MIDTERM EXAMINATION

Part 1 – Short Answer Questions (2 points per question – total 20 points)

Answer questions 1 to 10 given that the state of the memory and CPU contents shown on page 9. The diagram represents the initial conditions for each question.

1) Give the machine code for the following assembler instruction: NEG $CC.

$70 $00CC (extended addressing is used since direct addressing is not available for the instruction.)

2) What has changed in memory after the execution of the following sequence of instructions?

```
LSL 0,X
LSL 0,X
LSL 0,X
LSL 0,X
```

The byte at address $0904 becomes $ 90.

3) Describe the changes made to memory by the PSHY.

The bytes at $090A are changed from $C161 to $4470.

4) What will be the content of accumulator D (hexadecimal value) after the execution of the following instructions?

```
VAR1 EQU $0902
     LDAA #16
     LDAB VAR1
```

D = …**$1010**……..

5) Will the branch in the following instructions execute (i.e. will the program branch)? Give the contents of Accumulator A after the execution of the CMPA instruction.

```
VAR3 EQU  $0900
   LDAA 0,SP
   CMPA VAR3
   BLT next
.
.
NEXT ldab $CC
```

Loads value $FC into A at address $090C
Compares to value $F5 at address $0900 and since the difference FC-F5 = 7 > 0
NO BRANCH

FC = -4 and F5 = -11, -4 > -11, i.e., FC > F5

Branch (Y or N) ___**N**_____ Contents of the A register: **$FC**

BLT treats numbers as signed numbers and thus $FC is greater than $F5.

6) Give the instruction that loads the value at address $47C1 using indirect-indexed addressing into accumulator D.

LDD [5,X]

or

LDD [-3,SP]

7) Identify the destination and the value of the data stored in the last instruction of the following sequence.

```
     PULD
     INCD
     STD -10,SP
```

Destination : **$0904 and $0905** It's independent of question 6!

Data value : **(16 bits) $FCDE**

8) What are the contents of index registers X and Y after the execution of the following instructions?

```
LDY 1,X+
LDX 0,Y
```

X = **$47C1** …Y= **$0909**

9) What is the content of accumulator A and the condition code bits NZVC after the execution of the following instructions.

PULA **Pulls off the stack the value $FC**

LDAB $0902

ADDA 0,SP **Adds the value pointed to by SP: $DD**

A = .**$D9** NZVC = **1001**

Addition FC N = 1, Bit 7 of result = 0, Z= 0, result is non-0

DD V=0, Addition of 2 negative numbers gives a negative result => NO overflow

D9 C=1, Addition of 2 large positive numbers, overflow

10) Using indexed addressing with auto-increment, complete the operands of the load and add instructions (LDAA and ADDA) to add the values $09, $09 and $85 found in memory. Note that the last instruction (STAA) should store the results in the byte following the value $85.

LDAA **1,X+**

ADDA **1,X+**

ADDA **1,X+**

STAA 0,X

A digital temperature sensor (DTS) provides an 8-bit code as a function of temperature in the range of [0˚C - 60˚C]. The DTS is connected to a microcontroller to build a digital thermometer. The codes that correspond to every integer value of temperature are stored in the microcontroller's memory as an array starting at the address tempTbl, as shown bellow:

Addr	CODE	TEMP ^{0}C	Addr	CODE	TEMP ^{0}C	Addr	CODE	TEMP ^{0}C
tempTbl	196	0	tempTbl+20	142	20	tempTbl+40	89	40
tempTbl+1	193	1	tempTbl+21	139	21	tempTbl+41	87	41
tempTbl+5	184	5	tempTbl+25	128	25	tempTbl+45	78	45
tempTbl+6	181	6	tempTbl+26	125	26	tempTbl+46	76	46
tempTbl+7	178	7	tempTbl+27	122	27	tempTbl+47	74	47
tempTbl+8	175	8	tempTbl+28	120	28	tempTbl+48	72	48
tempTbl+9	173	9	tempTbl+29	117	29	tempTbl+49	70	49
tempTbl+17	151	17	tempTbl+37	96	37	tempTbl+57	56	57
tempTbl+18	148	18	tempTbl+38	94	38	tempTbl+58	54	58
tempTbl+19	145	19	tempTbl+39	91	39	tempTbl+59	53	59

The temperature that corresponds to a code is equal to the index in the array. For example, the temperature corresponding to the code **178** is **7** degrees Celsius. Any temperature that has a code which is not included in this table has to be rounded up to the next temperature, to be represented by an integer value.

The temperature table is defined as follows. Note that the values in this table are shown as decimal values.

```
tblSize  EQU 60
tempTbl dc.b 196,193,191,188,186,184,181,178,175,173; 0 to 9 degrees
    dc.b 170,167,165,162,159,156,154,151,148,145      ; 10 to 19 degrees
    dc.b 142,139,136,134,131,128,125,122,120,117      ; 20 to 29 degrees
    dc.b 114,112,109,106,104,101,99,96,94,91    ; 30 to 39 degrees
    dc.b 89,87,85,82,80,78,76,74,72,70              ; 40 to 49 degrees
    dc.b 68,66,64,62,61,59,57,56,54,53               ; 50 to 59 degrees
```

Develop a subroutine, "TEMP ← getTemp(CODE)", that converts any 8-bit CODE to a temperature integer value (TEMP) as explained bellow:

- If the CODE is greater than the first element in the array, return the value 0 (TEMP=0).
- If the CODE is less than the last value in the table, return 60 (TEMP=60).
- If the CODE is between the value of the first element and the value of the last element, then the conversion result TEMP will be rounded up to the integer value of the next higher temperature. For example, for the CODE = **153**, a temperature of **17** degrees is returned (TEMP=17).

First provide a flowchart that illustrates the design of the subroutine. Give a few sentences that describe your overall design approach.

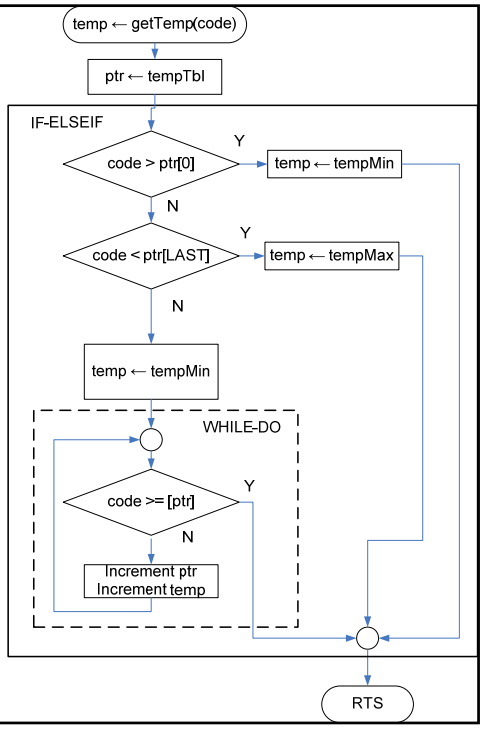
Then provide the code for the subroutine. Do not forget to comment your code.

Use the stack to receive CODE for translation from the calling routine and to return the temperature value (TEMP).

Design (flowchart and description):

The subroutine will take the following steps:

1. Compares the code value to the first element in the array, if greater than this first element, returns 0.
2. Compares the code value to the last element in the array, if less than this last element, return 60.
3. Scan the array searching for the index *i* where the code is greater than or equal to arr[i], return i.



```
;-------------------------------------------------------
; Subroutine: TEMP <- getTemp(CODE)
; Description: converts a CODE provided by a digital temperature sensor (DTS)  to
    TEMP, a Celsius Temperature.
; Arguments: CODE - on stack - put by the main program from DTS
; Returns: TEMP - on stack - The Celsius temperature corresponding to CODE
; Local variables: ptr - register X - pointer to the table's elements
; CODE - register A - value from DTS passed through the stack
;-------------------------------------------------------
; Stack Definitions
; 0,1 - Return Address; 2,3,4,5 – preserve X and D
off_TEMP EQU 6  ; subroutine will put here the conversion result TEMP,
    ; i.e., the temperature that corresponds to the CODE provided by DTS
off_CODE EQU 7  ; the main program puts here the CODE provided by DTS
          ; before calling the subroutine getTemp
tblSize   EQU 60
LAST EQU 59
tempMin EQU 0
tempMax EQU 60
tempTbl  dc.b 196,193,191,188,186,184,181,178,175,173 ; 0 a 9 degrees
    dc.b 170,167,165,162,159,156,154,151,148,145    ; 10 a 19 degrees
    dc.b 142,139,136,134,131,128,125,122,120,117    ; 20 a 29 degrees
    dc.b 114,112,109,106,104,101,99,96,94,91        ; 30 a 39 degrees
    dc.b 89,87,85,82,80,78,76,74,72,70              ; 40 a 49 degrees
    dc.b 68,66,64,62,61,59,57,56,54,53              ; 50 a 59 degrees
```

```
getTemp:
    pshx ; preserve registers
    pshd
    ldx #tempTbl
    ldaa off_CODE,SP ; load CODE in A
    cmpa 0,X        ; Compare CODE to tmpTBL[0]
    ble nextif1      ; in range
    movb #tempMin,off_TEMP,SP   ; TEMP < tempMIN => temp <- tempMin
    bra leave
nextif1 cmpa LAST,X         ; Compare CODE to tmpTBL[60]
    bge nextif2     ; it's in range
    movb #tempMax,off_TEMP,SP  ; TEMP <- tempMax since TEMP is > tempMax
    bra leave

nextif2
    movb #tempMin,off_TEMP,SP ; initially assume temp <- tempMin
loop    cmpa 1,X+  ; Compare CODE to tmpTBL[k], k=1..59
                   ; and ptr++ to check for the next tmpTBL value
    bge leave         ; found TEMP and return
    inc off_TEMP,SP
    bra loop
leave   puld  ; restore registers
    pulx
    rts
```