# 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