# ICS 3111:Microprocessor

Lec: Paul Musyoka

SCES, Strathmore University

June 27, 2023

# Course content

Strathmore UNIVERSITY

1. Microprocessors and Microcomputers
2. Microprocessor Architecture and Microcomputer systems
3. Microprocessor Architecture and Memory Interfacing
4. Interfacing I/O Devices
5. Interrupts
6. Programmable Interface Devices
7. General Purpose Programmable Peripheral Devices

# Microprocessors

- **What is a microprocessor**: A microprocessor is a multipurpose, programmable , clock-driven, register based electronic device that reads binary instructions from a storage device called memory, accepts binary data and provides results
- A typical programmable machine consists of: a **microprocessor**, a **memory** and **input/output** devices
- The components of a programmable system include:
  - **Hardware**: Which is the physical part of the system
  - **Program**: Which is a set of instructions written for the microprocessor to run
  - **Software**: Which a group of programs combine to perform a certain task

# Microprocessors

- A typical programmable system/microcomputer is shown in figure 1

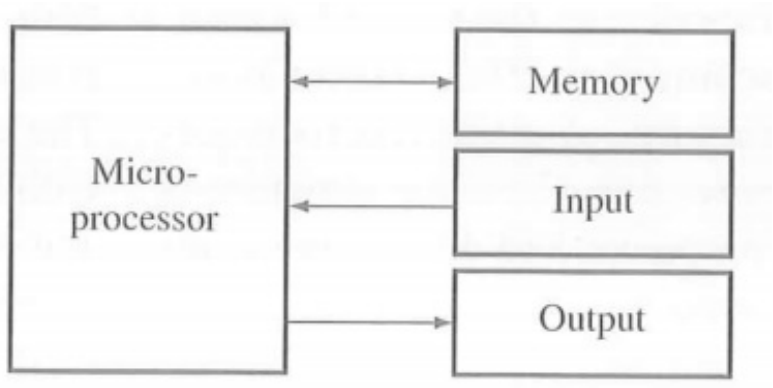



Figure 1: A programmable machine or microcomputer

# Microprocessor as a programmable device

- A microprocessor can receive instructions the perform a task based on those instructions.
- A modern microprocessor is designed to understand instructions and execute them.
- A programmer can select appropriate instructions and as the microcomputer to perform various tasks on a given data.
- The instructions are stored in a device called **memory**. A **memory** is a set of finite size registers.
- The registers are always grouped together in powers of two. For instance, a group of $1024(2^{10})$ 8-bit registers on a semiconductor chip is known as $1K$ byte memory

# Microprocessor as a programmable device

- Input/output devices allow users to interact with a microprocessor. Such devices include a keyboard, card reader, a scanner, monitors, LED displays etc
- The primary component of a microcomputer is a microprocessor also known as the central processing unit (CPU)
- The CPU contains various
  - Registers to store temporary data
  - The arithmetic logic unit (ALU) which performs arithmetic and logical operations
  - The instruction decoders, counters and control lines

# Microprocessor as a programmable device

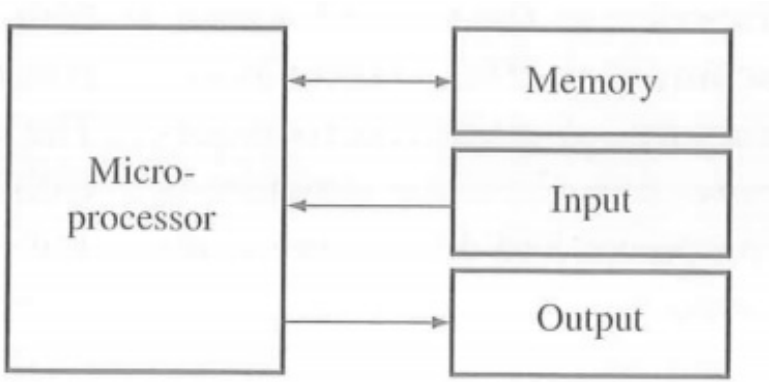



Figure 2: A block diagram of a computer

# Microprocessor as a programmable device

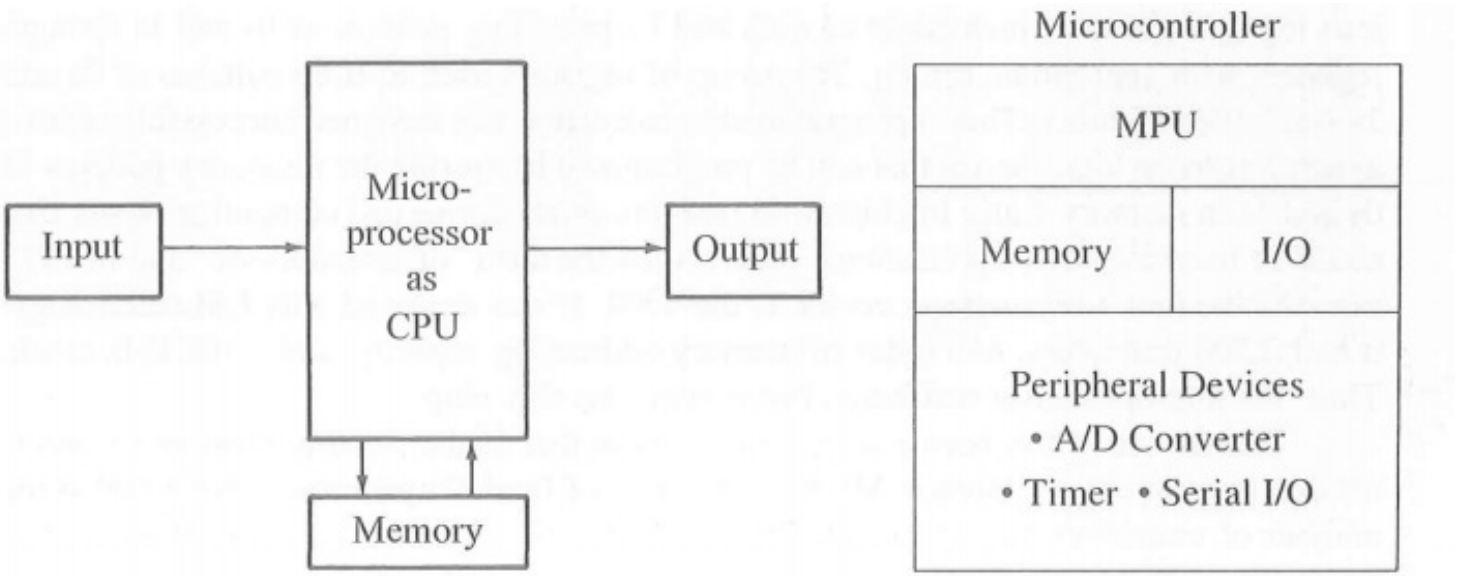



Figure 3: A block diagram of a computer with CPU and a block diagram of a microcontroller

# Advance semiconductor technology

- The invention of integrated circuits in the 1950s prompted development of large scale integration which eventually led to the manufacturing of microprocessors.

Intel Microprocessors: Historical Perspective

| Processor | Year of Introduction | Number of Transistors | Initial Clock Speed | Address Bus | Data Bus | Addressable Memory |
|---|---|---|---|---|---|---|
| 4004 | 1971 | 2,300 | 108 kHz | 10-bit | 4-bit | 640 bytes |
| 8008 | 1972 | 3,500 | 200 kHz | 14-bit | 8-bit | 16 K |
| 8080 | 1974 | 6,000 | 2 MHz | 16-bit | 8-bit | 64 K |
| 8085 | 1976 | 6,500 | 5 MHz | 16-bit | 8-bit | 64 K |
| 8086 | 1978 | 29,000 | 5 MHz | 20-bit | 16-bit | 1 M |
| 8088 | 1979 | 29,000 | 5 MHz | 20-bit | 8-bit* | 1 M |
| 80286 | 1982 | 134,000 | 8 MHz | 24-bit | 16-bit | 16 M |
| 80386 | 1985 | 275,000 | 16 MHz | 32-bit | 32-bit | 4 G |
| 80486 | 1989 | 1.2 M | 25 MHz | 32-bit | 32-bit | 4 G |
| Pentium | 1993 | 3.1 M | 60 MHz | 32-bit | 32/64-bit | 4 G |
| Pentium Pro | 1995 | 5.5 M | 150 MHz | 36-bit | 32/64-bit | 64 G |
| Pentium II | 1997 | 8.8 M | 233 MHz | 36-bit | 64-bit | 64 G |
| Pentium III | 1999 | 9.5 M | 650 MHz | 36-bit | 64-bit | 64 G |
| Pentium 4 | 2000 | 42 M | 1.4 GHz | 36-bit | 64-bit | 64 G |

Figure 4: The historical perspective of Intel microprocessors

# Organization of a Microprocessor-Based System

- The figure 5 shows a microprocessor based system with the main components including the system **bus**.

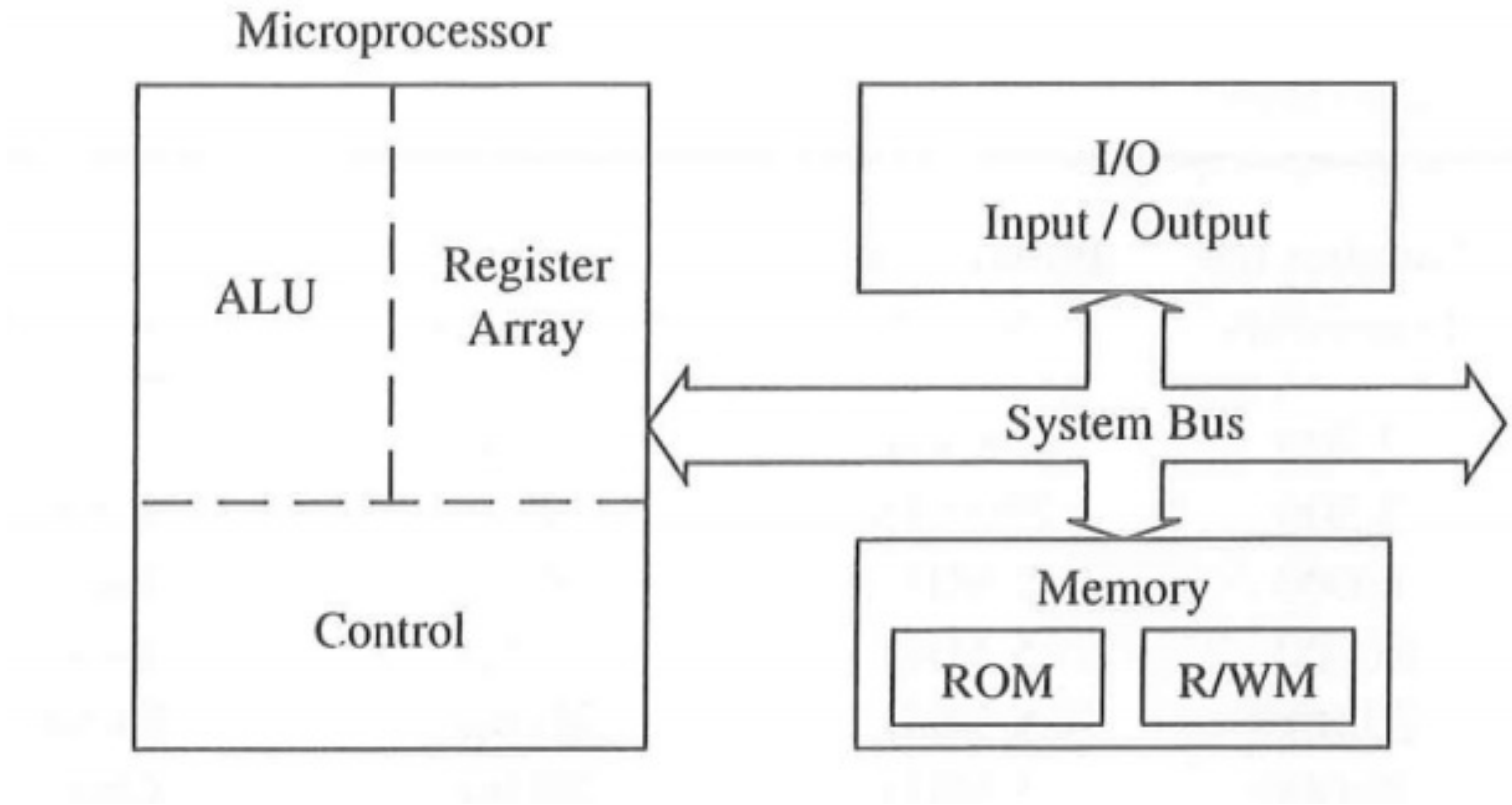



Figure 5: A microprocessor based system

# Overview of other processors

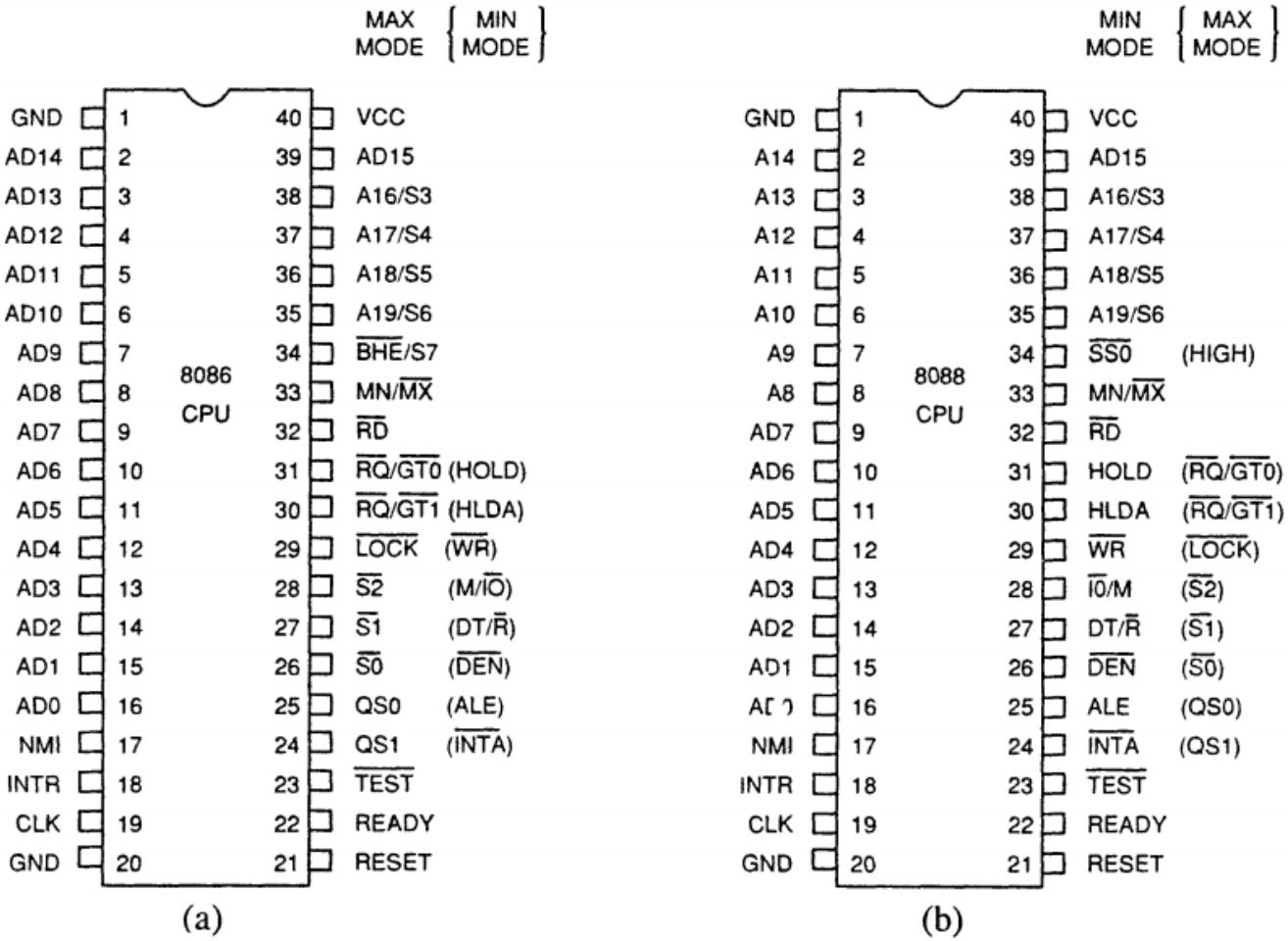



Figure 6: 8086 pin out

# Overview of other processors

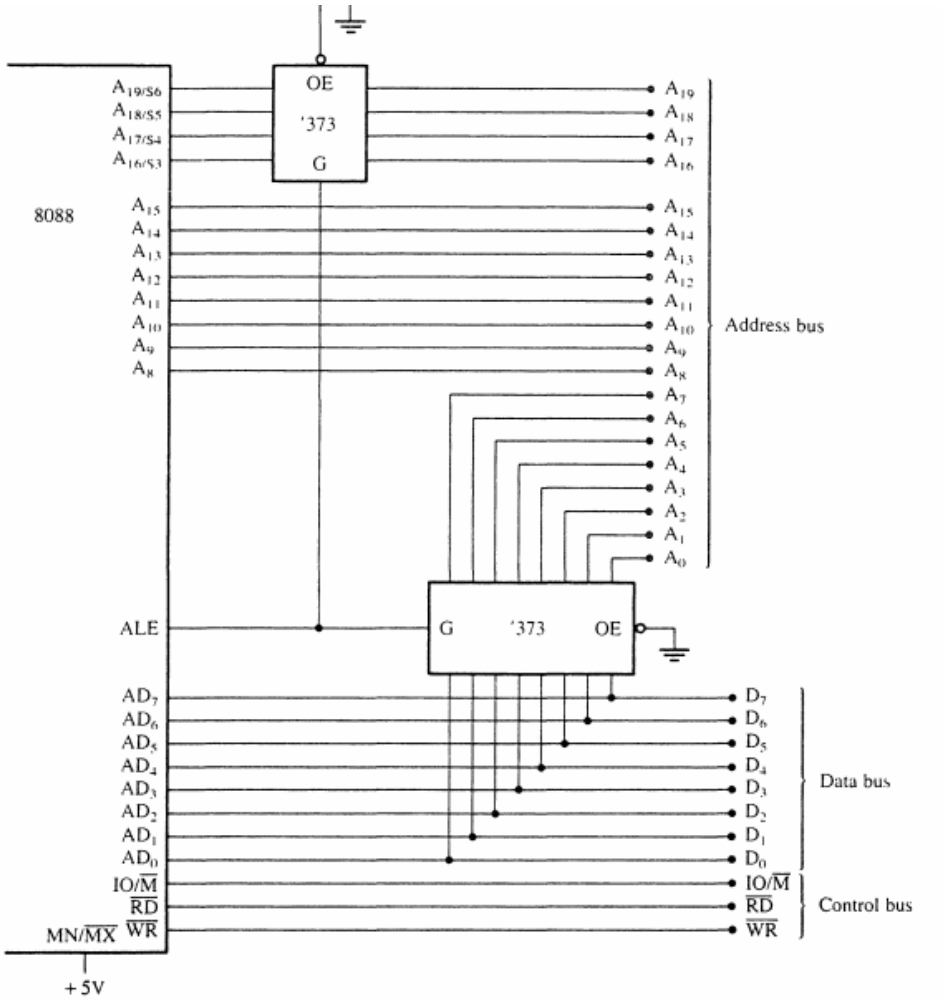



Figure 7: Bus latching

# Overview of other processors

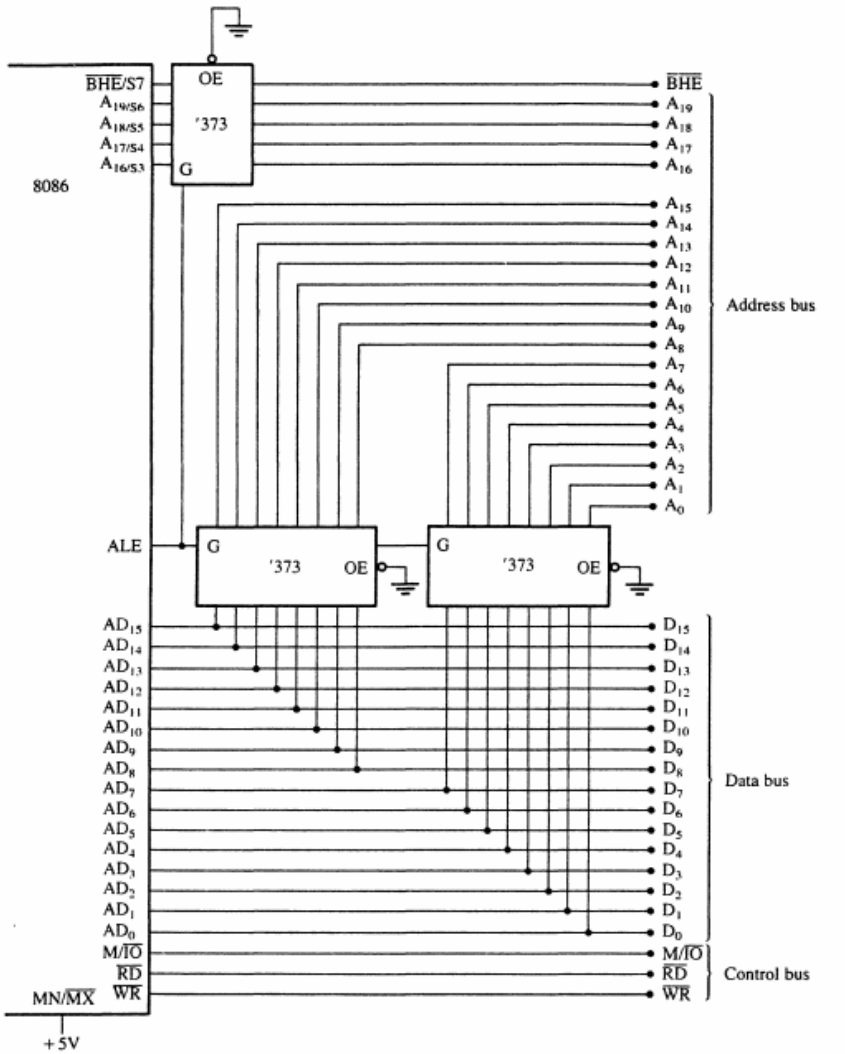



Figure 8: Bus latching most applicable

# Overview of other processors

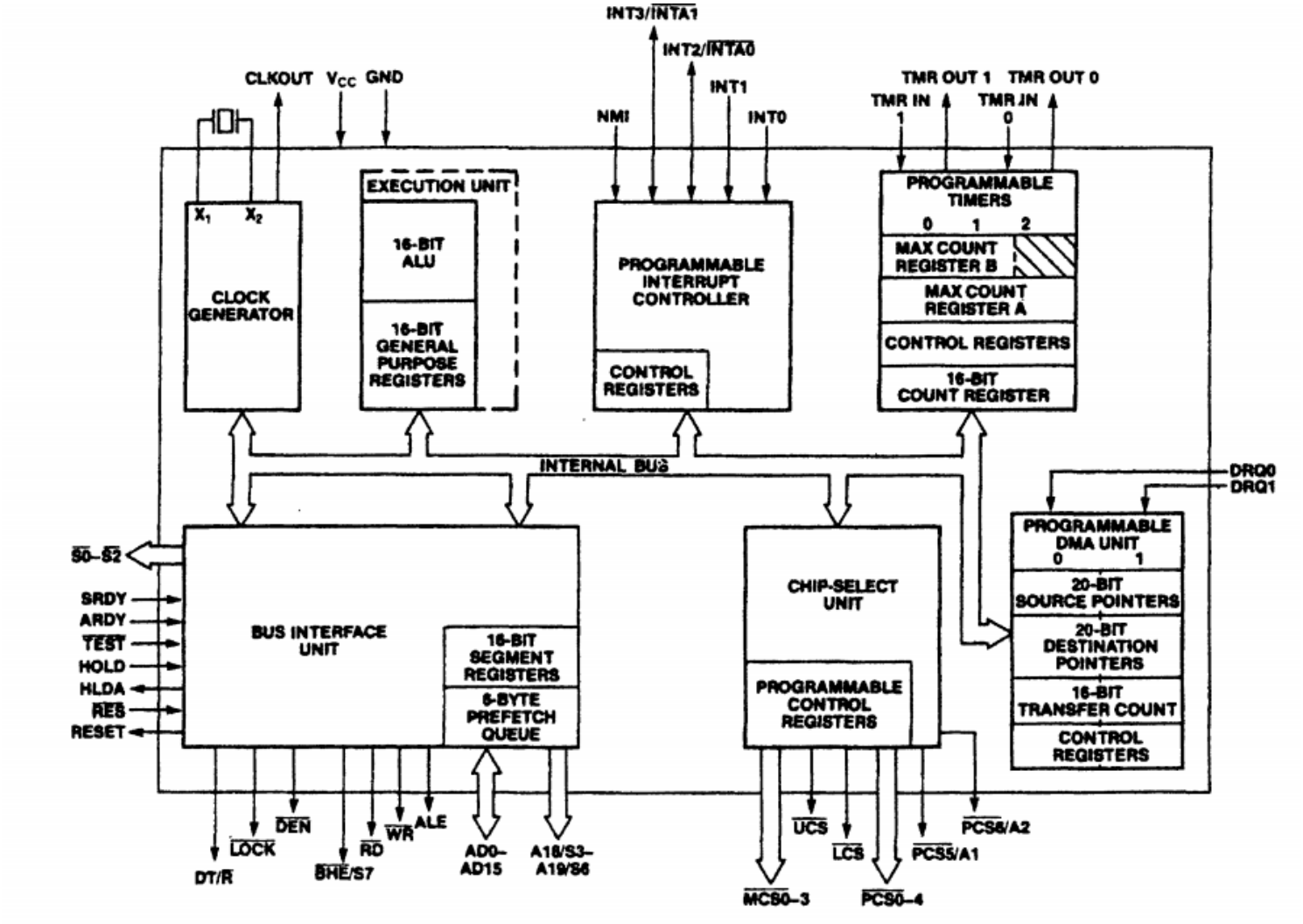



Figure 9: 80186 block diagram

# Overview of other processors

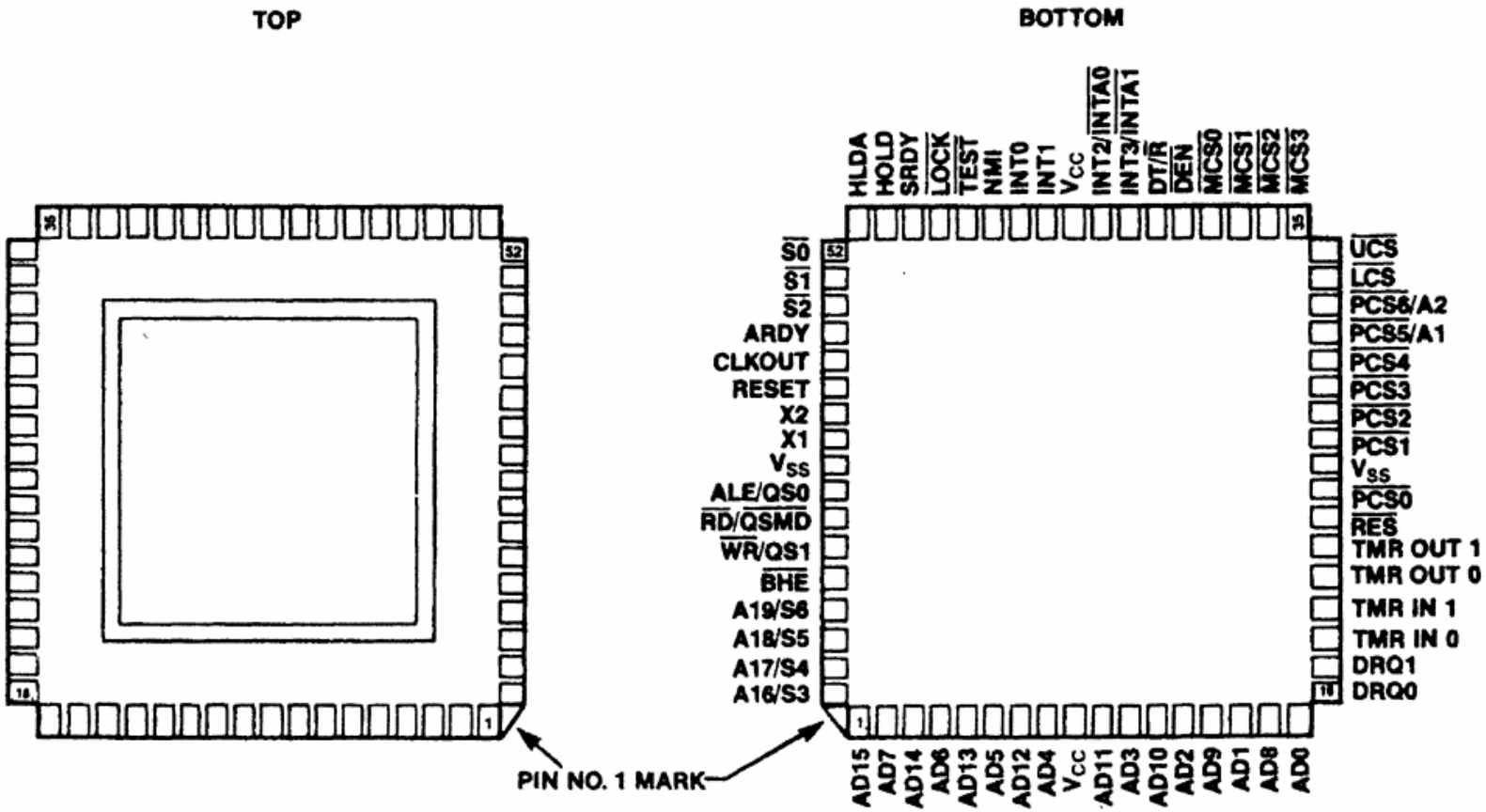



Figure 10: 80186 pin diagram

# Overview of other processors

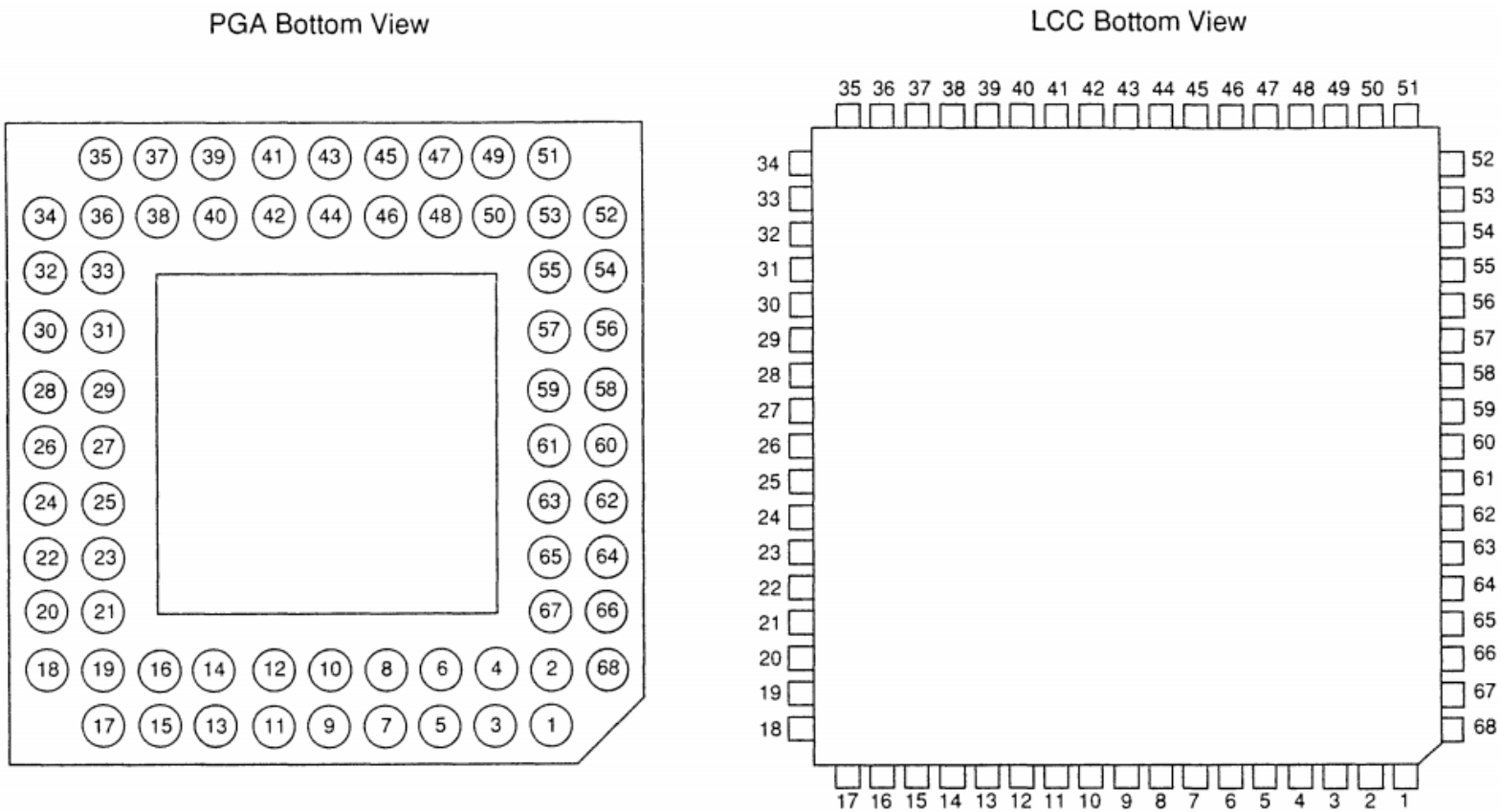



Figure 11: 80186 pin diagram

# Overview of other processors

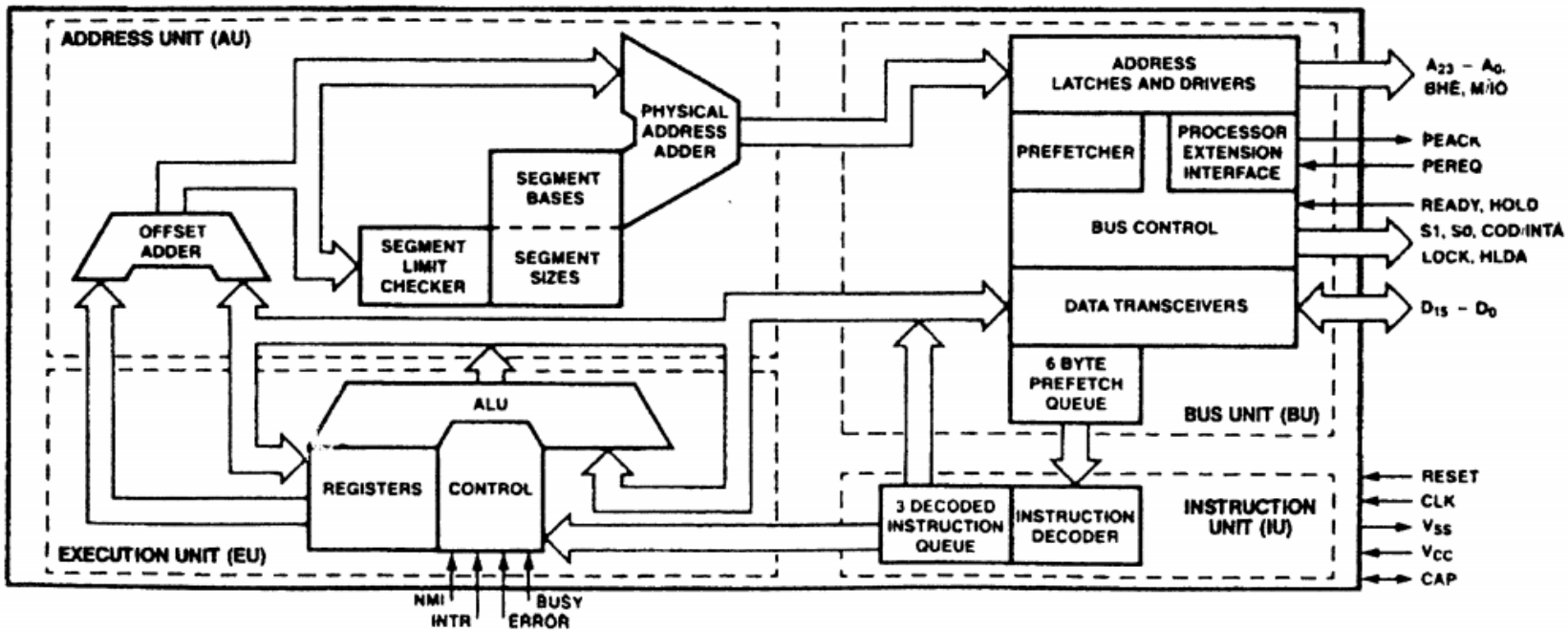



Figure 12: 80286 block diagram courtesy Intel Corporation

# Overview of other processors

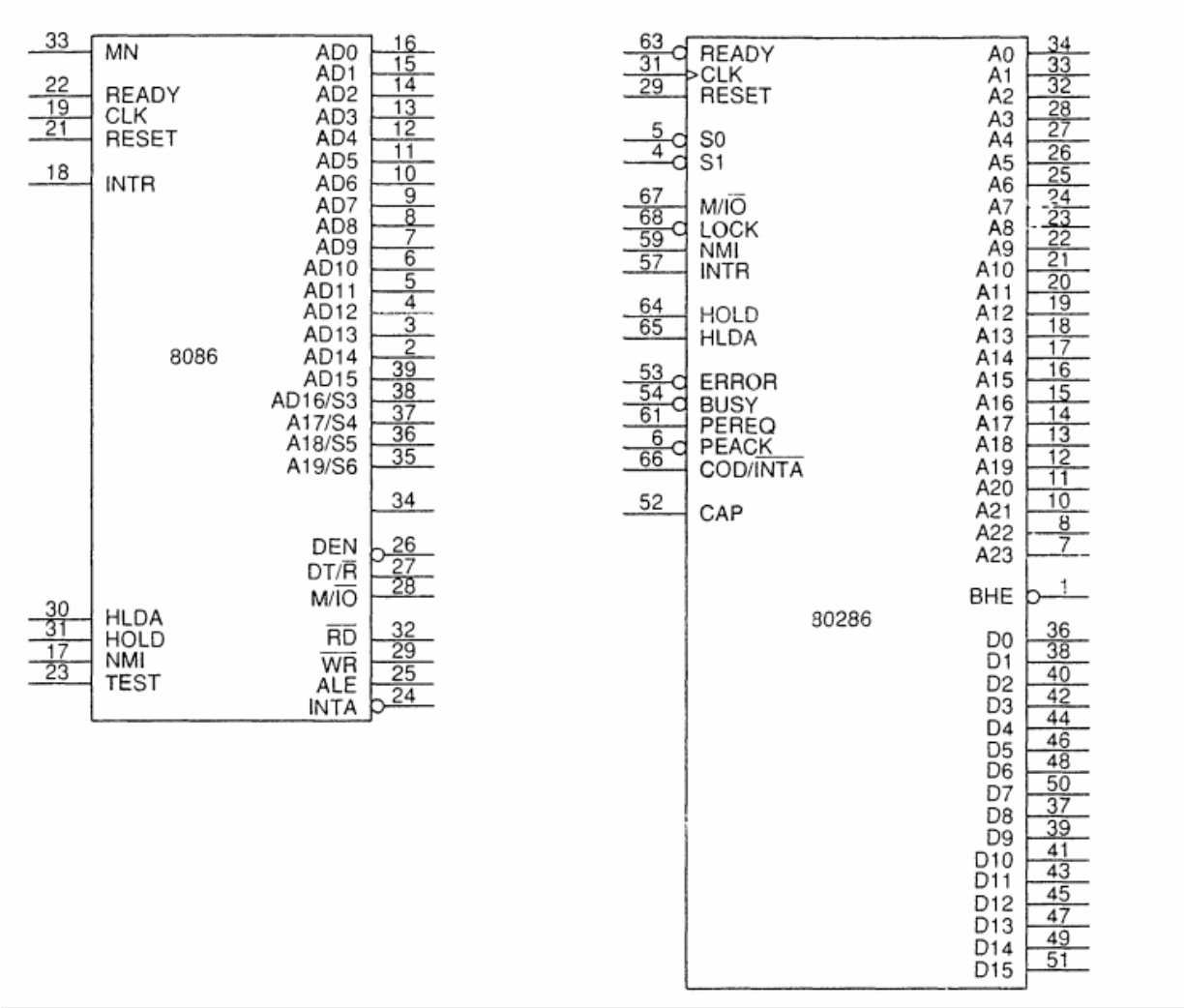



Figure 13: 80286 pin diagram courtesy Intel Corporation

# Overview of other processors

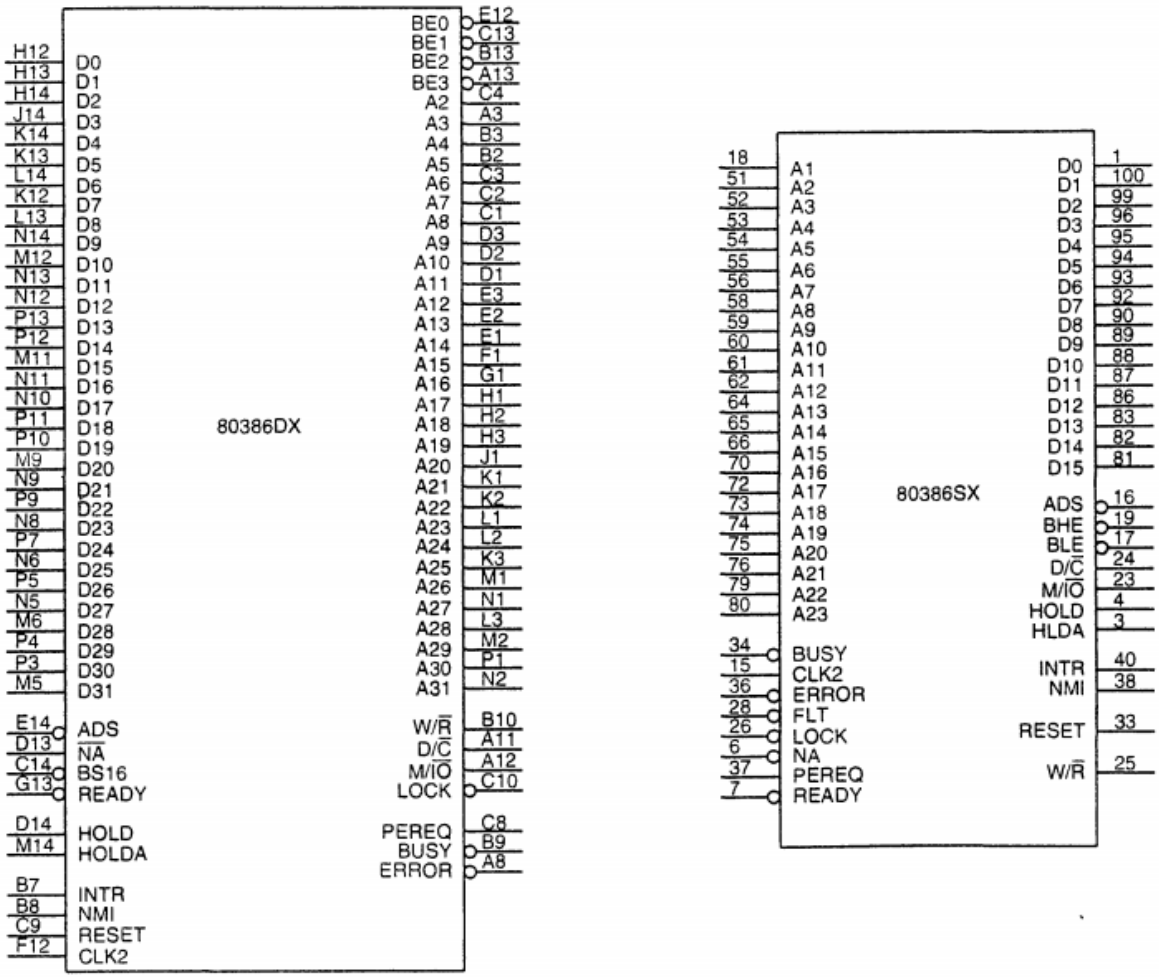
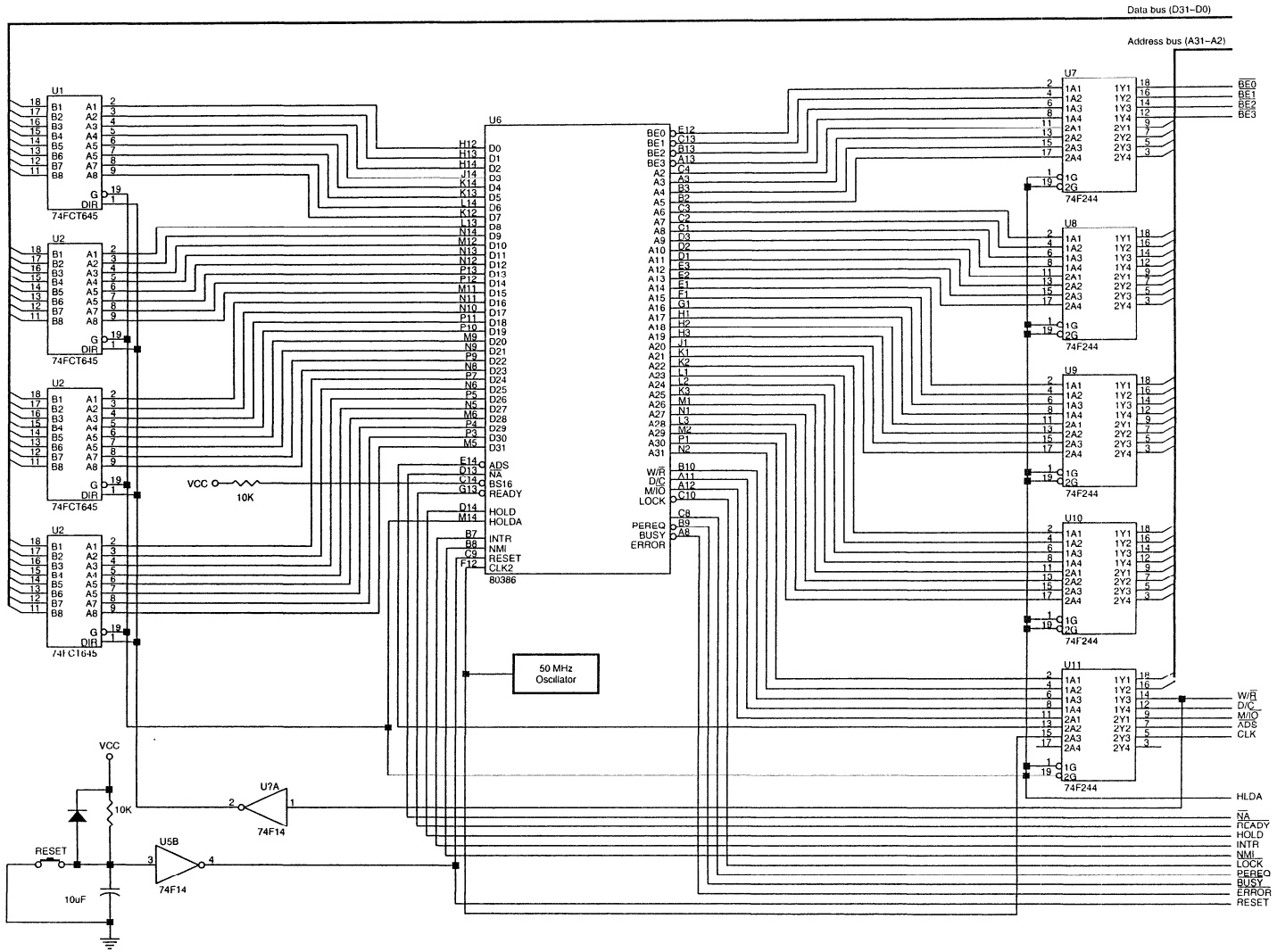



Figure 14: 80386 pin diagram courtesy Intel Corporation

# Overview of other processors



# Overview of other processors

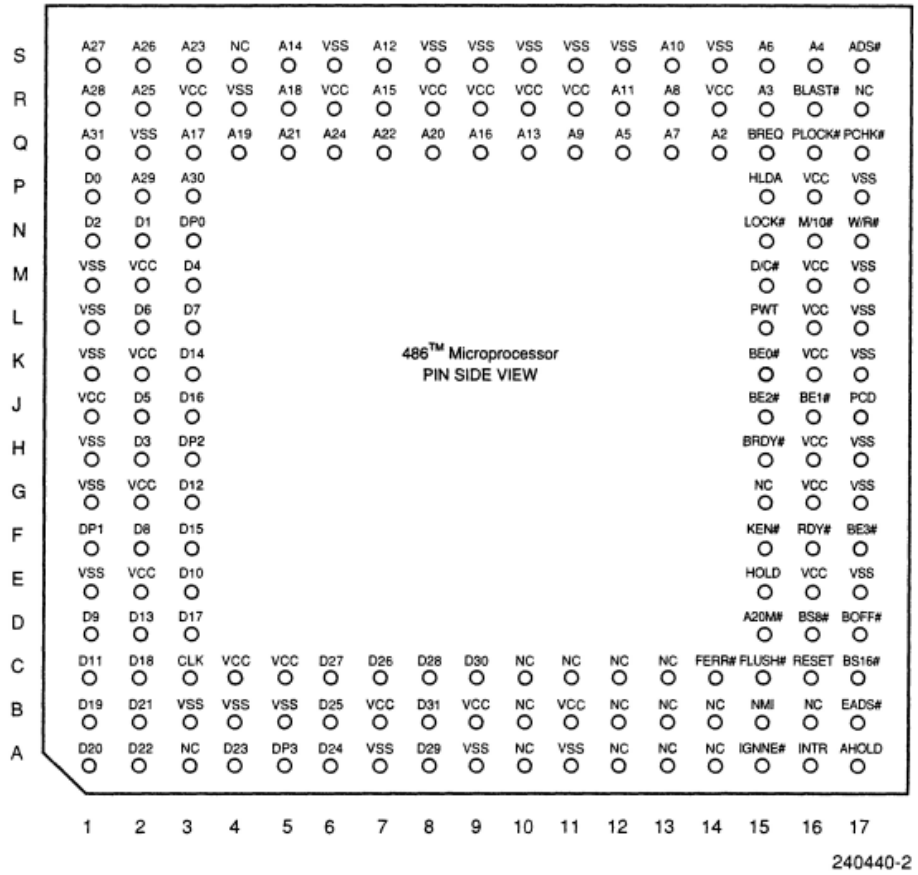



Figure 16: A pin diagram 80486 courtesy Intel Corporation

# Overview of other processors

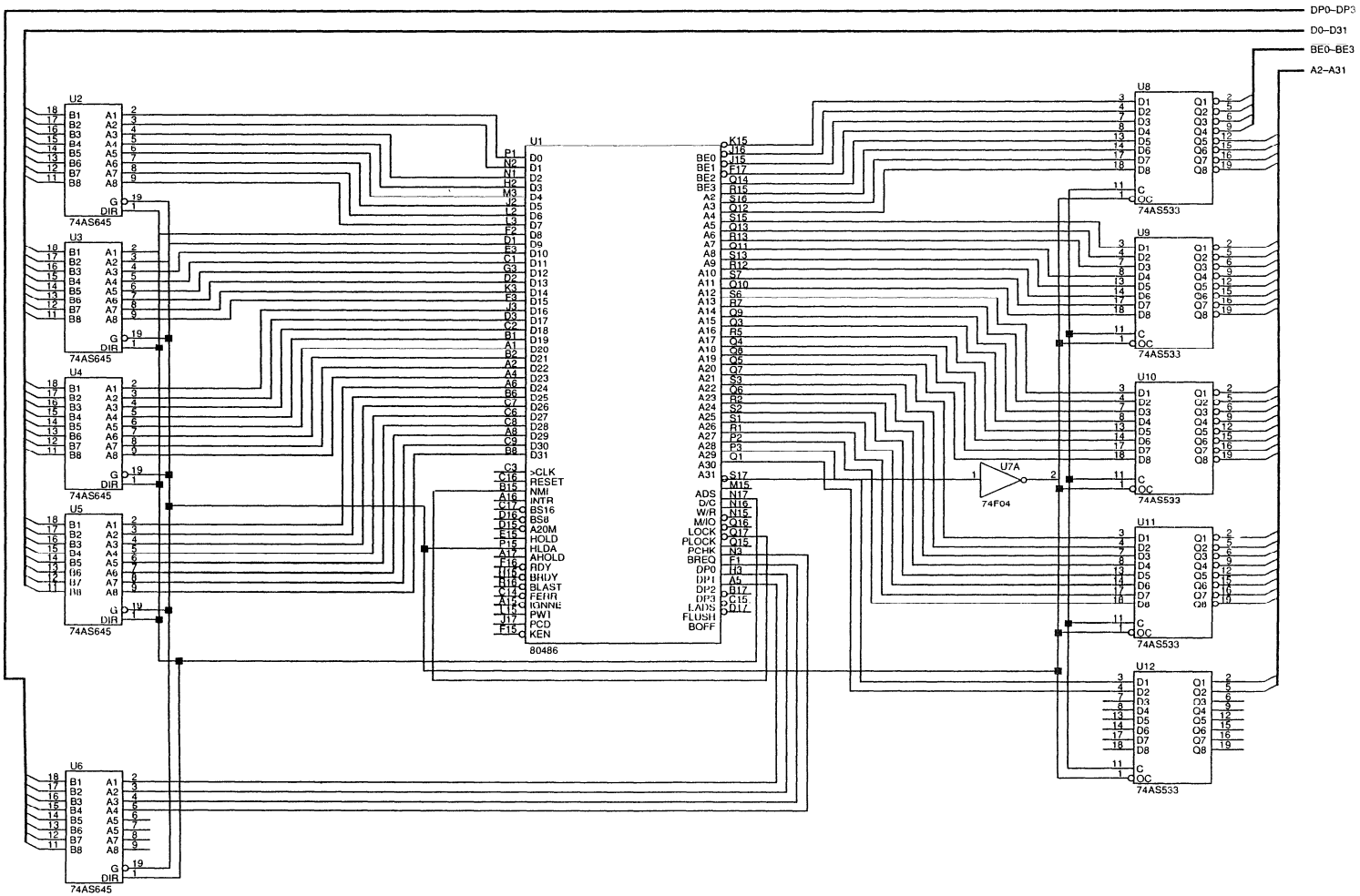
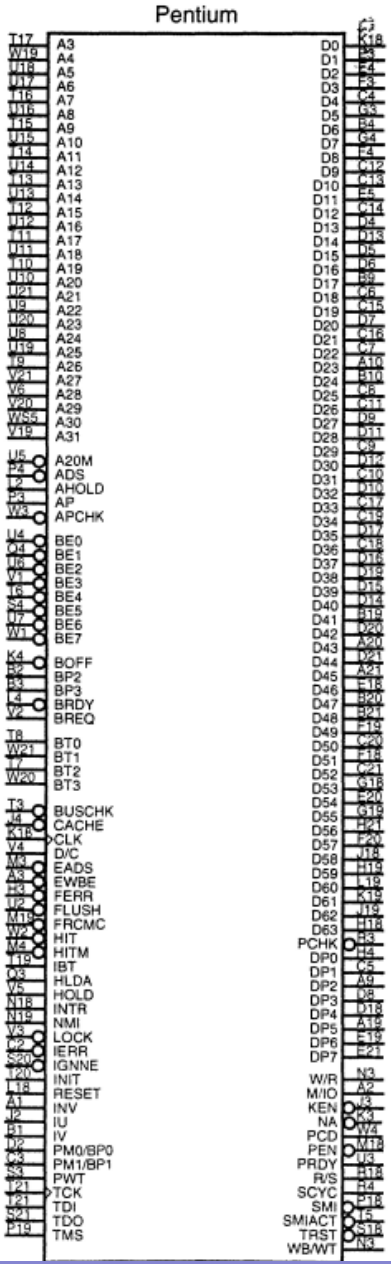



**FIGURE 16–29** An 80486 microprocessor showing the buffered address, data, and parity buses

# Overview of other processors



# Microprocessor Internal Architecture

- As seen in figure 5 the microprocessor unit is made up of the following internal components.
  - **Arithmetic Logic** Unit: It performs arithmetic operations such addition and subtraction and logic operation such as AND, OR, and exclusive OR.
  - **Register array**: It consists of various registers identified by letters such as A,B,C,D,E,H and L. These registers are primarily used to store data temporarily during the execution of a program and are accessible to the user.
  - **Control unit**: The control unit provides the necessary timing and control signals to all operations in the microcomputer. It also controls the flow of data between the microprocessor and memory and other peripherals.
- The memory can be divide into two categories namely RAM and ROM.
- The system is the communication path between the MPU and its peripherals

# How does the microprocessor work?

- Assume there is a program and data already entered in the RAM
- When the MPU is given a command to execute the program which is stored in the memory sequentially
- It fetches the first instruction from memory, it decodes it the executes it.
- This process is continued until all the instructions and executed.

# Detailed 8085 internal architecture

- The internal architecture of the Intel 8085 is given in the diagram in figure 19

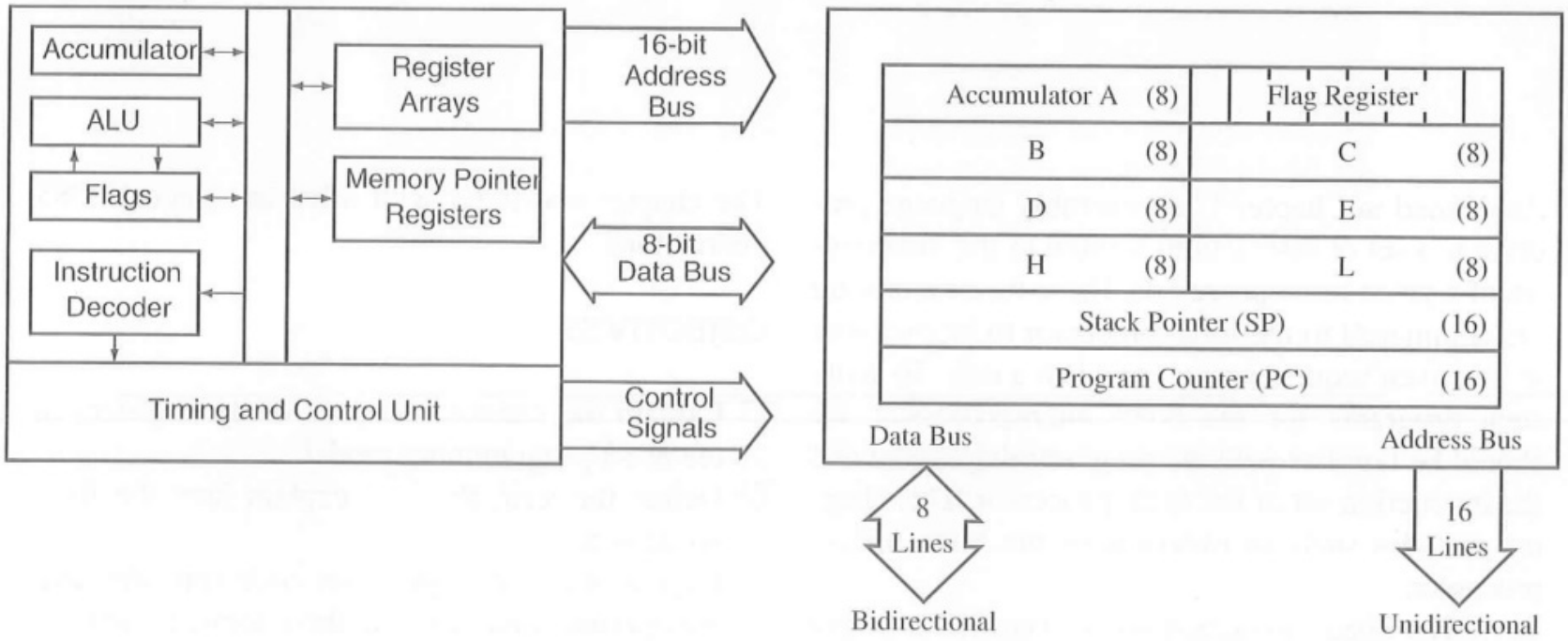



Figure 19: The internal architecture of the Intel 8085 MPU show the register arrangement and system buses

# The Intel 8085 Flag Register

- The flag register is shown in figure 20

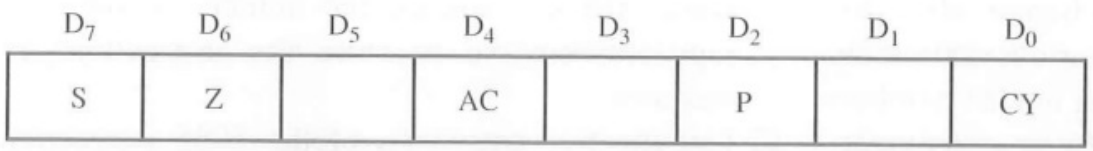



Figure 20: The Intel 8085 flag register

- The flags play critical role in decision during program execution. Their roles are explained in the next slide

# The Intel 8085 Flag Register

Strathmore UNIVERSITY

- Z-Zero flag: The Zero flag is set to one when the result in the accumulator or other registers is zero. Otherwise it is reset.
- CY-Carry flag: If an arithmetic operation results into a carry then it is set to one otherwise it is reset to zero.
- S-Sign Flag: The sign flag is set if bit D7 of the result is $= 1$.
- P-Parity: If the result in the accumulator has an even number of 1s the this flag is set to one otherwise it is reset.
- AC-Auxiliary carry: In an arithmetic operation when a carry is generated by bit D3 into D4. Then AC is set to 1. This flag is used internal for binary coded operation (BCD). There are no jump instruction associated with this flag.

# The Program counter and the stack pointer

- These are two sixteen (16) bit registers used to hold memory addresses.
- The size of these register is 16 bits because the addresses are 16 bit
- The microprocessor uses PC to sequence execution of instructions
- The function of PC is to point to the memory address from with the next instruction (1 byte) is going to be fetched.
- After the byte is fetched the content of PC is automatically incremented by one to point to the next location.

# The Program counter and the stack pointer

Strathmore UNIVERSITY

- The stack pointer is also a 16 bit register used as a memory pointer.
- It points to the read write memory called stack
- The beginning of the stack is defined by loading a 16 bit address to the stack pointer.
- The stack is important when dealing with subroutines (functions)

# Microprocessor Architecture and Microcomputer system

- The microprocessor is a programmable device.
- It can be programmed to perform and operation on data.
- This instructions are given to the microprocessor by writing the in the memory
- All functions initiated by the microprocessor can be initiated in three groups.
  - Microprocessor initiated operation
  - Internal operation
  - Peripheral (external) operation

# Microprocessor Architecture and Microcomputer system

- The microprocessor performs primarily four operations
  1. Memory read: Read instructions or data from the memory
  2. Memory write: Writes data or instructions from the memory
  3. I/O read accepts data from input devices
  4. I/O write: Sends data to output devices
- All this are part of the communication from the peripherals
- To communicate from the peripherals the MPU needs to do the following.
  - Identify the peripheral and the memory location with its address
  - Transfer binary information data or instructions
  - Provide timing or synchronization signals

# Address bus

- The address bus is group of 16 bit lines generally identified as $A_0 to A_{15}$
- The address bus are unidirectional data flow in one direction. That is from the MPU to the peripheral devices

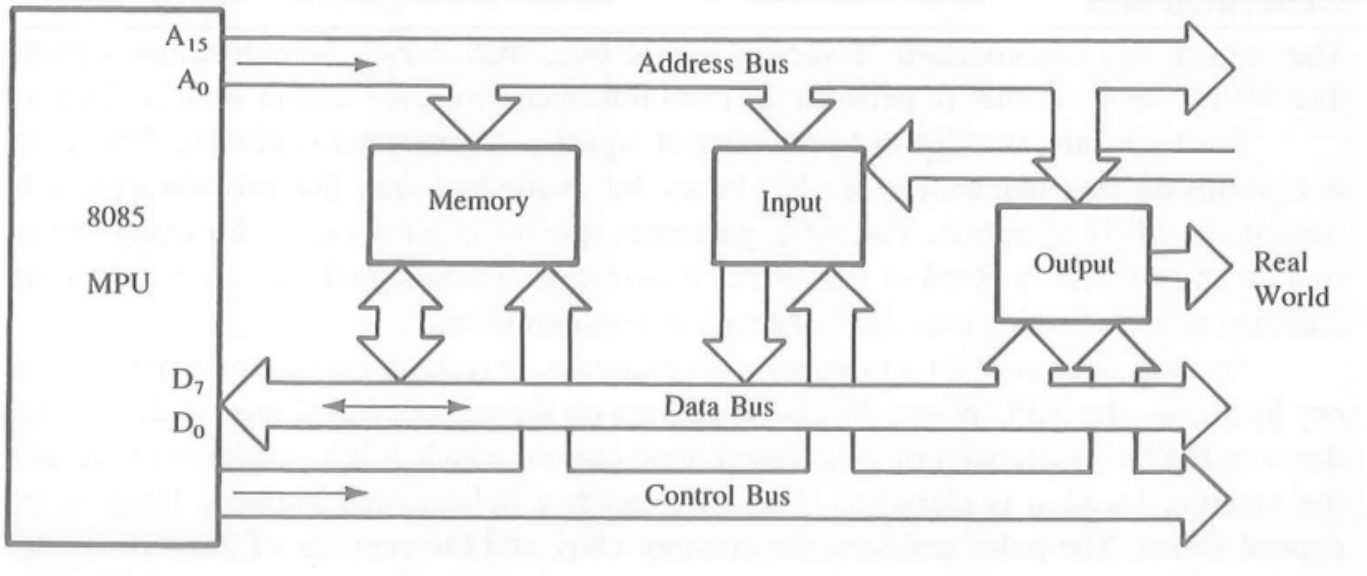



Figure 21: The Intel 8085 bus structure

- The 8085 with 16 address lines is capable of addressing upto $2^{16} = 65536$ known as 64K locations

# Data bus

- The data bus is a group of 8 lines used for data flow
- The lines are bidirectional - data can flow in both directions
- The 8 data lines enable the MPU to manipulate an 8 bit data ranging from $00H - FFH$
- The largest number that can appear on the data bus is 11111111 $(255_{10})$

# Control bus

- The control bus is comprised of various single lines that carry synchronization signals
- However control lines are not grouped together like address or data buses.
- They are individual lines which provide pulses which control operations
- It is the MPU which generates these control signals especially when it performs memory read or write operations
- These signals are used to identify the devices with which the MPU is communicating

# Memory read operation

- To communicate with memory, the MPU places a 16 bit address on the address lines
- The addresses are the decoded by an external logic circuit and the memory is identified
- The MPU sends a pulse called memory read as the control signal
- It is the MPU which generates these control signals especially when it performs memory read or write operations
- The pulse activates the memory chip and the contents of this memory are place on the 8 bit data bus and brought into the MPU

# Memory read operation

The fetch operation is shown in figure 22

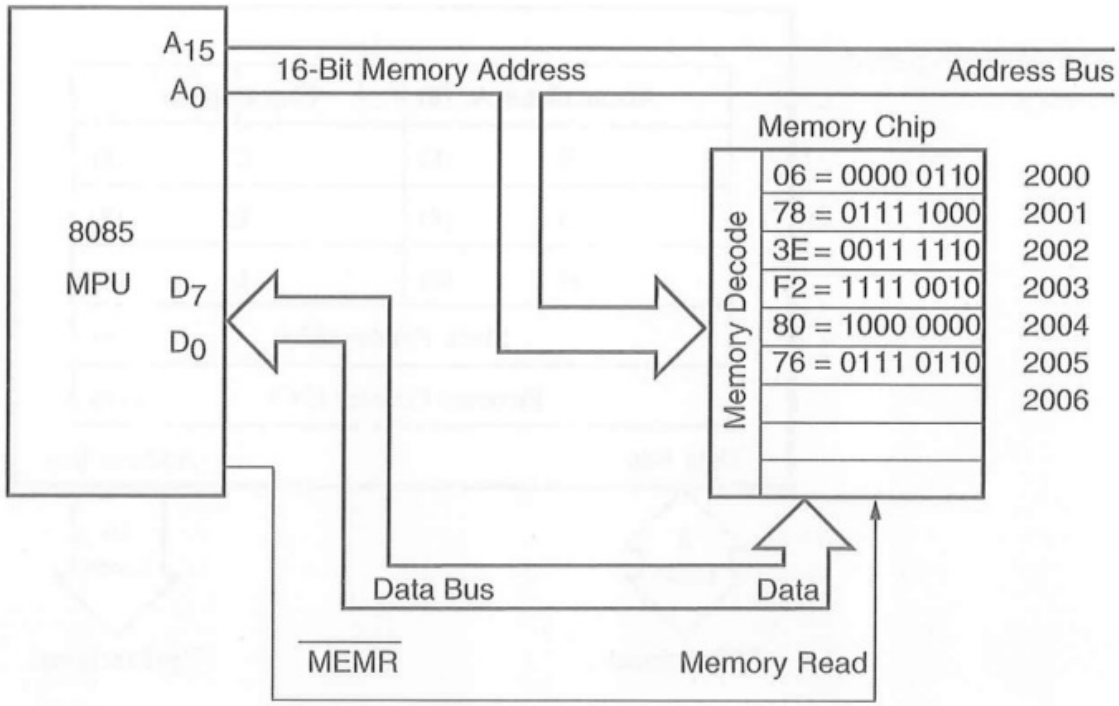



Figure 22: The Intel 8085 bus structure

# The Intel 8085 pin and functional diagram

The pin and functional diagram is shown in figure 23

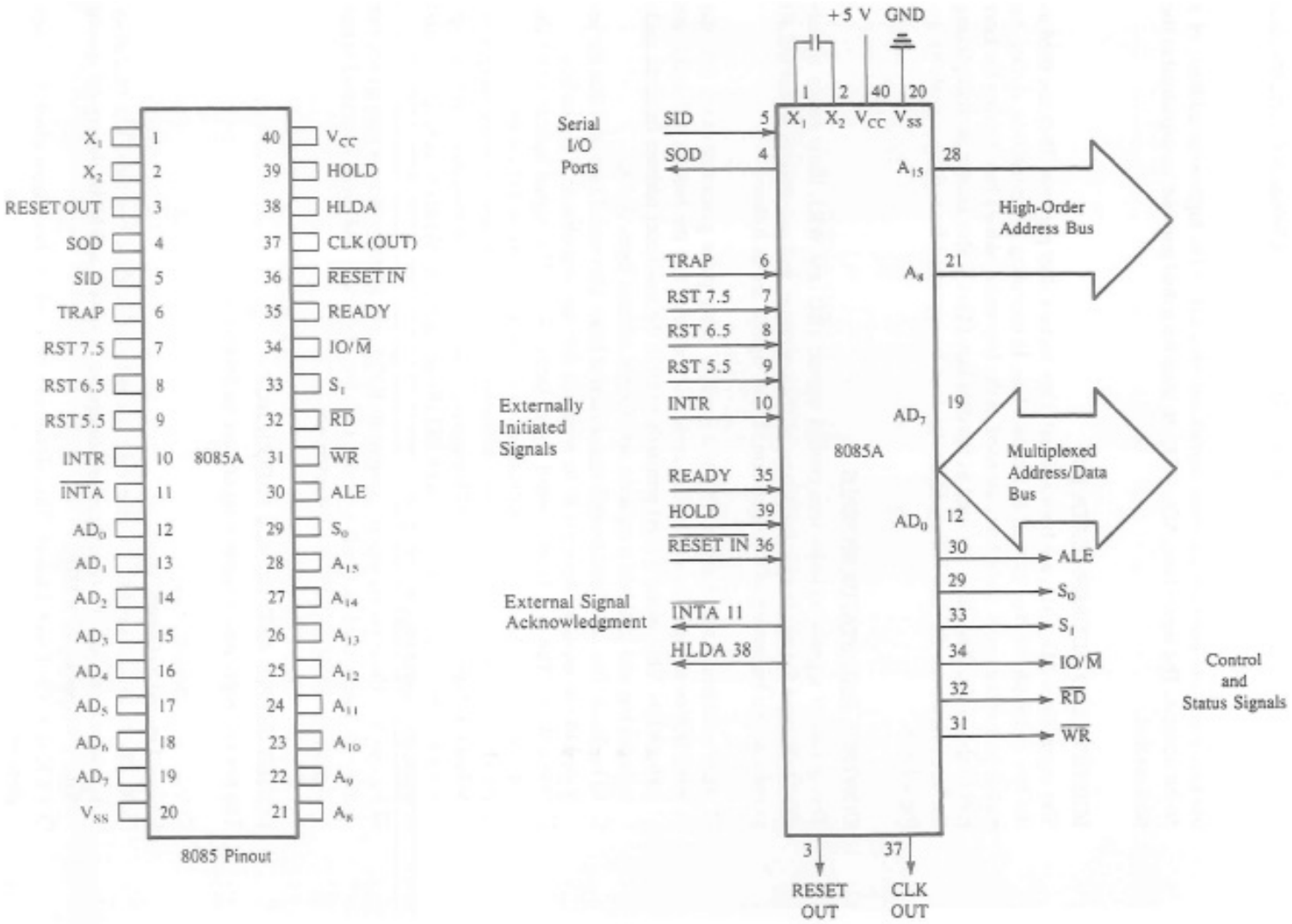



Figure 23: The Intel 8085 bus structure

# The Intel 8085 pin and functional diagram

## The buses

- The 16 lines(pins) are used for address bus however the lines are divided into two groups $A_{15} - A_8$ and $AD_7 - AD_0$
- The lines $A_{15} - A_8$ are unidirectional while $AD_7 - AD_0$ are bi-directional
- $AD_7 - AD_0$ are used for dual purpose. They serve dual purpose.
- They are used as the low-order address bus as well as the data bus.
- During the earlier part of the cycle these lines as low as low order address bus.
- During the later part of the cycle, these lines are used for data bus. (This is known as multiplexing )

# The Intel 8085 pin and functional diagram

## Control and Status Signals

- ALE-Address Latch Enable: This is a positive going pulse generated every time the 8085 begins and operation (machine cycle); it indicates that the bits on $AD_7 - AD_0$ are address bits.
- $\overline{RD}$Read: This is a Read control signal (active low). This signal indicates that the selected I/O or memory device is to be read and data is available on the data bus.
- $\overline{WR}$ - Write: This is a Write control signal (active low). This signal indicates that the data on the data bus are to be written into a selected memory or I/O
- $IO/\overline{M}$ This is a status signal used to differentiate between I/O and memory operations. When High-it is I/O operations. When low it indicates memory operations. This signal combines with $\overline{WR}$ and $\overline{RD}$ to generate memory and I/O operation.
- $S_1$ and $S_0$ These are status signals similar to $IO/\overline{M}$ can be used to identify different operations.

# The Intel 8085 pin and functional diagram

## Control and Status Signals

8085 Machine Cycle Status and Control Signals

| Machine Cycle | Status IO/$\overline{M}$ | Status $S_1$ | Status $S_0$ | Control Signals |
|---|---|---|---|---|
| Opcode Fetch | 0 | 1 | 1 | $\overline{RD} = 0$ |
| Memory Read | 0 | 1 | 0 | $\overline{RD} = 0$ |
| Memory Write | 0 | 0 | 1 | $\overline{WR} = 0$ |
| I/O Read | 1 | 1 | 0 | $\overline{RD} = 0$ |
| I/O Write | 1 | 0 | 1 | $\overline{WR} = 0$ |
| Interrupt Acknowledge | 1 | 1 | 1 | $\overline{INTA} = 0$ |
| Halt | Z | 0 | 0 | $\overline{RD}$, $\overline{WR}$ = Z and $\overline{INTA} = 1$ |
| Hold | Z | X | X | |
| Reset | Z | X | X | |

NOTE: Z = Tri-state (high impedance)
X = Unspecified

Figure 24: 8085 Machine Cycle Status and Control Signals

# The Intel 8085 pin and functional diagram

## Power Supply and Clock Frequency

- $V_{CC}$: +5V power supply
- $V_{SS}$: Ground reference
- $X_1, X_2$ Crystal clock- 6 MHz
- CLK(OUT): Clock output: This signal can be used as the system clock for other devices

# The Intel 8085 pin and functional diagram

## Interrupts and externally initiated signls

- INTR(input)- Interrupt request: this is used as a general purpose interrupt
- $\overline{INTA}$ (Output)- Interrupt acknowledge: This is used to acknowledge interrupt
- RST 7.5,RST 6.5, RST 5.5(Inputs) Restart Interrupts: These are vectored interrupts that transfer the program control to specific locations. They have higher priorities than the INTR interrupts. Among these three, the priority is 7.5,6.5,5.5
- TRAP Input (Input) This is a nonmaskable interrupt and has the highest priority.
- HOLD (Input): This signal indicates that a peripheral such as a DMA (Direct Memory Access) controller is requesting the use of the address and data buses.

# The Intel 8085 pin and functional diagram

## Interrupts and externally initiated signls

- HLDA (Output)- Hold Acknowledge: This signal acknowledge the HOLD request.
- Ready (Input): This signal si used to delay the microprocessor READ and WRITE cycles until a slow responding peripheral is ready to send or accept data.When this signal goes low, the microprocessor waits for a number of clock cycles until it goes high.
- $\overline{RESETIN}$:When the signal on this pin goes low, the program counter is set to zero and the MPU is reset.
- $\overline{RESETOUT}$: This signal indicates that the MPU is being reset.

# Memory Unit

- The memory is an essential component of a microcomputer.
- It stores binary instructions and data for the microprocessor.
- To communicate with memory the MPU should do the following.
  - Select the chip
  - Identify the register
  - Read from or write int the register.

# Flip-Flop or Latch as storage element

- A memory unit is a circuit that can store bits-high or low. Generally voltage levels or capacitive charge.
- A flip flop or a latch is a memory unit.
- Data is written through input data $D_{IN}$. The chip is enable through pin $EN$
- The data is read through pin $D_{OUT}$

# Flip-Flop or Latch as storage element

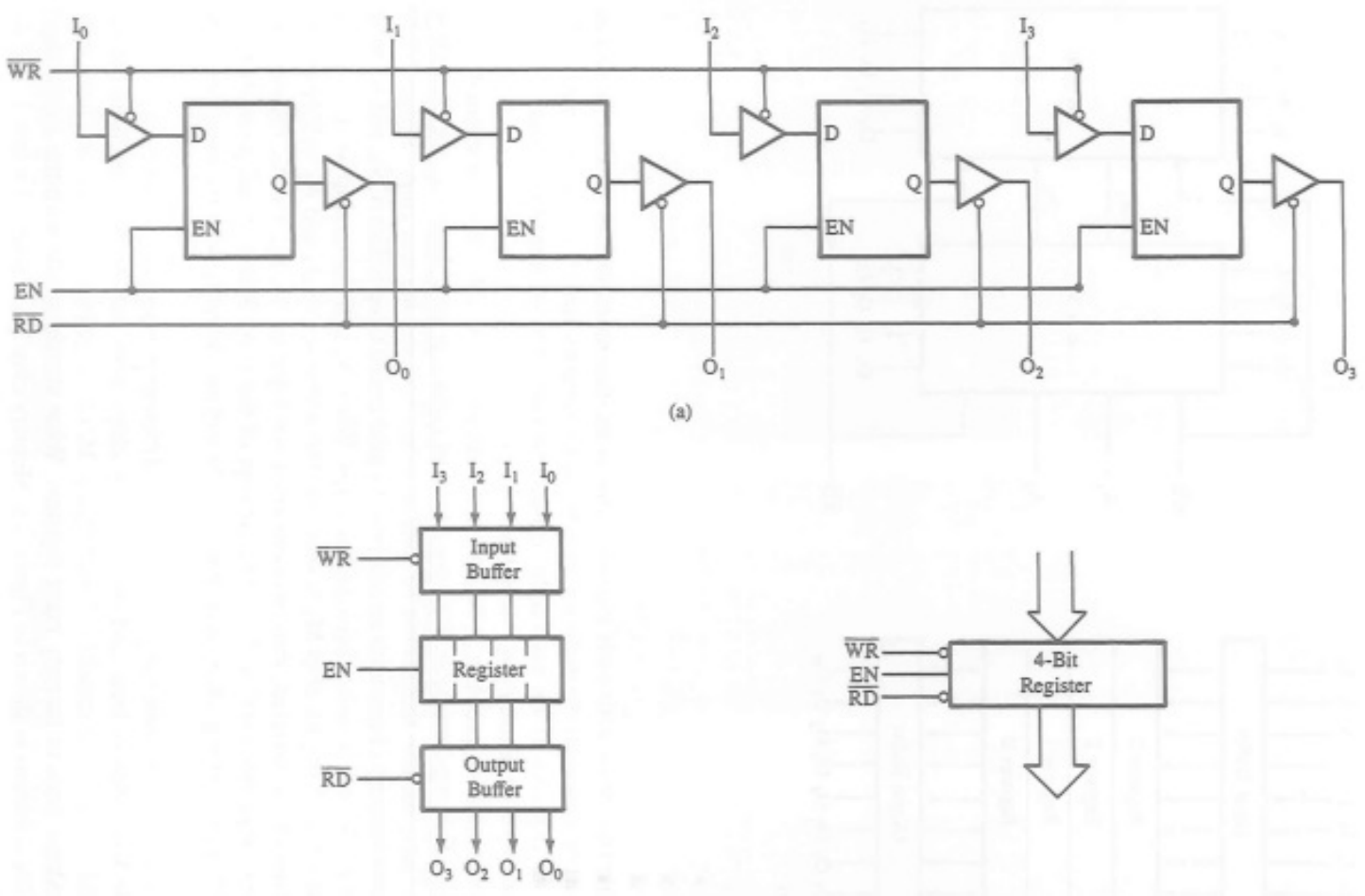



Figure 25: Latch as a four bit register

# 4X8 bit register

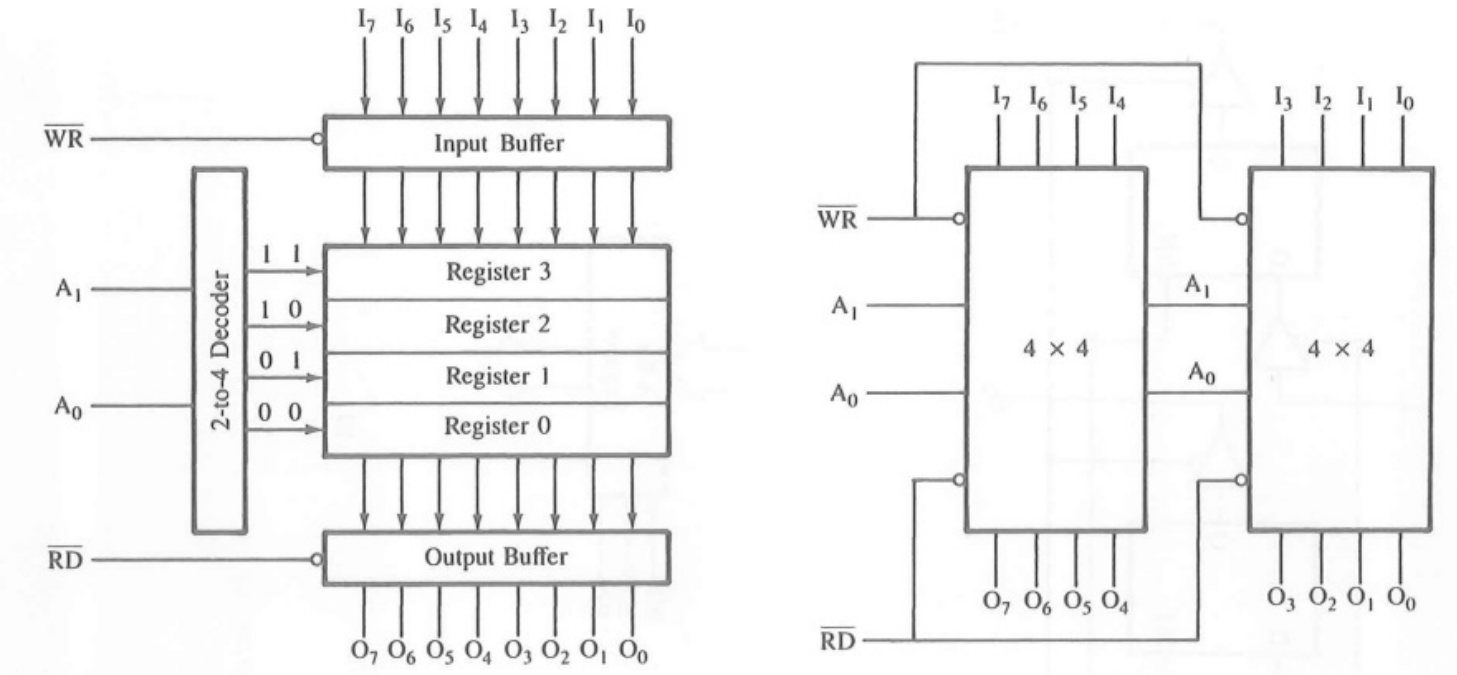



Figure 26: 4 X 8 bit register

# Flip-Flop or Latch

- Figure 25 Illustrates the arrangement of latches or flip flop in a 4 bit register.
- Each register can store a bit and can be activated by the $\overline{WR}$ and $\overline{RD}$ so that they can be written or read from.
- The $EN$ signal is part of the address bit the is used to enable a chip so that it can be written to or read from.

# 4X8 bit register

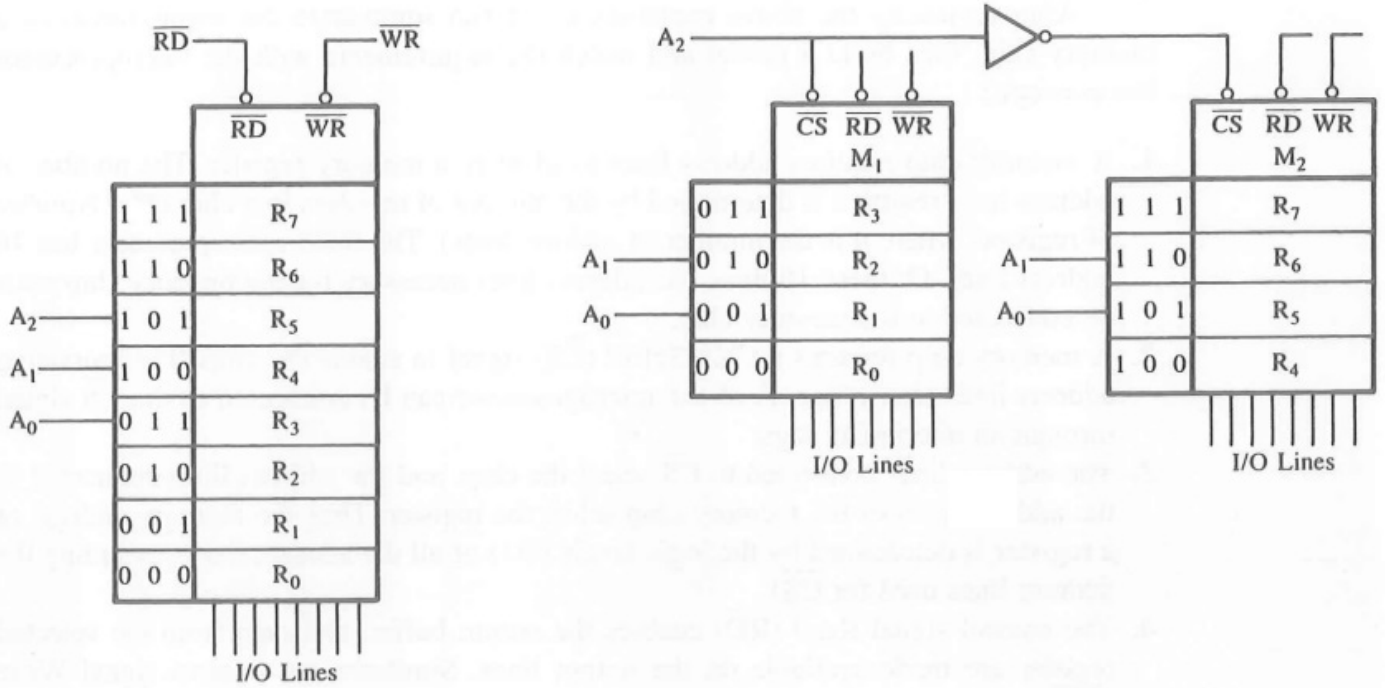



Figure 27: Two memory chips with four registers each and chip select

# RAM and ROM Chips

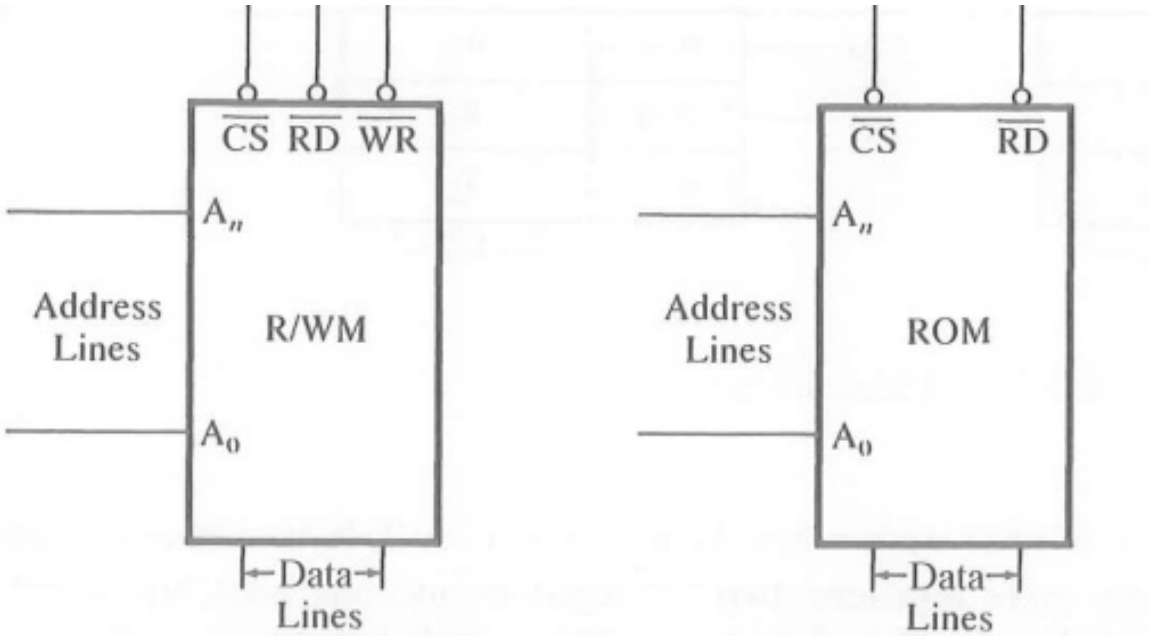



Figure 28: R/W memory model and ROM model

# Memory map and addresses

- Intel 8085 which is 8-bit has 16 address lines.
- It can identify upto 65536 memory registes each with 16 bit.
- A memory map is a pictorial representation in which memory devices are located in the entire range of addresses for each memory device
- Assume that a memory chip with 256 registers. Meaning we need only 256 out of 65236.
- The question is what should we do with the remaining address line of the microprocessor.

# Memory map and addresses illustration

- Figure 30 shows a memory chip with 256 registers and eight I/O lines.
- Memory size of the chip is expressed as 256 x 8.
- It has 8 address lines ($A_7 - A_0$), one chip select signal and two control signals $\overline{RD}$ and $\overline{WR}$
- Only 8 address lines $A_7 - A_0$ are required to identify 256 memory registers.
- The remaining eight lines $A_7 - A_0$ are connected to the chip select line through inverters and the NAND gate.

# Memory map and addresses illustration

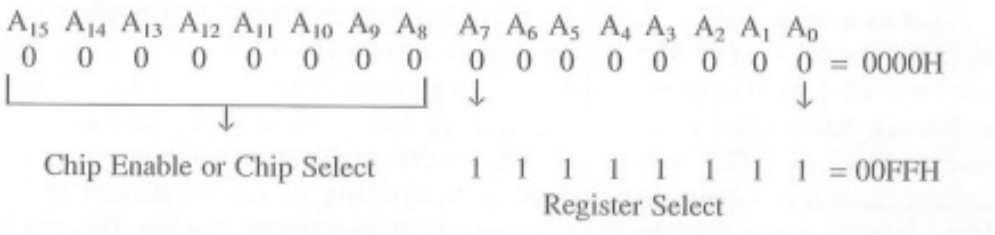



Figure 29: Memory map for 256 register memory

# Memory map and addresses illustration

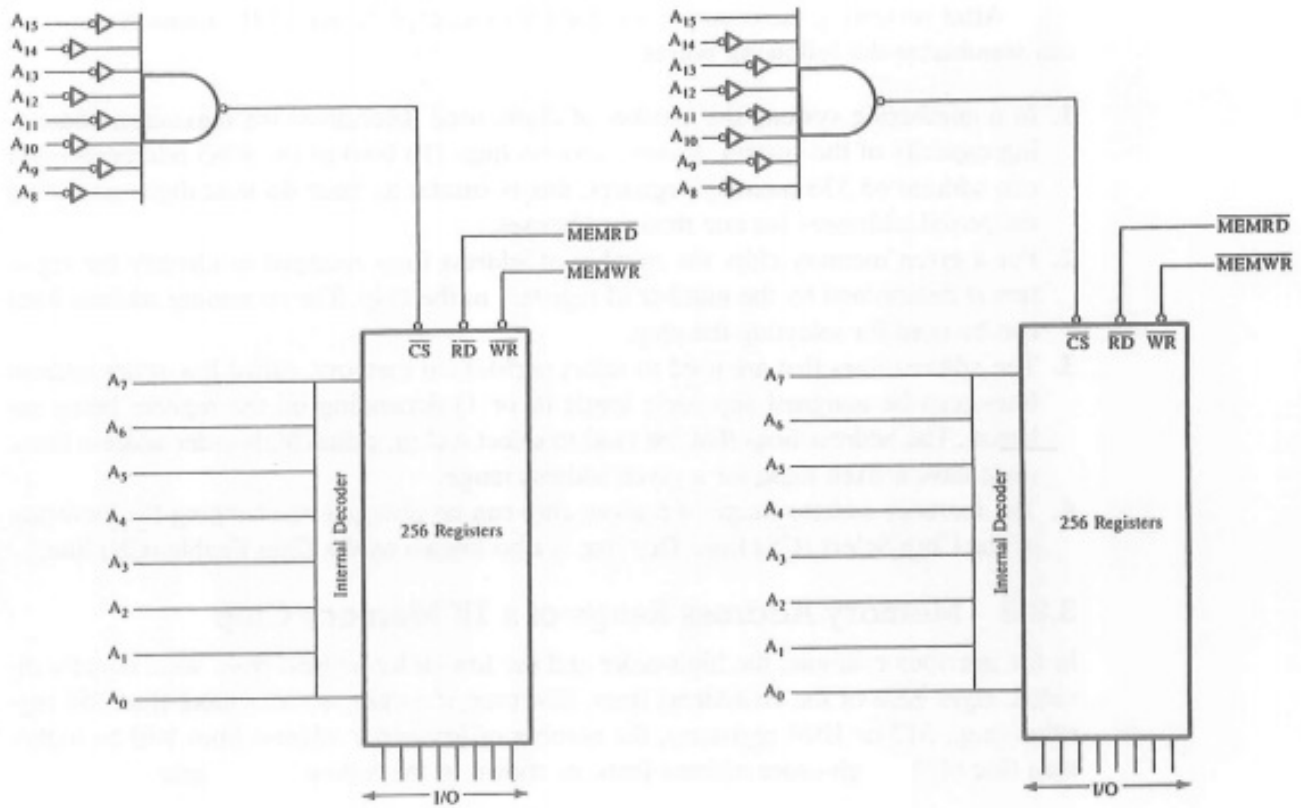



Figure 30: Address map for 256 register memory

## Example

Describe the address map with a diagram showing how 1K (1028x8) memory is mapped. Show all the control signals

# Memory word size

- Memory devices are aailable in various word sizes (1,4 and 8) and the size of the memory chip is generally specified in terms of the total number of bits it can store.
- For instance a memory chip of size 1024 x 4 has 1024 registers and each register has 4 bits, thus it can store a total of 4096 = 1024x4 bits.

## Example

- Calculate the number of memory chips needed to design 8K byte memory if the available chip size is 1024x1

# Input and Output devices

- There are two different methods by which I/O devices can be identified one uses 8-bit address and the other uses 16-bit address namely.
  - Peripheral-Mapped I/O (8bit)
  - Memory mapped I/O (16bit)
- Peripheral mapped I/O: Here MPU use 8 bit address lines to identify an input or an output device. The MPU can address upto 256 addresses.
- Memory mapped I/O: Here the MPU uses 16 address lines to identify an I/O device; an I/O is connected as if it is a memory register.
- The MPU uses the same control signal i.e memory read and memory write and instructions as those of the memory.

# Introduction to 8085 Instructions

## Data Transfer (Copy) Operations

- Several instructions are used to copy data. They are listed as follows

| Opcode | Operand | Description |
|---|---|---|
| MOV | Rd,Rs* | Move<br>☐ This is a 1-byte instruction<br>☐ Copies data from source register Rs to destination register Rd |
| MVI | R,8-bit* | Move Immediate<br>☐ This is a 2-byte instruction<br>☐ Loads the 8 bits of the second byte into the register specified |
| OUT | 8-bit port address | Output to Port<br>☐ This is a 2-byte instruction<br>☐ Sends (copies) the contents of the accumulator (A) to the output port specified in the second byte |
| IN | 8-bit port address | Input from Port<br>☐ This is a 2-byte instruction<br>☐ Accepts (reads) data from the input port specified in the second byte, and loads into the accumulator |

Figure 31: Data transfer instructions

# Introduction to 8085 Instructions

## Data Transfer (Copy) Operations

| | |
|---|---|
| HLT | Halt<br>☐ This is a 1-byte instruction<br>☐ The processor stops executing and enters wait state<br>☐ The address bus and data bus are placed in high impedance state. No register contents are affected |
| NOP | No Operation<br>☐ This is a 1-byte instruction<br>☐ No operation is performed<br>☐ Generally used to increase processing time or substitute in place of an instruction. When an error occurs in a program and an instruction needs to be eliminated, it is more convenient to substitute NOP than to reassemble the whole program |

Figure 32: Halt and NOP

# Introduction to 8085 Instructions

## Data Transfer (Copy) Operations

Strathmore UNIVERSITY

## Example

ASSEMBLY LANGUAGE PROGRAM

| Tasks | 8085 Mnemonics |
|---|---|
| 1. Load register B with 37H. | MVI B,37H* |
| 2. Copy the number from B to A. | MOV A,B |
| 3. Send the number to the output—port 01H. | OUT PORT1 |
| 4. End of the program. | HLT |

TRANSLATION FROM ASSEMBLY LANGUAGE TO MACHINE LANGUAGE

Now, to translate the assembly language program into machine language, look up the hexadecimal machine codes for each instruction in the 8085 instruction set and write each machine code in the sequence, as follows:

| 8085 Mnemonics | Hex Machine Code |
|---|---|
| 1. MVI B,37H | 06 |
| | 37 |
| 2. MOV A,B | 78 |
| 3. OUT PORT1 | D3 |
| | 01 |
| 4. HLT | 76 |

Figure 33: Example

# Introduction to 8085 Instructions

## Arithmetic Operations

- The 8085 microprocessor performs various arithmetic operations, such as addition, subtraction,increment and decrement. The operations have the following mnemonics.

| Opcode | Operand | Description |
|---|---|---|
| ADD | R† | Add<br>☐ This is a 1-byte instruction<br>☐ Adds the contents of register R to the contents of the accumulator |
| ADI | 8-bit | Add Immediate<br>☐ This is a 2-byte instruction<br>☐ Adds the second byte to the contents of the accumulator |
| SUB | R† | Subtract<br>☐ This is a 1-byte instruction<br>☐ Subtracts the contents of register R from the contents of the accumulator |
| SUI | 8-bit | Subtract Immediate<br>☐ This is a 2-byte instruction |

†R represents any of registers A, B, C, D, E, H, and L.

Figure 34: Arithmetic instructions

# Introduction to 8085 Instructions

## Arithmetic Operations

- The 8085 microprocessor performs various arithmetic operations, such as addition, subtraction,increment and decrement. The operations have the following mnemonics.

| | | |
|---|---|---|
| | | ☐ Subtracts the second byte from the contents of the accumulator |
| INR | R* | Increment<br>☐ This is a 1-byte instruction<br>☐ Increases the contents of register R by 1<br>*Caution:* All flags except the CY are affected |
| DCR | R* | Decrement<br>☐ This is a 1-byte instruction<br>☐ Decreases the contents of register R by 1<br>*Caution:* All flags except the CY are affected |

Figure 35: Arithmetic instructions

# Introduction to 8085 Instructions

## Arithmetic Operations

- An example to add two numbers

**Program:**

| Address | Mnemonics | Operand | Opcode | Remarks |
|---|---|---|---|---|
| 2000 | LXI | H, 3000H | 21 | Load H-L pair with address 3000H. |
| 2001 | | | 00 | Lower-order of 3000H. |
| 2002 | | | 30 | Higher-order of 3000H. |
| 2003 | MOV | A, M | 7E | Move the 1st operand from memory to reg. A. |
| 2004 | INX | H | 23 | Increment H-L pair. |
| 2005 | MOV | B, M | 46 | Move the 2nd operand from memory to reg. B. |
| 2006 | ADD | B | 80 | Add B with A. |
| 2007 | INX | H | 23 | Increment H-L pair. |
| 2008 | MOV | M, A | 77 | Move the result from reg. A to memory. |
| 2009 | HLT | | 76 | Halt. |

Figure 36: Addition of two numbers

# Introduction to 8085 Instructions

## Logical Operations

A microprocessor is basically a programmable logic chip. It can perform all the logic functions of the hard-wired logic through its instruction set. The 8085 instruction set includes such logic functions as AND, OR, Ex OR, and NOT (complement). The opcodes of these operations are as follows:*

| | | |
|---|---|---|
| ANA: | AND | Logically AND the contents of a register. |
| ANI : | AND Immediate | Logically AND 8-bit data. |
| ORA: | OR | Logically OR the contents of a register. |
| ORI : | OR Immediate | Logically OR 8-bit data. |
| XRA: | X-OR | Exclusive-OR the contents of a register. |
| XRI : | X-OR Immediate | Exclusive-OR 8-bit data. |

All logic operations are performed in relation to the contents of the accumulator. The instructions of these logic operations are described below.

Figure 37: The group of logical instructions

# Introduction to 8085 Instructions

## Logical Operations

| Opcode | Operand | Description |
|---|---|---|
| ANA | R | Logical AND with Accumulator<br>☐ This is a 1-byte instruction<br>☐ Logically ANDs the contents of the register R with the contents of the accumulator<br>☐ 8085: CY is reset and AC is set |
| ANI | 8-bit | AND Immediate with Accumulator<br>☐ This is a 2-byte instruction<br>☐ Logically ANDs the second byte with the contents of the accumulator<br>☐ 8085: CY is reset and AC is set |
| ORA | R | Logically OR with Accumulator<br>☐ This is a 1-byte instruction |

Figure 38: The group of logical instructions

# Introduction to 8085 Instructions

## Logical Operations

| | | |
|---|---|---|
| ORI | 8-bit | OR Immediate with Accumulator<br>☐ This is a 2-byte instruction<br>☐ Logically ORs the second byte with the contents of the accumulator |
| XRA | R | Logically Exclusive-OR with Accumulator<br>☐ This is a 1-byte instruction<br>☐ Exclusive-ORs the contents of register R with the contents of the accumulator |
| XRI | 8-bit | Exclusive-OR Immediate with Accumulator<br>☐ This is a 2-byte instruction<br>☐ Exclusive-ORs the second byte with the contents of the accumulator |
| CMA | | Complement Accumulator<br>☐ This is a 1-byte instruction that complements the contents of the accumulator<br>☐ No flags are affected |

Figure 39: The group of logical instructions

# Introduction to 8085 Instructions

## Logical Operations: Illustrative or ORing data from two inputs ports

- An additional input port with eight switches and the address 01H is connected to the microcomputer to control the same appliances and lights from the bedroom as well as from the kitchen. Write instructions to turn on the devices from any of the input ports
- To turn on the appliances from any one of the input port, the microprocessor needs to read the switches at both ports and logically OR the switch positions

# Introduction to 8085 Instructions

## Logical Operations

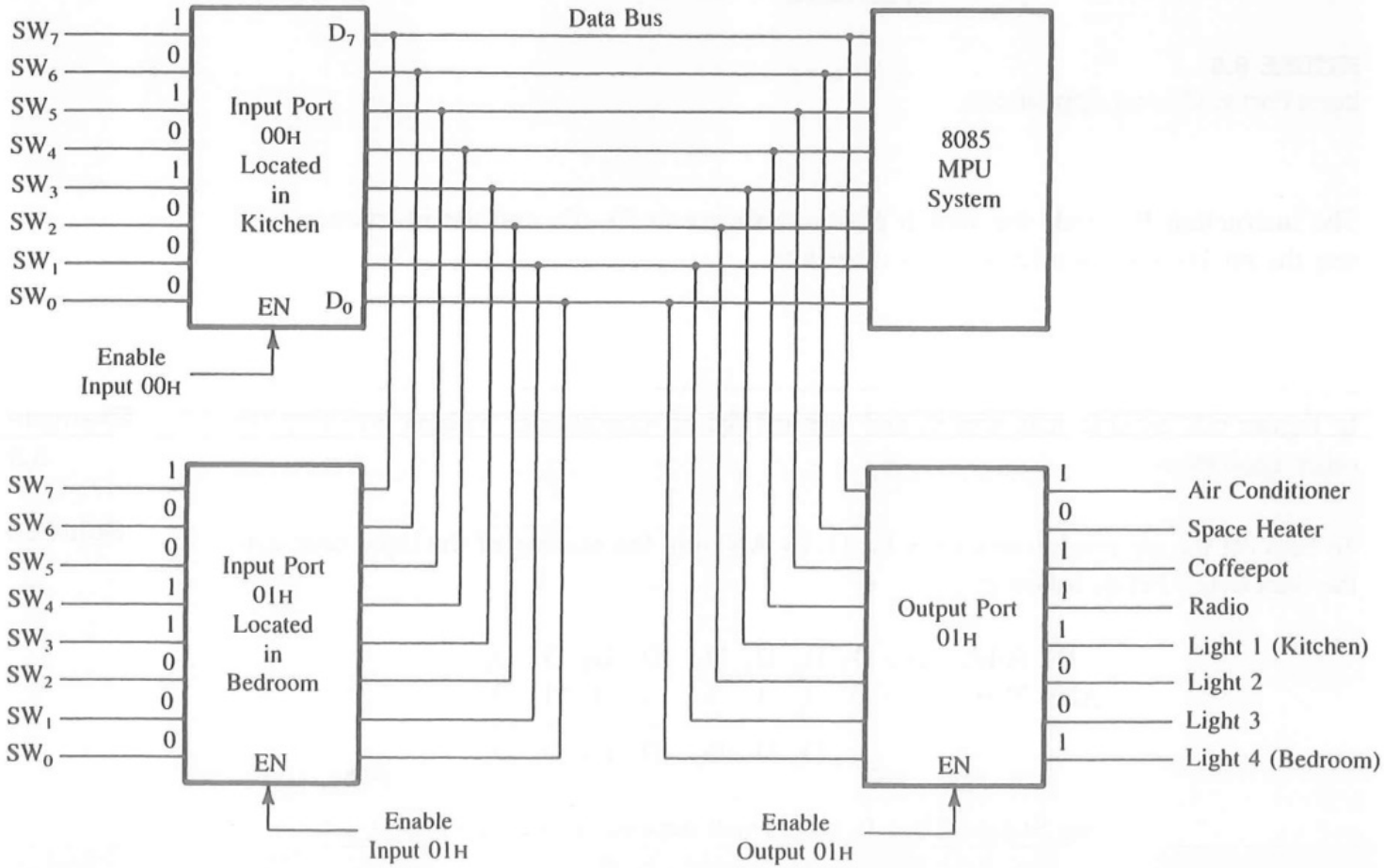



Figure 40: The diagram for switch control

# Introduction to 8085 Instructions

## Logical Operations

PROGRAM

| Memory Address | Machine Code | Instructions Opcode | Operand | Comments |
|---|---|---|---|---|
| HI-LO | | | | |
| XX00 | 06 | MVI | B,91H | ;This instruction simulates read- |
| 01 | 91 | | | ; ing input port 01H |
| 02 | 0E | MVI | C,A8H | ;This instruction simulates read- |
| 03 | A8 | | | ; ing input port 00H |
| 04 | 78 | MOV | A,B | ;It is necessary to transfer data |
| | | | | ; byte from B to A to OR with |
| | | | | ; C. B and C cannot be ORed |
| | | | | ; directly |
| 05 | B1 | ORA | C | ;Combine the switch positions |
| | | | | ; from registers B and C in the |
| | | | | ; accumulator |
| 06 | D3 | OUT | PORT1 | ;Turn on appliances and lights |
| 07 | PORT1 | | | |
| 08 | 76 | HLT | | ;End of the program |

PROGRAM OUTPUT

By logically ORing the data bytes in registers B and C

$$
\begin{aligned}
(B) \rightarrow (A) &= 1\ 0\ 0\ 1\ \ 0\ 0\ 0\ 1 \quad (91H)\\
(C) &= 1\ 0\ 1\ 0\ \ 1\ 0\ 0\ 0 \quad (A8H)\\
\hline
(A) &= 1\ 0\ 1\ 1\ \ 1\ 0\ 0\ 1 \quad (B9H)
\end{aligned}
$$

Flag Status: S = 1, Z = 0, CY = 0

Figure 41: The program for switch control

# Introduction to 8085 Instructions

## Branch instructions

INSTRUCTIONS

All conditional Jump instructions in the 8085 are 3-byte instructions; the second byte specifies the low-order (line number) memory address, and the third byte specifies the high-order (page number) memory address. The following instructions transfer the program sequence to the memory location specified under the given conditions:

| Opcode | Operand | Description |
|---|---|---|
| JC | 16-bit | Jump On Carry (if result generates carry and CY = 1) |
| JNC | 16-bit | Jump On No Carry (CY = 0) |
| JZ | 16-bit | Jump On Zero (if result is zero and Z = 1) |
| JNZ | 16-bit | Jump On No Zero (Z = 0) |
| JP | 16-bit | Jump On Plus (if $D_7 = 0$, and S = 0) |
| JM | 16-bit | Jump On Minus (if $D_7 = 1$, and S = 1) |
| JPE | 16-bit | Jump On Even Parity (P = 1) |
| JPO | 16-bit | Jump On Odd Parity (P = 0) |

Figure 42: Group of branch instructions

# Introduction to 8085 Instructions

## Branch instructions

### Example

Load the hexadecimal numbers 9BH and A7H in the registers D and E, respectively, and add the numbers. If the sum is greater FFH, display 01H at port PORT0; otherwise display the sum.

| Memory Address | Machine Code | Label | Mnemonics |
|---|---|---|---|
| 2000 | 16 | START: | MVI D,9BH |
| 2001 | 9B | | |
| 2002 | 1E | | MVI E,A7H |
| 2003 | A7 | | |
| 2004 | 7A | | MOV A,D |
| 2005 | 83 | | ADD E |
| 2006 | D2 | | JNC DSPLAY |
| 2007 | X | | |
| 2008 | X | | |
| 2009 | 3E | | MVI A,01H |
| 200A | 01 | | |
| 200B | D3 | DSPLAY: | OUT 00H |
| 200C | 00 | | |
| 200D | 76 | | HLT |

Figure 43: Solution

# Microprocessor Architecture and Memory Interfacing

## Data flow and Timing diagrams

- To understand how the microprocessor works its important to look at the timing diagrams.
- Consider the following scenario:

### Example

Illustrate the steps and timing when the and the data flow when the instruction code 0100 1111-(4F MOV C,A) stored in location 2005H is being fetched

# Microprocessor Architecture and Memory Interfacing

## Data flow and Timing diagrams

### Example

- Step 1: The MPU places the 16 bit address (2005H) on the address bus.
- Step 2: The control unit sends the control signal $\overline{RD}$ to enable the chip.
- Step 3: When The memory is enabled, 4FH is place on the data bus.
- Step 4: The byte (4FH) is the placed on the instruction decoder and then the MPU performs the function as in the instruction.

# Microprocessor Architecture and Memory Interfacing

## Demultiplexing low order address

### Example

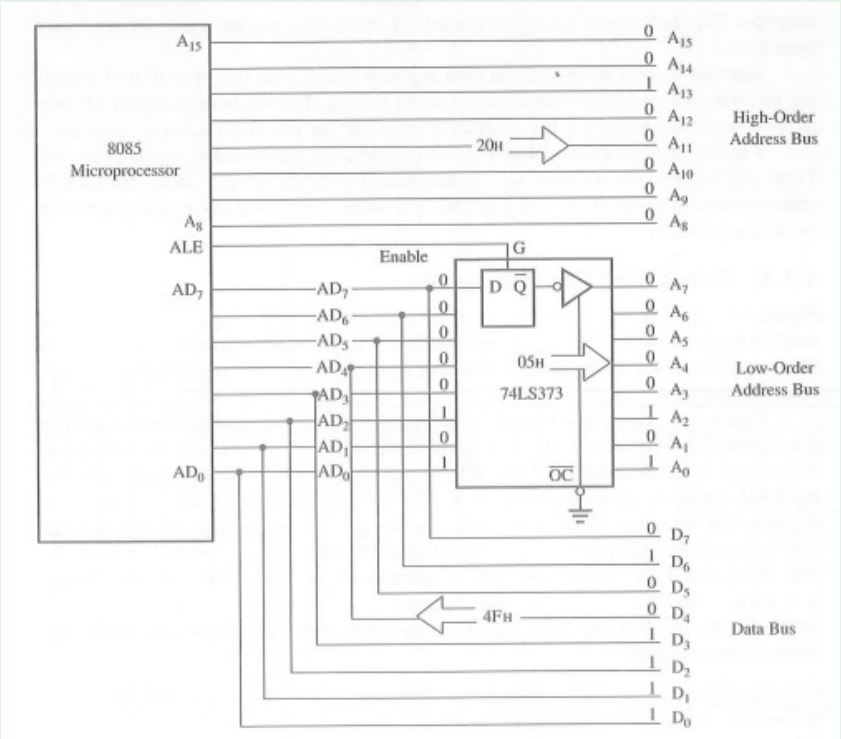



Figure 44: Schematic of latching of low order address

# Microprocessor Architecture and Memory Interfacing

## Data flow and Timing diagrams

### Example

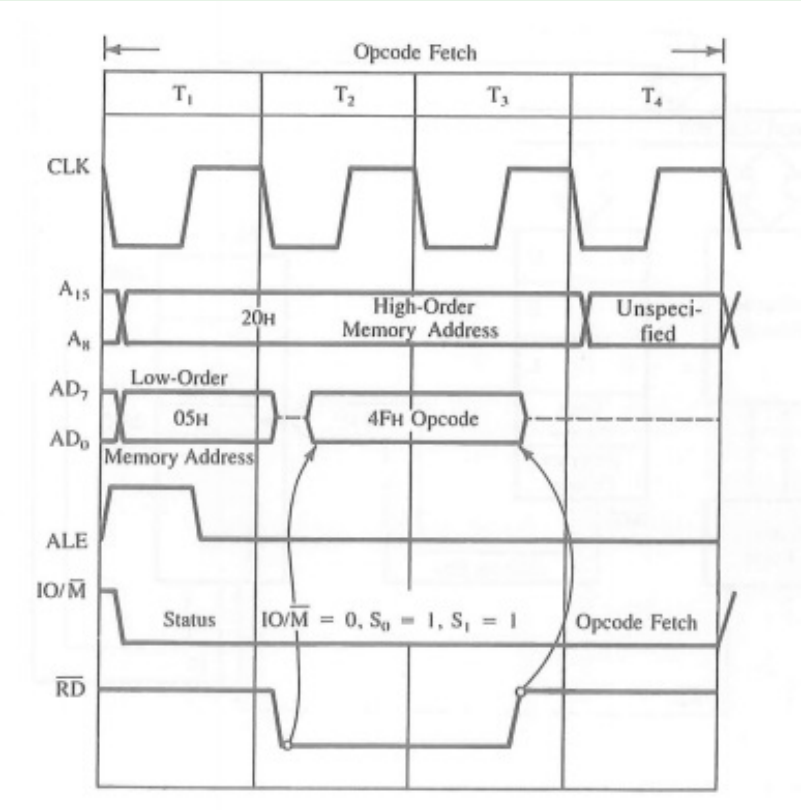



Figure 45: Opcode fetch timing diagram

# Microprocessor Architecture and Memory Interfacing

## Generation of control signals

- There are three elements that need to be defined.
  - T-state: This is the subdivision of the operation performed in one clock cycle. One T-state is equal to one clock cycle.
  - Machine cycle: This is the time require to complete one operation of accessing memory, I/O or acknowledge request.
  - Instruction cycle: This is the time required to complete the execution of an instruction. One instruction cycle consists of one to six machine cycle.

# Microprocessor Architecture and Memory Interfacing

## Generation of control signals

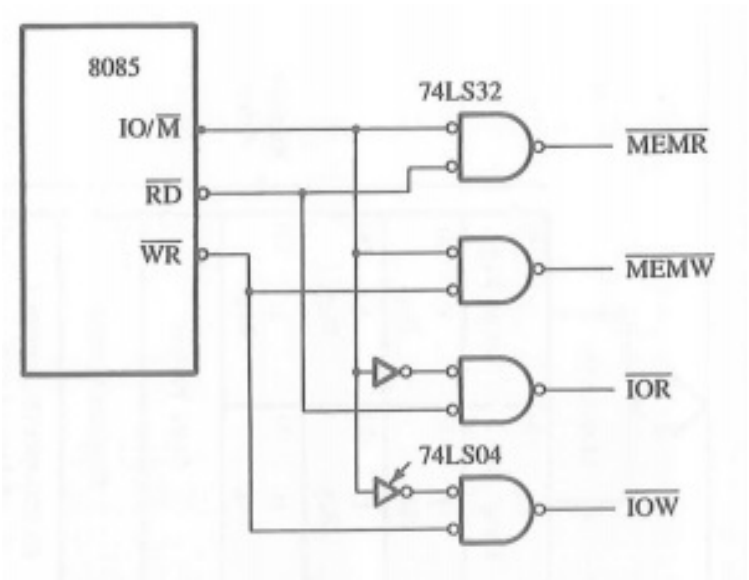



Figure 46: Schematic to generate read/write control signals for memory and I/O

# Microprocessor Architecture and Memory Interfacing

## Intel 8085 functional block diagram

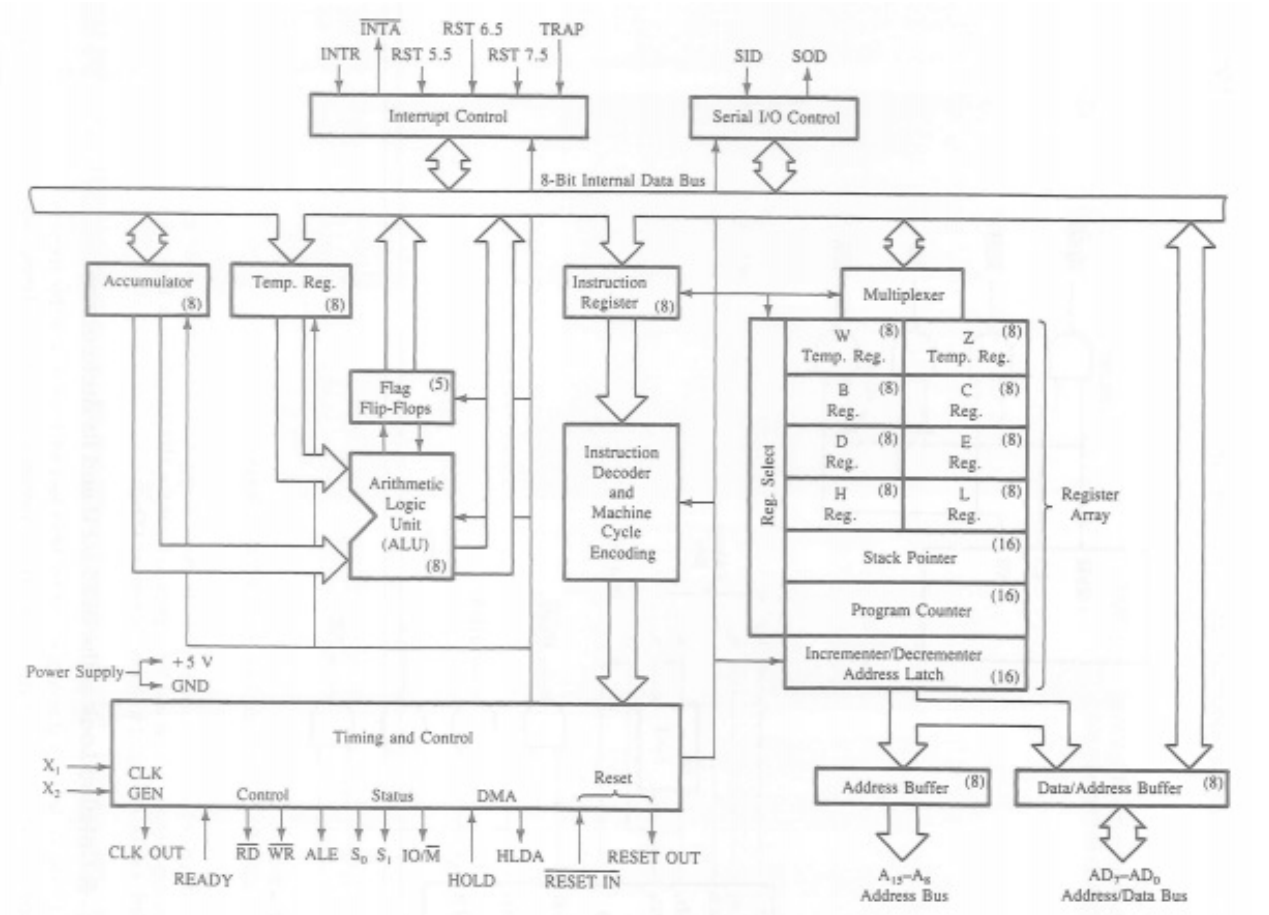



Figure 47: The functional block diagram of intel 8085

# Interfacing the 8155 Memory Mapped I/O

- It has eight address lines, one $\overline{CE}$ (Chip Enable) line and five lines compatible with the control and status signals of the 8085
- The five control lines include $IO/\overline{M}$, ALE, $\overline{RD}$,$\overline{WR}$ and RESET.
- These lines eliminate the need for external demultiplexing of the bus $AD_7 - AD_0$ and for generating separate control signals for memory and I/O
- The memory section includes 256x8 memory locations and an internal latch to demultiplex the bus line $AD_7 - AD_0$.
- The memory section also requires a Chip Enable ($\overline{CE}$) and a memory write $\overline{MEMW}$ and memory read $\overline{MEMR}$ control signals.

# Interfacing the 8155 Memory Mapped I/O

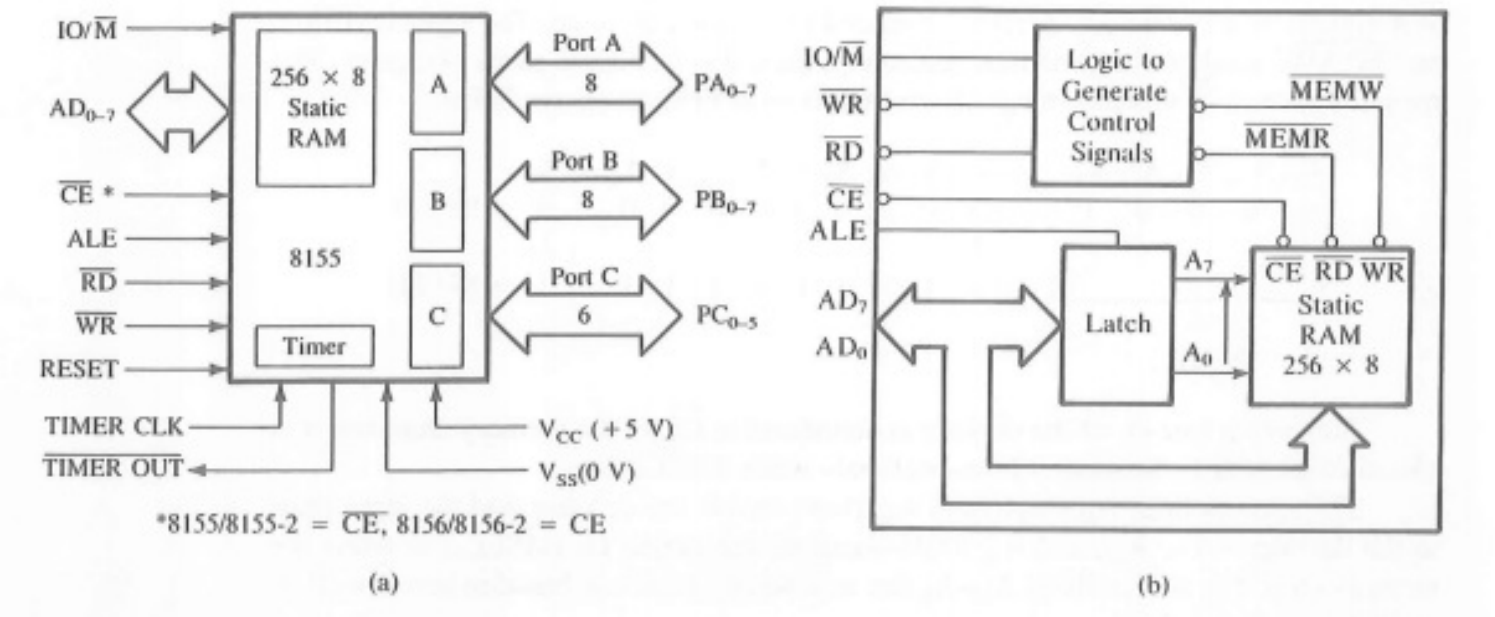



Figure 48: Intel 8155 programmable I/O with the internal architecture

# Interfacing the 8155 Memory Mapped I/O

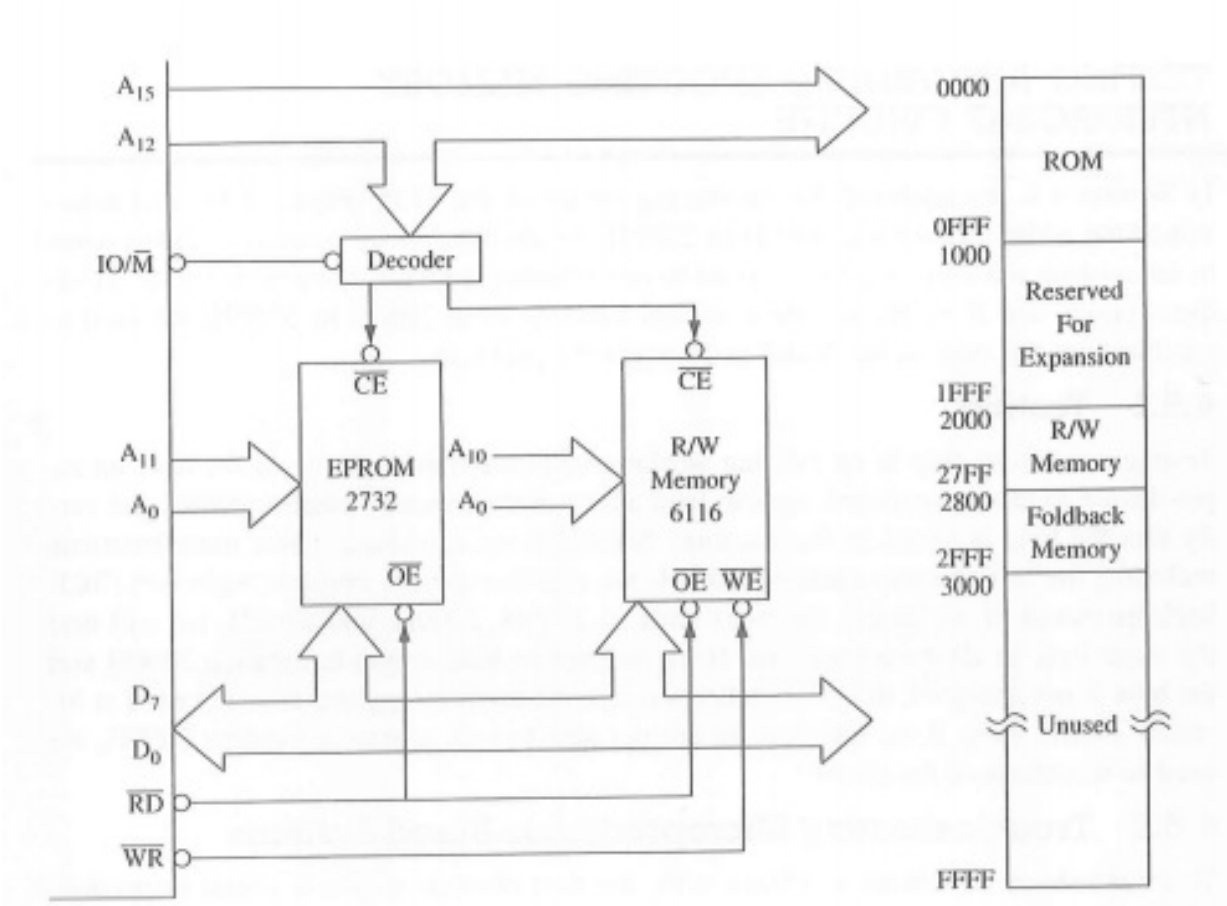



Figure 49: Memory System and mapping

# INTERFACING I/O

Strathmore UNIVERSITY

- Intel 8085 has two instructions for data transfer between the microprocessor and the I/O devices IN and OUT.
- The IN instruction is used to read data from an input device such as keyboard into the accumulator.
- The OUT instruction is used to send data from the accumulator to an output device such as an LED display.
- Each of this instructions are two byte. e.g IN 80H or OUT 81H

# OUT Instruction timing diagram

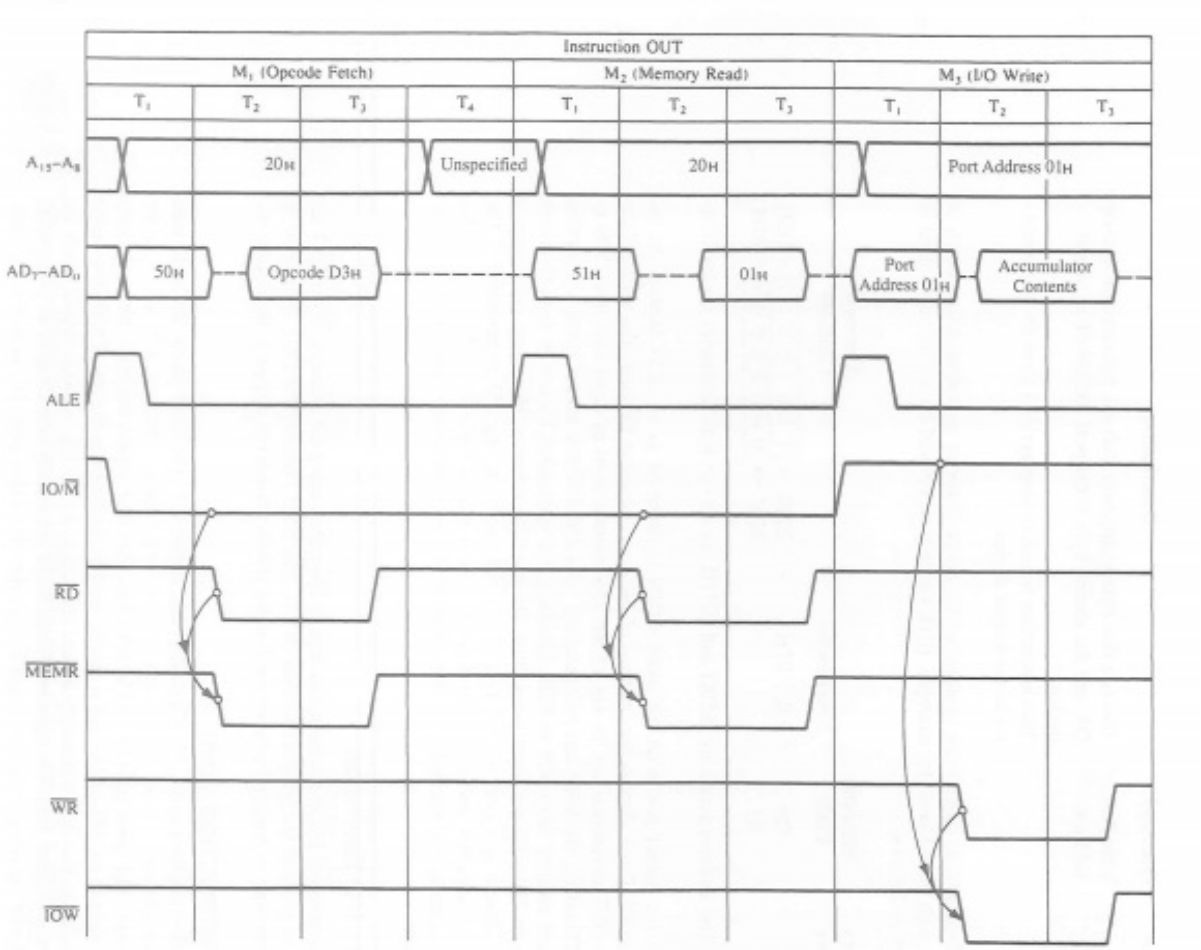



Figure 50: OUT instruction timing diagram

# OUT Instruction steps

| Memory Address | Machine Code | Mnemonics | | Memory Contents |
|---|---|---|---|---|
| 2050 | D3 | OUT 01H | ; 2050 → | 1 1 0 1 0 0 1 1 = D3H |
| 2051 | 01 | | ; 2051 → | 0 0 0 0 0 0 0 1 = 01H |

(Note: The memory locations 2050H and 2051H are chosen here arbitrarily for the illustration.)

Figure 51: OUT instruction timing diagram

# OUT Instruction steps

- In the first machine cycle M1 (Opcode fetch) Intel 8085 places the high order address byte 50H on the $A_{15} - A_8$ and the lower address 50H on $AD_8 - AD_0$
- At the same time ALE goes high $IO/\overline{M}$ goes low which indicates availability of lower address buses and the operation being a memory operation.
- At $T_2$ the microprocessor sends the $\overline{RD}$ control signal which is combined by $IO/\overline{M}$ to generate $\overline{MEMR}$ and the processor fetches the instruction using the data bus.
- In the second machine cycle $M_2$ (Memory read) the 8085 places the next address 51H on the address line and fetches the IO address 01H via the data bus.
- In the third machine cycle $M_3$ (IO write) the 8085 place the IO address on the address line $AD_7 - AD_0$

# I/O Interfacing

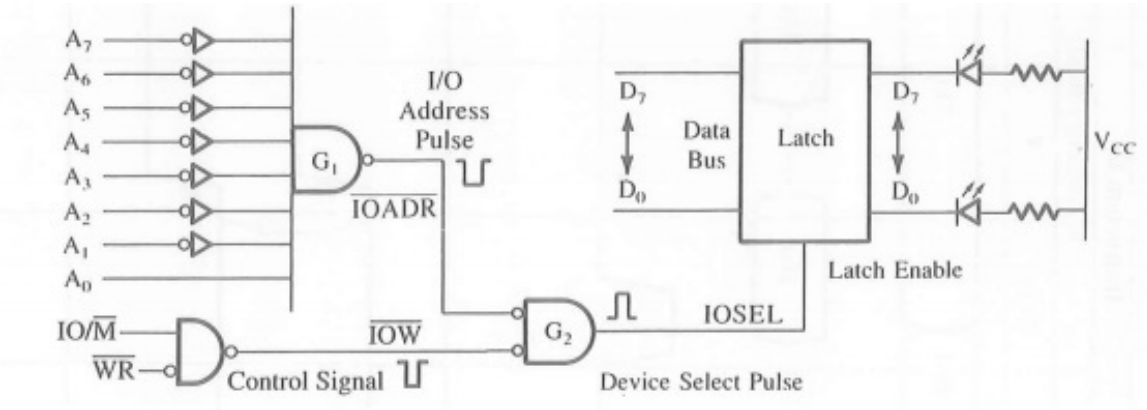



Figure 52: Interfacing an IO absolute decoding

- When address bus carries address the gate $G_1$ and $G_2$ are combined to generate $\overline{IOW}$
- When the contents of the accumulator are placed on the data bus the output latch is enable to display the data on the LED

# Absolute vs Partial decoding

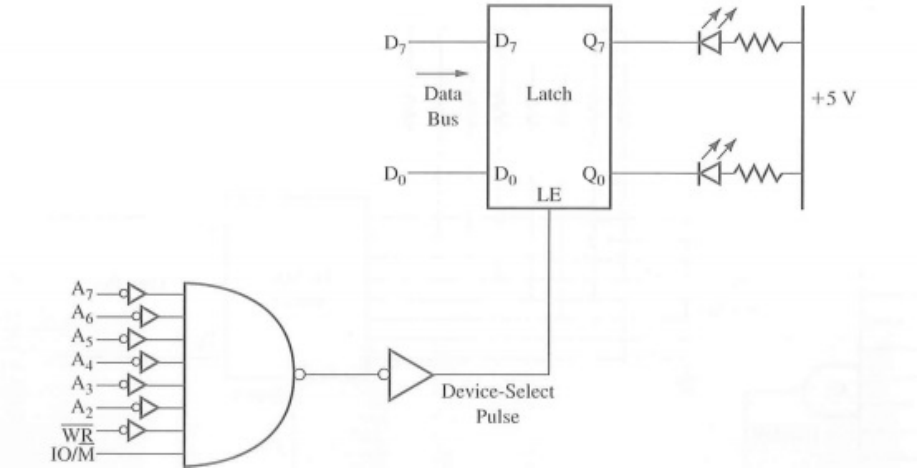



Figure 53: Interfacing an IO - **partial decoding**

- In figure 55 all the 8 address lines are combined to generate one pulse for enabling the IO latch- This called **absolute decoding**
- In the figure 53 only some address lines are combined to generate the IOSEL pulse. $A_1 - A_0$ are not used instead they are replaced $IO/\overline{M}$ and $\overline{WR}$

# Input Interface

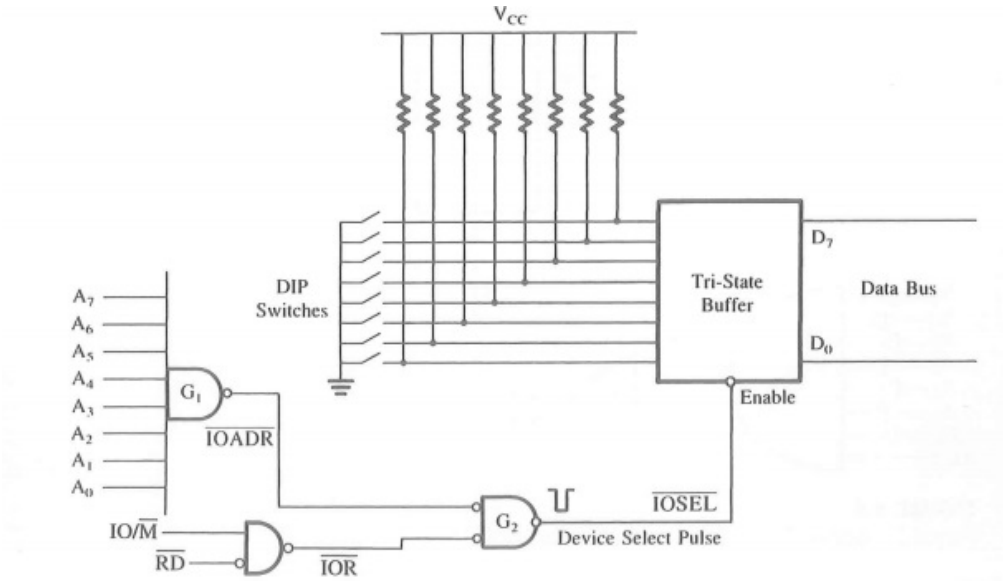



Figure 54: Interfacing an Input

- Data from the keys are put on the data bus $D_7 - D_0$ and loaded into the accumulator.

# Interfacing IO using decoders

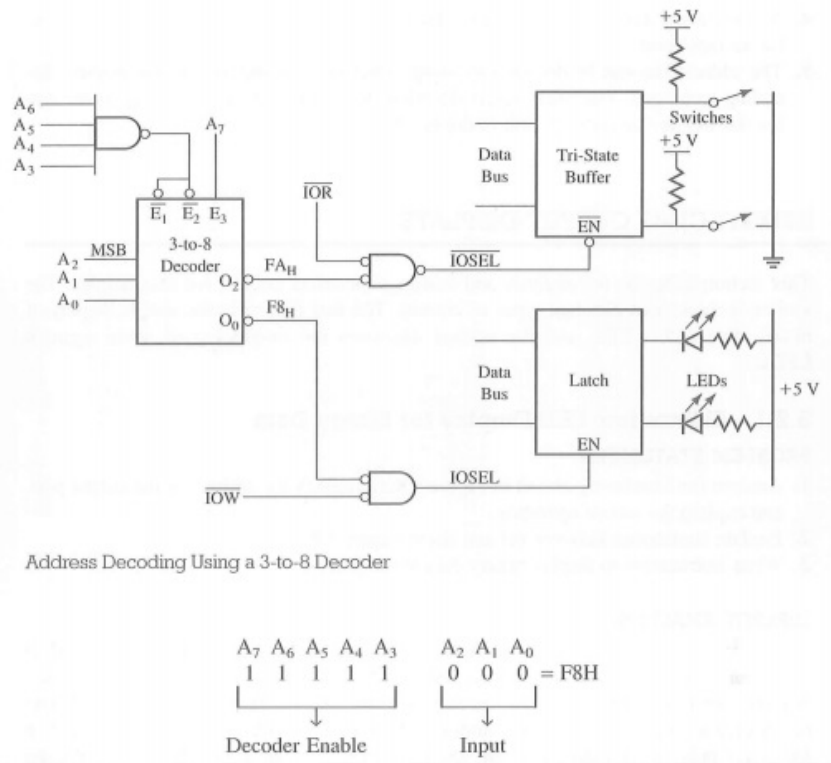



Figure 55: Interfacing an Input and output using decoder

# Interfacing IO using decoders

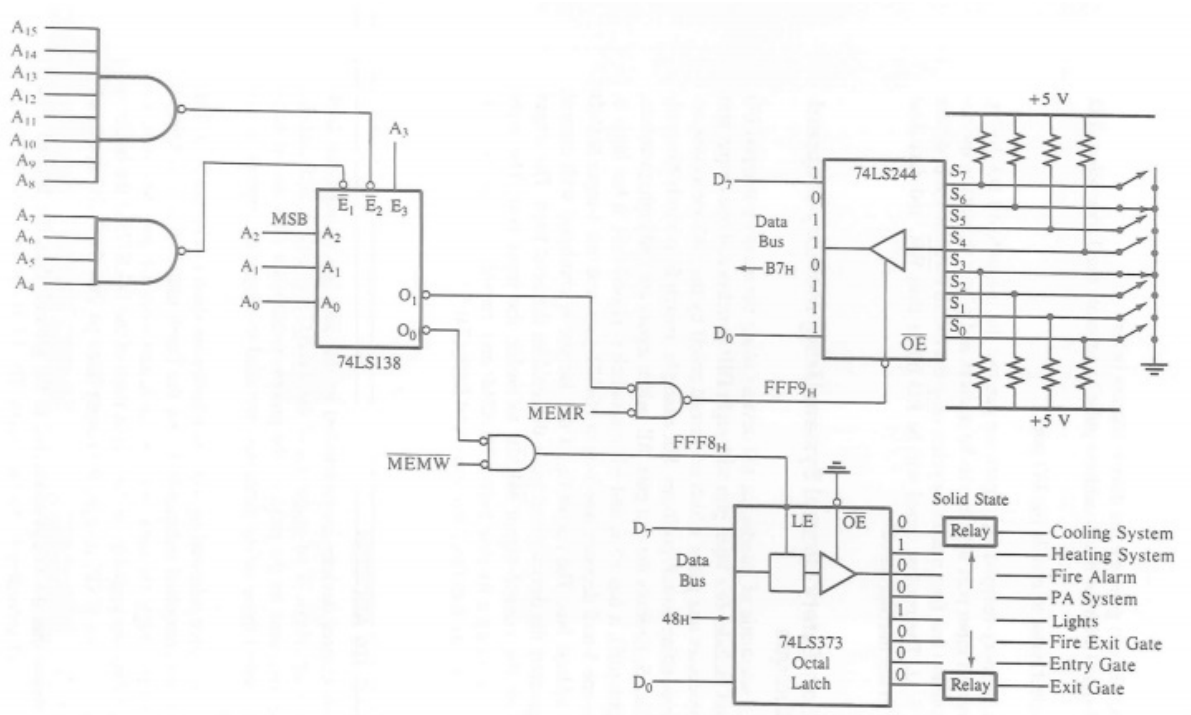



Figure 56: Memory mapped IO

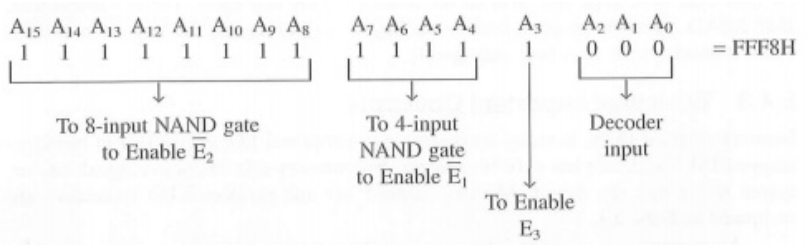



Figure 57: Memory mapped IO

# INTERFACING PERIPHERALS (IO) AND APPLICATIONS

Strathmore UNIVERSITY

## Interrupts

- An interrupt is a process by which a peripheral informs the microprocessor that it is ready for communication and requires its attention
- The process is initiated by an external device and is asychronous
- The response to the interrupt is controlled by the microprocessor.
- Interrupts are categorized into two categories **Maskable** and **non-Maskable**
- The microprocessor can ignore or delay a maskable interrupt when it is attending to something critical

# INTERFACING PERIPHERALS (IO) AND APPLICATIONS

Strathmore UNIVERSITY

## Interrupts Steps

- Step 1:Interrupt is enable by writing instruction IE in the main program. The instruction DI disables the interrupt
- Step 2:The MPU will check the INTR when executing the main program
- Step 3: If INTR pin is high the there is an interrupt. The Microprocessor completes the current instruction and then disables IE. then send $\overline{INTA}$. During this time the MPU cannot accepts any other interrupt.
- Step 4: $\overline{INTA}$ is used to insert RST in the main program. The RST transfers the program execution to page 00H after executing Step 5.

# INTERFACING PERIPHERALS (IO) AND APPLICATIONS

Strathmore UNIVERSITY

## Interrupts Steps

- Step 5: RST which is a CALL instruction makes the processor to save the memory address of the next instruction into stack. Execution is the transferred to the CALL location.
- Step 6: The task to be performed is then executed (subroutine).
- The subroutine contains IE to enable interrupt again.
- At the end of the RET is executed to return the execution to the main program.

# INTERFACING PERIPHERALS (IO) AND APPLICATIONS

## RST Instructions

- 8085 has a set of RST instructions which are executed in the same manner as CALL instructions.
- The interrupts are listed in the table below.

Restart Instructions

| Mnemonics | Binary Code $D_7$ | $D_6$ | $D_5$ | $D_4$ | $D_3$ | $D_2$ | $D_1$ | $D_0$ | Hex Code | Call Location in Hex |
|---|---|---|---|---|---|---|---|---|---|---|
| RST 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | C7 | 0000 |
| RST 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | CF | 0008 |
| RST 2 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | D7 | 0010 |
| RST 3 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | DF | 0018 |
| RST 4 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | E7 | 0020 |
| RST 5 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | EF | 0028 |
| RST 6 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | F7 | 0030 |
| RST 7 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | FF | 0038 |

Figure 58: RST table

# Description of the interrupt process

1. The main program initializes the stack pointer and enables interrupts
2. To interrupt the processor, push the INTR switch
3. Assume the switch is pushed when the processor is executing an instruction. The following sequence of event will occur
   a. Microprocessor completes executing the latest instruction
   b. It senses the INTR is high and IE is enabled
   c. It disables IE and sends out $\overline{INTA}$
   d. The $\overline{INTA}$ enables tristate buffer and the RST instruction is placed on the data bus
   e. The MPU saves the PC content, PSW contents and then jumps to the service routine.
   f. After execution of the service routine the MPU meets RET command the return to the main program after restoring the contents to PC

# 8085 Vectored Interrupts

- 8085 has five vectored interrupts
- These interrupts have locations as shown below
- The TRAP has highest priority but lower priority than HOLD and $\overline{INTA}$

| Interrupts | | Call Locations |
|---|---|---|
| 1. TRAP | ⟶ | 0024H |
| 2. RST 7.5 | ⟶ | 003CH |
| 3. RST 6.5 | ⟶ | 0034H |
| 4. RST 5.5 | ⟶ | 002CH |

Figure 59: Vectored Interrupt

- 8085 has five vectored interrupts
- These interrupts have locations as shown below
- The TRAP has highest priority but lower priority than HOLD and $\overline{INTA}$

| Interrupts | | Call Locations |
|---|---|---|
| 1. TRAP | → | 0024H |
| 2. RST 7.5 | → | 003CH |
| 3. RST 6.5 | → | 0034H |
| 4. RST 5.5 | → | 002CH |

Figure 60: Vectored Interrupt

# 8085 interrupts and vector locations

- TRAP ia a non-maskable interrupt and has the highest priority. It doesnt need to be enabled and cannot be disabled.
- RST 7.5,6.5 and 5.5: These maskable interrupts are enabled under program control using IE and SIM (set interrupt mask) and disabled using DI

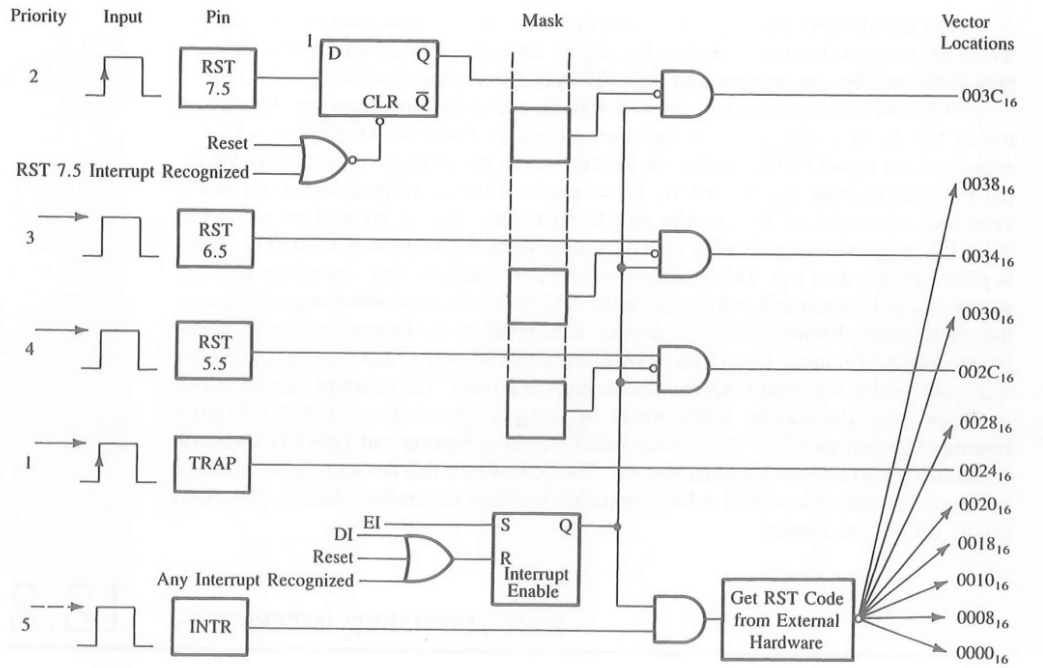



Figure 61: 8085 interrupts and vector locations

# SIM Register

- This a one byte instruction and can be used for 3 three different functions.
  - To set the mask for RST 7.5,6.5,5.5 on bit D0,D1,D2.
  - TO reset RST 7.5 on bit D4. This is additional function on RST 7.5. It is used to override ignore RST 7.5 without servicing it.
  - The third function is to implement serial I/O on bit D7 and D6.

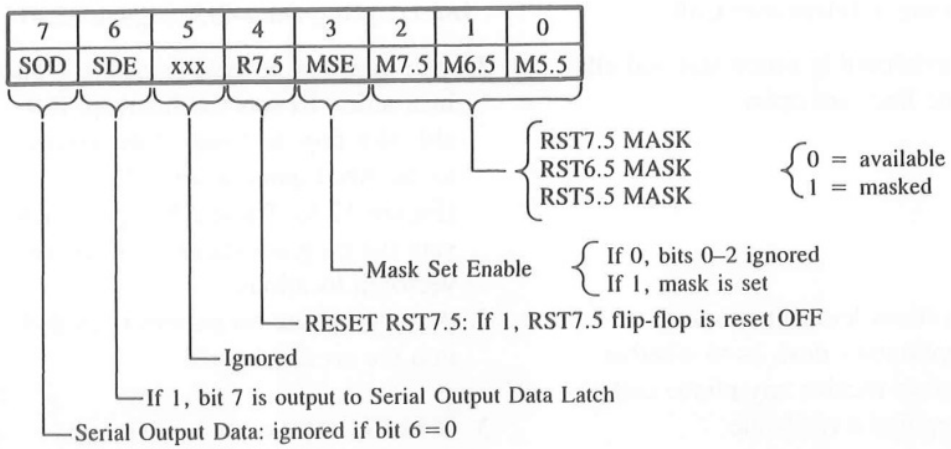



Figure 62: SIM Register

# Writing a control word on SIM

**Instructions**

```
EI          ;Enable interrupts
MVI A,08H   ;Load bit pattern to enable RST 7.5, 6.5, and 5.5
SIM         ;Enable RST 7.5, 6.5, and 5.5
```

Bit $D_3 = 1$ in the accumulator makes the instruction SIM functional, and bits $D_2$, $D_1$, and $D_0 = 0$ enable the interrupts 7.5, 6.5, and 5.5.

Reset the 7.5 interrupt

**Instructions**

```
MVI A,18H   ;Set D4 = 1
SIM         ;Reset 7.5 interrupt flip-flop
```

Figure 63

# Pending interrupts

Strathmore UNIVERSITY

## PENDING INTERRUPTS

Because there are several interrupt lines, when one interrupt request is being served, other interrupt requests may occur and remain pending. The 8085 has an additional instruction called RIM (Read Interrupt Mask) to sense these pending interrupts.

**Instruction** RIM: Read Interrupt Mask. This is a 1-byte instruction that can be used for the following functions.

- ☐ To read interrupt masks. This instruction loads the accumulator with 8 bits indicating the current status of the interrupt masks
- ☐ To identify pending interrupts. Bits $D_4$, $D_5$, and $D_6$ identify the pending interrupts.
- ☐ To receive serial data. Bit $D_7$ (Figure 12.7) is used to receive serial data.

Figure 64

# Pending interrupts cont..

Assuming the microprocessor is completing an RST 7.5 interrupt request, check to see RST 6.5 is pending. If it is pending, enable RST 6.5 without affecting any other interrupts; otherwise, return to the main program.

**Instructions**

```
        RIM         ;Read interrupt mask
        MOV B,A     ;Save mask information
        ANI 20H     ;Check whether RST 6.5 is pending
        JNZ NEXT
        EI
        RET         ;RST 6.5 is not pending, return to main program
NEXT:   MOV A,B     ;Get bit pattern; RST 6.5 is pending
        ANI 0DH     ;Enables RST 6.5 by setting D1 = 0
        ORI 08H     ;Enable SIM by setting D3 = 1
        SIM
        JMP SERV    ;Jump to service routine for RST 6.5
```

Figure 65

# Pending interrupts cont..

The instruction RIM checks for a pending interrupt. Instruction ANI 20H masks all the bits except $D_5$ to check pending RST 6.5. If $D_5 = 0$, the program control is transferred to the main program. $D_5 = 1$ indicates that RST 6.5 is pending. Instruction ANI 0DH sets $D_1 = 0$ (RST 6.5 bit for SIM), instruction ORI sets $D_3 = 1$ (this is necessary for SIM to be effective), and instruction SIM enables RST 6.5 without affecting any other interrupts. The JMP instruction transfers the program to the service routine (SERV) written for RST 6.5.

The RIM instruction loads the accumulator with the following information:

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| SID | I7.5 | I6.5 | I5.5 | IE | M7.5 | M6.5 | M5.5 |

Interrupt Masks: 1 = masked

Interrupt Enable Flag: 1 = enabled

Pending Interrupts: 1 = pending

Serial Input Data Bit, if any

Figure 66

# Stack and subroutine

- Stack is a group of memory location in the RAM which is used to store temporary data.
- These locations are defined in the main memory.
- The beginning of the stack is define by using instruction LXI SP XXXXH
- Data bytes are stored in pairs in the stack using instruction PUSH and retrieved using POP. eg PUSH B to store the contents of register B and C and POP B to retrieve the same content.

# Initializing stack

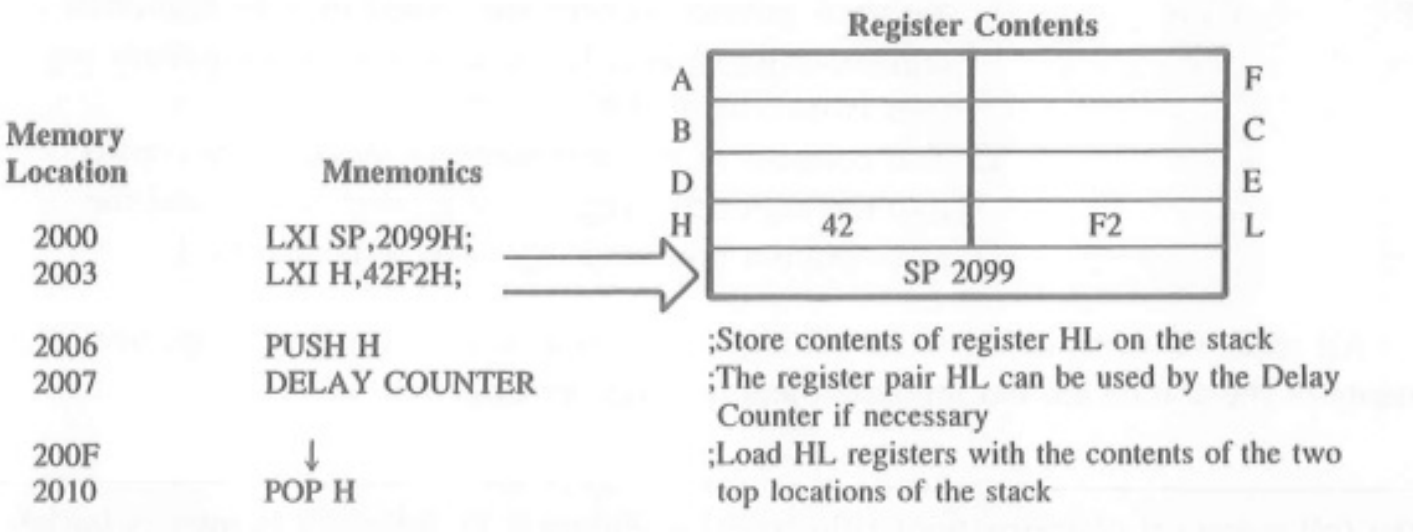



Figure 67

# Stack storage and retrieval

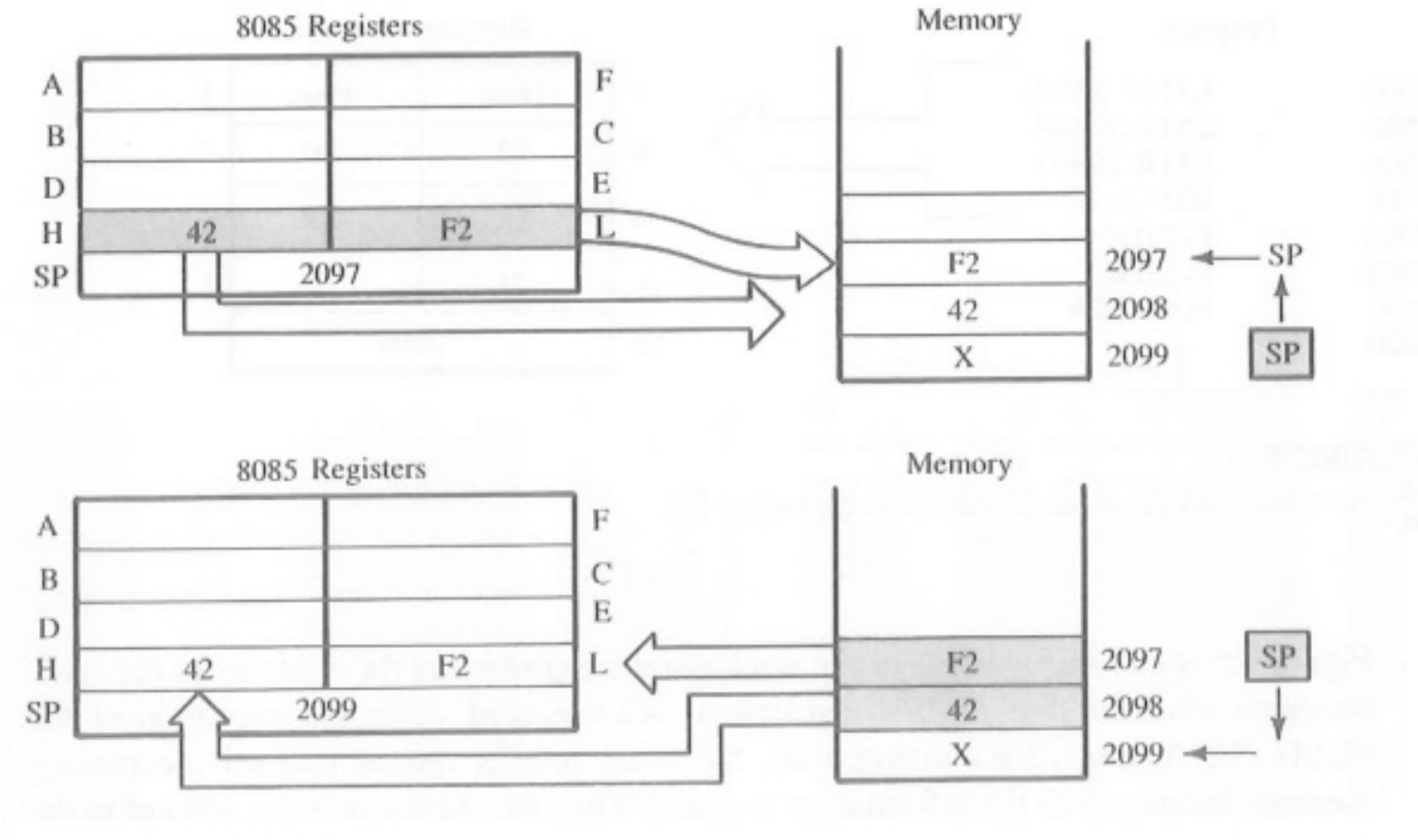



Figure 68

# The general purpose programmable peripheral device (8255A)

- The 8255A is a widely used programmable parallel I/O device.
- It can be programmed to transfer data under various conditions, from simple I/O to interrupt I/O.
- It is flexible, versatile and economical (when i/o ports are required), but somewhat comples.
- It is an important general purpose I/O device that can be used with almost every microprocessor
- The 8255A has 24 I/O pins that can be grouped primarily in two parallel ports: A and B with the remaining eight bits as port C.
- The eight bits of port C can be used as individual bits or be grouped in two 4bit ports $C_{UPPER}(C_U)$ and $C_{LOWER}(C_L)$

# The general purpose programmable peripheral device (8255A)

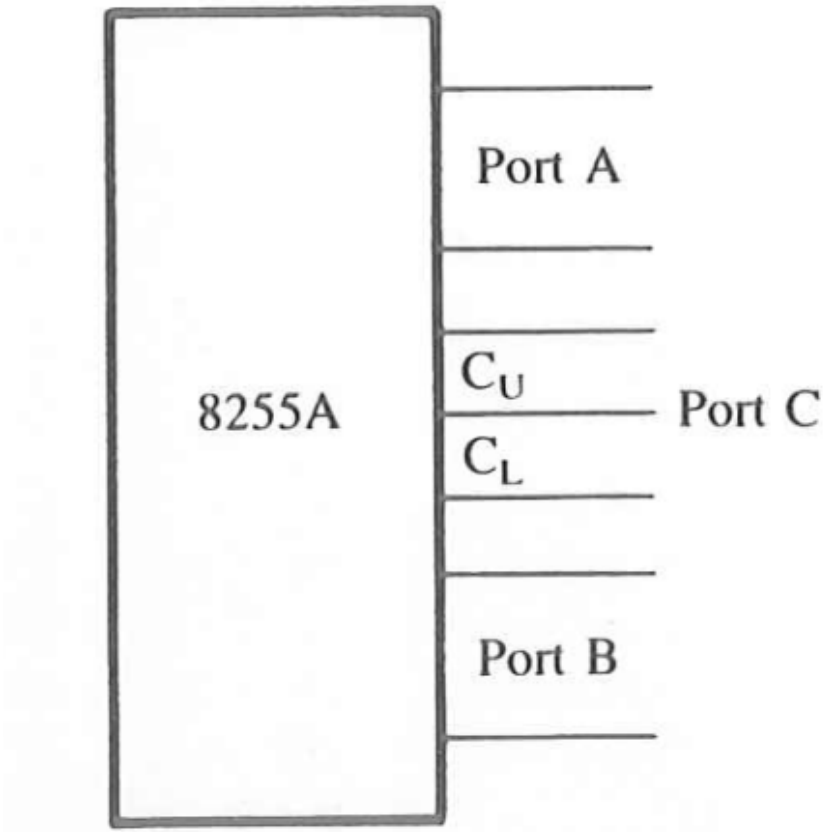



Figure 69: Pin functional diagram

# Functionality of (8255A)

- The functions of these ports are defined by writing a control word in the control register
- The functions are classified into two modes
  - The BIT Set/Reset mode and The I/O mode
  - The BSR mode is used to set or reset port C
  - The I/O mode is further divided into three modes: Mode 0, Mode 1 and Mode 2.
  - In Mode O all ports functions as simple I/O ports, Mode 1 is handshake mode where ports A and/or B use bits from port C for handshake signals, two types of data transfer are implemented status check and interrupt.
  - In mode 2, port A can be set up for bidirectional data transfer using handshake signal from port C and port B can be set up either in Mode 0 or Mode 1

# The general purpose programmable peripheral device (8255A) modes

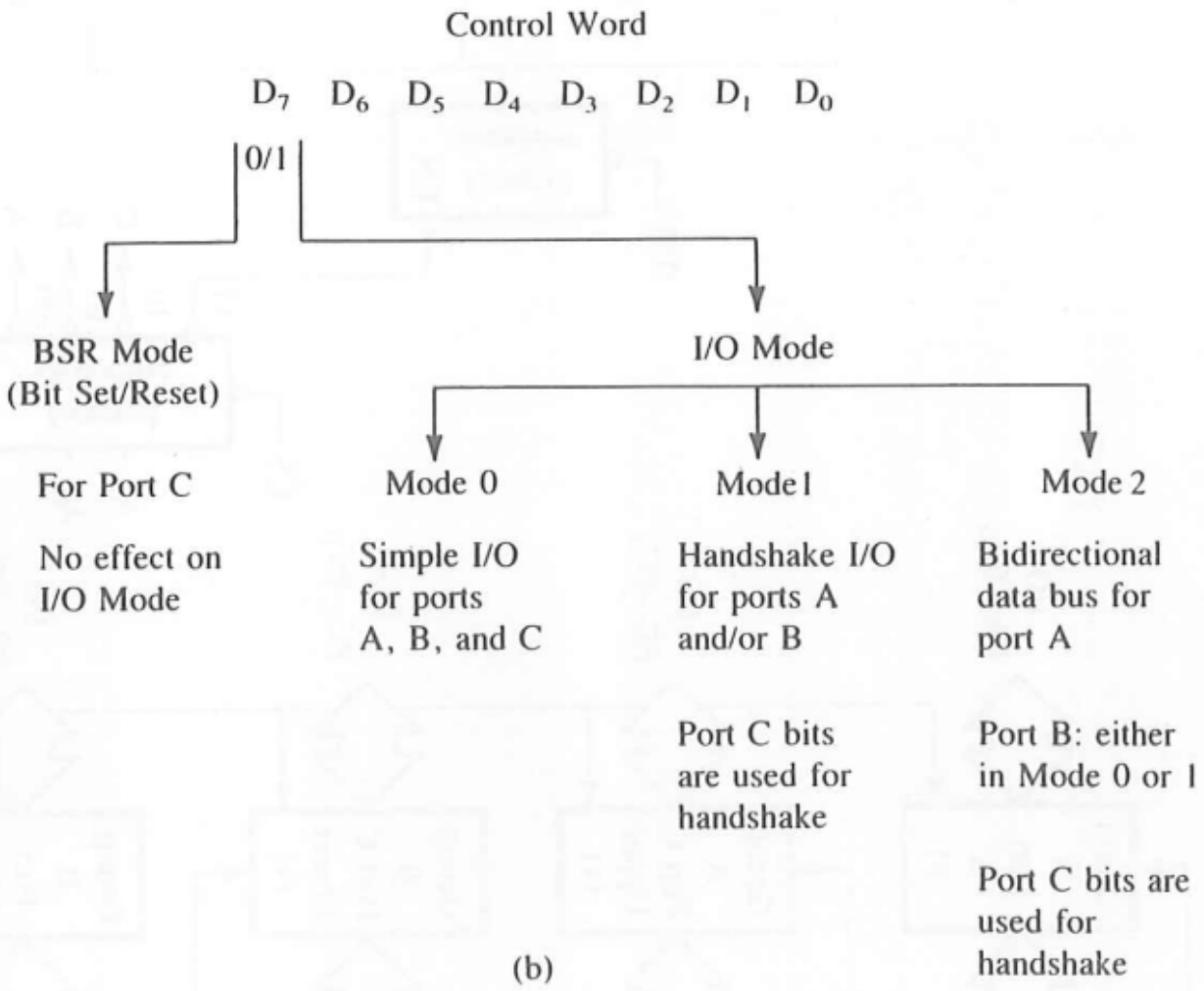



Figure 7.9: modes

# (8255A) Control logic

- The control logic has six lines. There functions and connections are as follows.
  - $\overline{RD}$(read): This control signal enable the Read operation. When the signal is lwo the MPU reads data from a selected port of thr 8255A
  - $\overline{WR}$(write) This control signal enables the Write operation. When the signal goes low., the MPU writes into a selected I/O port or the control register
  - RESET (Reset) This is an active high signal; it clears the control register and sets ports input mode.
  - $\overline{CS}, A_0A_1$ These are device select signals. $\overline{CS}$ signal is connected to a decoded address, and $A_0$ and $A_1$ are generally connected to MPU address $A_0$ and $A_1$ respectively

# Control Logic

The $\overline{CS}$ signal is the master Chip Select, and $A_0$ and $A_1$ specify one of the I/O ports or the control register as given below:

| $\overline{CS}$ | $A_1$ | $A_0$ | Selected |
|---|---|---|---|
| 0 | 0 | 0 | Port A |
| 0 | 0 | 1 | Port B |
| 0 | 1 | 0 | Port C |
| 0 | 1 | 1 | Control Register |
| 1 | X | X | 8255A is not selected. |

Figure 71: control logic

# Detailed Diagram

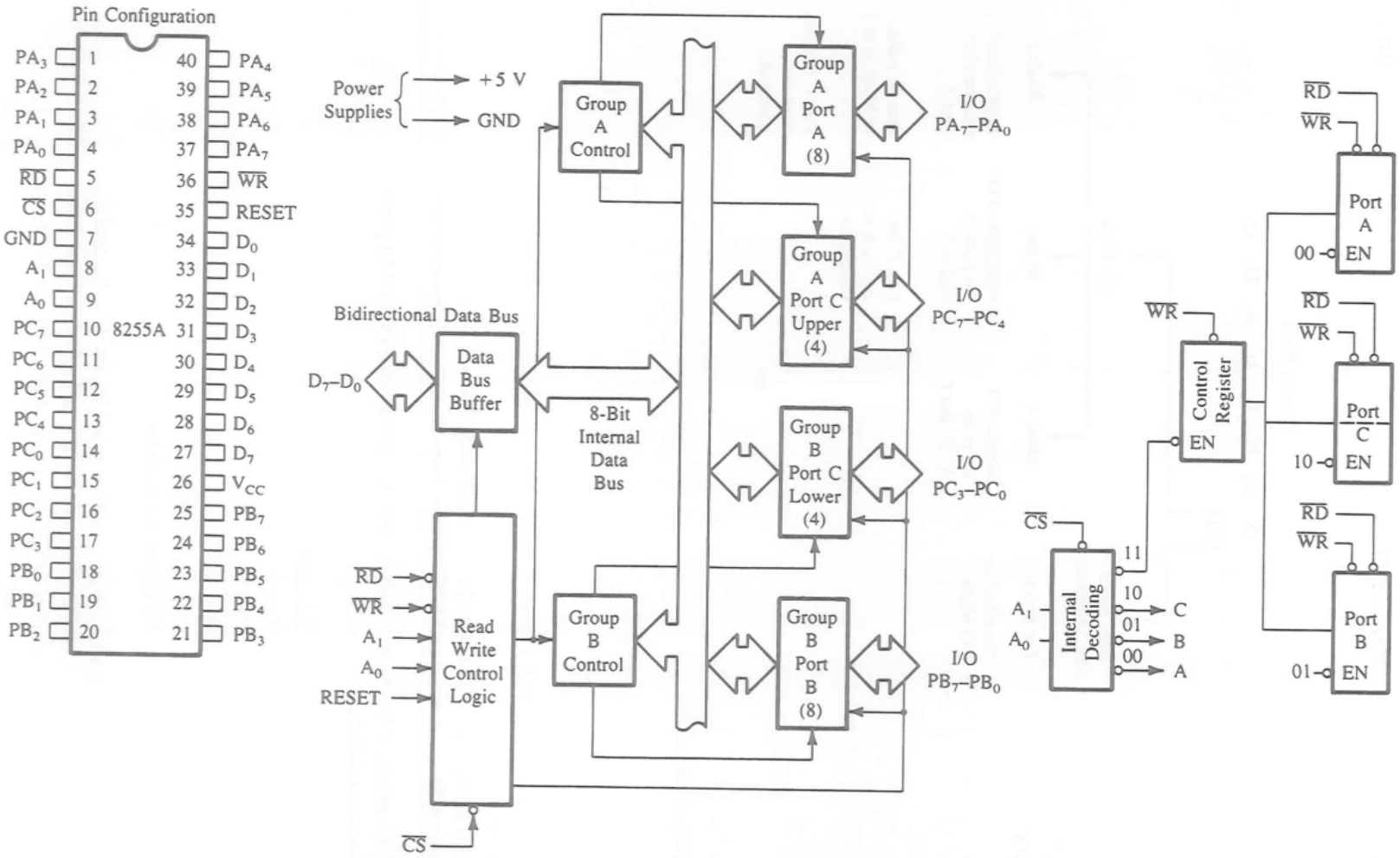
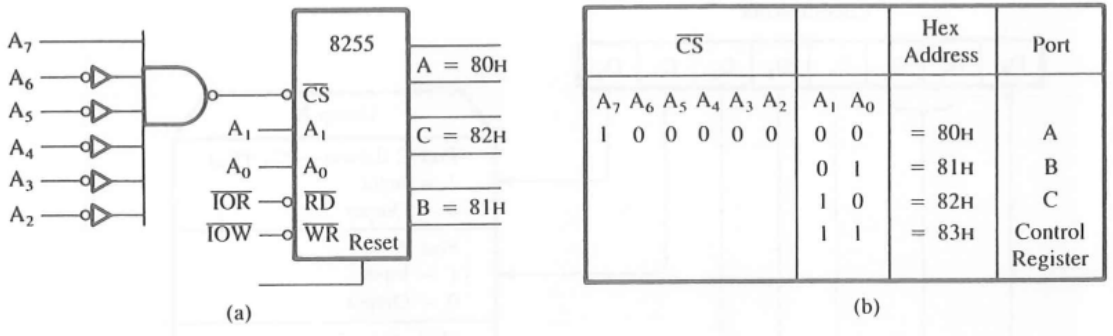



Figure 72: 8255 detailed diagram

# (8255A) Control Word

- The contents of this register called th control word, specify an I/O function for each port. This register can be accessed to write the control word when A0 and A1 are at logic 1. This register is not accessible for read operation.




| $\overline{CS}$ | | | | | | | | Hex Address | Port |
|---|---|---|---|---|---|---|---|---|---|
| $A_7$ | $A_6$ | $A_5$ | $A_4$ | $A_3$ | $A_2$ | $A_1$ | $A_0$ | | |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | = 80H | A |
| | | | | | | 0 | 1 | = 81H | B |
| | | | | | | 1 | 0 | = 82H | C |
| | | | | | | 1 | 1 | = 83H | Control Register |

(b)

Figure 73: 8255 ports and there addresses

# Control register

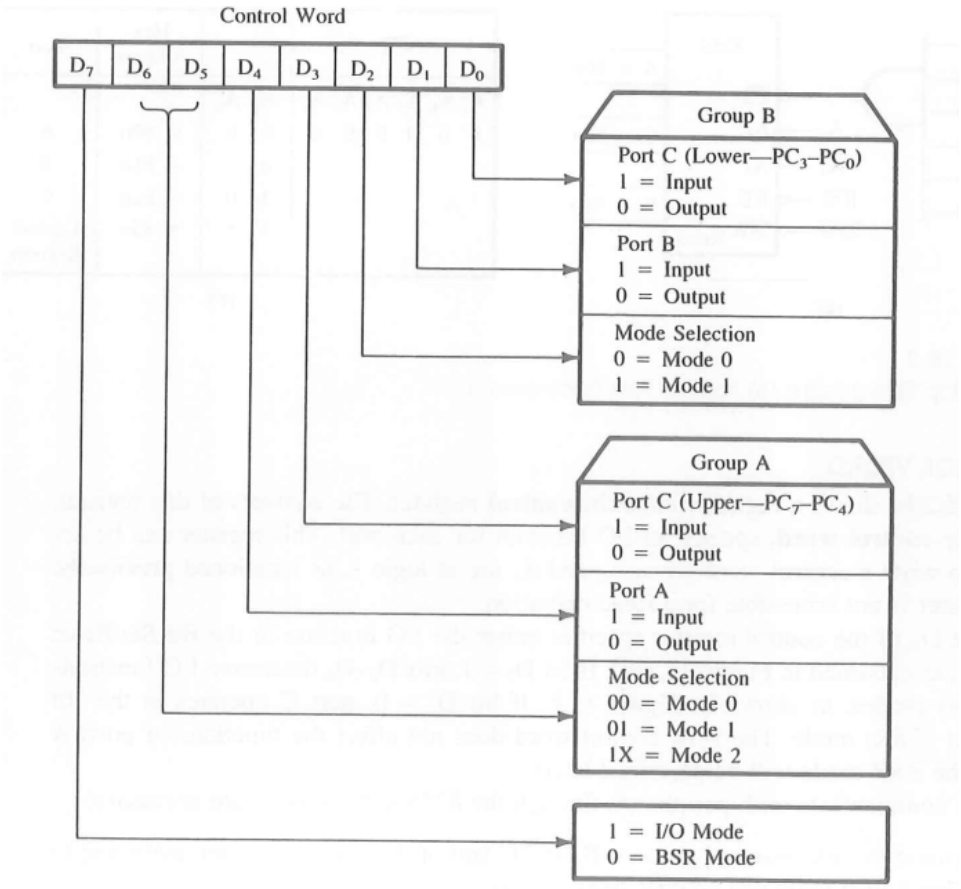



Figure 74: 8255 control register

# (8255A) Control Word explanation

- Bit D7 of the control register specifies either the I/O or BSR functions. If D7= 1 bits D6 -D0 determine I/O functions in various modes.
- If Bit D7=0 port C operates in the BSR mode.
- BSR does not affect port A and B
- To communicate with peripherals through the 8255A three steps are necessary:
  - Determine the addresses of ports A, B and C and the control register according to the Chip select logic and address lines A0 and A1
  - Write a control word in the control register
  - Write I/O instructions to communicate withe peripherals through ports A,B,C

# COUNTERS AND TIME DELAYS

Strathmore UNIVERSITY

- Counters are used primarily to keep track of events;
- Time delays are important in setting up reasonably accurate timing between two events.
- The process of designing counters and time delays using software instructions is far more flexible and less time consuming that hardware.
- **COUNTER:** A counter is designed by simply loading an appropriate number into one of the registers and using the INR or the DCR instructions
- A loop is established to update the count and each count is checked to determine whether it has reached the final number or not

# COUNTERS AND TIME DELAYS

- The flow chart below illustrates counter steps

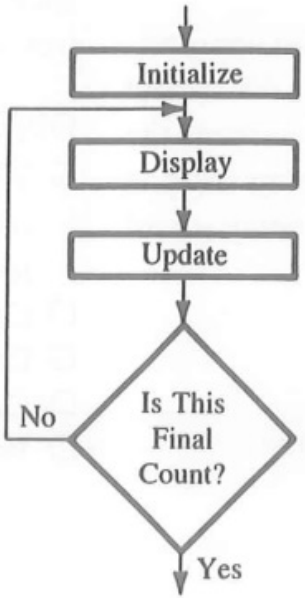
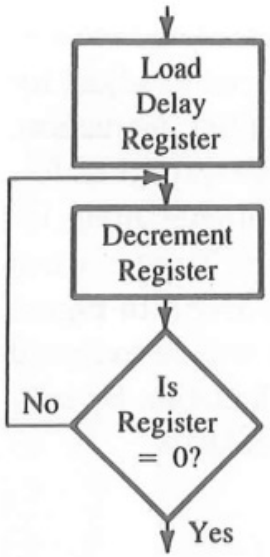



Figure 75: Counter flow chart

# COUNTERS AND TIME DELAYS

- **TIME DELAY**: The procedure used to design a time delay is similar to that used to set up a counter.
- A register is loaded with a number, depending on the time delay required, and then the register is decremented until it reaches zero




| Label | Opcode | Operand | Comments | T-states |
|---|---|---|---|---|
| | MVI | C,FFH | ;Load register C | 7 |
| LOOP: | DCR | C | ;Decrement C | 4 |
| | JNZ | LOOP | ;Jump back to ; decrement C | 10/7 |

Figure 76: time delay flow chart

- It is important to determine each instruction and the T-states

# Calculating time delay from instructions

- The instruction MVI requires 7 T-states
- An 8085 based microcomputer may use 2 MHz clock frequency, then the time it will take to execute the instruction MVI is calculated as follows.

Clock frequency of the system f = 2 MHz
Clock period $T = 1/f = 1/2 \times 10^{-6} = 0.5\ \mu s$
Time to execute MVI = 7 T-states × 0.5
$= 3.5\ \mu s$

Figure 77: time delay calculation

- To calculate the time delay in a loop we must account for the T-states required for each instruction and for the number of times the instructions are executed in the loop

# Calculating time delay from instructions

- The register C is loaded with the count FFH ($255_{10}$) by the instruction MVI which is executed once and takes 7 T-states.
- The next two instructions, DCR and JNZ form a loop with a total of 14 i.e (4+10) T-states.
- The loop is repeated 255 times until register C = 0

The time delay in the loop $T_L$ with 2 MHz clock frequency is calculated as

$$TL = (T \times \text{Loop T-states} \times N10)$$

where $T_L$ = Time delay in the loop
$T$ = System clock period
$N_{10}$ = Equivalent decimal number of the hexadecimal count loaded in the delay register

$$T_L = (0.5 \times 10^{-6} \times 14 \times 255)$$
$$= 1785\ \mu s$$
$$\approx 1.8\ ms$$

Figure 78: time delay calculation

# Calculating time delay from instructions

- To calculate the time delay more accurately, we need to adjust for the execution of the JNZ instruction and add the execution time of the initial instruction.
- The T-states for JNZ instruction are shown as 10/7.
- This can be interpreted as follows: The processor requires 10 T-states to execute a conditional Jump instruction and 7-T-states when the program falls through the loop.
- The loop will be executed 255 times; and in the last cycle the JNZ will be executed will be executed in 7-T-states.
- The difference can be accounted for in the delay calculation by subtracting the execution time of 3 T-states. Therefore the adjusted loop delay is

$$\begin{aligned} T_{LA} &= T_L - (3 \text{ T-states} \times \text{Clock period}) \\ &= 1785.0\ \mu s - 1.5\ \mu s = 1783.5\ \mu s \end{aligned}$$

Figure 79: more accurate time delay calculation

# Calculating time delay from instructions

- Now the total dealy must take into accoun the execution time of the instructions outside the loop.
- In the previous example we have only one instruction (MVI C) outside the loop.
- Therefore the total time delay is

$$\text{Total Delay} = \begin{matrix}\text{Time to execute instructions} \\ \text{outside loop}\end{matrix} + \begin{matrix}\text{Time to execute} \\ \text{loop instructions}\end{matrix}$$

$$\begin{aligned} T_D &= T_O + T_{LA} \\ &= (7 \times 0.5\ \mu s) + 1783.5\ \mu s = 1787\ \mu s \\ &\approx 1.8\ ms \end{aligned}$$

Figure 80: more accurate time delay calculation

# Time delay using a register pair

- The time delay can be considerably increased by setting a loop and using a register pair with a 16-bit (maximum FFFFH).
- Th 16 bit number is decremented by using the instruction DCX.
- However DCX doesnt set the zero flag and without testing the z-flag Jump instruction cannot be executed.
- Additional technique must be used to set the Zero flag. Consider the following instructions

| Label | Opcode | Operand | Comments | T-states |
|---|---|---|---|---|
| | LXI | B,2384H | ;Load BC with 16-bit count | 10 |
| LOOP: | DCX | B | ;Decrement (BC) by one | 6 |
| | MOV | A,C | ;Place contents of C in A | 4 |
| | ORA | B | ;OR (B) with (C) to set Zero flag | 4 |
| | JNZ | LOOP | ;If result ≠ 0, jump back to LOOP | 10/7 |

Figure 81: Using a double register

- The instruction MOV A,C and OR B are used to set the Zero flag if both B and C are zero.
- The loop will be repeated 2384H times, equal to the count set in the register pair.

# Time delay using a register pair calculating the time delay

The time delay in the loop is calculated as in the previous example. The loop includes four instructions: DCX, MOV, ORA, and JNZ, and takes 24 clock periods for execution. The loop is repeated 2384H times, which is converted to decimals as

$$\begin{aligned} 2384\text{H} &= 2 \times (16)^3 + 3 \times (16)^2 + 8 \times (16)^1 + 4(16^0) \\ &= 9092_{10} \end{aligned}$$

If the clock period of the system = 0.5 µs, the delay in the loop $T_L$ is

$$\begin{aligned} T_L &= (0.5 \times 24 \times 9092_{10}) \\ &\approx 109 \text{ ms (without adjusting for the last cycle)} \\ \text{Total Delay } T_D &= 109 \text{ ms} + T_O \\ &\approx 109 \text{ ms (The instruction LXI adds only 5 µs.)} \end{aligned}$$

Figure 82: Using a double register

# Time delay using a loop within a loop techniqu

A time delay similar to that of a register pair can also be achieved by using two loops; one loop inside the other loop, as shown in Figure 8.3(a). For example, register C is used in the inner loop (LOOP1) and register B is used for the outer loop (LOOP2).

```
        MVI B,38H     7T
LOOP2:  MVI C,FFH     7T
LOOP1:  DCR C         4T
        JNZ LOOP1     10/7T
        DCR B         4T
        JNZ LOOP2     10/7T
```

DELAY CALCULATIONS

The delay in LOOP1 is $T_{L1} = 1783.5\ \mu s$. These calculations are shown
We can replace LOOP1 by $T_{L1}$, as shown Now we can calculate the delay in LOOP2 as if it is one loop; this loop is executed 56 times because of the count (38H) in register B:

$$\begin{aligned} T_{L2} &= 56(T_{L1} + 21\ \text{T-states} \times 0.5\ \mu s) \\ &= 56(1783.5\ \mu s + 10.5\ \mu s) \\ &= 100.46\ ms \end{aligned}$$

Figure 83: Using a loop withing a loop