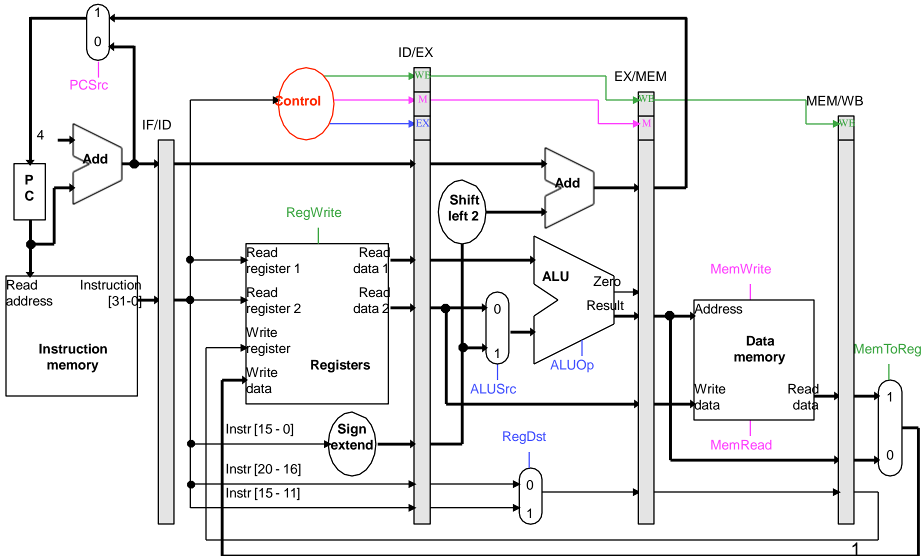
Advanced Computer Networks (ACA)

Prof. Santosh K C
Asst. Prof.
CSE Dept.
B.I.E.T.
Davangere-04

Pipeline Review



Our examples are too simple

Here is the example instruction sequence used to illustrate pipelining on the previous page

```
lw   $8, 4($29)
sub  $2, $4, $5
and  $9, $10, $11
or   $16, $17, $18
add  $13, $14, $0
```

The instructions in this example are independent

- Each instruction reads and writes completely different registers
- Our datapath handles this sequence easily

But most sequences of instructions are *not* independent!

An example with dependences

Read after Write dependences

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

Dependences are a property of how the computation is expressed

An example with dependences

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

There are several dependences in this code fragment

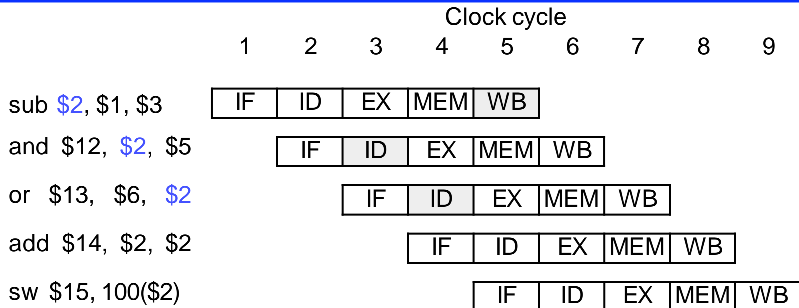
- ❖ The first instruction, SUB, stores a value into $2
- ❖ That register is used as a source in the rest of the instructions

This is no problem for 1-cycle and multicycle datapaths

- ❖ Each instruction executes completely before the next begins
- ❖ This ensures that instructions 2 through 5 above use the new value of $2 (the sub result), just as we expect.

How would this code sequence fare in our pipelined datapath?

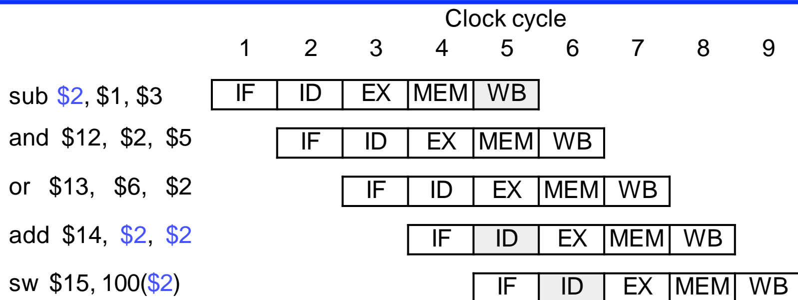
Data hazards in the pipeline diagram



The SUB does not write to register $2 until clock cycle 5 causeing 2 data hazards in our pipelined datapath

- The AND reads register $2 in cycle 3. Since SUB hasn't modified the register yet, this is the *old* value of $2
- Similarly, the OR instruction uses register $2 in cycle 4, again before it's actually updated by SUB

Things that are okay



The ADD is okay, because of the register file design

- ❖ Registers are written at the beginning of a clock cycle
- ❖ The new value will be available by the end of that cycle

The SW is no problem at all, since it reads $2 after the SUB finishes

One Solution To Data Hazards

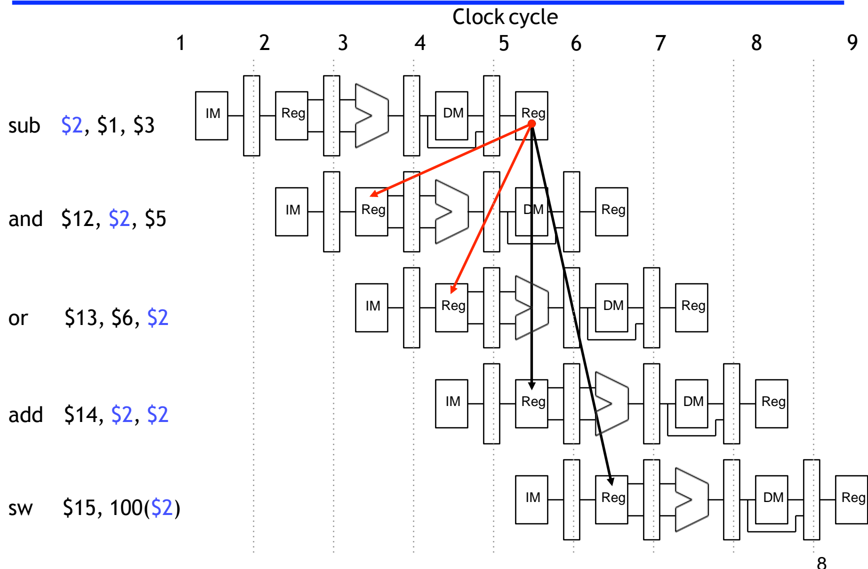
```
sub   $2, $1, $3
and   $12, $2, $5
or    $13, $6, $2
add   $14, $2, $2
sw    $15, 100($2)
```

```
sub   $2, $1, $3
sll   $0, $0, $0
sll   $0, $0, $0
and   $12, $2  $5
or    $13, $6, $2
add   $14, $2, $2
sw    $15, 100($2)
```

Since it takes two instruction cycles to get the value stored, one solution is for the assembler to insert no-ops or for compilers to reorder instructions to do useful work while the pipeline proceeds

A software solution to data hazards

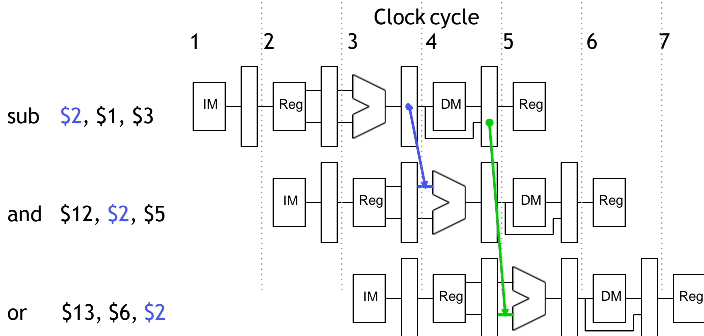
A fancier pipeline diagram



Forwarding

Since the pipeline registers already contain the ALU result, we could just forward the value to later instructions, to prevent data hazards

- In clock cycle 4, the AND instruction can get the value of $1 – $3 from the EX/MEM pipeline register used by SUB
- Then in cycle 5, the OR can get that same result from the MEM/WB pipeline register being used by SUB



Forwarding Implementation

Forwarding requires …

(a) Recognizing when a potential data hazard exists, and

(b) Revising the pipeline to introduce forwarding paths …

We'll do those revisions next time

What about stores?

Two "easy" cases: