Processor Structure
&
Function

Organization

- **Functions**
  - **Fetch Instruction**
  - **Interpret Instruction**
  - **Fetch Data** — **minor cycle**
  - **Execute Instruction, i.e., process data**
  - **Write data results, to memory or I/O module**

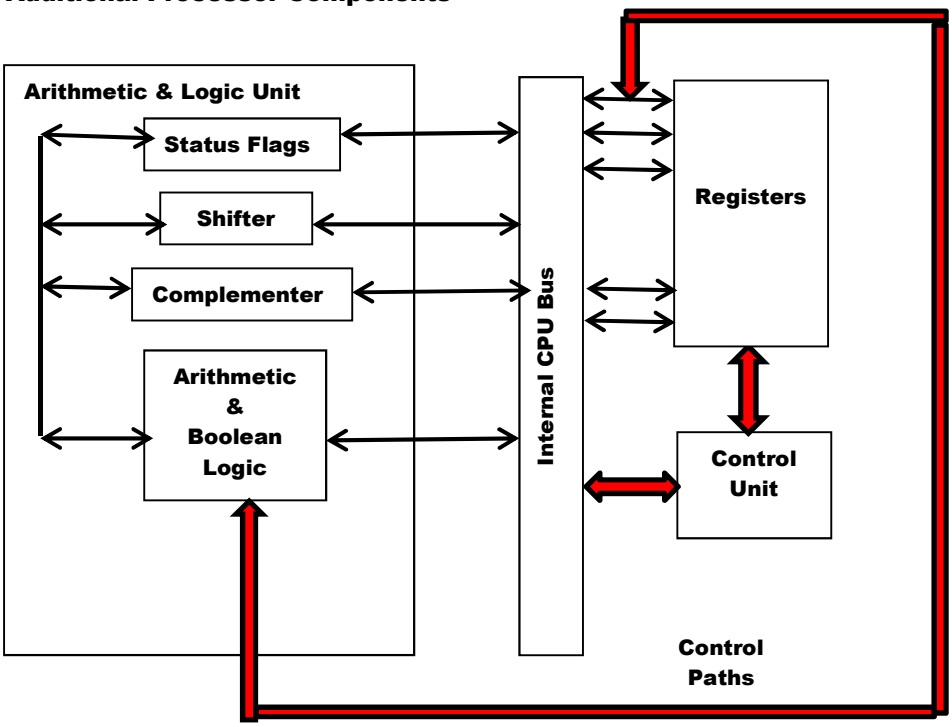
major cycle

Basic Machine Cycle

- **Major Components of Processor**
  - **ALU**
  - **Control Unit**
  - **Registers**
  - **Internal Cache**

Von Neumann computer

- **Additional Processor Components**



- **Register Organization**
  - **User Visible Registers, referenced by machine language**
    - **General Purpose Registers**
      - **Orthogonal – any register can contain the operand for any opcode**
      - **Dedicated – e.g., stack registers, floating-point registers**
    - **Data Registers**
      - **Used only to hold data, excluding addresses**
      - **Used to hold data including addresses**
    - **Address Registers**
      - **Partially General Purpose, e.g., X register in Pep/8**
      - **Dedicated**
        - ✓ **Segment Pointers, i.e., Registers**
          - ★ **holds the address of the base of the segment**
        - ✓ **Index Registers**
          - ★ **Indexed addressing, may autoindex**
        - ✓ **Stack Pointer**
          - ★ **Enables push, pop, etc.**
    - **Condition Code (Program Status Word) registers**
      - **Provide status of most recent instruction executions**
      - **Set condition codes for testing**

  - **Control Registers,**
    - **used by**
      - **control unit (control processor operations)**
      - **privileged instructions (control program execution)**

    - **registers**
      - **PC – address of next instruction to be fetched**
      - **IR – instruction most recently fetched**
      - **MAR – memory address register**
        - ✓ **holds memory address**
        - ✓ **connects directly to the address bus**
      - **MBR – memory buffer register**
        - ✓ **holds data**
          - ★ **To be written to memory**
          - ★ **Most recently read from memory**
        - ✓ **connects directly to the data bus**

ALU may connect
- **directly to the MBR and User-Visible Registers**

or
- **there may be additional buffering registers between the ALU and the MBR and the User-Visible Registers**

User-Visible Registers exchange data with the MBR

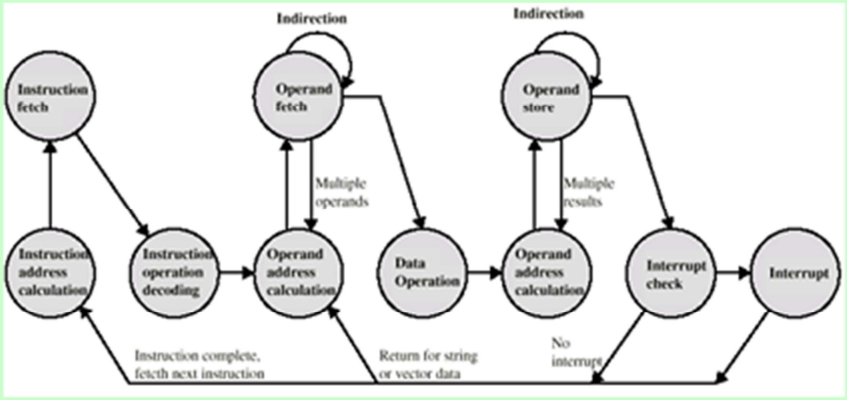
- **Status Registers**
  - **Single register or a set of registers that contain the program and operating status**
  - **PSW -- Program Status Word**
    - **Sign bit**
    - **Zero bit**
    - **Carry bit**
    - **Equal bit – set if logical compare result is equality**
    - **Overflow bit**
    - **Interrupt Enable/Disable bits**
    - **Interrupt Vector Register**
    - **upervisor bit – indicates whether current instruction is executing in supervisor or user mode**
    - **Pointer to PCB data structures – process control blocks**
    - **System Stack Pointer**
    - **Page Table Pointer**
    - **I/O Control Registers**

Microprocessor Register Organizations
Motorola MC68000 versus Intel 8086
pages 439-440

- **Instruction Cycle**
  - **Fetch**
    - **(PC) → MAR → Address Bus**
    - **Control Unit requests memory read**
    - **Content of memory location → Data Bus → MBR → IR (page 443, fig 12.6)**
    - **PC incremented**
  - **Interpret -- Control Unit**
    - **examines the contents of the IR**
    - **determines the addressing mode of the operand specifier, e.g., indirect**
      - **control unit executes the indirect cycle (page 443 figure 12.7)**
  - **Execute**
  - **Interrupt, e.g.,**
    - **(PC) → MBR → Memory**
    - **(SP) → MAR (page 444 figure 12.8)**

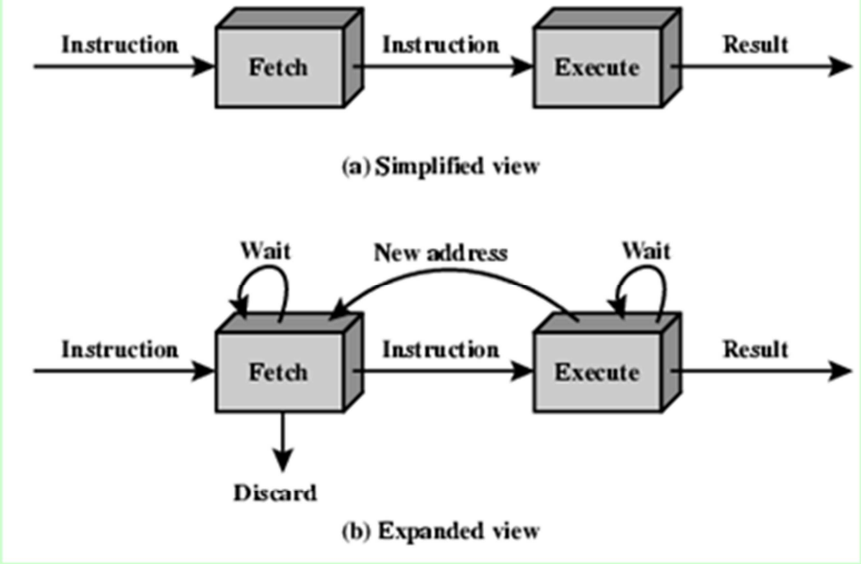
Instruction Pipelining

Instruction Cycle State Diagram



Fetch next instruction in parallel with the execution of the previous instruction
Instruction prefetch or fetch overlap

Two Stage Instruction Pipeline



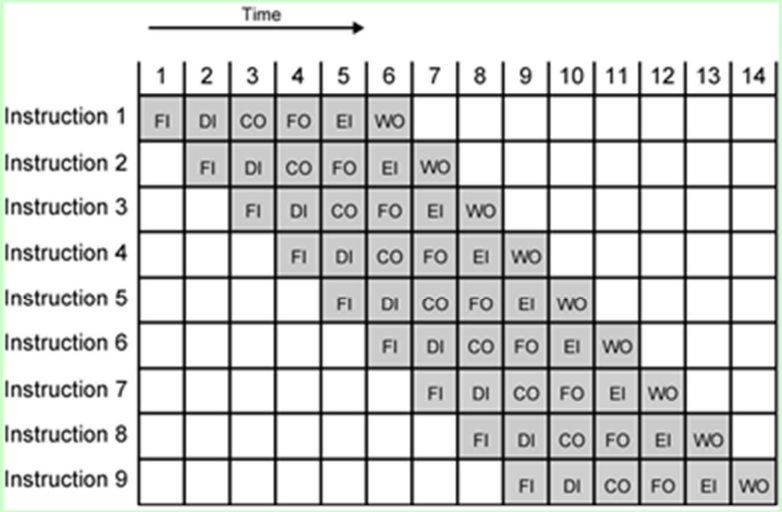
Execution time generally longer than fetch time, e.g., reading & storing operands, etc.
Conditional branch instruction → address of next instruction is unknown until execution is complete

Rule:
Conditional branch instruction → fetch the next one in memory after the conditional instruction.
If branch not taken, discard fetched instruction and fetch the appropriate instruction.

Pipeline Units

- **FI (Fetch Instruction)** — **fetch next expected instruction into buffer**
- **DI (decode Instruction)** — **determine opcode & operand specifiers**
- **CO (Calculate Operands)** — **calculate effective address of each operand, i.e., mode**
  - **Displacement**
  - **Register Indirect**
  - **Indirect**
  - **etc**
- **FO (Fetch Operands)** — **fetch each operand from memory**
  operands in Registers need not be fetched
- **EI (Execute Instruction)** — **perform operation & store result, if any, in the specified destination operand location**
- **WO (Write Operand)** — **store the result in memory**

Timing Diagram for Instruction Pipeline Operation

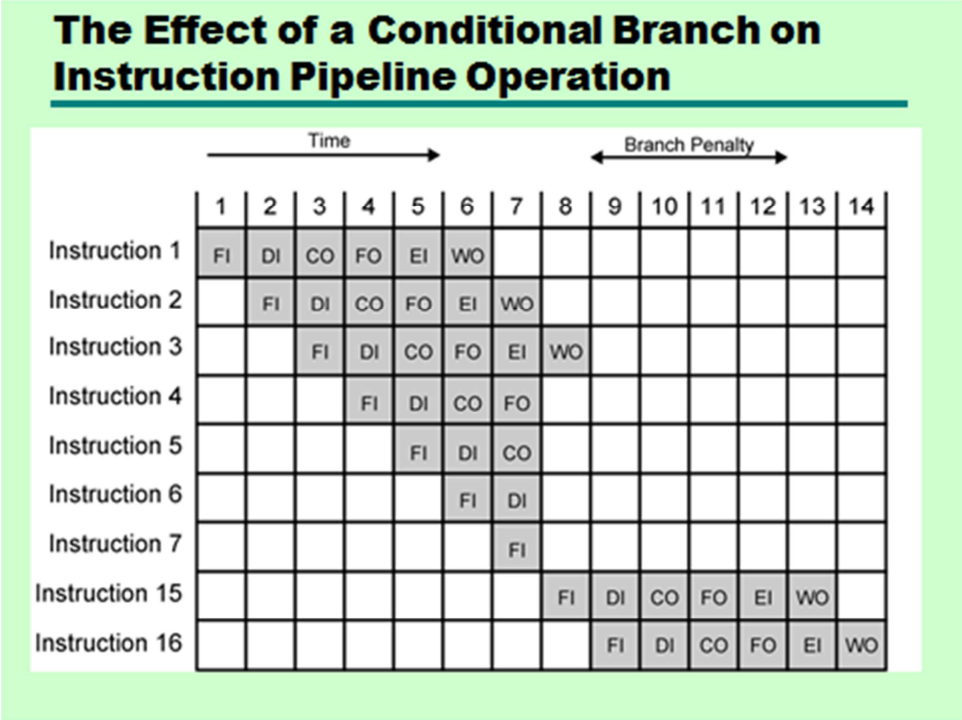


Under Ideal Conditions

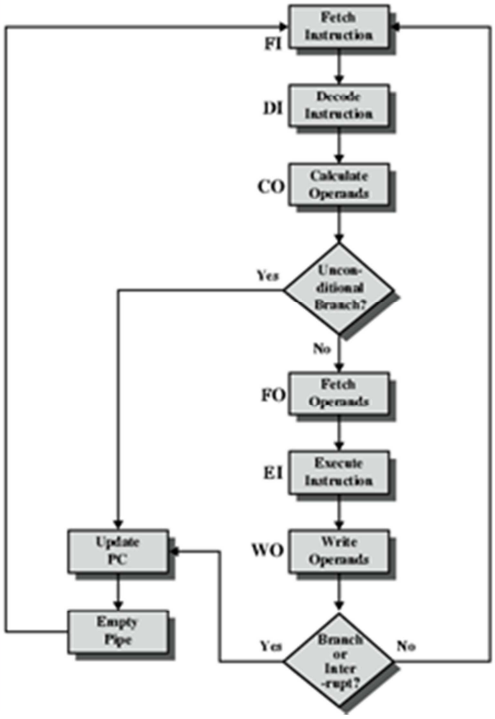
- ⋆ **All instructions go through all six stages**
- ⋆ **No memory conflicts**
- ⋆ **All stages execute simultaneously with the same execution time**

Unpredictable Events

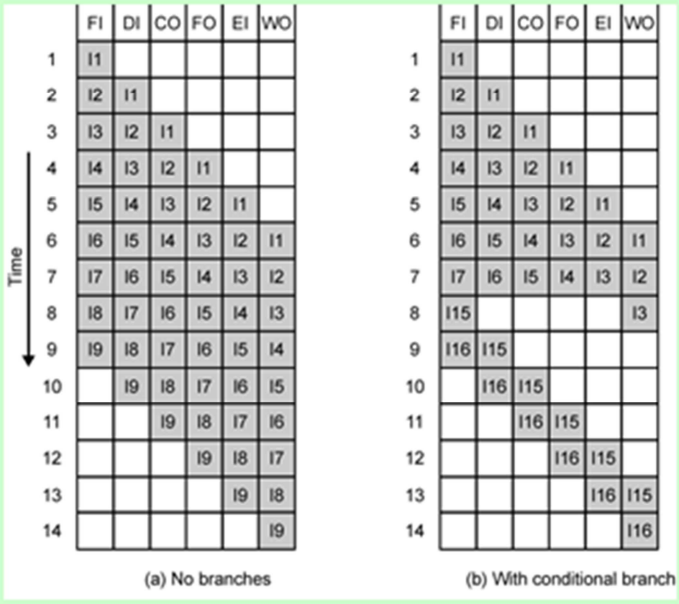
- **Not all instructions go through all six stages**
- **Memory conflicts exist**
- **All stages do not execute simultaneously with the same execution time**
- **CO Stage may depend on the contents of a register that could have been altered by an instruction that is still in the pipeline**
- **Interrupts occur → disrupt the ideal conditions**
- **Conditional branch instructions may invalidate multiple instruction fetches**
  - **Instruction 3 is a conditional branch to Instruction 15**
  - **Branch not taken is not determined until the end of time unit 7**
  - **Pipeline must be flushed**
  - **During time unit 8 instruction 15 enters the pipeline**
  - **No instructions complete during time units 9 through 12**



Six Stage Instruction Pipeline



Alternative Pipeline Depiction

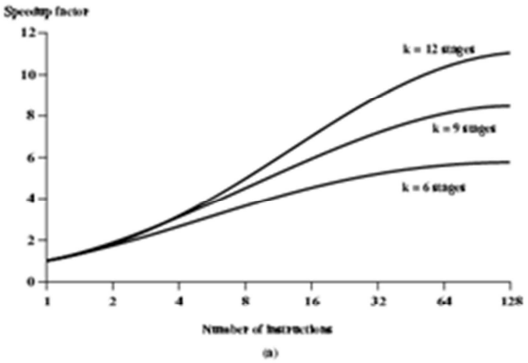


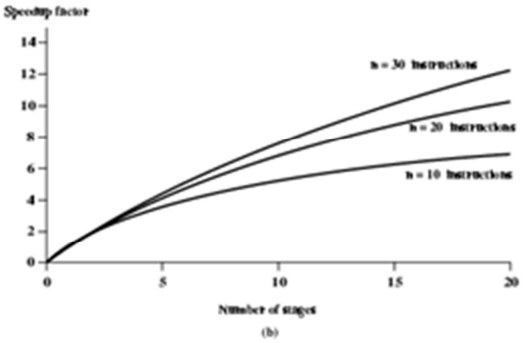
Design Limitations

- **There is overhead in moving from one pipeline unit to another**
- **This overhead can appreciably lengthen the execution time of a single instruction**
- **The amount of control logic required to handle memory and register dependencies increases enormously with the number of stages**
- **Latching Delay – pipeline buffers take time to operate → adds to the instruction cycle time**

Pipeline Performance Analysis (pages 450-451)

Speedup Factors with Instruction Pipelining





Pipeline Hazards (Pipeline Bubble)

Pipeline or some portion of the pipeline becomes stalled because conditions do not permit continued execution

- **Resource Hazard (Structural Hazard)**
  - **Two or more instructions that are already in the pipeline need the same resource**
  - **These instructions need to be executed serially, not in parallel to one another e.g., several instructions both are ready to enter the EI (execution stage) but there is only one ALU hence one instruction executes immediately, the next waits for one cycle, the next waits for two cycles, etc.**

- **Data Hazards**
  - **There is a conflict in the access to an operand location**
    - ★ **Two instructions to execute in parallel in the pipeline and both require access to the same operand location**
    - ★ **Operand value may be updated by one instruction in such a way as to produce a different result than if the two instructions had been executed serially**

  - **<u>Read after Write</u>**
    - ★ **Ideal**
      Instruction 1 writes to location A and then Instruction 2 reads from it
    - ★ **Hazard**
      Instruction 2 reads from location A before Instruction 1 writes to it

  - **<u>Write after Read</u>**
    - ★ **Ideal**
      Instruction 1 reads from location A and then Instruction 2 writes to it
    - ★ **Hazard**
      Instruction 2 writes to location A and then Instruction 1 reads from it

  - **<u>Write after Write</u>**
    - ★ **Ideal**
      Instruction 1 and Instruction 2 both write to location A
    - ★ **Hazard**
      The write operations take place in the reverse order of that intended

- **Control Hazards (Branch Hazard)**

(see Stallings PP Slides)