The Intel Microprocessors 8086/8088 Architecture

Module - 1

Prepared by:
Dakshata Panchal
SFIT, Mumbai

Syllabus to be covered

- Introduction
- 8086/8088 CPU Architecture
- Programmer's Model &Functional Pin Diagram
- Memory Segmentation
- Banking in 8086
- De-multiplexing of Address/Data bus
- Study of 8284 Clock Generator
- Study of 8288 Bus Controller
- Functioning of 8086 in Minimum mode and Maximum mode
- Timing diagrams for Read and Write operations in minimum mode and maximum mode
- Interrupt structure and its servicing

Introduction

Why learning Microprocessor???

Are Microprocessor used in this world???

Yes!!!! Everywhere from the one you are watching this video lecture!!

Routers/servers they all uses μ_p!!!!

Introduction

Why do you require Microprocessor??

Is μ_p there in fan?? **NO**

Tube light?? **NO**

Remote control?? **YES**

Mobile phones?? **YES**

Computer?? **YES**

So, basically wherever "**Programming is involved Microprocessor is present**"

E.g. **A C Remote Controller** (Various buttons are there to perform distinct operations)

(a)First, There is a program which reads the keys being pressed

(b)Sense appropriate data through serial wireless communication

(c)Receiver receives the data and analyze which button was pressed and performs appropriate action.

In other words, sending also requires μ_p and Receiving also requires μ_p

E.g. **Traffic Light**

For displaying three colors RED, YELLOW and GREEN Programming Involved and so μ_p is present to process these information.

E.g. **Railway Station**

Indicator which displays the train timings, platform no. and so on.

Inside the train you gets message

Next station is Borivali and so on!!!

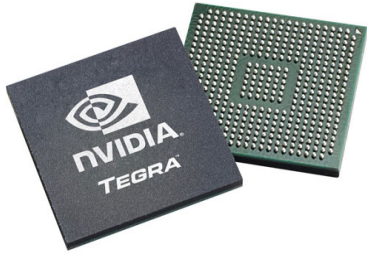
"All these are programs and all of them are executed by Microprocessor"

Where all are these microprocessors used??

- **Computers** – Desktops, laptops, tablets, calculators
- **All mobile phones** – smartphones and basic phones too!
- **Home Appliances** – washing machine, microwave oven, AC, smart TV, DVD players, home automation systems, children toys, etc.
- **Instrumentation** – frequency counters, signal generators, synthesizers, etc.
- **Others** – Traffic light control, game machines, industry controllers, communication systems, flight control systems,

MICROPROCESSOR??

Heart of the Computer is its Microprocessor



What is a microprocessor?

- Millions of transistors (tiny electronic devices that carry electric charge) are integrated on a single chip, called the microprocessor.
- Each transistor can be in “OFF” or “ON” state i.e.,

  welcome to the world of 0s and 1s
- With perfect circuitry combinations and programming logic used, these transistors states can be changed, for what?

  perform arithmetic and logical operations
- Is everything we perform on our computer and phones just arithmetic and logical operations?

  YES

How does this microprocessor work?

- On electricity, of course!!
- You can play around by

  giving power to a transistor, set to the ON state

  or shutting off power to it, reset to the OFF state
- When a user gives an instruction, e.g., press “a” on the keyboard
  1. The system recognizes a key is pressed
  2. Which key is pressed (ASCII value noted)
  3. Where should this input go to? (MS Word, browser search or error)
  4. What is its purpose? (type “a”, select all “ctrl+a”, etc)
  5. Perform the necessary action
- And **all these steps are a sequence of 0s and 1s**

Then, CPU = Microprocessor?

NO. The CPU is a Microprocessor, but not the only microprocessor in your system!

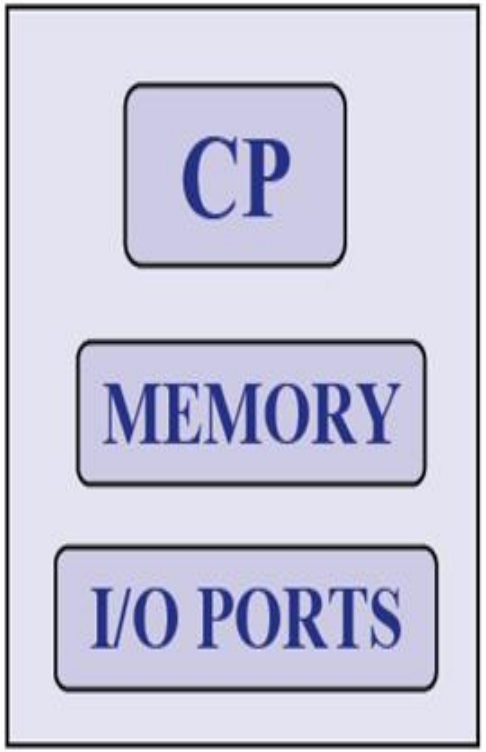
- The **Central Processing Unit** of a computing device is embedded on a single chip, often called as a **microprocessor**.
- But a microprocessor need not always be the CPU of that device!
- A microprocessor can be used for other purposes too, such as **NPU** (Network Processing Unit), **GPU** (Graphics Processing Unit), **APU** (Audio Processing Unit), etc.
- The quadcore microprocessor has four CPUs in it.

Microprocessor v/s Microcontroller

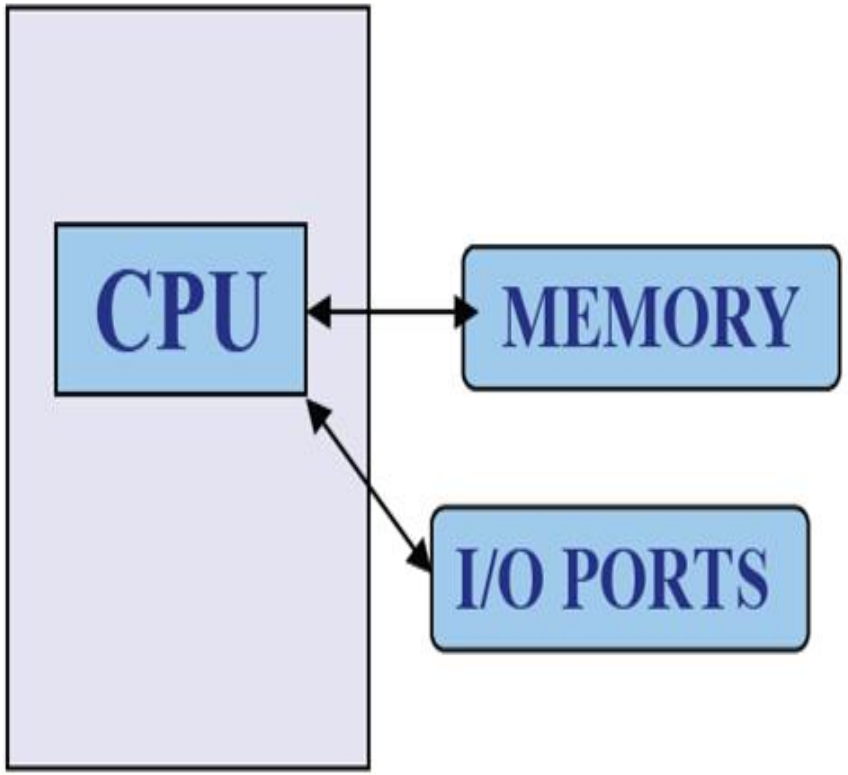
- The entire CPU on a single chip = Microprocessor
- The entire computer on a single chip = Microcontroller!
- Microprocessors are general purpose processors while microcontrollers are designed for a specific purpose.
- Most home appliances like washing machine, oven, control systems etc. have a microcontroller in it.

Microprocessor v/s Microcontroller

Microcontroller



Microprocessor



Before Microprocessors, CPUs were-

- 1st Gen (1940-56): Vacuum tubes and magnetic drums
- 2nd Gen(1956-63): Transistors
- 3rd Gen (1964-71): Integrated Circuits
- 4th Gen (1971-present): single chip **Microprocessors**
- 5th Gen (present-future): Multicore/ Parallel processing

Basics of μ_p

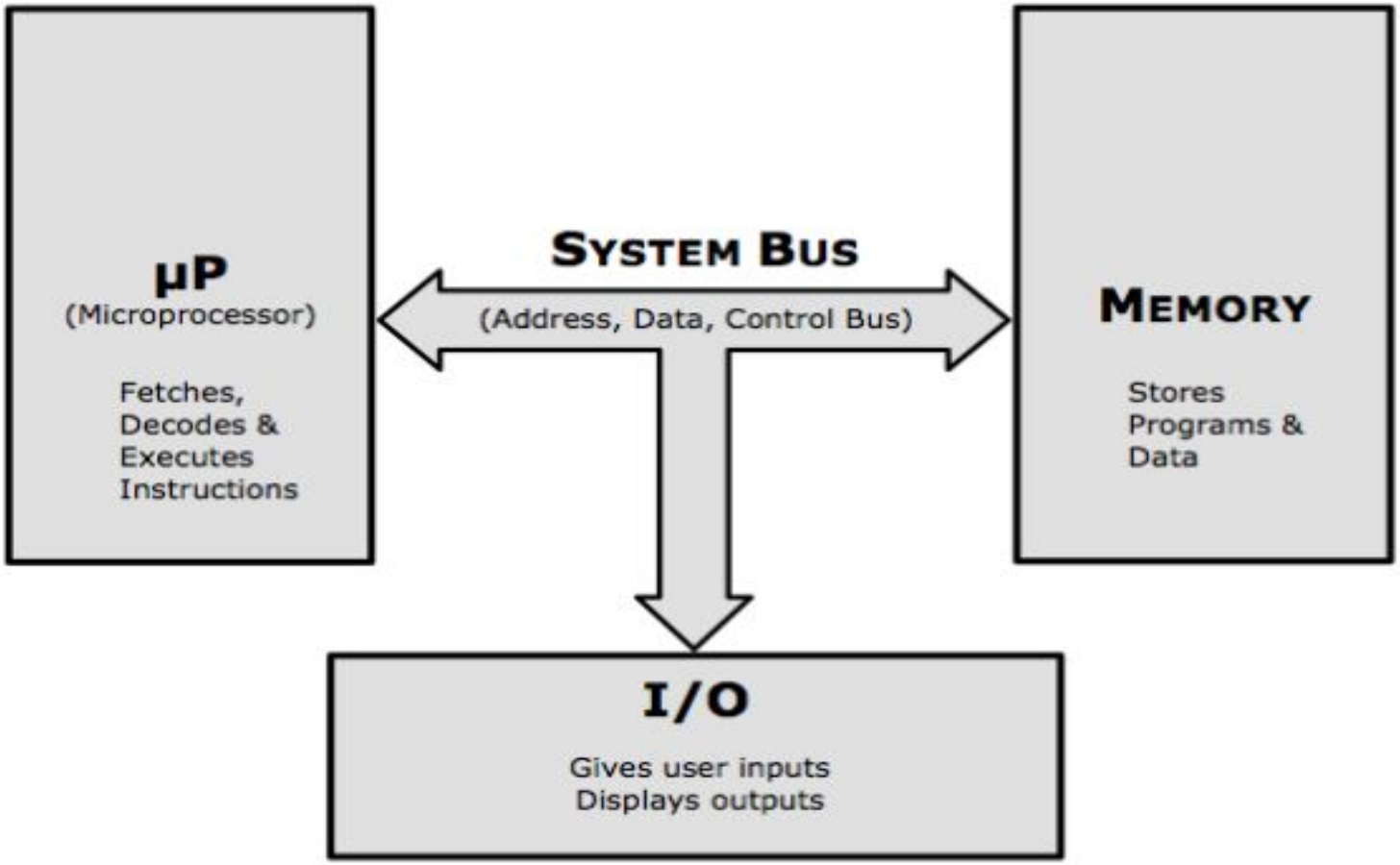
- 8085 - Successful Processor but not very powerful
- **8086** – Remarkable Processor, Intel combined with IBM and made the PC

-80186
-80286
-80386
-80486

} 8086 family

-80586(Pentium 1,P2,P3,P4…..Core2duo,**Ci7**)

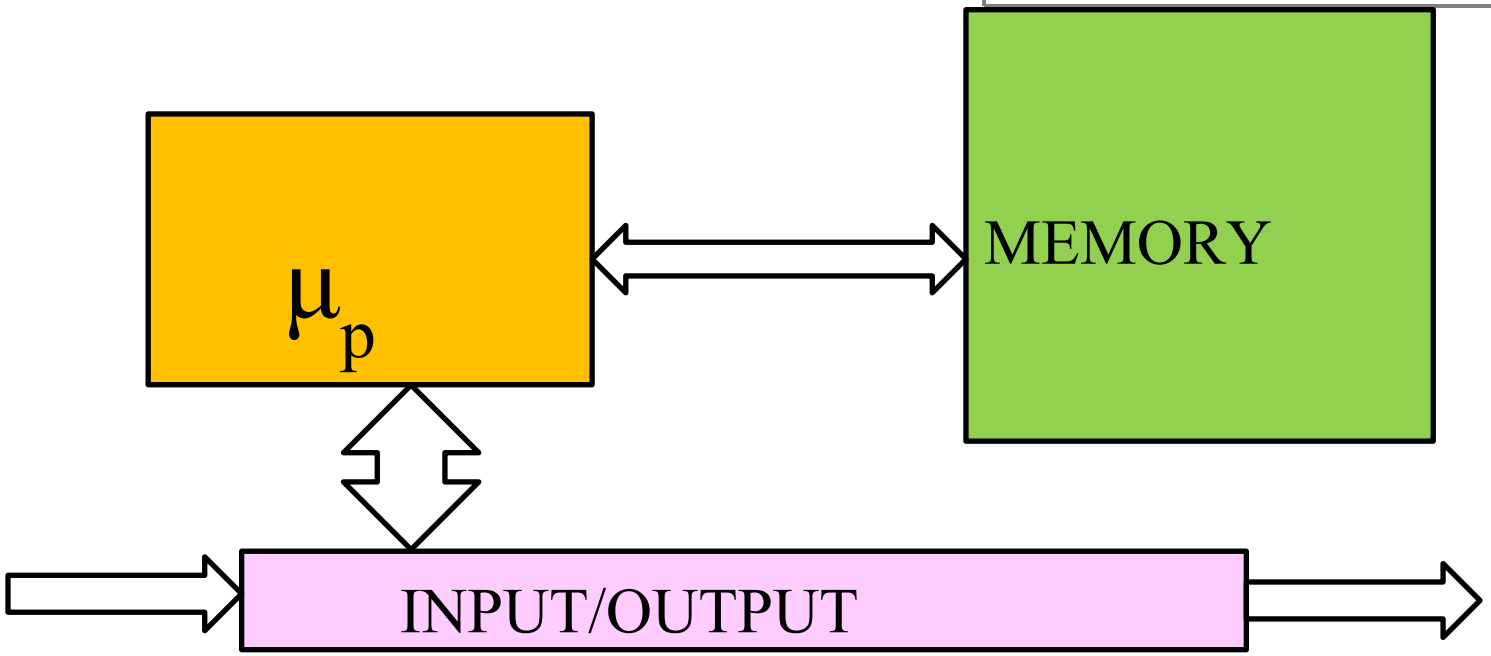
Basics of Computer System



The Processor- μ_p

- Main Functions of μ_p is
  - FETCH
  - DECODE
  - EXECUTE Instructions

Instructions are the part of a program and Program is stored in a memory



1. Firstly,μ_p fetches an instruction from the memory.
2. It then decodes the instruction.
3. μ_p understands the operation to be performed and so executes the instruction

This Entire Process is called INSTRUCTION CYCLE

Memory

- Memory is used to store information. It stores two kind of information.. **Programs** and **Data**
- Eg:

Programs	**Data**
MS Word	Word Documents
Video Player	Videos
Wats up	Messages
Songs player	Songs

- All programs and data are stored in the memory in digitized form i.e 0's and 1's.
- Memory is a series of locations. Each location is identified by its unique address and Each location contain 1 Byte(8 bits) of data

Memory

There are various forms of memory devices:

- The main memory is also called as Primary memory consists of RAM and ROM.
- Hard disk, Floppy Disk,CD/DVD, etc are Secondary devices.
- So, For major portion of this subject we will be dealing with 8086 processor, it will be in your best interest to think of **Primary memory** only, whenever we speak of memory
- Because secondary memory and high speed memories were implemented much later in the evolution of processors as the demand for mass storage and high speed performance started increasing
- **In short, Word "Memory" refers to primary memory i.e RAM and ROM**

I/O Devices

- I/O Devices are used to enter programs and data as inputs and display or print the results as outputs
- For eg: keyboard, mouse will take the data as an input and printer (prints) or monitor (displays) as an output.
- A device like touch screen performs dual functions of both input and output.

Quick Revision

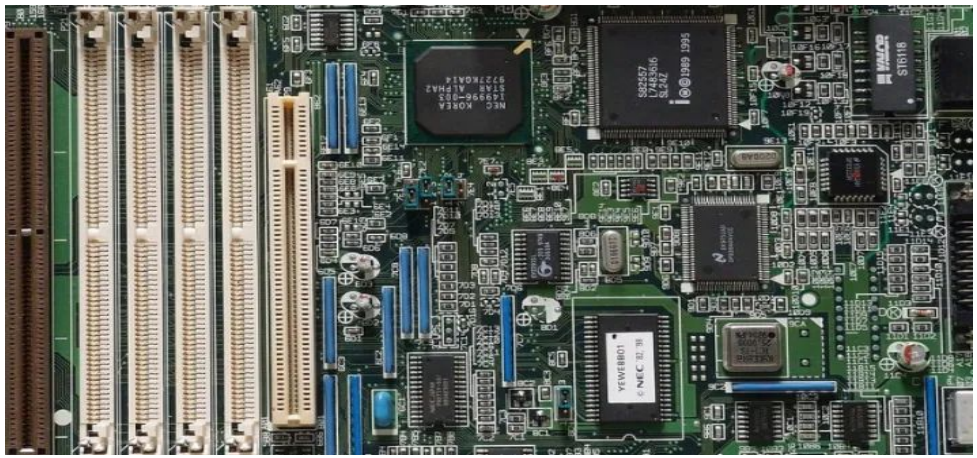
What does memory do? Store programs and data

What does Processor do? Fetch, Decode &Execute Instruction

What does I/O do? gives input and produces output

Bus : Set of lines

- Internal bus in Pen drive.
- Bus in motherboard.



Bus : Set of lines

Bus is used to transfer information.

For eg: while listening a song.

At first, song is in memory, processor will process it and finally you listen to it with the help of a speaker.

Size of the bus is in the form of bits.

8 bits = 8 lines

16 bit = 16 lines and so on.

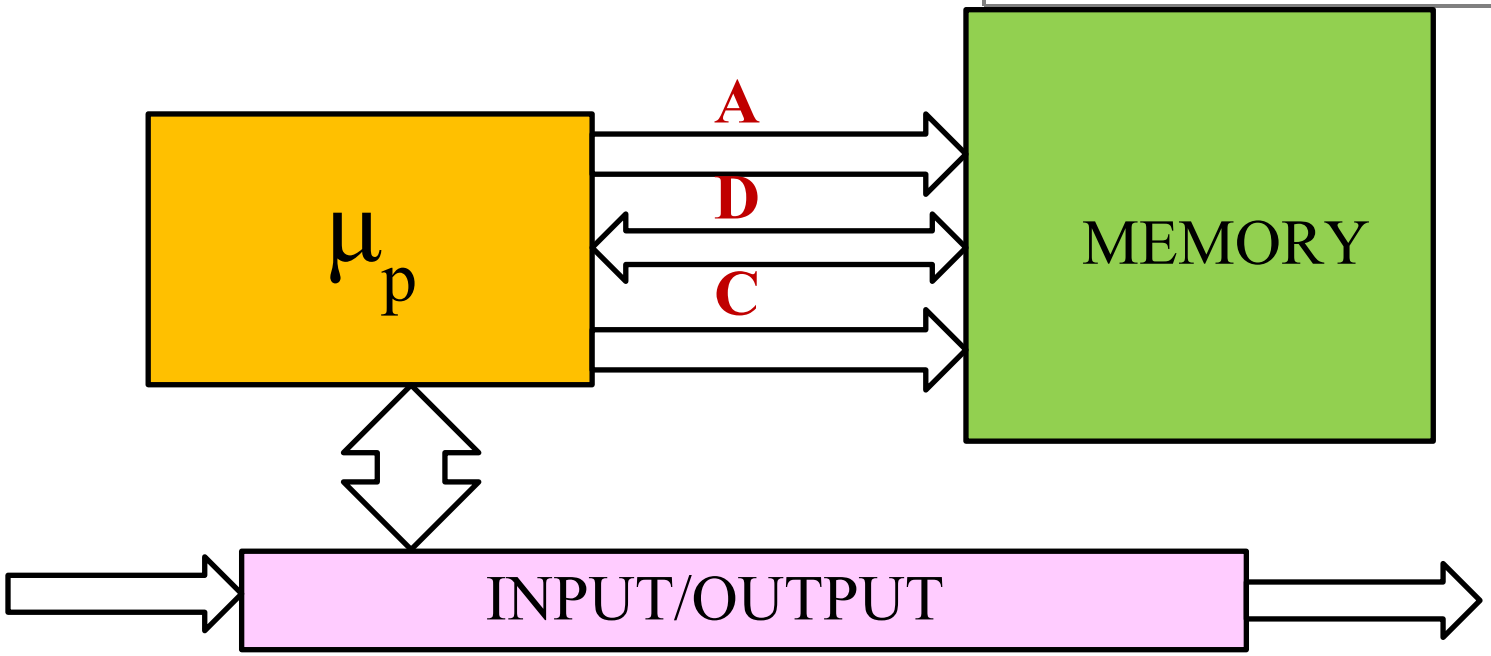
Bus : Set of lines

There are 3 types of Bus:

1. Address Bus Used to identify the locations
2. Control Bus Uses two signals READ & WRITE

   READ Processor is getting data

   WRITE Processor is sending data
3. Data Bus for transferring data



- Address bus and Control Bus are unidirectional
- Data Bus is Bi Directional.

Explanation of Read and Write operation

8086 Architecture

To understand the concept of 8086 architecture following concept need to clear.

- Pipelining
- Memory banking
- Memory Segmentation
- Flag Register

Pipelining

The Biggest reason, why a processor have become so faster is

" PIPELINING"

There are 3 major factors which makes processor faster

1. Increase the size of data bus
2. Increase the clock frequency
3. Piplining

Pipelining

Non pipelined 8085

T1	T2	T3	T4	T5	T6	T7	T8
Fetch1	Execute1	Fetch2	Execute2	Fetch3	Execute3	Fetch4	Execute4

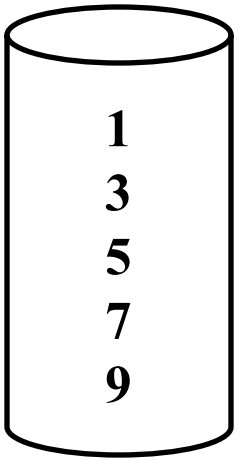
Pipelined 8086

T1	T2	T3	T4	T5
Fetch1	Execute1			
	Fetch2	Execute2		
		Fetch3	Execute3	
			Fetch4	Execute4

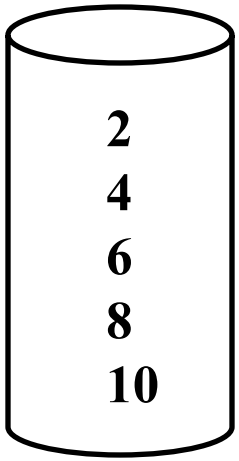
Pipelining means fetching the next instruction while executing the current instruction

Example

- Pentium uses 5 stage pipelining with 2 pipes (fetch, decode,execute,store,write back)
- Pipe 1



Pipe 2



- At a time it deals with 10 instruction at a time!!

Cont..

- Increasing the no. of pipelining = Increasing the parallelism. And so its performance.
- Pentium 4 has 4 such pipes with 20 stages i.e At a time it deals with 80 instruction. And so it will be massive increase in performance.
- It also has disadvantages like data dependency, Branching.

8086 Memory Segmentation

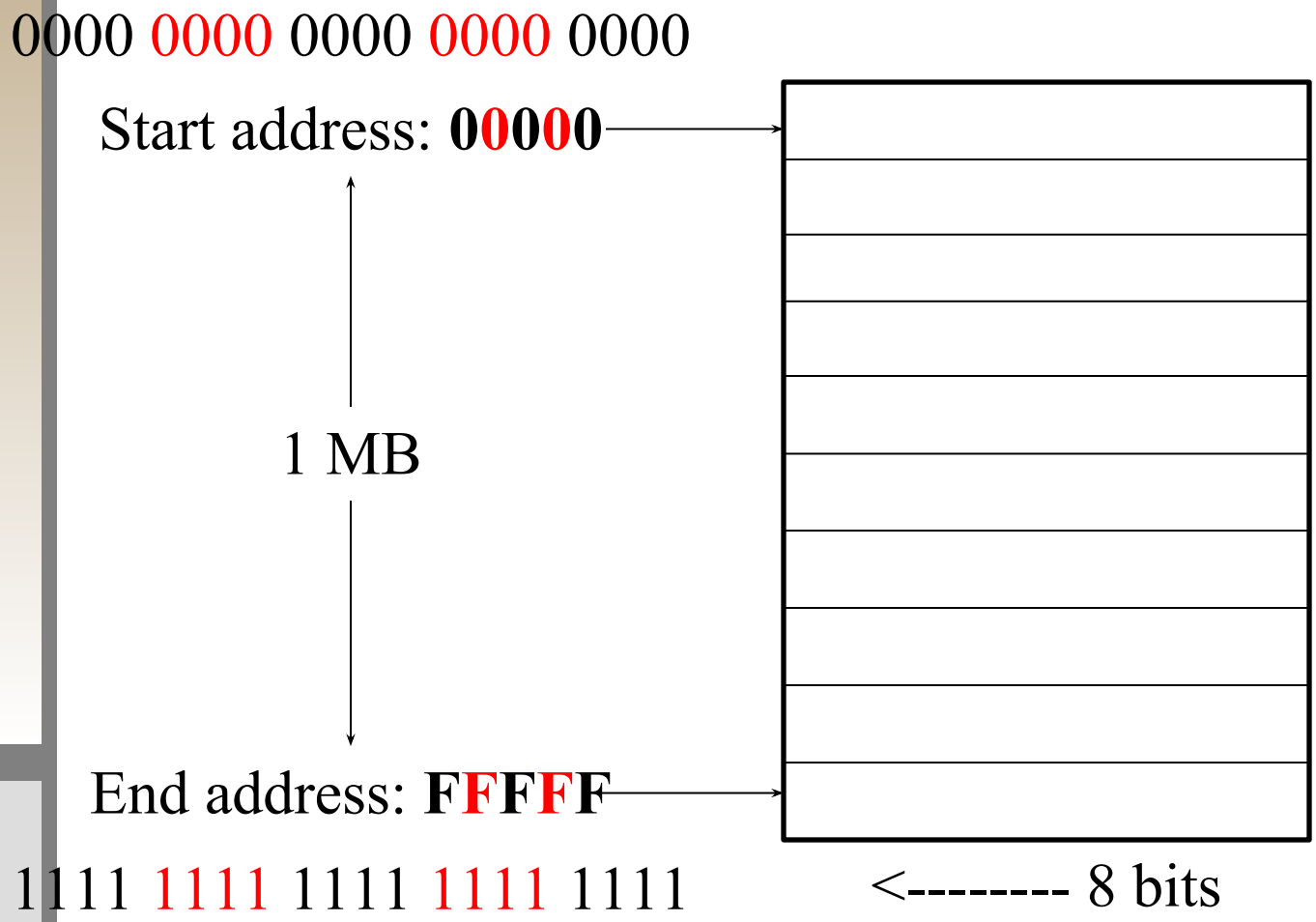
What is memory?

- A bank of 1 byte locations, each having its own unique address.
- 2^{20} = 1 MB = 1 Mega Bytes (for 8086 µp)
- Memory lies outside the processor, but it can be accessed by it.

What is the use of memory??

- To store programs
- To store data

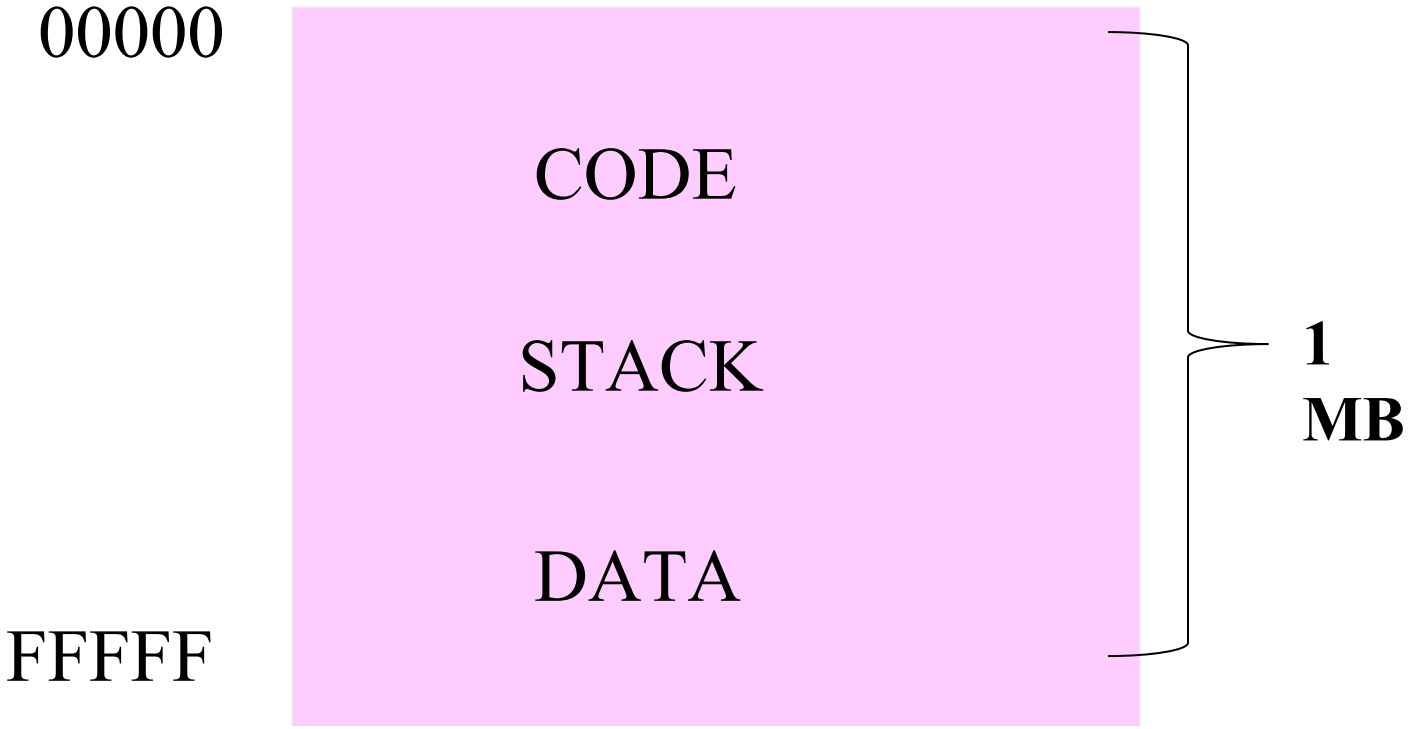
8086 Memory Bank



Is it a problem or a solution??

To understand this let's have glimpse on predecessor processor.

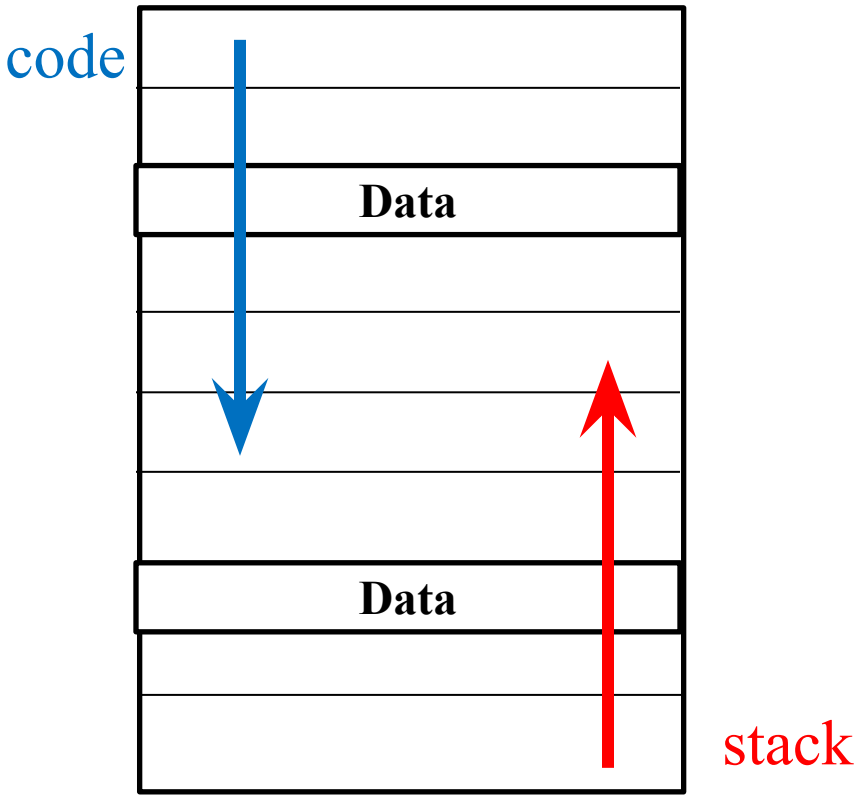
Assume Memory with no segmentation.



How Memory is Used?

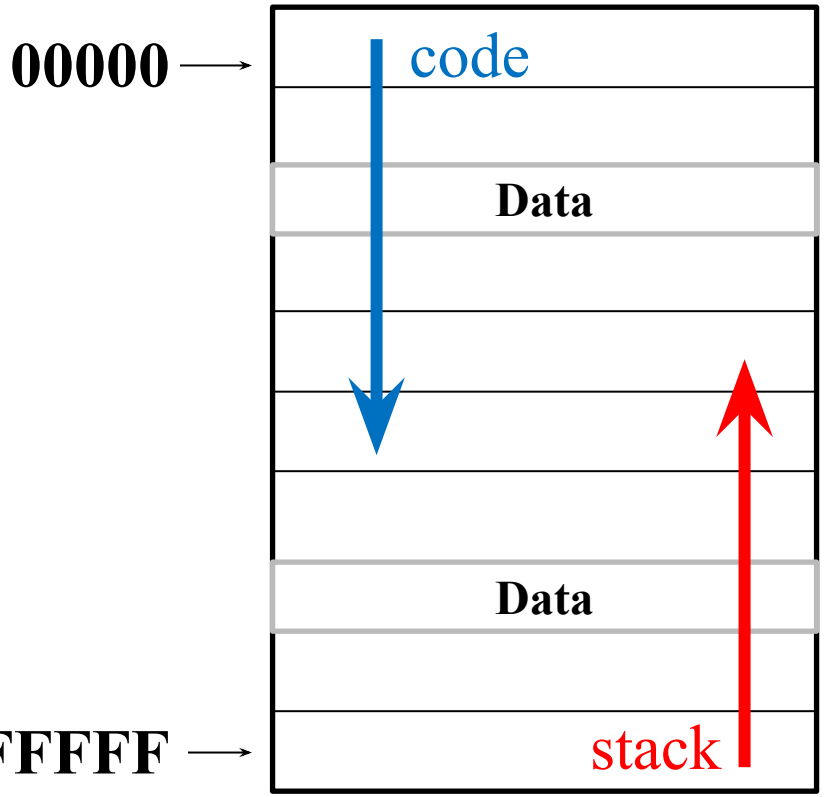
- Programs are stored sequentially
  from 00000 □ FFFFF
- Data can be stored randomly anywhere
- Stack is also stored sequentially, but opposite direction
  from FFFFF □ 00000

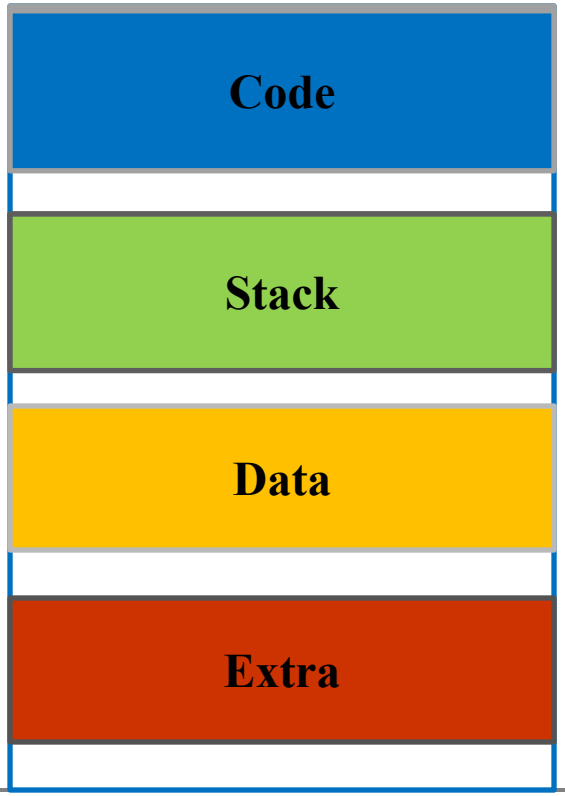
Memory Overwriting



Solution?

Memory Segmentation





Who create the segments??

Programmer

But it is managed by the processor

When a program is loaded, processor updates the segment registers with the start addresses of the segments in use.

An offset register starts from 0000

Features

- Prevents over-writing of memory
- Organized management of memory

  (Gave birth to the concept of files and folders)
- Allows 20-bit physical addressing with 16-bit registers.

???

In Computer, everything you store in a files???

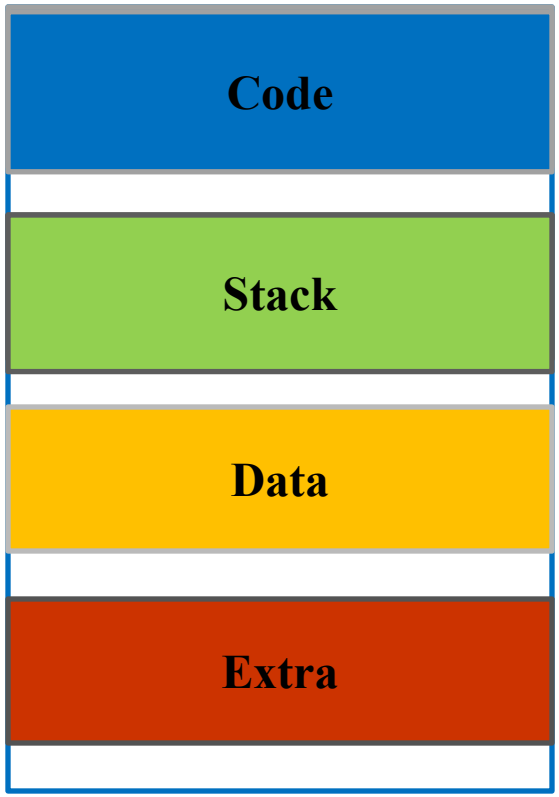
Yes

Example: when you write anything in your word file you never bother about overwriting.

Because files do not over write each other

What are these files ???

FILES ARE MORDERN VERSION OF SEGMENTS



All applications are independent Code segments

Songs, videos, images are all data segments

Who creates file??

We

Who manage segment??

Processor

8086 addresses are 20-bits long.

In a computer, 1 location is always 8 bits long.

- 8 bits = 1 byte
- 16 bits = 2 bytes
- 20 bits = 2.5 bytes

Problem: 20 bits is not computer-compatible

Solution: use 24 bits = 3 bytes (with 4 bits always set to 0000)

Again problem: huge amounts of data transfer wastage

- Who created these problems?

WE

- Why did we create these problems?

We are not happy with 2^{16} = 64 KB memory

We want to access 1 MB memory

with 16 bits!

How?

We started using the concept of virtual address

Example of a College

- There are 500 students in a college. Each student has a unique number for identification 001 – 500.
- There are 8 courses in this college.
- Given a student id will it be possible to guess the course he has enrolled for? (without looking at the database)

Example: student id 246

- Assume each course can handle only 100 students.
- The roll numbers of students are assigned from 00 – 99 for each course
- A course id is maintained 1 – 8.
- The student id is now assigned as xyy where x is the course id and yy is the class roll number.

Example: 685

- Consider a database that allows you to enter two digits for of student id

  1st enter 06 (course id)

  then 85 (class roll no)

- The first digit of the course id is always going to be 0, but it cannot be avoided

  because you have to enter two digits

In other words, every time you won't need to give segment address and offset address. Segment address are given only once in the beginning of the program to identify , where the segments located

- Basically we do whole programming in 16-bit address

Why 8085 did not have segmentation?

Because its address bus was 16-bit. So, its Physical address itself was compatible numbers so no need to create virtual address

In 8086, Segmentation is optional or compulsory??

Its Compulsory because it has 20-bit address and half of the byte will always waste

Memory Segmentation

What is Memory Segmentation?

- Access a 20-bit address using 16-bits!
- Split the 20-bit address into two parts
  - Segment address (16-bits)
  - Offset address (16-bits)
- Regain the original physical address with the help of a small calculator

  P.A. = Segment address * 10
  + offset address

Note: Segmentation is not optional for the 8086 architecture, it is a necessity.

Minimum Size of a Segment

- Segment Registers store the Most Significant 16 bits of the 20-bit physical address
- The Least Significant 4 bits are not recorded.
- Hence 8086 imposes that a segment can start only at such a location whose address is a multiple of 10
- Example: 20000 or 50030 or 12340
- It can never start at 12345 or 5003A, etc.

Maximum Size of a Segment

- Offset register is 16-bits
- Always starts at 0000
- Last address possible = FFFF
- 0000 – FFFF = 2^{16} = 64KB
- **Maximum size of a segment = 64 KB**

Minimum Size of a Segment

- Assume a segment starts at 51230
- Since it cannot start at 51231 – 5123F
- The next location available is 51240
- 51240 – 51230 = 16 bytes
- **Minimum size of a segment = 16 bytes**

Examples

CS = 5123

IP = 00C6

PA = ?

512F6

SS = 2000

IP = 123A

PA = ?

2123A

Questions for Practise

1. What is segmentation? What are the advantages of segmentation?
2. What is the minimum and maximum size of a segment? Justify.
3. What is effective address? Explain with examples

8086 Architecture

How is a Processor defined??

Processors are specified in terms of bits

How many bits does the Register of a Processor have??

16-bit Processor (8086)

- Has 16 data lines
- All internal Registers are 16-bit long

32-bit Processor

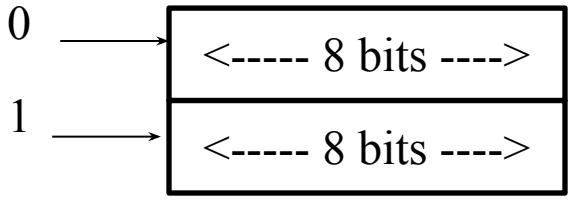
- All Internal Registers are 32-bit long
- Has 32-bit data bus

The 8086 has 16-bit data bus and a 20-bit address bus

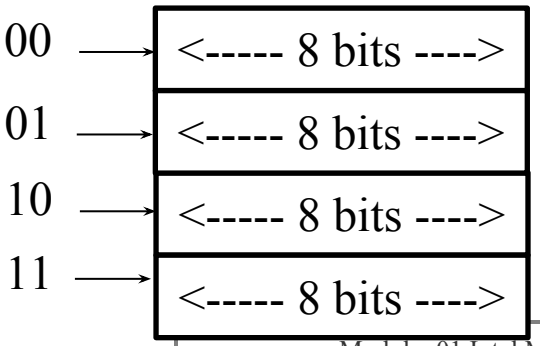
8086 architecture is 16 – bit Processor means--

- It can transfer 16 bits of data at a time.
- Its ALU is 16 bits, i.e. it can perform arithmetic and logical operations on 16- bit data at a time
- All its registers are 16 bits

- Suppose we have 1 bit for addressing, we can access two (2^1) memory locations.



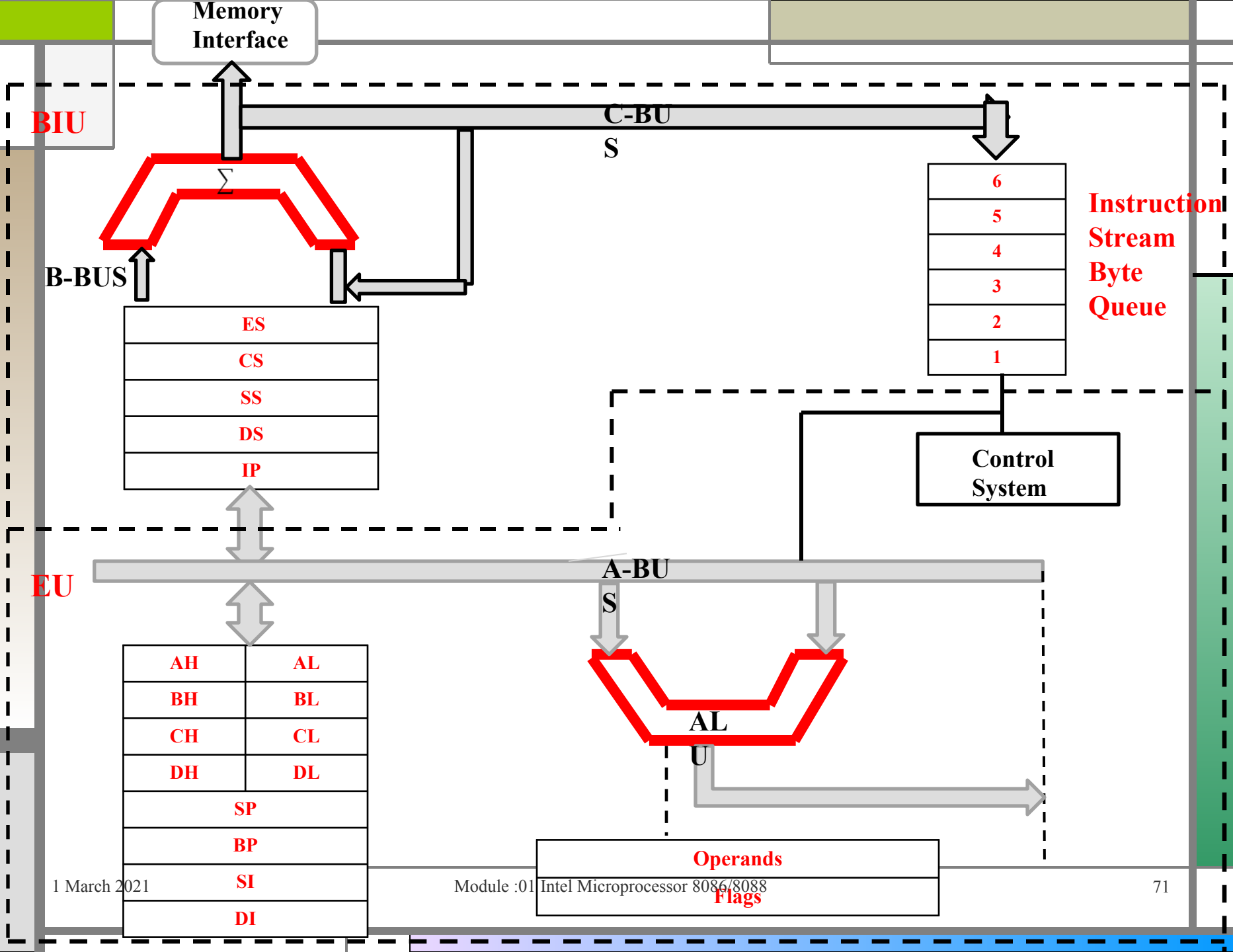
- If we have 2 bits for addressing, we can access four (2^2) memory locations



The 8086 has 20 bits for addressing, i.e., It can access 2^{20} bytes or 1 MB data

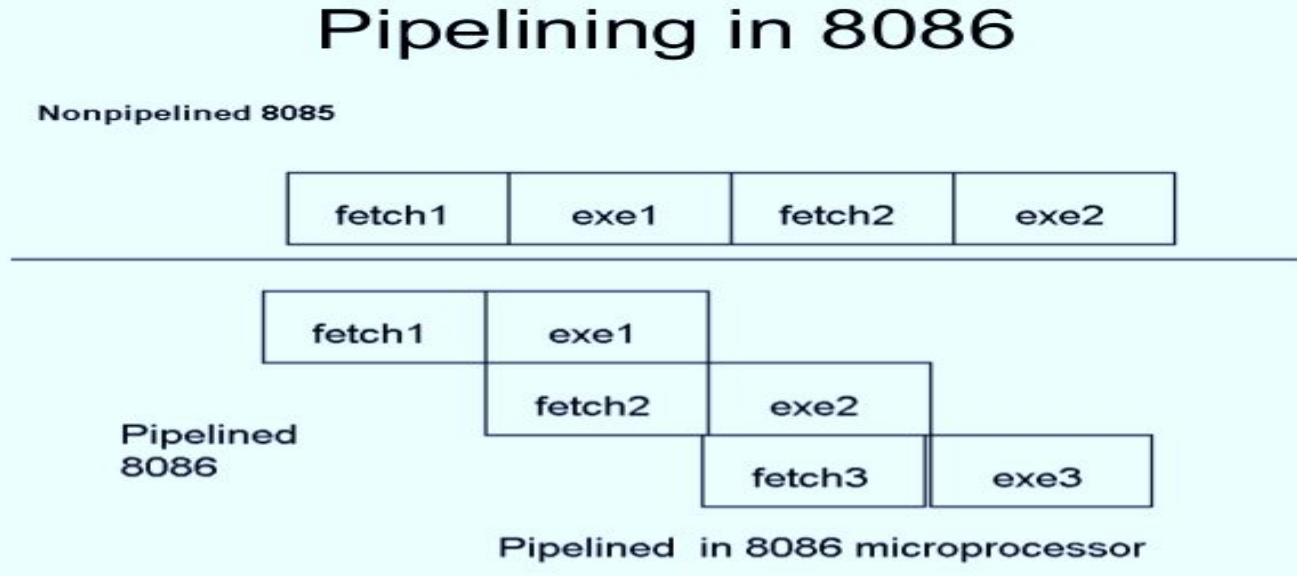
There are two sections in the architecture of a 8086 Processor.

- Bus Interface Unit (BIU)
- Execution Unit (EU)

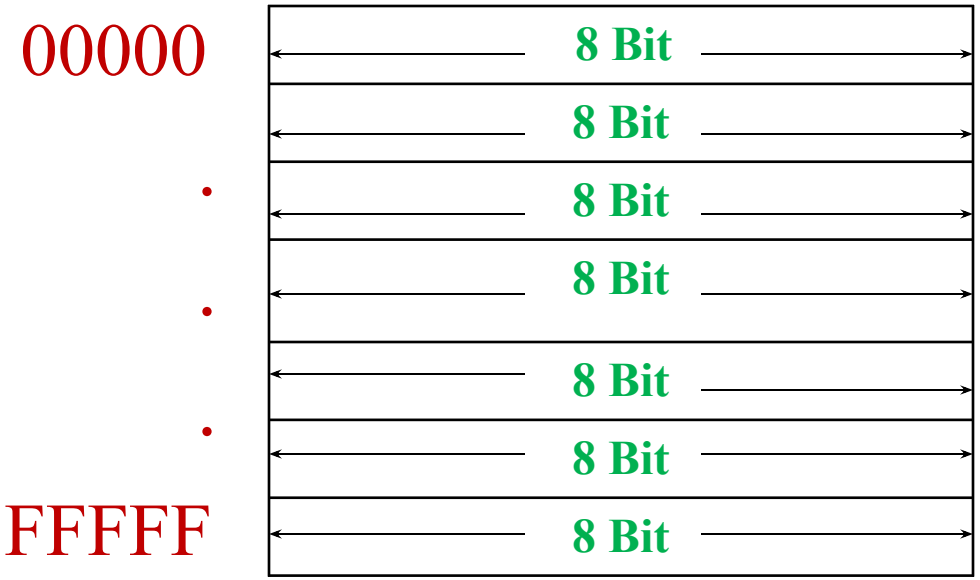

Memory Interface
BIU
C-BUS
∑
B-BUS
ES
CS
SS
DS
IP
6
5
4
3
2
1
Instruction Stream Byte Queue
Control System
EU
A-BUS
AH
AL
BH
BL
CH
CL
DH
DL
SP
BP
SI
DI
ALU
Operands
Flags

Why two sections????

- Because data transfer rates are usually slower compared to execution rate.
- Pipelining instructions will avoid processor being idle.



- This 16- bit processor has a 20-bit address bus
- Width of a memory locations is 8 bits.
- The address of that memory location is 20 bit long



- As a programmer or a user cannot remember the 20-bit address of everything stored in the memory, we use linguistic names.
- So,

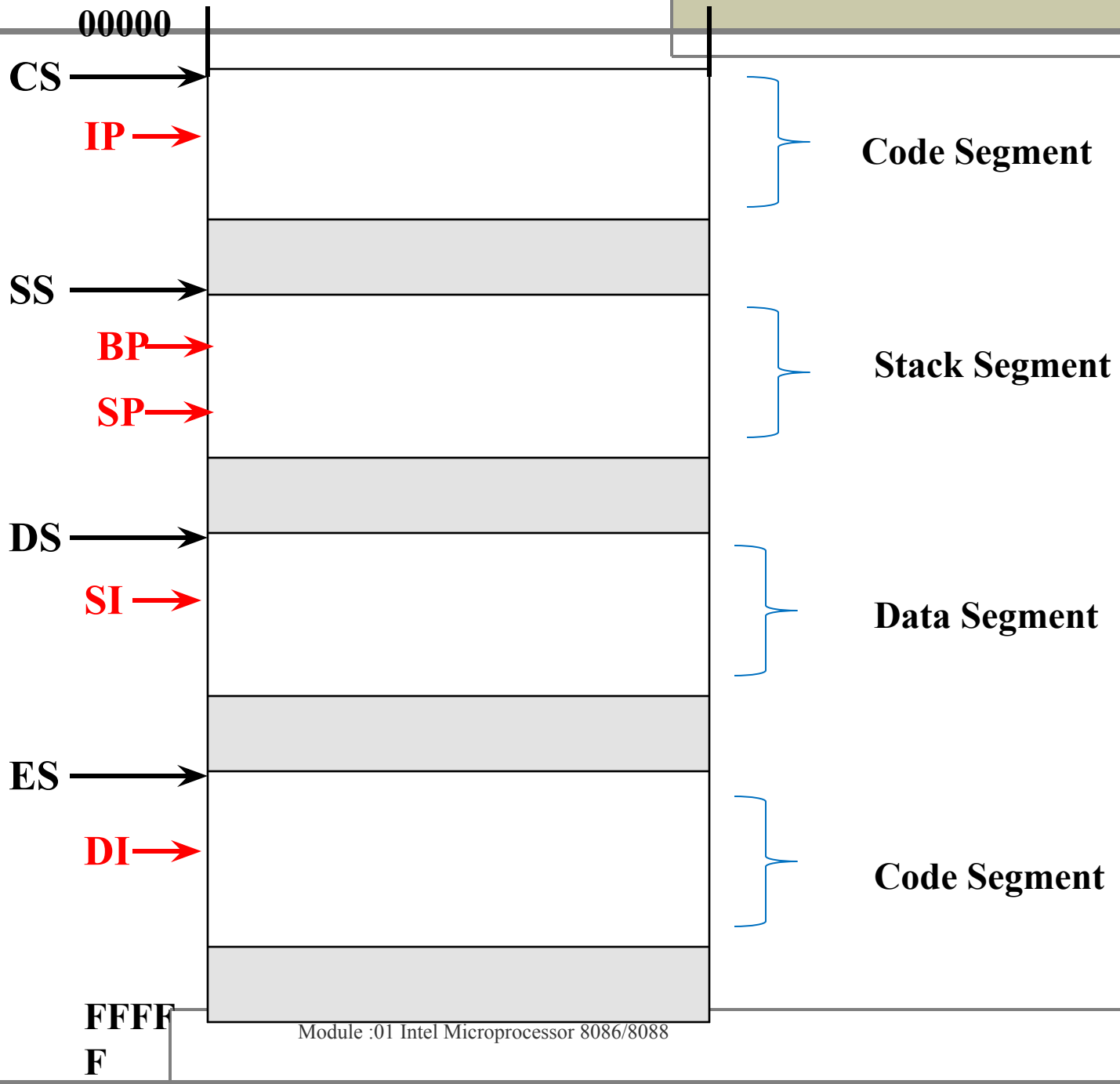
Entire memory is renamed as 4 segments:

CODE SEGMENT

STACK SEGMENT

DATA SEGMENT

EXTRA SEGMENT


00000
CS
IP
SS
BP
SP
DS
SI
ES
DI
FFFF
F
Code Segment
Stack Segment
Data Segment
Code Segment

Base address of each segment is maintained by the segment registers.(FOLDER NAME) and Offset with the segment is maintained by the respective registers(FILE NAME)

CS +IP

SS+ SP / SS+BP

DS + SI

ES +DI

Can you tell me the page number of topic , Cache Memory in COA book??

You cannot remember the content of a book by the page number
But You can remember by the chapter name and the sub headings

SEGMENT ADDRESS □ CHAPTER NAME
OFFSET □ SUB HEADINGS

How Processor knows the real address when we provide the segment name and the offset???

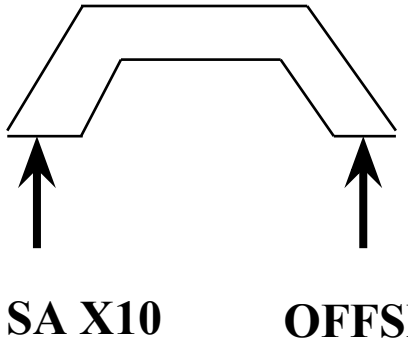
Physical Address = (Segment X 10) + Offset

For example :

CS = 1000H (16 bits)

IP = 1036H (16 bits)

PA= (1000 X10) + 1036 = 11036 (20 bits)



If Segment Address = 2000H

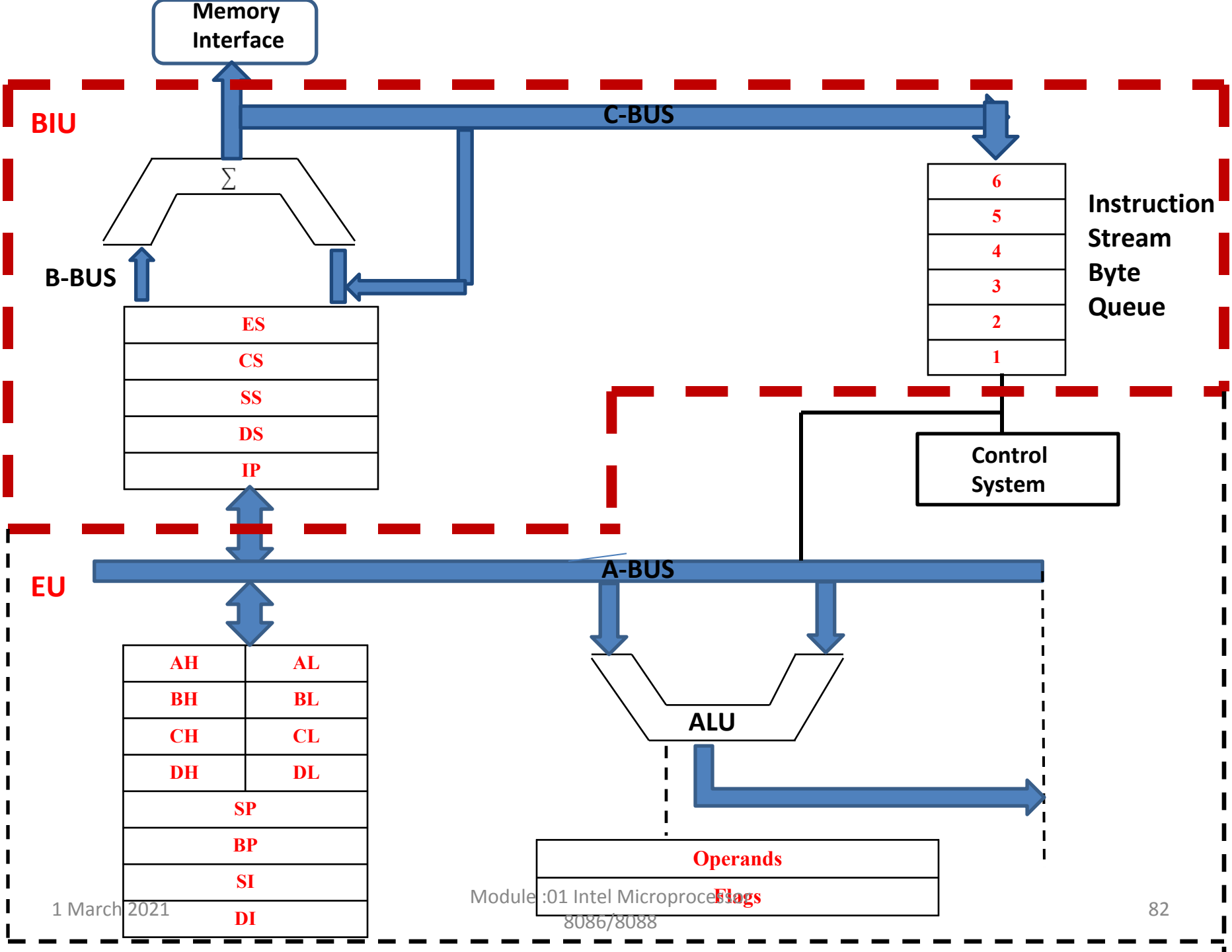
Offset Address = 4000H

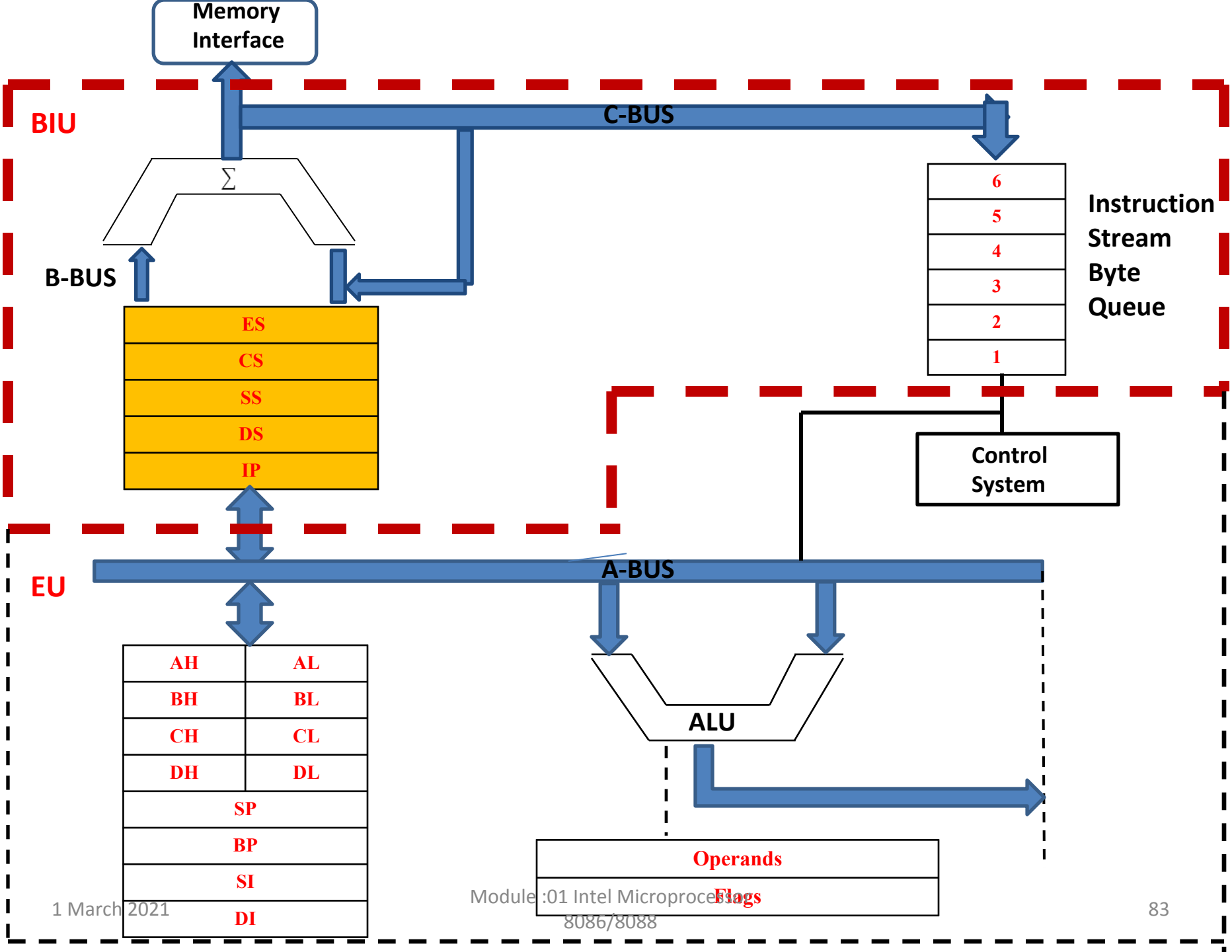
Then Calculate Physical Address??

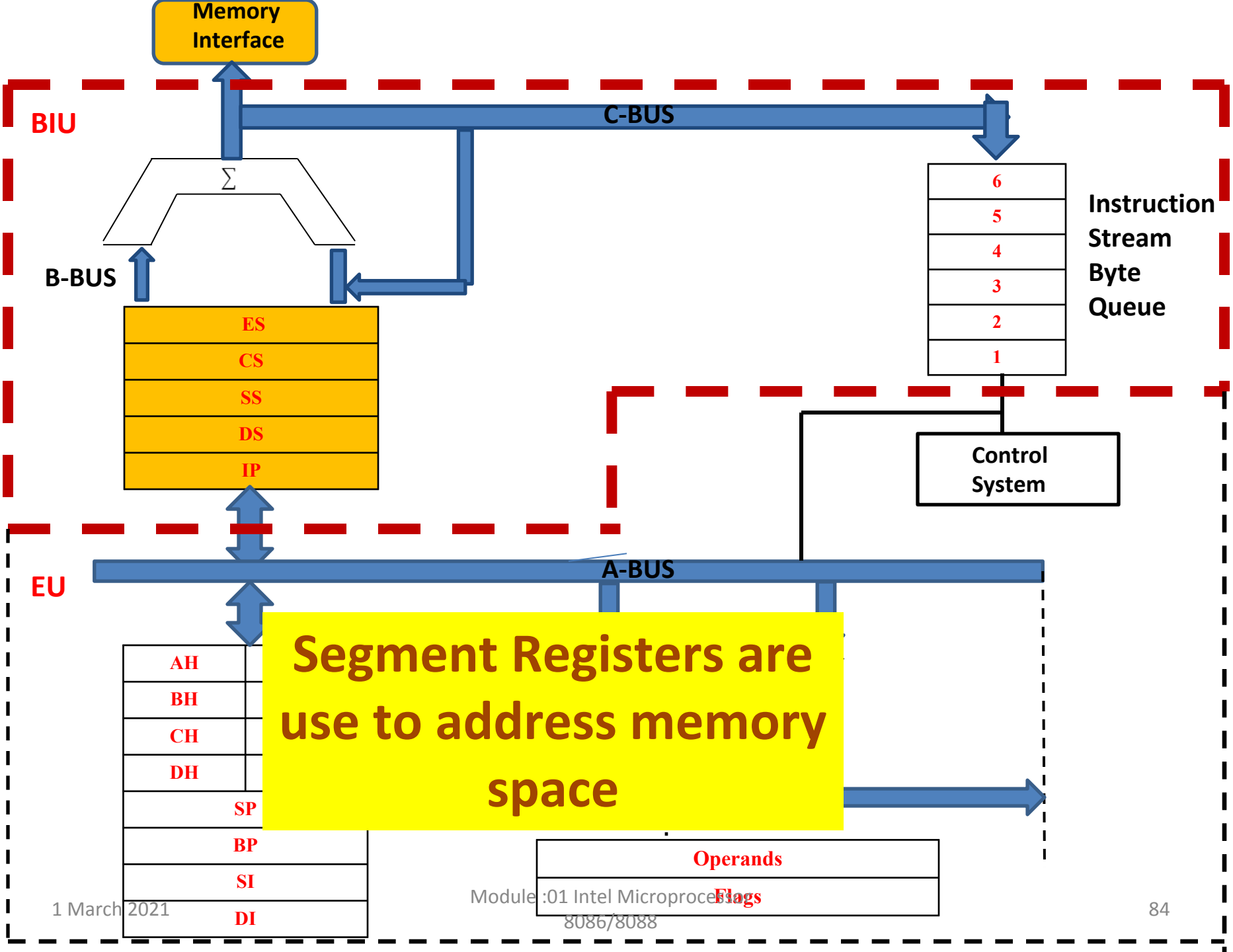
Physical Address =

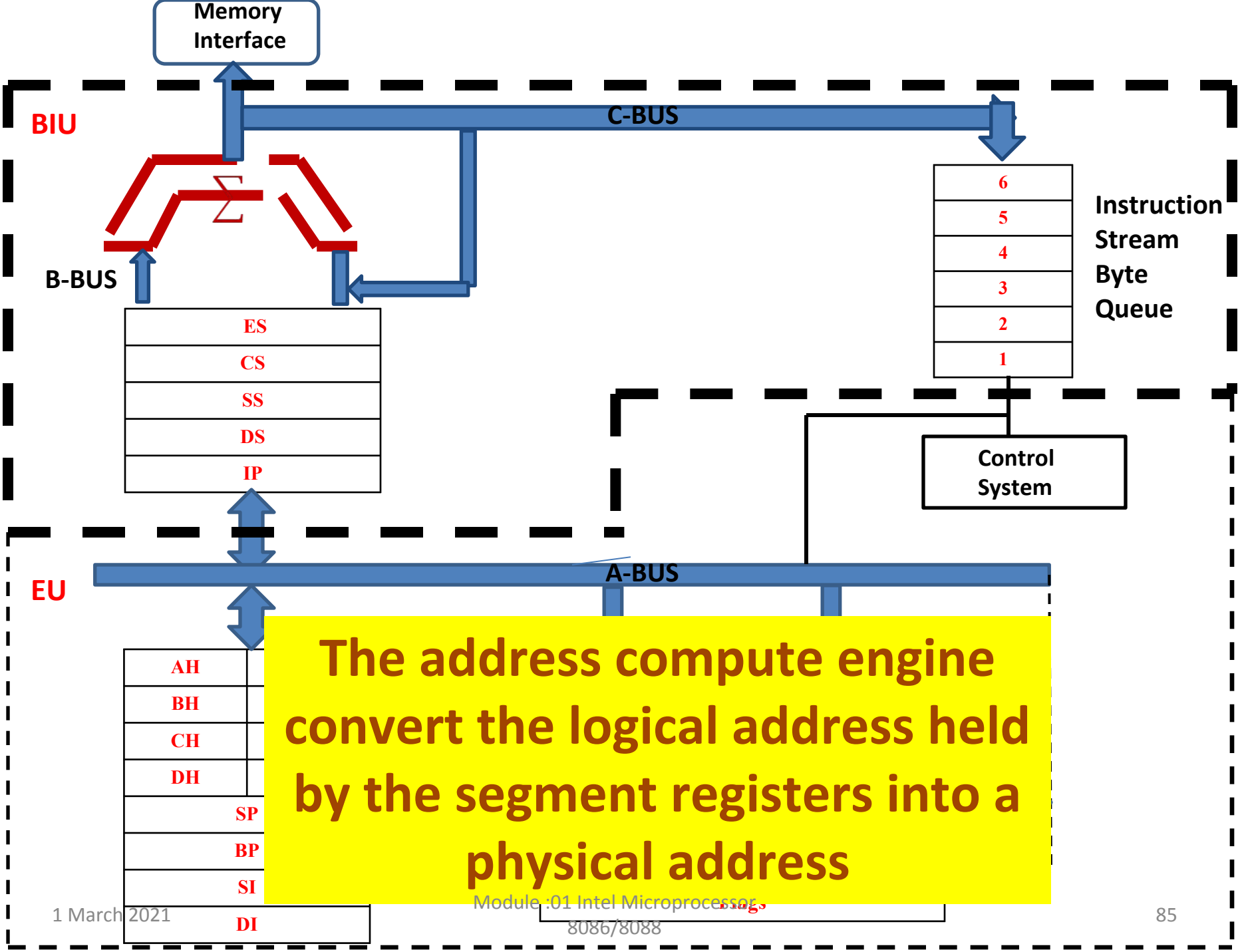
(2000 X 10) + 4000H = 24000

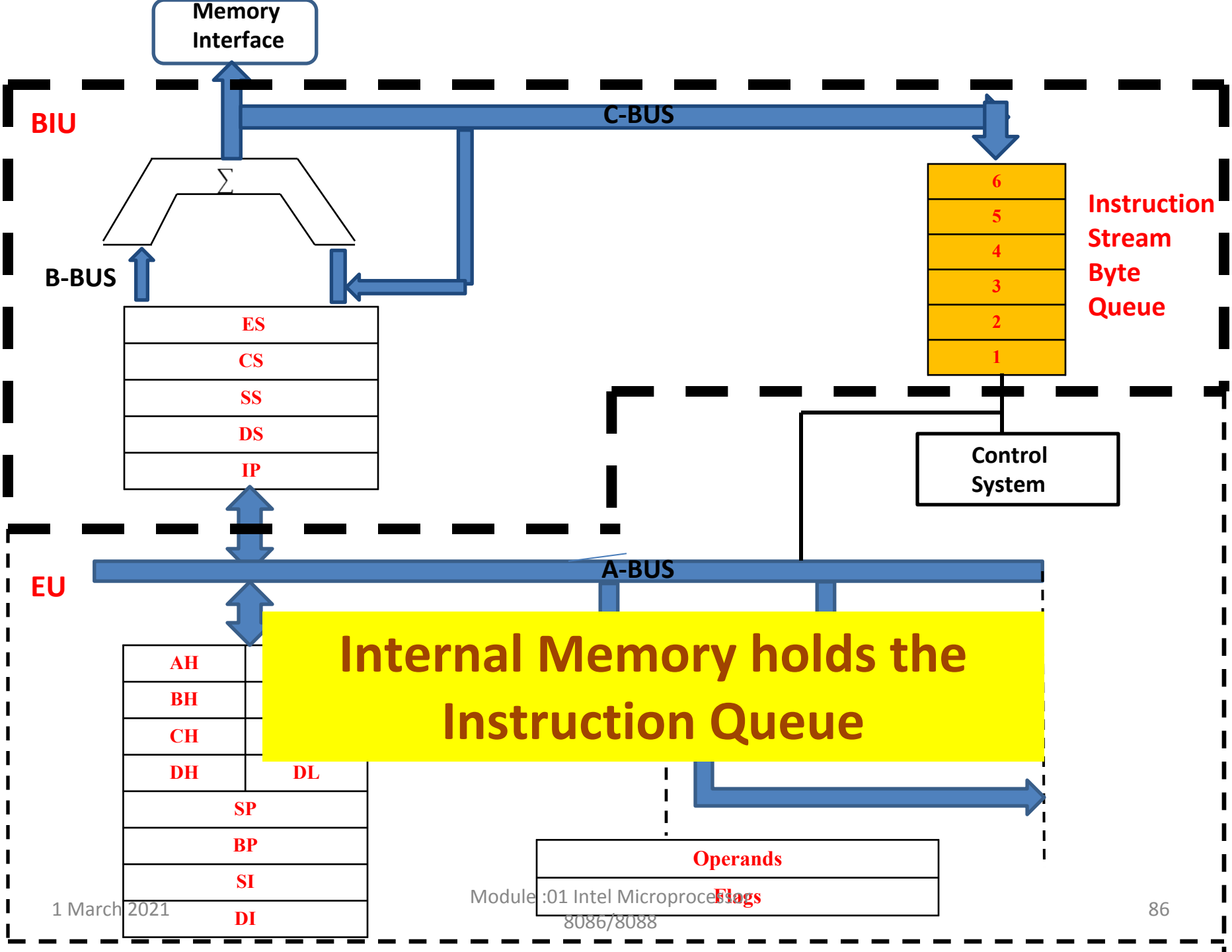
Note : After computing PA, instruction will come inside the processor through the data bus from the memory


Memory Interface
BIU
C-BUS
∑
B-BUS
6
5
4
3
2
1
Instruction Stream Byte Queue
ES
CS
SS
DS
IP
Control System
EU
A-BUS
AH
AL
BH
BL
CH
CL
DH
DL
SP
BP
SI
DI
ALU
Operands
Flags


Memory Interface
BIU
C-BUS
∑
B-BUS
6
5
4
3
2
1
Instruction Stream Byte Queue
ES
CS
SS
DS
IP
Control System
EU
A-BUS
AH
AL
BH
BL
CH
CL
DH
DL
SP
BP
SI
DI
ALU
Operands
Flags


Memory Interface
BIU
C-BUS
∑
B-BUS
ES
CS
SS
DS
IP
6
5
4
3
2
1
Instruction Stream Byte Queue
Control System
EU
A-BUS
AH
BH
CH
DH
SP
BP
SI
DI
Operands
Flags
Segment Registers are use to address memory space


Memory Interface
BIU
C-BUS
B-BUS
ES
CS
SS
DS
IP
6
5
4
3
2
1
Instruction Stream Byte Queue
Control System
EU
A-BUS
AH
BH
CH
DH
SP
BP
SI
DI
The address compute engine convert the logical address held by the segment registers into a physical address


Memory Interface
BIU
C-BUS
B-BUS
ES
CS
SS
DS
IP
6
5
4
3
2
1
Instruction Stream Byte Queue
Control System
EU
A-BUS
AH
BH
CH
DH
DL
SP
BP
SI
DI
Operands
Flags
Internal Memory holds the Instruction Queue

- BIU not only fetch the next instruction but it fetches 6 byte of the program and stores them in the queue.
- All instruction are not of same size.

6 instruction != 6 byte

In 8086, Any instruction can be at max of 6 bytes

Example

Eg:

MOV BL, CL Opcode → 1 Byte Instruction

Eg:

MOV BL, 25H

Opcode operand → 2 byte Instruction

(1byte) (1byte)

Eg:

MOV BX, 2000H

(1byte) (2byte) → 3byte Instruction

"Every Instruction has unique Opcode"

- All instruction are not of same size.
- Instruction may vary 1 byte, 2 byte, 3 byte, maximum up to 6 bytes.

When will BIU fill the instruction Queue????

1. Wait for the whole queue to become empty.
2. If 1 byte is empty then refill the queue
3. If 2 byte is empty then refill the queue.

Ans:

If 2 byte is empty then refill the queue.

Because 2byte= 16bit and 8086 is 16 bit procesor

So,

Architect of Processor builds a queue inside the processor, which permitted you to store in advance, specific instruction that would be executed in the future.

Whole concept is called **PIPELINING.**

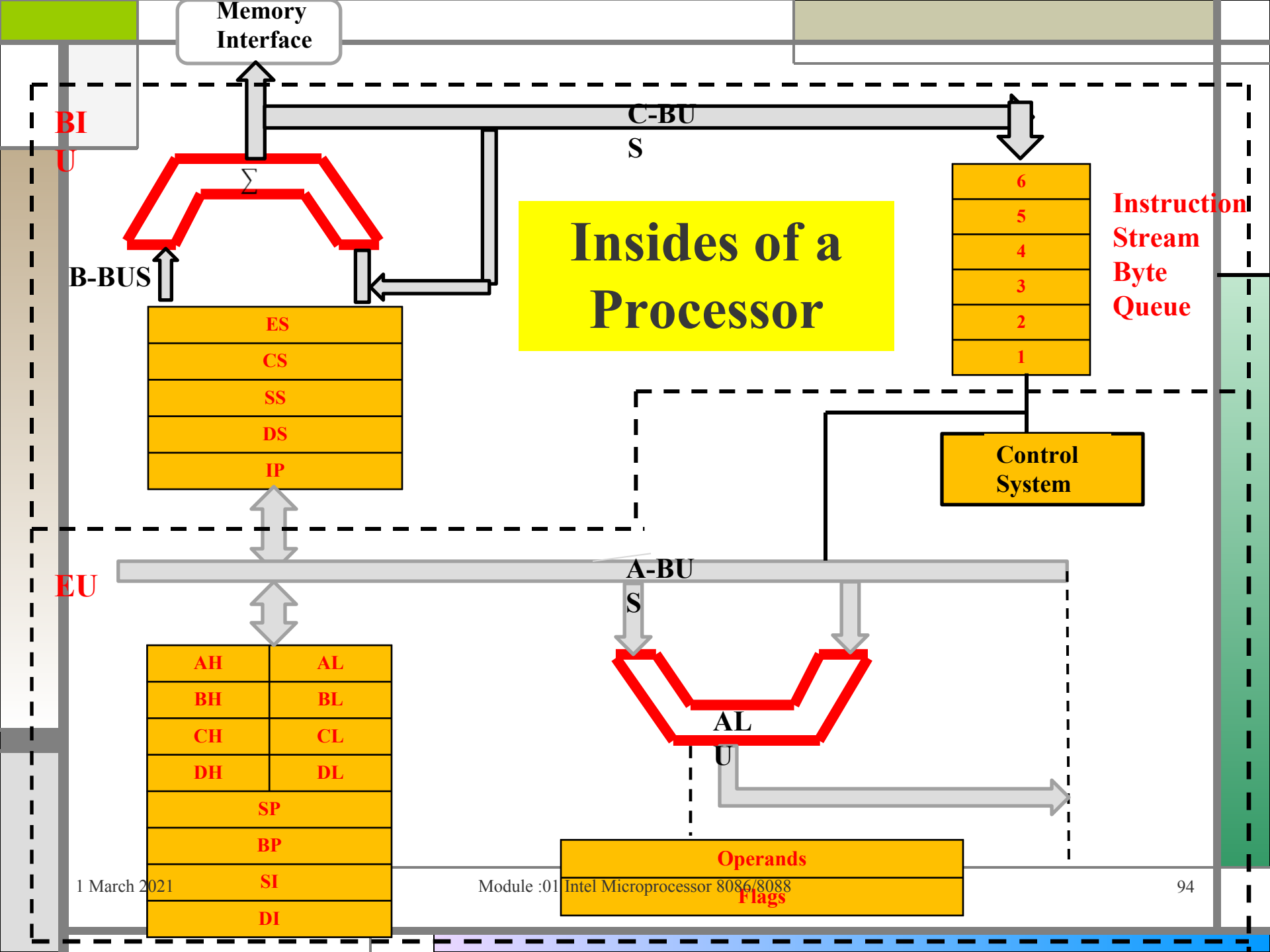
Pipelining is a technique wherein the Bus Interface Unit pre-fetches the instruction for the Execution Unit to use

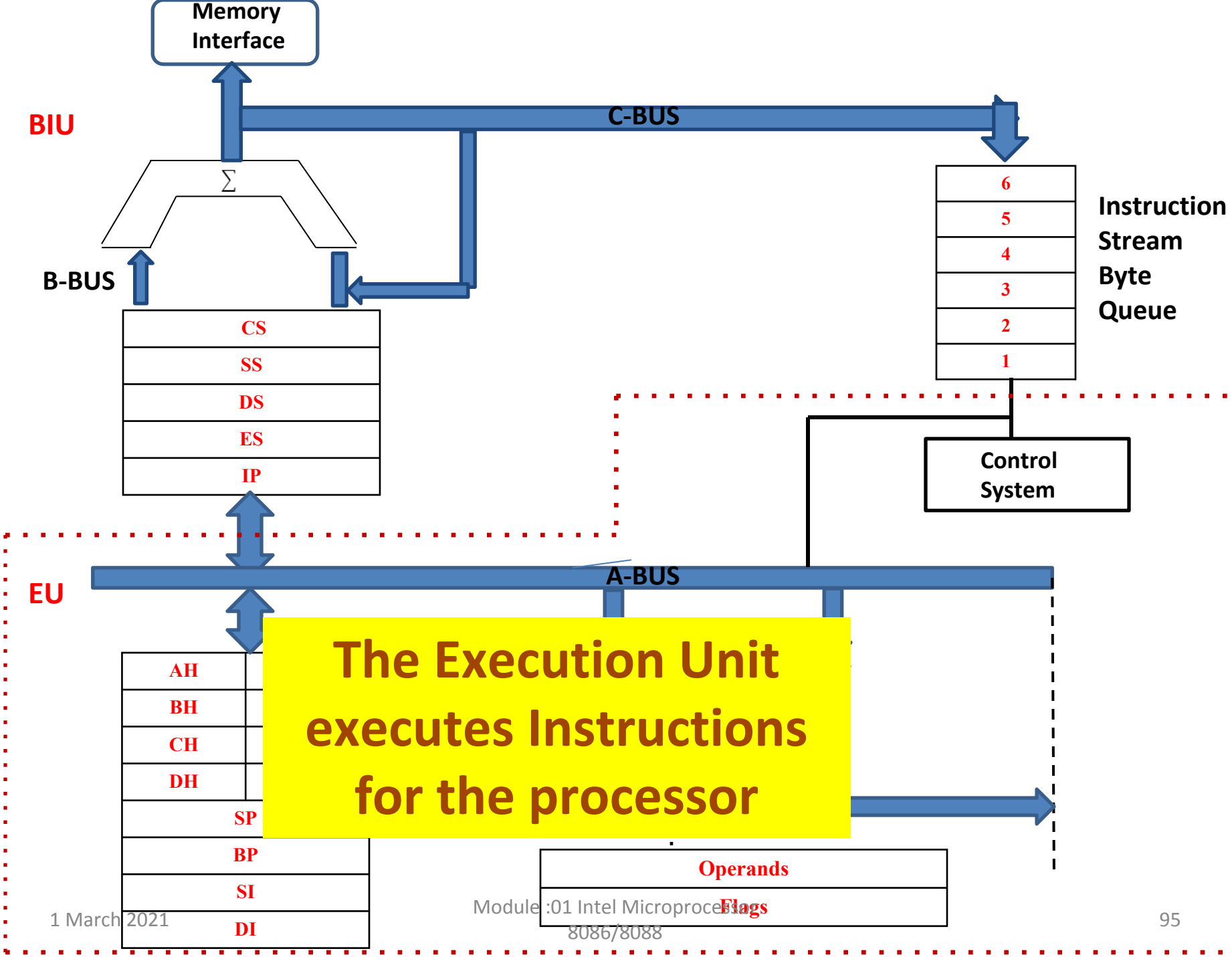
In case of 8086, up to six byte of instructions are fetched.

Note: if there is any JUMP instruction PIPE has to refreshed.

Bus Interface Unit

- Handles all the data and the addresses on the buses for the execution unit.
- Bus operations include instruction fetching, reading and writing operations for memory and calculating the addresses of the memory operands.
- The instructions are transferred to the instruction queue


Memory Interface
BIU
C-BUS
∑
B-BUS
ES
CS
SS
DS
IP
Insides of a Processor
6
5
4
3
2
1
Instruction Stream Byte Queue
Control System
EU
A-BUS
AH
AL
BH
BL
CH
CL
DH
DL
SP
BP
SI
DI
ALU
Operands
Flags


Memory Interface
BIU
C-BUS
∑
B-BUS
6
5
4
3
2
1
Instruction Stream Byte Queue
CS
SS
DS
ES
IP
Control System
EU
A-BUS
AH
BH
CH
DH
SP
BP
SI
DI
Operands
Flags
The Execution Unit executes Instructions for the processor

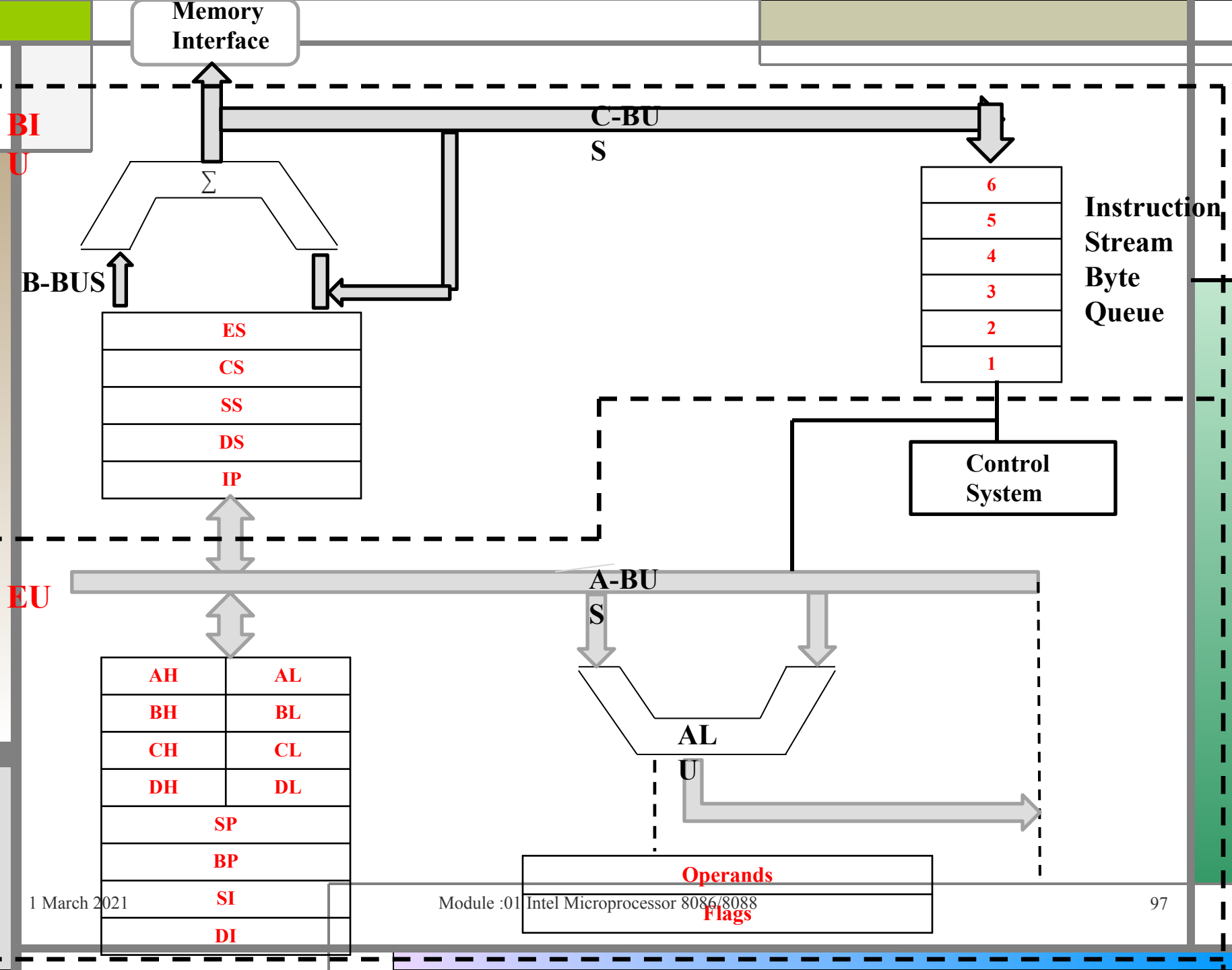
Execution Unit

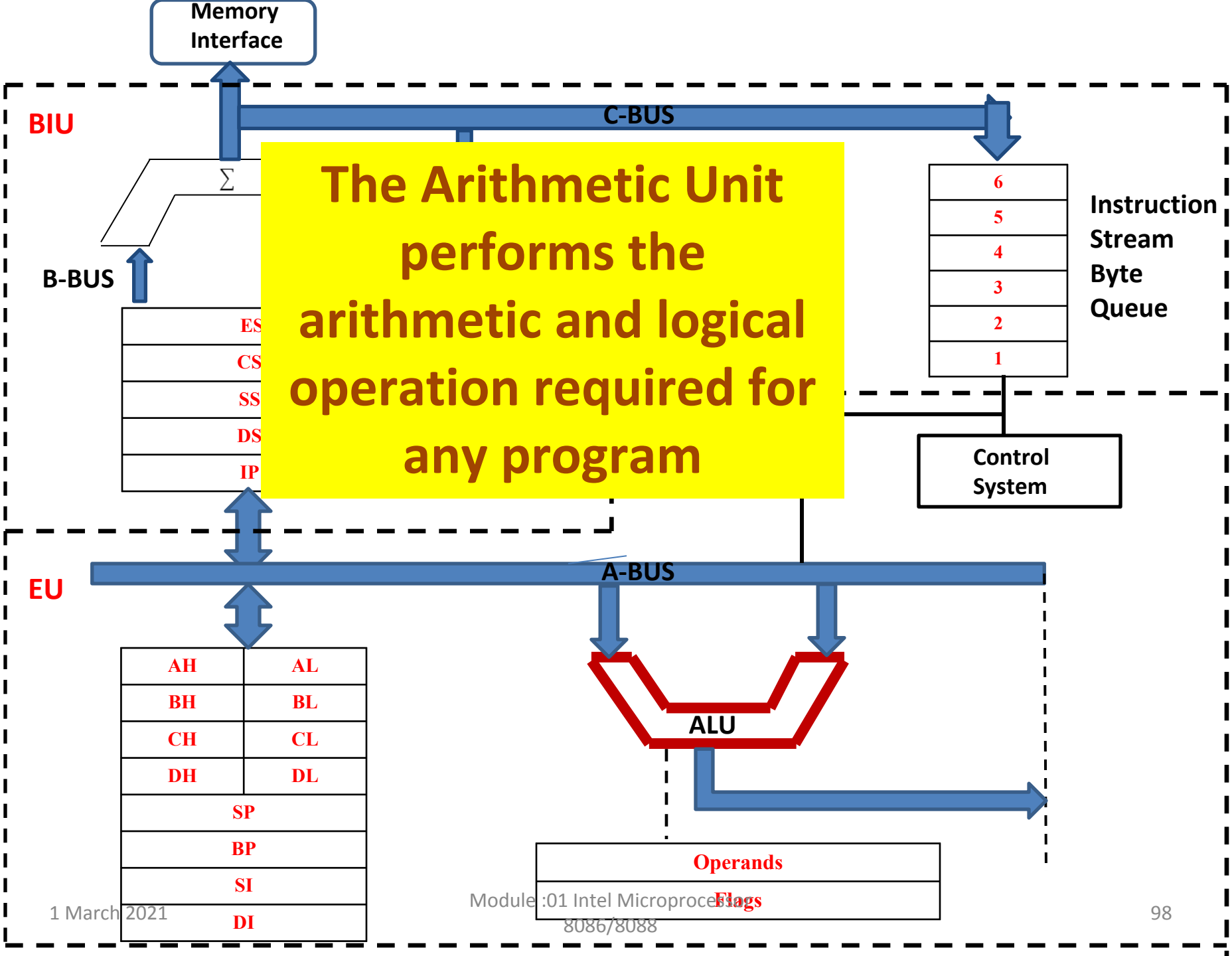
Control Section decodes the instruction and releases the control signals to every object of an architecture

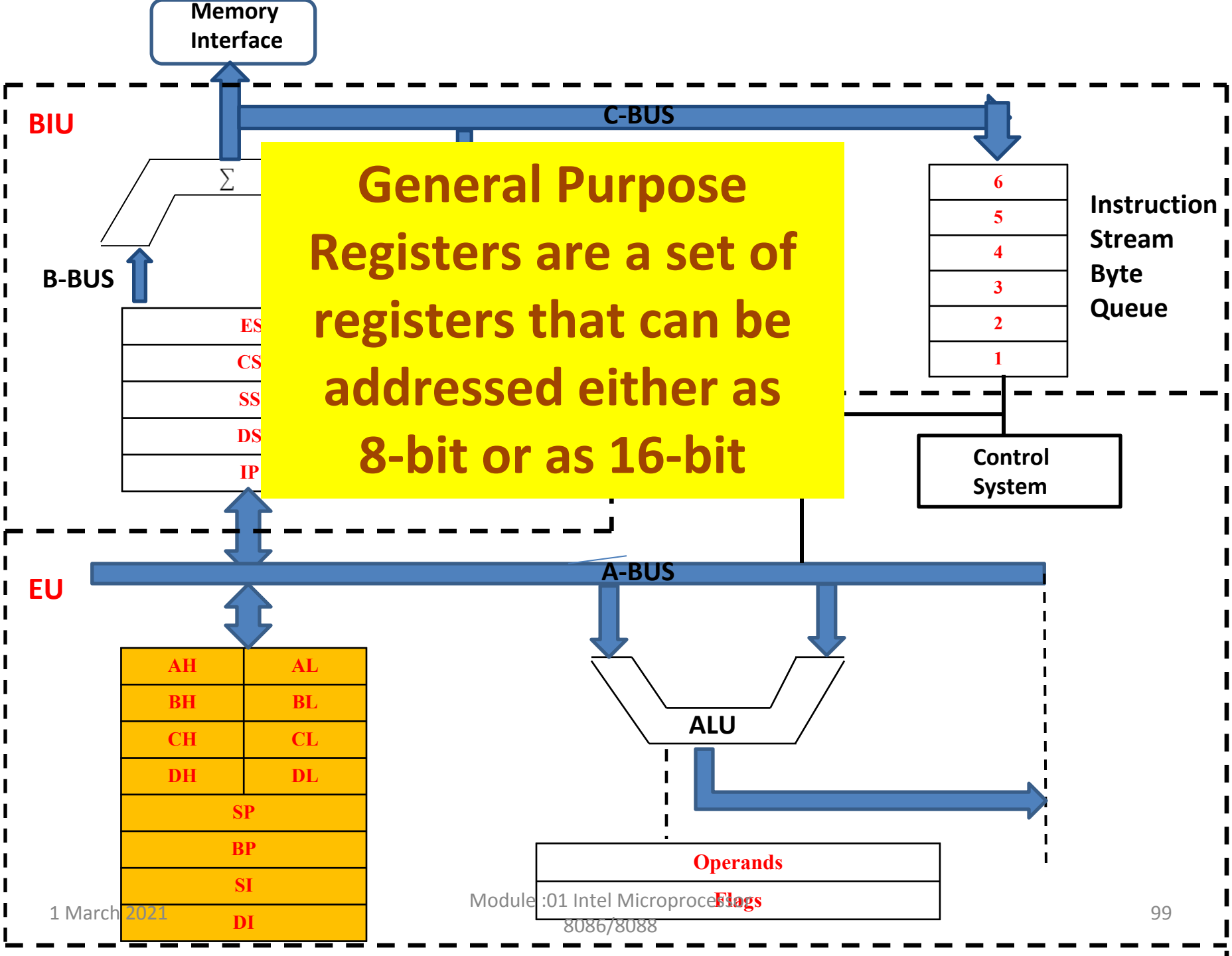
Example:

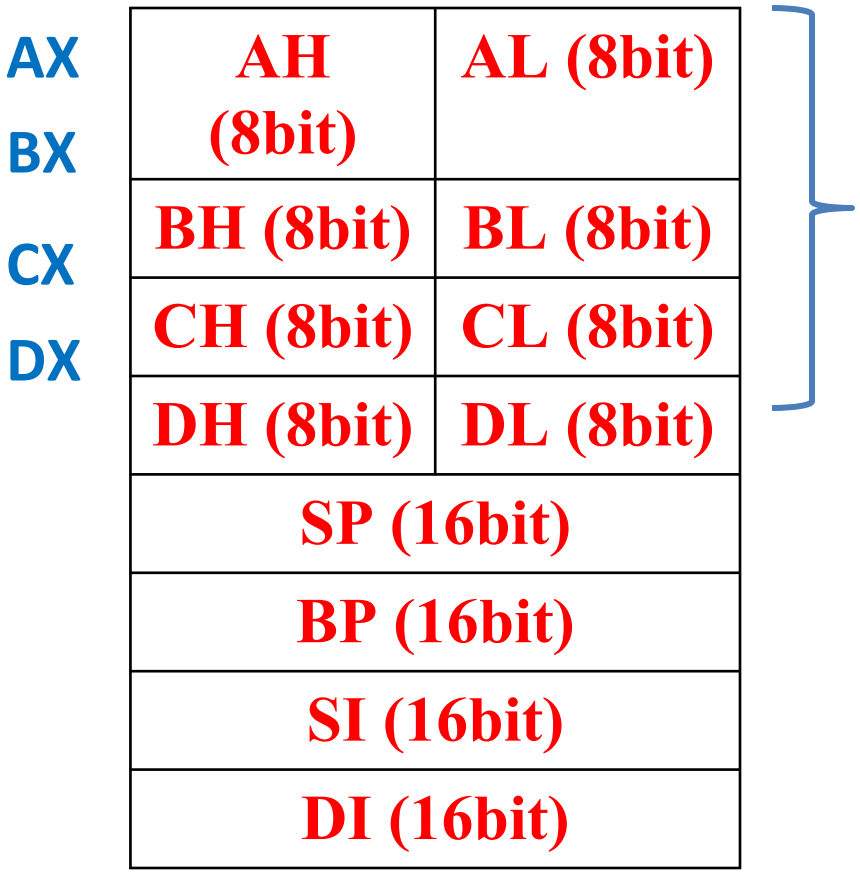
ADD BL, CL

(00110101010010….)


Memory Interface
BIU
C-BUS
Σ
B-BUS
ES
CS
SS
DS
IP
6
5
4
3
2
1
Instruction Stream Byte Queue
Control System
EU
A-BUS
AH
AL
BH
BL
CH
CL
DH
DL
SP
BP
SI
DI
AL
U
Operands
Flags


Memory Interface
BIU
C-BUS
The Arithmetic Unit performs the arithmetic and logical operation required for any program
B-BUS
ES
CS
SS
DS
IP
6
5
4
3
2
1
Instruction Stream Byte Queue
Control System
EU
A-BUS
AH
AL
BH
BL
CH
CL
DH
DL
SP
BP
SI
DI
ALU
Operands
Flags


Memory Interface
BIU
C-BUS
B-BUS
ES
CS
SS
DS
IP
6
5
4
3
2
1
Instruction Stream Byte Queue
Control System
General Purpose Registers are a set of registers that can be addressed either as 8-bit or as 16-bit
EU
A-BUS
AH
AL
BH
BL
CH
CL
DH
DL
SP
BP
SI
DI
ALU
Operands
Flags



General Purpose Register

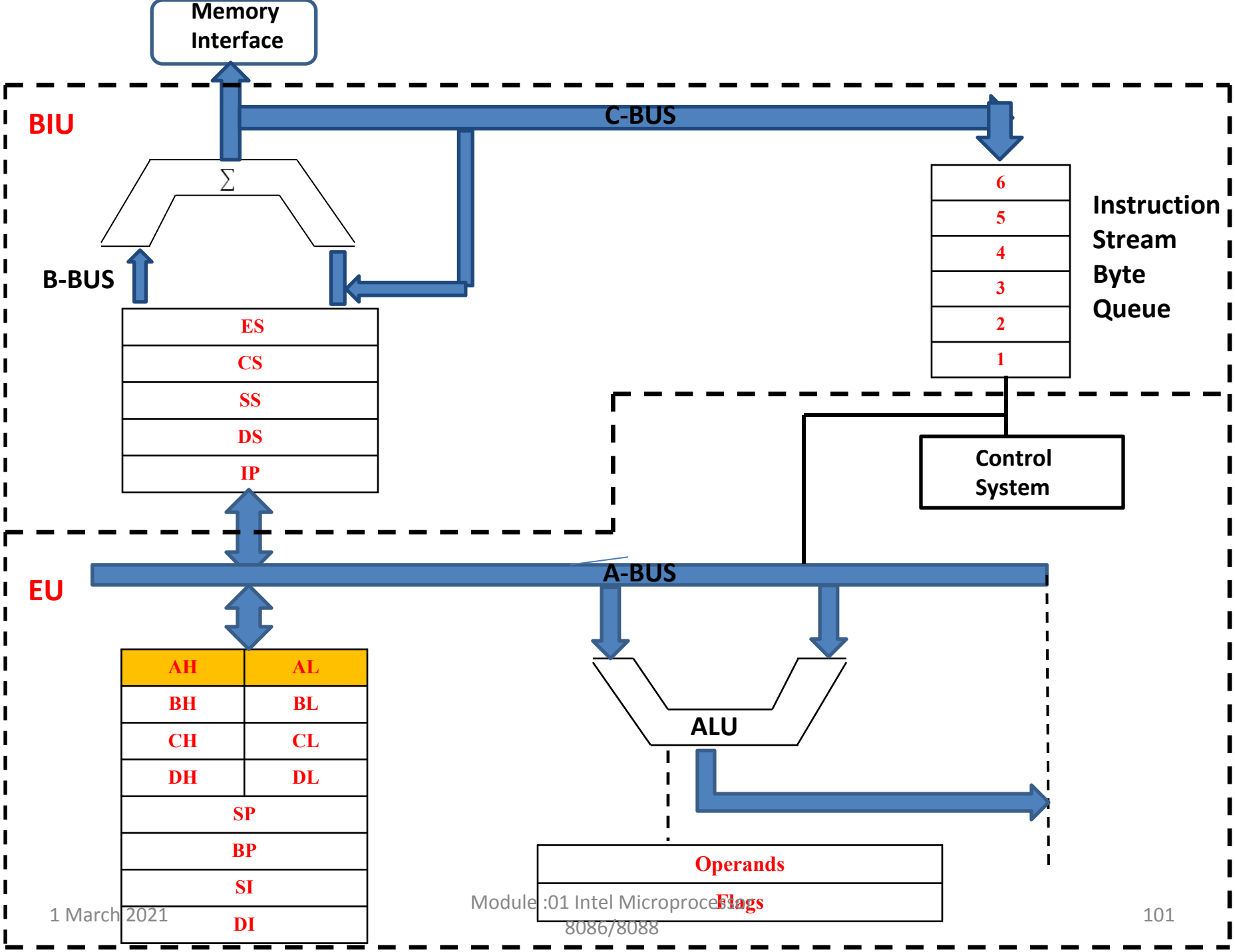
Though they are general purpose register they have some implicit purposes

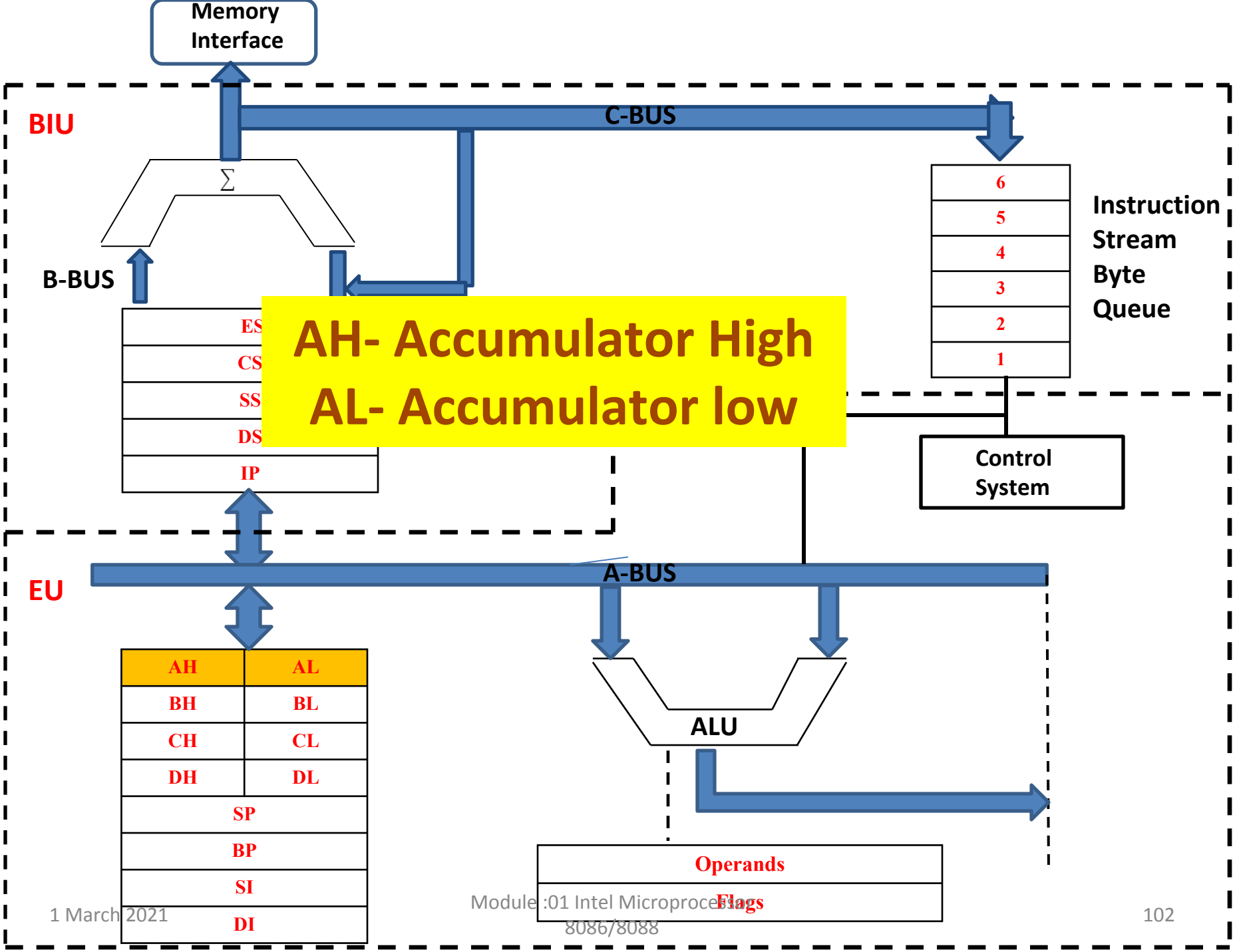
A: accumulator

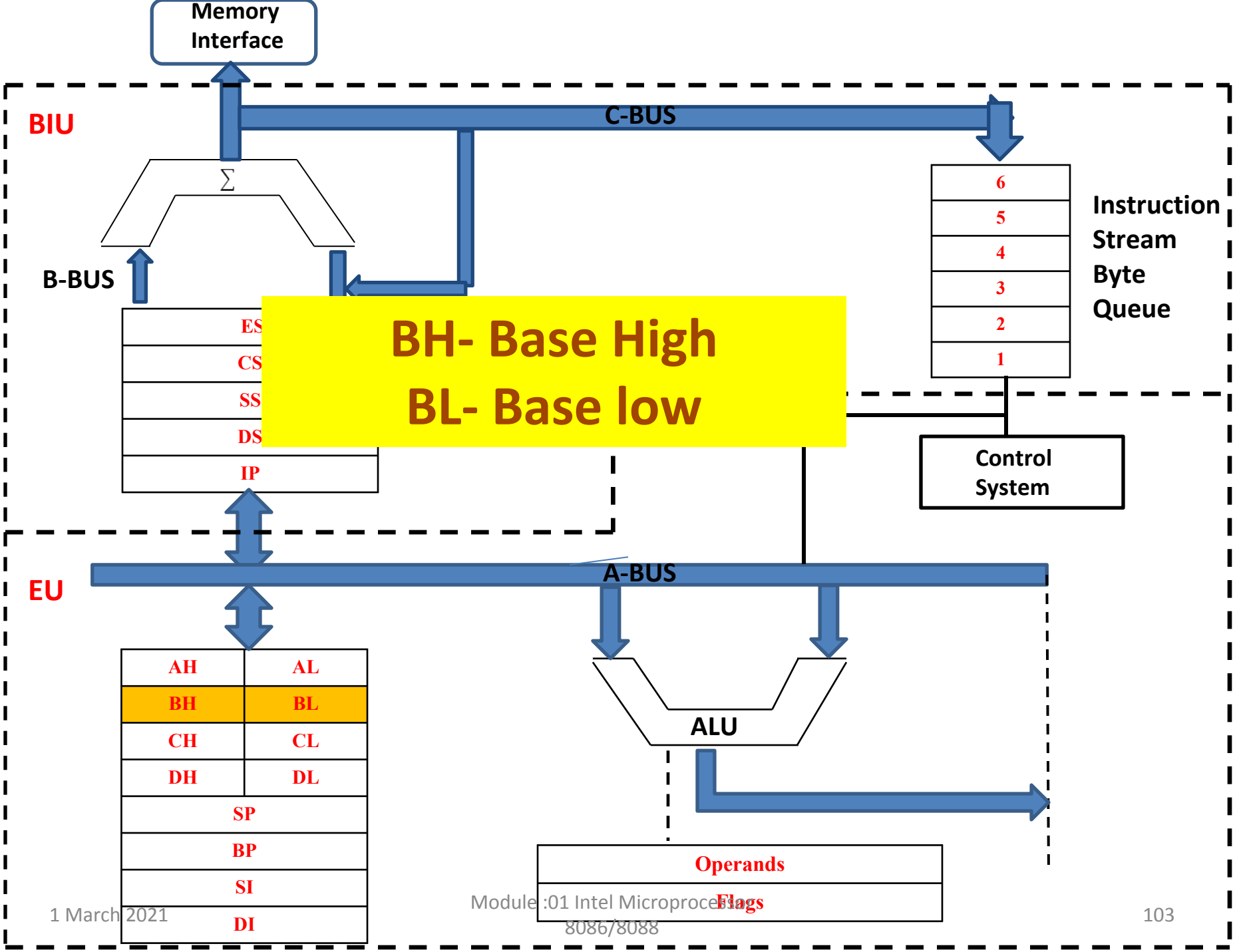
B: Base

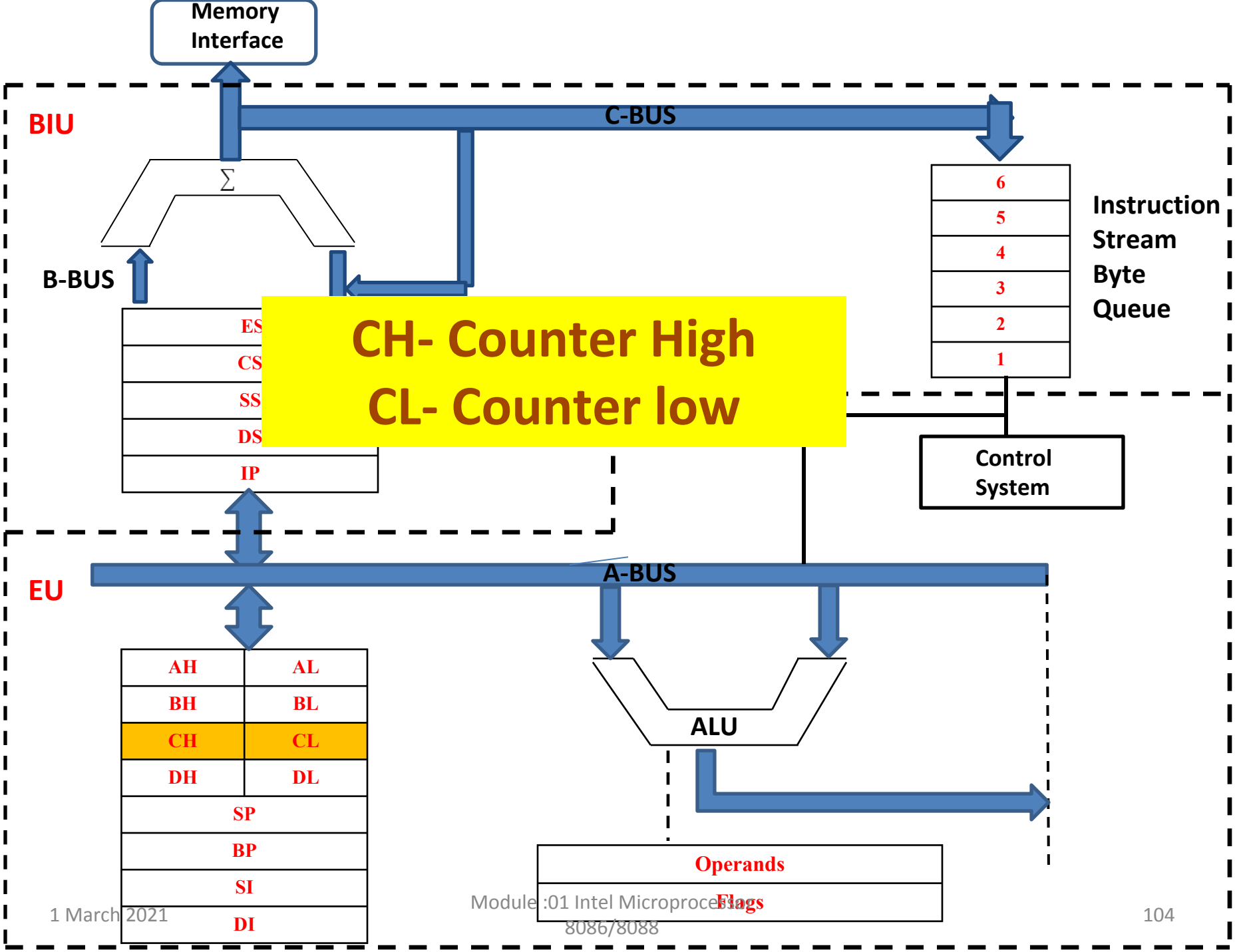
C: Count

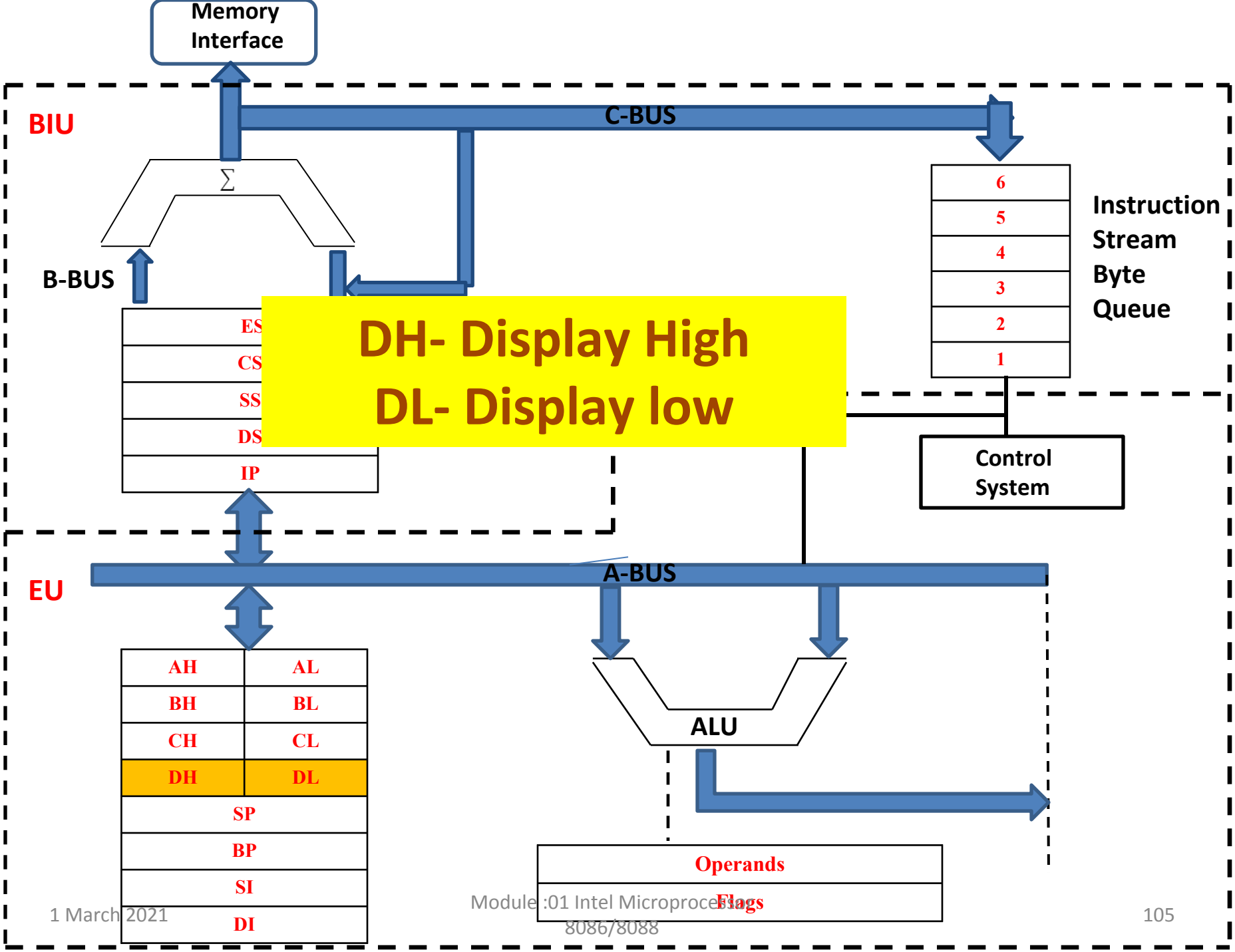
D: Display

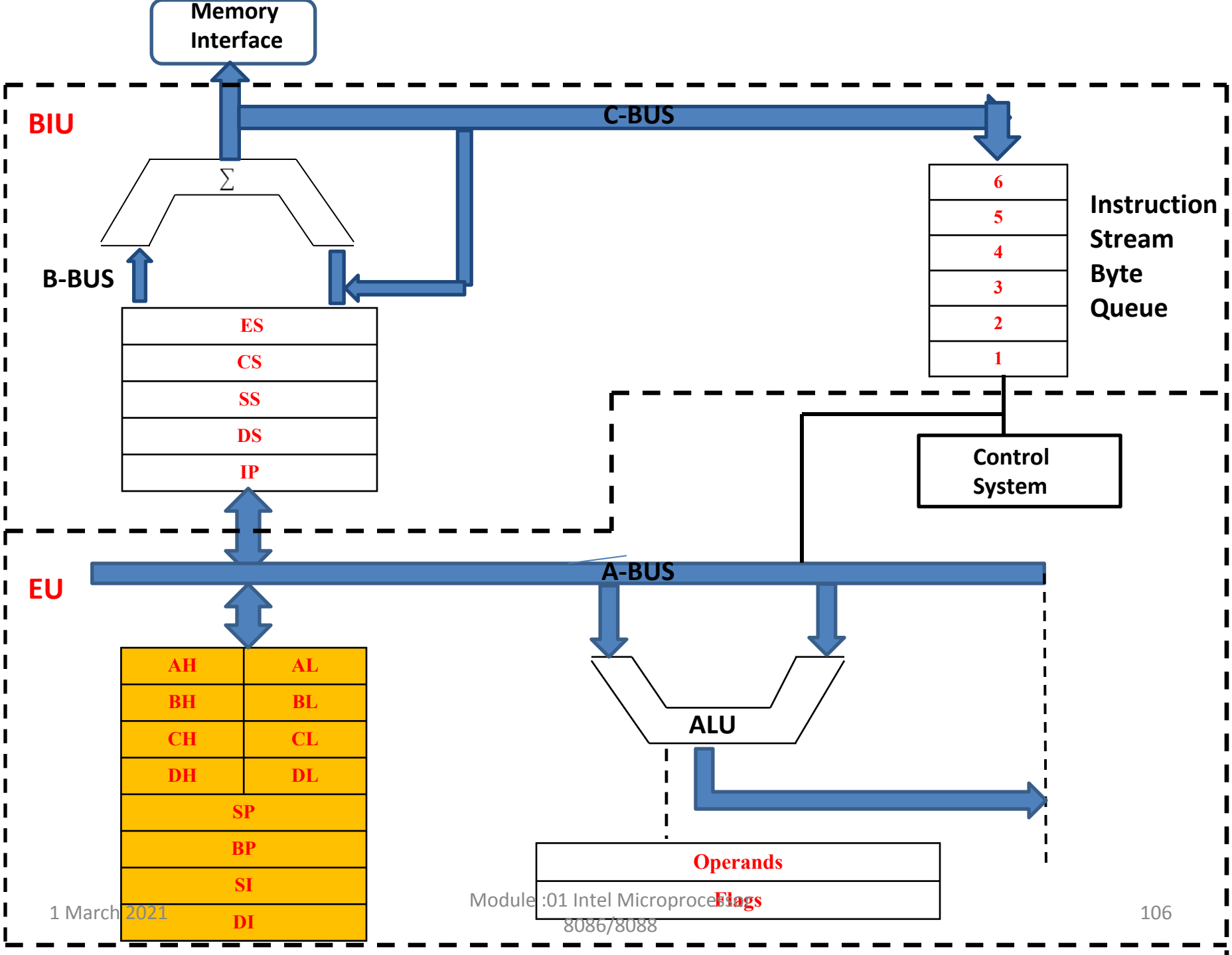

Memory Interface
BIU
C-BUS
∑
B-BUS
ES
CS
SS
DS
IP
6
5
4
3
2
1
Instruction Stream Byte Queue
Control System
EU
A-BUS
AH
AL
BH
BL
CH
CL
DH
DL
SP
BP
SI
DI
ALU
Operands
Flags


Memory Interface
BIU
C-BUS
∑
B-BUS
6
5
4
3
2
1
Instruction Stream Byte Queue
ES
CS
SS
DS
IP
AH- Accumulator High
AL- Accumulator low
Control System
EU
A-BUS
AH
AL
BH
BL
CH
CL
DH
DL
SP
BP
SI
DI
ALU
Operands
Flags


Memory Interface
BIU
C-BUS
∑
B-BUS
6
5
4
3
2
1
Instruction Stream Byte Queue
ES
CS
SS
DS
IP
BH- Base High
BL- Base low
Control System
EU
A-BUS
AH
AL
BH
BL
CH
CL
DH
DL
SP
BP
SI
DI
ALU
Operands
Flags


Memory Interface
BIU
C-BUS
B-BUS
6
5
4
3
2
1
Instruction Stream Byte Queue
ES
CS
SS
DS
IP
CH- Counter High
CL- Counter low
Control System
EU
A-BUS
AH
AL
BH
BL
CH
CL
DH
DL
SP
BP
SI
DI
ALU
Operands
Flags


Memory Interface
BIU
C-BUS
∑
B-BUS
6
5
4
3
2
1
Instruction Stream Byte Queue
ES
CS
SS
DS
IP
DH- Display High
DL- Display low
Control System
EU
A-BUS
AH
AL
BH
BL
CH
CL
DH
DL
SP
BP
SI
DI
ALU
Operands
Flags


Memory Interface
BIU
C-BUS
∑
B-BUS
ES
CS
SS
DS
IP
6
5
4
3
2
1
Instruction Stream Byte Queue
Control System
EU
A-BUS
AH
AL
BH
BL
CH
CL
DH
DL
SP
BP
SI
DI
ALU
Operands
Flags

Operand Register (16-bit)

It is a temporary register used by the programmer.

Example : XCHG BX, CX

For swapping we need temporary register

Flags (16 bit)

Contains the status of the current result produced by ALU

Eg: sign flag , carry flag, sign flag etc.

It access to the outside world either through the data bus, address bus or I/O Ports

Why is this required????

Execution Unit of a processor is much faster than the peripheral devices like memory (ROM or RAM). Earlier days, Memory was almost as fast as Processor unit but when 8086 come in picture, Processor became much faster than the peripheral devices

Questions

1.Draw and explain the architecture of 8086.

2.Explain pipelining in 8086.

Memory Banking

- 8086 uses 16 bit data bus i.e., it should be able to access 16 bit data in one cycle.
- And so it needs to read from 2 memory location as one memory location carries 1 Byte data.
- 16 bit data is stored in 2 consecutive location.
- However, if both these memory locations are in same memory chip then they cannot be accessed at the same time, as the address bus of the chip cannot contain two addresses simultaneously.
- **Hence, the memory of 8086 is divided into two banks, each bank provide 8 bits**.

Even Bank and Odd Bank

- Memory is divided in such a manner any two consecutive location lie in two different chips. Hence each chip contain an alternate location.
- One bank contains all even addresses called the **EVEN BANK** while the other containing odd addresses is called the **ODD BANK**.
- Generally for any 16 bit operation, even bank provides the lower byte and the odd bank provides higher byte. Due to which, **Even Bank is also known as Lower Bank and Odd Bank is also known as Higher Bank**

1 MB

512 KB

ODD Bank

- Also Called as “Higher Bank”
- Address range:

  00001H
  00003H
  00005H
  .
  .
  FFFFFH
- Selected when $\overline{BHE} = 0$

512 KB

EVEN Bank

- Also Called as “Lower Bank”
- Address range:

  00002H
  00004H
  00006H
  .
  .
  FFFFEH
- Selected when $A_0 = 0$

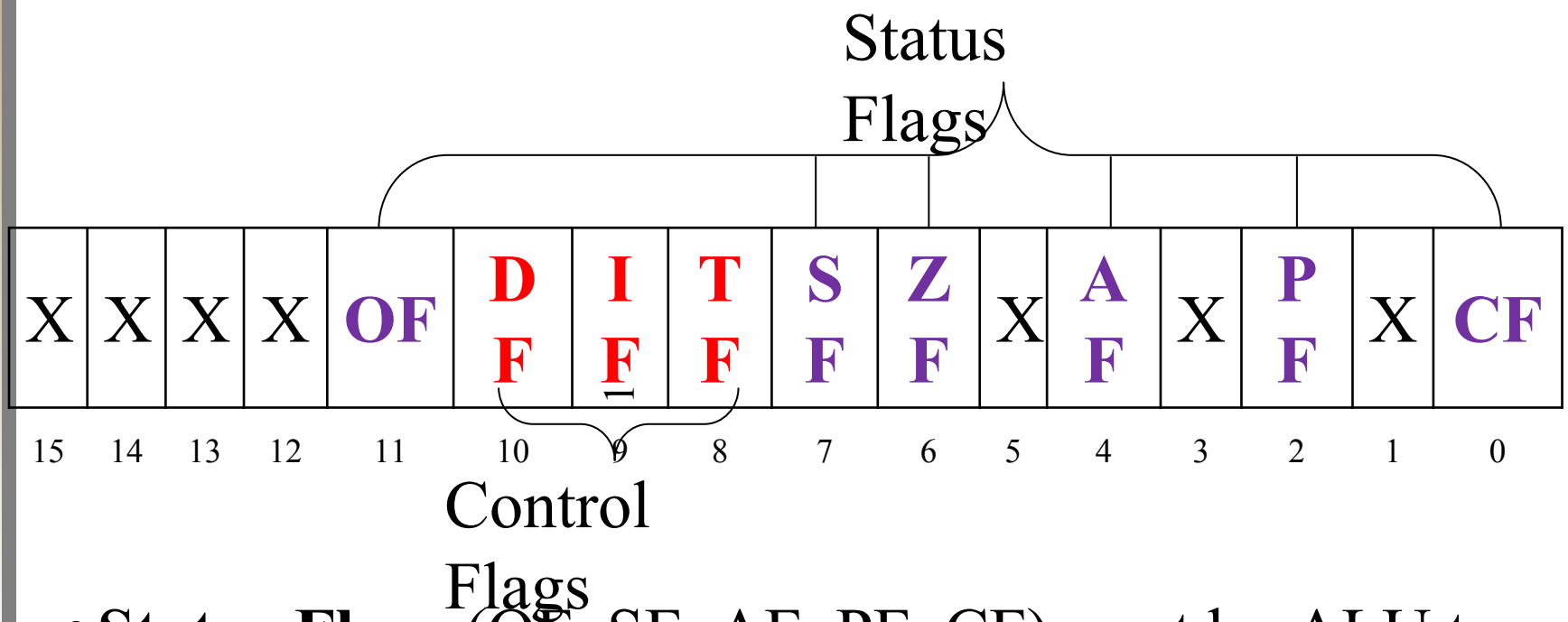
$\overline{BHE} = 0$	A_0	OPERATIONS
0	0	R/W 16 bit from both the banks
0	1	R/W 8 bit from Higher Bank
1	0	R/W 8 bit from Lower Bank
1	1	No Operation

Memory Banking is a pillar to understand memory Designing

8086 Flag Register

- 16-bit register
- Has 16 flip flops
- Each flip flop can be set, reset and viewed individually.
- Shows the status of an operation
- These flip flops are set/reset by the processor after execution of each instruction.
- Used for conditional branching (we will see later)

The Flag Bits



- **Status Flags** (OF, SF, AF, PF, CF) – set by ALU to show the status of execution of an instruction.
- **Control Flags** (DF, IF, TF) – set by the programmer to control an execution.

CF = Carry Flag

- Status Flag (set by ALU)
- Gets set when there is a CARRY out of the MSB

Example

MOV AX, 00FFh	0000 0000 1111 1111	AX
MOV BX, 0001h	0000 0000 0000 0001	BX
ADD AX, BX	0000 0001 0000 0000	AX
CF = 0		
ADD AL, BL	0000 0001 0000 0000	AX
CF = 1		

CF = 0


mov ax,00FF
mov bl,01
add ax,bx
add [bx+si],al
add [bx+si],al
add [bx+si],al
add [bx+si],al
add [bx+si],al
add [bx+si],al
ax 0100
bx 0001
cx 0000
dx 0000
si 0000
di 0000
bp 0000
sp 0000
ds 489D
c=0
z=0
s=0
o=0
p=1
a=1
i=1
d=0

CF = 1


mov ax,00FF
mov bl,01
add al,bl
add [bx+si],al
add [bx+si],al
add [bx+si],al
add [bx+si],al
add [bx+si],al
add [bx+si],al
ax 0000
bx 0001
cx 0000
dx 0000
si 0000
di 0000
bp 0000
sp 0000
ds 489D
c=1
z=1
s=0
o=0
p=1
a=1
i=1
d=0

PF = PARITY FLAG

- Status Flag (set by ALU)
- Gets set when the result has even number of 1s

Example:

MOV AX, 0006h

AX = 0006 (0000 0000 0000 0110)

PF = 1

ADD AX, 0002h

AX = 0008 (0000 0000 0000 1000)

PF = 0

Note: 0 is an even number.

AF = Auxiliary Flag

- Status Flag (set by ALU)
- Carry from lower nibble to higher nibble
- 1 nibble = 4 bits

Example:

MOV AX, 0007h 0000 0000 0000 0111
MOV BX, 0001h 0000 0000 0000 0001
ADD AX, BX 0000 0000 0000 1000

AF = 0

Auxiliary Flag...

MOV AX, 000Fh 0000 0000 0000 1111

MOV BX, 0001h 0000 0000 0000 0001

ADD AX, BX 0000 0000 000**1** 0000

AF = 1

ZF = Zero Flag

- Status Flag (set by ALU)
- If the result of an operation produces 0, ZF = 1

Example:

MOV AX, 0070h 0000 0000 0111 0000

XOR AX, AX 0000 0000 0111 0000

0000 0000 0000 0000

ZF = 1

SF = Sign Flag

- Is a Status Flag (set by ALU)
- Indicates if the MSB is 0 or 1.

If MSB = 1 □ SF = 1 □ Negative Number ?

If MSB = 0 □ SF = 0 □ Positive Number ?

MOV AX, 0006h	0000 0000 0000 0110	AX
MOV BX, 0008h	0000 0000 0000 1000	BX
SUB AX, BX	1111 1111 1111 1110	AX

SF = 1

Signed and Unsigned Numbers

- Unsigned Numbers are assumed to be positive
- 8-bits can have 0 ... 255 (d)
  or 00 ... FF(h)
  SF = 1 for 80 – FF , because MSB = 1
- Signed Numbers can be positive (MSB = 0)
  or negative (MSB = 1)
- 8-bits can have -128 ... -1 0 1 ... 127 (d)
  or - 80 ... -1 0 1 ... 7F (h)

Example

1000 0011 = +83 (unsigned)

= -7D (signed) – 2's C

OF = Overflow Flag

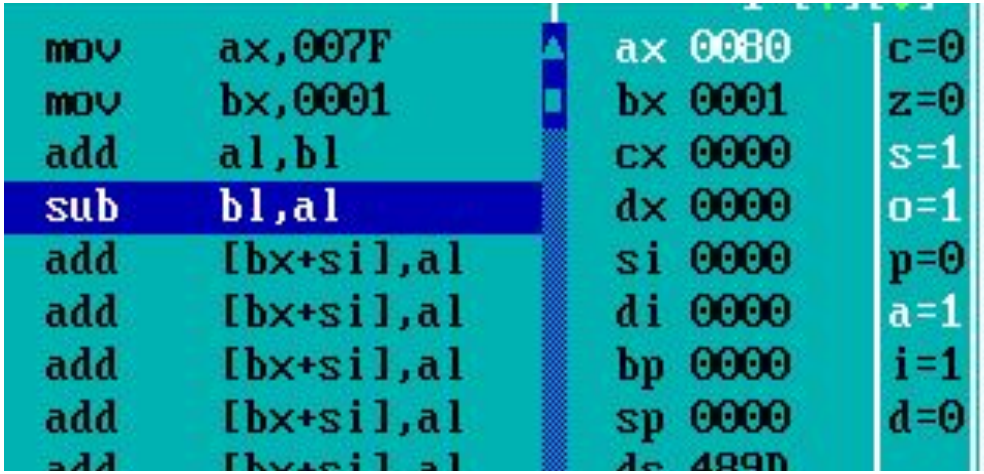
- Status Flag (set by ALU)
- If the result is out of range,
  then OF = 1
- In 8-bits 00 to 7F OF = 0

  0000 0000 0111 1111

  80 to FF OF = 1

  1000 0000 1111 1111
- A combination of SF and OF can say if the number is actually positive or negative. How?

Signed or Unsigned?

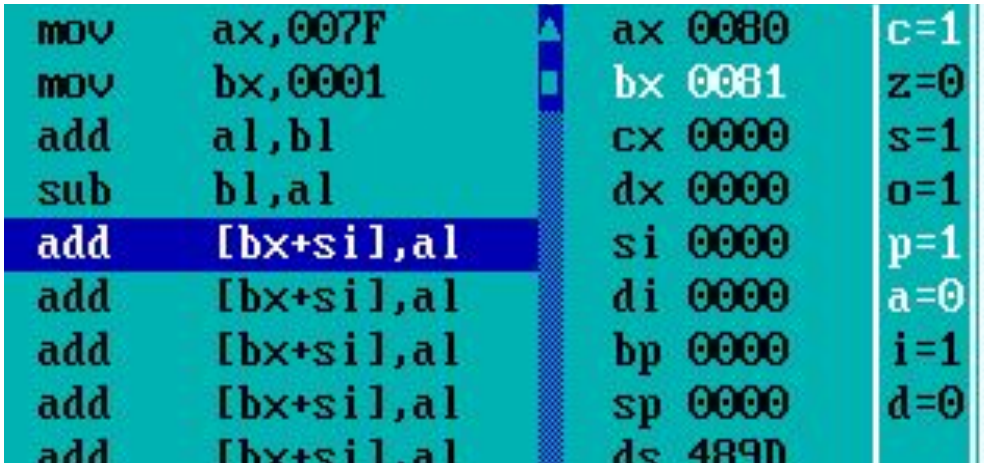


7F + 1 = +80

MSB =1 □ SF = 1

and OF = 1

=> SF is wrong



01 – 80 = -79

2's C of (-79) = 81

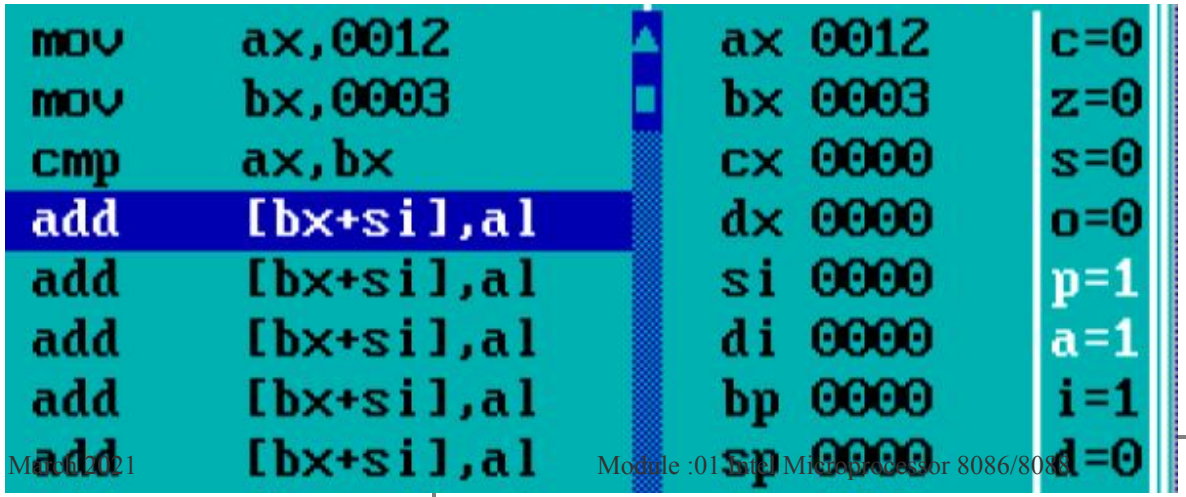
SF = 1

but OF = 0

=> SF is correct

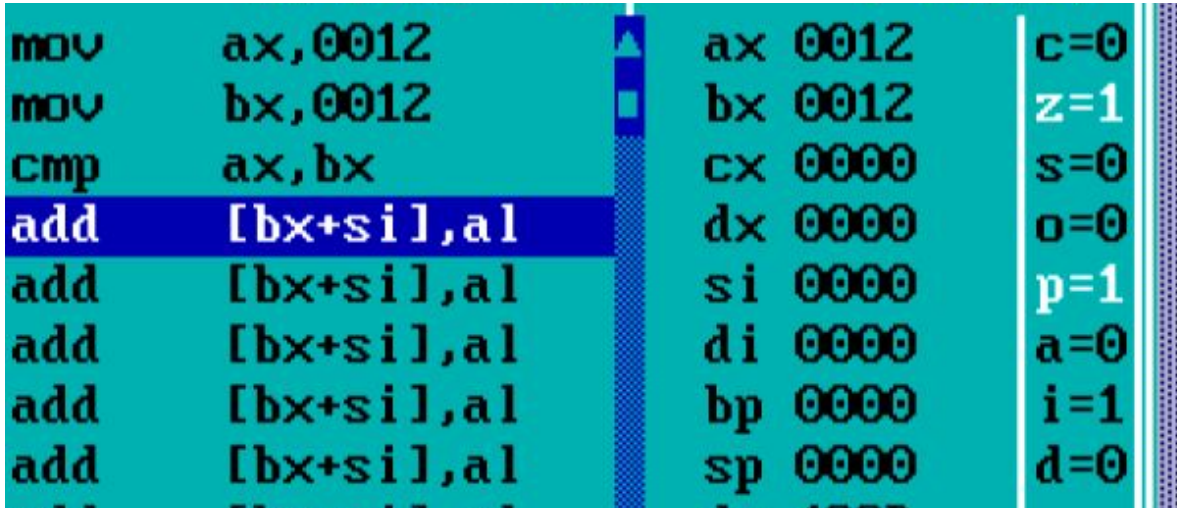
More combinations

If OF = 0
SF = 1
ZF = 0
AX < BX



If OF = 0
SF = 0
ZF = 0
AX > BX

More combinations...



If OF = 0

SF = 0

ZF = 1

AX = BX

Exercise 1

42+23

$$\begin{array}{rr} & {\scriptstyle 1} \\ & 0100\ 0010 \\ + & 0010\ 0011 \\ \hline & 0110\ 0101 \\ \hline \end{array}$$

				0				**0**	**0**		**0**		**1**		**0**
X	X	X	X	OF	DF	IF	TF	SF	ZF	X	AF	X	PF	X	CF

Exercise 2

37+29

```
    1 1 1  1 1 1
    0011 0111
+   0010 1001
-------------
    0110 0000
-------------
```

				0				**0**	**0**		**1**		**1**		**0**
X	X	X	X	OF	DF	IF	TF	SF	ZF	X	AF	X	PF	X	CF

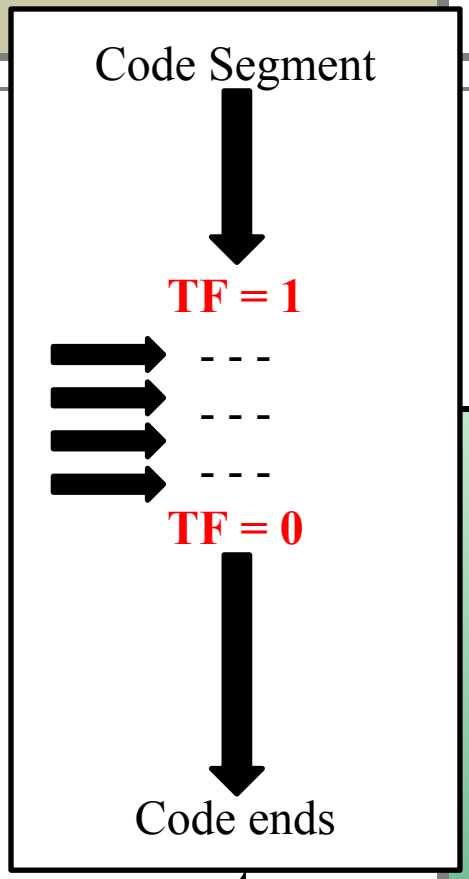
Exercise 3

42+43

$$\begin{array}{rr} & {\scriptstyle 1}\quad\quad{\scriptstyle 1} \\ & 0100\ 0010 \\ + & 0100\ 0011 \\ \hline & 1000\ 0101 \end{array}$$

				1				**1**	**0**		**0**		**0**		**0**
X	X	X	X	OF	DF	IF	TF	SF	ZF	X	AF	X	PF	X	CF

TF = Trap Flag



- Control Flag – set by programmer.
- TF = 1: Single Stepping Mode

Run entire code: F9

Single line execution: F7

- If only one block of code needs stepwise execution, set TF=1 before the block, and TF=0 at the end of the block, and then press F9.
- Only that block will be executed line-by-line.

IF = Interrupt Flag

- Control Flag – set by programmer.

  IF = 1 : allow interrupts – by default

  IF = 0 : disable interrupts

- If a critical block of code has to be executed without being interrupted, set IF = 0 at the start of the block and IF = 1 at the end of the block.

Example

- Hardware interrupts – key press, bus faults...
- Software interrupts – triggers...

DF = Direction Flag

- Control Flag – set by programmer.

  DF = 0 : auto increment (default)

  DF = 1 : auto decrement

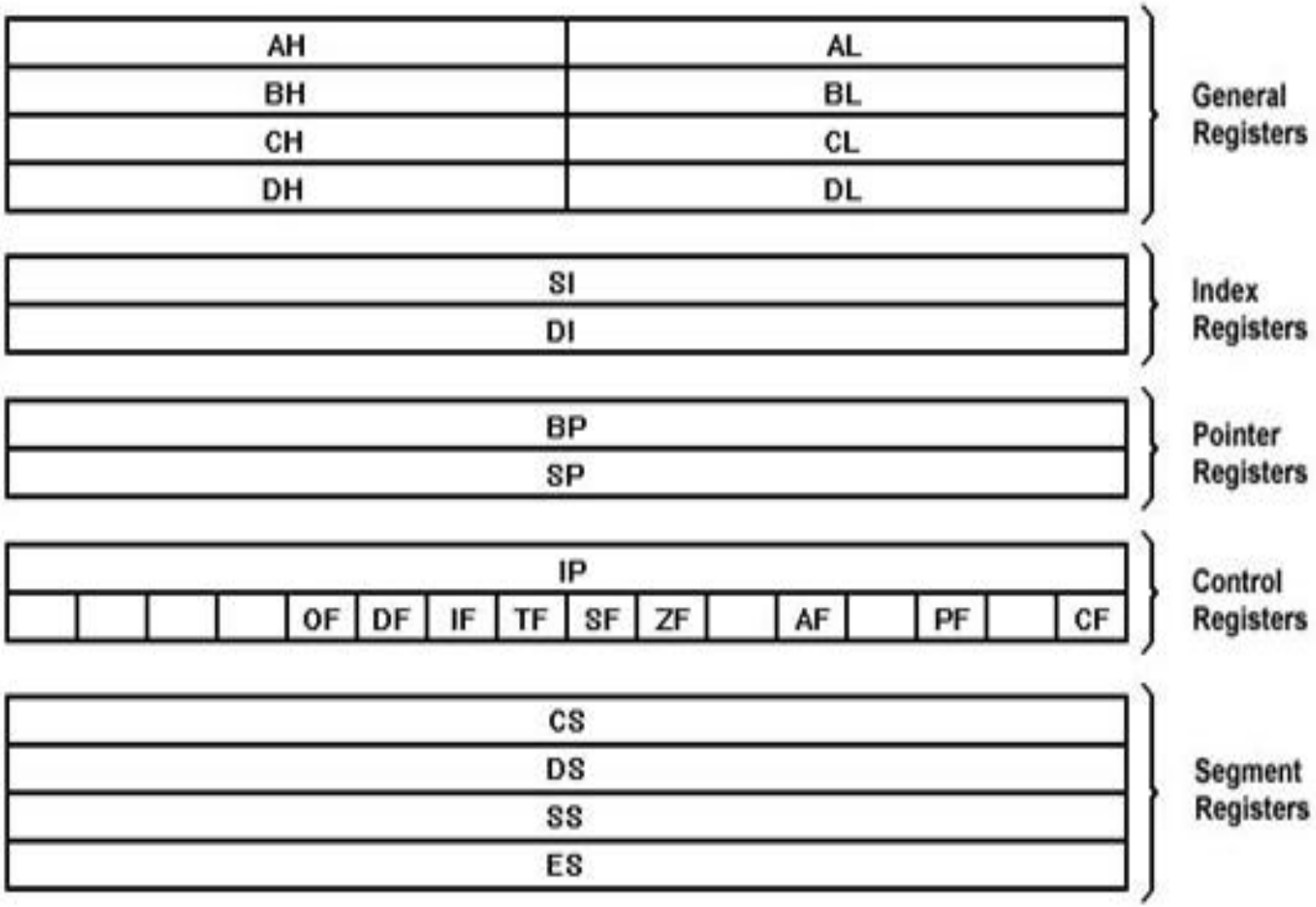
- Used in string traversals

Questions

1. Explain the flag register of 8086
2. START : MOV AX, 0048H
   MOV BX, 4AE0H
   SUB AL, BH

Give the contents of the Flag register after execution of the above instruction.

Programmer's model

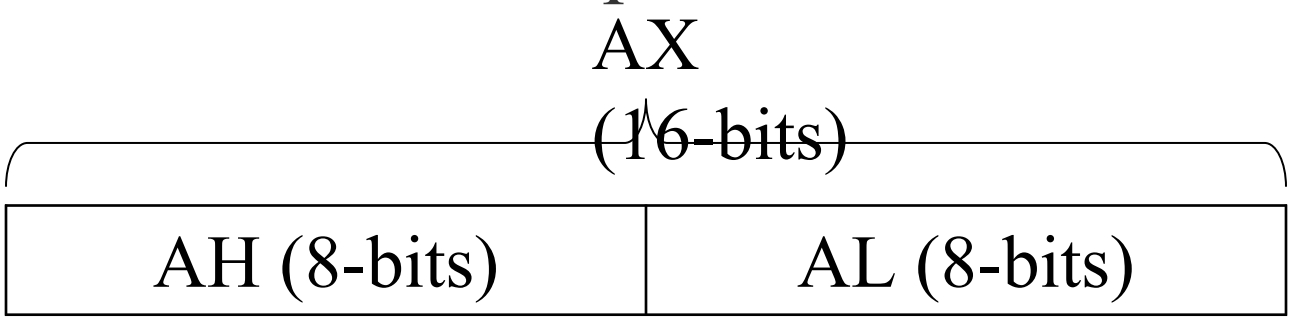


General Purpose Registers

- EU has 4 general purpose registers labeled as **AX, BX, CX** and **DX.**
- Each can be used as two 8-bit registers (H and L)
- They are used as
  - operands in most of the instructions
  - source or destination registers during computation and transfer of data
  - pointers to memory
  - counters.
- Programmer is free to use these registers in anyway.

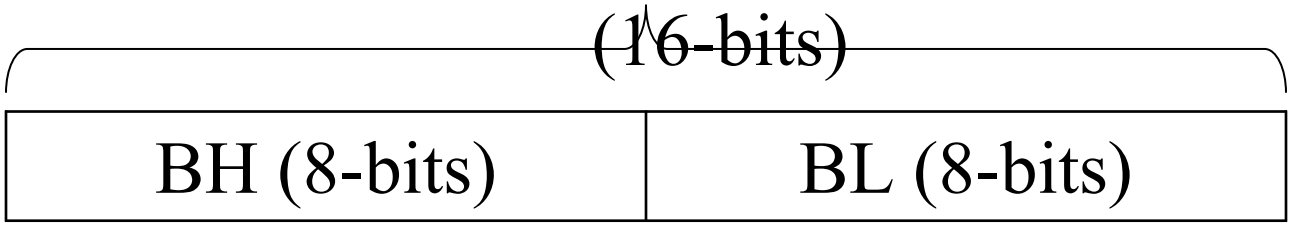
AX Register

- It functions as **Accumulator**
- It is the default operand in multiplication and division
- AL is the default location for port inputs.
- It is divided into two parts – AH and AL.



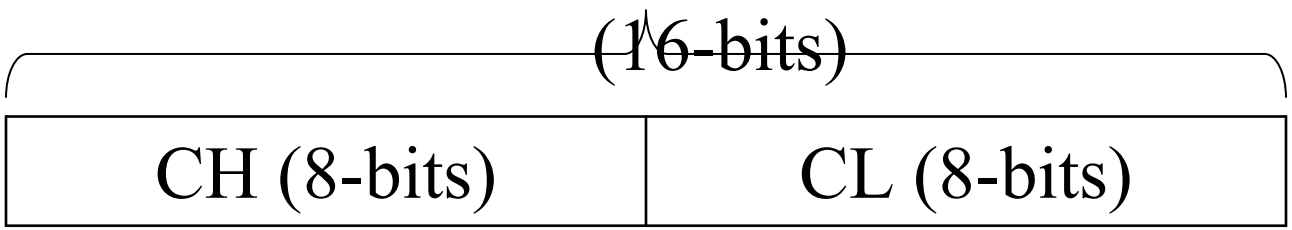
BX Register

- It functions as the **Base register**
- Can be used as a pointer to a memory location
- It is suitable for accessing the elements of an array.
- It can be treated as BH and BL



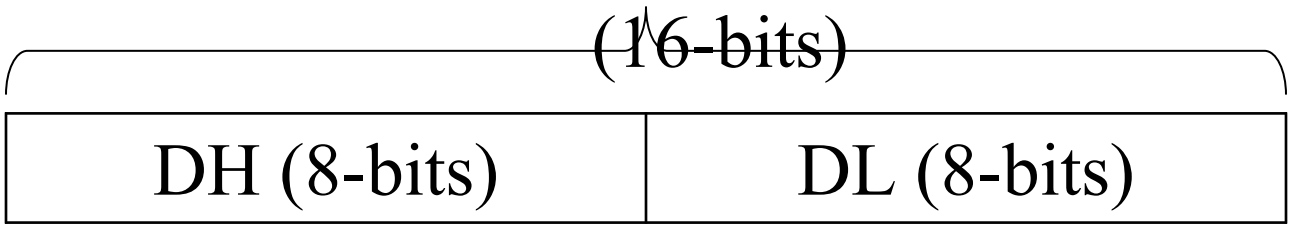
CX Register

- It functions as the **Count register**
- Loop and repeat instructions use CX register to hold repeat count value
- It can be treated as CH and CL



DX Register

- It functions as the **Display register**
- It is used as an I/O address pointer in the IN and OUT instructions
- It is also involved in 16-bit multiplication and division as a default operand
- It can be treated as DH and DL.



Pointer and Index Register

- The pointer and index registers are used to hold the offset within a segment.

- They are
  - Stack Pointer (SP) Register
  - Base Pointer (BP) Register
  - Source Index (SI) Register
  - Destination Index (DI) Register.

SI (Source Index Register) & DI (Destination Index Register)

- They are used as a pointer to memory location
- They are useful in accessing contiguous memory location, such as string or array
- In repeat string instructions, SI is used as a pointer to a source string element and DI is used as a pointer to a destination string element

Stack Pointer (SP) Register

- This register is dedicated for maintaining an area of memory used as a stack
- In stack memory, data can be stored and retrieved on the basis of Last-In-First-Out (LIFO)
- Stack is useful in implementing procedure calls, recursions, passing of parameters, creation of local variables, etc.
- SP always points to the top of the stack relative to SS

Base Pointer (BP) Register

- It points relative to the stack segment register
- It is generally used for accessing the parameters from the stack in the procedures
- It is designed to provide support for passing of parameters between procedures, local variables and other stack based allocation and operations
- It is used to access any location directly in the stack

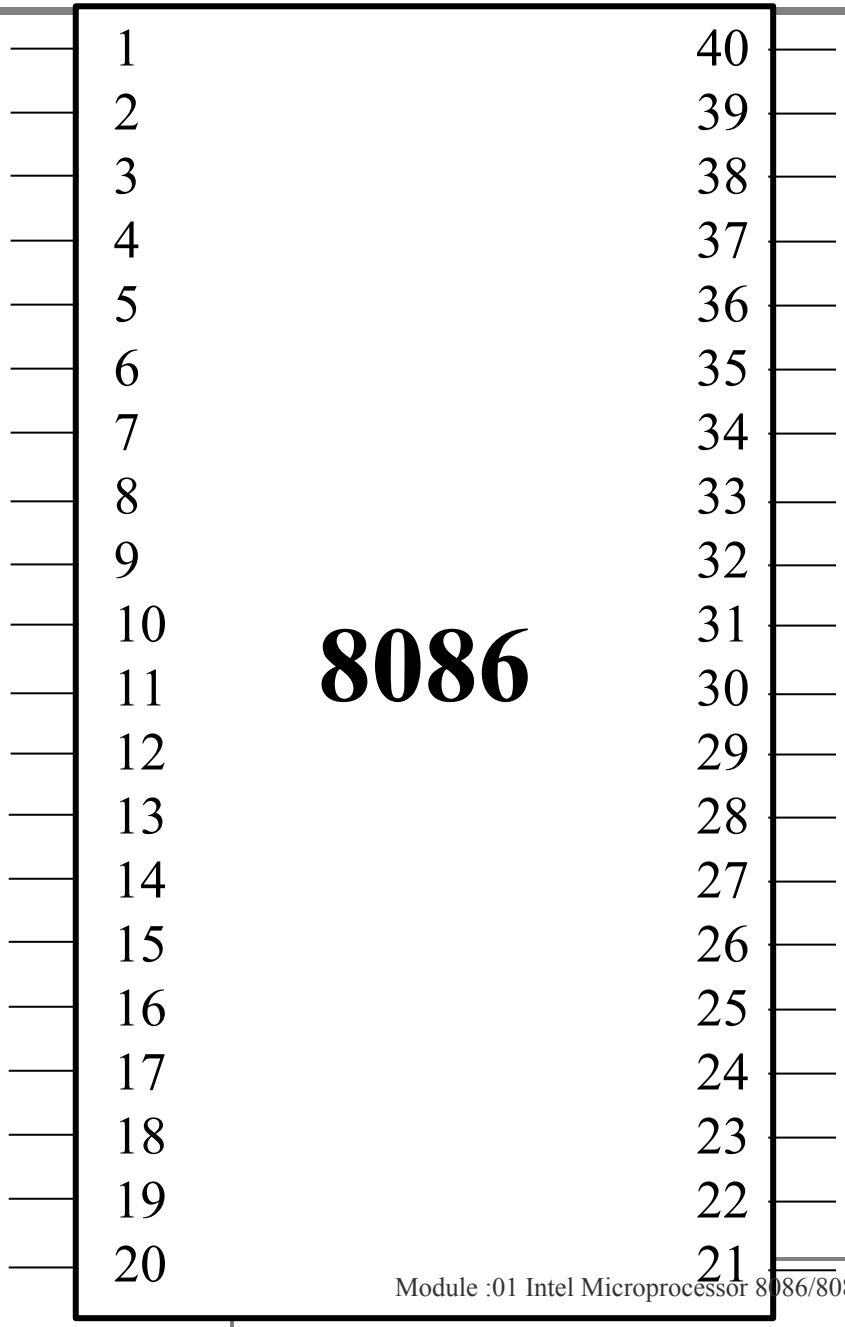
Instruction Pointer (IP) Register

- It holds the offset of next instruction to be executed
- Some instructions (JMP and CALL) can cause the IP to be loaded with a new value.

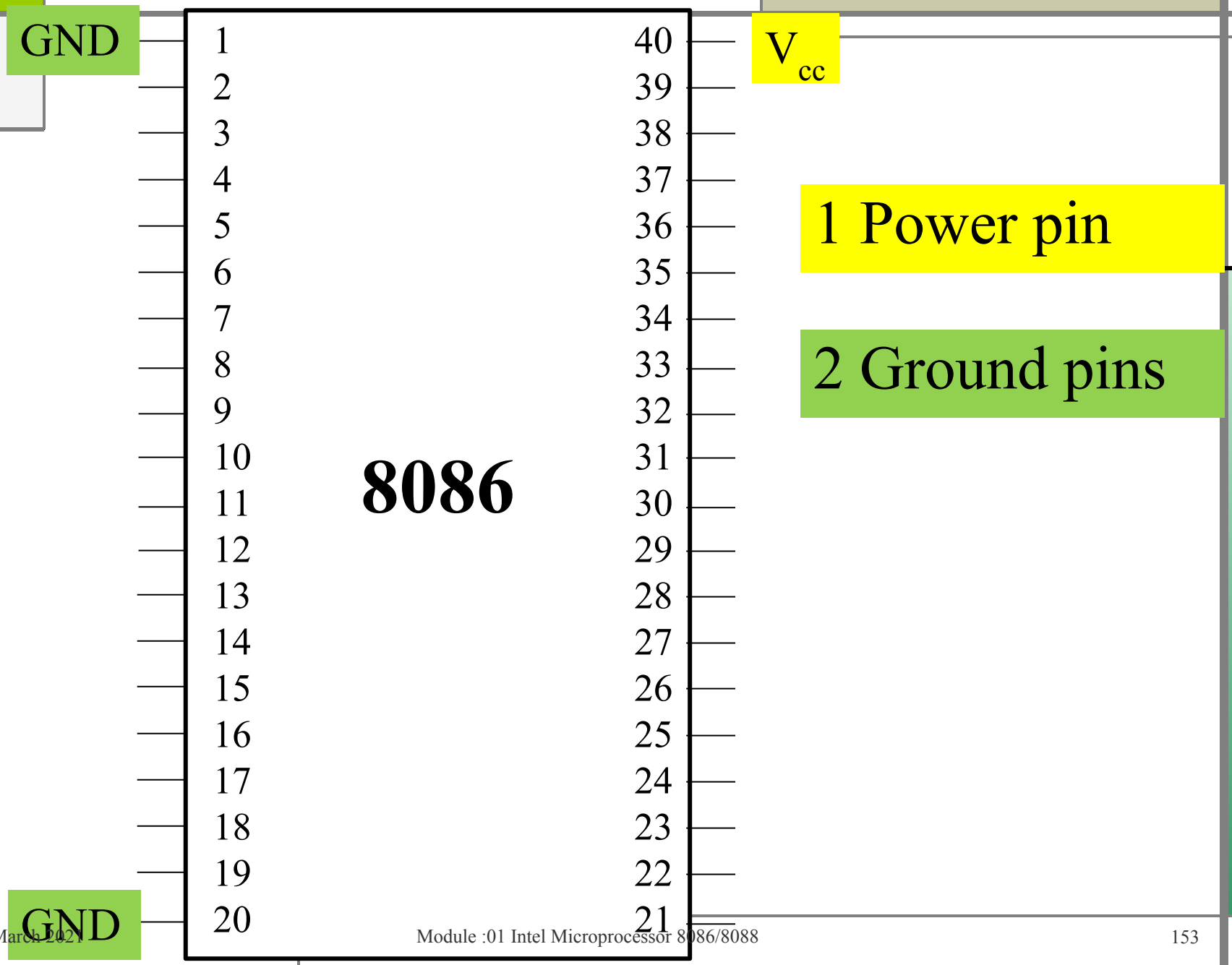
Question

- Explain the programming model of 8086?

8086 PIN CONFIGURATION

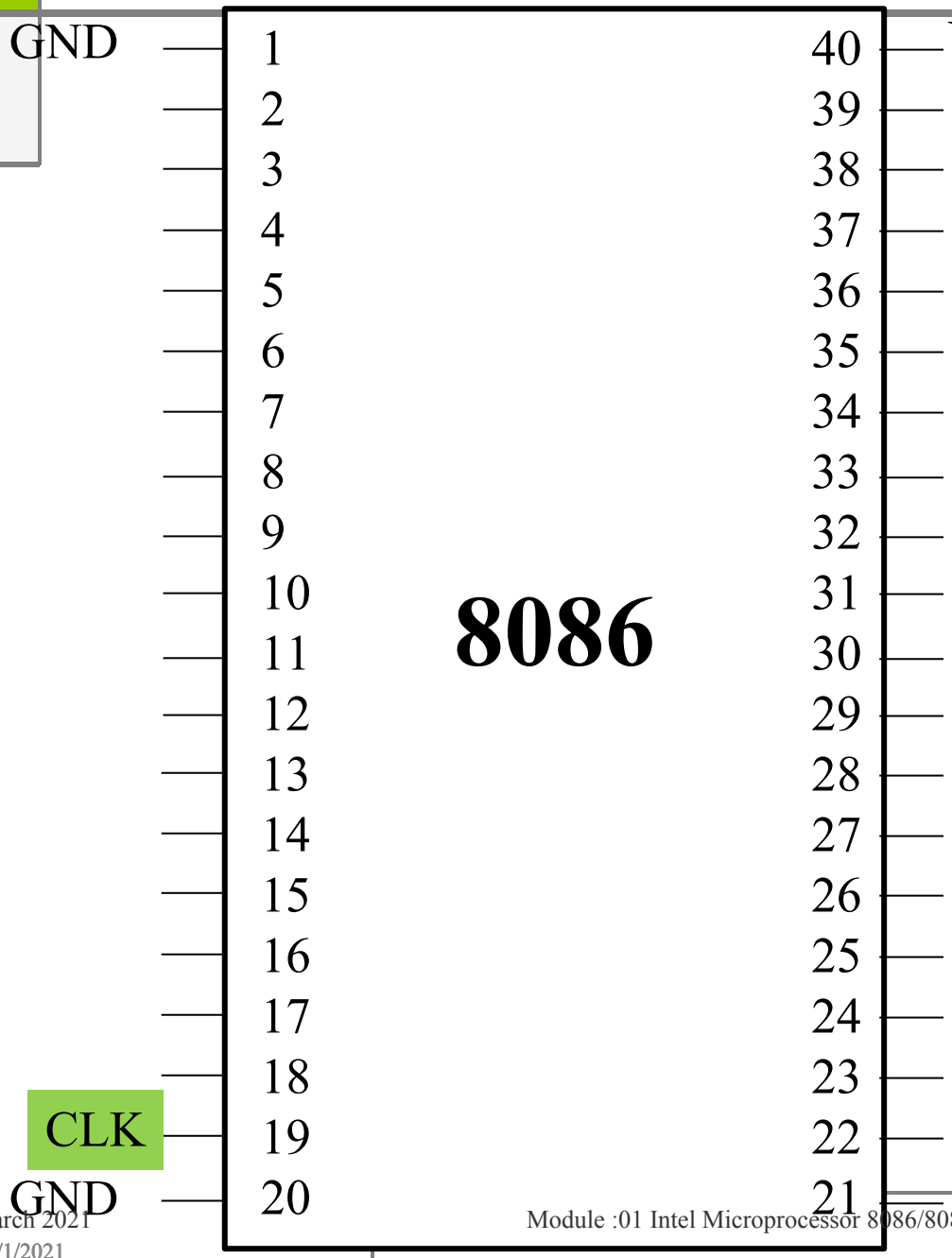


40 pin-DIP (Dual Inline Package)



1 Power pin

2 Ground pins



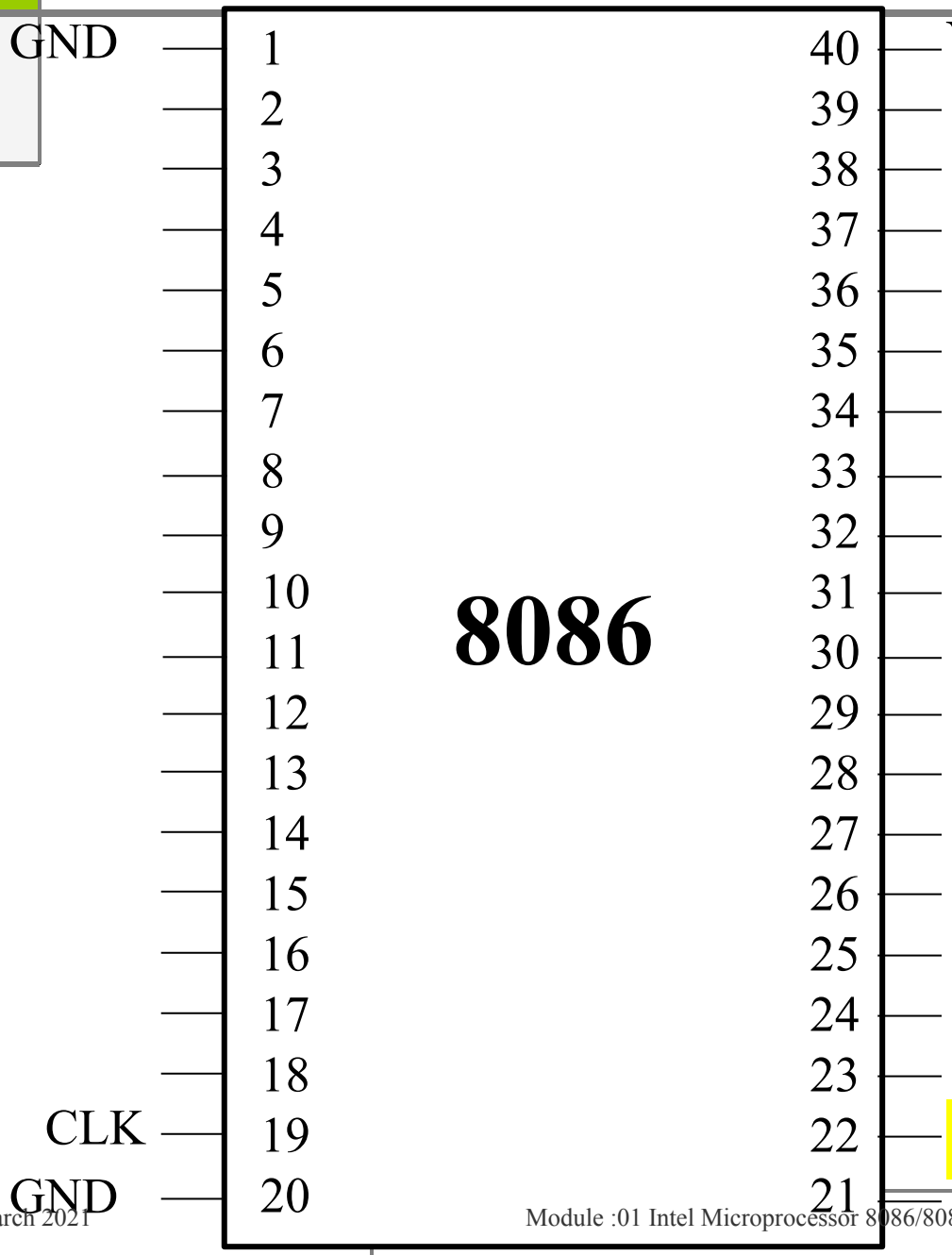
CLK

- Sync events with a 8284 (clock)

RESET = 1

(for 4 clk cycles)

- Terminate current activity.
- Clears all registers and empties the instruction queue



READY = 1

- Data transfer is complete.
- Processor is ready for execution

READY = 0

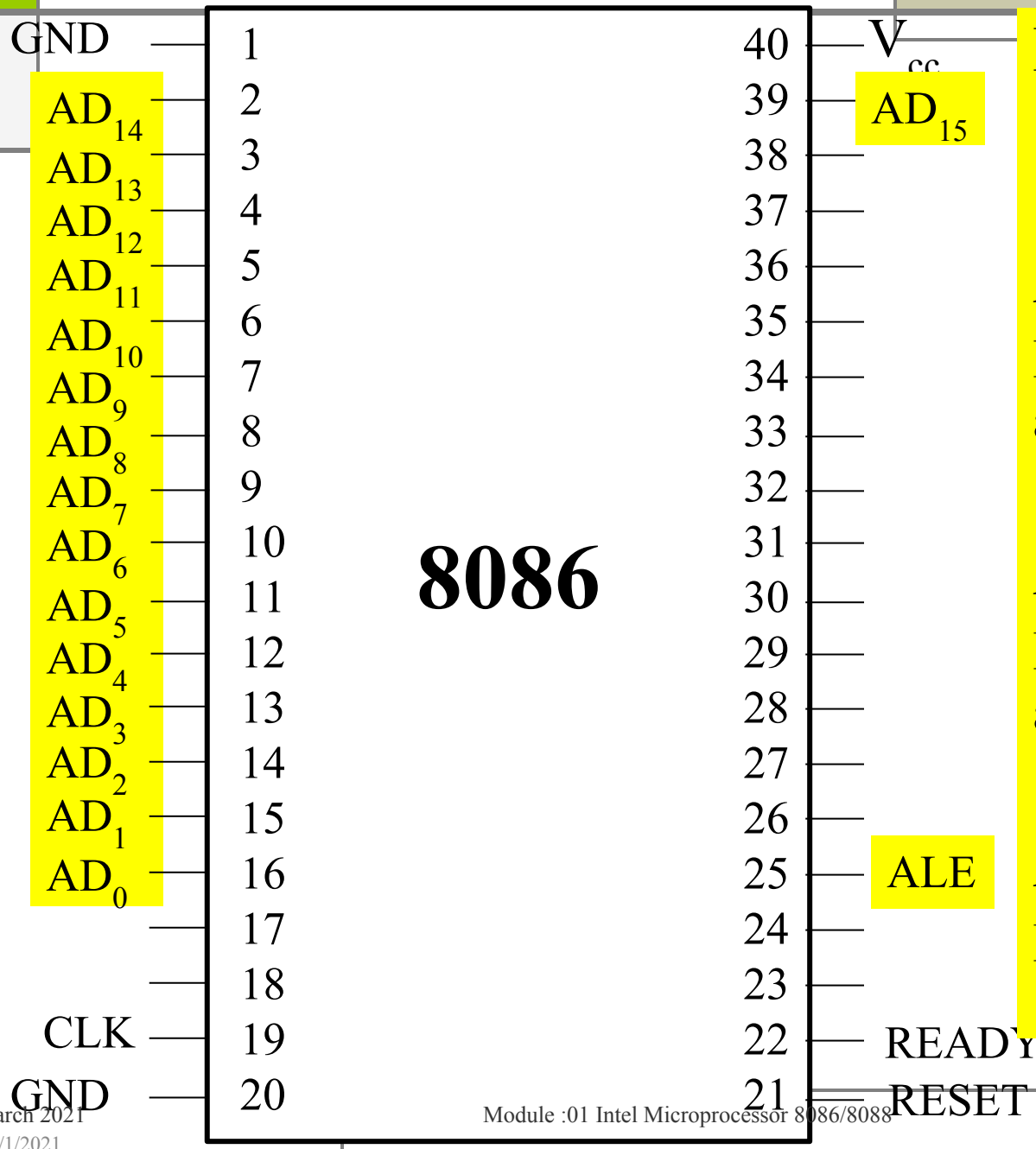
- Processor is waiting for some resource.

Time Multiplexing

When the same pin has different functions during different time cycles,

that pin is said to be time multiplexed.

Aren’t all humans time multiplexed?

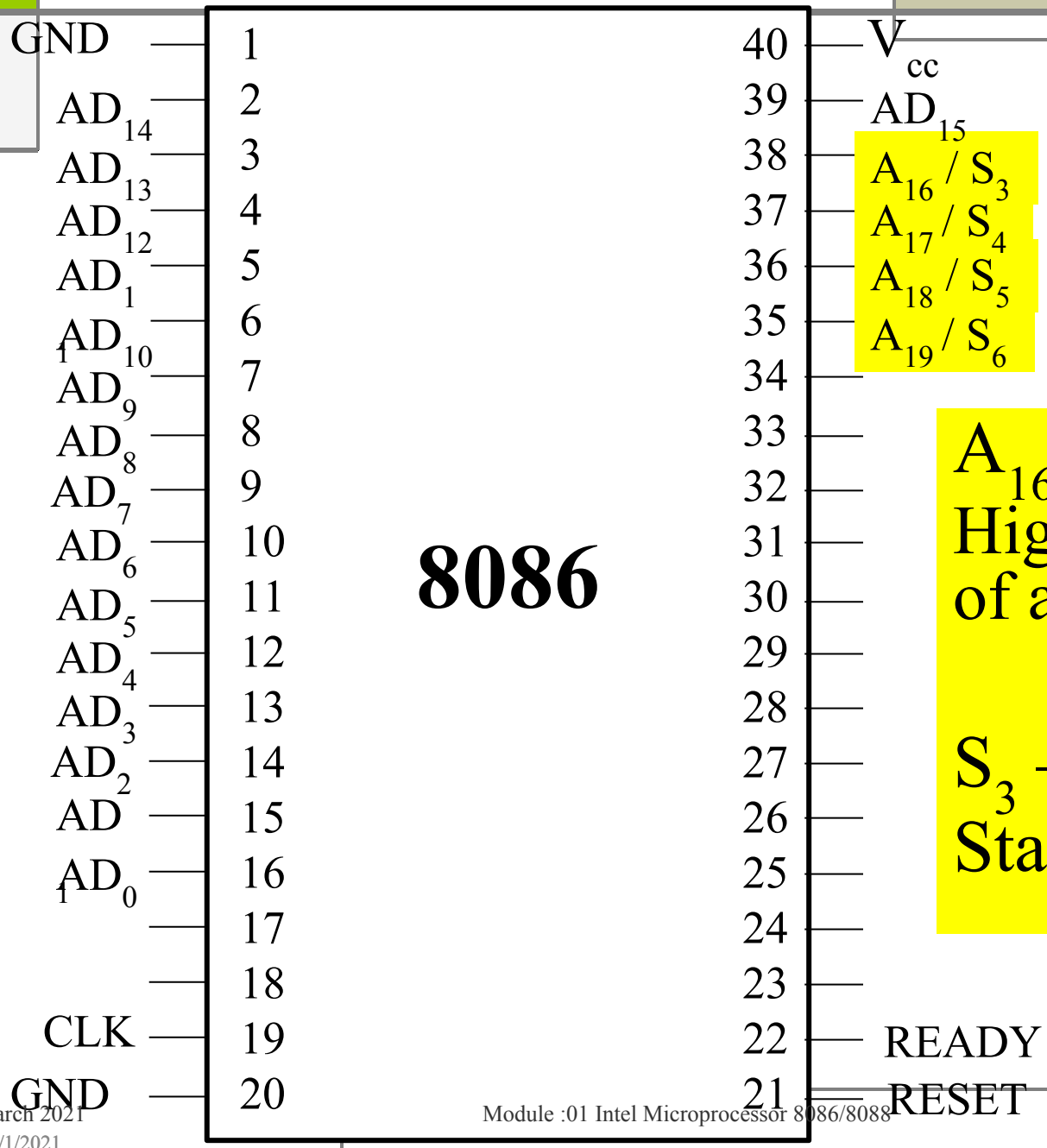


$D_0 - D_{15}$
16-bit data line

$A_0 - A_{15}$
Lower 16 bits of address line

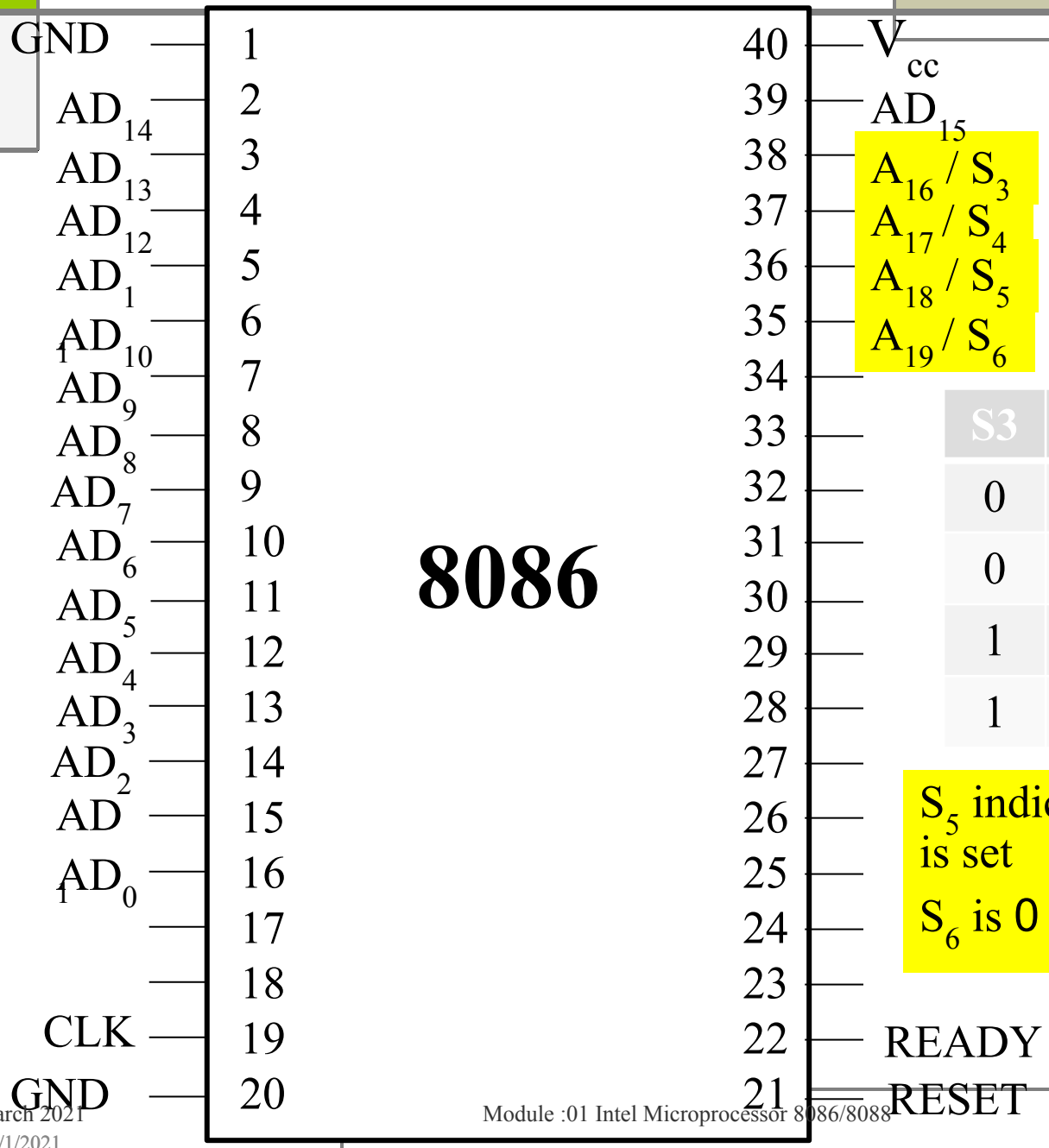
ALE = 1
Line carries address

ALE = 0
Line carries data



$A_{16} – A_{19}$
Higher 4 bits of address line

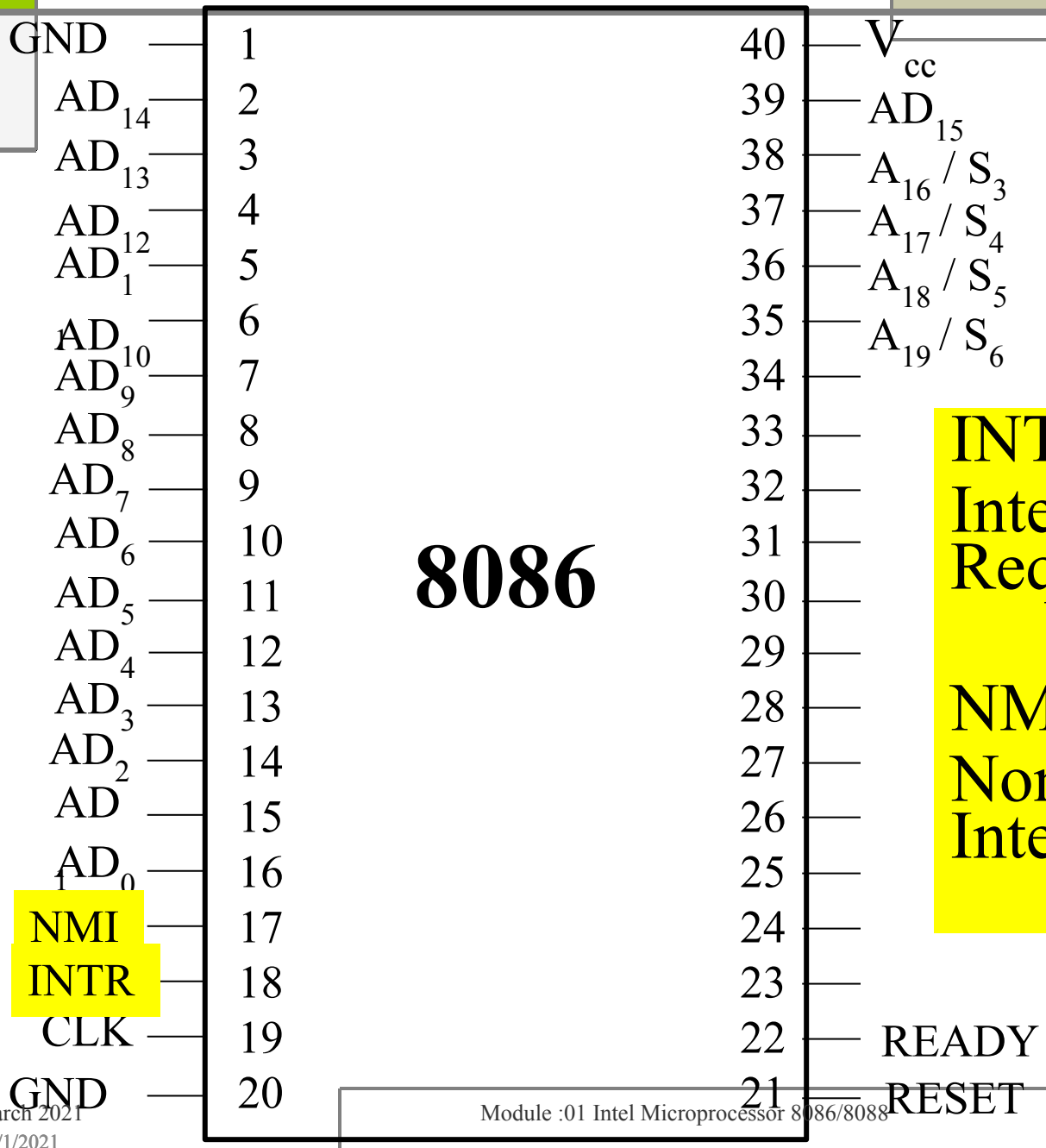
$S_3 – S_6$
Status Signals



S3	S4	Segment
0	0	Extra
0	1	Stack
1	0	Code
1	1	Data

S_5 indicates interrupt flag is set

S_6 is 0 when 8086 is BM



INTR
Interrupt Request

NMI
Non-Maskable Interrupt

Active High / Active Low?

- Describes how a pin is activated.
- Active high pins are enabled when set to 1
- Active low pins are enabled when set to 0
- By default all pins are directly connected to the Vcc.
- Active low pins are connected via NOT gate
- If we do not want certain pins to be active by default, we will reverse their role.

Why active low pins?

Consider a water tank.

When tank is filled more than half,

assume L = 1

When tank falls to less than half

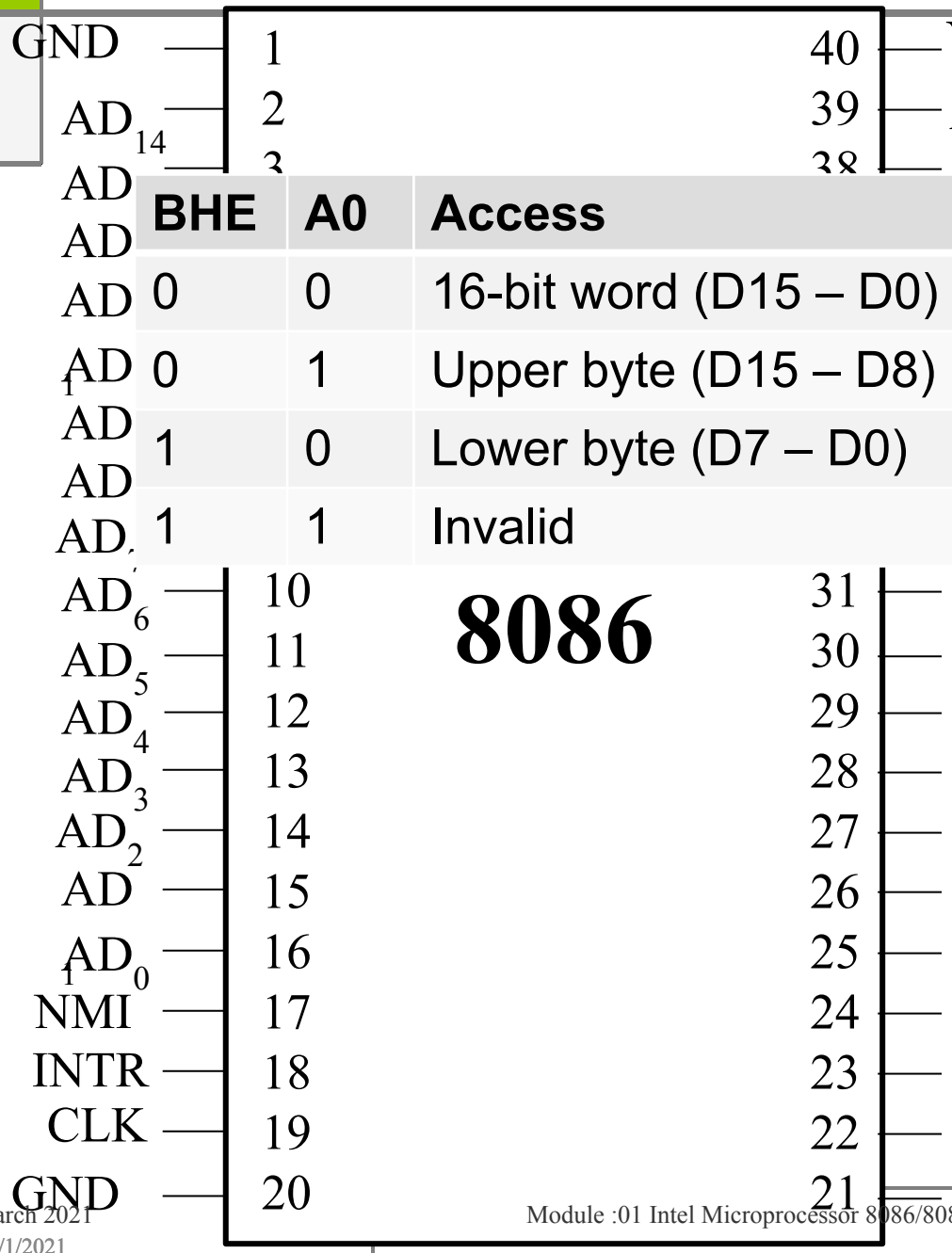
assume L = 0

i.e., L indicates the water level.

When should the water pump motor start?

When L = 0

Or L = 1 ??



BHE	A0	Access
0	0	16-bit word (D15 – D0)
0	1	Upper byte (D15 – D8)
1	0	Lower byte (D7 – D0)
1	1	Invalid

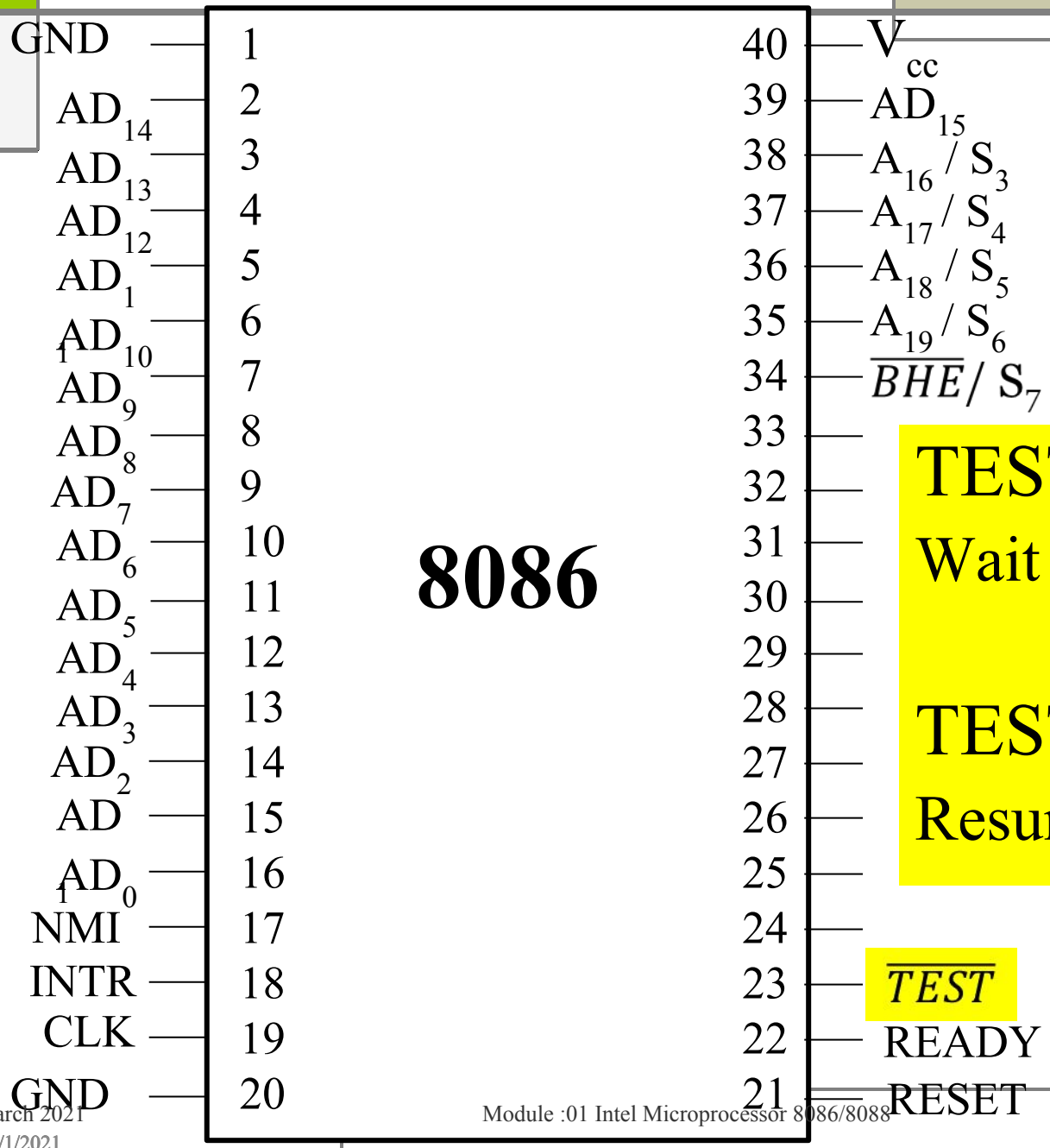
BHE = 0

Enable data on D_8 – D_{15}

BHE = 1

Enable data on D_0 – D_7

S7 reserved for future

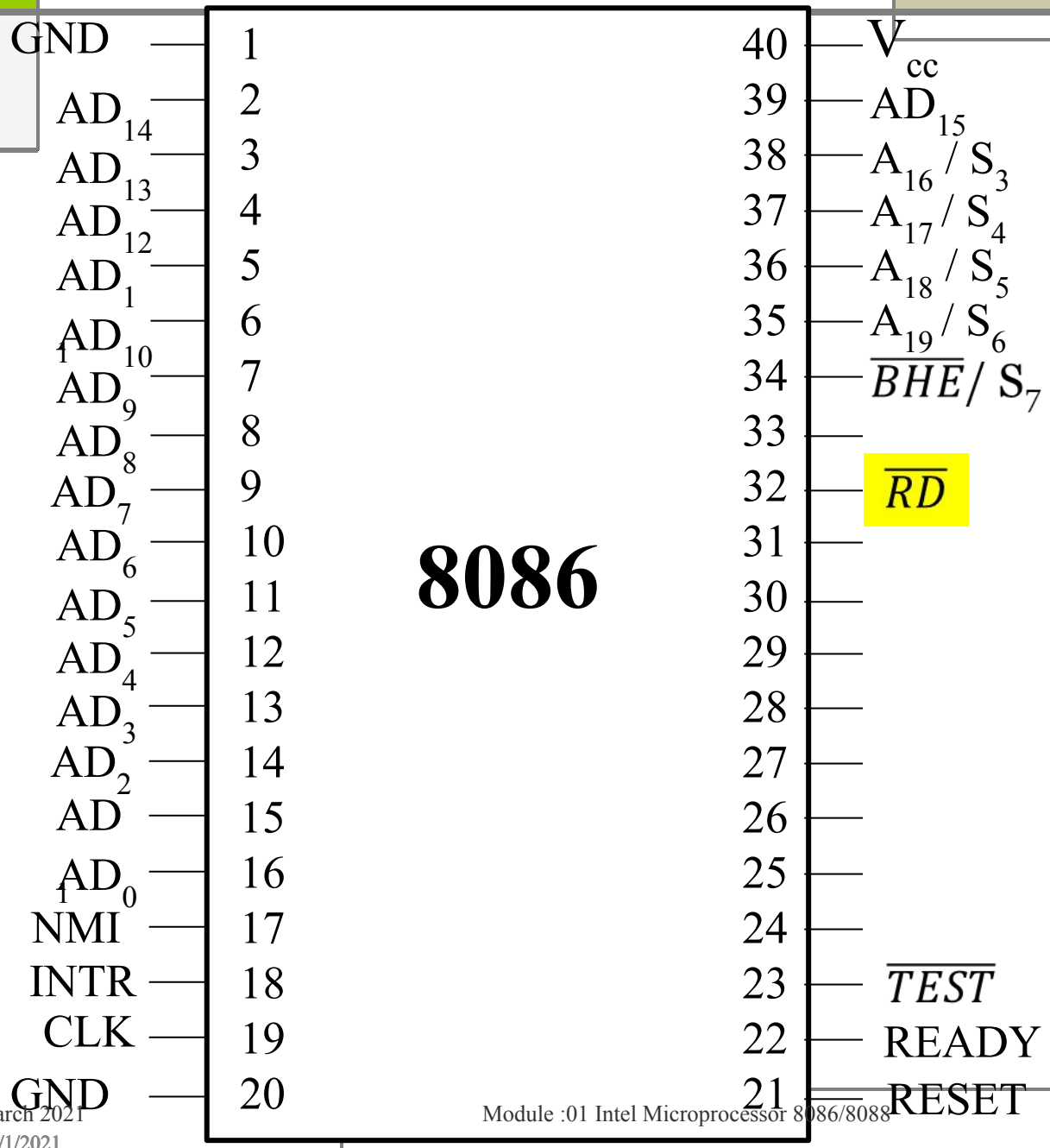


TEST = 0

Wait instruction

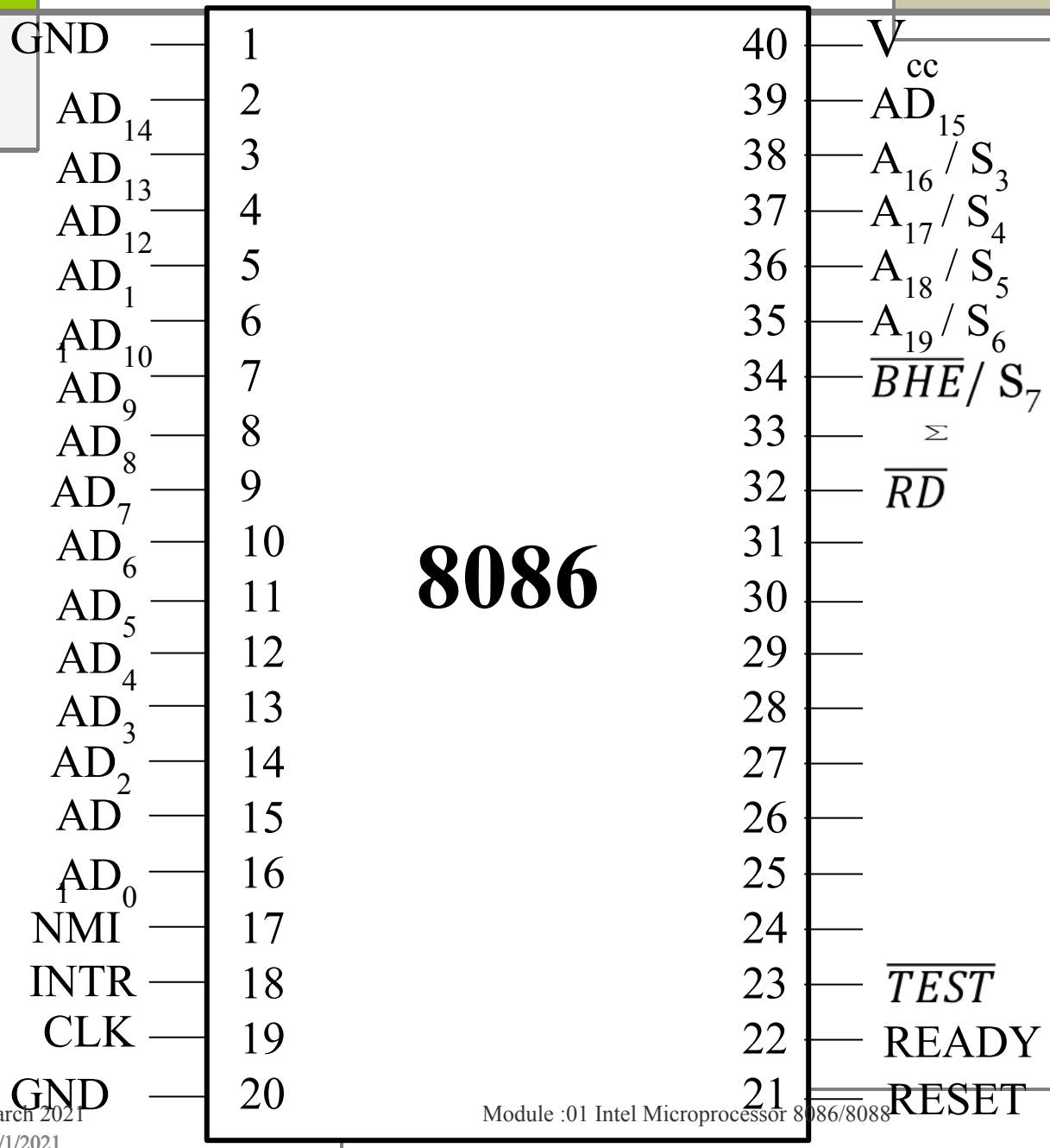
TEST = 1

Resume execution



RD = 0
Read

RD = 1
No read

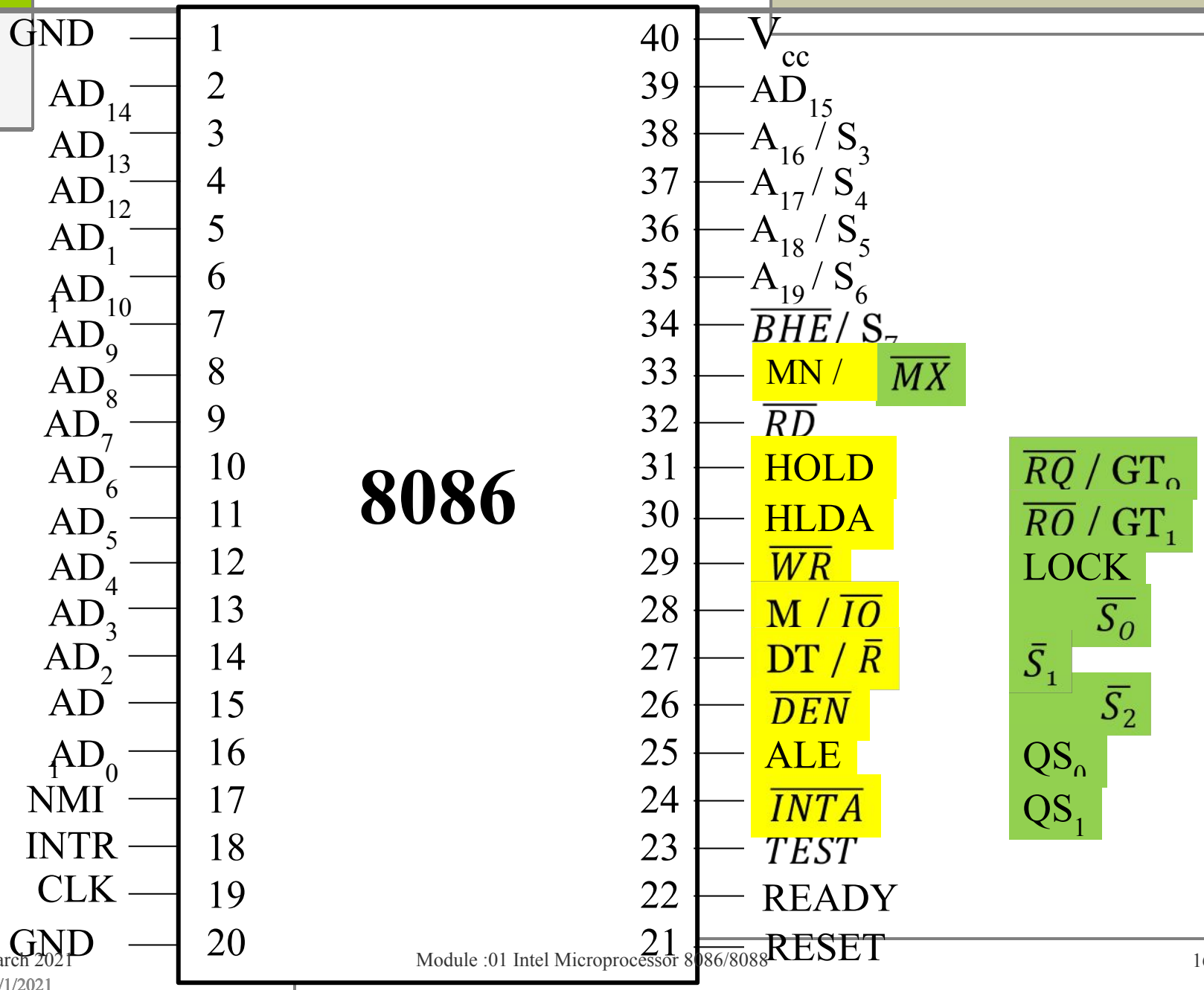

GND 1
AD14 2
AD13 3
AD12 4
AD1 5
AD10 6
AD9 7
AD8 8
AD7 9
AD6 10
AD5 11
AD4 12
AD3 13
AD2 14
AD 15
AD0 16
NMI 17
INTR 18
CLK 19
GND 20
8086
40 Vcc
39 AD15
38 A16 / S3
37 A17 / S4
36 A18 / S5
35 A19 / S6
34 BHE / S7
33 Σ
32 RD
31
30
29
28
27
26
25
24
23 TEST
22 READY
21 RESET
0
Max Mode
1
Min Mode

Modes of Operation

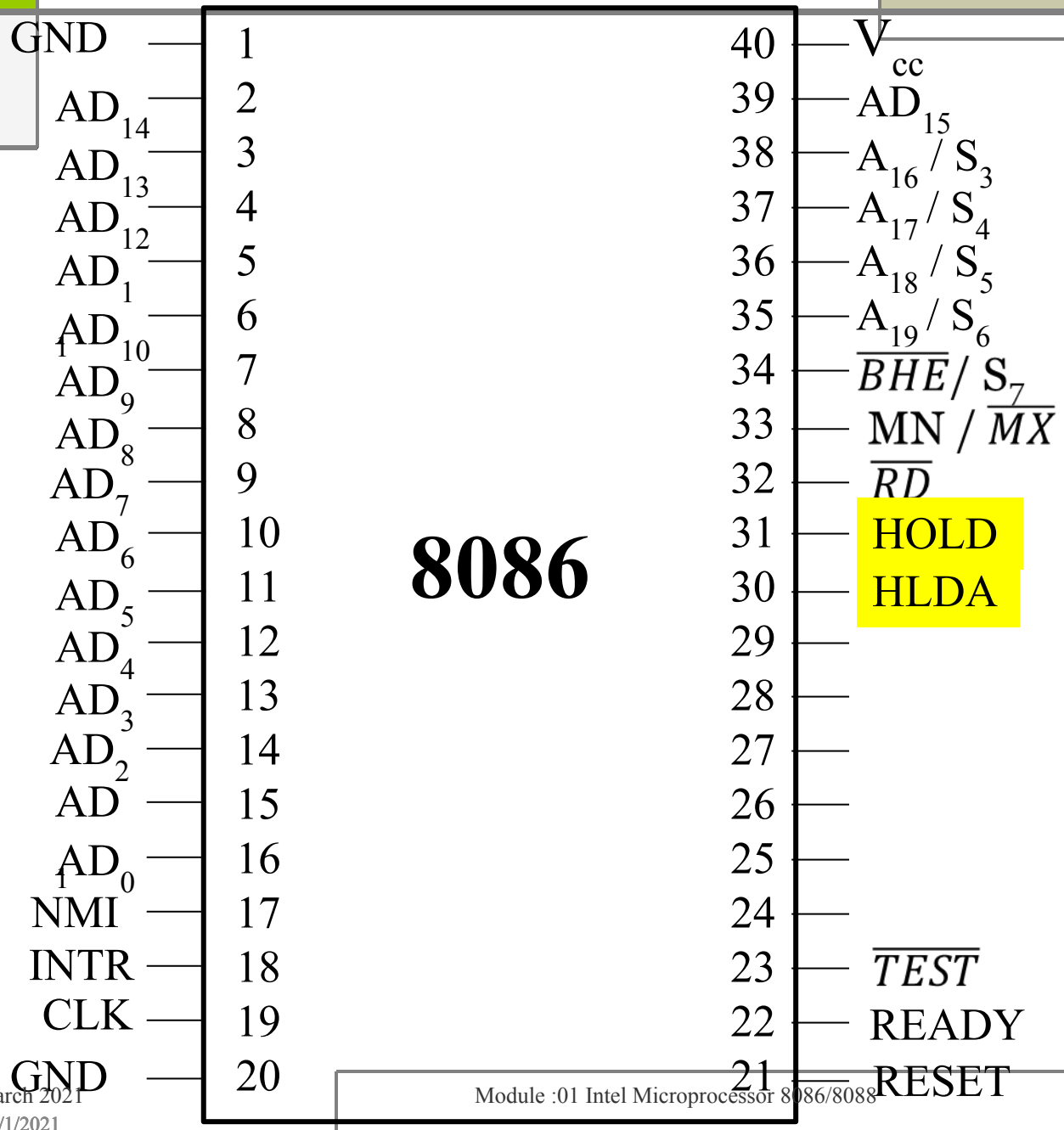
Processor needs control over the address, data and control buses to access memory and I/O devices.

- **Minimum mode** – single processor mode
  - Processor issues control signals
- **Maximum mode** – multi processor mode
  - The bus controller issues control signals

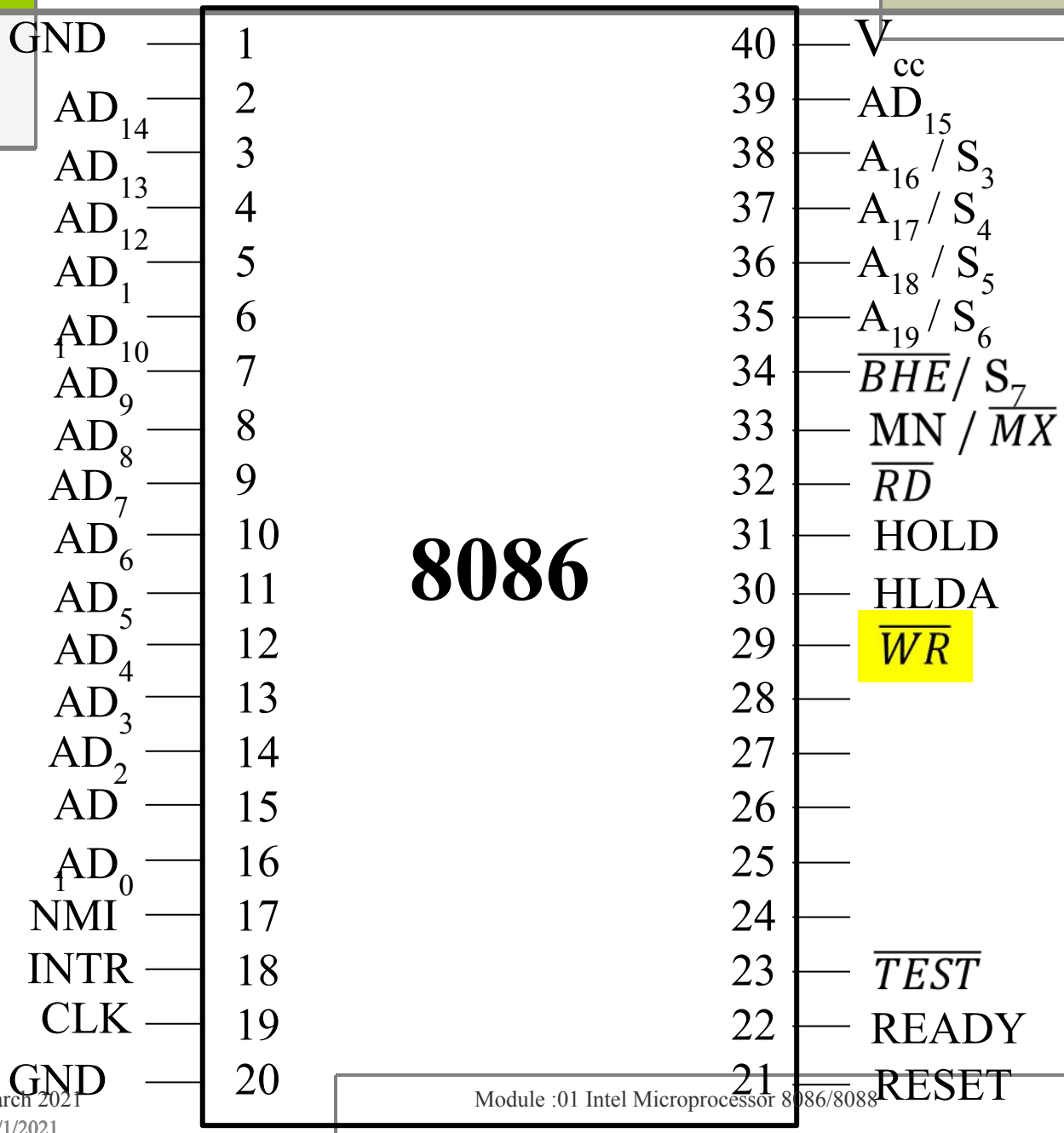
These modes of operations are available only in

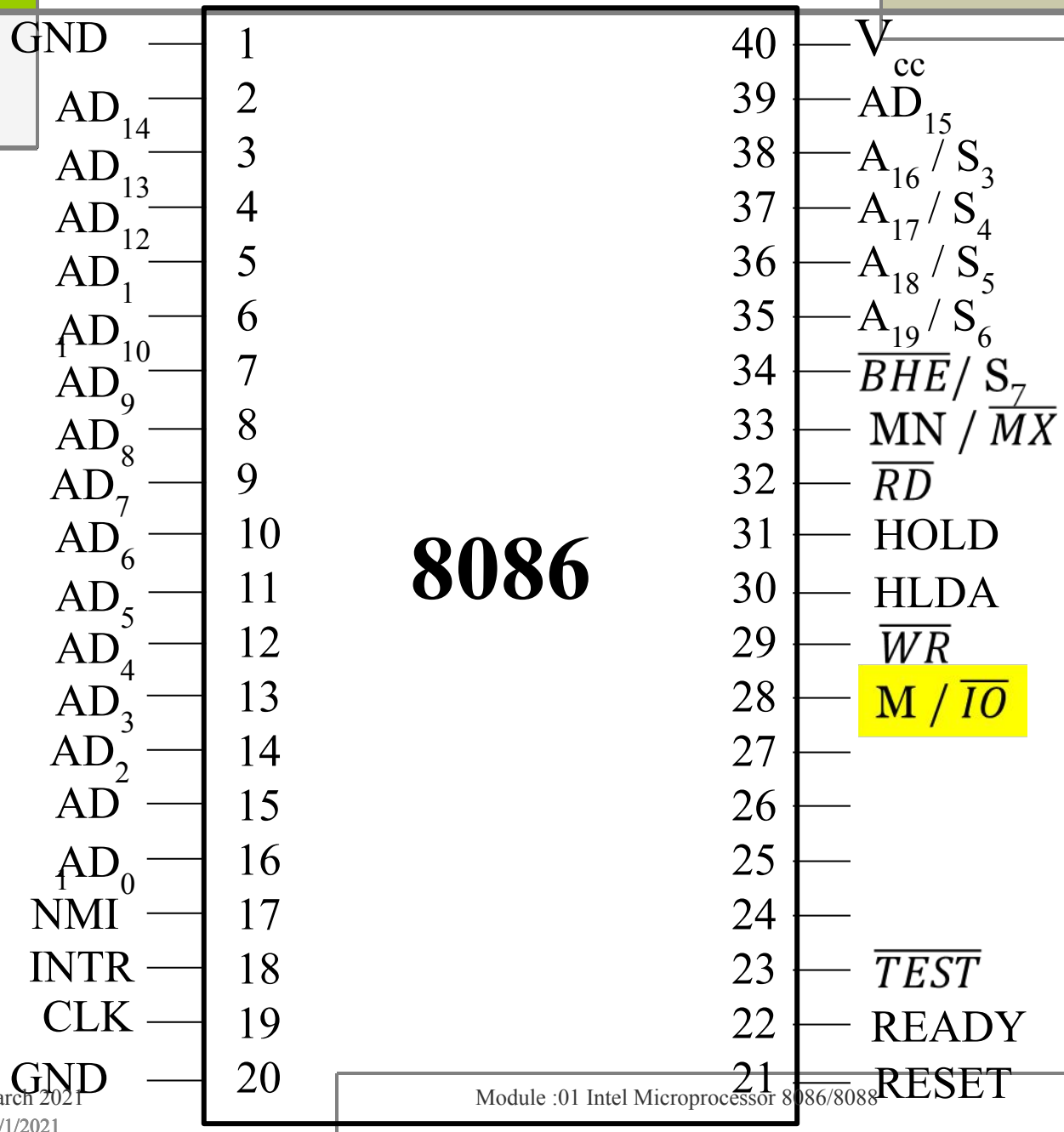

8086
GND 1
AD14 2
AD13 3
AD12 4
AD1 5
AD10 6
AD9 7
AD8 8
AD7 9
AD6 10
AD5 11
AD4 12
AD3 13
AD2 14
AD 15
AD0 16
NMI 17
INTR 18
CLK 19
GND 20
40 Vcc
39 AD15
38 A16 / S3
37 A17 / S4
36 A18 / S5
35 A19 / S6
34 BHE / S7
33 MN / MX
32 RD
31 HOLD
30 HLDA
29 WR
28 M / IO
27 DT / R
26 DEN
25 ALE
24 INTA
23 TEST
22 READY
21 RESET
RQ / GT0
RO / GT1
LOCK
S0
S1
S2
QS0
QS1

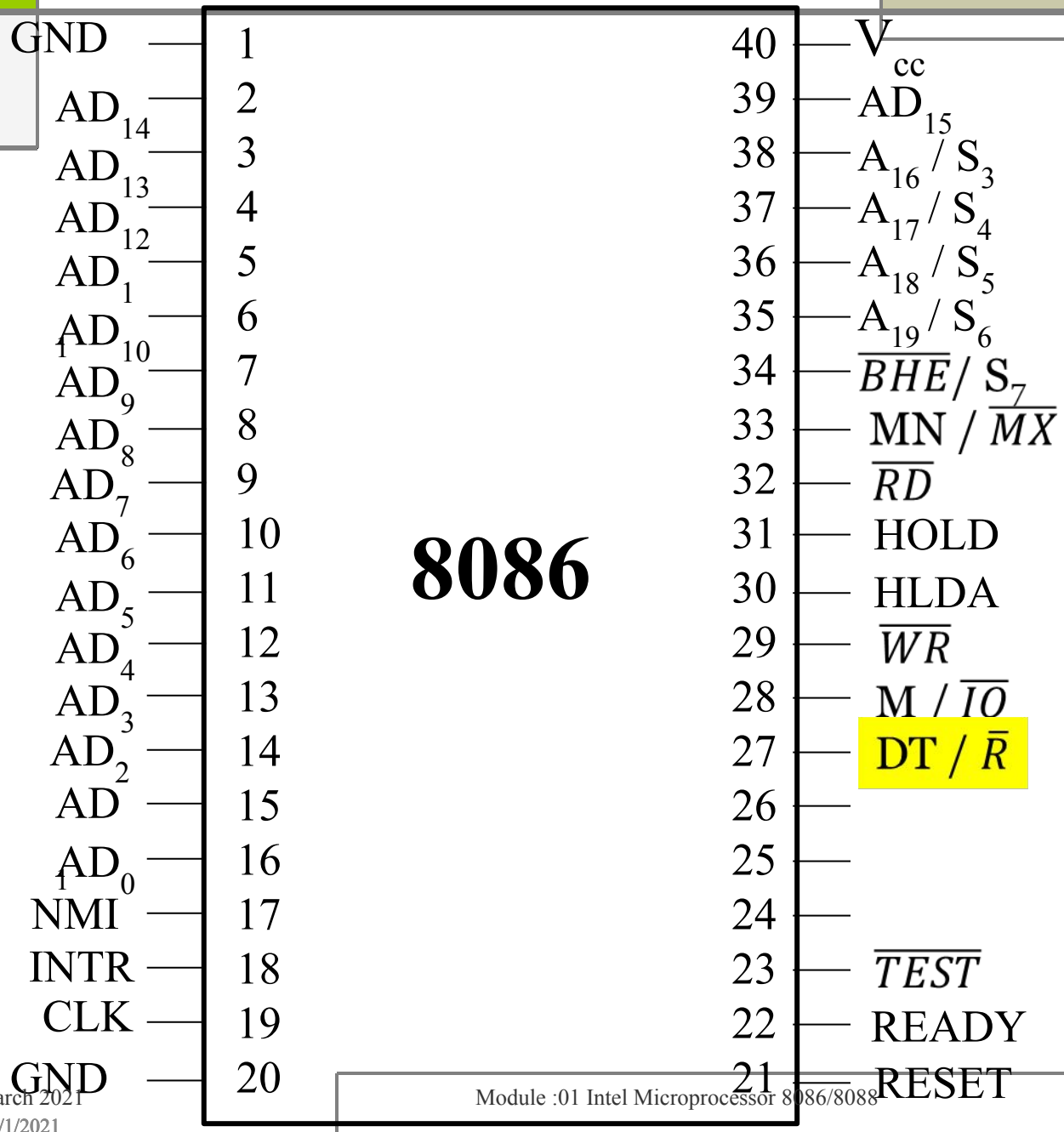
Minimum Mode



When the DMA controller wants to take control of the data bus, it seeks the permission of the processor by setting **HOLD**. Processor gives permission by setting **HLDA**.


8086
GND 1
AD14 2
AD13 3
AD12 4
AD1 5
AD10 6
AD9 7
AD8 8
AD7 9
AD6 10
AD5 11
AD4 12
AD3 13
AD2 14
AD 15
AD0 16
NMI 17
INTR 18
CLK 19
GND 20
40 Vcc
39 AD15
38 A16 / S3
37 A17 / S4
36 A18 / S5
35 A19 / S6
34 $\overline{BHE}$ / S7
33 MN / $\overline{MX}$
32 $\overline{RD}$
31 HOLD
30 HLDA
29 $\overline{WR}$
28
27
26
25
24
23 $\overline{TEST}$
22 READY
21 RESET


8086
GND 1
AD14 2
AD13 3
AD12 4
AD1 5
AD10 6
AD9 7
AD8 8
AD7 9
AD6 10
AD5 11
AD4 12
AD3 13
AD2 14
AD 15
AD0 16
NMI 17
INTR 18
CLK 19
GND 20
40 Vcc
39 AD15
38 A16 / S3
37 A17 / S4
36 A18 / S5
35 A19 / S6
34 $\overline{BHE}$ / S7
33 MN / $\overline{MX}$
32 $\overline{RD}$
31 HOLD
30 HLDA
29 $\overline{WR}$
28 M / $\overline{IO}$
27
26
25
24
23 $\overline{TEST}$
22 READY
21 RESET

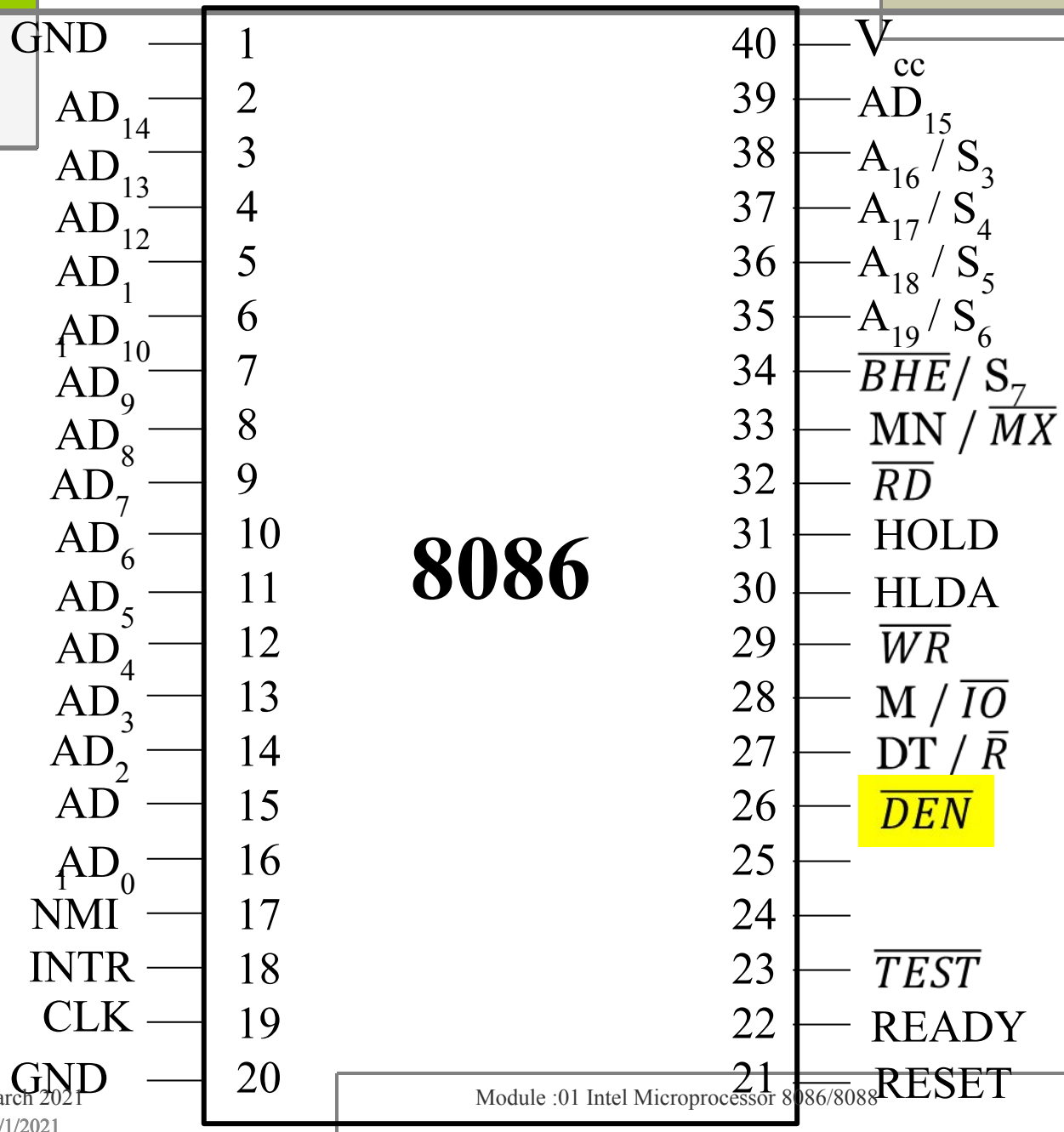


0

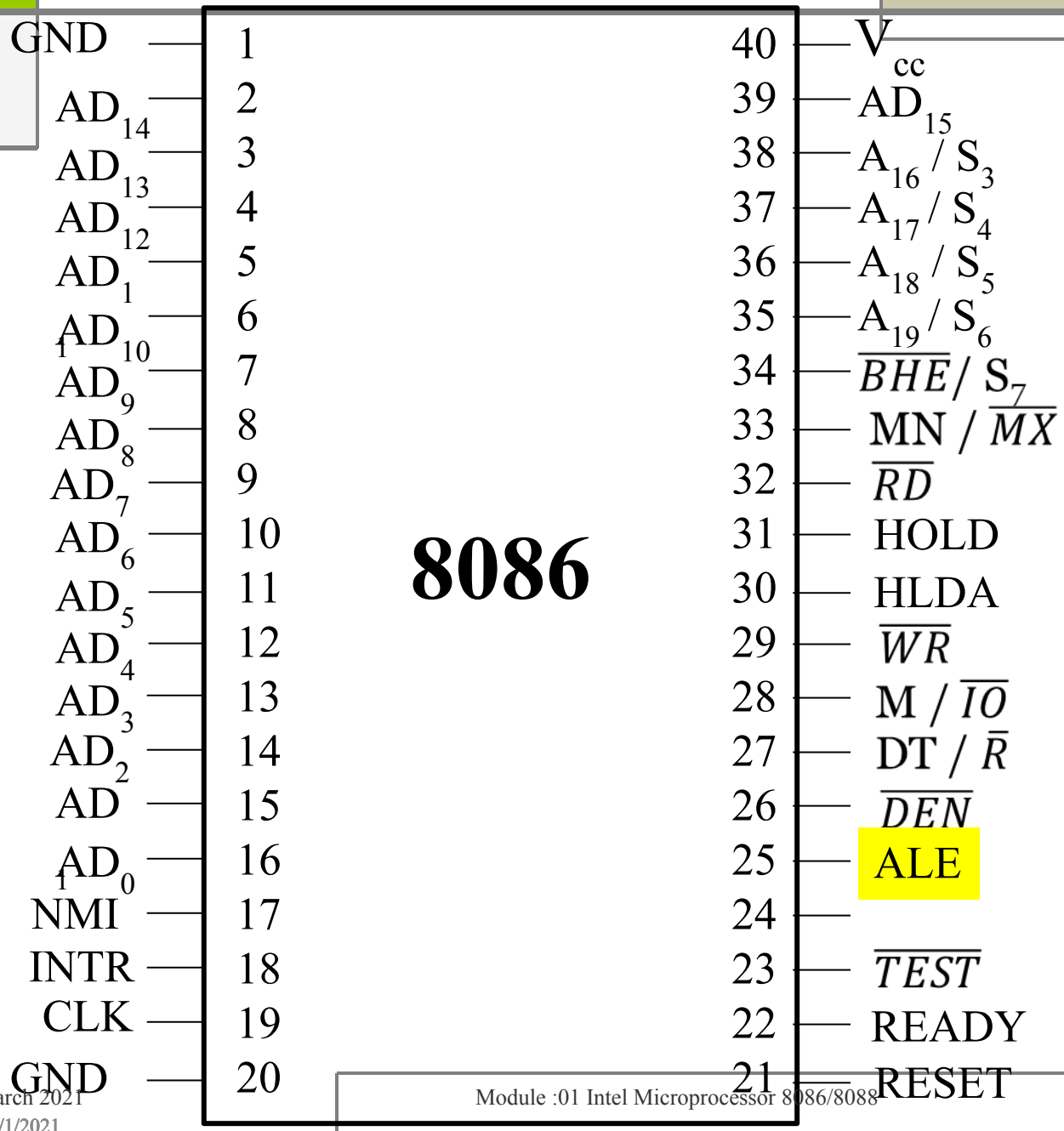
Receive

1

Transmit



Enables the data on the external buffers

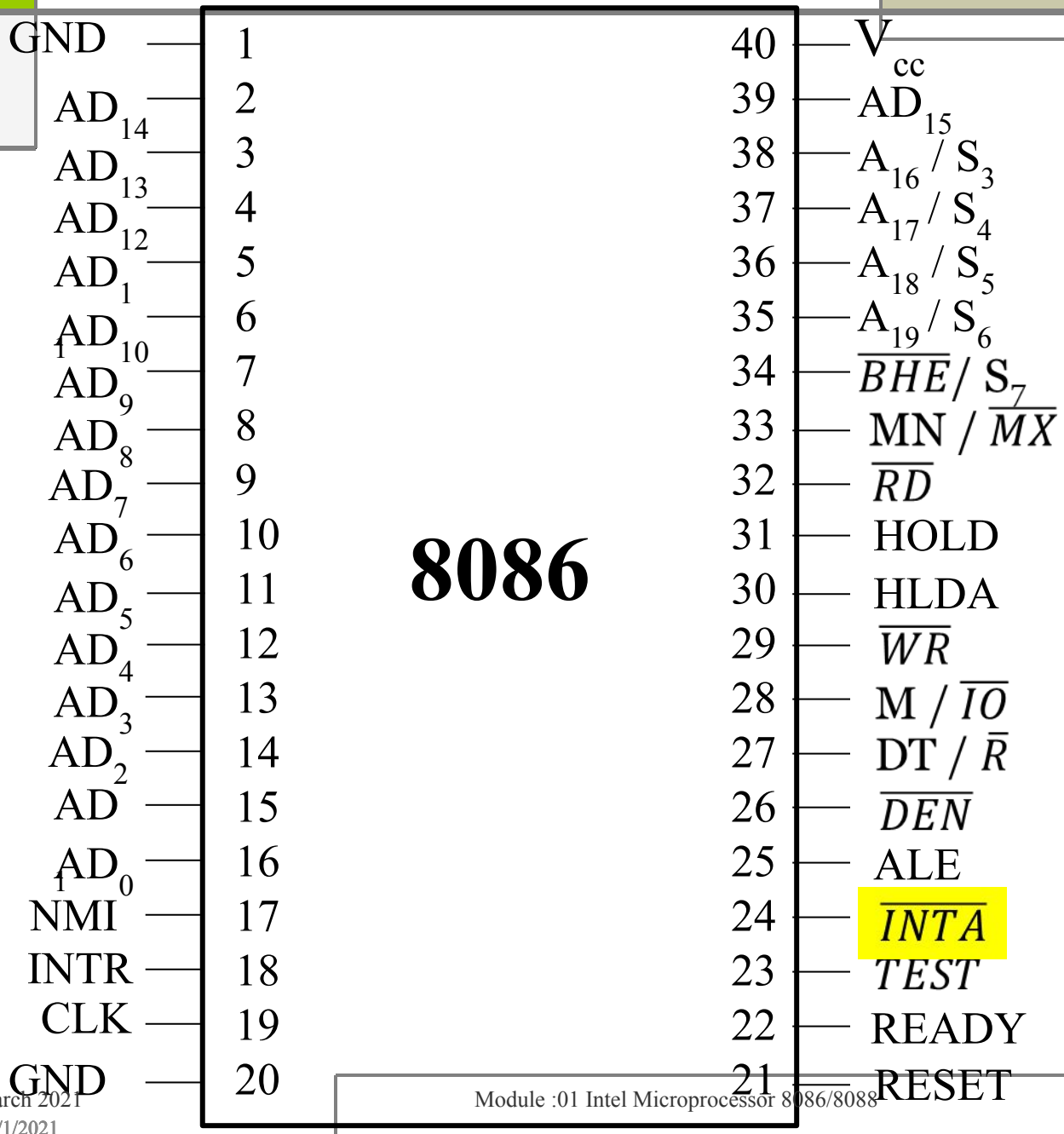


ALE=0

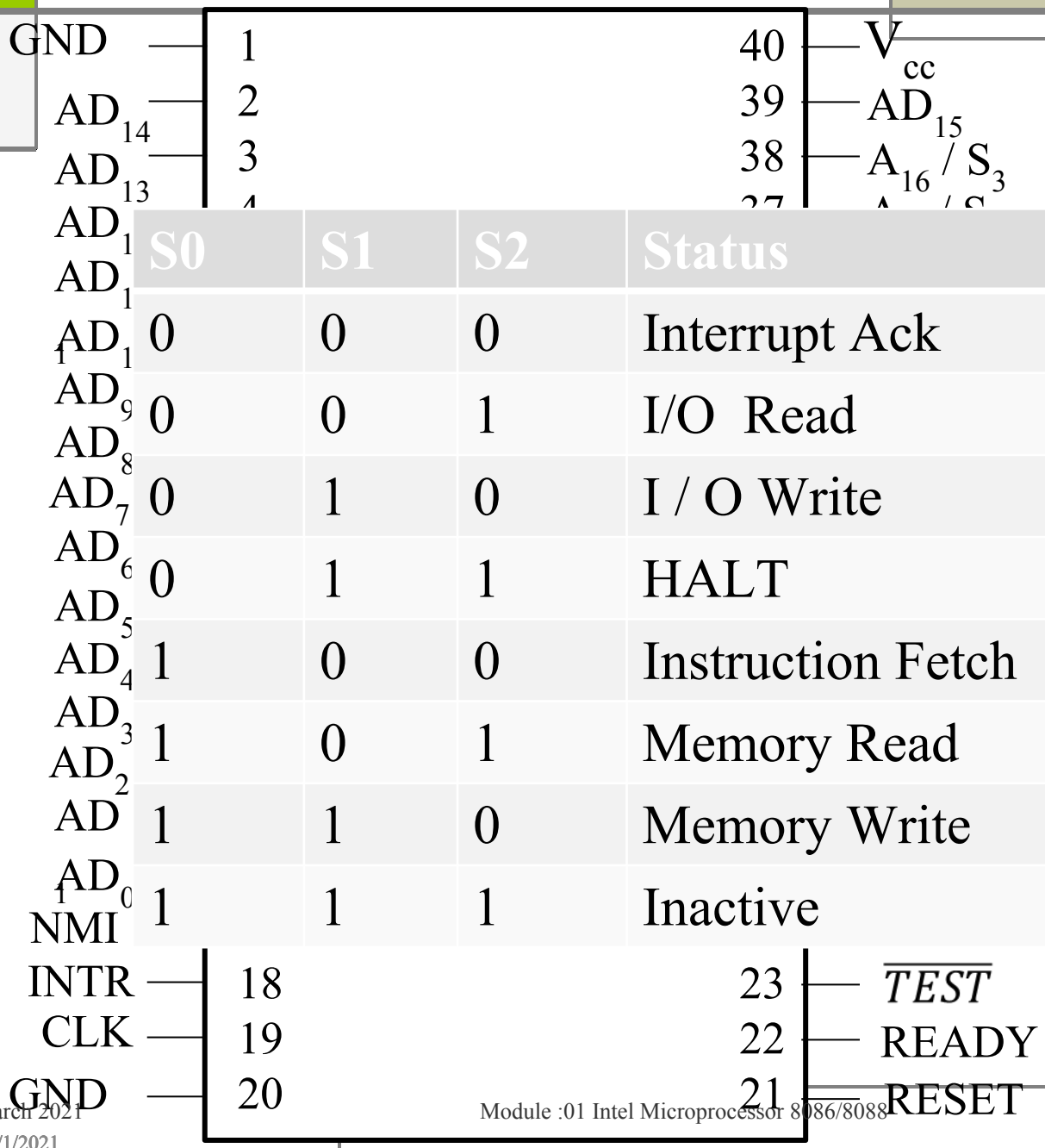
Carry Data

ALE =1

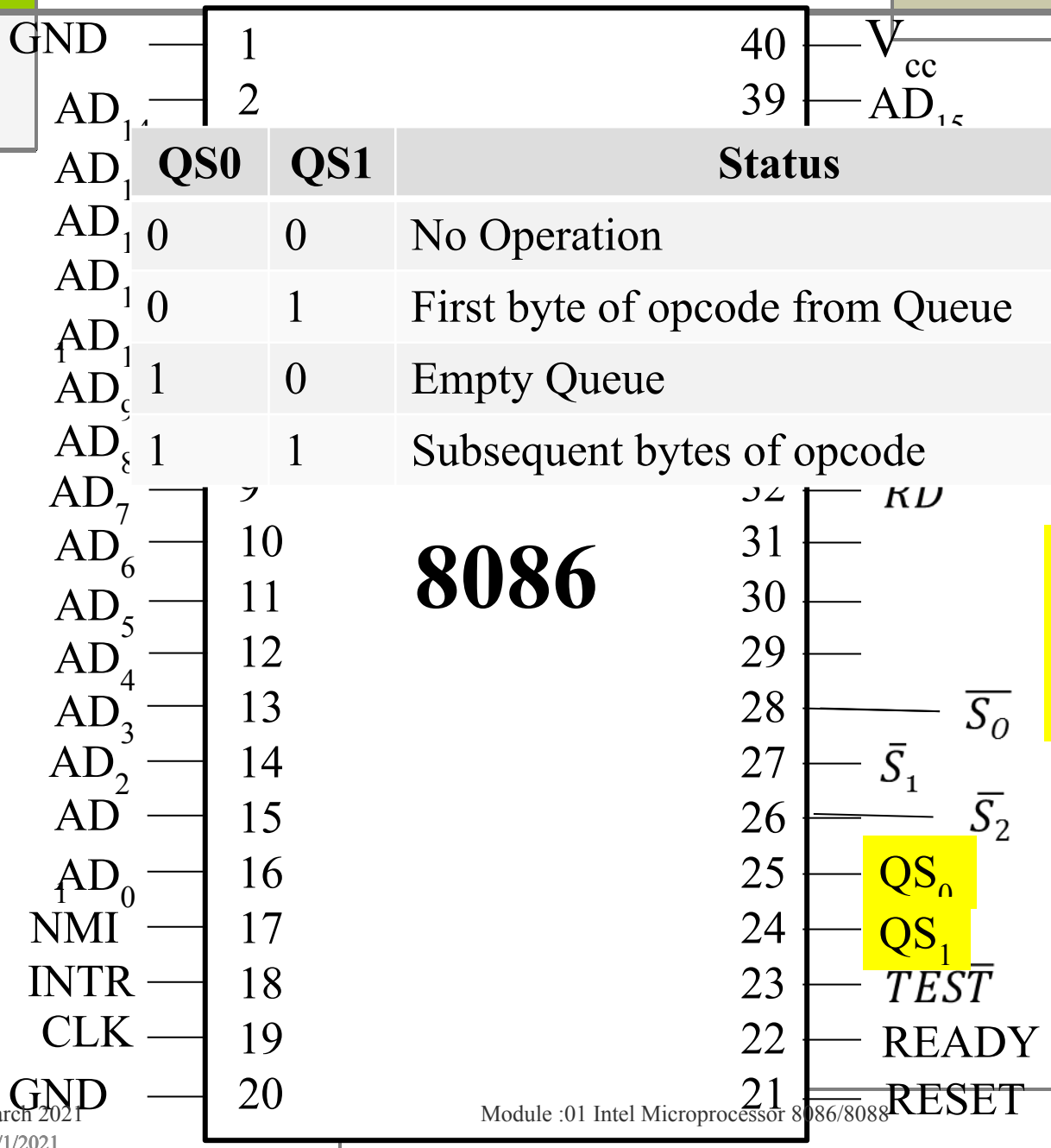
Carry Address


8086
GND 1
AD14 2
AD13 3
AD12 4
AD1 5
AD10 6
AD9 7
AD8 8
AD7 9
AD6 10
AD5 11
AD4 12
AD3 13
AD2 14
AD 15
AD0 16
NMI 17
INTR 18
CLK 19
GND 20
40 Vcc
39 AD15
38 A16 / S3
37 A17 / S4
36 A18 / S5
35 A19 / S6
34 $\overline{BHE}$ / S7
33 MN / $\overline{MX}$
32 $\overline{RD}$
31 HOLD
30 HLDA
29 $\overline{WR}$
28 M / $\overline{IO}$
27 DT / $\overline{R}$
26 $\overline{DEN}$
25 ALE
24 $\overline{INTA}$
23 TEST
22 READY
21 RESET

Maximum Mode

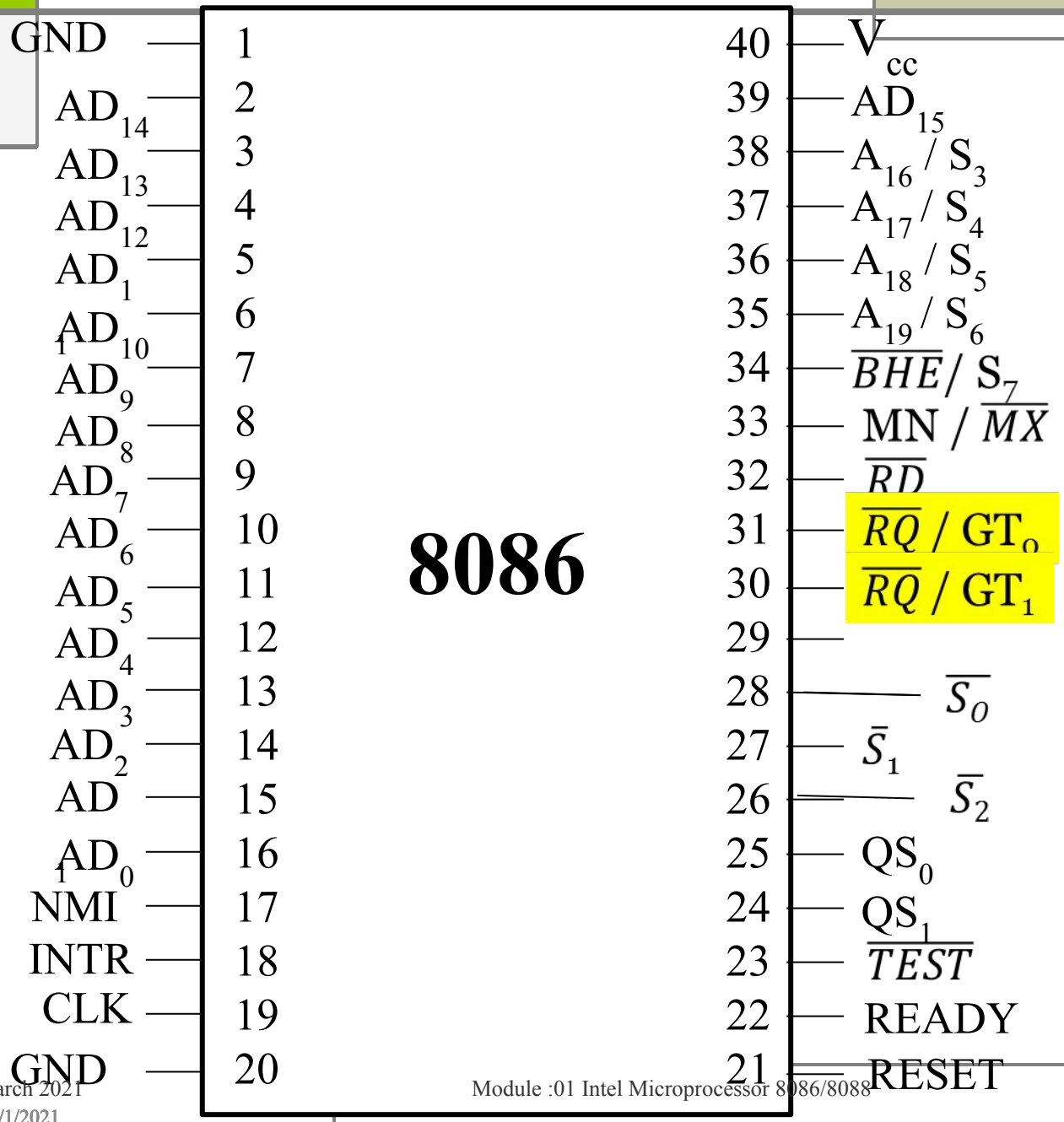


S0	S1	S2	Status
0	0	0	Interrupt Ack
0	0	1	I/O Read
0	1	0	I / O Write
0	1	1	HALT
1	0	0	Instruction Fetch
1	0	1	Memory Read
1	1	0	Memory Write
1	1	1	Inactive



QS0	QS1	Status
0	0	No Operation
0	1	First byte of opcode from Queue
1	0	Empty Queue
1	1	Subsequent bytes of opcode

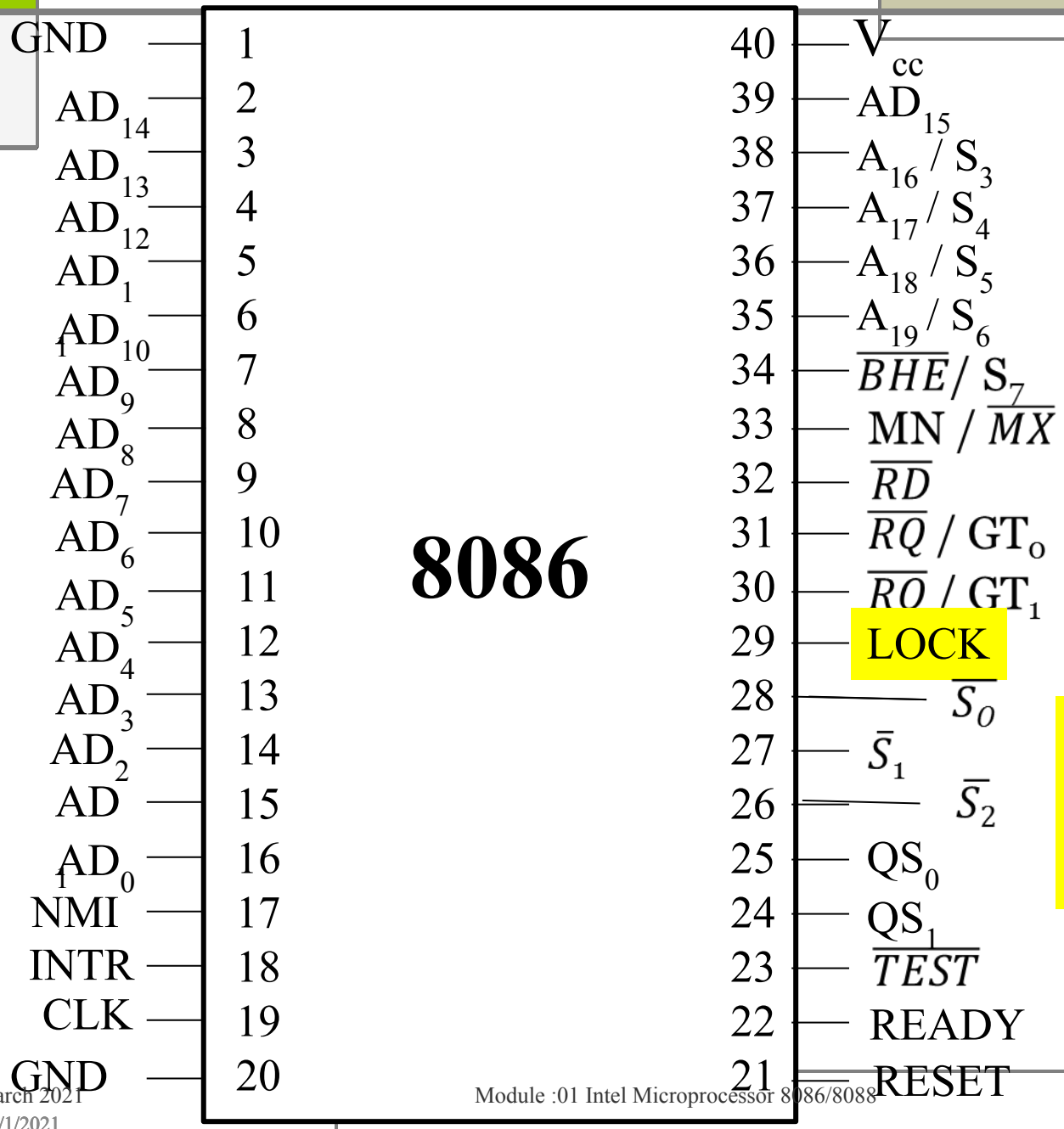
Instruction Queue Status pins



Signals for resource sharing between processors .

RQ – Request

GT - Grant



Lock the peripherals

Question

1. Draw and explain the Pin configuration of Intel 8086 chip?
2. Write a short note on the following pins:-
   (a) ALE
   (b) HOLD and HLDA
   (c) BHE
   (d) LOCK
   (e) DEN
   (f) DT/R
   (g) NMI

Are 8086 and 8088 different?

Features	8086	8088
Address Line	20-bits	**20-bits**
Data Line	16-bits	**8-bits**
ALU	16-bits	**16-bits**
Year of Invention	1976	**1979**

Story of 8088

In 1977

I want to make personal computers. Design a μp that is cheap & efficient

How about 8086? A 16-bit mp with 1MB

intel

How about 68000? A 16-bit mp with 24 MB

Motorola Inc.

That's great!
But all peripherals currently available are only 8-bit

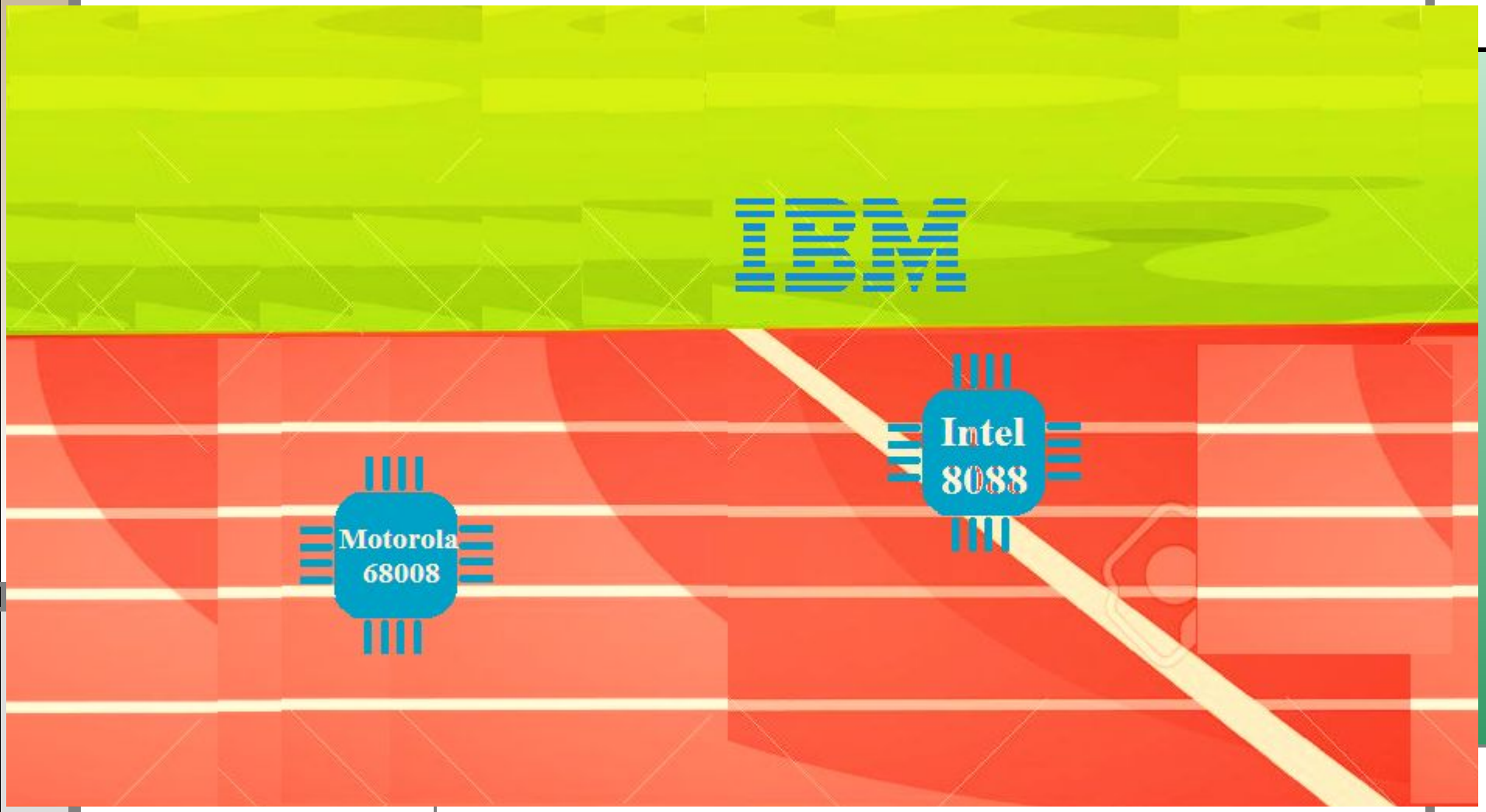
No problem.
I can redesign 8086 to work with 8-bit peripherals

intel

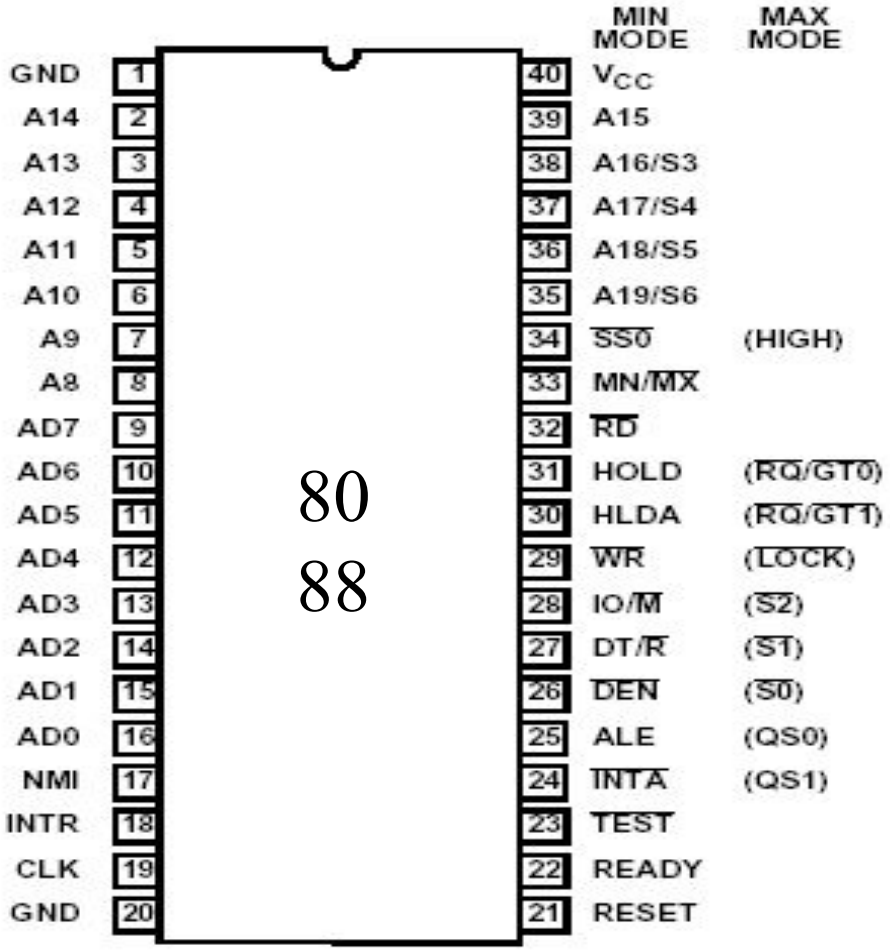
Me too!

Motorola Inc.

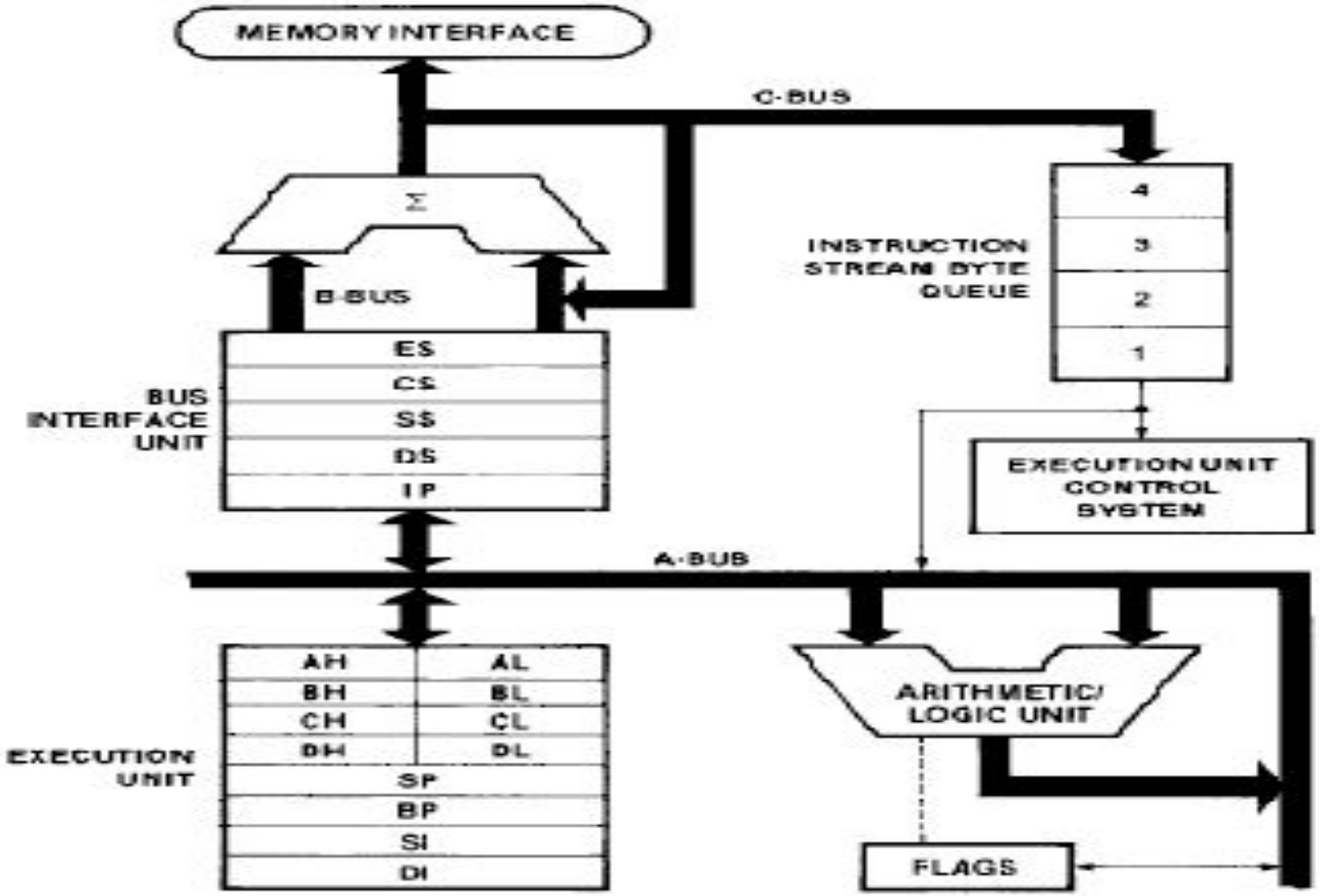
& Intel won the race with Intel-8088



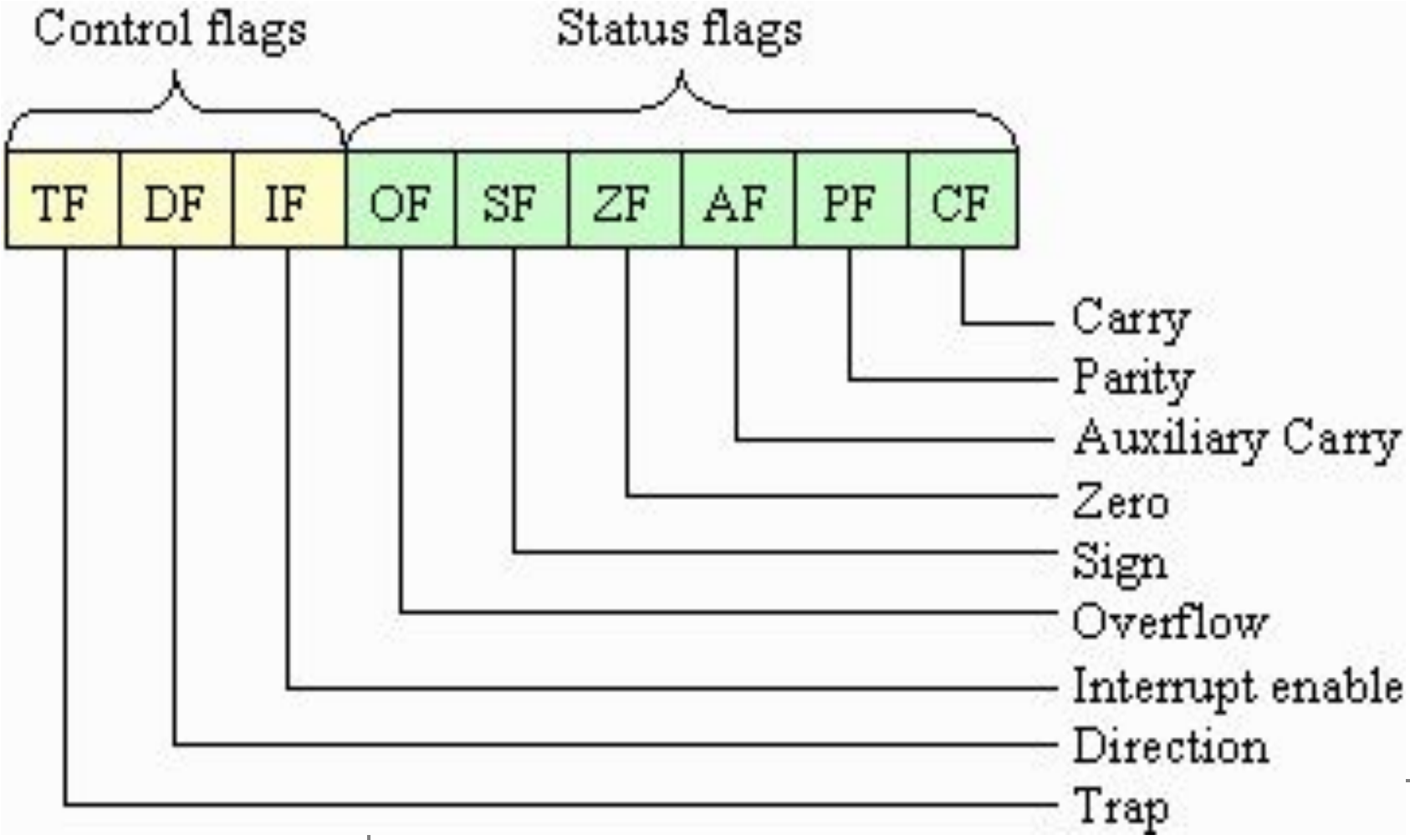
Pin Diagram of 8088



8088 Architecture



8088 Flag Register



Compare 8086 & 8088

8086	8088
16 bit microprocessor.	16 bit microprocessor.
It has 16 bit data bus.	It has 8 bit data bus.
It has 16 bit ALU.	It has 16 bit ALU.
Requires memory banking to transfer 16-bit data at a time. (BHE pin)	Does not require memory banking as it has 8-bit data bus. (No BHE)
Performs faster memory operations as it can transfer 16 bits in one cycle.	Performs slower memory operations as it can transfer only 8 bits in one cycle.

8086 v/s 8088...

8086	8088
Has a 6 byte pre-fetch queue for pipelining.	Has a 4 byte pre-fetch queue for pipelining.
Has an M/ pin to differentiate between memory and I/O operations.	Has an IO/ pin to differentiate between memory and I/O operations.
BIU will fetch new bytes into the pipelining queue when 2 bytes of the queue are empty.	BIU will fetch a new byte into the pipelining queue when 1 byte of the queue is empty.
Has 9 flags.	Has 9 flags.
Supports pipeline architecture.	Supports pipeline architecture.

Cold starting of 8086

- When power is first supplied to 8086, all its internal registers contain random data. This also applies to RAM
- Because of this, the processor may fetch its first instruction from any location leading to an unpredictable result
- Thus some mechanism must be provided for gaining control of machine when it is started and this is done by **RESET** input

RESET

CPU	Content
Flags	0000H
CS	FFFFH
DS	0000H
SS	0000H
ES	0000H
Queue	Empty

After RESET

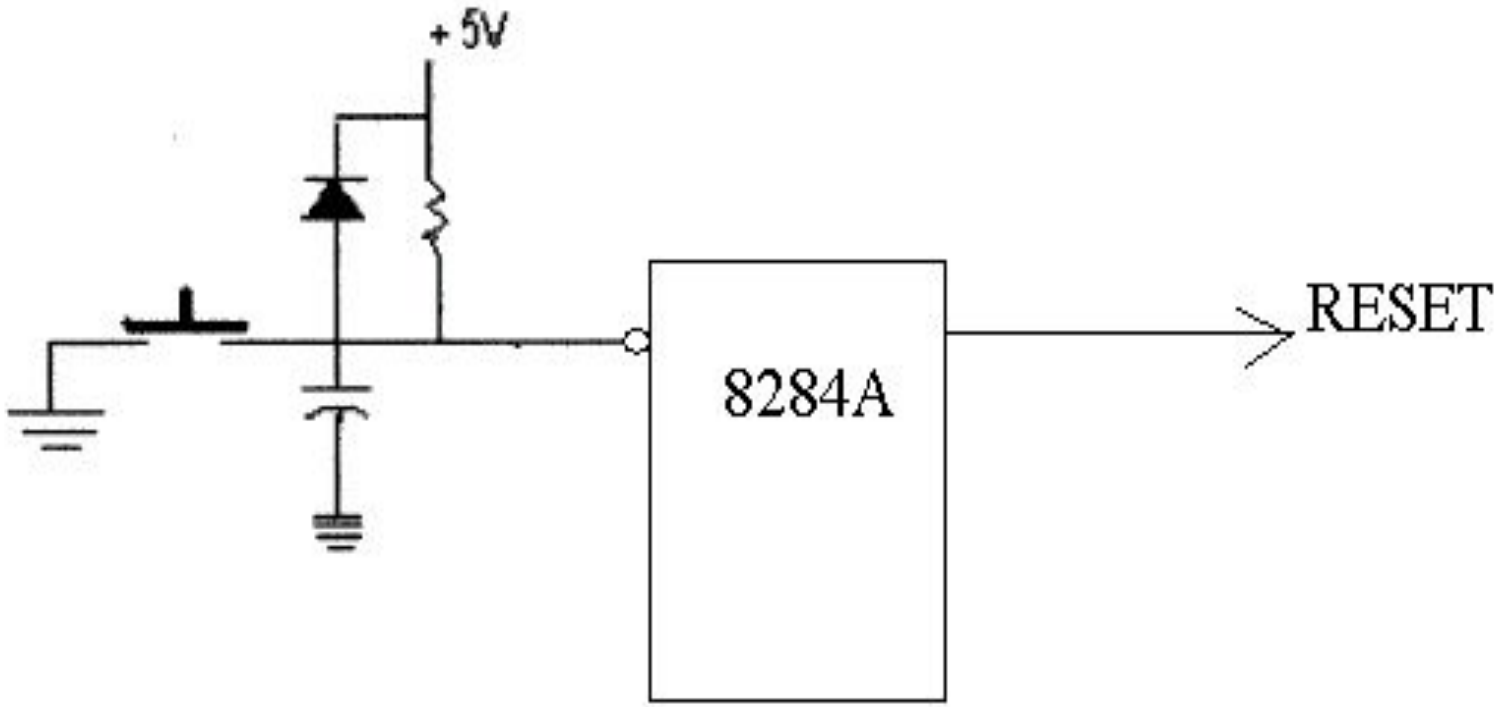
- The CPU will fetch its first instruction from FFFF0H
- RESET thus solves the random start problem associated with cold starting.
- However, we must ensure that a useful program resides in memory beginning at FFFF0H

@ FFFF0hlocation

- RAM cannot be used as this address and solution is to map ROM to this address space.
- Only 16 bytes are available, so the instruction at FFFF0H has to be a JUMP to some other location where a longer program can reside.
- This is a program that initializes computer and then loads a more complex program (OS) into RAM.

Generating the RESET?

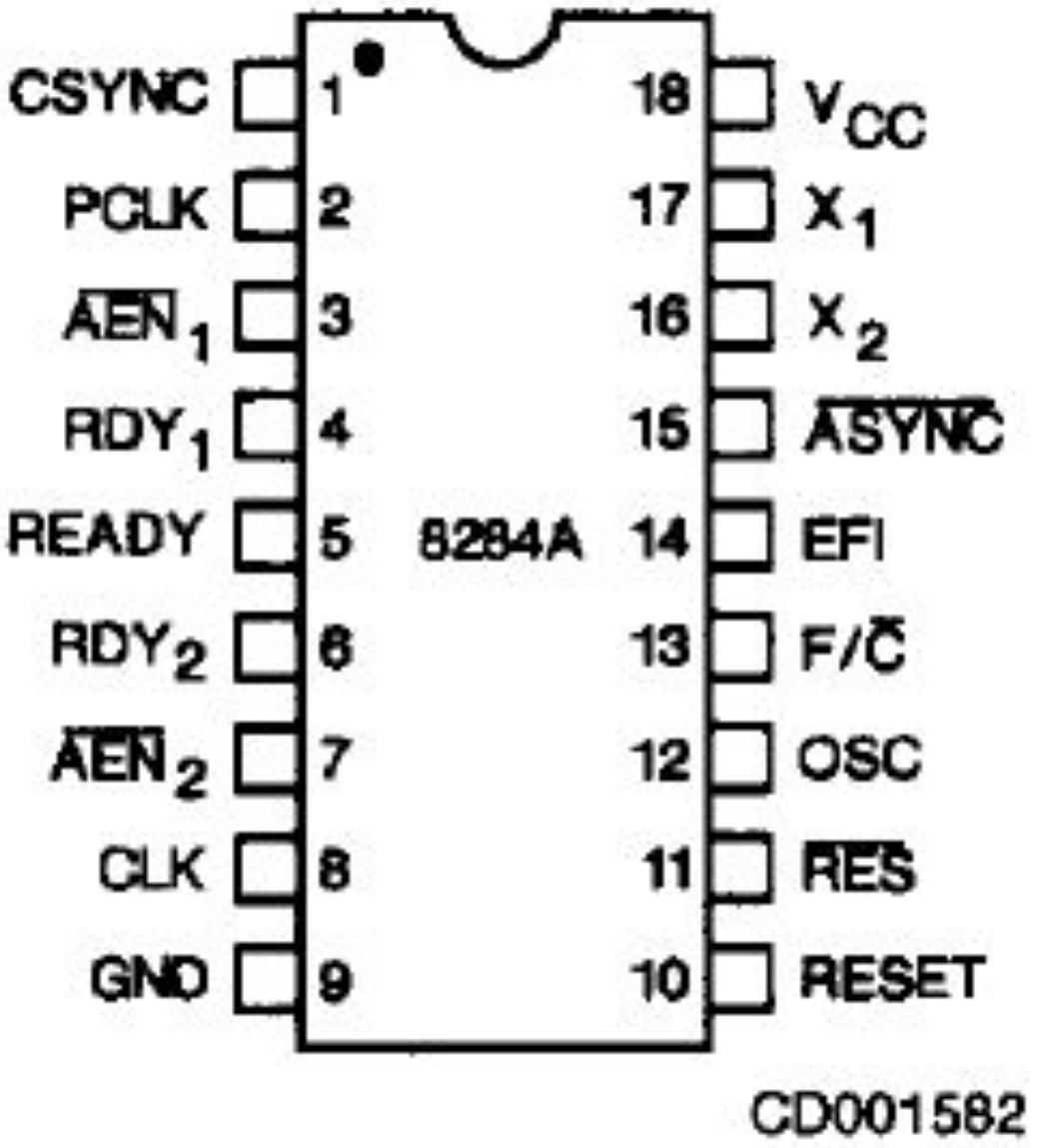
Resetting with 8284



Reset Signal

- This circuit allows the processor to be reset in two ways
  - One way is to depress the switch. The reset will be held high for an integral number of clock pulses until the switch is released
  - Second way is by providing power-on reset function. When power is first applied to the system, the capacitor will charge toward +5V. With sufficient time constant, the $\overline{RES}$ input will be held low long enough to reset the processor

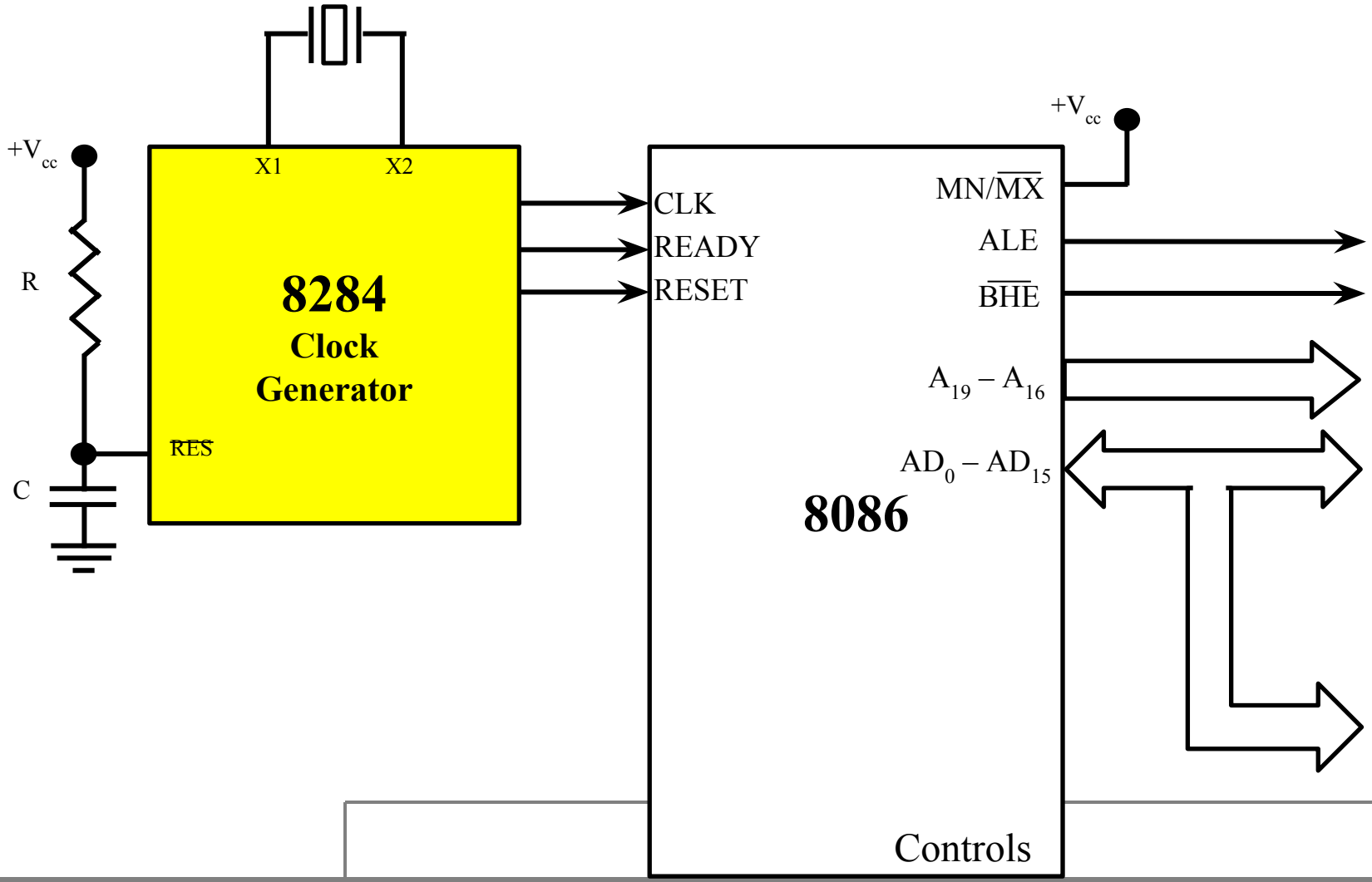
8284 Pin Diagram



8284 Clock Generator

- The clock source can be an internal crystal oscillator $F/\overline{C} = 0$ or an external frequency input $F/\overline{C} = 1$ applied to the EFI pin.
- 8284 has three frequency outputs:
  - OSC with frequency = crystal frequency
  - PCLK with frequency = crystal frequency/2 and 50% duty cycle
  - CLK with frequency = crystal frequency/3 and 33% duty cycle

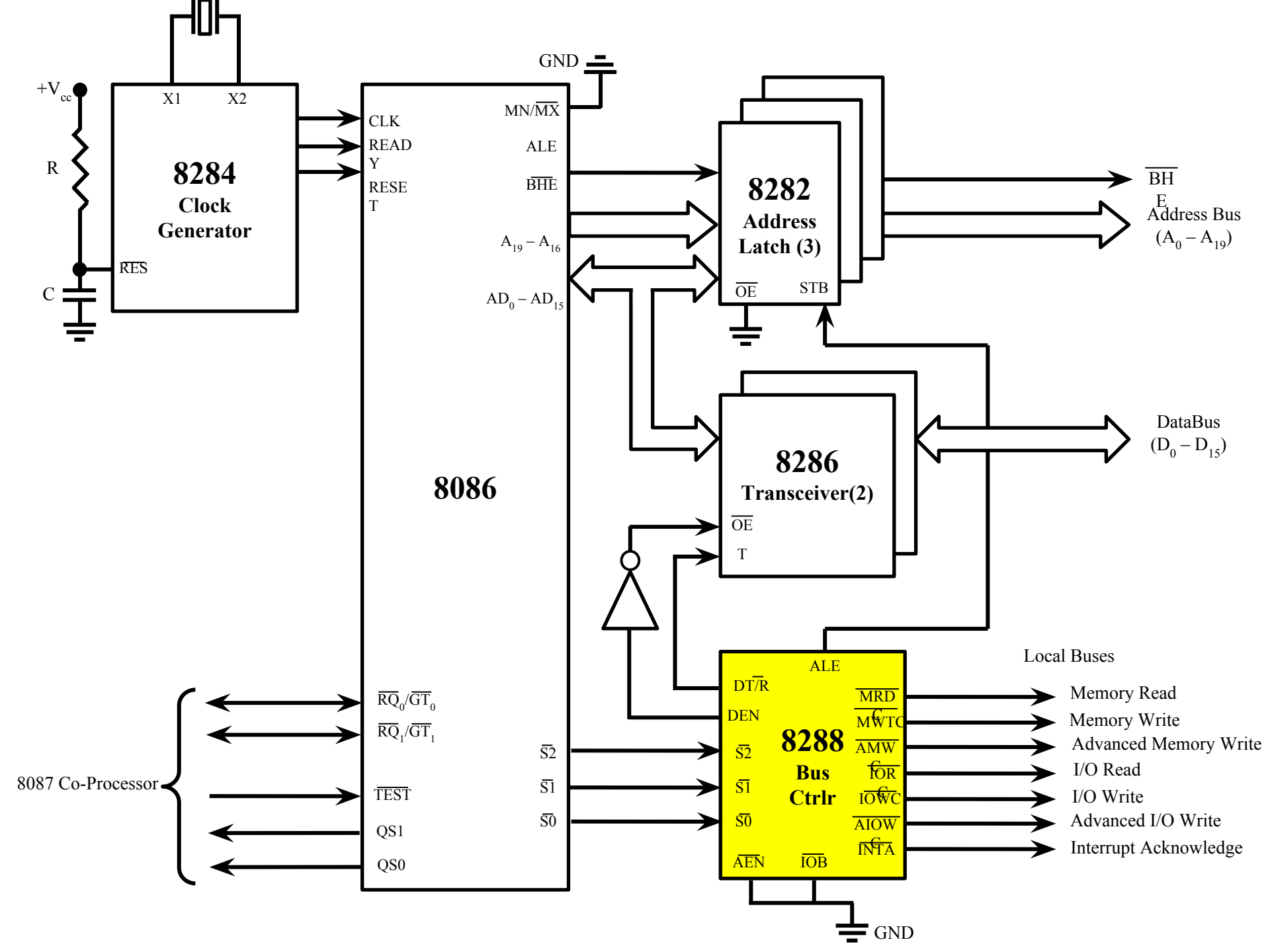
8284 with 8086



Questions

1. Explain the architecture of 8088 with neat block diagram.
2. Write a short note on Intel 8284 Clock Generator

Study of 8288 Bus Controller


+Vcc
R
C
X1
X2
8284
Clock
Generator
RES
CLK
READY
RESET
GND
MN/MX
ALE
BHE
A19 – A16
AD0 – AD15
8086
8282
Address
Latch (3)
OE
STB
BHE
Address Bus
(A0 – A19)
8286
Transceiver(2)
OE
T
DataBus
(D0 – D15)
RQ0/GT0
RQ1/GT1
TEST
QS1
QS0
8087 Co-Processor
S2
S1
S0
8288
Bus
Ctrlr
ALE
DT/R
DEN
S2
S1
S0
AEN
IOB
MRDC
MWTC
AMWC
IORC
IOWC
AIOW
INTA
Local Buses
Memory Read
Memory Write
Advanced Memory Write
I/O Read
I/O Write
Advanced I/O Write
Interrupt Acknowledge
GND

8288 – Bus Controller

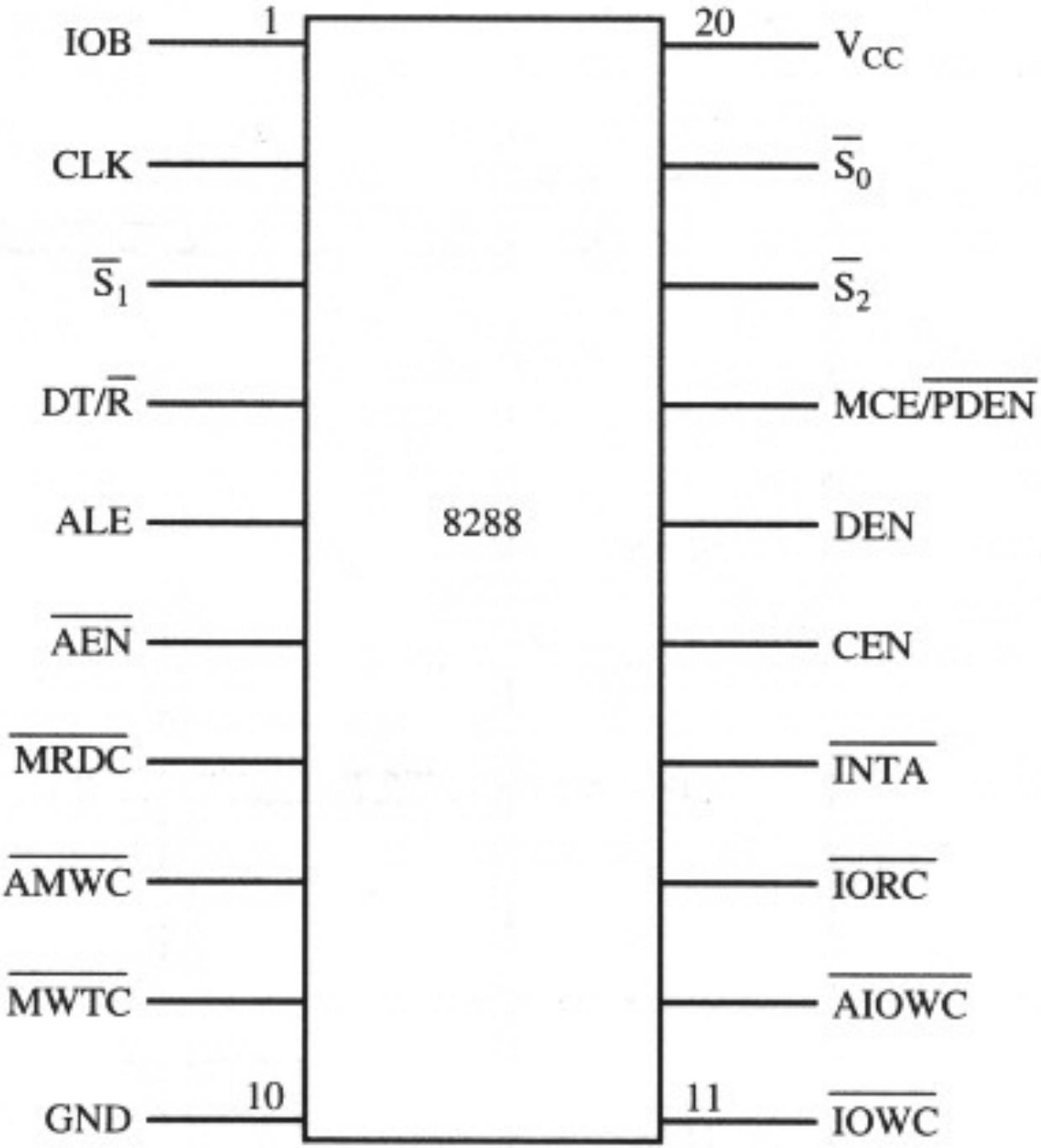
- Intel 8288 is a bus controller that decodes the maximum mode encoded bus control signal outputs to individual bus control signals
- S0, S1 and S2 are connected from 8086 and decoded by the 8282 bus controller
- The individual decoded bus control signals are used to control 8282 latch, 8286 transceiver and memory devices on the bus

8288 – inputs & outputs

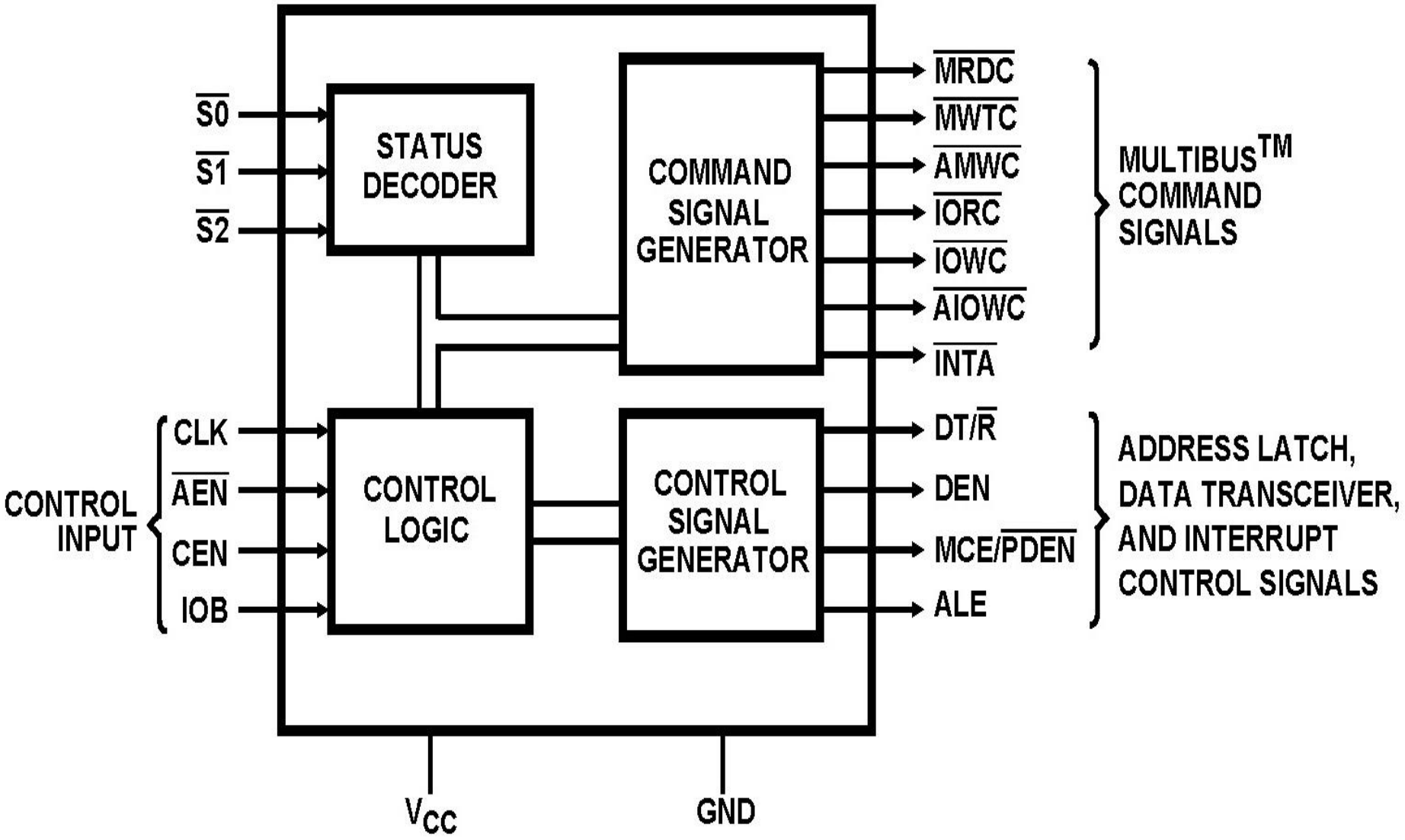
$\overline{S_2}$	$\overline{S_1}$	$\overline{S_0}$	**Processor State**	**8288 Active Output**
0	0	0	Interrupt Acknowledge	$\overline{INTA}$
0	0	1	Read I/O Port	$\overline{IORC}$
0	1	0	Write I/O Port	$\overline{IOWC}\,(\overline{AIOWC})$
0	1	1	Halt	**NONE**
1	0	0	Code Access	$\overline{MRDC}$
1	0	1	Read Memory	$\overline{MRDC}$
1	1	0	Write Memory	$\overline{MWTC}\,(\overline{AMWTC})$
1	1	1	Passive	**NONE**

8288...

- The CLK input permits 8288 activities to be synchronized with that of a processor
- AIOWC and AMWTC output are enabled one clock cycle earlier than normal write control signals because some device require wider cycle

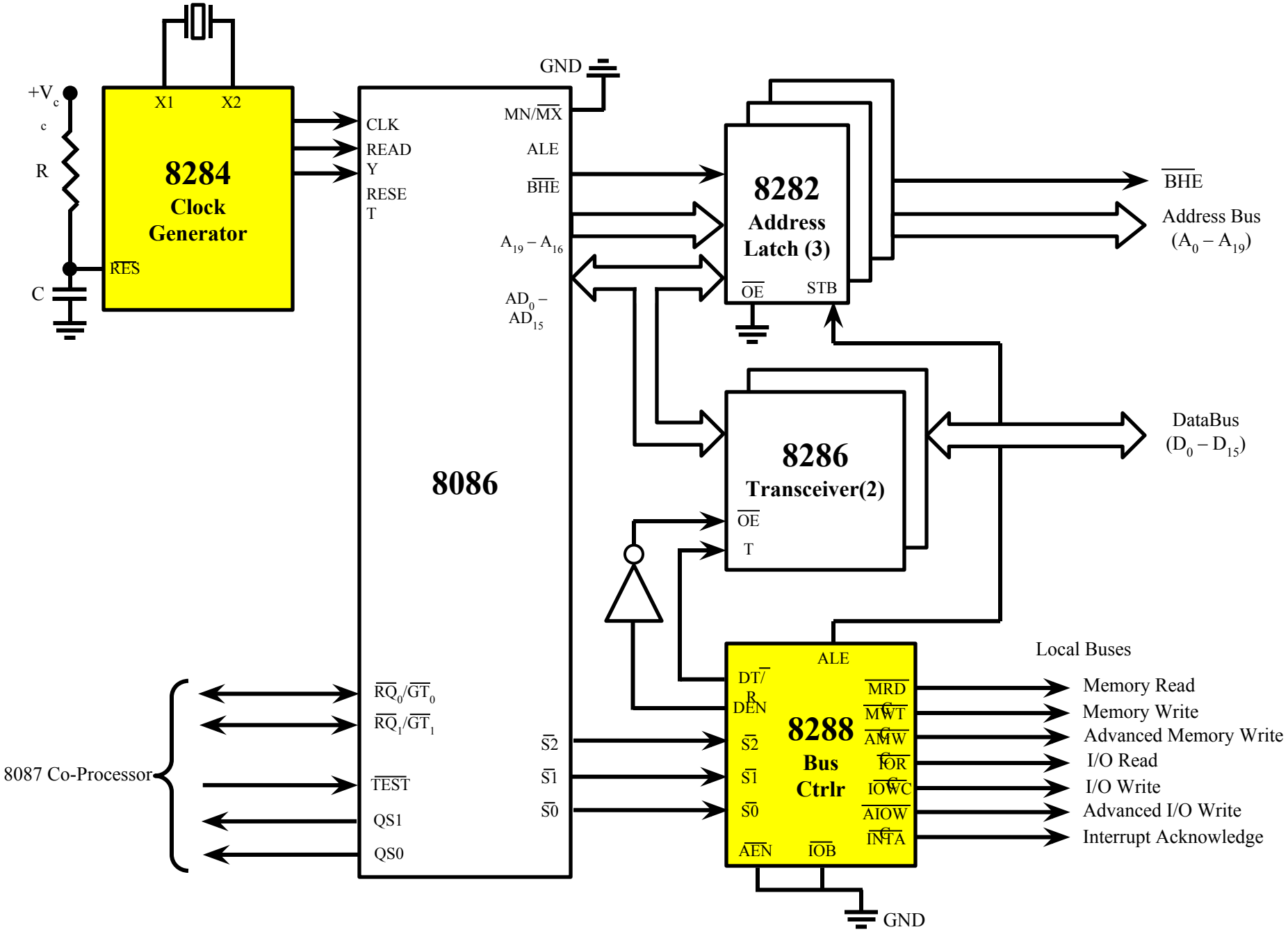

IOB
1
CLK
$\overline{S}_1$
DT/$\overline{R}$
ALE
$\overline{AEN}$
$\overline{MRDC}$
$\overline{AMWC}$
$\overline{MWTC}$
GND
10
8288
20
V_{CC}
$\overline{S}_0$
$\overline{S}_2$
MCE/$\overline{PDEN}$
DEN
CEN
$\overline{INTA}$
$\overline{IORC}$
$\overline{AIOWC}$
11
$\overline{IOWC}$

8288 – inside


S0
S1
S2
STATUS DECODER
COMMAND SIGNAL GENERATOR
MRDC
MWTC
AMWC
IORC
IOWC
AIOWC
INTA
MULTIBUS™ COMMAND SIGNALS
CONTROL INPUT
CLK
AEN
CEN
IOB
CONTROL LOGIC
CONTROL SIGNAL GENERATOR
DT/R
DEN
MCE/PDEN
ALE
ADDRESS LATCH, DATA TRANSCEIVER, AND INTERRUPT CONTROL SIGNALS
VCC
GND

8288 Pins

- ALE, DT/R and DEN pins are provided as output from 8288 to 8282 and 8286
- MCE/PDEN output depends on mode of 8288 (IO or not)
- CEN, IOB and AEN determine operating mode of 8288 – System bus or I/O bus
- When CEN (command enable) and IOB are high, 8288 operates in I/O bus mode. When CEN and AEN are low, it operates in System bus mode


+Vc
c
R
C
X1
X2
8284
Clock
Generator
RES
GND
CLK
READY
RESET
MN/MX
ALE
BHE
A19 – A16
AD0 –
AD15
8086
8282
Address
Latch (3)
OE
STB
BHE
Address Bus
(A0 – A19)
8286
Transceiver(2)
OE
T
DataBus
(D0 – D15)
RQ0/GT0
RQ1/GT1
8087 Co-Processor
TEST
QS1
QS0
S2
S1
S0
ALE
DT/R
DEN
8288
Bus
Ctrlr
S2
S1
S0
AEN
IOB
MRD
MWTC
AMWC
IOR
IOWC
AIOW
INTA
Local Buses
Memory Read
Memory Write
Advanced Memory Write
I/O Read
I/O Write
Advanced I/O Write
Interrupt Acknowledge
GND

8086 in minimum mode

Minimum Mode

- Single Processor Mode
- The Processor is in control of all the three buses – address, data and control.

Multiplexed Pins

- Multiplexed pins perform different functions at different time intervals
- These functions will never be required by the µp or its peripherals simultaneously.
- E.g., Address and Data pins are multiplexed.
- The µp first sends out address, and then from that location/ to that location, the µp sends/ receives data.
- Same pins act as address lines in one time state (T1), and data lines in another time state (T3)

Time in µP

- T-state is the smallest unit of time in a µp
- 1 clock cycle = 1 T-state
- In 8086, 1 machine cycle = 4 T-states
- 1 machine cycle (or bus cycle) is the time required to
  - T1 – send out an address – on address bus
  - T2 – send out a signal (read/ write) – on control bus
  - T3 – read/ write data on that location – on data bus
  - T4 – release all buses
- 1 instruction cycle = *n* machine cycles
  (depends on the instruction)

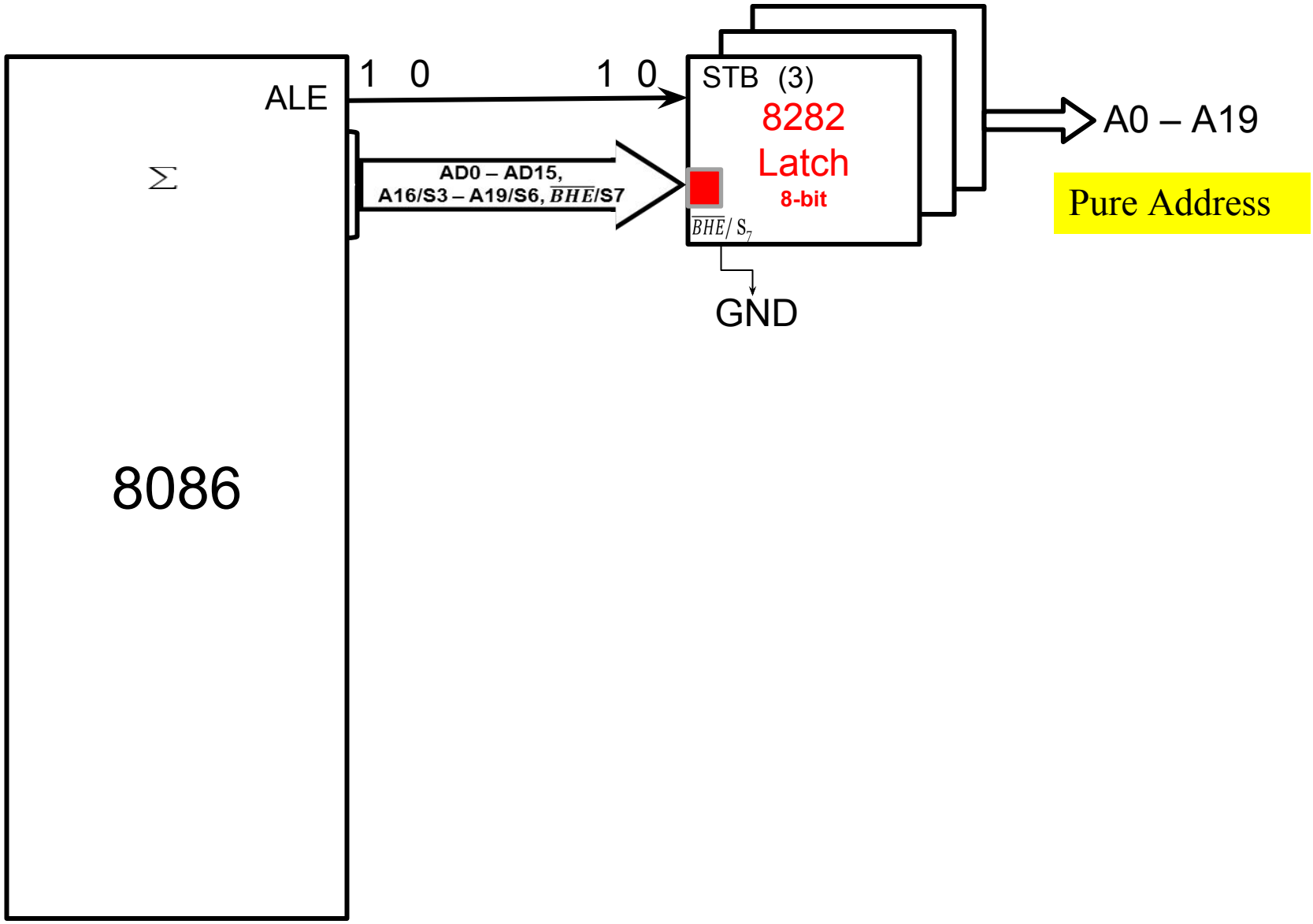
What is a “BUS” in real?

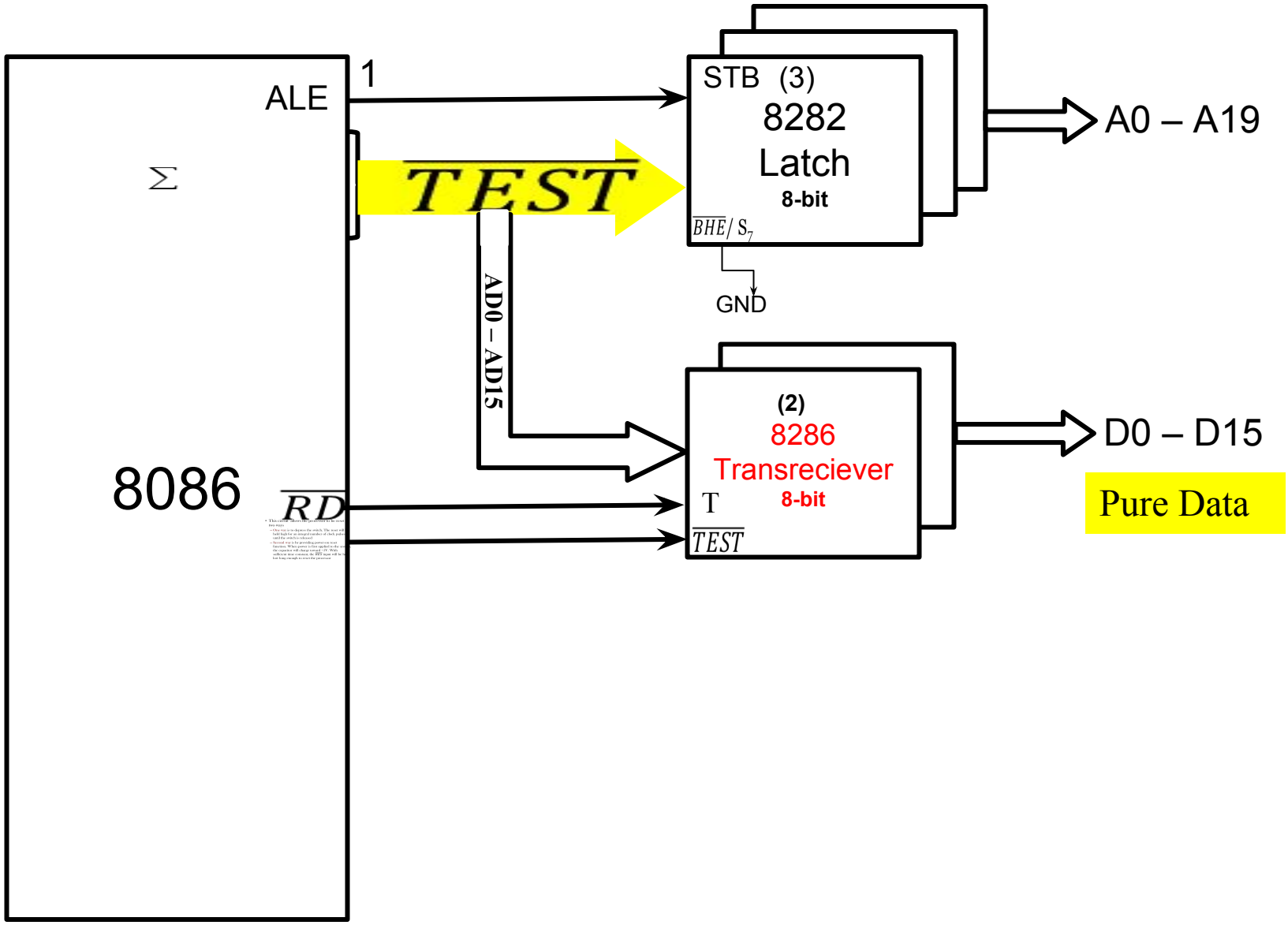
- Bus is a way of transport
- 20-bit Address Bus transports 20-bit Addresses
- 16-bit Data Bus transports 16-bit data
- Control Bus transports control signals

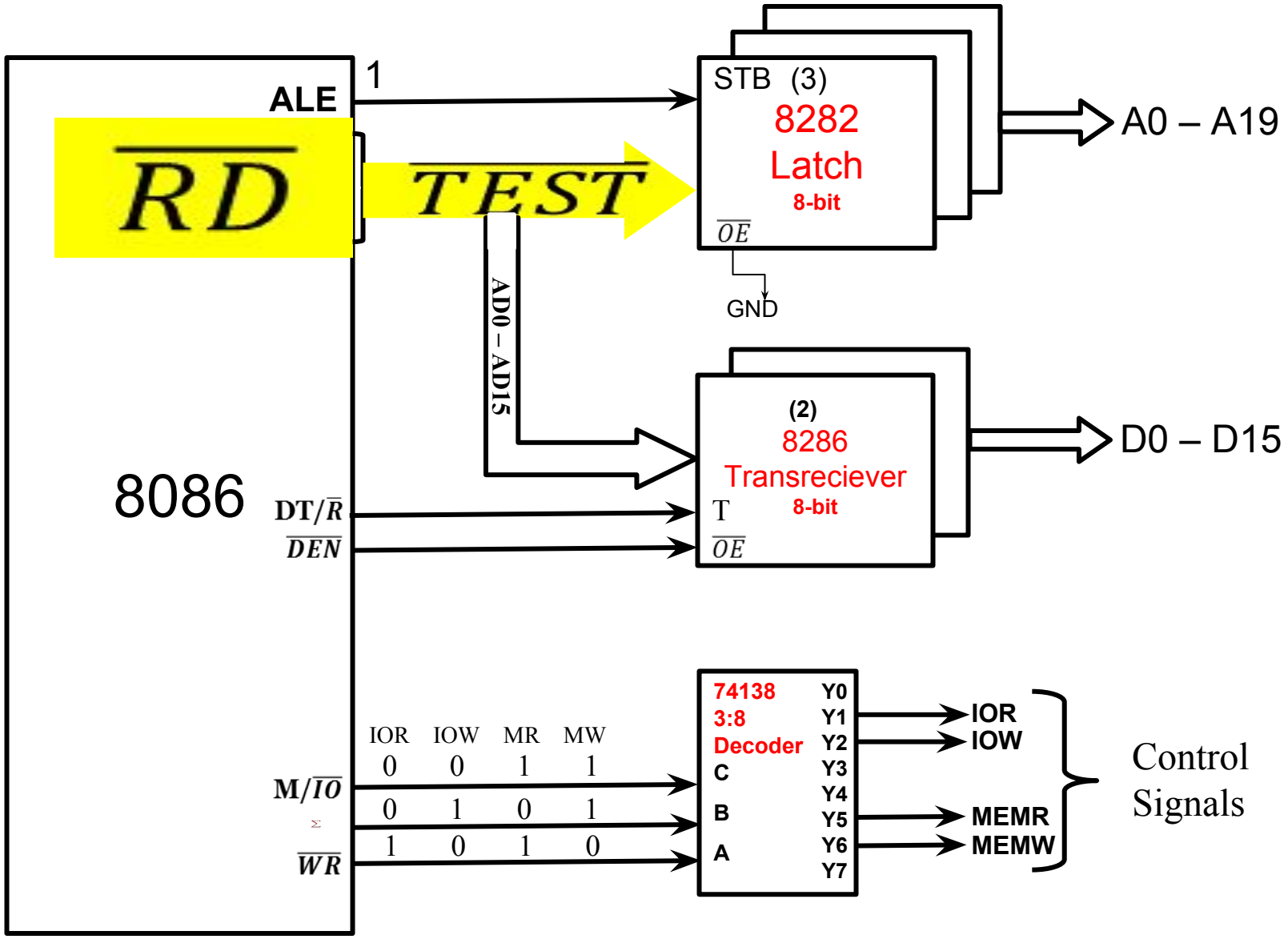
- The 8086 ALU sets the pins A0 – A19 to match the 20-bit address of the location it wants to access
- This value is not stored on these pins for a long time
- This value gets rewritten as D0 – D15 and status lines within the fraction of a second
- Where is A0 – A19 after that?
- In the address bus!
- But all these are given by the same set of pins
- How to get pure address from AD0 – AD15?

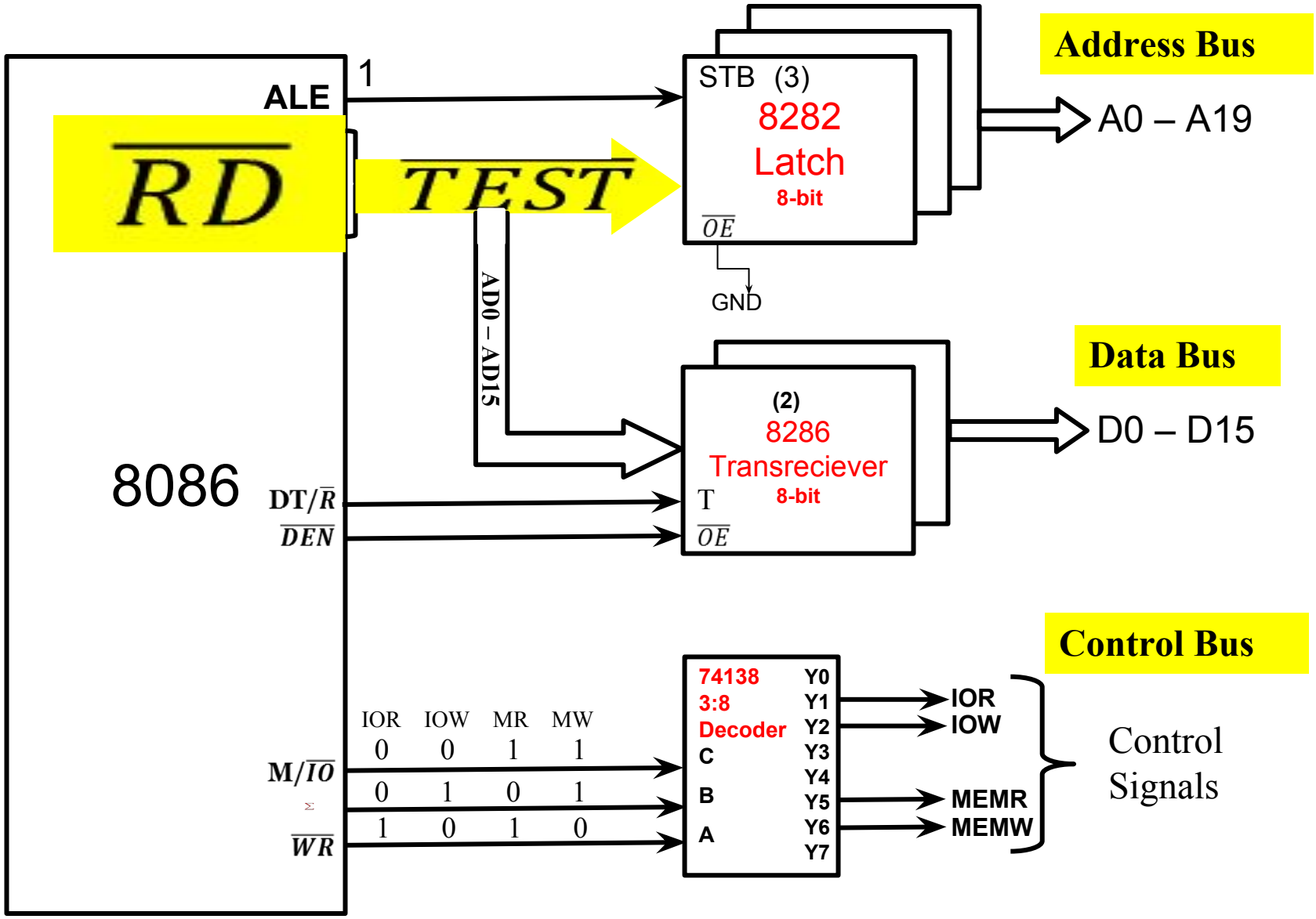
By using a LATCH

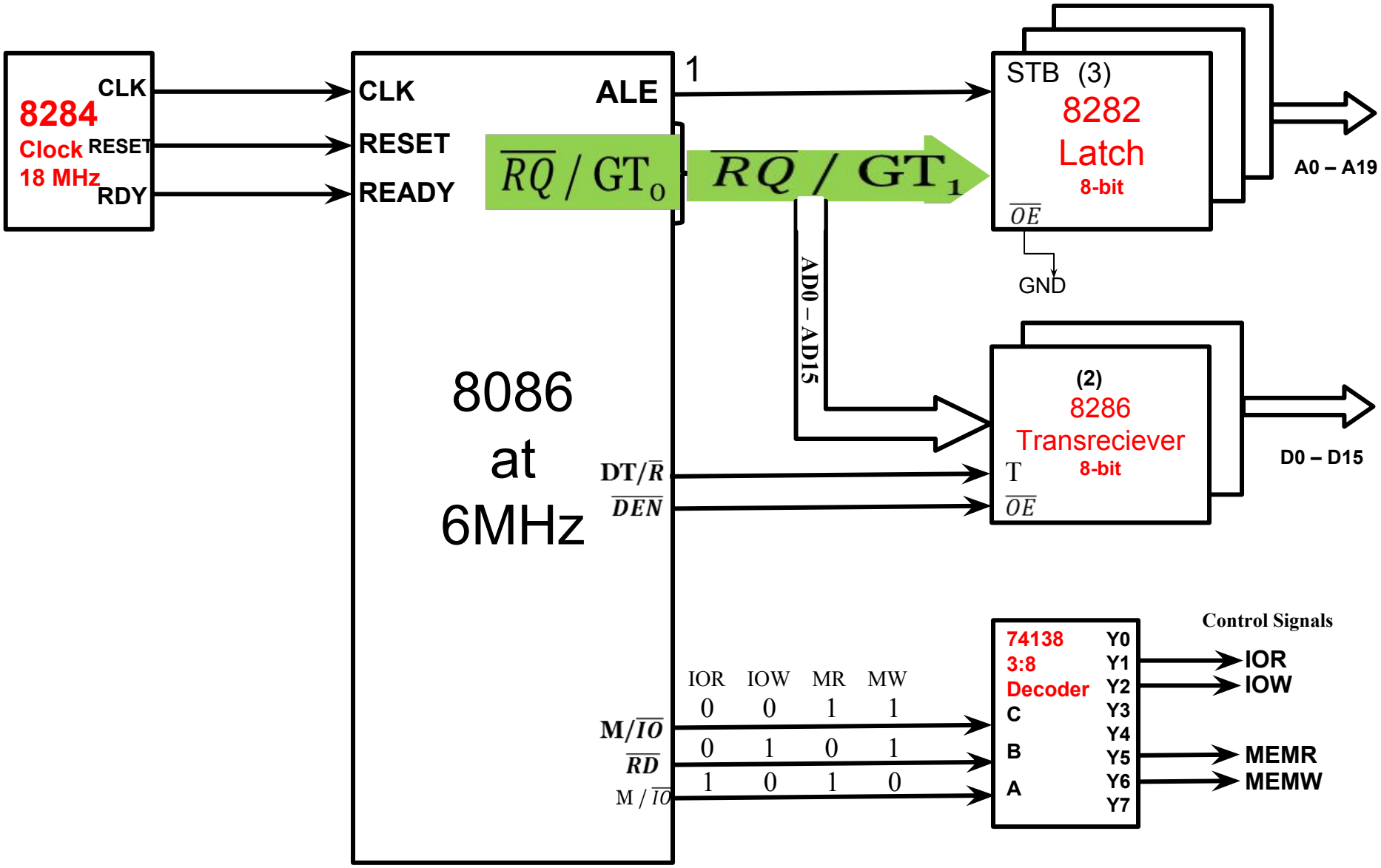
to **save only the address** from these pins,
and remain **cut-off when data is on** them.

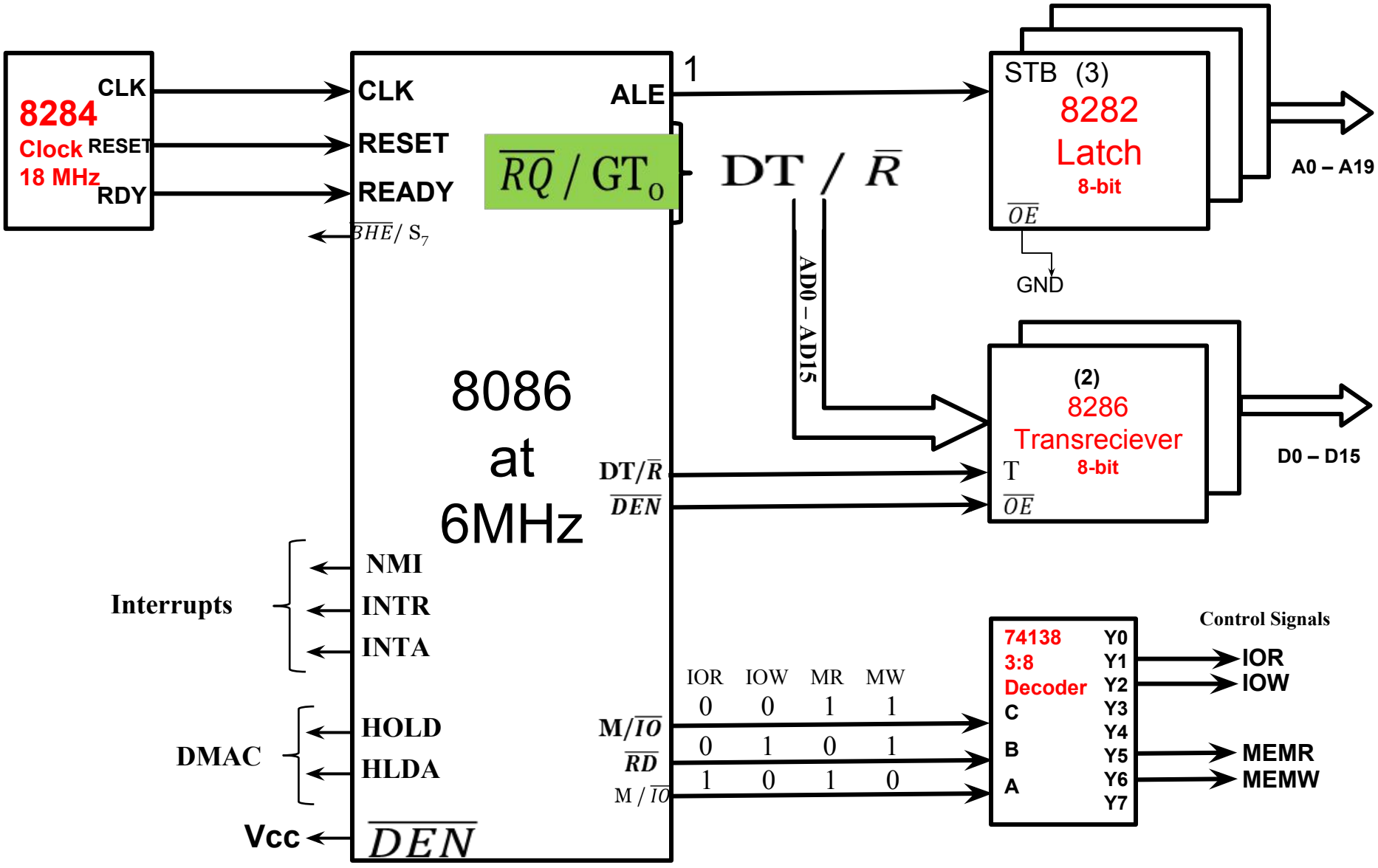

ALE
1 0
1 0
STB (3)
8282
Latch
8-bit
AD0 – AD15,
A16/S3 – A19/S6, BHE/S7
BHE/ S7
A0 – A19
Pure Address
GND
8086


ALE
1
Σ
TEST
STB (3)
8282
Latch
8-bit
BHE/ S7
GND
A0 – A19
AD0 – AD15
(2)
8286
Transreciever
8-bit
T
TEST
D0 – D15
Pure Data
8086
RD

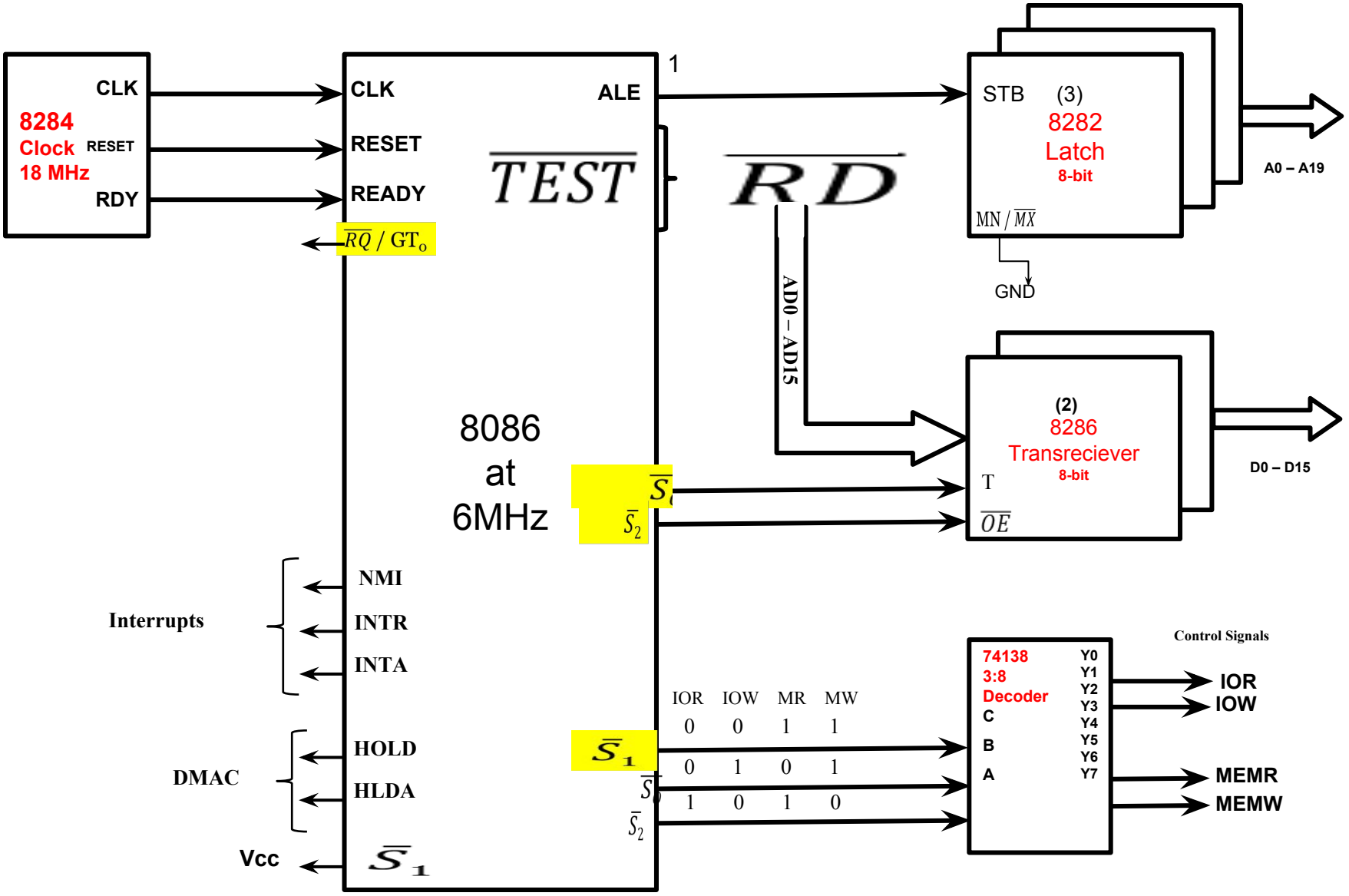

ALE
1
RD
TEST
STB (3)
8282
Latch
8-bit
OE
GND
A0 – A19
AD0 – AD15
(2)
8286
Transreciever
8-bit
T
OE
D0 – D15
8086
DT/R
DEN
M/IO
WR
IOR IOW MR MW
0 0 1 1
0 1 0 1
1 0 1 0
74138
3:8
Decoder
C
B
A
Y0
Y1
Y2
Y3
Y4
Y5
Y6
Y7
IOR
IOW
MEMR
MEMW
Control
Signals


Address Bus
ALE
1
STB (3)
8282
Latch
8-bit
OE
GND
A0 – A19
RD
TEST
AD0 – AD15
Data Bus
(2)
8286
Transreciever
8-bit
T
OE
D0 – D15
8086
DT/R
DEN
Control Bus
IOR IOW MR MW
0 0 1 1
0 1 0 1
1 0 1 0
M/IO
WR
74138
3:8
Decoder
C
B
A
Y0
Y1
Y2
Y3
Y4
Y5
Y6
Y7
IOR
IOW
MEMR
MEMW
Control
Signals

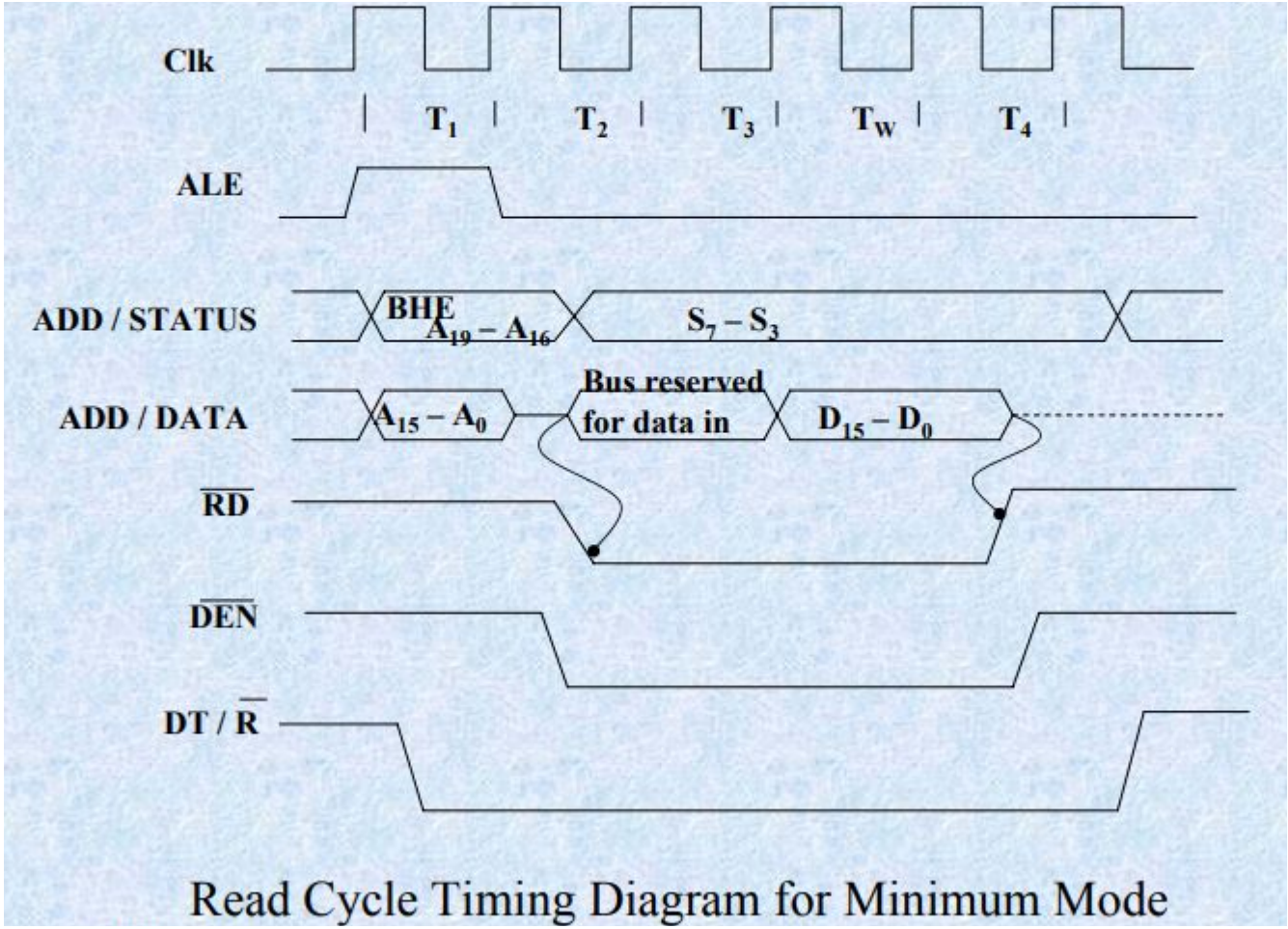

8284
Clock
18 MHz
CLK
RESET
RDY
CLK
RESET
READY
ALE
1
$\overline{RQ}$ / GT$_0$
$\overline{RQ}$ / GT$_1$
8086
at
6MHz
STB (3)
8282
Latch
8-bit
$\overline{OE}$
GND
A0 – A19
AD0 – AD15
(2)
8286
Transreciever
8-bit
T
$\overline{OE}$
D0 – D15
DT/$\overline{R}$
$\overline{DEN}$
M/$\overline{IO}$
$\overline{RD}$
M / $\overline{IO}$
IOR IOW MR MW
0 0 1 1
0 1 0 1
1 0 1 0
74138
3:8
Decoder
C
B
A
Y0
Y1
Y2
Y3
Y4
Y5
Y6
Y7
Control Signals
IOR
IOW
MEMR
MEMW


8284
Clock
18 MHz
CLK
RESET
RDY
CLK
RESET
READY
$\overline{BHE}$/ S_7
ALE
1
$\overline{RQ}$ / GT_0
DT / $\overline{R}$
8086
at
6MHz
$DT/\overline{R}$
$\overline{DEN}$
NMI
INTR
INTA
Interrupts
HOLD
HLDA
DMAC
Vcc
$\overline{DEN}$
M/$\overline{IO}$
$\overline{RD}$
M / $\overline{IO}$
IOR IOW MR MW
0 0 1 1
0 1 0 1
1 0 1 0
STB (3)
8282
Latch
8-bit
$\overline{OE}$
GND
A0 – A19
AD0 – AD15
(2)
8286
Transreciever
T
8-bit
$\overline{OE}$
D0 – D15
74138
3:8
Decoder
C
B
A
Y0
Y1
Y2
Y3
Y4
Y5
Y6
Y7
Control Signals
IOR
IOW
MEMR
MEMW

8086 in Minimum Mode!

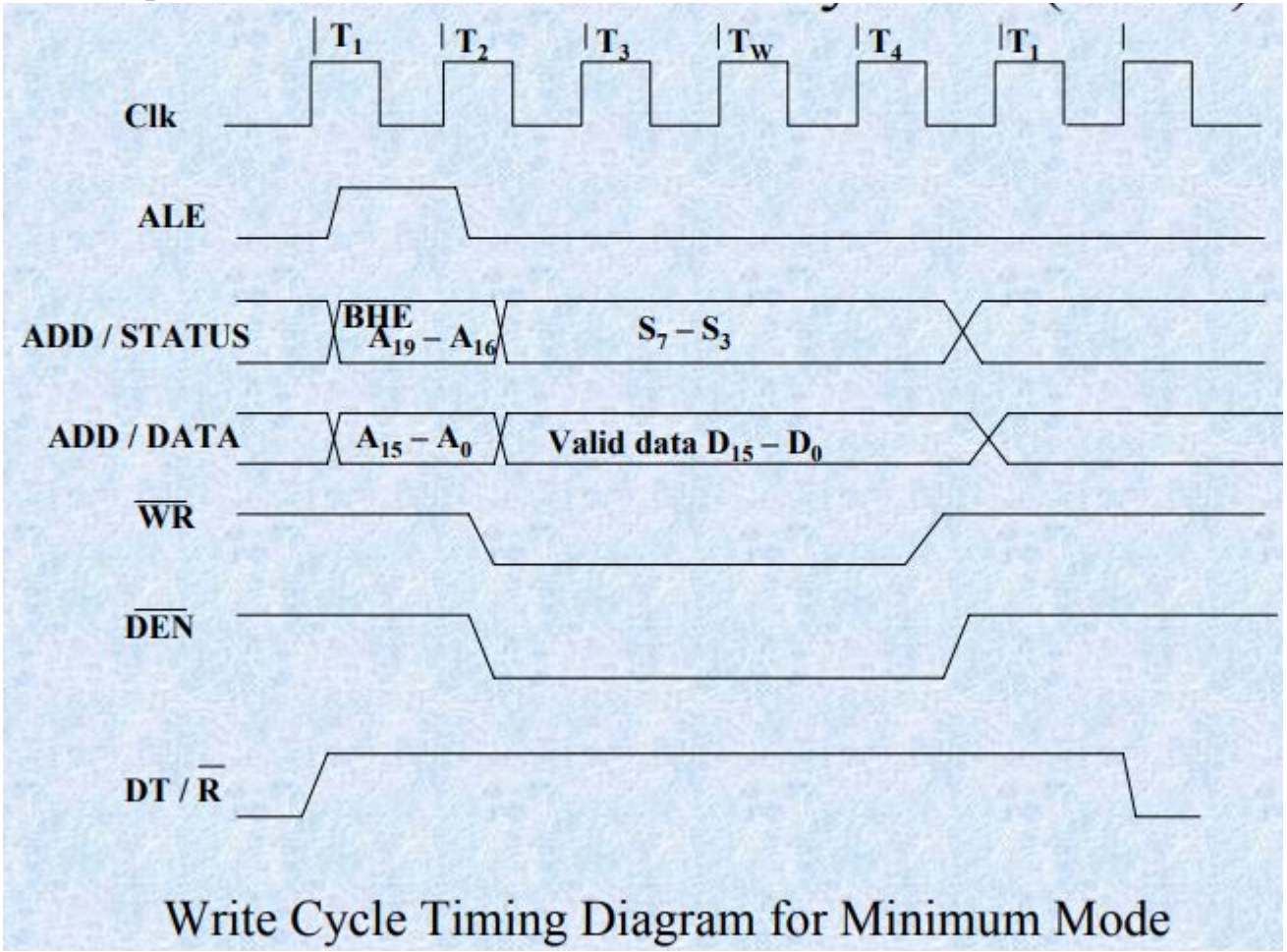


Timing diagrams for Read operation in minimum mode



Read Cycle Timing Diagram for Minimum Mode

Timing diagrams for Write operation in minimum mode



Write Cycle Timing Diagram for Minimum Mode

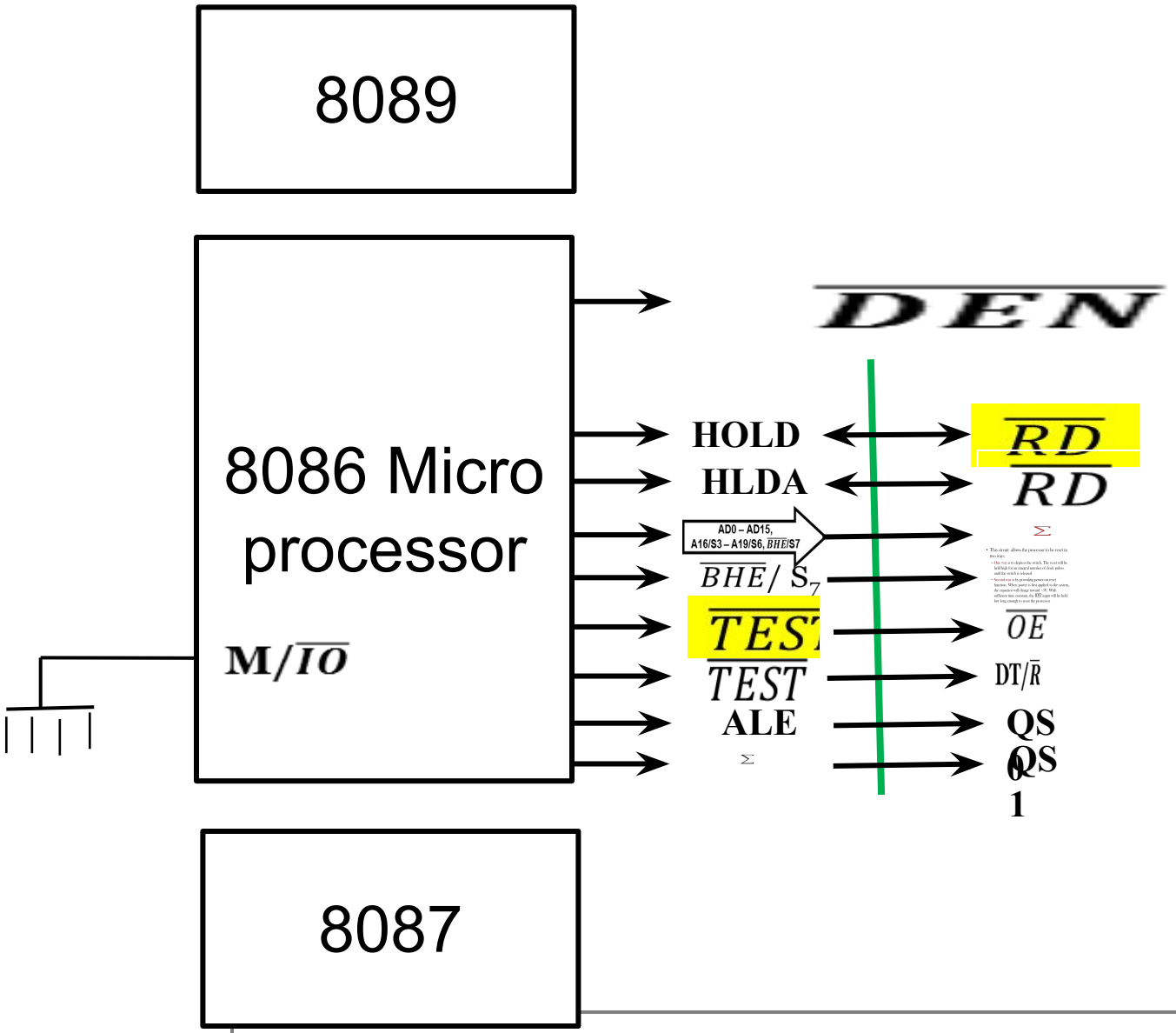
8086 in Maximum mode

Maximum Mode

- Multi Processor Mode
- In Maximum Mode, there are two more processors

  8087 (Math Co-processor)

  8089 (I/O Co-processor)


8089
8086 Micro processor
M/IO
8087
DEN
HOLD
HLDA
AD0 – AD15, A16/S3 – A19/S6, BHE/S7
BHE/ S7
TEST
ALE
RD
RD
OE
DT/R
QS
QS
1

Request and Grant for Processor 1

Request and Grant for Processor 2

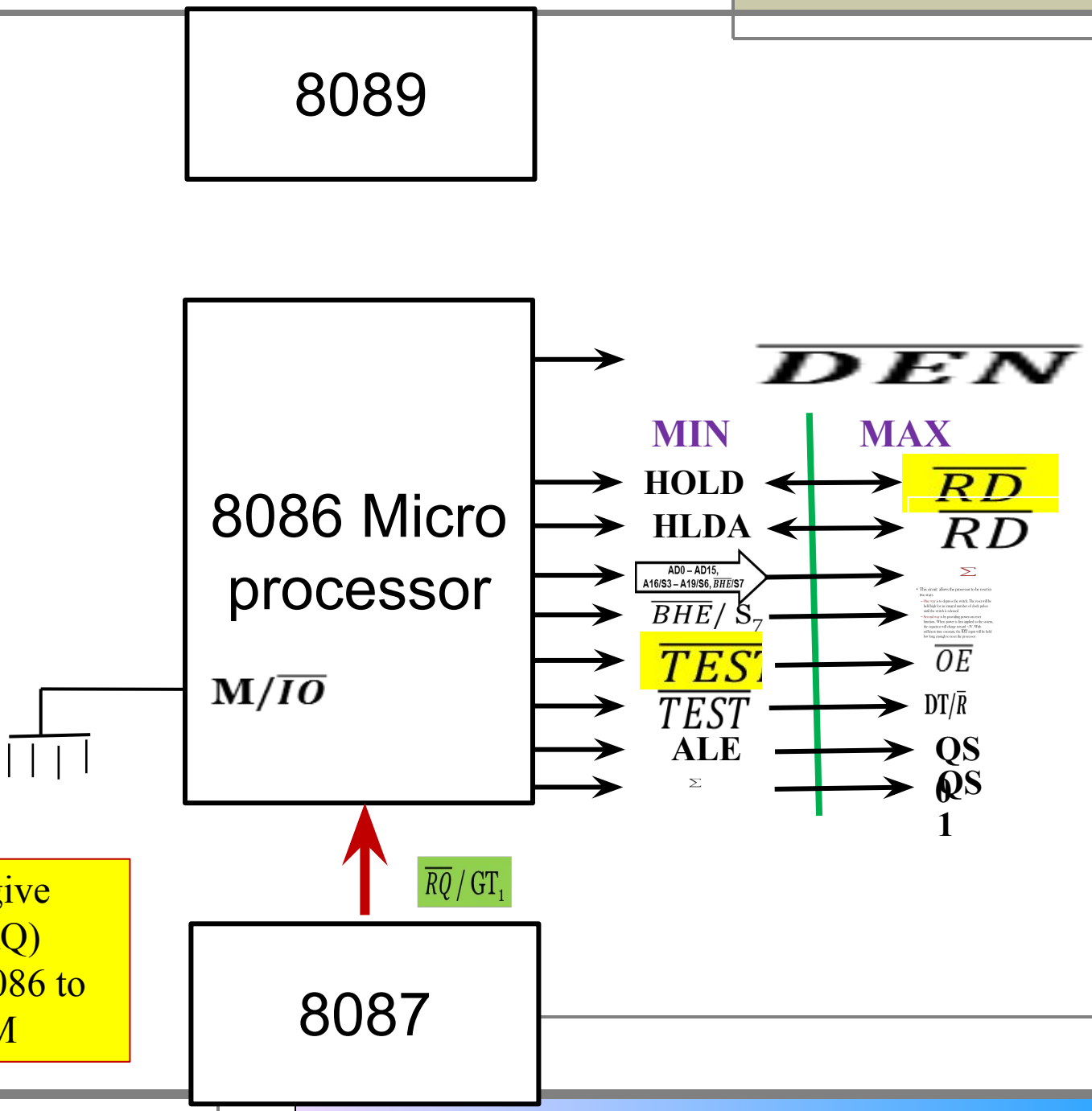
At a time, only one processor can become **Bus Master**.

(The processor who controls the bus is called Bus Master)

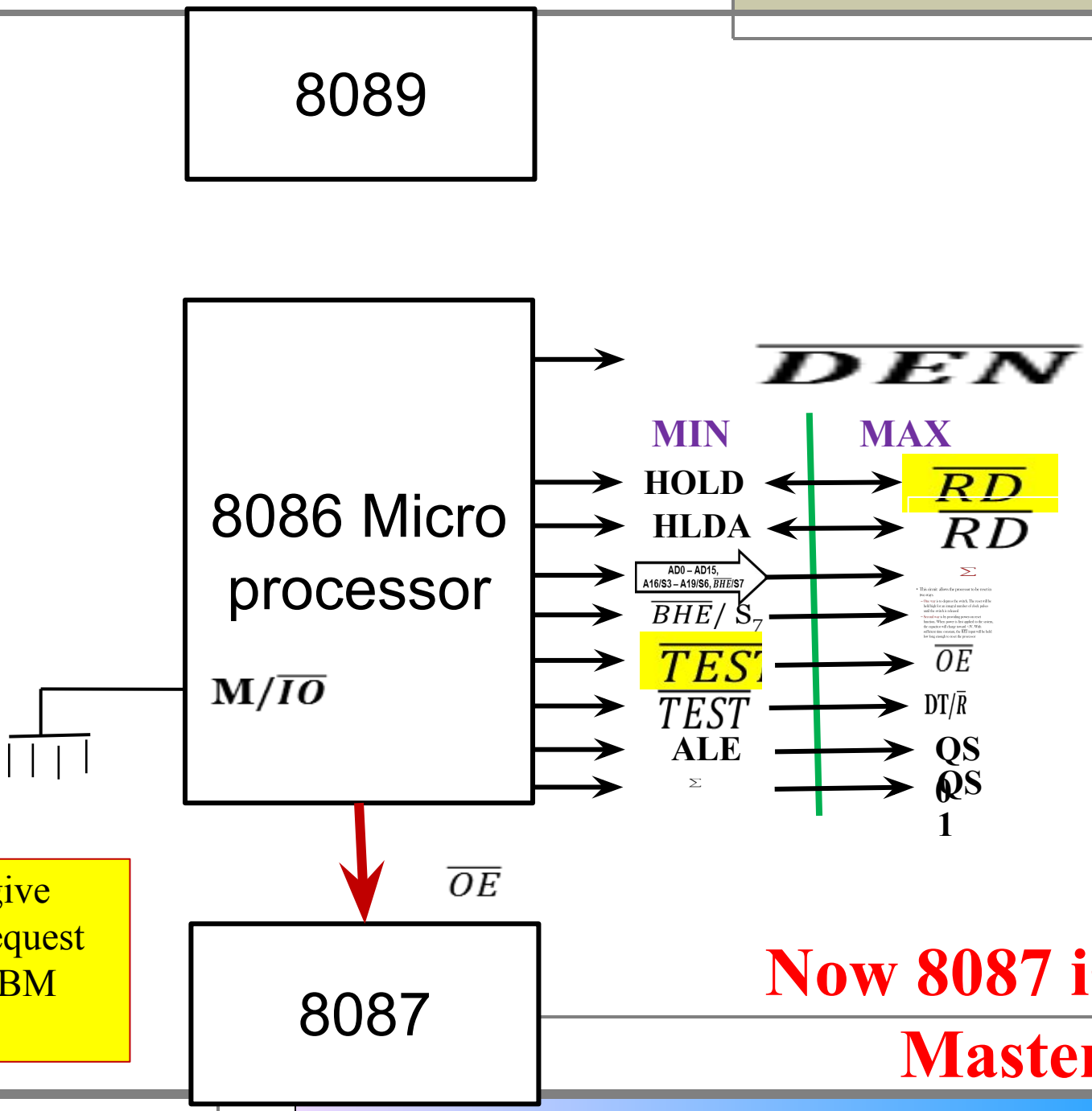
By default 8086, is the Bus Master.

If 8087 or 8089 wants to perform any operation on Memory then it has to become Bus Master.

It undergoes certain steps

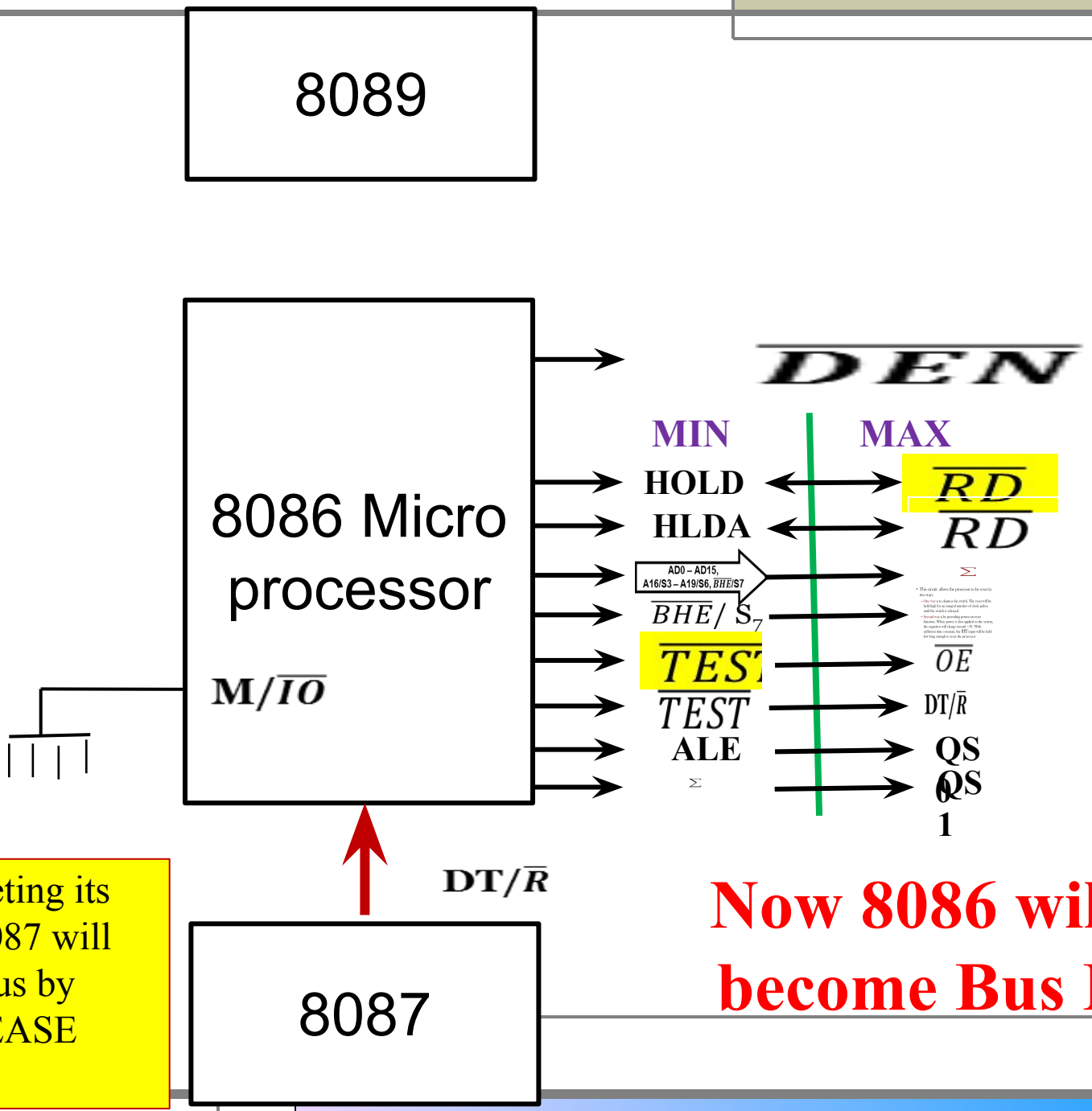


8087 will give Request (RQ) signal to 8086 to become BM



8086 will give grant the request to become BM

Now 8087 is Bus Master



After completing its operation, 8087 will release the bus by giving RELEASE SIGNAL

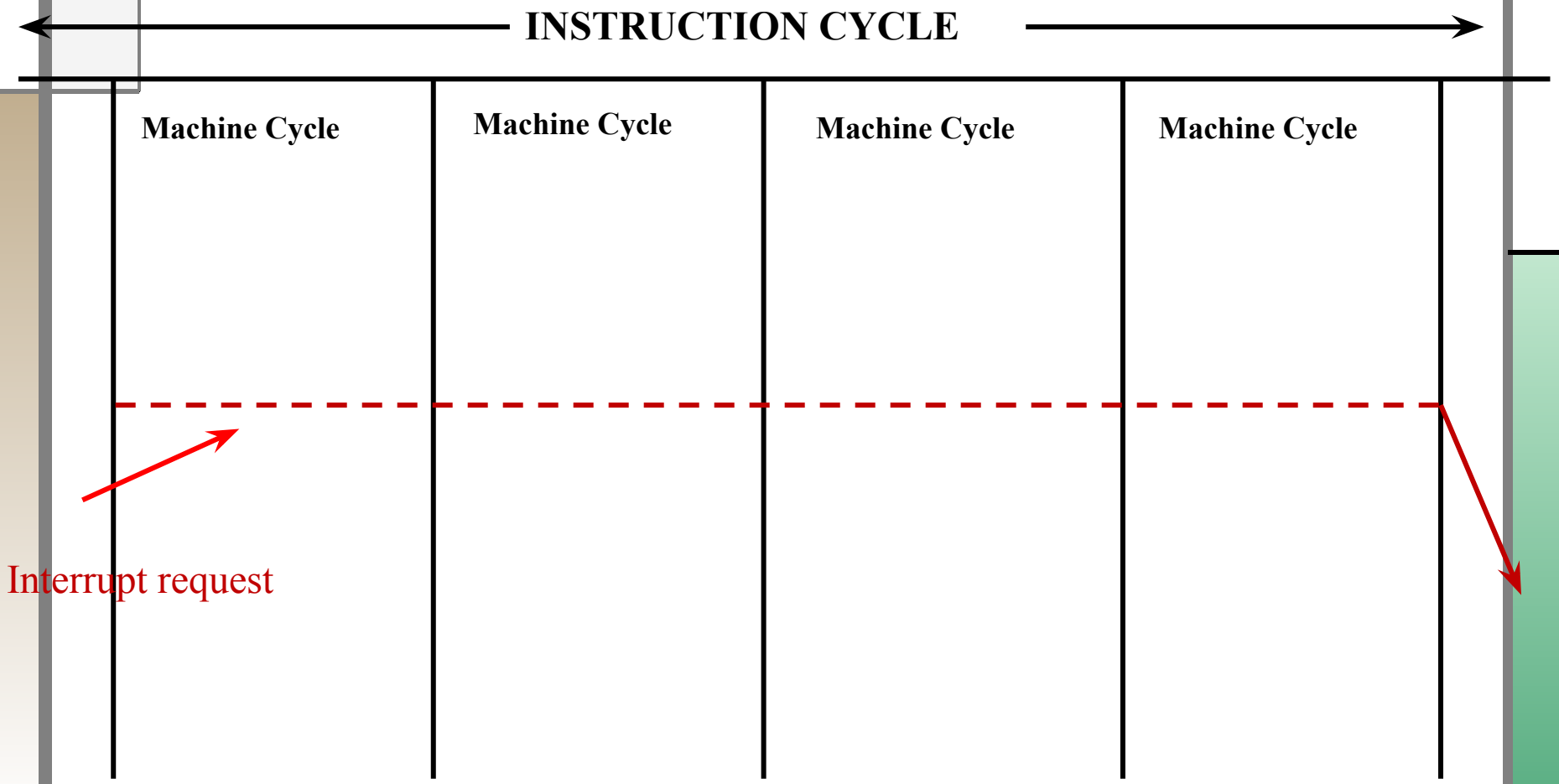
Now 8086 will again become Bus Master

$M/\overline{IO}$

Instruction Cycle Total time require to fetch, decode, execute an instruction.

" Entire process of an instruction is called INSTRUCTION CYCLE"

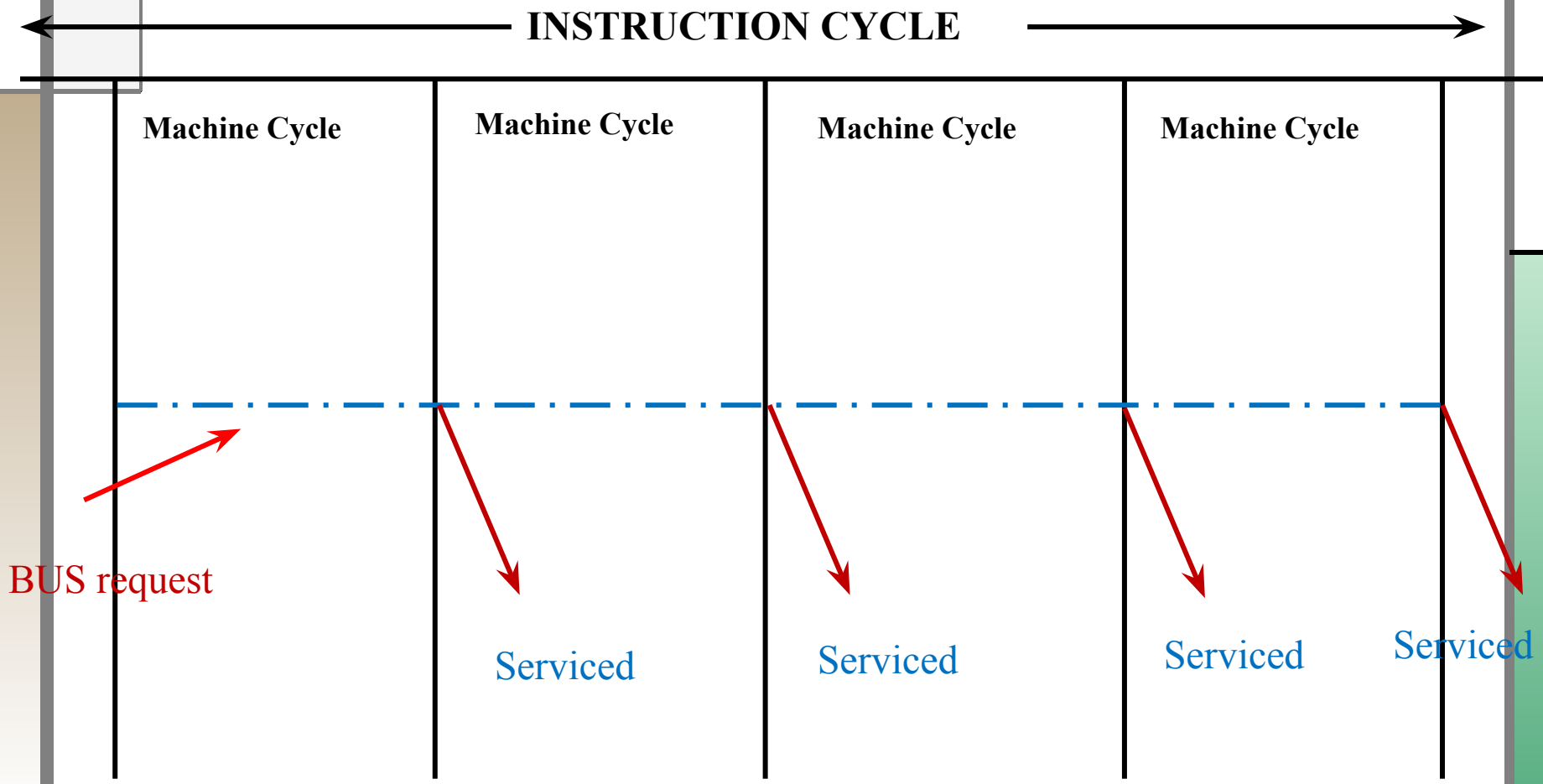
An instruction cycle have several Machine cycle like memory read, memory write, I/O Read, I/O write etc.



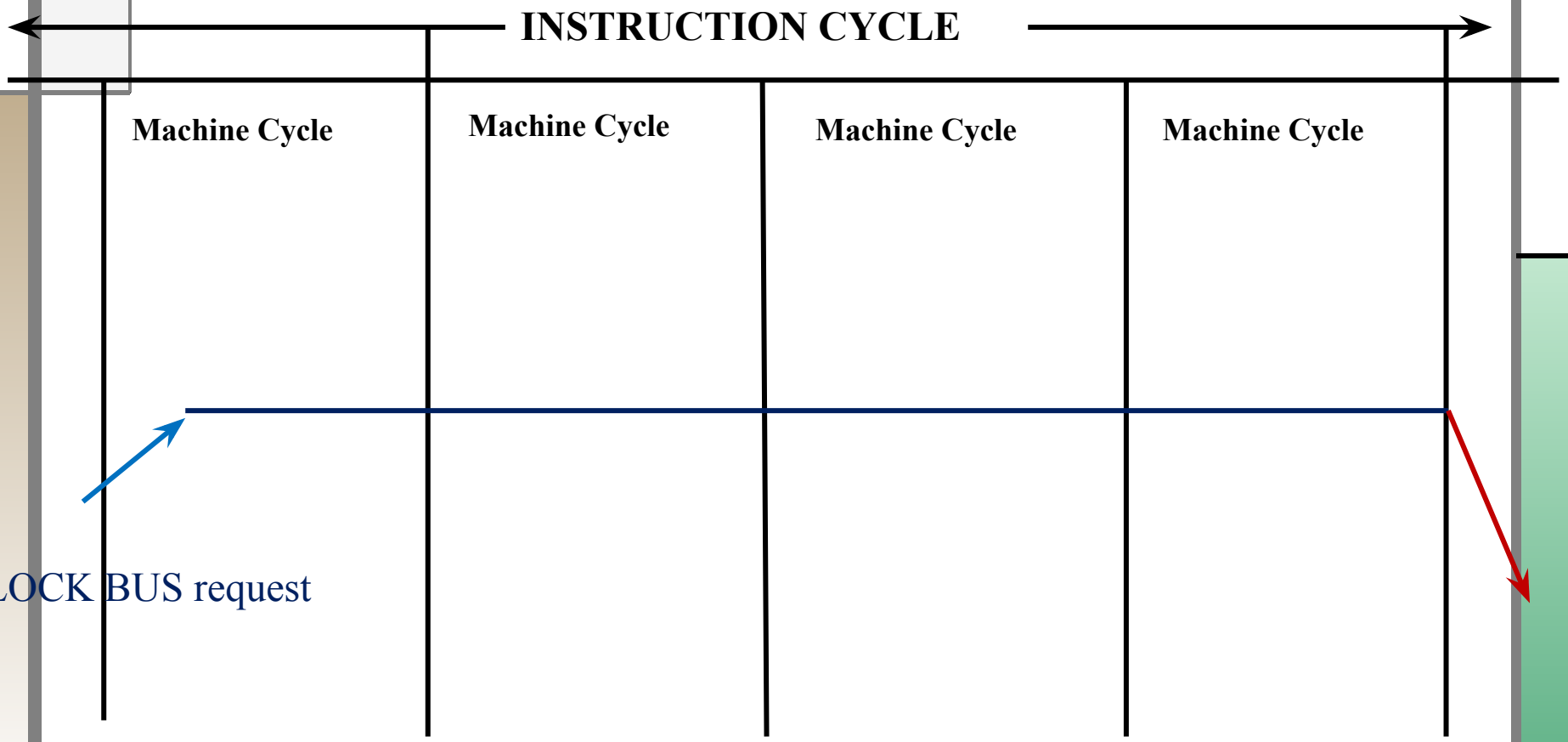
Whenever processor gets an interrupt it should first finish current instruction and then interrupt is serviced.

Time in μP

- T-state is the smallest unit of time in a μp
- **1** clock cycle = **1** T-state
- In 8086, **1** machine cycle = **4** T-states
- 1 machine cycle (or bus cycle) is the time required to
  - **T1** – send out an address – on address bus
  - **T2** – send out a signal (read/ write) – on control bus
  - **T3** – read/ write data on that location – on data bus
  - **T4** – stores the data and release all buses
- 1 instruction cycle = *n* machine cycles (depends on the instruction)



Whenever processor gives the bus request, it is not mandatory to finish the instruction, but it has to finish current machine cycle.



Example :

LOCK MOV DX, BX

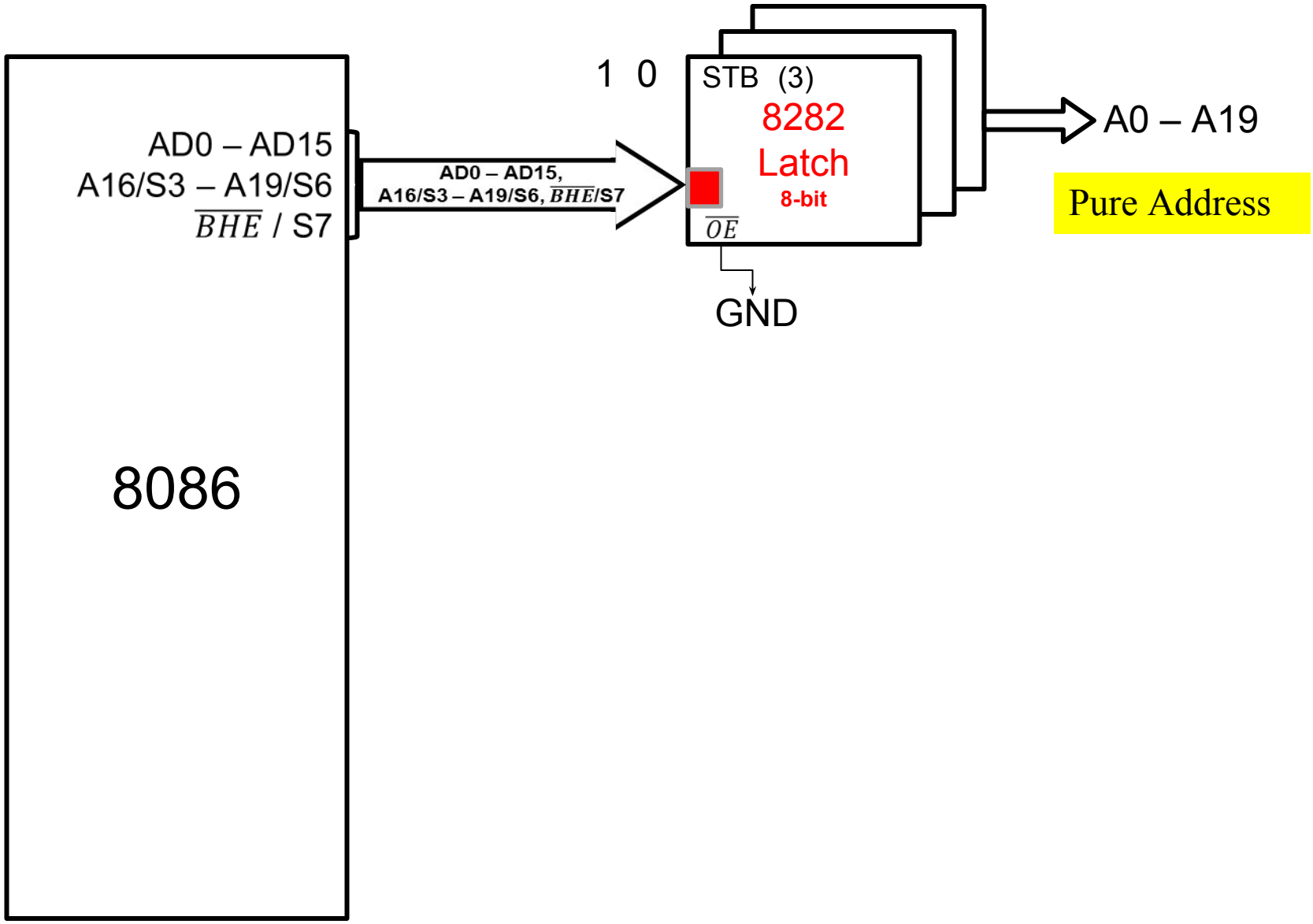
Processor will service the request after finishing whole instruction and also processor will release signal, $\overline{\textbf{LOCK}} = 0$

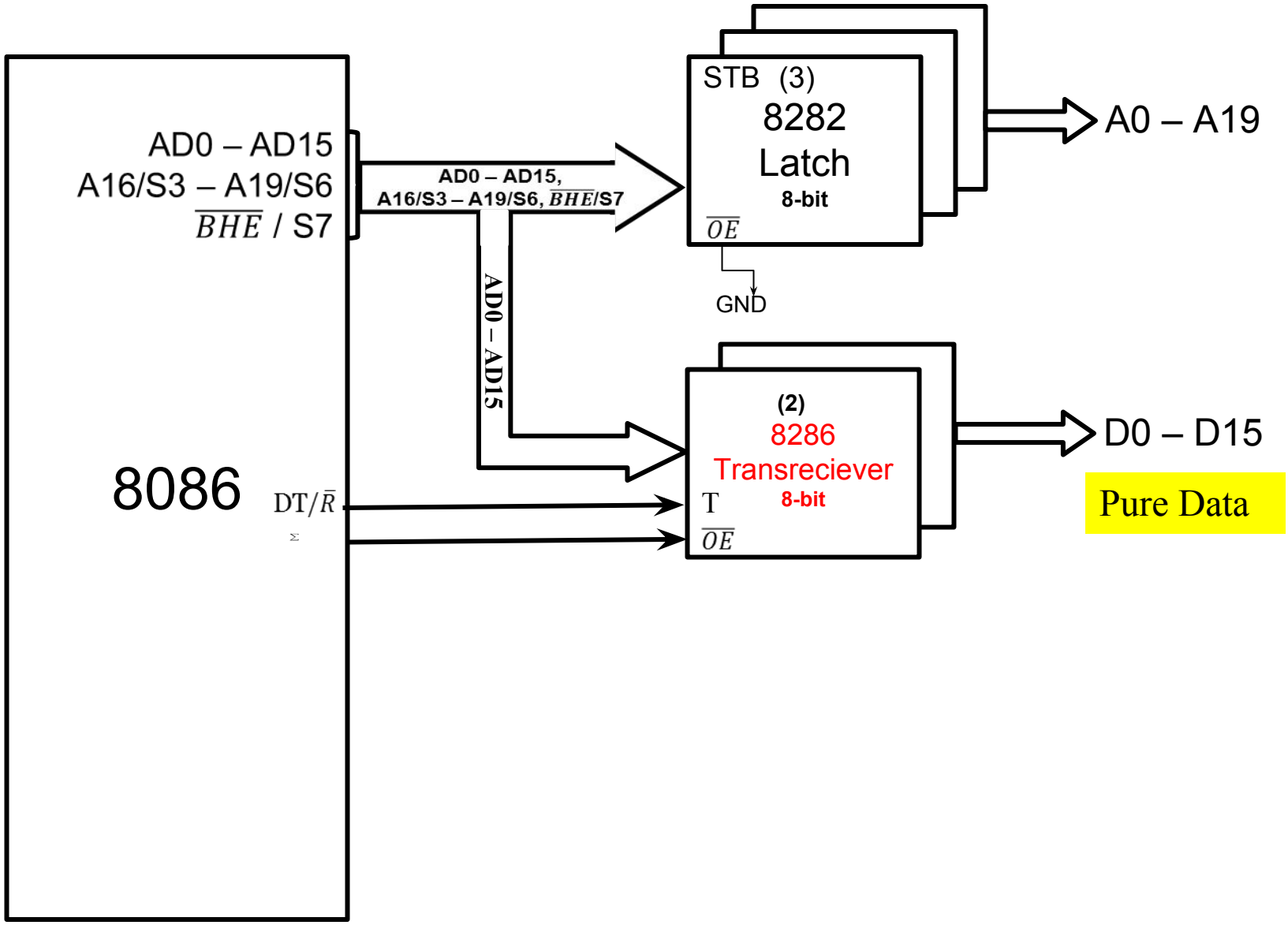
$\overline{S_0}, \overline{S_1}, \overline{S_2}$ Status signals

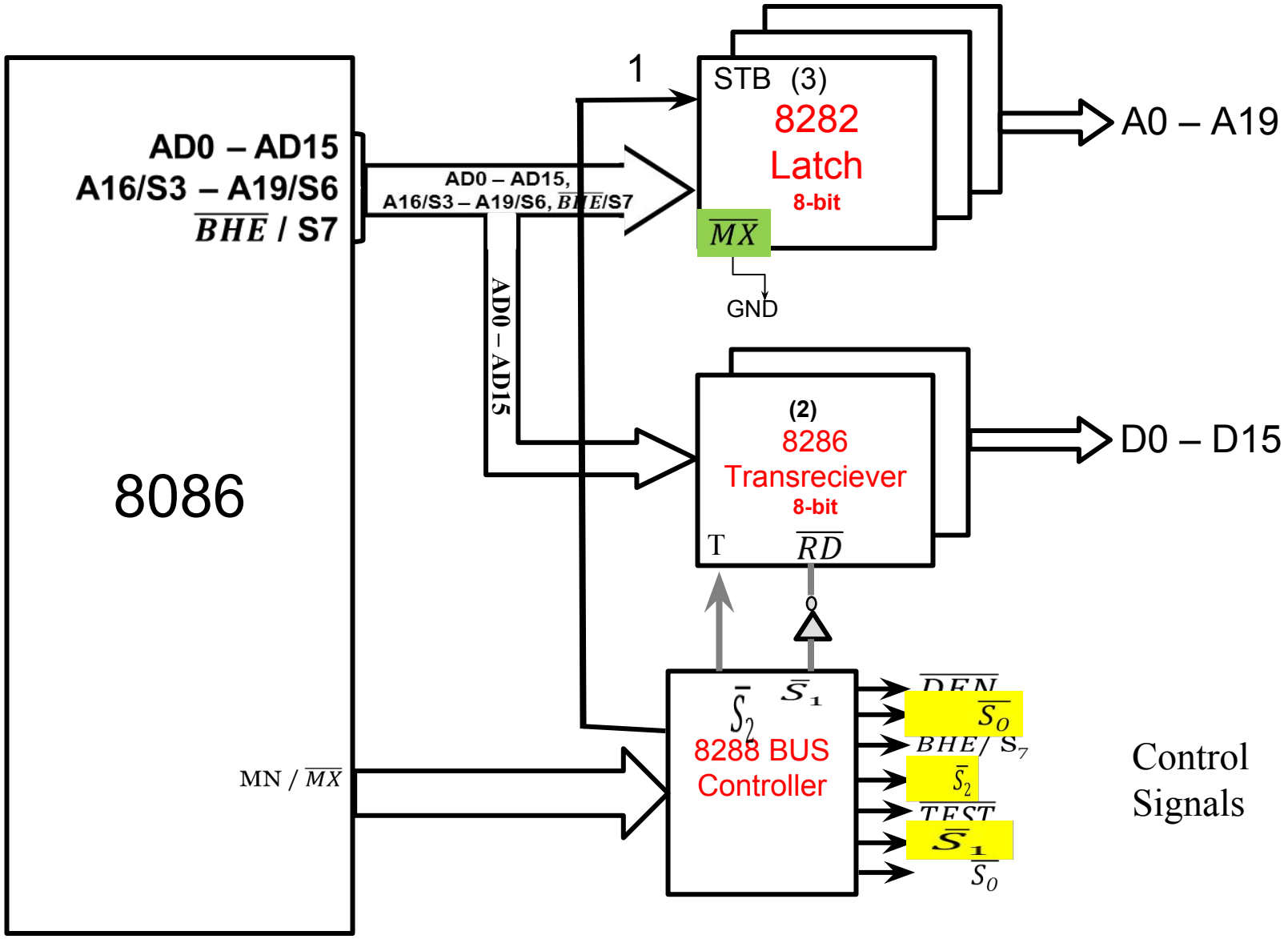
Used by the 8086 bus controller to generate bus timing and control signals. These are decoded as shown.

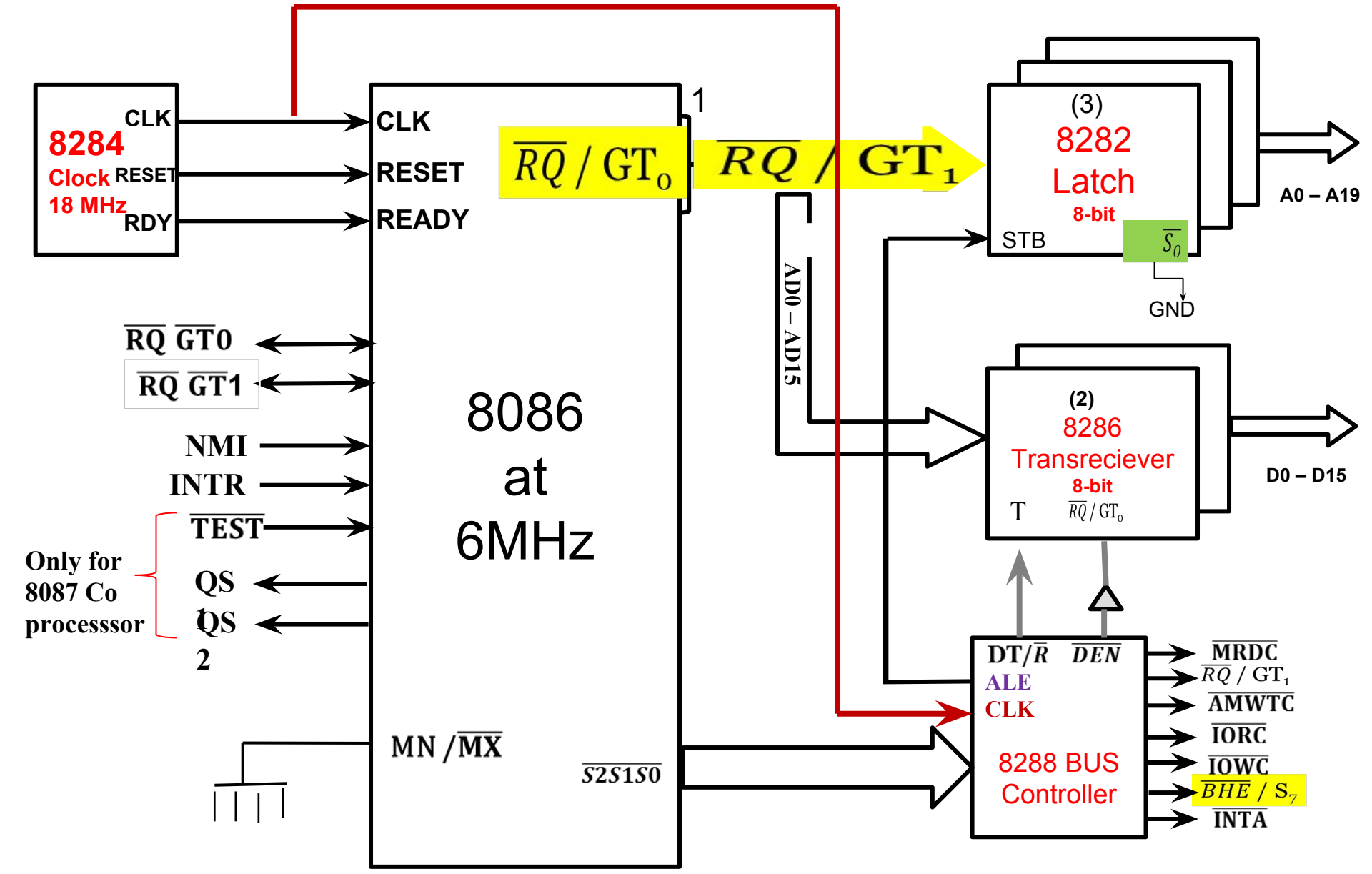
Status Signal			Machine Cycle
$\overline{S}_2$	$\overline{S}_1$	$\overline{S}_0$	
0	0	0	Interrupt acknowledge
0	0	1	Read I/O port
0	1	0	Write I/O port
0	1	1	Halt
1	0	0	Code access
1	0	1	Read memory
1	1	0	Write memory
1	1	1	Passive/Inactive

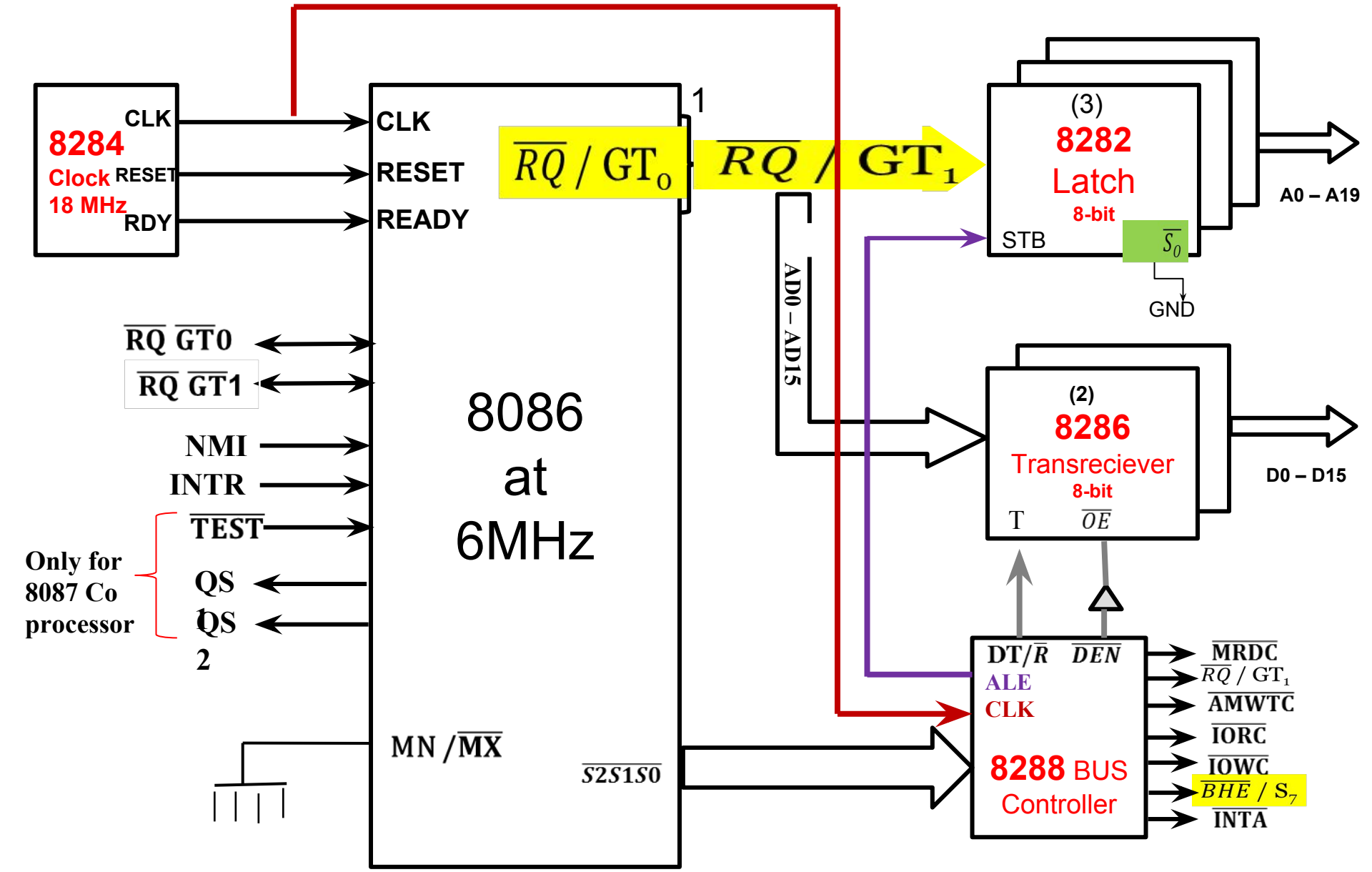
Only BM can set the status


AD0 – AD15
A16/S3 – A19/S6
$\overline{BHE}$ / S7
8086
AD0 – AD15,
A16/S3 – A19/S6, $\overline{BHE}$/S7
1 0
STB (3)
8282
Latch
8-bit
$\overline{OE}$
GND
A0 – A19
Pure Address


AD0 – AD15
A16/S3 – A19/S6
$\overline{BHE}$ / S7
8086
DT/$\bar{R}$
AD0 – AD15,
A16/S3 – A19/S6, $\overline{BHE}$/S7
AD0 – AD15
STB (3)
8282
Latch
8-bit
$\overline{OE}$
GND
A0 – A19
(2)
8286
Transreciever
8-bit
T
$\overline{OE}$
D0 – D15
Pure Data


AD0 – AD15
A16/S3 – A19/S6
$\overline{BHE}$ / S7
8086
MN / $\overline{MX}$
1
STB (3)
8282
Latch
8-bit
$\overline{MX}$
GND
A0 – A19
AD0 – AD15,
A16/S3 – A19/S6, $\overline{BHE}$/S7
AD0 – AD15
(2)
8286
Transreciever
8-bit
T
$\overline{RD}$
D0 – D15
$\bar{S}_2$
$\bar{S}_1$
8288 BUS
Controller
$\overline{DEN}$
$\overline{S_0}$
BHE/ S_7
$\bar{S}_2$
$\overline{TEST}$
$\bar{S}_1$
$\overline{S_0}$
Control
Signals


8284
Clock
18 MHz
CLK
RESET
RDY
CLK
RESET
READY
1
RQ / GT0
RQ / GT1
8086
at
6MHz
RQ GT0
RQ GT1
NMI
INTR
TEST
Only for
8087 Co
processsor
QS
QS
2
MN /MX
S2S1S0
AD0 – AD15
(3)
8282
Latch
8-bit
STB
S0
GND
A0 – A19
(2)
8286
Transreciever
8-bit
T
RQ / GT0
D0 – D15
DT/R
DEN
ALE
CLK
8288 BUS
Controller
MRDC
RQ / GT1
AMWTC
IORC
IOWC
BHE / S7
INTA



Entire System Bus is in a control of 8288 Bus Controller

Points to remember :-

- ALE in minimum mode produced by Microprocessor but in maximum mode ALE is not given by Microprocessor as we have multiprocessor.

- Only Bus Master will put the address in ALE i.e.
  if 8086 is BM, then 8086 will give ALE
  if 8087 is BM, then 8087 will give ALE
  if 8089 is BM, then 8089 will give ALE

ALE becomes 1 in first T state of Machine Cycle.

For Example :

101 status signal

Means memory read machine cycle (4T States), only in the first T state Bus is carrying address i.e. ALE =1

ALE takes care of two things:

1. Status
2. Which T State of Machine Cycle

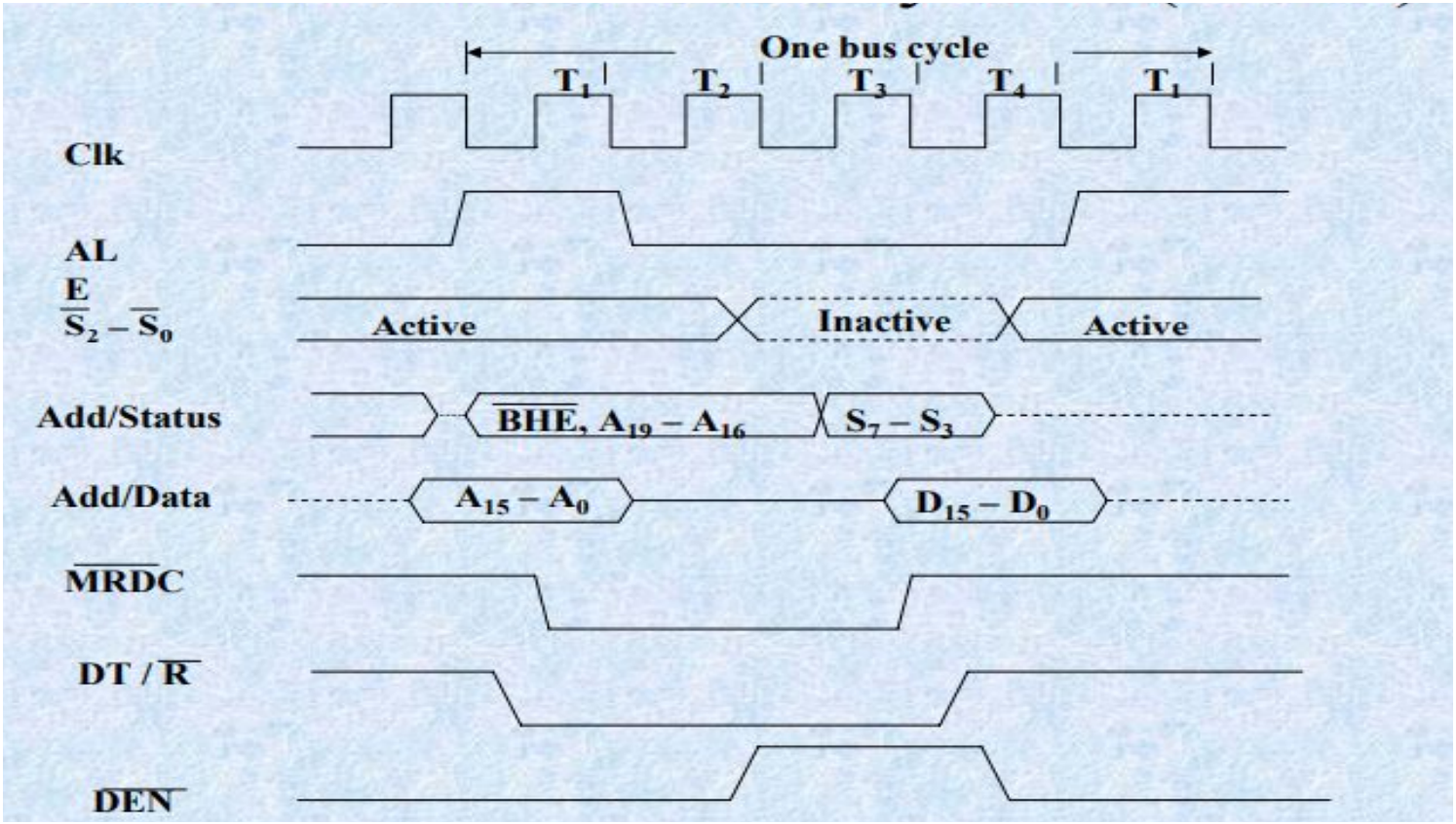
TEST

TEST pin is examined by the "WAIT" instruction. If the TEST pin is Low, execution continues. Otherwise the processor waits in an "idle" state. This input is synchronized internally during each clock cycle on the leading edge of CLK.

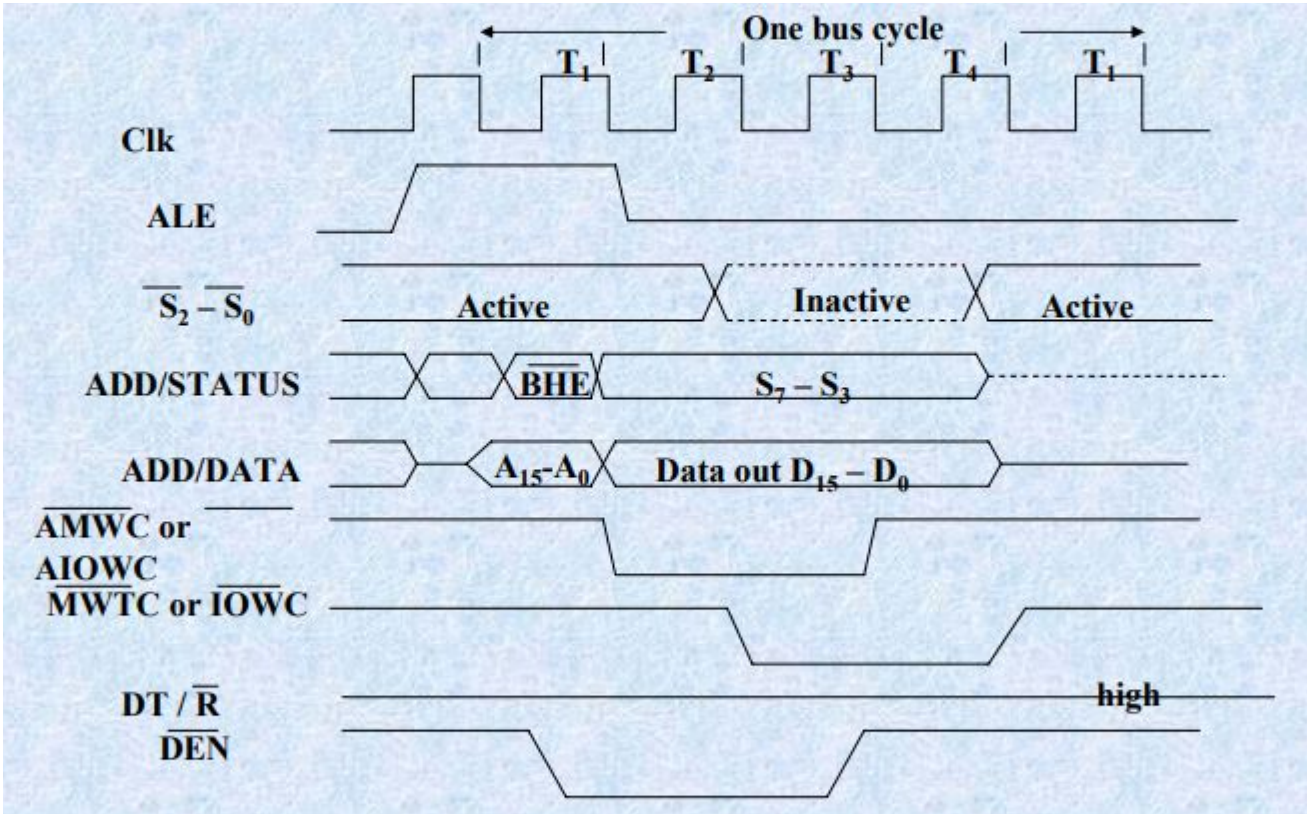
QS1 and QS0

QS1	QS1	Characteristics
0	0	No operation
0	1	First byte of opcode from queue
1	0	Empty the queue
1	1	Subsequent byte from queue

Timing diagrams for Read operation in maximum mode



Timing diagrams for Write operation in maximum mode



Questions

1. Draw and explain the working of 8086 in minimum mode.
2. Draw and explain the pin diagram of 8086 in minimum mode
3. Draw and explain the working of 8086 in maximum mode.
4. Draw and explain the pin diagram of 8086 in maximum mode

THANK YOU!!!