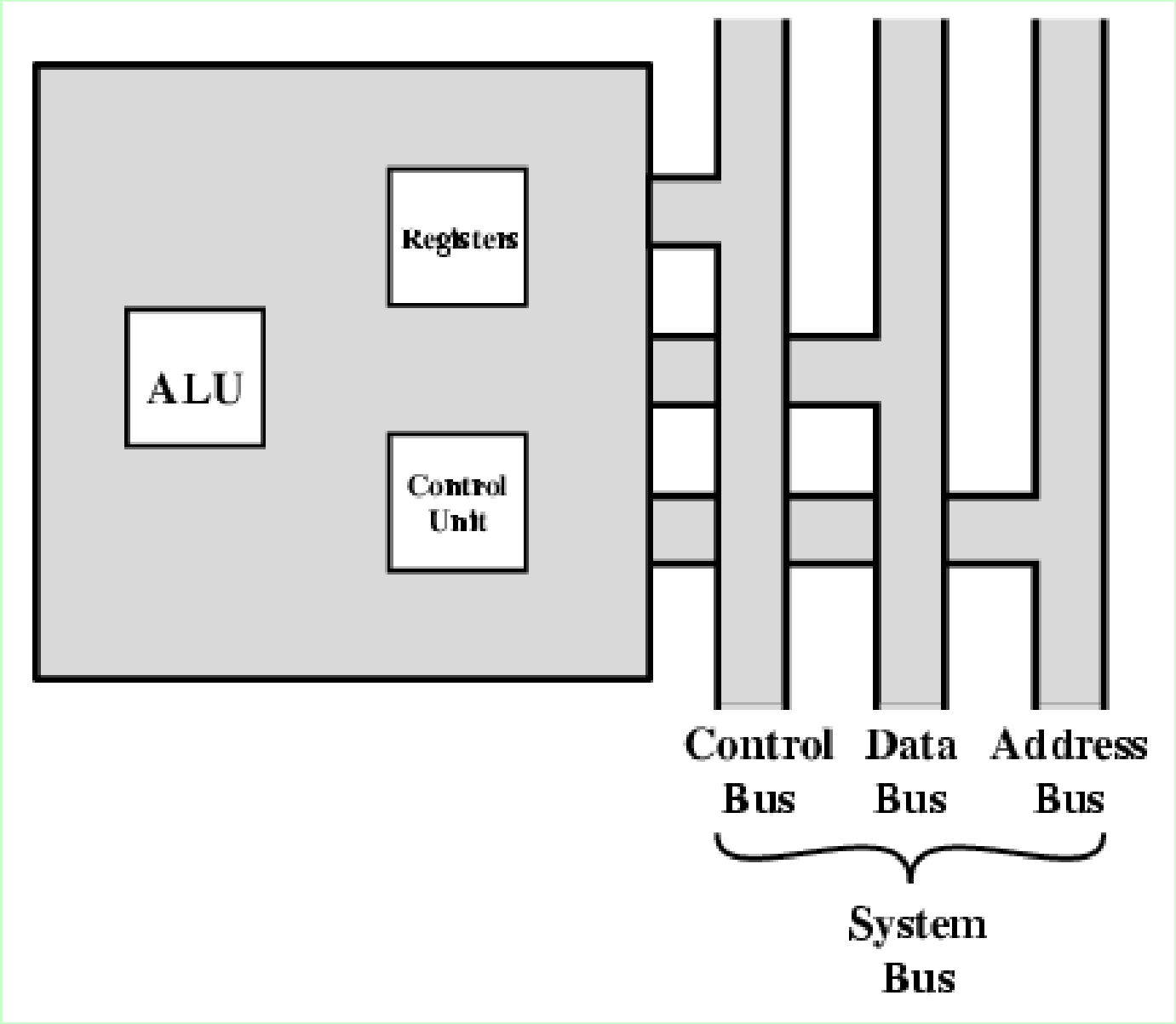
William Stallings
Computer Organization and Architecture
8th Edition

Chapter 12

Advanced Processor Structure and Function

CPU With Systems Bus



CPU Working cycle

- CPU must:
  - Fetch instructions
  - Interpret instructions
  - Fetch data
  - Process data
  - Write data

Prefetch

- Fetch accessing main memory
- Execution usually does not access main memory
- Can fetch next instruction during execution of current instruction
- Called instruction prefetch

Improved Performance

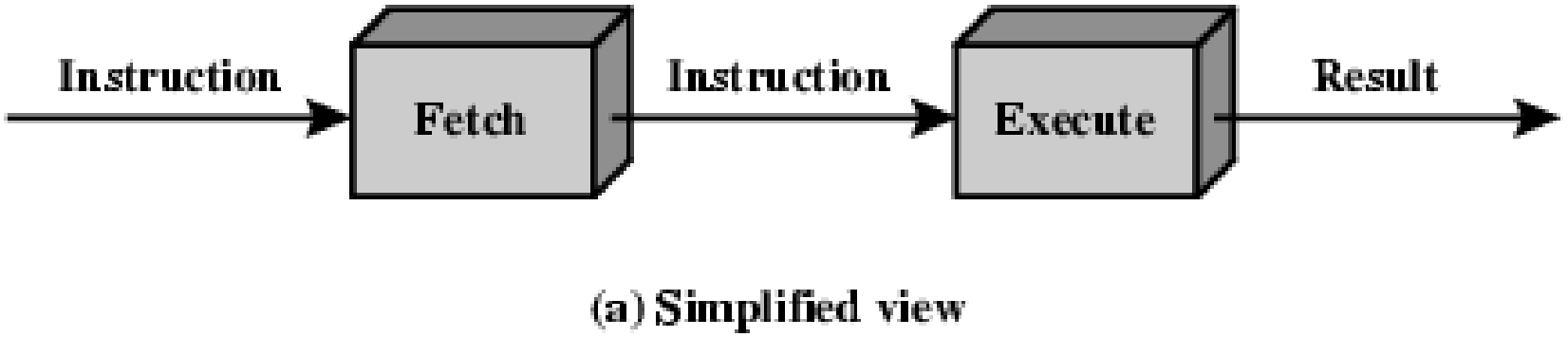
- But not doubled:
  - —Fetch usually shorter than execution
    - – Prefetch more than one instruction?
  - —Any jump or branch means that prefetched instructions are not the required instructions
- Add more stages to improve performance

Pipelining

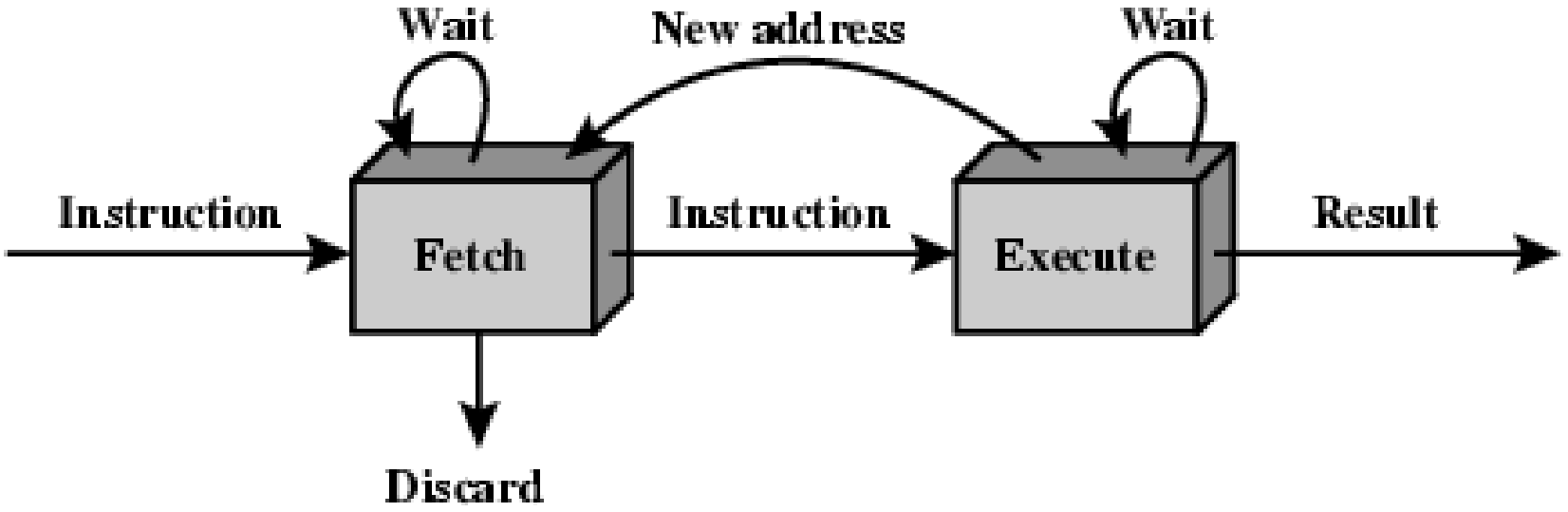
- Fetch instruction
- Decode instruction
- Calculate operands (i.e. EAs)
- Fetch operands
- Execute instructions
- Write result

- Overlap these operations

Two Stage Instruction Pipeline

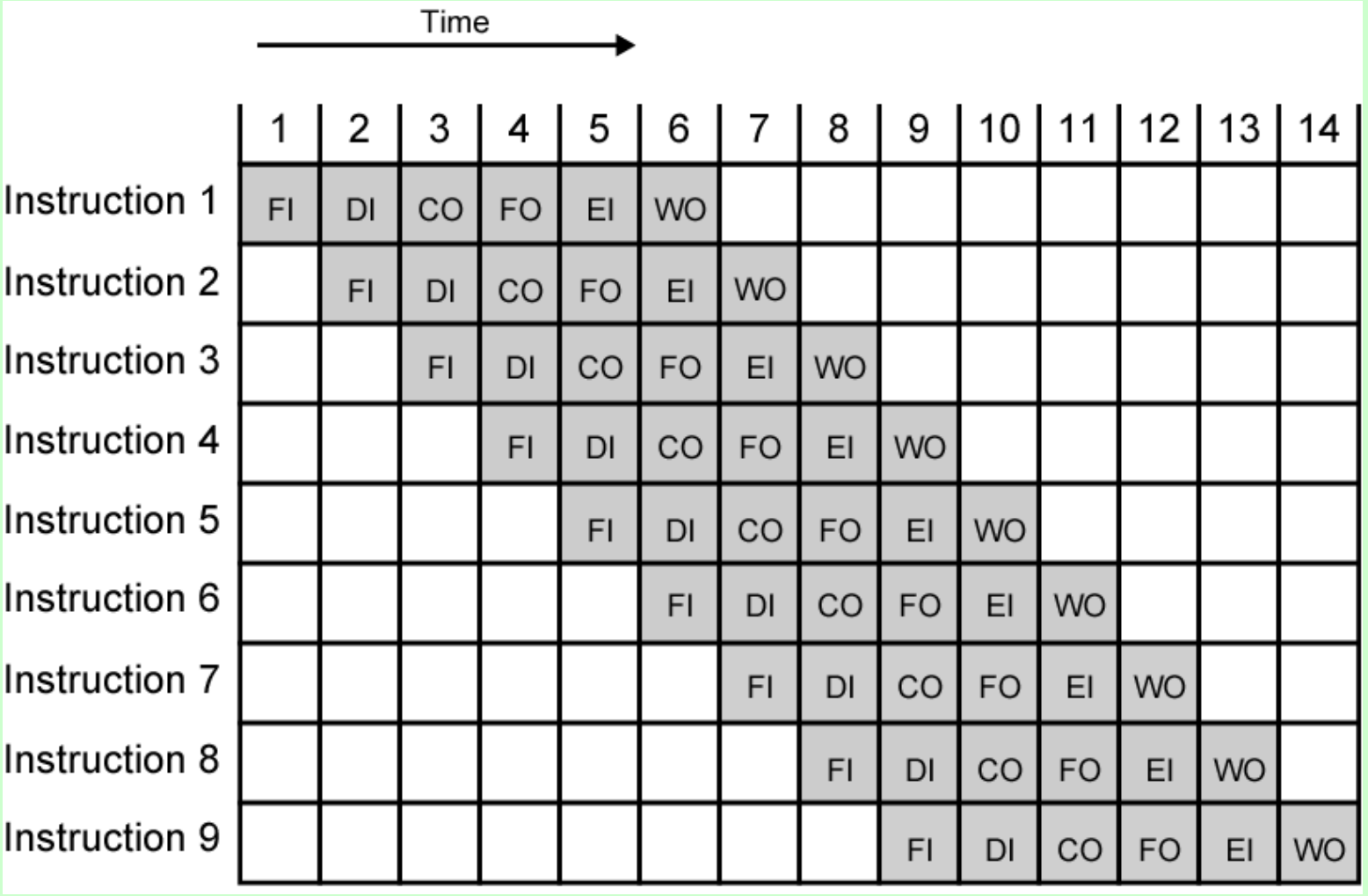


(a) Simplified view

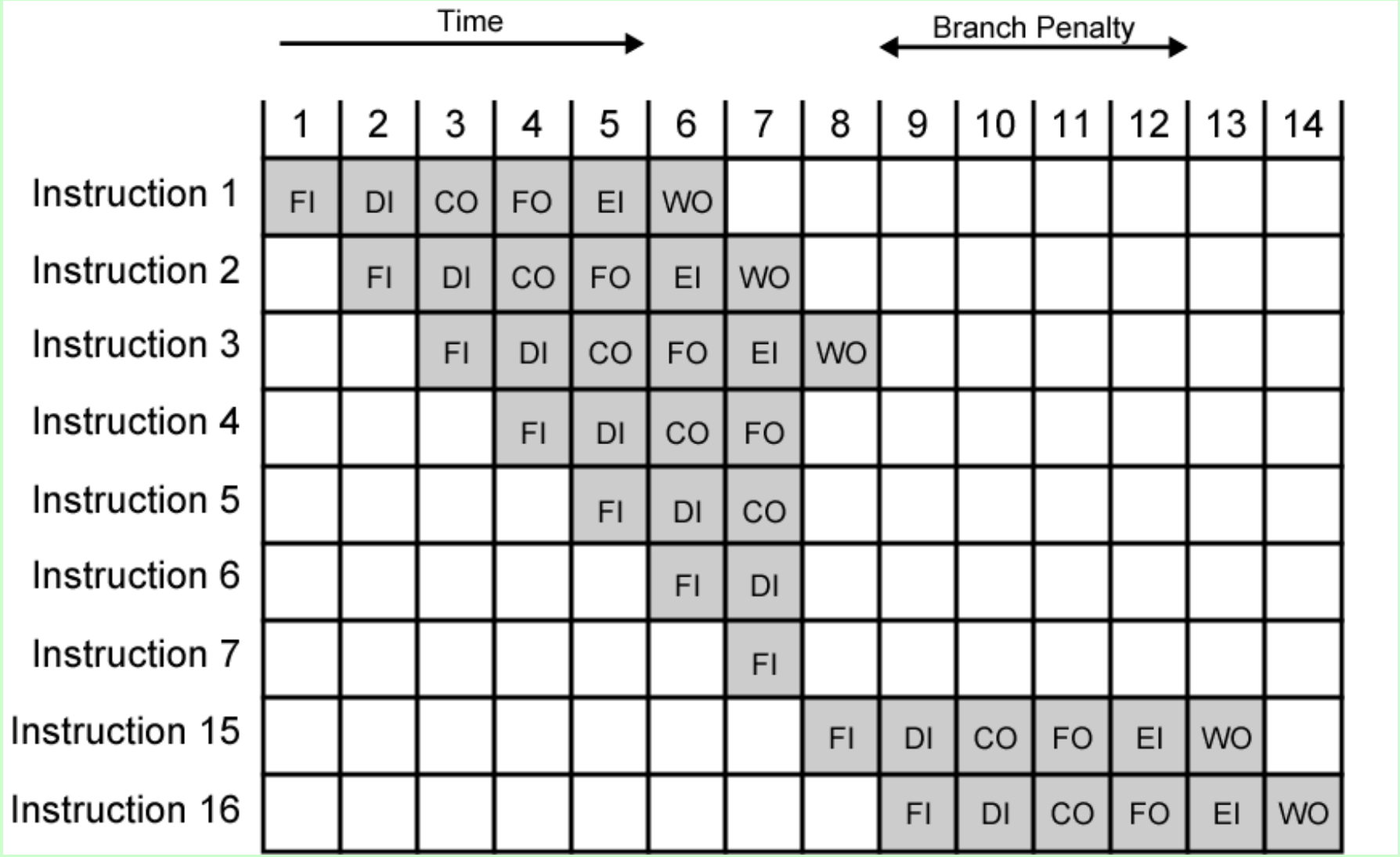


(b) Expanded view

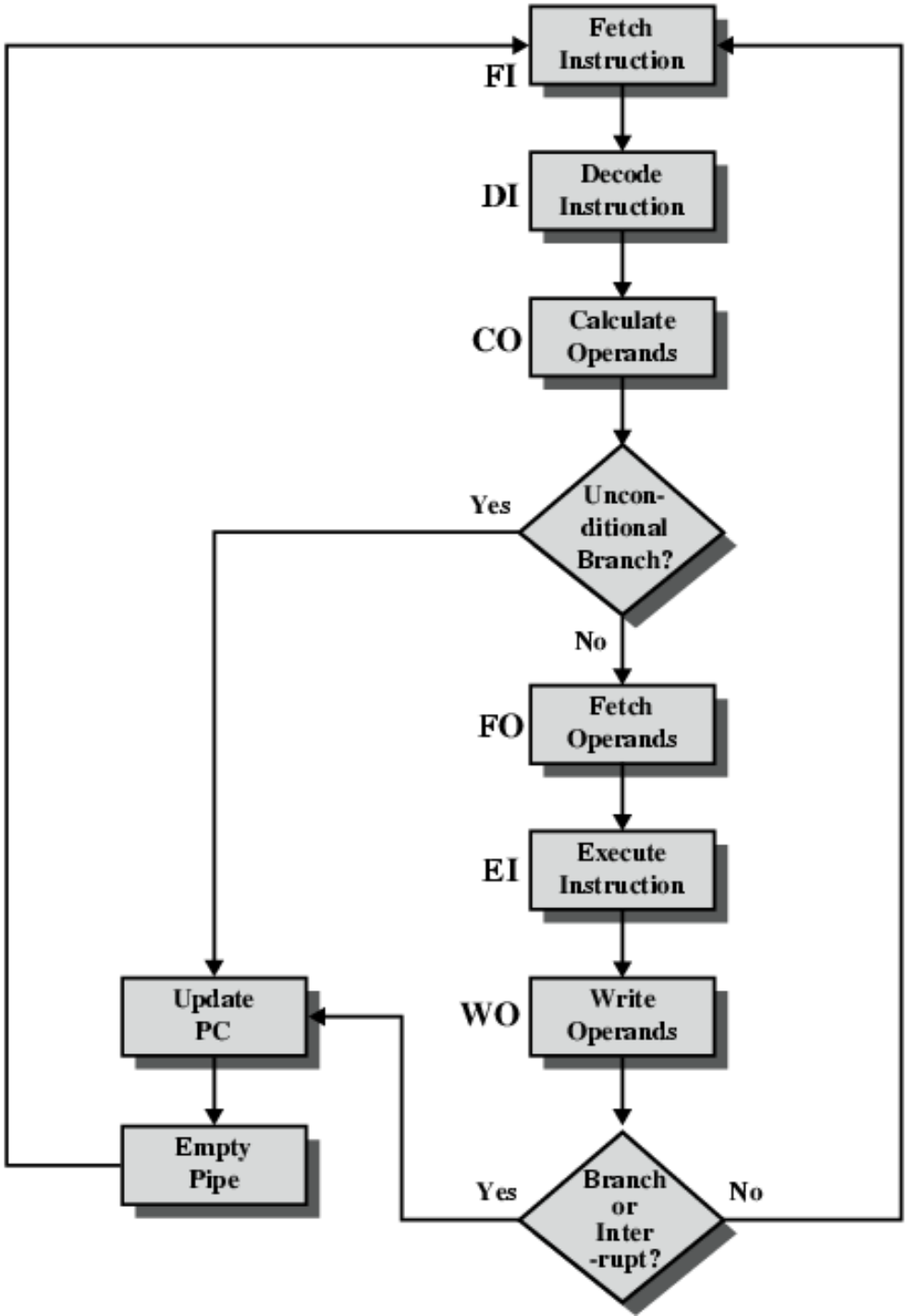
Timing Diagram for Instruction Pipeline Operation



The Effect of a Conditional Branch on Instruction Pipeline Operation



Six Stage Instruction Pipeline



Pipeline Hazards

- Pipeline, or some portion of pipeline, must stall
- Also called *pipeline bubble*
- Types of hazards
  - Resource
  - Data
  - Control

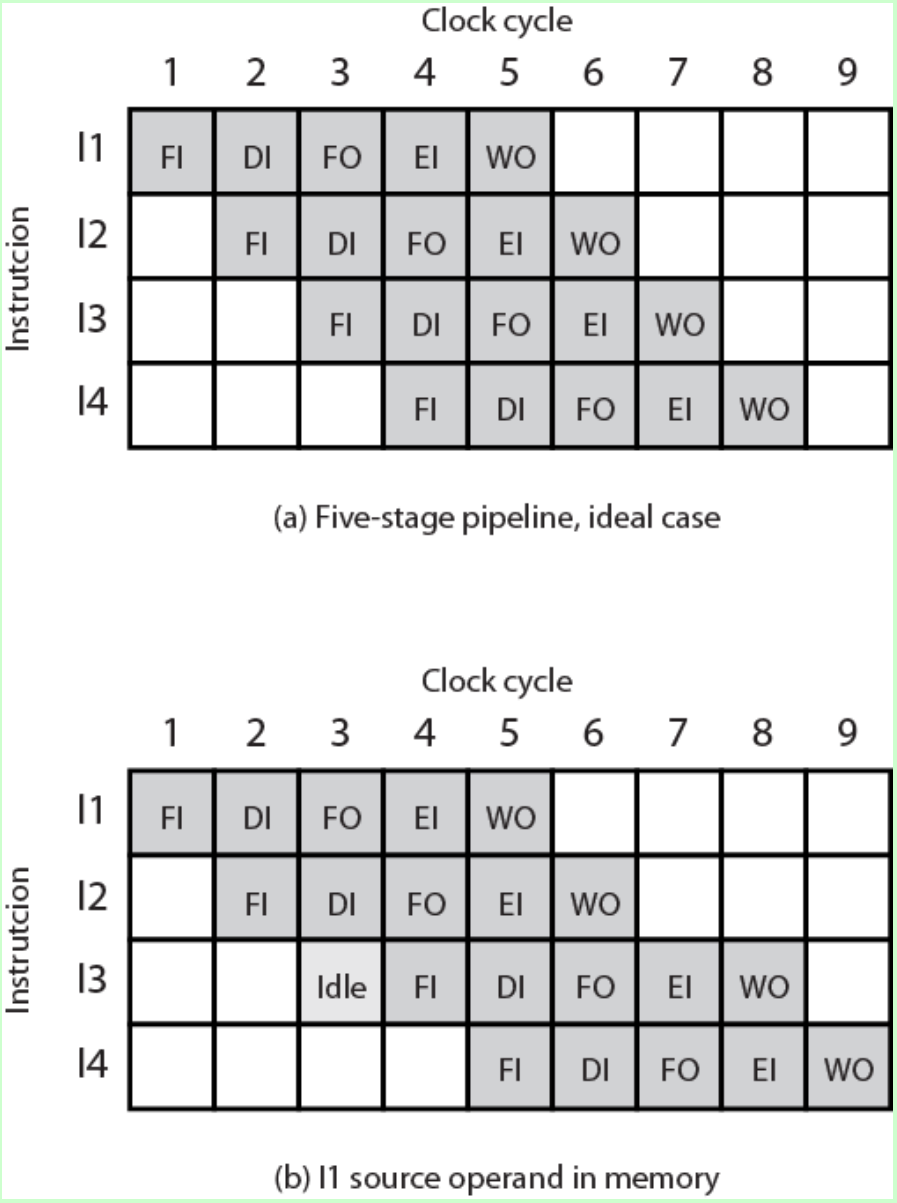
Resource Hazards

- Two (or more) instructions in pipeline need same resource
- Executed in serial rather than parallel for part of pipeline
- Also called *structural hazard*
- E.g. Assume simplified five-stage pipeline
  - Each stage takes one clock cycle
- Ideal case is new instruction enters pipeline each clock cycle
- Assume main memory has single port
- Assume instruction fetches and data reads and writes performed one at a time
- Ignore the cache
- Operand read or write cannot be performed in parallel with instruction fetch
- Fetch instruction stage must idle for one cycle fetching I3

- E.g. multiple instructions ready to enter execute instruction phase
- Single ALU

- One solution: increase available resources
  - Multiple main memory ports
  - Multiple ALUs

Resource Hazard Diagram



Fo in 11 is F1 in l3
Are conflicts
All fetch together

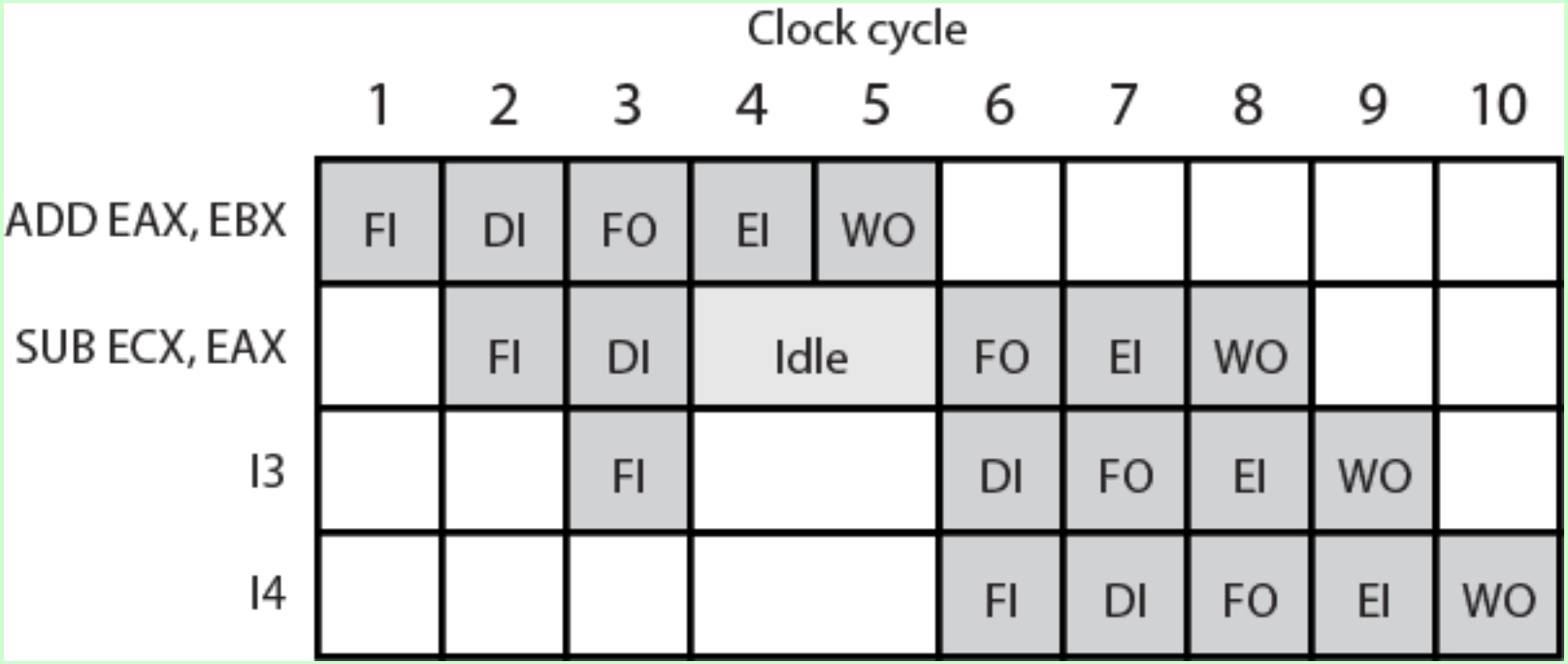
Data Hazards

- Conflict in access of an operand location
- Two instructions to be executed in sequence
- Both access a particular memory or register operand
- If in strict sequence, no problem occurs
- If in a pipeline, operand value could be updated so as to produce different result from strict sequential execution
- E.g. x86 machine instruction sequence:

- ADD EAX, EBX /* EAX = EAX + EBX
- SUB ECX, EAX /* ECX = ECX – EAX

- ADD instruction does not update EAX until end of stage 5, at clock cycle 5
- SUB instruction needs value at beginning of its stage 2, at clock cycle 4
- Pipeline must stall for two clocks cycles
- Without special hardware and specific avoidance algorithms, results in inefficient pipeline usage

Data Hazard Diagram



Types of Data Hazard

- Read after write (RAW), or true dependency
  - An instruction modifies a register or memory location
  - Succeeding instruction reads data in that location
  - Hazard if read takes place before write complete
- Write after read (RAW), or antidependency
  - An instruction reads a register or memory location
  - Succeeding instruction writes to location
  - Hazard if write completes before read takes place
- Write after write (RAW), or output dependency
  - Two instructions both write to same location
  - Hazard if writes take place in reverse of order intended sequence
- Previous example is RAW hazard
- See also Chapter 14

Control Hazard

- Also known as *branch hazard*
- Pipeline makes wrong decision on branch prediction
- Brings instructions into pipeline that must subsequently be discarded
- Dealing with Branches
  - Multiple Streams
  - Prefetch Branch Target
  - Loop buffer
  - Branch prediction
  - Delayed branching

Multiple Streams

- Have two pipelines(多个流水)
- Prefetch each branch into a separate pipeline
- Use appropriate pipeline

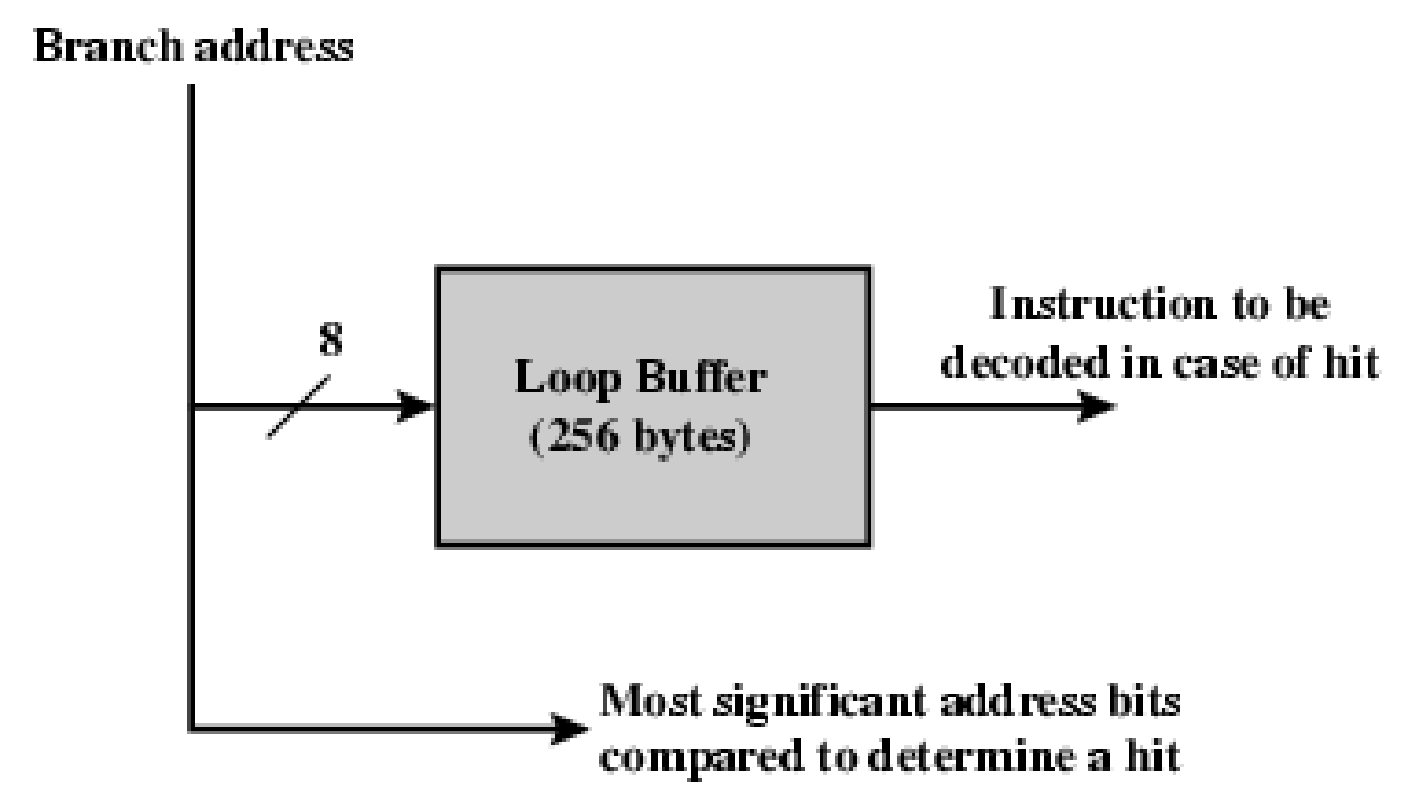
Prefetch Branch Target

- Target of branch is prefetched in addition to instructions following branch
- Keep target until branch is executed
- Used by IBM 360/91
- 提前取出分支指令

Loop Buffer

- Very fast memory
- Maintained by fetch stage of pipeline
- Check buffer before fetching from memory
- Very good for small loops or jumps
- c.f. cache
- Used by CRAY-1
- 高速缓冲存一组指令
- 若一个转移将要发生，硬件首先检查转移目标是否在此缓冲器种。若是，则下一条指令由此缓冲器取得。

Loop Buffer Diagram



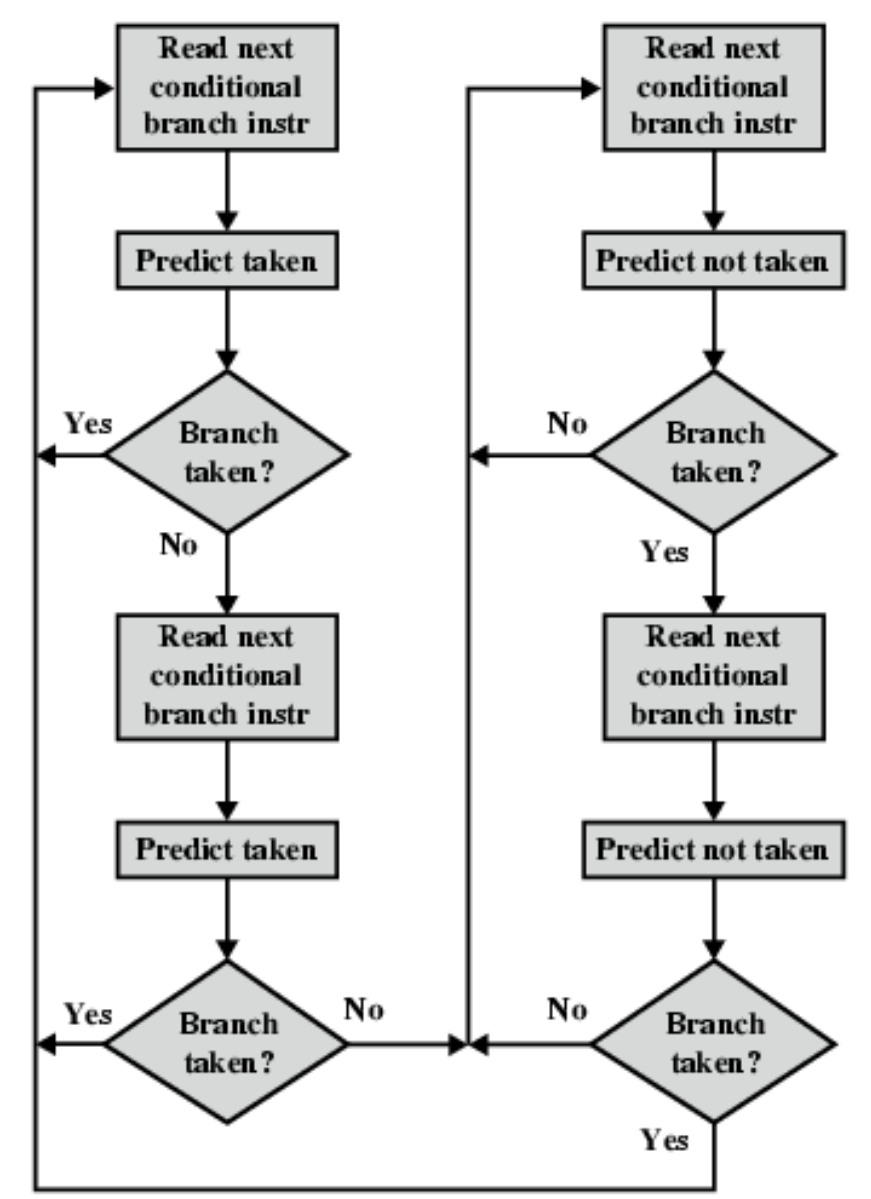
Branch Prediction (1)

- Predict never taken
  - —Assume that jump will not happen
  - —Always fetch next instruction
  - —68020 & VAX 11/780
  - —VAX will not prefetch after branch if a page fault would result (O/S v CPU design)
- Predict always taken
  - —Assume that jump will happen
  - —Always fetch target instruction

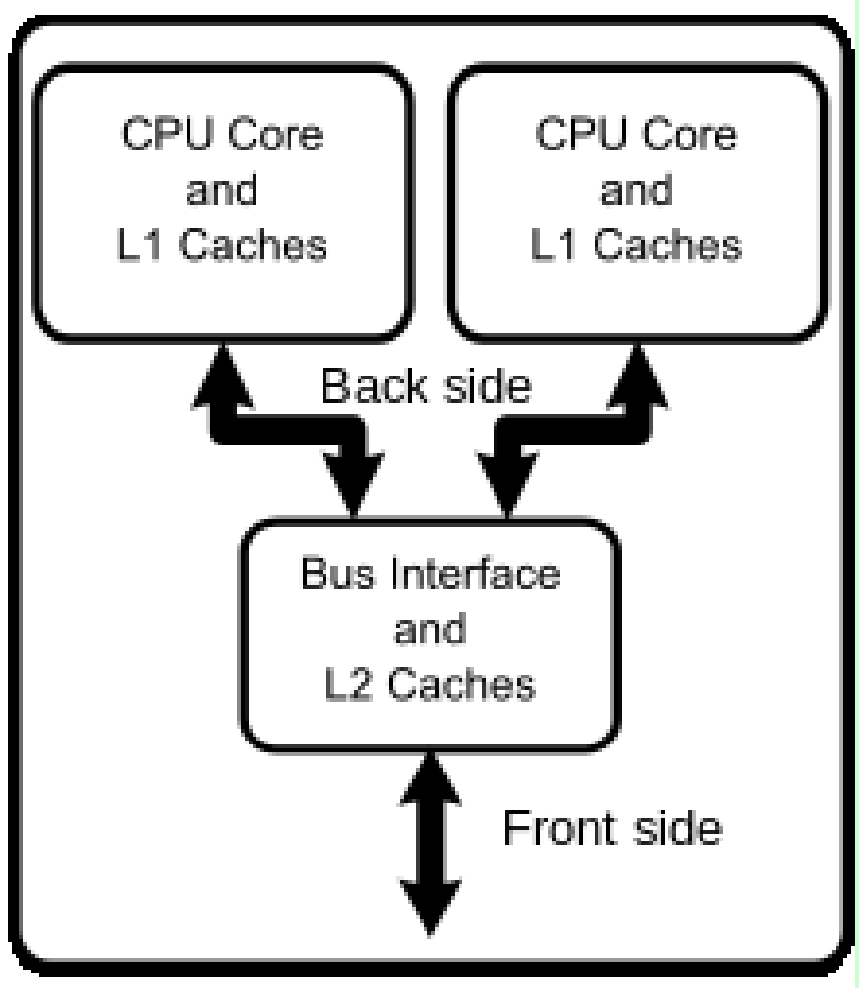
Branch Prediction (3)

- Delayed Branch
  - —Do not take jump until you have to
  - —Rearrange instructions

Branch Prediction Flowchart



Multi-cores


CPU Core
and
L1 Caches
CPU Core
and
L1 Caches
Back side
Bus Interface
and
L2 Caches
Front side