

Solutions to Exam Two

CS130 - Computer Organization and Assembly Language Drake University - Fall, 2004

Directions: Do six out of seven problems. Important: Cross out the problem that you don't want corrected. All problems have equal weight. Show all work. Please work first on problems with which you are more comfortable.

Problem 1 - Recursion. Here and on the next page is the code for Mergesort that we looked at (minus the comments and with line numbers added). Answer the questions that follow it.

```
( 1)          ; MSORT (MERGESORT)
( 2)          INCLUDE IRVINE16.INC
( 3)          .DATA
( 4)  ARRAY   WORD 13, 24, 32, 56, 47, 63, 44, 35, 26, 73, 32, 53, 28, 77, 23, 42
( 5)          N = ($-ARRAY)/2
( 6)  LOWHLFTOP WORD ?
( 7)  UPPHLFTOP WORD ?
( 8)          .CODE
( 9)  MAIN    PROC
(10)          MOV  AX, @DATA
(11)          MOV  DS, AX
(12)          MOV  CX, N
(13)          MOV  SI, OFFSET ARRAY
(14)  L1:     MOV  AX, [SI]
(15)          PUSH AX
(16)          ADD  SI, 2
(17)          LOOP L1
(18)          MOV  SI, SP
(19)          MOV  BX, N
(20)          CALL MSORT
(21)  L2:     POP  AX
(22)          CALL WRITEINT
(23)          CALL CRLF
(24)          LOOP L2
(25)          MOV  AX, 4CH
(26)          INT  21H
(27)  MAIN    ENDP
```

```

(28) MSORT PROC
(29) CMP BX, 1
(30) JG L1
(31) RET
(32) L1: MOV BP, SI
(33) MOV CX, BX
(34) L2: MOV AX, [BP]
(35) PUSH AX
(36) ADD BP, 2
(37) LOOP L2
(38) MOV AX, SP
(39) PUSH SI
(40) PUSH BX
(41) MOV SI, AX
(42) SHR BX, 1
(43) CALL MSORT
(44) POP BX
(45) POP SI
(46) MOV AX, SP
(47) ADD AX, BX
(48) PUSH SI
(49) PUSH BX
(50) MOV SI, AX
(51) SHR BX, 1
(52) CALL MSORT
(53) POP BX
(54) POP SI
(55) MOV CX, BX
(56) MOV DX, SP
(57) MOV DI, SP
(58) ADD DI, BX
(59) MOV AX, DI
(60) MOV LOWHLFTOP, AX
(61) ADD AX, BX
(62) MOV UPPHLFTOP, AX
(63) L3: CMP DX, LOWHLFTOP
(64) JE L4
(65) CMP DI, UPPHLFTOP
(66) JE L5
(67) MOV BP, DX
(68) MOV AX, [BP]
(69) MOV BP, DI
(70) CMP AX, [BP]
(71) JL L5
(72) L4: MOV BP, DI
(73) MOV AX, [BP]
(74) ADD DI, 2
(75) JMP L6
(76) L5: MOV BP, DX
(77) MOV AX, [BP]
(78) ADD DX, 2
(79) L6: MOV BP, SI
(80) MOV [BP], AX
(81) ADD SI, 2
(82) LOOP L3
(83) MOV AX, BX
(84) SHL AX, 1
(85) ADD AX, SP
(86) MOV SP, AX
(87) RET
(88) MSORT ENDP
(89) END MAIN

```

(a) What is the value of N and why?

Solution: 16. By the time the assembler gets to the point where N is defined, it will have already put 32 bytes of data into the data segment, and since ARRAY marks the start of the data segment, \$-ARRAY will equal 32.

(b) How many times will MSORT be called in total (during a single execution of the program)? Explain your answer.

Solution: Mergesorting 16 items involves twice mergesorting two 8-item lists, which involves mergesorting four 4-item lists, which involves mergesorting eight 2-item lists, which involves mergesorting sixteen 1-item lists. $16+8+4+2+1 = 31$ calls to MSORT in total.

(c) Draw a picture of the stack, showing the data stored there, and showing where the DX, DI and SI registers are pointing at the moment that the loop at line 63 begins (for the first invocation of the MSORT procedure).

Solution:

```
stack
-----
...
...
...
13
24
32
56
47
63
44
35
26
73
32
53
28
77
23
42 <----- SI
(ret.  addr.)
77
73
53
42
32
28
26
23 <----- DI
63
56
47
44
35
32
24
13 <----- DX
```

Problem 2 - Data movement, etcetera. Assuming

```
.data
list1 BYTE 1, -2, 3, -4, 5, -6
list3 DWORD -1, 2, -3, 4, -5, 6
```

Show what the contents of `eax` will be, *expressed in hexadecimal*, after executing each of the following.

Before looking at the solutions, we need to see byte-by-byte what is being stored in the data segment:

01 FE 03 FC 05 FA FF FF FF FF 02 00 00 00 FD FF FF FF 04 00 00 00 FB FF FF FF 06 00 00 00.

(a) `movzx eax, list1`

Solution: 00000001h

(b) `movsx eax, [list1 + 3]`

Solution: FFFFFFFFCh

(c) `mov eax, -3`
`mov ah, [list1 + 1]`

Solution: FFFFFFFDh

(d) `movzx eax, WORD PTR list3`

Solution: 0000FFFFh

(e) `movsx eax, WORD PTR [list3 + 8]`

Solution: FFFFFFFDh

(f) `mov eax, TYPE list3`

Solution: 00000006h

(g) `mov eax, LENGTHOF list3`

Solution: 00000004h

(h) `mov eax, SIZEOF list3`

Solution: 00000018h

Problem 3 - Indexed addressing. Consider the executing the following MASM program. Show in detail what happens along the way to the array.

```

include Irvine16.inc
.data
array WORD 10, 9, 8, 7, 6, 5, 4, 3, 2, 1
.code
main PROC
mov ax, @data
mov ds, ax
mov si, 0
mov di, 8
mov cx, 5
mov ax, 1
L: mov bx, array[si]
   mov array[di], bx
   mul bx
   add si, 2
   sub di, 2
   loop L
exit
main ENDP
END main

```

Solution:

ax	bx	cx	si	di	array
1		5	0	8	10 9 8 7 6 5 4 3 2 1
1	10	5	0	8	10 9 8 7 10 5 4 3 2 1
10	10	4	2	6	10 9 8 7 10 5 4 3 2 1
10	9	4	2	6	10 9 8 9 10 5 4 3 2 1
90	9	3	4	4	10 9 8 9 10 5 4 3 2 1
90	8	3	4	4	10 9 8 9 10 5 4 3 2 1
270	8	2	6	2	10 9 8 9 10 5 4 3 2 1
270	9	2	6	2	10 9 8 9 10 5 4 3 2 1
2430	9	1	8	0	10 9 8 9 10 5 4 3 2 1
19440	8	1	8	0	10 9 8 9 10 5 4 3 2 1
19440	8	0	10	-2	10 9 8 9 10 5 4 3 2 1

Problem 4 - Stack usage. By means of a table with columns labeled by registers, show precisely what the following code will do:

```

MOV    bx, 1
MOV    dx, 2
MOV    cx, 3
L: PUSH    bx
   PUSH    dx
   POP     bx
   POP     ax
   MUL    bx
   MOV    dx, ax
   INC    dx
   LOOP   L

```

Solution:

ax	bx	cx	dx	stack
	1	3	2
	1	3	2 1
	1	3	2 1 2
	2	3	2 1
1	2	3	2
2	2	2	3
2	2	2	3 2
2	2	2	3 2 3
2	3	2	3 2
2	3	2	3
6	3	1	7
6	3	1	7 3
6	3	1	7 3 7
6	7	1	7 3
3	7	1	7
21	7	0	22

Problem 5 - Conditional branching.

(a) Consider the pair of instructions

```
CMP ax, bx
JL skip
```

Explain carefully why the Intel folks designed the JL instruction so that it branches if and only if the SF and OF registers differ.

Solution: We want to branch iff ax-bx is *really* negative (in real-world signed sense). If OF = 0, then SF correctly contains the sign of the result after subtraction. So we should branch if OF=0 and SF=1. But if OF = 1, then SF has is wrong. So here we want to branch if SF = 0. So branch if OF=1 and SF=0. In general, branch when OF and SF are different.

(b) Consider the pair of instructions

```
CMP ax, bx
JB skip
```

Explain carefully why the Intel folks designed the JB instruction so that it branches if and only if the CF bit is set (i.e. = 1).

Solution: We want to branch iff ax-bx is negative in an unsigned sense, and hence cannot be expressed in an unsigned sense, and hence CF = 1.

Problem 6 - C-to-Masm translation. Write some MASM code that would correspond to the following C/Java code:

```
while ( m < n )
{
    if ( m%2 == 0 && n%2 != 0 ) // i.e.  if m is even and n is odd
        { m++; n--; }
    else
        { m--; n++; }
}
```

Solution:

```

loop: MOV AX, m
      CMP AX, n
      JGE done
      MOV AX, m
      AND AX, 1
      JNZ skip
      MOV BX, n
      AND BX, 1
      JZ skip
      INC m
      DEC n
      JMP loop
skip: DEC m
      INC n
      JMP loop
done: .....

```

Problem 7 - Using frames. Write a procedure that receives an arbitrary number of numbers on the stack, and also receive the number of numbers in the CX register. Before returning, it should wipe out these numbers (by adjusting the stack pointer), but should leave two other numbers on the stack: the maximum and minimum of the original numbers. The procedure should use the base pointer register to set up a frame with two local variables. It should use the local variables to compute the maximum and minimum, arranging for these to be the max and min of the numbers examined “so far” inside a loop. For full credit, avoid using directives like `LOCAL` and `USES` and explicitly indicating the parameter list. In other words, use a “bare bones” approach that does not rely on automatic code generation via special Masm features.

We'll do this one in class together.