AI224: Operating Systems — Lecture 2

Dr. Karim Lounis

Jan - May 2025

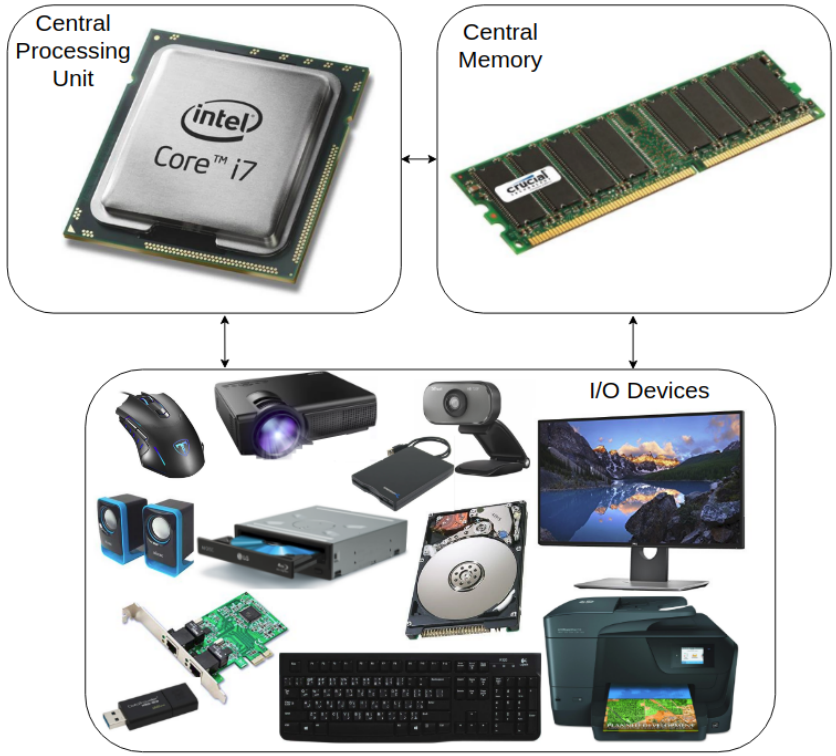
Review: Computer Systems Architecture

Hardware Components of a Computer System

A computer systems is composed of: (1) Hardware Components, (2) Software Components — Application and System software, and (3) User.

The main **hardware components** are (Von Neumann architectural model):

- Central Processing Unit.
- Central memory — RAM.
- Peripherals — I/O devices.



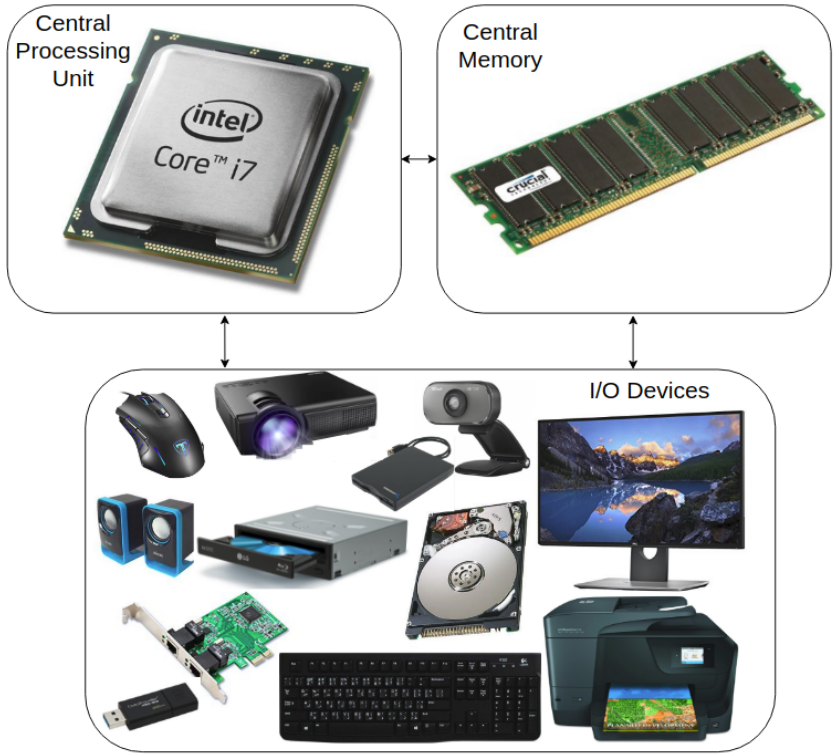
These components are connected together in a **motherboard** using electrical lines (**system bus:** address bus, data bus, control bus), connectors, ports, slots, sockets, etc.

Hardware Components of a Computer System

A computer systems is composed of: (1) Hardware Components, (2) Software Components — Application and System software, and (3) User.

The main **hardware components** are (Von Neumann architectural model):

- Central Processing Unit .
- Central memory — RAM .
- Peripherals — I/O devices.

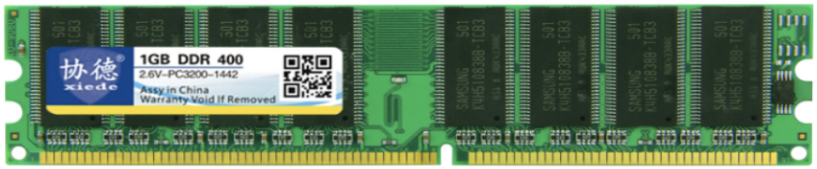


These components are connected together in a **motherboard** using electrical lines (**system bus:** address bus, data bus, control bus), connectors, ports, slots, sockets, etc.

Central Memory — RAM

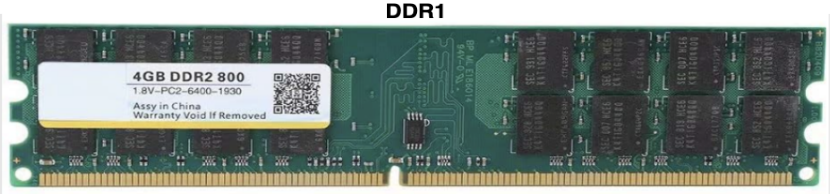
Central Memory — RAM

The **Central memory** or **RAM**, is a volatile type of memory circuit that is used to store information (e.g., text, programs, videos, images, etc).



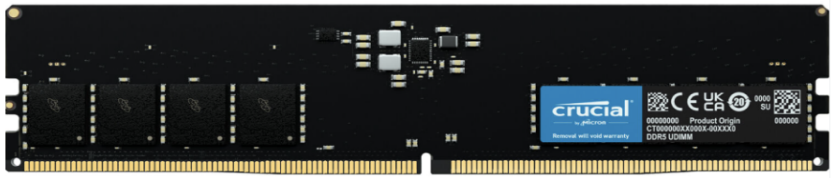
DDR1

DDR3



DDR2

DDR4



DDR5

RAMs are featured by their: **frequency**, **storage capacity**, **data rate**, and **latency**. They r connected to the motherboard thru the **DIMM** slots.

There are various technologies: SDR SDRAM, DDR SDRAM, RDRAM, DDR2, DDR3, DDR4, and DDR5, LPDDR, and GDDR.

Central Memory — RAM (Parameters)

For a better understanding, consider the central memory as a vertical furniture of drawers. Each drawer is identified by a lebel on `M` bits, and stores `W` bits. With this abstraction, the parameters of a central memory are:

- **Address Size.** The number of bits `M` to express one address. With `M` we can address 2^M locations (drawer).
- **Word Size.** The number of bits `W` stored per one memory location. If `W=8`, the memory is byte-addressable.
- **Control Signal.** Read operation or write operation.

Example. A memory with `(M,W)=(4,8)` is a 16B memory.

When the value of `M` is greater than 10, multiples of the bytes (octets) are used: 2^{10} is Kilo, 2^{20} is Mega, 2^{30} is Giga, 2^{40} is Tera, 2^{50} Peta, etc.

Question: What is the size of a byte-addressable memory with `M`=39?

Central Memory — RAM (Parameters)

For a better understanding, consider the central memory as a vertical furniture of drawers. Each drawer is identified by a lebel on `M` bits, and stores `W` bits. With this abstraction, the parameters of a central memory are:

- **Address Size.** The number of bits `M` to express one address. With `M` we can address 2^M locations (drawer).
- **Word Size.** The number of bits `W` stored per one memory location. If `W=8`, the memory is byte-addressable.
- **Control Signal.** Read operation or write operation.

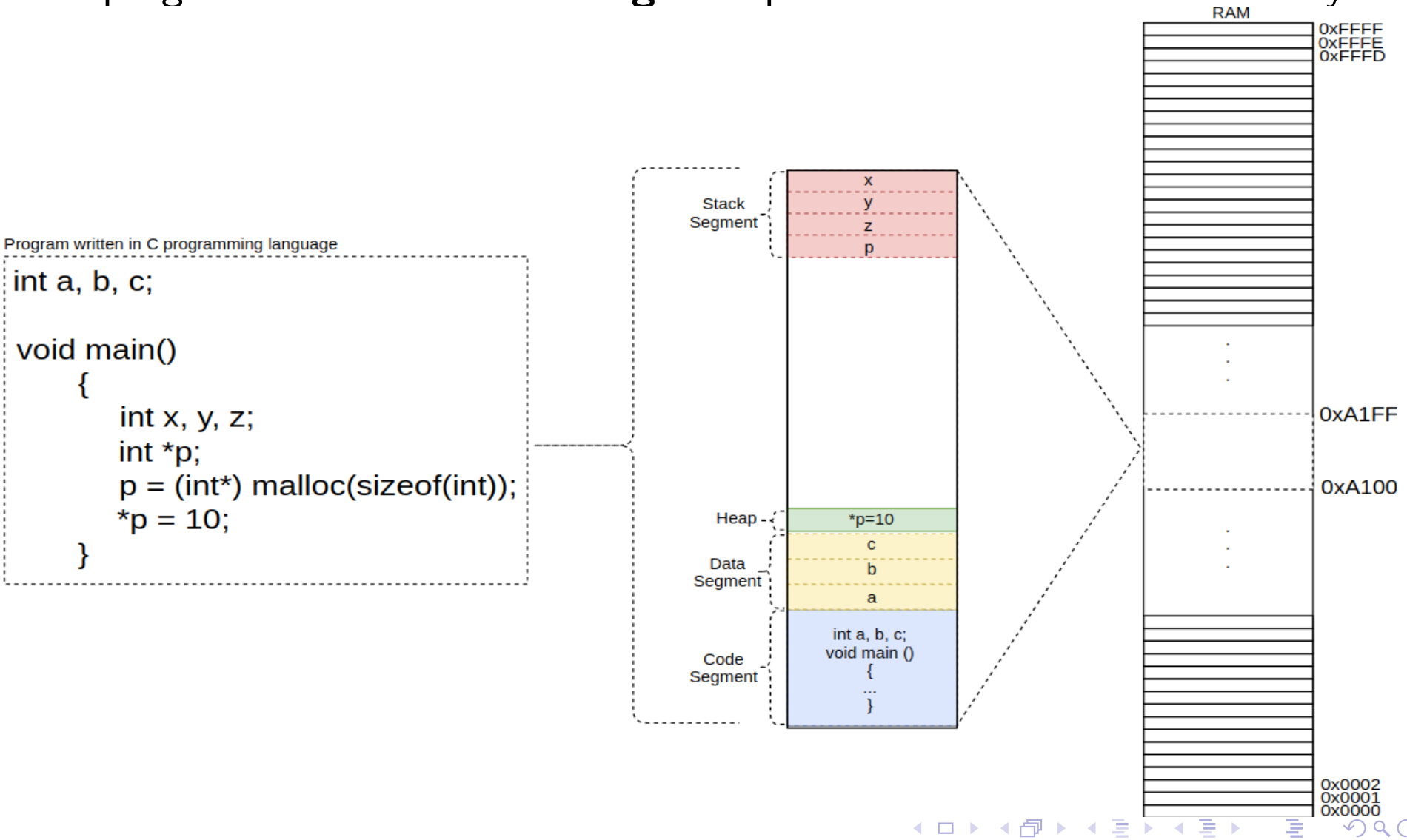
Example. A memory with `(M,W)=(4,8)` is a 16B memory.

When the value of `M` is greater than 10, multiples of the bytes (octets) are used: 2^{10} is Kilo, 2^{20} is Mega, 2^{30} is Giga, 2^{40} is Tera, 2^{50} Peta, etc.

Question: What about a word-addressable memory `(M,W)=(24,16)`?

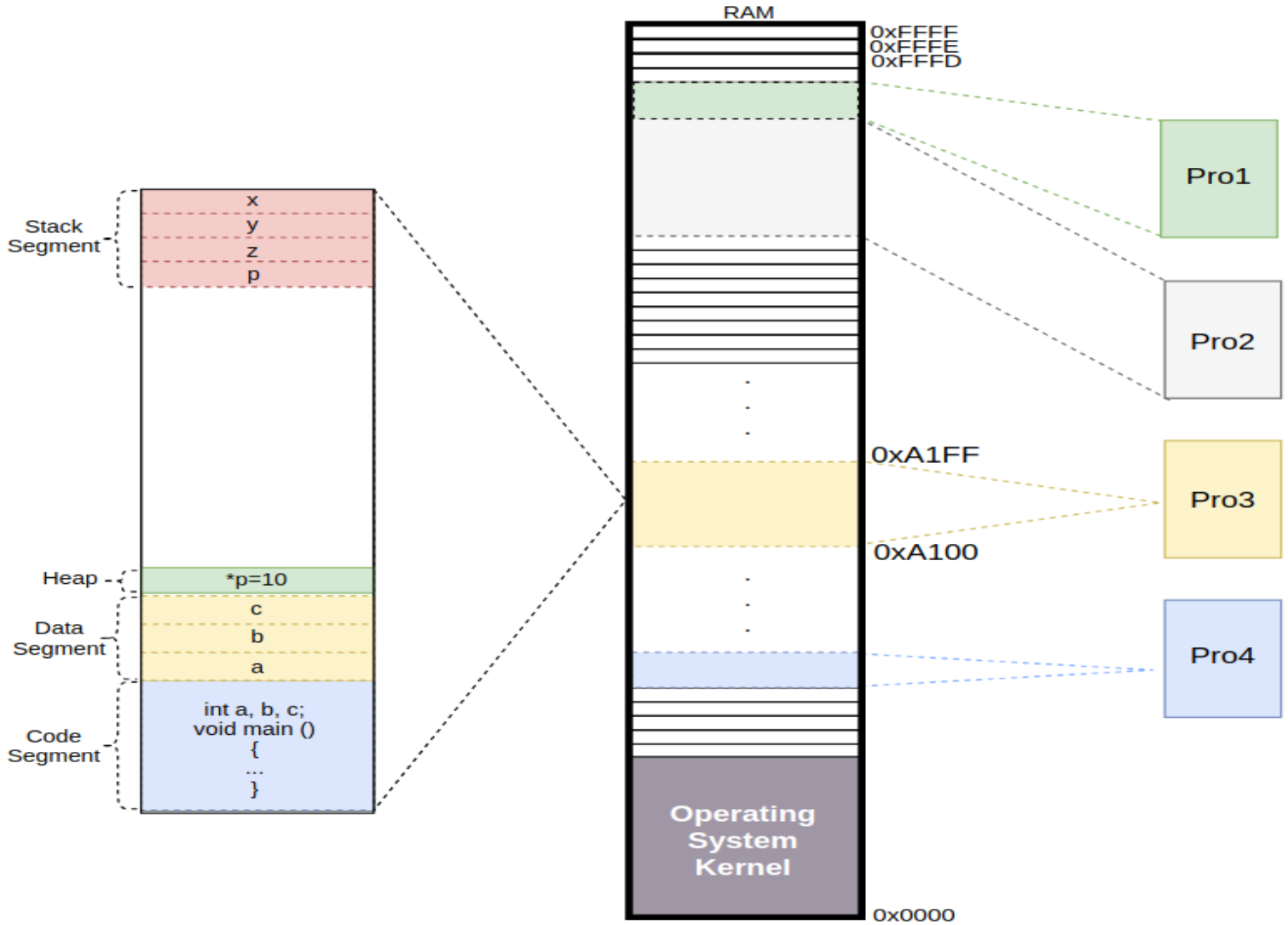
Central Memory — RAM (Old-fashion allocation)

Each program is allocated a **contiguous** partition in the central memory:



Central Memory — RAM (Old-fashion allocation)

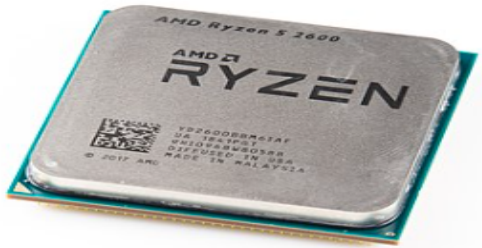
Each program is allocated a **contiguous** partition in the central memory:

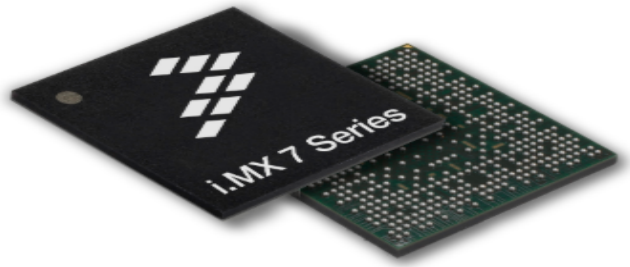


Central Processing Unit

CPU — Central Processing Unit

Processor or **Microprocessor**, is a complex electronic circuit designed to execute machine instructions at a very high speed.





Frequency: The number of clock cycles per second (Hz)
Word size: The size of the processing unit (4, 8, 16, 32, 64 bits)
Internal cache: L1, L2, and L3 SRAM with different sizes (KB/MB)
Registers: Number of registers (ax, bx, ..., or, r0, r1, ..., or, x0, x1, ...)
Number of cores: Multiple processing cores inside the processor
Hyper-threading: Number of threads per core (♯ logical cores)
IPC: The average number of instructions per clock cycle
Instruction Set: The set of possible instructions
Architecture: RISC vs CISC processor.

CPU Specs (`lscpu`)

On a GNU/Linux system, you can see the computer's CPU features by executing the command `lscpu`:

```
Architecture:            x86_64
CPU op-mode(s):          32-bit, 64-bit
Byte Order:              Little Endian
CPU(s):                  8
On-line CPU(s) list:     0-7
Thread(s) per core:      2
Core(s) per socket:      4
Socket(s):               1
NUMA node(s):            1
Vendor ID:               AuthenticAMD
CPU family:              21
Model:                   2
Model name:              AMD FX(tm)-8350 Eight-Core Processor
Stepping:                0
CPU MHz:                 1400.000
CPU max MHz:             4000.0000
CPU min MHz:             1400.0000
BogoMIPS:                8000.05
Virtualization:          AMD-V
L1d cache:               16K
L1i cache:               64K
L2 cache:                2048K
L3 cache:                8192K
NUMA node0 CPU(s):       0-7
Flags:                   fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca
cmov pat pse36 clflush mmx fxsr sse sse2 ht syscall nx mmxext fxsr_opt pdpe1gb r
dtscp lm constant_tsc rep_good nopl nonstop_tsc extd_apicid aperfmperf pni pclmu
lqdq monitor ssse3 fma cx16 sse4_1 sse4_2 popcnt aes xsave avx f16c lahf_lm cmp_
legacy svm extapic cr8_legacy abm sse4a misalignsse 3dnowprefetch osvw ibs xop s
kinit wdt lwp fma4 tce nodeid_msr tbm topoext perfctr_core perfctr_nb cpb hw_pst
ate ssbd vmmcall bmi1 arat npt lbrv svm_lock nrip_save tsc_scale vmcb_clean flus
hbyasid decodeassists pausefilter pfthreshold
```

CPU Registers

A **register** It is a very high speed one-word memory inside the CPU, generally made of electronic latches. The CPU holds contains registers for different purposes:

General-purpose Registers:

AX	Accumulator register: Used for arithmetic and logical operations
BX	Base register: Used for addressing memory locations
CX	Counting registers: used in loops
DX	Data register: holds operands and results of operations

These registers are denoted eax, ebx, ecx, edx on 32-bit CISC processors, as rax, rbx, ... on 64-bit CISC processors, as r0, r1, r2, ... on 32-bit RISC processors, and x0, x1, ... on 64-bit RISC processors.

Indexing Registers:

BP	Base Pointer: Points stack frames and helps access local variables
SP	Stack Pointer: Used to point the top of the stack
IP	Instruction Pointer: Points the address of the next instruction (PC)
LR	Link register: Holds the return address (during function calls)

These registers are denoted ebp, esp, eip, elr on 32-bit CISC processors, as rbp, rsp, ... on 64-bit CISC processors, as r13, r14, r15, ... on 32-bit RISC processors, and bp, sp, pc, lr, ... on 64-bit RISC processors.

CPU Registers (Con't)

A **register** It is a very high speed one-word memory inside the CPU, generally made of electronic latches. The CPU holds contains registers for different purposes:

Status and Memory Registers:

FL	Flags[1]: Holds various flags (carry flag, zero flag, sing flag, interrupt flag, parity flag, overflow flag, ... trap flag)
SR	Segment registers point base address of segments
MAR	Memory address register: Holds address to read from/write to RAM.
MBR	Memory buffer register: Holds data 2b written, or just read from RAM.
CRI	Holds the current instruction to be decoded.
ISRSP	Used to manage nested interrupts (points interrupt stack in kernel space)

[1] You may find the PSW (Program Status Word) register, which grouped PC register and other status/flag registers in IBM System/360.

CPU Instruction Set

Processor Instruction Set: is the set of machine instructions that a given processor is able to decode and execute (supported instructions).

In this course, most of the examples given in assembly language use the x86 instructions set (on a 16-bit computer, e.g., Intel 8086 microprocessor):

`MOV Ax, 0x1111`	Moves the value 0x1111 to register Ax
`MOV Al, 0x11`	Moves the value 0x11 to the lower part of Ax
`INC Ax`	Increments the current value stored in Ax
`DEC Ax`	Decrements the current value stored in Ax
`DIV Bx`	Divides the content of Ax by Bx and store it in Ax
`MUL Cx`	Multiplies the content of Ax with Cx and store it in Ax
`PUSH Ax`	Pushes the content of Ax onto the stack (SP=SP-2)
`POP Ax`	Pops the top of the stack (pointed by SP) into Ax
`JMP ABCD`	Jumps to label ABCD
`CMP Ax, 1`	Compares the content of Ax with value 1 and sets the Flags
`JE ABCD`	Jumps to ABCD if Ax was equal to 1 (previous operation)
`LOOP ABCD`	Jumps to ABCD and decrements the Cx while Cx is non-zero
`HLT`	Terminates the program

CPU Instruction Set (Con't)

Each instruction in the assembly language is transformed into a **machine code** by the assembler program. And each instruction in machine code is generally composed of an **Operation code** (a.k.a., opcode) and **operand**.

```
MOV Ax, [0xF565]  |  JMP Label  |  INC Ax  |  ADD Ax, Bx  |  MOV Ax, Bx
```

Some instructions just consists of an **operation code**:

```
NOP  |  HLT  |  RET  |  CLI  |  STI
```

Hardware Components (CPU — Central Processing Unit)

Consider the following x86-assembly program (Result stored in `Ax`):

```
Start:
MOV Ax, 0X000F
MOV Bx, 0X0014
ADD Ax, Bx
MOV Cx, 0x0005
DIV Cx
INC Ax
MOV [0x335F], Ax
HLT
```

What is the value that will be stored in the memory at address `0x335F` after the above assembly code is executed?

Hardware Components (CPU — Central Processing Unit)

Consider the following x86-assembly program (Result stored in `Ax`):

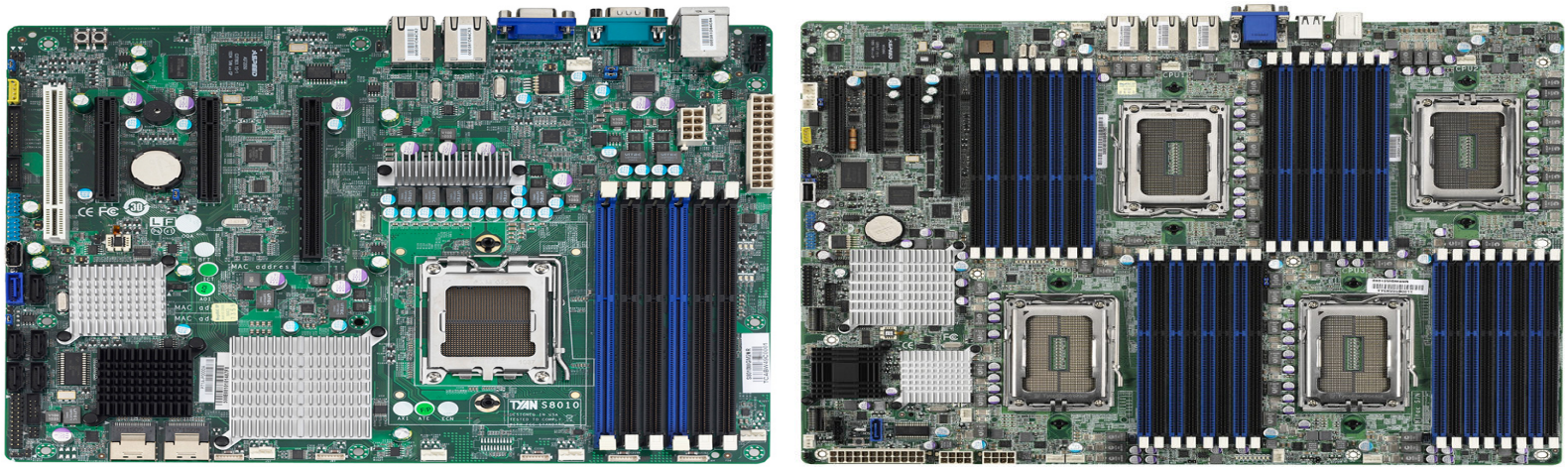
```
Start:
MOV Ax, 0X0001
MOV Cx, 0X0001
T:
 MUL Cx
 INC Cx
 CMP Cx, 0X0005
 JBE T
HLT
```

Try it at: https://yjdoc2.github.io/8086-emulator-web/compile.

Single-processor, Multi-processor, and Multicore Systems

Based on the number of processors, a computer may either be a **single-processor system** or a **multiprocessors system**.

In a **single-processor system**, there is only one CPU (Central Process Unit) capable of executing a general-purposeinstruction set, whereas in a **multiprocessor system**, there are two or more CPUs capable of executing a general-purpose instruction set, and sharing the computer bus, [clock], memory, and peripheral devices. Also, modern processors include multiple computing cors on a single chip, they are called multicore e.g., Intel Core i5, i7, i9 and Apple Silicon M1/2/3.



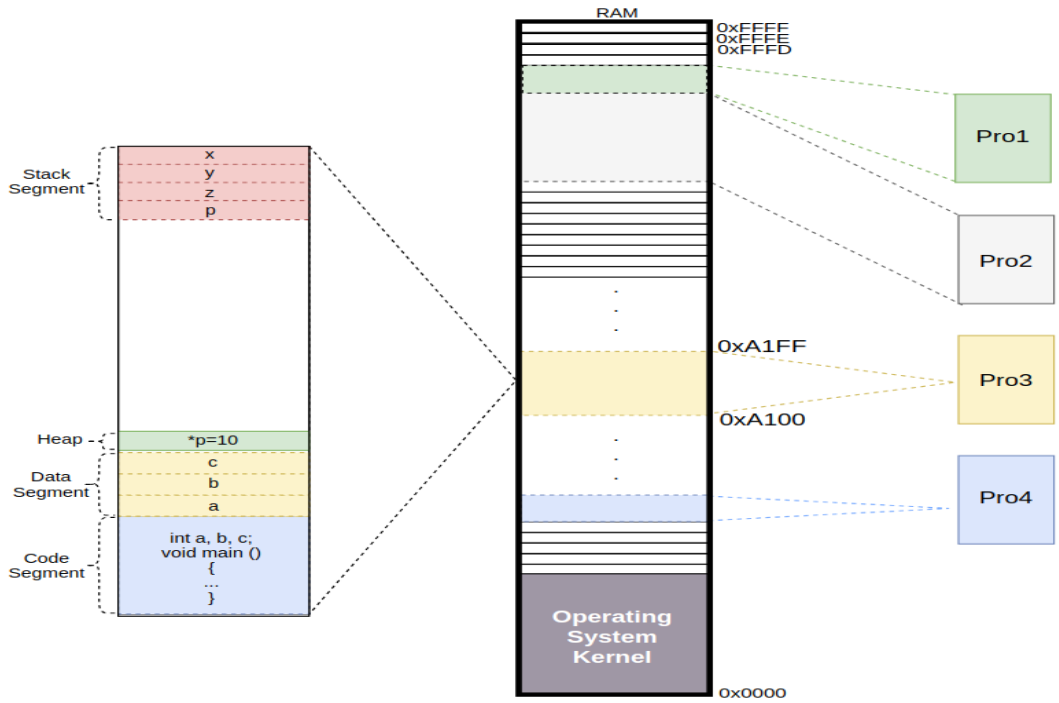
Motherboard of a single-processor (Left) & a multiprocessor computer (Right)

Instruction Execution Cycle

Some Assumptions

Let us consider the following simplifying assumptions:

1. We work on a uniprocessor computer (one single-core processor).
2. We execute one single instruction per clock cycle (IPC=1).
3. No interaction between processes i.e., if `n` processes are executing, no process should affect the execution (behavior) of the other ones.
4. The OS adopts the old-fashioned memory allocation scheme.



Instruction Execution Cycle

Program instructions go through three main phases: **FETCH, DECODE, & EXECUTE** (A.k.a., Machine cycles)

1. **FETCH:** Getting program's next instruction from memory.

   1.1. Get instruction address from IP, place it in the MAR and send a read-request to RAM to retrieve the content of [MAR].

   1.2. After access time, the content is placed on the MBR.

   1.3. Store the instruction code in CIR.

2. **DECODE:** Instruction decoding and operands fetching.

   2.1. The CU decodes and transforms the instruction into a sequence of elementary operations.

   2.2. If the instruction requires operands from memory, the CPU gets them in the MBR after issuing a fetch operand operation.

   2.3. The operand is stored in one of the general purpose registers and the IP is updated.

Instruction Execution Cycle

3. **EXECUTE:** Instruction execution

   3.1. The ALU executes the instruction.

   3.2. The state register is updated (i.e., FLAGS register).

Instructions could be:

- Data transfer: from and to memory or between registers.

  e.g., `Mov Ax, [0x52F3]` or `Mov Ax, Bx`
- Arithmetic operation: Addition, subtraction, division, and product.

  e.g., `Div Ax, Cx`, `Sub Ax, Cx` , or `Add Ax, Bx`
- Logical operation: AND, OR, NOT, and comparison.

  e.g., `Cmp Ax, 0x0001` or `XOR Ax, Ax`
- Sequence control: Branch and tests.

  e.g., `Je X` or `jmp X`

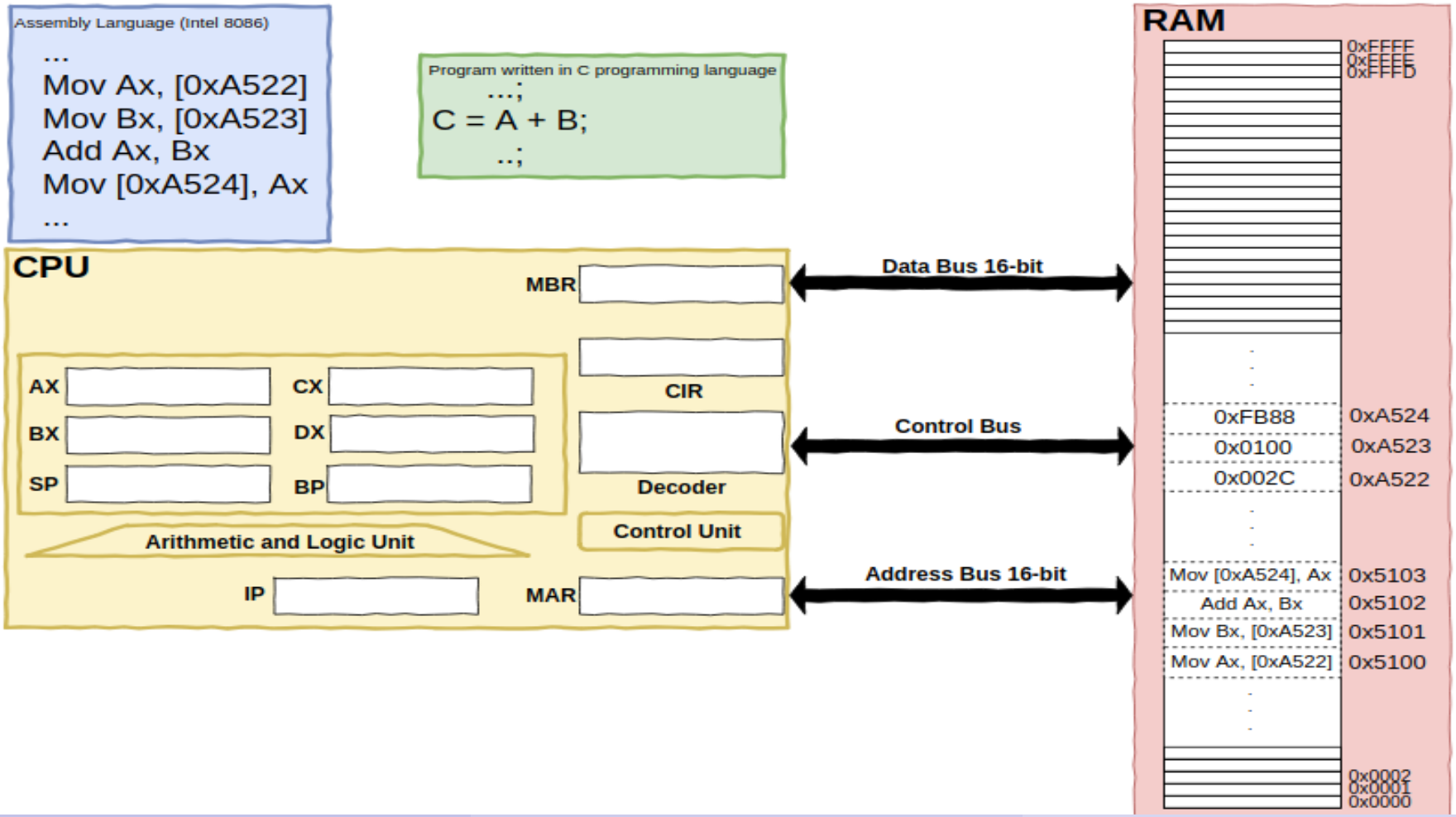
Example (Instruction Execution Cycle)

Recall the assumptions:

- We are working on a CISC computer.
- We are working on a 16-bit CPU (e.g., Intel 8086).
- We are working on a single-core processor.
- The memory is word-addressable.
- The instruction cycles consit of: FETCH, DECODE, & EXECUTE.
- One instruction per clock cycle.
- Operating system adopts the old-fashion memory allocaiton.

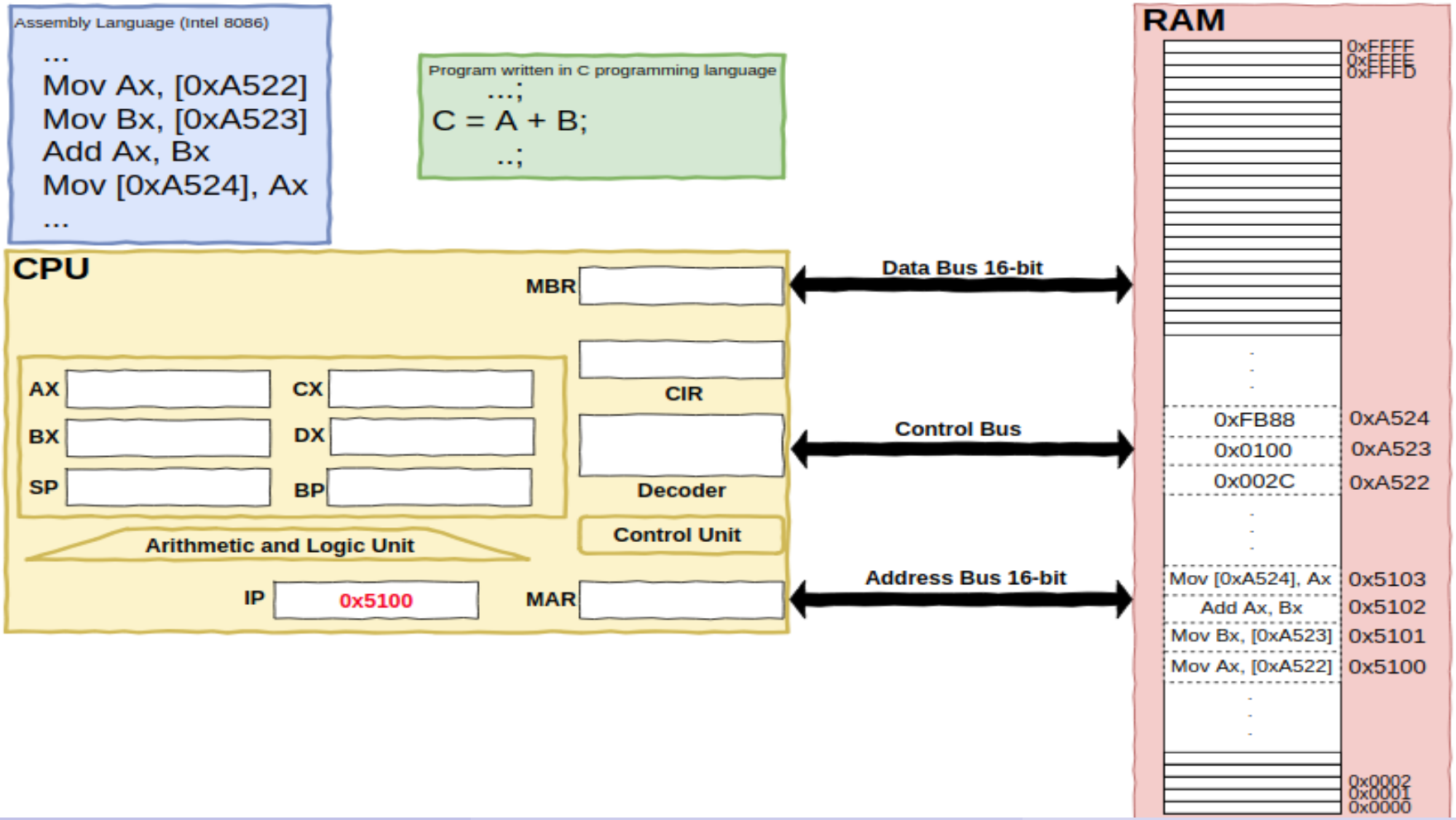
Example (Instructions Execution Cycle)

FETCH



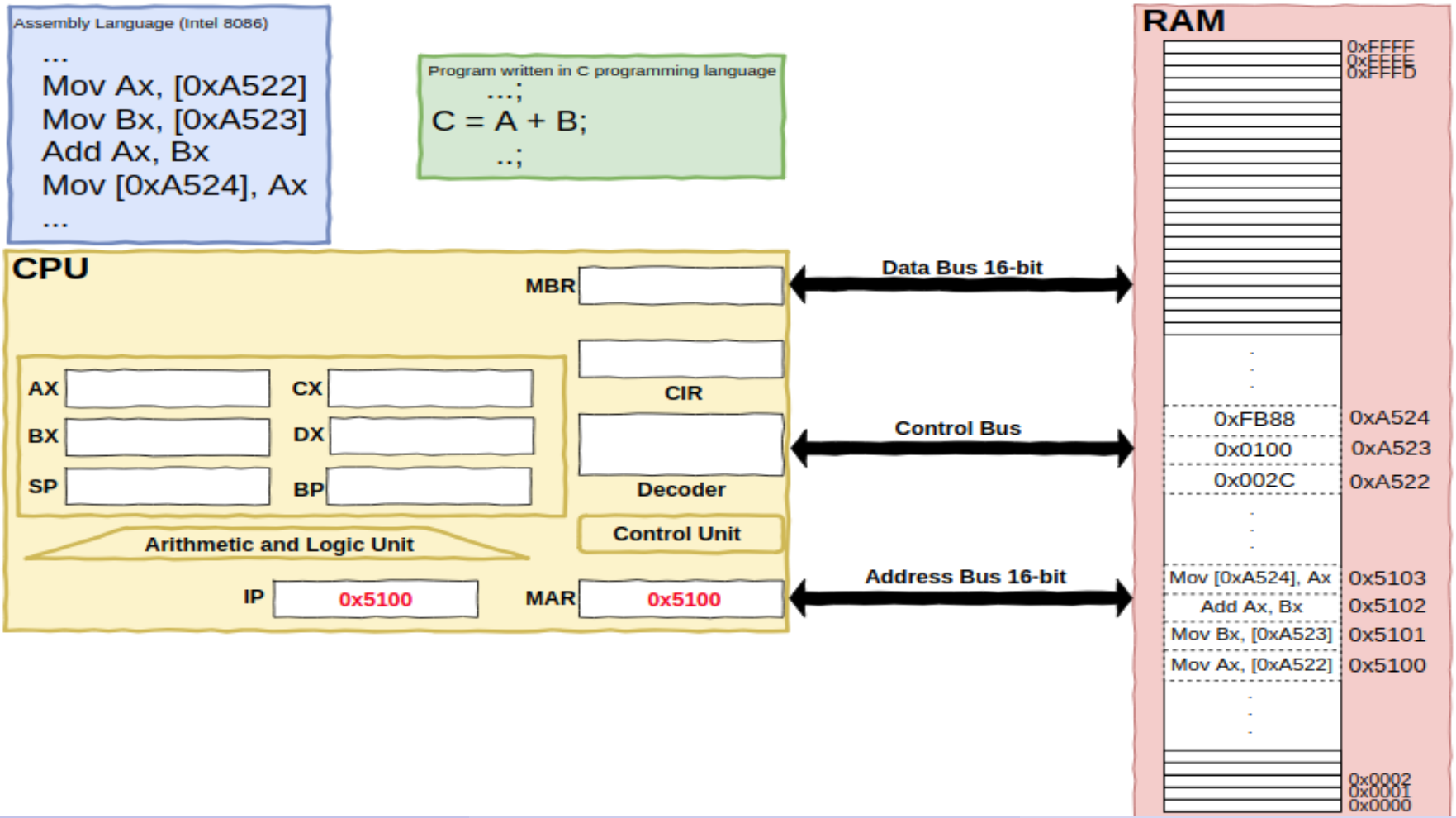
Example (Instructions Execution Cycle)

FETCH



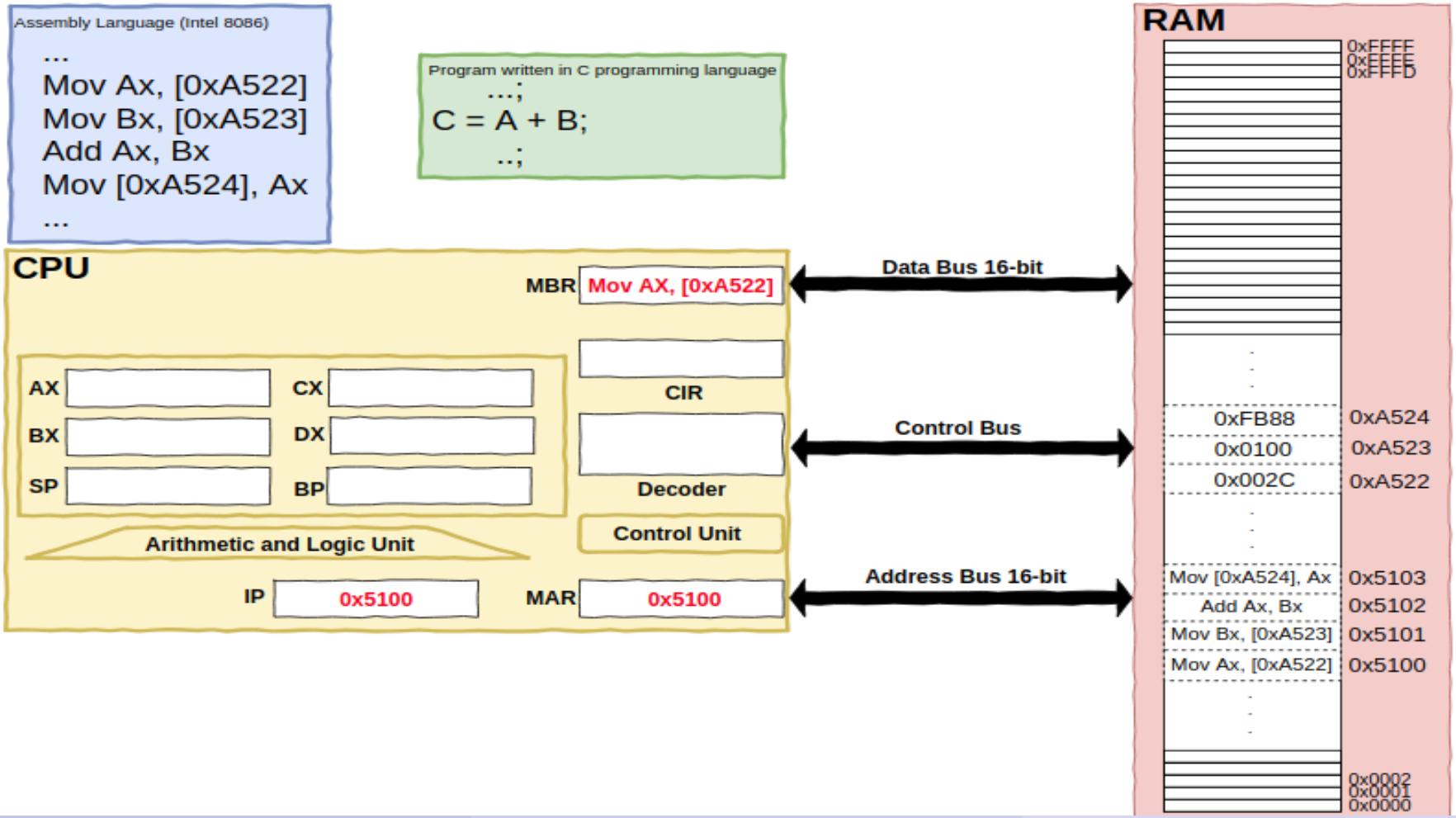
Example (Instructions Execution Cycle)

FETCH



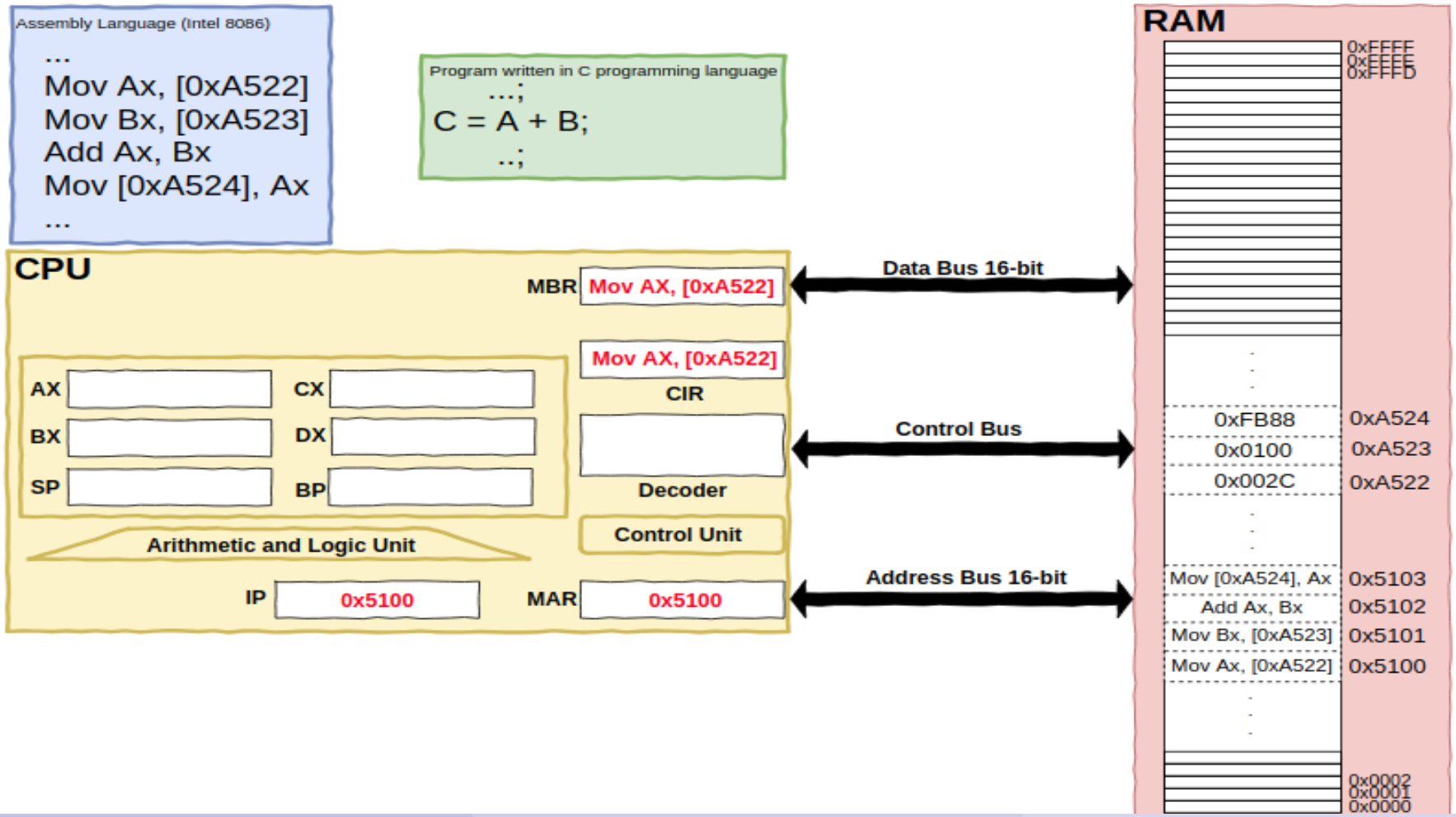
Example (Instructions Execution Cycle)

FETCH



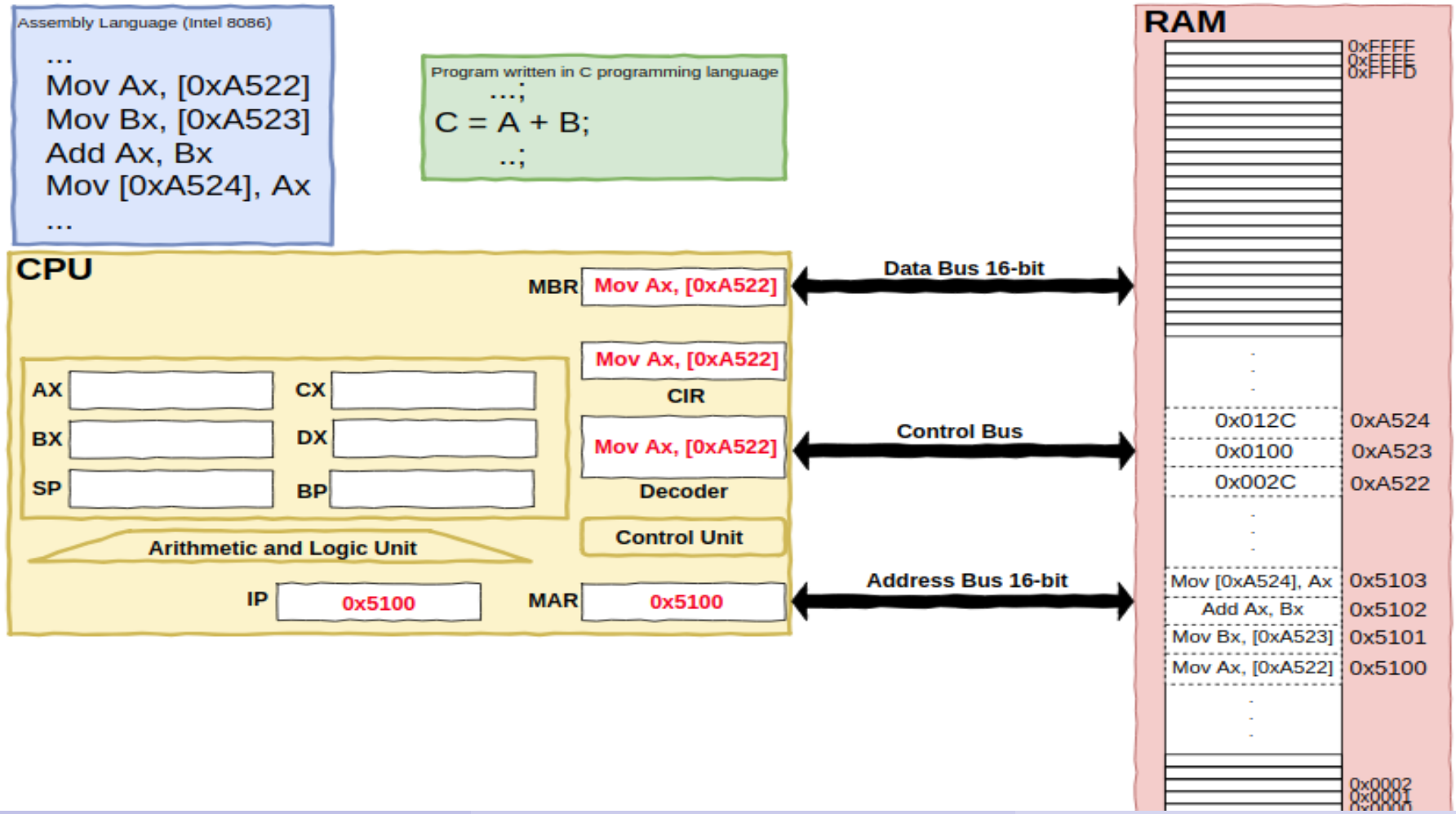
Example (Instructions Execution Cycle)

FETCH



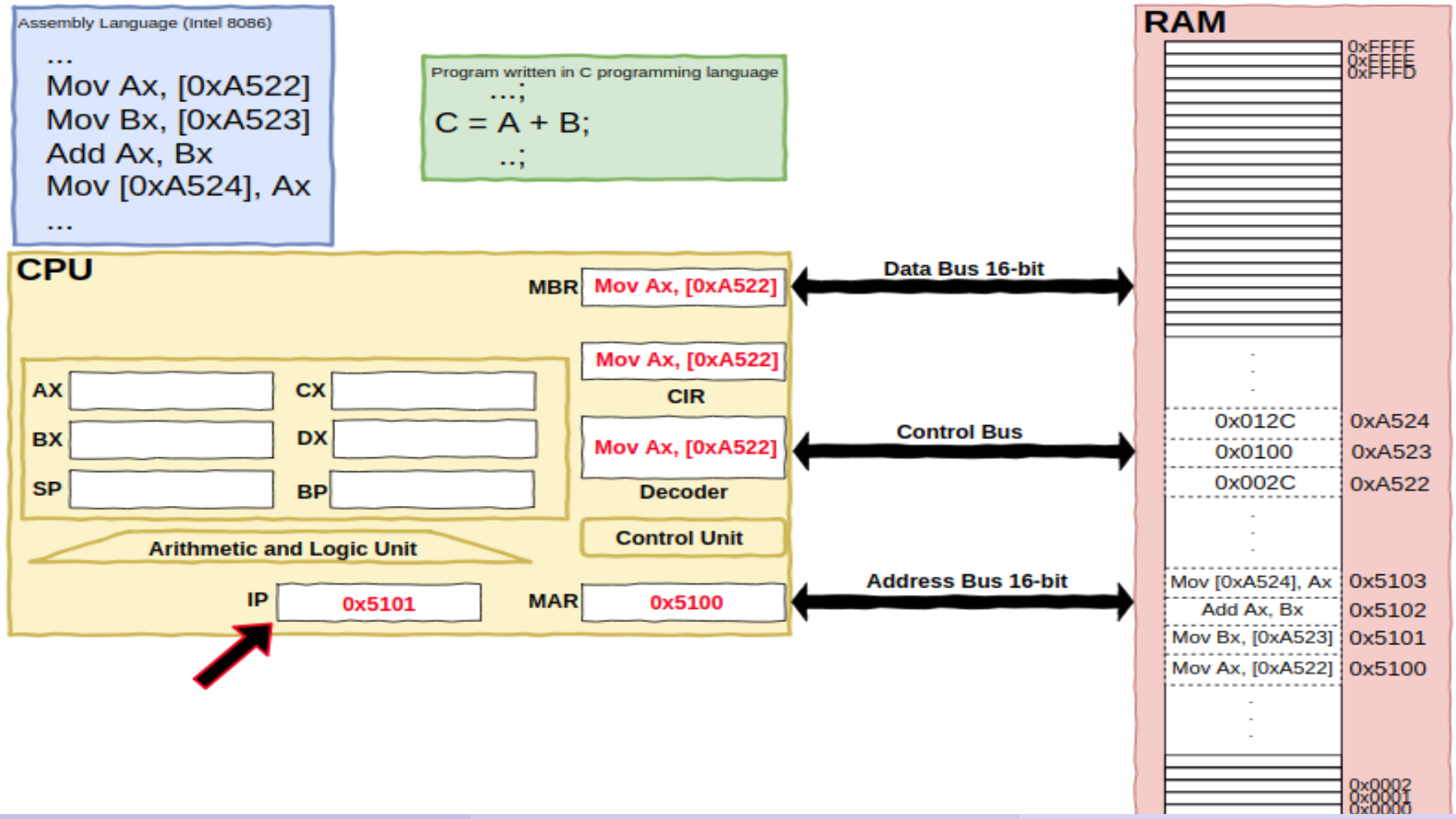
Example (Instructions Execution Cycle)

DECODE



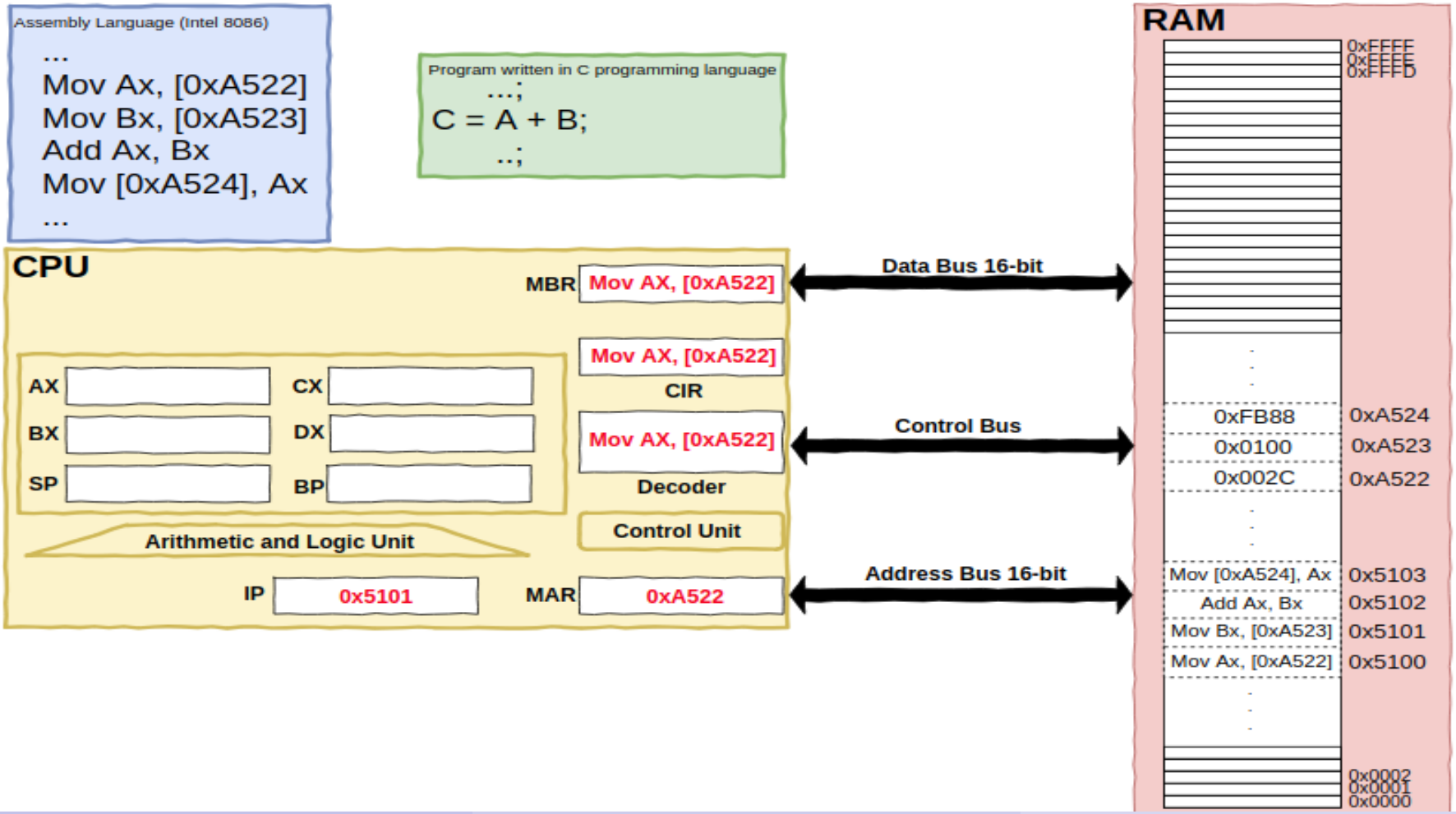
Example (Instructions Execution Cycle)

DECODE



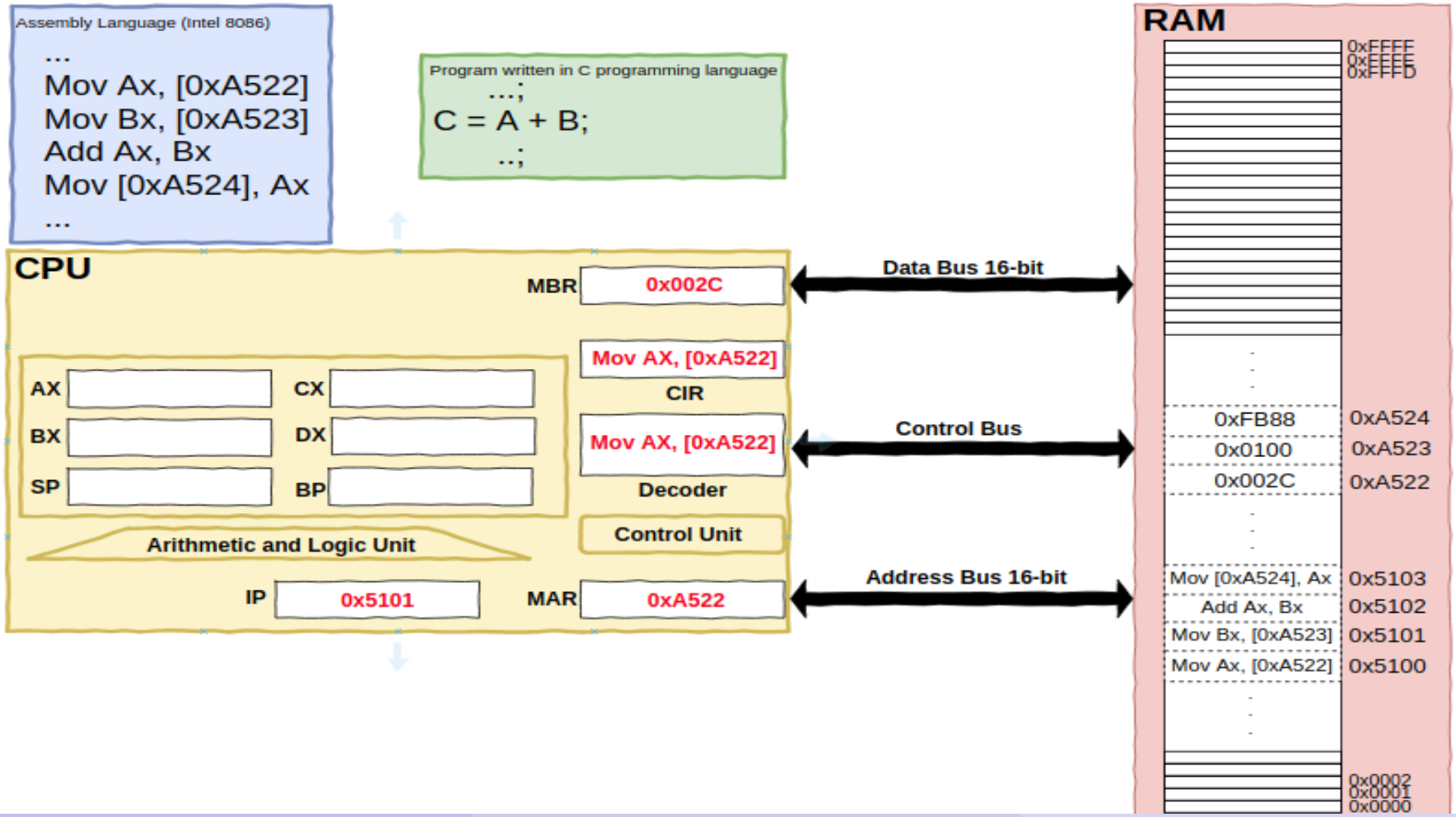
Example (Instructions Execution Cycle)

DECODE



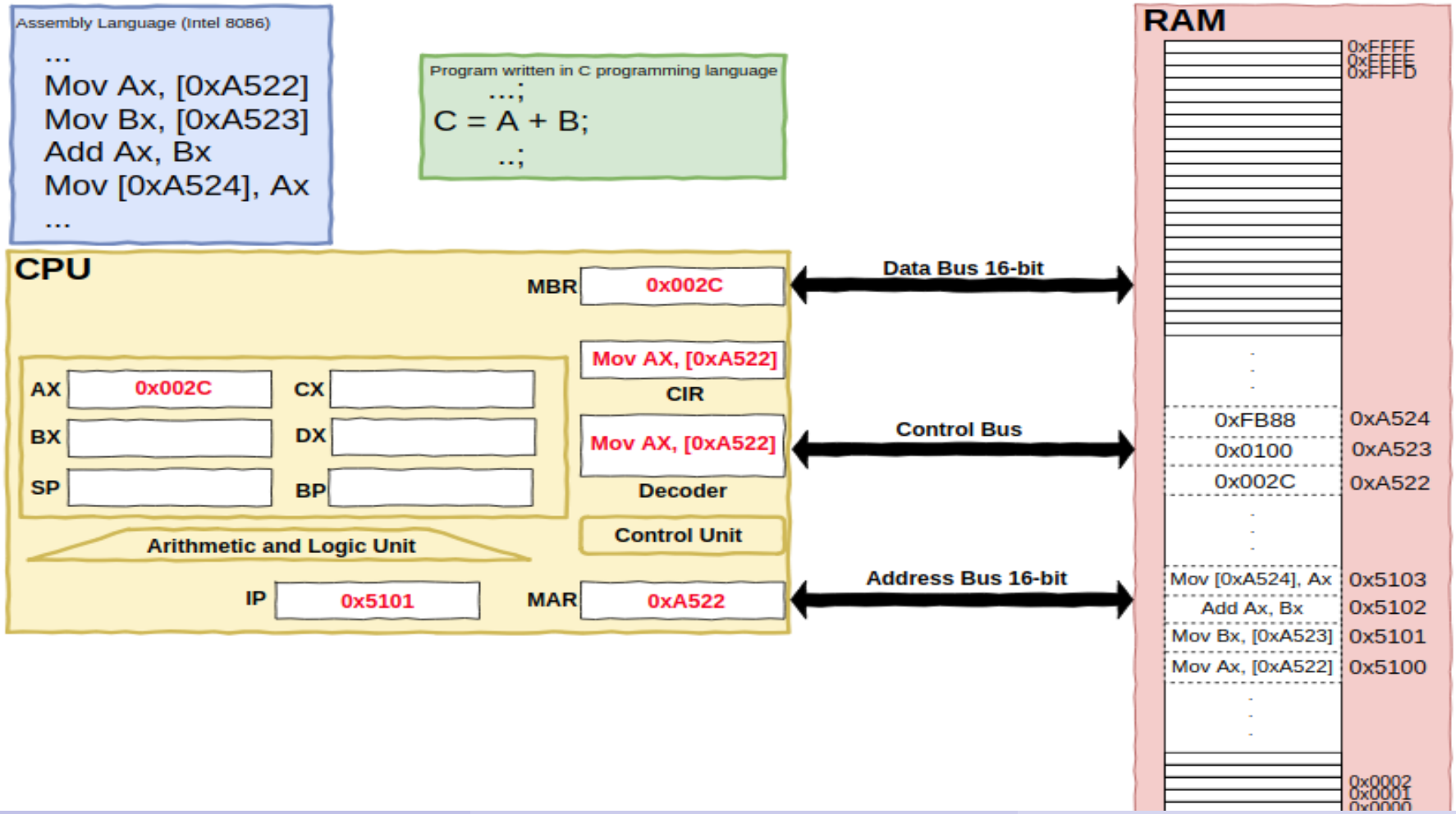
Example (Instructions Execution Cycle)

DECODE



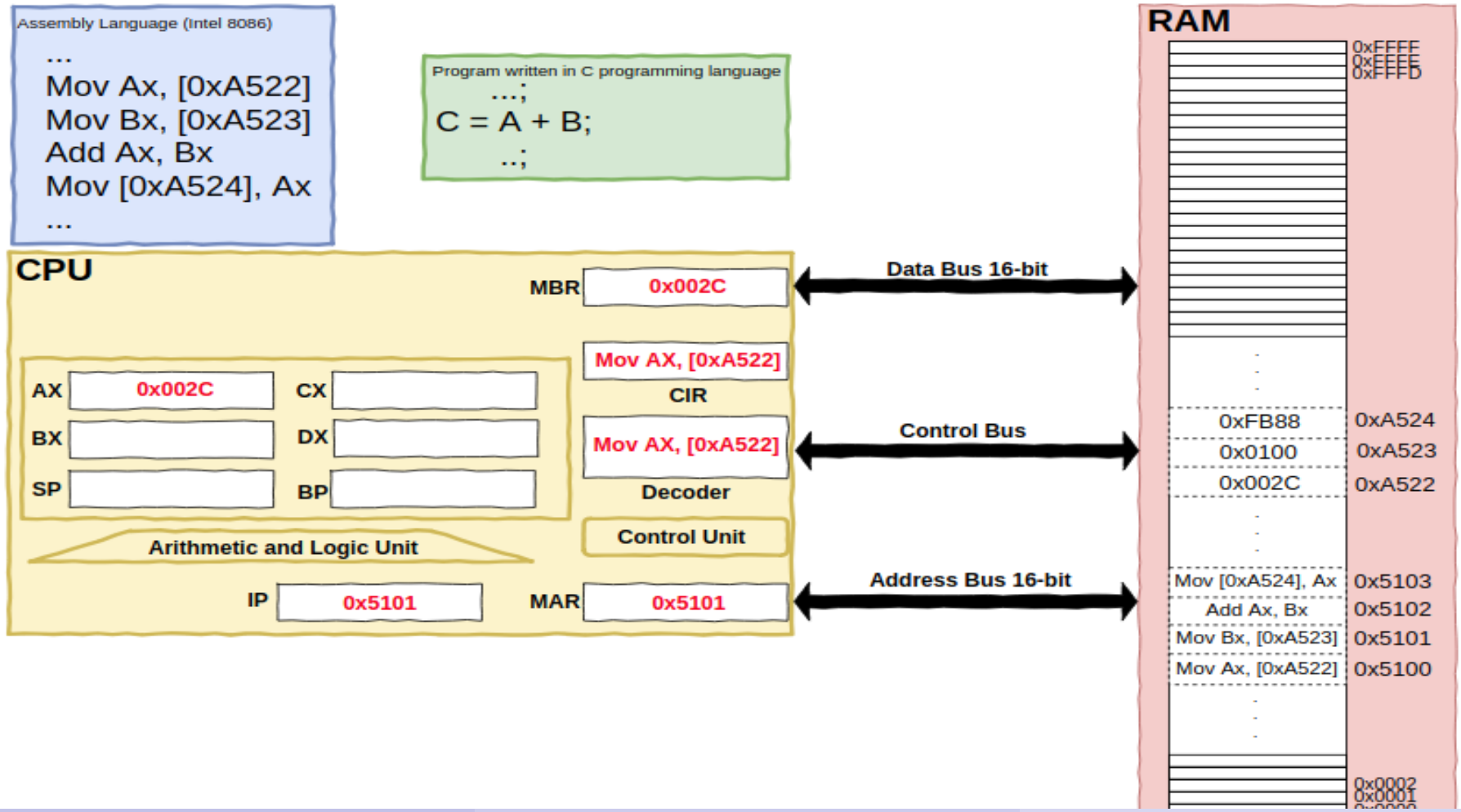
Example (Instructions Execution Cycle)

EXECUTE



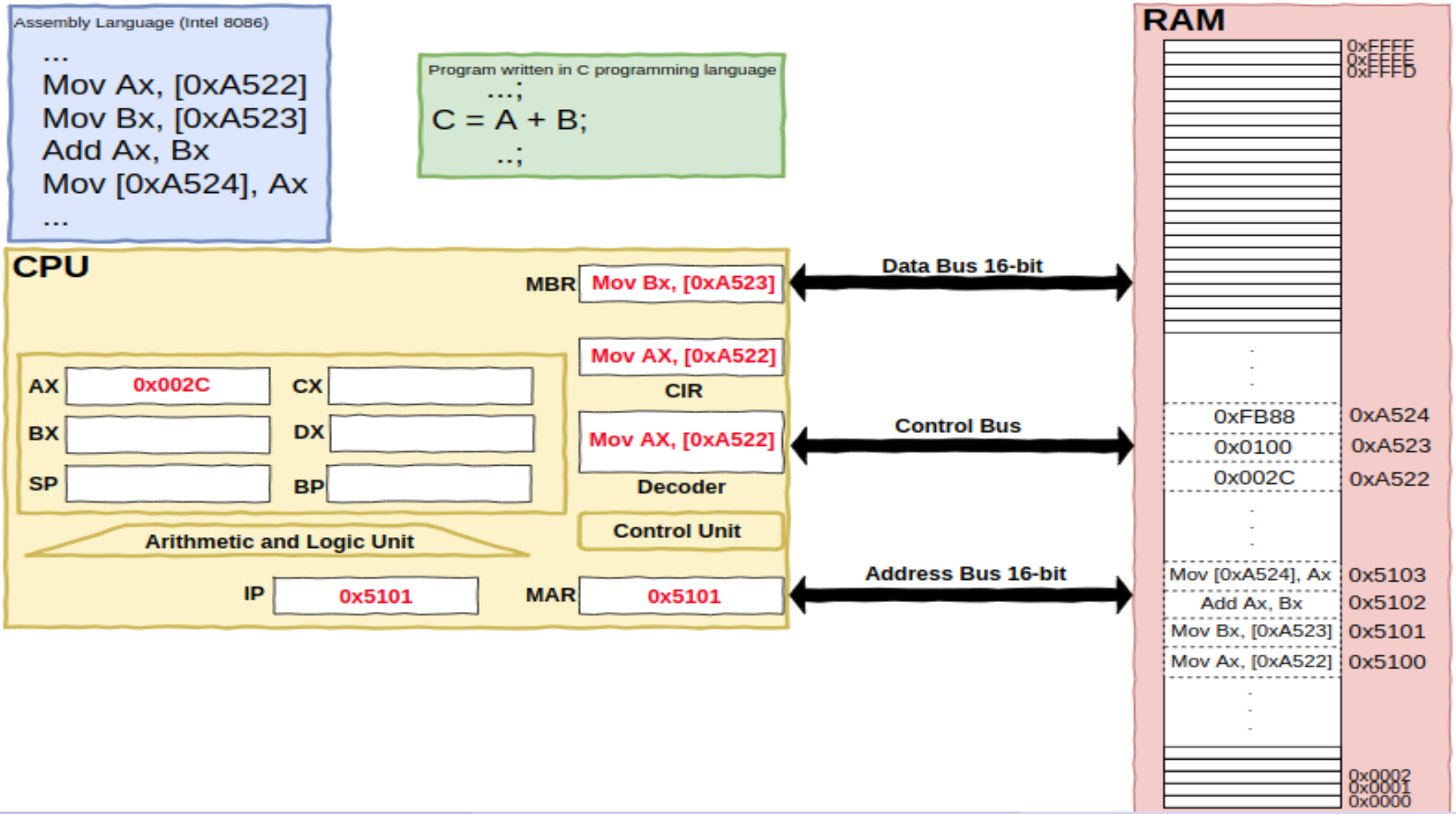
Example (Instructions Execution Cycle)

FETCH



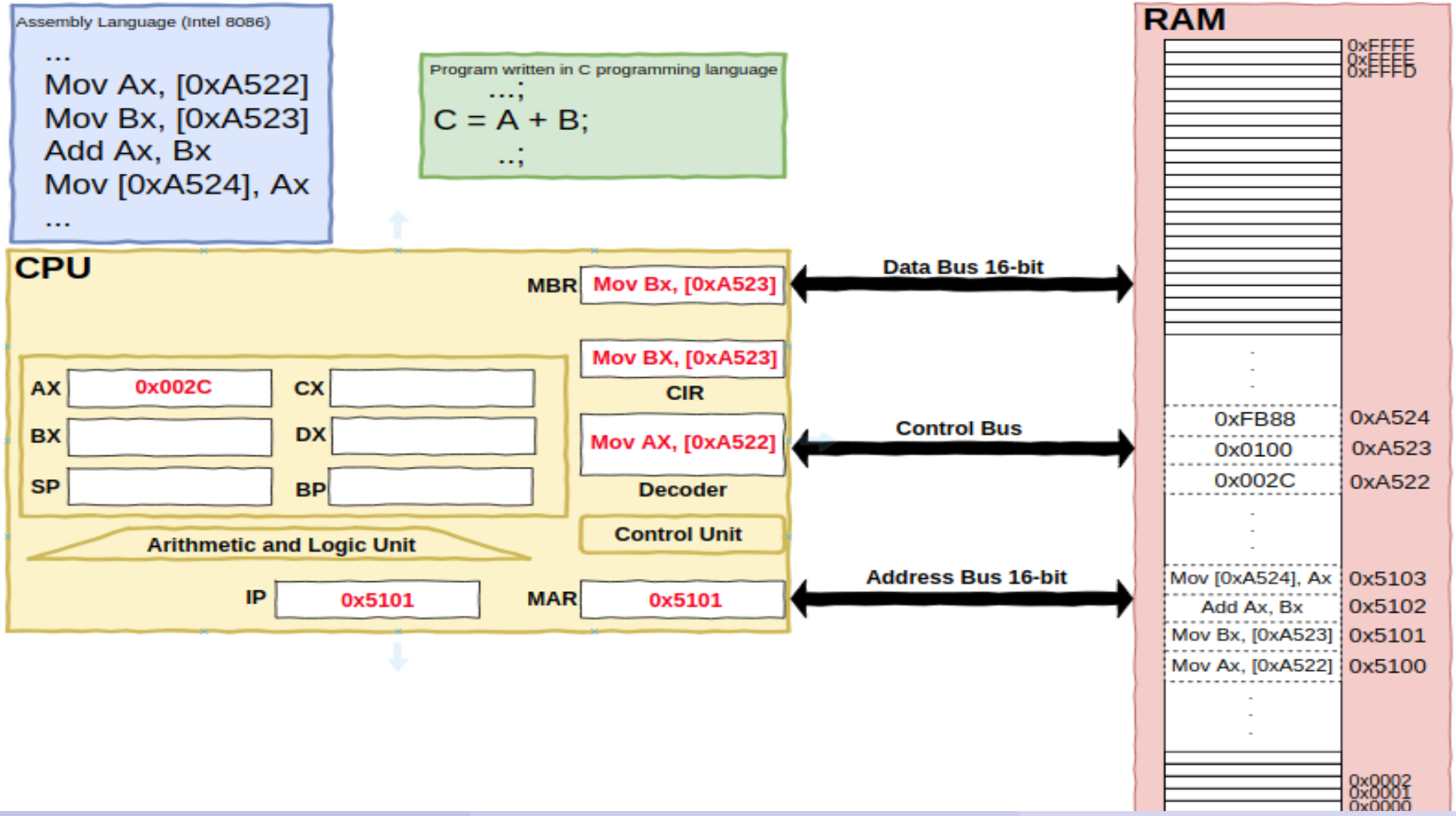
Example (Instructions Execution Cycle)

FETCH



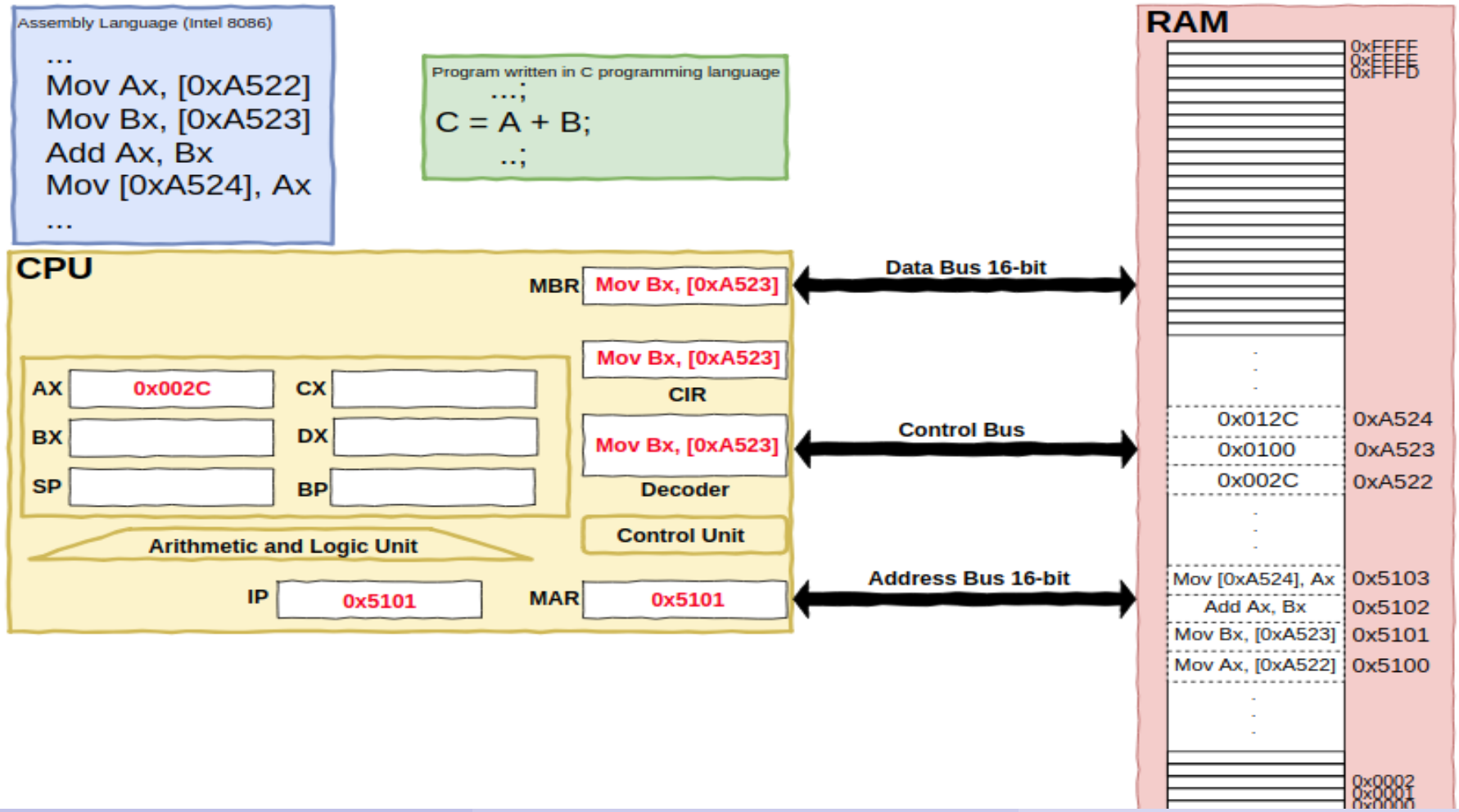
Example (Instructions Execution Cycle)

FETCH



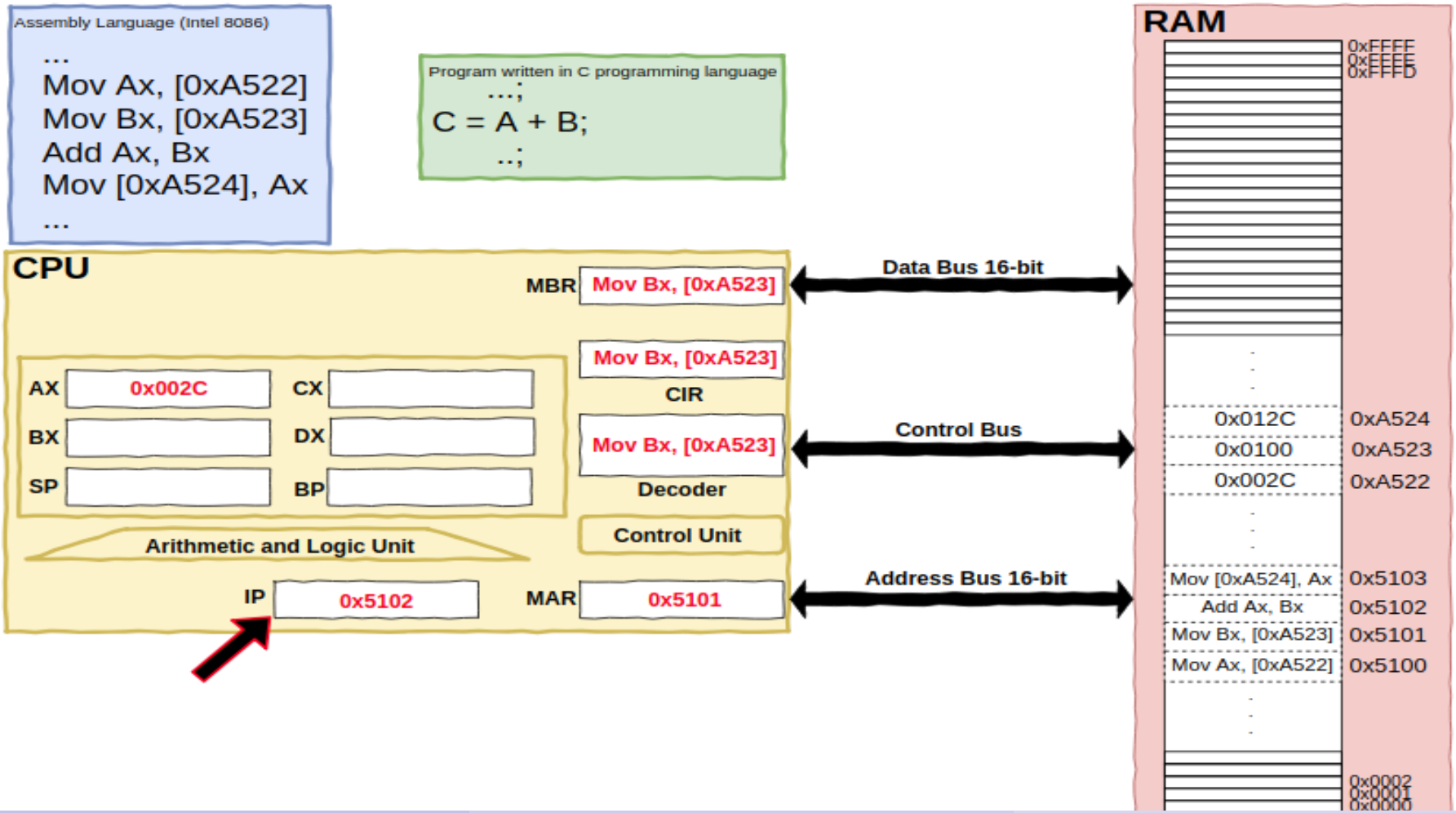
Example (Instructions Execution Cycle)

DECODE



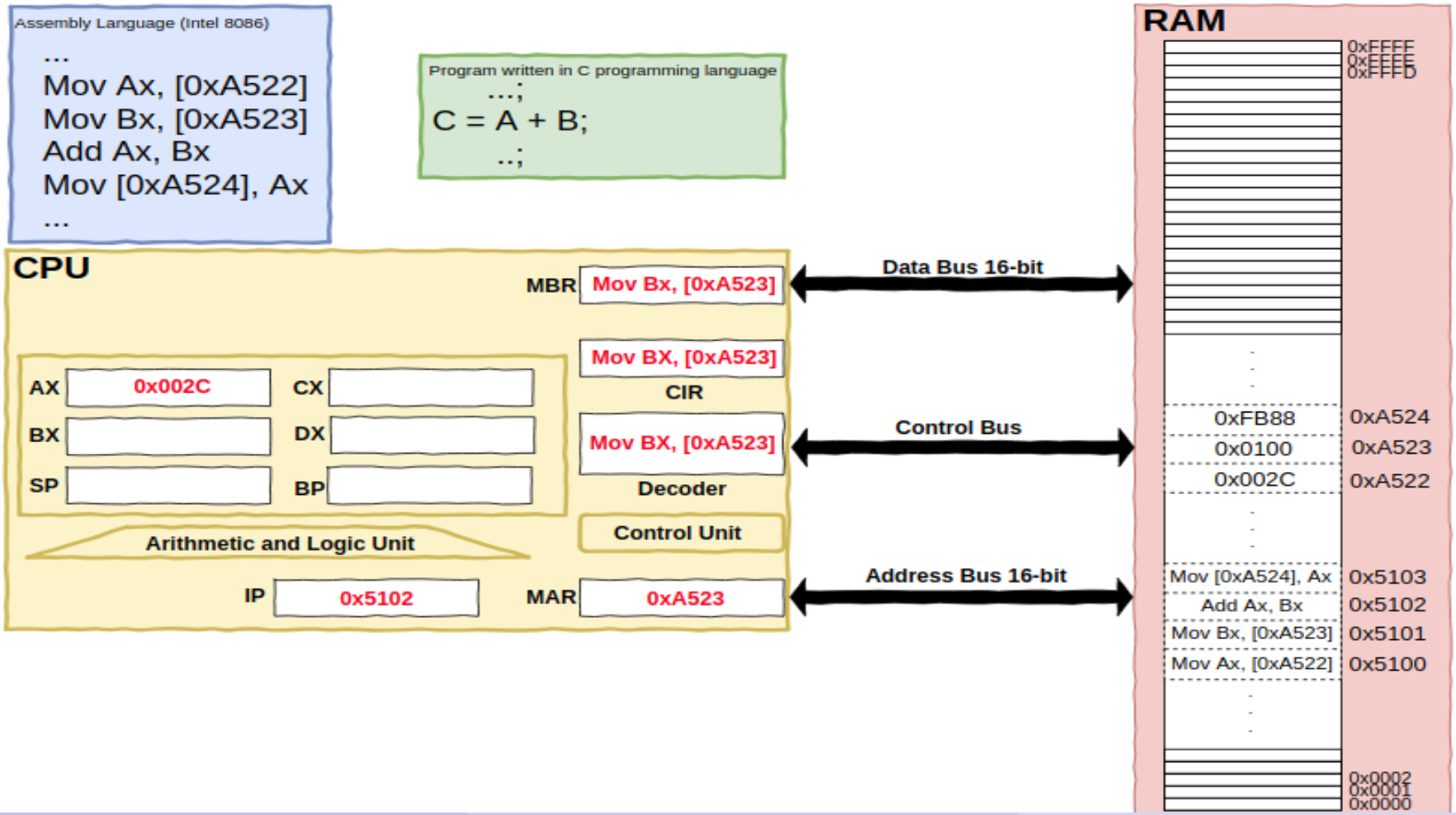
Example (Instructions Execution Cycle)

DECODE



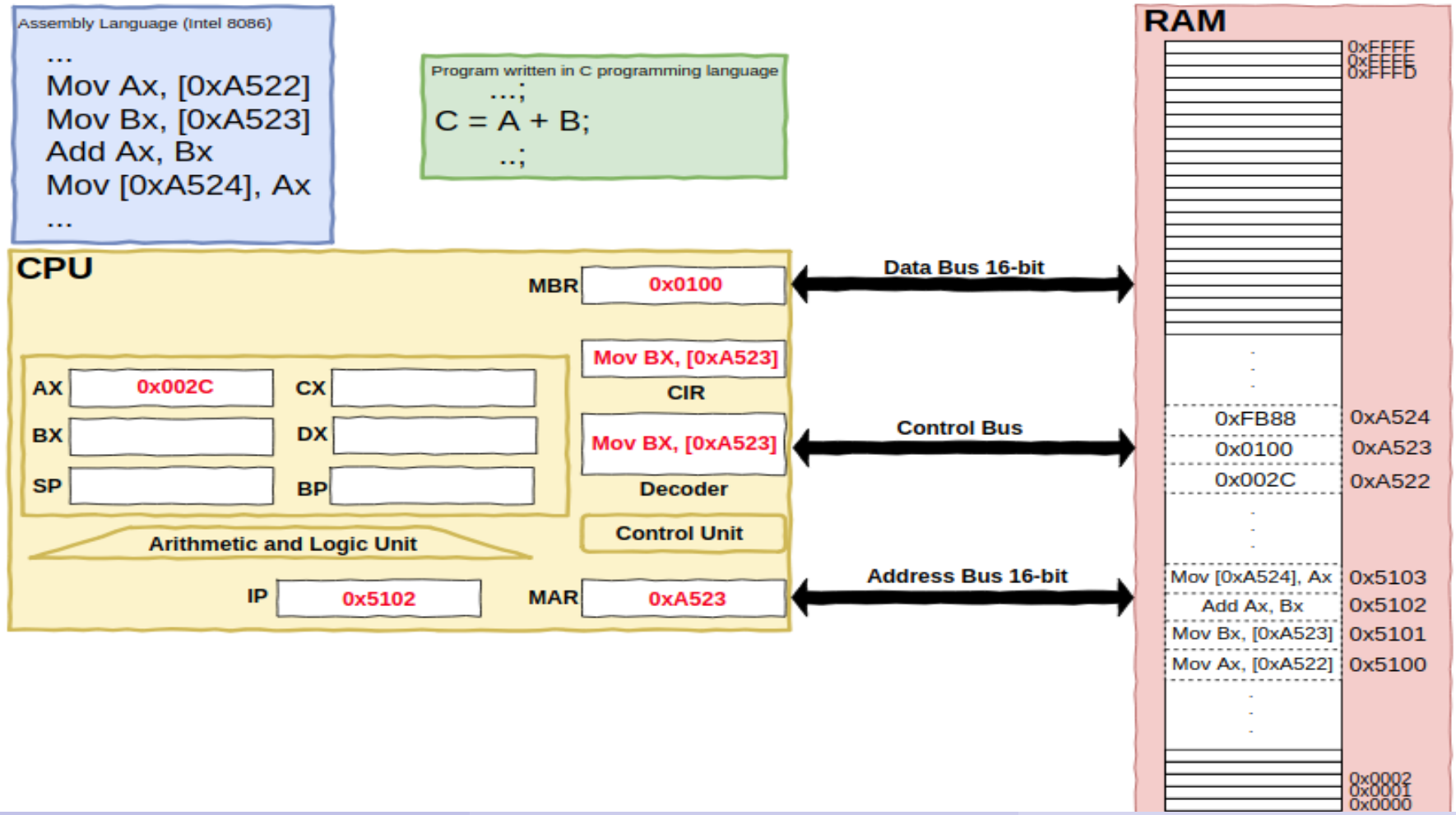
Example (Instructions Execution Cycle)

DECODE



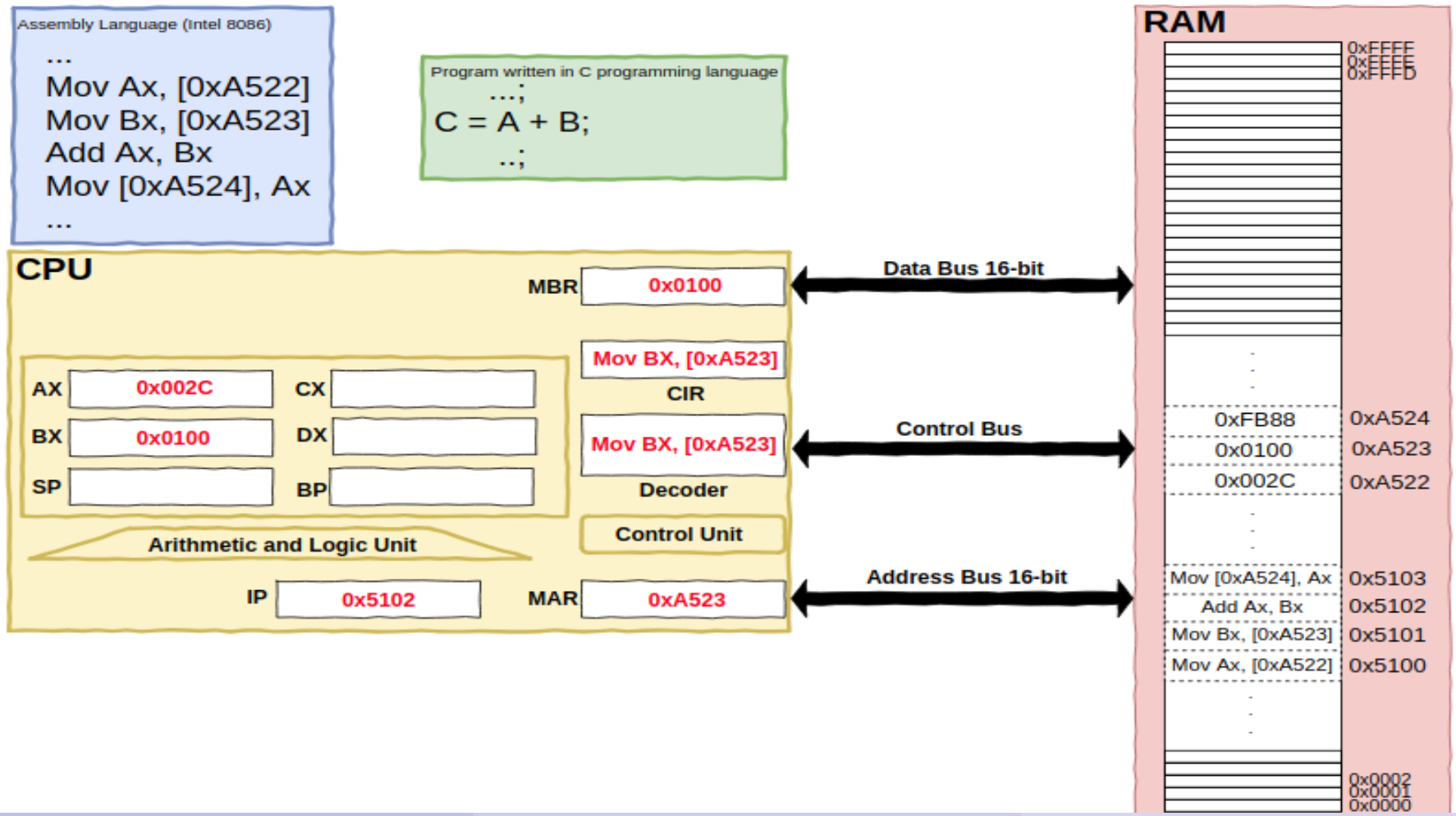
Example (Instructions Execution Cycle)

DECODE



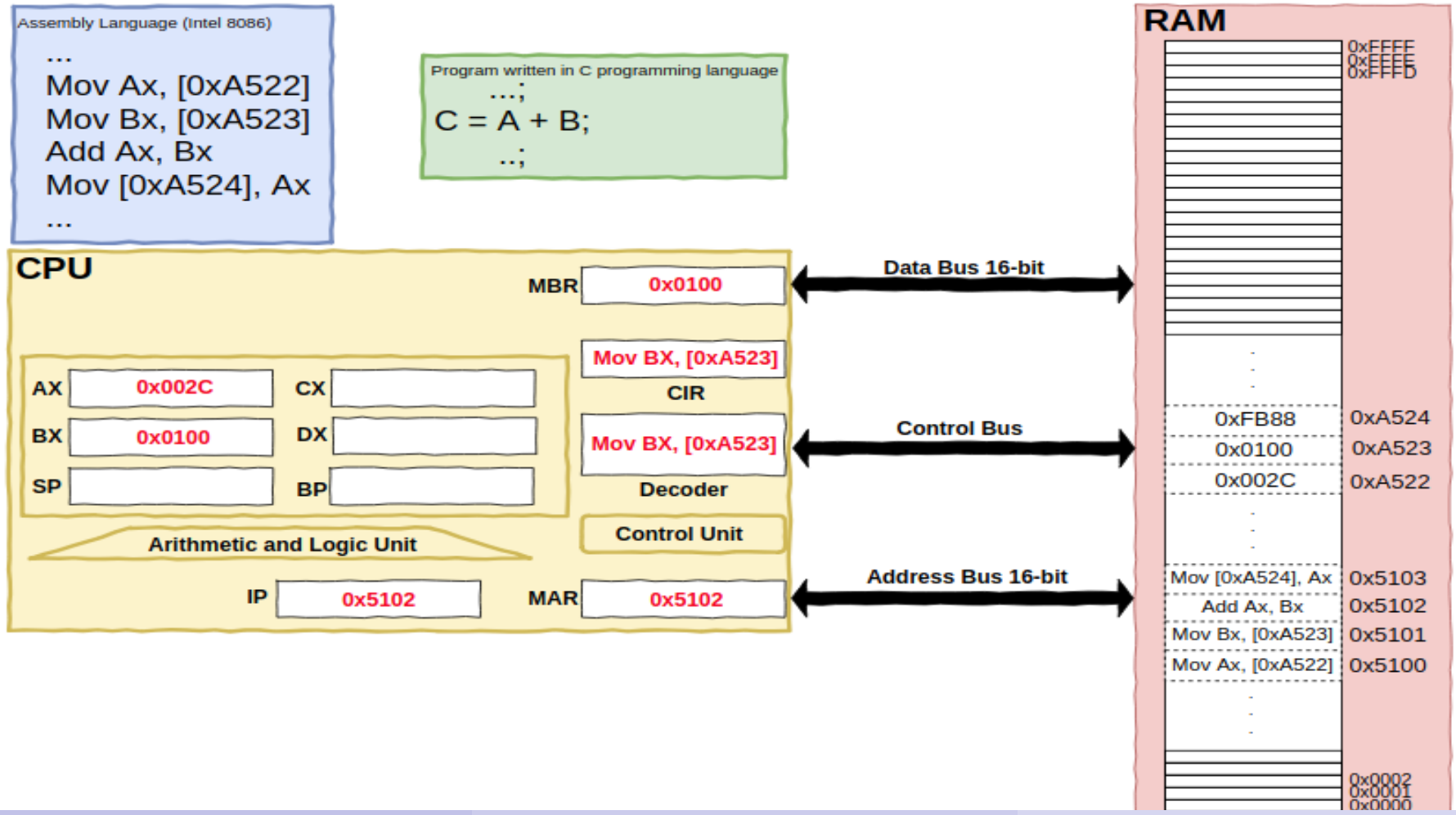
Example (Instructions Execution Cycle)

EXECUTE



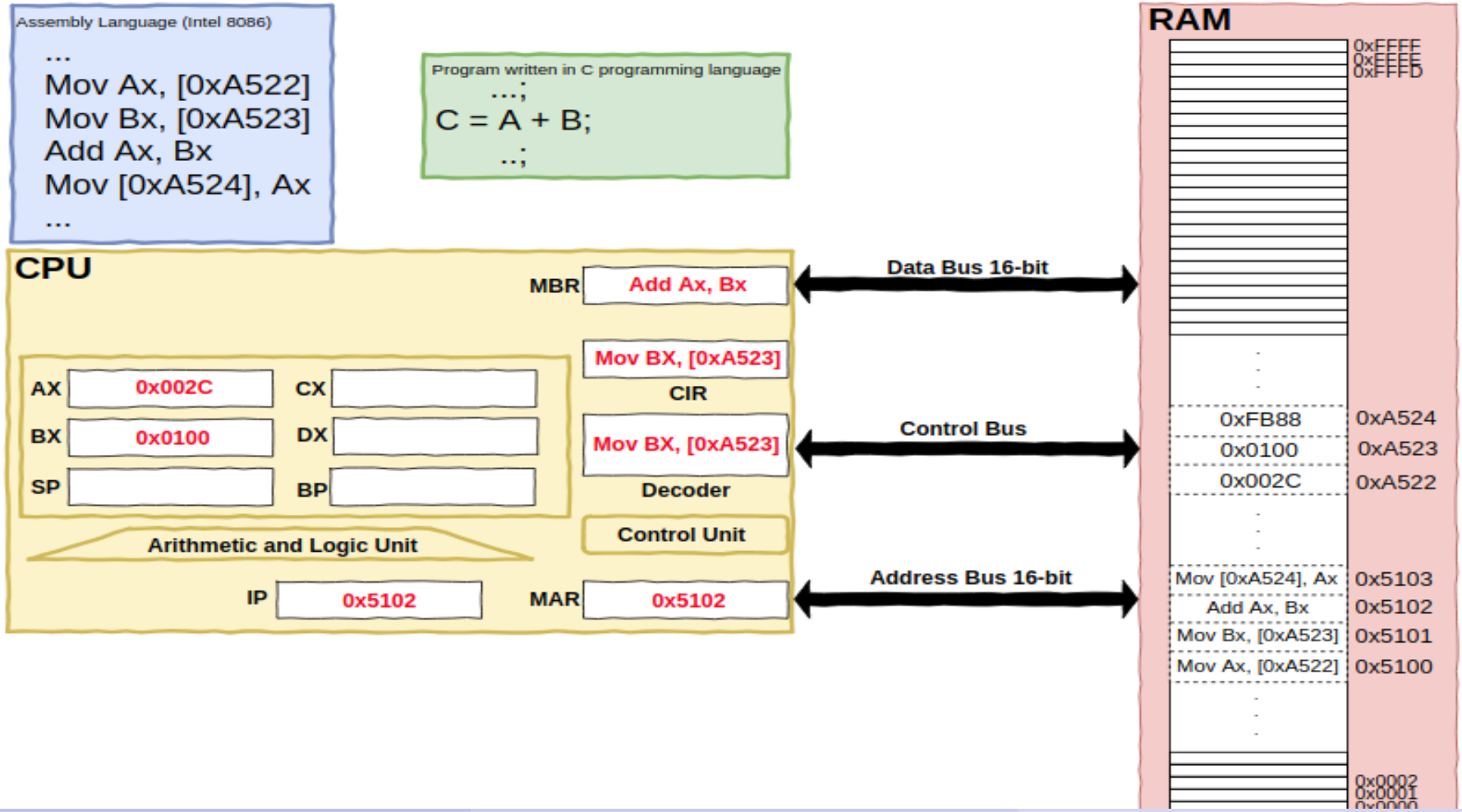
Example (Instructions Execution Cycle)

FETCH



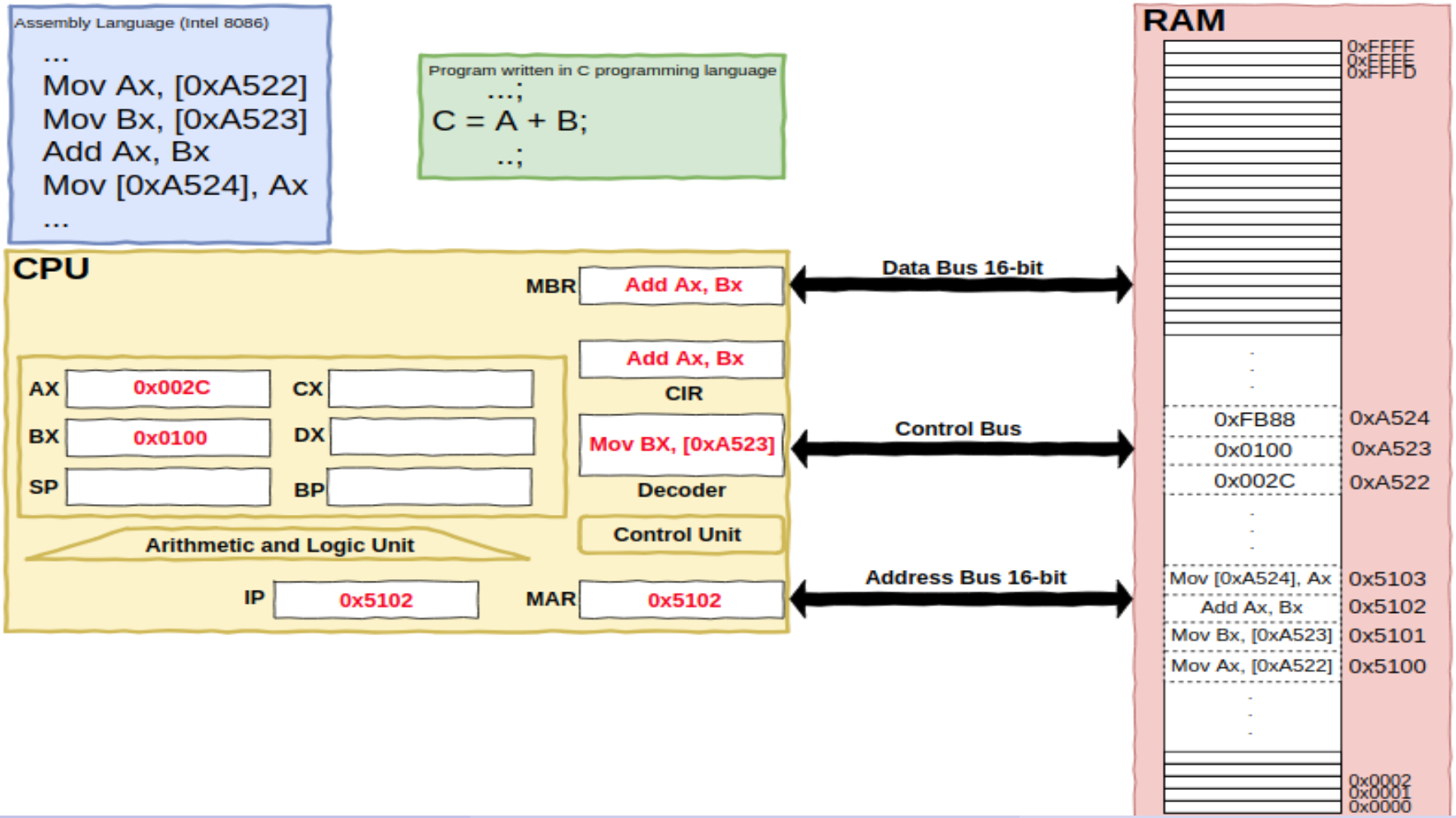
Example (Instructions Execution Cycle)

FETCH



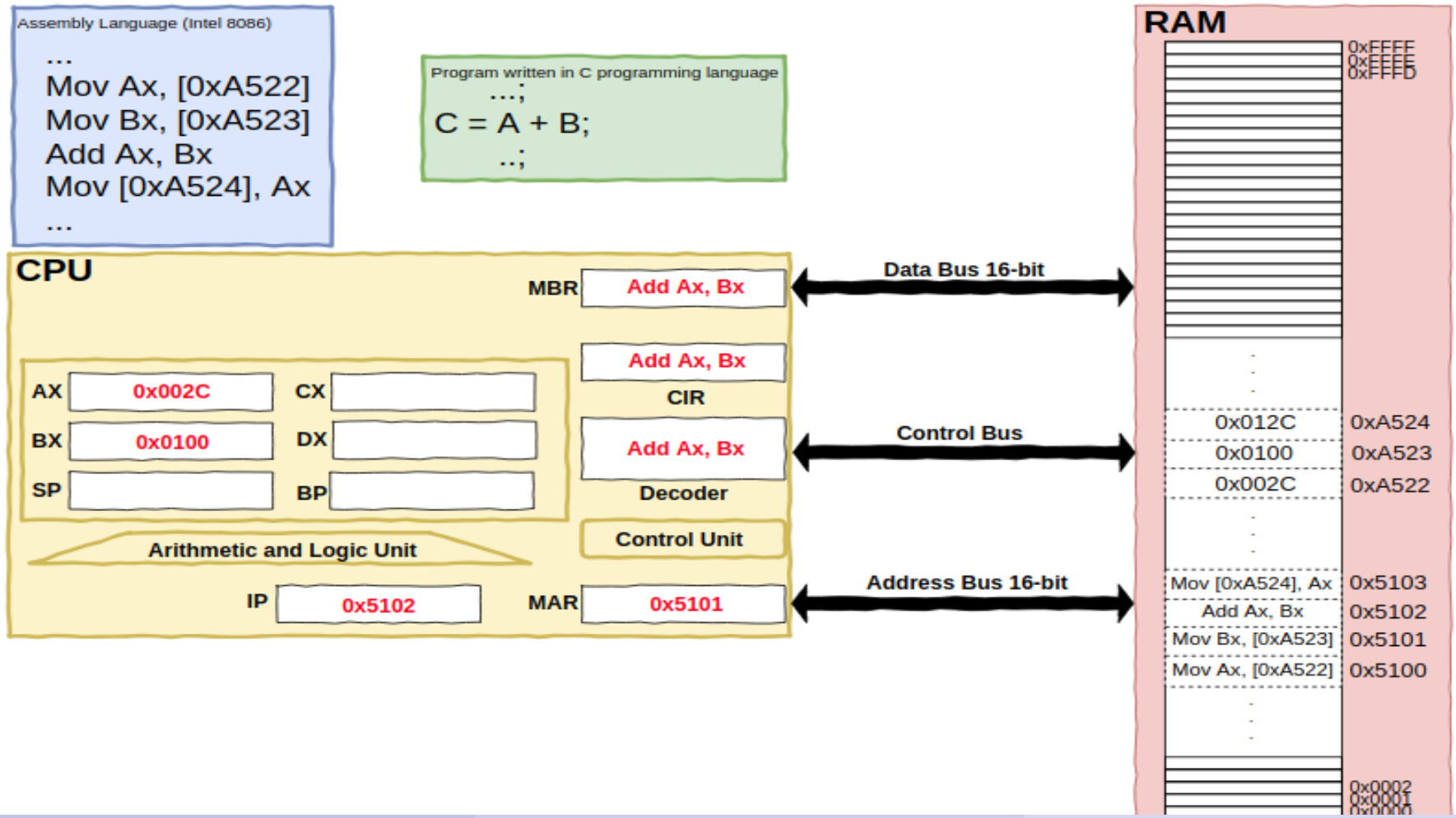
Example (Instructions Execution Cycle)

FETCH



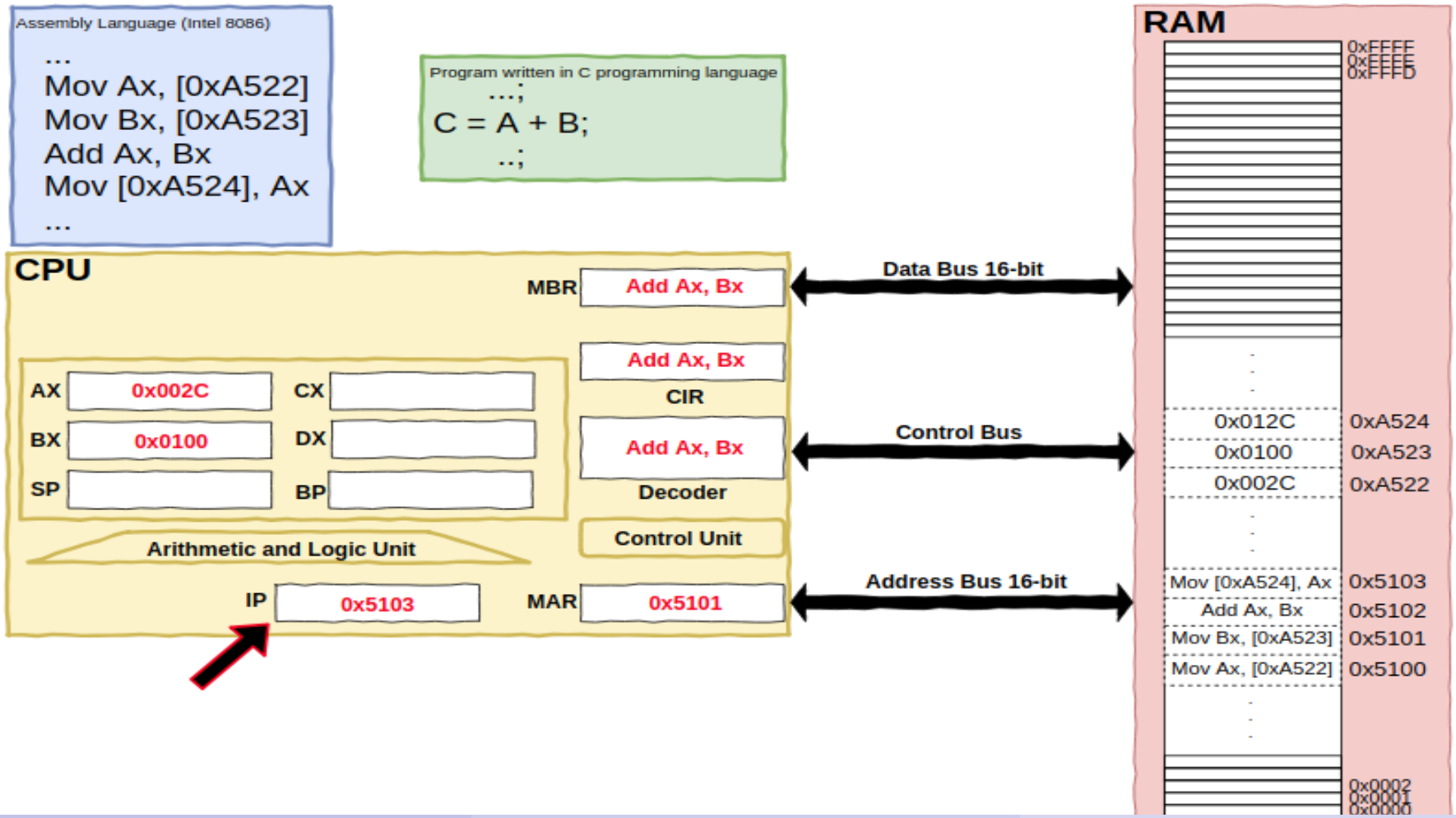
Example (Instructions Execution Cycle)

DECODE



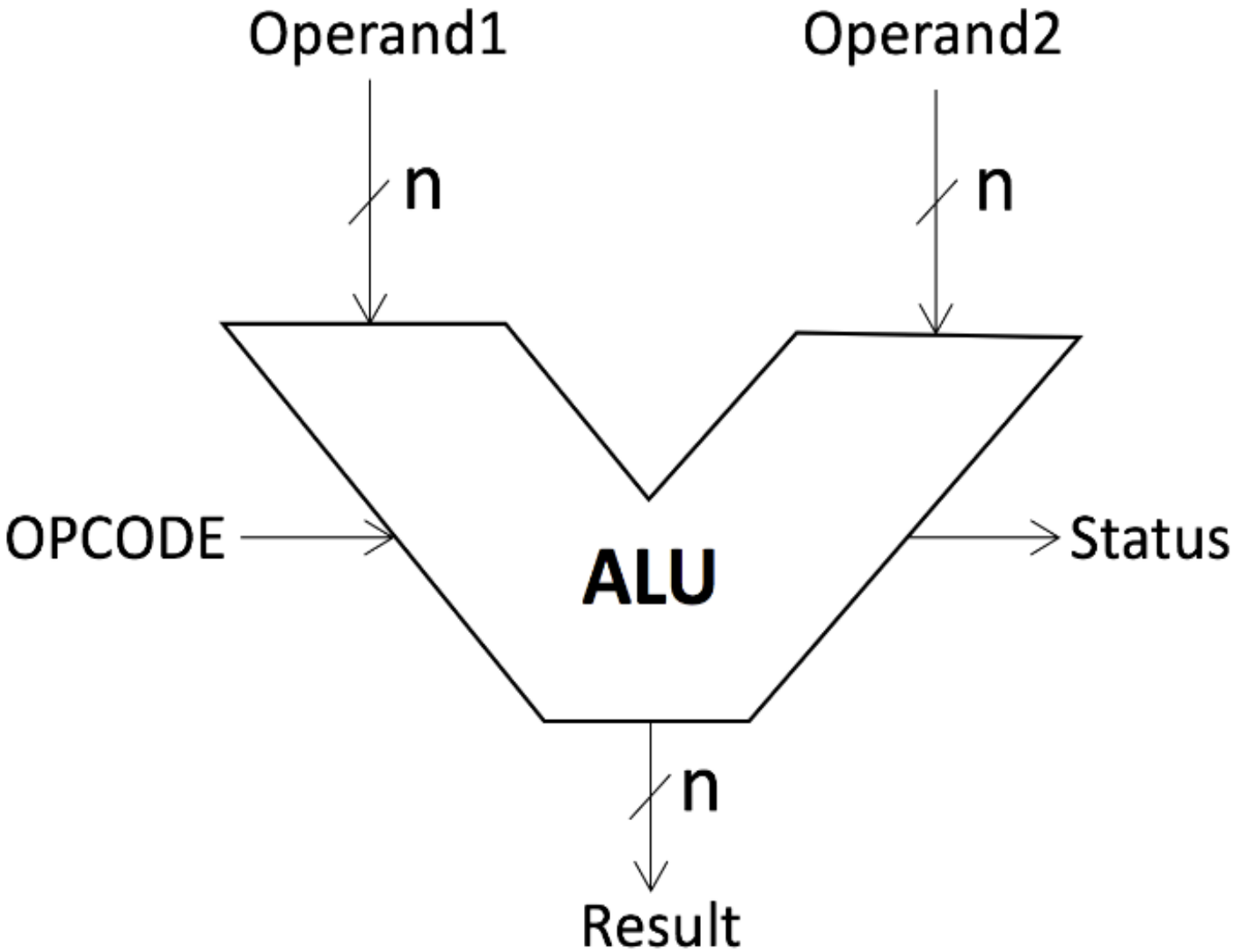
Example (Instructions Execution Cycle)

DECODE



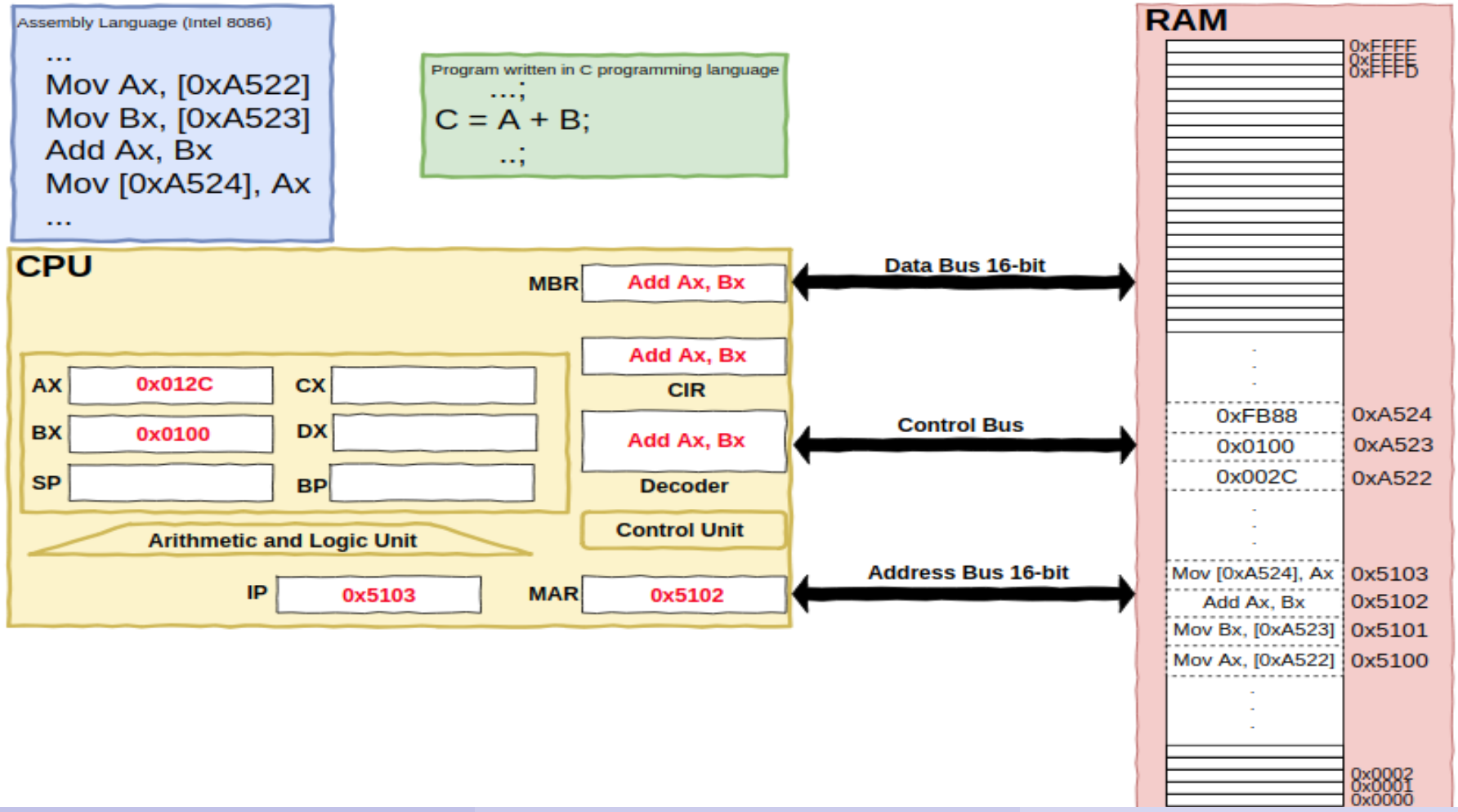
Example (Instructions Execution Cycle)

EXECUTE



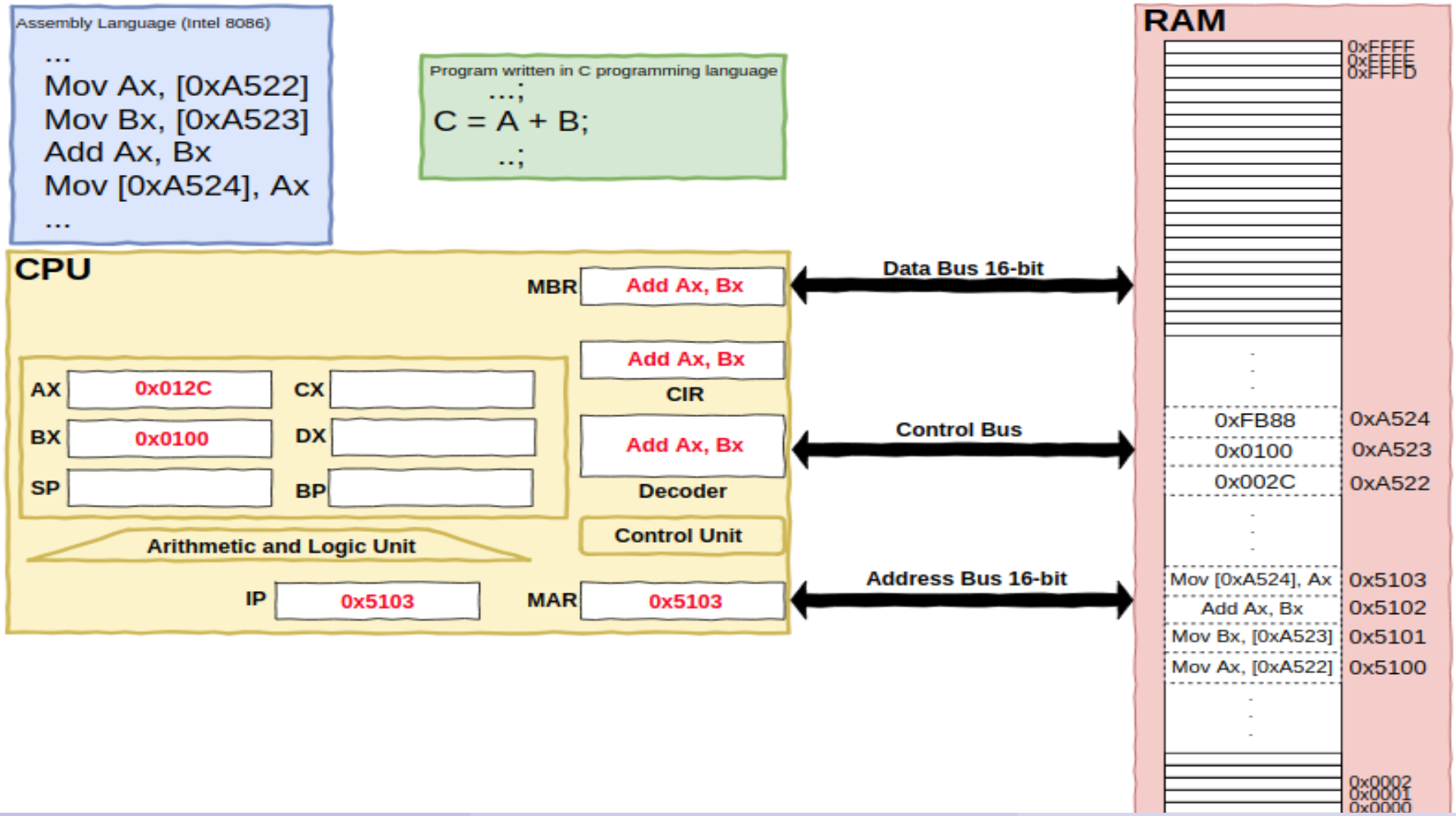
Example (Instructions Execution Cycle)

EXECUTE



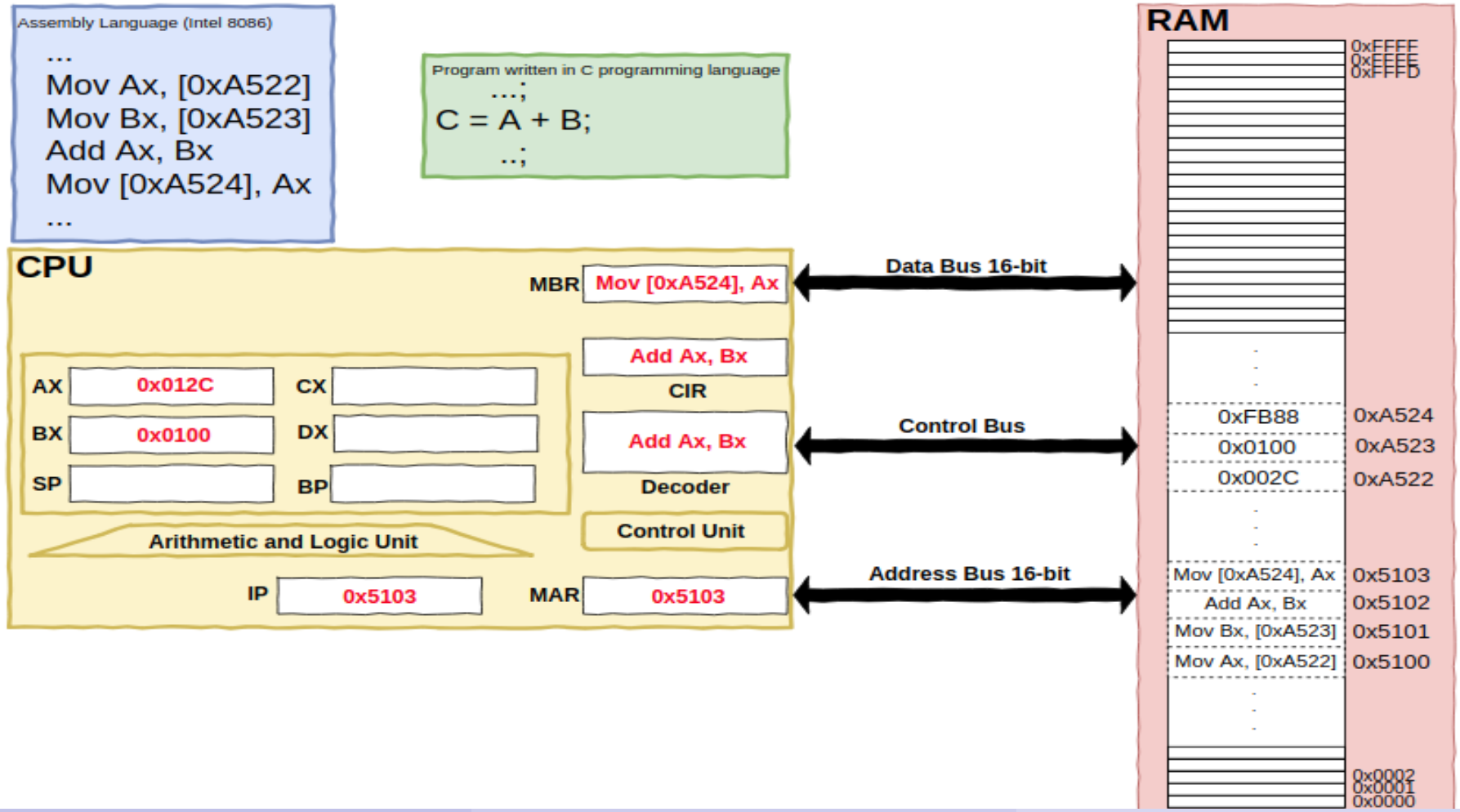
Example (Instructions Execution Cycle)

FETCH



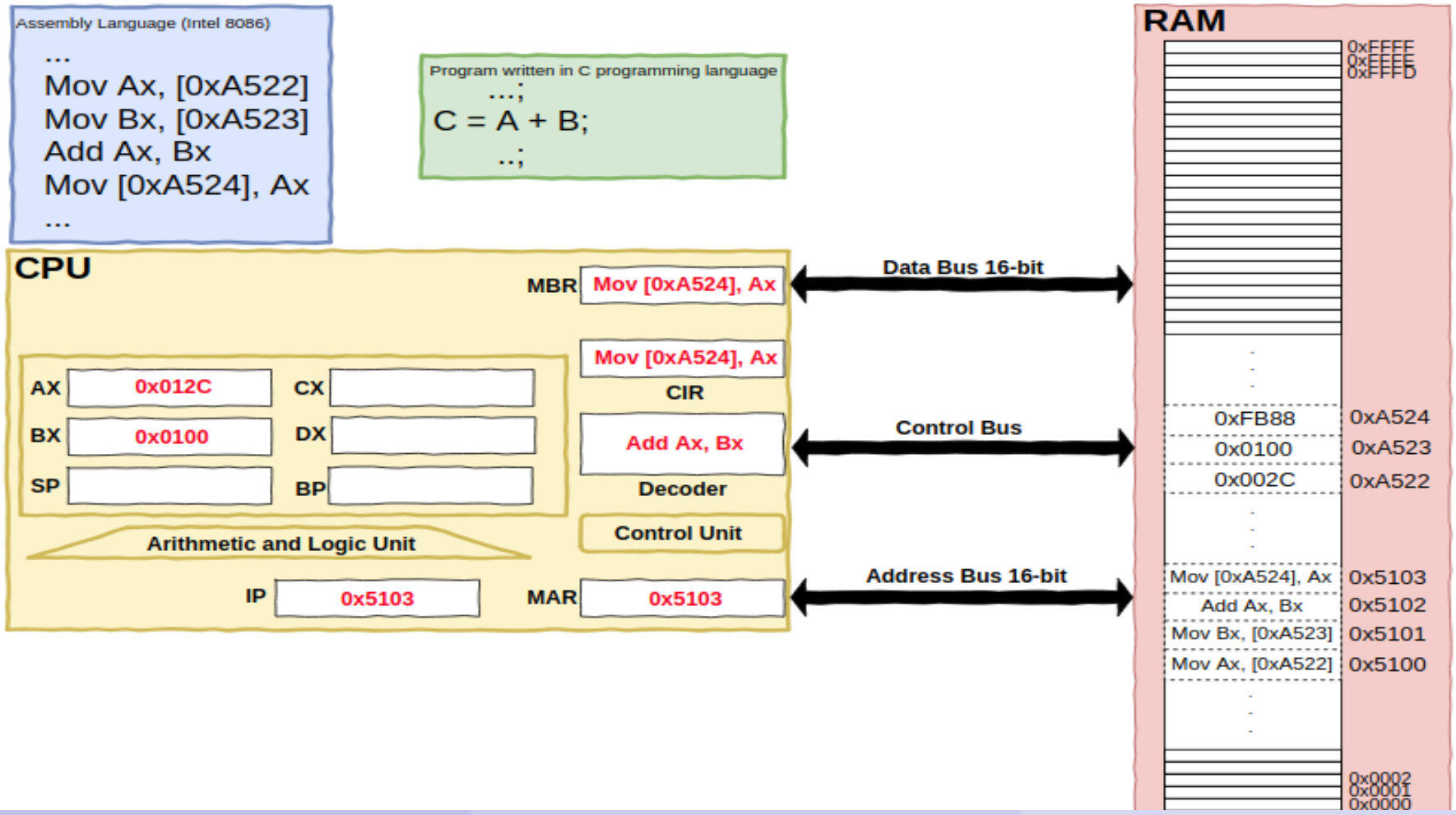
Example (Instructions Execution Cycle)

FETCH



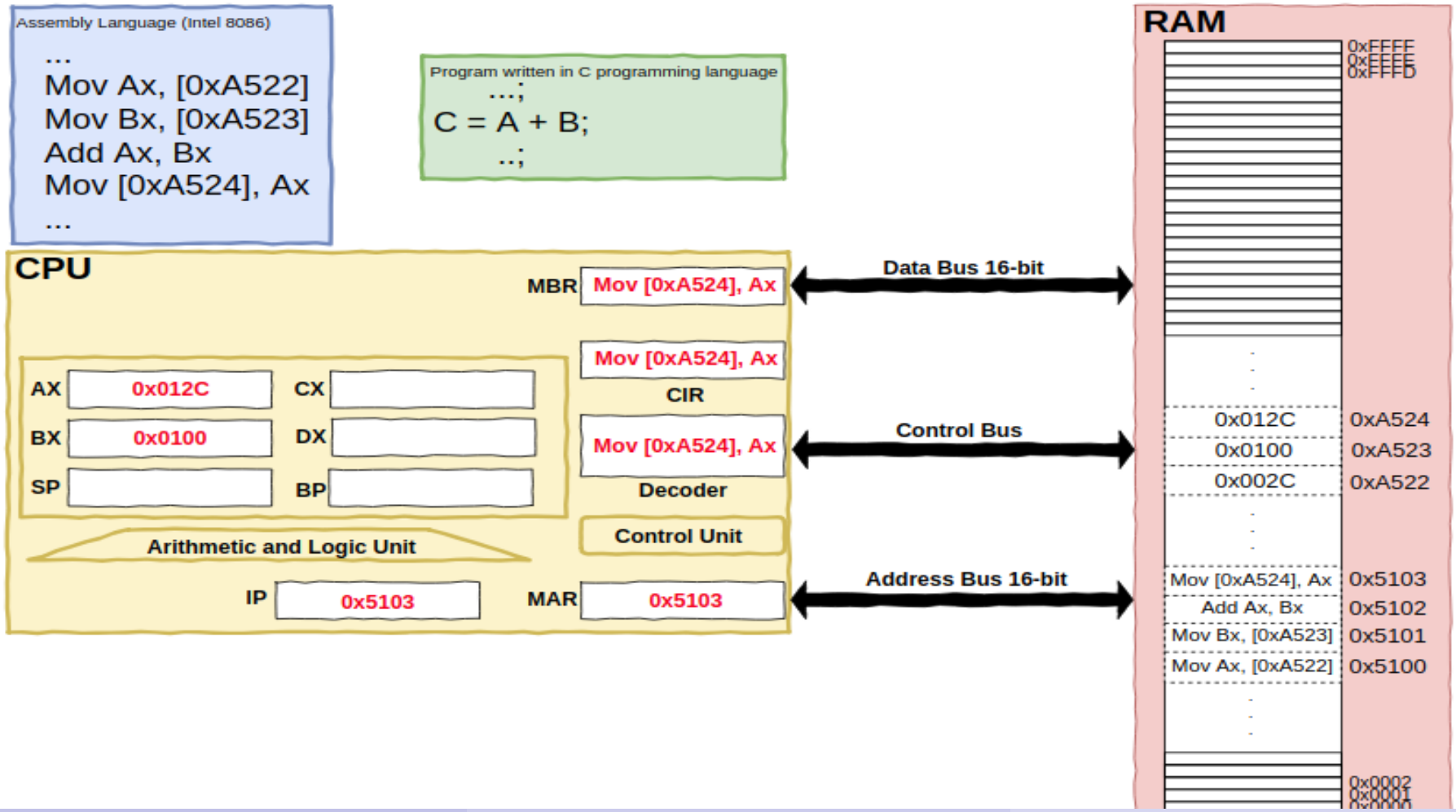
Example (Instructions Execution Cycle)

FETCH



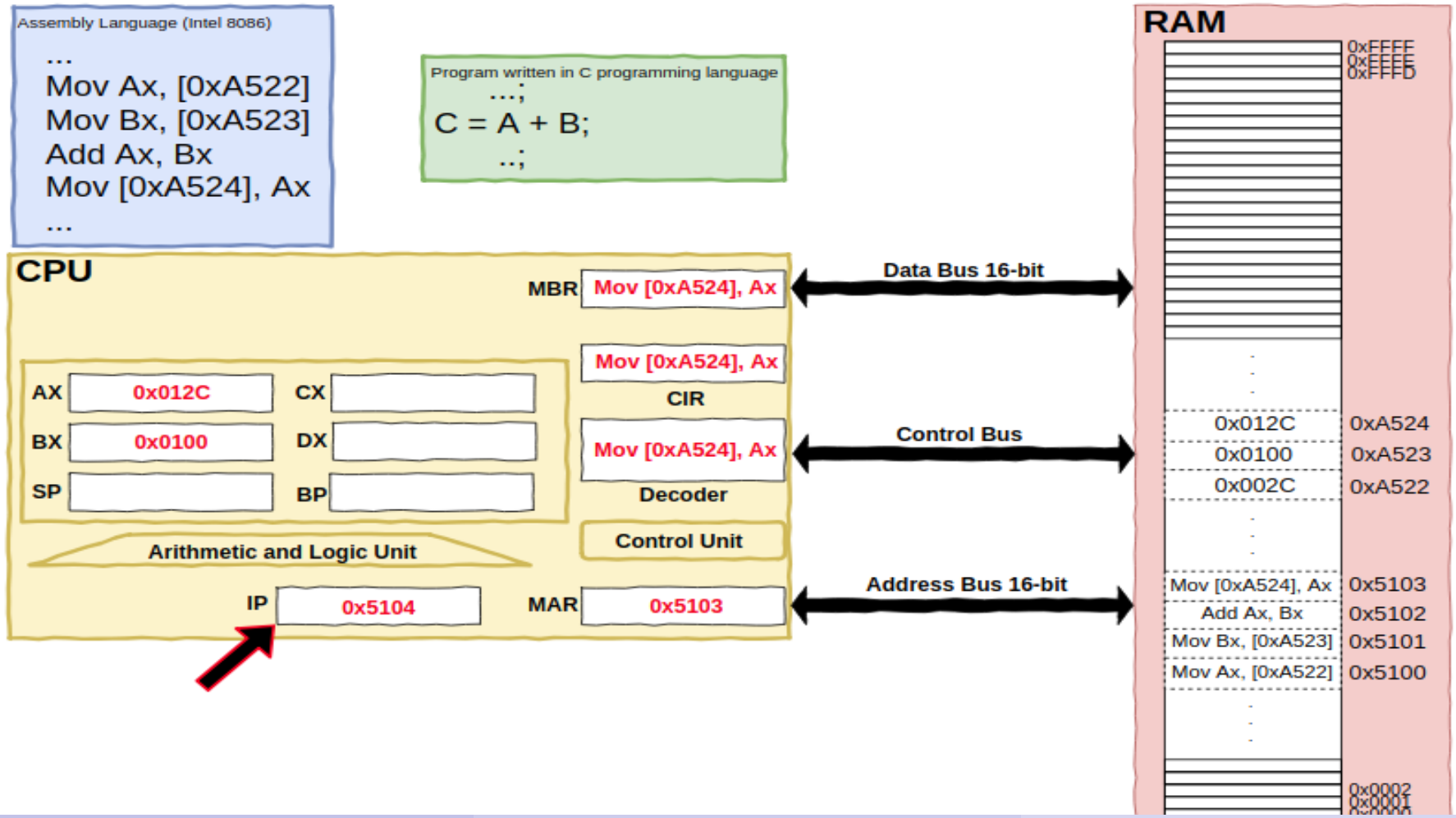
Example (Instructions Execution Cycle)

DECODE



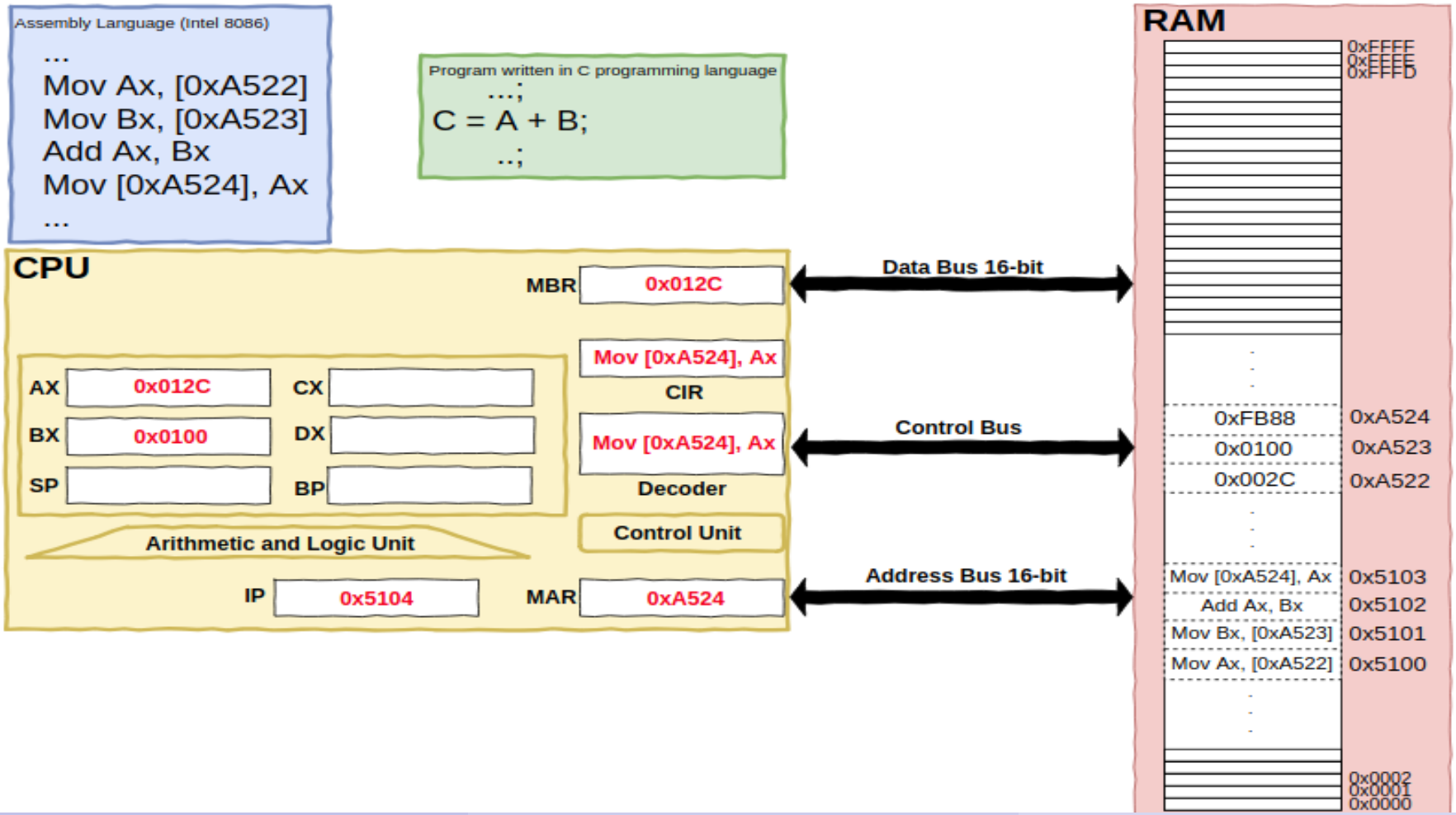
Example (Instructions Execution Cycle)

DECODE



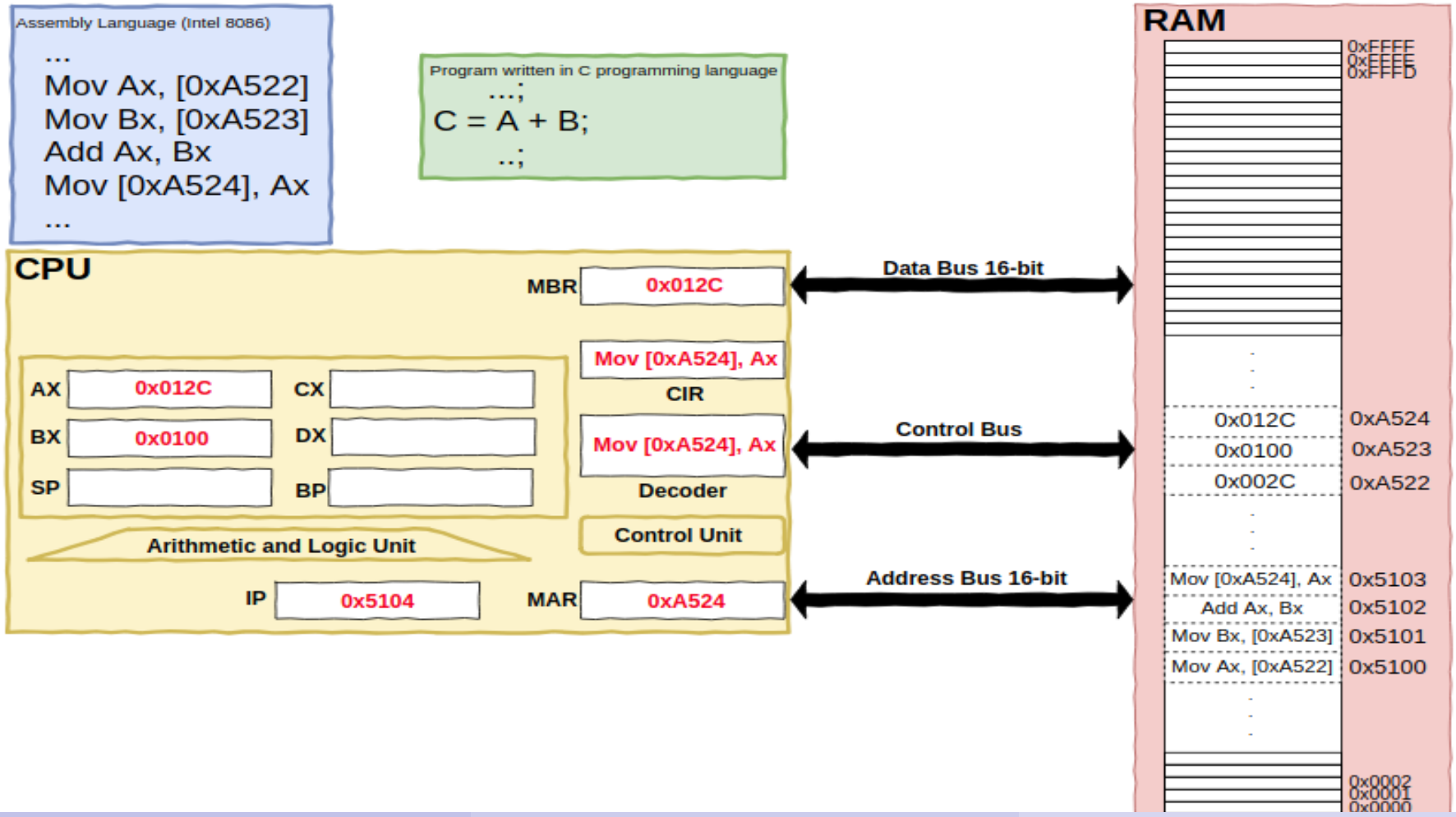
Example (Instructions Execution Cycle)

DECODE



Example (Instructions Execution Cycle)

EXECUTE



CPU: A Shared Resource

The operating system has to arrange that one CPU is used to execute: the **operating system kernel code** & a set of **processes** (from OS & user).

Q_1 : What happens if the user executes one of the C-programs below?

```
        ..              while (1)       for (int i=0; i<1;)     do
X:      goto Y;         {               {                       {
Y:      goto X;                 ;               ;                       ;               X: goto X;
        ..              }               }                       } while(!0);
```

Q_2 : What happens if the user types something on the keyboard?

Q_3 : What happens if network traffic arrives through the network card?

Q_4 : What happens if the user's program wants to open a file in the HDD?

Q_5 : What happens if the user moves the mouse (peripheral)?

Q_6 : What happens if the a program makes an error (e.g., 0/0)?

- End