/A



Hazards

- Data hazards: adjacent instructions use results not yet produced/saved

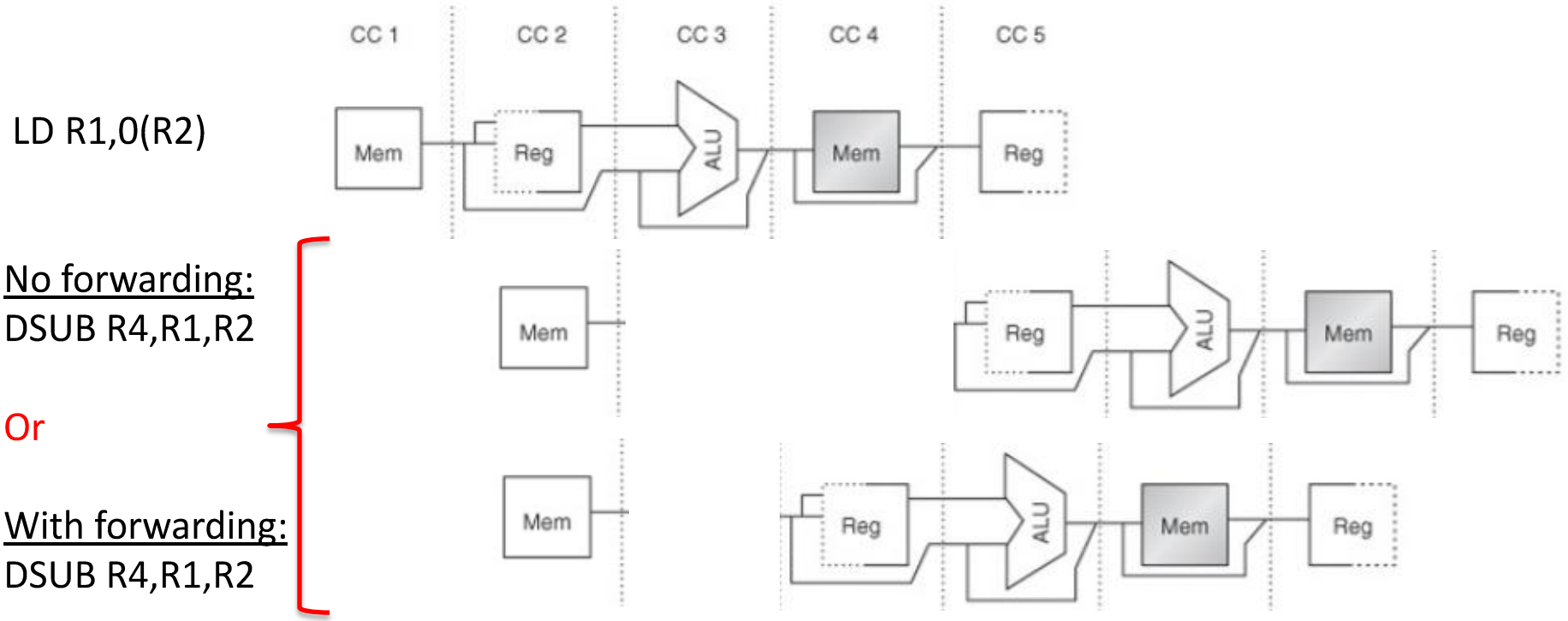
Solution? **Data forwarding**!



Hazards

- Data hazards: adjacent instructions use results not yet produced/saved

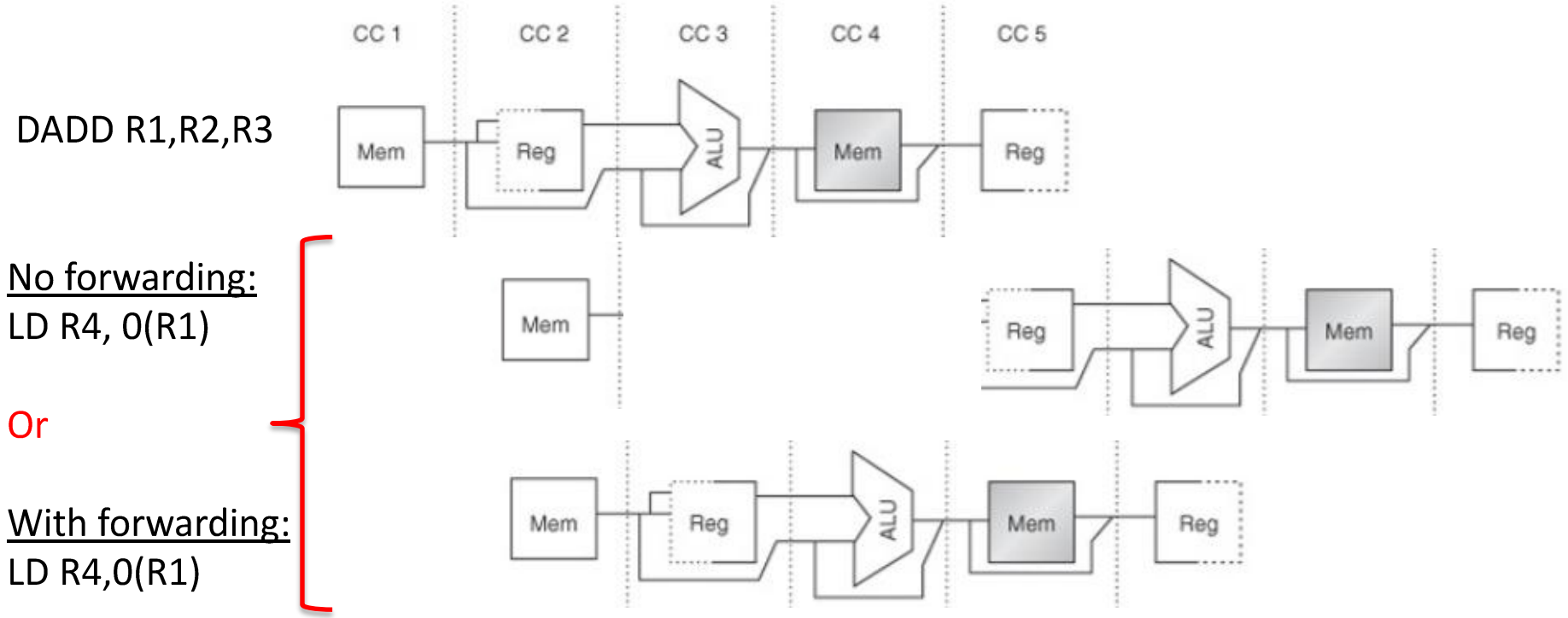
Solution? **Data forwarding**!



Hazards

- Data hazards: adjacent instructions use results not yet produced/saved

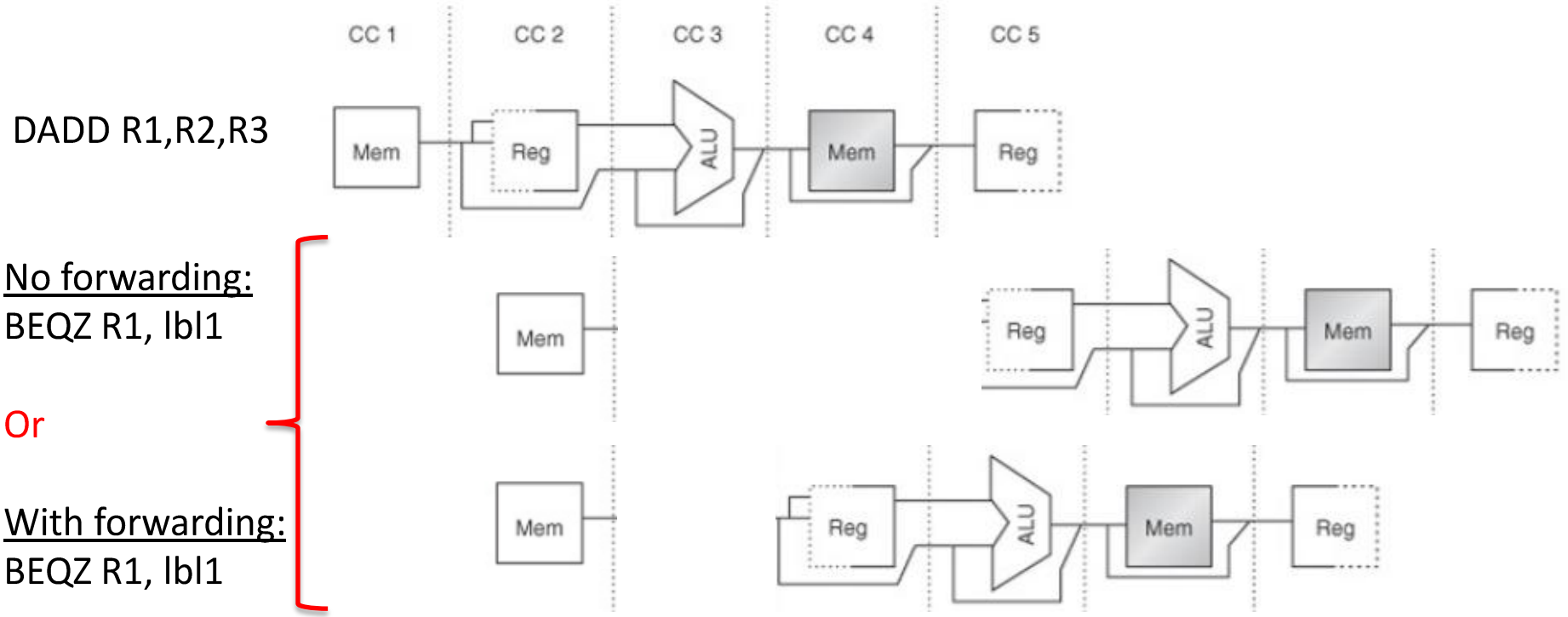
Solution? **Data forwarding**!



Hazards

- Data hazards: adjacent instructions use results not yet produced/saved

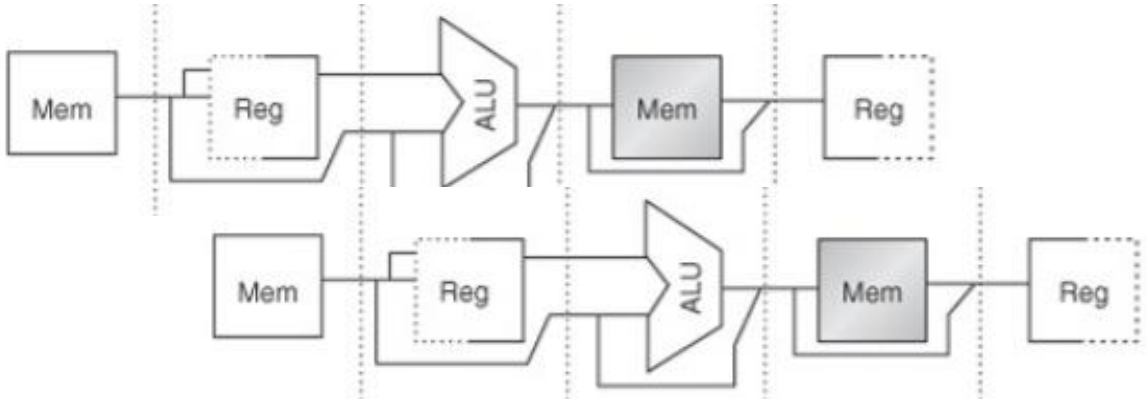
Solution? **Data forwarding**!



Hazards

- Branch hazards: you need to jump in the instruction flow but only figure out some number of cycles later

```
        DADD R8, R2, R3
        BEQZ R1, LBL1
        DADD R2, R4, R5
        …
LBL1:   DADD R4, R2, R4
```



Problem! 1 cycle stall required

Branch strategies

- Consider this code:

```
        DADD   R8,R2,R3
        BEQ    R2,R4,L1
        DADD   R2,R4,R5
        ...
L1:     DADD   R4,R2,R4
```

DADD R2,R4,R5 should only un if the branch is <u>not taken</u>

How do we deal with this?

- **Freeze/flush the pipeline -- Figure A.11**
- **Predicted-not-taken scheme – Figure A.12**
- **Predicted-taken scheme**
- **Branch delay slot – Figure A.13**

Branch strategies – Freeze pipeline

In both cases, we incur a 1-cycle penalty on a branch instruction

- Branch taken

```
cc                          1   2   3   4   5   6   7   8
     DADD   R8,R2,R3   IF  ID  EX  ME  WB
     BEQ    R2,R4,L1       IF  ID  EX  ME  WB
     DADD   R2,R4,R5           IF  ignored
L1:  DADD   R4,R2,R4               IF  ID  EX  ME  WB
```

- Branch not taken

```
cc                          1   2   3   4   5   6   7   8
     DADD   R8,R2,R3   IF  ID  EX  ME  WB
     BEQ    R2,R4,L1       IF  ID  EX  ME  WB
     DADD   R2,R4,R5           IF  ignored
     DADD   R2,R4,R5               IF  ID  EX  ME  WB   restart
     ...
```

Branch strategies – predict-not-taken

We incur a 1-cycle penalty only if the branch is taken

- Branch taken

```
cc                          1  2  3  4  5  6  7  8
     DADD  R8,R2,R3    IF ID EX ME WB
     BEQ   R2,R4,L1       IF ID EX ME WB
     DADD  R2,R4,R5          IF cancelled in pipeline
L1:  DADD  R4,R2,R4             IF ID EX ME WB
```

- Branch not taken

```
cc                          1  2  3  4  5  6  7  8
     DADD  R8,R2,R3    IF ID EX ME WB
     BEQ   R2,R4,L1       IF ID EX ME WB
     DADD  R2,R4,R5          IF ID EX ME WB
     DADD  R2,R4,R5             IF ID EX ME WB
     ...
```

Branch strategies – branch delay slot

We have seen that the instruction that comes after a branch is *special:*

- It must only be executed if the branch is not taken.
- The problem is that we might only figure out one cycle too late, letting this instruction start executing in the pipeline even if we are taking the branch.
- Multiple schemes exist to solve this issue. What if we could do simpler?

Solution: make use of a **better compiler** which reorders instructions, putting an instruction in this *slot* that should execute regardless of the branch outcome.

Branch strategies – branch delay slot

If we find an appropriate instruction, no penalty!

- Branch taken

```
cc                              1   2   3   4   5   6   7   8
      BEQ    R2,R4,L1           IF  ID  EX  ME  WB
      DADD   R8,R2,R3               IF  ID  EX  ME  WB
L1:   DADD   R4,R2,R4                   IF  ID  EX  ME  WB
```

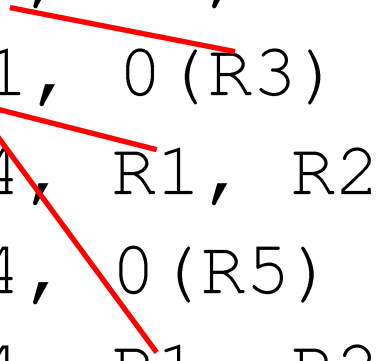
- Branch not taken

```
cc                              1   2   3   4   5   6   7   8
      BEQ    R2,R4,L1           IF  ID  EX  ME  WB
      DADD   R8,R2,R3               IF  ID  EX  ME  WB
      ...
```

Data Dependences

- True dependence (RAW)
- Anti-dependence (WAR)
- Output dependence (WAW)

```
DADD R3, R1, R2
LW   R1, 0(R3)
DADD R4, R1, R2
SW   R4, 0(R5)
MUL  R4, R1, R2
```



Data Dependences

- True dependence (RAW)
- Anti-dependence (WAR)
- Output dependence (WAW)

```
DADD R3, R1, R2
LW   R1, 0(R3)
DADD R4, R1, R2
SW   R4, 0(R5)
MUL  R4, R1, R2
```

Data Dependences

- True dependence (RAW)
- Anti-dependence (WAR)
- Output dependence (WAW)

```
DADD R3, R1, R2
LW   R1, 0(R3)
DADD R4, R1, R2
SW   R4, 0(R5)
MUL  R4, R1, R2
```

Hazard - Example

First *two* loop iterations:

CC	1	2	3	4	5	6	7	8	9	10	11	12	13	14
LD R1,0(R2)	IF	ID	EX	ME	WB									
DADDI R1,R1,#1		IF	ID	ID	EX	ME	WB							
SD R1,0(R2)			IF	IF	ID	EX	ME	WB						
DADDI R2,R2,#8					IF	ID	EX	ME	WB					
DSUB R4,R3,R2						IF	ID	EX	ME	WB				
BNEZ R4,loop							IF	ID	ID	EX	ME	WB		
DADD R9,R9,R9									IF	(flush)				
LD R1,0(R2)										IF	ID	EX	ME	WB
DADDI R1,R1,#1											IF	ID	ID	EX
SD R1,0(R2)												IF	IF	ID
...														

9 cycles per iteration
(ideal is 6)

Important: when doing this kind of exercises, make sure that each stage is only active once for each clock cycle [each column of the table].