Datorteknik

— — —

Exempeluppgifter
Laborativ Examination

Michael Josefsson

Version 0.2

Om LAX i Datorteknik

Den laborativa examinationen (LAX) är en avslutande examinationspunkt i labserien. LAX genomförs enskilt och utan samarbete med andra studenter.

Vid LAX-tillfället är datorerna i labsalen i *examinationsläge* vilket innebär att nätåtkomst är avstängd. Inloggning sker lokalt på datorn med eget användarnamn och lösenord.

Hjälpmedel På LAX-en behöver du ta med dig penna. Kladdpapper finns i labsalen. På varje labplats finns dessutom en kopia av kursens Infoblad som delades ut på första föreläsningen. Linjal har du på LiU-kortet. Inget övrigt skall behövas.

Hårdvara Ingen hårdvara utöver den som använts i laborationsserien kommer användas. Tangentbordet och knappar är vanliga insignaler och lysdioder och displayer är utenheter. Högtalare kommer inte användas (tack och lov), inte heller oscilloskopet eller IR-mottagaren.

Alla laxar innehåller åtminstone följande moment:

- konfigurera stack
- sätta in- och utportar
- villkorliga hopp
- någon databehandling
- hårdvara från labbarna

Exempellaxar finns på hemsidan.

Ingen LAX *kräver* avbrott för sin lösning men en del kan bli *enklare* om avbrott används.

Det finns en uppsättning LAX-uppgifter och de slumpas till varje tillfälle.

Ett LAX-tillfälle är 90 minuter långt och gången vid laxtillfället är denna:

1. Bladet med LAX delas ut. Labtiden börjar.
2. När du har en lösning, påkalla assistentens uppmärksamhet.
3. Assistenten testar funktionen enligt LAX-bladet samt kontrollerar i koden att hårdvaruinitieringen utförts i en subrutin:
   a) Vid korrekt funktion och kod: Du antecknas med G i listan, är färdig, monterar ner uppkopplingen och lämnar lokalen.
   b) Vid felaktig funktion och/eller kod är du inte klar och får fortsätta med uppgiften. Se punkt 2 ovan.
4. Vid full tid avlsutar assistenten laxtillfället.

Inget facit till LAX-en kommer delas ut. Inte heller kommer lösningar att diskuteras varken under själva LAX-tillfället eller senare.

Lämna kvar LAX-bladet och Infobladet på labplatsen efter LAX-en. Eventuella anteckningar skall lämnas i papperskorgen innan du lämnar salen.

Detta dokument innehåller förslag på några representativa demonstrationslaxar. Uppgifterna är konstruerade så att de ska innehålla *sekvens*, *iteration* och *selektion* enligt JSP. Dessutom bör de komma ihåg någon tidigare händelse. De ”skarpa” LAX-arna är av samma komplexitetsgrad, har samma eller liknande hårdvara men är *inte* dessa uppgifter. Laxarna kräver inte avbrott men får användas.

Inlärningen sker i den kreativa processen i hjärnan när du själv konstruerar en lösning. Titta inte på lösningsförslagen om du inte har ett eget förslag att jämföra med!

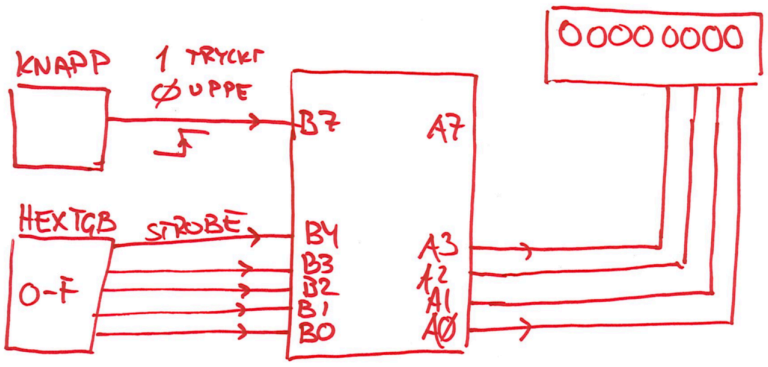
Namn	In-enhet	Ut-enhet
`LAX-DEMO 1`	Hextangentbord	2 x 7-segmentsdisplay
`LAX-DEMO 2`	2 x Tryckknapp	2 x 7-segmentsdisplay
`LAX-DEMO 4`	Hextangentbord	2 x 7-segmentsdisplay
`LAX-DEMO 5`	Hextangentbord	Lysdioddisplay

Öva så att du kan ta fram en fungerande lösning på 30–45 minuter.

Datorteknik LAX
Förslag på lösningsgång

Först och främst är det avgörande att **ha en plan!** Lös inte hela uppgiften på en gång, det är **alltid** fel approach, i all programmering. Man måste gå i **små steg**. Till exempel dessa steg (i en hypotetisk LAX med ett hextangentbord och lysdiodsrampen med 8 dioder där uppgiften är att skicka tangentbordsvärdet till dioderna ”rakt av” eller inverterat beroende på en ytterligare insignal från en tryckknapp):

1. Rita skiss/schema över uppkopplingen med alla väsentliga kopplingar!



2. Koppla in lysdioderna. Kanske `PortA` bit 7 – 0.
3. Konfigurera in- och utportar.

```
        ldi     r16,$FF
        out     DDRA,r16
        ldi     r16,$00
        out     DDRB,r16
```

4. **Kontrollera** att portarna är korrekt konfigurerade genom att tända och släcka dioderna. Kod för att testa dioderna kan vara något så enkelt som

```
        ldi     r16,$AA
        out     PORTA,r16
        ldi     r16,$55
        out     PORTA,r16
```

   som du kompilerar, och sen stegar igenom för att tända och släcka alla dioder. Nu vet du att hela kedjan fungerar (kompilering/nedladdning/stegning/portar/lysdioder). Hoppa inte över denna testning.

5. Koppla in tangentbordets fem bitar, `STROBE` samt `D3`–`D0` (typiskt på pinnarna 4–0). **Kontrollera** att DALIA-kortets lysdioder tänds som du förväntar dig när du trycker på olika tangenter dvs utan programmet inblandat. OK, tangentbordet funkar.
6. **Kontrollera** tangentbordets funktion genom att skriva programkod för att testa. För att veta om tangentbordet lästs in korrekt skickar du lämpligen ut inlästa värdet på dioderna:

```
TEST:
        in      r16,PINB
        out     PORTA,r16
        jmp     TEST
```

   Kompilera/stega osv. OK, du kan läsa in tangentbordets bitar.

   Om vi lånar begrepp från reglertekniken har du nu ett system som är **styrbart** (du vet att du kan läsa in värden) och **observerbart** (du vet att du kan skicka ut värden). Utan båda dessa kommer systemet aldrig att fungera.

7. Koppla in tryckknappen. Skriv ett program som läser av den biten (typiskt på pinne 7 på porten du redan använder som ingång, men det är inte slagit i sten) och gör något med displayen bara för att **kontrollera** att du har **styrbarhet** från tryckknappen:

```
KNAPP_KONTROLL:
        ldi     r16,$FF
        sbis    PINB,7
        clr     r16
        out     PORTA,r16
        jmp     KNAPP_KONTROLL
```

Kör du programmet i full fart ska alla dioder tändas vid nedtryckning av knappen och släckas annars.

8. Sista steget här är att inte tända/släcka alla dioder utan bara de som tangentbordet vill enligt uppgiften ovan. Bäst är nu om hela programmet redan är uppdelat i klippfärdiga bitar. Här menar jag förstås subrutiner. Skriv subrutiner!
"Bra" rutiner som du kanske redan har vid det här laget är:

   - `READ_KBD` (läser tangentbord och returnerar inläst värde i `r16`)
   - `EMIT` (skicka ut värdet i `r16` till dioderna)

   Programmet kan då se ut så här:

```
MAIN_LOOP:
        call    READ_KBD
        call    EMIT
        jmp     MAIN_LOOP
```

   Hela koden kan skrivas som (med två rader för att känna av tryckknappen):

```
MAIN_LOOP:
        call    READ_KBD
        sbis    PINB,7
        clr     r16
        call    EMIT
        jmp     MAIN_LOOP
```

   Provkör. Asch, funkar bara nästan! "`clr r16`" gör ju att displayen släcks om tangenten inte är nedtryckt. Prova med complement-instruktionen istället:

```
MAIN_LOOP:
        call    READ_KBD
        sbis    PINB,7
        com     r16     ; complement istf clr
        call    EMIT
        jmp     MAIN_LOOP
```

   Provkör. Asch, funkar bara nästan! Inverterat värde skulle ut när man tryckte på knappen, inte när man *inte* tryckte på knappen. En sista ändring då:

```
MAIN_LOOP:
        call    READ_KBD
        sbic    PINB,7  ; skip on no key
        com     r16
        call    EMIT
        jmp     MAIN_LOOP
```

   Det där borde funka.

Det blev långt det här men avsikten är att påminna att man måste göra allt i små steg. Varje steg är i sig inte så komplicerat, men att sätta ihop till subrutiner, ändra `clr`→`com` och `sbis`→`sbic` kräver många bollar i luften och en koncentration och lugn som kanske inte infinner sig vid examinationen.

För att fullfölja subrutinstänket skulle programmet slutligen kunna vara

```
MAIN_LOOP:
        call    READ_KBD
        call    INVERT_ON_KEY
        call    EMIT
        jmp     MAIN_LOOP
```

där `INVERT_ON_KEY` utför inverteringen som önskat på argumentet `r16` och returnerar rätt värde i `r16` à la "`r16 = INVERT_ON_KEY(r16)`".

En minimal lösning (utan subrutiner, så inte giltig på LAX) kan göras på mindre än 20 bytes. Kan du hitta den?

-o-O-o-

Datorteknik
LAX-DEMO 1

Tidsomfattning: 90 minuter
inkl redovisning

Uppgift I labsatsen finns ett hexadecimalt tangentbord. Det ger ut fyra bitar data (D,C,B,A) vid nedtryckt tangent men även en *strobe*-signal som är hög så länge *någon* tangent är nedtryckt.

Din uppgift är att läsa av det hexadecimala tangentbordet, en siffra åt gången, och presentera dess decimala motsvarighet på två sjusegmentsdisplayer, tiotalssiffran till vänster och entalssiffran till höger. Displayerna kan visa de hexadecimala siffrorna `0-F`, men ska här bara visa `0-9`. Senaste decimaltal ska kontinuerligt visas tills en ny siffra trycks ned på tangentbordet.

Tangentbordet kan inte ge flera utsignaler även om flera tangenter trycks ned samtidigt, följaktligen behöver inte programmet ta någon hänsyn till detta fall.

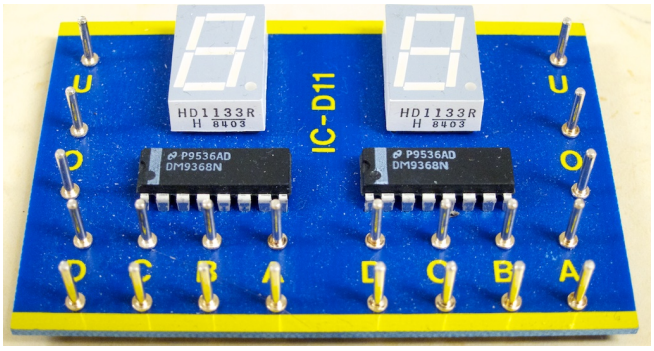
Obs! Hårdvaruinitieringen **måste** utföras som en subrutin.

Hårdvara

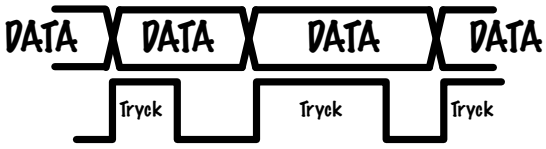
- labsats
- sjusegmentsdisplay
- hexadecimalt tangentbord

Funktionskontroll och examination Funktionen prioriteras! Någon kodgranskning, utöver kontroll av att hårdvaruinitieringen är utförd som subrutin, kommer inte ske. Funktionen kontrolleras genom upprepade tryckningar på det hexadecimala tangentbordet och kontroll på sjusegmentsdisplayen. Nöjaktig funktion resulterar i godkänd LAX.

Sjusegmentsdisplayen Matningsspänning är U (5 V) och 0 (0 V). Indata till respektive segment är de fyra bitarna D, C, B, A.



Tangentbordet med kopplingsplatta Matningsspänning +5 V påförs pinne 1, 0 V pinne 8. Utdata, fyra bitar, återfinns på pinnarna 3, 5, 7 och 9. *Strobe*-signalen är pinne 2. Stroben är hög så länge någon knapp är nedtryckt. Datat kommer samtidigt och ligger kvar tills nästa knappnedtryckning:



Datorteknik
LAX-DEMO 2

Tidsomfattning: 90 minuter
inkl redovisning

Uppgift I labsatsen finns två tryckknappar. Dessa ger en positiv och en negativ flank som utsignal från var sina stift, för respektive tryckknapp.

Din uppgift är att räkna antalet nedtryckningar av den vänstra tryckknappen. När den högra tryckknappen trycks ned ska detta antal visas på en sjusegmentsdisplay, och fortsätta att visas även efter att den högra tryckknappen släppts upp. Därefter ska man kunna börja om med att räkna nedtryckningar av den vänstra tryckknappen. Displayerna kan visa de hexadecimala siffrorna `0`–`F`. Trycker man mer än 15 gånger på den vänstra tryckknappen så ska displayen visa `F`, dvs 15_{10} hexadecimalt.

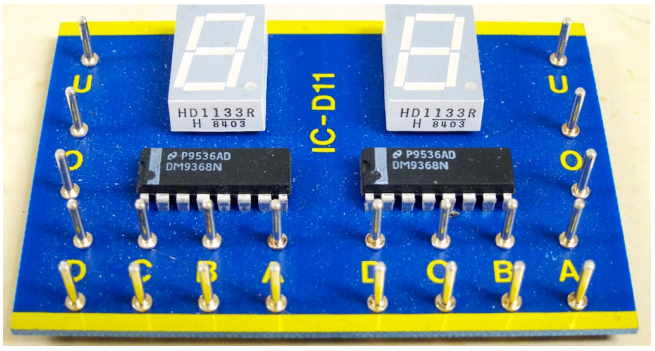
Obs! Hårdvaruinitieringen **måste** utföras som en subrutin.

Hårdvara

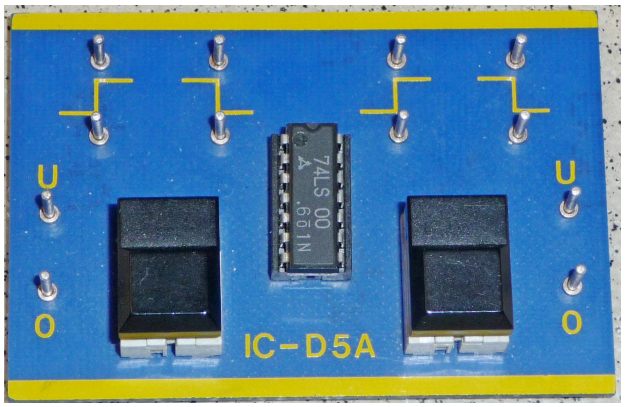
- sjusegmentsdisplay
- tryckknappar

Funktionskontroll och examination Funktionen prioriteras! Någon kodgranskning, utöver kontroll av att hårdvaruinitieringen är utförd som subrutin, kommer inte ske. Funktionen kontrolleras genom upprepade tryckningar på tryckknapparna och kontroll på sjusegmentsdisplayen. Nöjaktig funktion resulterar i godkänd LAX.

Sjusegmentsdisplayen Matningsspänning är U (5 V) och 0 (0 V). Indata till respektive segment är de fyra bitarna D, C, B, A.



Tryckknappar Matningsspänning är U (5 V) och 0 (0 V). Varje knapp ger en positiv och en negativ flank som utsignal från var sina stift då knappen trycks ned. Utsignalen återgår sedan när knappen släpps upp.



Datorteknik
LAX-DEMO 4

Tidsomfattning: 90 minuter
inkl redovisning

Uppgift: I labsatsen finns ett hexadecimalt tangentbord. Det ger ut fyra bitar data (D,C,B,A) vid nedtryckt tangent men även en *strobe*-signal som är hög så länge *någon* tangent är nedtryckt.

Din uppgift är att visa nedtryckta decimala tangentvärden från tangentbordet på vänster alternativt höger indikator på sjusegmentsdisplayen. Med tangenten F ska man kunna växla (toggla) indikator så att efterföljande tangentvärden hamnar till vänster ifall höger indikator tidigare användes, och vice versa. Gamla värden ska dock alltid ligga kvar tills dom ersätts av nya från tangentbordet. Tangenterna A, B, C, D och E ska inte ha någon funktion eller påverkan.

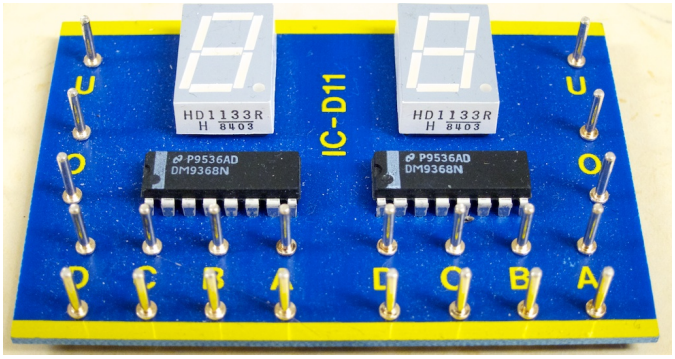
Obs! Hårdvaruinitieringen **måste** utföras som en subrutin.

Hårdvara

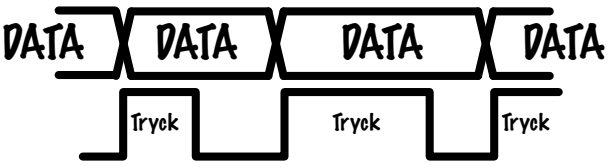
- labsats
- sjusegmentsdisplay
- hexadecimalt tangentbord

Funktionskontroll och examination Funktionen prioriteras! Någon kodgranskning, utöver kontroll av att hårdvaruinitieringen är utförd som subrutin, kommer inte ske. Funktionen kontrolleras genom upprepade tryckningar på det hexadecimala tangentbordet och kontroll på sjusegmentsdisplayen. Nöjaktig funktion resulterar i godkänd LAX.

Sjusegmentsdisplay Displayen har två sjusegments indikatorer. Matningsspänning är U (5 V) och 0 (0 V). Indata till respektive segment är de fyra bitarna D, C, B, A.



Tangentbordet med kopplingsplatta Matningsspänning +5 V påförs pinne 1, 0 V pinne 8. Utdata, fyra bitar, återfinns på pinnarna 3, 5, 7 och 9. *Strobe*-signalen är pinne 2. Stroben är hög så länge någon knapp är nedtryckt. Datat kommer samtidigt och ligger kvar tills nästa knappnedtryckning:



Datorteknik
LAX-DEMO 5

Tidsomfattning: 90 minuter
inkl redovisning

Uppgift: I labsatsen finns ett hexadecimalt tangentbord. Det ger ut fyra bitar data (D,C,B,A) vid nedtryckt tangent men även en *strobe*-signal som är hög så länge *någon* tangent är nedtryckt.

Din uppgift är att visa ett nedtryckt tangentvärde binärt på lysdioddisplayens fyra mest vänstra lysdioder. Samma tangentvärde ska även visas på dom fyra mest högra lysdioderna, inverterat (obs **ej** speglat) eller normalt. Inverterad eller normal visning för dom fyra mest högra lysdioderna togglas med tangent 0, som också ska visas på samma sätt som övriga tangentvärden. Dvs, om normal visning råder ska lysdioderna 7–4 och lysdioderna 3–0 visa samma sak, men om inverterad visning råder ska lysdioderna 3–0 visa det inverterade värdet av lysdioderna 7–4, vilka alltså alltid ska visa tangentvärdet.

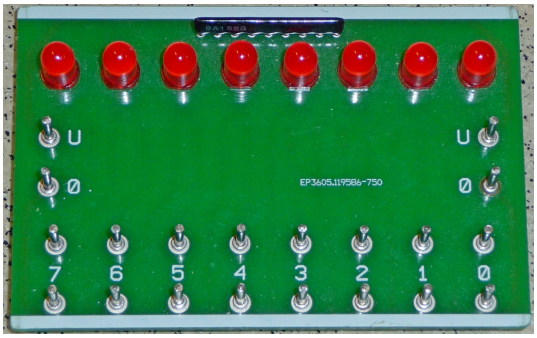
Obs! Hårdvaruinitieringen **måste** utföras som en subrutin.

Hårdvara

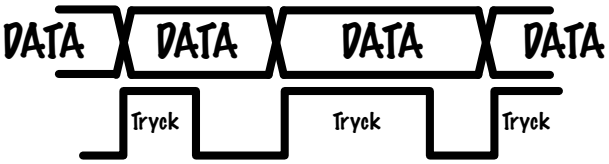
- labsats
- lysdioddisplay
- hexadecimalt tangentbord

Funktionskontroll och examination Funktionen prioriteras! Någon kodgranskning, utöver kontroll av att hårdvaruinitieringen är utförd som subrutin, kommer inte ske. Funktionen kontrolleras genom upprepade tryckningar på det hexadecimala tangentbordet och kontroll på lysdioddisplayen. Nöjaktig funktion resulterar i godkänd LAX.

Lysdioddisplay Matningsspänning är U (5 V) och 0 (0 V). Lysdioderna styrs individuellt av ingångarna 7 till 0 nederst på plattan.



Tangentbordet med kopplingsplatta Matningsspänning +5 V påförs pinne 1, 0 V pinne 8. Utdata, fyra bitar, återfinns på pinnarna 3, 5, 7 och 9. *Strobe*-signalen är pinne 2. Stroben är hög så länge någon knapp är nedtryckt. Datat kommer samtidigt och ligger kvar tills nästa knappnedtryckning:



Lösningsförslag

Instruktioner

Inlärningen sker i den kreativa processen i hjärnan när du själv konstruerar en lösning. Titta inte på förslagen nedan om du inte har ett eget förslag att jämföra med!

Man lär sig koda bättre genom att läsa mycket kod. Jämför förslagen med din egen lösning och förbättra dem båda. Vad kan göras för att få mer lättläst kod? Är strukturen den bästa? Skulle koden tjäna på globala konstanter? Variabelnamn? Finns det alternativa lösningsmetoder?

Vid LAX-tillfället sker visserligen ingen kodgranskning men man tjänar ändå på att ha ett strukturerat angreppssätt med subrutiner och bra namngivning av *labels*.

Namn	In-enhet	Ut-enhet
`LaxDemo1.asm`	Hextangentbord	2 x 7-segmentsdisplay
`LaxDemo2.asm`	2 x Tryckknapp	2 x 7-segmentsdisplay
`LaxDemo2a.asm`	Dalia	Dalia
`LaxDemo4.asm`	Hextangentbord	2 x 7-segmentsdisplay
`LaxDemo5.asm`	Hextangentbord	Lysdioddisplay
`LaxDemo5mini.asm`	Hextangentbord	Lysdioddisplay

`LabDemo5mini.asm` är ett försök att konstruera en resurssnål lösning. Här har man eliminerat kod genom att bland annat koppla om hårdvaran och ta bort — i det här fallet — onödiga instruktioner. Lösningen tillhör kategorin ”ful kod”.

```
        /* LaxDemo1.asm
         * Compiles to 50 bytes (42 if call etc)
         */

COLD:
        ldi     r16,HIGH(RAMEND)
        out     SPH,r16
        ldi     r16,LOW(RAMEND)
        out     SPL,r16
        call    HW_INIT
MAIN:
        sbis    PINA,4          ; wait for strobe/key press
        jmp     MAIN
        in      r16,PINA        ; read key
        andi    r16,$0F
        cpi     r16,10
        brmi    PRINT
        subi    r16,$FA
PRINT:
        out     PORTB,r16
WAIT:
        sbic    PINA,4          ; wait for key release
        jmp     WAIT
        jmp     MAIN            ; process next digit

        ; --- Config I/O
HW_INIT:
        ldi     r16,0
        out     DDRA,r16        ; PORTA<4> strobe, PORTA<3-0> data
        dec     r16
        out     DDRB,r16        ; PORTB<7-0> display
        ret
```

```
        /* LaxDemo2.asm
         * Compiles to 70 bytes (62 if call etc)
         */

COLD:
        ldi     r16,HIGH(RAMEND)
        out     SPH,r16
        ldi     r16,LOW(RAMEND)
        out     SPL,r16
        call    HW_INIT
WARM:
        call    GETKEYS
        sbrc    r16,1
        inc     r17             ; left pressed
        sbrc    r16,0
        call    SHOWIT          ; right pressed
        jmp     WARM

SHOWIT:
        cpi     r17,$0F
        brmi    SHOWIT2
        ldi     r17,$0F
SHOWIT2:
        out     PORTB,r17
        ; clr   r17             ; show progress
        ret

GETKEYS:
        in      r16,PIND
        andi    r16,$03
        cpi     r16,$00
        brne    GETKEYS         ; wait for release
GETKEYS2:
        in      r16,PIND
        andi    r16,$03
        cpi     r16,$00
        breq    GETKEYS2        ; wait for press
        ret

HW_INIT:
        ldi     r16,$00
        out     DDRD,r16        ; bit1 - left, bit0 - right
        ldi     r16,$FF
        out     DDRB,r16
        clr     r17             ; no sum yet
        ret
```

```
        /* LaxDemo2a.asm
         * Compiles to 72 bytes (64 if call etc)
         * Version for Dalia
         * Input: Buttons INT1 and INT0
         * Output: on-board LED
         */

COLD:
        ldi     r16,HIGH(RAMEND)
        out     SPH,r16
        ldi     r16,LOW(RAMEND)
        out     SPL,r16
        call    HW_INIT
WARM:
        call    GETKEYS
        sbrs    r16,3
        inc     r17             ; left pressed
        sbrs    r16,2
        call    SHOWIT          ; right pressed
        jmp     WARM

SHOWIT:
        cpi     r17,$0F
        brmi    SHOWIT2
        ldi     r17,$0F
SHOWIT2:
        out     PORTB,r17
        ; clr   r17             ; show progress
        ret

GETKEYS:
        in      r16,PIND
        andi    r16,$0C
        cpi     r16,$0C
        brne    GETKEYS         ; wait for release
GETKEYS2:
        in      r16,PIND
        andi    r16,$0C
        cpi     r16,$0C
        breq    GETKEYS2        ; wait for press
        ret

HW_INIT:
        ldi     r16,$00
        out     DDRD,r16
        ldi     r16,$FF
        out     DDRB,r16
        out     PORTD,r16       ; pull-up, PD3 left, PD2 right
        clr     r17             ; no sum yet
        ret
```

```
        /* LaxDemo4.asm
         * Compiles to 86 bytes (72 if rcall etc)
         */

COLD:
        ldi     r16,HIGH(RAMEND)
        out     SPH,r16
        ldi     r16,LOW(RAMEND)
        out     SPL,r16
        call    HW_INIT
WARM:
        call    GETKEY
        cpi     r16,$0F         ; "F"?
        brne    NO_TOGGLE
        com     r18             ; yep!
NO_TOGGLE:
        cpi     r16,10          ; 0-9?
        brpl    WARM            ; A-F no update
        in      r17,PORTB       ; get displayed
        sbrs    r18,0           ; 0 -> right, 1 -> left
        jmp     RIGHT
LEFT:
        andi    r17,$0F         ; clear left
        swap    r16             ; put digit in place
        jmp     SHOWIT
RIGHT:
        andi    r17,$F0         ; clear right
SHOWIT:
        or      r17,r16         ; merge
        out     PORTB,r17       ; and display
        jmp     WARM

        ; --- GETKEY returns key in r16
GETKEY:
        sbic    PINA,4          ; wait for release
        jmp     GETKEY
GETKEY2:
        sbis    PINA,4
        jmp     GETKEY2         ; wait for press
        in      r16,PINA        ; get key
        andi    r16,$0F
        ret

        ; --- I/O
        ; PA4   STROBE
        ; PA3-0 Data
        ; PB7-4 Left digit
        ; PB3-0 Right digit
HW_INIT:
        ldi     r16,$00
        out     DDRA,r16
        ldi     r16,$FF
        out     DDRB,r16
        ldi     r16,$00
        out     PORTB,r16       ; "00"
        clr     r18             ; toggle byte
        ret
```

```
        /* LaxDemo5.asm
         * Compiles to 70 bytes (60 if rcall etc)
         */

COLD:
        ldi     r16,HIGH(RAMEND)
        out     SPH,r16
        ldi     r16,LOW(RAMEND)
        out     SPL,r16
        rcall   HW_INIT
WARM:
        rcall   GETKEY
        mov     r17,r16         ; r16 rightmost
        brne    NO_ZERO
        com     r19             ; was a "0"
NO_ZERO:
        cpi     r19,0
        breq    WARM3
        ldi     r18,$0F         ; invert right
        eor     r16,r18
WARM3:
        swap    r17
        or      r16,r17
        out     PORTB,r16
        rjmp    WARM

        ; --- GETKEY Return pressed key in r16
GETKEY:
        sbic    PINA,4          ; wait for release
        rjmp    GETKEY
GETKEY2:
        sbis    PINA,4          ; wait for press
        rjmp    GETKEY2
        in      r16,PINA
        andi    r16,$0F         ; return key
        ret

        ; --- I/O init, initial state
HW_INIT:
        clr     r16
        out     DDRA,r16
        ldi     r16,$FF
        out     DDRB,r16
        clr     r19             ; 0 -> normal, 1 -> inverted
        ret
```

```
        /*
         * LaxDemo5mini.asm
         * Attempt at minimal, and hence ugly, code
         * Compiles to 34 bytes
         */

COLD:
        ldi     r16,HIGH(RAMEND)
        out     SPH,r16
        ldi     r16,$FF
        out     DDRB,r16
        ldi     r17,$0F
WARM:
        sbic    PIND,0          ; wait for release
        rjmp    WARM
KEY:
        sbis    PIND,0          ; wait for press
        rjmp    KEY
        in      r16,PINA
        cpi     r16,0
        brne    CONT
        com     r17             ; was a "0"
CONT:
        sbrc    r17,0
        eor     r16,r17
        out     PORTB,r16
        rjmp    WARM
```