90

You must show how you get your answer for a full credit in each problem. The final answer only will not get a full credit even if correct.

1. Represent the following **decimal** integers in the signed 2's complement binary number system which uses **9 bits** for each number. And then express them in **12 bits** through sign-extension.
   - 140
   - -200

decimal	binary (9-bit)	binary (12-bit)
140	010001100	000010001100
-200	100111000	111100111000

140
-128
12

256 128 64 32 16 8 4 2 1
0 1 0 0 0 1 1 0 0
0 1 1 0 0 1 0 0 0

200
-128
72
-64
8

100110111
+1
100111000

2's Complement

2. (a) Derive the range of effective address **for each of V1, V2 and V3.**
   (b) For each of the 7 highlighted instructions, derive the effective address (in hexadecimal) of the memory operand (source or destination) involved. All the bytes of the memory variables are initialized to $\boxed{\$05}$.

```
      ORG   $500
V1    DS.B  5
V2    DS.W  5
V3    DS.B  40

      ORG   $8000
      movb  #$AB,V1
      ldaa  $502
      staa  V2
      ldx   #V2
      staa  9,x
      staa  2,-x
      ldy   #$507
      ldx   #V3
      sty   2,x+
      ldy   #$50F
      ldab  #2
      sty   b,x
      ldd   #2
      ldaa  [d,x]
```

5 — 500, 501, 502, 503, 504

x = $505

x = $503 before

after x = 511

2, x

x = $50F

5

40
-32
8

$28 = 40

50F
+28
537

Variable	range of effective address	instruction	effective address
V1	$500 – $504	**ldaa** $502	$502
V2	$505 – $50E	**staa** V2	$505
V3	$50F – $536	**staa** 9,x	$50E
		staa 2,-x	$503
		sty 2,x+	$50F
		sty b,x	$51B -2
		ldaa [d,x]	$50F

3. For each of the arithmetic instructions, derive the result of arithmetic operation and determine if each flag is set ("1") or cleared ("0"). The instructions are executed in the order they appear.

```
ldaa   #$30
adda   #$60
ldab   #$80
addb   #$70
ldaa   #$A0
suba   #$30
ldab   #$90
subb   #$E0
```

8 4 2 1

1

A0

$70

90

0 1 0 1 0000

1 0 1 1 0000

2's comp

instruction	result of operation	Z	N	C	V
adda #$60	a = $90	0	1	0	1
addb #$70	b = $F0	0	1	0	1
suba #$30	a = $70	0	0	0	[illegible]
subb #$E0	a = $B0	0	1	[illegible]	0

-2

10 A
11 B
12 C
13 D

4. The stack pointer sp has been initialized to $1FFF before the program below is executed. Draw the top of stack right after each of the stack instructions, showing the top byte(s) in the stack and the contents of **sp**. Also, show the address of each byte in the stack.

```
ldaa   #$10
psha
ldaa   #$20
psha
pulb
```

address	byte
$1FFD	$20
$1FFE	$10

SP_{after} = $1FFE

-4

SP - 1 then a → address SP

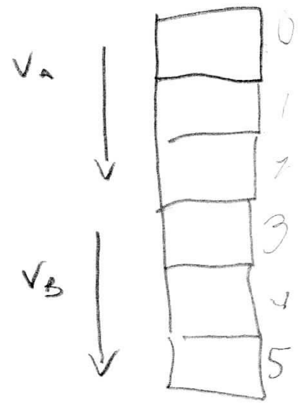
1FFE

5. The array `VA` has been loaded with three 1-byte numbers. **Using only the load and store instructions, and the constant offset indexing**, write a program which copies the three numbers in `VA` into the respective locations of the array `VB`. Minimize the number of instructions.

```
VA    DS.B  3
VB    DS.B  3
```

x = 0

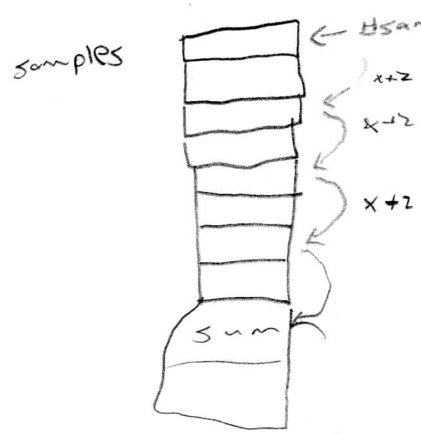
```
ldx  #VA
ldaa 0,X
staa 3,X
ldaa 1,X
staa 4,X
ldaa 2,X
staa 5,X
```



6. Assume that the array **samples** has been loaded with 4 numbers, one word (**not byte**) each. Write a program which adds the 4 numbers and stores the total sum in the variable **sum**. You must use the **auto-post-increment indexing** (for accessing any memory location). Minimize the number of instructions.

```
samples   DS.W  4
sum       DS.W  1
```

```
ldx  #samples
ldy  2,x+
addy 2,x+
addy 2,x+
addy 2,x+
sty  2,x+
```



85

In this exam., the microcontroller HCS12 is assumed unless specified otherwise. You must show how you get your answer for a full credit in each problem. **The final answer only will not get a full credit even if correct.**

1. The array `SAMPLE` consists of 100 bytes. Using a loop, write a program which initializes all elements (bytes) of the array to \$FF. Minimize the number of instructions. Do not assume a specific address for `SAMPLE`.

```
SAMPLE      DS.B    100
```

```
        ldx   #SAMPLE
        ldaa  #100
again:  movb  #$FF,0,x
        inc   X
        dbne  a, again
        rts
```

2. The variable `VAR1` is loaded with a 1-byte number. Write a program which shifts `VAR1` to the left 1-bit position if `DIRECTION` = 1, shifts it to the right 1-bit position if `DIRECTION` = 2, or does nothing otherwise (`DIRECTION` is neither 1 nor 2). Minimize the number of instructions.

```
DIRECTION DS.B   1
VAR1      DS.B   1
```

```
        ldaa  DIRECTION
        cmpa  #1
        beq   left
        cmpa  #2
        beq   right
        bra   end
left:   lsl   VAR1
        bra   end
right:  lsr   VAR1
end:    rts
```

3. The variables NUM1 and NUM2 contain two **unsigned** numbers which are passed to the subroutine `find_smaller` through the stack. Write the subroutine `find_smaller` which gets the two numbers from the stack, and returns the ~~larger~~ smaller of the two to the main program through the stack. The subroutine should not access the variables NUM1, NUM2 and SMALLER and must be consistent with the main program below. Minimize the number of instructions.

```
NUM1      DS.B  1
NUM2      DS.B  1
SMALLER   DS.B  1

...
ldaa      NUM1
psha
ldaa      NUM2
psha
bsr       find_smaller
pula
staa      SMALLER
...
```

Num1
Num2

r.a. -2

```
find_smaller:  pula          ; Num 2
               pulb          ; Num 1
               cba           ; compare b to a
               blo   one                          -1
               psh a
               bra   end
one:           psh ba
end:           rts
```

4. *Polling:* Suppose that the 1-bit control signal `READY` from an output device is raised to 1 only when it is ready to take another byte of data and reset to 0 once a byte is received by the output device. `READY` is connected to the MSB of Port B and the data byte is sent to the output device through Port A. Write a program which outputs the two bytes in the variable `OUTDATA` to the output device such that no byte is lost. Minimize the number of instructions.

```
PORTA     EQU   $0000
DDRA      EQU   $0002
PORTB     EQU   $0001

OUTDATA   DS.B  2
```

0
1000 0000
8 4 2 1

```
        movb  #$FF  DDRA
        ldx   #OUTDATA
poll:   brclr $80, PORTB, poll     -1
        movb  X, PORTA
        inx
        bra   poll      infinite loop
```

-2

5. *Interrupt-driven IO:* Whenever an input device gets a new byte to be input, it interrupts the CPU generating an IRQ interrupt. Upon the IRQ interrupt, the CPU reads in the new byte from the input device through Port A (to which the input device is connected). Assume that the IRQ interrupt is automatically cleared once the interrupt is recognized. Write a program which inputs two bytes from the input device, one byte at a time, and store them in `INDATA`, utilizing the interrupt mechanism and minimizing the power consumption by the CPU.

```
        PORTA  EQU  $0000
        INTCR  EQU  $001E

        INDATA DS.B 2

        lds         #$2000

        ldx    #INDATA
again:  movb #$40, INTCR
        cli
        wai
        bra  again

ISR:    ldab #2
        ldx  #INDATA
again:  movb PORTA, x
        inx
        dbne b, again
        rti

        ORG $FFF2
DC.W    ISR
```

-4

6. (a) What is saved in the stack during the interrupt sequence? List all (in any order).
(b) Why is the CCR saved during the interrupt sequence, but not in the subroutine call sequence?

a) AB, X, Y, CCR, PC

b) It is needed to return to the program with proper flags ?

-4