DIT150 Maskinorienterad programmering GU

DAT015 Maskinorienterad programmering IT

Tentamen

Söndag 17 december 2006, kl. 14.00 - 18.00 i V-salar

Examinatorer
Rolf Snedsböl, tel. 772 1665
Jan Skansholm, tel. 772 1012

Kontaktpersoner under tentamen
Som ovan

Tillåtna hjälpmedel

Häftet

Instruktionslista för CPU12

I den får rättelser och understrykningar vara införda, inget annat.

Du får också använda bladet

C Reference Card

samt boken

Vägen till C, Bilting, Skansholm, Studentlitteratur

Tabellverk och miniräknare får ej användas!

Allmänt
Siffror inom parentes anger full poäng på uppgiften. **Full poäng kan fås om**:

- redovisningen av svar och lösningar är läslig och tydlig. **OBS!** Ett lösningsblad får endast innehålla redovisningsdelar som hör ihop med en uppgift.
- din lösning ej är onödigt komplicerad.
- du motiverat dina val och ställningstaganden
- redovisningen av en hårdvarukonstruktion innehåller funktionsbeskrivning, lösning och realisering.
- redovisningen av en mjukvarukonstruktion i assembler är fullständigt dokumenterad, d v s är redovisad både i strukturform (flödesplan eller pseudospråk) och med kommenterat program i assemblerspråk, om inget annat anges i uppgiften.
- C-program är utformade enligt de råd och anvisningar som givits under kursen. I programtexterna skall raderna dras in så att man tydligt ser programmens struktur. När så anges skall programtexten också vara indelad i moduler med användning av include-filer.

Betygsättning
För godkänt slutbetyg på kursen fordras att både tentamen och laborationer är godkända. På tentamen fordras 20p, varav minst 10p på datorteknikdelen (uppg 1-3) och 7p på C-delen (uppg 4). Tentamen ger slutbetyget:

20p ≤ **betyg 3** < 30p ≤ **betyg 4** < 40p ≤ **betyg 5**

Lösningar
anslås på kursens www hemsida.

Betygslistan
anslås såsom anges på kursens hemsida.

Granskning
Tid och plats anges på kursens hemsida.

Institutionen för data- och informationsteknik
CHALMERS TEKNISKA HÖGSKOLA

1. ***Adressavkodning***. Konstruera adressavkodningslogiken för ett MC12-system där vi önskar en yttre ROM-modul och en yttre RWM-modul. Dessutom skall två 8-bitars IO-portar finnas. Alla chip select signaler är aktiva låga

ROM-modulen är 16-kbyte stor och skall ha sin sista adress på $FFFF. RWM-modulen vi har tillgång till är 32-kbyte och skall placeras i adressområdet $4000 till $7FFF.

Använd fullständig adressavkodningslogik för minnesmodulerna.

Inporten och utporten får placeras på valfri (valfria) adress (adresser) i det lediga adressutrymmet. Använd så få grindar som möjligt (ofullständig adressavkodning) när du konstruerar adressavkodningen för IO-portarna.

I din lösning (innehållande argument för dina val) vill vi ha
adressavkodningslogiken (ritad med valfria grindar) för minnerna och I/O-portarna
ett blockdiagram på hur RWM-modulen skall anslutas till systemet. **(5p)**

2. ***Avbrott och assemblerprogrammering***. Ett MC12-system är bestyckad med en pulsgenerator som genererar avbrott varje millisekund och en klockmodul som kan visa tid.

Du skall skriva avbrottsrutinerna som räknar ner till ”12-slaget” på nyårsafton.

Du behöver en rutin (IRQINIT) som initierar systemet och en avbrottsrutin (IRQ) som minskar en klock-variabel. Klockvariabeln skrivs till en display av huvudprogrammet. När programmet startas skall displayen visa (börja på) 23:59:59. (Vi skall räkna ner det sista dygnet). Avbrott kvitteras genom en skrivning på den symboliska adressen IRQRES.

Initieringsrutinen (IRQINIT): Skriv en rutin som initierar nödvändiga variabler och systemet för avbrott. Det finns inga andra avbrottskällor än pulsgeneratorn i systemet.

Avbrottsrutinen (IRQ): Skriv en avbrottsrutin som uppdaterar klock-variabeln. Variabeln hittas på den symboliska adressen CLOCK och består av 3 bytes enligt

```
CLOCK   RMB 3   Variabel innehållande klockan tt:mm:ss
```

där tt är 0-23 timmar, mm är 0-59 minuter och ss 0-59 sekunder. Alla tre lagras som NBCD-tal. När klockan räknat ner till noll skall den stanna och huvudprogrammet fortsätta som vanligt.

Du får själv skapa ytterligare hjälpvariabler för klockavbrotten. **(10p)**

3. ***Småfrågor***.

a. *Seriekommunikation*. Vi diskuterade problem med att bibehålla synkroniseringen mellan sändare och mottagare när man skickar långa sekvenser av data.
Hur är detta löst i CAN-protokollet?
Ge ett annat exempel på hur man göra för att bibehålla synkroniseringen? **(2p)**

b. *Realtidssystem*. Händelsestyrda (Event triggered) och tidsstyrda (Time triggered) styrsystem diskuterades under föreläsningen. Beskriv kort skillnaderna på dessa system och ge någon fördel och nackdel med dessa system. **(2p)**

c. *Grafik***.** Det fokuserades på ”trianglar” under grafikföreläsningen.
Vad innehåller en triangel för information. **(1p)**

d. *Grafik***.** Vad är den största prestandabegränsningen i samband med grafikkort? **(1p)**

.

e. *Borrmaskinlabbet.* Skriv assemblerrutinen för OUTONE enligt följande specifikation.

```
* Subrutin OUTONE. Läser kopian av
* borrmaskinens styrord på adress DCCopy.
* Ettställer en av bitarna och skriver det
* nya styrordet till utporten DCTRL samt
* tillbaka till kopian DCCopy.
* Biten som ska ettställs ges av innehållet
* i B-registret (0-7) vid anrop.
* Om (B)> 7 utförs ingenting.
*
* Anrop:  LDB    #bitnummer
*         JSR    OUTONE
*
* Bitnumrering framgår av följande figur.
```

7	6	5	4	3	2	1	0

```
* Utdata:               Inga
* Registerpåverkan:     Ingen
* Anropade subrutiner:  Inga
```

(5p)

f. *Handskakning.* Vad menas ”busy wait” och ”polling”. Ange kort fördelar och nackdelar. Rita även flödesplaner *eller* ge enkla exempel på programkod i ditt svar. **(2p)**

g. *Instruktionslistan för CPU12.* Instruktionen **`BRCLR Inport,#3,Loop`** fungerar ej i labsystemet. Ange en instruktionssekvens som utför denna operation **(2p)**

4 *C-programmering*

Uppgiften (en stor) innehåller mycket text men det är nödvändigt och den är inte så svår som det kanske verkar vid första anblicken. Uppgiften knyter an till sådant du gjort på labbarna 2, 4 och 5.

Obs! Filen queue.h finns som bilaga

I laboration nr 2 fick du prova på att konstruera ett program som lät två programdelar, s.k. processer, exekvera ”samtidigt”, dvs. pseudoparallellt. I denna uppgift skall du få skriva några delar av en realtidskärna som tillåter att man kör ett godtyckligt antal processer samtidigt. Meningen är att man från ett användarprogram skall kunna skapa och starta parallella processer genom att anropa funktioner från realtidskärnan. Bl.a. skall följande funktioner kunna anropas. (Du skall inte skriva alla dessa!)

```
void init_processes();                          // initierar realtidskärnan
typedef void (*function)(void);
process *create_process(function f, int prio); // skapar ny process
void start_process(process *);                  // startar en process
process *running_process();                     // ger aktuell process
void delay_process(process *p, long int t);     // fördröjer p t tidsenheter
void change_priority(process *p, int prio);     // ändrar p:s prioritet
void terminate(process *p);                     // avslutar p
```

Typen `process` är en struct-typ (se deluppgift b) och parametern `f` till funktionen `create_process` pekar ut den funktion som den nya processen skall exekvera.
Förutom assemblerrutinerna i deluppgift a skall alla funktioner i realtidskärnan skrivas i C.

Deluppgift a

Processbyte kan bara göras när ett avbrott har skett. Då har som du vet alla register i processorn (inklusive återhoppsadressen) automatiskt lagts överst på stacken. Själva processbytet sker sedan genom att man ändrar stackpekarregistret så att det pekar ut en annan stack. Detta måste göras i en avbrottsrutin skriven i assembler. En sådan avbrottsrutin kan emellertid vara mycket enkel. Antag att själva hanteringen av processer görs i realtidskärnan i en intern C-funktion med namnet `reschedule` och att realtidskärnan dessutom innehåller en extern variabel som har definitionen

```
struct stack_frame *inter_sp;
```

(Structen `stack_frame` beskriver hur processorns register lagras på stacken vid avbrott. Du behöver inte känna till detaljerna för att lösa denna uppgift.) Vad en avbrottsrutin behöver göra är att kopiera stackpekarregistrets värde till variabeln `inter_sp` och sedan anropa funktionen `reschedule`. Denna funktion sparar undan det värde som finns i variabeln `inter_sp` och uppdaterar sedan denna variabel så att den pekar på den nya stacken. Vid returen från `reschedule` skall därför avbrottsrutinen kopiera det värde som finns i variabeln `inter_sp` till stackpekarregistret och sedan återvända från avbrottet.

Processbyten skall kunna göras vid två olika tillfällen, dels vid klockavbrott och dels när man uttryckligen begär det i programmet (man vill t.ex. fördröja den aktuella processen ett visst tidsintervall). För att åstadkomma ett avbrott från programmet kan man exekvera maskininstruktionen `SWI` (software interrupt).

Din uppgift är att skriva följande fyra assemblerrutiner:

- `void sw_interrupt()` Anropas från C. Åstadkommer ett avbrott genom att utföra instruktionen `SWI`.
- `void enableinter()` Anropas från C. Nollställer I-flaggan så att avbrottsmekanismen aktiveras.
- `void swtrap()` Anropas vid software interrupt. Skall anropa `reschedule` så som beskrivits ovan.
- `void clocktrap()` Anropas vid klockavbrott. Fungerar som `swtrap`, men skall dessutom anropa den färdigskrivna C-funktionen `clock_inter` vilken bl.a. räknar upp den externa variabeln `current_time`.

Du får anta att det finns en färdigskriven C–funktion `init_vect()` som initierar avbrottsvektorerna för klockavbrott och för software interrupt.

(6p)

Deluppgift b

Processerna beskrivs med hjälp av följande struct-typ:

```
struct process_struct {
  struct stack_frame *stack_ptr;  // pekare till toppen på processens stack
  int pid;                        // processens nummer
  int priority;                   // processens prioritet
  unsigned long int start_time;   // tidigaste starttid
};
typedef struct process_struct process;
```

Varje process har en egen stack och pekaren `stack_ptr` pekar ut stackens topp. Varje process tilldelas automatisk ett eget unikt nummer när processen skapas. När man skapar en ny process skall man ange processens prioritet. Denna avgör vilken process som får exekvera om flera processer tävlar

om processorn. Man kan fördröja en process. I så fall används komponenten `start_time` för att hålla reda på tidigaste tidpunkt när processen får starta.

I realtidskärnan finns bl.a. följande tre variabler definierade:

```
static process *running;        // pekare till den process som körs
static Queue   waiting;         // en kö med pekare till övriga processer
static Iterator waiting_it;     // iterator som är kopplad till kön waiting
```

Kön `waiting` innehåller pekare av typen `process *`. Processerna läggs in i prioritetsordning i kön. (Hur typerna `Queue` och `Iterator` är deklarerade framgår av bilagan. Det är exakt samma typer som du själv arbetade med i laboration nr 4 och du får anta att de är implementerade och att alla deras funktioner kan anropas.)

Du får förutsätta att funktionerna `init_processes` och `create_process` är färdigskrivna. Funktionen `init_processes` initierar variablerna `running`, `waiting` och `waiting_it`. Den skapar dessutom två processer, en som beskriver exekveringen i funktionen `main` och en som beskriver en ”vänteprocess” med låg prioritet. Vänteprocessen har lagts in i kön `waiting`, medan processen som beskriver `main` blir den aktuella processen och pekas ut av variabeln `running`. Funktionen `init_processes` anropar också funktionerna `init_vect` och `enable_inter` från deluppgift a.

Din uppgift är att skriva funktionen `reschedule` som anropas när ett avbrott har inträffat. (Se deluppgift a.) Funktionen `reschedule` skall eventuellt byta ut den aktuella processen mot en annan. Funktionen skall ta fram den första processen i kön `waiting`. Men den måste hoppa över sådana processer som är fördröjda. Den aktuella tidpunkten finns i den externa variabel `current_time` och denna kan jämföras med processernas tidigaste starttid. Du kan förutsätta att kön aldrig är tom eftersom det finns en ”vänteprocess”. Därför hittar man alltid en körbar process. Denna skall jämföras med den aktuella processen. Om den aktuella processen har lägre prioritet än den som valts ur kön skall processbyte ske. Detta skall man också göra om den aktuella processen blivit fördröjd (dess komponent `start_time` har givits ett värde som är större än den aktuella tidpunkten.) Om processbyte skall göras skall den nya processen tas ut ur kön `waiting` och den gamla processen läggas dit istället. Dessutom skall den gamla stackpekaren som finns i variabeln `inter_sp` sparas i komponenten `stack_ptr`. Istället skall stackkpekaren för den nya processen läggas i variabeln `inter_sp`.

(8p)

***Deluppgift* c**

Din uppgift är att implementera de tre funktionerna `running_process`, `start_process` och `delay_process`. För att en process skall kunna startas måste man lägga den i väntekön. En process fördröjs genom att man sätter komponenten `start_time` i structen `process`. Tänk på att man om man ändrat i väntekön eller ändrat starttiden för en process så måste man sedan åstadkomma ett avbrott så att funktionen `reschdule` kommer att anropas.

(6p)

Bilaga 1 - Assemblerspråket för mikroprocessorn CPU12.

Assemblerspråket använder sig av de mnemoniska beteckningar som processorkonstruktören MOTOROLA specificerat för maskininstruktioner och instruktioner till assemblatorn, s k pseudoinstruktioner eller assemblatordirektiv. Pseudoinstruktionerna framgår av följande tabell:

Direktiv	Förklaring
ORG N	Placerar den efterföljande koden med början på adress N (ORG för ORiGin = ursprung)
L RMB N	Avsätter N bytes i följd i minnet (utan att ge dem värden), så att programmet kan använda dem. Följden placeras med början på adress L. (RMB för Reserve Memory Bytes)
L EQU N	Ger label L konstantvärdet N (EQU för EQUates = beräknas till)
L FCB N1, N2	Avsätter i följd i minnet en byte för varje argument. Respektive byte ges konstantvärdet N1, N2 etc. Följden placeras med början på adress L. (FCB för Form Constant Byte)
L FDB N1, N2	Avsätter i följd i minnet ett bytepar (två bytes) för varje argument. Respektive bytepar ges konstantvärdet N1, N2 etc. Följden placeras med början på adress L. (FDB för Form Double Byte)
L FCS "ABC"	Avsätter en följd av bytes i minnet, en för varje tecken i teckensträngen "ABC". Respektive byte ges ASCII-värdet för A, B, C etc. Följden placeras med början på adress L. (FCS för Form Caracter String)

Bilaga 2 - ASCII-koden.

0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1	$b_6b_5b_4$ $b_3b_2b_1b_0$
NUL	**DLE**	**SP**	**0**	**@**	**P**	**`**	**p**	0 0 0 0
SOH	**DC1**	**!**	**1**	**A**	**Q**	**a**	**q**	0 0 0 1
STX	**DC2**	**“**	**2**	**B**	**R**	**b**	**r**	0 0 1 0
ETX	**DC3**	**#**	**3**	**C**	**S**	**c**	**s**	0 0 1 1
EOT	**DC4**	**$**	**4**	**D**	**T**	**d**	**t**	0 1 0 0
ENQ	**NAK**	**%**	**5**	**E**	**U**	**e**	**u**	0 1 0 1
ACK	**SYN**	**&**	**6**	**F**	**V**	**f**	**v**	0 1 1 0
BEL	**ETB**	**‘**	**7**	**G**	**W**	**g**	**w**	0 1 1 1
BS	**CAN**	**(**	**8**	**H**	**X**	**h**	**x**	1 0 0 0
HT	**EM**	**)**	**9**	**I**	**Y**	**i**	**y**	1 0 0 1
LF	**SUB**	*****	**:**	**J**	**Z**	**j**	**z**	1 0 1 0
VT	**ESC**	**+**	**;**	**K**	**[Ä**	**k**	**{ä**	1 0 1 1
FF	**FS**	**,**	**<**	**L**	**\Ö**	**l**	**|ö**	1 1 0 0
CR	**GS**	**-**	**=**	**M**	**]Å**	**m**	**}å**	1 1 0 1
S0	**RS**	**.**	**>**	**N**	**^**	**n**	**~**	1 1 1 0
S1	**US**	**/**	**?**	**O**	**_**	**o**	**RUBOUT (DEL)**	1 1 1 1

Bilaga 3 – queue.h

```
// Makrot DATA skall ange typen för sådana data som skall läggas i kön.
// Det bör ha definierats av användaren, annars sätts det här till void.

 #ifndef DATA
 #define DATA void
 #endif

 #ifndef QUEUE_H
 #define QUEUE_H

 struct qstruct;                  // anger att qstruct och qiteratorstruct
 struct qiteratorstruct;          // definieras på annat ställe

 typedef struct qstruct *Queue;            // typerna Queue och Iterator
 typedef struct qiteratorstruct *Iterator; // skall utnyttjas av användaren

 Queue new_queue();                // allokerar minnesutrymme för en ny kö
 void delete_queue(Queue q);       // tar bort kön helt och hållet
 void clear(Queue q);              // tar bort köelementen men behåller kön
 int  size(Queue q);                        // ger köns aktuella längd
 void add(Queue q, int priority, DATA *d); // lägger in d på rätt plats
 DATA *get_first(Queue q);                  // avläser första dataelementet
 void remove_first(Queue q);                // tar bort det första elementet

 Iterator new_iterator(Queue q);    // allokerar utrymme för en ny iterator
 void delete_iterator(Iterator it); // tar bort iteratorn
 void go_to_first(Iterator it);     // går till köns första element
 void go_to_last(Iterator it);      // går till köns sista element
 void go_to_next(Iterator it);      // går till till nästa element
 void go_to_previous(Iterator it);  // går till föregående element
 DATA *get_current(Iterator it);    // ger pekare till aktuellt
                                    // dataelementet eller 0,
                                    // om inget refereras
 void change_current(Iterator it,
                     DATA *d);      // ändrar aktuellt dataelementet
 void remove_current(Iterator it);  // tar bort aktuellt dataelement
 void find(Iterator it, DATA *d);   // söker d, iteratorn kommer att
                                    // referera till *d eller ge värdet 0
 #endif
```

Preliminära, kortfattade lösningar och svar

1. ROM. 16kbyte $\Rightarrow 2^4 \bullet 2^{10}$ byte $\Rightarrow$ 14 Adressbitar $\Rightarrow$ [A13,A0] direkt till ROM-kapsel.
RWM. 32kbyte $\Rightarrow 2^5 \bullet 2^{10}$ byte $\Rightarrow$ 15 Adressbitar $\Rightarrow$ [A14,A0] direkt till RWM-kapsel – Fast detta funkar inte när adressområde är [$4000,$7FFF] – Se tabellen nedan.
Minnesmodulerna och I/O-portarna tar upp följande adressområden:
Väljer att lägga IN- och UTport på samma adress då detta ger färre grindar.

		A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
ROM	Start: C000	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Stop: FFFF	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
RWM	Start: 4000 -	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Stop: 7FFF	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
I/O	0000	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

CS-ROM: Rita en NAND-grind enligt {A_{15}·A_{14}· ·E R/W}'
CS-RWM: Rita en NAND-grind enligt {A_{15}'· A_{14}· E}'
I/O: Bilda först IOSEL för både portarna enligt {A_{14}'·E} Rita sedan OUTSEL enligt {IOSEL R/W'}'och sedan INSEL enligt {IOSEL R/W}'

Blockdiagram för RWM-modul innehåller:
[A13,A0] direkt till kapsel, A14 till GND, R/W', CS_RWM och DATA [7,0]

2.
```
IRQINIT   psha
          pshx
          movw    #$2359,CLOCK    Init kolckan tt:mm:ss
          movb    #$59,CLOCK+2
          ldx     #1000           Avbrottsräknare
          stx     TEMP
          clr     IRQRES          nollställ avbrottsvippan
          ldx     #IRQ            avbrottsvektor
          stx     $fff2           (alt 3ff2)
          cli
          pulx
          pula
          rts

TEMP      rmb     2    Avbrottsräknare (1000 IRQ = 1s)

IRQ       clr     IRQRES          nollställ avbrottsvippan
          ldx     TEMP            1000 avbrott?
          dex
          stx     TEMP
          bne     IExit             nej
          ldx     #1000           Avbrottsräknare
          stx     TEMP
 * Minska sekunder
          ldaa    CLOCK+2
          adda    #-1
          daa
          staa    CLOCK+2         Hel minut?
          bpl     IExit             nej
 * Minska minuter
          movb    #$59,CLOCK+2    59 nya sekunder
          ldaa    CLOCK+1
          adda    #-1
          daa
          staa    CLOCK+1         Hel timme?
          bpl     IExit             nej
 * Minska timmar
          movb    #$59,CLOCK+1    59 nya minuter
          ldaa    CLOCK
          adda    #-1
          daa
          staa    CLOCK           24 timmar?
          bpl     IExit             nej
```

```
* Stanna klockan på något sätt!
* Använd någon global variabel o kolla om klockan är noll eller
* se till att förhindra framtida avbrott
          ldaa    0,sp            Ettställ I-flaggan
          oraa    #$10
          staa    0,sp
          movw    #0,CLOCK        Nolla klockan (som nu borde stå på -1:59:59)
          clr     CLOCK+2
IExit     rti                     (Plus programhuvud och flödesplan)
```

Upg 3

a. Stuffbitar. Se CAN-OH:n

b. Se RTS-OH:n

c. 3 hörn (3*xyz); 3 textur koordinater

d. Minnesbandbredd

e. Se Laboration 2

f. Se s 34 i Arb

g.

```
Loop   psha
       ldaa   Inport
       anda   #$03
       pula
       beq    Loop
```

Upg 4

// Deluppgift a

```
                segment text
                define   _clocktrap
                define   _swtrap
                define   _enableinter
                define   _sw_interrupt
                extern   _inter_sp
                extern   _reschedule
                extern _clock_inter
_sw_interrupt:  SWI
                RTS
_enableinter:   CLI
                RTS
_swtrap:        STS  _inter_sp
                JSR  _reschedule
                LDS  _inter_sp
                RTI
_clocktrap:     STS  _inter_sp
                JSR   _clock_inter
                JSR  _reschedule
                LDS  _inter_sp
                RTS
```

// Deluppgift b

```
void reschedule() {  // anropas vid avbrott, byter process om så behövs
    process *p;
    // leta reda på första körbara process i kön
    go_to_first(waiting_it);
    p = get_current(waiting_it);
    while (p->start_time > current_time) {
      go_to_next(waiting_it);
      p = get_current(waiting_it);
    }
    // Undersök om processbyte skall ske
    if (p->priority > running->priority || running->start_time >
current_time) {
      running->stack_ptr = inter_sp;   // spara stackpekaren för
processen
      inter_sp = p->stack_ptr;         // byt till annan  stack
      remove_current(waiting_it);      // ta bort den nya processen från
väntekön
      add(waiting, running->priority, running);   // lägg den gamla
processen i kön
      running = p;
    }
}
```

// Deluppgift c

```
process *running_process() {
  return running;
}

void start_process(process *p) {
  if (p) {
    add(waiting, p->priority, p);
    sw_interrupt();
  }
}

void delay_process(process *p, long int t) {
  if (p) {
    p->start_time = current_time + t;
    sw_interrupt();
  }
}
```