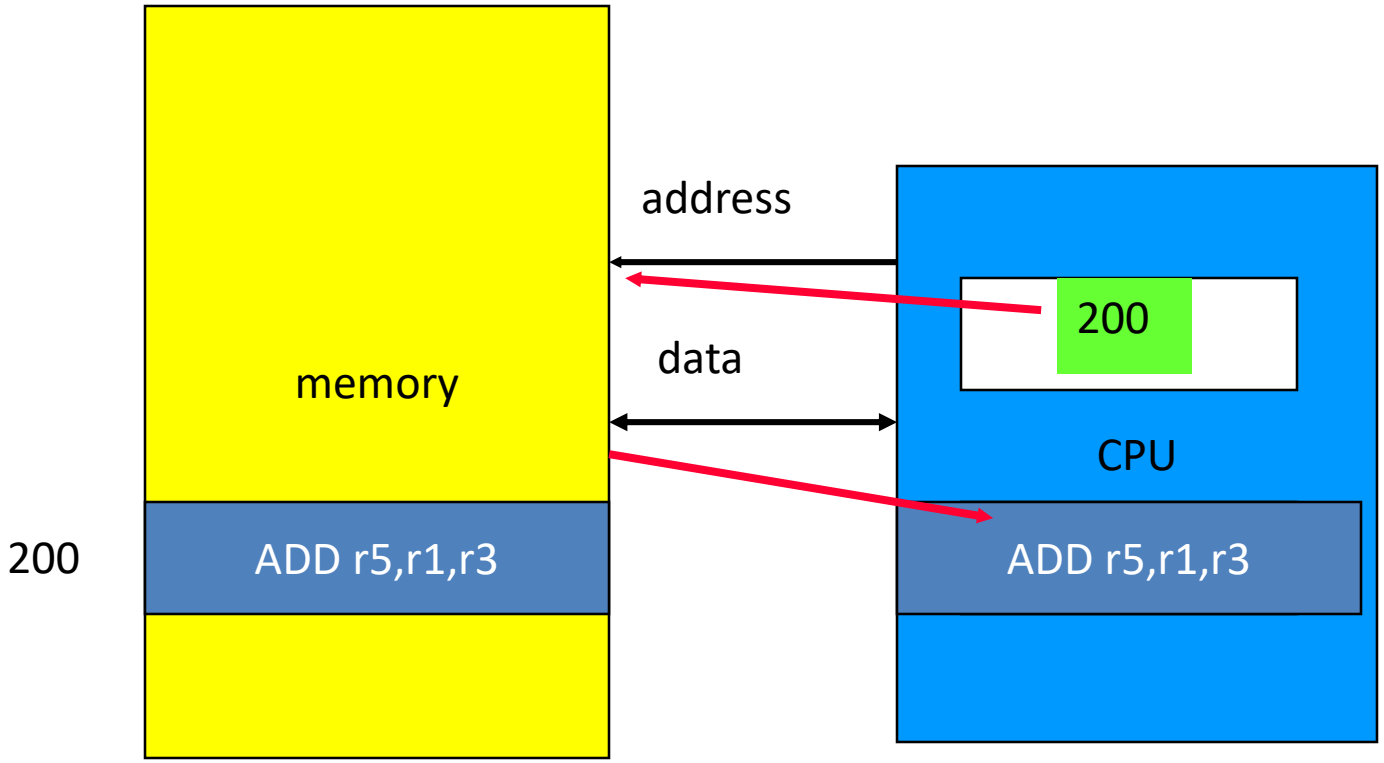
L2 (CHAPTER 5)

Programming in Assembly
Part 1: Computer Organization

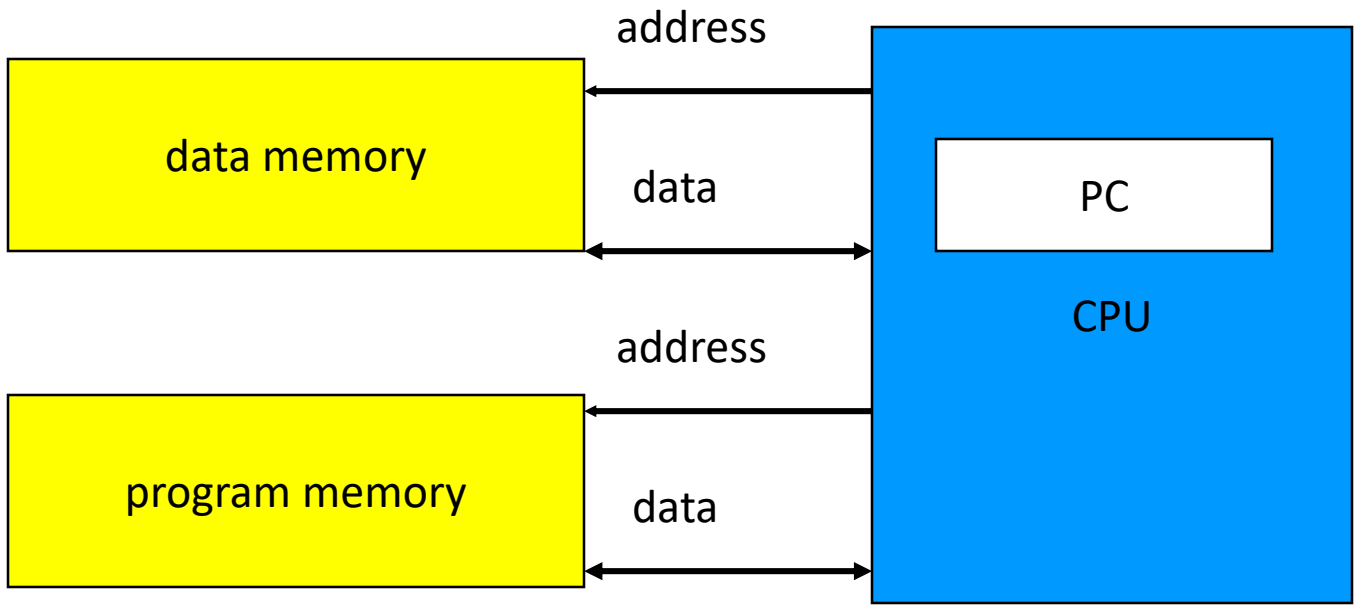
von Neumann Architecture

- Memory holds data, instructions.
- CPU fetches instructions from memory.
- CPU registers: program counter (PC), instruction register (IR), general-purpose registers, etc.

CPU + Memory



Harvard Architecture

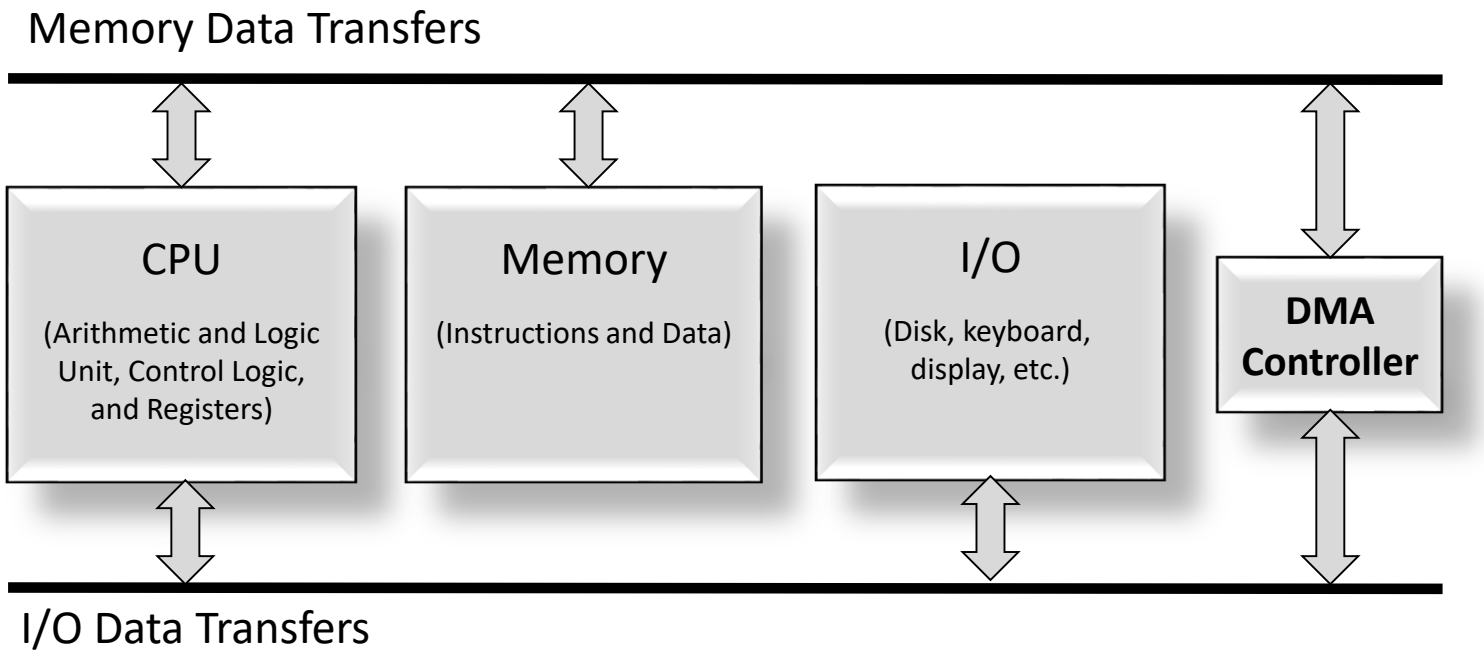


von Neumann vs. Harvard

- Harvard allows two simultaneous memory fetches.
- Most DSPs use Harvard architecture for streaming data:
  - greater memory bandwidth;
  - more predictable bandwidth.
- Harvard can’t use self-modifying code.
  - Instruction memory is read-only

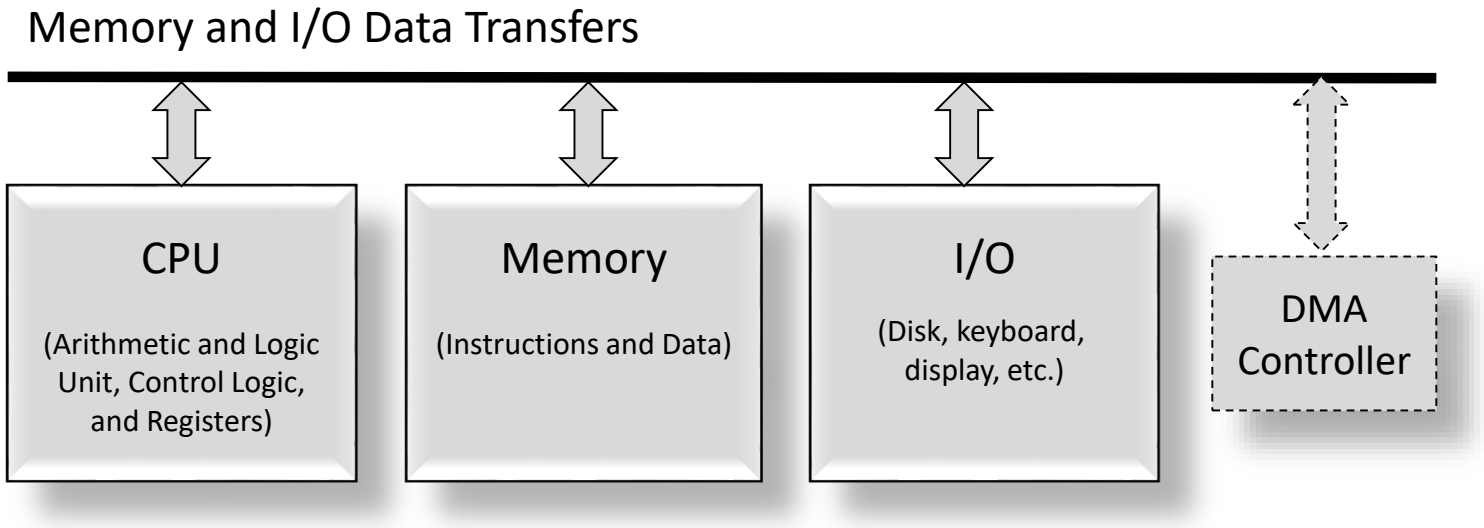
A Typical Desktop Processor (von Neumann Arch. as example)

- Three major components:
  - (1) *CPU* , where operations are performed,
  - (2) *memory* , where data is stored, and
  - (3) *input/output (I/O) system*, to gather input data or to output results.
- The three units are interconnected by groups of wires called buses.
- Data transferred between memory and I/O devices normally goes through and is coordinated by the CPU. Direct Memory Access (DMA) controller can be added to perform these transfers without involving the CPU



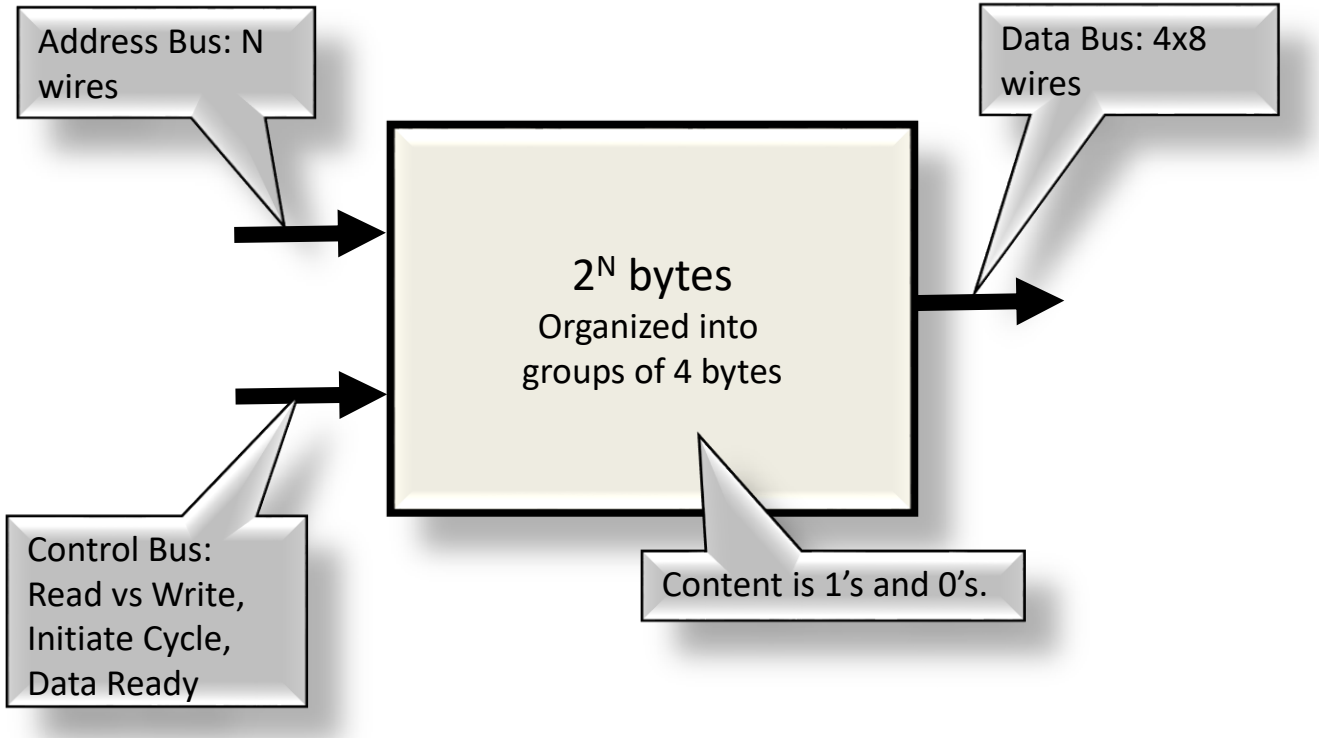
A Typical Embedded Processor (von Neumann Arch. as Example)

- Memory-mapped I/O
  - A 32-bit memory address is capable of referencing 4 GB (2^{32} bytes) of information
  - A portion of the address space is reserved for and decoded by the I/O devices. This eliminates the need for separate I/O instructions since all of the regular instructions for accessing memory can then be used to access I/O devices.
- DMA controller is often not present, since most embedded applications do not require very high data transfer rates



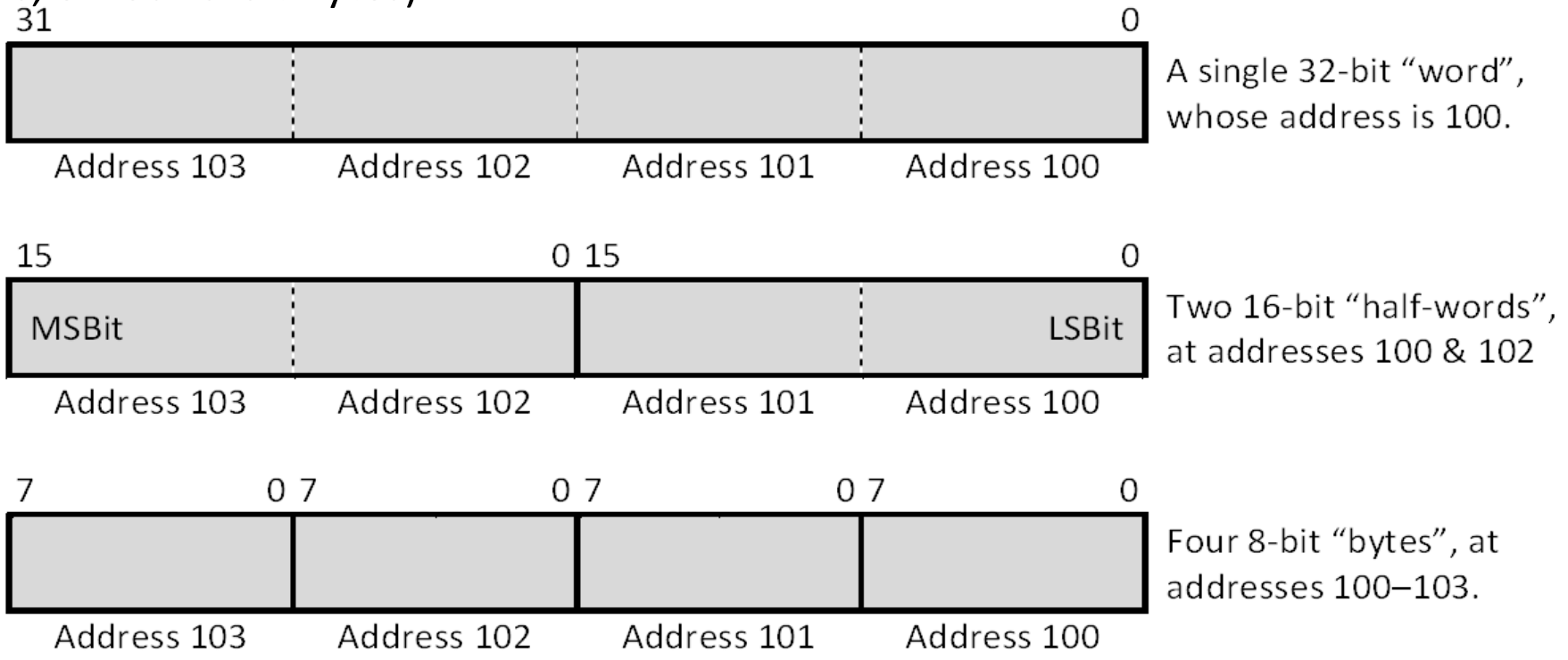
Memory

- Information flows between the CPU and memory over three sets of wires called the address bus , the data bus , and the control bus.
  - The CPU places information on the address bus to tell the memory which data to access
  - uses the control bus to determine the direction of information transfer and to coordinate timing
  - The data itself is transferred over the data bus
- Address bus has N wires, where N is # bits in each address (8,16,32,64…)
- Data bus has 4x8 wires, hence each data transfer is 4 Bytes (also called 1 word)



Memory Byte Ordering

- Two possible byte orderings: “little endian” and “big endian”
  - Little-endian: the LSB (Least Significant Byte) is stored at the lowest address.
  - Big-endian: the LSB (Least Significant Byte) is stored at the highest address.
  - Intel processors use Little-Endian; ARM processors can be configured as either Little- or Big-endian.
- Below are examples of Little Endian ordering (A 32-bit entity can contain one 32-bit word, or two 16-bit half-words, or four 8-bit Bytes)



Data Alignment

- Assume a byte-addressable memory with a data bus that is 32 bits (4 bytes) wide
- Consider 16 bytes of memory (addresses 0 to 15) arranged as four 32-bit words (4 bytes each)

Address 15	Address 14	Address 13	Address 12
Address 11	Address 10	Address 9	Address 8
Address 7 (MSbyte)	Address 6	Address 5	Address 4 (LSbyte)
Address 3	Address 2	Address 1	Address 0

Well-aligned: each word begins on a mod-4 address, which can be read in a single memory cycle

Address 15	Address 14	Address 13	Address 12
Address 11	Address 10	Address 9 (MSbyte)	Address 8
Address 7	Address 6 (LSbyte)	Address 5	Address 4
Address 3	Address 2	Address 1	Address 0

Ill-aligned: a word begins on address 6, not a mod-4 address, which can be read in 2 memory cycles

The first read cycle would retrieve 4 bytes from addresses 4 through 7; of these, the bytes from addresses 4 and 5 are discarded, and those from addresses 6 and 7 are moved to the far right;
The second read cycle retrieves 4 bytes from addresses 8 through 11; the bytes from addresses 10 and 11 are discarded, and those from addresses 8 and 9 are moved to the far left;
Finally, the two halves are combined to form the desired 32-bit operand:

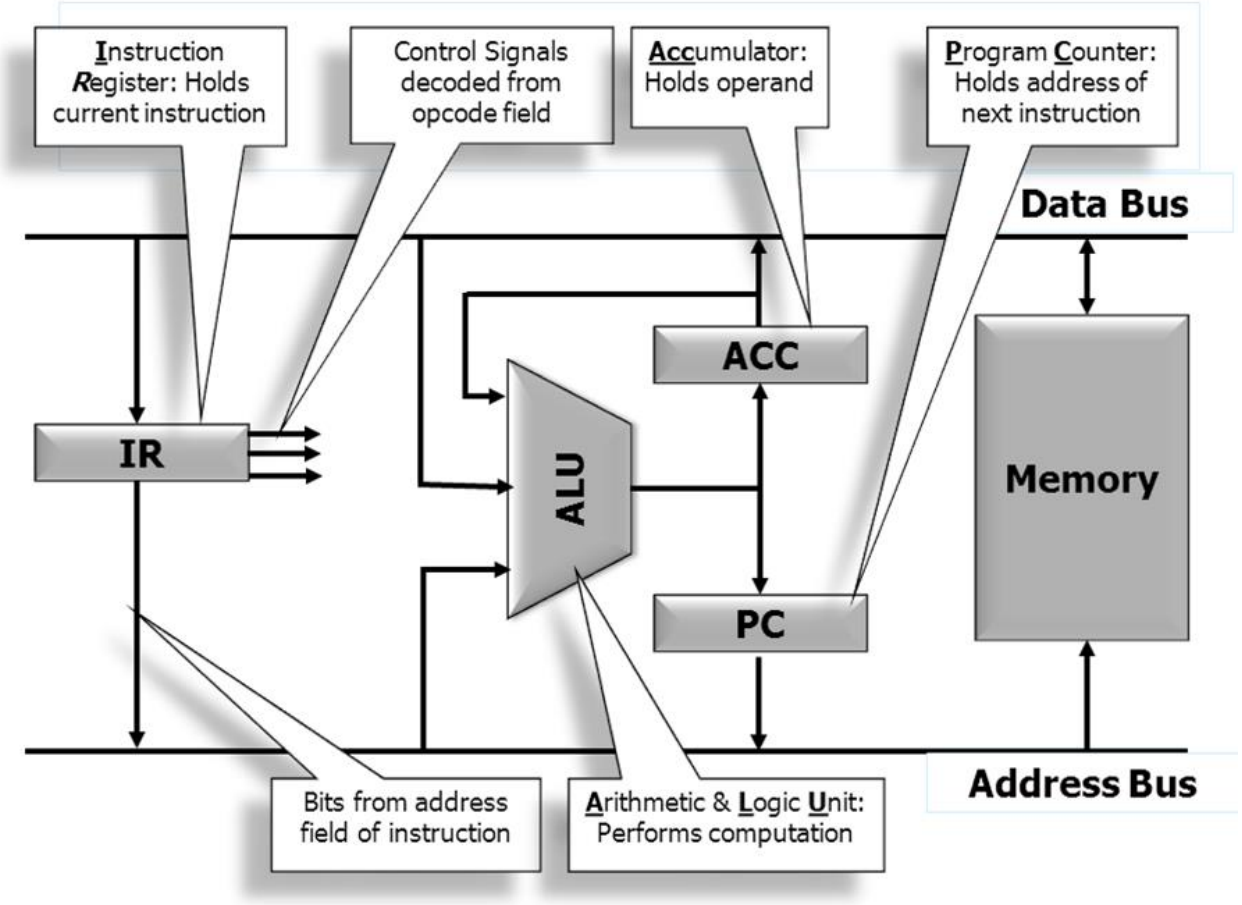
		Address 7	Address 6 (LSbyte)

Address 9 (MSbyte)	Address 8		

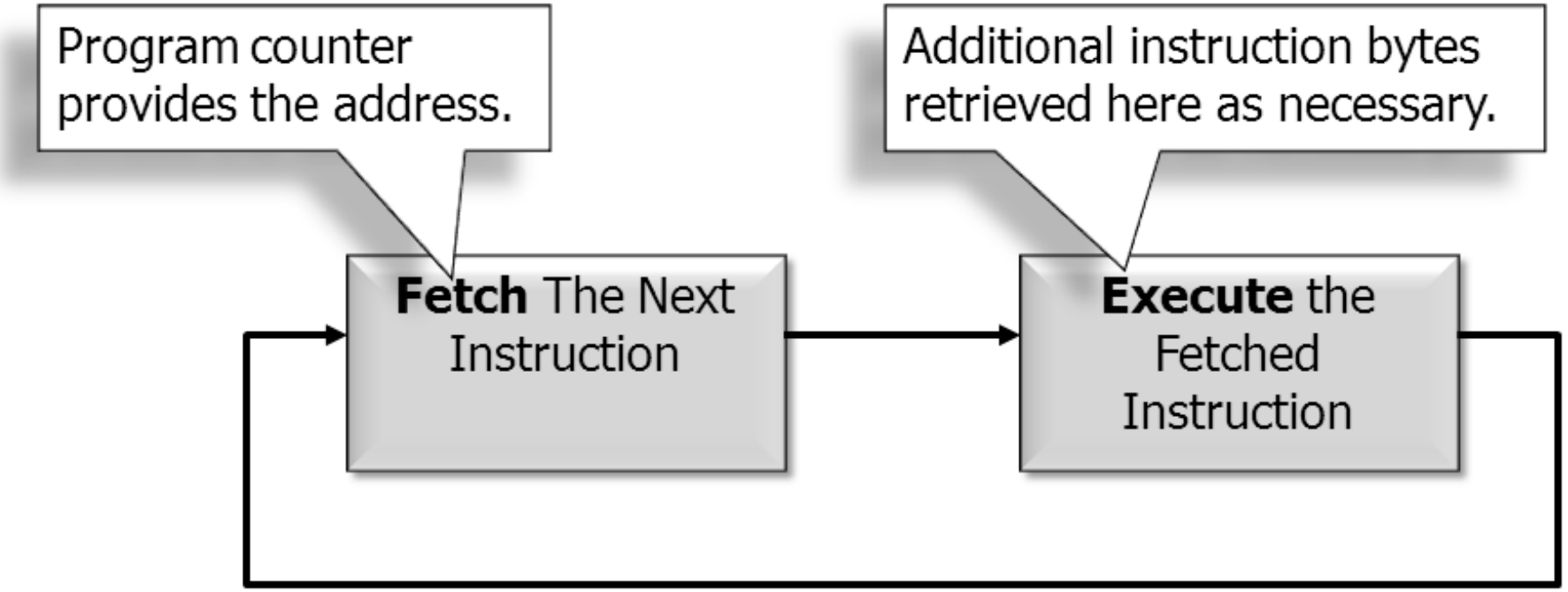
Address 9 (MSbyte)	Address 8	Address 7	Address 6 (LSbyte)

Single Accumulator Architecture

- A CPU consists of several components, including the arithmetic and logic unit (ALU) , a set of registers, and a control unit . One or more data and address buses interconnect these components.
- Instruction registers hold the current instruction; a program counter (PC) to specify the address of the next instruction to be executed, and others
- Data registers hold the operands and the result. In the simplest configuration, a single data register called the accumulator (ACC) is used



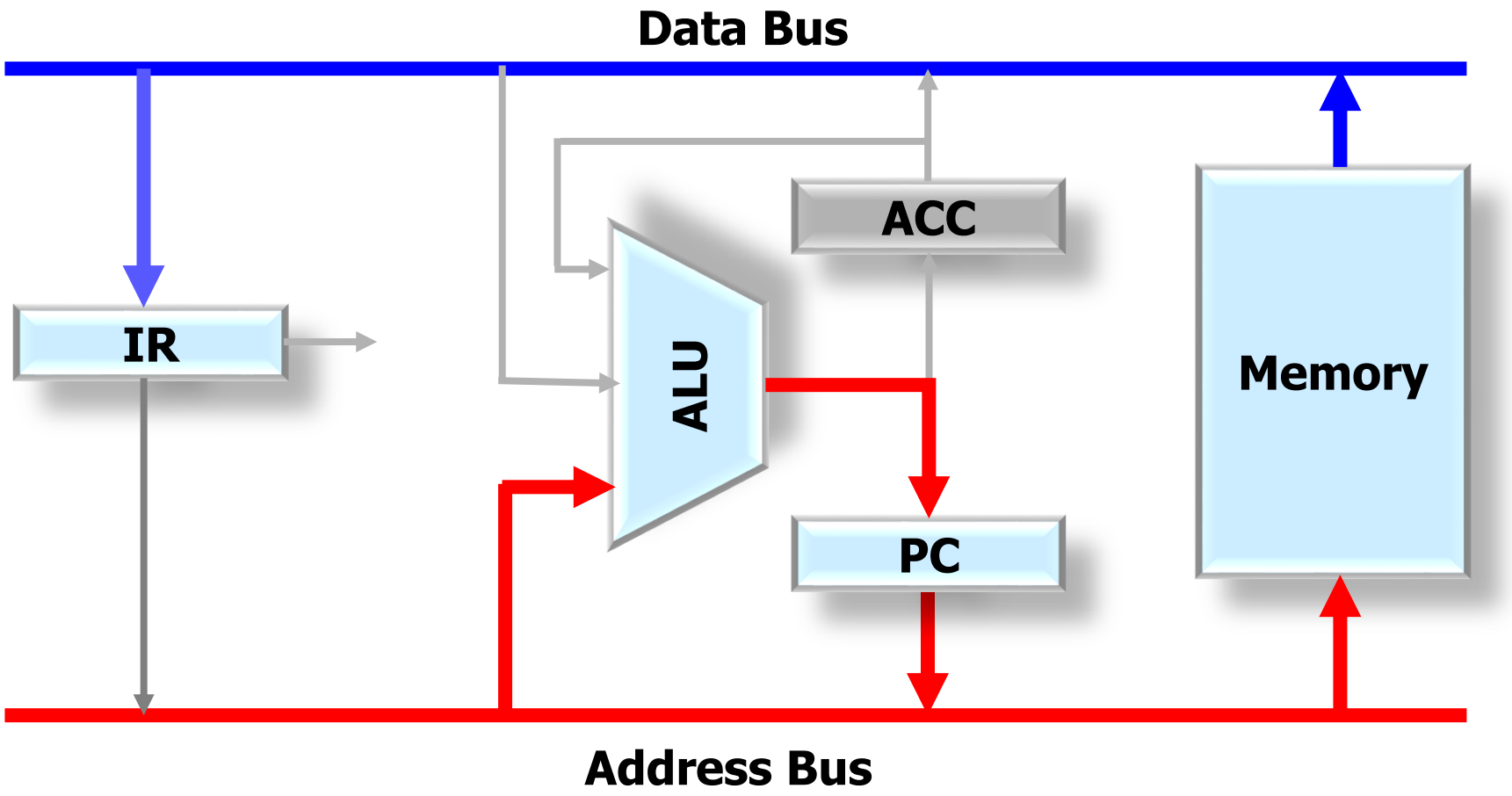
The Fetch-Execute Cycle



(There is typically an instruction decode stage that is omitted here for simplicity)

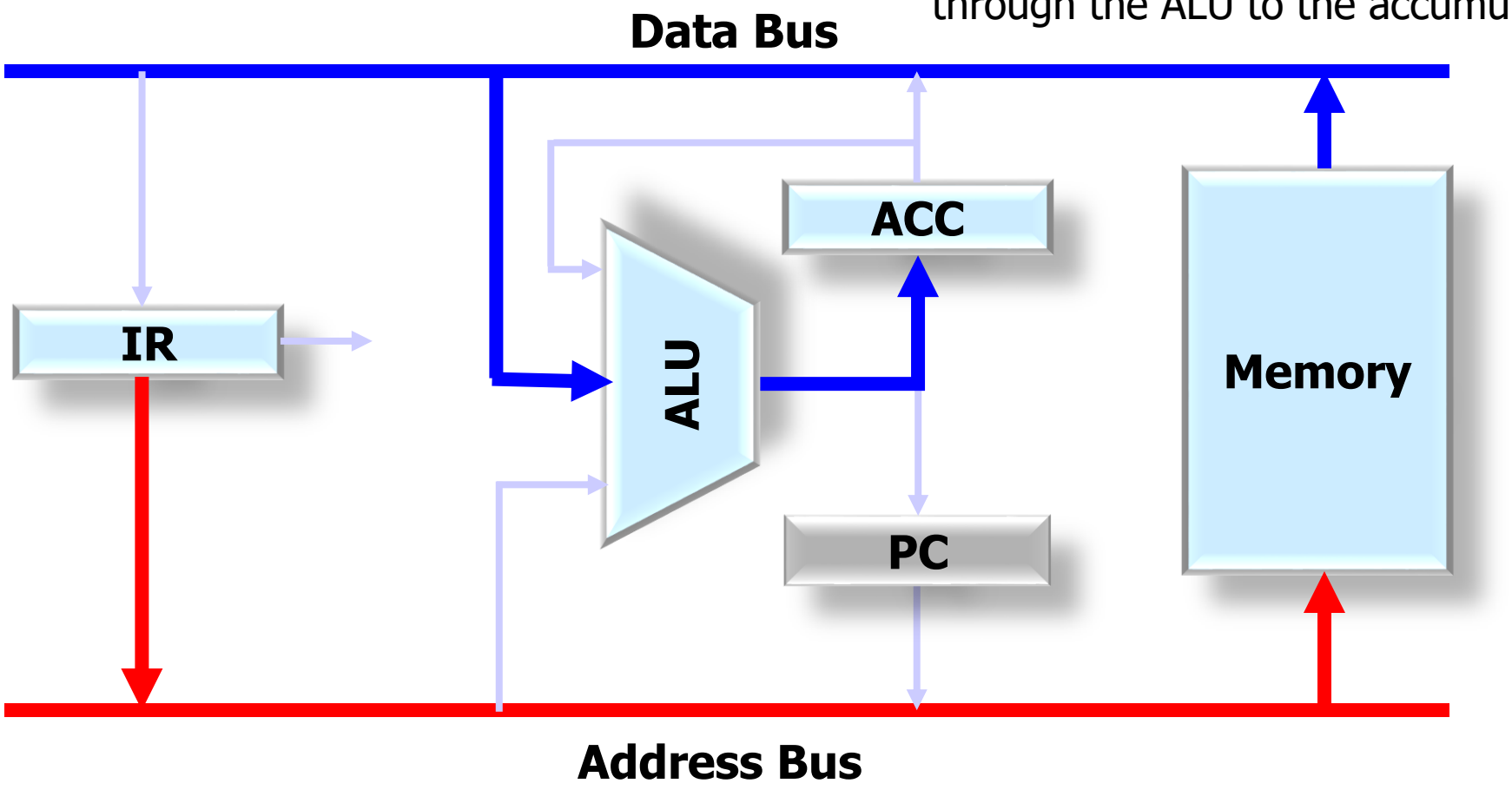
Instruction Fetch

1. Memory_Address_Bus ← Program_Counter
2. Start Memory Read Operation
3. Increment Program_Counter
4. Wait for Memory Read to Complete
5. Instruction_Register ← Memory_Data_Bus
6. Go to execute phase.



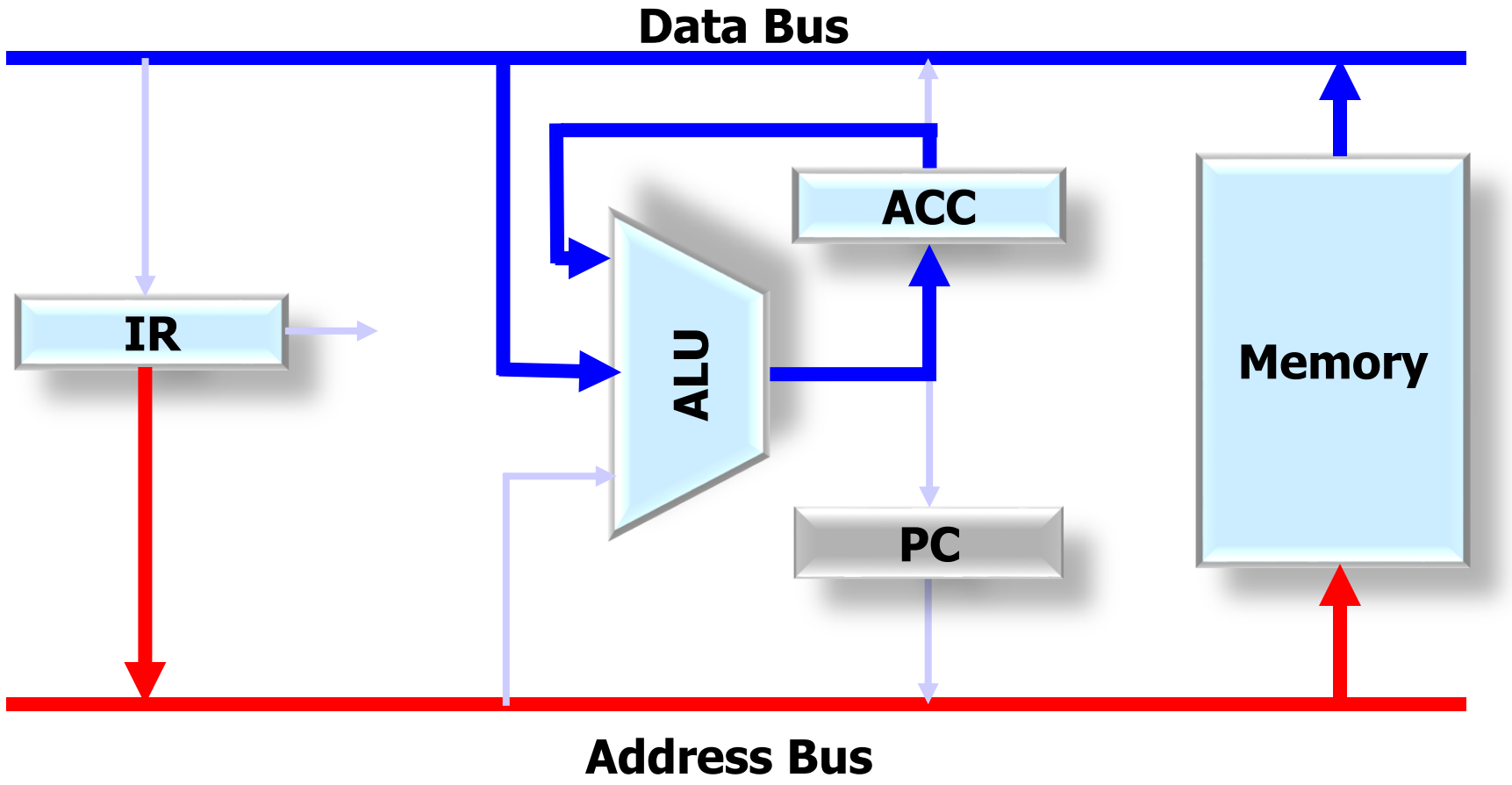
Execution Phase: Load ACC

An instruction that loads data from memory into the accumulator takes the address of the memory operand from the instruction register, places it on the address bus, initiates a memory read operation, and then transfers the memory data through the ALU to the accumulator.



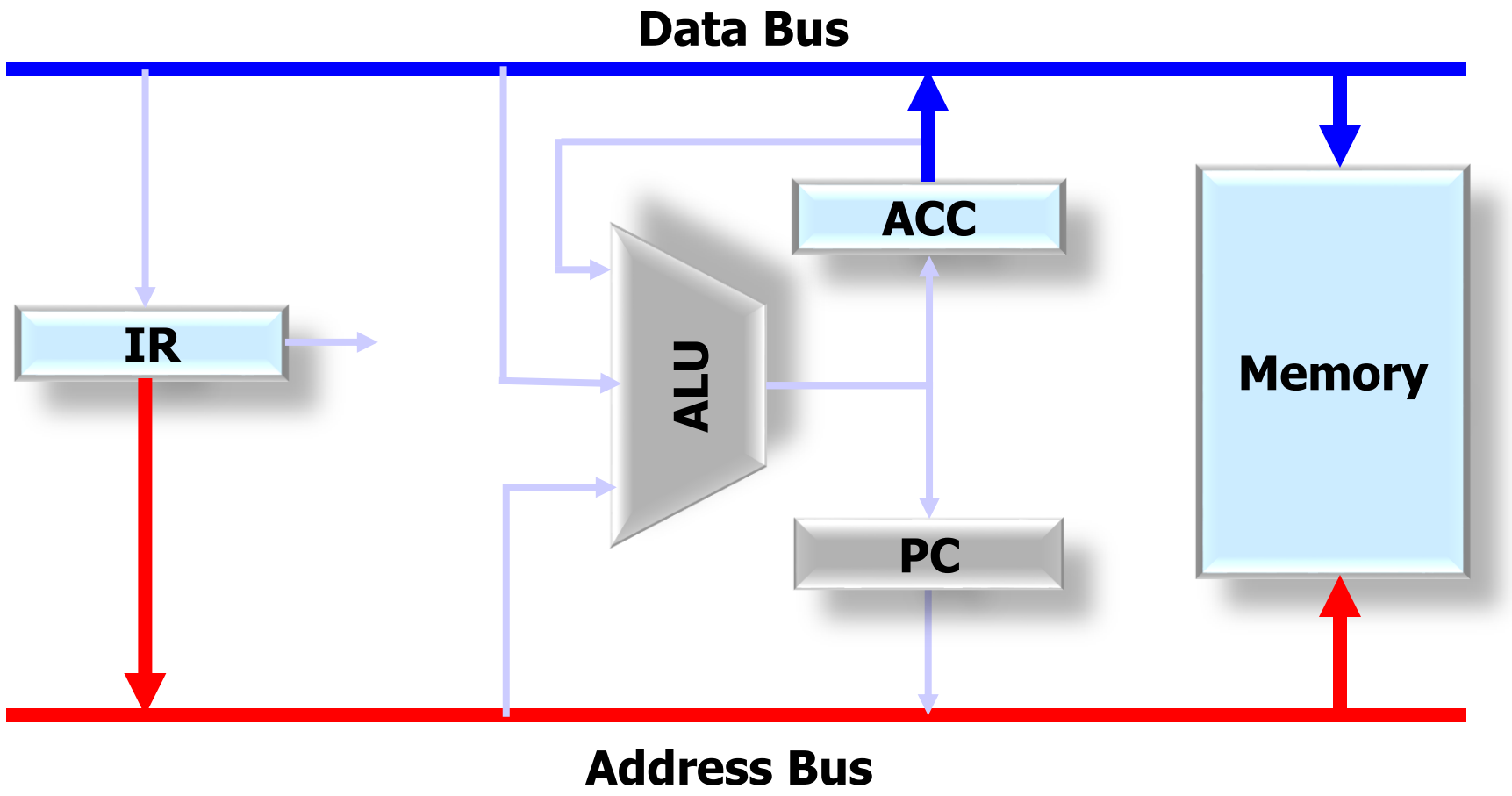
Execution Phase: ADD

Once an operand has been loaded into the accumulator, a subsequent instruction can perform an addition between it and a second operand, taking one operand from memory, a second from the accumulator



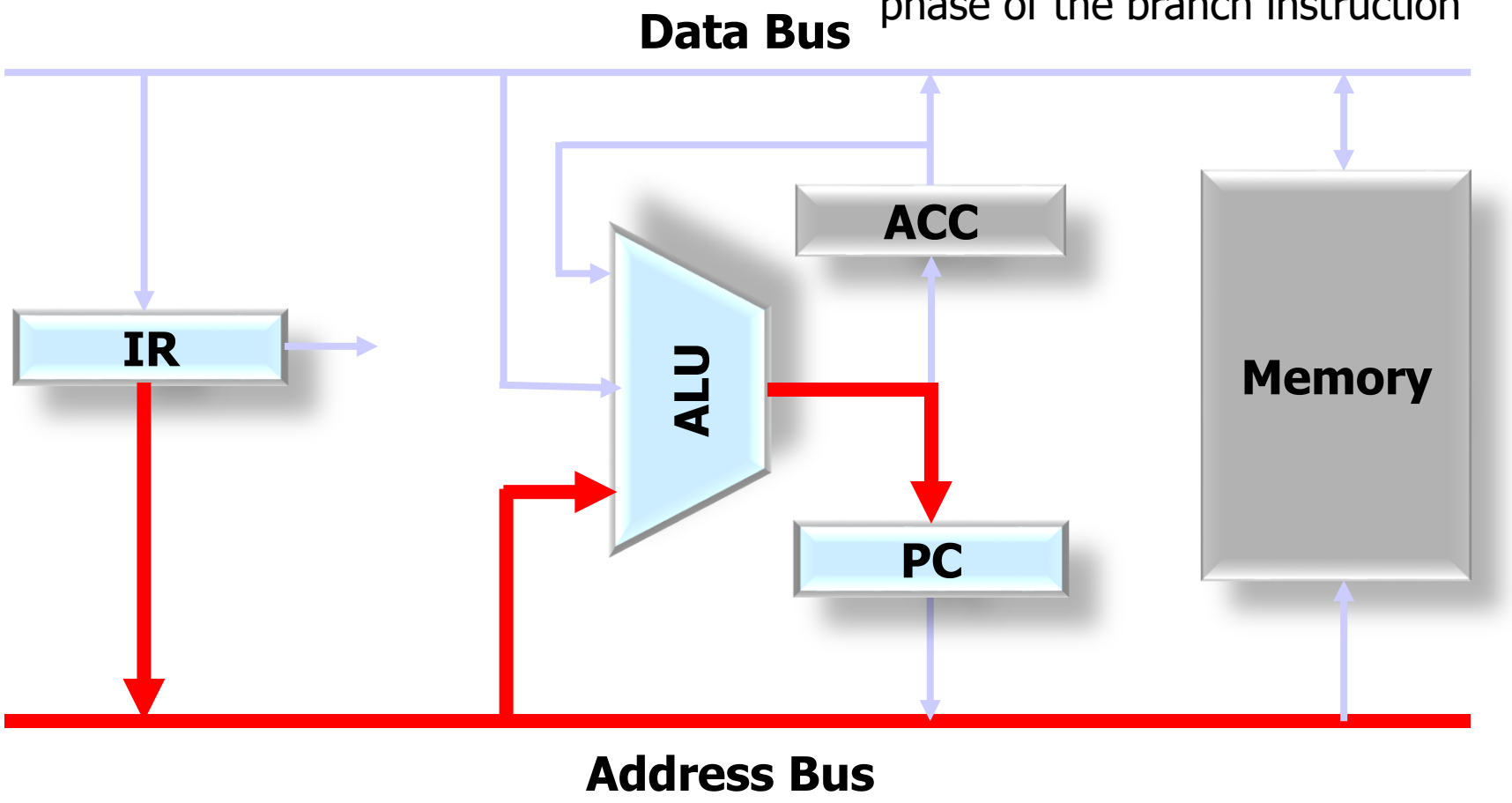
Execution Phase: Store ACC

Replaces content of ACC with the result of ADD.



Execution Phase: Branch

A branch instruction loads a new address into the PC during the execute phase. This address may have been stored as a constant, as a value retrieved from a register or read from memory, or as the result of some calculation performed during the execute phase of the branch instruction



result ← op1 + op2

- The past slides illustrate the single-accumulator machine, where an ALU instruction can perform arithmetic operation between register and memory in a single cycle;
- Modern CPUs are typically register machines, where an ALU instruction can only perform arithmetic operation between two registers; separate load/store instructions are used to transfer data between register and memory.

Single Accumulator Architecture:

ACC ← MEM[adrs_of_op1]
ACC ← ACC + MEM[adrs_of_op2]
MEM[adrs_of_result] ← ACC

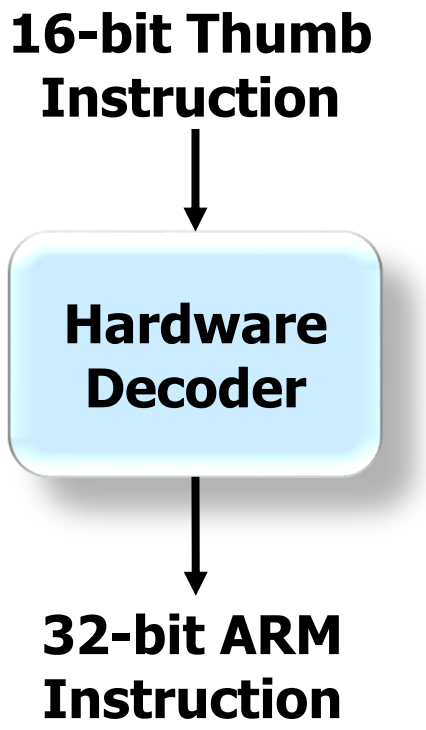
Load/Store Architecture(almost all modern CPUs):

REG[R1] ← MEM[adrs_of_op1] ;Copy 1st operand from memory into register ‘R1'
REG[R2] ← MEM[adrs_of_op2] ;Copy 1st operand from memory into register ‘R2'
REG[R3] ← REG[R1] + REG[R2] ;Perform the addition, leaving the result in a 3rd register
MEM[adrs_of_result] ← REG[R3] ;Store contents of register ‘R3' into memory

The ARM Processor Family

Three Instruction Sets

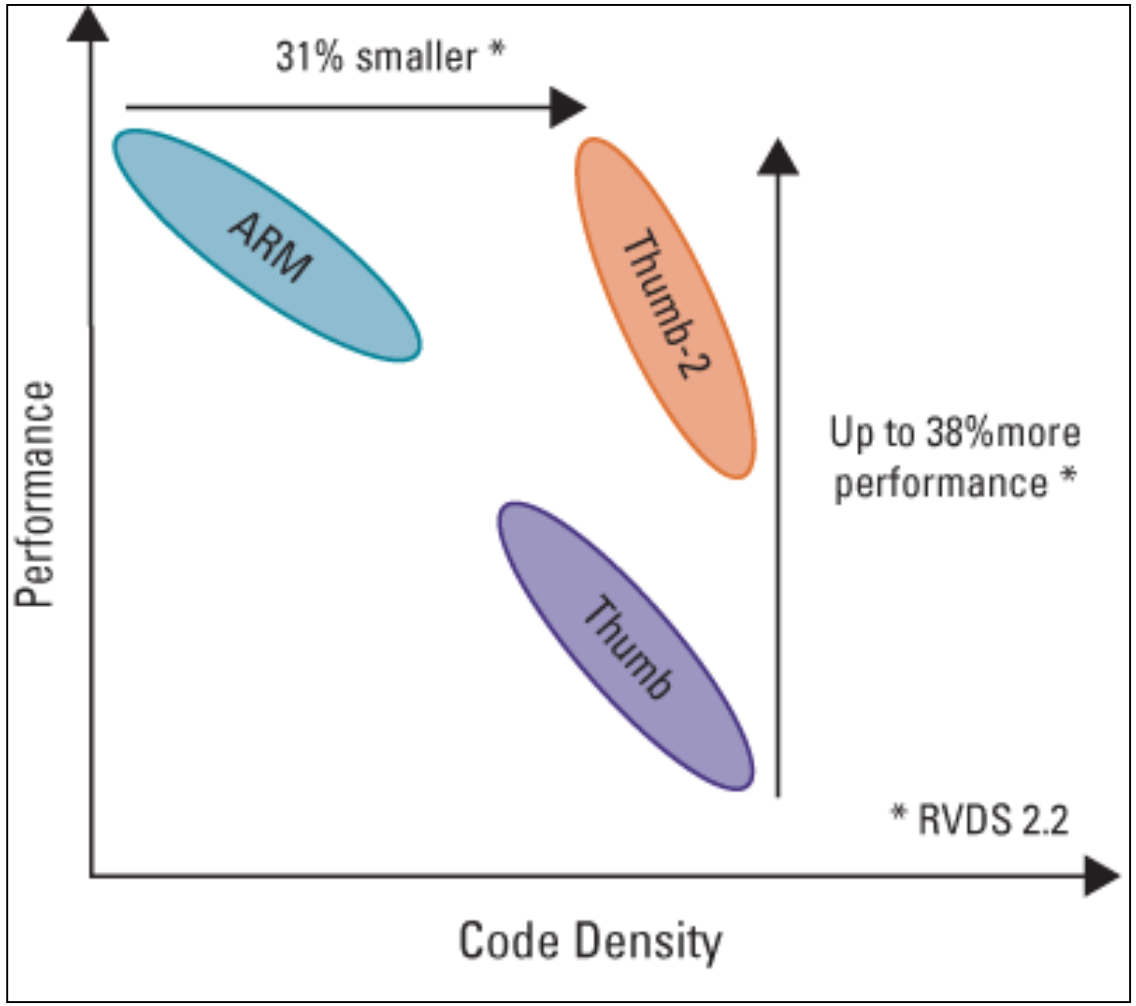
- ARM Instruction Set
  - Instructions are 32 bits wide
  - Original RISC (lots of parallelism)
  - “Load/Store” Architecture
- Thumb Instruction Set
  - Subset of ARM instructions, some restrictions
  - Instructions are 16 bits wide
  - Less parallelism, longer instruction sequences
  - but total code size is 30% smaller
- Jazelle Instruction Set
  - Java byte codes



Three Instruction Sets

	ARM	Thumb	Jazelle
Instruction Size	32 bits	16 bits	8 bits
Core instructions	58	30	> 60% of Java byte codes in hardware; rest in software
Conditional Execution	most	Only branch instructions or in an IT block	N/A
Data processing instructions	Access to barrel shifter and ALU	Separate barrel shifter and ALU instructions	N/A
Program status register	Read/write in privileged mode	No direct access	N/A
Register usage	15 general purpose registers + pc	8 general purpose registers + 7 high registers + pc	N/A

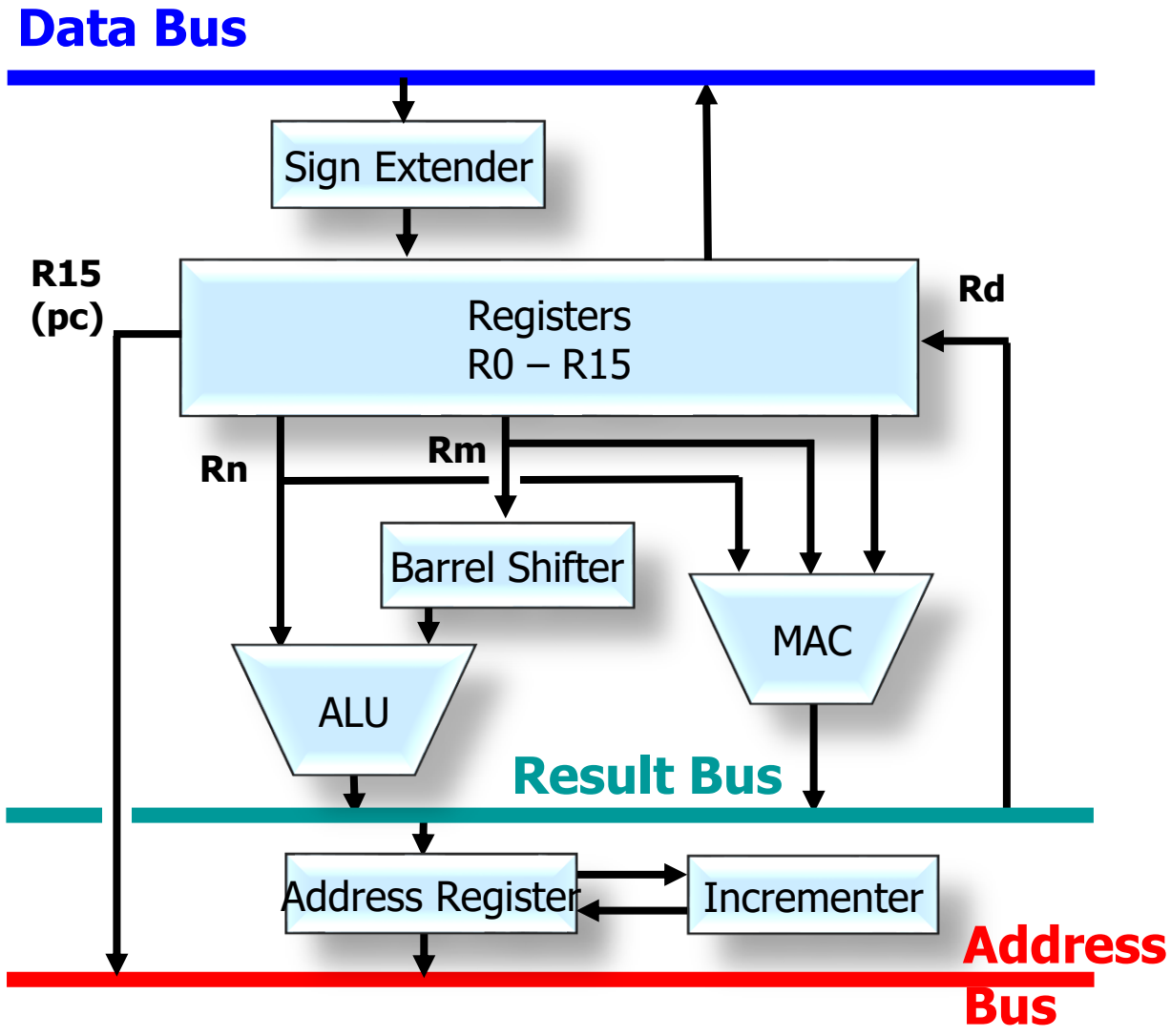
Code Density vs. Performance



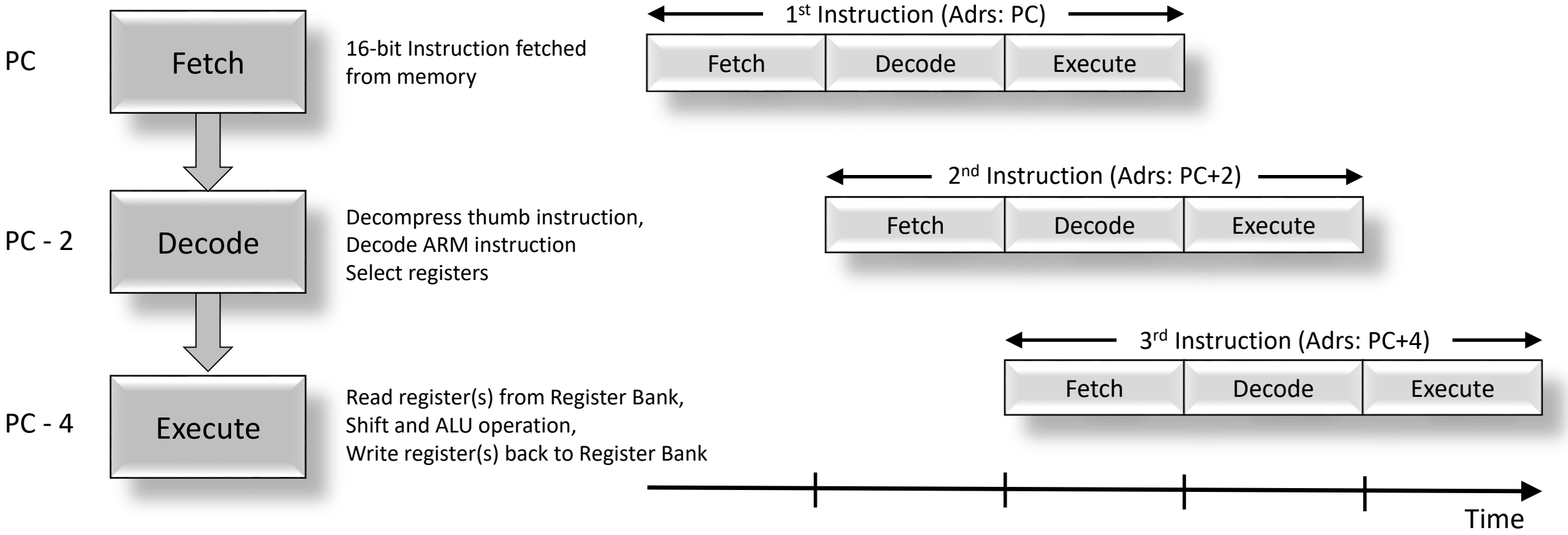
The ARMv7-M “Cortex-M3” Architecture

ARM Cortex-M3 CPU

- Load-store architecture
- A barrel shifter allows operand Rm to be shifted left or right any number of bit positions before being combined with operand Rn in the ALU, thus allowing calculations of the form Rn +/- 2kRm.
- A MAC (Memory Address Calculator) prepares subscripted and relative address references for memory access, and a special incrementer can adjust the memory address on subsequent instructions without having to recalculate the address.

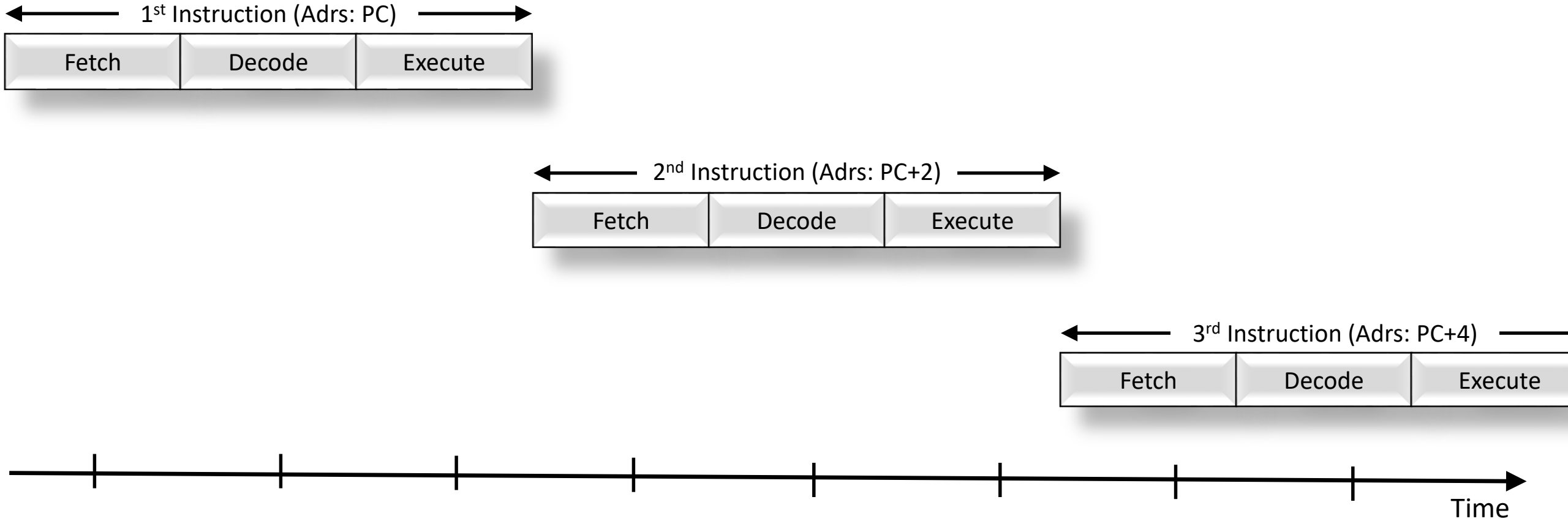


Pipelined Instruction Execution



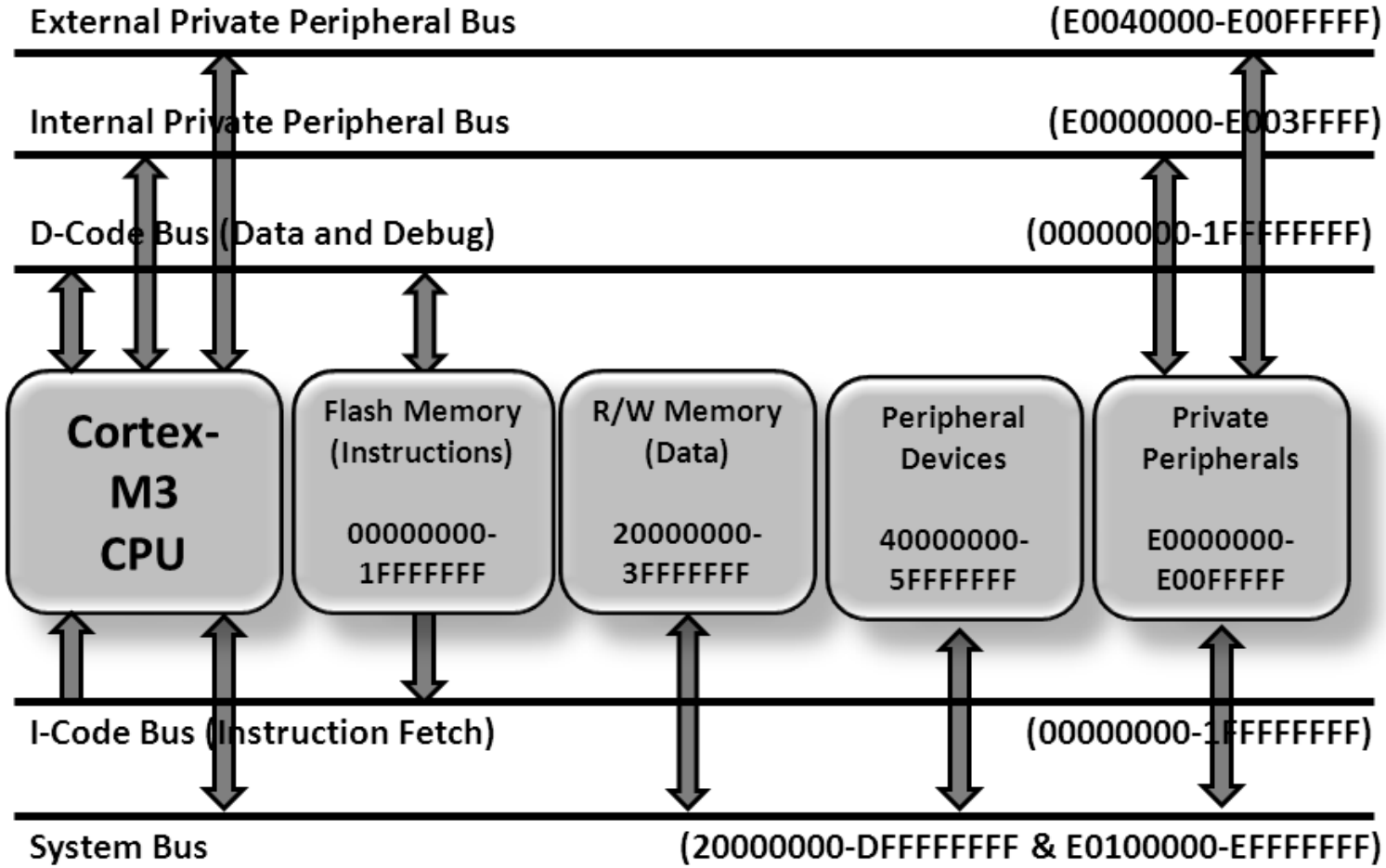
Pipelining allows three instructions to be in progress simultaneously during any one processor clock cycle. While one instruction is fetched, another is decoded, and a third is executed. Thumb instructions are 16 or 32 bits wide, with the majority being 16 bits. Thus, while the address held in PC is being used to read one 16-bit instruction from memory, the previous instruction (from address PC-2) is decoded, and the one before that (from address PC-4) is executed. One instruction completed every cycle.

Non-Pipelined Instruction Execution (hypothetical)



One instruction completed every 3 cycles. Performance is much lower than pipelined execution.

ARM Cortex-M3 Bus Structure (Harvard arch.)



Code and data are stored in separate and independent hardware; instructions are fetched over the I-Code bus from a nonvolatile flash memory while the system bus is used to access data in static read/write memory (or peripherals). Since the two memories are physically separate, overall performance is enhanced by a *Harvard* –style architecture with separate data paths that allow simultaneous instruction and data access

Summary

- von Neumann vs. Harvard architectures
- Instruction fetch/execute cycle
- Instruction sets for ARM processors
  - ARM, Thumb, Thumb-2
- Pipelined Instruction Execution