Carleton
UNIVERSITY

Department of Electronics

ELEC 4601: Laboratory 1
Introduction to Microprocessors

Carleton University, Ottawa, Ontario, K1S 5B6, Canada

Introduction

This lab will introduce various concepts related to microprocessors:

- How to test and debug simple code for a given microprocessor—in this case, an AM386SX-40 (one of the early variations of the classic 80x86 processor)
- The idea of addressing and communicating with devices that are electrically connected to the address/data/control buses of the microprocessor using memory-mapped and port-mapped operations
- How to operate a logic analyzer that lets you capture and evaluate these microprocessor bus signals
- The idea that the execution of code, even a single instruction, results in the generation of electrical signals that you can use as inputs to your own digital circuitry

Hardware

The following hardware is used in this lab:

- HP 16702B logic analyzer
- AM386SX-40 single board computer (PC/104 SBC)

The single board computer (SBC) uses a PC/104 industrial interface bus (a derivative of the classic ISA bus) as a peripheral device bus. The PC/104 uses the Microsoft DOS operating system. DOS has network functionality to allow you to connect to a more sophisticated Windows-based network for file transfer and, more importantly, it allows you to have direct access to all the hardware attached to the SBC without having to work with setting up access permission from the operating system. This is a standard embedded microprocessor development setup.

Logic analyzer

Logic analyzers are used to capture and store the states of a large number of time related signals and then examine the relationship of these signals to each other. They are like digital cameras for digital signals with a variable flash rate (sampling rate) and a fixed memory card size (finite memory) that can be commanded to take a series of snapshots when certain conditions occur (trigger conditions). For example, you can trigger a snapshot of the signals when two of them have a particular logical state or when a particular hexadecimal value appears on the address or data bus. Note that if you take a lot of very fast snapshots, you are limited by memory to a small time-window to look at. If you take slow snapshots, you have a bigger time window for examination, but you may miss fast-changing signals. Note that you will not actually have to change the sampling rate in these exercises.

PC/104

PC/104 is an industrial standard that defines how the bus signals from a microprocessor are brought out to particular pins of a particular connector (with a very well-defined size and shape). If you follow this standard, you can create boards and devices that will connect to any microcomputer that follows the standard. For the purposes of this lab, you need only consider it as a convenient way to bring the signals from the microprocessor out to place where the logic analyzer can connect to them. All hardware connections have already been made for you—***take a look at the pods connected at the back of the PC/104 running to the logic analyzer.***

Part 1: Setting up the logic analyzer

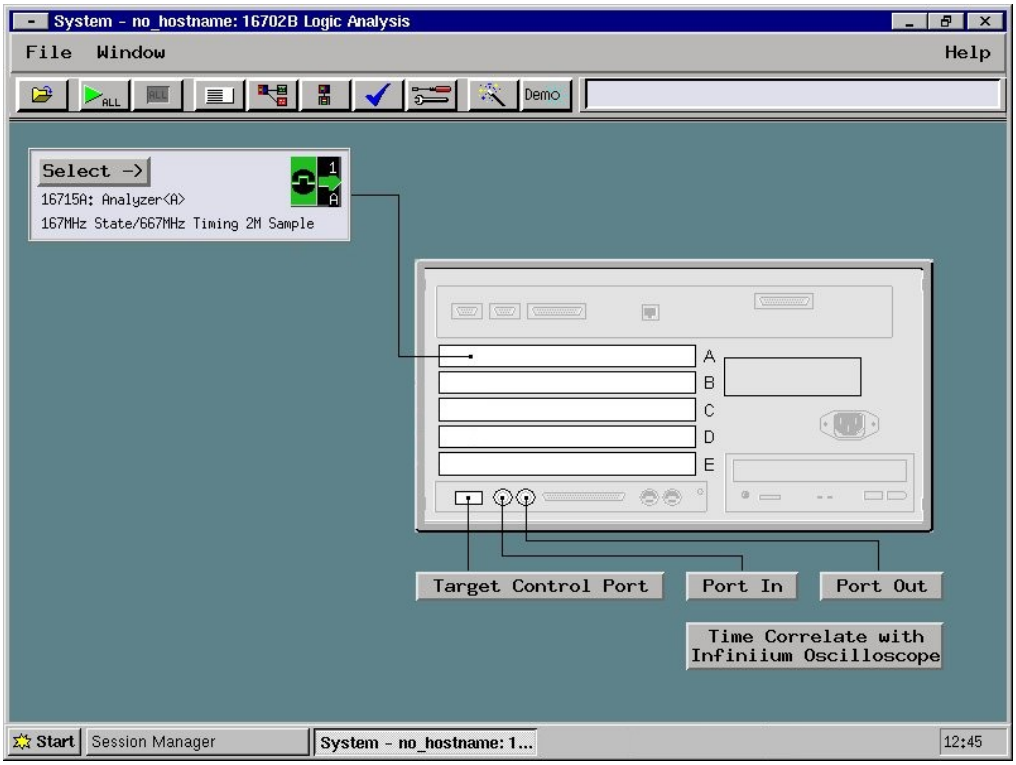

Figure 1 – System configuration (home screen)

Procedure

1) To make navigation easier, remotely connect to the logic analyzer
   a) Open **Exceed** on the Windows machine and click **Passive**
   b) Open a **Firefox** window. In the address bar, type in the name of your logic analyzer. The name of your analyzer is on the top right: "LAXX.VLSI1" where "XX" is some number between 01 and 14
   c) Click on **X-Window Web Control**
   d) Click on **Join Session**
2) Click the **Select ->** icon in the **167XXA: Analyzer<A>** box. Note that the last 'A' of this title may also be a 'D/E/something-else,' and just reflects the location of the interface board plugged into the back of the logic analyzer. Next, select the **Setup and Trigger…** option, usually at the top of the dropdown list

   a) Under the **Sampling** tab, ensure that **Type of the Analyzer** is set to **Timing Mode**

   b) Under **Format**, select **Pod Assignment**. Move (drag) all of the Pods to **Analyzer 1**. If you look at the back of the PC/104, you will see 3 pods connected to the logic analyzer

At this point, your **Pod Assignment** setup should appear as shown in Figure 2. Your setup may not yet show the up–down arrows; do not worry.

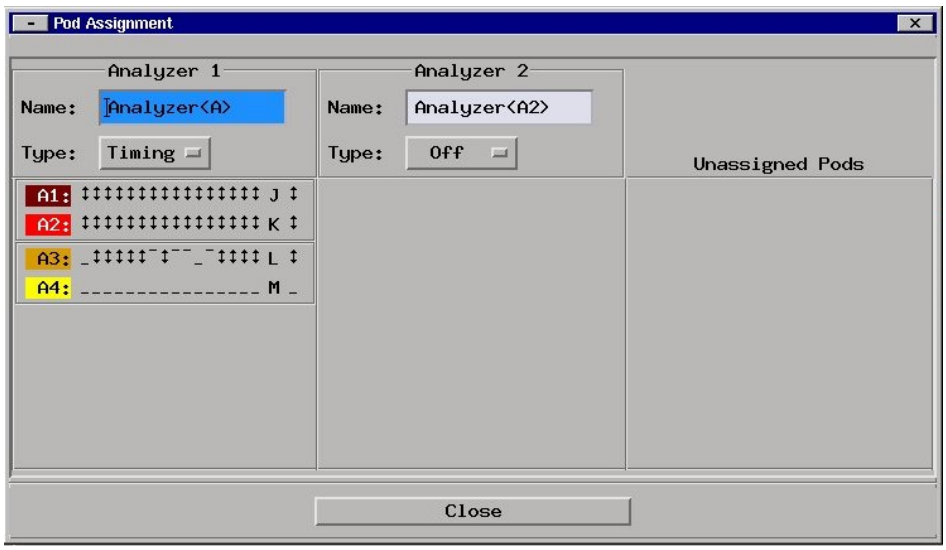

Figure 2 – Pod assignments

3) Select **Close** to return to **Format**. The **Format** window is used to select the pods and pins to be used, and how they are labeled. Please refer to Table 1 for the appropriate pin assignments

   a) Setting up the pods and pins:
      i) Familiarize yourself with what is on the screen of the logic analyzer. You will notice labels running down the left of the screen (see Figure 3)
      ii) Click on the first label in the list (usually "Label1" or something similar) and select **Rename**. Rename it to "DATA"
      iii) Use your cursor to scroll through Pods 1–3 and the CLK Inputs (horizontal scroll is located at the bottom right of your screen). *The pods shown correspond to the ones selected in the **Pod Assignment** view. Note that, depending on the location of the logic analyzer module, the first letter of the pod name might also be D, E, or just something other than A*

   b) Finish the pin assignments, as shown in Table 1
      i) In the **Format** view, under each pod, is a listing **15…8 7…0**. These correspond to the pins of each pod
      ii) For example, looking at Table 1, you will see that for Pod 1, all pins (0 through 15) are assigned to **DATA**. An asterisk **'*'** denotes that this pin is assigned to the label
      iii) Along **DATA**, click on the corresponding Pod 1 segment, where you can set all pins to **'*'**. In the pop-up window, selecting **'*********'** will quickly set all pins
      iv) Click on an existing label (e.g., **DATA**) for options on adding new labels
      v) Do this for all of the other bus lines in Table 1 and have a TA verify this

Table 1 – Pin Assignments for the Logic Analyzer

Pin #	CLK Inputs	Pod 3	Pod 2	Pod 1
	Label Name	Label Name	Label Name	Label Name
GND	—	GND	GND	GND
0	—	ADDR16	ADDR0	DATA0
1	—	ADDR17	ADDR1	DATA1
2	—	ADDR18	ADDR2	DATA2
3	—	ADDR19	ADDR3	DATA3
4	—	IRQ4	ADDR4	DATA4
5	—	IRQ5	ADDR5	DATA5
6	—	IRQ3	ADDR6	DATA6
7	—	IRQ7	ADDR7	DATA7
8	—	MEMR	ADDR8	DATA8
9	—	MEMW	ADDR9	DATA9
10	—	BALE	ADDR10	DATA10
11	—	REFRESH	ADDR11	DATA11
12	—	AEN	ADDR12	DATA12
13	—	OSC	ADDR13	DATA13
14	—	SYSCLK	ADDR14	DATA14
15	—	—	ADDR15	DATA15
CLK Input J	IOR	—	—	—
CLK Input K	IOW	—	—	—
	Note: CLK inputs are at the end (left) of the scrollable list of pods			

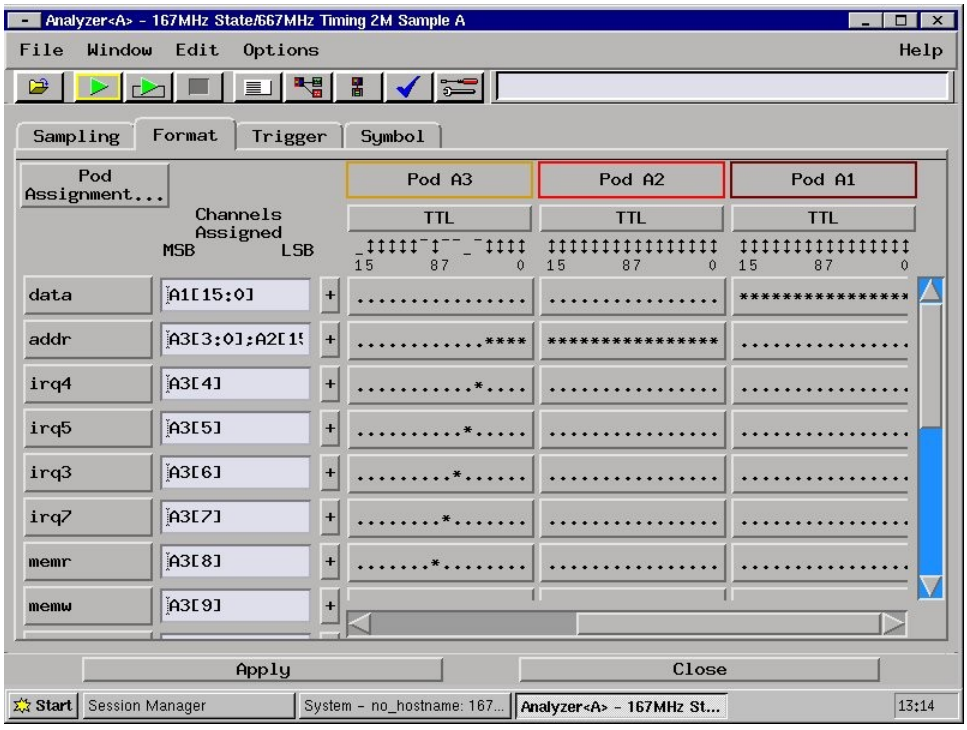

Figure 3 – Example pin assignments screen

Note the following about Figure 3:

- The labels (which can refer to a single signal on a single pod, or a cluster of signals across several pods) are shown in the leftmost column
- The array of buttons with the dots **'.'** and asterisks **'*'** are used to assign one or more pins of the selected pod to a particular label

At this point, you have defined the electrical connections. Signals of interest on the microprocessor bus are connected to actual inputs on the logic analyzer and are marked with labels that make analyzing the waveforms easier for us.

Important (run a sample program)

In this lab, we are learning how to use a logic analyzer to view the digital signals/patterns generated by a microprocessor. Therefore, we must generate some sample signals before we tell the logic analyzer what to look for. Follow the steps in Appendix A to run a simple program that prints "Hello World, Welcome to ELEC4601 !!!" at the top of the video monitor. In the next steps, we will program the logic analyzer to take a snapshot of the signals we assigned in the previous section when it finds the first few letters of the printed message.

Continue setting up the logic analyzer (triggering)

Since the logic analyzer does not have infinite memory, you must precisely define when you want to take a snapshot of the signals you are interested in. This process involves setting up the logical conditions that 'trigger' the first snapshot. Snapshots will then be taken at a speed that you define until memory is full. The trick is to define the speed of the snapshots so that they occur fast enough to see what you want over a fixed period of time. Too fast, and you may not see how the signals are related; too slow, and you may not see the fast signals that you are interested in. In this exercise, you can leave the default sampling rate.

4) Select the **Trigger** tab

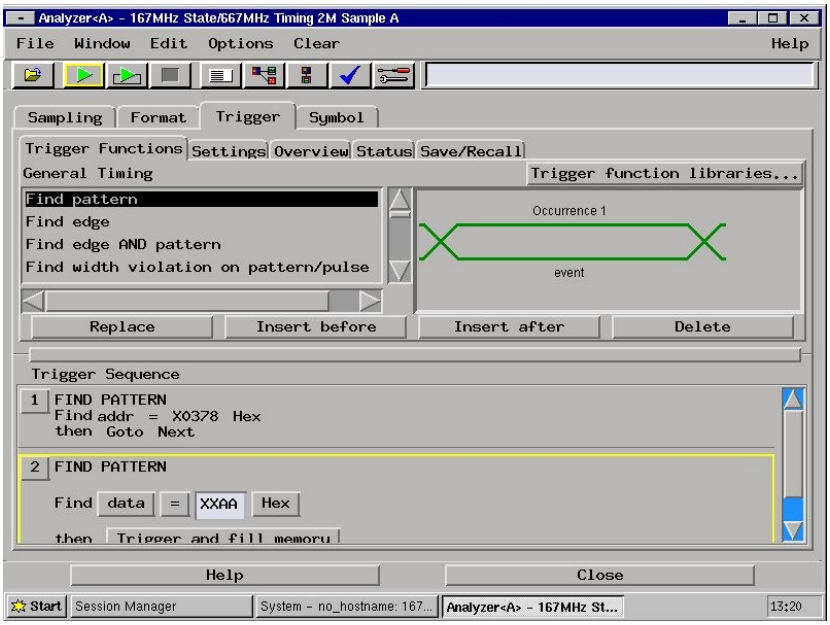

Figure 4 – Example trigger setup (DO NOT USE THIS SEQUENCE)

5) We are going to trigger when the logic analyzer finds the first four letters in sequence. Under the **General Timing** list, select **Find pattern for > duration** and then press **Replace**. Proceed to enter the following sequence of logical conditions. ***Note:*** *to add another level to the sequence, press* ***Insert after****. To add another condition to each level, press the current label (e.g.,* ***DATA*** *or* ***ADDR****) and select* ***Add label after...***

   1 Set **DATA** to **XX48** {H}, **ADDR** to **B8XXX**, for **> 6 ns**. Then **Goto Next**
      - Note that the **6 ns** may be slightly different on your machine—do not worry
   2 Set **DATA** to **XX65** {e}, **ADDR** to **B8XXX**, for **> 6 ns**. Then **Goto Next**
   3 Set **DATA** to **XX6C** {l}, **ADDR** to **B8XXX**, for **> 6 ns**. Then **Goto Next**
   4 Set **DATA** to **XX6C** {l}, **ADDR** to **B8XXX**, for **> 6 ns**. Then **Trigger and fill memory**

   Click the RUN icon, then execute the LAB1A.EXE program again.

   To view the waveform, select **Window** followed by **Analyzer<A>, Waveform<1>…**

You can save the screen shown on the logic analyzer to a .jpeg file. Select **Get Image** on the Welcome Page of your remote connection. Save the image to your H: drive (never save to your desktop! Ever!).

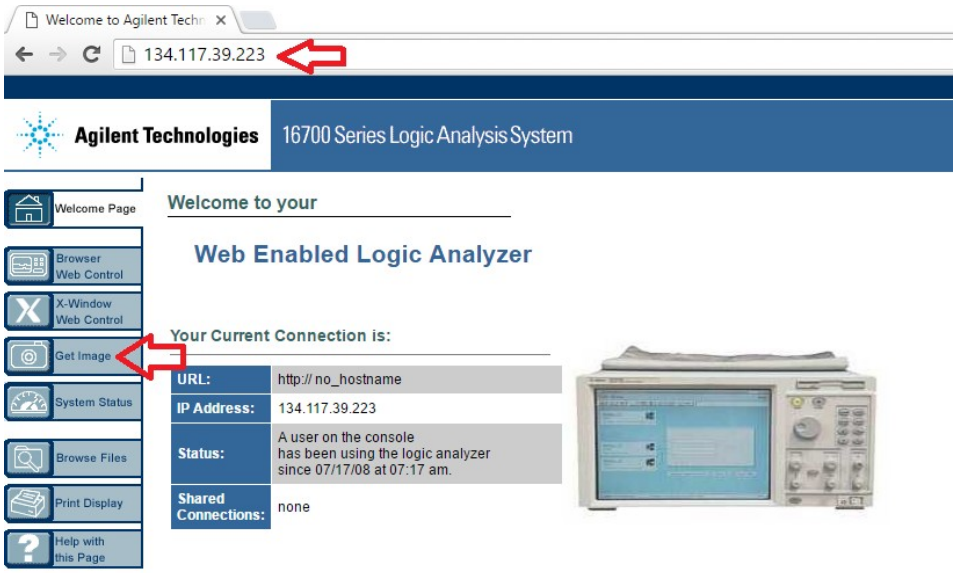

Figure 5 – Remote screen capture

Deliverables (make sure to have these before leaving the lab)

1) Take a screenshot of the first four WRITE cycles on the logic analyzer
2) Find the frequency of SYSCLK
3) Find the time taken for one complete memory-mapped I/O cycle to video memory. Think about this one for a bit. Review your lecture notes and the x86 instruction cycle (fetch–decode–execute). Brainstorm with your partner

Note: you must include screenshots of your waveforms with timing markers indicating the beginning and ending times of your measurements. Under the **Markers** tab, make sure to show the time difference between the two markers (G1 and G2).

Part 2: Investigating port-mapped I/O

The other technique used to talk with peripheral devices on the 80x86 buses is referred to as port-mapped I/O. This involves talking to a device using 'IN' and 'OUT' opcode instructions. Memory-mapped I/O can use any instruction that involves memory operations (there are lots); port-mapped I/O is restricted to a smaller and less-powerful instruction set.

Memory-mapped I/O uses the full 20-bit address space of the processor and produces **MEMR** and **MEMW** strobe signals. Port-mapped I/O uses a smaller 16-bit address space and produces **IOR** and **IOW** strobe signals.

For this demonstration, we are going to use the DOS Debug command to create a simple program to flash some LEDs that are attached to the output pins of what is referred to as the 'printer' or 'parallel' port (the input/output port at address 0x0378).

Procedure

1) From the PC/104 computer, run the Debug utility by typing **debug** at the DOS prompt. From there, type:

```
o 378 55        (To set the state of the LEDs to 0x55)
o 378 AA        (To set the state of the LEDs to 0xAA)
i 378           (To read the state of the LEDs)
```

2) Capture individual waveforms for each of the three operations above (***note:*** *you will have to change the trigger levels from what they were in Part 1*)

3) Use the logic analyzer to measure the time that it takes to produce one complete I/O cycle for each of the three examples in 1). Capture the 3 timing diagrams for your report

4) You can also do port-mapped READS and WRITES by using the simple DEBUG assembler and instruction trace features. For a port-mapped READ using assembly, type:

```
a 100
mov dx, 378
in al, dx       (note: this does the same as "i 378" in 1). i.e., read from port 378)
```

5) Trace the little program above and note exactly when the logic analyzer triggers:
   a) Press **Enter** to exit the assembler
   b) Type **r ip** to reposition the instruction pointer
   c) Type **100** to bring the pointer to the first line of our newly written code (i.e., "mov dx, 378")

6) Press **t** and **Enter** multiple times to step through the program until you get to the last instruction. Be **CAREFUL**, if you step past the last instruction, there is a chance that your machine will lock up. If you wish to restart the program, follow b) and c) of 5)

7) Use the logic analyzer to measure the time that it takes to produce the port-mapped READ cycle

8) For a port-mapped WRITE using assembly, type:

```
a 100
mov dx, 378
mov al, 55
out dx, al
```

(***note:*** *this does the same as "o 378 55" in 1). i.e., write 55 to port 378)*

9) Use the logic analyzer to measure the time that it takes to produce the port-mapped WRITE cycle

Deliverables

1) Have at least two screenshots (one READ, one WRITE) with timing markers for both methods (DEBUG commands and assembly instructions)
2) Note your observations on the LEDs, the PC/104, and the logic analyzer

Part 3: Investigating video memory

The CGA text mode video memory starts at B800:0000 (20-bit address of 0xB8000). This memory holds all the text characters that you see on the monitor attached to the PC/104 computer.

Address B800:0000 references the character in the upper left corner of the screen. Each line on the display is 80 characters wide. There are 25 lines on the display. Each character is coded with one word—one byte of the word is the actual ASCII character code, and the other byte of the code indicates the foreground and background colors of the character (search DOS color codes). This gives you a memory block of 80x2x25 = 4,000 bytes in size.

Procedure

1) Using DEBUG and some simple assembly code, you now have a relatively easy way to access any character on the screen. Type:

```
a 100
mov ax, B800
mov es, ax
mov di, 0
mov ax, 4235
es:
mov [di + 640], ax
```

2) Step through the program, like in 5) and 6) of Part 2. Be **CAREFUL**, if you step past the last line of code, there is a chance that your machine will lock up. If you wish to restart the program, follow b) and c) of 5) in Part 2.

Deliverables

1) Note the little change that you see on the screen. Take a picture of the screen for later reference
2) Be able to explain and comment each of the above lines of code

Bonus

With this basic code, you can now change any character on the screen, along with its attributes. Figure out how to change the color of every character on the screen so that it is WHITE on BLUE instead of LIGHTGREY on BLACK, regardless of what the character itself is.

Lab report

Please complete and submit the Google Form posted on cuLearn. The deadline is one week after your lab session.

Appendix A: Assembling and linking the Part 1 executable

Procedure

1) Download the **CODE.zip** file from cuLearn
2) Extract **CODE.zip** into your H:\ drive
3) Login to the PC/104 station with the username and password you were given at the beginning of the lab. A yes-or-no prompt will be given to you—just press the **Enter** key
4) Navigate to the **H:\CODE** folder on the PC/104 drive by typing:
   cd CODE (this gets us into the "H:\CODE" folder)
   dir (for your reference, this shows you the files in the current folder)
5) Close any open file explorer windows on the Windows machine
6) On the PC/104 computer, assemble the program by typing **ml LAB1A.ASM**. It produces "LAB1A.EXE"
7) You must type **cls** to clear the screen first, as a message will be written at the top of the screen's memory
8) Run the executable at the DOS prompt by typing **LAB1A.EXE**
9) Make sure you see the message at the top of the screen before going back to the lab procedure

- For your reference, the code that you are running is shown below

```
; *********** lab1a.asm *************
; Displays the string (including the spaces)
; at various locations about the screen
; - Code uses MASM 6.11 syntax
; - To assemble: ml lab1a.asm
; *********************************
.MODEL small
.STACK 100h
.386
.data
       msg DB "Hello World, Welcome to ELEC4601 !!!", 0
       nSize DW ($ - msg)
.code
_main PROC
       XOR SI, SI
       XOR DI, DI
       MOV DX, @data
       MOV DS, DX
       MOV CX, nSize
       MOV SI, OFFSET msg
       MOV DX, 0B800h
       MOV ES, DX
scanLoop:
       MOV AL, byte ptr [SI]
       MOV byte ptr ES: [DI], AL
       INC DI
       INC SI
       INC DI
       LOOP scanloop
terminate:
       MOV AX, 4C00h
INT 21h
_main ENDP
END _main
```