Computer Architecture & Program Execution

Lecture 2

Yeongpil Cho

Hanyang University

Topics

- Basics in Computer Architecture
- Program Execution

Basics in Computer Architecture

What is a Computer?

- Computer
  - a **programmable** electronic device that can store, retrieve, and process data

data center

server rack

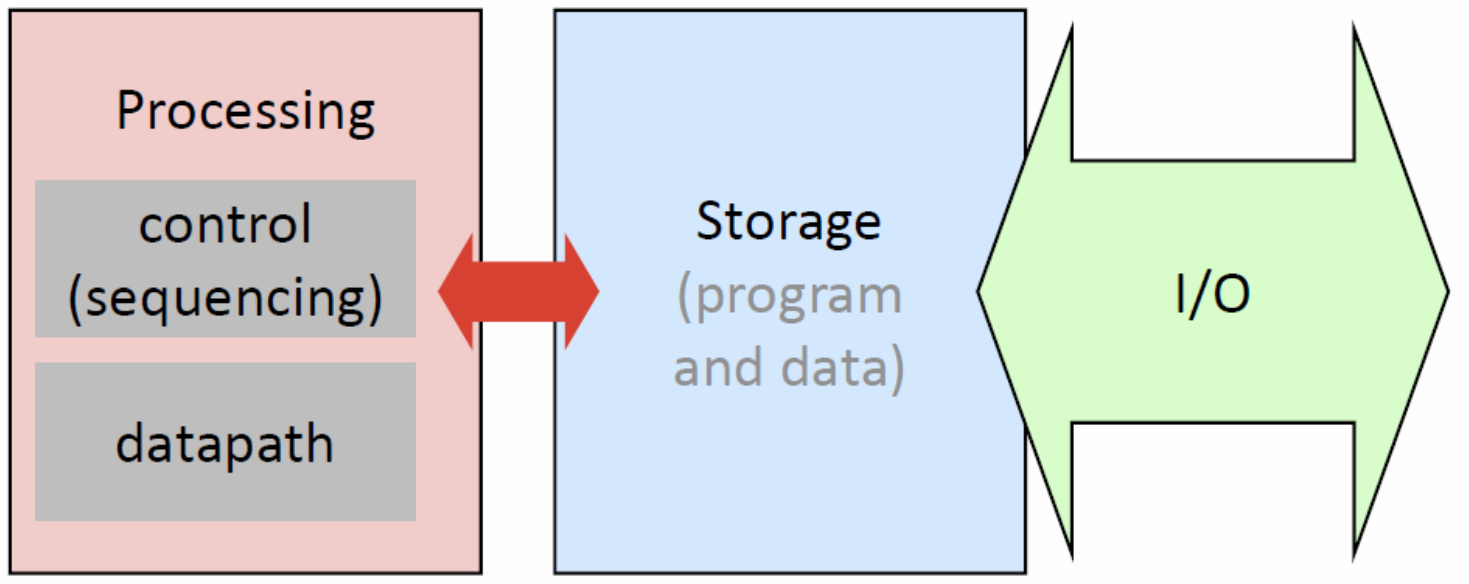
desktop

system-on-chip

components

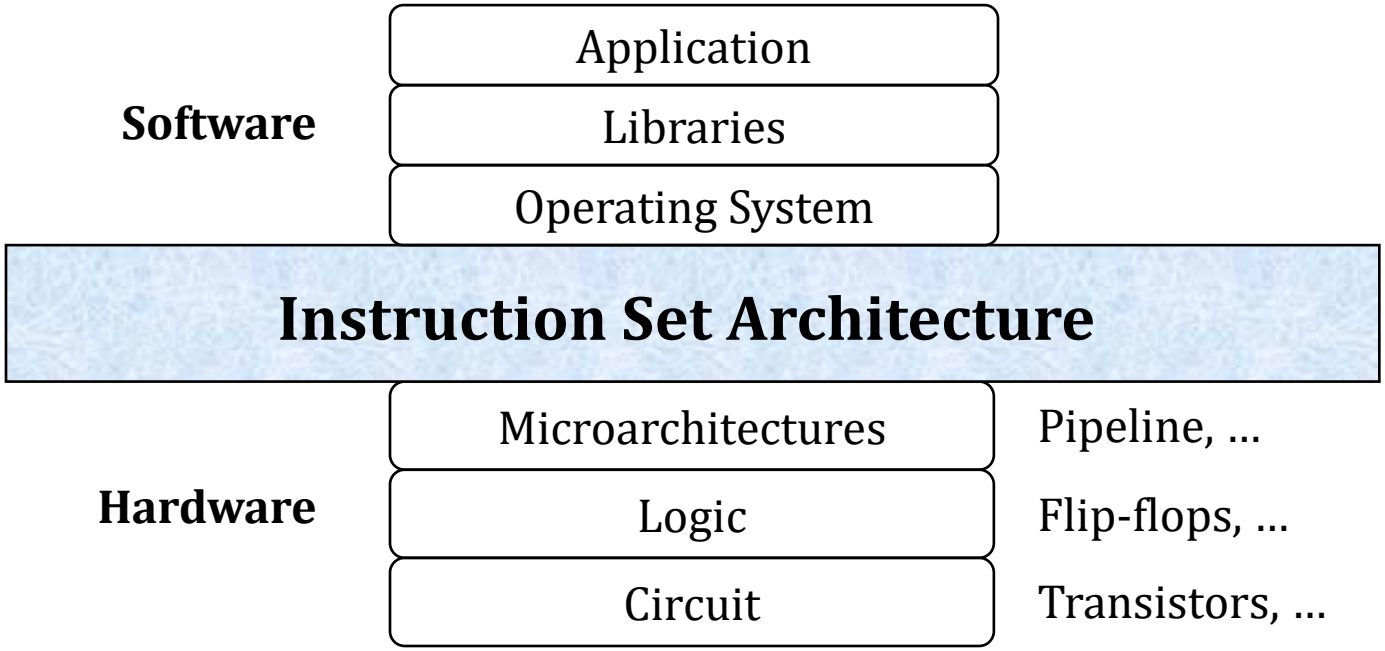
Stored-Program Computer

- A computer runs a program stored in a storage, usually memory.



Computer System Abstraction

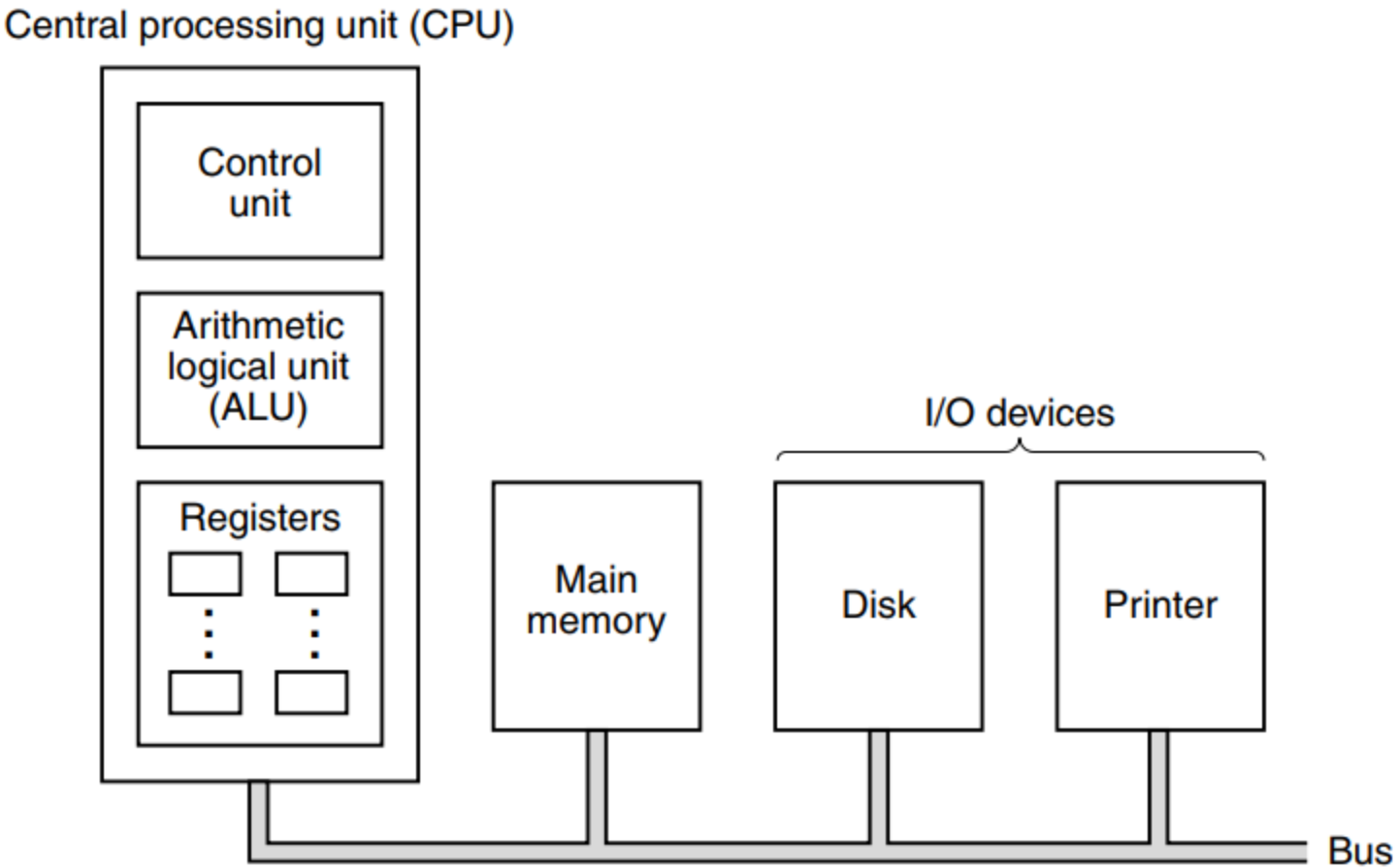
- Computer Architecture
  - The design and organization of a computer system
  - ISA + Microarchitectures + Sytem-level Organization
- Instruction Set Architecture (ISA)
  - The attributes **visible to the programmer** and which affect the logical execution of a program.



Attributes in ISA

- Data Type
  - Integers, floating-points
- Memory Model
  - Cache Memory, Memory Consistency
- Program Visible Processor State
  - General registers, Program counter, Processor status
- Instruction Set
  - Instructions and formats, Addressing modes
- System Model
  - Privilege, Interrupts, Multi-Core
- External Interfaces
  - PCIe, USB

A Typical Computer Organization

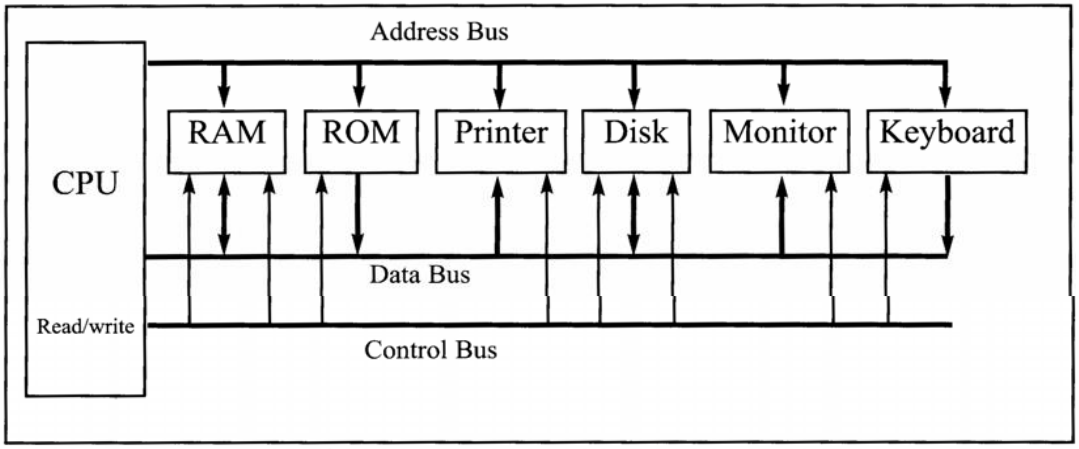


Computer: Structure

- CPU
  - Process data in memory
- Memory
  - **RAM** (Random Access Memory):
    temporary storage of programs that it is running
    - The data is lost if computer is turned off (Volatile memory)
  - **ROM** (Read-Only Memory):
    contains programs and information essential for computer operation
    - Permanent and usually cannot be changed by the user (Non-volatile memory)
- Peripherals
  - Serial ports, Parallel port, Keyboard, Monitor, …

Computer: Structure

- Bus
  - Strip of wires used to connect CPU with memory and peripherals
  - **Data bus**
    - The data lines used to carry information in and out of CPU
    - Typical values: 8-bit, 16-bit, 32-bit, 64-bit
    - Bidirectional
      - Data can get in or out of CPU



Computer: Structure

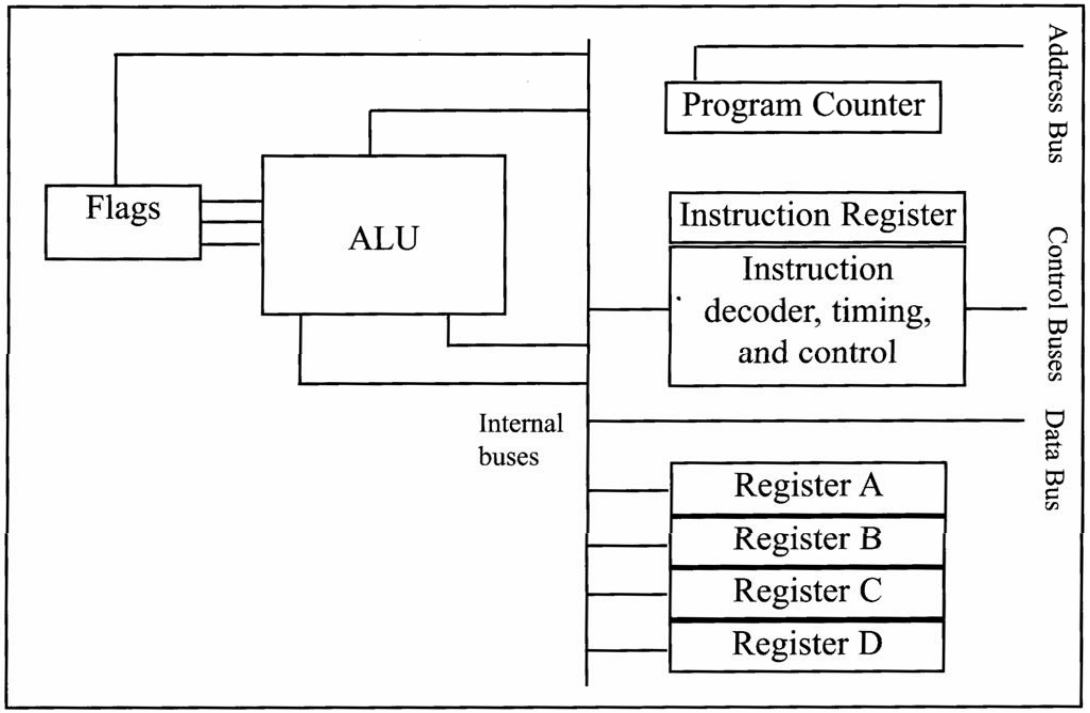
- Bus (Cont'd)
  - **Address bus**
    - Address bus is used to identify device and memory connected to CPU
    - Unidirectional
      - Its value can only be changed by CPU
    - Each device is assigned a range of addresses
      - E.g. 8-bit address line with 64-byte RAM, 32-byte ROM, 16 I/O ports
        - RAM: address 0 – 63
        - ROM: address 64 – 95
        - I/O ports: address 96 – 112
    - If there are *n* address lines, then the total address range is 2^n bytes
      - E.g. 8-bit address line: $2^8 = 256$ bytes address range
      - E.g. 32-bit address line: $2^{32} = 4GB$ address range
        - If the system has a total of 32-bit address lines, the maximum supported memory is 4GB

Computer: Structure

- Bus (Cont'd)
  - **Control bus**
    - CPU sends control information to devices to control their operations
      - e.g., read/write control
  - The operation of a computer relies on the combination of the three different buses
    - E.g. CPU wants to read a data byte from memory location 32
      - 1. CPU sets the value of the address bus to 32
      - 2. CPU uses the control bus to put the memory in read mode
      - 3. CPU reads the data byte on the data bus

Computer: Structure

- Inside CPU
  - **ALU (Arithmetic Logic Unit)**
    - Arithmetic functions (add, subtract, …)
    - Logic functions (and, or, not, …)



Computer: Structure

- Inside CPU (Cont'd)
  - **Registers**
    - Temporarily store information
    - Data read from memory or device will first be stored in registers, then CPU will process data in register
    - Calculation results will be store in register, then send out to memory
    - E.g. 3 + 5
      1. 3 will be first loaded to register A
         5 will be first loaded to register B
      2. ALU calculates 3+5
      3. the result 8, will be stored in register A
    - Typical size: 8-bit, 16-bit, 32-bit, 64-bit

Computer: Structure

- Inside CPU (Cont'd)
  - **Instruction**
    - a special binary pattern corresponds to a certain operation by CPU
      - E.g. 1011 0000 (B0H): move data to register A
      - E.g. 0000 0100 (04H): add a value to register A
    - Program is **a sequence of instructions**, and it is stored in memory
  - **Instruction Register (IR)**
    - CPU reads the program from the memory, one instruction at a time, and temporarily stores it in the IR
  - **Program Counter (PC)**
    - points to the memory address of the instruction to be fetched
  - **Instruction Decoder**
    - interprets the meaning of the instruction, so CPU can execute according to the instruction

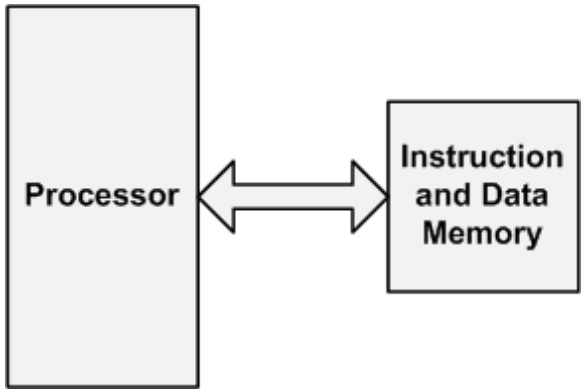
ISA: RISC and CISC

- **CISC** (Complex Instruction Set Computer)
  - A large number of instructions
  - Variable-length instruction formats
  - Some instructions that perform specialized task
  - Maybe small code size
  - Increase architecture complexity
- **RISC** (Reduced Instruction Set Computer)
  - Relatively few instructions
  - Fixed-length, easily decoded instruction format
  - Maybe large code size
  - Decrease architecture complexity
  - More trendy architecture

Memory Models

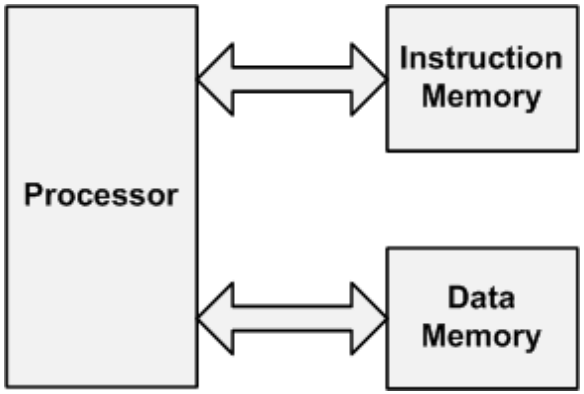
Von-Neumann

Instructions and data are stored in the same memory.



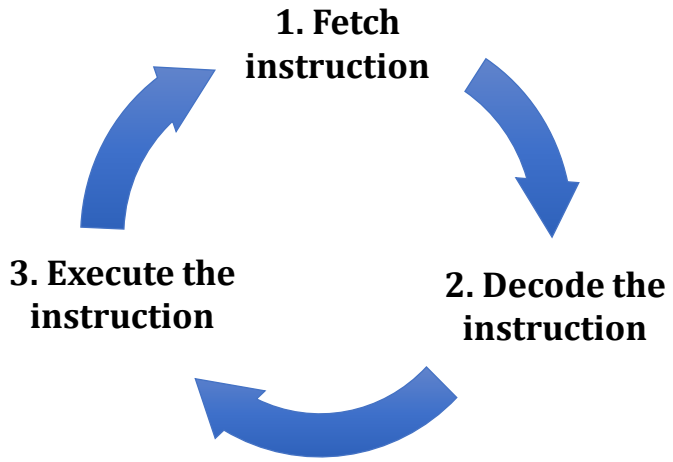
Harvard

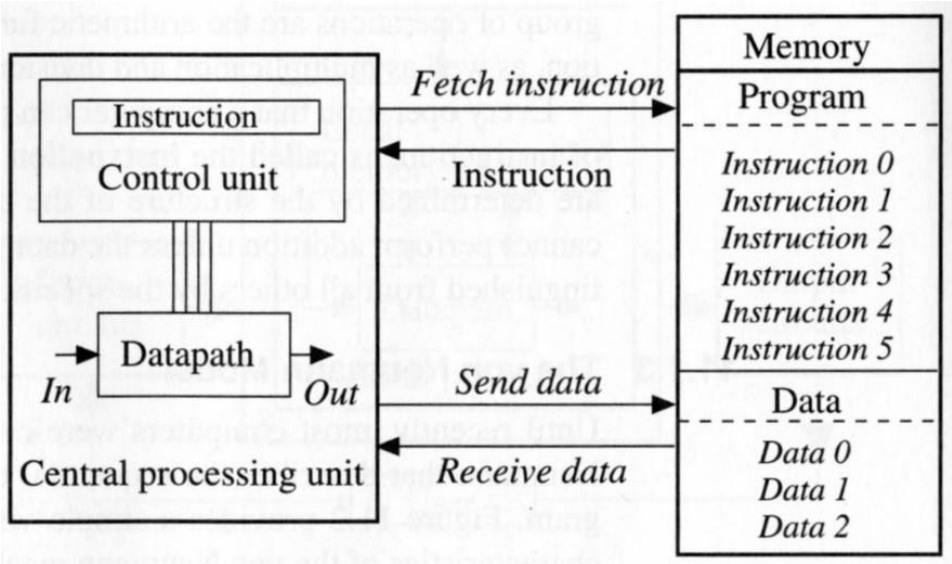
Data and instructions are stored into separate memories.



Instruction Execution Process

- **Instruction Fetch**
  - Reads next instruction into the IR.
  - The PC has the instruction address.
- **Instruction Decoding**
  - Decodes the op-code, gets the required operands and routes them to ALU.
- **Execution**
  - Generates control signals of ALU for execution.

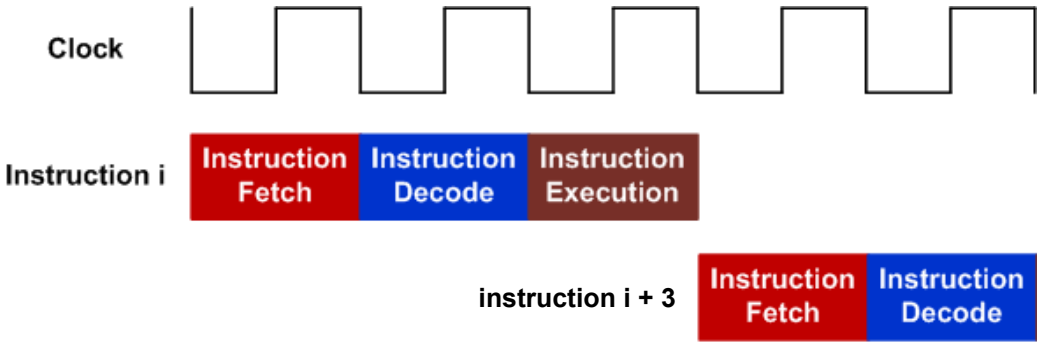




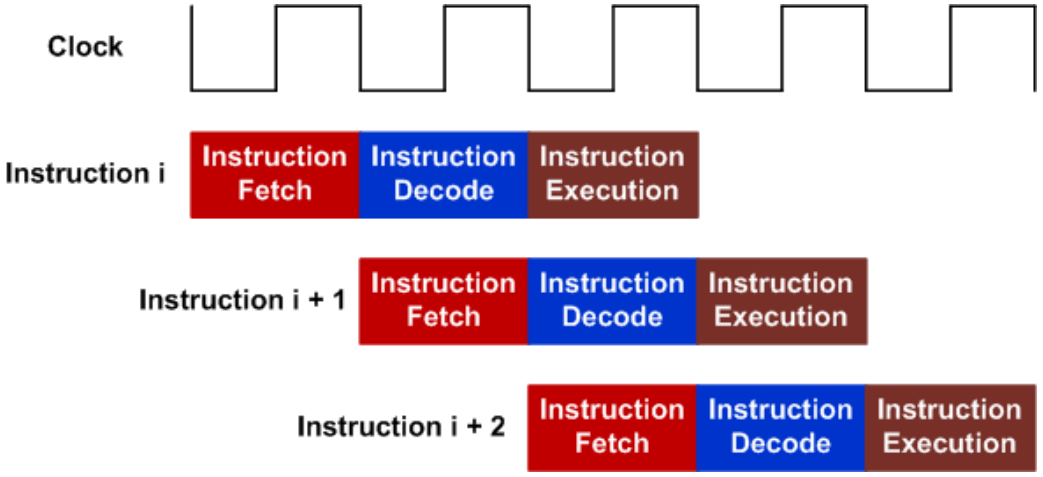
CPU Pipelining

- Improve performance by increasing instruction throughput.
- **Pipelining** allows hardware resources to be fully utilized

without pipelining

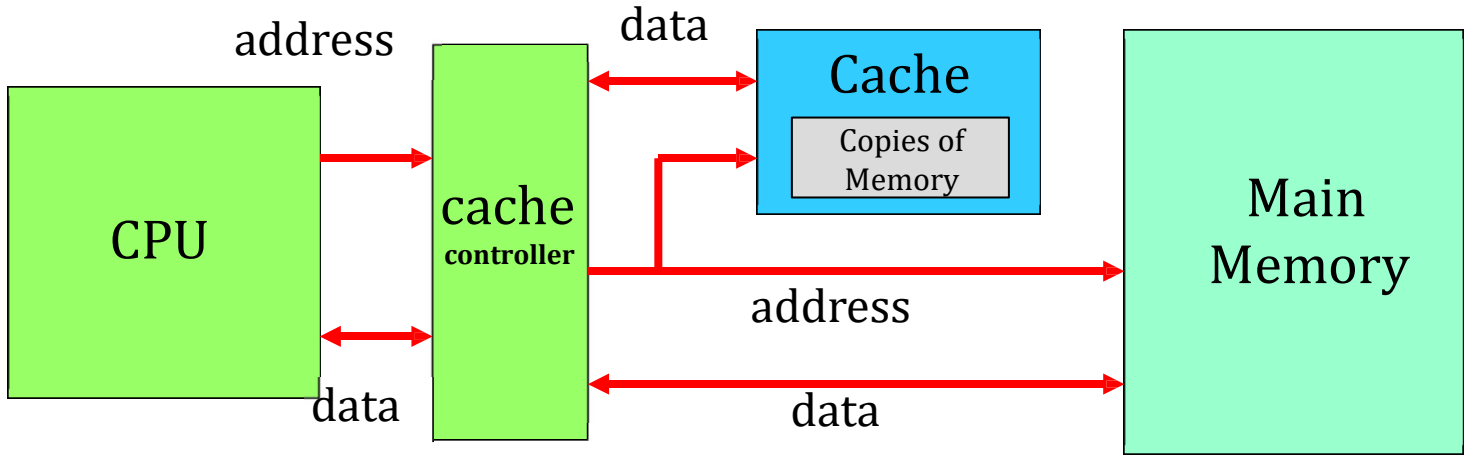


with pipelining



Cache Memory

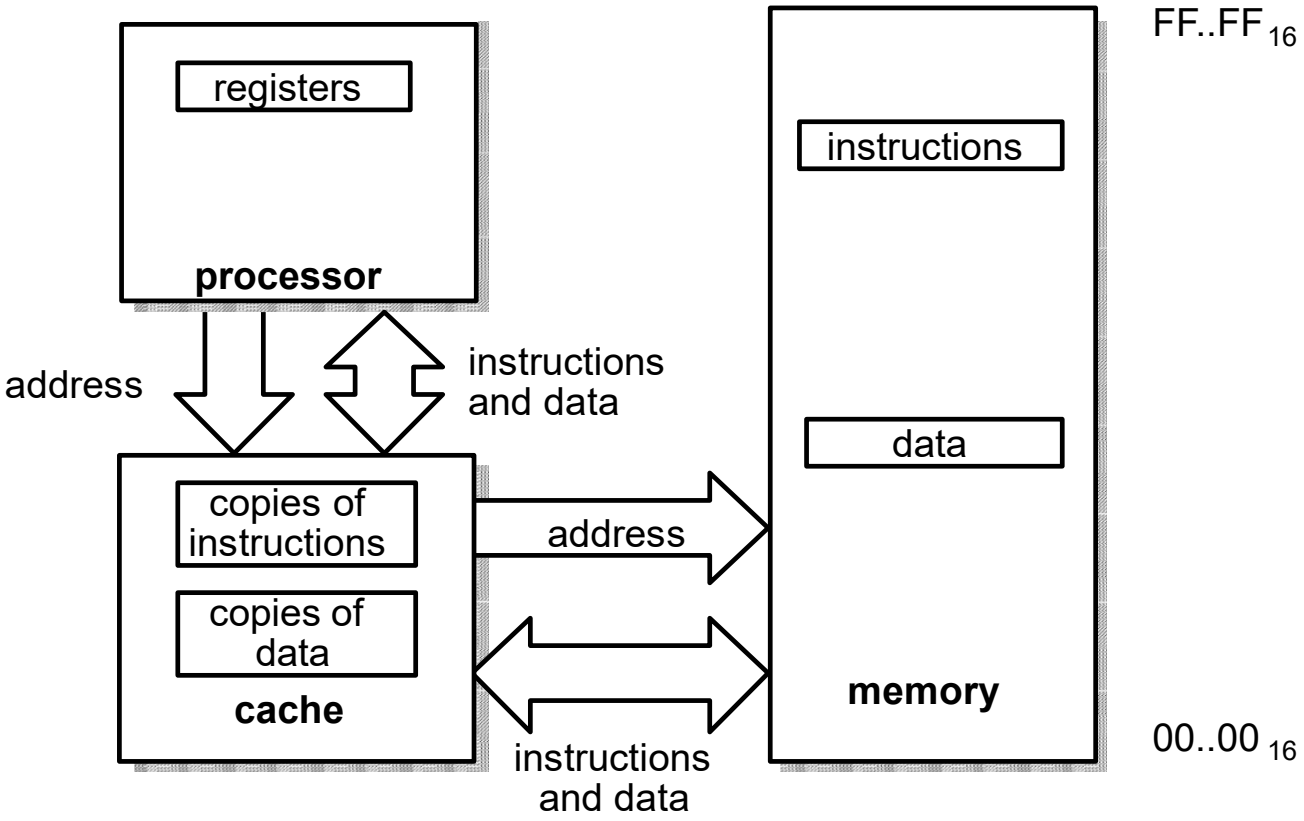
- **Cache:** Expensive but very fast memory directly connected to CPU interacting with slower but much larger main memory.
- Processor first checks if the addresses word is in cache.
- If the word is not found in cache, a block of memory containing the word is moved to the cache.



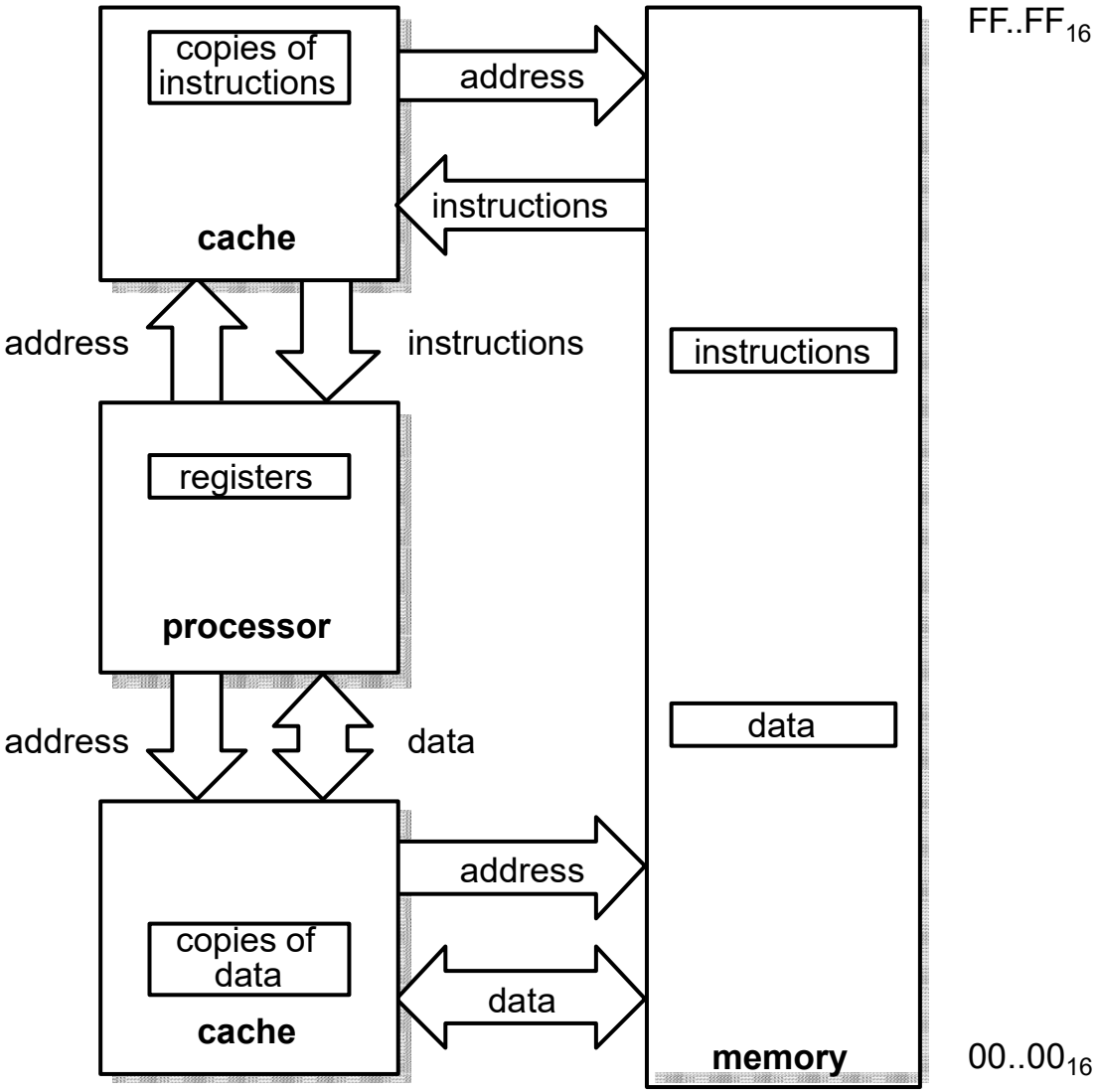
Effectiveness of Caching

execute typical instruction	1/1,000,000,000 sec = 1 nanosec
fetch from L1 cache memory	0.5 nanosec
branch misprediction	5 nanosec
fetch from L2 cache memory	7 nanosec
Mutex lock/unlock	25 nanosec
fetch from main memory	100 nanosec
send 2K bytes over 1Gbps network	20,000 nanosec
read 1MB sequentially from memory	250,000 nanosec
fetch from new disk location (seek)	8,000,000 nanosec
read 1MB sequentially from disk	20,000,000 nanosec
send packet US to Europe and back	150 milliseconds = 150,000,000 nanosec

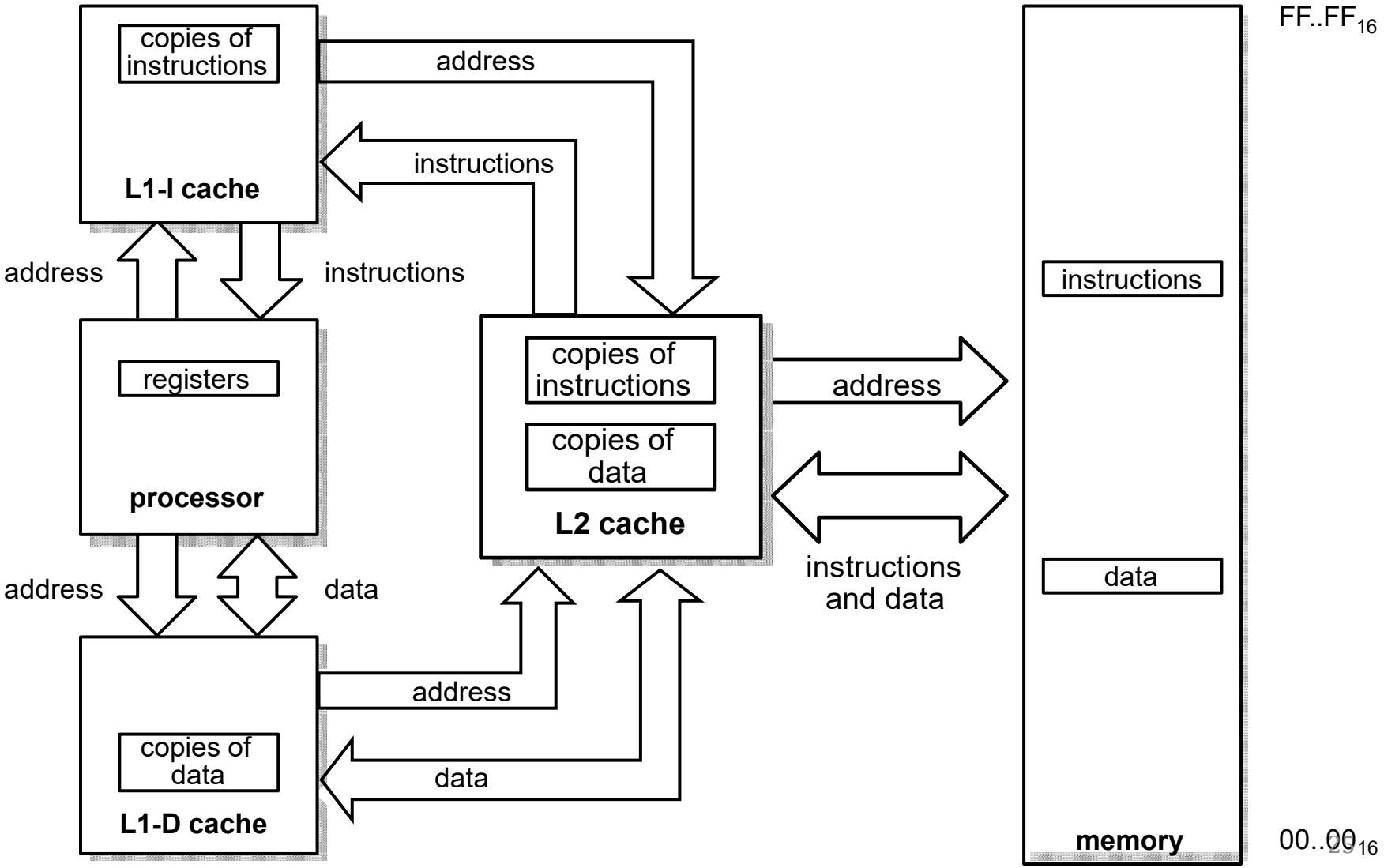
A Unified Instruction and Data Cache



Separate Data and Instruction Caches

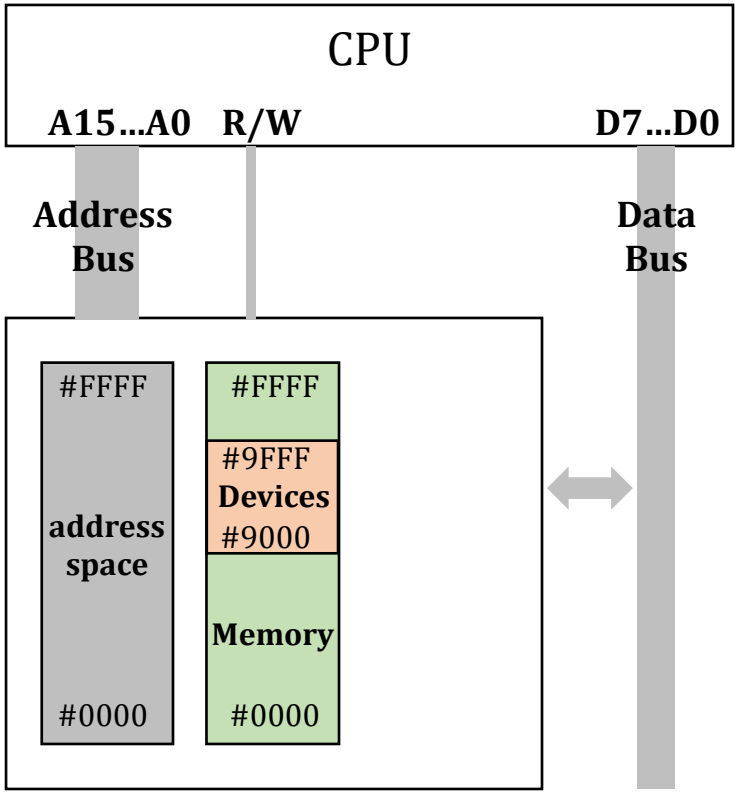


Hybrid Caches



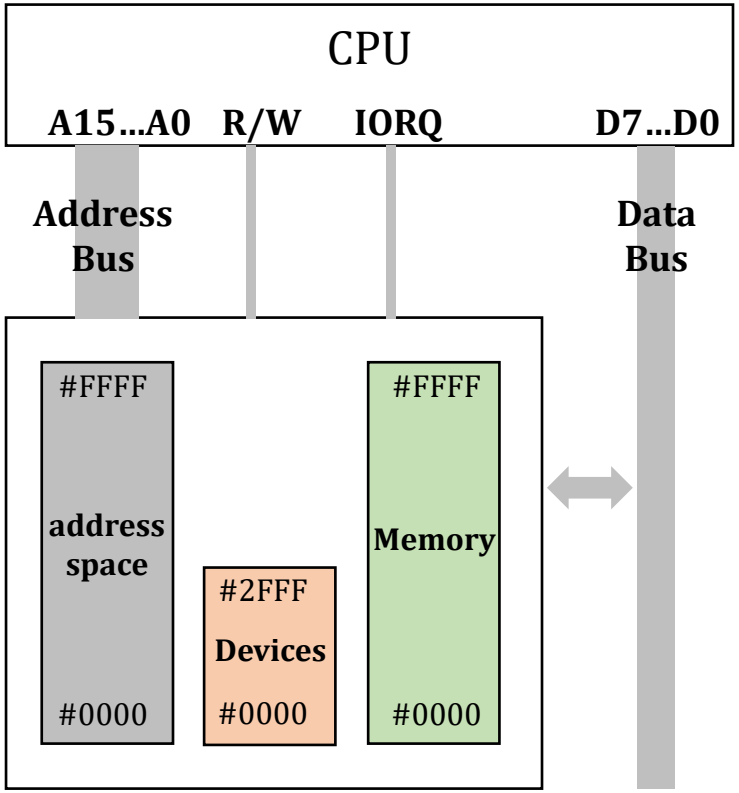
I/O Interface

- Memory-mapped I/O
  - Memory and I/O share the same address space
  - CPU accesses I/O using ordinary memory instructions



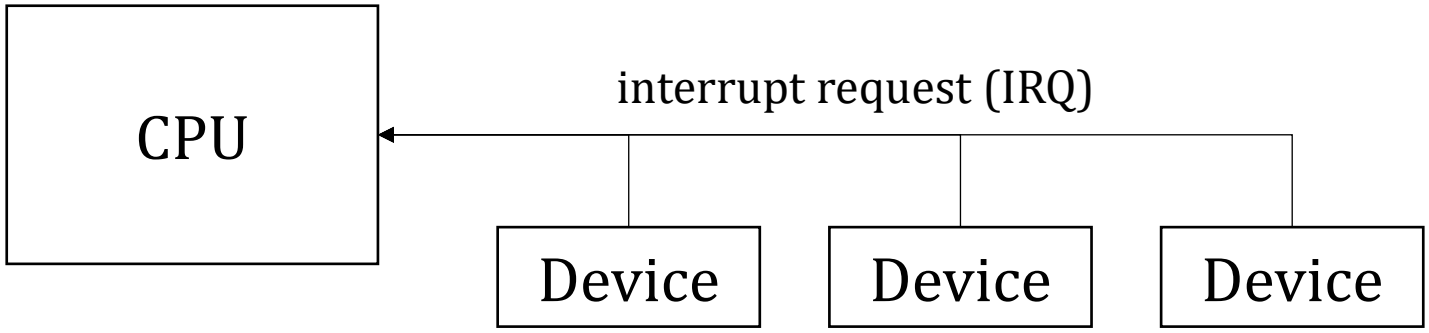
I/O Interface

- Port-mapped I/O
  - Memory and I/O have their own address spaces
  - CPU accesses I/O using dedicated instructions



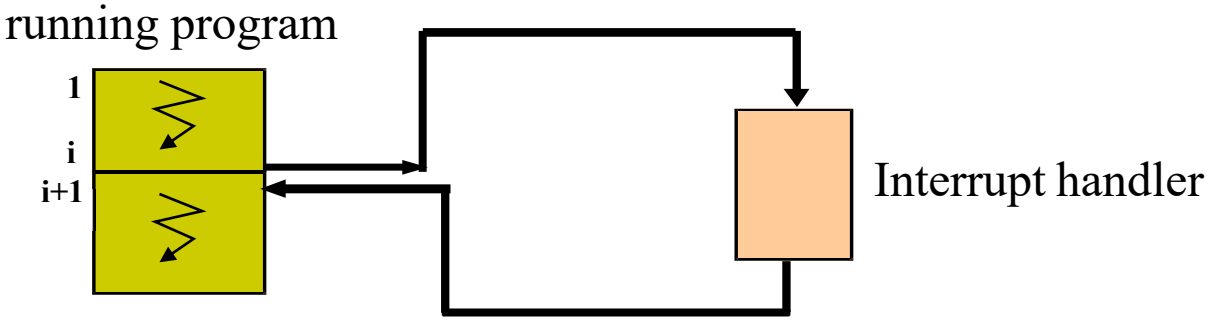
Interrupts

- A processor should react to device events.
- **Interrupt** is a mechanism to notify processors of the occurrence of a device event.
  - If an interrupt request signal is asserted from a device, the processor stops what it's doing and handles the request by calling an interrupt handler (or interrupt service routine).



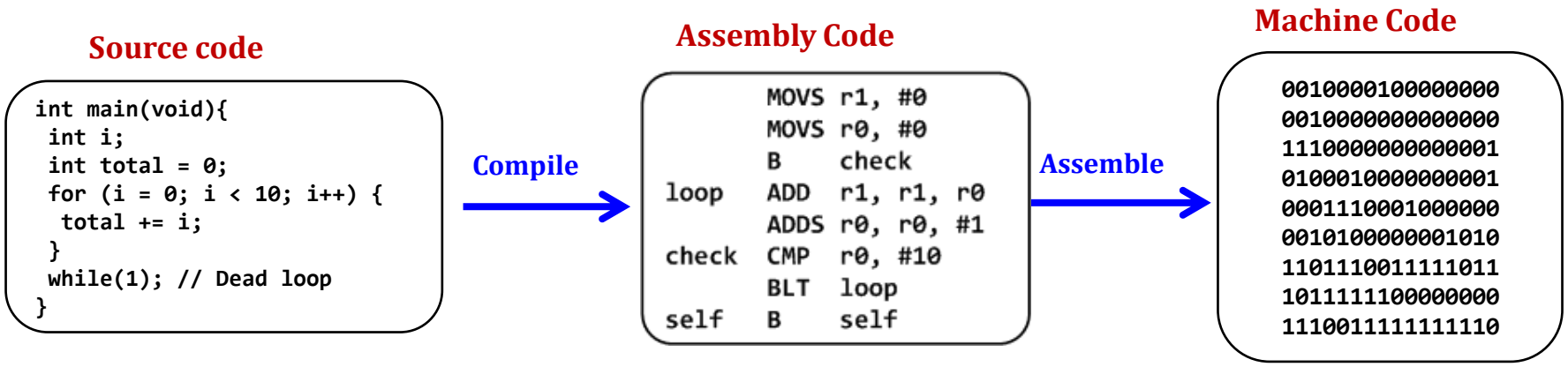
Instruction Cycle with Interrupts

- Generally, CPU checks for interrupts at the end of each instruction and executes the interrupt handler if required.
- The interrupt Handler identifies the nature/source of an interrupt and performs whatever actions are needed.
  - It takes over the control after the interrupt.
  - Control is transferred back to the interrupted program that will resume execution from the point of interruption.
  - Point of interruption can occur anywhere in a program.
  - State of the program is saved.



Program Execution

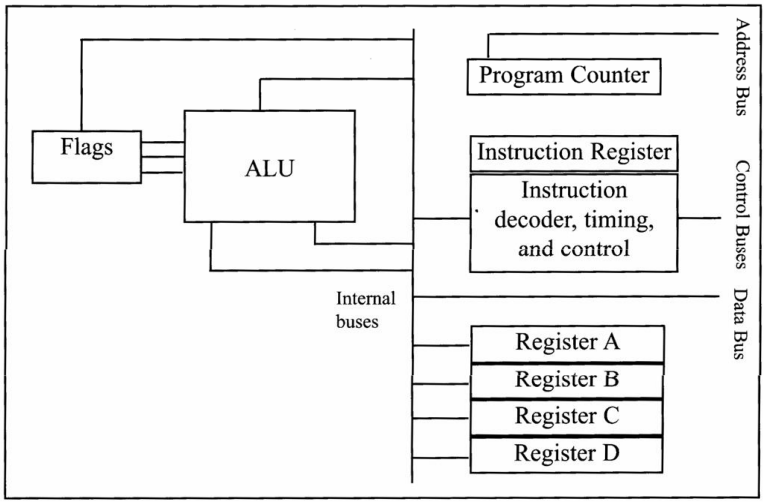
Levels of Program Code

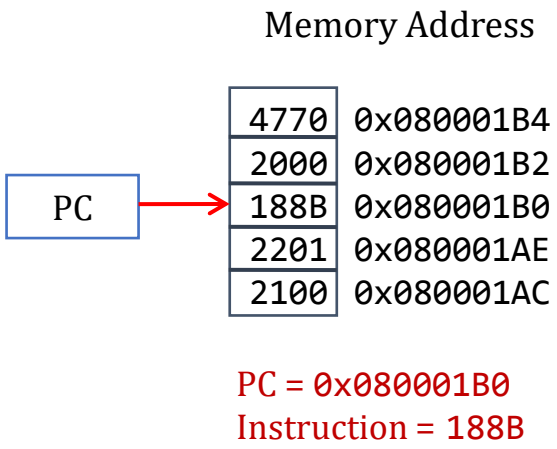


- High-level language
  - Level of abstraction closer to problem domain
  - Provides for productivity and portability
- Assembly language
  - Textual representation of instructions
- Hardware representation
  - Binary digits (bits)
  - Encoded instructions and data

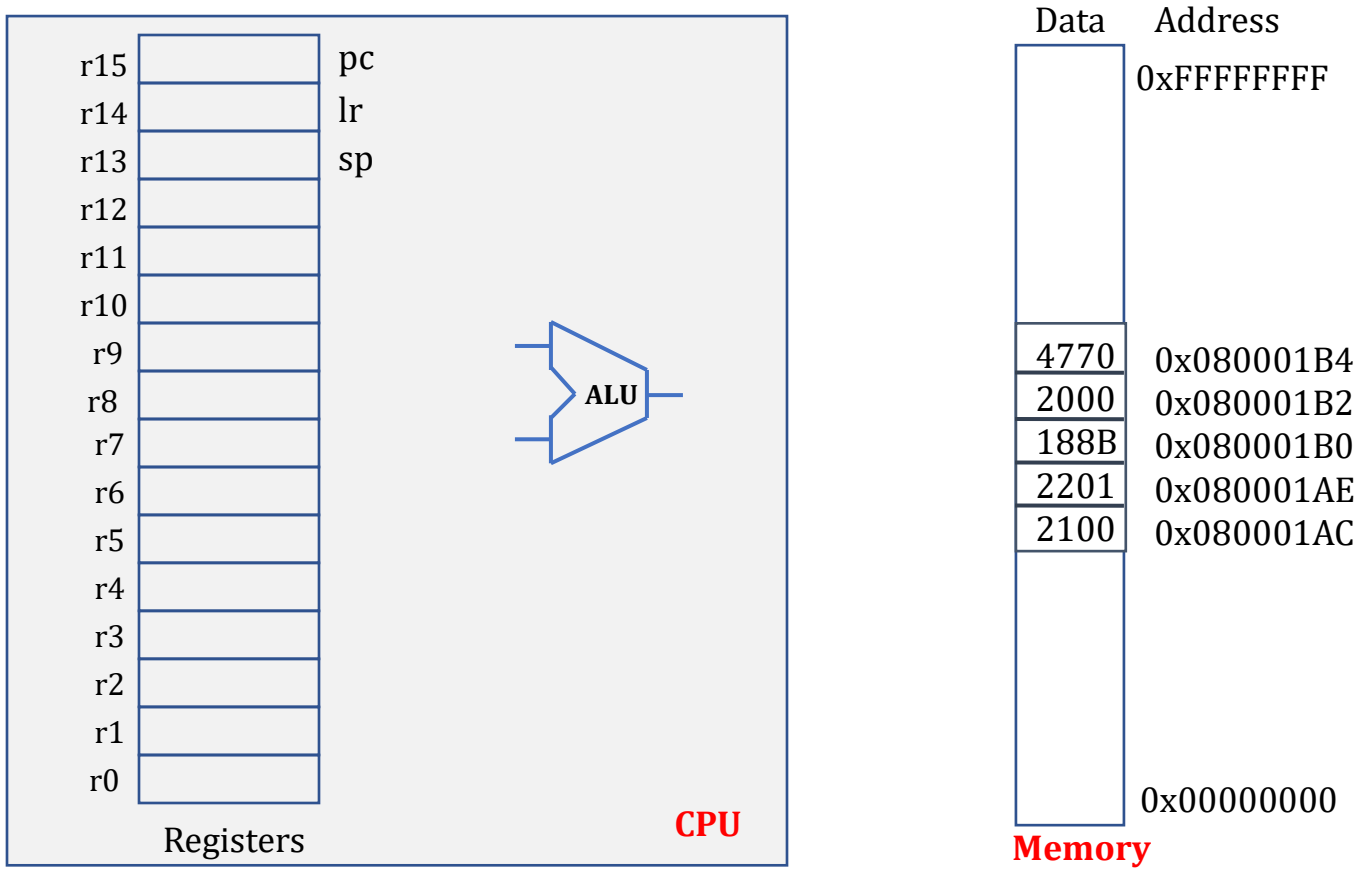
Program Execution

- Processor runs a program by executing instructions one by one
  - Program's instructions are stored on memory.
  - Program Counter (PC) holds the memory address of the next instruction to be fetched.
  - Processor interprets the meaning of each instruction and executes accordingly.
  - Processor puts data (i.e., operands) into registers and perform operations

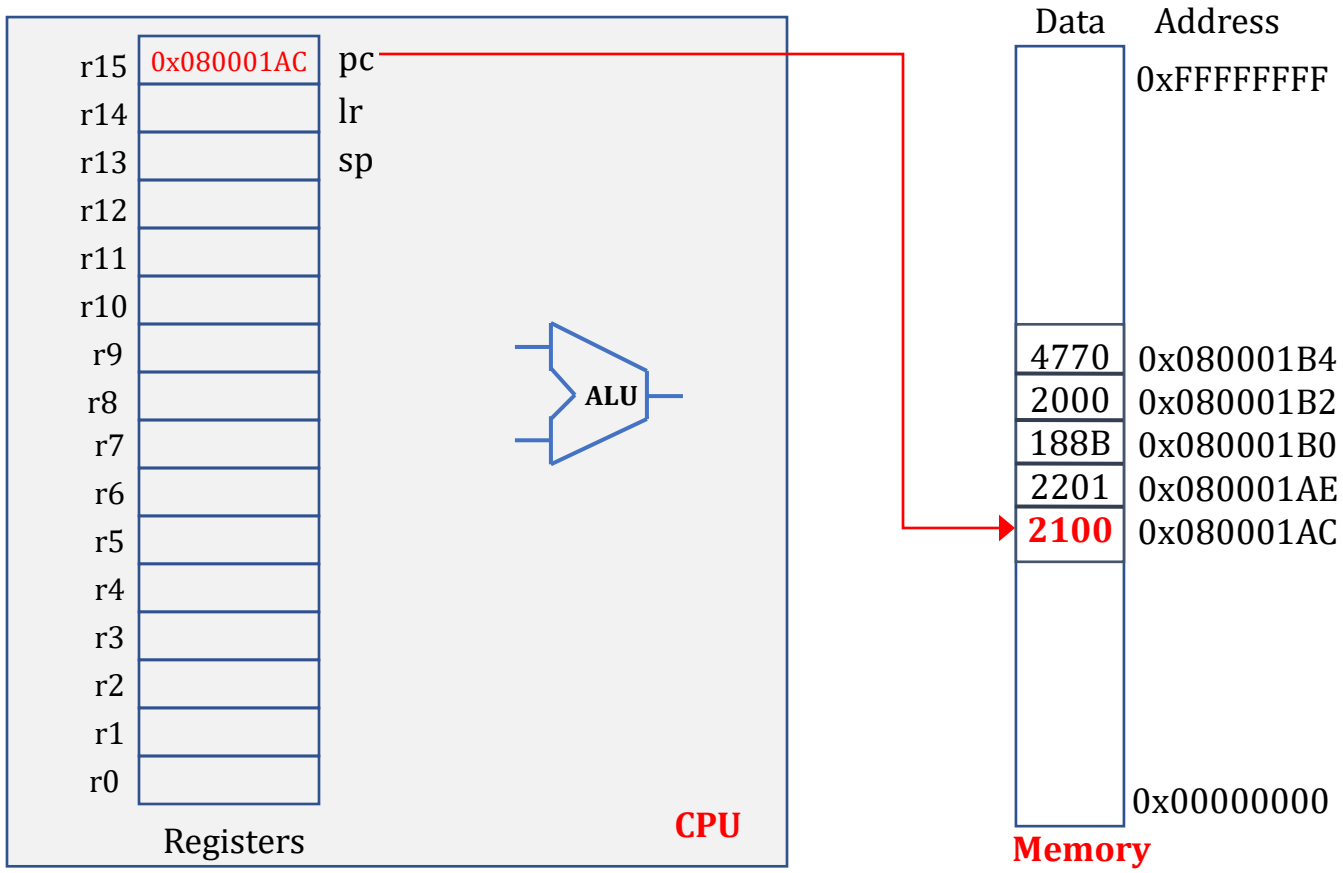




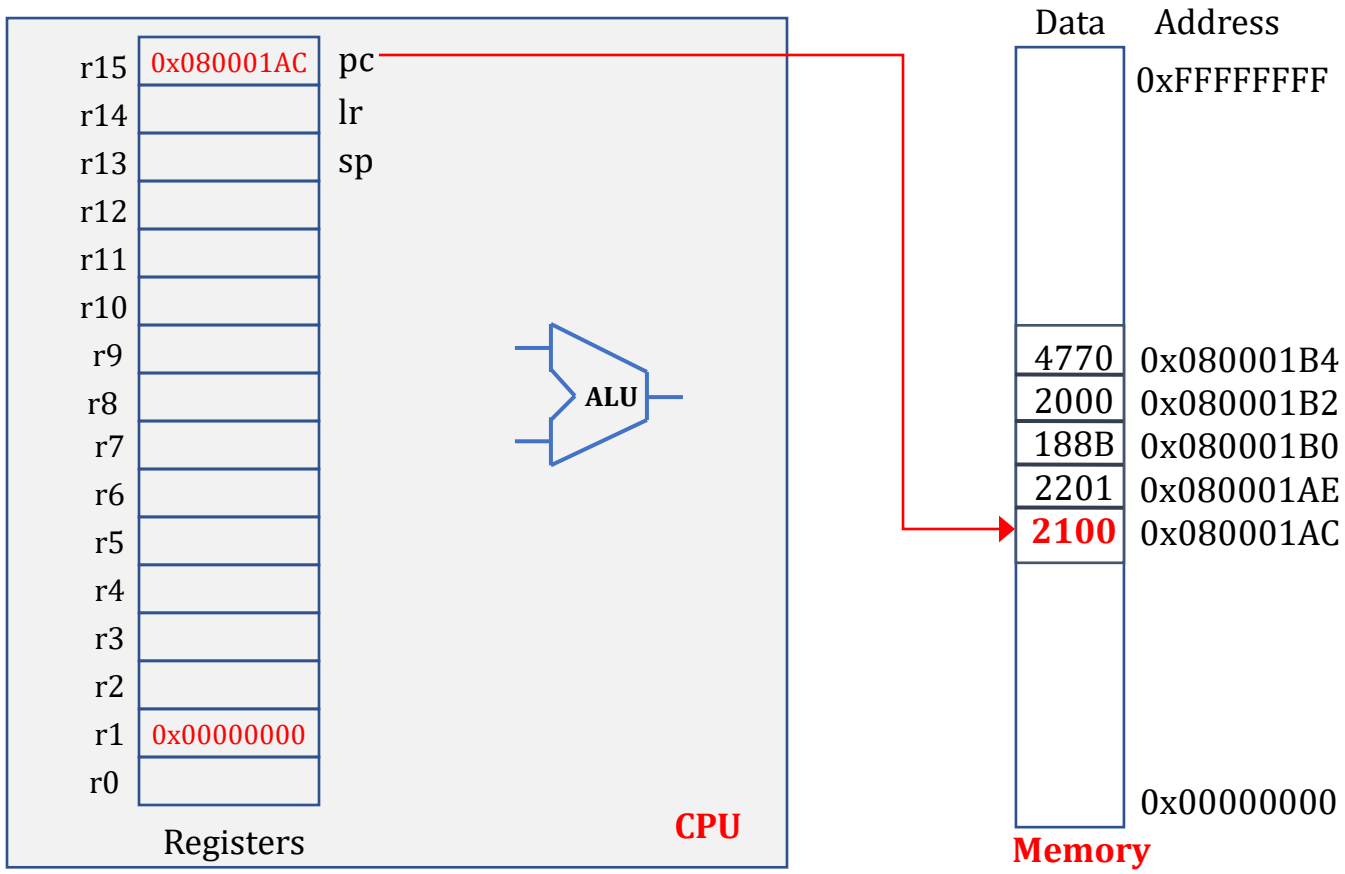
Machine codes are stored in memory



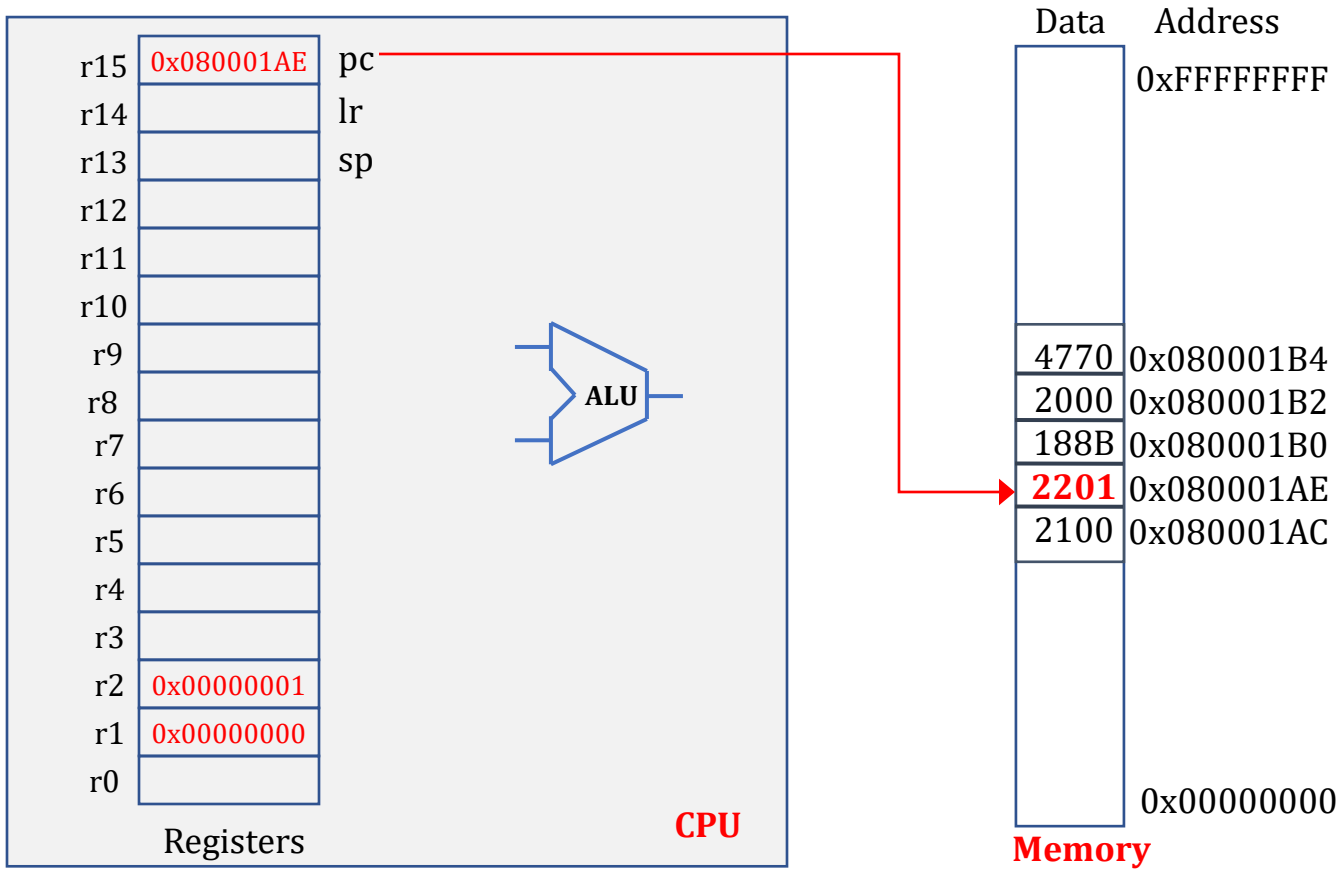
Fetch Instruction: pc = 0x08001AC
Decode Instruction: 2100 = MOVS r1, #0x00



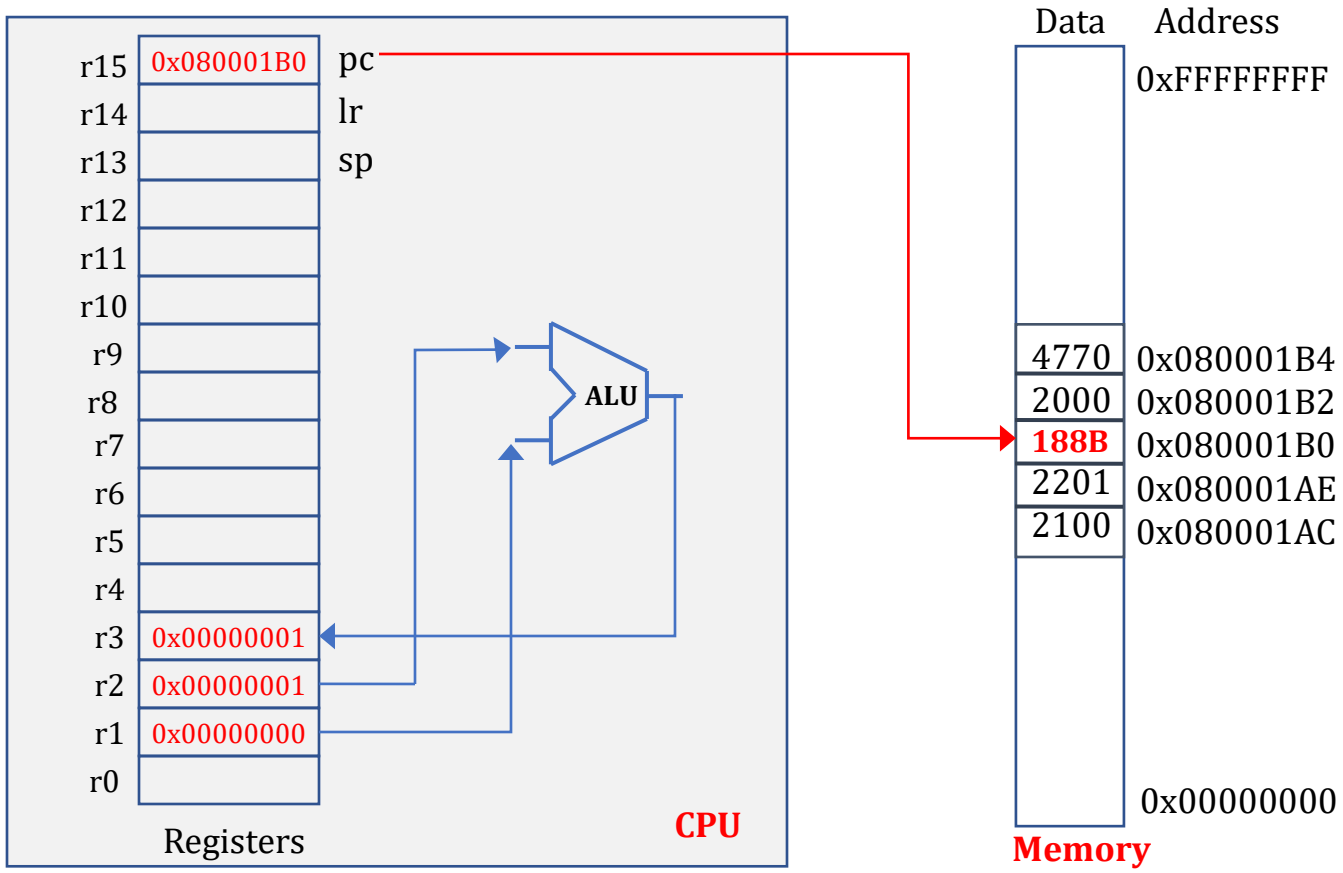
Execute Instruction: MOVS r1, #0x00



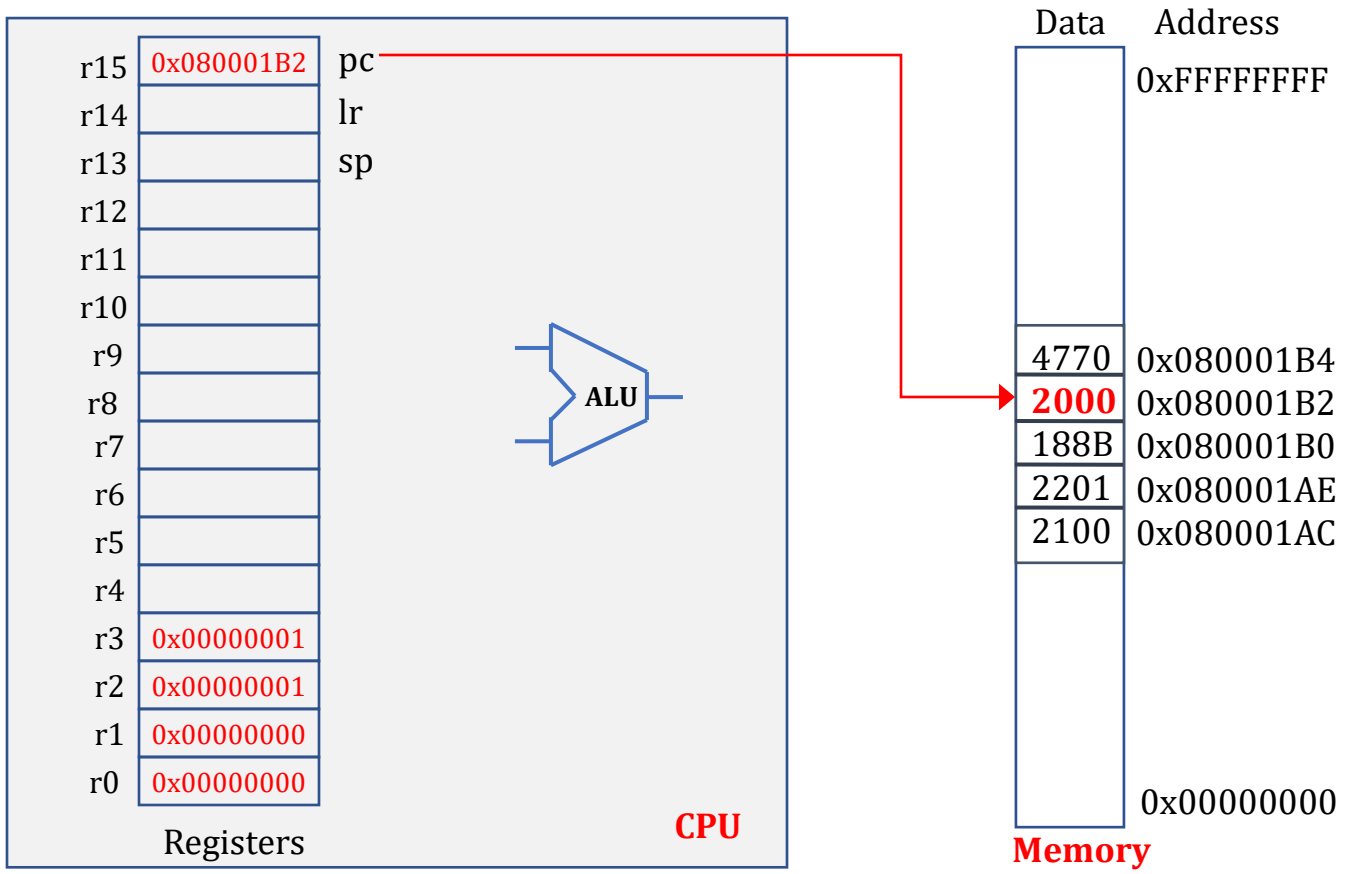
Fetch Next Instruction: pc = pc + 2
Decode & Execute: 2201 = MOVS r2, #0x01



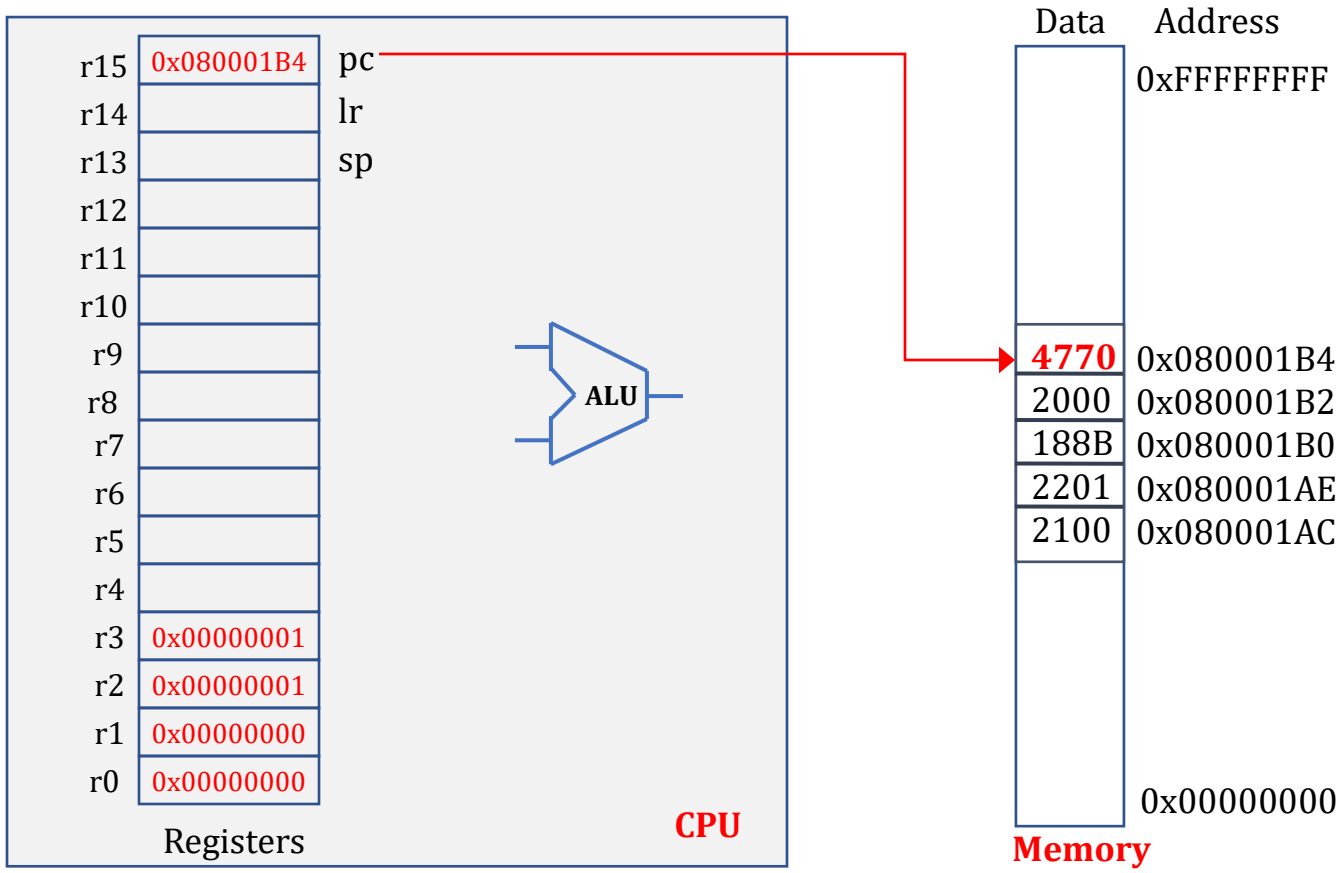
Fetch Next Instruction: pc = pc + 2
Decode & Execute: 188B = ADDS r3, r1, r2



Fetch Next Instruction: pc = pc + 2
Decode & Execute: 2000 = MOVS r0, #0x00



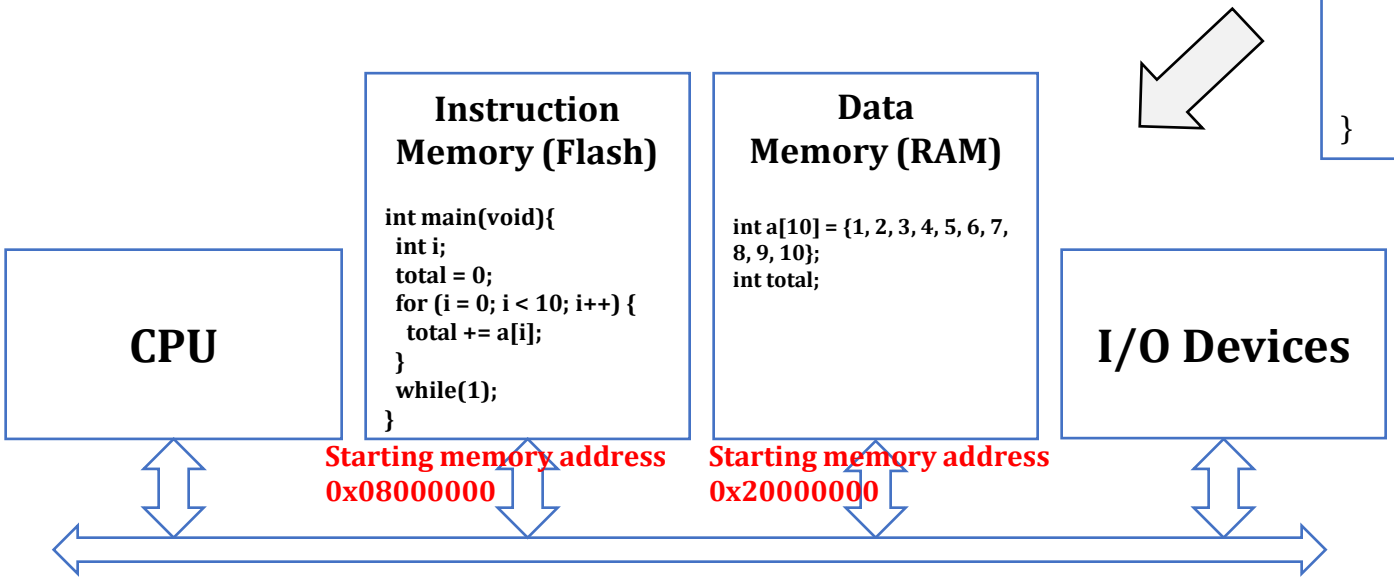
Fetch Next Instruction: pc = pc + 2
Decode & Decode: 4770 = BX lr



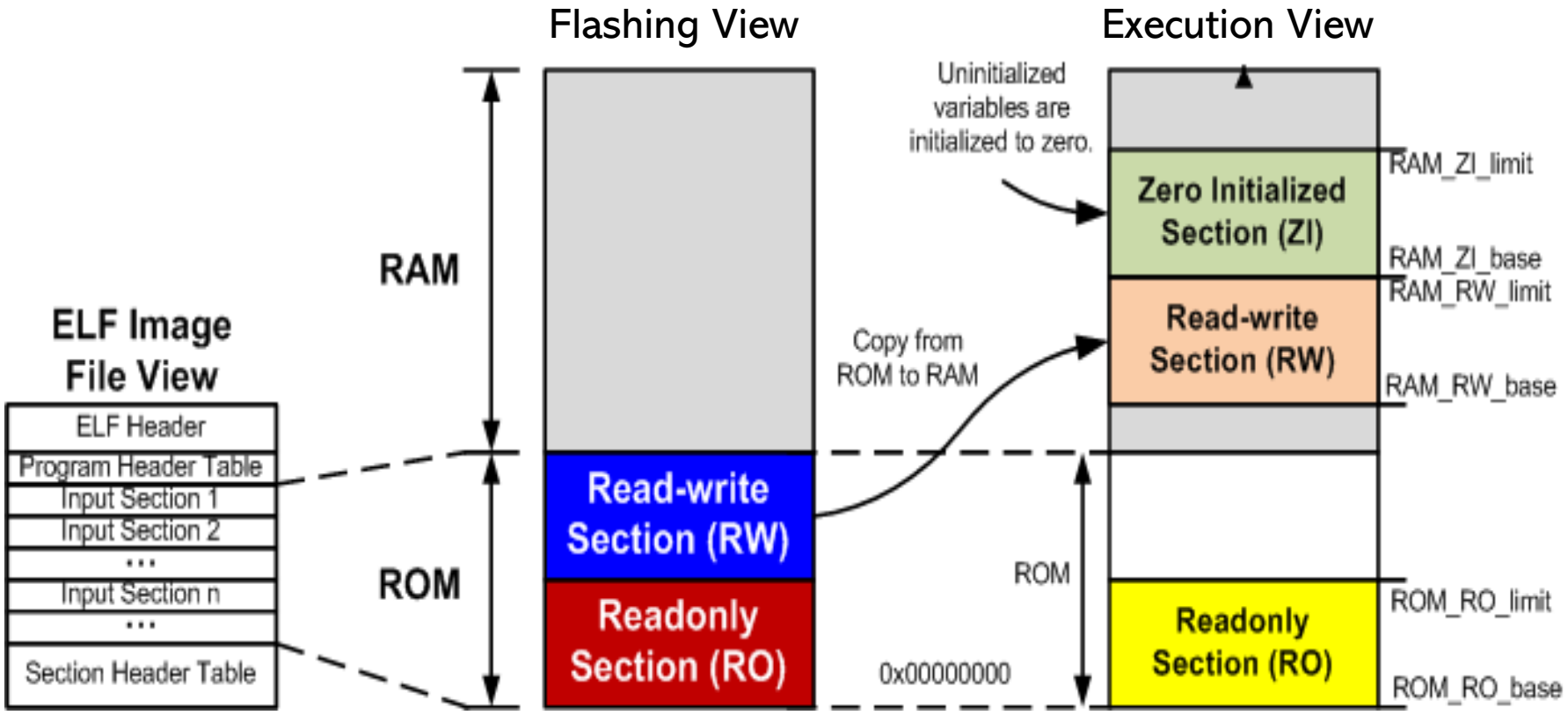
How are programs loaded?

- Program is stored on storage (e.g., flash memory)
  - Program (Binary file) = Code + Data
- They are loaded into memory separately
  - Code → Instruction Memory (Flash)
  - Data → Data Memory (SRAM or DRAM)

```
int a[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
int total;

int main(void){
        int i;
        total = 0;
        for (i = 0; i < 10; i++) {
                total += a[i];
        }
        while(1);
}
```



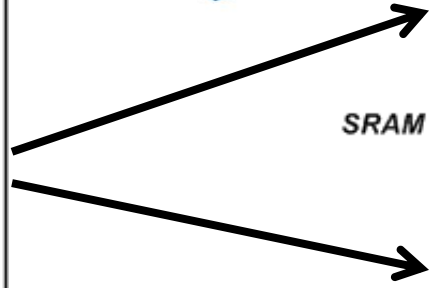
A detailed breakdown: Loading a Binary File into Memory

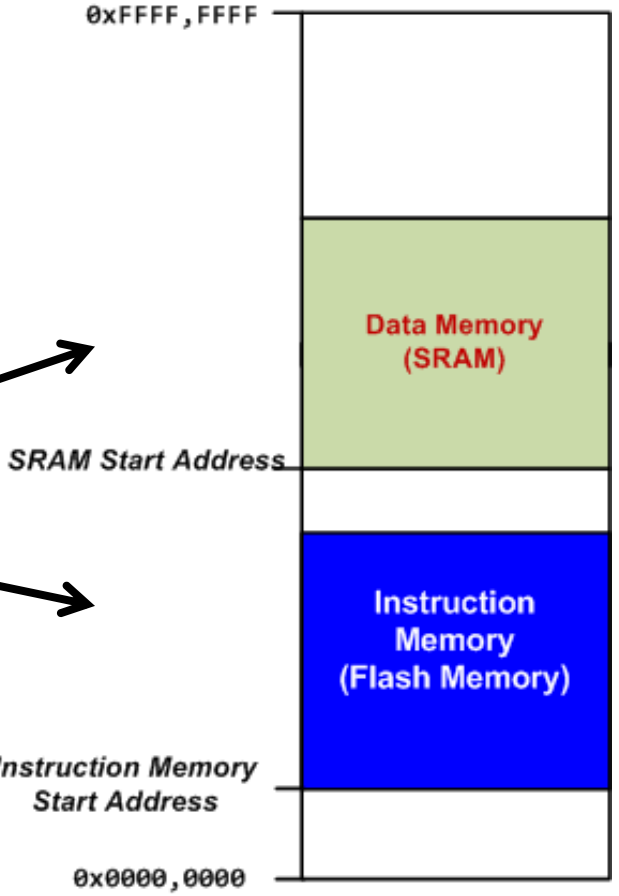


A detailed breakdown: Loading Code and Data into Memory

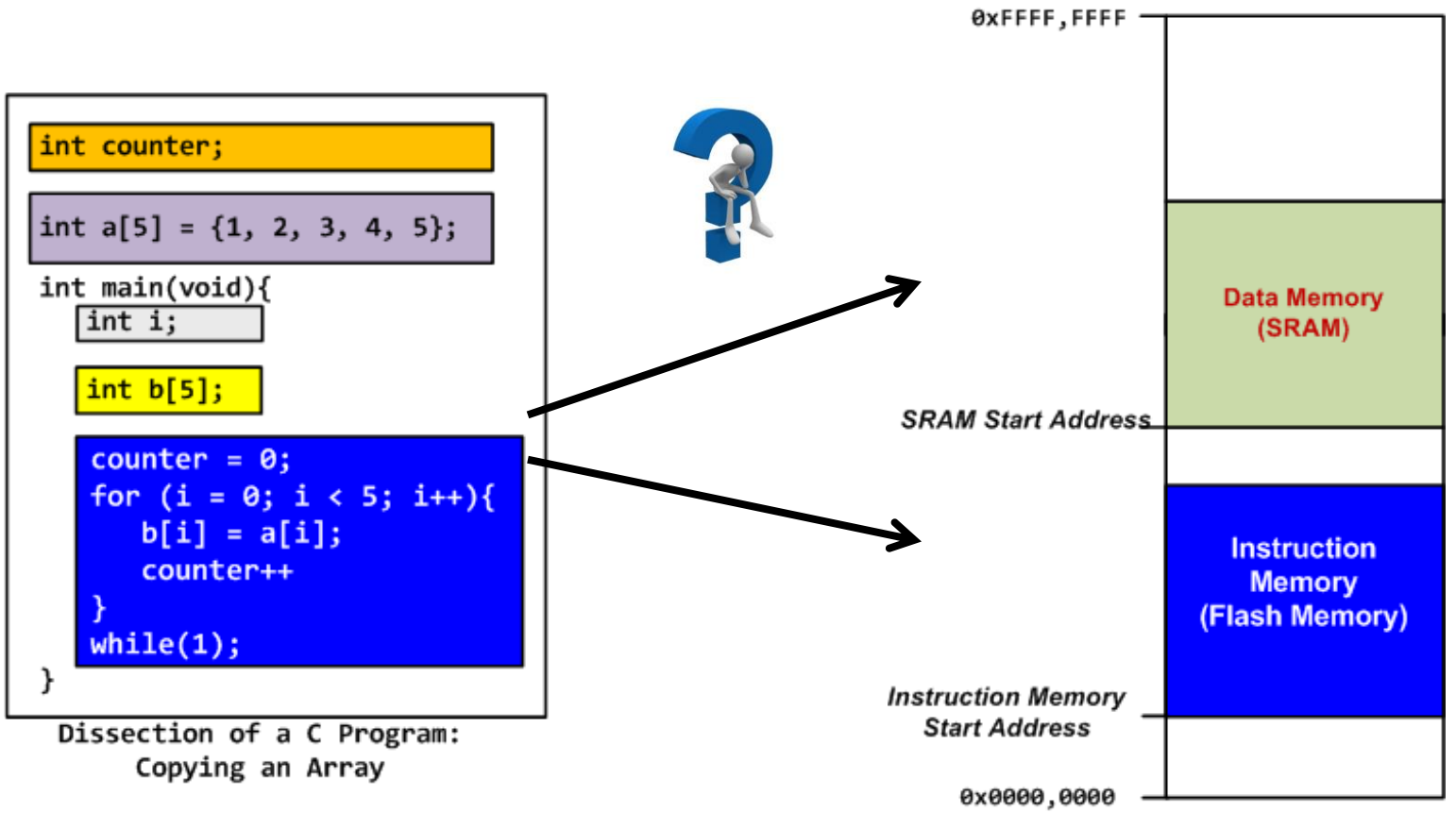
```
int counter;

int a[5] = {1, 2, 3, 4, 5};

int main(void){
   int i;

   int b[5];

   counter = 0;
   for (i = 0; i < 5; i++){
      b[i] = a[i];
      counter++
   }
   while(1);
}
```

C Program:
Copying an Array

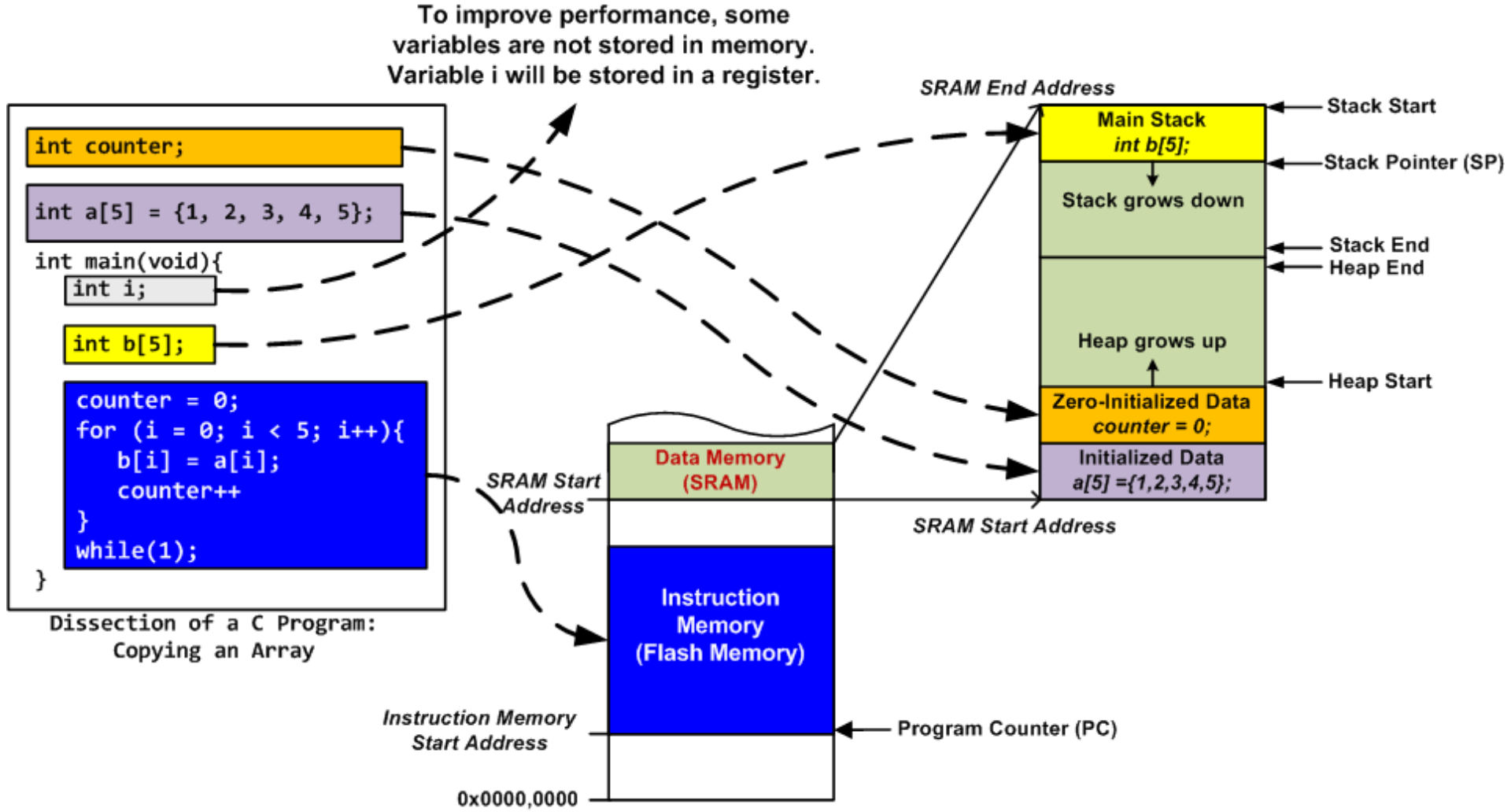




A detailed breakdown:
Loading Code and Data into Memory



A detailed breakdown: Loading Code and Data into Memory



Summary

- Computer Architecture Review
  - Basic Structure
  - Core Components
- Program Execution
  - Execution Sequence of Instructions
  - Program Loading