Institutionen för data- och informationsteknik
CHALMERS TEKNISKA HÖGSKOLA

Tentamen

EDA480 Maskinorienterad programmering D
EDA485 Maskinorienterad programmering Z
DIT151 Maskinorienterad programmering GU

Måndag 9 Mars 2008, kl. 14.00 - 18.00 i V-salar

Examinatorer
Roger Johansson, tel. 772 5729
Jan Skansholm, tel. 772 1012
Rolf Snedsböl, tel 772 1665

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

Även i denna får rättelser och understrykningar vara införda, inget annat.

Tabellverk och miniräknare får ej användas!

Allmänt
Siffror inom parentes anger full poäng på uppgiften. **För full poäng krävs att**:

- redovisningen av svar och lösningar är läslig och tydlig. Ett lösningsblad får endast innehålla redovisningsdelar som hör ihop med en uppgift.
- lösningen ej är onödigt komplicerad.
- du har motiverat dina val och ställningstaganden
- assemblerprogram är utformade enligt de råd och anvisningar som givits under kursen och tillräckligt dokumenterade.
- 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.

Lösningar
anslås såsom anges på kursens hemsida.

Betygslistan
anslås såsom anges på kursens hemsida.

Granskning
Tid och plats anges på kursens hemsida.

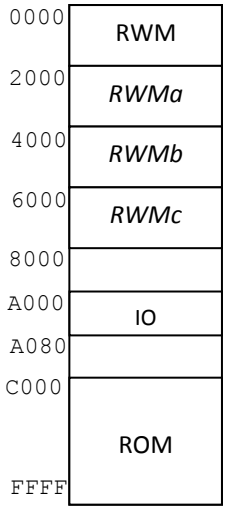
Betygsättning
För godkänt slutbetyg på kursen fordras att både tentamen och laborationer är godkända.

Tentamenspoäng ger slutbetyg enligt:

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

Uppgift 1 *Adressavkodning*

I ett befintligt MC12-system finns en yttre ROM-modul (ROM), en RWM-modul (RWM) och en IO-modul (IO). Modulerna är placerade i processorns adressrum enligt figuren till höger.



Fullständig adressavkodningslogik har använts för ROM-modulen och för IO-modulen.

För RWM-modulen däremot används ofullständig adressavkodning. Att ofullständig adressavkodning används illustreras i figuren genom att RWM-modulen speglas upprepade gånger som *RWMa, RWMb* och *RWMc*. Samtliga CS (”chip select”) signaler är aktivt låga.

Utgå ifrån informationen ovan och besvara följande:

a) Hur stora (hur många kByte) är RWM- respektive ROM-modulerna? **(1p)**

b) Ange adressavkodningslogiken för ROM- och IO-modulen. **(2p)**

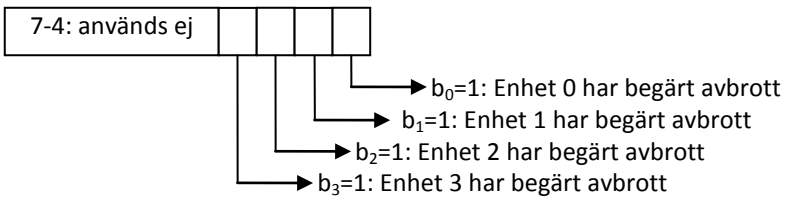
c) Ange adressavkodningslogiken för RWM-modulen. **(2p)**

d) Rita ett **blockdiagram** som visar hur processor, minne och IO-modul kopplas samman. Du har tillgång till separata adress- och databussar från processorn (ej multiplexad). Det skall klart framgå vilka signaler och vilka bussar som ingår i kopplingen. Ange även bussarnas bredd. **(2p)**

Uppgift 2 *Avbrott*

Ett MC12-system har fyra yttre enheter [0,3] som sporadiskt kan begära avbrott. Enhet 0 har högst prioritet och enhet 3 har lägst prioritet. Utöver dessa finns en realtidsklocka som ger avbrott varje millisekund. Det finns inga andra avbrottskällor och du skall nu skriva en avbrottshanterare för systemet.

De fyra yttre enheterna har ett gemensamt statusregister (`IrqStat`) innehållande fyra statusflaggor [bit 0,3] som anger vilken enhet som begärt avbrott (se figur). När någon av de fyra yttre enheterna begär avbrott skall tillhörande avbrottssubrutin anropas. Dessa heter `IrqRut0`, `IrqRut1`, `IrqRut2` och `IrqRut3`. Rutinerna kvitterar sitt avbrott genom att tillhörande bit i statusregistret nollställs.



När avbrott inträffat och ingen av de yttre enheterna har begärt avbrott så måste det vara ett klockavbrott. Avbrottssubrutinen för klockan heter `ClkIrq`. Subrutinen kvitterar avbrottet för klockan.

a) Skriv en avbrottshanterare (`IrqHandl`) för systemet. **(4p)**

b) Under vissa omständigheter ”tappar” systemet avbrott från klockan, förklara varför. Ge ett förslag på hur problemet kan lösas. **(4p)**

Uppgift 3 *Småfrågor*

a) Varför används så kallade ”stuff-bitar” i CAN-protokollet? **(1p)**

b) Kommunikation mellan processor och minne sker via adress- och databussar och kan vara synkron eller asynkron. Redogör kortfattat för skillnaden mellan dessa metoder. **(2p)**

c) Vad är viktigast för att öka prestanda vid utveckling av ett nytt grafikkort? **(1p)**

d) Vad menas med ett ”basic block” i samband med exekveringstidsanalys? **(1p)**

e) Varför ettställs I-flaggan i CC-registret vid RESET av MC68HCS12? **(1p)**

f) Hur bör du fördela din C-kod mellan *.c respektive *.h-filer i en C-applikation? **(1p)**

Uppgift 4

En C-funktion deklareras enligt följande:

```
void change_priority(process *p, int prio);
```

a) Visa aktiveringsposten i funktionen. **(2p)**

b) Beskriv hur respektive parameter refereras i funktionen, dvs. visa hur varje parameter placeras i register D (HC12 assemblerspråk). **(1p)**

Uppgift 5

Följande funktion finns given i “C”. Skriv motsvarande funktion i assemblerspråk för HC12. Förutsätt samma konventioner vid översättningen som i XCC12. **(5p)**

```
#define PPORT  *( char *) 0x700
#define EOT  4

void printerprint( char *s )
{
   while( *s != EOT )
   {
      PPORT = *s;
      s++;
   }
}
```

Uppgift 6

Skriv ett C-program som läser in en befintlig textfil och som skriver ut filens innehåll i kommandofönstret. Vid utskriften skall inga rader som enbart innehåller s.k. vita tecken skrivas ut. (Vita tecken är blanka tecken, tabulatortecken och radslutstecken.) I övrigt skall utskriften ha samma radstruktur som den som finns i filen.

Programmet skall kunna användas för vilken textfil som helst och skall därför börja med att fråga efter och läsa in filens namn från kommandofönstret. Om den angivna filen saknas skall en felutskrift ges och programmet avslutas.

Du får för enkelhets skull anta att ingen rad i textfilen är längre än 500 tecken och att filens namn inte är längre än 100 tecken. Försök att göra din lösning så enkel som möjligt genom att utnyttja C:s standardfunktioner. **(10p)**

Uppgift 7

Standardfunktionen **strspn** kan beskrivas på följande sätt:

```
size_t strspn ( const char * str1, const char * str2 );
```

Returns the length of the initial portion of *str1* which consists only of characters that are part of *str2*.

Parameters

str1 C string to be scanned.
str2 C string containing the characters to match.

Return value

The length of the initial portion of *str1* containing only characters that appear in *str2*.
Therefore, if all of the characters in *str1* are in *str2*, the function returns the length of the entire *str1* string, and if the first character in *str1* is not in *str2*, the function returns zero.

Exempel på användning:

```
int main()
{
  char string[]="7803 Elm St.";
  printf("The number length is %d.\n",strspn(string,"1234567890"));
  return 0;
}
```

Utskriften ska bli: The number length is 4.

Din uppgift är att, i C, skriva en egen definition av funktionen **strspn**. Du får inte använda dig av indexering, utan måste utnyttja pekare. Du får inte anropa någon annan standardfunktion. Du måste alltså skriva all kod själv. **(10p)**

Bilaga 1 - Assemblerdirektiv för MC68HC12.

Assemblerspråket använder sig av mnemoniska beteckningar som tillverkaren Freescale specificerat för maskininstruktioner och instruktioner till assemblatorn, s k pseudoinstruktioner eller assemblerdirektiv. 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: Kompilatorkonvention XCC12:

Parametrar överförs till en funktion via stacken.
Då parametrarna placeras på stacken bearbetas parameterlistan från höger till vänster.
Prolog kallas den kod som reserverar utrymme för lokala variabler.
Epilog kallas den kod som återställer (återlämnar) utrymme för lokala variabler.
Den del av stacken som används för parametrar och lokala variabler kallas *aktiveringspost*.
Beroende på datatyp används för returparameter HC12's register enligt följande tabell:

Storlek	**Benämning**	**C-typ**	**Register**
8 bitar	byte	char	B
16 bitar	word	short int	D
32 bitar	long	long int	Y/D

LÖSNINGSFÖRSLAG

Uppgift 1

a) Rita tabell, undersök RWM, IO och ROM-modulerna

			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
RWM	Början	0000	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Slut	1FFF	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1
Spegling, upp till 7FFF			0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
I/O	Början	A000	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
	Slut	A07F	1	0	1	0	0	0	0	0	0	1	1	1	1	1	1	1
ROM	Början	C000	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Slut	FFFF	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

RWM1: [A12,A0] direkt till RWM-modul ⇒ 13 Adressbitar ⇒ 2^{13} byte ⇒ 8kbyte
IO: [A6,A0] direkt till IO-modul ⇒ 7 Adressbitar ⇒ 2^7 byte ⇒ 128byte
ROM: [A13,A0] direkt till ROM-modul ⇒ 14 Adressbitar ⇒ 2^{14} byte ⇒ 16kbyte

ROM-modulen är 16 kByte, RWM-modulen 8 kByte och IO-modulen 128byte stor. **(1p)**

b)

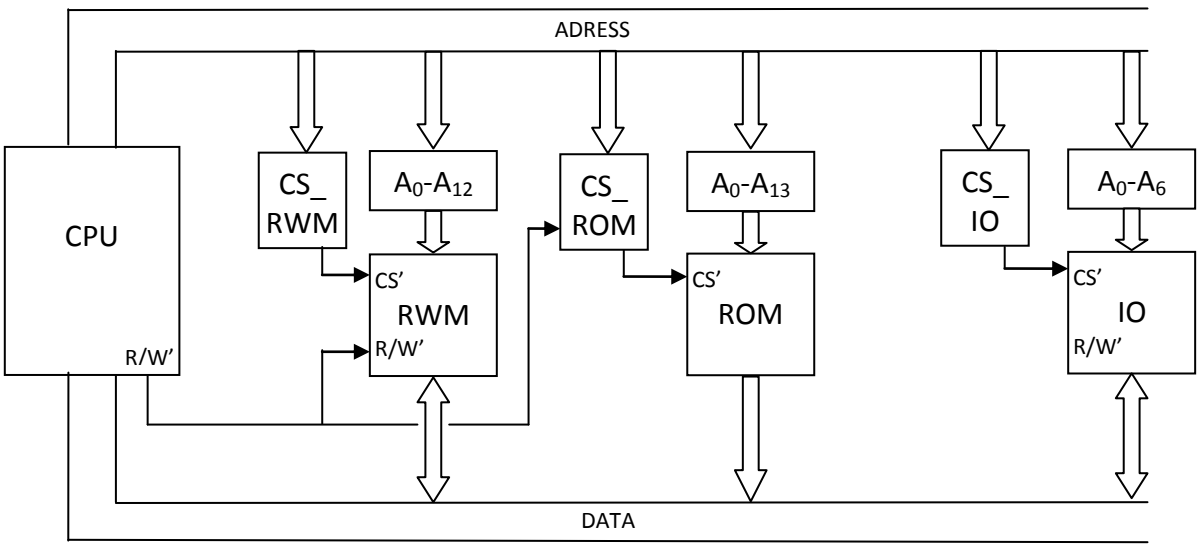
ROM: Av tabellen ser vi att A_{15} och A_{14} är konstanta och ingår därför i cs-logiken.
CS_ROM = $[A_{15}\bullet A_{14}\bullet E\bullet (R/W')\ ]'$

IO: Av tabellen ser vi att A_{15} och A_7 är konstanta och ingår därför i cs-logiken.
CS_IO = $[A_{15}\bullet A_{14}'\bullet A_{13}\bullet A_{12}'\bullet A_{11}'\bullet A_{10}'\bullet A_9'\bullet A_8'\bullet A_7'\bullet E]'$ **(2p)**

c)

Studera figuren i tesen och tabellen ovan där RWM-modulen speglas ända upp till 7FFF. Vi kan utläsa att A_{15} är konstant (= 0) för samtliga speglade adressintervall. A_{15} ingår därför i cs-logiken. är enda villkoret för CS-signalen för RWM-modulen.
CS_RWM = $[A_{15}'\bullet E]'$ **(2p)**

d)



(2p)

Uppgift 2

a) **(4p)**

```
IrqRut:
     LDAA       IrqStat     ; Läs statusflaggorna
     BITA       #%1111      ; Någon yttre enhet?
     BNE        Yttre
     JSR        ClkIrq      ; Serva Klockan
     BRA        IrqExit     ; .. och avsluta

Yttre:
     LSRA                   ; Enhet 0
     BCC        EjB0
     JSR        IrqRut0     ; Serva enhet 0
     BRA        IrqExit
EjB0
     LSRA
     BCC        EjB1
     JSR        IrqRut1     ; Serva enhet 1
     BRA        IrqExit
EjB1
     LSRA
     BCC        EjB2
     JSR        IrqRut2     ; Serva enhet 2
     BRA        IrqExit
EjB2
     JSR        IrqRut3     ; Serva enhet 3

IrqExit
     RTI
```

b)

Som det fungerar nu så detekteras avbrott från klockan när det inte är andra aktiva avbrott samtidigt. Inträffar yttre avbrott med så hög frekvens så att deras avbrottsubrutiner exekverar kontinuerlig i 2 ms kommer vi att tappa ett klockavbrott (ty avbrott varje ms).
Exekverar avbrottsrutinerna kontinuerlig i 3 ms tappas 2 klockavbrott. 4 ms – 3 klockavbrott etc. Det fungerar bättre om klockan har en statusvippa/register som kan läsas och kvitteras. **(4p)**

Uppgift 3

a) Stuffbitar är extrabitar som skickas på bussen för att behålla bitsynkroniseringen. **(1p)**

b) Synkron kommunikation: Dataöverföring sker vid förutbestämd tidpunkt (exvis nästa positiva flank hos systemklockan).
Asynkron kommunikation: Dataöverföring sker med hjälp av handskakningssignaler. **(2p)**

c) Minnesbandbredden. **(1p)**

d) Ett stycke kod med en ingång (en label) och en utgång (en branch). **(1p)**

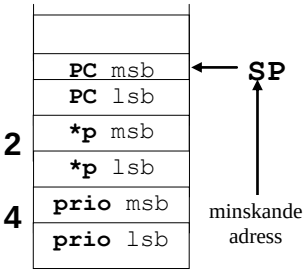
e) För att RESET-fas ska kunna utföras och initiera avbrottshantering korrekt. **(1p)**

f) Programkonstruktioner som EJ genererar kod (prototypdeklarationer, makrodefinitioner etc) placeras i `.h`. Datadeklarationer och funktionsdefinitioner placeras i `.c`. **(1p)**

Uppgift 4

a) **(2p)**

Aktiveringspost i funktionen:



b) **(1p)**

Referens p:

```
LDD 2,SP
```

Referens prio:

```
LDD 4,SP
```

Uppgift 5 (5p)

kompilatorgenererad kod

```
; void printerprint( char *s )
_printerprint:
; {
;       while( *s != EOT )
_printerprint1:
        LDAB    [2,SP]
        CMPB    #4
        BEQ     _printerprint2
;     {
;       PPORT = *s;
        STAB    $0700
;           s++;
        LDX     2,SP
        INX
        STX     2,SP
        BRA     _printerprint1
_printerprint2:
;       }
; }
        RTS
```

handoptimerad kod

```
; void printerprint( char *s )
_printerprint:
; {
;       while( *s != EOT )
        LDX     2,SP
_printerprint1:
        LDAB    1,X+    (även s++; nedan)
        CMPB    #4
        BEQ     _printerprint2
;       {
;           PPORT = *s;
        STAB    $0700
;           s++; (se ovan)
        BRA     _printerprint1
_printerprint2:
;       }
; }
    RTS
```

Uppgift 6 (10p)

```
#define MAXRAD 500
#define MAXNAMN 100
#include <stdio.h>
#include <string.h>
int main() {
  char namn[MAXNAMN+1], rad[MAXRAD+2];
  int n;
  FILE *fin;
  printf("Filens namn? ");
  scanf("%s", namn);
  fin = fopen(namn, "r");
  if (!fin){
    fprintf(stderr, "Filen %s kan ej öppnas.", namn);
    exit(1);
  }
  while (fgets(rad, MAXRAD, fin)) {
    if (strspn(rad, " \t\n") != strlen(rad))
      fputs(rad, stdout);
  }
}
```

Uppgift 7 (10p)

```
#include<stdio.h>

size_t strspn(const char* cs, const char* ct) {
        size_t n;
        const char* p;
        for(n=0; *cs; cs++, n++) {
          for(p=ct; *p && *p != *cs; p++)
            ;
          if (!*p)
            break;
        }
        return n;
}
```