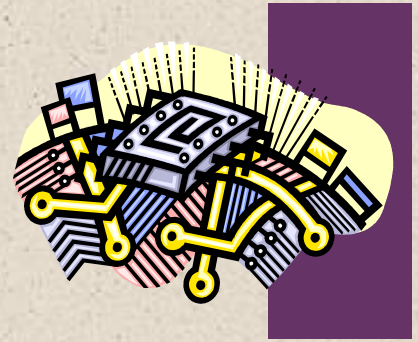
Unit 6

Processor Organization

Content..

- Register organization- User visible registers, Control and Status registers, Instruction Cycle-Indirect cycle and Data flow, Instruction Pipelining- Pipelining Strategy, Pipeline performance, Pipeline hazards, Fundamental Concepts- Register transfer, Performing arithmetic or logic operations, Fetching a word from memory, Storing a word in memory.

Processor Organization



Processor Requirements:

- 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
- In order to do these things the processor needs to store some data temporarily and therefore needs a small internal memory

CPU With the System Bus

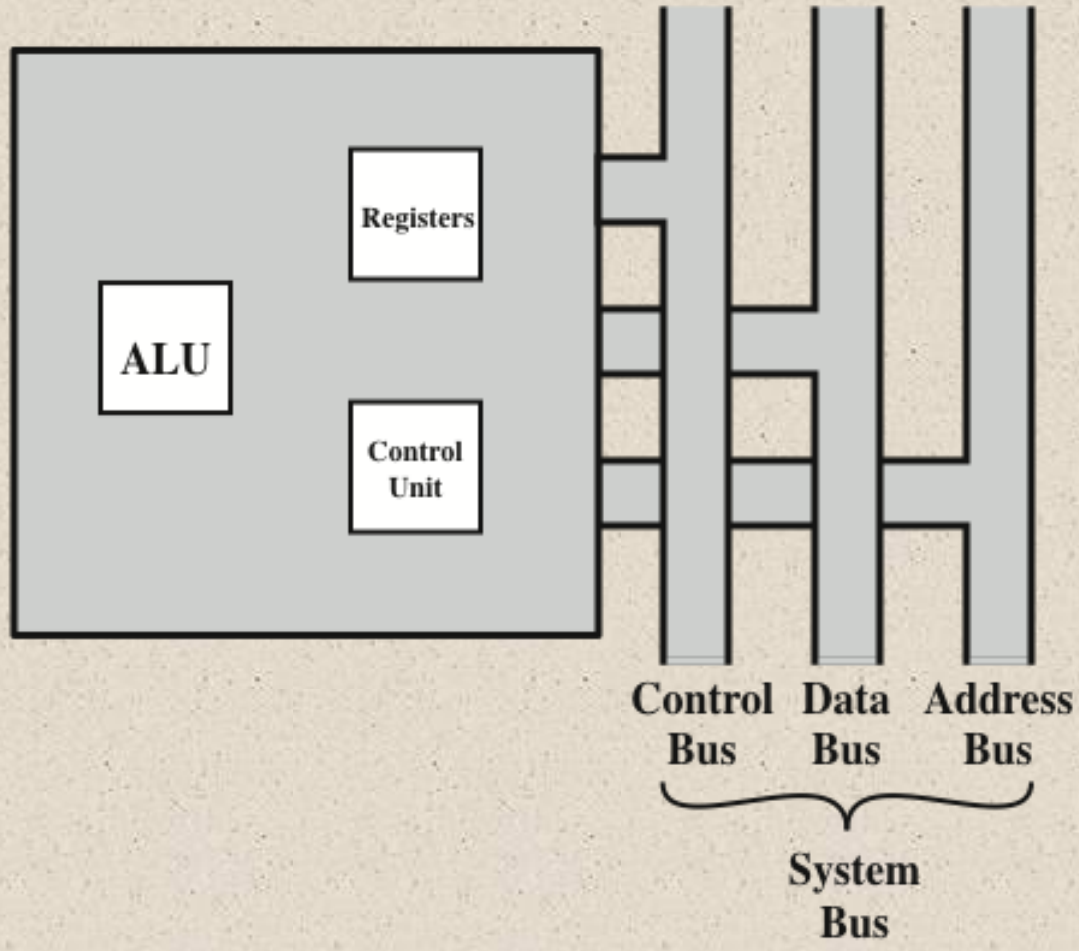


Figure 14.1 The CPU with the System Bus

CPU Internal Structure

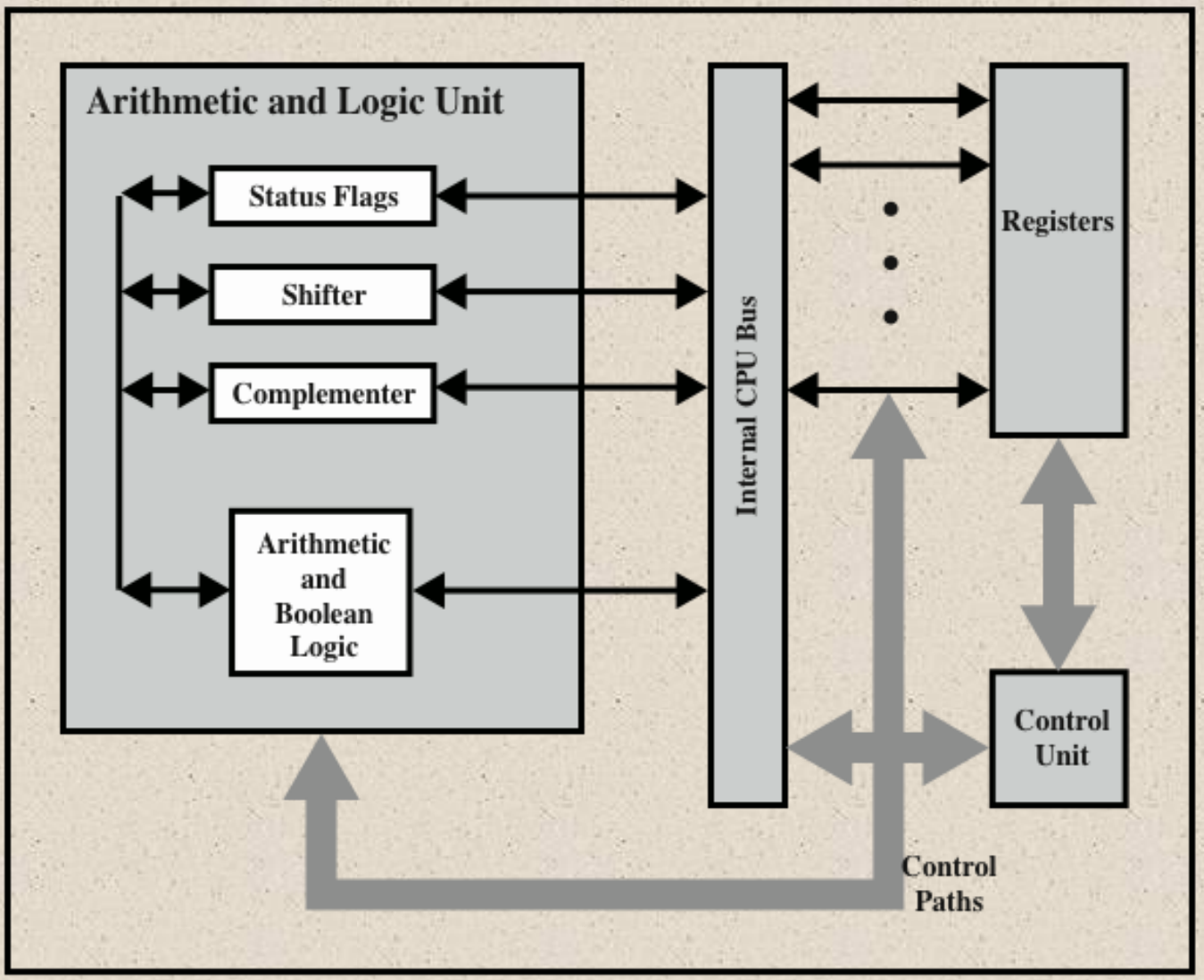


Figure 14.2 Internal Structure of the CPU

Register Organization

- Within the processor there is a set of registers that function as a level of memory above main memory and cache in the hierarchy
- 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 Registers

Referenced by means of the machine language that the processor executes

Categories:

- **General purpose**
  - Can be assigned to a variety of functions by the programmer
- **Data**
  - May be used only to hold data and cannot be employed in the calculation of an operand address
- **Address**
  - May be somewhat general purpose or may be devoted to a particular addressing mode
  - Examples: segment pointers, index registers, stack pointer
- **Condition codes**
  - Also referred to as *flags*
  - Bits set by the processor hardware as the result of operations

Condition Codes

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.	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. Conditional instructions, such as BRANCH are simplified relative to composite instructions, such as TEST AND BRANCH.	2. Condition codes are irregular; they are typically not part of the main data path, so they require extra hardware connections.
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.	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. Condition codes can be saved on the stack during subroutine calls along with other register information.	4. In a pipelined implementation, condition codes require special synchronization to avoid conflicts.

Control and Status Registers

Four registers are essential to instruction execution:

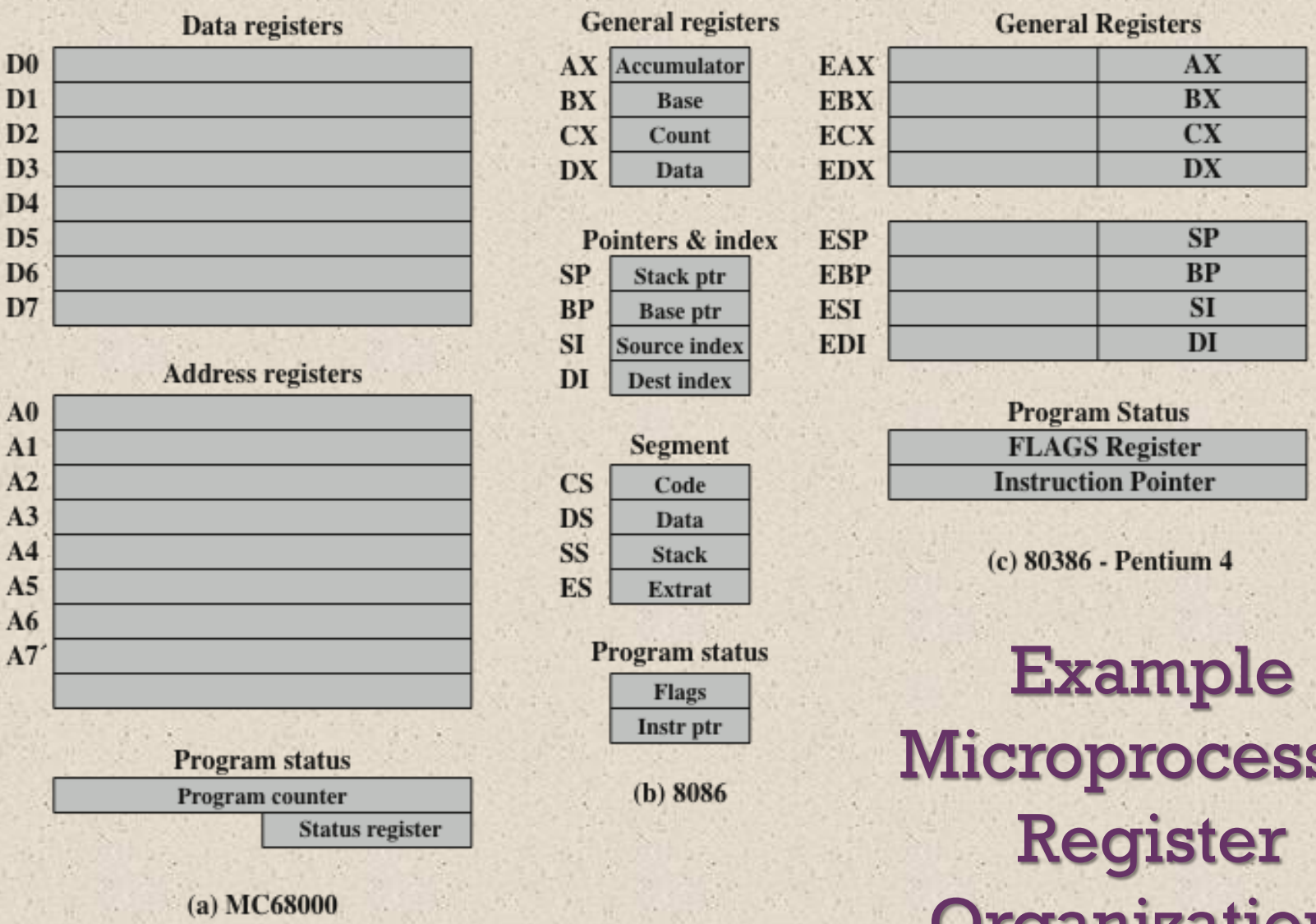
- Program counter (PC)
  - Contains the address of an instruction to be fetched
- Instruction register (IR)
  - Contains the instruction most recently fetched
- Memory address register (MAR)
  - Contains the address of a location in memory
- Memory buffer register (MBR)
  - Contains a word of data to be written to memory or the word most recently read

Program Status Word (PSW)

Register or set of registers that contain status information

Common fields or flags include:

- Sign
- Zero
- Carry
- Equal
- Overflow
- Interrupt Enable/Disable
- Supervisor



Example Microprocessor Register Organizations

Figure 14.3 Example Microprocessor Register Organizations

Instruction Cycle

Includes the following stages:

- Fetch
  - Read the next instruction from memory into the processor
- Execute
  - Interpret the opcode and perform the indicated operation
- Interrupt
  - If interrupts are enabled and an interrupt has occurred, save the current process state and service the interrupt

Indirect Cycle

- May require memory access to fetch operand
- Indirect access require more memory accesses
- Can be thought of as additional instruction subcycle

Instruction Cycle with indirect

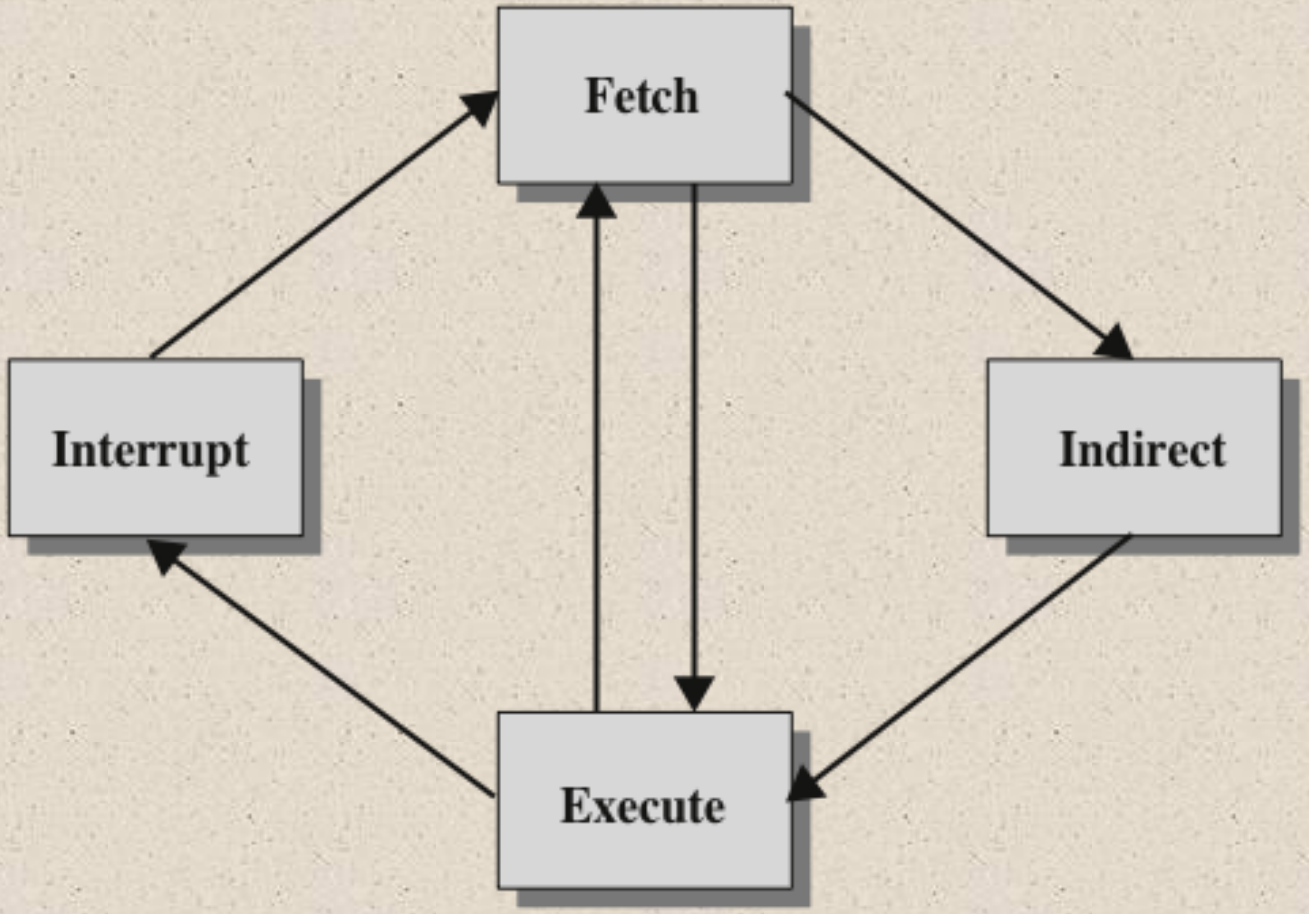


Figure 14.4 The Instruction Cycle

Instruction Cycle State Diagram

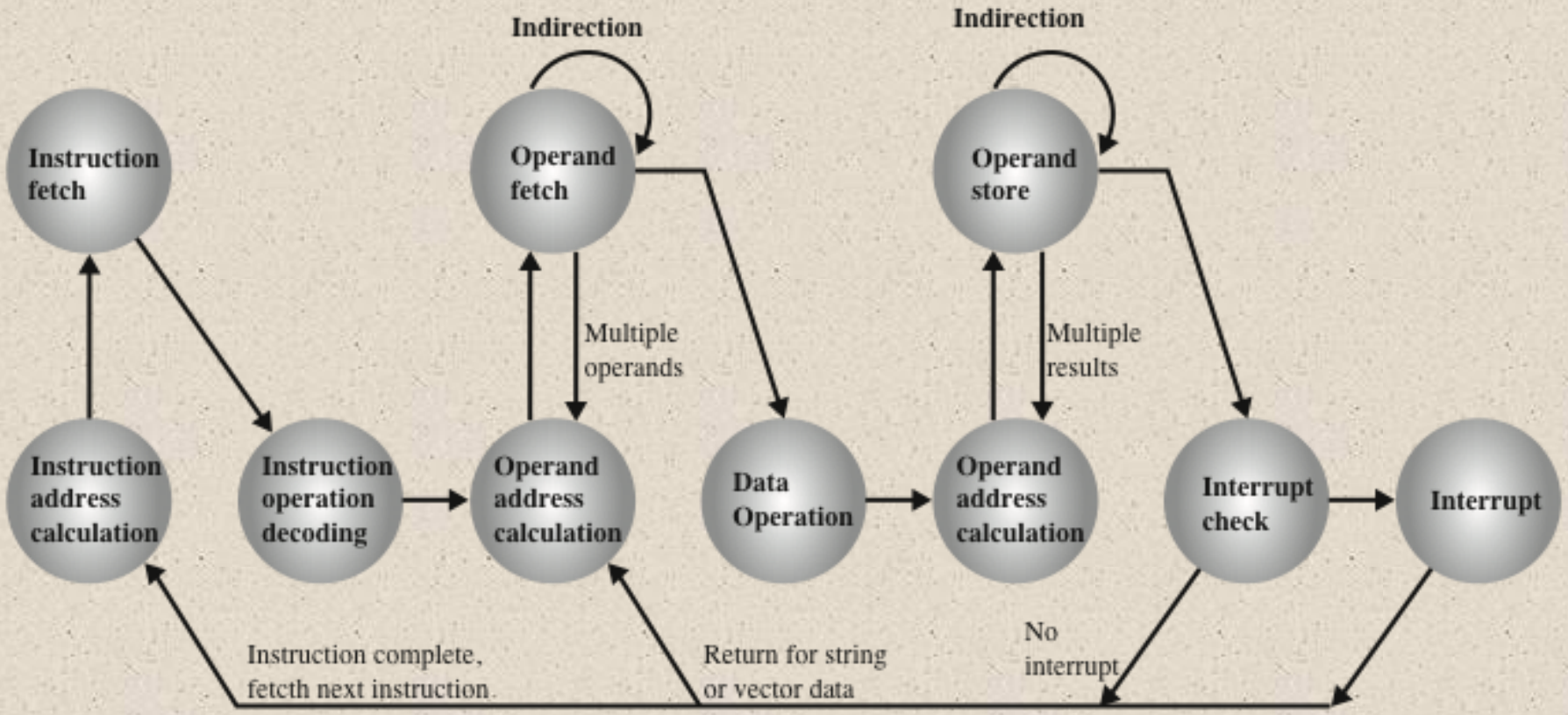
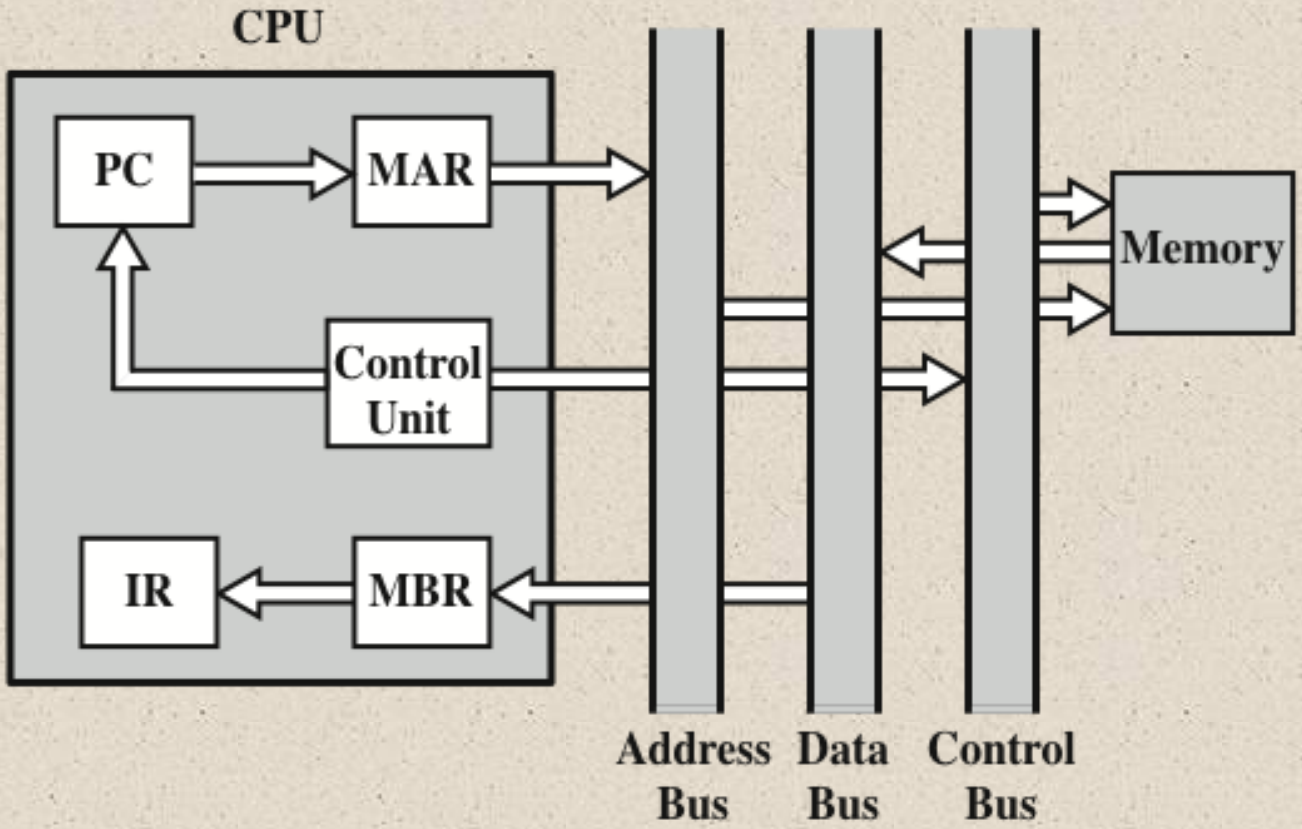


Figure 14.5 Instruction Cycle State Diagram

Data Flow (Instruction Fetch)

- Depends on CPU design
- In general
- Fetch
  - PC contain address of next instruction.
  - Address move to MAR
  - Address placed on address line
  - Control units request memory read
  - Result placed on data bus copied to MBR then to IR
  - Meanwhile PC incremented by 1

Data Flow, Fetch Cycle



MBR = Memory buffer register
MAR = Memory address register
IR = Instruction register
PC = Program counter

Figure 14.6 Data Flow, Fetch Cycle

Data Flow (Data Fetch)

- IR is examined
- If indirect addressing , indirect cycle is performed
  - Right most N bits of MBR transferred to MAR (add of OP)
  - Control units request memory read
  - Result (address of operand) moved to MBR

Data Flow, Indirect Cycle

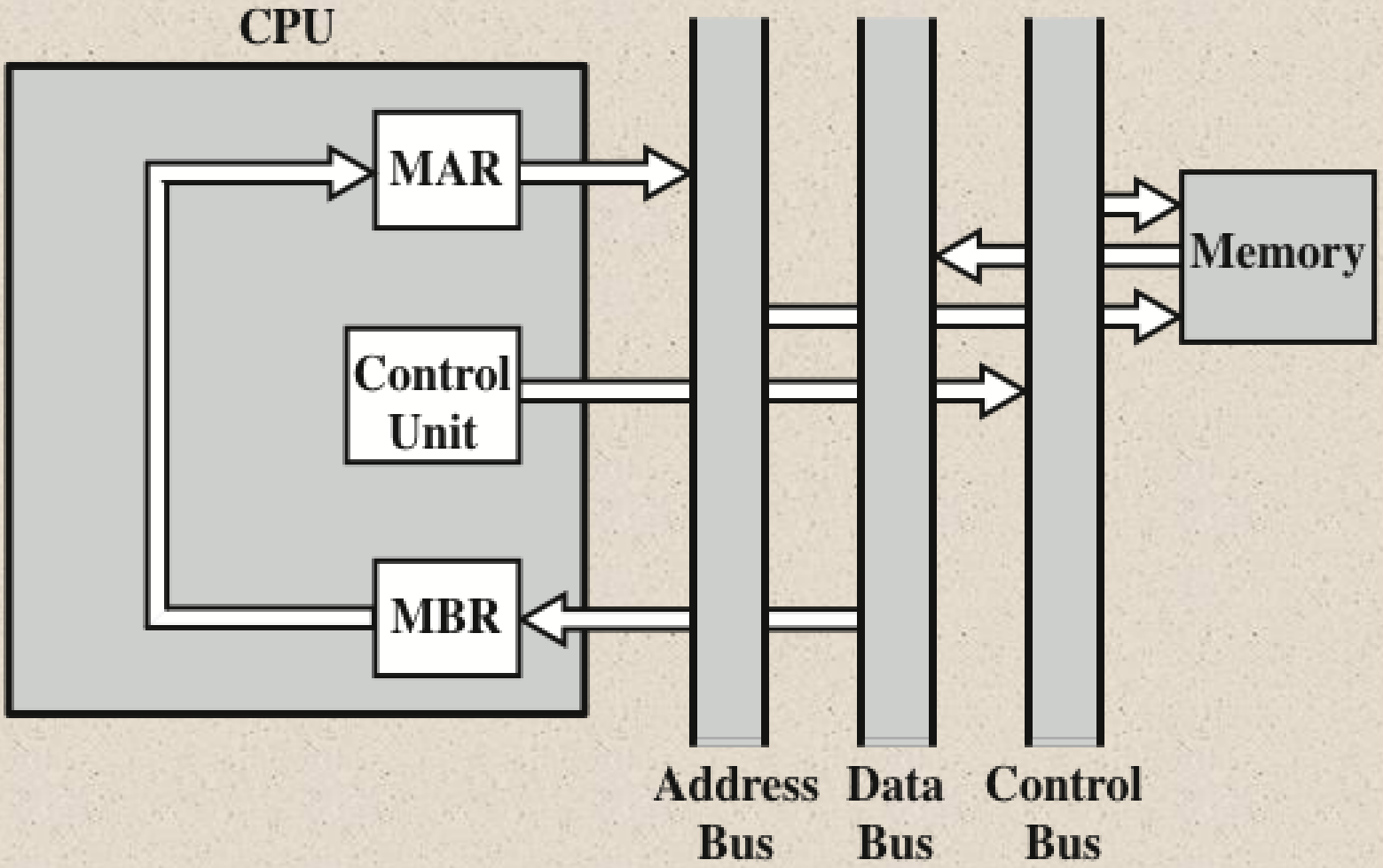


Figure 14.7 Data Flow, Indirect Cycle

Data Flow (Interrupt)

- Simple predictable
- Current PC saved to allow resumption after interrupt
- Content of PC copied to MBR
- Special memory location (e.g. SP) loaded to MAR
- MBR written to memory
- PC loaded with address of interrupt handling Routine
- Next instruction (1st of interrupt handler) can be fetched

Data Flow, Interrupt Cycle

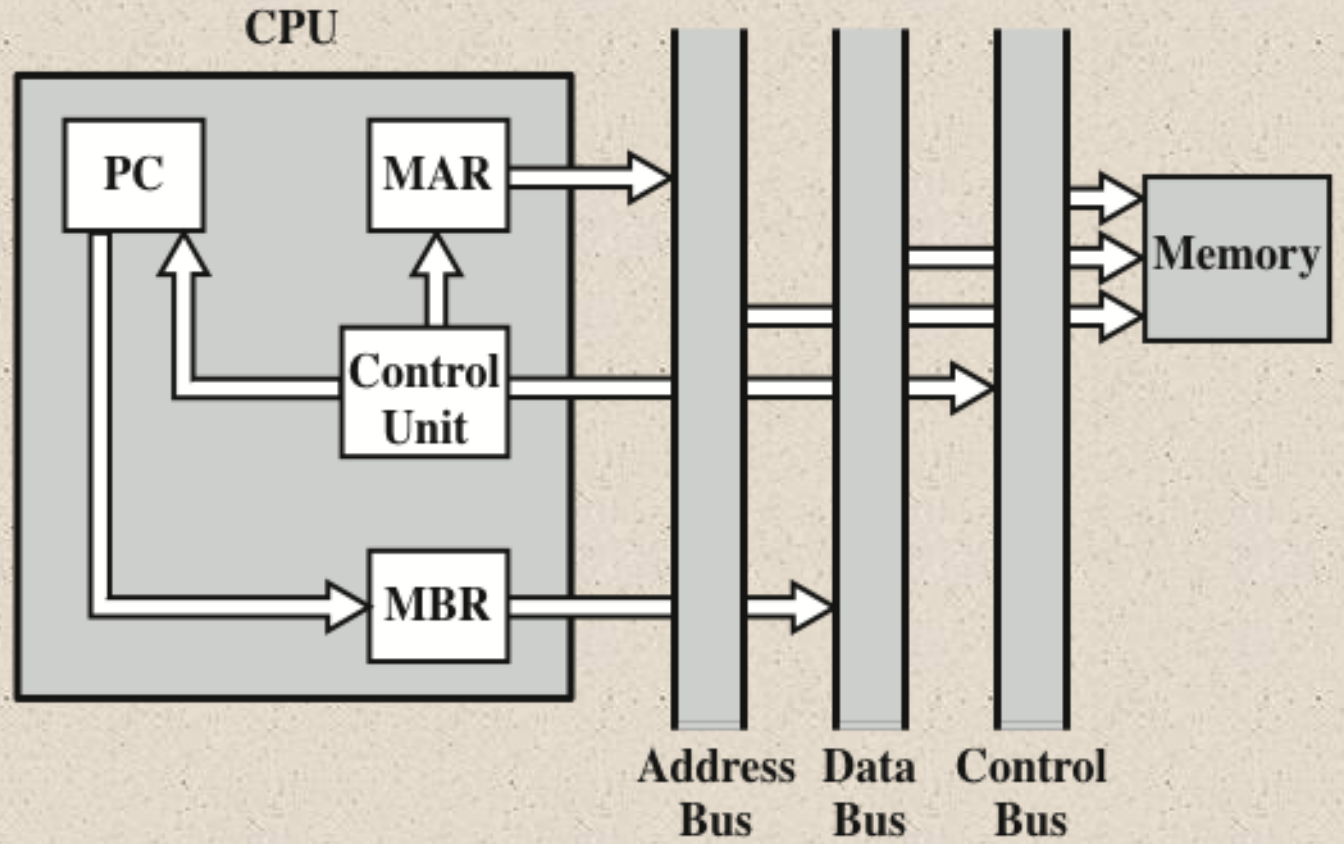


Figure 14.8 Data Flow, Interrupt Cycle

Prefetch

- Fetch accessing main memory
- Execution usually does not access main memory
- Can fetch next instruction during execution of current instruction
- Called instruction prefetch
  - Any Jump or branch means that prefetched instructions are not the required instructions
- Add more stages to improve performace

Pipelining Strategy

Similar to the use of an assembly line in a manufacturing plant

New inputs are accepted at one end before previously accepted inputs appear as outputs at the other end

To apply this concept to instruction execution we must recognize that an instruction has a number of stages

Two-Stage Instruction Pipeline

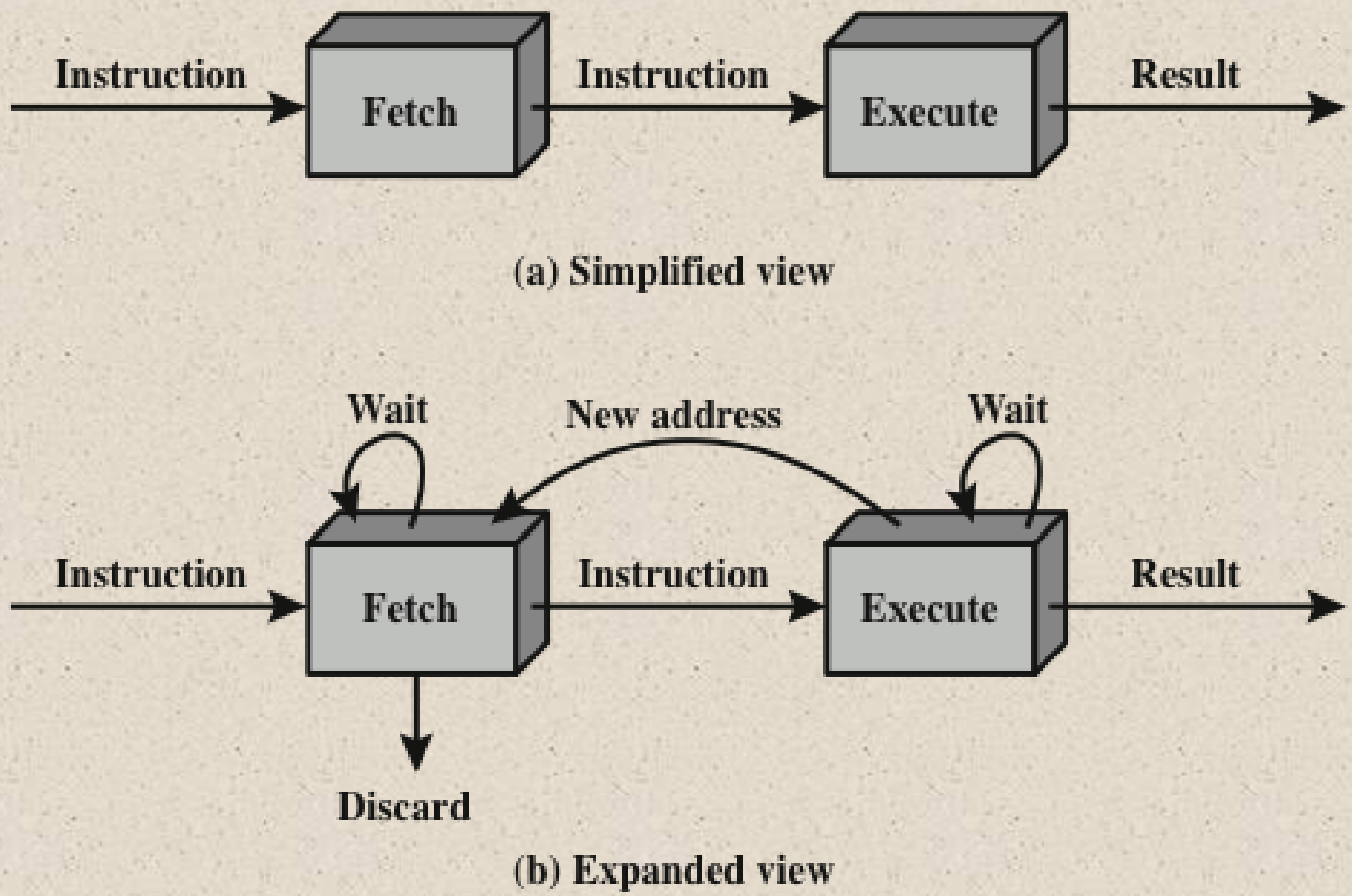


Figure 14.9 Two-Stage Instruction Pipeline

Additional Stages

- Fetch instruction (FI)
  - Read the next expected instruction into a buffer
- Decode instruction (DI)
  - Determine the opcode and the operand specifiers
- Calculate operands (CO)
  - Calculate the effective address of each source operand
  - This may involve displacement, register indirect, indirect, or other forms of address calculation
- 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

Timing Diagram for Instruction Pipeline Operation

Time →

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Instruction 1	FI	DI	CO	FO	EI	WO								
Instruction 2		FI	DI	CO	FO	EI	WO							
Instruction 3			FI	DI	CO	FO	EI	WO						
Instruction 4				FI	DI	CO	FO	EI	WO					
Instruction 5					FI	DI	CO	FO	EI	WO				
Instruction 6						FI	DI	CO	FO	EI	WO			
Instruction 7							FI	DI	CO	FO	EI	WO		
Instruction 8								FI	DI	CO	FO	EI	WO	
Instruction 9									FI	DI	CO	FO	EI	WO

Figure 14.10 Timing Diagram for Instruction Pipeline Operation

The Effect of a Conditional Branch on Instruction Pipeline Operation

Time →

Branch Penalty ↔

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Instruction 1	FI	DI	CO	FO	EI	WO								
Instruction 2		FI	DI	CO	FO	EI	WO							
Instruction 3			FI	DI	CO	FO	EI	WO						
Instruction 4				FI	DI	CO	FO							
Instruction 5					FI	DI	CO							
Instruction 6						FI	DI							
Instruction 7							FI							
Instruction 15								FI	DI	CO	FO	EI	WO	
Instruction 16									FI	DI	CO	FO	EI	WO

Figure 14.11 The Effect of a Conditional Branch on Instruction Pipeline Operation

Not all instructions go thru all six stages; FI, FO, & WO stages involve a memory access, i.e., potential memory conflicts; however desired values may be in cache or one or more stages may be null; CO stage may be dependent upon register contents to be modified by previous instruction still in the pipeline;

Six Stage Instruction Pipeline

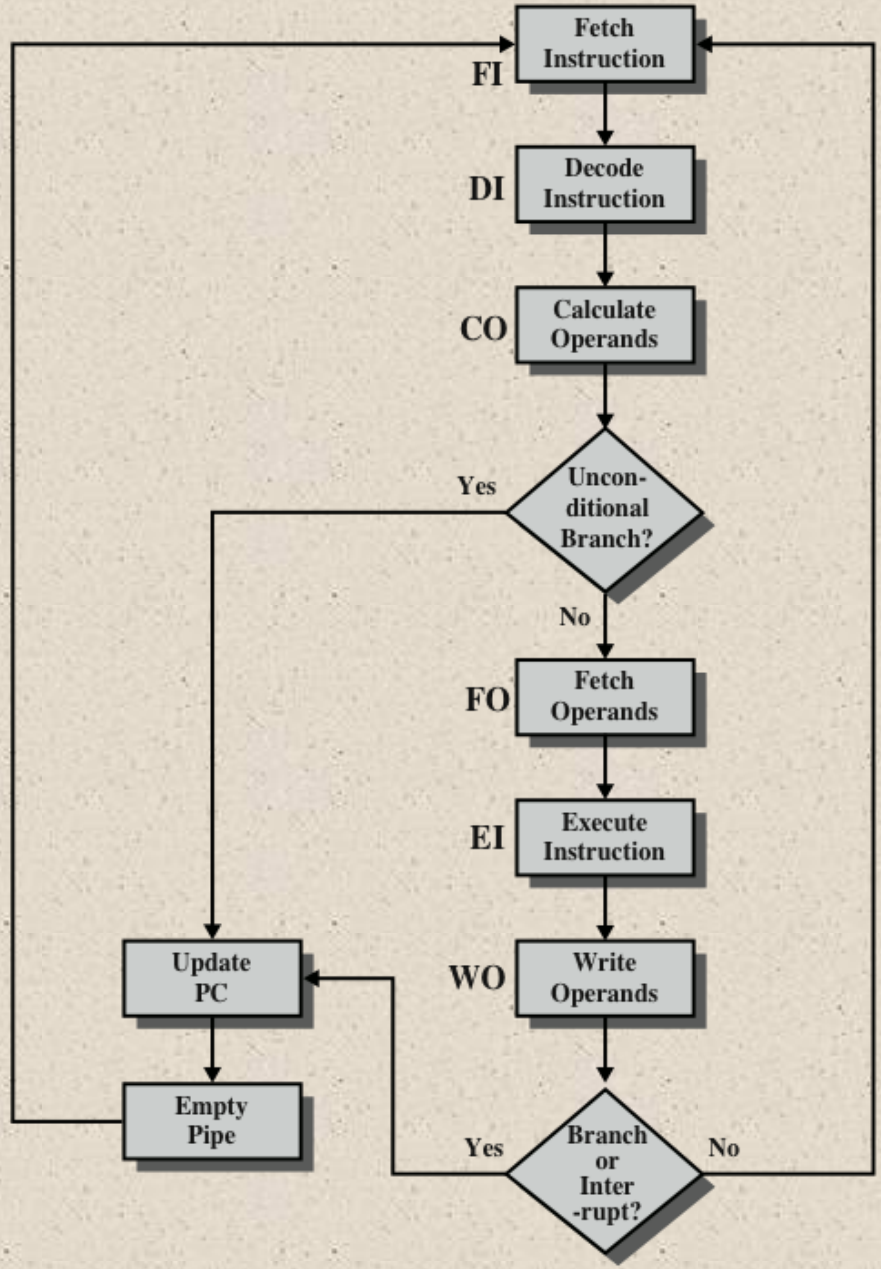


Figure 14.12 Six-Stage Instruction Pipeline

Alternative Pipeline Depiction

In Figure 14.13b, (which corresponds to Figure 14.11), the pipeline is full at times 6 and 7. At time 7, instruction 3 is in the execute stage and executes a branch to instruction 15. At this point, instructions I4 through I7 are flushed from the pipeline, so that at time 8, only two instructions are in the pipeline, I3 and I15.

Time

	FI	DI	CO	FO	EI	WO
1	I1					
2	I2	I1				
3	I3	I2	I1			
4	I4	I3	I2	I1		
5	I5	I4	I3	I2	I1	
6	I6	I5	I4	I3	I2	I1
7	I7	I6	I5	I4	I3	I2
8	I8	I7	I6	I5	I4	I3
9	I9	I8	I7	I6	I5	I4
10		I9	I8	I7	I6	I5
11			I9	I8	I7	I6
12				I9	I8	I7
13					I9	I8
14						I9

(a) No branches

	FI	DI	CO	FO	EI	WO
1	I1					
2	I2	I1				
3	I3	I2	I1			
4	I4	I3	I2	I1		
5	I5	I4	I3	I2	I1	
6	I6	I5	I4	I3	I2	I1
7	I7	I6	I5	I4	I3	I2
8	I15					I3
9	I16	I15				
10		I16	I15			
11			I16	I15		
12				I16	I15	
13					I16	I15
14						I16

(b) With conditional branch

Figure 14.13 An Alternative Pipeline Depiction

Pipeline Hazards

Occur when the pipeline, or some portion of the pipeline, must stall because conditions do not permit continued execution

There are three types of hazards:

- Resource
- Data
- Control

Also referred to as a *pipeline bubble*

Resource Hazards

A resource hazard occurs when two or more instructions that are already in the pipeline need the same resource

The result is that the instructions must be executed in serial rather than parallel for a portion of the pipeline

A resource hazard is sometimes referred to as a *structural hazard*

Assume a simplified five-stage pipeline, in which each stage takes one clock cycle. Now assume that main memory has a single port and that all instruction fetches and data reads and writes must be performed one at a time. Further, ignore the cache. In this case, an operand read to or write from memory cannot be performed in parallel with an instruction fetch. Therefore, the fetch instruction stage of the pipeline must idle for one cycle before beginning the instruction fetch for instruction I3. The figure assumes that all other operands are in registers.

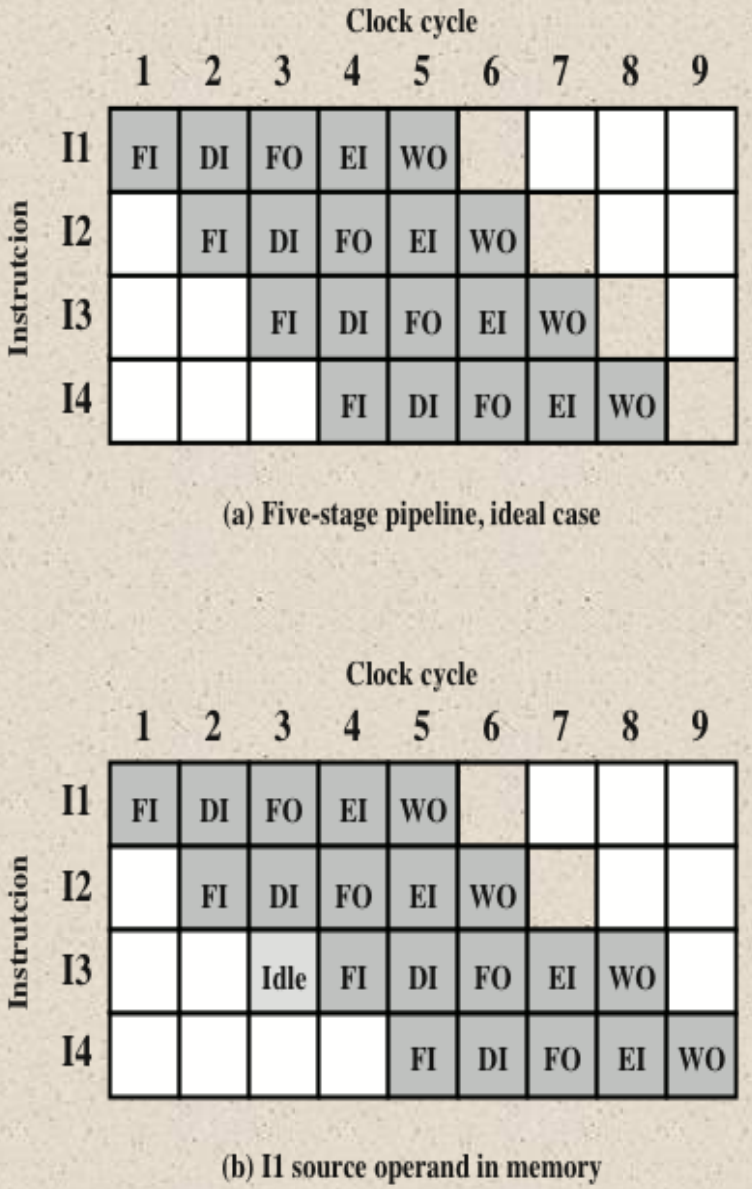


Figure 14.15 Example of Resource Hazard

	Clock cycle 1	2	3	4	5	6	7	8	9	10
ADD EAX, EBX	FI	DI	FO	EI	WO					
SUB ECX, EAX		FI	DI	Idle		FO	EI	WO		
I3			FI			DI	FO	EI	WO	
I4						FI	DI	FO	EI	WO

Figure 14.16 Example of Data Hazard

RAW

Hazard

Data Hazards

A data hazard occurs when there is a conflict in the access of an operand location

Two instructions in a program are to be executed in sequence and both access a particular memory or register operand. If the two instructions are executed in strict sequence, no problem occurs. However, if the instructions are executed in a pipeline, then it is possible for the operand value to be updated in such a way as to produce a different result than would occur with strict sequential execution. In other words, the program produces an incorrect result because of the use of pipelining.

Types of Data Hazard

- Read after write (RAW), or true dependency
  - An instruction modifies a register or memory location
  - Succeeding instruction reads data in memory or register location
  - Hazard occurs if the read takes place before write operation is complete
- Write after read (WAR), or antidependency
  - An instruction reads a register or memory location
  - Succeeding instruction writes to the location
  - Hazard occurs if the write operation completes before the read operation takes place
- Write after write (WAW), or output dependency
  - Two instructions both write to the same location
  - Hazard occurs if the write operations take place in the reverse order of the intended sequence

Control Hazard

- Also known as a *branch hazard*
- Occurs when the pipeline makes the wrong decision on a branch prediction
- Brings instructions into the pipeline that must subsequently be discarded
- Dealing with Branches:
  - Multiple streams
  - Prefetch branch target
  - Loop buffer
  - Branch prediction
  - Delayed branch

Multiple Streams

A simple pipeline suffers a penalty for a branch instruction because it must choose one of two instructions to fetch next and may make the wrong choice

A brute-force approach is to replicate the initial portions of the pipeline and allow the pipeline to fetch both instructions, making use of two streams

Drawbacks:

- With multiple pipelines there are contention delays for access to the registers and to memory
- Additional branch instructions may enter the pipeline before the original branch decision is resolved

Prefetch Branch Target

- When a conditional branch is recognized, the target of the branch is prefetched, in addition to the instruction following the branch
- Target is then saved until the branch instruction is executed
- If the branch is taken, the target has already been prefetched
- IBM 360/91 uses this approach

Loop Buffer

- Small, very-high speed memory maintained by the instruction fetch stage of the pipeline and containing the *n* most recently fetched instructions, in sequence
- Benefits:
  - Instructions fetched in sequence will be available without the usual memory access time
  - If a branch occurs to a target just a few locations ahead of the address of the branch instruction, the target will already be in the buffer
  - This strategy is particularly well suited to dealing with loops
- Similar in principle to a cache dedicated to instructions
  - Differences:
    - The loop buffer only retains instructions in sequence
    - Is much smaller in size and hence lower in cost

Loop Buffer

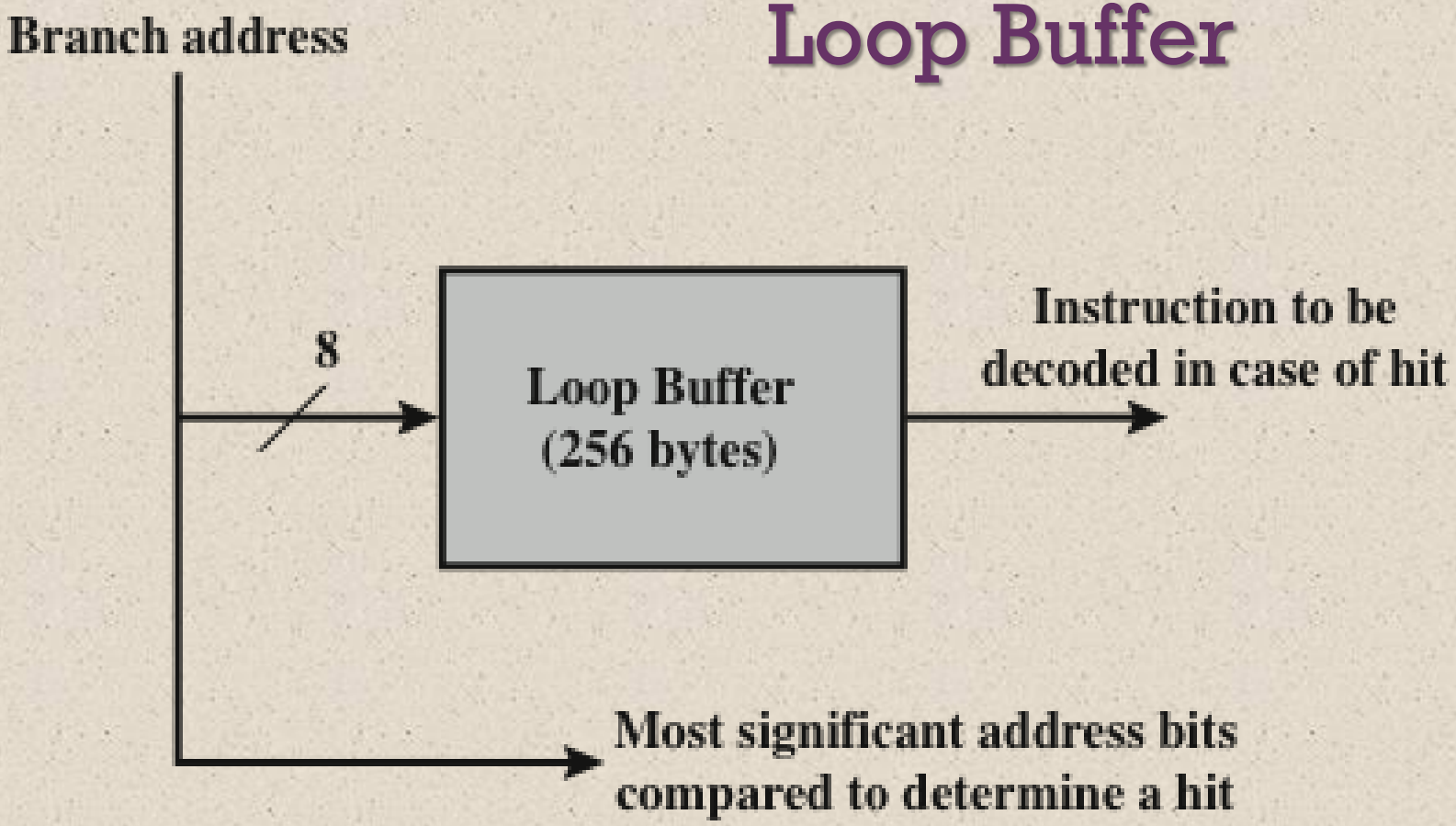


Figure 14.17 Loop Buffer

Branch Prediction

- Various techniques can be used to predict whether a branch will be taken:

1. Predict never taken
2. Predict always taken
3. Predict by opcode

- These approaches are static
- They do not depend on the execution history up to the time of the conditional branch instruction

1. Taken/not taken switch
2. Branch history table

- These approaches are dynamic
- They depend on the execution history

Branch Prediction Flow Chart

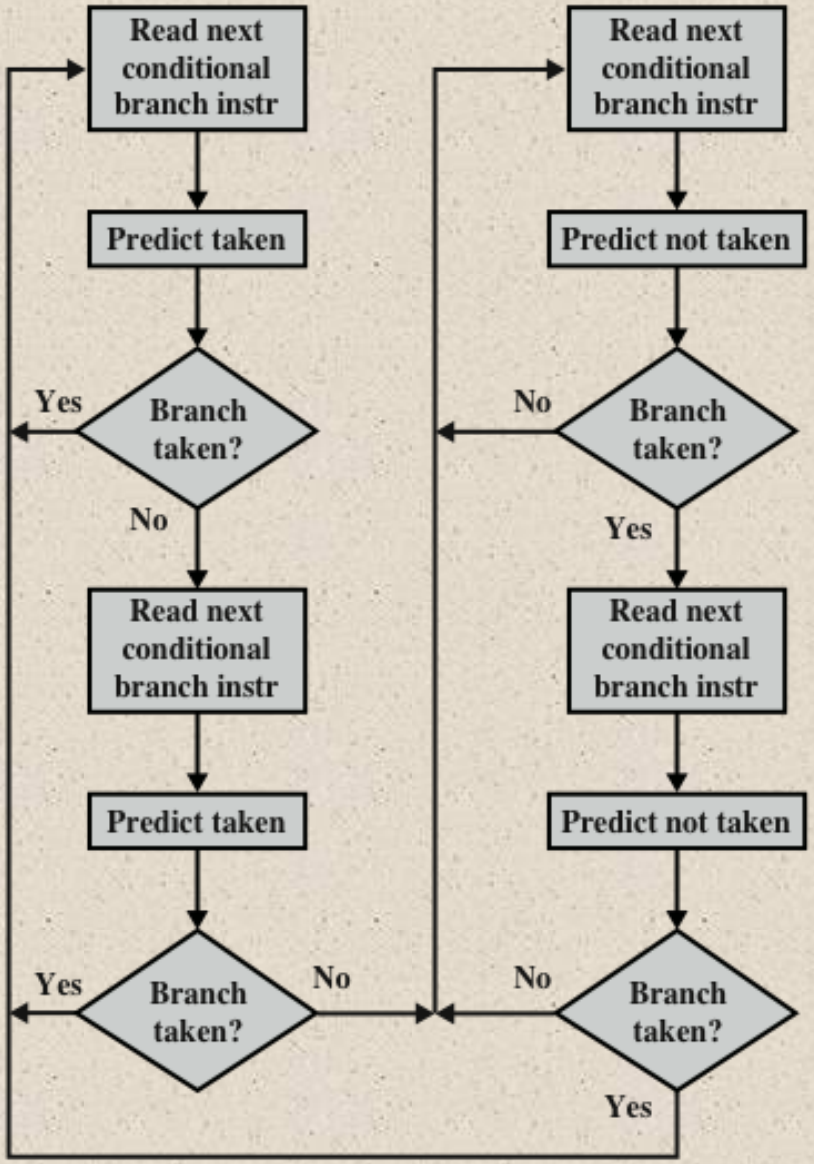


Figure 14.18 Branch Prediction Flow Chart

Branch Prediction State Diagram

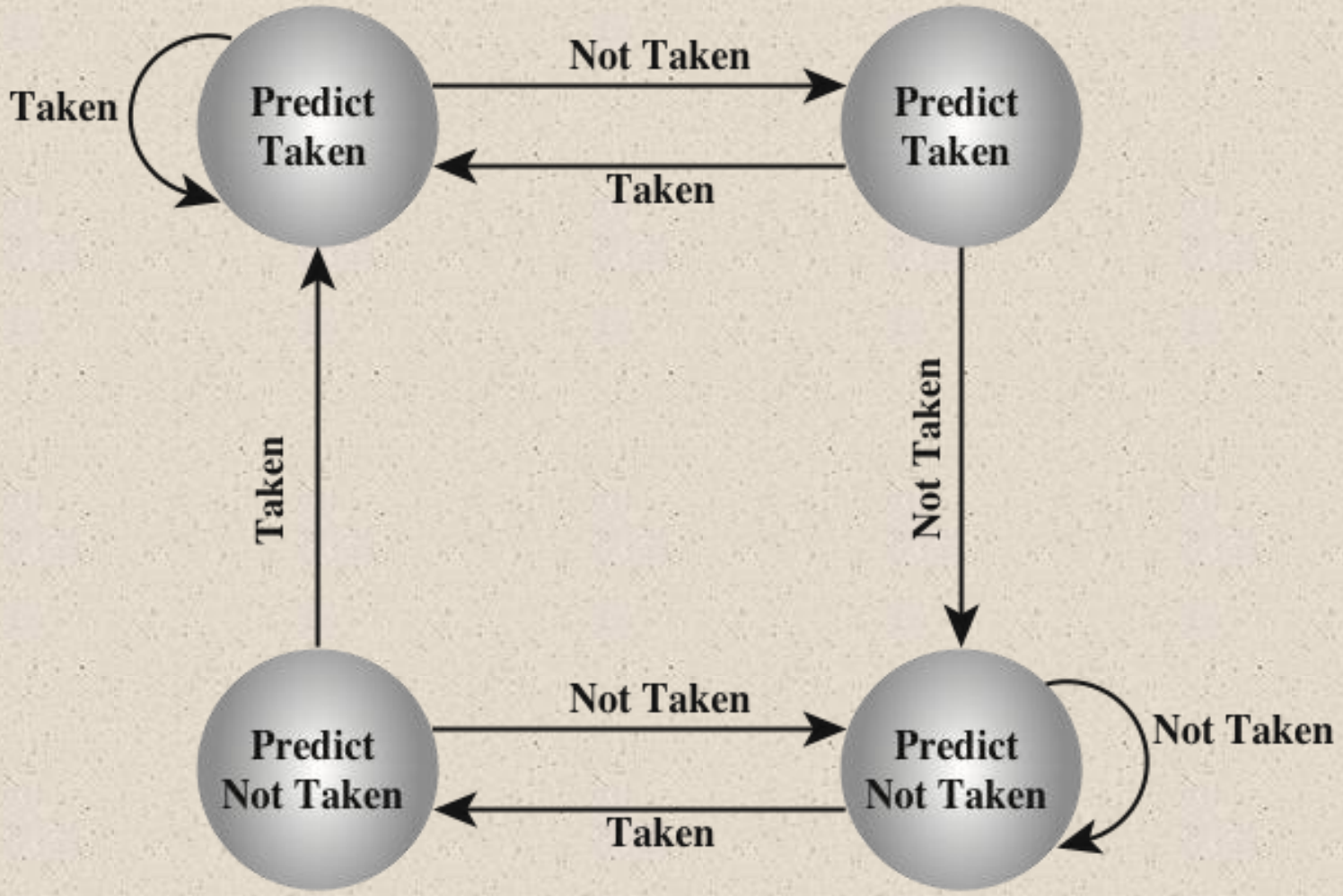


Figure 14.19 Branch Prediction State Diagram

Summary

Chapter 14

Processor Structure and Function

- Processor organization
- Register organization
  - User-visible registers
  - Control and status registers
- Instruction cycle
  - The indirect cycle
  - Data flow
- The x86 processor family
  - Register organization
  - Interrupt processing
- Instruction pipelining
  - Pipelining strategy
  - Pipeline performance
  - Pipeline hazards
  - Dealing with branches
  - Intel 80486 pipelining
- The Arm processor
  - Processor organization
  - Processor modes
  - Register organization
  - Interrupt processing