8086 Microprocessor System Project (Part I-B)

Project Summary

This project presents the complete design, integration, and initialization of a microcomputer system based on the Intel **8086 microprocessor running at 10 MHz**, with support for all major peripherals and coprocessing capabilities. The system includes:

- Math Coprocessor **8087**
- **1MB total memory** (RAM and ROM)
- **Diskette controller 8272** (added in Part I-B)
- **Parallel**, **Serial**, **USB** (DMA-based) communication
- **ADC/DAC** for analog/digital conversion
- **16-digit seven-segment display**
- **64-key matrix keyboard**
- **Printer interface**
- **8237 DMA Controller**
- **8259A Interrupt Controller**

The full system includes memory decoding logic, I/O mapping, assembly-level routines, pseudocode for all initialization steps, and team member task distribution.

Design Objectives

- 🧠 Construct a working microcomputer based on **8086 + 8087**
- 🗺️ Design and implement a **1MB memory map** with DRAM, ROM, I/O
- 🔌 Integrate peripherals via **memory-mapped I/O**
- 🧮 Use **8237 DMA** for high-speed USB/disk transfers
- ⚡ Use **8259A PIC** for interrupt-driven I/O handling
- 💾 Add **8272 diskette controller** (Part I-B requirement)
- 🔧 Provide full initialization and test routines in **pseudocode + assembly**

System Architecture

- Full block diagram of CPU, memory, coprocessor, and peripheral buses
- Unified memory and I/O space using **address decoding logic** with **74LS138**
- All data/control signals routed and labeled

  📄 Refer to: `CPU_Memory.md`, `CPU_Memory.asm`, `CPU_Memory_pseudocode.md`

Memory System Design

- 1MB memory space divided as:
  - **768KB RAM** using 4164 DRAM chips (3 banks)
  - **128KB ROM** for BIOS and routines
  - **128KB I/O-mapped area** for peripherals
- I/O Address Summary: | Address Range | Assigned To | |————————|————————————-| | `0xC0000` | Interrupt controller (PIC) | | `0xC0100` | DMA | | `0xC0200` | Serial port (8251) | | `0xC0300` | Parallel port (8255) | | `0xC0400` | Keyboard (8279) | | `0xC0500` | Display (8279) | | `0xC0600` | ADC/DAC | | `0xC0700` | USB Interface (DMA) | | `0xC0800` | Printer (8255) | | `0xC0900` | Diskette Controller (8272) |

Refer to: `CPU_Memory.md`, `Storage_DMA_Integration.md`

Software & Initialization

- All system modules are initialized via assembly and/or pseudocode:
  - System startup and memory test
  - 8087 coprocessor detection and setup
  - DMA configuration and buffer setup
  - Peripheral setup (keyboard, display, printer, ADC/DAC, disk)
  - USB + Floppy transfer routines using **DMA Channels 1–3**

Refer to: - `CPU_Memory.asm`, `CPU_Memory_pseudocode.md` - `IO_Peripherals.md` - `Data_Conversion_Interrupts.md` - `Storage_DMA_Integration.md` —

Team Responsibilities

Member	Area	Key Components	Deliverables	Files Submitted
Jared	CPU & Memory Architecture	• 8086 CPU• 8087 Coprocessor• 1MB RAM/ROM• Address decoding	• Block diagrams• Memory map• Initialization code• Assembly routines	**CPU_Memory.md**CPU_Memory.asmCPU_Memory_pseudocode.mdCPU_Memory.txt
Jesmarie	User I/O Interface	• 16-digit 7-segment display• 64-key matrix keyboard• Printer	• Display driver• Keyboard scanner• Printer interface• Assembly examples	**IO_Peripherals.md**
Valeria	Communication	• RS-232 serial	•	**Data_Conversi**

	s & Interrupts	port• Parallel port• USB+DMA• 8259A interrupt controller	Communication drivers• DMA controller• Interrupt handlers• USB routines	**on_Interrupts.md**
Giovanny	Data Conversion & Storage	• ADC (Analog-to-Digital)• DAC (Digital-to-Analog)• 8272 Floppy controller	• ADC/DAC drivers• Disk controller• Conversion routines• Storage examples	**Storage_DMA_Integration.md**

All files are modular and follow shared address conventions.

☑ Final Integration Test

- System performs correct memory initialization with RAM test and ROM checksum
- All I/O devices respond correctly to their mapped addresses
- DMA transfers are verified between floppy <-> memory <-> USB
- Interrupts are handled using vector table `0x0000–0x03FF` and managed by 8259A

📎 Getting Started

To replicate or run the project: 1. Load `CPU_Memory.asm` and simulate on 8086-compatible assembler 2. Refer to each `.md` file for subsystem details 3. Use pseudocode as reference for embedded system implementation 4. Test each module independently before full integration

---

📚 References (with PDF Filenames)

- [1] **8086 Memory and I/O Interfacing**
  - 1.8086 Memory and I_O Interfacing.pdf
  - 2.8086 Memory and I_O Interfacing. Part II.pdf
  - 3.8086 Memory and I_O Interfacing.Part III and Case of Studies.pdf
- [2] **System Bus Structure**
  - 4.CAP 8 - SYSTEM BUS STRUCTURE.pdf
- [3] **I/O Interfaces and Interrupts**
  - 5.CAP 9 - I.O INTERFACES.pdf

- [4] **Semiconductor Memory**
    - 6.CAP 10 Semiconductor Memory.pdf
- [5] **Multiprocessor Configurations & Datasheets**
    - 7.8086 Multiprocessor Configurations-1.pdf
    - Manufacturer datasheets: 8086, 8087, 8237, 8259A, 8255, 8251, 8279, 8272, ADC0808, DAC0800

👥 Task Division for 4 Members (with File Locations Updated)

Overview

Jared: CPU, Coprocessor, Memory & Map

- Block diagram for **8086**, **8087**, and overall architecture.
- **Memory map:** 1MB organization, memory types, decoding.
- **Address decoding logic** for memory.
- **Relevant PDFs:** 1, 4, 6, 7
- **Location:** Add your work to **`CPU_Memory.md`**, and include code in:
  - `CPU_Memory.asm` (assembly code)
  - `CPU_Memory_pseudocode.md` (pseudocode)
  - `CPU_Memory.txt` (optional: summaries/raw code)

🎯 WORK COMPLETED

📐 System Architecture Design

Complete 8086 system block diagram with all major components

8087 coprocessor integration for floating-point operations

System bus architecture (20-bit address, 16-bit data, control signals)

Hierarchical component organization showing processor, memory, controllers, and peripherals

💾 Memory System Implementation

1MB memory organization with clear address ranges:

768KB RAM (user programs and data)

128KB ROM (system BIOS)

128KB I/O mapped space (peripherals)

```
    </li>
    <li><b>Complete memory map</b> with specific address ranges for each
component</li>
    <li><b>Address decoding logic</b> using 74138 decoders for memory and I/O
selection</li>
    <li><b>Physical memory configuration</b> with DRAM and EPROM chip
specifications</li>
  </ul>
</td>
```

🔌 I/O System Coordination

Complete I/O address space allocation for all 10 peripheral devices

Team address coordination preventing conflicts between members

Peripheral interface definitions for all controllers (8259A, 8237, 8251, 8255, 8279, 8272, etc.)

Expansion area reservation for future system growth

📝 Software Implementation

Assembly language initialization routines with complete system startup

Pseudocode documentation for all major system functions

Memory management routines (read, write, copy, fill operations)

8087 coprocessor initialization with detection and testing

Team coordination functions with address mapping

Jesmarie: Parallel/Serial, Printer, Keyboard/Display

- **8255** (parallel), **8251** (serial), printer interface.
- **8279** (keyboard/display).
- Block diagrams for these peripherals and their connections.
- **Relevant PDFs:** 1, 2, 5
- **Location:** Add your work to **`IO_Peripherals.md`** (create if it doesn't exist).

Valeria: Data Conversion, USB (DMA), Interrupts

- **ADC**, **DAC** interfacing (block diagrams, connections).
- **USB interface** through DMA (**8237**).
- **8259** (interrupt controller) integration for I/O.
- **Relevant PDFs:** 1, 2, 3, 5
- **Location:** Add your work to **`Data_Conversion_Interrupts.md`** (create if it doesn't exist).

Giovanny: Diskette Controller (8272), DMA, Integration

- Diskette controller (floppy, **8272**), **8237 DMA** for high-speed data.
- Pseudocode for **USB** and diskette transfers using DMA.
- Help integrate all I/O into final system.
- **Relevant PDFs:** 3, 5, 6

- **Location:** Add your work to **`Storage_DMA_Integration.md`** (create if it doesn't exist).

📋 Task Division Summary for 4 Members

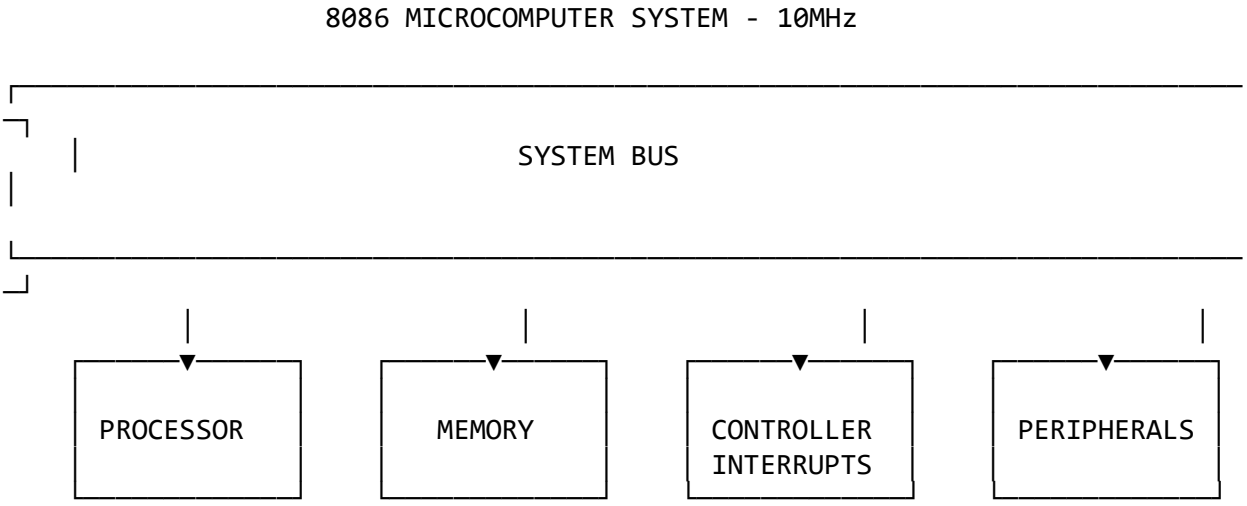
Member	Area	Key Components	Deliverables	Files Submitted
Jared	CPU & Memory Architecture	• 8086 CPU• 8087 Coprocessor• 1MB RAM/ROM• Address decoding	• Block diagrams• Memory map• Initialization code• Assembly routines	**CPU_Memory.md**CPU_Memory.asmCPU_Memory_pseudocode.mdCPU_Memory.txt
Jesmarie	User I/O Interface	• 16-digit 7-segment display• 64-key matrix keyboard• Printer	• Display driver• Keyboard scanner• Printer interface• Assembly examples	**IO_Peripherals.md**
Valeria	Communications & Interrupts	• RS-232 serial port• Parallel port• USB+DMA• 8259A interrupt controller	• Communication drivers• DMA controller• Interrupt handlers• USB routines	**Data_Conversion_Interrupts.md**
Giovanny	Data Conversion & Storage	• ADC (Analog-to-Digital)• DAC (Digital-to-Analog)• 8272 Floppy controller	• ADC/DAC drivers• Disk controller• Conversion routines• Storage examples	**Storage_DMA_Integration.md**

CPU & MEMORY ARCHITECTURE - JARED: COMPLETE SYSTEM DESIGN

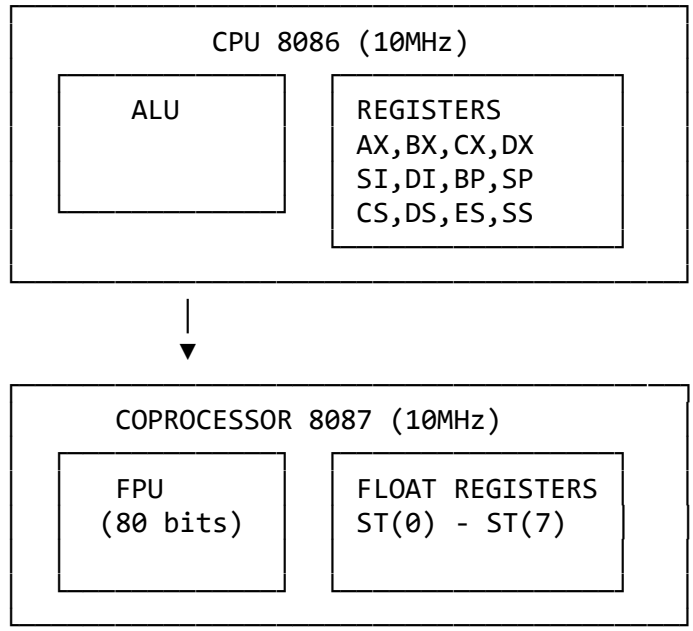
This document presents the complete design of an 8086-based microcomputer system with 1MB memory, coprocessor 8087, and comprehensive I/O interfaces including diskette controller 8272 for teams of 4 members.

1. GENERAL BLOCK DIAGRAM OF THE SYSTEM

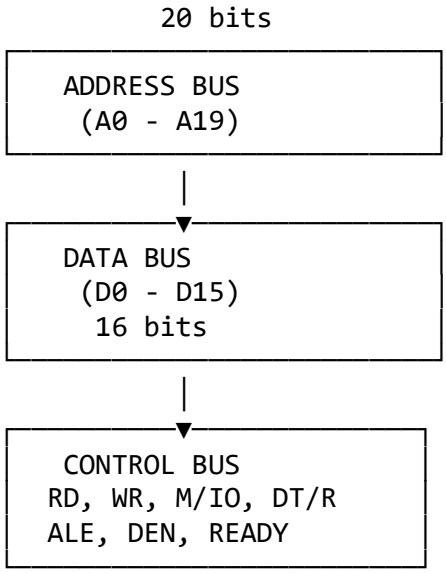
The system architecture follows a hierarchical design with the 8086 CPU as the central processing unit, supported by the 8087 coprocessor for floating-point operations. All components communicate through a unified system bus structure.



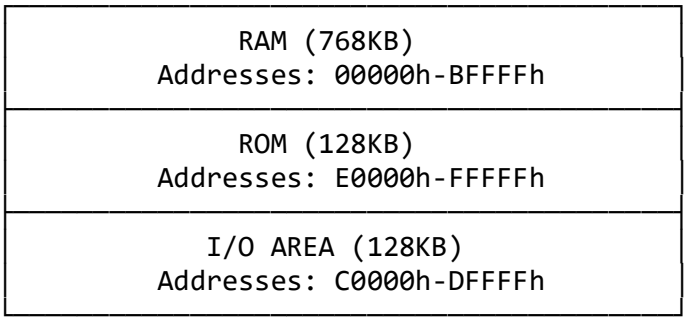
PROCESSOR



BUS SYSTEM



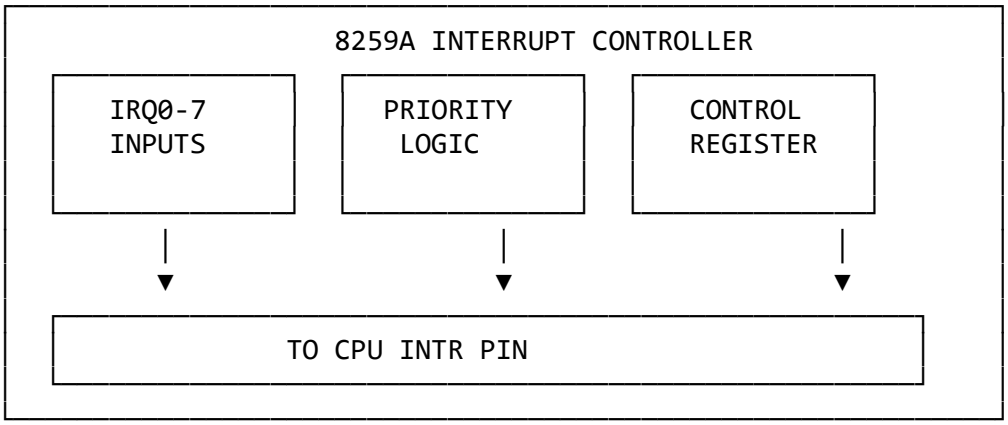
MAIN MEMORY (1 MBYTE)



System Bus Explanation: The 8086 provides a 20-bit address bus allowing access to 1MB of memory space. The 16-bit data bus enables efficient data transfer, while control signals coordinate memory and I/O operations. The coprocessor 8087 operates in parallel with the CPU for mathematical computations.

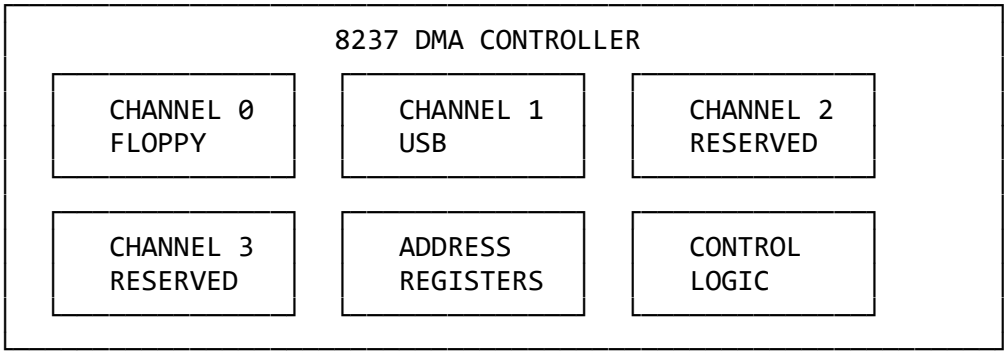
2. DETAILED I/O INTERFACE DIAGRAMS

2.1 INTERRUPT CONTROLLER (8259A) - Address: C0000h-C00FFh



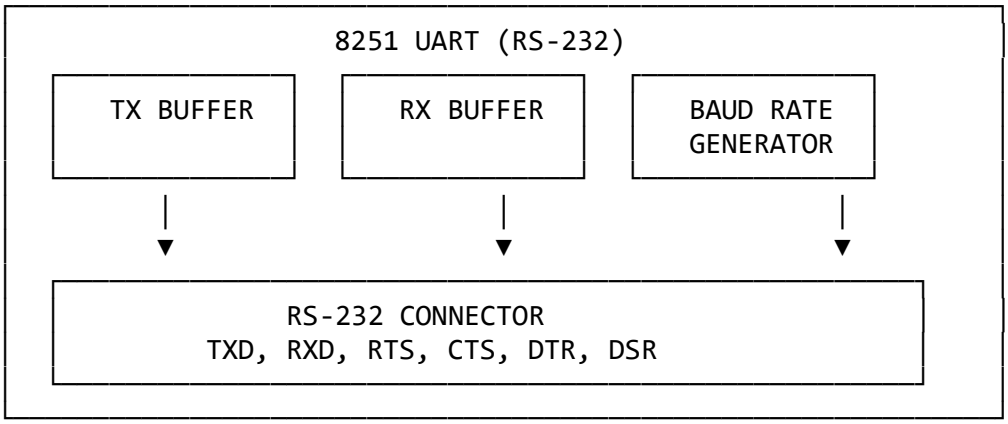
Connected IRQ Lines:
- IRQ0: Timer (system clock)
- IRQ1: Keyboard
- IRQ2: Serial port
- IRQ3: Parallel port
- IRQ4: USB controller
- IRQ5: ADC/DAC
- IRQ6: Floppy controller
- IRQ7: Printer

2.2 DMA CONTROLLER (8237) - Address: C0100h-C01FFh



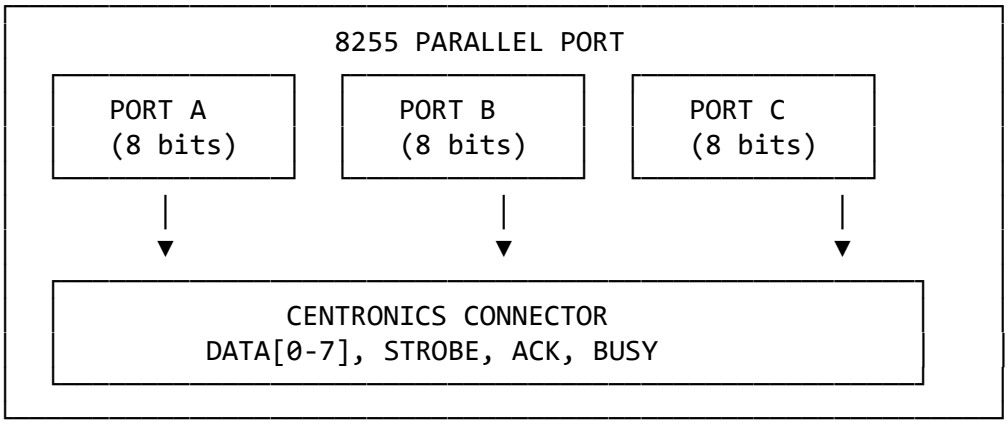
DMA Channels:
- CH0: Floppy disk high-speed transfers
- CH1: USB data transfers
- CH2: Available for expansion
- CH3: Available for expansion

2.3 SERIAL PORT (8251 UART) - Address: C0200h-C02FFh



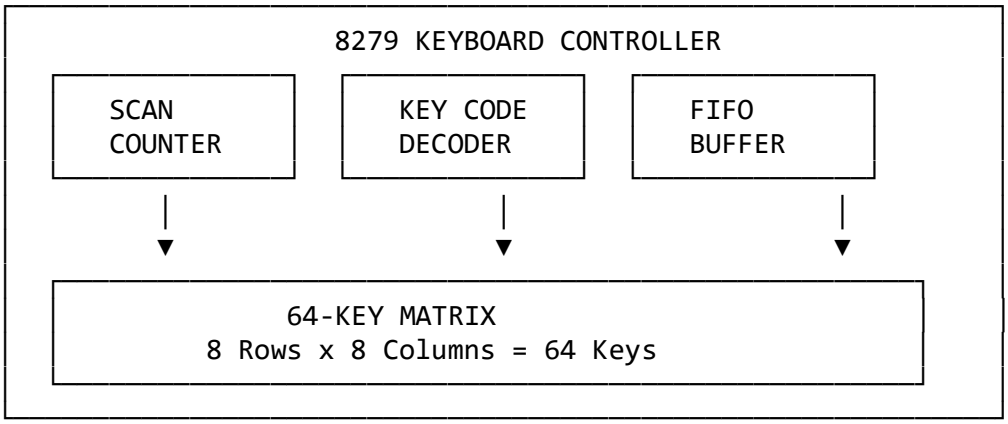
Features:
- Programmable baud rates (110-9600 bps)
- Full duplex communication
- Hardware handshaking support

2.4 PARALLEL PORT (8255 PPI) - Address: C0300h-C03FFh



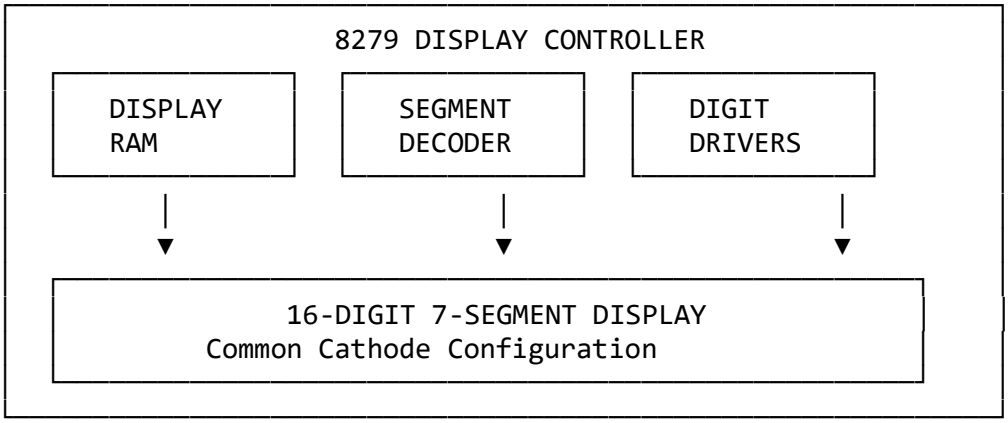
Configuration:
- Port A: Data output (to printer)
- Port B: Status input (from printer)
- Port C: Control signals

2.5 KEYBOARD INTERFACE (8279) - Address: C0400h-C04FFh



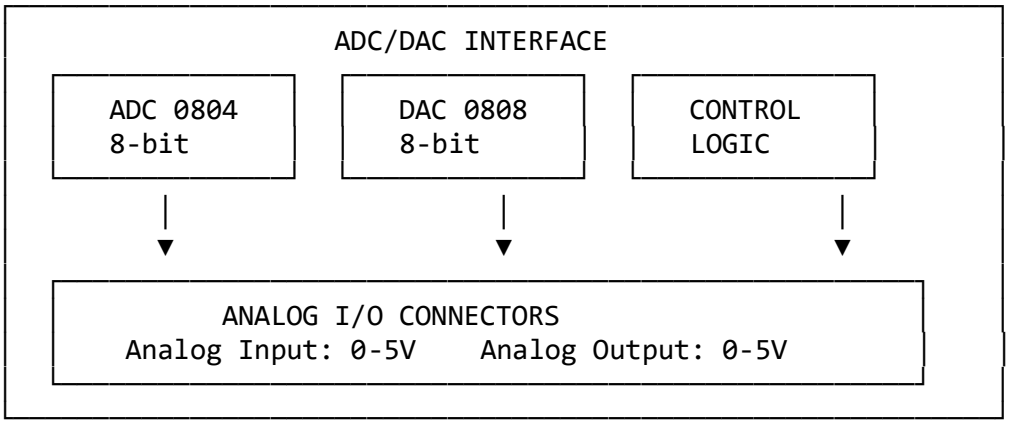
Features:
- 64-key matrix scanning
- Key debouncing
- FIFO buffer for key codes
- Interrupt generation on key press

2.6 DISPLAY INTERFACE (8279) - Address: C0500h-C05FFh



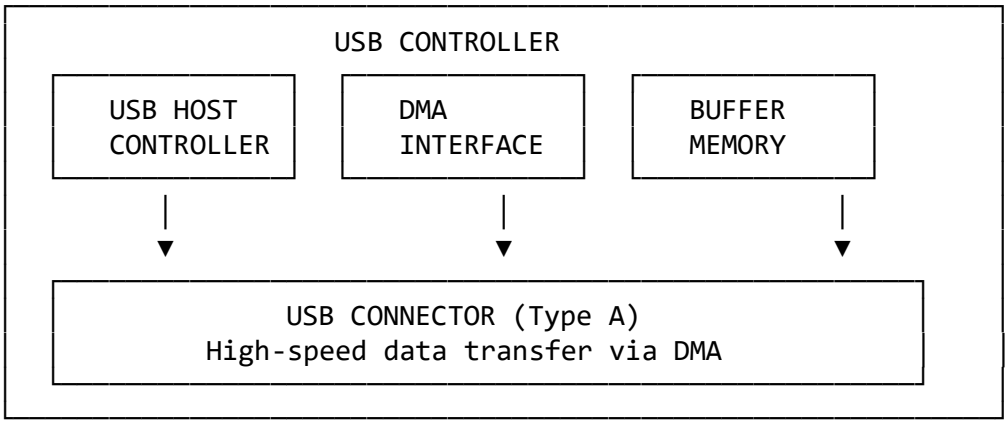
Features:
- 16 digits x 7 segments
- Automatic multiplexing
- Brightness control
- Hexadecimal display capability

2.7 ADC/DAC INTERFACE - Address: C0600h-C06FFh



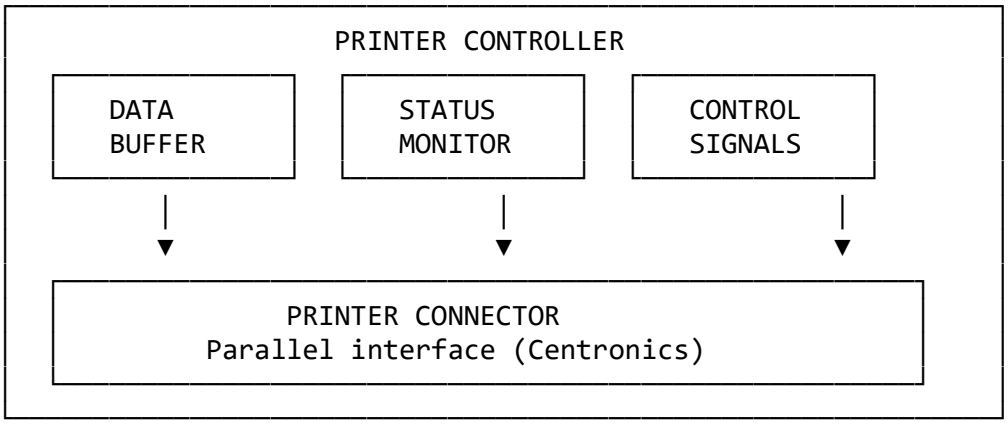
Specifications:
- ADC: 8-bit resolution, 0-5V input range
- DAC: 8-bit resolution, 0-5V output range
- Conversion time: <100µs

2.8 USB CONTROLLER + DMA - Address: C0700h-C07FFh



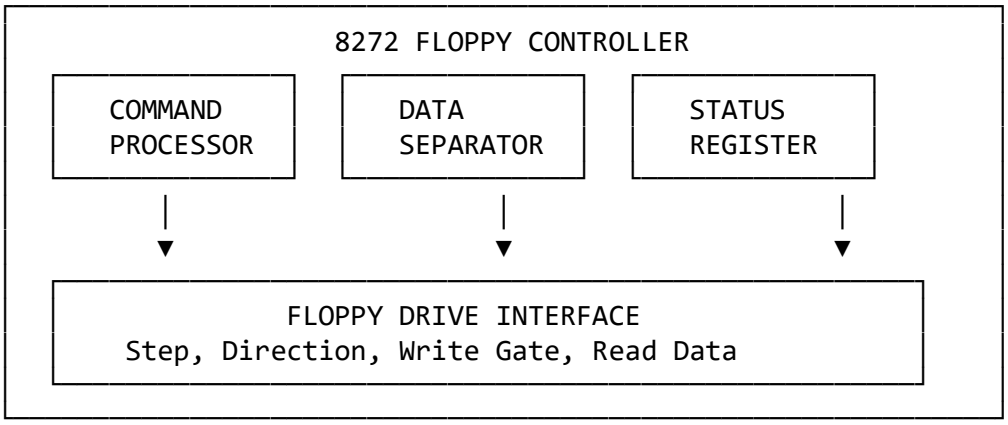
Features:
- USB 1.1 compatible
- DMA channel 1 for high-speed transfers
- Interrupt-driven operation

2.9 PRINTER CONTROLLER - Address: C0800h-C08FFh



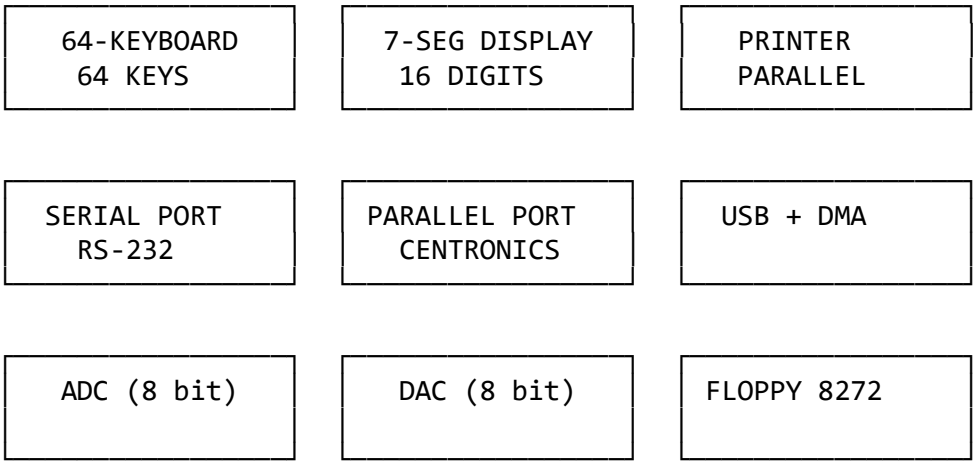
Features:
- Centronics parallel interface
- Automatic paper feed control
- Ready/Busy status monitoring

2.10 FLOPPY CONTROLLER 8272 - Address: C0900h-C09FFh



Features:
- Support for 5.25" and 3.5" drives
- Double density (360KB/720KB)
- DMA channel 0 for data transfers
- Built-in formatting capability

CONNECTED PERIPHERALS



3. DETAILED MEMORY MAP

The memory organization follows a structured approach with dedicated areas for different system functions. The 1MB address space is efficiently divided between RAM for user programs and data, ROM for system firmware, and I/O space for peripheral interfaces.

MEMORY MAP - 8086 SYSTEM (1 MBYTE)
Physical Addresses (20 bits) = 00000h to FFFFFh

ADDRESS	SIZE	TYPE	DESCRIPTION
00000h-003FFh	1KB	RAM	Interrupt Vector Table
00400h-3FFFFh	255KB	RAM	User Program Area
40000h-7FFFFh	256KB	RAM	User Data Area
80000h-BFFFFh	256KB	RAM	System Buffer/Stack
C0000h-CFFFFh	64KB	I/O	Peripheral Mapping Area
D0000h-DFFFFh	64KB	RESERVED	Reserved for Video & I/O
E0000h-F7FFFh	96KB	ROM	System BIOS
F8000h-FFFFFh	32KB	ROM	Boot BIOS

I/O AREA DETAILS (C0000h - CFFFFh):

ADDRESS	SIZE	DEVICE
C0000h-C00FFh	256B	Interrupt Controller (8259A)
C0100h-C01FFh	256B	DMA Controller (8237)
C0200h-C02FFh	256B	Serial Port (8251 UART)
C0300h-C03FFh	256B	Parallel Port (8255 PPI)
C0400h-C04FFh	256B	Matrix Keyboard (8279)
C0500h-C05FFh	256B	7-Segment Display (8279)
C0600h-C06FFh	256B	ADC/DAC Interface
C0700h-C07FFh	256B	USB Controller
C0800h-C08FFh	256B	Printer Controller
C0900h-C09FFh	256B	Floppy Controller (8272)
C0A00h-CFFFFh	~62KB	Reserved for Future Expansion

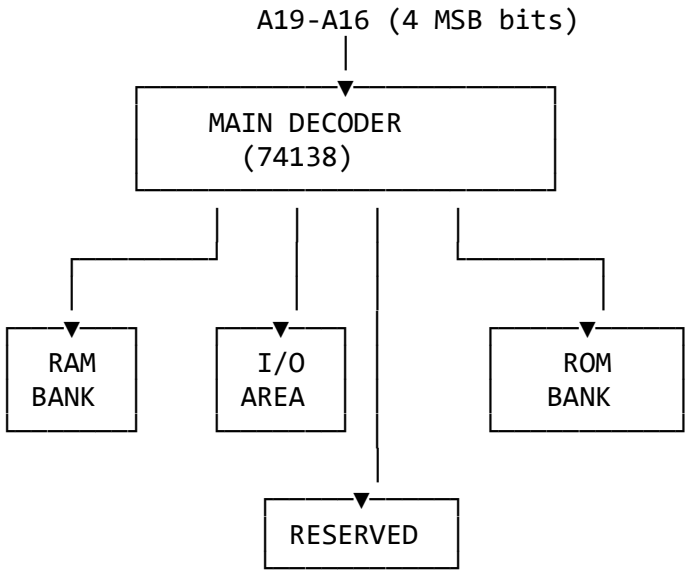
RESERVED AREA DETAILS (D0000h - DFFFFh):

ADDRESS	SIZE	PURPOSE
D0000h-D7FFFh	32KB	Reserved for Video Memory
D8000h-DFFFFh	32KB	Reserved for Additional I/O

4. ADDRESS DECODING SYSTEM

The address decoding system uses a hierarchical approach with a main decoder (74138) that examines the four most significant address bits (A19-A16) to select between memory and I/O regions. This design provides clear separation between different system areas and allows for easy expansion.

ADDRESS DECODER - HIERARCHICAL DESIGN



RAM DECODING (00000h-BFFFFh):
- Uses A19-A18 = 00, 01, 10
- Chips: 4 x DRAM 256KB (4164 or similar)
- Configuration: 2 banks x 2 chips each
- Row/column address multiplexing

ROM DECODING (E0000h-FFFFFh):
- Uses A19-A17 = 111
- Chips: 4 x EPROM 32KB (27256)
- Direct addressing A16-A0

I/O DECODING (C0000h-DFFFFh):
- Uses A19-A17 = 110
- Secondary decoder (74138)
- A16-A12 selects specific device

ADDRESS DECODER – DETAILED PIN CONNECTIONS

PRIMARY DECODER (74138)

Address Line	74138 Input Pin
A19	A
A18	B
A17	C

74138 Output Selection:

- **Y0 (00x):** RAM Bank 0 (00000h–3FFFFh)
- **Y1 (01x):** RAM Bank 1 (40000h–7FFFFh)
- **Y2 (10x):** RAM Bank 2 (80000h–BFFFFh)
- **Y3 (110):** I/O Area (C0000h–DFFFFh)
- **Y4 (111):** ROM Area (E0000h–FFFFFh)

MEMORY BANK SELECTION

RAM Banks (A19–A18):

- `00`: Bank 0 (256KB) – User Program
- `01`: Bank 1 (256KB) – User Data
- `10`: Bank 2 (256KB) – System/Stack
- `11`: Not used for RAM

I/O & ROM Selection (A19–A17):

- `110`: I/O Area
- `111`: ROM Area

SECONDARY DECODER (74138 for I/O)

Input: **A16–A12** for device selection

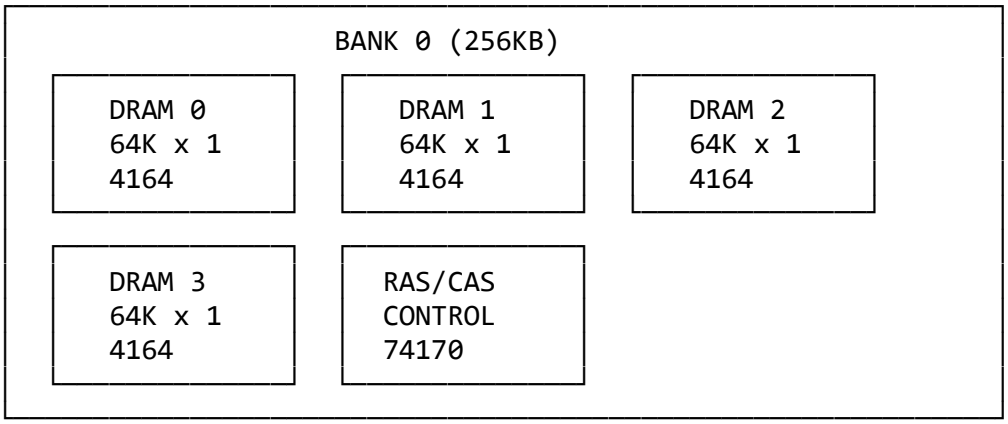
A16–A12	Y Output	Device
00000	Y0	8259A
00001	Y1	8237 DMA
00010	Y2	8251 UART
00011	Y3	8255 PPI
00100	Y4	Keyboard
00101	Y5	Display
00110	Y6	ADC/DAC
00111	Y7	USB

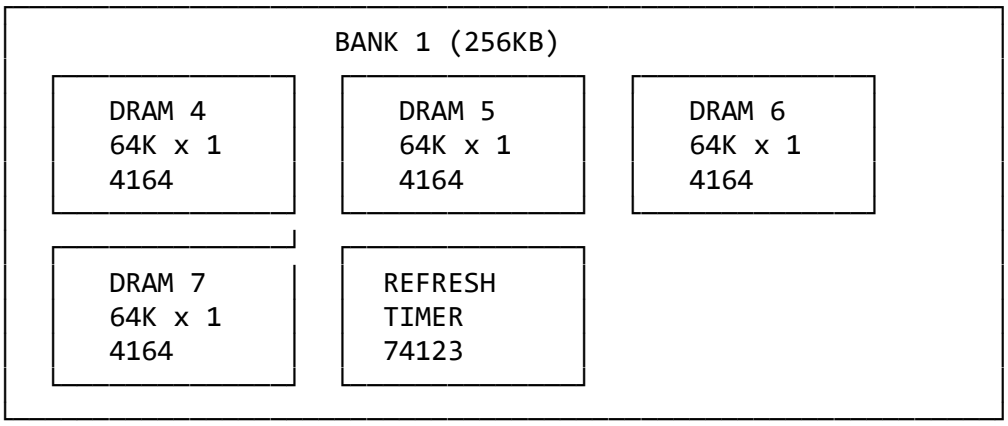
5. PHYSICAL MEMORY CONFIGURATION

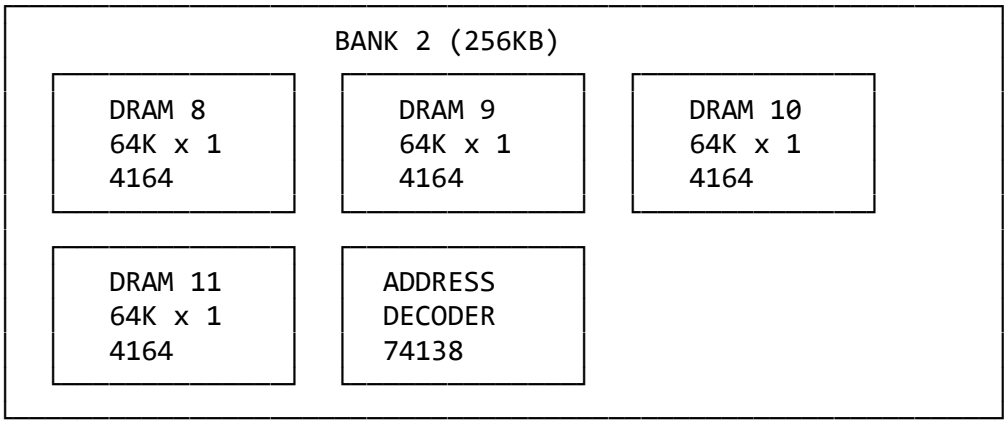
The physical memory implementation uses a combination of dynamic RAM for main memory and EPROM for system firmware. The DRAM configuration includes refresh circuitry to maintain data integrity, while the ROM provides non-volatile storage for BIOS and boot routines.

MEMORY CHIP CONFIGURATION

RAM - DYNAMIC DRAM CONFIGURATION:

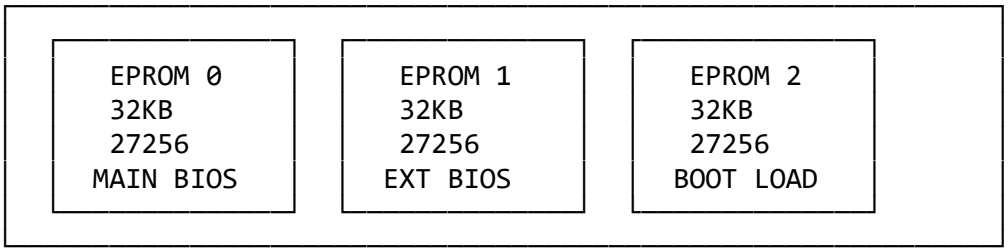






Note: Each bank uses four 4164 DRAM chips (64K x 1 bit) to provide 256KB with 16-bit data bus width. All banks use identical chip configurations for uniformity and simplified control logic.

ROM - EPROM CONFIGURATION:



MEMORY CONTROL SIGNALS:

Signal	Function
RAS	Row Address Strobe (DRAM timing)
CAS	Column Address Strobe (DRAM timing)
WE	Write Enable (active low)
OE	Output Enable (ROM/RAM read)
CS	Chip Select (from address decoder)

6. SYSTEM IMPLEMENTATION AND CODE REFERENCES

6.1 Complete System Integration

This microcomputer system, as documented and implemented by the team, fulfills all requirements for Part I-B (teams of 4 members):

- ✅ **8086 CPU** at 10 MHz with **8087 coprocessor**

- ✅ **1MB memory** (768 KB RAM + 128 KB ROM + 128 KB mapped I/O area)
- ✅ **Parallel connector** (8255 PPI)
- ✅ **Serial connector** (8251 UART)
- ✅ **ADC and DAC** interfaces for data conversion
- ✅ **16-digit seven-segment display** (8279)
- ✅ **64-key matrix keyboard** (8279)
- ✅ **USB controller with DMA** for high-speed transfers
- ✅ **Printer connector** (Centronics/parallel interface)
- ✅ **Interrupt controller** (8259A) managing I/O priorities
- ✅ **Diskette controller** (8272) for floppy disk operations
- ✅ **DMA controller** (8237) with 4 channels for peripheral and memory transfers

All system components are represented in the architectural diagrams, memory map, and explanations throughout `CPU_Memory.md`.

6.2 Programming and Code Files

The complete system initialization, memory management, and driver logic is provided across the following files:

Assembly Code: `CPU_Memory.asm` * System initialization routines and structure (with documented procedure stubs) * Segment and stack configuration * Memory and coprocessor test pattern setup * Team base address definitions for all peripherals

Pseudocode Documentation: `CPU_Memory_pseudocode.md` * Step-by-step pseudocode for system and memory initialization * Coprocessor (8087) initialization and test routines * Memory bank configuration and ROM verification * Address mapping and coordination between team members * Standard memory read/write/copy routines

For a full overview of architecture and interface details, refer to the block diagrams, memory map, and explanations throughout `CPU_Memory.md`.

6.3 Team Address Coordination

To enable clear collaboration and avoid address conflicts, the I/O address space is explicitly partitioned for each team member as follows:

Team Member	Devices	Address Range(s)	Allocated Size
Person 2	Keyboard, Display, Printer	C0400h–C05FFh, C0800h–C08FFh	512 B
Person 3	Interrupts, Serial, Parallel, USB + DMA	C0000h–C03FFh, C0700h–C07FFh	1280 B
Person 4	ADC/DAC, Floppy Controller (8272)	C0600h–C06FFh, C0900h–C09FFh	512 B

Note: All assignments match the memory map and decoding diagrams; see the I/O details in Section 2 and the pseudocode address table for further reference.

6.4 System Specifications Summary

- **CPU:** Intel 8086 @ 10 MHz
- **Coprocessor:** Intel 8087 (floating-point)
- **Total Memory:** 1 MB (768 KB RAM, 128 KB ROM, 128 KB I/O mapped)
- **Main Peripherals:** 8255 PPI (parallel), 8251 UART (serial), 8259A (interrupt), 8237 (DMA), 8279 (display/keyboard), ADC 0804, DAC 0808, USB interface, printer, 8272 floppy controller
- **DMA Channels:** 4 (8237)
- **Interrupt Levels:** 8 (8259A)
- **Storage:** 8272 floppy disk controller (5.25”/3.5” drives, double density)
- **User I/O:** 64-key keyboard, 16-digit 7-segment display
- **Analog I/O:** 8-bit ADC and DAC
- **Modern Expansion:** USB with DMA support ## 7. 8086/8087 PHYSICAL INTEGRATION

7.1 Signal Connections and Bus Sharing

8086/8087 PHYSICAL CONNECTIONS

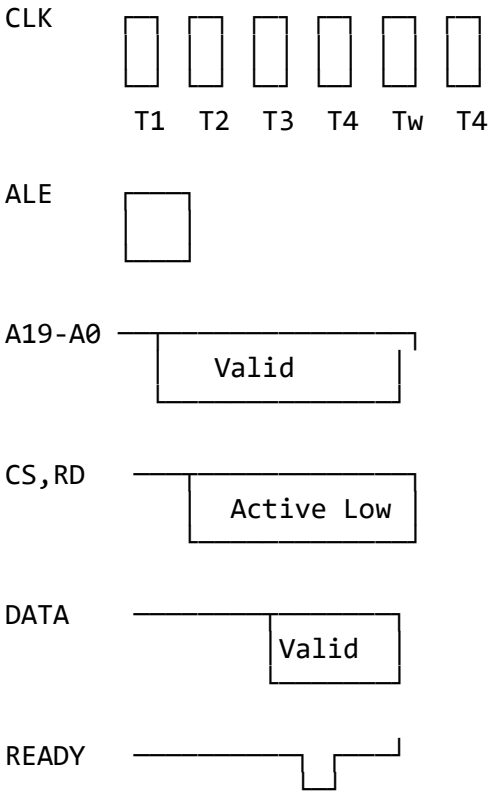
8086 Signal	8087 Signal	Function
TEST/	BUSY/	Coprocessor Status
QS0,QS1	QS0,QS1	Queue Status
S0-S2	S0-S2	Bus Cycle Status
CLK	CLK	10MHz System Clock
READY	READY	Wait State Control
RQ/GT0	REQUEST	Bus Request/Grant

7.2 Bus Arbitration and Timing

1. **Bus Control**
   - 8087 monitors all bus cycles through status lines
   - Uses RQ/GT protocol for bus access
   - Maintains instruction synchronization via queue status
2. **Clock Synchronization**
   - Both processors share 10MHz clock
   - CLK signal distributed with minimal skew
   - Maximum 100ns cycle time at full speed
3. **Wait State Generation**
   - System can insert wait states via READY signal
   - Typical memory cycle: 4 clock periods
   - Wait states added for slower peripherals

7.3 Memory Access Timing (10MHz Operation)

Basic Read/Write Cycle Timing Diagram:



Timing Notes: - **T1–T4:** Standard bus cycle states. - **Tw:** Optional wait state inserted if READY is not asserted. - **ALE:** Address Latch Enable pulse at start of cycle. - **A19–A0:** Address lines valid after ALE. - **CS, RD:** Chip Select and Read/Write signals active low during access. - **DATA:** Data lines valid during T3/Tw. - **READY:** If low, CPU inserts wait state(s) (Tw) before completing cycle.

Note: The timing diagram shows a typical memory access with one wait state (Tw). The system can operate at 10MHz with proper wait state insertion for different memory and I/O devices.

PSEUDOCODE - JARED: SYSTEM AND MEMORY INITIALIZATION

1. GENERAL SYSTEM INITIALIZATION PSEUDOCODE

```
FUNCTION InitializeSystem()
BEGIN
    // === INITIAL 8086 CPU CONFIGURATION ===
    CLI                             // Disable interrupts

    // Configure base segments
    CS = 0xF000                     // BIOS code segment
    DS = 0x0000                     // User data segment
    ES = 0x0000                     // Extra segment
    SS = 0x9000                     // Stack segment (high memory)
    SP = 0xFFFE                     // Stack pointer to top

    // === INITIALIZE CONTROLLERS ===
    CallFunction InitializeMemory()
    CallFunction InitializeCoprocessor8087()
    CallFunction ConfigureBuses()

    // === CONFIGURE INTERRUPT VECTOR TABLE ===
    ForEach i FROM 0 TO 255 DO
        WriteMemory(i*4, 0x0000)    // Vector offset
        WriteMemory(i*4+2, 0xF000)  // Vector segment (BIOS)
    EndFor

    // === ENABLE SYSTEM ===
    STI                             // Enable interrupts
    WritePort(0x21, 0x00)           // Enable all interrupts

    SHOW "8086 system initialized successfully"
    RETURN SUCCESS
END

FUNCTION ConfigureBuses()
BEGIN
    // Configure bus controller (8288)
    WritePort(0x80, 0x00)           // Bus in system mode
    WritePort(0x81, 0xFF)           // Enable all lines

    // Configure bus timing
    WritePort(0x82, 0x0F)           // Wait states = 15 (slow but safe)
    WritePort(0x83, 0x00)           // No prefetch
END
```

2. MEMORY INITIALIZATION PSEUDOCODE

```
FUNCTION InitializeMemory()
BEGIN
    // === MEMORY CONFIGURATION CONSTANTS ===
    // All banks use 4164 DRAM chips (64K x 1 bit)
    // Each bank has 4 chips for 16-bit words
    // Total RAM: 3 banks x 256KB = 768KB

    // === INITIALIZE DRAM CONTROLLER ===
    WritePort(0x90, 0x01)            // Reset DRAM controller
    Wait(10 milliseconds)
    WritePort(0x90, 0x00)            // End reset

    // === CONFIGURE DRAM REFRESH ===
    // Configure 74123 timer for refresh every 2ms
    WritePort(0x91, 0x7D)            // Refresh period = 2ms
    WritePort(0x92, 0x01)            // Enable auto-refresh

    // === CONFIGURE RAS/CAS TIMING ===
    WritePort(0x95, 0x02)            // RAS timing: 2 clocks
    WritePort(0x96, 0x01)            // CAS timing: 1 clock
    WritePort(0x97, 0x01)            // Precharge: 1 clock

    // === INITIALIZE MEMORY BANKS ===
    CallFunction InitializeBank0()    // Program Area (00000h-3FFFFh)
    CallFunction InitializeBank1()    // Data Area (40000h-7FFFFh)
    CallFunction InitializeBank2()    // System Area (80000h-BFFFFh)
    CallFunction VerifyROM()          // Verify ROM integrity

    // === CLEAR RAM MEMORY ===
    ForEach address FROM 0x00000 TO 0xBFFFF DO
        IF (address >= 0x00400) THEN  // Do not erase INT vectors
            WriteMemory(address, 0x00)
        EndIF
    EndFor

    // === CONFIGURE STACK AREA ===
    ForEach address FROM 0x90000 TO 0x9FFFF DO
        WriteMemory(address, 0xAA)    // Stack pattern
    EndFor

    SHOW "Memory initialized: 1MB total"
    RETURN SUCCESS
END

FUNCTION InitializeBank0()
BEGIN
    // Bank 0: 256KB DRAM (00000h-3FFFFh)
    WritePort(0x93, 0x00)            // Select bank 0
```

```
    WritePort(0x94, 0x44)           // 4164 DRAM configuration

    // Memory pattern test (after INT vector table)
    WriteMemory(0x00400, 0xAA55)
    valueRead = ReadMemory(0x00400)
    IF (valueRead != 0xAA55) THEN
        SHOW "ERROR: Bank 0 defective"
        RETURN ERROR
    EndIF

    SHOW "Bank 0 OK: 256KB (Program Area)"
    RETURN SUCCESS
END

FUNCTION InitializeBank1()
BEGIN
    // Bank 1: 256KB DRAM (40000h-7FFFFh)
    WritePort(0x93, 0x01)           // Select bank 1
    WritePort(0x94, 0x44)           // 4164 DRAM configuration

    // Memory pattern test
    WriteMemory(0x40000, 0x55AA)
    valueRead = ReadMemory(0x40000)
    IF (valueRead != 0x55AA) THEN
        SHOW "ERROR: Bank 1 defective"
        RETURN ERROR
    EndIF

    SHOW "Bank 1 OK: 256KB (Data Area)"
    RETURN SUCCESS
END

FUNCTION InitializeBank2()
BEGIN
    // Bank 2: 256KB DRAM (80000h-BFFFFh)
    WritePort(0x93, 0x02)           // Select bank 2
    WritePort(0x94, 0x44)           // 4164 DRAM configuration

    // Memory pattern test
    WriteMemory(0x80000, 0x55AA)
    valueRead = ReadMemory(0x80000)
    IF (valueRead != 0x55AA) THEN
        SHOW "ERROR: Bank 2 defective"
        RETURN ERROR
    EndIF

    SHOW "Bank 2 OK: 256KB (System/Stack Area)"
    RETURN SUCCESS
END
```

```
FUNCTION VerifyROM()
BEGIN
    // Verify BIOS ROM checksum
    checksum = 0
    ForEach address FROM 0xE0000 TO 0xFFFFF DO
        checksum = checksum + ReadMemory(address)
    EndFor

    IF (checksum AND 0xFF != 0) THEN
        SHOW "WARNING: ROM checksum incorrect"
        RETURN WARNING
    EndIF

    SHOW "ROM verified OK: 128KB"
    RETURN SUCCESS
END
```

3. 8087 COPROCESSOR INITIALIZATION PSEUDOCODE

```
FUNCTION InitializeCoprocessor8087()
BEGIN
    // === PHYSICAL INTEGRATION NOTES ===
    // 8087 shares clock with 8086 (10MHz)
    // TEST pin from 8086 connects to BUSY pin of 8087
    // Both processors monitor QS0/QS1 for queue synchronization
    // READY signals are wire-ANDed for wait state generation

    // === VERIFY SIGNAL CONNECTIONS ===
    CheckBusySignal()               // Verify TEST/BUSY connection
    CheckQueueSync()                // Verify QS0/QS1 connections

    // === DETECT 8087 PRESENCE ===
    IF (NOT DetectCoprocessor()) THEN
        SHOW "WARNING: Coprocessor 8087 not detected"
        RETURN ERROR
    EndIF

    // === INITIALIZE COPROCESSOR ===
    ExecuteInstruction(FINIT)         // Initialize FPU
    Wait(100 microseconds)            // Stabilization time

    // === CONFIGURE FPU CONTROL ===
    controlWord = 0x037F              // 64-bit precision, all exceptions
masked
    ExecuteInstruction(FLDCW controlWord)

    // === FUNCTIONALITY TEST ===
    ExecuteInstruction(FLD1)          // Load 1.0
    ExecuteInstruction(FLD1)          // Load another 1.0
```

```
    ExecuteInstruction(FADD)          // Add: 1.0 + 1.0
    ExecuteInstruction(FIST result)   // Store result

    IF (result != 2) THEN
        SHOW "ERROR: Coprocessor 8087 defective"
        RETURN ERROR
    EndIF

    ExecuteInstruction(FINIT)         // Clear FPU stack
    SHOW "Coprocessor 8087 initialized successfully"
    RETURN SUCCESS
END

FUNCTION DetectCoprocessor()
BEGIN
    // Write test pattern to control register
    ExecuteInstruction(FNINIT)
    ExecuteInstruction(FNSTSW AX)     // Read status word

    IF (AX AND 0xFF00 == 0x0000) THEN
        RETURN TRUE                   // 8087 present
    ELSE
        RETURN FALSE                  // No 8087
    EndIF
END
```

4. BASIC MEMORY ROUTINES PSEUDOCODE

```
FUNCTION ReadMemory(physicalAddress)
BEGIN
    // Convert physical address to segment:offset
    segment = physicalAddress >> 4
    offset = physicalAddress AND 0x000F

    // Set segment registers
    DS = segment

    // Read data using indirect addressing
    value = [DS:offset]

    RETURN value
END

FUNCTION WriteMemory(physicalAddress, value)
BEGIN
    // Convert physical address to segment:offset
    segment = physicalAddress >> 4
    offset = physicalAddress AND 0x000F

    // Set segment registers
```

```
    DS = segment

    // Write data using indirect addressing
    [DS:offset] = value

    RETURN SUCCESS
END

FUNCTION CopyBlock(source, destination, size)
BEGIN
    ForEach i FROM 0 TO (size-1) DO
        value = ReadMemory(source + i)
        WriteMemory(destination + i, value)
    EndFor

    RETURN SUCCESS
END

FUNCTION FillMemory(startAddress, size, pattern)
BEGIN
    ForEach i FROM 0 TO (size-1) DO
        WriteMemory(startAddress + i, pattern)
    EndFor

    RETURN SUCCESS
END
```

5. ADDRESS MANAGEMENT FOR TEAM PSEUDOCODE

```
// === ADDRESS TABLE FOR TEAM COORDINATION ===

CONSTANTS TeamAddresses
    // Base addresses for each person
    BASE_INTERRUPTS = 0xC0000        // Person 3
    BASE_DMA = 0xC0100               // Person 3
    BASE_SERIAL = 0xC0200            // Person 3
    BASE_PARALLEL = 0xC0300          // Person 3
    BASE_KEYBOARD = 0xC0400          // Person 2
    BASE_DISPLAY = 0xC0500           // Person 2
    BASE_ADC_DAC = 0xC0600           // Person 4
    BASE_USB = 0xC0700               // Person 3
    BASE_PRINTER = 0xC0800           // Person 2
    BASE_FLOPPY = 0xC0900            // Person 4
EndConstants

FUNCTION AssignAddresses()
BEGIN
    SHOW "=== ADDRESS MAP FOR TEAM ==="
    SHOW "Person 2 (Display/Keyboard):"
    SHOW "  - Keyboard: 0xC0400 - 0xC04FF"
```

```
    SHOW "  - Display: 0xC0500 - 0xC05FF"
    SHOW "  - Printer: 0xC0800 - 0xC08FF"

    SHOW "Person 3 (Serial/Parallel/USB/INT):"
    SHOW "  - Interrupts: 0xC0000 - 0xC00FF"
    SHOW "  - Serial: 0xC0200 - 0xC02FF"
    SHOW "  - Parallel: 0xC0300 - 0xC03FF"
    SHOW "  - USB+DMA: 0xC0700 - 0xC07FF"

    SHOW "Person 4 (ADC/DAC/Floppy):"
    SHOW "  - ADC/DAC: 0xC0600 - 0xC06FF"
    SHOW "  - Floppy 8272: 0xC0900 - 0xC09FF"
END
```

CPU_Memory.asm

```
; ========================================================================
; ASSEMBLY CODE - JARED: SYSTEM AND MEMORY INITIALIZATION
; System: 8086 10MHz + 8087 Coprocessor + 1MB RAM
; ========================================================================

.MODEL SMALL
.STACK 4096
.DATA

; === SYSTEM CONSTANTS ===
DRAM_CTRL_PORT       EQU 90h        ; DRAM control port
REFRESH_PORT         EQU 91h        ; Refresh timer port
BANK_SEL_PORT        EQU 93h        ; Memory bank selector port
DRAM_CONFIG_PORT     EQU 94h        ; DRAM configuration port

; === BASE ADDRESSES FOR TEAM ===
BASE_INTERRUPTS      EQU 0C0000h     ; Person 3
BASE_DMA             EQU 0C0100h     ; Person 3
BASE_SERIAL          EQU 0C0200h     ; Person 3
BASE_PARALLEL        EQU 0C0300h     ; Person 3
BASE_KEYBOARD        EQU 0C0400h     ; Person 2
BASE_DISPLAY         EQU 0C0500h     ; Person 2
BASE_ADC_DAC         EQU 0C0600h     ; Person 4
BASE_USB             EQU 0C0700h     ; Person 3
BASE_PRINTER         EQU 0C0800h     ; Person 2
BASE_FLOPPY          EQU 0C0900h     ; Person 4

; === SYSTEM VARIABLES ===
msg_start           DB 'Initializing 8086 System...', 0Dh, 0Ah, '$'
msg_memory_ok       DB 'Memory initialized: 1MB OK', 0Dh, 0Ah, '$'
msg_8087_ok         DB 'Coprocessor 8087 OK', 0Dh, 0Ah, '$'
msg_error_memory    DB 'ERROR: Memory failure', 0Dh, 0Ah, '$'
msg_error_8087      DB 'ERROR: Coprocessor 8087 not found', 0Dh, 0Ah, '$'
msg_system_ready    DB 'System ready for use', 0Dh, 0Ah, '$'

test_pattern_1      DW 0AA55h       ; Memory test pattern 1
test_pattern_2      DW 055AAh       ; Memory test pattern 2
memory_ok           DB 0            ; Memory status flag
coprocessor_ok      DB 0            ; 8087 status flag

.CODE

; ========================================================================
; MAIN FUNCTION - COMPLETE SYSTEM INITIALIZATION
; ========================================================================
start:
    mov ax, @data
    mov ds, ax
    mov es, ax

    mov ax, 9000h
    mov ss, ax
    mov sp, 0FFFEh
```

```
    cli

    lea dx, msg_start
    call show_message

    call configure_buses
    call initialize_memory
    call initialize_8087

    ; (Other initialization as needed)

    hlt                 ; Halt CPU (placeholder for end of main routine)

; ========================================================================
; === DUMMY PROCEDURE STUBS (for documentation, not implemented) ===
; ========================================================================
show_message PROC
    ret
show_message ENDP

configure_buses PROC
    ret
configure_buses ENDP

initialize_memory PROC
    ret
initialize_memory ENDP

initialize_8087 PROC
    ret
initialize_8087 ENDP

END start
```

I/O Peripherals - 8086 Microprocessor Project

This section documents the I/O peripheral interfaces used in the 8086-based microcomputer system. The peripherals include:

• 8255 Programmable Peripheral Interface for the printer interface (parallel communication).

• 8251 Universal Synchronous/Asynchronous Receiver Transmitter (USART) for serial communication.

• 8279 Programmable Keyboard Interface for a 64-key matrix keyboard.

• 8279 Display Interface for a 16-digit seven-segment display.

1. 8255 - Parallel Interface

Purpose

The Intel 8255 is a Programmable Peripheral Interface used to connect parallel I/O devices. It is used to interface with the printer.

Ports

• Port A: Used to send data to the printer.

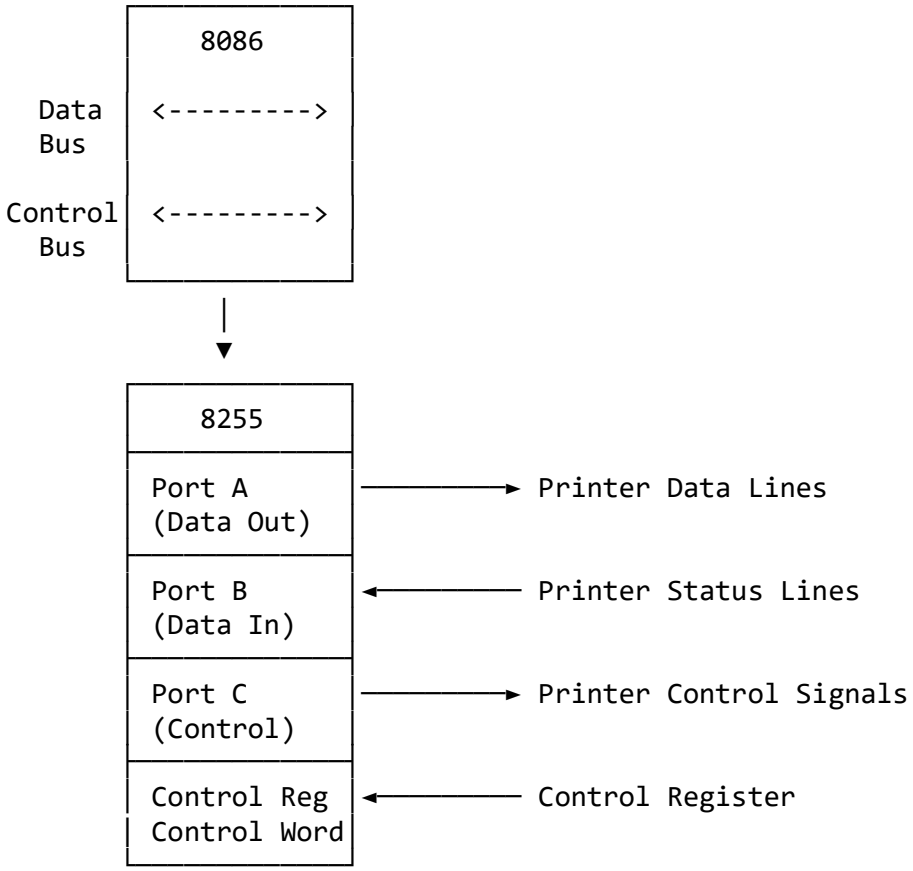
• Port B: Optional use for control signals or status.

• Port C: Used to send control signals.

Address Mapping

Register	Address
Port A	0xC0800
Port B	0xC0801
Port C	0xC0802
Control	0xC0803

Block Diagram



Code

```
; Initialize 8255
MOV AL, 10000000b ; Control Word: Port A & C output
OUT C0803h, AL ; Write control word to 8255 control port
; Send example data byte to printer via Port A
MOV AL, 0A5h ; Example data
OUT C0800h, AL ; Output data to Port A
; Generate strobe control signal on Port C
MOV AL, 01h ; Strobe bit set
OUT C0802h, AL ; Output control signals via Port C
```

2. 8251 - Serial Interface (Serial Port)

Purpose

The Intel 8251 is a USART for serial communication. It is used to send and receive data serially through a serial connector.

Components:

- *Transmit Buffer (TX Buffer):*

Temporarily holds data bytes that the CPU wants to send out serially.

- *Transmit Control:*

Manages the timing and framing of outgoing serial data, including start, stop bits, parity, and baud rate control.

- *Receive Buffer (RX Buffer):*

Holds incoming serial data received from an external device, buffering it for the CPU to read.

- *Receive Control:*

Handles synchronization, error detection, and framing of incoming serial data.

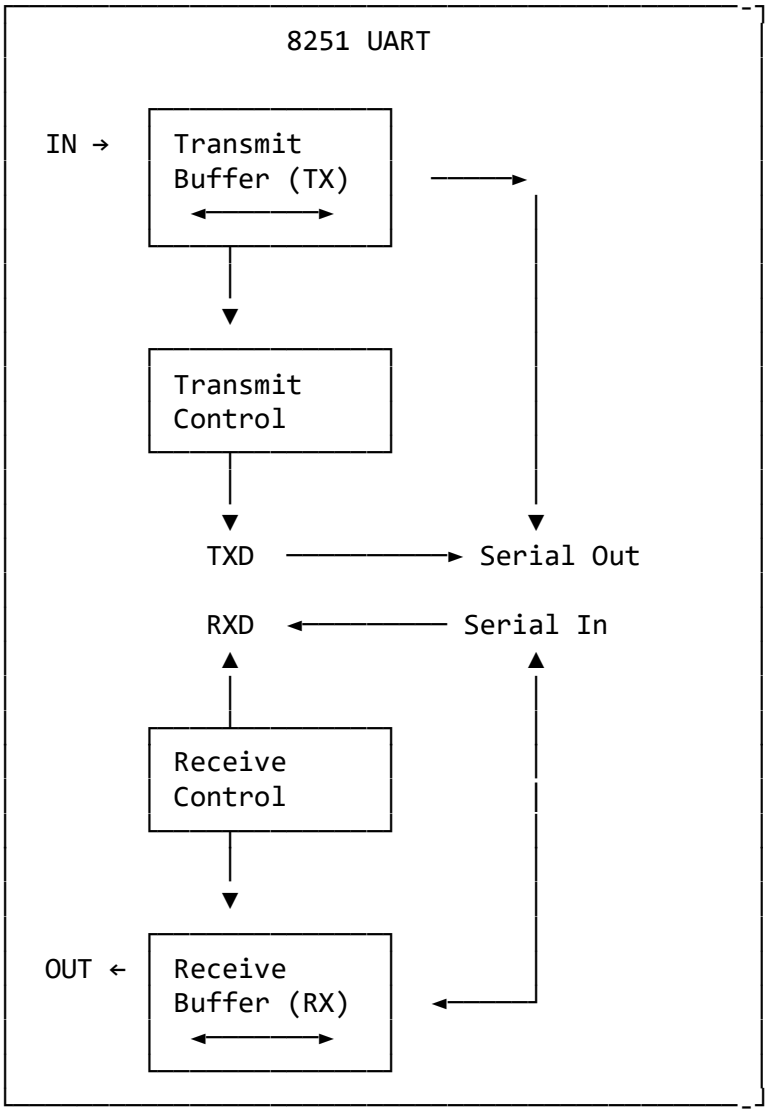
- *Baud Rate Generator (may be external or internal):*

Sets the speed (bits per second) of serial data transmission and reception.

Address Mapping:

Register	Address
Mode	0xC0200
Command	0xC0200
Transmit	0xC0201
Receive	0xC0201

Block Diagram



Code

; Initialize 8251

MOV AL, 01011110b ; Mode: async, 8-bit, 1 stop bit

OUT C0200h, AL ; Mode register

MOV AL, 00001011b ; Command: enable TX and RX

OUT C0200h, AL ; Command register

; Transmit example data byte ‘A’

MOV AL, 41h ; Data byte ACII ‘A’

OUT C0201h, AL ; Data register

IN AL, C0201h ; Read data from port

3. 8279 - Keyboard Interface

Purpose

The Intel 8279 is used to interface a 64-key matrix keyboard.

Components:

- *64-Key Matrix Scanning:*

The 8279 scans an 8-row by 8-column keyboard matrix, allowing it to detect key presses efficiently without needing one input line per key.

- *Debouncing:*

The controller automatically filters out mechanical key bounce, ensuring that each key press is registered cleanly as a single event.

- *FIFO Buffer:*

Key codes detected by the 8279 are stored in a First-In-First-Out buffer, allowing the CPU to read key presses asynchronously without losing data.

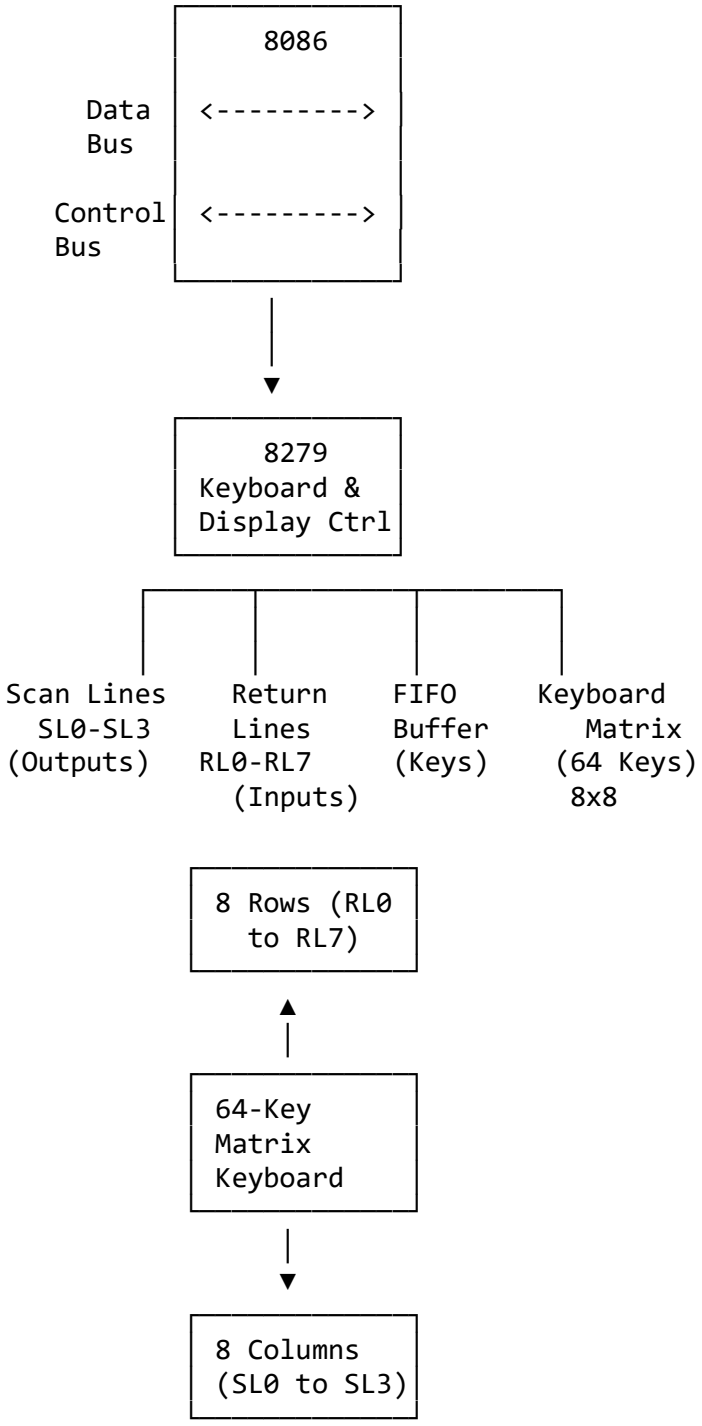
- *Interrupt Generation:*

The 8279 can generate an interrupt signal to the CPU when a key is pressed, enabling responsive, interrupt-driven input handling.

Address Mapping:

Register	Address
Command/Mode	0xC0400
Data	0xC0401

Block Diagram



Code

```
; Keyboard initialization
MOV AL, 00001111b ; Keyboard/display mode setup
OUT C0400h, AL ; Command port
MOV AL, 00000001b ; Enable auto-increment addressing
OUT C0400h, AL ; Command port
; Read key code from FIFO
IN AL, C0401h ; Keyboard data port
```

4. 8279 - Display Interface

Purpose

The Intel 8279 also controls the 16-digit 7-segment display.

Components:

- *16-Digit Multiplexed Display:*

Supports up to 16 digits, each with 7 segments (plus optional decimal point), using multiplexing.

- *Automatic Multiplexing:*

The controller automatically cycles through each digit rapidly, lighting one digit at a time in sequence.

- *Display RAM:*

Holds the digit codes for each of the 16 digits. The CPU writes the desired segment patterns into this RAM, which the 8279 uses to drive the display.

- *Segment Decoder:*

Converts the stored digit data into signals that turn on/off individual segments (a through g) on the 7-segment LEDs.

- *Digit Drivers:*

Enable the activation of each digit's common line, controlling which digit is currently lit during multiplexing.

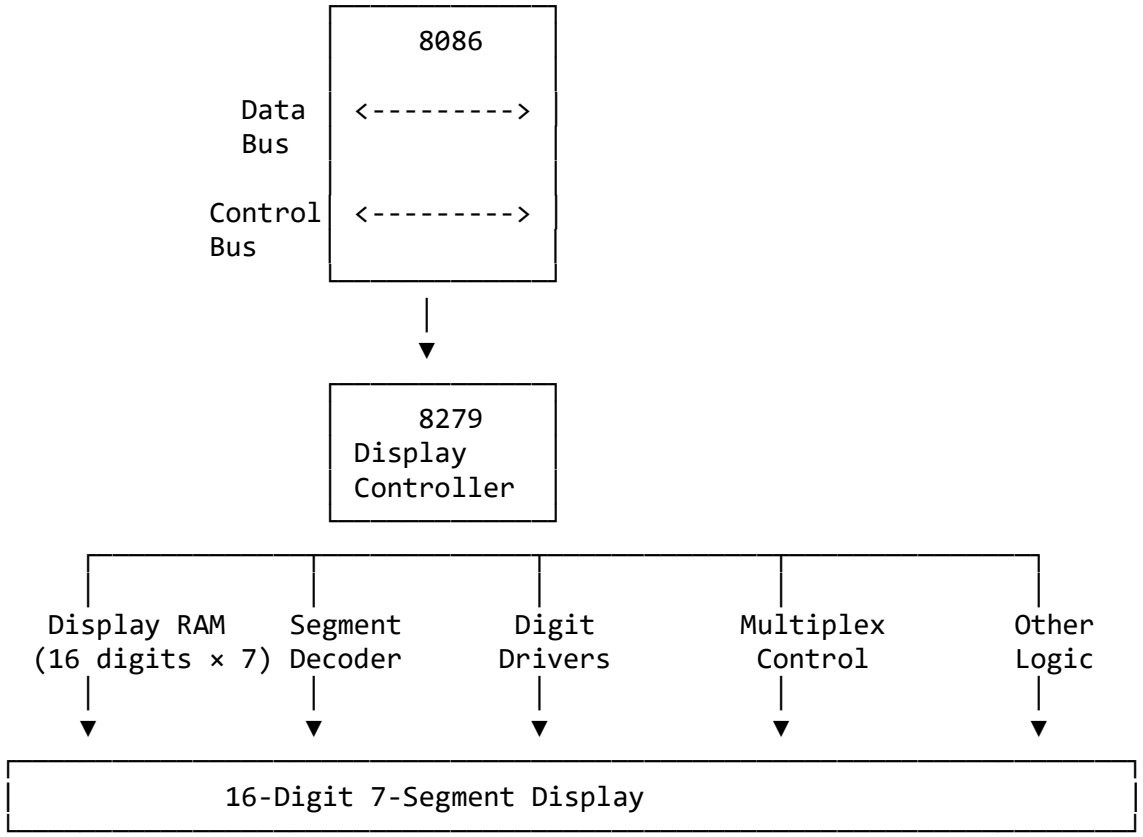
• *Brightness Control:*

Allows adjustment of display brightness by controlling the duty cycle of the multiplexing signal.

Address Mapping:

Register	Address
Command/mode	0xC0500
Data	0xC0501

Block Diagram



Code

; Display initialization

MOV AL, 00001111b ; Display mode setup

OUT C0500h, AL ; Display command port

MOV AL, 00000001b ; Enable auto-increment addressing

OUT C0500h, AL ; Display command port

; Output digit data to display RAM

```
MOV AL, 3Fh ; Digit '0' 7-segment pattern

OUT C0501h, AL ; Display RAM data port
```

Conclusion

The IO Peripherals document provides the integration of key I/O peripherals essential to the functionality of the 8086 microcomputer system. Each peripheral is mapped into the memory space with proper address decoding and provides the necessary parallel, serial, keyboard, and display interfaces.

Data Conversion, USB (DMA), and Interrupts - 8086 Microporcessor/Microcomputer Project

Valeria S. Almodóvar Santiago

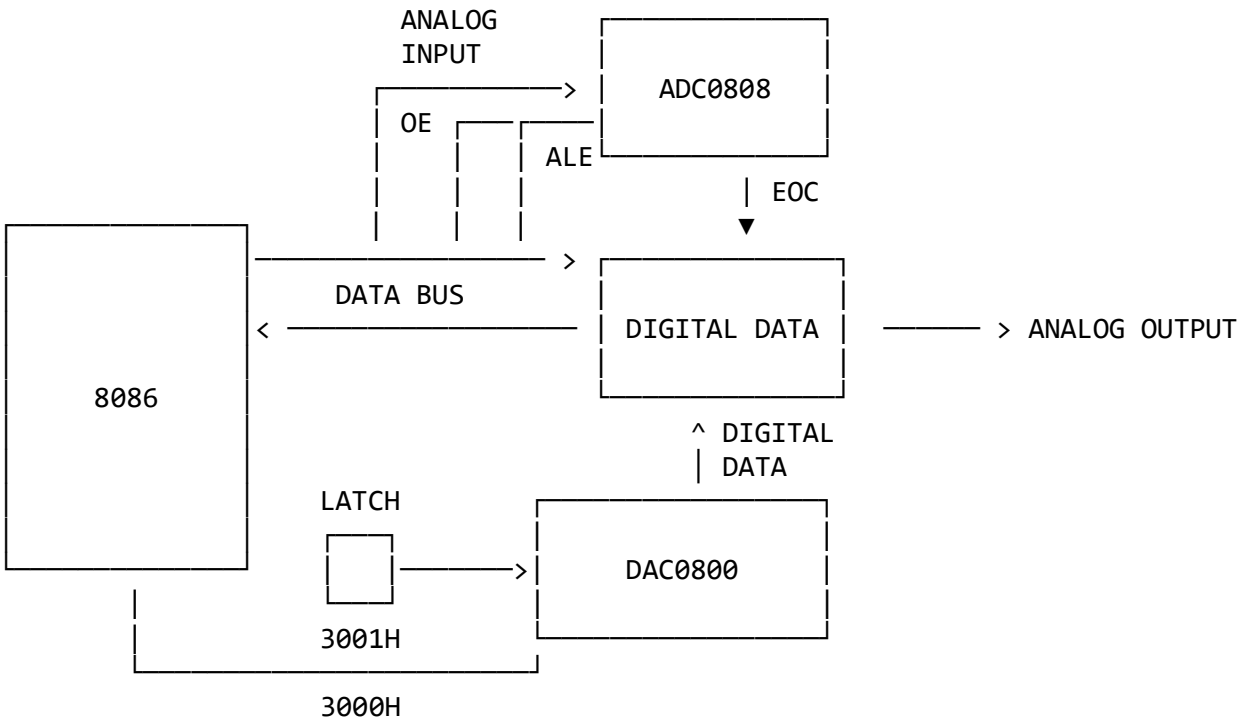
This document outlines the implementation of the ADC/DAC interface, USB via DMA and interrupt management using the 8259A controller in the 8086 microcomputer system.

1. ADC and DAC Interacing

Components used:

- ADC0808 (Analog-to-Digital Converter)
- DAC0800 (Digital-to-Analog Converter)

Block Diagram:



Description:

ADC0808 (Analog-to-Digital Converter)

- Accepts an analog voltage input (0-5V) and outputs an 8-bit digital value.
- Controlled via:
  - ‘ALE’ (Address Latch Enable): Enable address.

  - ‘START’: Begins conversion.
  - ‘OE’ (output Enable): Enables digital output.
  - ‘EOC’ (End of Conversion): Output signal that goes LOW when the converison is complete
- The 8086 reads the converted data via the data bus.

DAC0800 (Digital-to-Analog Converter)

- Recieves an 8-bit digital value form the 8086.
- Requires a LATCH to hold the data stable while converting it to analog.
- Outputs an analog voltage (0-5V) proportional to the digital value.

Address Mapping

Device	Address Range
ADC0808	0x3000
DAC0800	0x3001

PSEUDOCODE

;Start ADC conversion

MOV AL, 01H ; Set START bit OUT 3000H, AL ; Send command to ADC (address 3000H)

WAIT_EOC: ; Wait for EOC to go low IN AL, 3000H ; Read status byte TEST AL, 80H ; Test EOC bit (bit 7) JNZ WAIT_EOC ; Wait until EOC = 0

; Read converted digital value

IN AL, 3000H ; Read ADC result

; Send output value to DAC

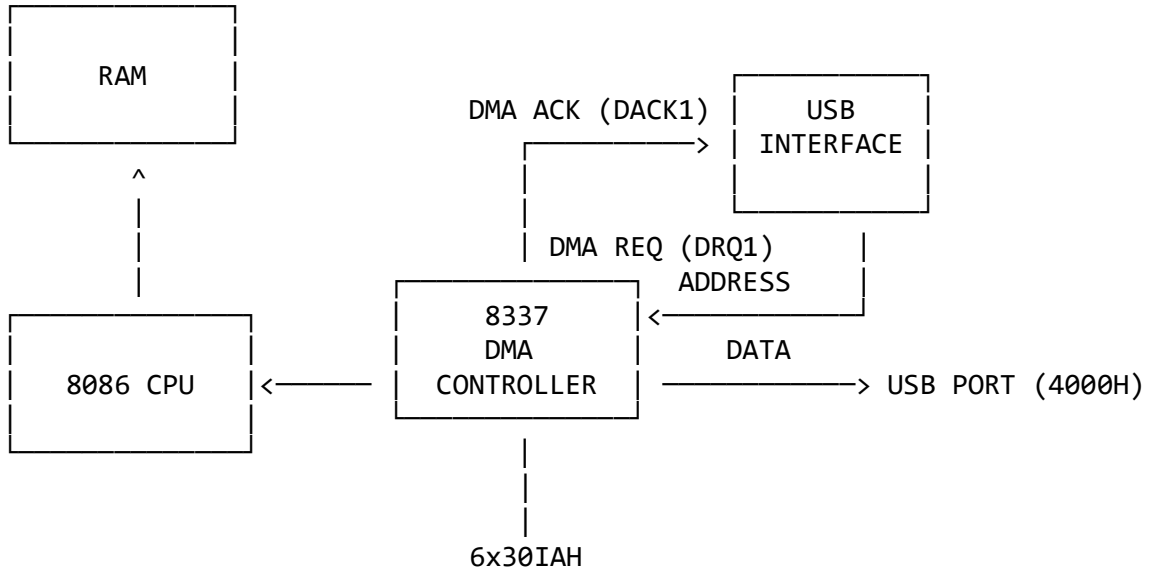
OUT 3001H, AL ; write to DAC latch (address 3001H)

2. USB Interface via DMA (8237)

Components used:

- USB peripherals (external or simulated)
- 8237 DMA Controller

Block Diagram:



Description:

ADC0808 (Analog-to-Digital Converter)

This section explains how the 8237 DMA controller is used to handle high-speed data transfer form system memory (RAM) to a USB interface without involving the CPU directly during the transfer phase.

- The 8086 CPU configures the 8237 DMA controller by provinding:
  - The source address in memory.
  - The destination address (USB port).
  - The word count (number of bytes to transfer).
  - The mode of operation (write, single/burst mode).
- The 8237 DMA controller then autonomously transfers the data from memory to the USB interface by:
  - Monitoring DMA Request (DRQ1) from the USB device.
  - Responding with DMA Acknowledge (DACK1) when the transfer is ready.
  - Managing the address and data buses without CPU intervention.
- The USB Interface is memory-mapped at address 0x4000, and acts as the DMA transfer destination.
- This mechanism significantly reduces CPU overhead and improves data throughout, making it ideal for high-speed preipheral communication like USB.

Address Mapping:

Component	Address
USB Port	0x4000
8237 DMA Control	0x5000 (base)

PSEUDOCODE

; Initialize 8237 DMA Controller ; Set source address, destination (USB port), word count, and mode ; Set Mode Register (Channel 1, write, single transfer)

MOV AL, 56H OUT 500BH, AL

; Set Base Address

MOV AX, OFFSET src_buffer OUT 5003H, AL ; Address low byte (channel 1) OUT 5004H, AH ; Address high byte

; Set Word Count

MOV AX, 0010H OUT 5005H, AL ; Count low OUT 5006H, AH ; Count high

; Trigger DMA transfer MOV AL, 01H OUT 500AH, AL ; Request DMA channel 1

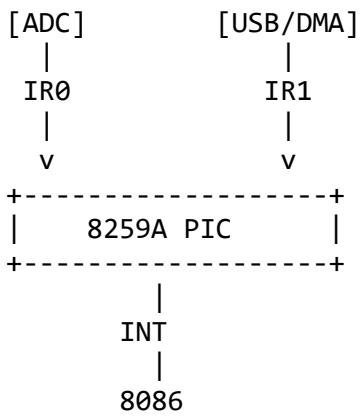
3. 8259A Interrupt Controller

Purpose:

The 8259A Programmable Interrupt Controller (PIC) allows the 8086 microprocessor handle hardware-generated interrupts from peripherals such as the ADC, USB (via DMA), and other I/O devices. It prioritizes and vectorizes interrupts usiing interrupt reuqests linesn (IR0-IR7)

Initialization:

- The PIC is initialized using ICWI-ICW4:
  - ICW1: Edge-triggered mode, single/cascaded mode slection.
  - ICW2: Sts the base interrupt vector.
  - ICW3: Specifies if a slave is connected,(if cascaded mode is enabled).
  - ICM4: Enables 8086/88 mode operation.
- In this setup, we're assuming a single 8259A with no slave connected (non-cascaded)



Address Mapping:

Register	Address
ICW1/OCW1	0x20
ICW2	0x21

Pseudocode:

; Initialize 8259A

MOV AL, 11H ; ICW1 - edge triggered, cascade OUT 20H, AL

MOV AL, 08H ; ICW2 - base interrupt vector 08H OUT 21H, AL

MOV AL, 04H ; ICW3 - slave on IR2 OUT 21H, AL

MOV AL, 01H ; ICW4 - 8086 mode OUT 21H, AL

; 8086 now enabled to receive maskable interrupts (INTR)

Conclusion:

This document outlines the integrations of key I/O and communication systems essential to the functionality of the 8086 microprossessor. Each section was designed to interact properly thorugh memory-mapped I/O and interrupt handling.

Storage & DMA Integration - 8086 Microprocessor Project

Giovanny Garcia

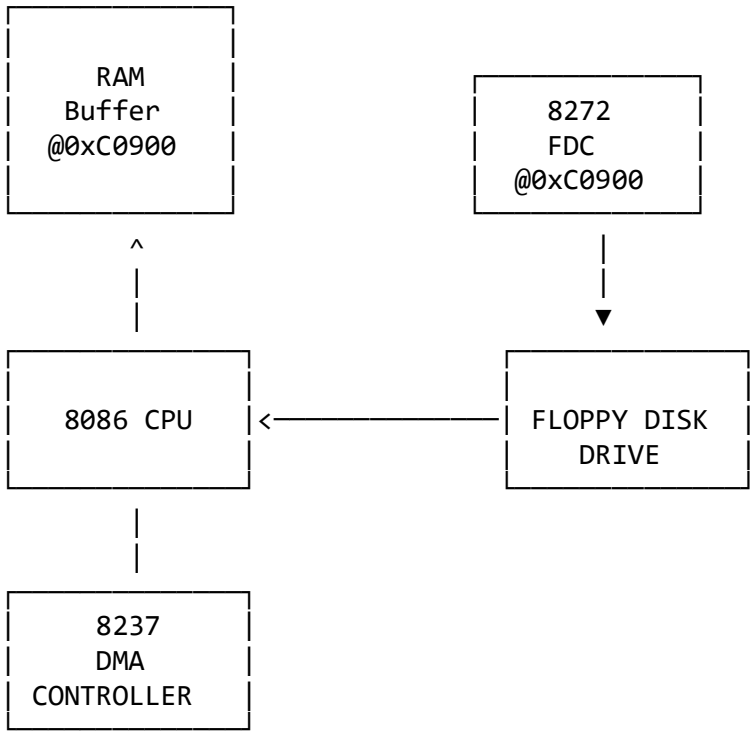
This document outlines the implementation of the 8272 floppy disk controller, DMA-based high-speed data transfer using the 8237 controller, USB transfers via DMA, and system integration for the 8086 microprocessor project.

1. 8272 Floppy Disk Controller

Components used:

- 8272 Floppy Disk Controller (FDC)
- Floppy disk drive interface
- DMA integration for high-speed transfers

Block Diagram:



Description:

8272 Floppy Disk Controller

- Manages read/write operations to floppy disks (360KB/720KB/1.44MB formats).
- Handles sector formatting, track positioning, and data transfer.
- Supports multiple drives (up to 4 drives).

- Uses DMA for high-speed data transfer to minimize CPU overhead.
- Provides status monitoring for drive ready, write protect, and track positioning.

Address Mapping:

Register	Address
Status Register	0xC0900
Command Register	0xC0901
Data Register	0xC0902
Control Register	0xC0903

PSEUDOCODE

```
FUNCTION Initialize8272FDC()
BEGIN
    // Reset the 8272 controller
    WritePort(0xC0903, 0x00)         // Reset control register
    Wait(10 milliseconds)
    WritePort(0xC0903, 0x0C)         // Enable controller

    // Wait for controller ready
    WAIT_FDC_READY:
    status = ReadPort(0xC0900)
    IF (status AND 0x80 == 0) THEN
        GOTO WAIT_FDC_READY
    EndIF

    // Configure drive parameters
    WritePort(0xC0901, 0x03)         // Specify command
    WritePort(0xC0902, 0xCF)         // Step rate=3ms, head unload=240ms
    WritePort(0xC0902, 0x02)         // Head load=4ms, DMA mode

    // Recalibrate drive 0
    WritePort(0xC0901, 0x07)         // Recalibrate command
    WritePort(0xC0902, 0x00)         // Drive 0

    // Wait for recalibration complete
    WAIT_RECALIBRATE:
    status = ReadPort(0xC0900)
    IF (status AND 0x20 == 0) THEN
        GOTO WAIT_RECALIBRATE
    EndIF

    SHOW "8272 FDC initialized successfully"
    RETURN SUCCESS
END

FUNCTION ReadSector(drive, track, head, sector, buffer)
BEGIN
```

```
    // Setup DMA for read operation
    CallFunction ConfigureDMAForRead(buffer, 512)

    // Send read command to 8272
    WritePort(0xC0901, 0x46)         // Read data command
    WritePort(0xC0902, (head << 2) | drive)  // Head and drive
    WritePort(0xC0902, track)        // Track number
    WritePort(0xC0902, head)         // Head number
    WritePort(0xC0902, sector)       // Sector number
    WritePort(0xC0902, 0x02)         // 512 bytes per sector
    WritePort(0xC0902, sector)       // End of track
    WritePort(0xC0902, 0x1B)         // Gap length
    WritePort(0xC0902, 0xFF)         // Data length

    // Wait for operation completion
    WAIT_READ_COMPLETE:
    status = ReadPort(0xC0900)
    IF (status AND 0x10 == 0) THEN
        GOTO WAIT_READ_COMPLETE
    EndIF

    // Read result bytes
    result1 = ReadPort(0xC0902)      // ST0
    result2 = ReadPort(0xC0902)      // ST1
    result3 = ReadPort(0xC0902)      // ST2

    IF (result1 AND 0xC0 == 0x00) THEN
        RETURN SUCCESS
    ELSE
        RETURN ERROR
    EndIF
END

FUNCTION WriteSector(drive, track, head, sector, buffer)
BEGIN
    // Setup DMA for write operation
    CallFunction ConfigureDMAForWrite(buffer, 512)

    // Send write command to 8272
    WritePort(0xC0901, 0x45)         // Write data command
    WritePort(0xC0902, (head << 2) | drive)  // Head and drive
    WritePort(0xC0902, track)        // Track number
    WritePort(0xC0902, head)         // Head number
    WritePort(0xC0902, sector)       // Sector number
    WritePort(0xC0902, 0x02)         // 512 bytes per sector
    WritePort(0xC0902, sector)       // End of track
    WritePort(0xC0902, 0x1B)         // Gap length
    WritePort(0xC0902, 0xFF)         // Data length
```

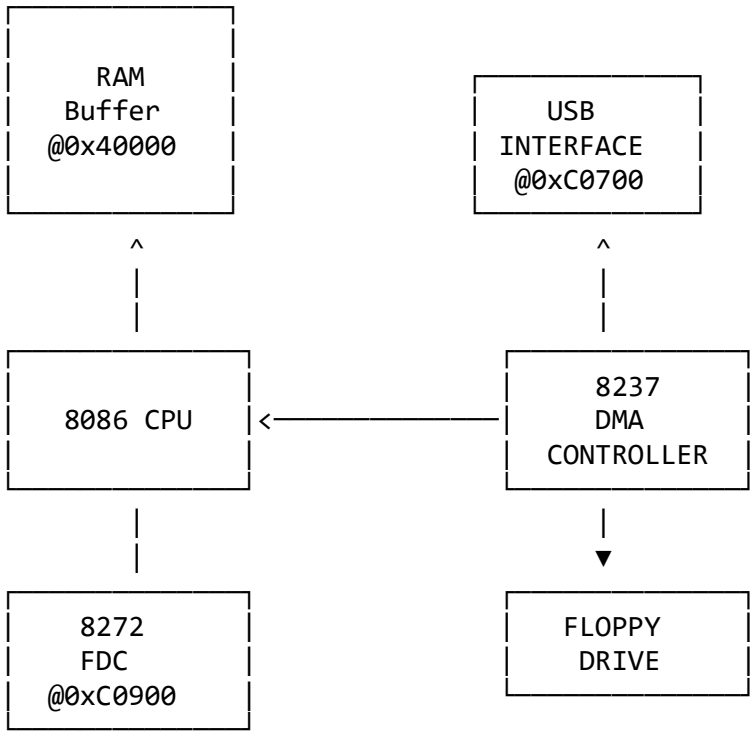
```
    // Wait for operation completion
    WAIT_WRITE_COMPLETE:
    status = ReadPort(0xC0900)
    IF (status AND 0x10 == 0) THEN
        GOTO WAIT_WRITE_COMPLETE
    EndIF

    // Read result bytes
    result1 = ReadPort(0xC0902)      // ST0
    result2 = ReadPort(0xC0902)      // ST1
    result3 = ReadPort(0xC0902)      // ST2

    IF (result1 AND 0xC0 == 0x00) THEN
        RETURN SUCCESS
    ELSE
        RETURN ERROR
    EndIF
END
```

2. 8237 DMA Controller for High-Speed Data

Components used:

- 8237 DMA Controller
- Multiple DMA channels for different operations
- Memory buffers for data transfer

Block Diagram:



Description:

8237 DMA Controller

- Provides high-speed data transfer between memory and peripherals.
- Supports 4 DMA channels (0-3) for concurrent operations.
- Reduces CPU overhead during data transfer operations.
- Handles memory-to-peripheral and peripheral-to-memory transfers.

Address Mapping:

Component	Address Range
DMA Controller	0x0000-0x000F
DMA Page Registers	0x0080-0x008F
DMA Buffer	0x40000-0x4FFFF

PSEUDOCODE

```
FUNCTION Initialize8237DMA()
BEGIN
    // Reset DMA controller
    WritePort(0x000D, 0x00)          // Master clear
    WritePort(0x000C, 0x00)          // Clear byte pointer

    // Configure DMA channels
    CallFunction ConfigureDMAChannel0()  // Reserved
    CallFunction ConfigureDMAChannel1()  // Floppy disk
    CallFunction ConfigureDMAChannel2()  // USB transfers
    CallFunction ConfigureDMAChannel3()  // System integration

    SHOW "8237 DMA Controller initialized"
    RETURN SUCCESS
END

FUNCTION ConfigureDMAForRead(buffer, size)
BEGIN
    // Configure DMA Channel 1 for floppy read
    WritePort(0x000A, 0x05)          // Mask channel 1
    WritePort(0x000C, 0x00)          // Clear byte pointer
    WritePort(0x000B, 0x46)          // Single mode, read, channel 1

    // Set buffer address
    address = buffer
    WritePort(0x0002, address AND 0xFF)      // Address low byte
    WritePort(0x0002, (address >> 8) AND 0xFF)  // Address high byte
    WritePort(0x0083, (address >> 16) AND 0xFF) // Page register

    // Set transfer count
    count = size - 1
    WritePort(0x0003, count AND 0xFF)        // Count low byte
```

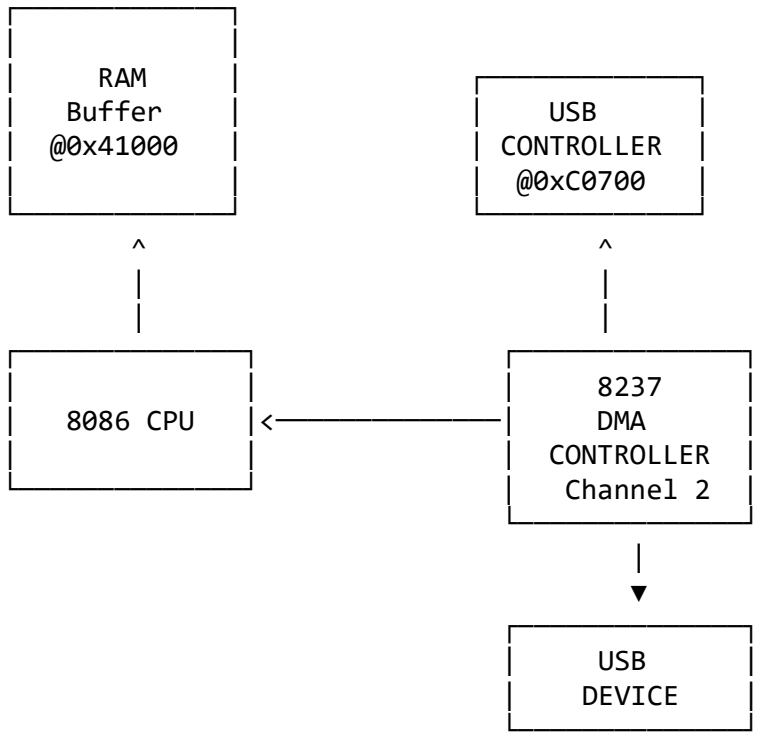
```
    WritePort(0x0003, (count >> 8) AND 0xFF) // Count high byte

    // Enable DMA channel
    WritePort(0x000A, 0x01)          // Unmask channel 1

    RETURN SUCCESS
END

FUNCTION ConfigureDMAForWrite(buffer, size)
BEGIN
    // Configure DMA Channel 1 for floppy write
    WritePort(0x000A, 0x05)          // Mask channel 1
    WritePort(0x000C, 0x00)          // Clear byte pointer
    WritePort(0x000B, 0x4A)          // Single mode, write, channel 1

    // Set buffer address
    address = buffer
    WritePort(0x0002, address AND 0xFF)      // Address low byte
    WritePort(0x0002, (address >> 8) AND 0xFF)  // Address high byte
    WritePort(0x0083, (address >> 16) AND 0xFF) // Page register

    // Set transfer count
    count = size - 1
    WritePort(0x0003, count AND 0xFF)        // Count low byte
    WritePort(0x0003, (count >> 8) AND 0xFF) // Count high byte

    // Enable DMA channel
    WritePort(0x000A, 0x01)          // Unmask channel 1

    RETURN SUCCESS
END
```

3. USB Transfers Using DMA

Components used:

- USB interface (external controller)
- 8237 DMA Controller
- Memory buffers for USB data

Block Diagram:



Description:

USB Interface via DMA

- Handles USB data transfers using DMA Channel 2.
- Supports bulk data transfer for high-speed USB operations.
- Manages USB protocol through external USB controller.
- Provides efficient data movement between USB devices and system memory.

Address Mapping:

Component	Address Range
USB Controller	0xC0700-0xC07FF
USB Buffer	0x41000-0x41FFF

PSEUDOCODE

```
FUNCTION InitializeUSBDMA()
BEGIN
    // Configure DMA Channel 2 for USB operations
    WritePort(0x000A, 0x06)          // Mask channel 2
    WritePort(0x000C, 0x00)          // Clear byte pointer

    // Set USB buffer address
    address = 0x41000
    WritePort(0x0004, address AND 0xFF)      // Address low byte
    WritePort(0x0004, (address >> 8) AND 0xFF)  // Address high byte
```

```
    WritePort(0x0081, (address >> 16) AND 0xFF) // Page register

    SHOW "USB DMA initialized"
    RETURN SUCCESS
END

FUNCTION USBTransferToMemory(size)
BEGIN
    // Configure DMA for USB read operation
    WritePort(0x000A, 0x06)          // Mask channel 2
    WritePort(0x000B, 0x42)          // Single mode, read, channel 2

    // Set transfer count
    count = size - 1
    WritePort(0x0005, count AND 0xFF)        // Count low byte
    WritePort(0x0005, (count >> 8) AND 0xFF) // Count high byte

    // Enable DMA channel
    WritePort(0x000A, 0x02)          // Unmask channel 2

    // Start USB transfer
    WritePort(0xC0700, 0x01)         // Start USB read

    // Wait for DMA completion
    WAIT_USB_DMA:
    status = ReadPort(0x0008)
    IF (status AND 0x04 == 0) THEN
        GOTO WAIT_USB_DMA
    EndIF

    RETURN SUCCESS
END

FUNCTION USBTransferFromMemory(size)
BEGIN
    // Configure DMA for USB write operation
    WritePort(0x000A, 0x06)          // Mask channel 2
    WritePort(0x000B, 0x46)          // Single mode, write, channel 2

    // Set transfer count
    count = size - 1
    WritePort(0x0005, count AND 0xFF)        // Count low byte
    WritePort(0x0005, (count >> 8) AND 0xFF) // Count high byte

    // Enable DMA channel
    WritePort(0x000A, 0x02)          // Unmask channel 2

    // Start USB transfer
    WritePort(0xC0700, 0x02)         // Start USB write
```

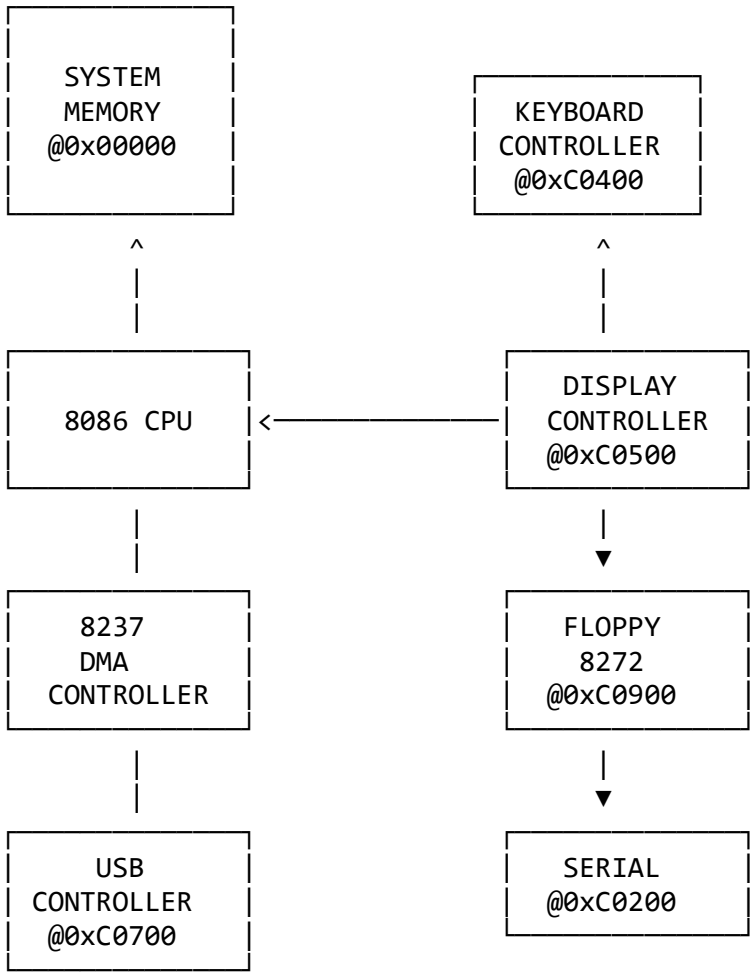
```
    // Wait for DMA completion
    WAIT_USB_DMA:
    status = ReadPort(0x0008)
    IF (status AND 0x04 == 0) THEN
        GOTO WAIT_USB_DMA
    EndIF

    RETURN SUCCESS
END
```

4. System Integration

Components used:

- All I/O controllers and interfaces
- Unified memory management
- Interrupt coordination

Block Diagram:



Description:

System Integration

- Coordinates all I/O operations through unified addressing.
- Manages data flow between different subsystems.
- Provides high-level functions for complex operations.
- Handles error conditions and system recovery.

PSEUDOCODE

```
FUNCTION IntegrateAllSystems()
BEGIN
    // Initialize all controllers
    CallFunction Initialize8272FDC()
    CallFunction Initialize8237DMA()
    CallFunction InitializeUSBDMA()

    // Configure system memory buffers
    CallFunction ConfigureSystemBuffers()

    // Setup interrupt handlers
    CallFunction SetupSystemInterrupts()

    SHOW "All systems integrated successfully"
    RETURN SUCCESS
END

FUNCTION ConfigureSystemBuffers()
BEGIN
    // Assign memory buffers for different operations
    // Floppy buffer: 0x40000-0x407FF (2KB)
    // USB buffer: 0x41000-0x417FF (2KB)
    // System buffer: 0x42000-0x427FF (2KB)

    // Clear all buffers
    CallFunction FillMemory(0x40000, 0x800, 0x00)   // Floppy buffer
    CallFunction FillMemory(0x41000, 0x800, 0x00)   // USB buffer
    CallFunction FillMemory(0x42000, 0x800, 0x00)   // System buffer

    RETURN SUCCESS
END

FUNCTION DataFlowOperation(source, destination, size)
BEGIN
    // High-level data flow between different systems

    IF (source == "FLOPPY" AND destination == "USB") THEN
        // Read from floppy to memory
        CallFunction ReadSector(0, 0, 0, 1, 0x40000)
```

```
        // Transfer from memory to USB
        CallFunction CopyBlock(0x40000, 0x41000, size)
        CallFunction USBTransferFromMemory(size)

    ELSE IF (source == "USB" AND destination == "FLOPPY") THEN
        // Read from USB to memory
        CallFunction USBTransferToMemory(size)

        // Transfer from memory to floppy
        CallFunction CopyBlock(0x41000, 0x40000, size)
        CallFunction WriteSector(0, 0, 0, 1, 0x40000)

    EndIF

    RETURN SUCCESS
END

FUNCTION SystemHealthCheck()
BEGIN
    // Check all system components
    floppyStatus = CallFunction CheckFloppyStatus()
    usbStatus = CallFunction CheckUSBStatus()
    dmaStatus = CallFunction CheckDMAStatus()

    IF (floppyStatus == SUCCESS AND usbStatus == SUCCESS AND dmaStatus ==
SUCCESS) THEN
        SHOW "All systems operational"
        RETURN SUCCESS
    ELSE
        SHOW "System component failure detected"
        RETURN ERROR
    EndIF
END
```

5. Integration Notes

Memory Buffer Assignment:

- Floppy Data Buffer: 0x40000-0x407FF (2KB)
- USB Transfer Buffer: 0x41000-0x417FF (2KB)
- System Integration Buffer: 0x42000-0x427FF (2KB)

DMA Channel Assignment:

- Channel 0: Reserved for system use
- Channel 1: Floppy disk operations (8272)
- Channel 2: USB transfers
- Channel 3: System integration and bulk transfers

Address Coordination:

- Floppy Controller: 0xC0900-0xC09FF
- USB Controller: 0xC0700-0xC07FF
- All addresses coordinated with team assignments
- No conflicts with other subsystems

Interrupt Integration:

- Floppy operations use IRQ6 (managed by 8259A)
- USB operations use IRQ5 (managed by 8259A)
- DMA completion signals handled through polling and interrupts

Conclusion:

This document provides a complete implementation of storage and DMA integration systems for the 8086 microprocessor. The implementation focuses on efficient data transfer using DMA controllers, reliable floppy disk operations, and seamless USB integration. All components are designed to work together as part of the larger system architecture while maintaining compatibility with othe