Computer Organization and Architecture

15 CPU Structure and Function

Tongwei Ren

Dec. 5, 2019

南京大學
NANJING UNIVERSITY

Review

- Addressing modes
  - Immediate, direct, indirect, register, register indirect, displacement, stack
- Instruction format
  - Instruction length, allocation of bits, variable-length instructions

Task of CPU

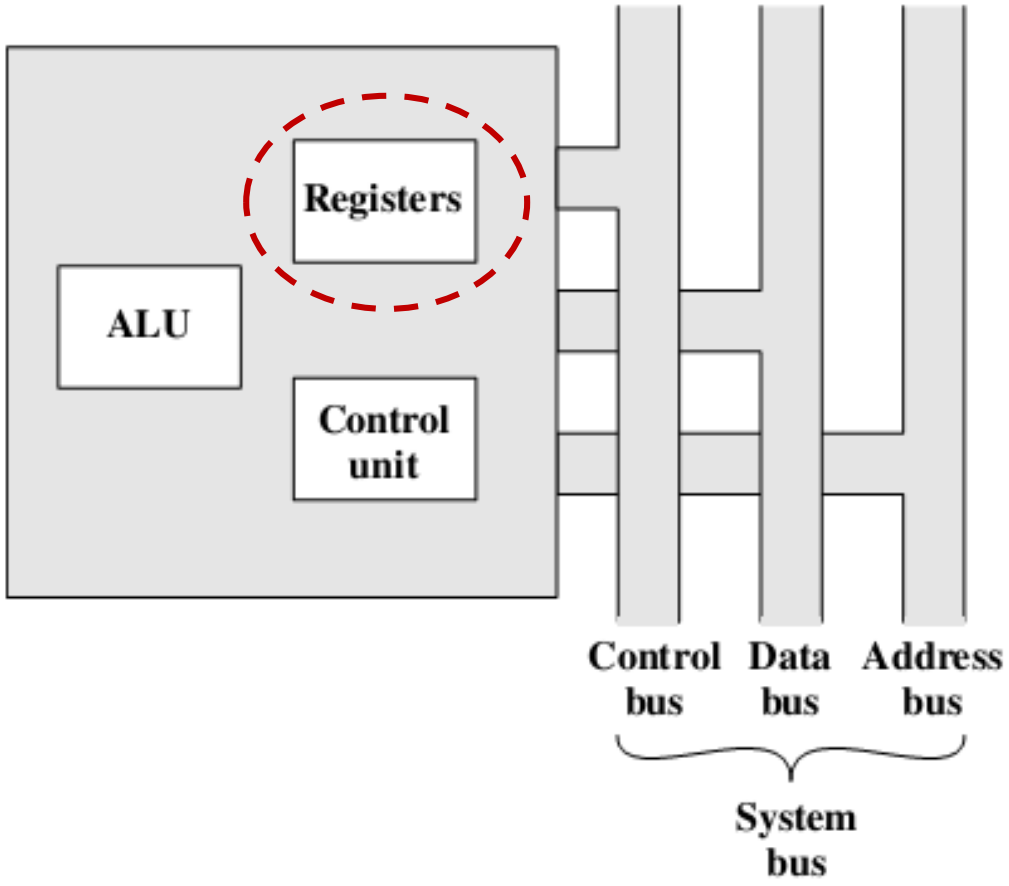
- Fetch instruction: The processor reads an instruction from memory (register, cache, main memory)
- Interpret instruction: The instruction is decoded to determine what action is required
- Fetch data: The execution of an instruction may require reading data from memory or an I/O module
- Process data: The execution of an instruction may require performing some arithmetic or logical operation on data
- Write data: The results of an execution may require writing data to memory or an I/O module

CPU Requirement

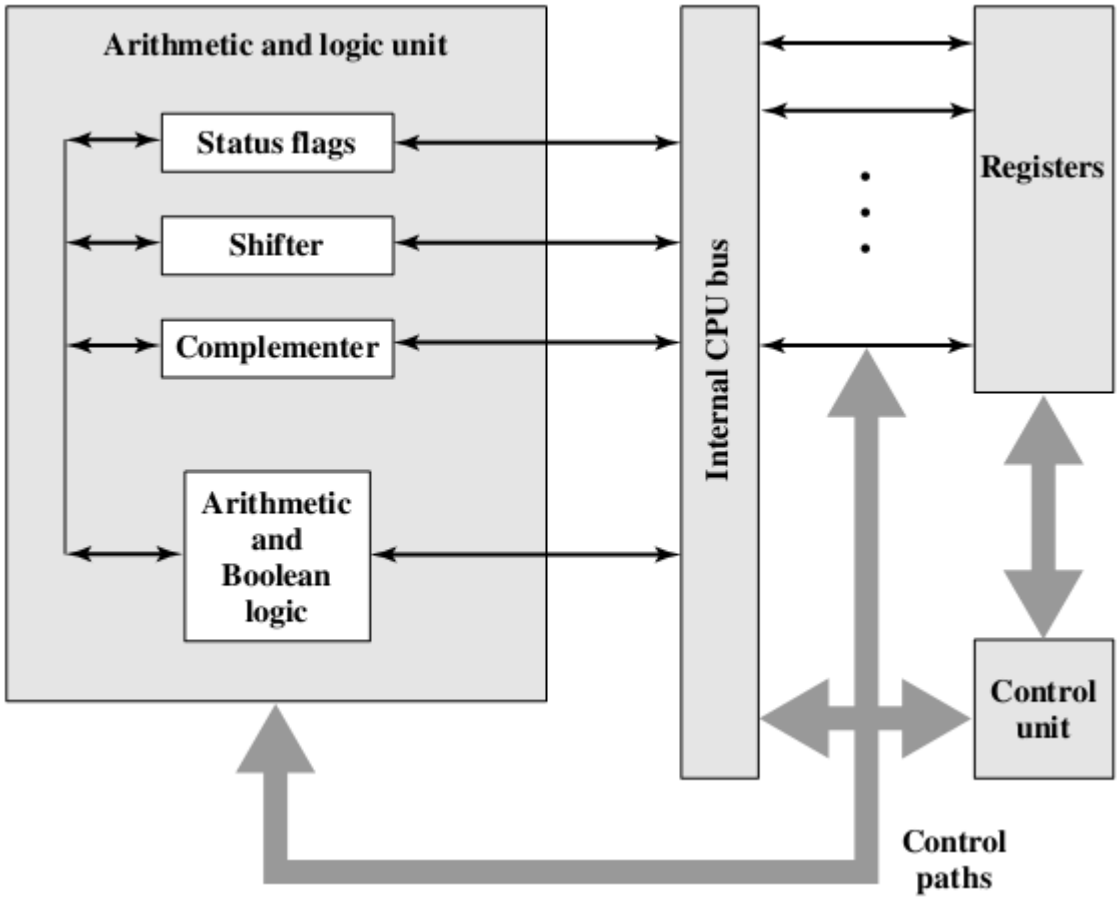
- The processor needs to store some data temporarily
- It must remember the location of the last instruction so that it can know where to get the next instruction
- It needs to store instructions and data temporarily while an instruction is being executed

The processor needs a small internal memory

Simplified View of Processor



Detailed View of Processor



Register Organization

- The registers in the processor perform two roles:
  - User-visible registers: Enable the machine- or assembly language programmer to minimize main memory references by optimizing use of registers
  - Control and status registers: Used by the control unit to control the operation of the processor and by privileged, operating system programs to control the execution of programs

User-visible Register

- General purpose register
  - Assigned to a variety of functions by programmer
- Data register
  - Used only to hold data and cannot be employed in the calculation of an operand address
- Address register
  - General purpose or devoted to a particular addressing mode
  - E.g.: segment pointer, index register, stack register, …

User-visible Register (cont.)

- Design issues
  - Whether to use completely general-purpose registers or to specialize their use
  - Number of registers
    - Fewer registers result in more memory references
    - More registers do not noticeably reduce memory references
  - Register length
    - Must be at least long enough to hold the largest address or values of most data types
    - Some machines allow two contiguous registers to be used as one for holding double-length values

User-visible Register (cont.)

- Condition codes register
  - Condition codes are bits set by the processor hardware as the result of operations
  - At least partially visible to the user

Advantages	Disadvantages
1. Because condition codes are set by normal arithmetic and data movement instructions, they should reduce the number of COMPARE and TEST instructions needed. 2. Conditional instructions, such as BRANCH are simplified relative to composite instructions, such as TEST AND BRANCH. 3. Condition codes facilitate multiway branches. For example, a TEST instruction can be followed by two branches, one on less than or equal to zero and one on greater than zero.	1. Condition codes add complexity, both to the hardware and software. Condition code bits are often modified in different ways by different instructions, making life more difficult for both the microprogrammer and compiler writer. 2. Condition codes are irregular; they are typically not part of the main data path, so they require extra hardware connections. 3. Often condition code machines must add special non-condition-code instructions for special situations anyway, such as bit checking, loop control, and atomic semaphore operations. 4. In a pipelined implementation, condition codes require special synchronization to avoid conflicts.

User-visible Register (cont.)

- Store and recovery
  - A subroutine call will result in the automatic saving of all user-visible registers, to be restored on return
    - The processor performs the saving and restoring as part of the execution of call and return instructions
    - This allows each subroutine to use the user-visible registers independently
  - The programmer should save the contents of the relevant user-visible registers prior to a subroutine call, by including instructions for this purpose in the program

Control and Status Register

- Employed to control the operation of the processor
- Most of these, on most machines, are not visible to user
  - Some of them may be visible to machine instructions executed in a control or operating system mode

Control and Status Register (cont.)

- Program counter (PC)
  - Contains the address of an instruction to be fetched
  - Typically, the processor updates the PC after each instruction fetch so that the PC always points to the next instruction to be executed
  - A branch or skip instruction will also modify the contents of the PC
- Instruction register (IR)
  - Contains the instruction most recently fetched
  - The fetched instruction is loaded into an IR, where the opcode and operand specifiers are analyzed

Control and Status Register (cont.)

- Memory address register (MAR)
  - Contains the address of a location in memory
  - MAR connects directly to the address bus
- Memory buffer register (MBR)
  - Contains a word of data to be written to memory or the word most recently read
  - MBR connects directly to the data bus, and user-visible registers, in turn, exchange data with the MBR
  - ALU may have direct access to the MBR and user-visible registers

Control and Status Register (cont.)

- Program status word (PSW): A register or set of registers contain status information
  - Sign: Sign bit of the result of the last arithmetic operation
  - Zero: Set when the result is 0.
  - Carry: Set if an operation resulted in a carry (addition) into or borrow (sub-traction) out of a high-order bit
  - Equal: Set if a logical compare result is equality
  - Overflow: Indicate arithmetic overflow
  - Interrupt enable/disable: Enable or disable interrupts
  - Supervisor: Indicates whether the processor is executing in supervisor or user mode

Control and Status Register (cont.)

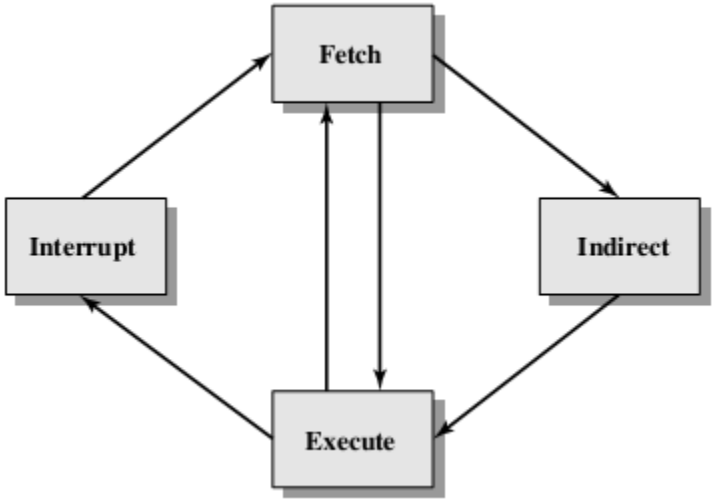
- Other registers related to status and control
  - A pointer to a block of memory containing additional status information
  - In machines using vectored interrupts, an interrupt vector register may be provided
  - If a stack is used to implement certain functions, a system stack pointer is needed
  - A page table pointer is used with a virtual memory system

Control and Status Register (cont.)

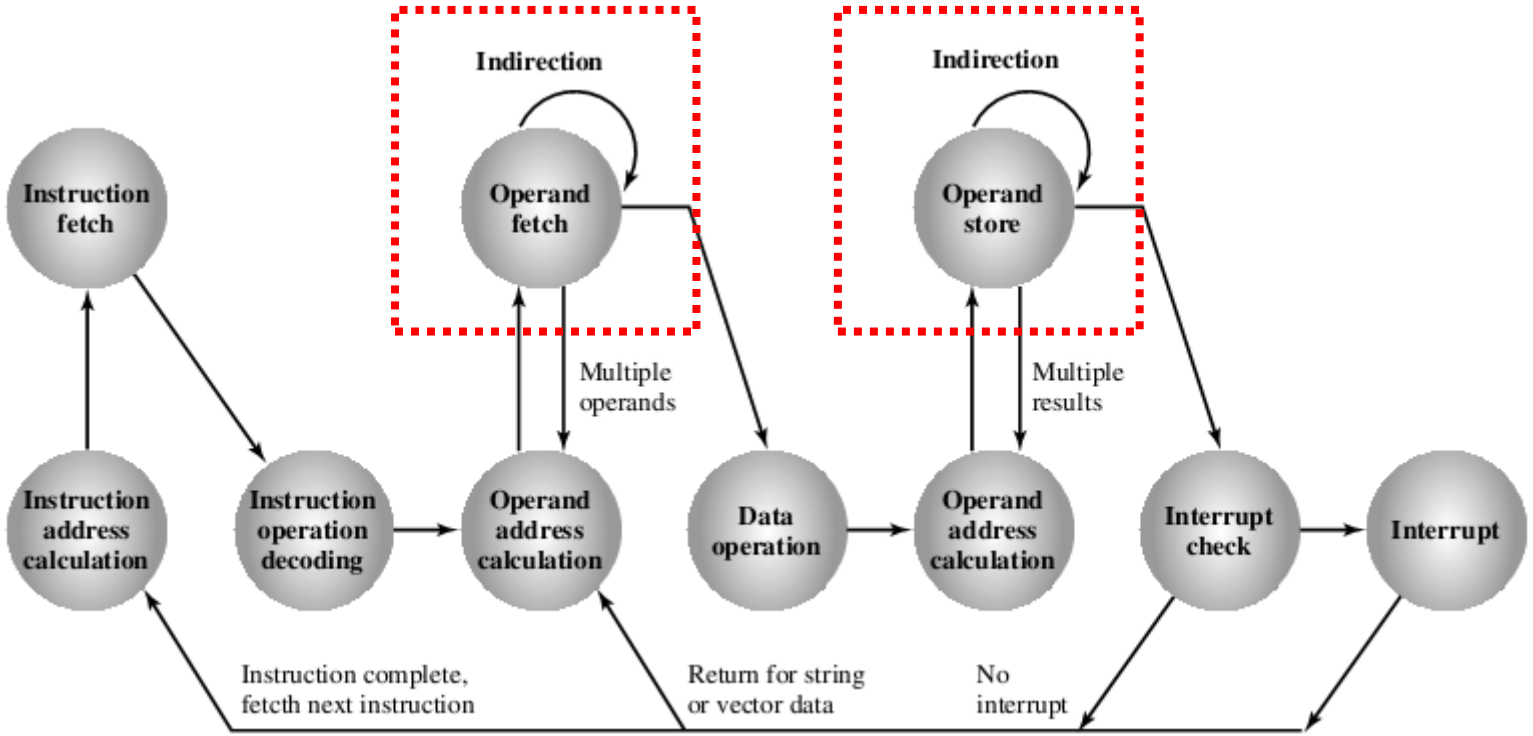
- Design issues
  - Operating system support
    - Certain types of control information are of specific utility to the operating system
    - If the processor designer has a functional understanding of the operating system to be used, register organization can to some extent be tailored to the operating system
  - Allocation of control information between registers and memory
    - It is common to dedicate the first (lowest) few hundred or thousand words of memory for control purposes
    - Trade-off of cost versus speed arises

Indirect Cycle

- The execution of an instruction may involve one or more operands in memory, each of which requires a memory access
- If indirect addressing is used, additional memory accesses are required
- Treat the fetching of indirect addresses as one more instruction stages

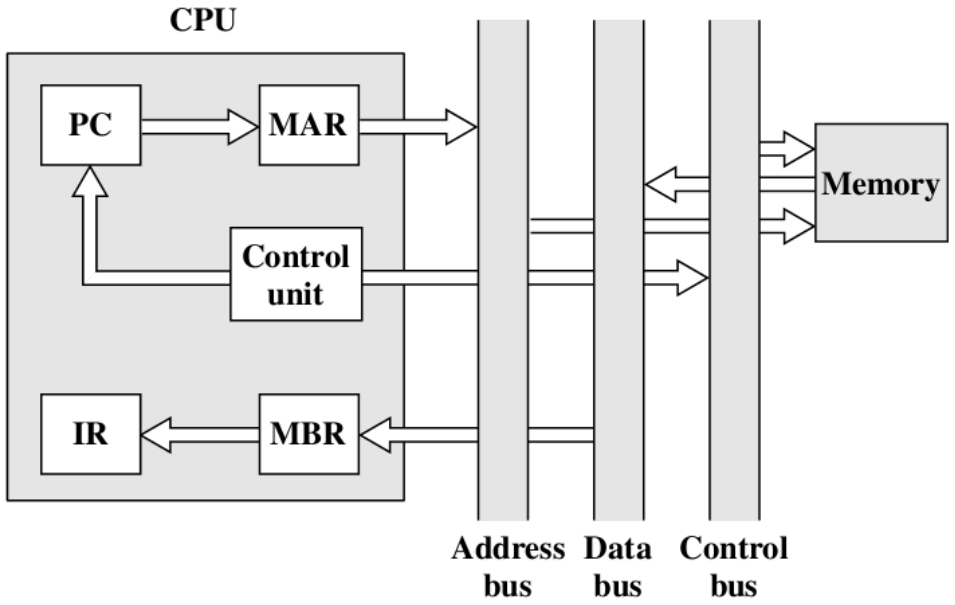


Indirect Cycle (cont.)



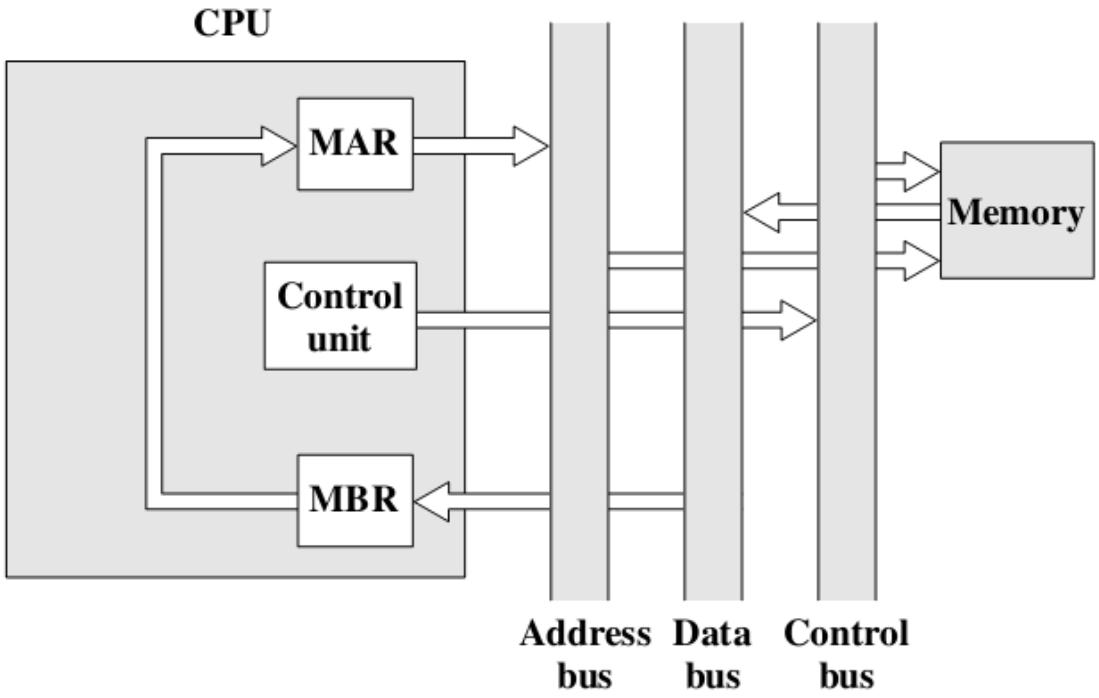
Data Flow

- Assume a processor that employs a memory address register (MAR), a memory buffer register (MBR), a program counter (PC), and an instruction register (IR)
- Fetch cycle



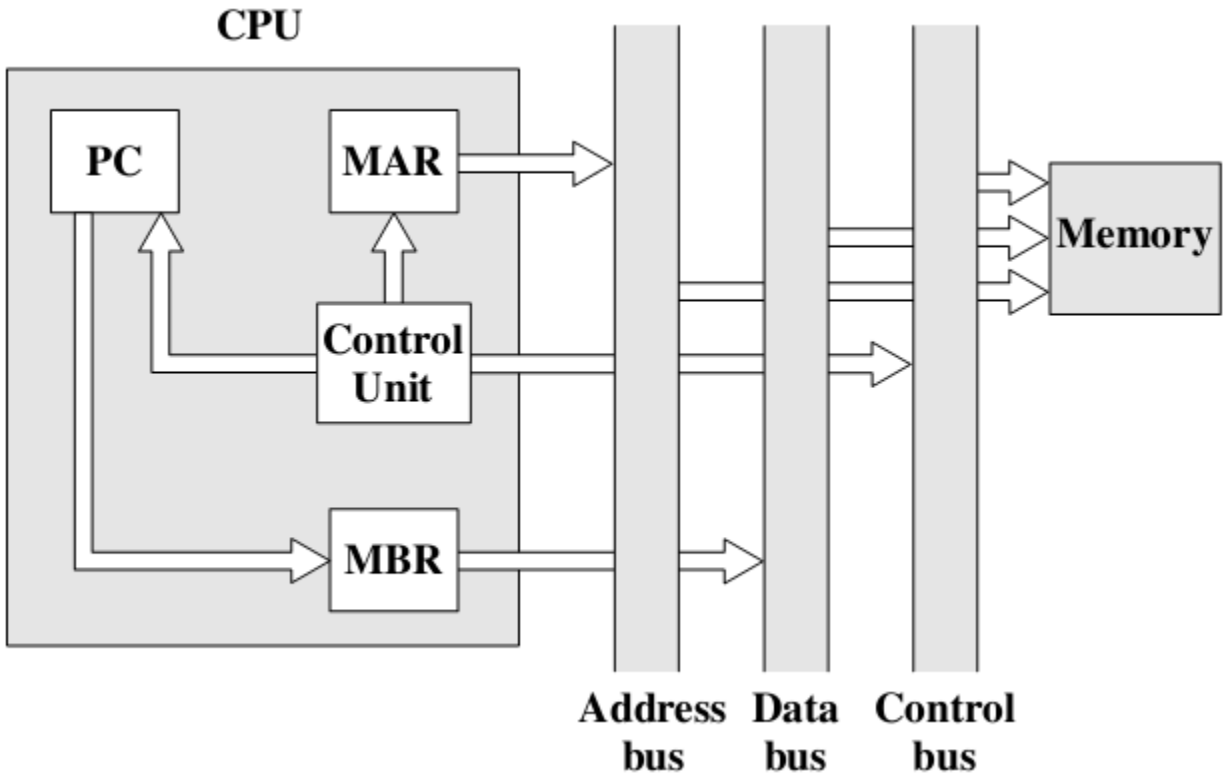
Data Flow (cont.)

- Indirect cycle



Data Flow (cont.)

- Interrupt cycle

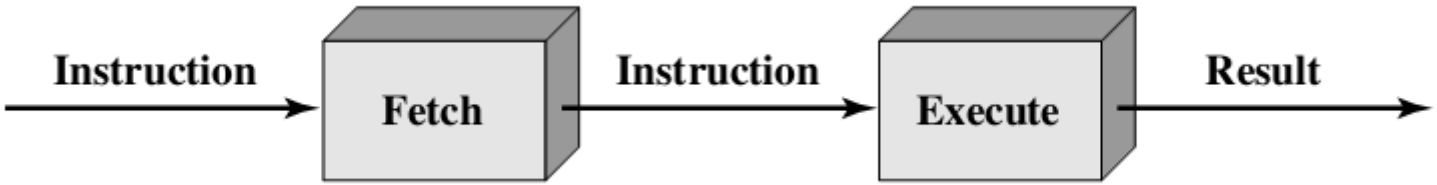


Instruction Pipelining

- Pipelining
  - If a product goes through various stages of production, products at various stages can be worked on simultaneously by laying the production process out in an assembly line
- In fact, an instruction has a number of stages

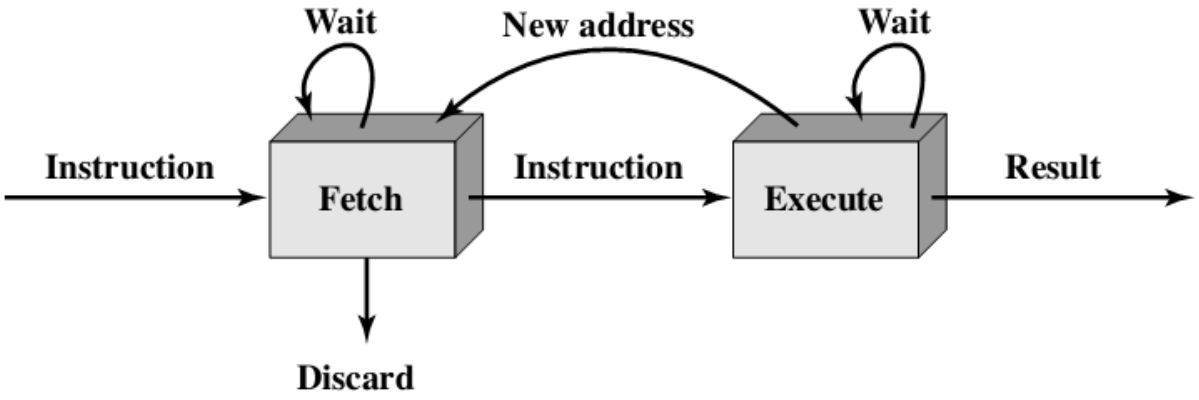
Two Stages Solution

- Subdivide instruction processing into two stages: fetch instruction and execute instruction
- Fetch the next instruction in parallel with the execution of the current one
- Problem: memory access conflict



Two Stages Solution (cont.)

- There are times during the execution of an instruction when main memory is not being accessed
- More problems
  - Execution time will generally be longer than fetch time
  - A conditional branch instruction makes the address of the next instruction to be fetched unknown

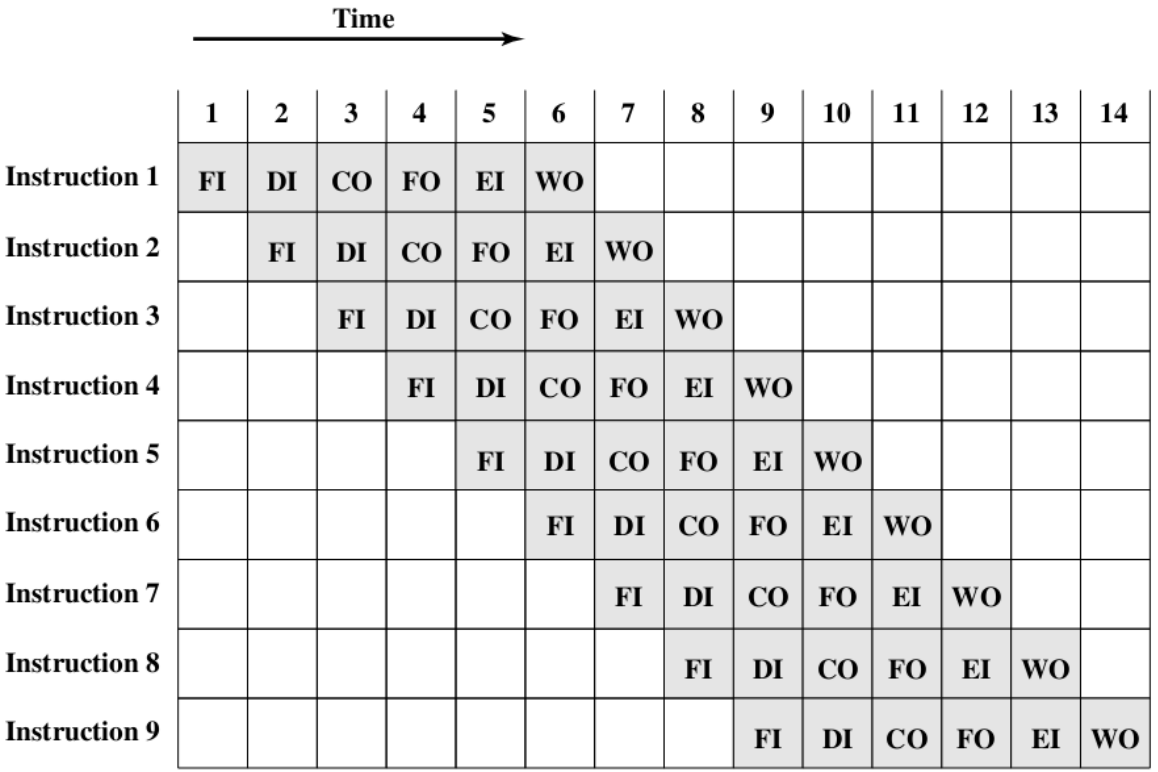


Six Stages Solution

- To gain further speedup, the pipeline must have more stages
  - Fetch instruction (FI): Read the next expected instruction into a buffer
  - Decode instruction (DI):Determine opcode and operand specifiers
  - Calculate operands (CO): Calculate effective address of each source operand
  - Fetch operands (FO): Fetch each operand from memory. Operands in registers need not be fetched
  - Execute instruction (EI): Perform the indicated operation and store the result, if any, in the specified destination operand location
  - Write operand (WO):Store the result in memory
- The various stages will be of more nearly equal duration

Six Stages Solution (cont.)

- Example: reduce the execution time for 9 instructions from 54 time units to 14 time units

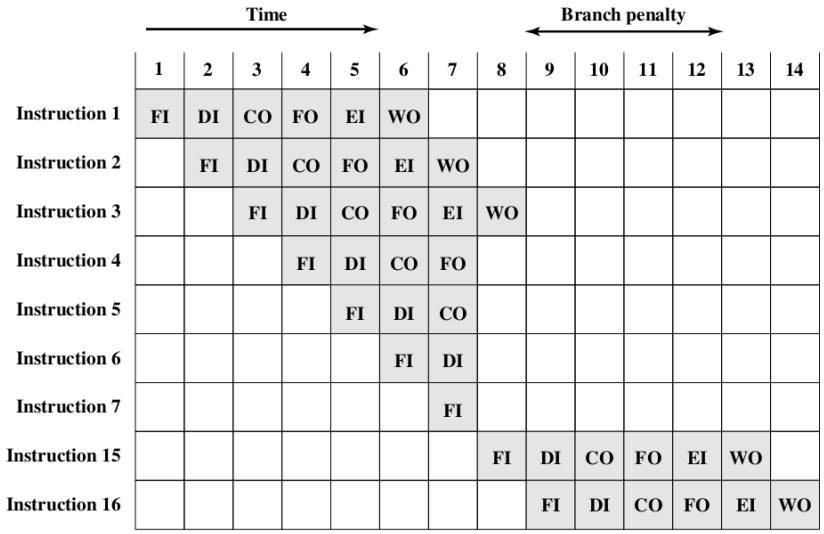


Six Stages Solution (cont.)

- Comment
  - Not all the instructions contain six stages
    - E.g.: Load instruction does not need the WO stage
    - To simplify hardware, the timing is set up assuming that each instruction requires all six stages
  - Not all the stages can be performed in parallel
    - E.g.: FI, FO, and WO stages involve a memory access
    - The desired value may be in cache, or the FO or WO stage may be null

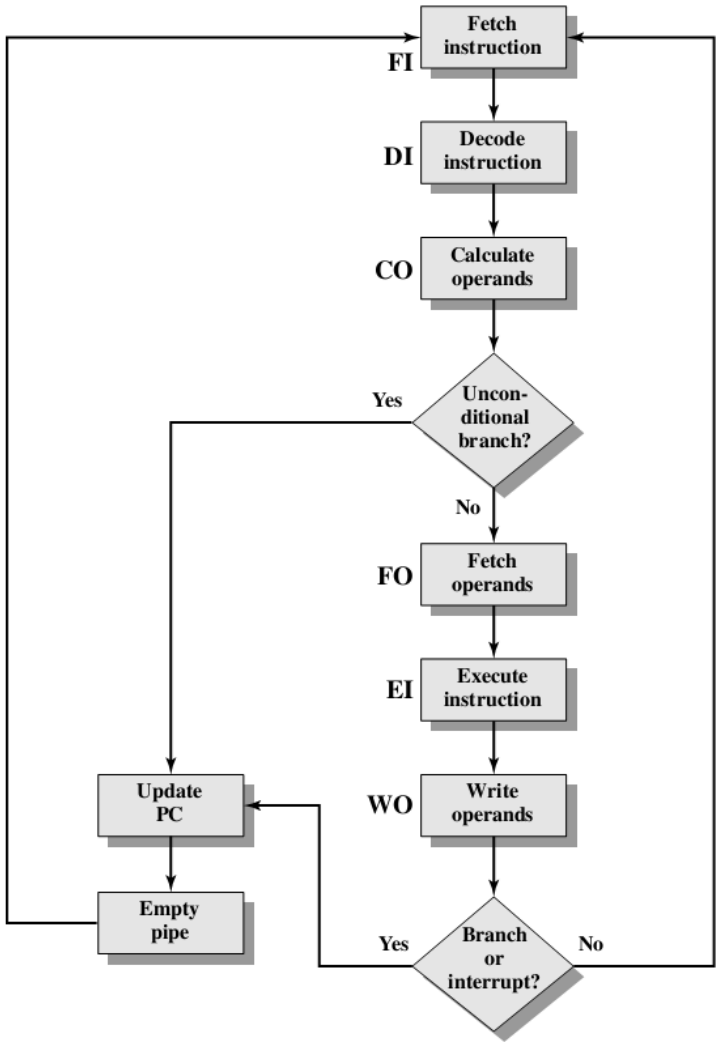
Six Stages Solution (cont.)

- Limitation
  - If the six stages are not of equal duration, there will be some waiting involved at various pipeline stages
  - Conditional branch instruction can invalidate several instruction fetches



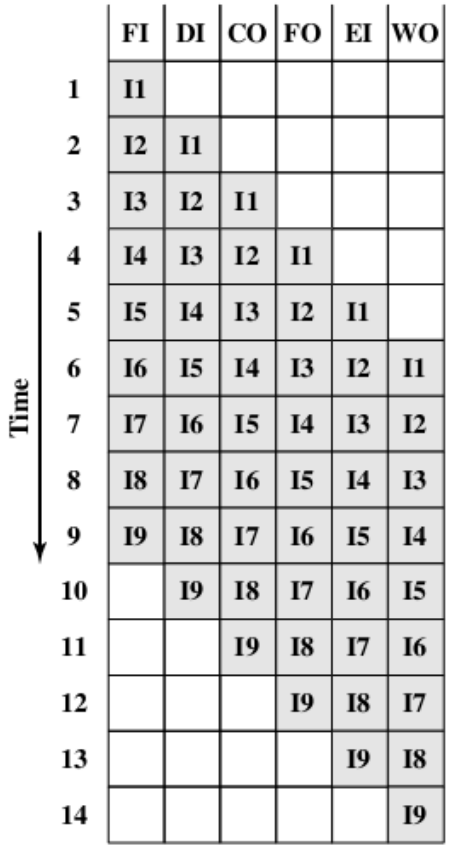
Six Stages Solution (cont.)

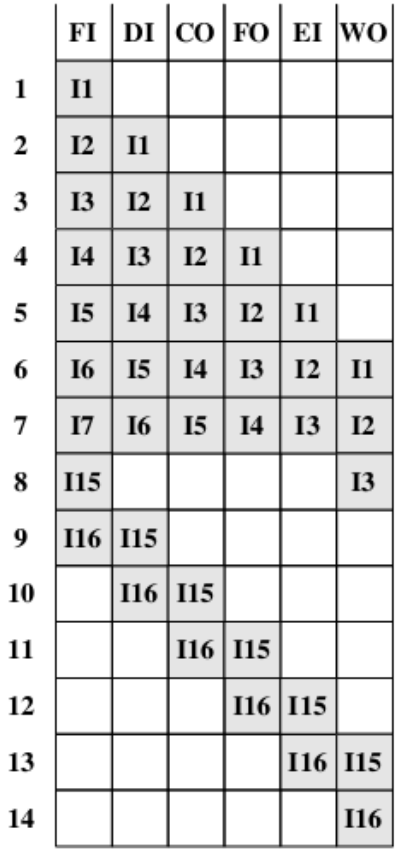
- Limitation (cont.)
  - Interrupt



Six Stages Solution (cont.)

- Another viewpoint





Pipeline Performance

- Assume
  - t_i: time delay of the circuitry in the *i*th stage of pipeline
  - t_m: maximum stage delay
  - k: number of stages in the instruction pipeline
  - d: time delay of a latch, needed to advance signals and data from one stage to the next

- Cycle time

$$t = \max[t_i] + d = t_m + d$$

Pipeline Performance (cont.)

- Total time required for a pipeline with k stages to execute n instructions

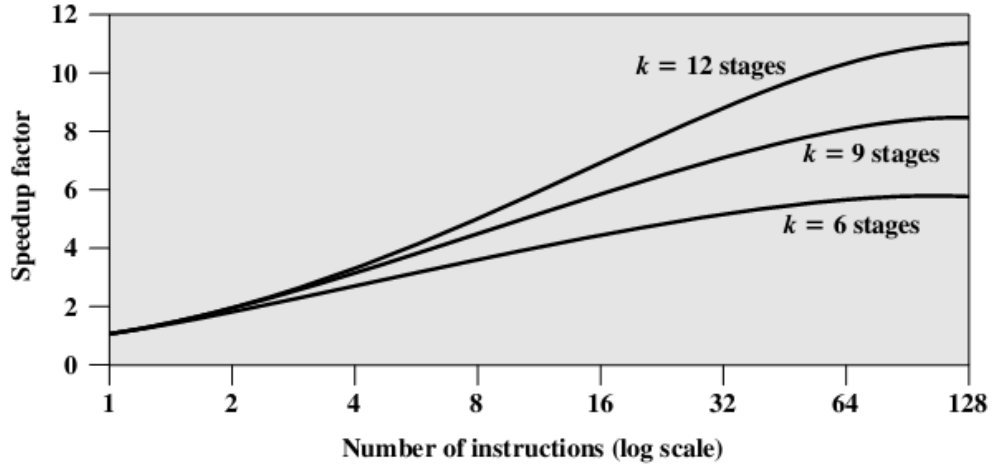
$$T_{k,n} = [k + (n-1)]t$$

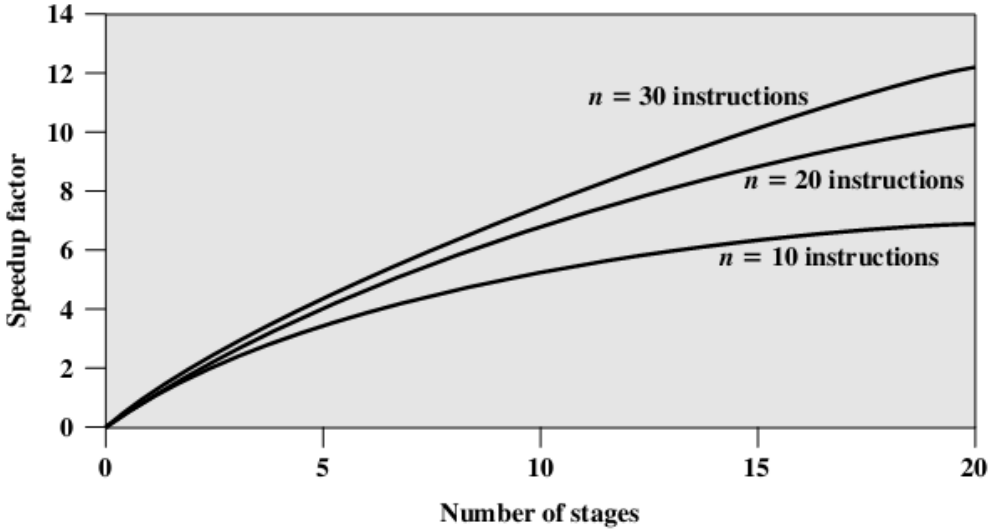
- Speedup factor

$$S_k = \frac{T_{1,n}}{T_{k,n}} = \frac{nkt}{[k + (n-1)]t} = \frac{nk}{k + (n-1)} = \frac{n}{1 + \frac{n-1}{k}}$$

[缪晓伟，121250101]

Pipeline Performance (cont.)





Pipeline Performance (cont.)

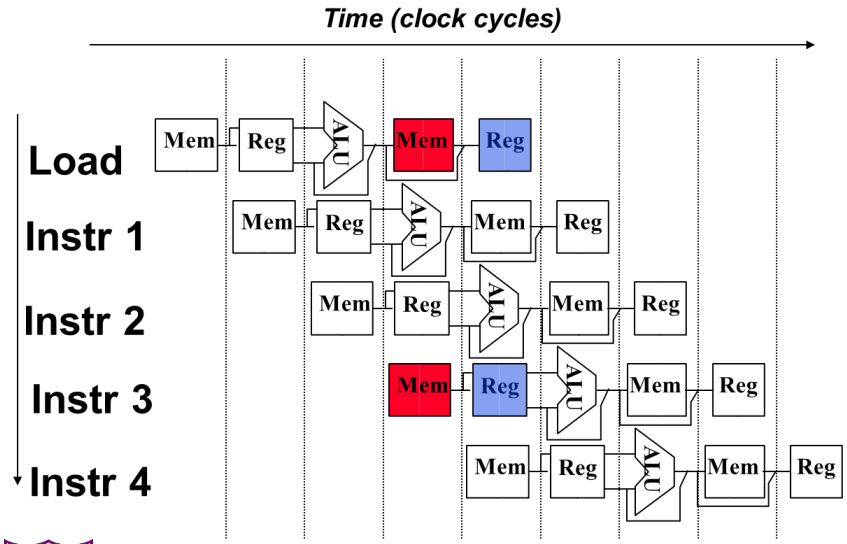
- Misunderstanding
  - The greater the number of stages in the pipeline, the faster the execution rate
- Reason
  - At each stage of the pipeline, there is some overhead involved in moving data from buffer to buffer and in performing various preparation and delivery functions
  - The amount of control logic required to handle memory and register dependencies and to optimize the use of pipeline increases enormously with the number of stages

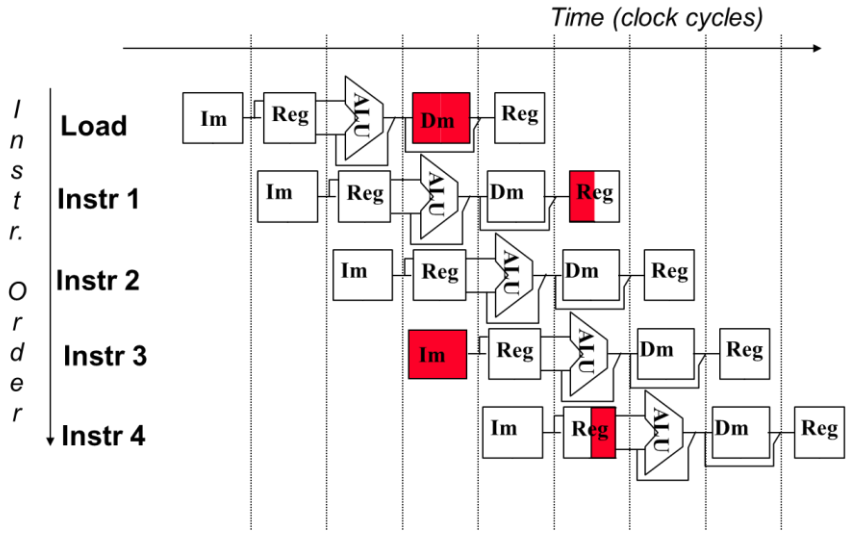
Hazard

- In some cases, instruction pipeline will be blocked or stalled the subsequent instructions cannot be correctly executed
- Type
  - Structure hazard / hardware resource conflict
  - Data hazard / data dependency
  - Control hazard

Structure Hazard

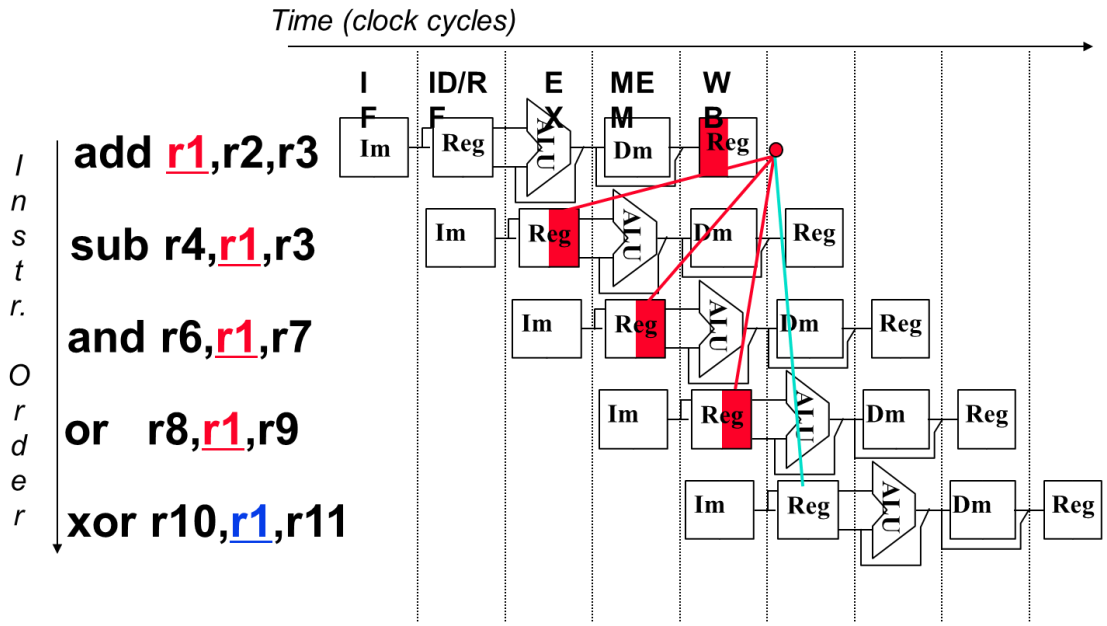
- Reason
  - The same device is accessed by different instructions
- Solution
  - A device can be accessed once in one instruction, and use multiple different devices





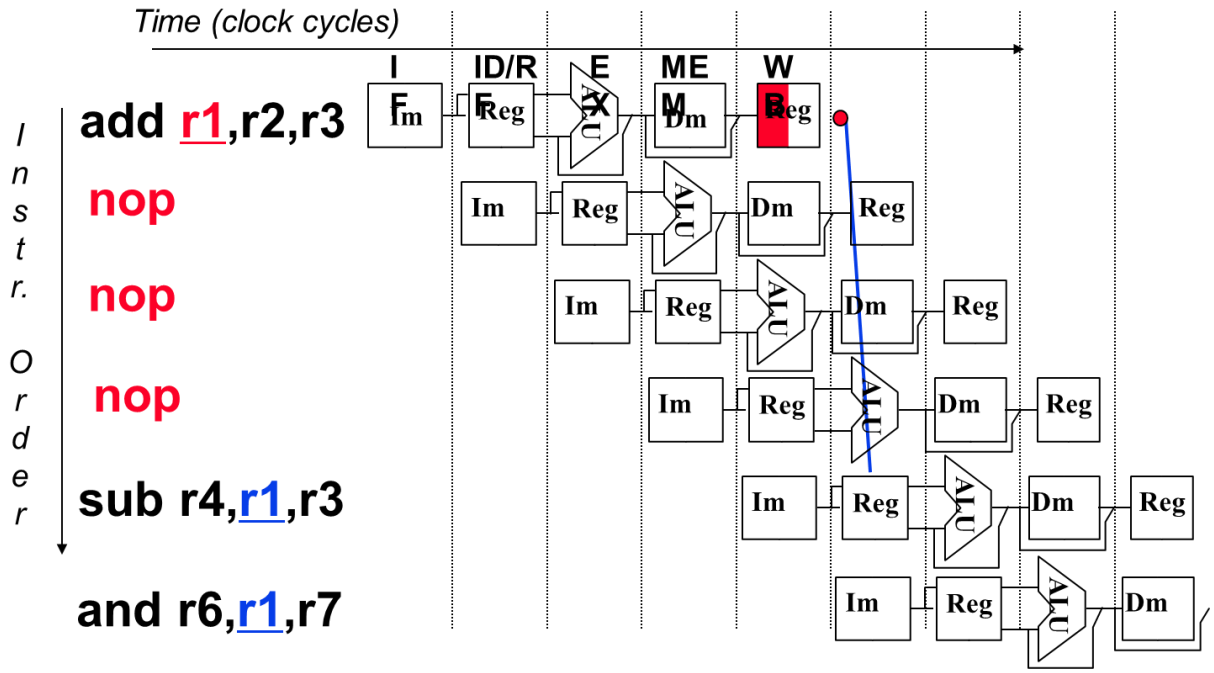
Data Hazard

- Reason
  - The data required by a instruction is not generated



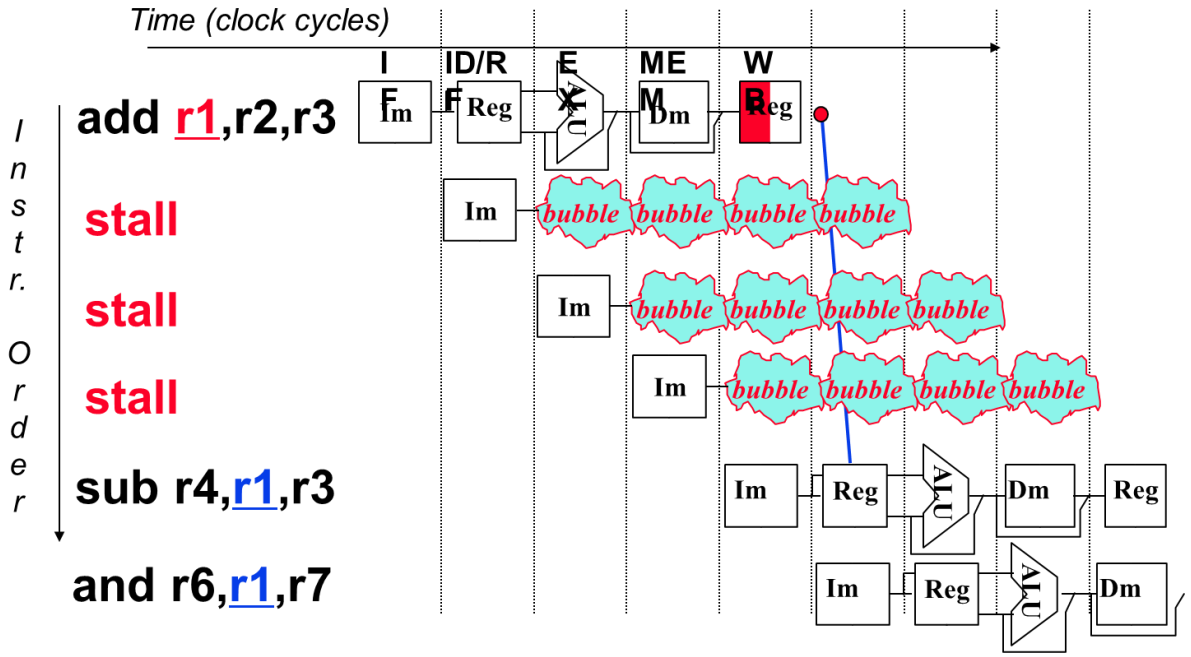
Data Hazard (cont.)

- Solution
  - Insert nop instruction



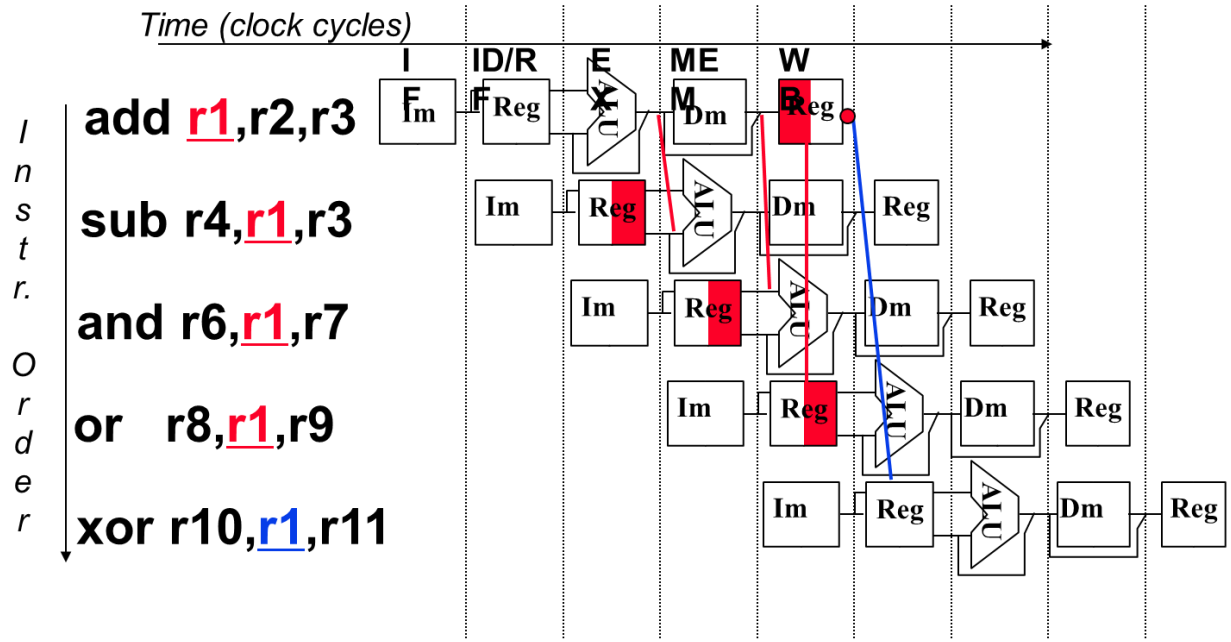
Data Hazard (cont.)

- Solution (cont.)
  - Insert bubble



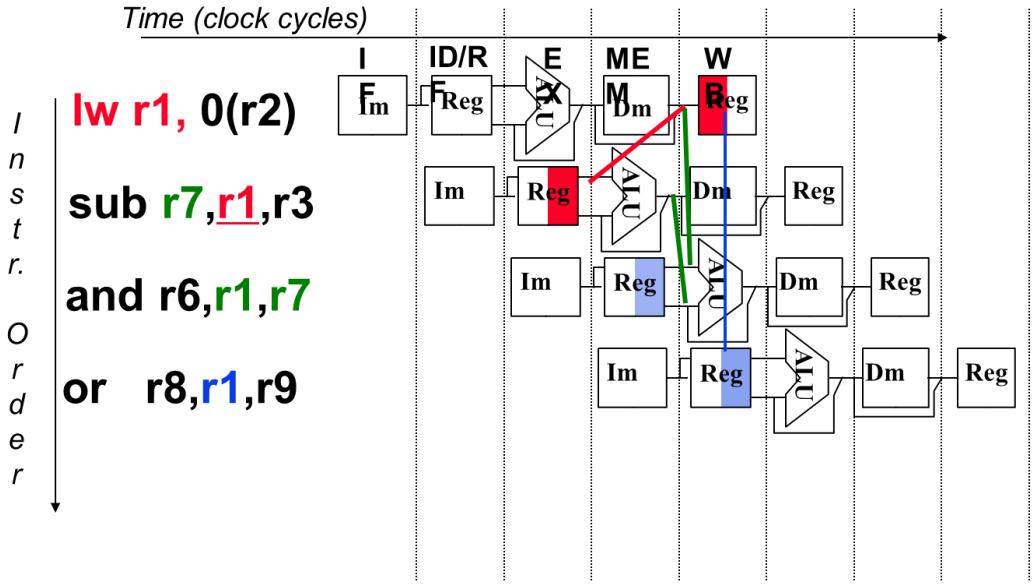
Data Hazard (cont.)

- Solution (cont.)
  - Forwarding / bypassing



Data Hazard (cont.)

- Solution (cont.)
  - Exchange instruction orders



Slow code:

```
lw    $2, b
lw    $3, c
add   $1, $2, $3
sw    a, $1
lw    $5, e
lw    $6, f
sub   $4, $5, $6
sw    d, $4
```

Fast code:

```
lw    $2, b
lw    $3, c
lw    $5, e
add   $1, $2, $3
lw    $6, f
sw    a, $1
sub   $4, $5, $6
sw    d, $4
```

Control Hazard

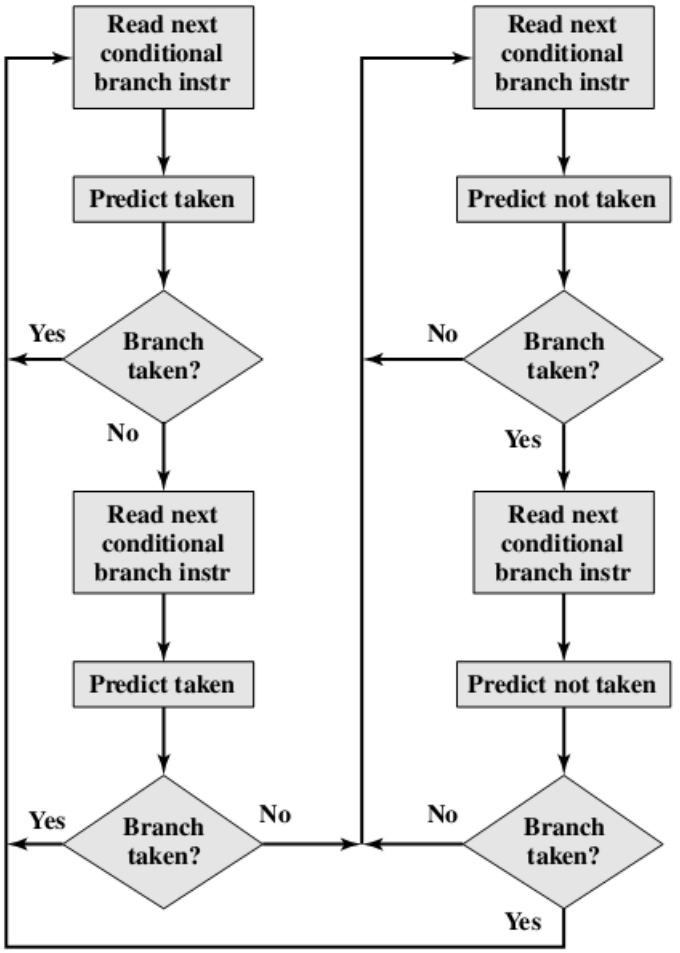
- Reason
  - The order of instruction execution is changed
    - Transfer: branch, loop, …
    - Interrupt
    - Exception
    - Call / return

Control Hazard (cont.)

- Solution
  - Multiple streams: replicate the initial portions of the pipeline and allow the pipeline to fetch both instructions, making use of two streams
  - Prefetch branch target: When a conditional branch is recognized, the target of the branch is prefetched, in addition to the instruction following the branch
  - Loop buffer: use a small, very-high-speed memory maintained by the instruction fetch stage of the pipeline and containing the n most recently fetched instructions, in sequence
  - Delayed branch: exchange instruction orders

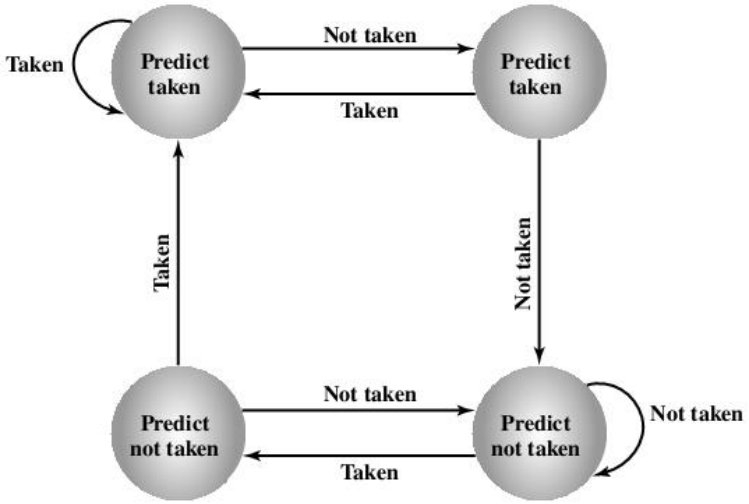
Control Hazard (cont.)

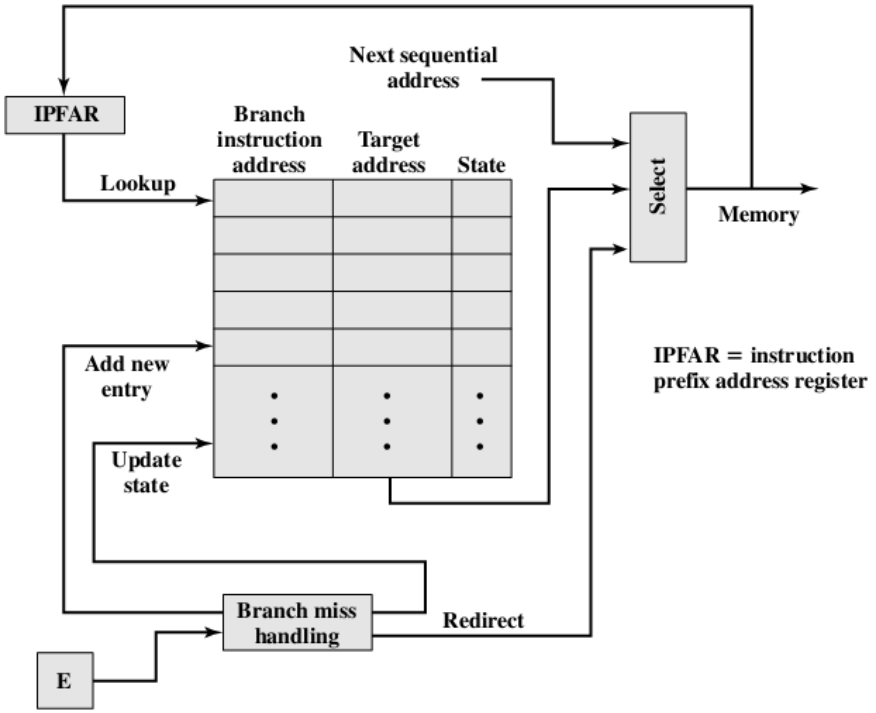
- Solution (cont.)
  - Branch prediction
    - Predict never taken
    - Predict always taken
    - Predict by opcode



Control Hazard (cont.)

- Solution (cont.)
  - Branch prediction
    - Taken/not taken switch
    - Branch history table





Summary

- CPU structure
- Register
  - General purpose, control and status
- Indirect cycle
- Data flow
- Pipeline
  - Two stages, six stages, performance, hazard (structure hazard, data hazard, control hazard)

Thank You

rentw@nju.edu.en

南京大學
NANJING UNIVERSITY