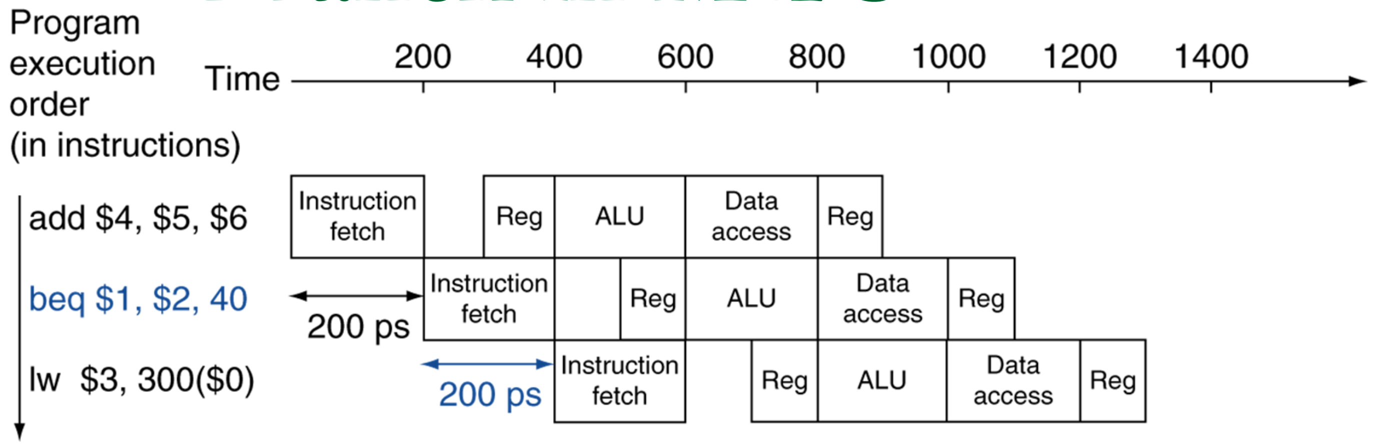
# COMPUTER ORGANIZATION (IS F242)

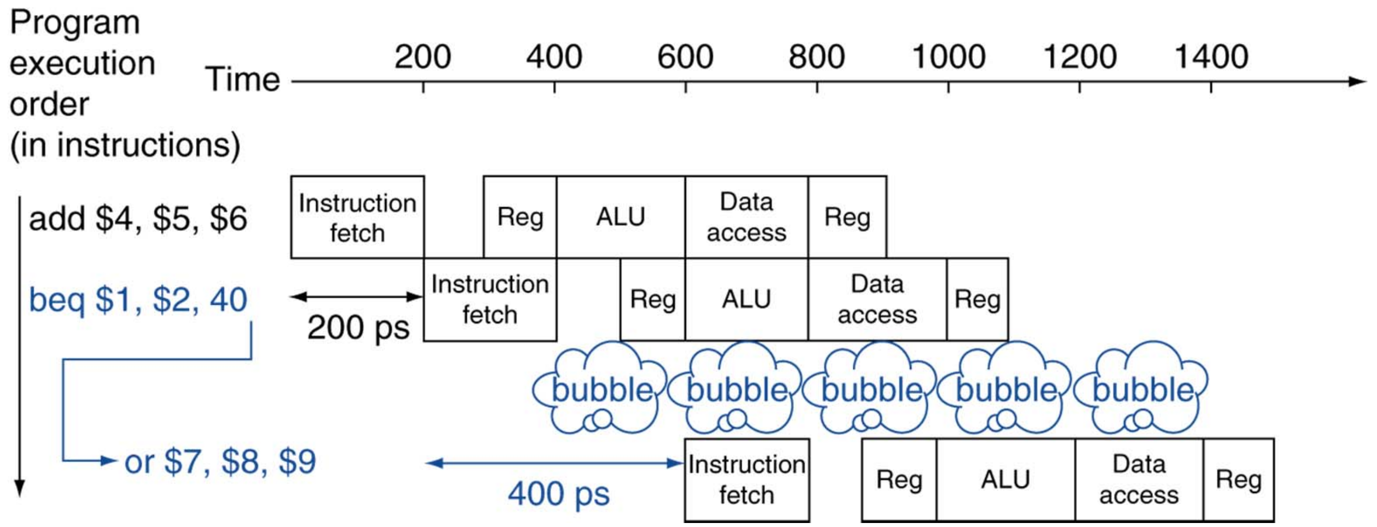
## LECT 46: PIPELINING

# Branch in MIPS

Prediction correct



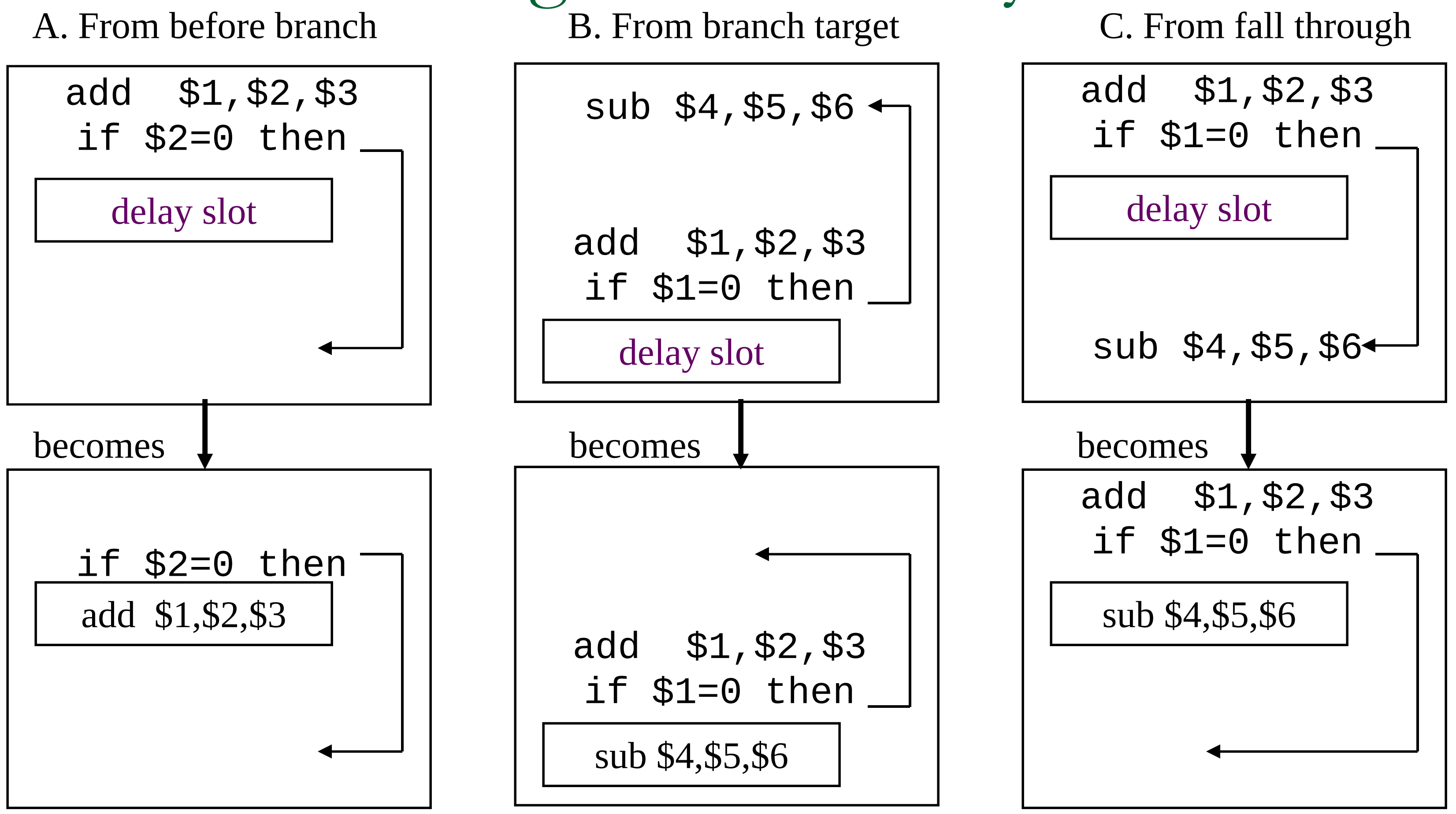
Prediction incorrect



# Delayed Branches

- If the branch hardware has been moved to the ID stage, then we can eliminate all branch stalls with delayed branches which are defined as always executing the next sequential instruction after the branch instruction – the branch takes effect *after* that next instruction
  - MIPS compiler moves an instruction to immediately after the branch that is not affected by the branch (a safe instruction) thereby hiding the branch delay

# Scheduling Branch Delay Slots

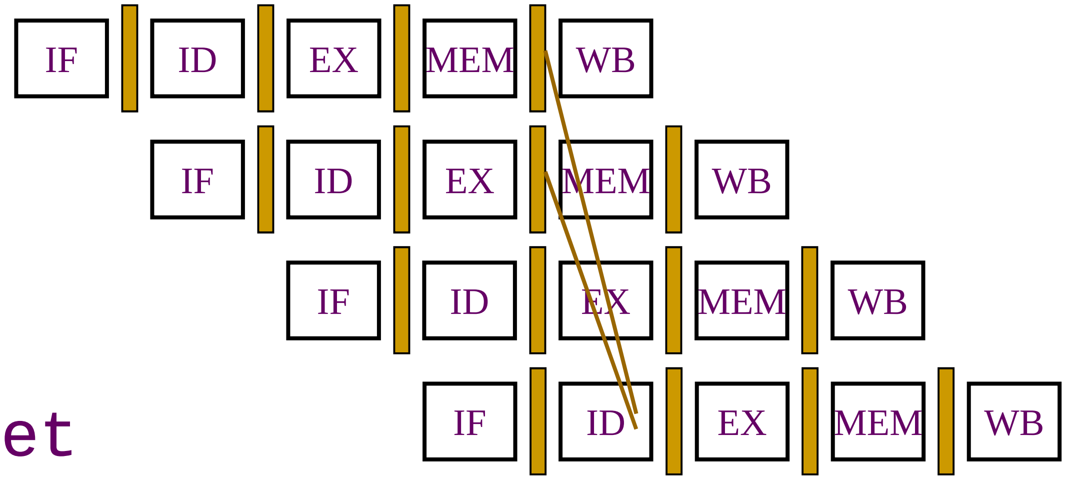


- A is the best choice, fills delay slot and reduces IC
- In B and C, the `sub` instruction may need to be copied, increasing IC
- In B and C, must be okay to execute `sub` when branch fails

# Data Hazards for Branches

- If a comparison register is a destination of 2nd or 3rd preceding ALU instruction

```
add $1, $2, $3
add $4, $5, $6
…
beq $1, $4, target
```

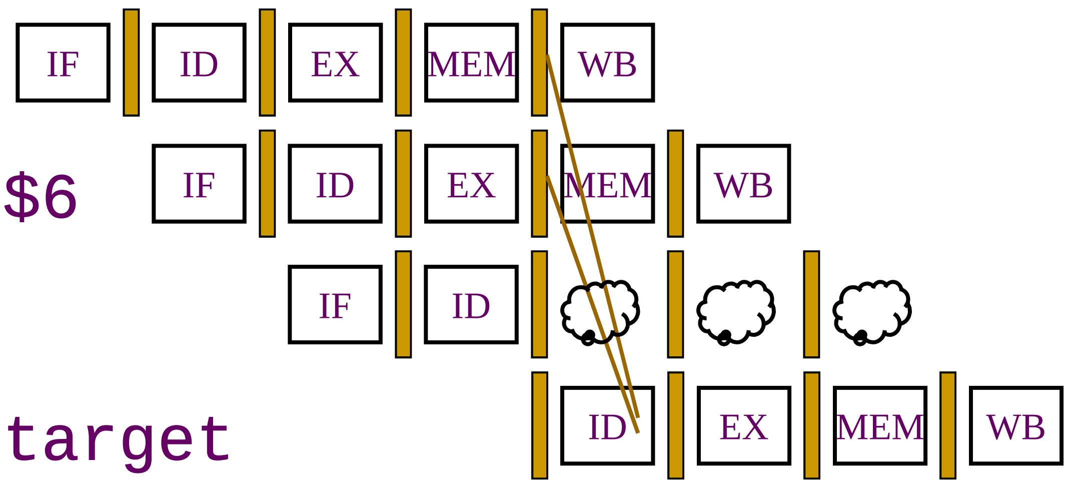


- Can resolve using forwarding

# Data Hazards for Branches

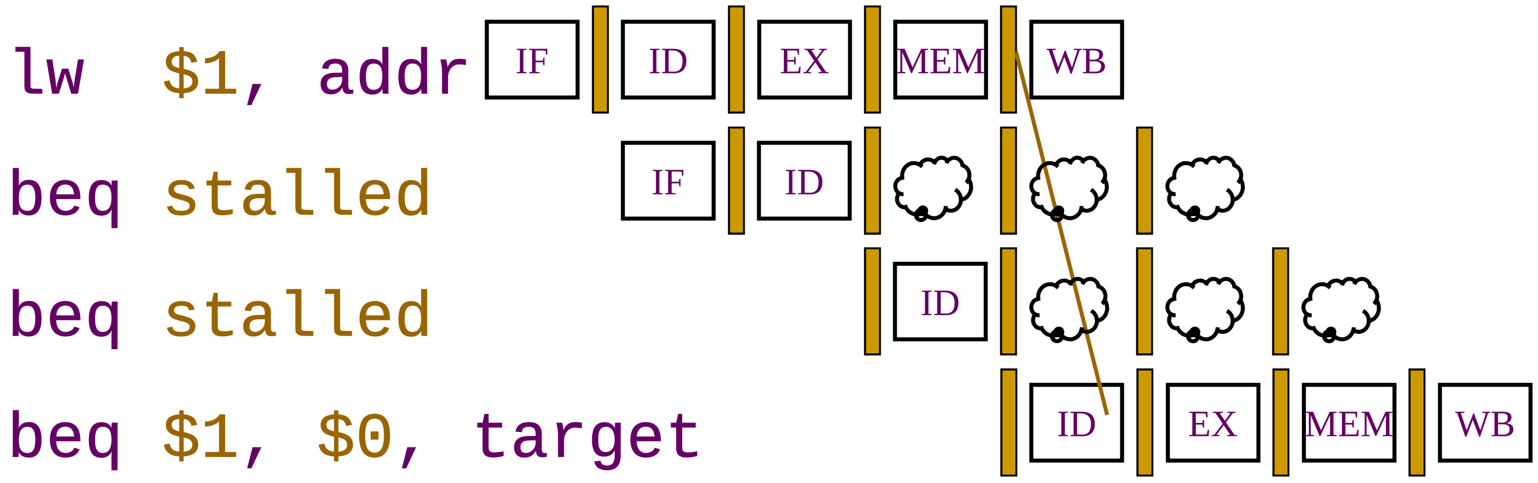
- If a comparison register is a destination of preceding ALU instruction or 2nd preceding load instruction
  - Need 1 stall cycle

```
lw  $1, addr
add $4, $5, $6
beq stalled
beq $1, $4, target
```



# Data Hazards for Branches

- If a comparison register is a destination of immediately preceding load instruction
  - Need 2 stall cycles



## Pipeline Performance a relook

- Pipeline CPI = Ideal Pipeline CPI + Structural Stalls + Data Hazard Stalls + Control Stalls

## Instruction-Level Parallelism (ILP)

- Pipelining:
  - Executing multiple instructions in parallel
  - Exploits potential parallelism among instructions
    - Is called Instruction Level Parallelism
- ILP – the parallelism among instructions
- To increase ILP
  - Deeper pipeline
    - Less work per stage $\Rightarrow$ shorter clock cycle
  - Multiple issue
    - Multiple instructions are launched in 1 clock cycle
    - Replicate pipeline stages $\Rightarrow$ multiple pipelines
    - Start multiple instructions per clock cycle
    - CPI < 1, so use Instructions Per Cycle (IPC)

- Ideal example
  - 6 GHz four-way multiple issue processor can execute a peak rate of 24 billion instructions per second
  - CPI = 0.25
  - IPC = 4
- But dependencies reduce this in practice

## Multiple Issue

- Multiple Issues can be implemented in 2 major ways [Based on division of work between compiler & Hardware]
- Static multiple issue
  - Compiler groups instructions to be issued together
  - Packages them into “issue slots”
  - Compiler detects and avoids hazards
- Dynamic multiple issue
  - CPU examines instruction stream and chooses instructions to issue each cycle
  - Compiler can help by reordering instructions
  - CPU resolves hazards using advanced techniques at runtime

## Speculation

- An approach that allows compiler or processor to “guess” the outcome of an instruction to remove it as a dependence in executing other instructions
- Speculation can be done in compiler or by the hardware
- Example
  - Speculate on the outcome of a branch so that the instructions after the branch could be executed earlier
  - Speculate that store that precedes a load does not refer to the same address

## Speculation

- "Guess" can be wrong
- Need method to check if guess was right
  - Check whether guess was right
    - If so, complete the operation
    - If not, roll-back and do the right thing
- Common to static and dynamic multiple issue
- Examples
  - Speculate on branch outcome
    - Roll back if path taken is different
  - Speculate on load
    - Roll back if location is updated
- Need method to unroll or back out the effects of the instructions that were executed speculatively
- Adds complexity to any processor supporting speculation

## Recovery

- Compiler speculation – compiler inserts additional instructions that checks the accuracy of the speculation & provide a fix-up routine to use when speculation is incorrect
- Hardware speculation – Processor usually buffers the speculative results until it knows they are no longer speculative
    - If speculation is correct, write the buffers to register/ Memory
    - Else flush the buffers and reexecute the correct instruction sequence