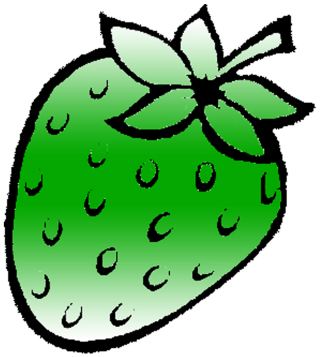
STRAWBERRY



/strawberrydevelopers

/strawberry_app

For more visit:

Strawberrydevelopers.weebly.com

UNIT V: CENTRAL PROCESSING UNIT

Agenda

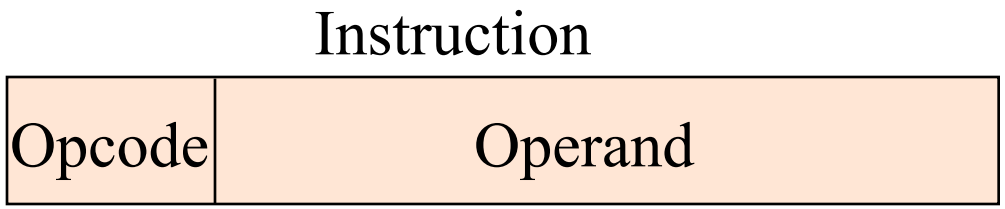
- Basic Instruction Cycle & Sets
- Addressing
- Instruction Format
- Processor Organization
- Register Organization
- Pipeline Processors
- Instruction Pipelining
- Co-Processors
- RISC computers vs. CISC computers – Assignment 2

Addressing Modes

- Immediate
- Direct
- Indirect
- Register
- Register indirect
- Displacement
- Stack

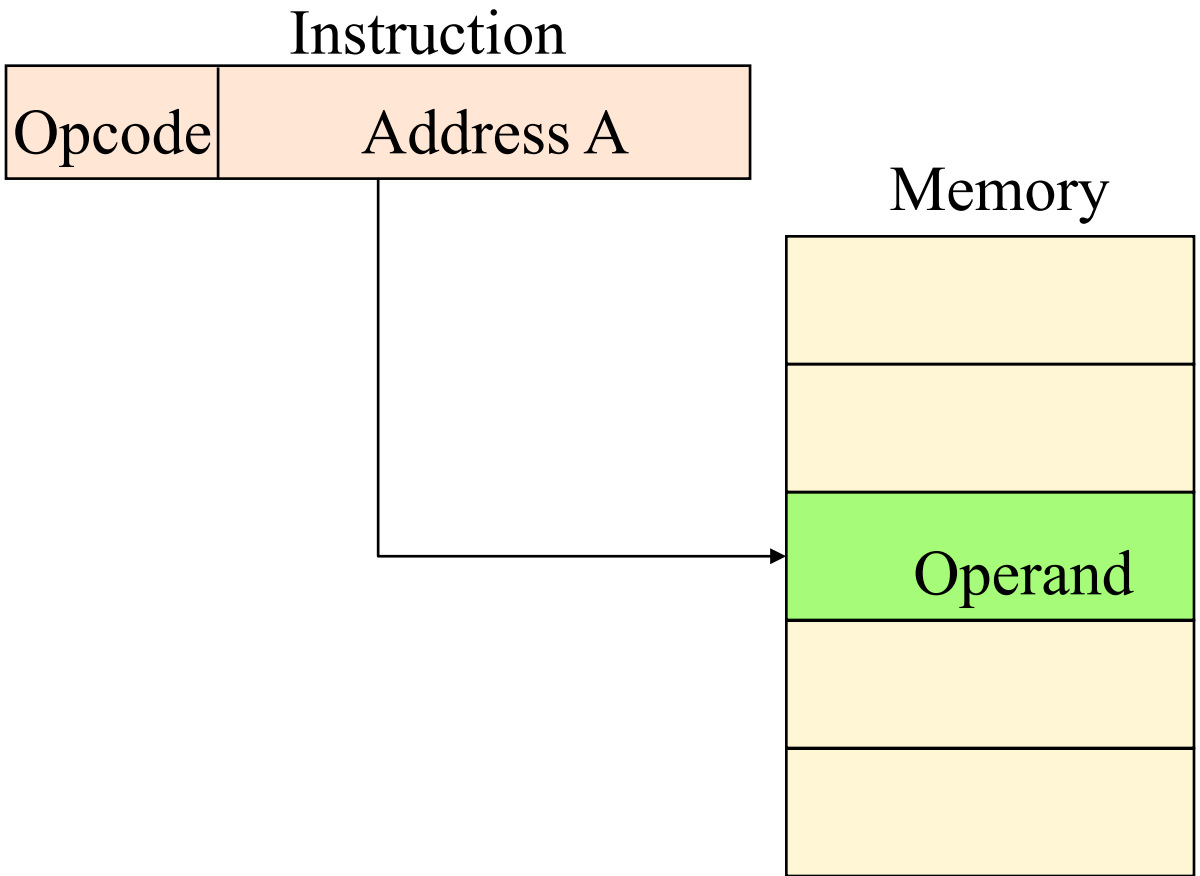
Immediate Addressing

- Operand is part of instruction
- e.g. ADD 5
  - Add 5 to contents of accumulator
  - 5 is operand
- No memory reference to fetch data
- Fast
- Limited range



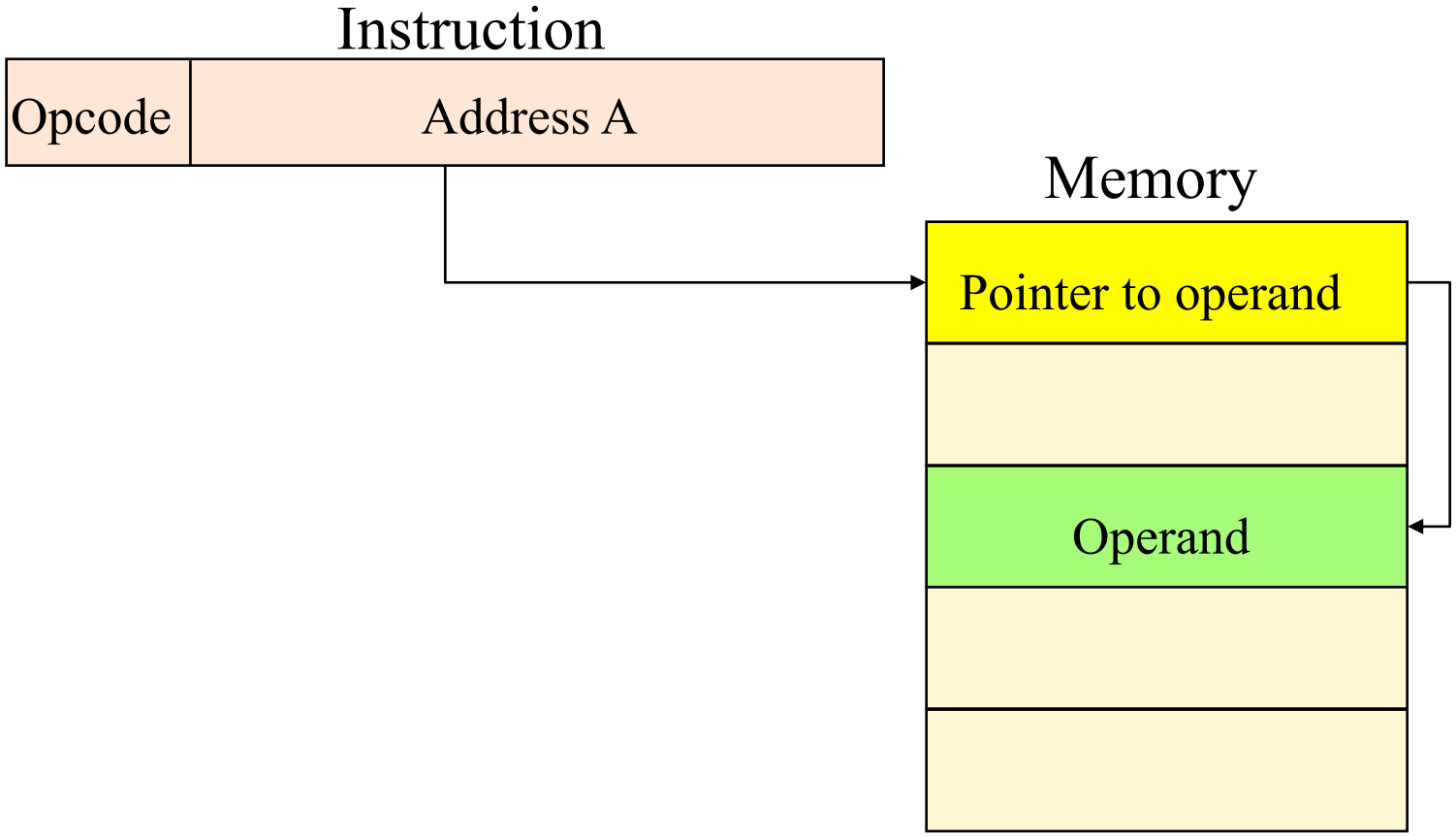
Direct Addressing

- Address field contains address of operand
- Effective address (EA) = address field (A)
- e.g. ADD A
  - Add contents of cell A to accumulator
  - Look in memory at address A for operand
- Single memory reference to access data
- Limited address space



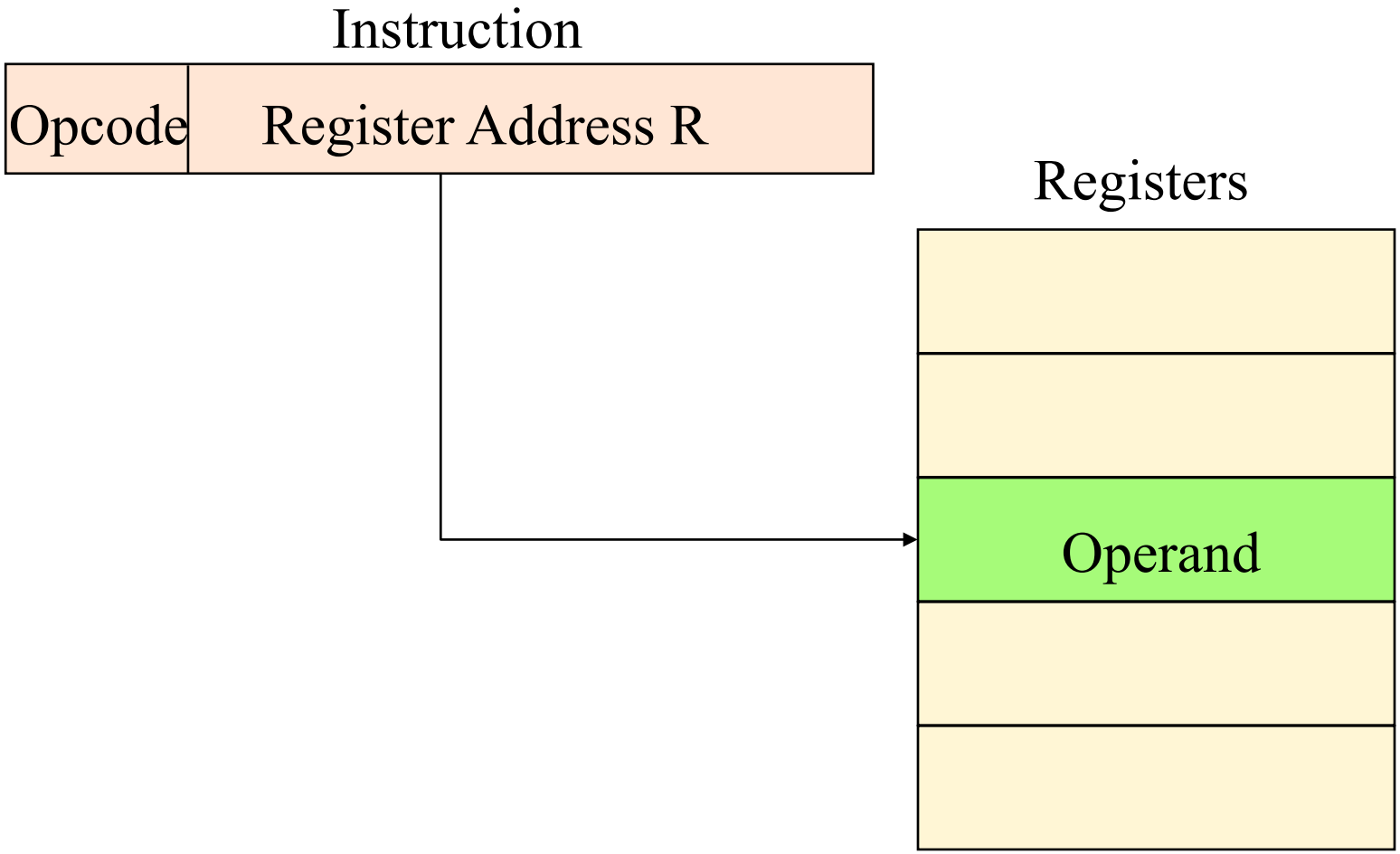
Indirect Addressing

- Memory cell pointed to by address field contains the address of (pointer to) the operand
- e.g. ADD (A)
  - Add contents of cell pointed to by contents of A to accumulator



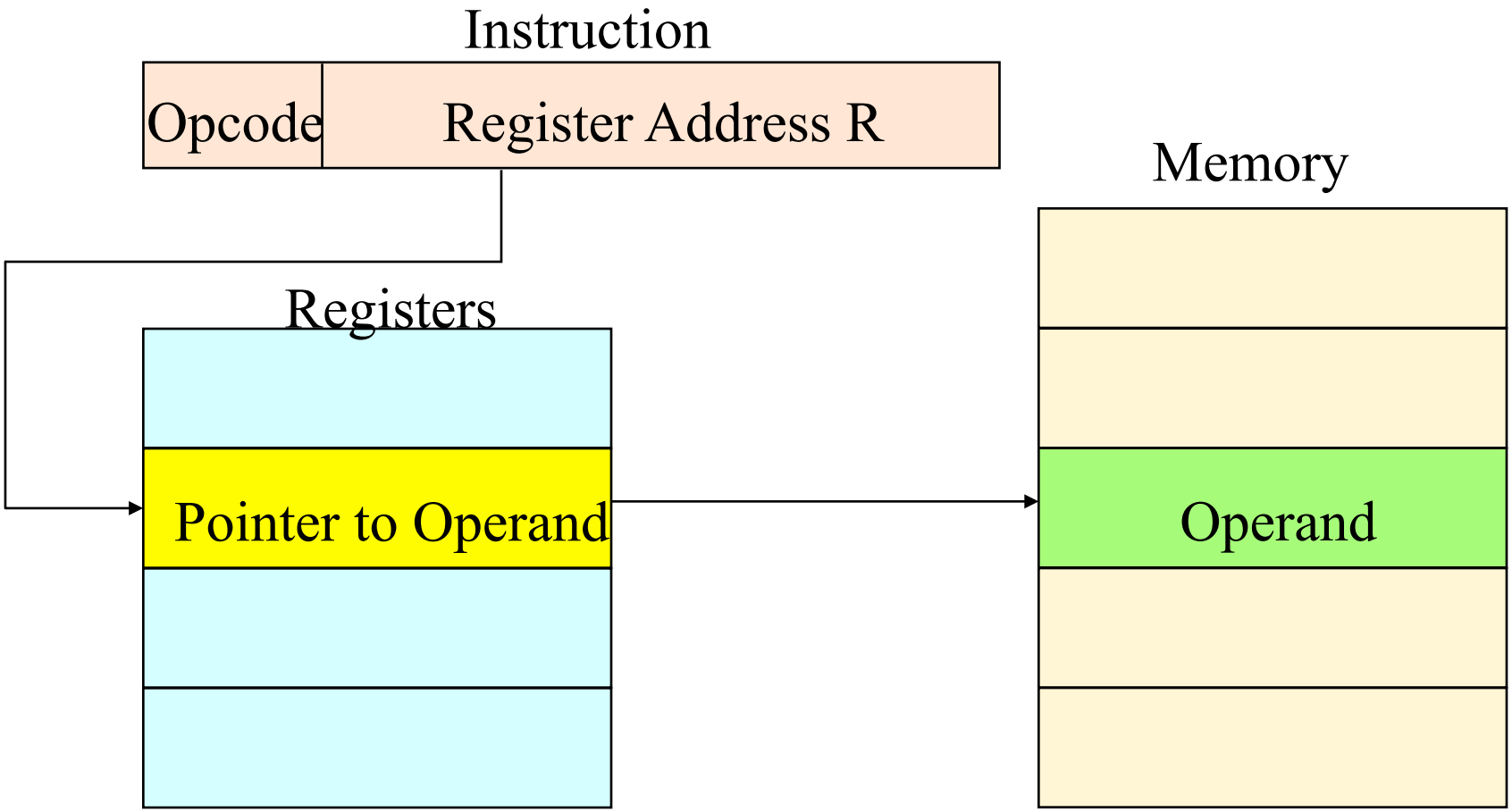
Register Addressing

- Operand is held in register named in address filed
- EA = R
- Limited number of registers



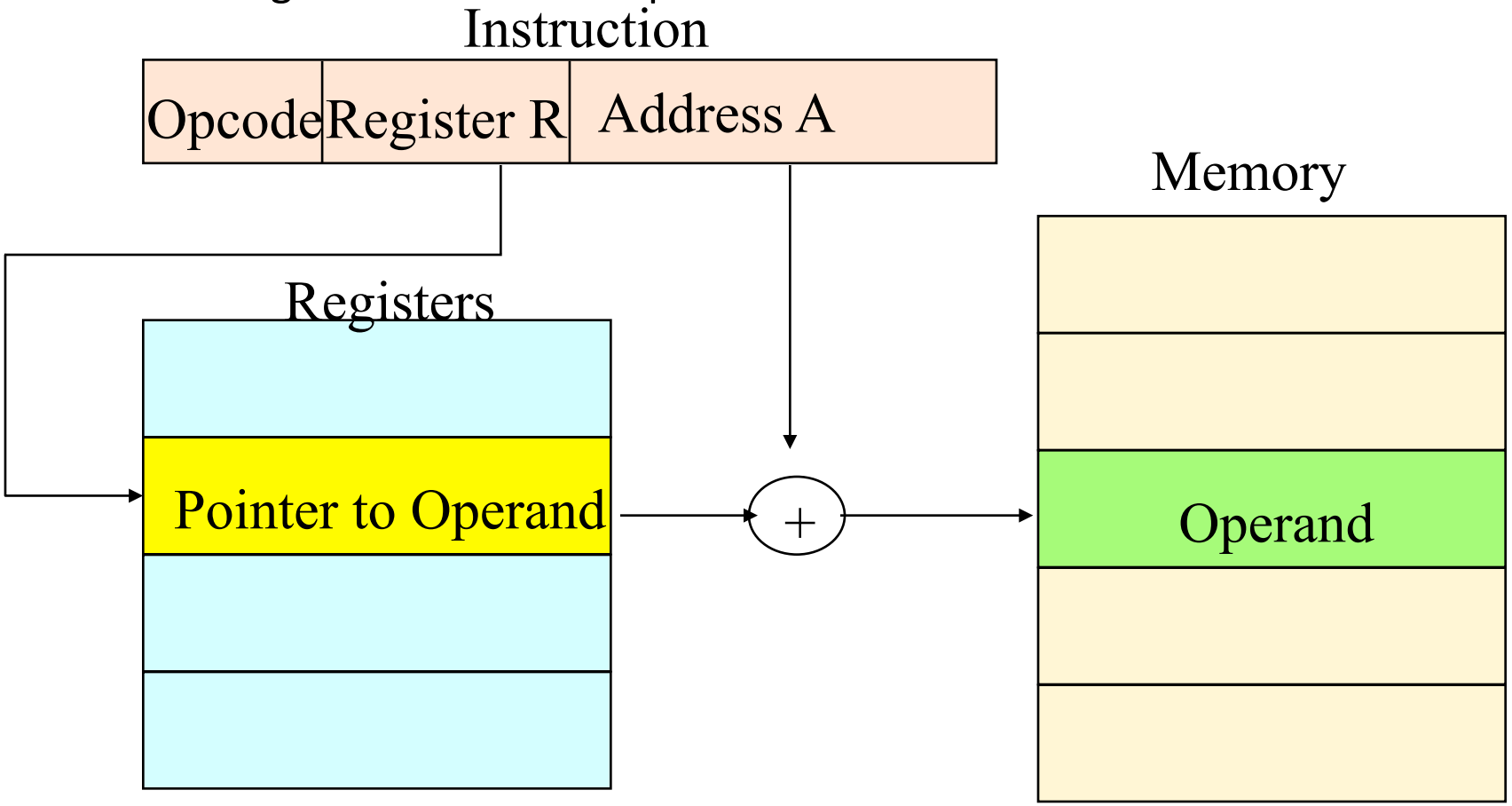
Register Indirect Addressing

- Operand is in memory cell pointed to by contents of register R



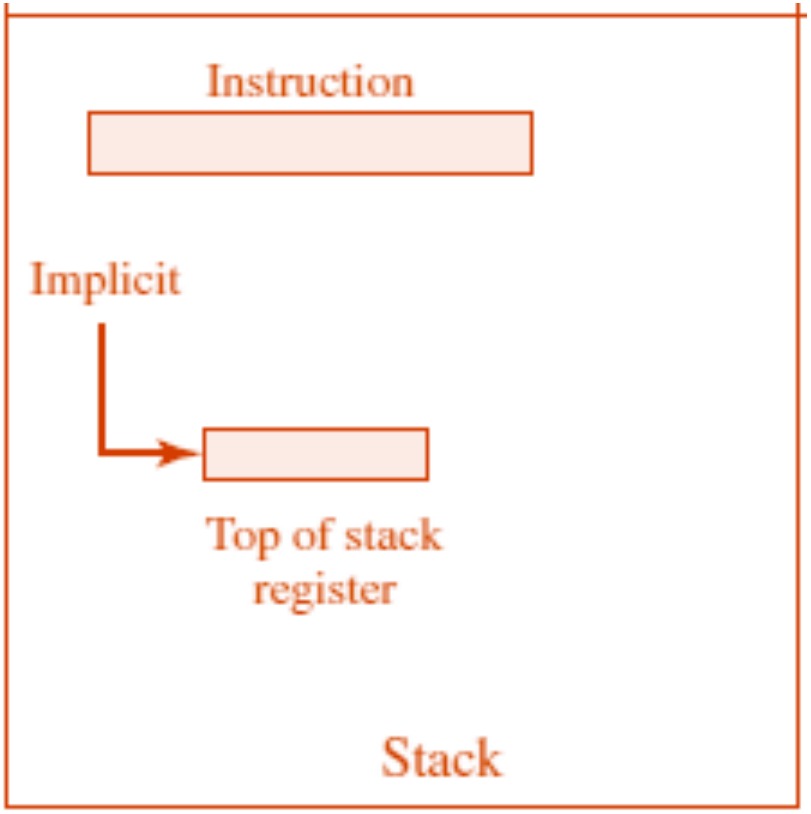
Displacement Addressing

- EA = A + (R)
- Address field hold two values
  - A = base value
  - R = register that holds displacement



Stack Addressing

- Operand is (implicitly) on top of stack
- e.g.
  - ADD Pop top two items from stack and add



Instruction Formats

- Layout of bits in an instruction
- Includes opcode
- Includes (implicit or explicit) operand(s)
- Usually more than one instruction format in an instruction set

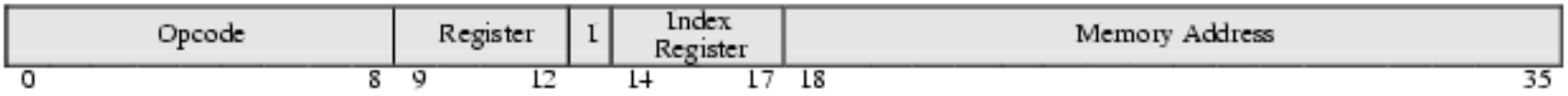
Instruction Length

- Affected by and affects:
  - Memory size
  - Memory organization
  - Bus structure
  - CPU complexity
  - CPU speed

Allocation of Bits

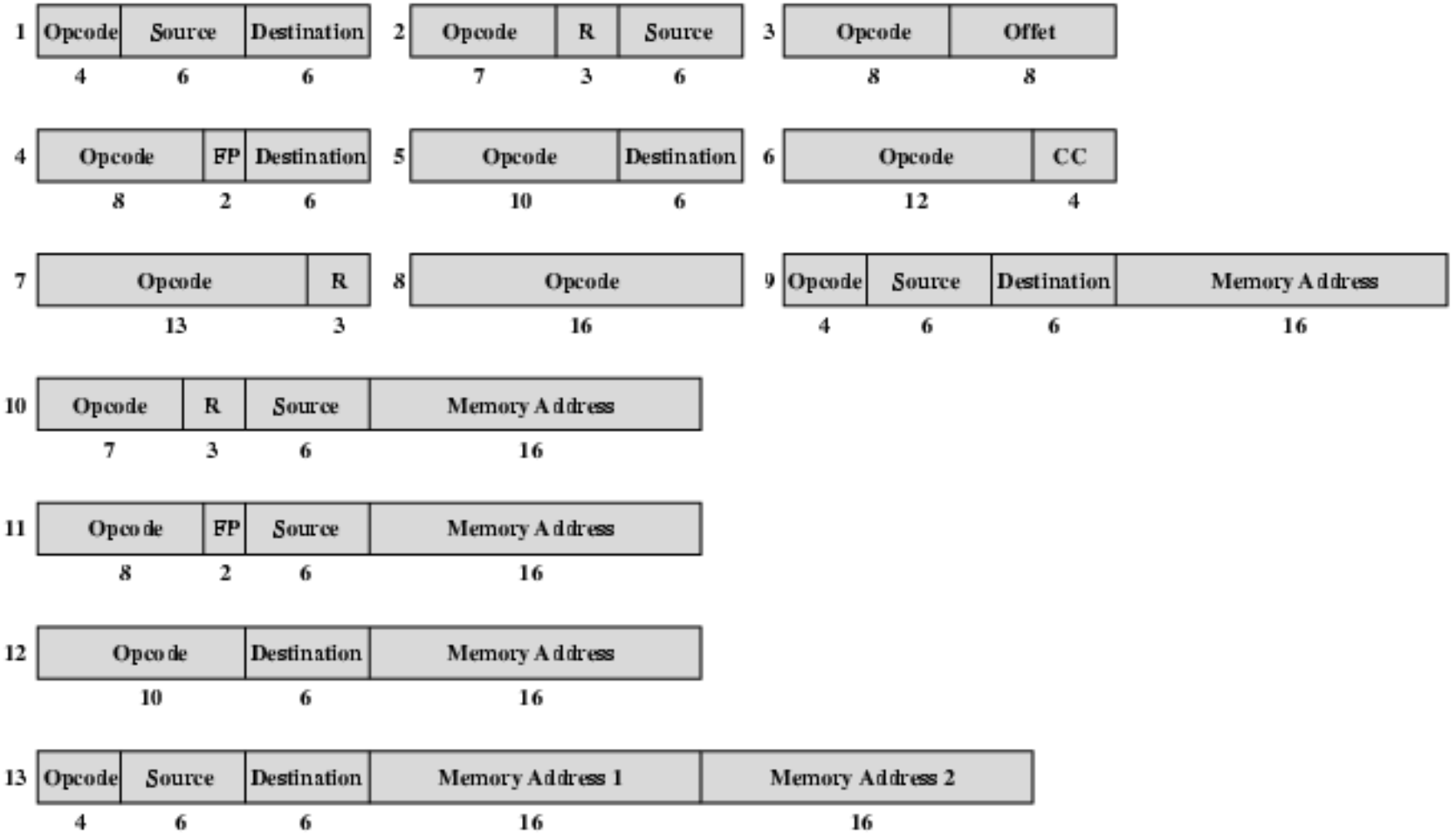
- Number of addressing modes
- Number of operands
- Register versus memory
- Number of register sets
- Address range
- Address granularity

Example: PDP-10 Instruction Format



I = indirect bit

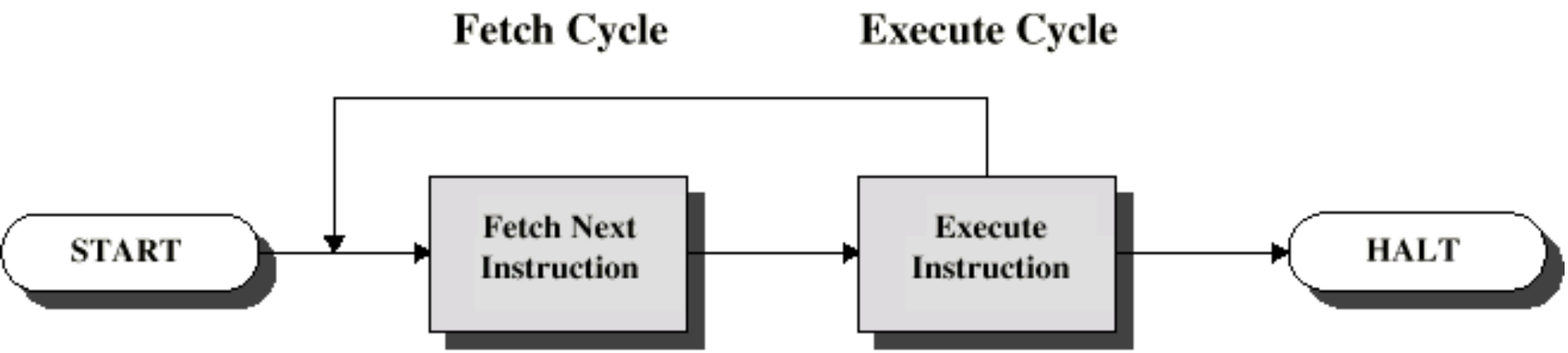
Example: PDP-11 Instruction Format



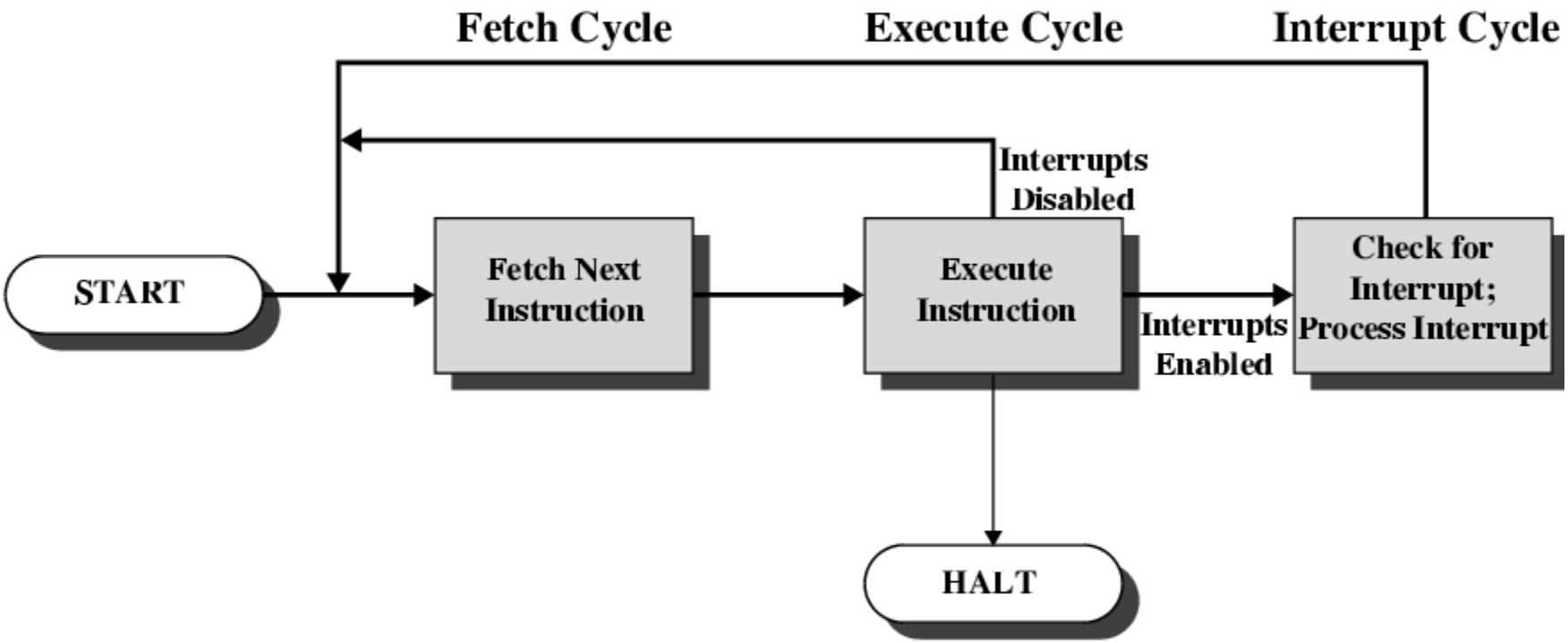
Numbers below fields indicate bit length
Source and Destination each contain a 3-bit addressing mode field and a 3-bit register number
FP indicates one of four floating-point registers
R indicates one of the general-purpose registers
CC is the condition code field

Instruction Cycle

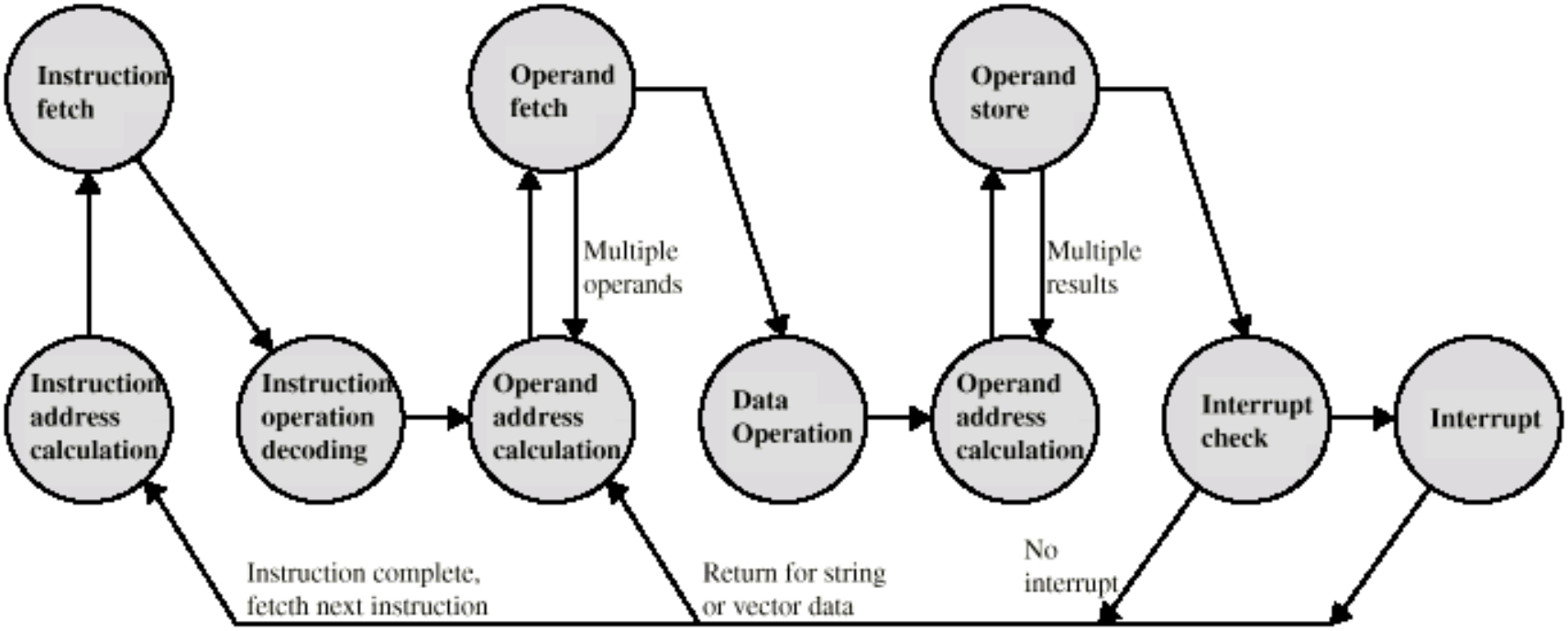
- **Fetch:** Read the next instruction from memory into the processor.
- **Execute:** Interpret the opcode and perform the indicated operation.
- **Interrupt:** If interrupts are enabled and an interrupt has occurred, save the current process state and service the interrupt.



Instruction Cycle with Interrupts

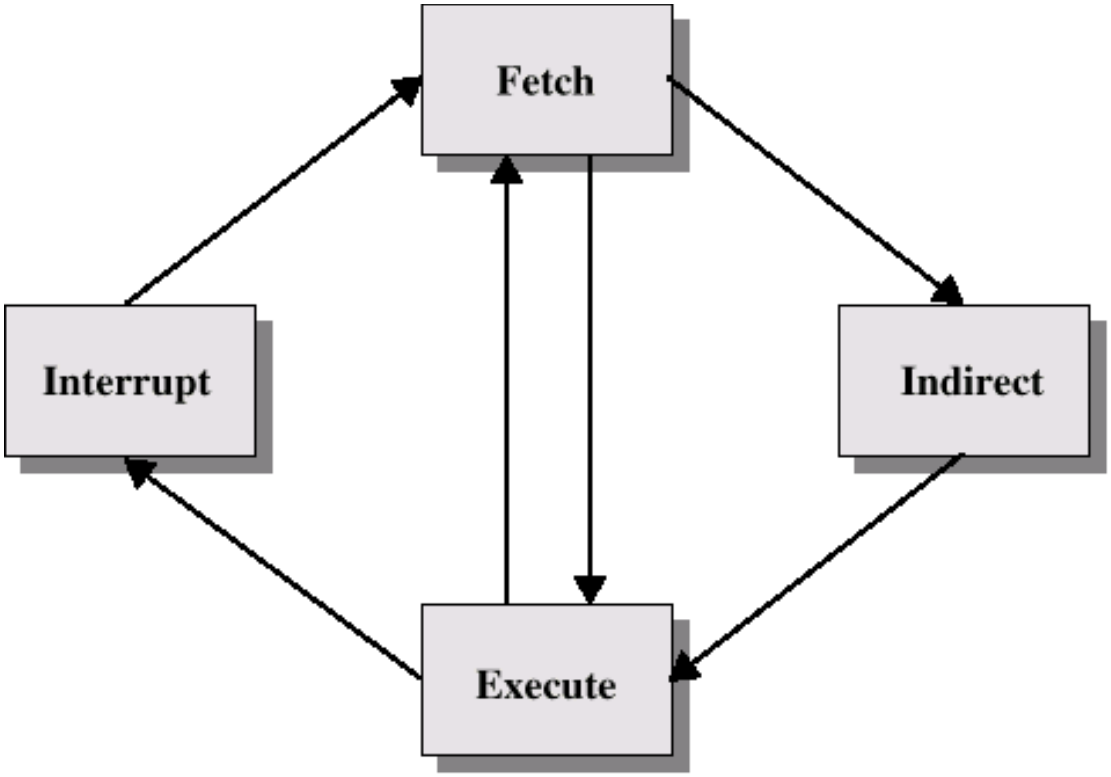


Instruction Cycle (with Interrupts) - State Diagram

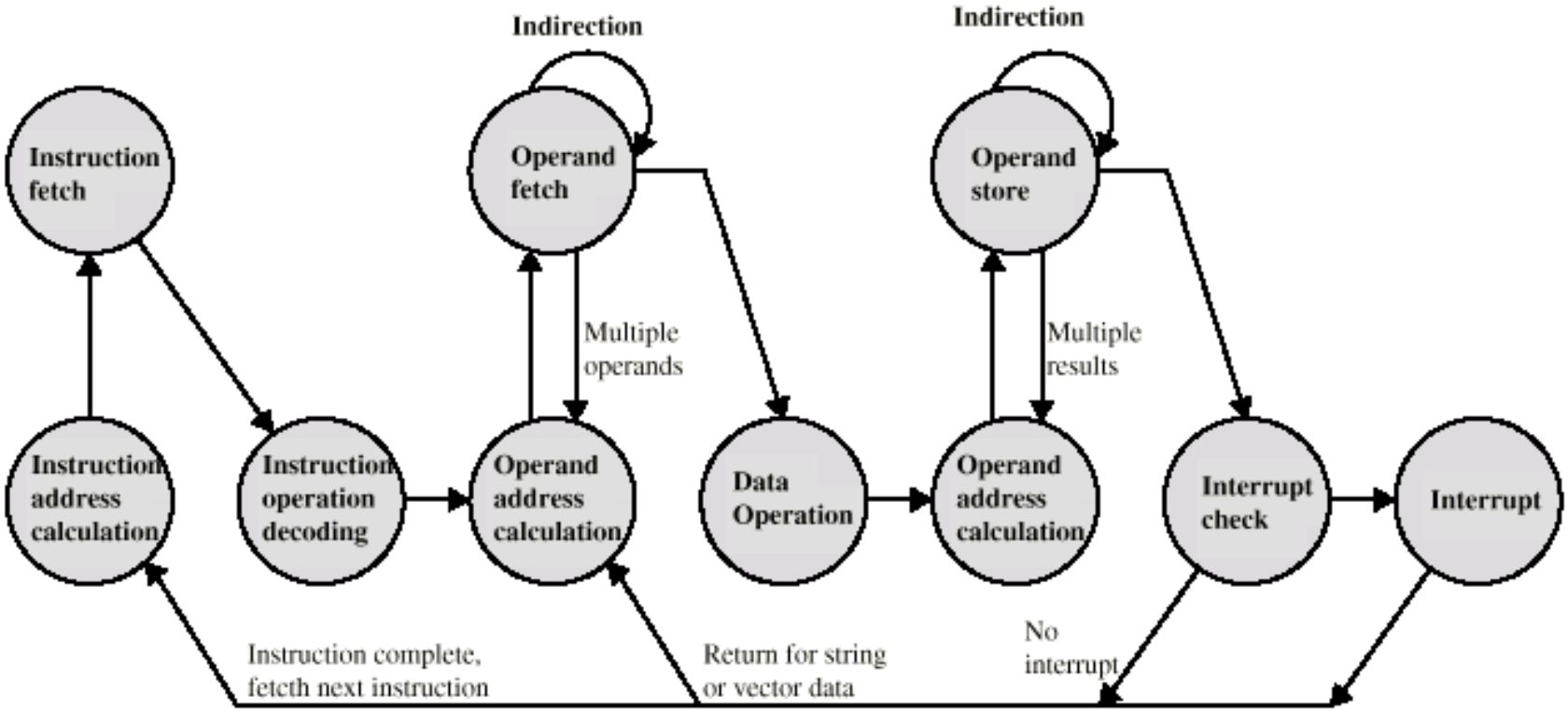


Indirect Cycle

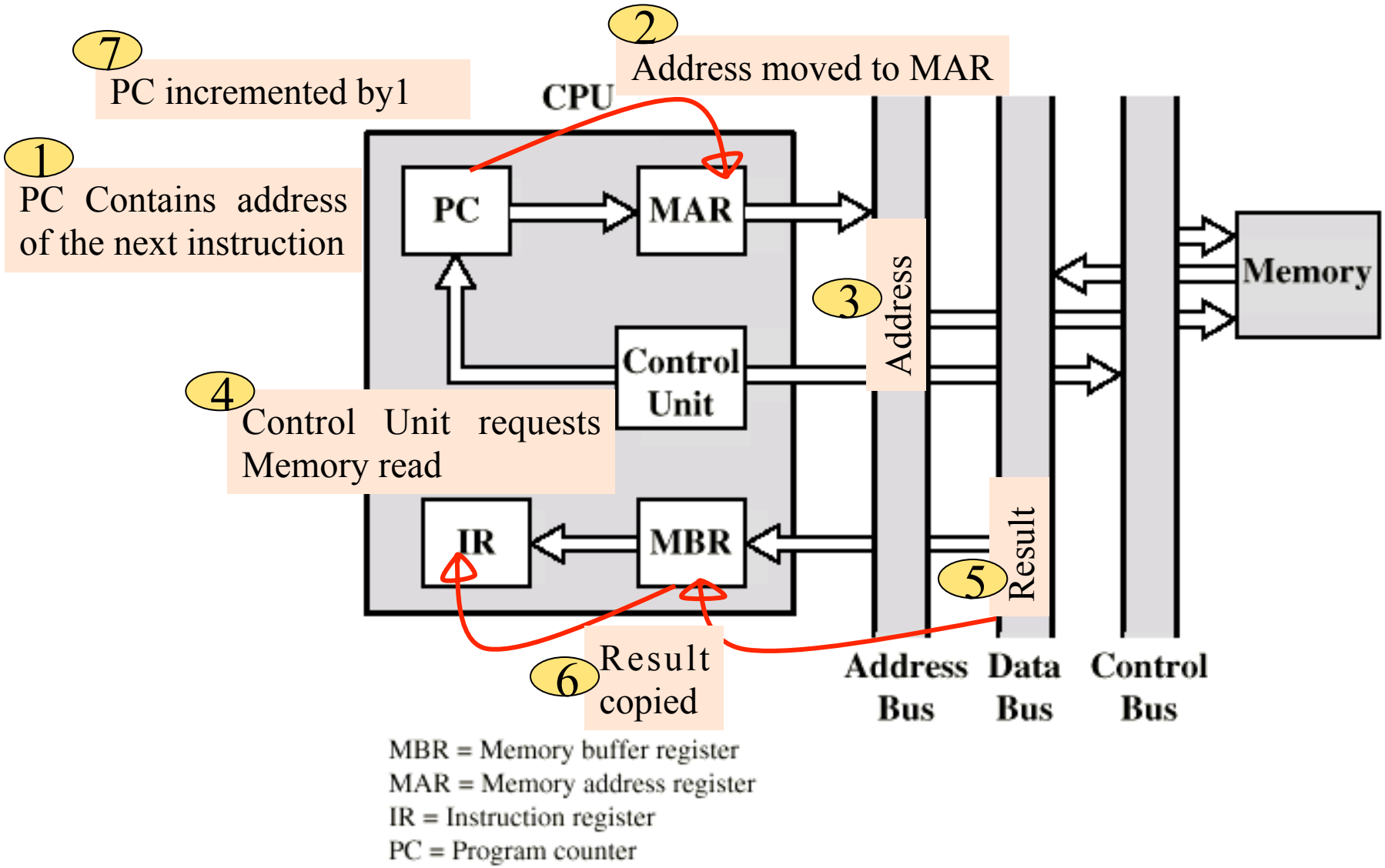
- The main line of activity consists of alternating instruction fetch and instruction execution activities.
- After an instruction is fetched, it is examined to determine if any indirect addressing is involved. If so, the required operands are fetched using indirect addressing. Following execution, an interrupt may be processed before the next instruction fetch.



Instruction Cycle State Diagram

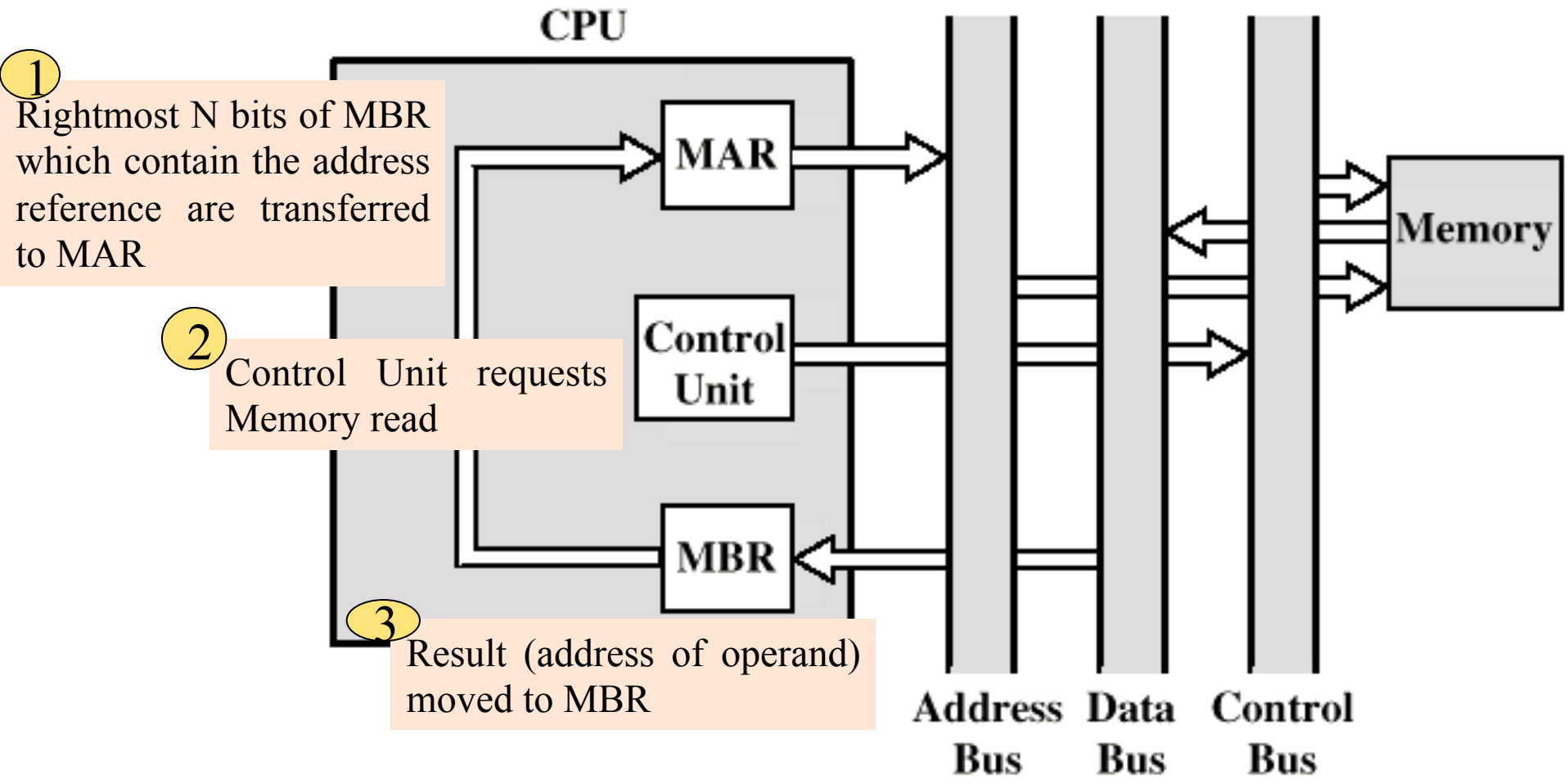


Data Flow (Instruction Fetch)



Data Flow (Indirect Cycle)

- IR is examined
- If indirect addressing, indirect cycle is performed:

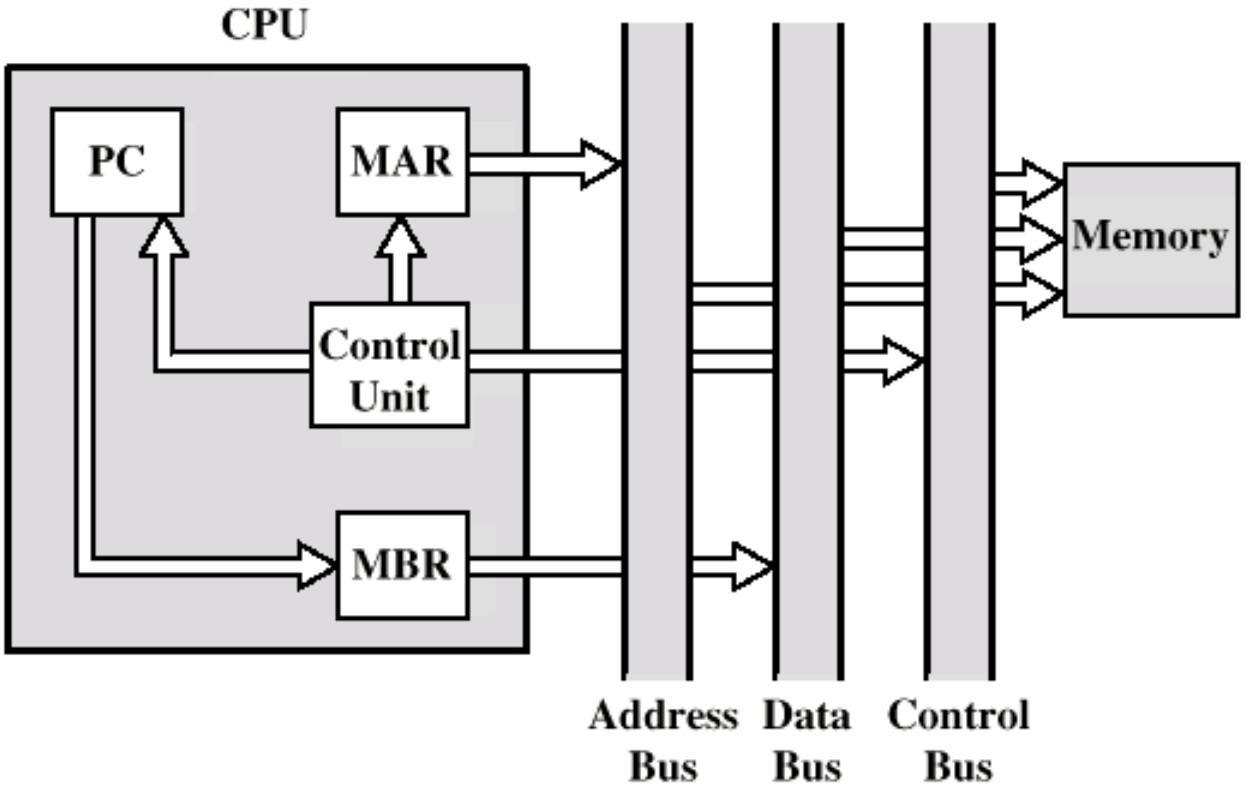


Data Flow (Execute)

- May take many forms
- Depends on instruction being executed
- May include
  - Memory read/write
  - Input/Output
  - Register transfers
  - ALU operations

Data Flow (Interrupt Cycle)

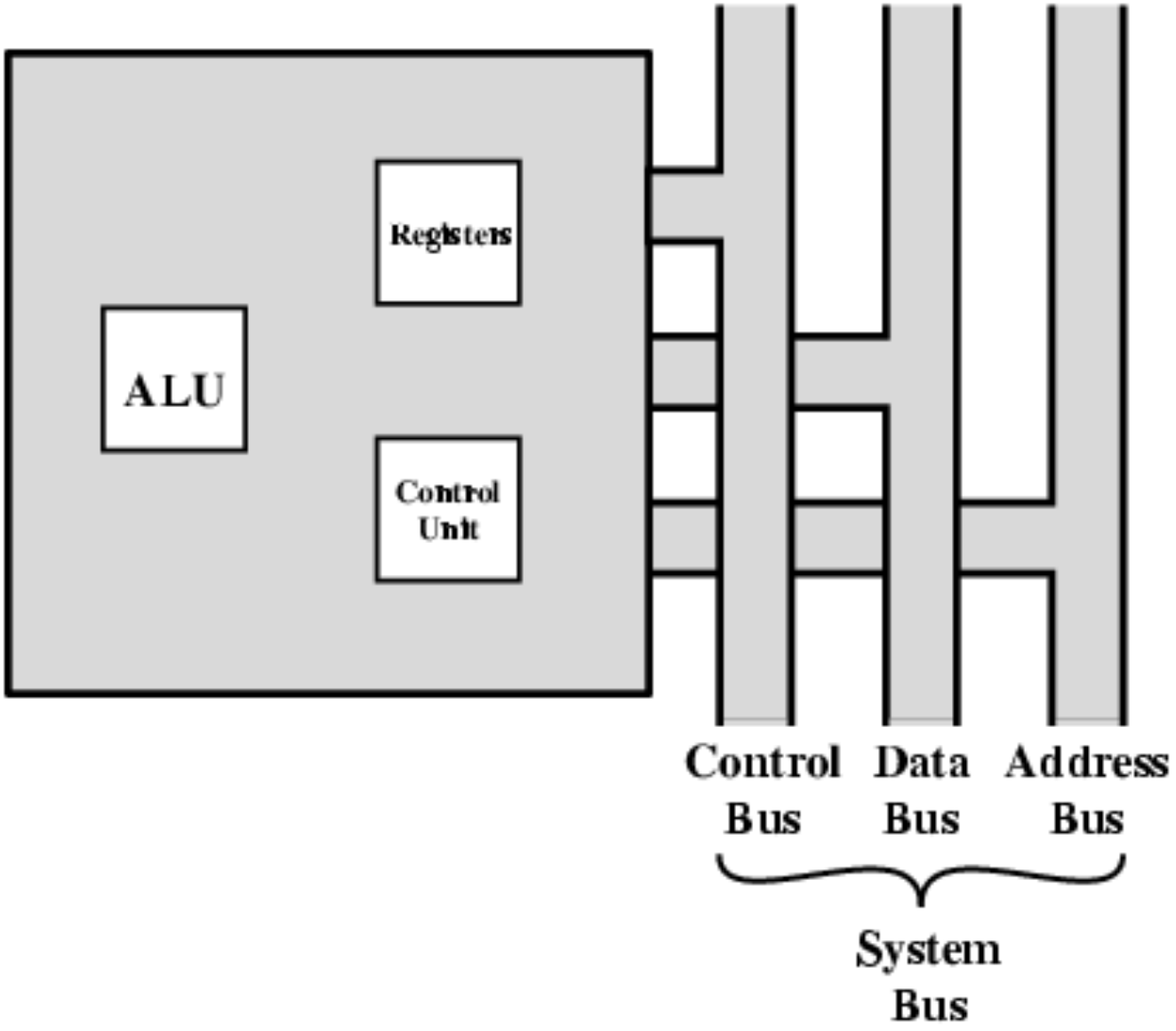
- The current contents of the PC must be saved so that the processor can resume normal activity after the interrupt. Thus, the contents of the PC are transferred to the MBR to be written into memory.
- The special memory location reserved for this purpose is loaded into the MAR from the control unit.
- The PC is loaded with the address of the interrupt routine. As a result, the next instruction cycle will begin by fetching the appropriate instruction.



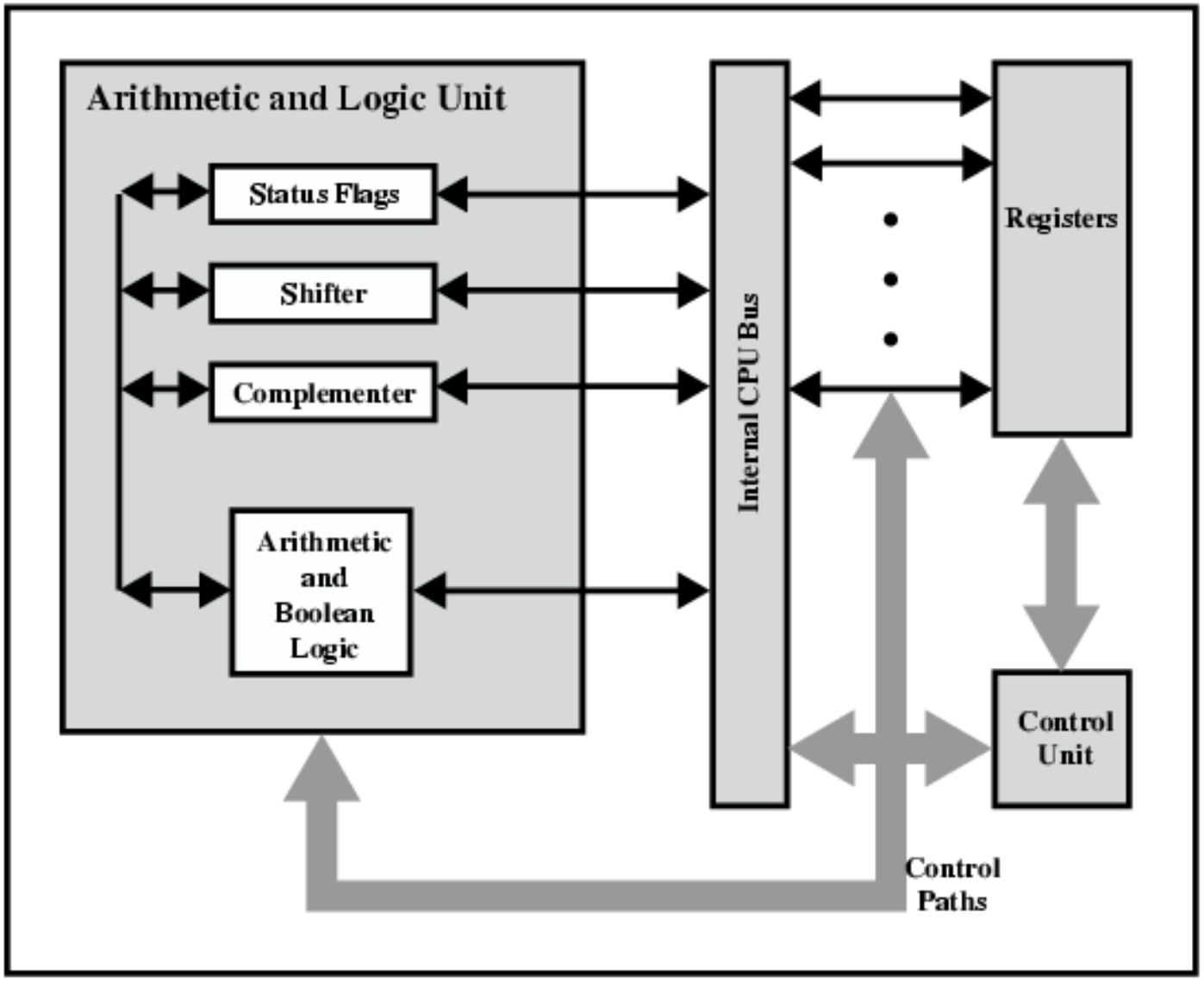
Processor Organization

- CPU must:
  - **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 With Systems Bus



CPU Internal Structure



Register Organization

- **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

- **General Purpose Registers:** May be used for data or addressing
- **Data registers: U**sed only to hold data
- **Address registers:** Devoted to a particular addressing mode. Examples- Index registers, Stack pointer
- **Condition** codes: Bits set by the processor hardware as the result of operations.
  - For example, an arithmetic operation may produce a positive, negative, zero, or overflow result. In addition to the result itself being stored in a register or memory, a condition code is also set. The code may subsequently be tested as part of a conditional branch operation.
  - Can be read (implicitly) by programs – e.g. Jump if zero

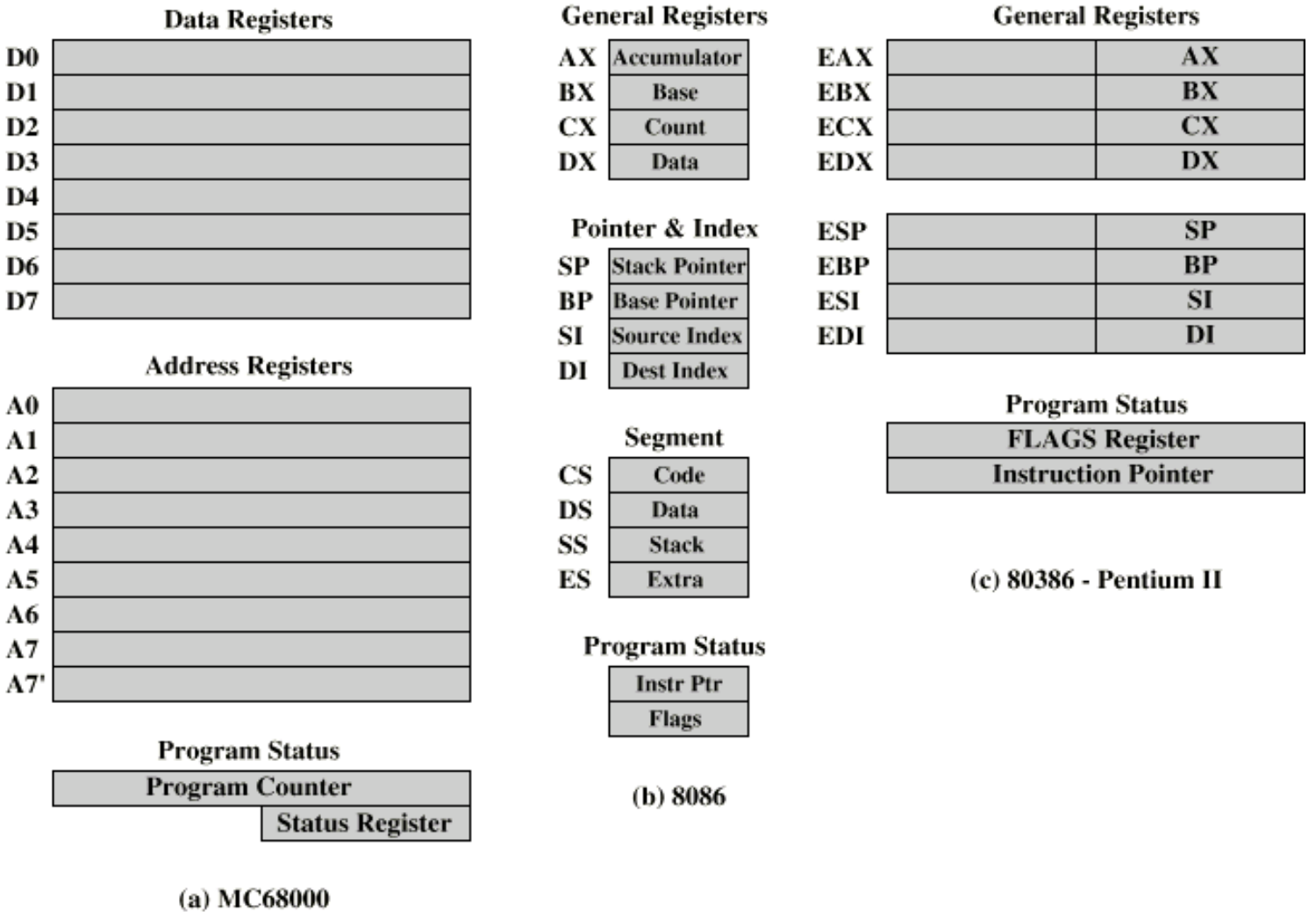
Control & Status Registers

- **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

- **Sign:** Contains the 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 (subtraction) out of a high-order bit. Used for multiword arithmetic operations.
- **Equal:** Set if a logical compare result is equality.
- **Overflow:** Used to indicate arithmetic overflow.
- **Interrupt Enable/Disable:** Used to enable or disable interrupts.
- **Supervisor:** Indicates whether the processor is executing in supervisor or user mode. Certain privileged instructions can be executed only in supervisor mode, and certain areas of memory can be accessed only in supervisor mode.

Example Register Organizations



Pipeline Processor

- Pipelining: A means of introducing parallelism into the essentially sequential nature of a machine instruction program.
- Example: Instruction Pipelining
- A processor which works on the property of parallelism is known as pipeline processor.

Pipelining

- Two stages: fetch instruction and execute instruction.
- The first stage fetches an instruction and buffers it. When the second stage is free, the first stage passes it the buffered instruction. While the second stage is executing the instruction, the first stage takes advantage of any unused memory cycles to fetch and buffer the next instruction. This is called *instruction prefetch* or *fetch overlap*.
- Advantage:
  - This process will speed up instruction execution. If the fetch and execute stages were of equal duration, the instruction cycle time would be halved.

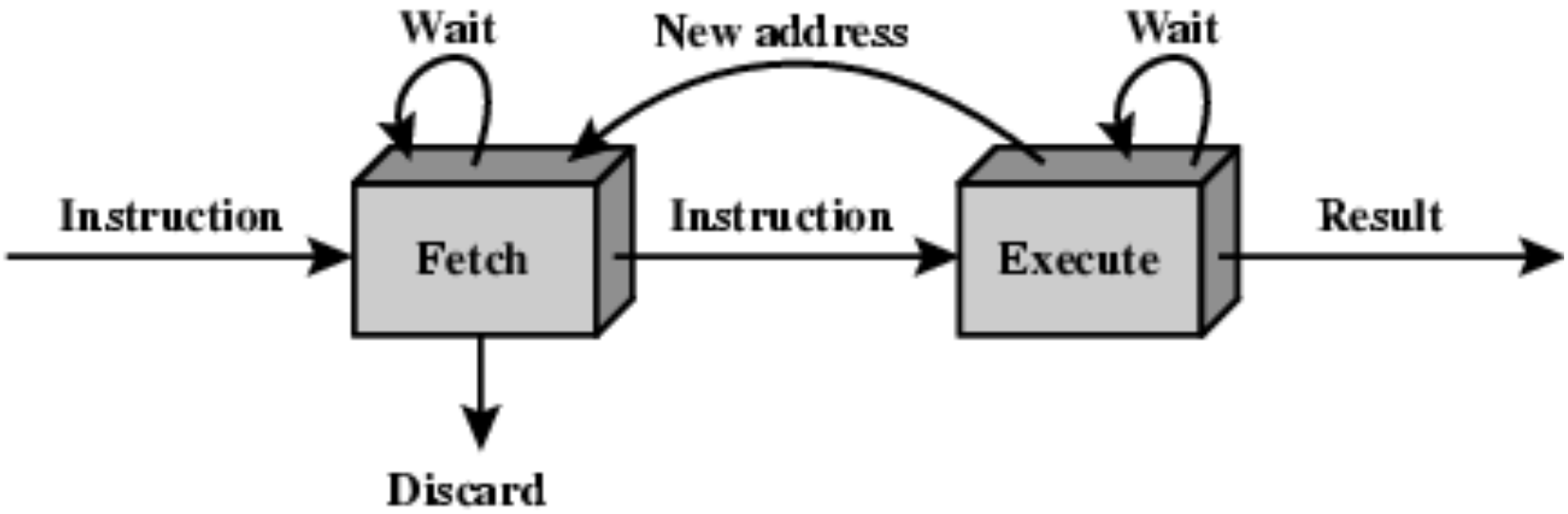
(a) Simplified view

Pipelining

- Disadvantage:
  - **1.** The execution time will generally be longer than the fetch time. Execution will involve reading and storing operands and the performance of some operation. Thus, the fetch stage may have to wait for some time before it can empty its buffer.
  - **2.** A conditional branch instruction makes the address of the next instruction to be fetched unknown. Thus, the fetch stage must wait until it receives the next instruction address from the execute stage. The execute stage may then have to wait while the next instruction is fetched.

Pipelining

- A simple rule for the above problem:
  - When a conditional branch instruction is passed on from the fetch to the execute stage, the fetch stage fetches the next instruction in memory after the branch instruction. Then, if the branch is not taken, no time is lost. If the branch is taken, the fetched instruction must be discarded and a new instruction fetched.

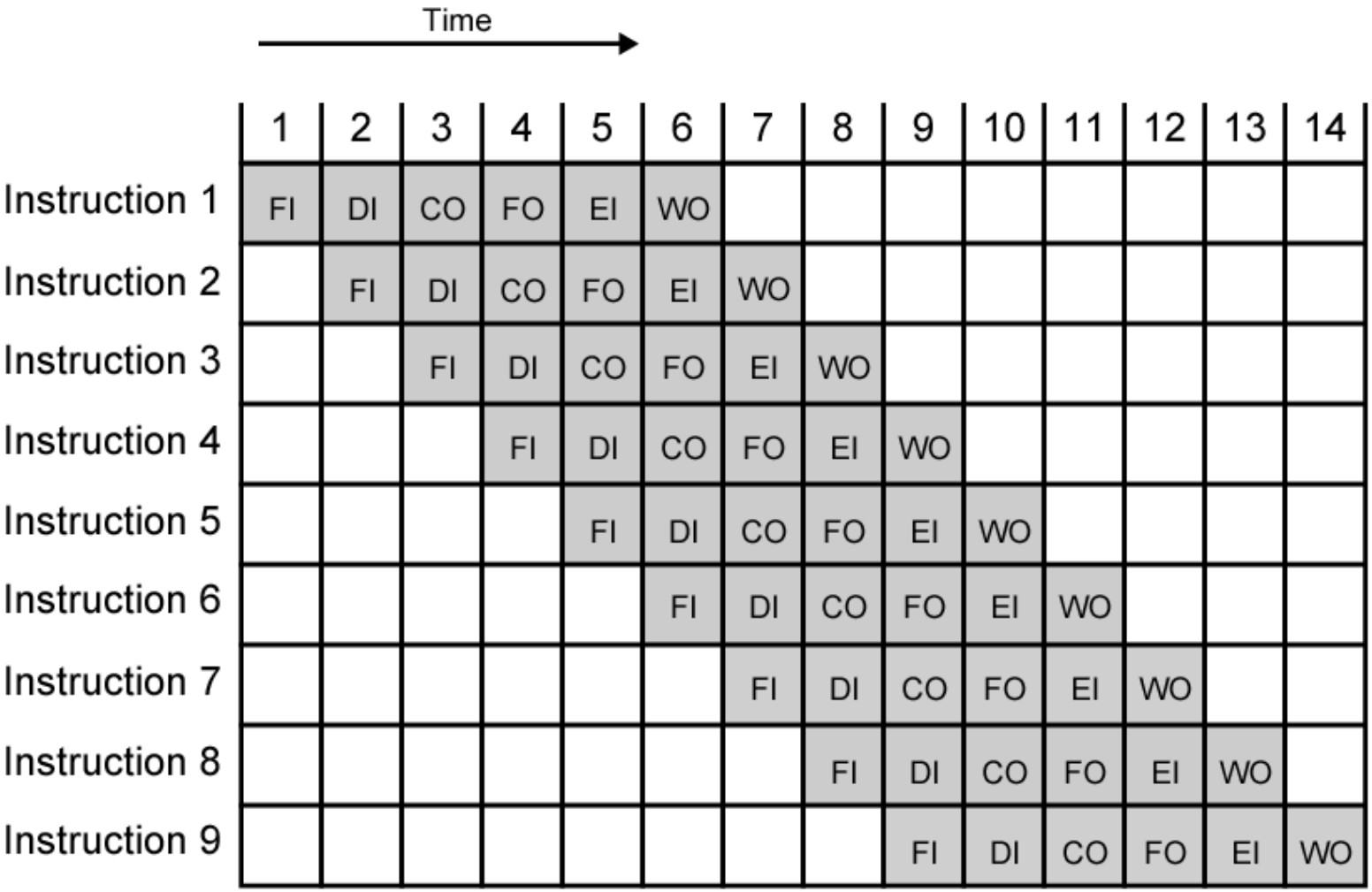


(b) Expanded view

Pipelining

- **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.
- **Fetch operands (FO):** Fetch each operand from memory.
- **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: Instruction Pipeline Operation

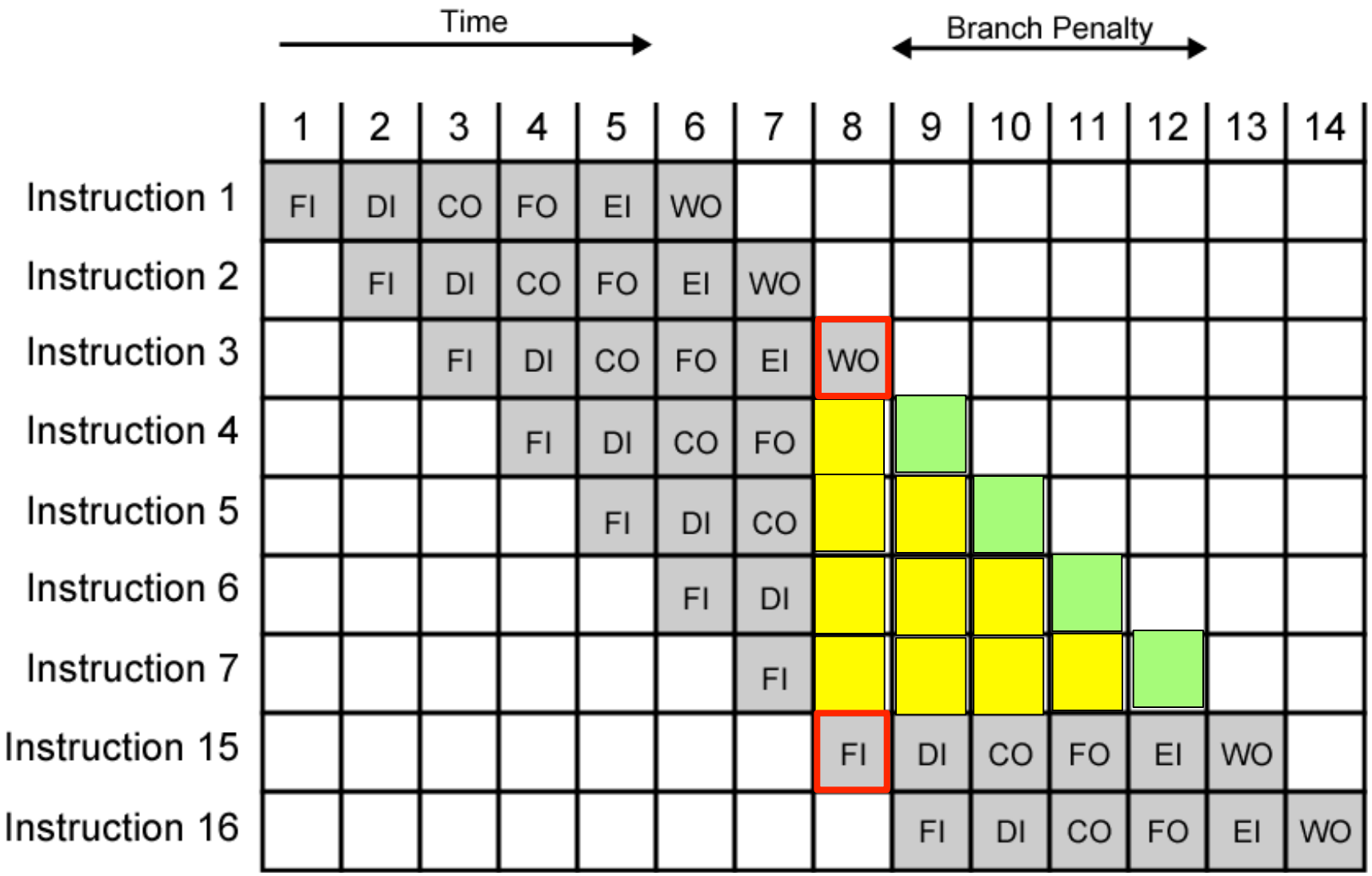


Six-stage pipeline can reduce the execution time for 9 instructions from 54 time units to 14 time units. (Assumption: All stages takes equal duration)

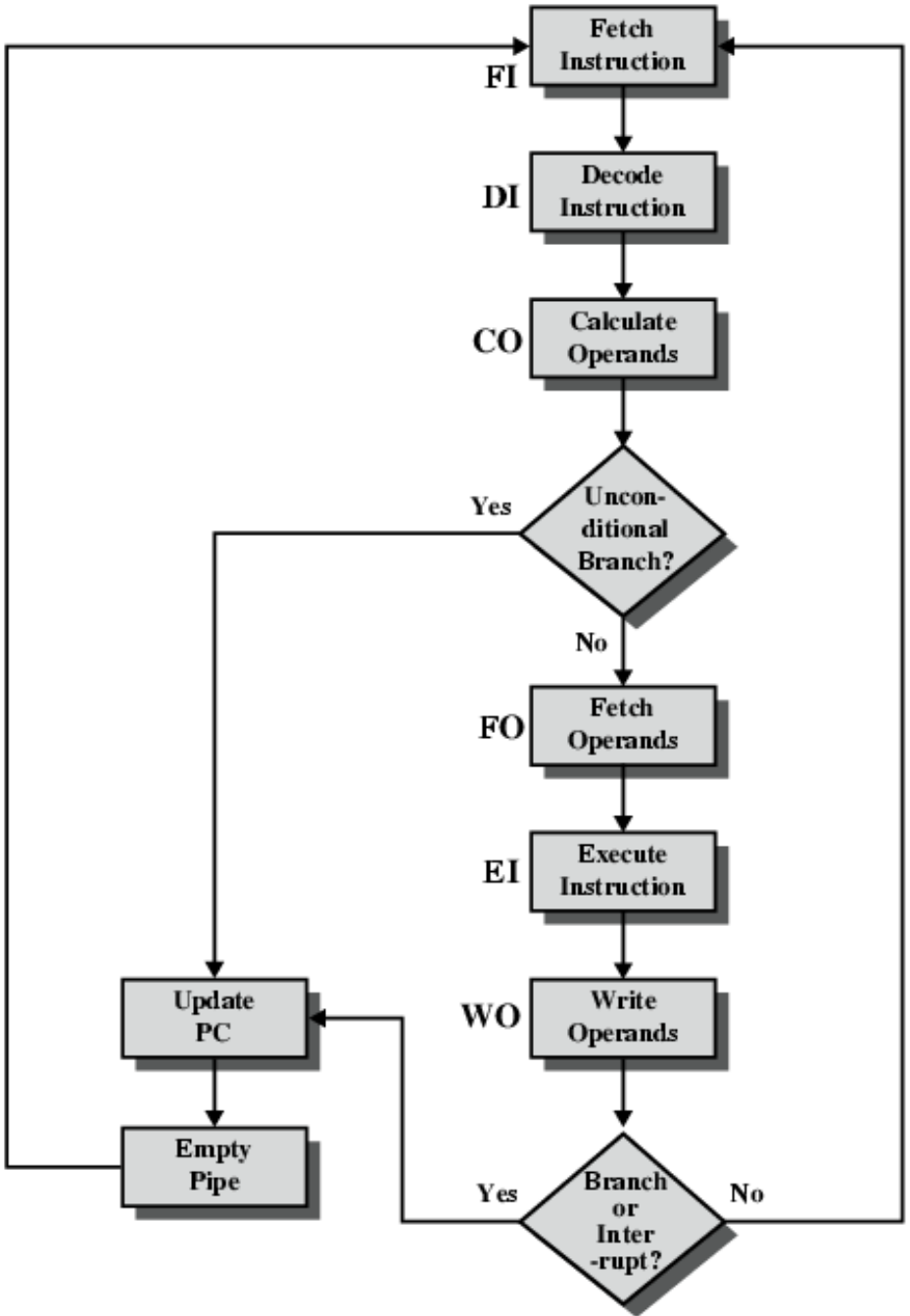
Pipelining: example (conditional branch)

- Assume that instruction 3 is a conditional branch to instruction 15.
- Until the instruction is executed, there is no way of knowing which instruction will come next.
- The pipeline, in this example, simply loads the next instruction in sequence (instruction 4) and proceeds.
- This is not determined until the end of time unit 7. At this point, the pipeline must be cleared of instructions that are not useful.
- During time unit 8, instruction 15 enters the pipeline.
- No instructions complete during time units 9 through 12; this is the performance penalty incurred because we could not anticipate the branch.

The Effect of a Conditional Branch on Instruction Pipeline Operation



Six Stage Instruction Pipeline



Alternative Pipeline Depiction

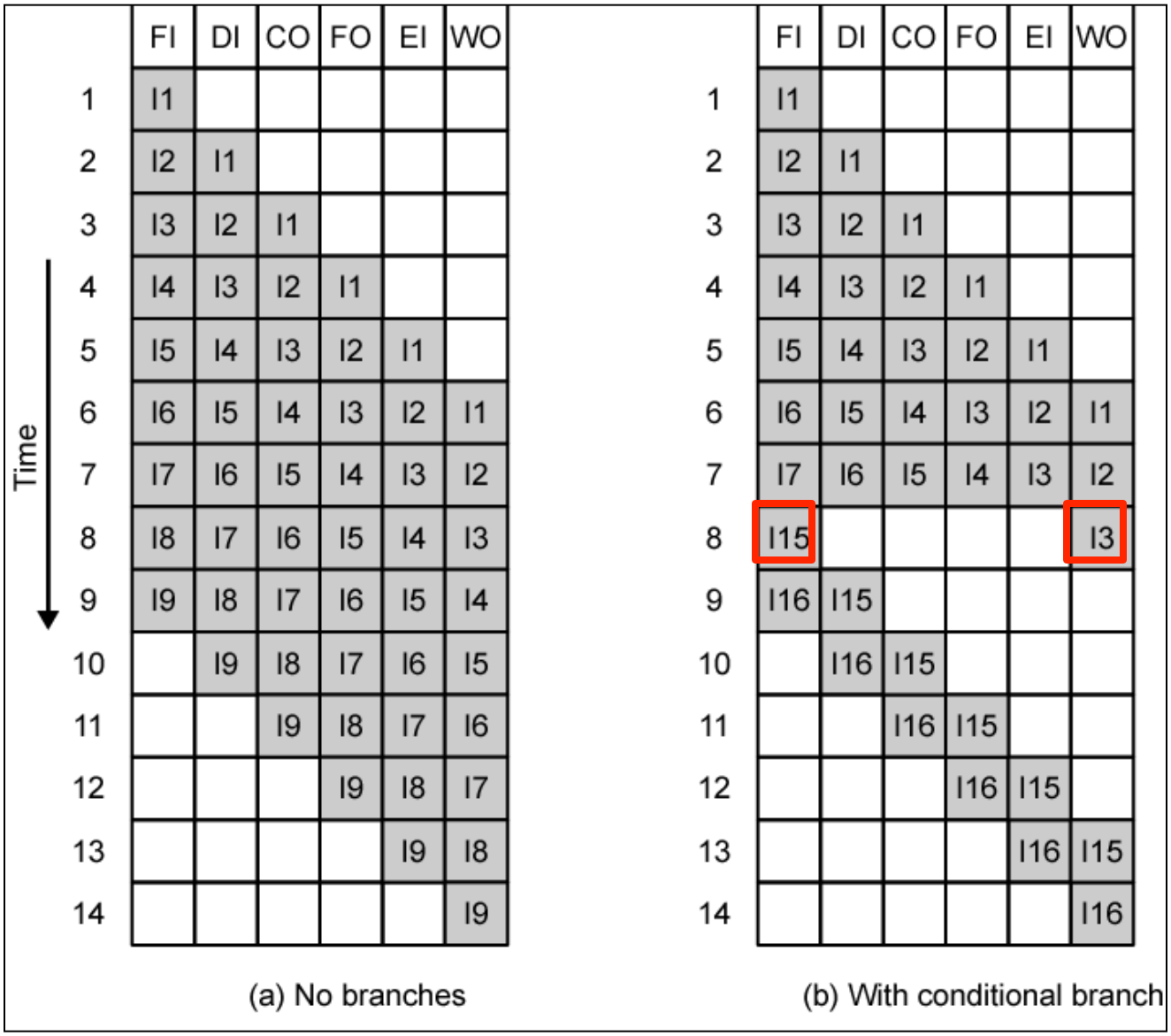
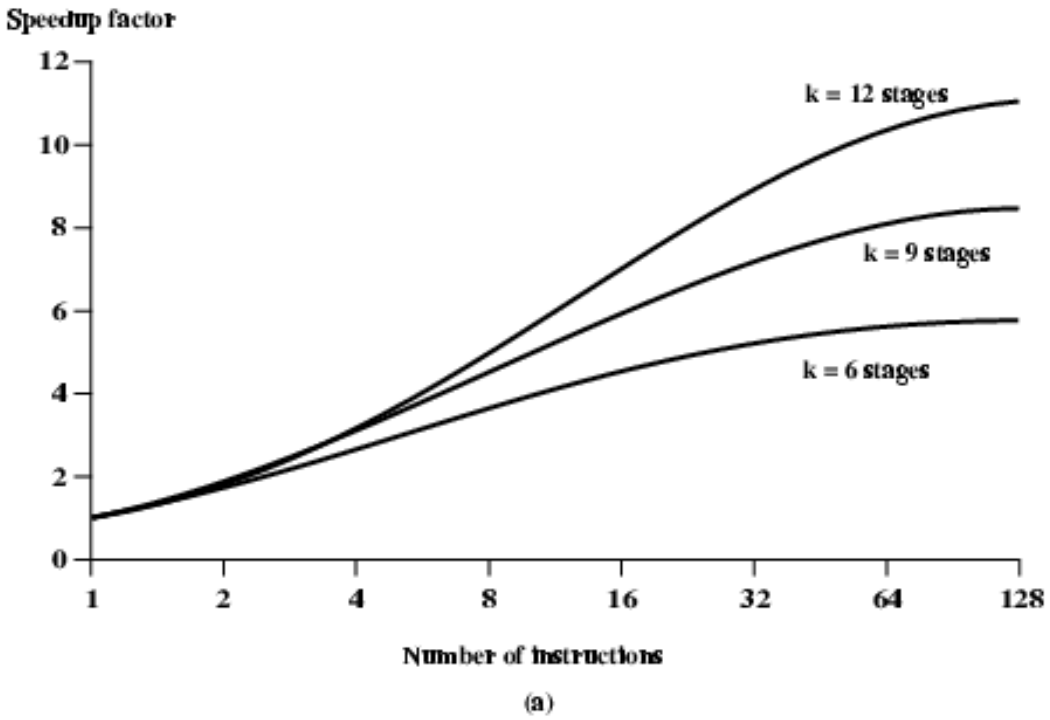
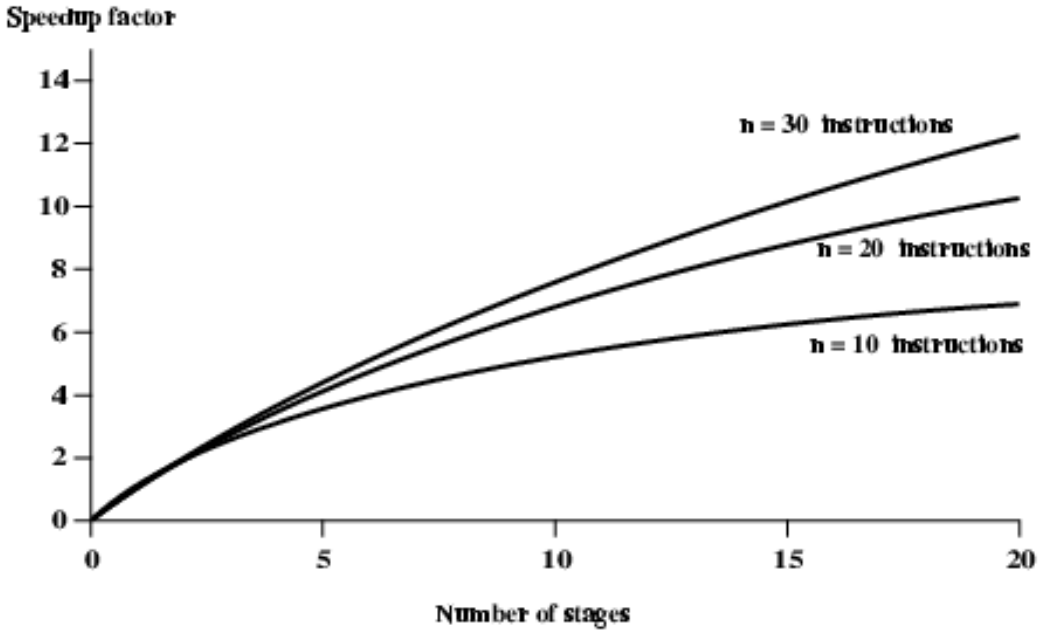


Fig b: The pipeline is full at times 6 and 7. At time 7, instruction 3 is in the execute stage and executes a branch to instruction 15. At this point, instructions I4 through I7 are flushed from the pipeline, so that at time 8, only two instructions are in the pipeline, I3 and I15.

Speedup Factors with Instruction Pipelining



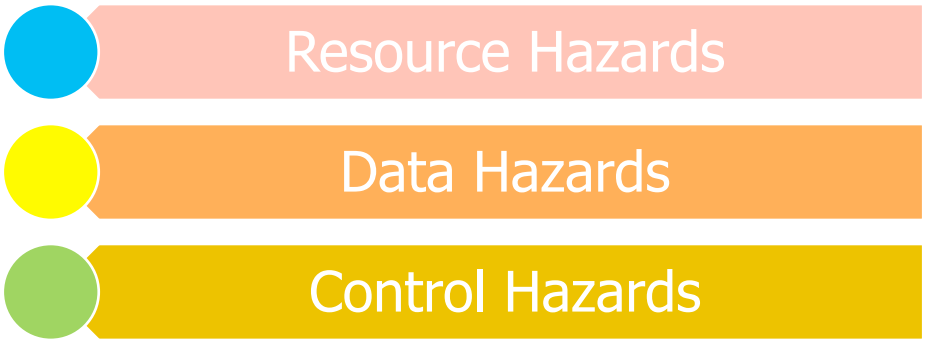
(a)



(b)

Pipeline Hazards

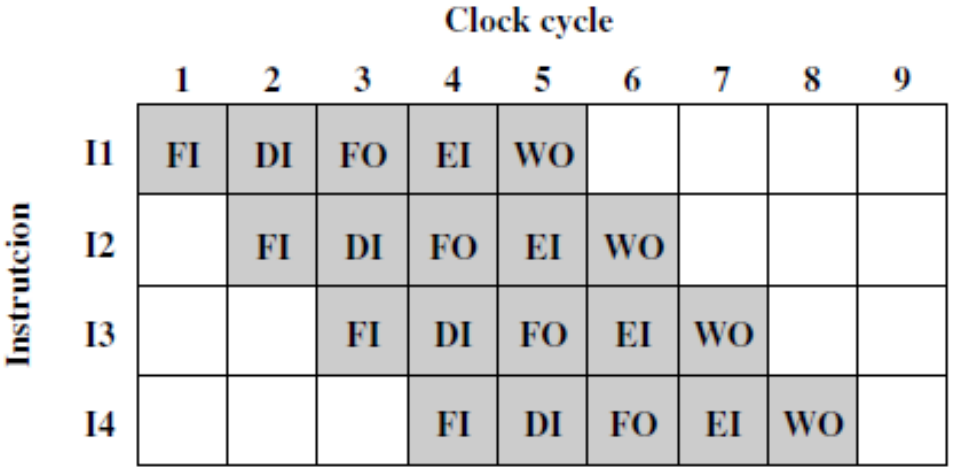
- A **pipeline hazard** occurs when the pipeline, or some portion of the pipeline, must stall because conditions do not permit continued execution. Such a pipeline stall is also referred to as a *pipeline bubble*. There are three types of hazards:



Resource Hazard

- 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.

Resource Hazard: Example

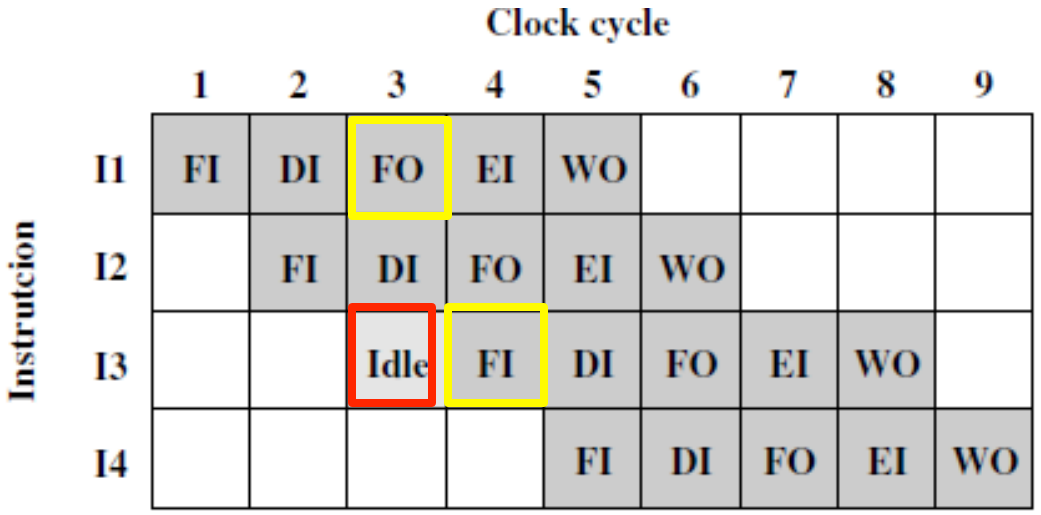


(a) Five-stage pipeline, ideal case

Now assume that main memory has a single port. In this case, an operand read to or write from memory cannot be performed in parallel with an instruction fetch.

Assume that the source operand for instruction I1 is in memory, rather than a register. Therefore, the fetch instruction stage of the pipeline must idle for one cycle before beginning the instruction fetch for instruction I3.

Assume that all other operands are in registers.



(b) I1 source operand in memory

Data Hazard

- A data hazard occurs when there is a conflict in the access of an operand location. In general terms, we can state the hazard in this form: 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.

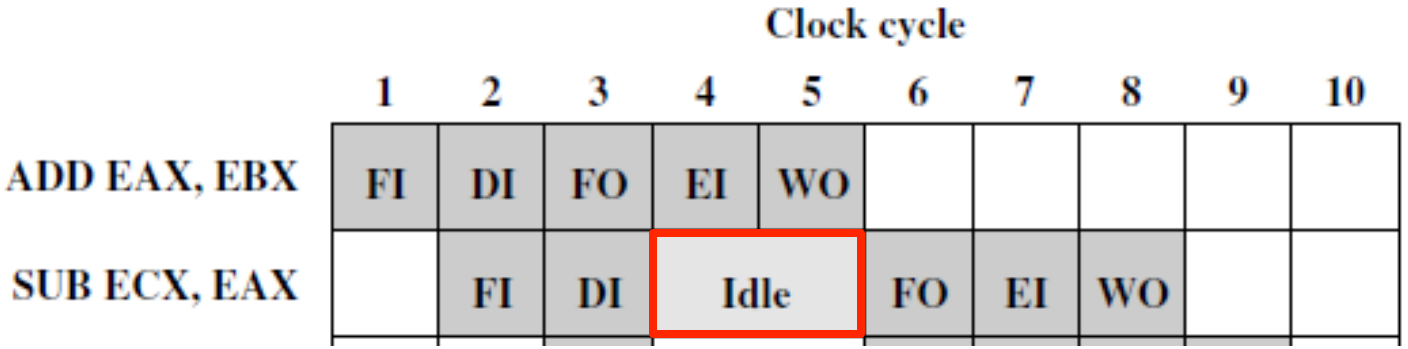
Data Hazard: Example

- As an example, consider the following x86 machine instruction sequence:

  ADD EAX, EBX /* EAX = EAX + EBX

  SUB ECX, EAX /* ECX = ECX - EAX

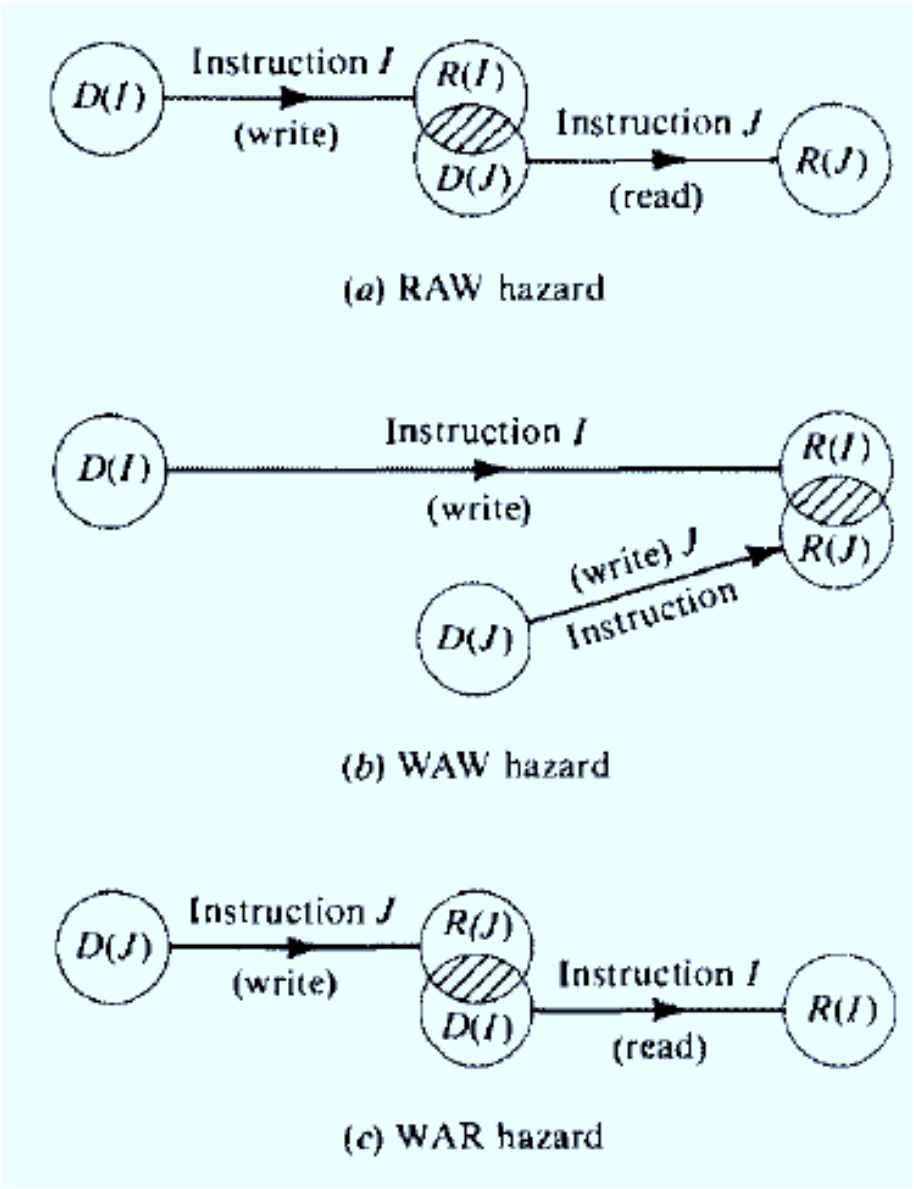
- The ADD instruction does not update register EAX until the end of stage 5, which occurs at clock cycle 5.
- But the SUB instruction needs that value at the beginning of its stage 2, which occurs at clock cycle 4.
- To maintain correct operation, the pipeline must stall for two clocks cycles.
- Thus, a data hazard results in inefficient pipeline usage.



Data Hazard: Types

- **Read after write (RAW), or true dependency:** An instruction modifies a register or memory location and a succeeding instruction reads the data in that memory or register location. A hazard occurs if the read takes place before the write operation is complete.
- **Write after read (WAR), or antidependency:** An instruction reads a register or memory location and a succeeding instruction writes to the location. A 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. A hazard occurs if the write operations take place in the reverse order of the intended sequence.

Data Hazard: Types



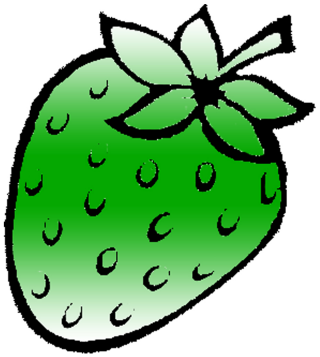
Control Hazards

- A control hazard, also known as a *branch hazard,* occurs when the pipeline makes the wrong decision on a branch prediction and therefore brings instructions into the pipeline that must subsequently be discarded.

Co-Processors

- A coprocessor is a computer processor used to supplement the functions of the primary processor (the CPU). Operations performed by the coprocessor may be floating point arithmetic, graphics, signal processing, string processing, encryption or I/O Interfacing with peripheral devices.

STRAWBERRY



/strawberrydevelopers

/strawberry_app

For more visit:

Strawberrydevelopers.weebly.com