# Resolving Pipeline Hazards

- **Data hazard:** Data dependency arises when the destination of one instruction is used as a source in the next instruction. For example, the result (R4) of the first instruction $I_1$ is used as an operand in the next instruction $I_2$.

  $I_1$: MUL R2,R3,R4

  $I_2$: ADD R5,R4,R6

  Here, the decode stage of instruction $I_2$ is delayed until the write stage of the instruction $I_1$ has been completed (Figure 8.6).

  The delay can be reduced or possibly eliminated, if the result of the instruction $I_1$ can be forwarded (**Data forwarding**) directly for use in the instruction $I_2$ (Figure 8.7).

# Resolving Pipeline Hazards (contd.)

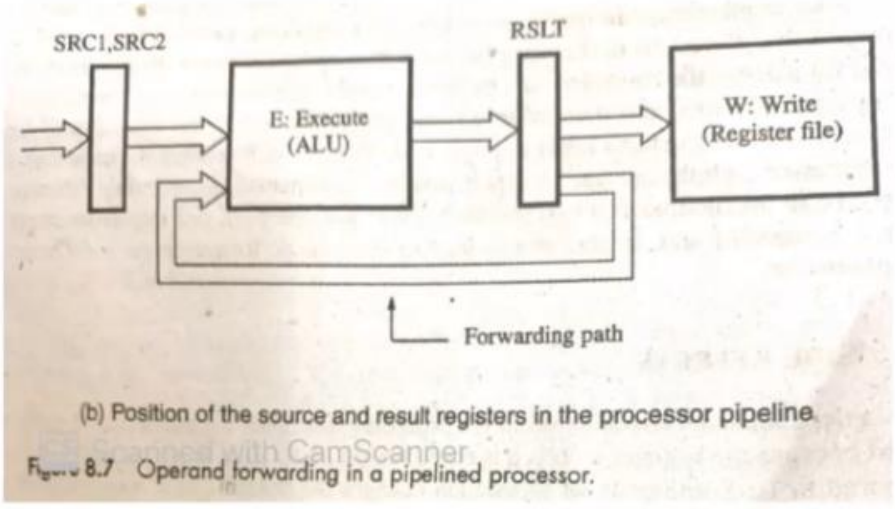



Figure: Resolving data hazard

# Resolving Pipeline Hazards(contd.)

Instead of implementing hardware, data dependency can be detected and dealt with **software**.

In this case, the compiler can insert two NOP (No Operation) instructions between $I_1$ and $I_2$to avoid the two cycle delays.

$I_1$: MUL R2,R3,R4
NOP
NOP
$I_2$: ADD R5,R4,R6

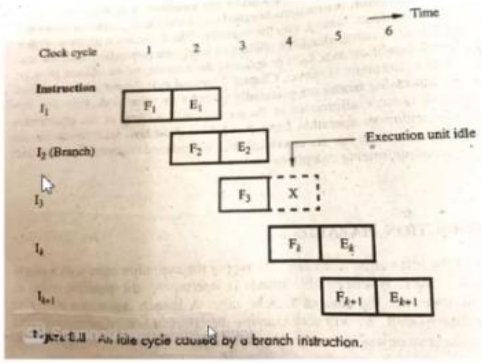
This leads to simpler hardware. On the other hand, the code size becomes larger.

- **Instruction hazard**: Instruction hazard due to a cache miss can be dealt with using two **separate caches** for data and instructions.

# Resolving Pipeline Hazards(contd.)

A branch instruction may also cause instruction hazard.

In the figure shown here, $I_2$ is a branch instruction (branch target $I_k$). In clock cycle 4, the processor must discard $I_3$, which has been fetched incorrectly, and fetch instruction $I_k$. Thus the pipeline is stalled for one clock cycle. This is referred to as **branch penalty**.



An idle cycle caused by a branch instruction.

# Resolving Pipeline Hazards(contd.)

To resolve this issue, the branch address needs to be computed earlier in the pipeline.

To reduce the effects of these pipeline stalls (both cache miss and branch instruction), many processors employ **sophisticated fetch units** that can fetch instructions before they are needed and put them in a **queue**.

To be effective, the fetch unit must have sufficient decoding and processing capabilities to recognize and execute branch instructions.

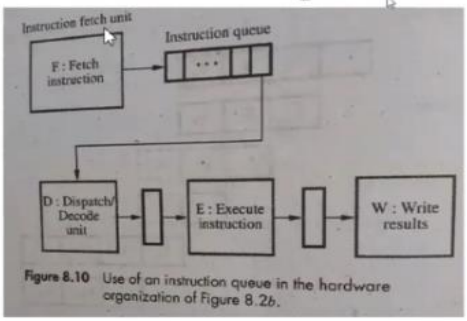


Figure 8.10 Use of an instruction queue in the hardware organization of Figure 8.2b.

- **Structural hazard**: Structural hazard may be avoided by adding **more hardware** or modifying the existing hardware to support **concurrent operations**.

# Superscalar Operation/Execution

- In pipelining, several instructions are present in the pipe line, but they are in different stages of their execution.
- In the absence of hazards, one instruction enters the pipeline and one instruction completes execution in each clock cycle.
- The maximum throughput of a pipelined processor is one instruction/clock cycle.
- A more aggressive approach is to equip the processor with multiple processing units to handle several instructions in parallel in each processing stage.
- In this case, several instructions start execution in the same clock cycle.
- Such processors are capable of achieving an instruction execution throughput of **more than one instruction/clock cycle**.
- These type of processors are known as **superscalar processors**.

# Superscalar Operation/Execution(contd.)

- In a superscalar processor, the compiler can avoid many hazards.

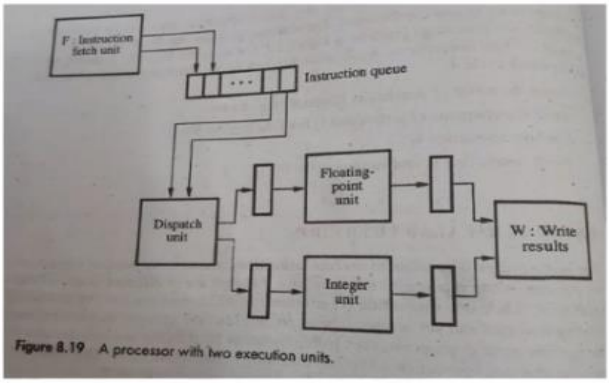


Figure 8.19 A processor with two execution units.

# Memory Mapping

- **Memory mapping** is the translation/transformation between the logical address space and the physical memory.
- The transformation of data from main memory to cache memory is called mapping.
- **Address** uniquely identifies a location of a memory.
- There are two types of address: **logical address** and **physical address**.
- Logical address is generated by the CPU while a program is running.
- Logical address is a virtual address. User can see it.
- Physical address is the actual location in the memory unit. User can not see it.
- Logical address is used as a reference to access the physical address.
- Physical address is computed by the memory management unit.

# Memory Mapping

- There are three types of memory mapping:
  - ➢Direct mapping
  - ➢Associative mapping
  - ➢Set associative mapping