ELE 655
Microprocessor System Design

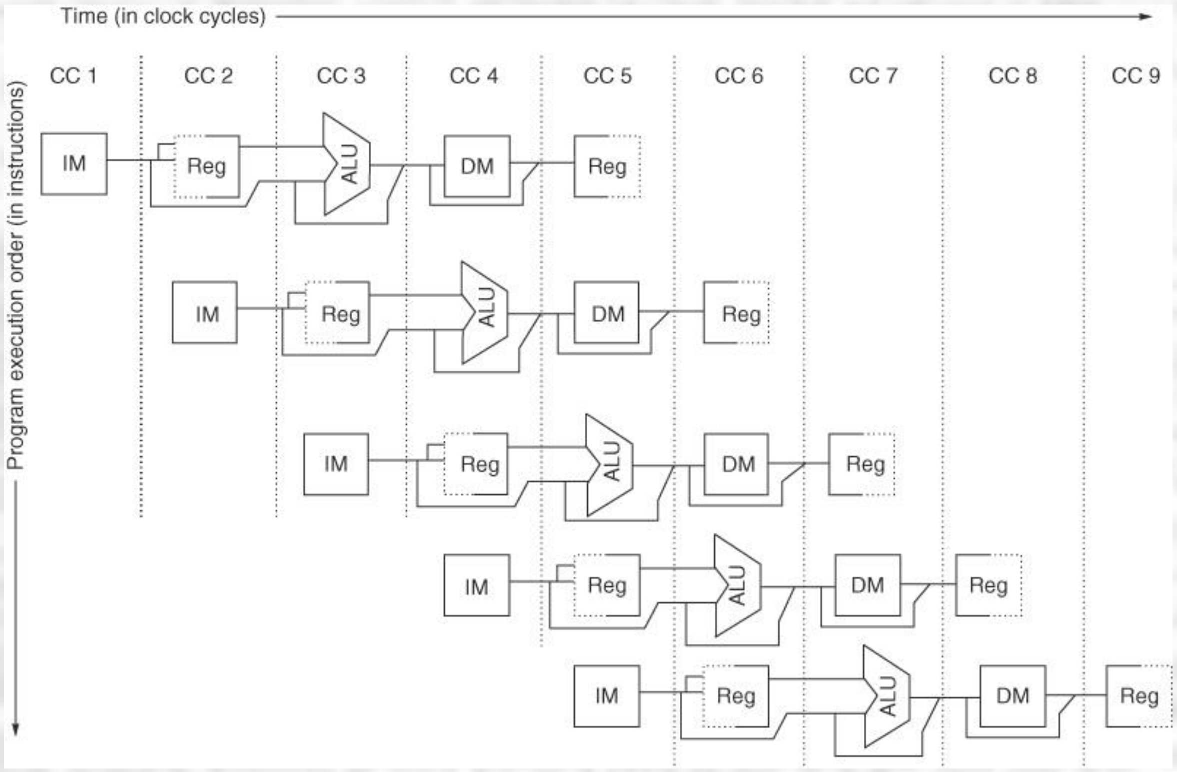
Section 2 – Instruction Level Parallelism

Class 1 – Pipeline Review

ILP

Pipeline Review

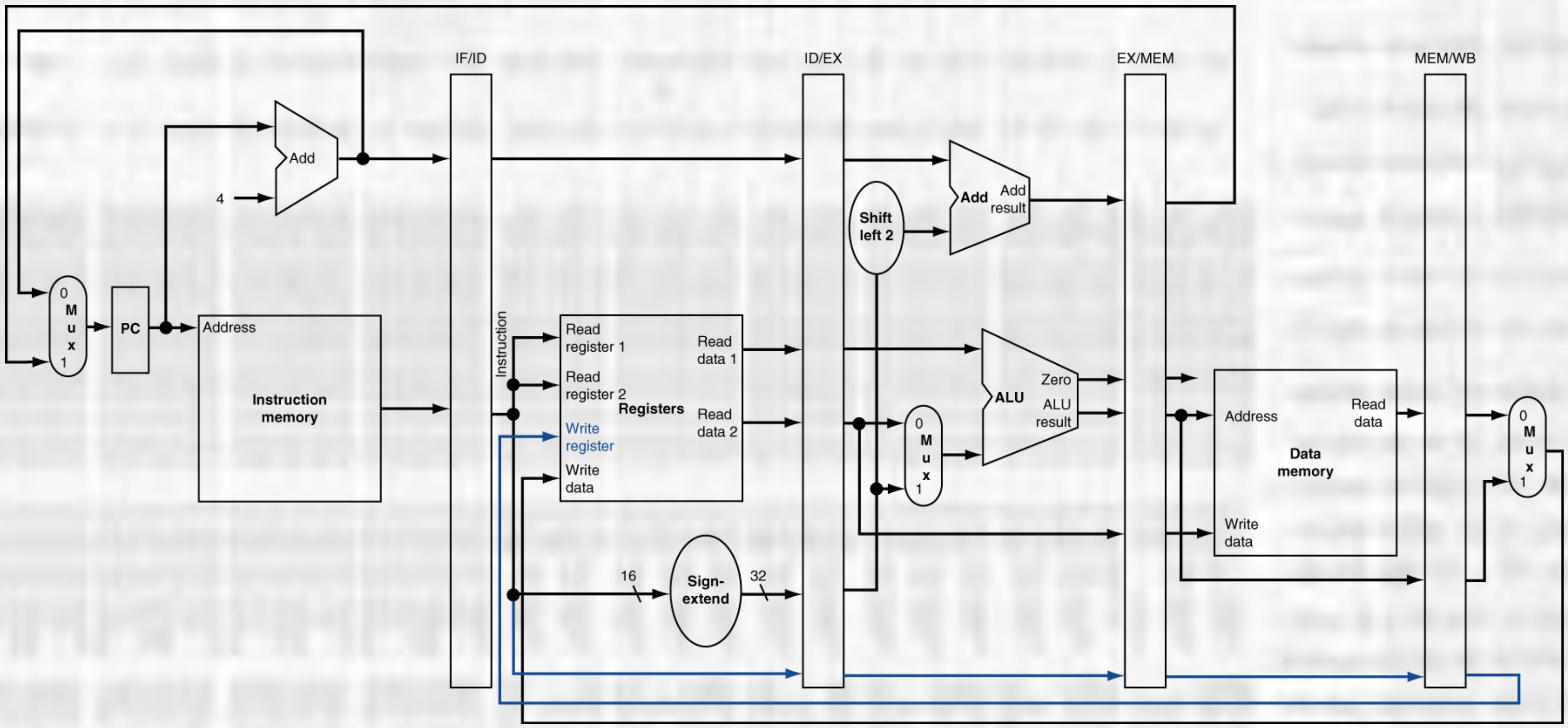
- Basic Pipeline



Notes: Reg shows up two places but actually is the same register file
Writes occur on the second half of the clock cycle, reads on the first half

ILP
Pipeline Review

- Basic Pipeline Implementation

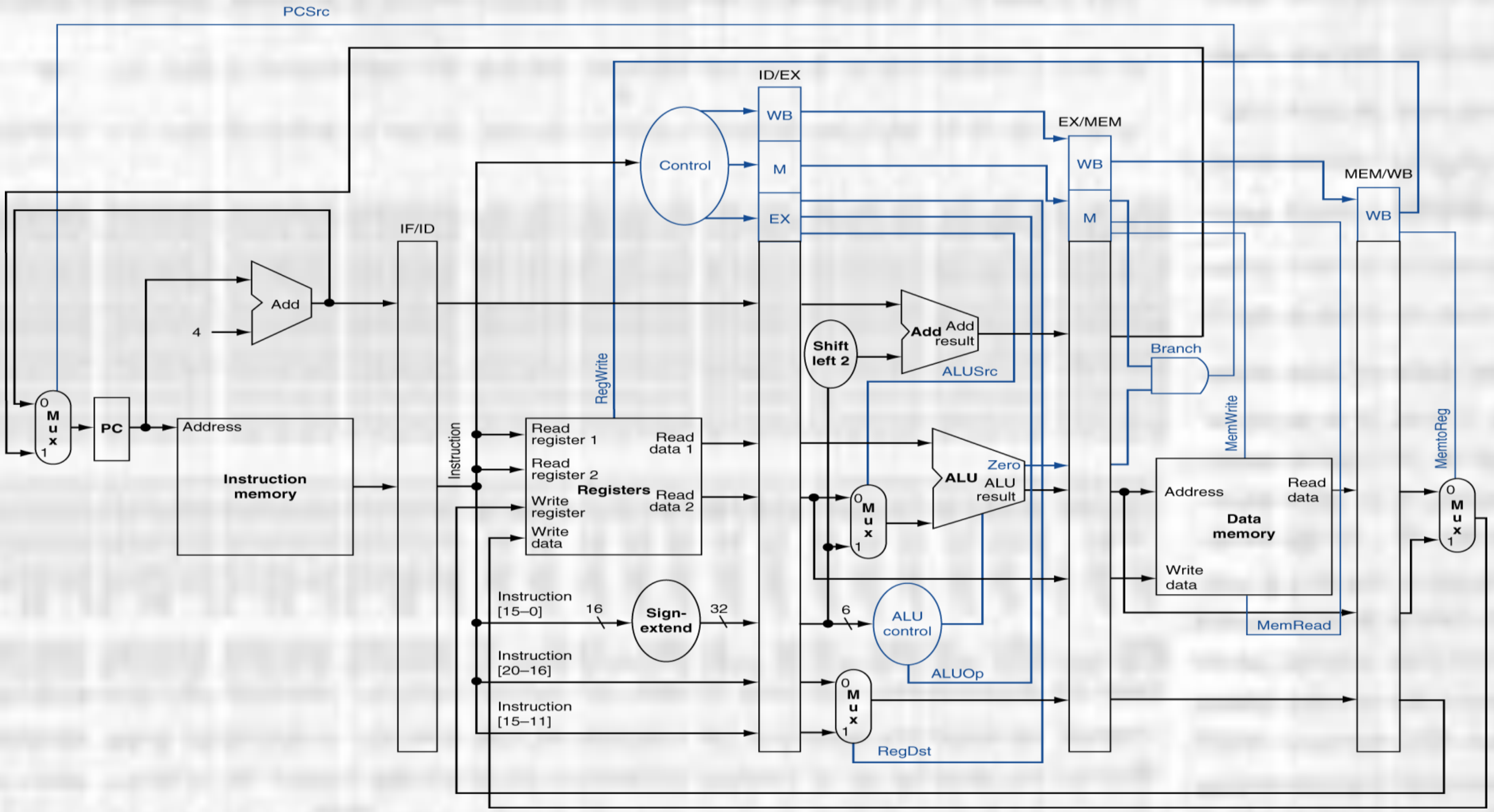


Notes: Writes occur on the second half of the clock cycle, reads on the first half

ILP

Pipeline Review

- Basic Pipeline with Control



ILP

Pipeline Review

- Pipeline Hazards
  - Hazards are conditions where the next instruction cannot perform its assigned pipeline action in the next clock cycle
  - 3 types
    - Structural
    - Data
    - Control

ILP

Pipeline Review

- Structural Hazards
  - These hazards result from a **resource** conflict
  - Classic case is Harvard vs. vonNeuman memory architectures
    - vonNeuman architectures share a single memory for program and data
    - A lw or sw command requires access to data memory to load or store the data value
    - It would not be possible to fetch the appropriate instruction during this clock cycle since the memory would be in use
    - The IF would be stalled and a "bubble" would be created in the pipeline

ILP
Pipeline Review

- Structural Hazards
  - vonNeuman memory architecture

Time	200	200	200	200	200	200	200	200	200	200	200	200	200	200	200
IF	LW	2	3	Stall	4	5	6	7	8	9	10	11	12	13	14
ID		LW	2	3	bubble	4	5	6	7	8	9	10	11	12	13
EX			LW	2	3	bubble	4	5	6	7	8	9	10	11	12
MEM				$\boxed{\text{LW}}$	2	3	bubble	4	5	6	7	8	9	10	11
WB					LW	2	3	bubble	4	5	6	7	8	9	10

data memory access prevents a concurrent instruction fetch

ILP

Pipeline Review

- Structural Hazards
  - Too few registers
  - Complex instructions → multi resource requirements (dual write)
  - Non-pipelined functions
  - Mismatched pipelines

ILP

Pipeline Review

- Data Hazards
  - These hazards result from a **dependence** of one instruction on another instruction still in the pipeline
  - Consider the following code snippit

```
add   $s0, $t0, $t1
sub   $t2, $s0, $t3
```

- The value of $s0 is needed to perform the subtraction

ILP
Pipeline Review

- Data Hazards

```
add   $s0, $t0, $t1
sub   $t2, $s0, $t3
```

Time	200	200	200	200	200	200	200	200	200	200	200	200	200	200	200
IF	add	sub	3	3	4	5	6	7	8	9	10	11	12	13	14
ID		add	stall	stall	3	4	5	6	7	8	9	10	11	12	13
EX			add	bubble	bubble	3	4	5	6	7	8	9	10	11	12
MEM				add	bubble	bubble	3	4	5	6	7	8	9	10	11
WB					add	bubble	bubble	3	4	5	6	7	8	9	10

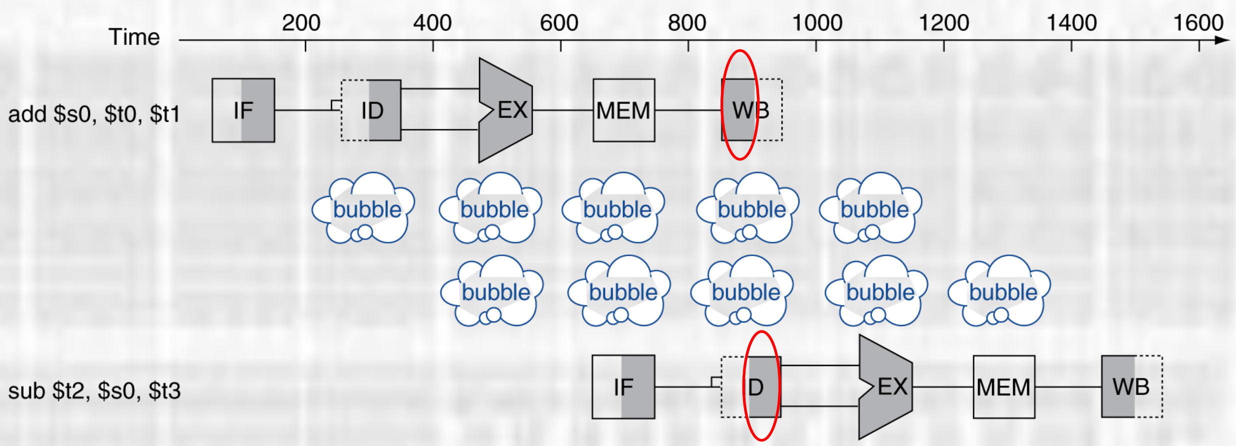
- 2 clock cycle bubbles are created
- It would be 3 bubbles – except we can take advantage of our convention
  - writes occur in the first half of the clock cycle
  - reads occur in the second half of the clock cycle
  - the WB occurs during the same clock cycle as the register read

ILP

Pipeline Review

- Data Hazards

```
add   $s0, $t0, $t1
sub   $t2, $s0, $t3
```



- 2 clock cycle bubbles are created
- It would be 3 bubbles – except we can take advantage of our convention
  - writes occur in the first half of the clock cycle
  - reads occur in the second half of the clock cycle
  - the WB occurs during the same clock cycle as the register read

ILP

Pipeline Review

- Data Hazards

  - In many cases the compiler can avoid a data hazard

```
add   $s0, $t0, $t1
sub   $t2, $s0, $t3
or    $s2, $t0, $t1
and   $s3, $t0, $t3
add   $s4, $t1, $t3

add   $s0, $t0, $t1
or    $s2, $t0, $t1
and   $s3, $t0, $t3
add   $s4, $t1, $t3
sub   $t2, $s0, $t3
```

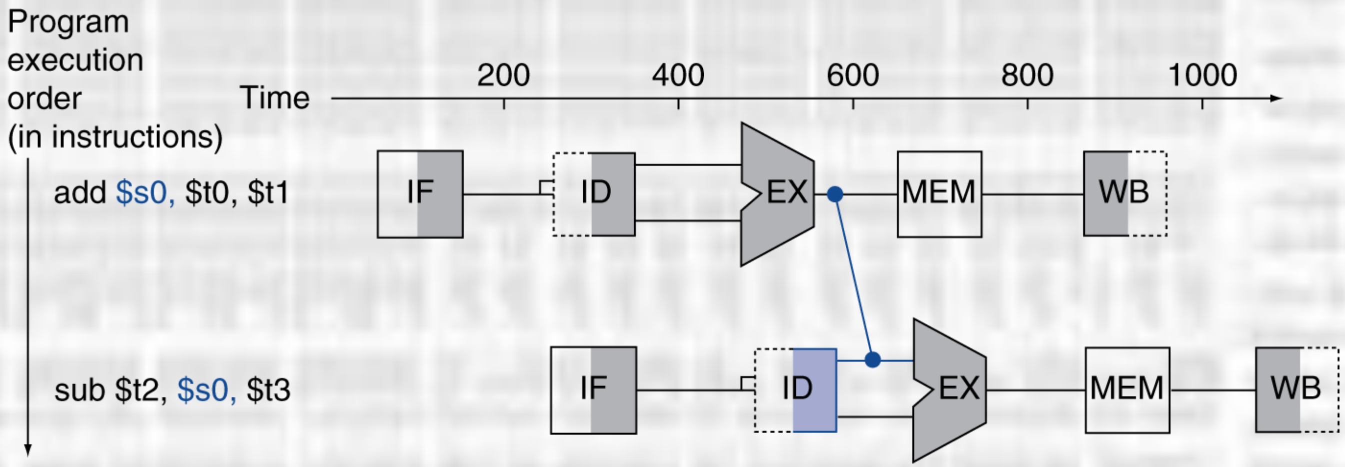
re-order the instruction to remove the hazard condition

ILP

Pipeline Review

- Data Hazards

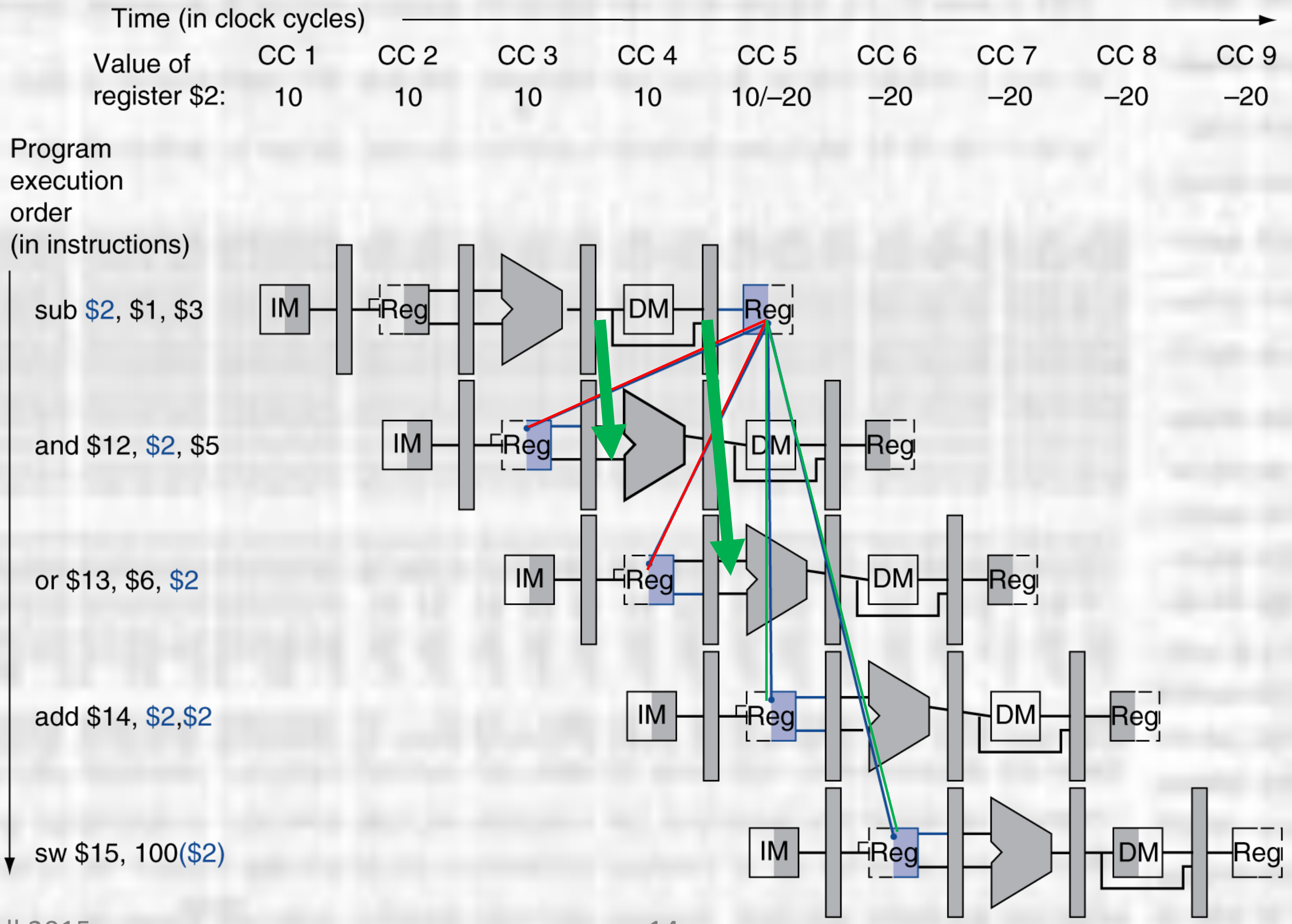
  - Hardware can also be used to avoid data hazards
    - called forwarding or bypassing
    - provide the needed data as soon as it is valid
    - requires extra circuitry



ILP

Pipeline Review

- Data Hazards



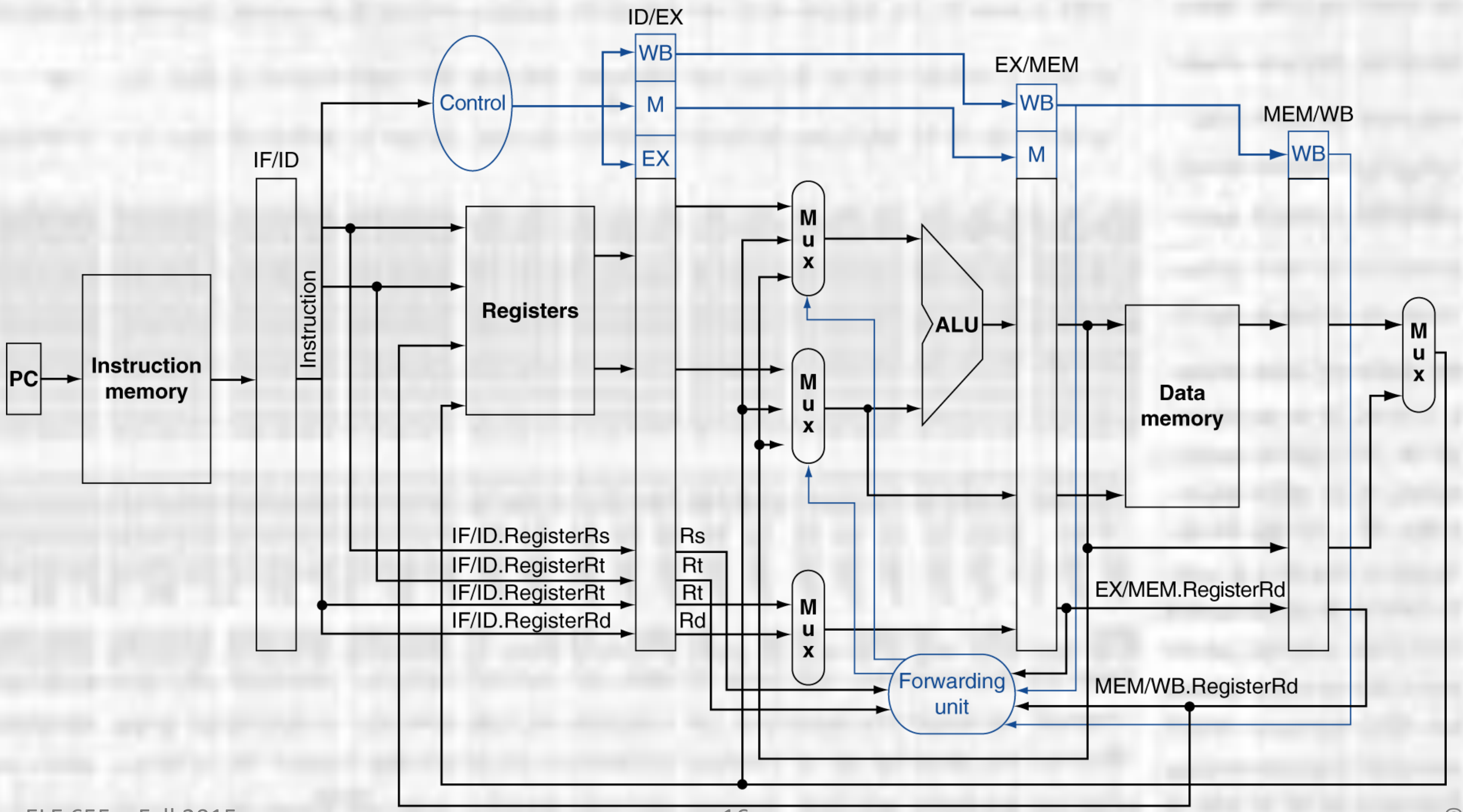
ILP

Pipeline Review

- Detecting the need for Forwarding
  - Conditions:
    - 1a) EX/MEM.RegisterRd = ID/EX.RegisterRs
      - EX/MEM currently holds a value needed by an instruction about to enter EX
    - 1b)) EX/MEM.RegisterRd = ID/EX.RegisterRt
      - EX/MEM currently holds a value needed by an instruction about to enter EX
    - 2a) MEM/WB.RegisterRd = ID/EX.RegisterRs
      - MEM/WB currently holds a value needed by an instruction about to enter EX
    - 2b) MEM/WB.RegisterRd = ID/EX.RegisterRt
      - MEM/WB currently holds a value needed by an instruction about to enter EX

ILP
Pipeline Review

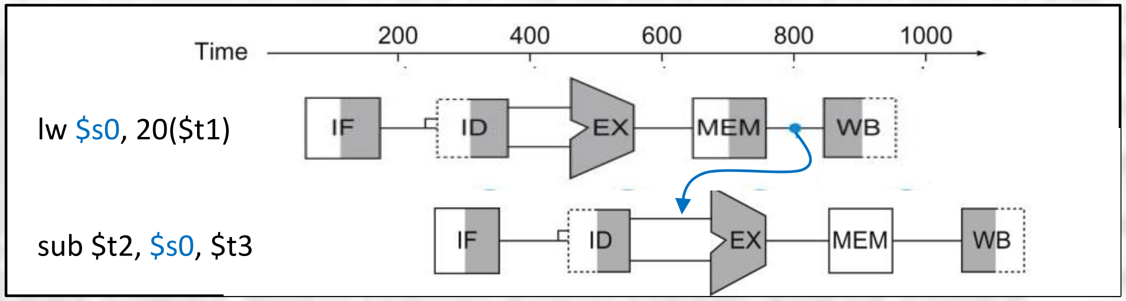
- Pipeline with Forwarding

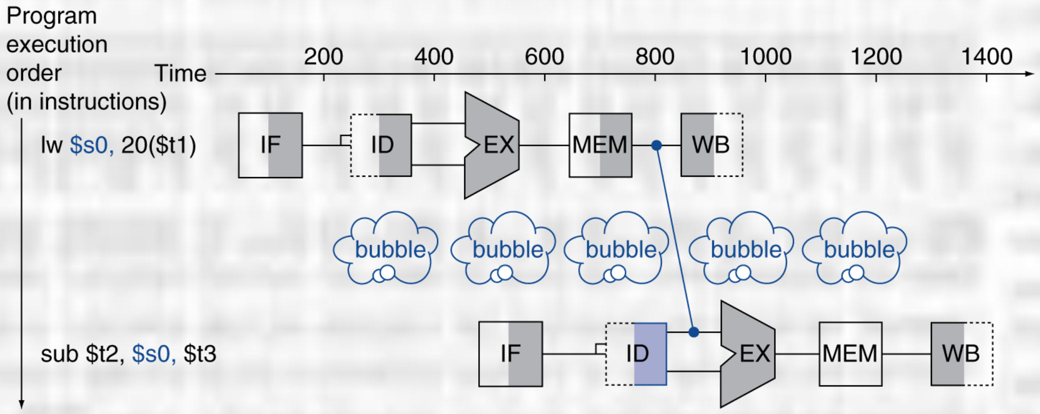


ILP
Pipeline Review

- Data Hazards

  - Hardware cannot avoid all data hazards
    - cannot go backwards in time !

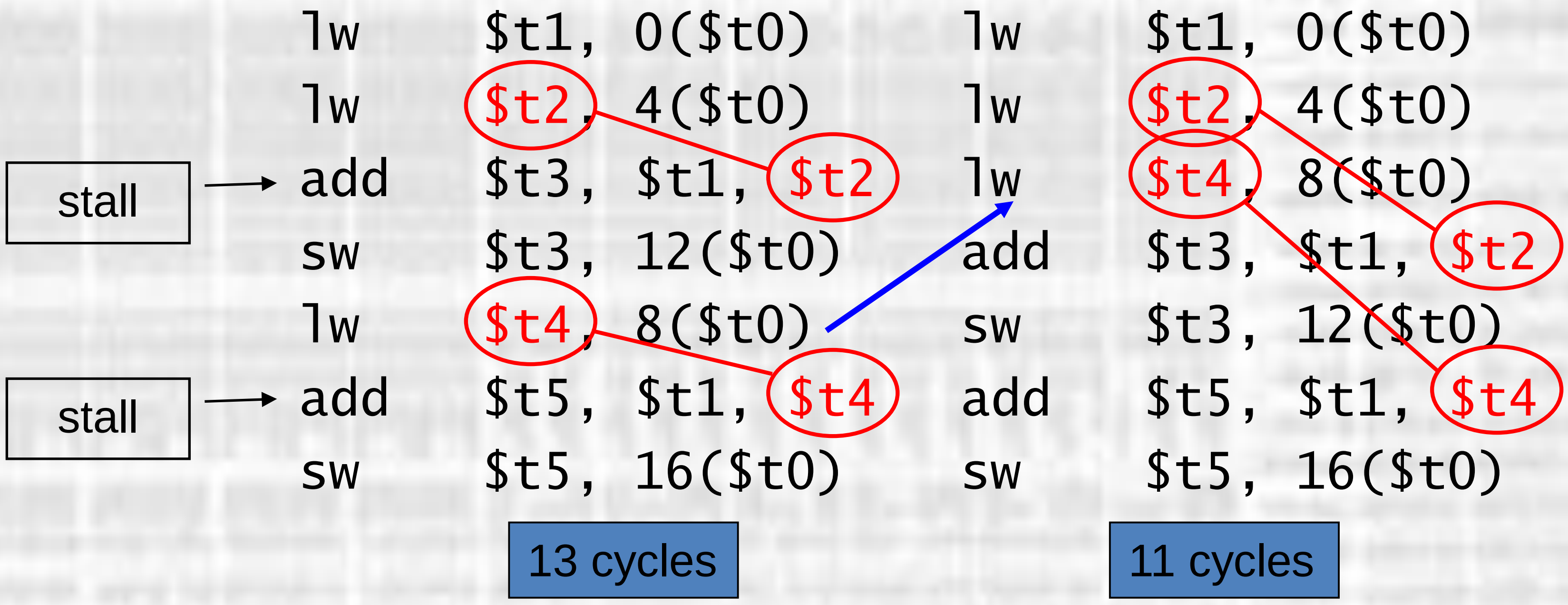




ILP

Pipeline Review

- Data Hazards
  - Forwarding plus compiler optimizations can avoid additional data hazards



```
lw    $t1, 0($t0)
lw    $t2, 4($t0)
add   $t3, $t1, $t2
sw    $t3, 12($t0)
lw    $t4, 8($t0)
add   $t5, $t1, $t4
sw    $t5, 16($t0)
```

13 cycles

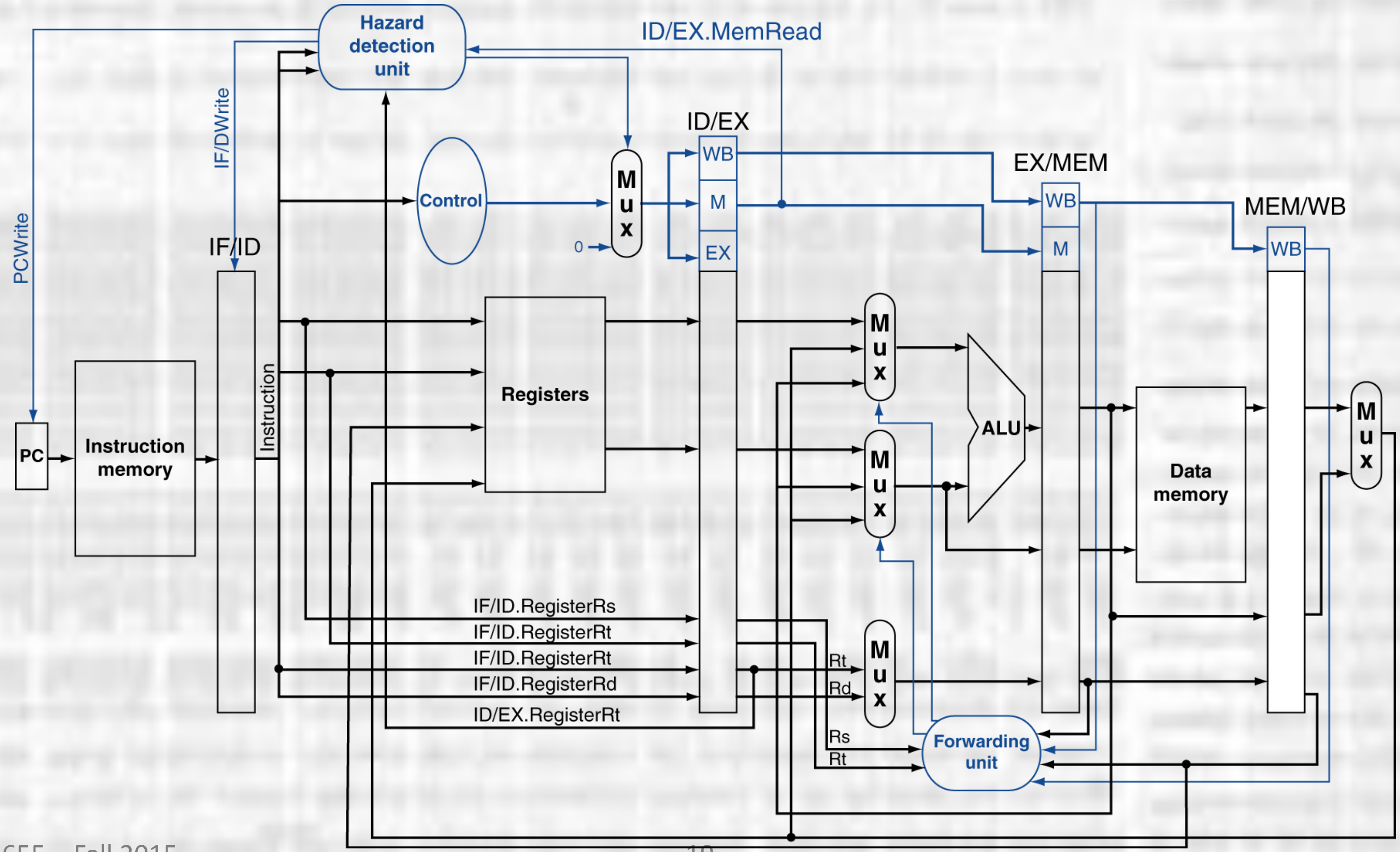
```
lw    $t1, 0($t0)
lw    $t2, 4($t0)
lw    $t4, 8($t0)
add   $t3, $t1, $t2
sw    $t3, 12($t0)
add   $t5, $t1, $t4
sw    $t5, 16($t0)
```

11 cycles

ILP

Pipeline Review

- Stalling the pipeline and inserting a bubble



ILP

Pipeline Review

- Branch Hazard

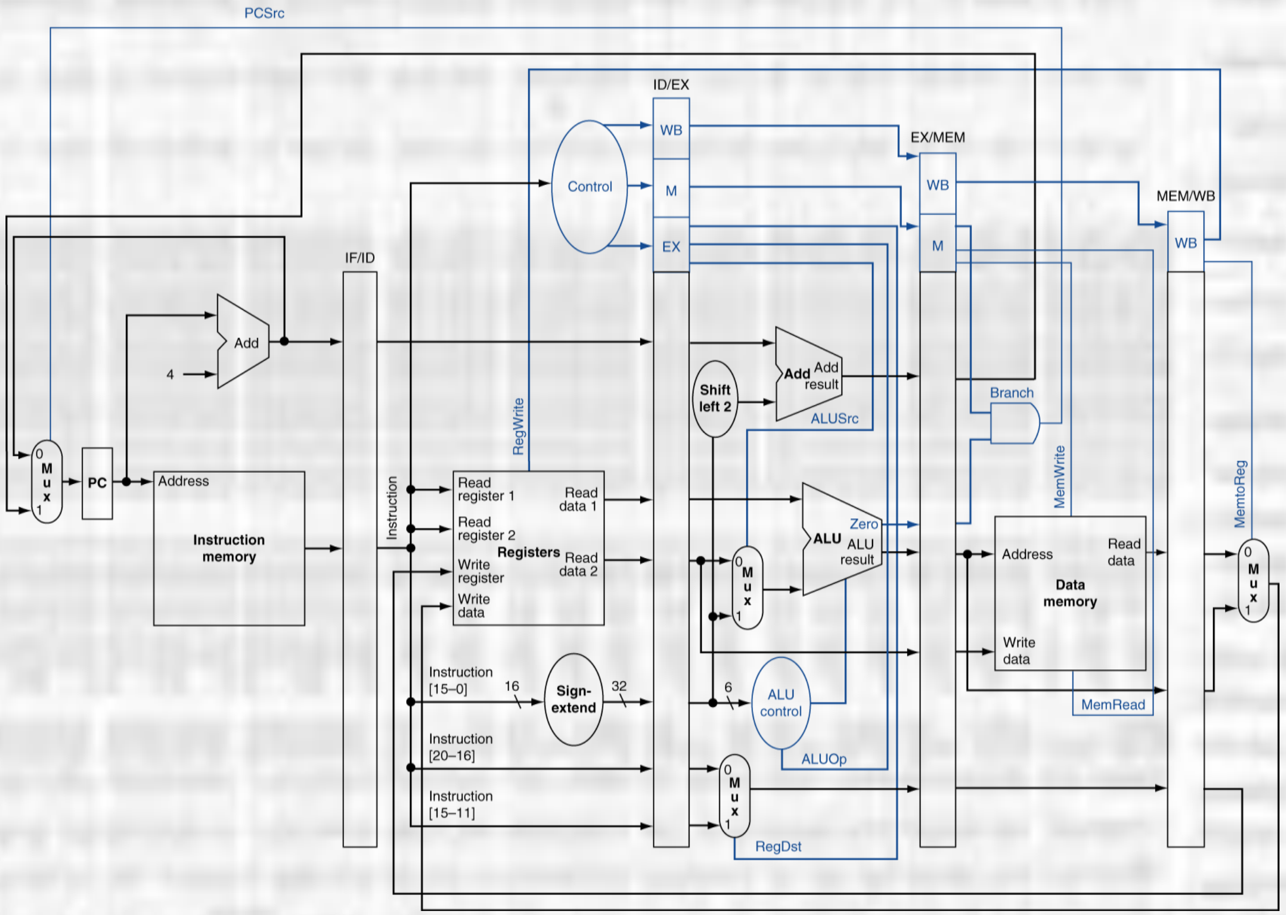
  - Consider the following code snippit

```
        beq     $1, $3, skip
        and     $13, $6, $2
        add     $14, $2, $2
skip    lw      $4, 50($7)
```

  - The branch decision is known after the calculation of the branch address and the comparison (subtract and check for zero), and is available in the MEM stage
  - If the branch is ignored – we will have the and, add and lw instructions in the pipeline – all is well
  - If the branch is taken we will have the and, add and lw instructions in the pipeline – but we do not want them to execute

ILP
Pipeline Review

- Addition of Branch Logic



ILP

Pipeline Review

- Branch Hazard
  - In our current implementation
    - We assume branches are not taken
    - We would need to flush the pipeline for taken branches
      - Branch decision is available in the MEM stage
      - Assuming the branch target is not already in the pipeline
      - $\rightarrow$ inserting 3 bubbles into the pipeline
  - ? – How far can we move the decision forward to reduce the impact of taken branches

ILP

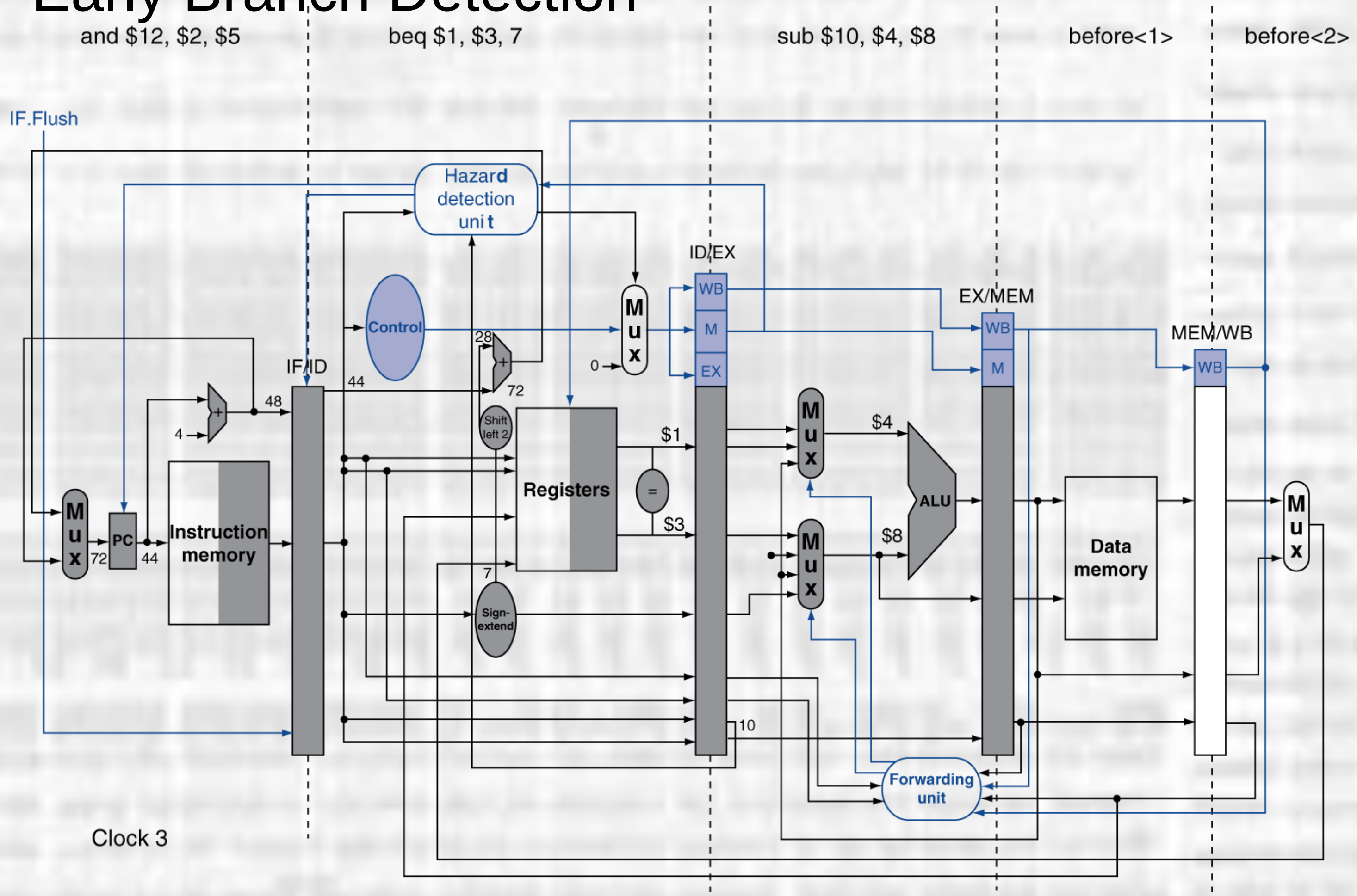
Pipeline Review

- Branch Hazard
  - Most branches are simple comparisons
    - equal → all bits the same
    - negative/positive → look at msb
    - We can move most of the branch prediction logic forward to the ID stage
      - We have register values (some may be forwarded!)
      - Need to modify the forwarding logic to account for branches
    - We can move the branch address calculation forward to the ID stage
    - Still have a single cycle stall – IF of the next instruction is occurring in parallel with ID detection of the taken branch

ILP

Pipeline Review

- Early Branch Detection



ILP

Pipeline Review

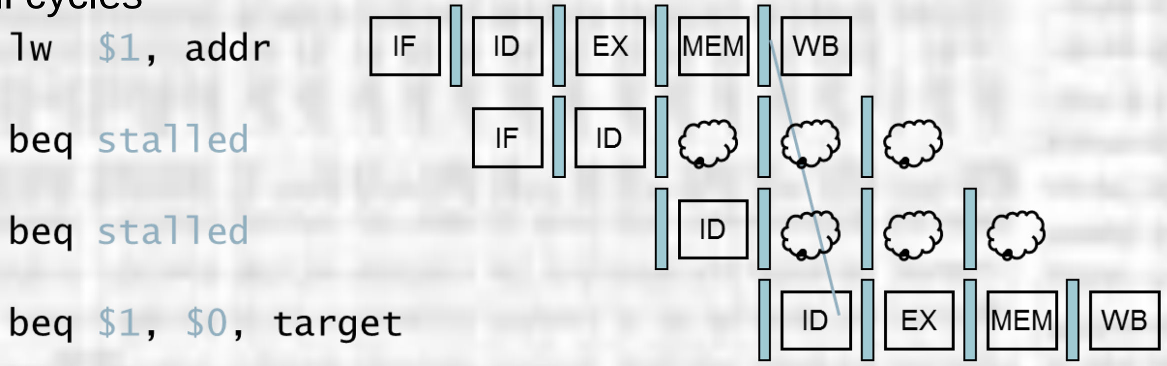
- Branch Hazard
  - This approach reduces the impact of branch hazards
  - There may still be data hazards that cannot be avoided
    - Branch dependent on previous lw instruction

      lw $1, 50($7)

      beq $1, $4, skip

    the lw result will not be available until the MEM cycle is complete

    → 2 stall cycles



ILP

Pipeline Review

- Branch Prediction
  - For deeper pipelines – the cost of missing a branch decision can be significant – many clock cycles lost
  - Leads to branch prediction
    - Static – Compile Time
    - Dynamic – Execution time

ILP
Pipeline Review

- Static Branch Prediction

  - Freeze the pipeline until decision known
    - Do not allow additional instructions until branch decision known
    - Insert No-op instruction(s)

  - Assume NOT taken
    - Allow sequential instruction into pipeline
    - HW must prevent commits until decision known
    - Must have HW to nop currently executing instructions
    - This is our simple pipeline solution

  - Assume Taken
    - As soon as target address available – start reading instructions
    - Used when complicated conditional instructions are part of the IS

Compiler can help
by ordering instructions
to match the HW

ILP

Pipeline Review

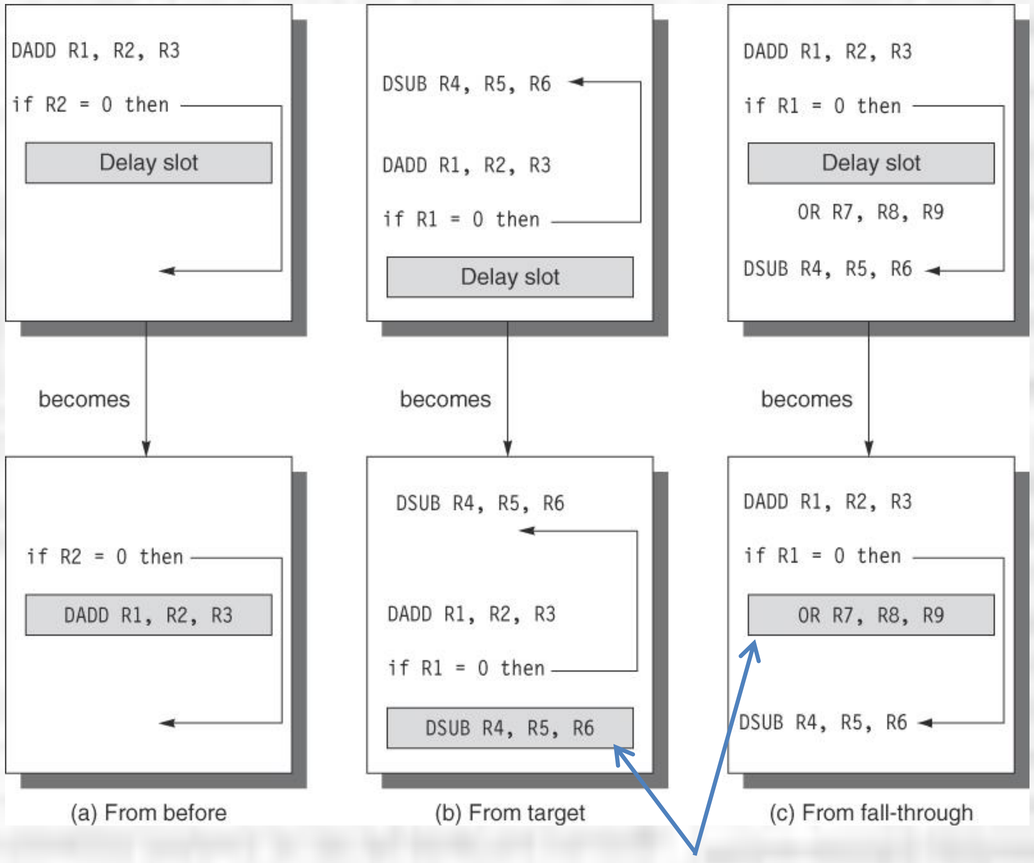
- Static Branch Prediction
  - Delayed Branch

Branch instruction
Sequential successor (always executed instruction) - delay slot
Branch target if taken

ILP
Pipeline Review

- Static Branch Prediction
  - Delayed Branch

Branch instruction
Sequential successor
Branch target if taken

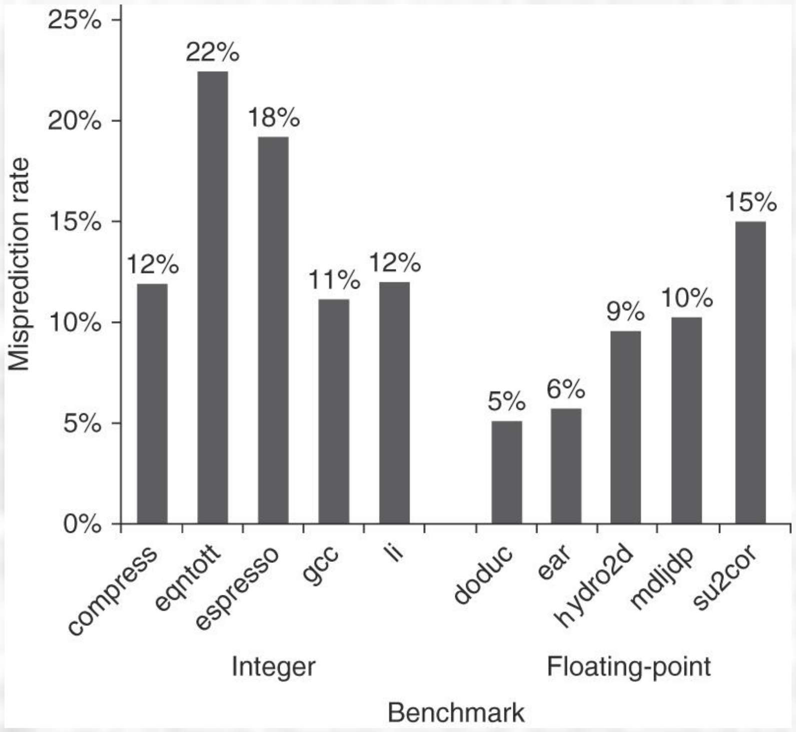


Must be OK whether branch taken or not

ILP

Pipeline Review

- Static Branch Prediction
  - Better than nothing – but not good enough for complex pipelines



ILP

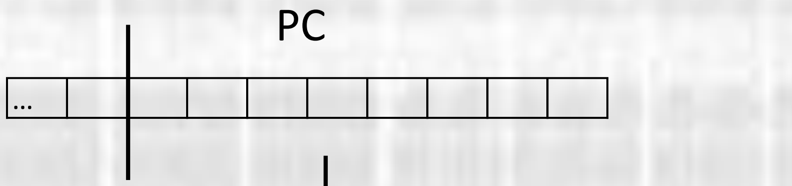
Pipeline Review

- Dynamic Branch Prediction – 1 bit
  - Use a small branch prediction buffer
    - n words deep
    - 1 bit of prediction value (1 bit word)
  - n is derived from the PC value
    - last 8 bits of PC → 256 words deep
  - The PC value references one of n predictions values
    - Assuming a branch instruction
    - Take the branch if the prediction value is set to 1
    - Don’t take the branch if the prediction value is set to 0
  - If the prediction was wrong – invert the prediction value

ILP

Pipeline Review

- Dynamic Branch Prediction – 1 bit



Branch Table

ADDR	B
0x00000000	0
0x00000001	0
0x00000010	1
…	
0x11111110	1
0x11111111	1

ILP
Pipeline Review

- Dynamic Branch Prediction – 1 bit

  - Issues

    - Multiple PC values point to the same branch table location
      - over write each other

        → wrong guesses

    - Each incorrect guess can lead to 2 wrong guesses
      - eg. Assume mostly loop back – bit set to 1

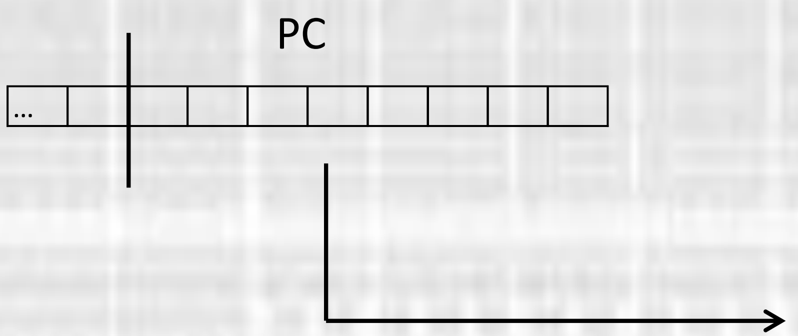
        when you do not loop back – you stall and set bit to 0

        next cycle you want to loop back but bit is 0 – stall and set bit to 1
      - 2 stalls

ILP

Control Hazards

- Dynamic Branch Prediction – 2 bit
  - Use 2 bits to make prediction decisions
    - Only change the prediction on 2 successive mispredictions
    - Resolves the 2 stall issue of 1 bit prediction



Branch Table

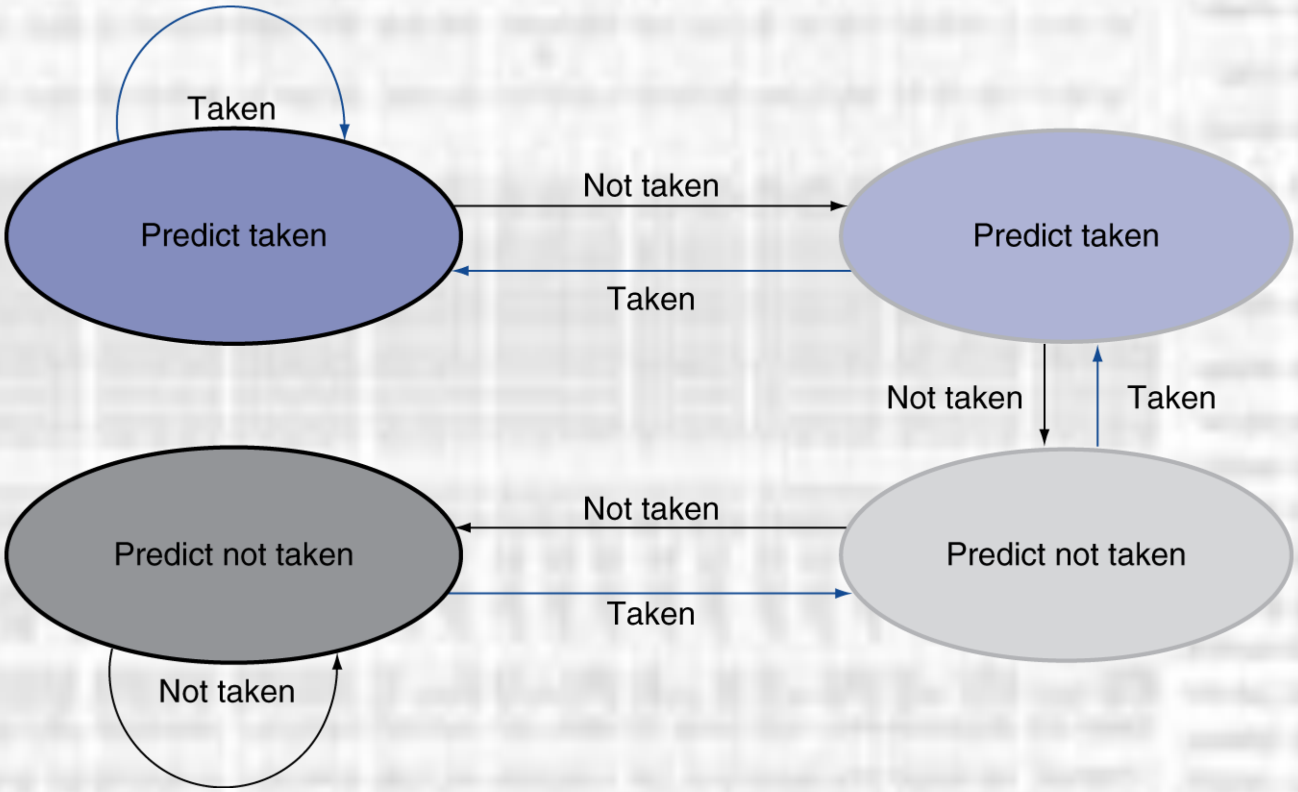
ADDR	B1	B0
0x00000000	0	0
0x00000001	0	0
0x00000010	1	0
…		
0x11111110	0	1
0x11111111	1	1

Option: Include bits in cache block instead of separate location – issue?

ILP

Pipeline Review

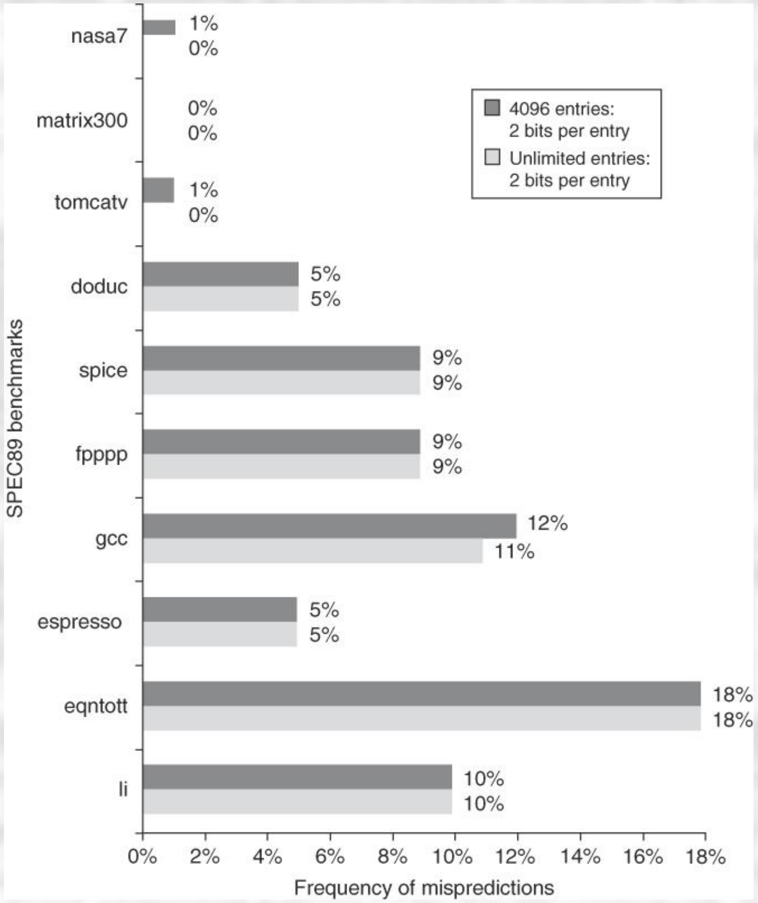
- Dynamic Branch Prediction – 2 bit

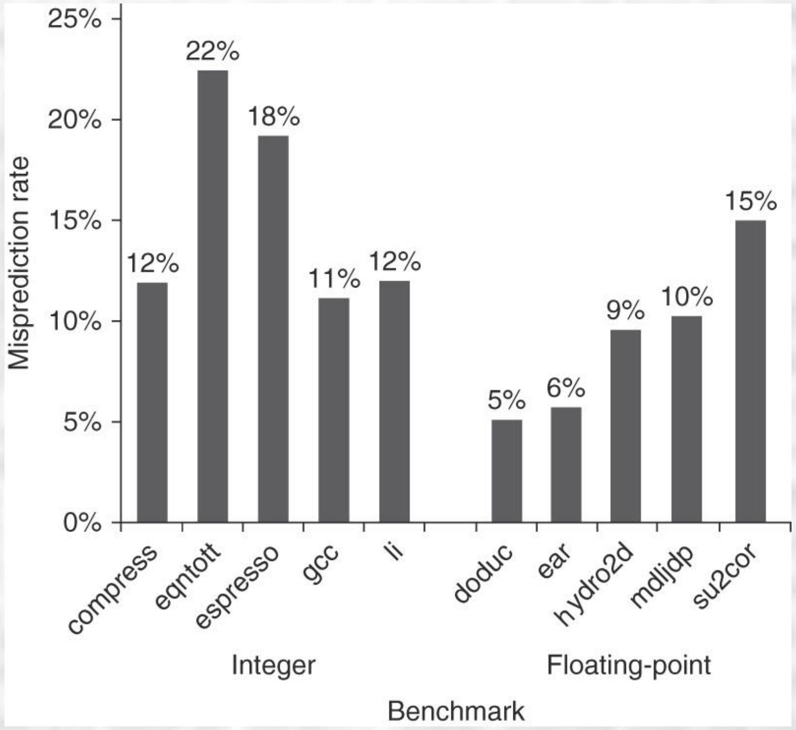


ILP

Pipeline Review

- Dynamic Branch Prediction – 2 bit


SPEC89 benchmarks
nasa7 1% 0%
matrix300 0% 0%
tomcatv 1% 0%
doduc 5% 5%
spice 9% 9%
fpppp 9% 9%
gcc 12% 11%
espresso 5% 5%
eqntott 18% 18%
li 10% 10%
4096 entries: 2 bits per entry
Unlimited entries: 2 bits per entry
Frequency of mispredictions


Misprediction rate
compress 12%
eqntott 22%
espresso 18%
gcc 11%
li 12%
doduc 5%
ear 6%
hydro2d 9%
mdljdp 10%
su2cor 15%
Integer
Floating-point
Benchmark