The Intel 80x86

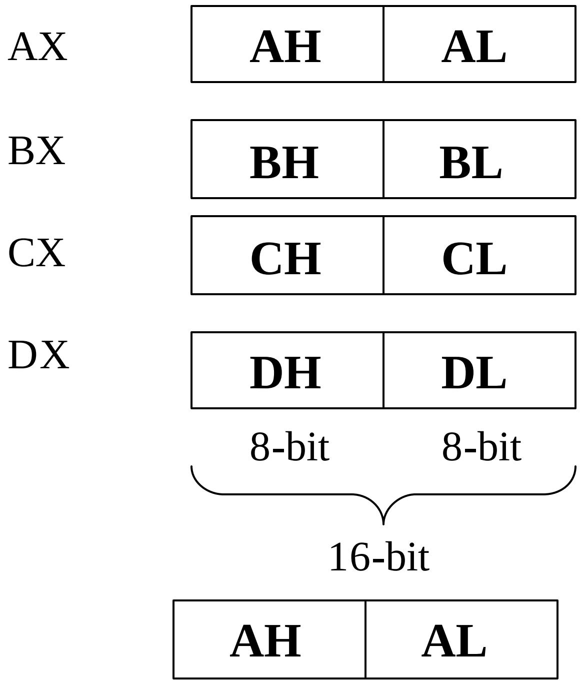
Getting to Know a Microprocessor.

- Processor is characterized by its :
  - **Register** Set
    - **General purpose** registers,
    - **addressing** registers,
    - **control/status** registers
  - **Instruction** set
    - Includes addressing modes
  - **Interrupt** mechanism (later!)

- Intel 8086: start of the 80x86 family tree.
  - All **registers**: **16-bit**
  - **16-bit data** and **20-bit address** bus
  - I/O mapped with 8-bit and 16-bit ports (later)
  - Each descendant – right up to the P6– are backward compatible
    - Same basic set of registers … but wider
    - Same basic instructions … but more
    - Same interrupt mechanism

8086 Register Set

- 16-Bit General Purpose Registers
  - can access all 16-bits at once
  - can access just high (H) byte, or low (L) byte



only General Purpose registers allow access as 8-bit High/Low sub-registers

8086 Register Set

16-Bit **Segment Addressing** Registers

CS	Code Segment
DS	Data Segment
SS	Stack Segment
ES	Extra Segment

16-Bit **Offset Addressing** Registers

SP	Stack Pointer
BP	Base Pointer
SI	Source Index
DI	Destination Index

8086 Register Set

16-Bit **Control/Status** Registers

IP Instruction Pointer (Program Counter for execution control)

FLAGS 16-bit register

- Not a 16-bit value: a collection of 9 bit-flags (six are unused)
- Flag is set when it is equal to 1
- Flag is clear when it is equal to 0

Control Flags

Direction: Used in string instructions for
moving forward/backward through strings

Interrupt: Used to enable/disable interrupts (Later)

Trap: Used to enable/disable single-step trap (Later)

8086 Register Set

Status Flags

- Flags set/cleared as “**side-effects**” of an instruction
- Part of learning an instruction is learning what flags is writes
- There are instructions that “read” a flag and indicate whether or not that flag is set or cleared.

Status Flag	Name	Description
C	Carry	
A	Auxiliary Carry	
O	Overflow	
S	Sign	
Z	Zero	

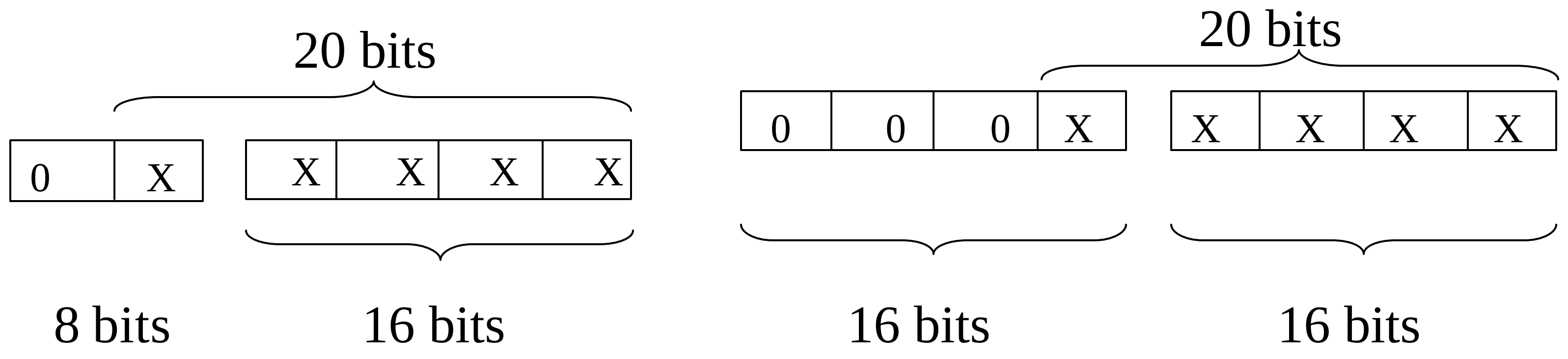
8086 Register Set

- Other registers **internal to the CPU**
  - They support the **execution of instructions**
    - Example: IR Instruction Register
    - Example: ALU input/output registers are temporary registers (scratchpad values)
  - They **cannot be accessed** directly by programmers
  - May be **larger than 16-bits**

Intel Segmented Memory Model for 20-bit Address Space

- How can **16-bit registers** and values be used to specify **20-bit addresses**?
  - Want to use 16-bit registers to refer to memory addresses

- Use two registers "side-by-side"



Intel Segmented Memory Model for 20-bit Address Space

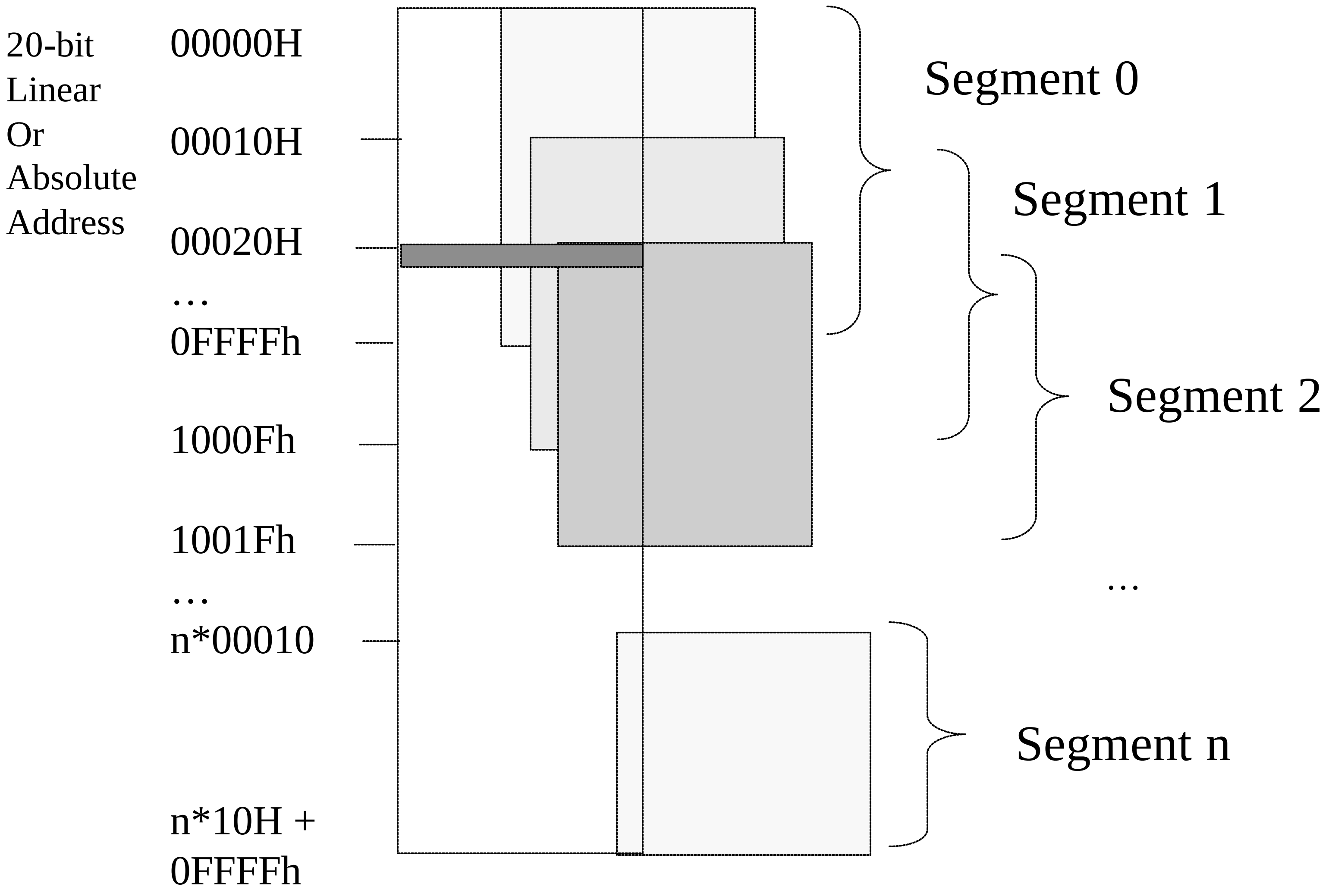
- Real-Address Mode (8086 and not later family members)
- On top of linear address space (from 0 to 1 Meg-1), **overlay overlapping "segments"**
  - Linear address: absolute address (20-bit value)
  - Segment defined as sequence of bytes that
    - Starts **every 16-bytes** (starts on absolute address that ends in 0h)
    - **Length: 64K** consecutive bytes (64K = FFFFh)
      - Hints : 2^{16} = 64K and all the 8086 registers are 16-bits wide

Segment i overlaps segment i + 1

- Segment 0 starts at absolute address 00000H and goes to 0FFFFh
- Segment 1 starts at absolute address 00010H and goes to 1000Fh
- Segment 2 starts at absolute address 00020H and goes to 1001FH

1. A particular byte can be located by giving segment number and offset within segment.
2. A particular byte located within more than one segment

Intel Segmented Memory Model for 20-bit Address Space



Intel Segmented Memory Model for 20-bit Address Space

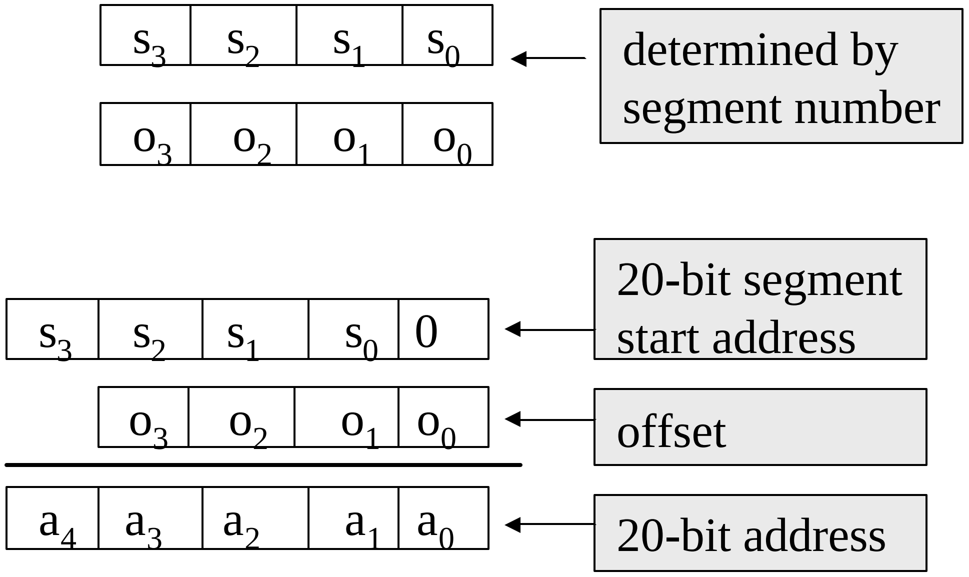
- At the hardware level :
  - Address put on the Address Bus as a **20-bit linear address**

- From the Software (Programmer's) Perspective:
  - Addresses NEVER specified as 20-bit values
  - Addresses ALWAYS specified as **two 16-bit values: `segment:offset`**

- Who does the conversion ?
  - The CPU (e.g. during the fetch of an instruction)
  - As a programmer, you always use `segment:offset`

Intel Segmented Memory Model for 20-bit Address Space

- How does the CPU convert from segment:offset to absolute ?
  - Recall: each segment starts at 16-byte boundary
  - Start address of a segment = segment number * 16_{10}
  - Hint: shortcut for multiplying by 16 when working in binary(hex) ?


s_3 s_2 s_1 s_0
o_3 o_2 o_1 o_0
determined by segment number
s_3 s_2 s_1 s_0 0
20-bit segment start address
o_3 o_2 o_1 o_0
offset
a_4 a_3 a_2 a_1 a_0
20-bit address

Intel Segmented Memory Model for 20-bit Address Space

- Example: Suppose we have segment number = 6020H and
  offset = 4267H

```
    segment * 10H   →   60200 H
         + offset   →    4267 H
                      ___________
  20-bit address        64467 H
```

$\boxed{\text{20-bit address}}$

Intel Segmented Memory Model for 20-bit Address Space

- Remember : ugly Side Effect of Segmented Memory
  - **Each memory byte** can be referred to by **many different SEG:OFS** pairs

- Example: The (unique) byte at address **00300 H** can be referred to by:

  20-bit address

  0 H : 300 H

  1 H : 2F0 H

  30 H : 0 H

  (more too !)

How is segmented memory managed by the 8086 ?

- 8086 includes four 16-bit SEGMENT registers:
  - CS : Code Segment Register
  - DS : Data Segment Register
  - SS : Stack Segment Register
  - ES : Extra Segment Register

- Segment registers are used by default as the segment values during certain memory access operations
  - All **instruction fetches**: CS : IP
  - "most" **data access**: DS : offset

BUT segments must be initialized before use (Later!)

Processor uses contents of DS as 16-bit **segment** value when fetching data => programmer only needs to supply 16-bit **offset** in instructions)

Let’s refine the Instruction Execution Cycle …

- Processor executes instruction by repeating:

```
do {

    Fetch instruction:  **IR := mem[CS:IP]**  and adjust IP to point to
                 next sequential instruction
    **Execute instruction in IR**


} until HLT instruction has been executed
```

Notation
:= “gets loaded from”

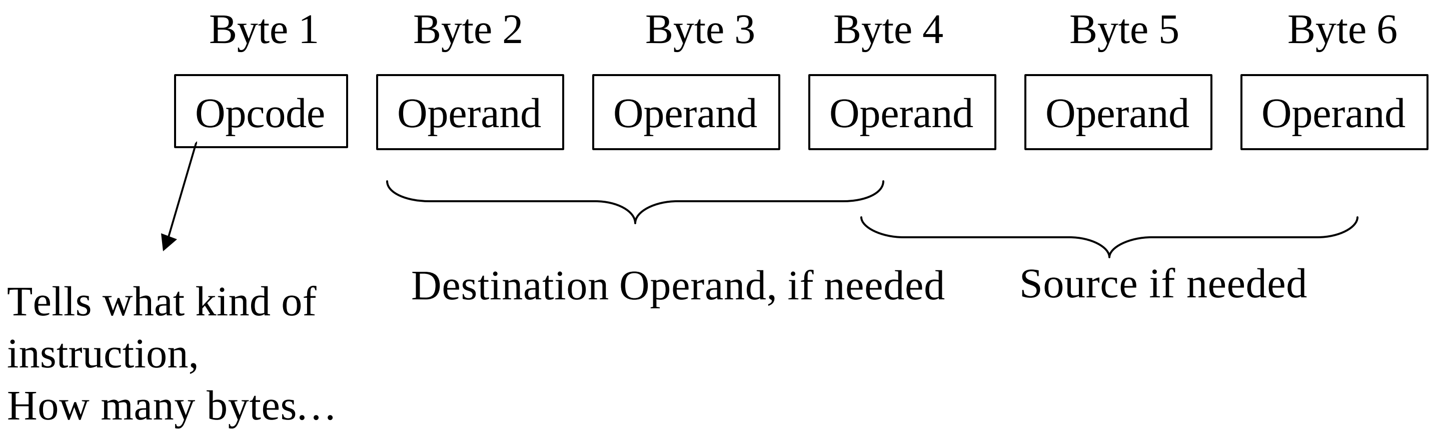
some interrupt stuff goes here ! more later!

inherently sequential behaviour!

Instruction Execution Cycle

- What is an instruction ?
  - On Intel 8086, an instruction is a **sequence of 1..6 bytes**

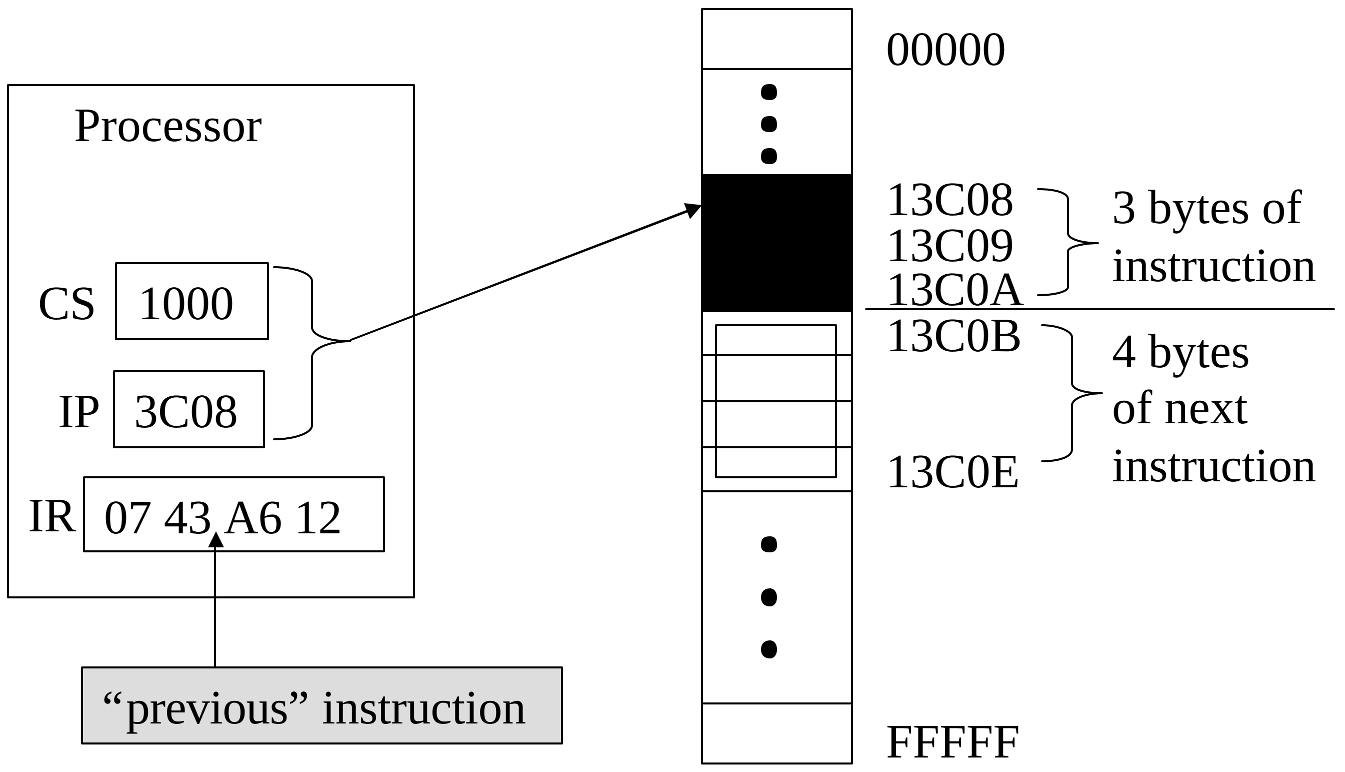
- A simple (incomplete) model of an instruction is as follows :



- Common mistake: **do not apply little endian to an instruction.**
  - Little endian only applies to word operations, not sequences of bytes.

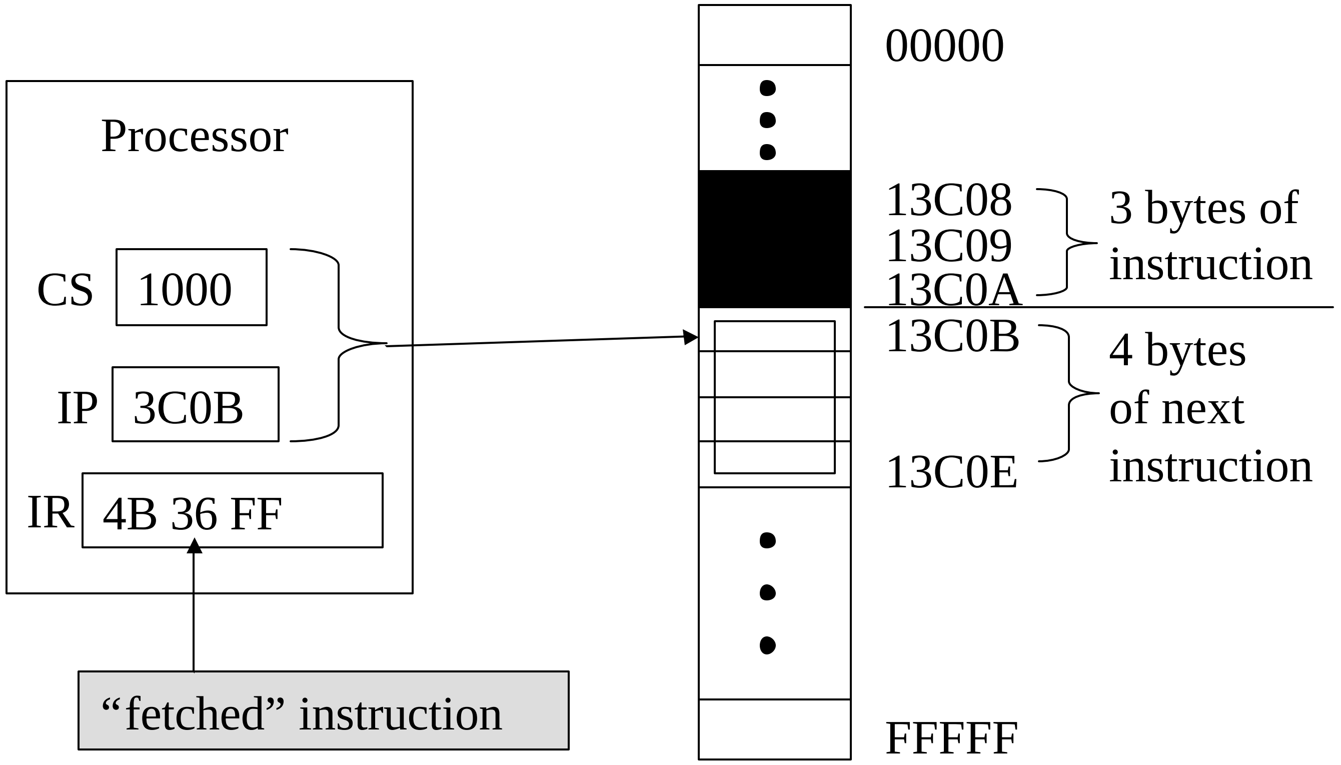
Instruction Execution Cycle

Before fetch:



Instruction Execution Cycle

After fetch:


Processor
CS
1000
IP
3C0B
IR
4B 36 FF
"fetched" instruction
00000
13C08
13C09
13C0A
3 bytes of instruction
13C0B
13C0E
4 bytes of next instruction
FFFFF