1 ASC: A Simple Computer Exercises

Each question is worth 20 points.

1. For each of the following load functions, give the effective address of the memory location being addressed and the contents of the accumulator resulting from the command. Assume that the contents of memory and the index registers are as follows and that all indirect indexing mode addresses are preindexed. Also, the symbol Z has value C and Y has value 11, both in hexadecimal. Memory and register contents are also in hexadecimal.

Address	Contents
0	4
1	E
2	3
3	12
4	A
5	2
6	5
7	10
8	C
9	A
A	3
B	5
C	7
D	1
E	0
F	0
10	0
11	AA
12	1D
13	3
14	5
15	C1
16	D
17	8
18	2
19	F
1A	9
1B	E

Index Register	Contents
1	A
2	7
3	4

(a) `LDA Z`

Address	Accumulator
C	7

(b) `LDA* Z`

Address	Accumulator
7	10

(c) `LDA Y, 2`

Address	Accumulator
18	2

(d) `LDA* Z, 1`

Address	Accumulator
D	1

(e) `LDA 8, 1`

Address	Accumulator
12	1D

(f) `LDA* 17`

Address	Accumulator
8	C

(g) `LDA* 17, 3`

Address	Accumulator
9	A

2. Trace through the following ASC program with the following inputs and give the outputs of the program. Assume that the program is named prog.

```
        ORG  0
START   RWD
        STA  Z
        LDX  Z, 1
        RWD
        STA  Z
        LDX  Z, 2
        LDX  =2, 3
        LDA  X, 2
        STA  Z
        LDA  =5
LOOP    STA  Y
        ADD  X, 1
        WWD
        TIX  NEXT, 1
NEXT    TDX  LOOP, 2
        LDA  Y
        WWD
        LDA  Z
        WWD
        HLT
X       BSC  1A, 2, B, C1, 4, A, 3
Y       BSS  1
Z       BSS  1
        END  START
```

(a) `./prog 1 3`

Solution:

```
7
12
D3
12
C1
```

(b) `./prog 2 4`

Solution:

```
10
D1
D5
DF
D5
4
```

3. Construct the Symbol Table for the ASC program in the previous exercise. Then assemble the code in both binary and hexadecimal form. The ASC opcode table is below.

Mn.	Op.
HLT	0
LDA	1
STA	2
ADD	3

Mn.	Op.
TCA	4
BRU	5
BIP	6
BIN	7

Mn.	Op.
RWD	8
WWD	9
SHL	A
SHR	B

Mn.	Op.
LDX	C
STX	D
TIX	E
TDX	F

Solution:

Sym.	Add.
START	0
LOOP	A
NEXT	E
X	14
Y	1B
Z	1C
=2	1D
=5	1E

START	RWD		1000 0 0 00 0000 0000	8000
	STA	Z	0010 0 0 00 0001 1100	201C
	LDX	Z, 1	1100 0 0 01 0001 1100	C11C
	RWD		1000 0 0 00 0000 0000	8000
	STA	Z	0010 0 0 00 0001 1100	201C
	LDX	Z, 2	1100 0 0 10 0001 1100	C21C
	LDX	=2, 3	1100 0 0 11 0001 1101	C31D
	LDA	X, 2	0001 0 0 10 0001 0100	1214
	STA	Z	0010 0 0 00 0001 1100	201C
	LDA	=5	0001 0 0 00 0001 1110	101E
LOOP	STA	Y	0010 0 0 00 0001 1011	201B
	ADD	X, 1	0011 0 0 01 0001 0100	3114
	WWD		1001 0 0 00 0000 0000	9000
	TIX	NEXT, 1	1110 0 0 01 0000 1110	E10E
NEXT	TDX	LOOP, 2	1111 0 0 10 0000 1010	F20A
	LDA	Y	0001 0 0 00 0001 1011	101B
	WWD		1001 0 0 00 0000 0000	9000
	LDA	Z	0001 0 0 00 0001 1100	101C
	WWD		1001 0 0 00 0000 0000	9000
	HLT		0000 0 0 00 0000 0000	0000
X	BSC		0000 0000 0001 1010	001A
	BSC		0000 0000 0000 0010	0002
	BSC		0000 0000 0000 1011	000B
	BSC		0000 0000 1100 0001	00C1
	BSC		0000 0000 0000 0100	0004
	BSC		0000 0000 0000 1010	000A
	BSC		0000 0000 0000 0011	0003
Y	BSS	1	---	dddd
Z	BSS	1	---	dddd
=2			0000 0000 0000 0010	0002
=5			0000 0000 0000 0101	0005

2 NASM Coding

4. (*25 points*) Write the following program using the NASM assembly language. You may use the atoi, itoa, iprint, iprintLF, sprint, sprintLF, slen, and quit subroutines that we constructed in class, remember to add the `%include 'functions.asm'` at the beginning.

   Fibonacci Number Program: The Fibonacci sequence starts with two ones and then every number after that is the sum of the two previous entries in the sequence. So the sequence is: 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ... We define Fib(n) to be the n^{th} Fibonacci number, specifically, Fib(1) = 1, Fib(2) = 1, Fib(3) = 2, Fib(4) = 3, Fib(5) = 5, Fib(6) = 8, and so on. Write a program in NASM that computes the n^{th} Fibonacci number. Hard code the value of n into one of the registers so if the programmer would change that single value the new Fibonacci number would be calculated. This should be the only line that a programmer should have to change to get the altered result.

   Solution:

```
; Calculator (fibonacci)

%include        'functions.asm'

SECTION .text
global  _start

_start:
    mov     eax, 12 ; store n in eax for calculation.
    mov     ecx, eax    ; store in ecx for calculation.

    mov     eax, 1      ; Store answer for base cases
    cmp     ecx, 1  ; if ecx is 1 then halt
    je      finish  ; jump to finish if ecx is 1

    cmp     ecx, 2  ; if ecx is 2 then halt
    je      finish  ; jump to finish if ecx is 2

    mov     ebx, 1      ; Set fib(2) for calculation.
    mov     eax, 2      ; Set fib(3) for calculation.

continue:
    dec     ecx     ; decrement ecx
    cmp     ecx, 2  ; if ecx is 2 then halt
    je      finish  ; jump to finish if ecx is 0
    mov     edx, ebx    ; Store ebx in edx
    mov     ebx, eax    ; Store eax in ebx
    add     eax, edx    ; add edx on to eax
    jmp     continue    ; otherwise continue with the next number.

finish:
    call    iprintLF    ; call our integer printing with linefeed function
    call    quit
```

5. (*25 points*) Do one and only one of the following programs. Write the program using the NASM assembly language. You may use the atoi, itoa, iprint, iprintLF, sprint, sprintLF, slen, and quit subroutines that we constructed in class, remember to add the `%include 'functions.asm'` at the beginning.

   (a) Factorial Program: A simple loop based program to calculate the factorial of a number. Recall that $n! = n \cdot (n-1) \cdot (n-1) \cdots 2 \cdot 1$, if n is greater than 0, and we define $0! = 1$. Hard code the value of n into one of the registers so if the programmer would change that single value the new factorial value would be calculated. This should be the only line that a programmer should have to change to get the altered result.

Solution:

```
; Calculator (factorial)

%include        'functions.asm'

SECTION .data
msg1        db      '! =  '      ; a message string to correctly output result

SECTION .text
global  _start

_start:
    mov     eax, 1      ; move 1 into eax
    mov     ecx, 10     ; move n into ecx
    mov     ebx, ecx    ; Store n for printing later.

continue:
    mul     ecx         ; multiply eax by ecx
    dec     ecx     ; decrement ecx
    cmp     ecx, 0  ; if ecx is 0 then halt
    jz      finish  ; jump to finish if ecx is 0
    jmp     continue    ; otherwise continue with the next number.

finish:
    mov     ecx, eax    ; store the result of n! in ecx
    mov     eax, ebx    ; load n into eax for printing.
    call    iprint      ; call our integer printing with linefeed function
    mov     eax, msg1   ; move our message string into eax
    call    sprint      ; call our string print function
    mov     eax, ecx    ; move our remainder into eax
    call    iprintLF    ; call our integer printing with linefeed function

    call    quit
```

(b) Factoring Program: A brute-force factoring program that will find the first prime factor of a number n. Hard code the value of n (the number to be factored) into one of the registers so that if a programmer would change that single value the altered factorization would result. This should be the only line that a programmer should have to change to get the altered result.

Solution:

```
; Calculator (factor)

%include        'functions.asm'

SECTION .data
msg1        db      'Factor: '      ; a message string to correctly output result

SECTION .text
global  _start

_start:
    mov     eax, 7429   ; move our number into eax
    mov     ecx, eax    ; save the number in ecx
    mov     ebx, 2      ; move 2 into ebx
    mov     edx, 0  ; move 0 into edx

continue:
```

```
    div     ebx         ; divide eax by ebx
    cmp     edx, 0  ; check if the remaincder is 0
    jz      finish  ; if so jump to the end

    inc     ebx     ; increment ebx
    mov     eax, ecx    ; reset eax
    mov     edx, 0  ; clear out edx

    jmp     continue    ; jump to continue to test the next number

finish:
    mov     eax, msg1   ; move our message string into eax
    call    sprint      ; call our string print function
    mov     eax, ebx    ; move our remainder into eax
    call    iprintLF    ; call our integer printing with linefeed function

    call    quit
```

(c) Greatest Common Divisor Program: Using the Euclidean Algorithm one can find the Greatest Common Divisor of two numbers, without factoring them. The algorithm is fairly simple, first write $a = b \cdot q_1 + r_1$ where q_1 and r_1 are the quotient and remainder, respectively, of $\frac{a}{b}$. Then shift b to the other side of the equation, shift the first remainder r_1 to the position of b and repeat the process. Then shift and repeat, continue until we get a remainder of 0. When that happens, our answer is the previous remainder.

$$
\begin{aligned}
a &= b \cdot q_1 + r_1 \\
b &= r_1 \cdot q_2 + r_2 \\
r_1 &= r_2 \cdot q_3 + r_3 \\
r_2 &= r_3 \cdot q_4 + r_4 \\
&\vdots \\
r_t &= r_{t+1} \cdot q_{t+2} + 0
\end{aligned}
$$

So $\gcd(a, b) = r_{t+1}$.

Hard code the values of a and b (the numbers to be gcded) into two of the registers. If a programmer would change those numbers, reassemble and link the program, then the result would be the GCD of the altered numbers.

Solution:

```
; Calculator (gcd)

%include        'functions.asm'

SECTION .data
msg1        db      'gcd = '      ; a message string to correctly output result

SECTION .text
global  _start

_start:

    mov     eax, 391    ; move our first number into eax
    mov     ebx, 850    ; move our second number into ebx

continue:
    div     ebx         ; divide eax by ebx
```

```
    cmp     edx, 0      ; check if the remainder is 0, if so GCD was last remainder
    je      finish      ; jump to the end and print result

    mov     eax, ebx    ; move the last divisor (ebx) to eax
    mov     ebx, edx    ; move the remainder to the divisor (ebx)
    mov     edx, 0      ; set the memory contents of edx to 0

    jmp     continue    ; go to the next step in the Euclidean Algorithm

finish:
    mov     eax, msg1   ; move our message string into eax
    call    sprint      ; call our string print function
    mov     eax, ebx    ; move our last remainder into eax
    call    iprintLF    ; call our integer printing with linefeed function

    call    quit
```