

SYSC3601

Microprocessor Systems

Unit 2:

The Intel 8086 Architecture and Programming Model

1. Registers and internal architecture (Ch 2)
2. Address generation (Ch 2)

- There are two main functional logic blocks in the 8086/88 processors:
 - **EU** Execution Unit - execution of program instructions
 - **BIU** Bus Interface Unit - provides interface to memory and I/O
 1. controls the address, data, and control busses.
 2. handles instruction fetch and data read/write functions

8086 Registers and Internal Architecture

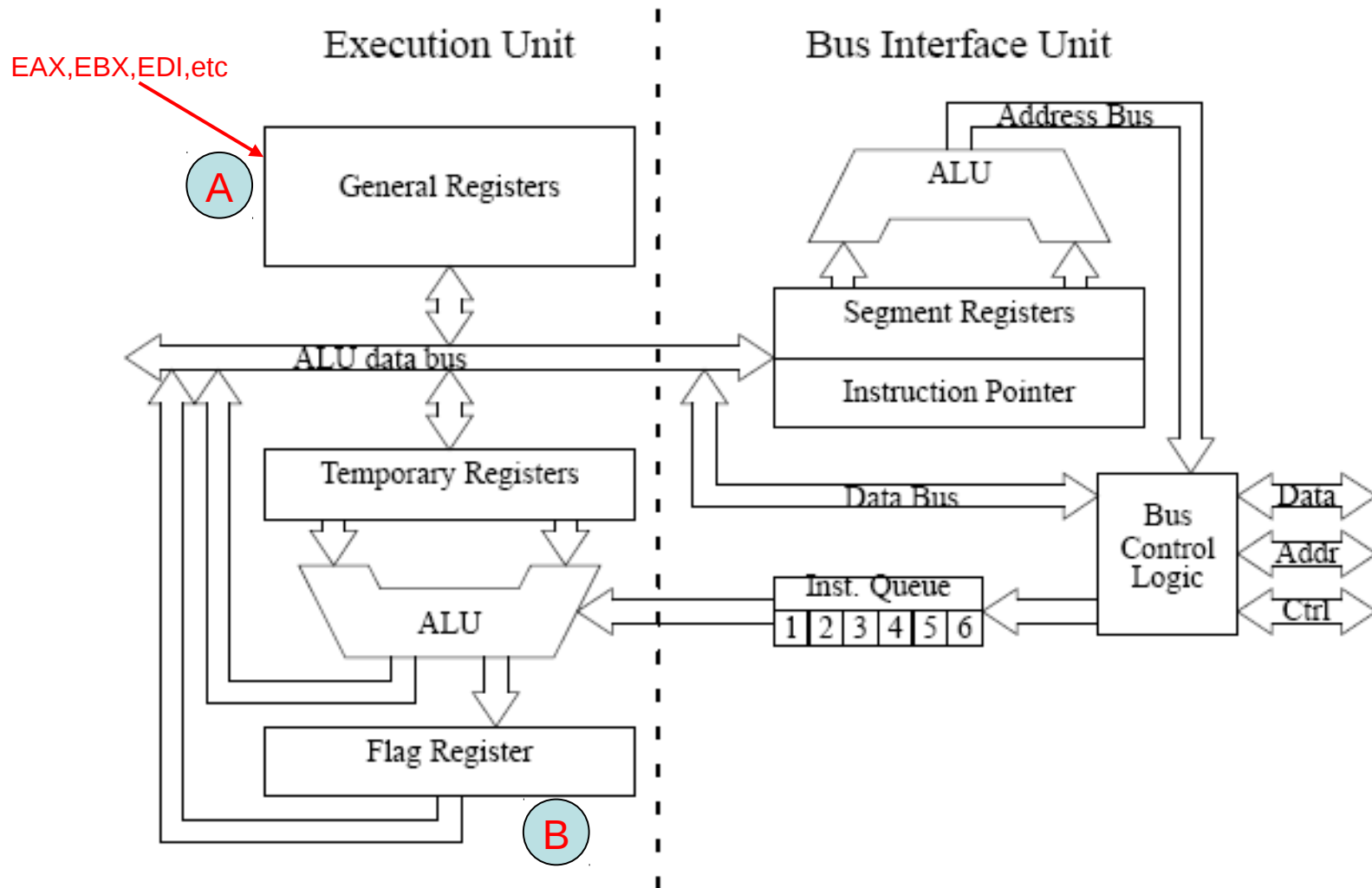
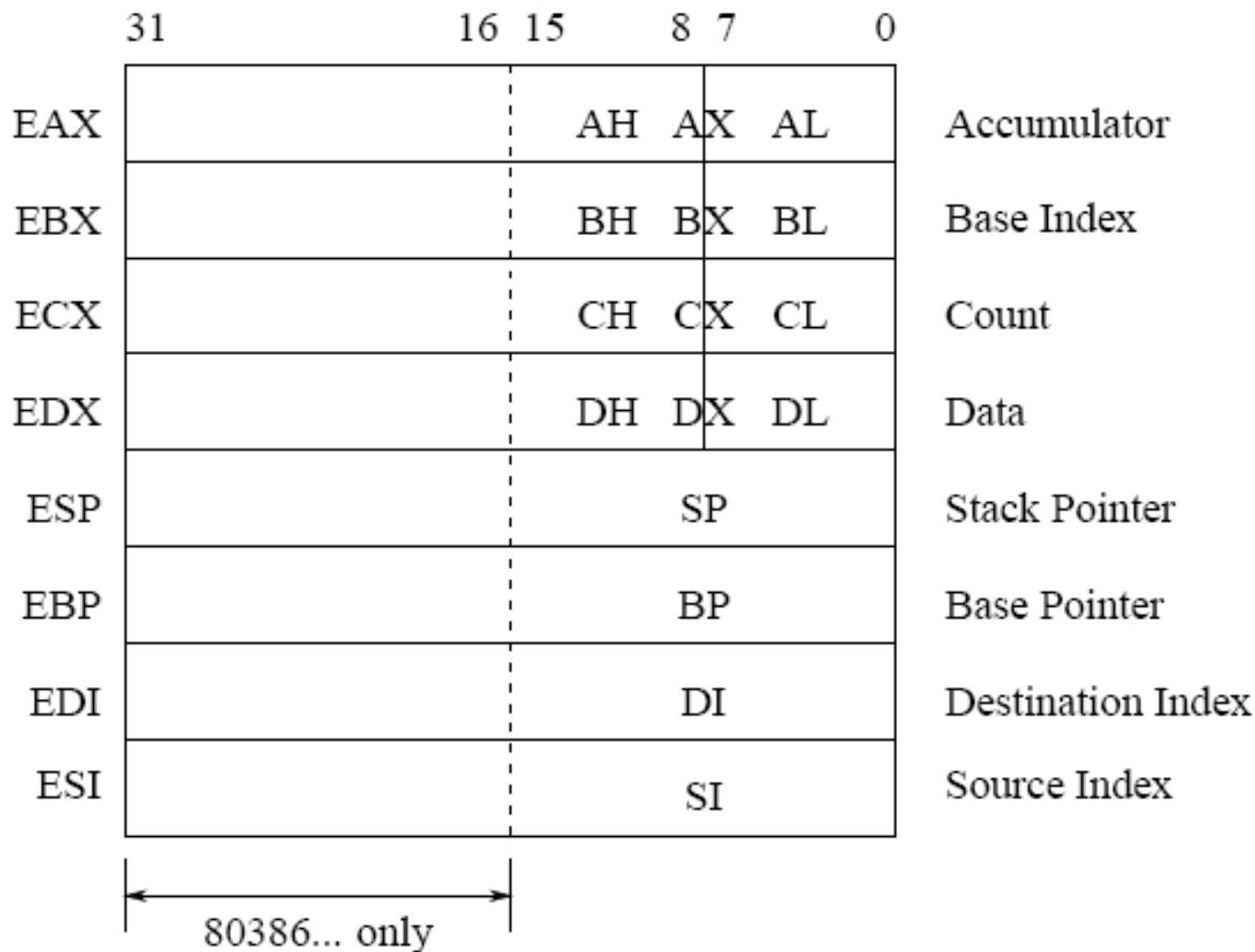


Figure 1: Internal Architecture of the 8086/88

A Execution Unit – Multipurpose Registers



Execution Unit – Multipurpose Registers

- **EAX** Accumulator: used for arithmetic and logic operations. Destination for MUL and DIV.
- **EBX** Base Index: Typically used to hold *offset* addresses.
- **ECX** Count: Typically used to hold a count value for various instructions (repeated strings, LOOP/LOOPD, Shift/rotate).

```
MOV CX, 080H
```

```
HERE ... ;
```

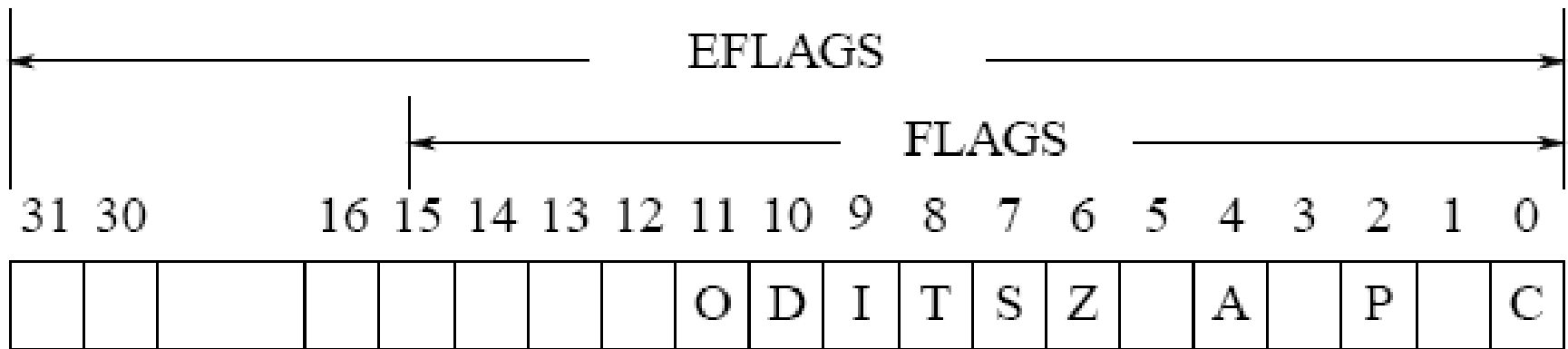
```
LOOP HERE ; Decrement CX, JNZ to label 'HERE'
```

- **EDX** Data: temporary data storage for part of a result from a multiplication (Most significant result) or division (dividend, remainder).

Execution Unit – Multipurpose Registers

- **ESP** Stack Pointer: Used to offset into the stack segment to address the stack. PUSH/POP, JSR
- **EBP** Base Pointer: Used to store a base memory location for data transfers.
- **EDI** Destination Index: Typically used as an offset for the destination memory location for string/byte transfers.
- **ESI** Source Index: Typically used as an offset for the source memory location for string/byte transfers.
- *The use of the base and offset registers EBX, ESP, EBP, EDI and ESI will become clearer when addressing modes are covered.*

B Execution Unit – Flag Register



- Note: O,Z,A,P & C are changed by most arithmetic and logic instructions but are *unchanged by data transfers*.
- **C** Carry: Holds the carry after addition, or the borrow after subtraction.
- **P** Parity: '0' - odd parity. '1' - even parity.
- **A** Auxiliary carry: Holds the “half-carry/borrow” after addition/subtraction. (BCD operations on nibbles).
- **Z** Zero: '1' if the result of an arithmetic or logic operation is zero.

Execution Unit – Flag Register

- **S** Sign: holds the sign of the result after a arithmetic or logical operation. This is the value of the sign bit of the result of the operation.
- **T** Trap: enables trapping if '1'. Program flow is interrupted based on the values of the control and debug registers.
- **I** Interrupt: Controls the operation of the INTR (interrupt request) pin. If '1', interrupts from INTR are enabled.
- **D** Direction: Selects either increment or decrement for the SI and/or DI registers during string and loop functions. If '1', the registers are decremented.
- **O** Overflow: Indicates that a result has exceeded the capacity of a register during *signed* operations.

Execution Unit – Extended Flag Register

- **IOP** (80286+) I/O Privilege level: Two bits correspond to privilege level for I/O operation. 00 is the highest, 11 is the lowest.
- **NT** (80286+) Nested task: Set when a task is nested within another task.
- **RF** (80386+) Resume Flag: Used during debugging.
- **VM** (80386+) Virtual Mode: Virtual mode execution (multiple 8086s running in protected mode).
- **AC** (80486SX+) Alignment Check: Non-aligned address (for co-processor).
- **VIF** (Pentium+) Virtual Interrupt Flag: A copy of the interrupt flag.
- **VIP** (Pentium+) Virtual Interrupt Pending:
- **ID** (Pentium+) ID: The CPUID instruction is supported.

Bus Interface Unit

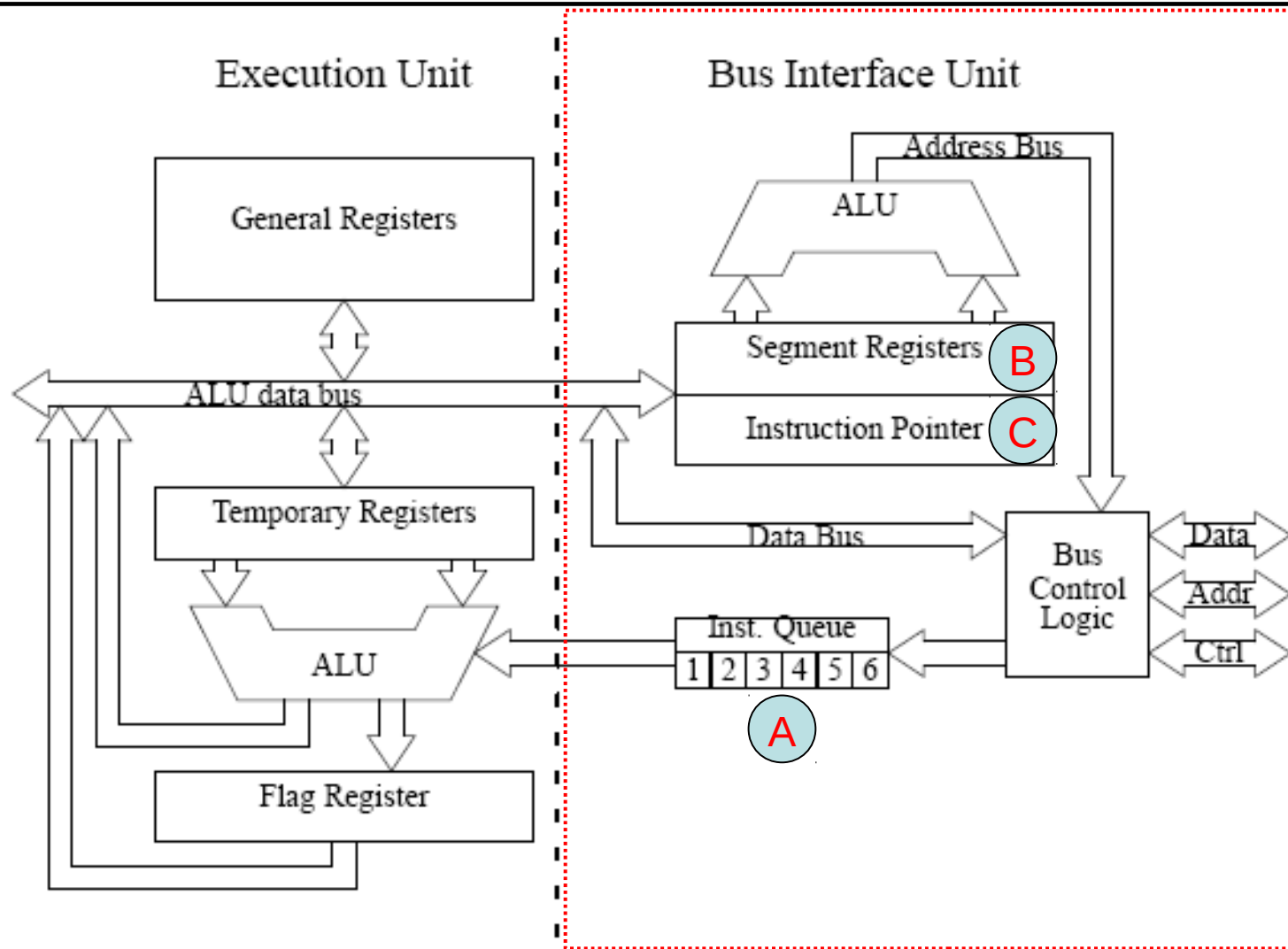
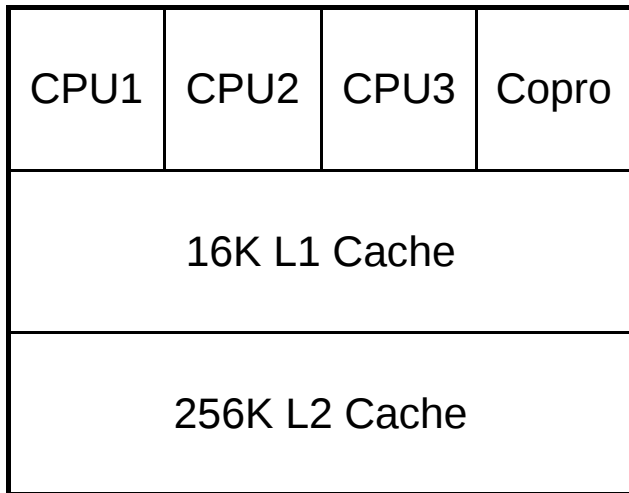
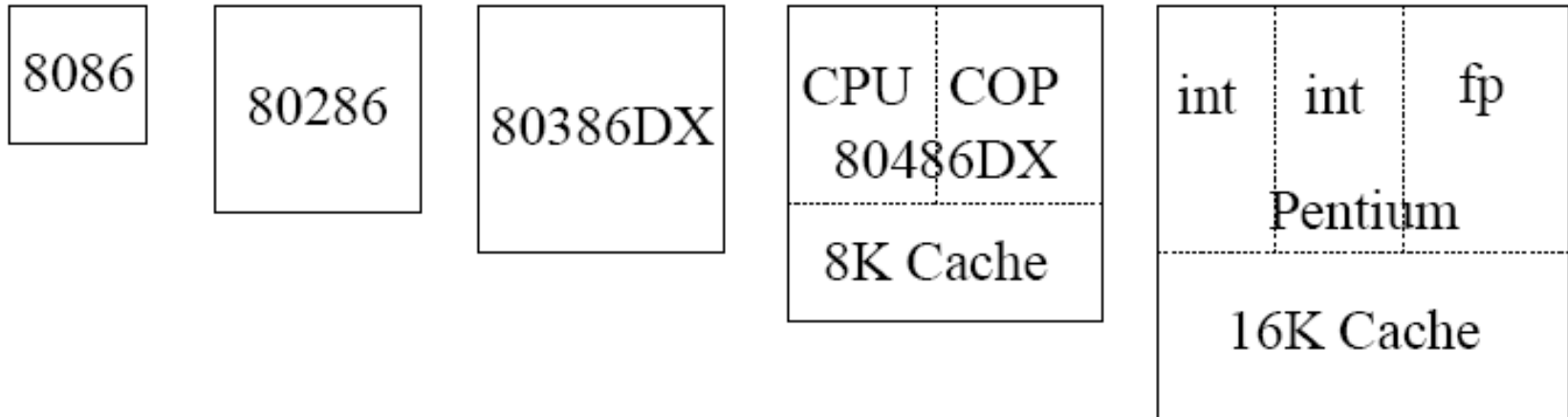


Figure 1: Internal Architecture of the 8086/88

The Bus Interface Unit (BIU)

- The BIU can operate in parallel with the EU
- The instruction queue
 - One task of the BIU is instruction “*pre-fetch*”
 - Whenever the external busses are idle, the BIU fetches the next instruction and places it in the instruction queue.
 - The instruction queue is now replaced by L1/L2 cache.
 - The 8086 can have up to 6 bytes of information in the instruction queue, the 8088 is limited to 4.
 - The instruction queue must be flushed for some instructions (change of program flow, e.g., JMP)
- We will be spending more time later in the course on bus control.
- BIU computes the Physical Address (explained later)

Intel x86 cores



Pentium Pro

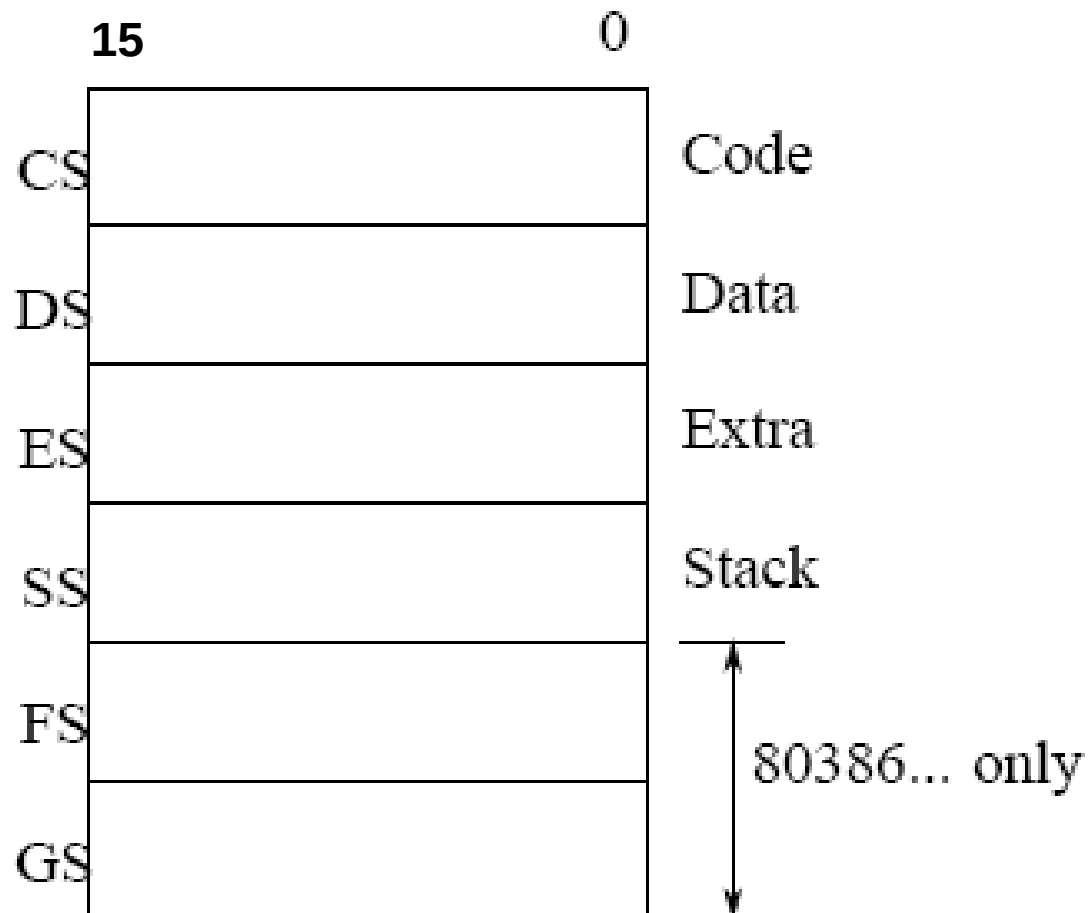
Pentium II, III, 4 same as Pentium Pro with increased L1 & L2 cache sizes.

B Bus Interface Unit – Instruction Pointer

- The Instruction Pointer (IP) is updated by the BIU.
 - EIP (32 bits) in 80386 and up
- IP contains the offset of the next instruction to be *fetch*ed from the beginning of the code segment.
- Whenever the instruction pointer is saved on the stack, it is automatically adjusted to point to the next instruction to be *execu*ted (as opposed to *fetch*ed).
- Can be modified with a JMP or CALL instruction.
- Used with CS (see next few slides...)
 - Physical address of next instruction = CS:IP

C Bus Interface Unit– Segment Registers

- Segment registers are combined with other registers to generate 20-bit addresses.



Bus Interface Unit – Segment Registers

- **CS** Code Segment: Used to compute the starting address of the section of memory holding code (restricted to 64K in REAL mode).
- **DS** Data Segment: Used to compute the starting address of the section of memory holding data (restricted to 64K in REAL mode).
- **SS** Stack Segment: Used to compute the starting address of the section of memory holding the stack (restricted to 64K in REAL mode).
- **ES** Extra Segment: Additional data segment used by some string instructions.
- **FS&GS** Additional segment registers in the 80386 (and up) for program use.

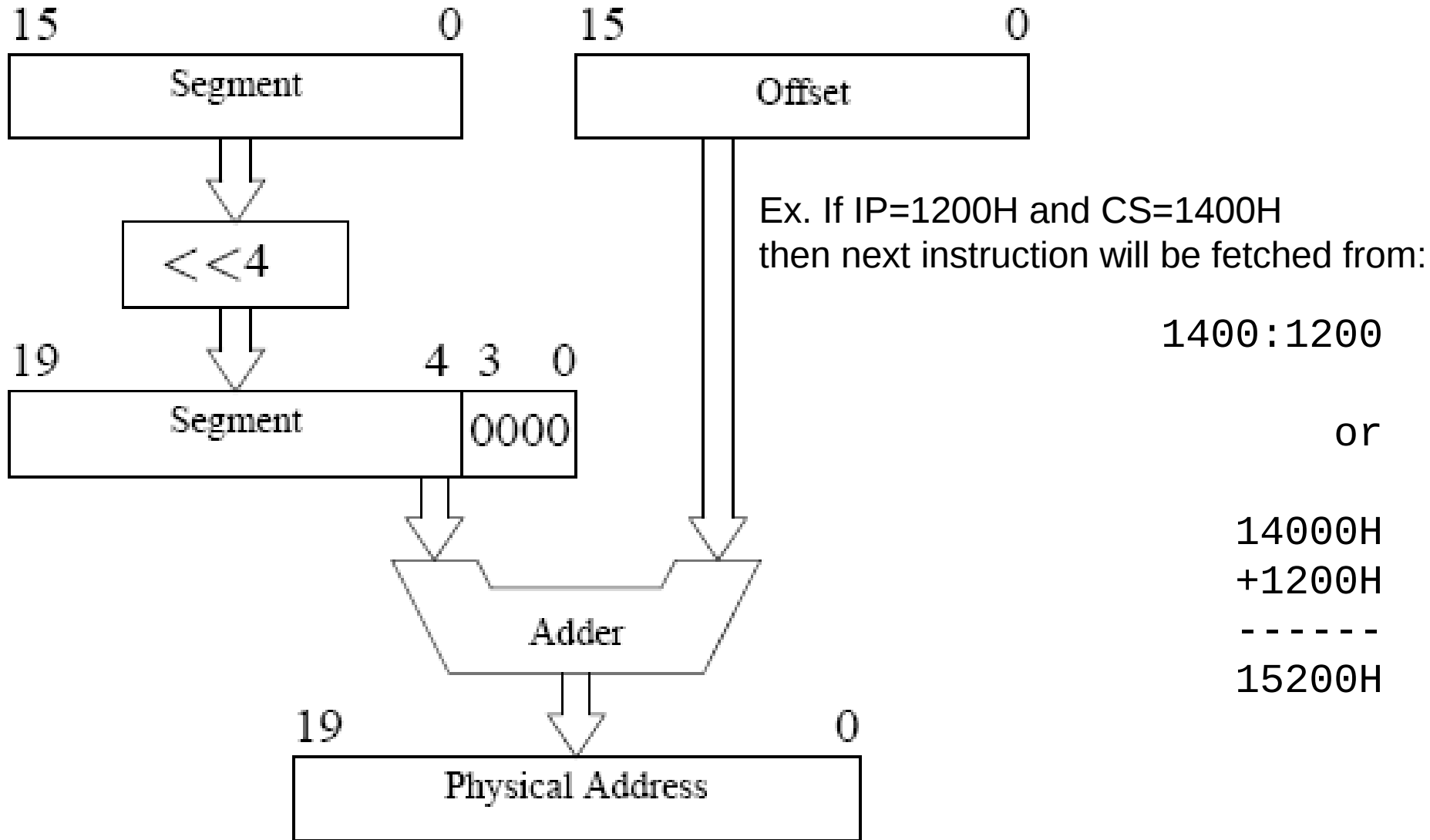
Address Generation

- Two types of address generation:
 1. Real Mode (the 8086/8088/186 can only operate in this mode)
 - Allows the μ P to address the first 1Mbyte of memory only.
 - The first Mbyte of memory is called *real* or *conventional* memory.
 2. Protected mode (80286...)
 - This mode uses the segment register contents (called a *selector*) to access a *descriptor* from a descriptor table.
 - The descriptor describes the memory segment's location, length and access rights.

Real Mode Address Generation

- Memory addresses consist of a segment address plus an offset address.
 - The *segment address* defines the start of a 64K block of memory.
 - The *offset address* selects a location within the 64K memory segment.
 - Memory locations are often written as:
segment:offset
C000:04BA

Real Mode Address Generation



Real Mode Address Generation – Funky Rules...

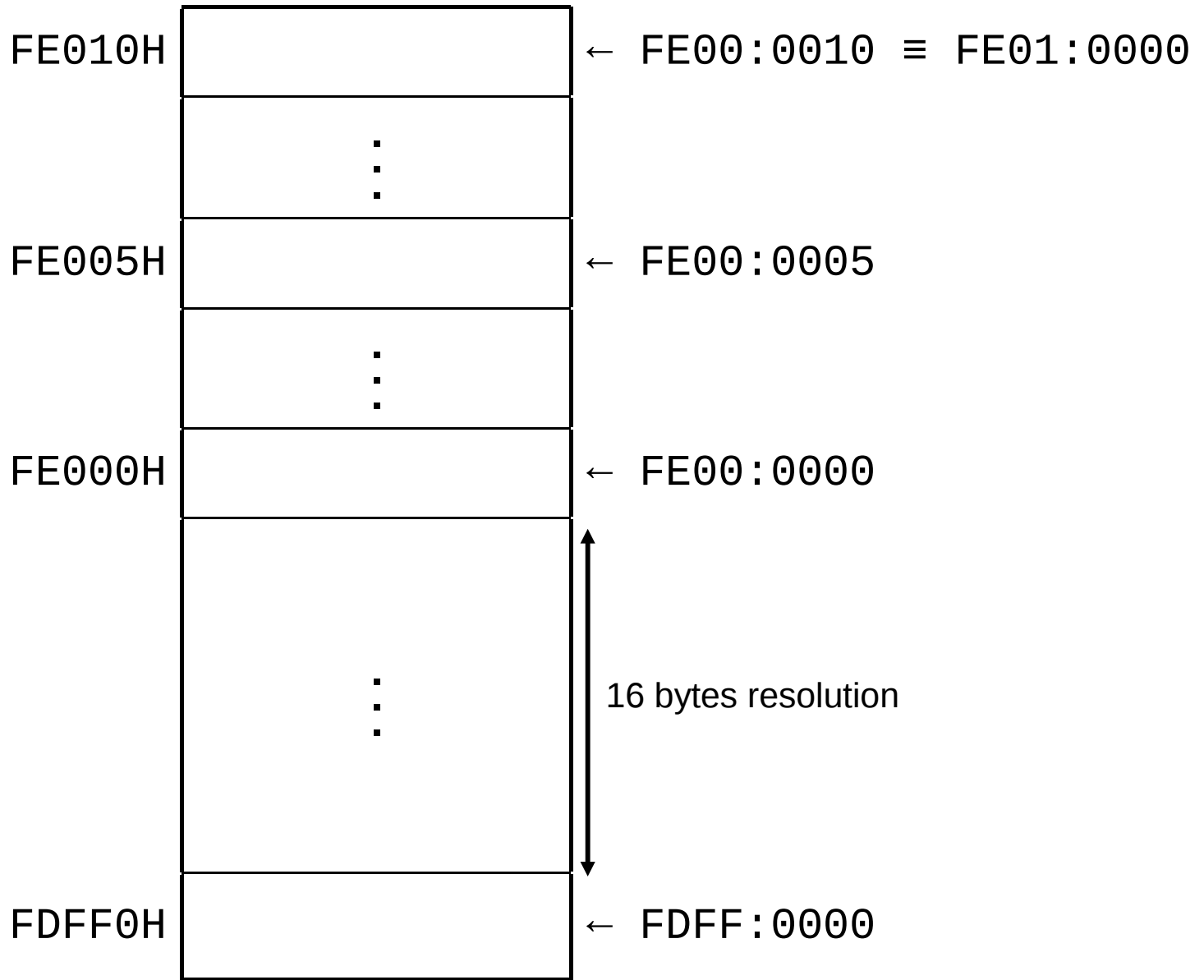
- The μP has a set of rules that apply whenever memory is addressed, which define the segment and offset register combination used by certain addressing modes.

Segment	Offset	Special Purpose
CS	IP	Instruction address
SS	SP or BP	Stack address
DS	BX,DI,SI, 8bit # or 16bit #	Data address
ES	DI (for string instruction)	String destination

Real Mode Address Generation

- Notes:
 1. Memory segments (i.e. the 64K blocks) may overlap if full 64K are not needed.
 2. The segment-offset scheme allows programs to be relocated in memory (on 16 byte boundaries).
 - Move the existing contents to the new physical location, then update the segment register.

Segment Resolution



Real Mode Address Generation - Examples

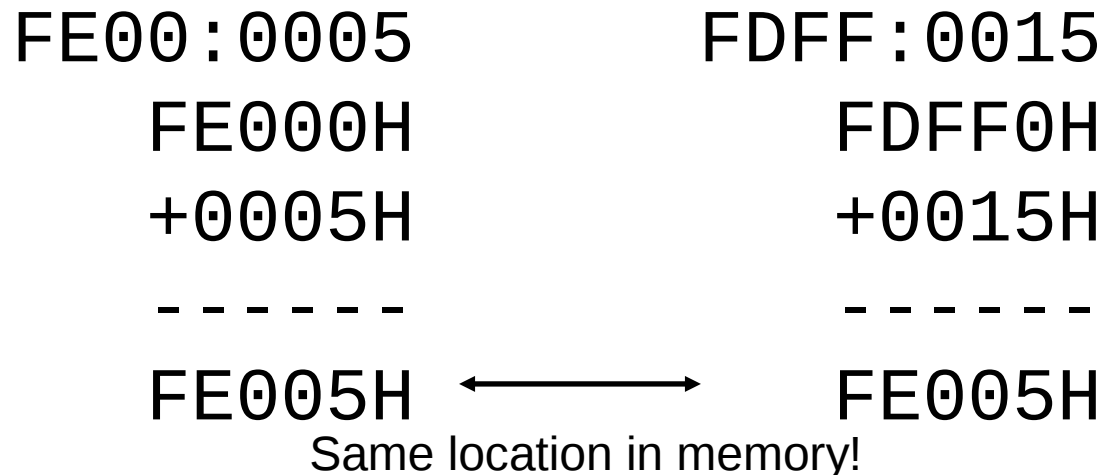
- Ex 1:

MOV DL, [BP]

Uses an absolute (i.e. physical) source address of:

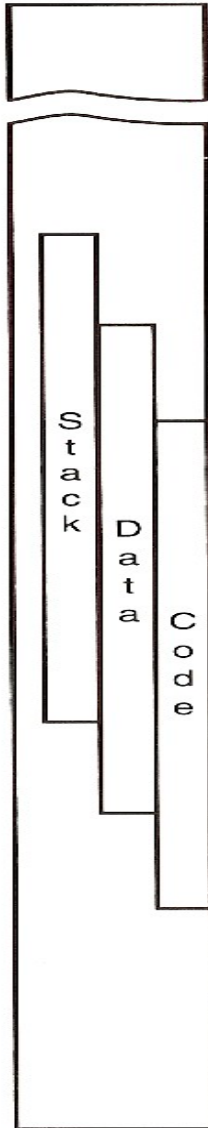
$$SS \times 16 + BP$$

- Ex 2 (overlap):

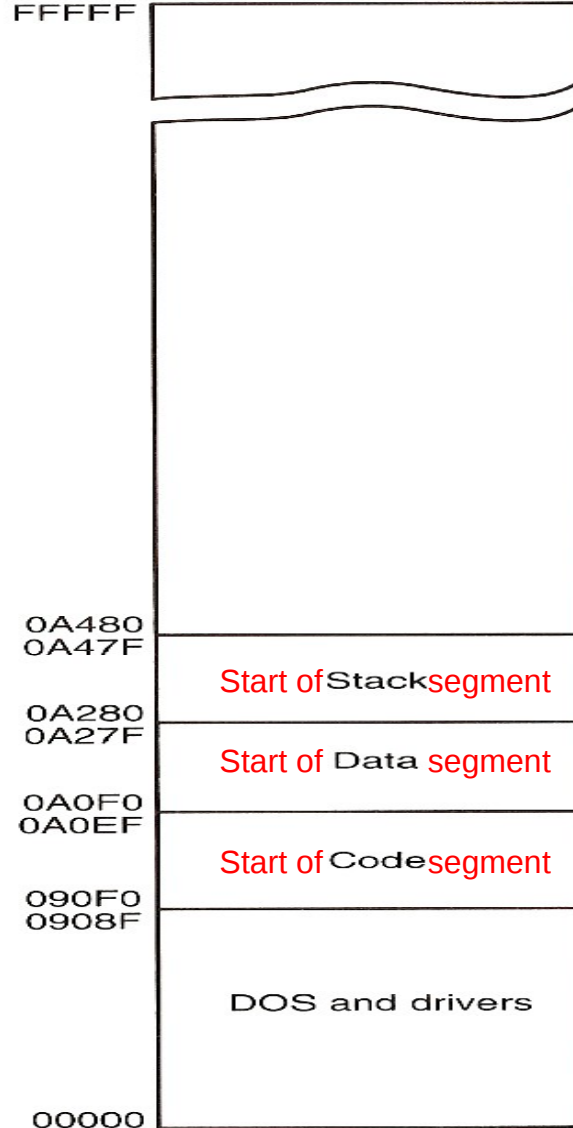


Real Address Mode Generation

Imaginary side view detailing segment overlap



Memory



Start of Stack segment

Start of Data segment

Start of Code segment

0 A 2 8 SS

0 A 0 F DS

0 9 0 F CS

Stack Operation

- The stack is a Last-In, First Out (LIFO) queue.
- The stack grows **down** in memory (i.e., towards 0).
- Only **words** (8086-80286) and double words (80386...) can be pushed/popped on/off the stack.
- POP CS is **NOT** allowed.
- Typically, initialize SP to 0H. Will decrement to 0FFFFH on first PUSH to point to top of segment.

Stack Operation Example

- 1 MOV BX, 1234H ; BX ← 1234H
 - 2 PUSH BX ; PUSH 1234H onto stack
 - 3 POP AX ; AX ← 1234H from stack
- High byte transferred first so that value is LITTLE ENDIAN in memory.
- 2a $SS \times 10H + SP - 1 \leftarrow 12H$ High order
 - 2b $SS \times 10H + SP - 2 \leftarrow 34H$ Low order
 - 2c $SP \leftarrow SP - 2$
 - 3a $AL \leftarrow SS \times 10H + SP$ (34H)
 - 3b $AH \leftarrow SS \times 10H + SP + 1$ (12H)
 - 3c $SP \leftarrow SP + 2$