

Dealing With Data Hazards

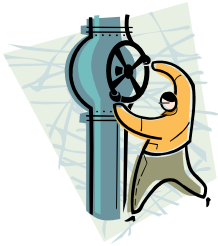
- In Software
 -
- In Hardware
 -
 -

Data Hazards are caused by *instruction dependences*. For example, the add is data-dependent on the subtract:

```

subi $5, $4, #45
add  $8, $5, $2
    
```

Pipeline Data Hazards



Warning, warning, warning!



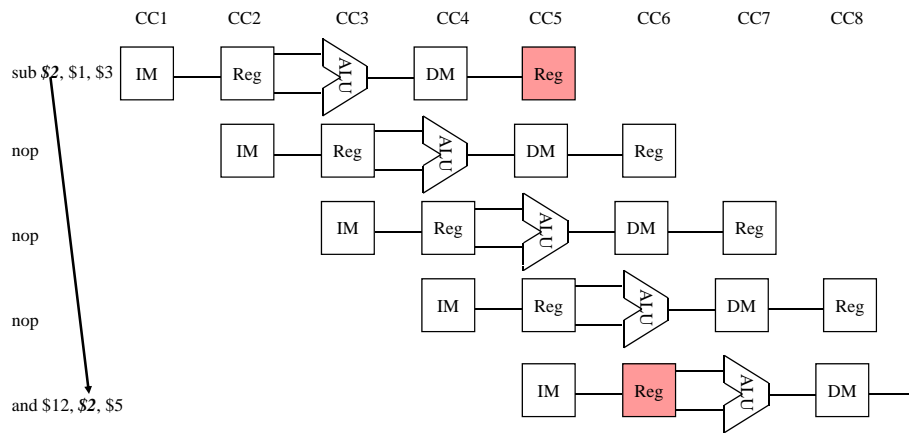
CSE 141

Dean Tullsen

CSE 141

Dean Tullsen

Dealing with Data Hazards in Software



CSE 141

Dean Tullsen

sub \$2, \$1,\$3
 and \$4, \$2,\$5
 or \$8, \$2,\$6
 add \$9, \$4,\$2
 slt \$1, \$6,\$7

How Many No-ops?

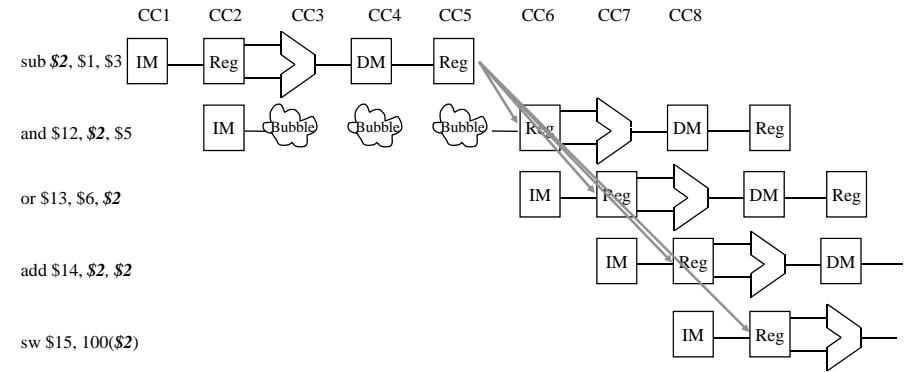
CSE 141

Dean Tullsen

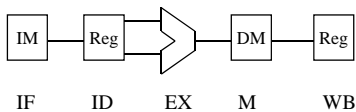
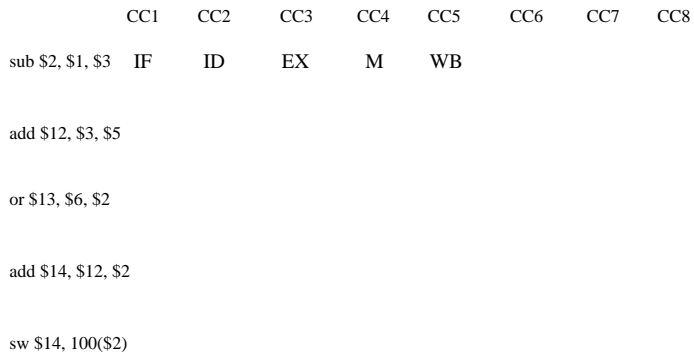
Are No-ops Really Necessary?

```
sub $2, $1,$3
and $4, $2,$5
or $8, $3,$6
add $9, $2,$8
slt $1, $6,$7
```

Dealing with Data Hazards in Hardware Part II-Pipeline Stalls



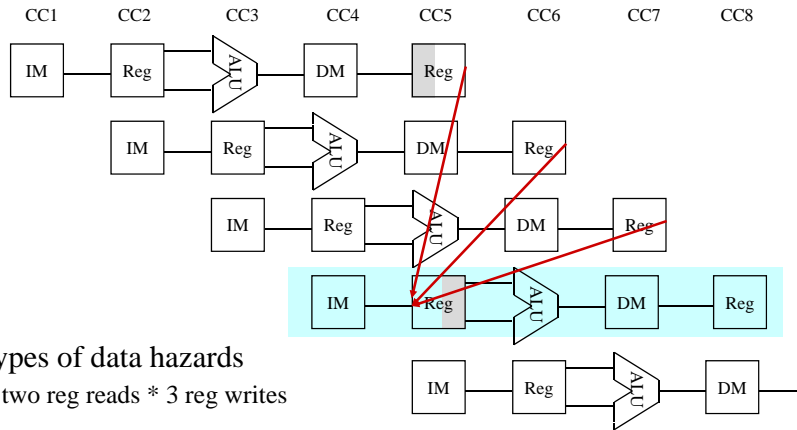
Pipeline Stalls



Pipeline Stalls

- To insure proper pipeline execution in light of register dependences, we must:
 - the hazard
 - the pipeline

Knowing When to Stall

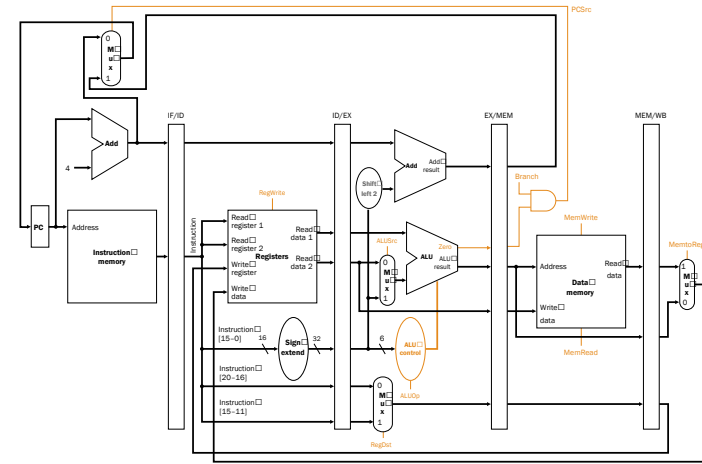


- 6 types of data hazards
 - two reg reads * 3 reg writes

CSE 141

Dean Tullsen

The Pipeline



- What comparisons tell us when to stall?

CSE 141

Dean Tullsen

Stalling the Pipeline

- Once we detect a hazard, then we have to be able to stall the pipeline (insert a bubble).
- Stalling the pipeline is accomplished by
 - (1) preventing the `IF` and `ID` stages from making progress
 - the `ID` stage because it cannot proceed until the dependent instruction completes
 - the `IF` stage because we do not want to lose any instructions.
 - (2) essentially, inserting “`nops`” in hardware

CSE 141

Dean Tullsen

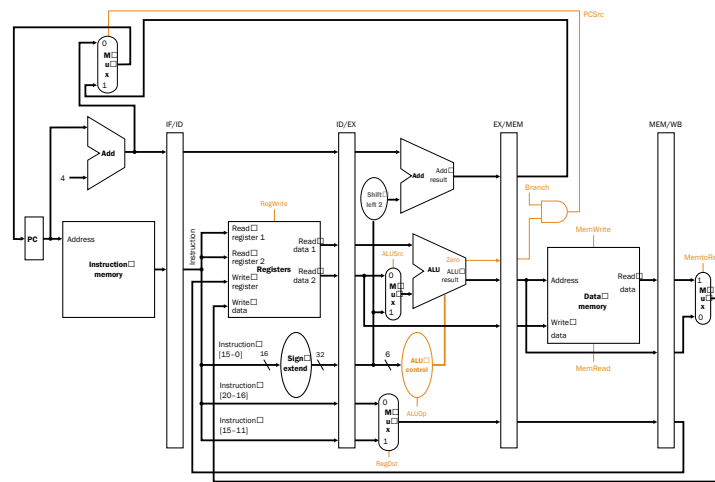
Stalling the Pipeline

- Preventing the `IF` and `ID` stages from proceeding
 - don't write the PC (`PCWrite = 0`)
 - don't rewrite `IF/ID` register (`IF/IDWrite = 0`)
- Inserting “`nops`”
 - set all control signals propagating to `EX/MEM/WB` to **zero**

CSE 141

Dean Tullsen

The Pipeline

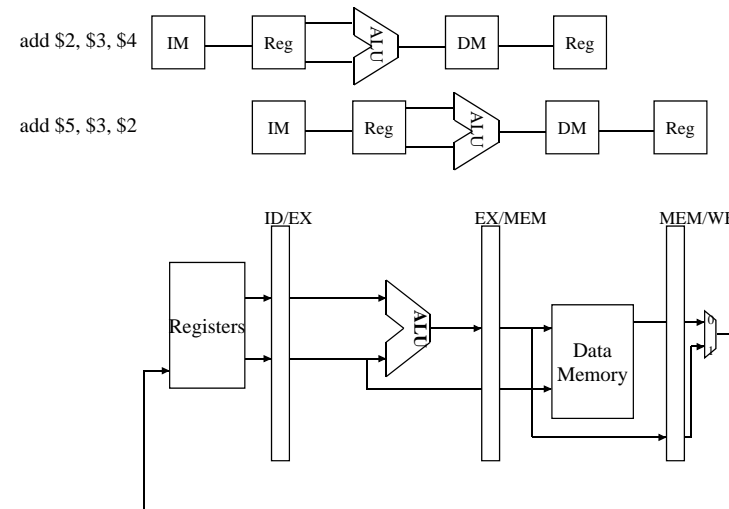


CSE 141

How else can we deal with data hazards?

Dean Tullsen

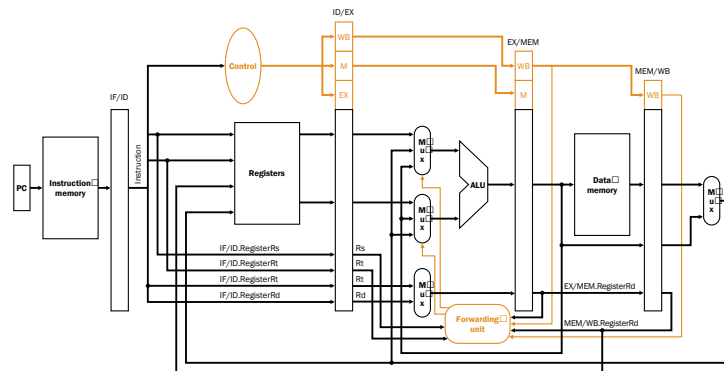
Reducing Data Hazards Through Forwarding



CSE 141

Dean Tullsen

Reducing Data Hazards Through MEM Forwarding



EX Hazard:

```

if (EX/MEM.RegWrite
and (EX/MEM.RegisterRd != 0)
and (EX/MEM.RegisterRd = ID/EX.RegisterRs)) ForwardA = 10
if (EX/MEM.RegWrite
and (EX/MEM.RegisterRd != 0)
and (EX/MEM.RegisterRd = ID/EX.RegisterRt)) ForwardB = 10
    
```

CSE 141

(similar for the MEM stage)

Dean Tullsen

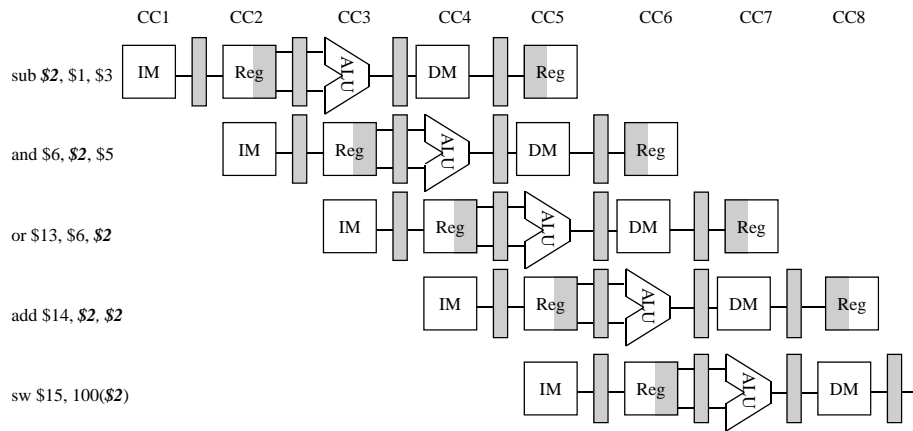
Data Forwarding

- The Previous Data Path handles two types of data hazards
 - hazard
 - hazard
- We assume the register file handles the third (WB hazard)
 - if the register file is asked to read and write the same register in the same cycle, we assume that the reg file allows the write data to be forwarded to the output
 - We're still going to call that forwarding.

CSE 141

Dean Tullsen

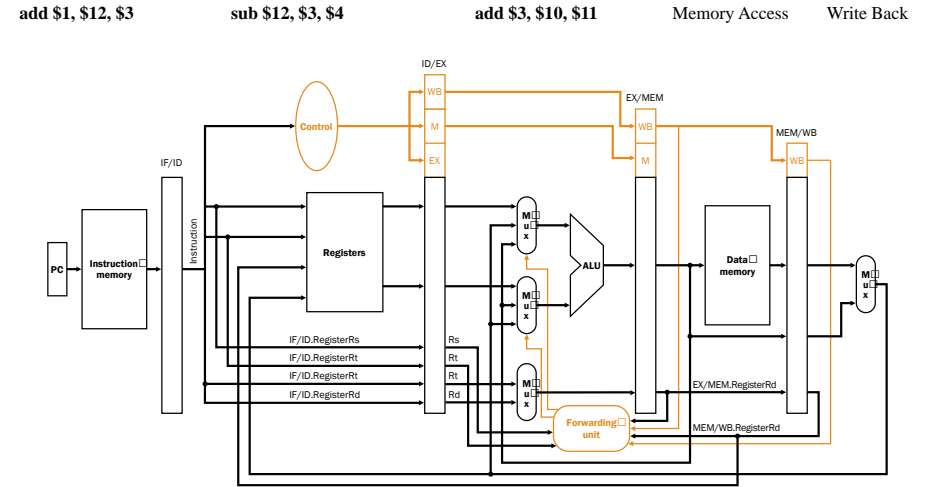
Eliminating Data Hazards via Forwarding



CSE 141

Dean Tullsen

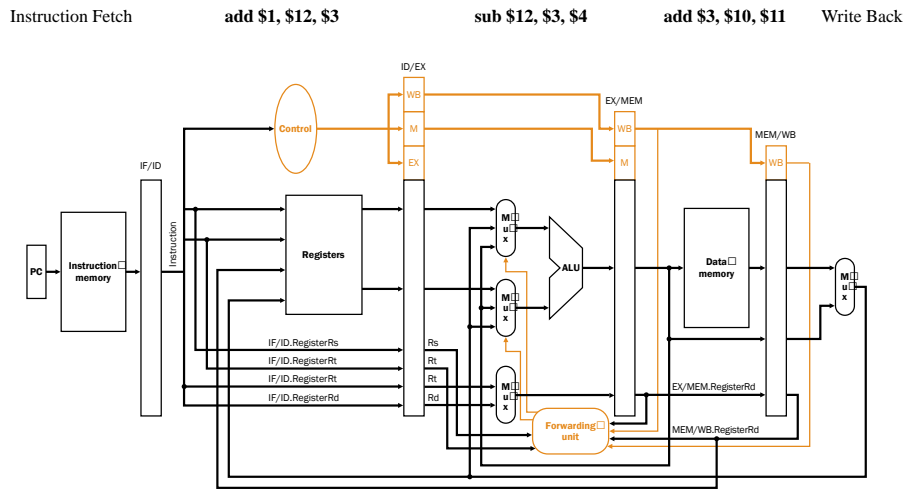
Forwarding in Action



CSE 141

Dean Tullsen

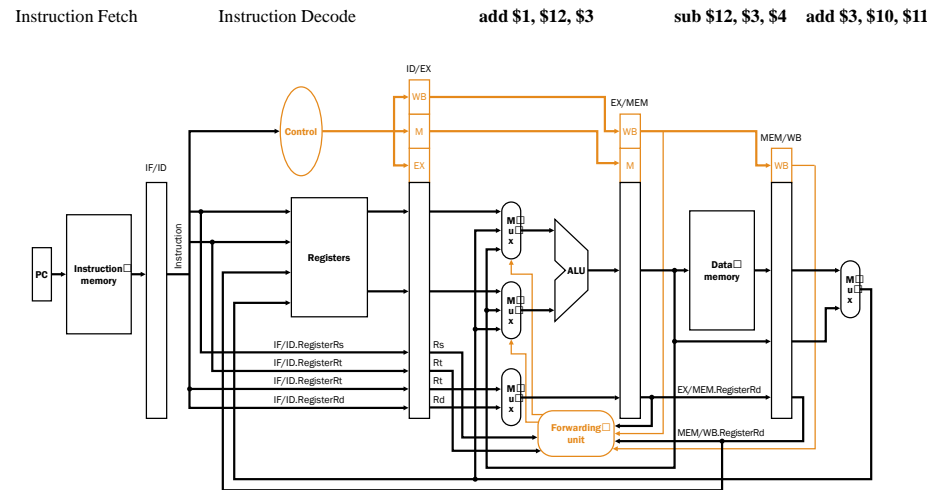
Forwarding in Action



CSE 141

Dean Tullsen

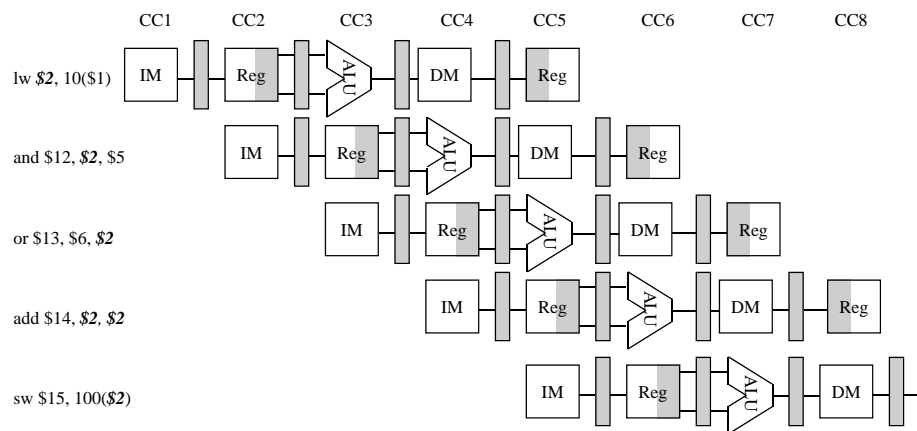
Forwarding in Action



CSE 141

Dean Tullsen

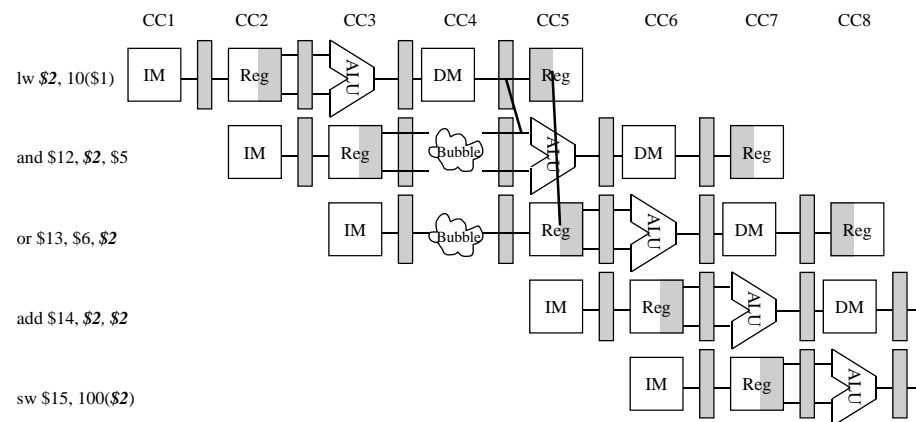
Eliminating Data Hazards via Forwarding??



CSE 141

Dean Tullsen

Eliminating Data Hazards via Forwarding and stalling



CSE 141

Dean Tullsen

Try this one...

Show stalls and forwarding for this code

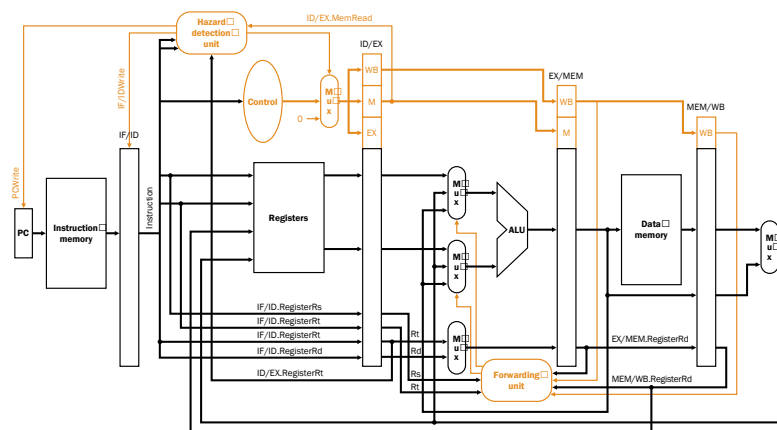
```

add $3, $2, $1
lw $4, 100($3)
and $6, $4, $3
sub $7, $6, $2
add $9, $3, $6
    
```

CSE 141

Dean Tullsen

Datapath with Hazard-Detection



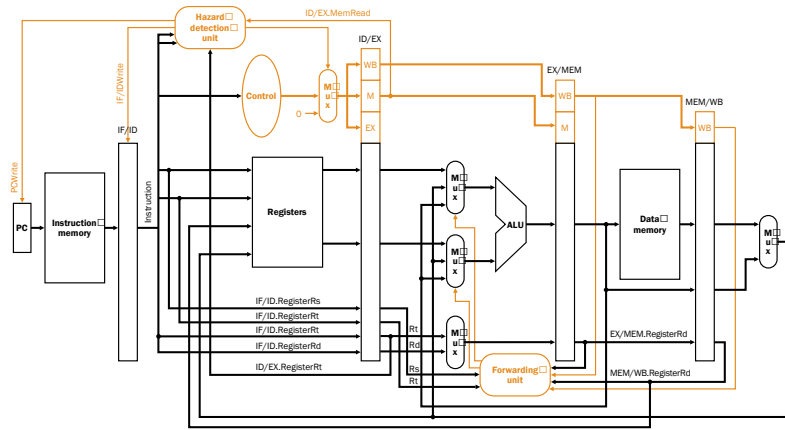
if (ID/EX.MemRead and
 ((ID/EX.RegisterRt = IF/ID.RegisterRs) or
 (ID/EX.RegisterRt = IF/ID.RegisterRt)))
 then stall the pipeline

CSE 141

Dean Tullsen

Hazard Detection

and \$4, \$2, \$5 lw \$2, 20(\$1)

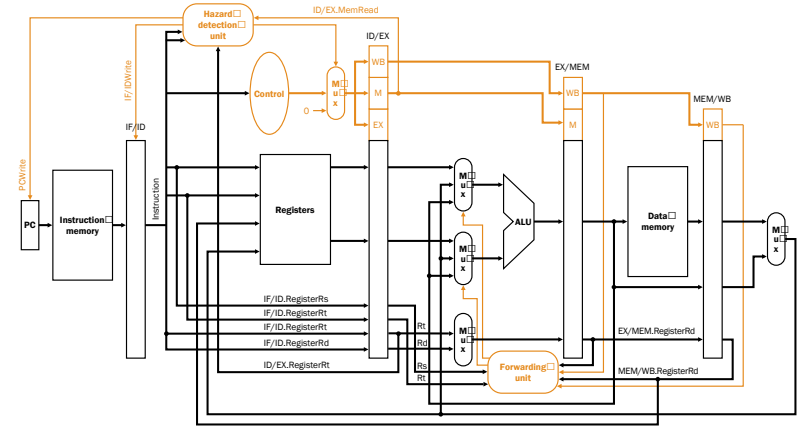


CSE 141

Dean Tullsen

Hazard Detection

and \$4, \$2, \$5 nop (bubble) lw \$2, 20(\$1)



CSE 141

Dean Tullsen

Data Hazard Key Points

- Pipelining provides high throughput, but does not handle data dependences easily.
- Data dependences cause *data hazards*.
- Data hazards can be solved by:
 - software (nops)
 - hardware stalling
 - hardware forwarding
- Our processor, and indeed all modern processors, use a combination of forwarding and stalling.
- $ET = IC * CPI * CT$

CSE 141

Dean Tullsen