CEG 3136 – Computer Architecture II
Tutorial 2 – Basic Programming - Solution
Tutorial 2 – Translating C Code to Assembler Fall 2019

1) Complete the translation of the C function to Assembler.

```
/*-----------------------------
Function: addByteArray
Parameters: arrPt – pointer to array
            num – number of elements
                  to sum
Description: Adds the contents of
             an integer array.
Assumption: array contains at
            least one element.
-----------------------------*/
int addByteArray(byte *arrPt,
                 byte num)
{
   int sum;
   sum=0;
   do
   {
      sum=sum+*arrPt++;
      num--;
   }while(num != 0);
   return(sum);
}
```

```
;-------------Assembler Code-------------
; Subroutine – addByteArray
; Parameters – arrPt – register X
;              num – register Y
; Results: sum – register D
; Description: Adds contents of an integer
;              array.

addByteArray: PSHX
              PSHY ; preserve Registers
              LDD #0 ; sum=0;

                       ; do {
loop          ADDB 1,X+ ;     sum=sum+
              ADCA #0   ;         *arrPt++;
              DEY             num--;
              BNE loop  ; } while(num != 0);


              PULY ; restore registers
              PULX
              RTS  ; return(sum);
```

2) Translate the following short C program into assembler. Use the stack to exchange ALL parameters and the result (assume *int*'s take 2 bytes of storage, and *byte*'s take 1 byte of storage). Ensure that the registers used by the subroutine are not changed after the subroutine has executed. Define the stack usage with the OFFSET directive and labels as offsets into the stack.

```
/*------------------------------
Function: addInts
Description: Adds two 8-bit integers.
------------------------------*/
int addInts(byte val1, byte val2)
{
  int sum;
  sum = val1 + val2;
  return(sum);
}
```

```
;-------------Assembler Code-------------
; Subroutine – addInts
; Parameters – val1 – on stack
;              val2 – on stack
; Results sum – on stack
; Description: Adds two integers.
; Stack usage:
      OFFSET 0
         DS.W 1 ; preserve Register D
         DS.W 1 ; return address
ADI_SUM DS.W 1  ; sum and return value
ADI_VAL1 DS.B 1 ; byte val1
ADI_VAL2 DS.B 1 ; byte val2

addInts: PSHD ; preserve acc D
         CLRA
         LDAB ADI_VAL1,SP
         ADDB ADI_VAL2,SP
         ADCA #00
         STD ADI_SUM,SP
         PULD
         RTS
```

3) The C standard library provides a function to concantenate two strings:

```
char *strcat(char *str1, char *str2)
```

A *string* of characters terminated with a null character is stored in the memory starting at address found in the pointer variable *str1* and a second string at address found in the pointer variable *str2*. Develop a structured assembly subroutine that concantenates the contents pointed to by *str2* to the end of the contents in string *str1*. The function returns the address of string *str1* (that is the address received in the *str1* variable). Assume single byte ASCII characters.

1. First provide a C function that illustrates the algorithm of the subroutine (or functions/subroutines – you may use additional functions/subroutines to provide a solution).
2. Then translate the C functions to assembler code to subroutine(s). Do not forget to comment your code.

Algorithm (C program and description):

```
/*-------------------------------------------------------------
Function: strcat
Parameters: str1 - reference to string 1.
            str2 - reference to string 2.
Returns: Reference to string 1 (which was modified).
Description: Concatenates the string referenced by str2 to the end
             of the string referenced by str1.
-------------------------------------------------------------*/
char *strcat(char *str1, char *str2)
{
      char *pt;  // working pointer

      pt = findEnd(str1);  // find end of string, i.e. nul character
      while(*str2 != 0)
      {
         *pt++ = *str2++;
      }
      *pt = *str2;
      return(str1);
}
/*-------------------------------------------------------------
Function: findEnd
Parameters: str - reference to a string.
Returns: Reference to the end of the string (position of '\0').
Description: Finds the address of the end of the string, that is,
             the position of the null character.
-------------------------------------------------------------*/
char *findEnd(char *str)
{
      while(*str != 0) str++;
      return(str);
}
```

Assembler Source Code:

```
; Subroutine: char* strcat(char *str1, char *str2)
; Parameters
;       str1 - address of first string - on stack
;       str2 - address of second string to append to first string
;              on stack and reg Y
; Returns
;       str1 - pointer to concatenated string, str1 - on stack
; Local Variables
;       pt - register X
; Description: appends the second string to the first string.
;
; Stack Usage:
        OFFSET 0
SCAT_PRY  DS.W 1  // preserve Y
SCAT_PRX  DS.W 1  // preserve X
SCAT_RA   DS.W 1  // return address
SCAT_STR1 DS.W 1  // address of first string
SCAT_STR2 DS.W 1  // address of second string

strcat: pshx    ; preserve registers
        pshy
        ldx SCAT_STR1,SP ; get address of first string
        jsr $findEnd  ; x points to end, i.e. pt
        ldy SCAT_STR2,SP  ; y points to str2
scat_loop:
        tst 0,Y ; (*str2 == 0)
        beq scat_endwhile ; exit loop if true
        movb 1,Y+,1,X+  ; *pt++ = *str2++;
        bra scat_loop
scat_endwhile:
        movb 0,Y,0,X ; *pt = *str2 // nul char
        puly    ; restore registers
        pulx
        rts

; Subroutine: char *findEnd(str)
; Parameters
;       str - address to a string - x
; Returns
;       adr - points to end of string in x
; Description: Finds the end of the string (nul char) and returns its address.
;
; Stack usage:
FEND_PRX DS.W 1 ; preserve X
FEND_RA  DS.W 1 ; 0 return address

findEnd:
fend_loop:
        tst 0,x         ; (*pt == 0)
        beq fend_endwhile       ; yes exit loop

        inx             ; pt++
        bra fend_loop
fend_endwhile:
        rts             ; restore X
```

4) The following C functions were developed as part of the design for the Alarm System Simulator in Lab 1 to check validity of codes being entered by the user (1 digit at a time). Translate the functions to an assembler subroutines. The `isCodeValid` function accesses a global array that contains 4 integer (2 bytes) alarm codes. Note that the ASCII digit is translate to its numeric value.

```
// Declare alarm code array
#define NUMCODES 5
int alarmCodes[NUMCODES];

/*----------------------------
 * Functions: checkCode
 * Parameters: input - input character
 * Returns: TRUE - alarm code detected
 *          FALSE - alarm code not detected
 * Descriptions: Creates alarm code using digits entered until
 *               4 digits are seen. After 4th digit, see if
 *               alarm code is valid using isCodeValid().
 *----------------------------*/

byte checkCode(byte input)
{
   static int mult = 1000; // current multiplier of digit
   static int alarmCode = 0;   // alarm code value
   byte retval = FALSE;

   if(isdigit(input))
   {
      alarmCode = alarmCode + (input-ASCII_CONV_NUM)*mult;
      mult = mult/10;
      if(mult == 0)
      {
         retval = isCodeValid(alarmCode);
         alarmCode = 0;
         mult = 1000;
      }
   }
   else
   {
      alarmCode = 0;
      mult = 1000;
   }

   return(retval);
}
```

```
/*----------------------------
 * Functions: isCodeValid
 * Parameters: alarmCode - integer alarmCode
 * Returns: TRUE - alarm code valid
 *          FALSE - alarm code not valid
 * Descriptions: Checks to see if alarm code is in the
 *               alarmCodes array.
 *----------------------------*/
byte isCodeValid(int alarmCode)
{
   int *ptr;  // pointer to alarmCodes
   byte cnt = NUMCODES;
   byte retval = FALSE;
   ptr = alarmCodes;
   do
   {
      if(*ptr++ == alarmCode)
      {
         retval = TRUE;
         break;
      }
      cnt--;
   } while(cnt != 0);
   return(retval);
}
```

```
;-------------------------------------------------------
; Subroutine: checkCode
; Parameters: input - accumulator A
; Returns: TRUE when a valid alarm code is detected, FALSE otherwise - stored in
;          accumulator A
; Local Variables: retval - on stack
; Global Variables:
;       mult - initilased to 1000 in inithw (Alarm System Module)
;       alarmCode - initialised to 0 in inithw (Alarm System Module)
; Descriptions: Creates alarm code using digits entered until
;               4 digits are seen. After 4th digit, see if
;               alarm code is valid using isCodeValid().
;-------------------------------------------------------
; Stack usage
        OFFSET 0
CKC_INPUT  DS.B 1 ; parameter input
CKC_RETVAL DS.B 1 ; variable retval
CKC_VARSIZE
CKC_PR_B          DS.B 1 ; preserve B
CKC_PR_X   DS.W 1 ; preserve X
CKC_PR_Y   DS.W 1 ; preserve Y
CKC_RA    DS.W 1 ; return address

checkCode: pshy
   pshx
   pshb
   leas -CKC_VARSIZE,SP
                              ; static int mult = 1000; // current multiplier of
digit
                              ; static int alarmCode = 0;   // alarm code value
   movb #FALSE,CKC_RETVAL,SP  ; byte retval = FALSE;
   staa CKC_INPUT,SP          ; save paramater value

   jsr isdigit                ; if(isdigit(input))
   tsta
   beq ckc_else               ; {
   ldaa CKC_INPUT,SP          ;    alarmCode = alarmCode + (input-
ASCII_CONV_NUM)*mult
   suba #ASCII_CONV_NUM
   tab
   clra
   ldy mult
   emul                       ; //*mult - result in D
   addd alarmCode
   std alarmCode
   ldd mult                   ;    mult = mult/10;
   ldx #10
   idiv
   stx mult
   ldd mult                   ;    if(mult == 0)
   bne ckc_endif1             ;    {
   ldd alarmcode              ;
   bsr isCodeValid            ;       retval = isCodeValid(alarmCode);
   staa CKC_RETVAL,SP
   ldd #0                     ;       alarmCode = 0;
   std alarmCode
   ldd #1000                  ;       mult = 1000;
   std mult
```

```
ckc_endif1:                     ;     }
   bra ckc_endif                ; }
ckc_else:                       ; else {
   ldd #0                       ;       alarmCode = 0;
   std alarmCode
   ldd #1000                    ;       mult = 1000;
   std mult                     ; }
ckc_endif:

   ldaa CKC_RETVAL,SP           ; return(retval);
   ; Restore registers and stack
   leas CKC_VARSIZE,SP
   pulb
   pulx
   puly
   rts

;---------------------------------------------------
; Subroutine: isCodeValid
; Parameters: alarmCode stored in register D
; Local Variables
;   ptr - pointer to array - in register X
;   cnt, retval - on the stack.
; Returns: TRUE/FALSE - Returned in accumulator A
; Description: Checks to see if alarm code is in the
;              alarmCodes array.
;---------------------------------------------------

; Stack usage
        OFFSET 0
CDV_ALARMCODE   DS.W 1 ; alarmCode
CDV_CNT    DS.B 1 ; cnt
CDV_RETVAL DS.B 1 ; retval
CDV_VARSIZE:
CDV_PR_X   DS.W 1 ; preserve x register
CDV_RA     DS.W 1 ; return address

isCodeValid: pshx
   leas -CDV_VARSIZE,SP
   std CDV_ALARMCODE,SP
   ; int *ptr;  // pointer to alarmCodes
   movb #NUMCODES,CDV_CNT,SP ; byte cnt = 5;
   movb #FALSE,CDV_RETVAL,SP ; byte retval = FALSE;
   ldx #alarmCodes           ; ptr = alarmCodes;
cdv_while                    ; do
   ldd 2,X+                  ; {
   cpd alarmCode             ;    if(*ptr++ == alarmCode)
   bne cdv_endif             ;    {
   movb #TRUE,CDV_RETVAL,SP  ;      retval = TRUE;
   bra cdv_endwhile          ;      break;
cdv_endif:                   ;    }
   dec CDV_CNT,SP            ;   cnt--;
   bne cdv_while             ; } while(cnt != 0);
cdv_endwhile:
   ldaa CDV_RETVAL,SP         ;   return(retval);
   ; restore registers and stack
   leas CDV_VARSIZE,SP
   pulx
```

```
rts
```