Datorteknik
Laborativ Examination (LAX)
med Arduino

Uppdateras löpande. Se datumet:

7 december 2020

Innehåll

1 Inledning

Labserien i kursen Datorteknik avslutas med en Laborativ Examintion, LAX.

LAX-ens syfte är att visa att studenten självständigt kan lösa en hårdvarunära programmeringsuppgift med assembler. Den är också ett kvitto på att kursens lärandemål kan anses vara uppfyllda.

Laxens resultat bedöms efter **funktion**, inget annat. Det finns inga krav på varken kodens utseende eller storlek. Detta betyder att om du till exempel fastnar på en lösning med pekare, men ser tydligt hur en pekarlös lösning skulle utföras, gör då den pekarlösa. Som vanligt leder naturligtvis ett strukturerat angreppssätt snabbare mot målet.

Observera! All examination vid LiU är individuell. Även om du gjort de tidigare laborationsuppgifterna i två-grupp har själva inlämningen varit individuell. Och det är din prestation i din egna inlämning som bedömts.

I fallet med LAX är hela uppgiften individuell och skall utföras individuellt av en (1) student. Det är alltså i detta fall inte tillåtet med samarbete mellan studenter.

Kurshemsidans samtliga dokument är tillåtna hjälpmedel.

Vi förbehåller oss möjligheten att köra den inlämnade koden genom Urkund eller motsvarande för att identifiera möjligen cirkulerande lösningar.

2 Mjukvara

Till din hjälp vid laxtillfället har du hela din kodbas som du utvecklat under labserien. Du tjänar alltså på att vara väldigt bekant med din kod. Du tjänar också på att hela kodbasen är strukturerad och konstruerad så att rutinernas sidoeffekter på annan kod minimeras (dvs subrutiner, enbart lokala variabler, `jmp`/`br` inom rutiner och `call` mellan rutiner).

LAX-uppgiften kommer inte *kräva* avbrott, men kan möjligen göras enklare med avbrott. En avbrottsbaserad lösning accepteras men uppgifterna utformas inte med avbrott som centralt tema. Känner du sig bekväm med avbrott på processorn *får* du alltså använda det, men det kommer inte *behövas*.

LAX-uppgiften utförs med de funktioner (knappar, display, backlight) som återfinns på KPAD.

Följande rutiner från labserien kan användas. Dessa är dina *handtag* till din tidigare kod. Du ska inte behöva modifiera koden dessa rutiner byggs upp av. Du ska inte heller behöva leta i KPAD:s datablad för andra funktioner.

- `LCD_HOME:`, för att sätta cursorn längs till vänster (DDRAM-pekaren = 0)
- `LCD_ERASE:`, som `LCD_HOME:` men tömmer också displayen
- `LCD_COL:`, för att sätta skrivposition på displayen (kolumn 0–15)
- `LCD_ASCII:`, för att skriva ut ASCII-tecknet i `r16` på displayen
- `LINE_PRINT:`, för att skriva ut den NUL-terminerade strängen som `Z` pekar på (antingen SRAM eller FLASH)
- `KEY:` och `KEY_READ:`, för att returnera `0`–`5` beroende på nedtryckt knapp

Ovanstående rutiner har redan använts i labserien och bör fungera. För att säkerställa att dina rutiner fungerar på avsett sätt finns här nedan några förslag på testfall.

Testfall. Skriv kod som säkerställer att din `LINE_PRINT:` kan skriva ut strängen i `LINE`. Strängen ska kunna vara från tom sträng till full rad på sexton tecken.

Testfall. Skriv kod som säkerställer att din `KEY_READ:` inte kan returnera annat än `0`, `1`, `2`, `3`, `4` eller `5`.

Testfall. Skriv kod som säkerställer att din `LCD_COL:` kan skriva text var som helst på displayen övre rad.

3 Övningslaxar

Till dessa övningslaxar finns lösningar. Titta inte på dem innan du löst uppgiften själv. Tidsomfattning 90 minuter.

Tips: Ha som vana att rita en figur över vad som ska hända. Rita gärna också figurer över använda delar av SRAM osv. Det underlättar.

LAX1 Med knapparna UP och DOWN ska du kunna räkna upp, respektive ned, ett räknarvärde, ett steg för varje knapptryckning. Räknaren ska bara kunna ha värdena 0, 1, 2, 3, 4, 5, och om man försöker räkna upp från 5 så ska räknaren bli 0, samt om man försöker räkna ned från 0 så ska räknaren bli 5. Räknarens värde ska efter varje knapptryckning visas på LCD:n på första raden i kolumnen längst till vänster.

LAX2 Skriv ut en textsträng lagrad i programminnet, framlänges eller baklänges, med början längst från vänster (framlänges) respektive längst från höger (baklänges) på den första raden i LCD:n. Utskrift från vänster (framlänges) ska ske en gång vid varje knapptryckning på knappen LEFToch utskrift från höger (baklänges) ska ske en gång vid varje knapptryckning på knappen RIGHT.

T ex: Om textsträngen är '`Anders Nilsson`' ska knapptryckning på LEFT ge: '`Anders Nilsson` '. Och en knapptryckning på RIGHT: ' `nossliN srednA`'

LAX3 Med knapparna RIGHT och LEFT ska du flytta en asterisk '*' åt höger respektive vänster så länge knappen hålls nedtryckt. Förflyttningarna ska ske med ungefär en sekunds mellanrum. När man flyttar asterisken '*' utanför den mest högra kolumnen på raden så ska den dyka upp längst till vänster på raden och fortsätta att förflyttas så länge man håller knappen nedtryckt. När man flyttar asterisken utanför den mest vänstra kolumnen på raden ska den dyka upp längst till höger på raden och fortsätta förflyttas så länge man håller knappen nedtryckt.

LAX4 Knapparna SELECT, LEFT, DOWN, UP och RIGHT antas motsvaras av returvärdet 1, 2, 3, 4, respektive 5 från rutinen `KEY_READ`. För varje enskild knapptryckning på någon av knapparna ska returvärdet ackumuleras till en summa. Denna summa ska efter varje knapptryckning visas som ett decimaltal på LCD:ns två första kolumner på första raden. Om additionen av något returvärde resulterar i att summan blir större än 15 så ska summan nollställas.

T ex, om summan för tillfället är 13 och man trycker på UP (dvs returvärde 4) så blir ju summan egentligen 17, men då ska summan nollställas och `00` visas på LCD:n.

LAX5 Med knapparna RIGHT och LEFT ska du få en mask att krypa fram längs första raden på LCD:n. Masken ritas ut med asterisker '*' och är från början en asterisk lång och befinner sig längst till vänster på LCD:ns första rad. Vid varje knapptryckning på RIGHT växer masken åt höger med en asterisk. Vid varje knapptryckning på LEFT minskar masken från vänster med en asterisk. Masken får dock aldrig bli kortare än en asterisk. När maskens "huvud" kommit till den mest högra kolumnen på LCD:ns första rad så kan den inte växa mer och då kan bara "svansen" krympa tills masken endast är en enda asterisk som befinner sig längst till höger på LCD:ns första rad. Därefter måste programmet startas om med reset för att man ska kunna spela"igen.

LAX6 Du ska med knappen UP först räkna antalet knapptryckningar på UP och sedan när man trycker på knappen SELECT ska antalet knapptryckningar på UP visas i LCD:ns första kolumn på första raden. Dvs, antalet knapptryckningar på UP visas inte förrän man trycker på SELECT. Knappen UP ska som mest hålla reda på 7 tryckningar, dvs trycker man mer än 7 gånger så ska resultatet fortfarande bli 7. När man tryckt på SELECT så ska det därefter vara möjligt att på nytt räkna nya knapptryckningar (från 0 till 7) med knappen UP.

4 Lösningsförslag till övningslaxar

Koden är exempel på hur en lösning kan se ut. Din kod kan och får skilja från detta. För laxen räknas enbart funktionen. Vid laxtillfället skall din kod skickas in, men bedöms inte för examination.

Rutinerna

- `LCD_ASCII:`
- `LINE_PRINT:`
- `LCD_HOME:`
- `LCD_COL:`
- `LCD_ERASE:`
- `KEY:`
- `KEY_READ:`

har du redan med dig från labserien och redovisas inte nedan.

LAX1

```
    clr     r17         ; clear counter value

L01:
    rcall   LCD_HOME
    mov     r16,r17
    ori     r16,$30
    rcall   LCD_ASCII
    rcall   KEY_READ
    cpi     r16,4
    breq    L01_UP
    cpi     r16,3
    breq    L01_DOWN
    rjmp    L01

L01_UP:
    inc     r17
    cpi     r17,6
    brne    L01
    clr     r17
    rjmp    L01

L01_DOWN:
    dec     r17
    cpi     r17,$FF
    brne    L01
    ldi     r17,5
    rjmp    L01
```

LAX2

```
L02:
    rcall   LINE_INIT_TO_SPACE
    ldi     ZH,HIGH(L02_TEXT*2)
    ldi     ZL,LOW(L02_TEXT*2)
    rcall   KEY_READ
    cpi     r16,2
    breq    L02_LEFT
    cpi     r16,5
    breq    L02_RIGHT
    rjmp    L02

L02_LEFT:
    ldi     XH,HIGH(LINE)
    ldi     XL,LOW(LINE)
L02_LEFT_AGAIN:
    lpm     r16,Z+
    cpi     r16,0
    breq    L02_DISPLAY
    st      X+,r16
    rjmp    L02_LEFT_AGAIN

L02_RIGHT:
    ldi     XH,HIGH(LINE+16)
    ldi     XL,LOW(LINE+16)
L02_RIGHT_AGAIN:
    lpm     r16,Z+
    cpi     r16,0
    breq    L02_DISPLAY
    st      -X,r16
    rjmp    L02_RIGHT_AGAIN

L02_DISPLAY:
    rcall   LCD_HOME
    rcall   LINE_PRINT
    rjmp    L02

LINE_INIT_TO_SPACE:
    ldi     XH,HIGH(LINE)
    ldi     XL,LOW(LINE)
    ldi     r16,' '
    clr     r17
LITS_AGAIN:
    st      X+,r16
    inc     r17
    cpi     r17,16
    brne    LITS_AGAIN
    clr     r16
    st      X,r16
    ret

L02_TEXT:
    .db     "Anders Nilsson",0
```

LAX3

```
    clr     r17
L03:
    rcall   KEY
    cpi     r16,2
    breq    L03_LEFT
    cpi     r16,5
    breq    L03_RIGHT
    rjmp    L03

L03_LEFT:
    dec     r17
    cpi     r17,$FF
    brne    L03_DISPLAY
    ldi     r17,15
    rjmp    L03_DISPLAY

L03_RIGHT:
    inc     r17
    cpi     r17,16
    brne    L03_DISPLAY
    ldi     r17,0
    rjmp    L03_DISPLAY

L03_DISPLAY:
    rcall   LCD_ERASE
    mov     r16,r17
    rcall   LCD_COL
    ldi     r16,'*'
    rcall   LCD_ASCII
    rcall   WAIT1S
    rjmp    L03


    ; --- WAIT 1 s
WAIT1S:
    ldi     r18,61
WAIT1S_0:
    clr     r25
    clr     r24
WAIT1S_1:
    adiw    r24,1
    brne    WAIT1S_1
    dec     r18
    brne    WAIT1S_0
    ret
```

LAX4

```
    .dseg
NUM01:                          ; 1:th digit of sum
    .byte   1
NUM10:                          ; 10:th digit of sum
    .byte   1
    .cseg

    clr     r16
    sts     NUM01,r16
    sts     NUM10,r16

    rcall   DISP_SUM
L04:
    rcall   KEY_READ
L04_AGAIN:
    rcall   INC_SUM
    dec     r16
    brne    L04_AGAIN
    rcall   CHECK_SUM
    rcall   DISP_SUM
    rjmp    L04

INC_SUM:
    lds     r17,NUM01
    inc     r17
    sts     NUM01,r17
    cpi     r17,10
    brne    INC_SUM_EXIT
    clr     r17
    sts     NUM01,r17
    lds     r17,NUM10
    inc     r17
    sts     NUM10,r17
INC_SUM_EXIT:
    ret

CHECK_SUM:
    lds     r18,NUM10
    ldi     r17,10
    mul     r18,r17             ; Multiply r18 by 10
    mov     r18,r0
    lds     r17,NUM01           ; Now add NUM01 to result
    add     r18,r17
    cpi     r18,16              ; Compare with 16 to see if result > 15
    brlo    CHECK_SUM_EXIT
    clr     r17
    sts     NUM01,r17           ; Clear sum
    sts     NUM10,r17
CHECK_SUM_EXIT:
    ret

DISP_SUM:
    rcall   LCD_HOME
    lds     r16,NUM10
    ori     r16,$30
    rcall   LCD_ASCII
    lds     r16,NUM01
    ori     r16,$30
    rcall   LCD_ASCII
    ret
```

LAX5

```
    clr     r17     ; worm start pos
    clr     r18     ; worm end pos

L05:
    rcall   DISP_WORM
L05_KEY:
    rcall   KEY_READ
    cpi     r16,5
    breq    L05_RIGHT
    cpi     r16,2
    breq    L05_LEFT
    rjmp    L05_KEY

L05_RIGHT:
    inc     r18
    cpi     r18,16
    brne    L05
    ldi     r18,15
    rjmp    L05

L05_LEFT:
    inc     r17
    cp      r18,r17
    brge    L05
    mov     r17,r18
    rjmp    L05

DISP_WORM:
    rcall   LCD_HOME
    clr     r19
DISP_WORM_NEXT:
    cp      r19,r17
    brge    DISP_WORM1
    rjmp    DISP_WORM_SPACE
DISP_WORM1:
    cp      r18,r19
    brge    DISP_WORM_ASTER
    rjmp    DISP_WORM_EXIT
DISP_WORM_SPACE:
    ldi     r16,' '
    rjmp    DISP_WORM_COMMON
DISP_WORM_ASTER:
    ldi     r16,'*'
DISP_WORM_COMMON:
    rcall   LCD_ASCII
    inc     r19
    rjmp    DISP_WORM_NEXT
DISP_WORM_EXIT:
    ret
```

LAX6

```
    clr     r17 ; clear sum

L06:
    rcall   KEY_READ
    cpi     r16,4
    breq    L06_UP
    cpi     r16,1
    breq    L06_SELECT
    rjmp    L06

L06_UP:
    inc     r17
    cpi     r17,8
    brne    L06
    ldi     r17,7
    rjmp    L06

L06_SELECT:
    rcall   LCD_HOME
    mov     r16,r17
    ori     r16,$30
    rcall   LCD_ASCII
    clr     r17
    rjmp    L06
```