Midterm Exam #1 Answer Key

Name: __________________________
Student ID #: __________________

I have read and understand Washington State University’s policy on academic dishonesty and cheating.
Signed: _______________________

Problem 1) (10 Points)
Identify the operand addressing mode used in each of these instructions.

a) AND DX,AX _____________ REGISTER
b) JMP JMPTAB[BX] ___________ BASE+DISP
c) ADD DX,15 ___________ IMMEDIATE
d) CMP WORD PTR [BX+DI],10 ___________ BASE+INDEX
e) MOV IVAL[DI+4],CX ___________ INDEX+DISP

Problem 2) (10 Points)
What will be the value in AX after executing the following instructions? Give the answer in both hexadecimal and binary.

\[
\begin{aligned}
\text{mov al,15} & \quad \text{AX}=??0F \quad \text{CL}=?? \\
\text{mov ah,15} & \quad \text{AX}=0F0F \quad \text{CL}=?? \\
\text{xor al,al} & \quad \text{AX}=0F00 \quad \text{CL}=?? \\
\text{mov cl,3} & \quad \text{AX}=0F00 \quad \text{CL}=03 \\
\text{shr ax,cl} & \quad \text{AX}=01E0 \quad \text{CL}=03 \\
\text{add al,90h} & \quad \text{AX}=0170 \quad \text{CL}=03 \quad \text{CY}=1 \\
\text{adc ah,0} & \quad \text{AX}=0270 \quad \text{CL}=0 \\
\end{aligned}
\]

___0270h__________

___0000 0010 0111 0000
Problem 3)
Suppose you had a different processor that was designed and operated similarly to the 8086/8088 architecture with the following differences: All of the registers are 8-bit registers, and the physical address (PA) is a 10-bit number.

Question A (5 points) Given what you know about the 8086/8088 architecture, what would be the size of the total addressing space on this new device?

___1KB or 1,024 Bytes____

Question B (5 points) Given what you know about 8086/8088 addressing, what would be the size of the “offset window” at each segment location through which you could address memory?

___256 Bytes____

Problem 4) (10 Points)
What will be the value in AX after executing the following instructions? Assume that DS and ES are set up appropriately to access the variable ‘array’. Give the answer in hexadecimal:

Byte Offset

<table>
<thead>
<tr>
<th>Offset</th>
<th>1</th>
<th>0</th>
<th>3</th>
<th>2</th>
<th>5</th>
<th>4</th>
<th>7</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>array</td>
<td>dw 11 11h, 22 22h, 33 33h, 44 44h</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- mov bx, 1
- mov si, 6
- mov ax, array[bx][si-2]

BX + SI – 2 = 5 ___3322h____

mov ax, array[bx][di] ; move bytes 3 and 4 into AX, byte 3 into AL, byte 4 into AH
Problem 5)
Consider the following fragment of assembly code:

```
array    dw  7, 6, 5, 4
count    dw  4
.
.
.
xor     ax, ax
stc
mov     cx, count
mov     si, offset array
label1:  adc  ax, word ptr [si]
         add    si, 2
         loop   label1
label2:
```

Question A:

The body of the loop will execute 4 times (CX = 4). On each pass through the loop, AX will have the following values:

<table>
<thead>
<tr>
<th>AX</th>
<th>Array[SI]</th>
<th>CF</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>AX = 0</td>
<td>7</td>
<td>+ 1</td>
<td>= 8</td>
</tr>
<tr>
<td>AX = 8</td>
<td>6</td>
<td>+ 0</td>
<td>= 14</td>
</tr>
<tr>
<td>AX = 14</td>
<td>5</td>
<td>+ 0</td>
<td>= 19</td>
</tr>
<tr>
<td>AX = 19</td>
<td>4</td>
<td>+ 0</td>
<td>= 23 = 17h</td>
</tr>
</tbody>
</table>

a) (5 Points): What will be the value in AX when control reaches label2?

_____23 or 17h ________

Question B:

b) (5 Points) What is the purpose of the line:

```
xor     ax, ax
```

_It zeroes the AX register._

Question C:

c) (10 Points) Write an efficient and functionally equivalent code segment for the line:

```
loop     label1
```

```
DEC     CX
CMP     CX, 0
JNZ     label1
```
Problem 6) (25 Points)
Write an 8088/8086 assembly language subroutine which will count the number of times that a specified ASCII character occurs within a ZERO terminated string. The subroutine will be called with DS:SI pointing to the string to be searched, and AL containing the ASCII character to search for, and should return with the count in AX. The subroutine should make no assumptions about the state of any flags, and should return with all registers preserved (except AX, which contains the return value). The subroutine should be declared using the PROC directive, should be callable from outside the module where it is defined (i.e. PUBLIC), and should be callable from any segment (i.e. a FAR proc).

Note: A zero terminated ASCII string is a sequence of ASCII character codes with the end of the sequence indicated by a byte containing the value 0.

For example: If the subroutine were called with a pointer to the following string in DS:SI, and the value ‘m’ in AL, it would return 3 in AX:

```
str  db  "Programming in assembler is easy",0
```

Your subroutine will conform to this interface description:

```
;----------
; Count the number of times that the given character code
; occurs in the specified string
;
; Entry:  AL      - character code to count
;         DS:SI   - pointer to zero terminated string
; Exit:   AX      - count of times character occurs
; Uses:   AX modified, all else preserved

public    chrcnt

chrcnt    proc   far
  push  cx       ;save reg’s we use
  push  si
  xor   cx,cx    ;use CX as counter

chct10:   cmp    byte ptr ds:[si],0  ;end of string?
  jz     chct90  ;if so, then done
  cmp    al,ds:[si]  ;does this char match?
  jnz    chct20  ;if not don’t count it
  inc    cx      ;count this char

chct20:   inc    si      ;go to next char
  jmp    chct10  ;repeat until end of str

chct90:   mov    ax,cx  ;return value in AX
  pop     si      ;restore reg’s
  pop     cx
  ret

chrcnt    endp
```
Problem 7)
You are stepping through the execution of an 8088 assembly language program. The following information shows the state of the machine. Shown are memory dumps, a disassembled listing of the part of the program that is currently executing, and the current contents of the CPU registers.

Dump of Interrupt Vector Table:
0000:0000 BB 08 0B 02 65 04 70 00-16 05 DA 09 65 04 70 00 ....e.p......e.p.
0000:0010 65 04 70 00 D7 04 00 C0-85 98 00 F0 53 FF 00 F0 e.p............S...
0000:0020 00 00 00 C9 28 00 DA 05-3A 00 DA 05 52 00 DA 05 .......(.....R...
0000:0030 6A 00 DA 05 82 00 DA 05-9A 00 DA 05 65 04 70 00 j............e.p.

Dump of the Program’s Data Segment:
125A:0000 00 00 57 65 6C 63 6F 6D-65 20 74 6F 20 45 45 33 ..Welcome to EE31
125A:0010 31 34 20 45 78 61 6D 20-23 31 00 57 65 6C 63 6F 14 Exam #1.Welco
125A:0020 6D 65 20 74 6F 20 45 45-33 31 34 20 4D 69 64 74 14 me to EE314 Midt
125A:0030 65 72 6D 20 45 78 61 6D-20 23 31 00 18 00 00 00 erm Exam #1.....

Dump of the Program’s Stack Segment:
125E:0060 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
125E:0070 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................

Listing of the program code:
1266:0033 EB02          JMP     0037
1266:0035 46            INC     SI
1266:0036 47            INC     DI
1266:0037 803C00        CMP     BYTE PTR [SI],00
1266:003A 7505          JNZ     0041
1266:003C 803D00        CMP     BYTE PTR [DI],00
1266:003E 7612          JZ      0053
1266:0041 8A04          MOV     AL,[SI]
1266:0043 3A05          CMP     AL,[DI]
1266:0045 74EE          JZ      0035
1266:0047 7305          JNB     004E
1266:0049 B8FFFF        MOV     AX,FFFF
1266:004C EB07          JMP     0055
1266:004E B80100        MOV     AX,0001
1266:0051 EB02          JMP     0055
1266:0053 33C0          XOR     AX,AX
1266:0055 C3            RET

Current Contents of the CPU Registers:
AX=0065  BX=0000  CX=0018  DX=0000  SP=007E  BP=0000  SI=0003  DI=001C
DS=125A  ES=125A  SS=125E  CS=1266  IP=0043  FLAGS=0220

NMI Interrupt Service Routine:
NMIISR:  PUSH AX
         PUSH SI
         CALL HANDLENMI ;Process the NMI, doesn’t modify any
         ;registers or flags except AX and SI
         POP SI
         POP AX
         IRET
Problem 7 Questions)
a) (10 Points) The instruction shown in bold in the program listing is the current instruction being executed. While this instruction is executing, an NMI occurs. The NMI will be serviced before the next instruction begins executing. What is the address of the NMI interrupt service routine?

**Interrupt Vector Table:**

<table>
<thead>
<tr>
<th>Offset</th>
<th>0000</th>
<th>0000</th>
<th>BB</th>
<th>08</th>
<th>0B</th>
<th>02</th>
<th>65</th>
<th>04</th>
<th>70</th>
<th>00</th>
<th>16</th>
<th>05</th>
<th>DA</th>
<th>09</th>
<th>65</th>
<th>04</th>
<th>70</th>
<th>00</th>
</tr>
</thead>
</table>

The Interrupt Vector Table is an array of DWORD entries (each entry is 4 bytes). The NMI Interrupt uses **vector 2**. The offset of entry 2 in the Interrupt Vector Table is at: $2 \times 4 = 8$. This entry is made up of the bytes underlined above. Each entry in the table is a **SEG:OFF** pair giving the CS and IP values for the entry point of the interrupt service routine. Remembering the Intel byte ordering convention, the address of the NMI ISR is:

____09DA:0516_________

b) (15 Points) Show the contents of the program stack at the point in execution of the NMI interrupt service routine just before the call to HANDLENMI occurs. Use one row of the table for each byte of memory used by the stack.

When the NMI occurs, the values of FLAGS, CS, and IP are pushed onto the stack by the interrupt logic in the CPU. The NMI interrupt service routine then pushed AX and SI before calling HANDLENMI. So, just before the call the handle NMI, the stack had the following values on it:

001Dhalready there from before
0220hFLAGS
1266hCS
0045hIP address of next instruction
0065hAX
0003hSI

Remember that the stack grows down (i.e. SP is decremented on a push), and that SP always points to the top of the stack. Remember also, that each entry on the stack is a WORD (2 BYTES). At the time that the NMI occurred, the SP register contained 007Eh. So at memory locations 007Eh and 007Fh is the value 001D. Starting with that location and value, the table would be filled in as follows:
c) (10 Points) Following the instruction shown in bold is a conditional jump instruction (JZ). Given the current state of the registers and the contents of memory, determine the address of the next instruction to be executed after the conditional jump.

C) 1266:0035

Just before the conditional jump is a CMP AL,[DI] instruction. This will set the flags, which determines whether the jump will be taken or not. Using the value of DS:DI, which is 125A:001C to examine the Dump of the Program’s Data Segment, we find that the memory location contains 63h, which is the same value as is in AL. This means that the zero flag will be set by the compare instruction, and so the jump will be taken. Therefore, the next instruction to execute after the JZ instruction will be at 1255:0035