EDA485 Maskinorienterad programmering Z

DAT015 Maskinorienterad programmering IT

Tentamen

Tisdag 18 april 2006, kl. 8.30 - 12.30 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 4-5). 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. Konstruera adressavkodningen 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 skall ha sin sista adress på $FFFF och är 8-kbyte stor. RWM-modulen som är 32-kbyte skall placeras med start på adress $0000. 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. **(8p)**

2. *Avbrott och assemblerprogrammering*. Ett HC12-system är bestyckad med en pulsgenerator som genererar avbrott var 10:de ms.

Du behöver en rutin (IRQINIT) som initierar systemet och en avbrottsrutin (IRQ) som uppdaterar en klock-variabel. Klockvariabeln skrivs till en display av huvudprogrammet. När programmet startar skall displayen visa (börja på) 00:00:00. Avbrott kvitteras genom en skrivning på den symboliska adressen IRQRES.

Initieringsrutinen (IRQINIT): Skriv en rutin som initierar 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 som alla tre lagras på NBCD-form.

Du får själv skapa ytterligare hjälpvariabler för klockavbrotten **(8p)**

3. Småfrågor.

a. I OH-serien över CAN-protokollet angavs 9 bra egenskaper som CAN-protokollet har. Ange 4 av dessa egenskaper. **(2p)**

b. "Deadline" är ett vanligt begrepp för ett realtidssystem och det beskrivs med en värdefunktion. Vad menar man med en ”deadline’s värdefunktion”? **(3p)**

c. Vad menas ”busy wait” och ”polling”. Ange fördelar och nackdelar. Rita även flödesplaner och ge exempel på programkod i ditt svar. **(4p)**

d. 2-pass-assemblatorn översätter assemblerprogram till maskinkod. Vad utförs i pass 1 respektive pass 2? **(2p)**

e. Assemblerinstruktionen **BRSET** fungerar inte i ETERM-simulatorn för HC12 pga en ”bugg”. Ange en instruktionssekvens som utför **BRSET** för att ”komma runt” problemet. **(3p)**

4

I C-standarden beskrivs standardfunktionen `strstr` på följande sätt:

```
char *strstr(const char *s1, const char *s2);
```

Description

The `strstr` function locates the first occurrence in the string pointed to by `s1` of the sequence of characters (excluding the terminating null character) in the string pointed to by `s2`.

Returns

The `strstr` function returns a pointer to the located string, or a null pointer if the string is not found. If `s2` points to a string with zero length, the function returns `s1`.

Skriv en egen version av funktionen `strstr`. Du får *inte* använda dig av någon annan standardfunktion, utan måste skriva allt själv. (Bortse från att det egentligen är felaktigt att returnera s1 eftersom den har typen **`const char *`** medan returvärdet har typen **`char *`**.)

(9 p)

5

Deluppgift a.

Konstruera en C-funktion `monitor` som har till uppgift att kontinuerligt, med jämna tidsmellanrum, kontrollera att en viss aktivitet i ett realtidssystem fungerar som den skall. Funktionen `monitor` skall ha tre parametrar. Den första skall vara en heltalsvariabel av typen `time_type`. Denna parameter anger hur ofta kontrollen skall ske. Enheten är millisekunder. Den andra parametern skall vara en pekare till en testfunktion som skall anropas vid varje kontrolltillfälle. Denna funktion skall vara parameterlös och som retur skall den ge ett sanningsvärde (en **`int`**) som indikerar om den aktivitet som testas fungerar som den skall eller inte. (Värdet **`true`** betyder att den fungerar och **`false`** att den inte fungerar.) Den tredje och sista parametern skall vara en pekare till en parameterlös funktion som saknar returvärde. Denna funktion är en alamfunktion som skall anropas om det vid ett kontrolltillfälle visar sig att den testade aktiviteten inte fungerar.

I denna deluppgift får du förutsätta att funktionen `hold` och typen `time_type` från den sista laborationen redan är definierade och kan användas.

(4 p)

Deluppgift b.

Antag att en dator används i ett patientövervakningssystem. En sensor känner av patientens puls och ger ett avbrott till datorn vid varje hjärtslag. Adressen till avbrottsvektorn är FF90 (hex). Sensorn styrs med hjälp av ett 8-bitars register på adressen 0D00. Sensorn startas och stängs av genom att bit nr 0 i registret sätts till 1 resp. 0. När ett hjärtslag inträffar sätter sensorn bit nr 1 i registret till 1. Om man har satt bit nr 6 i registret till 1 genererar sensorn även en avbrottssignal och sätter bit nr 7 till 1. Sensorn kan återställas efter ett avbrott genom att man sätter bit 7 till 0. (Detta nollställer även bit nr 1.)

Systemet tillåter även att man vid behov ger patienten en lätt elektrisk stöt för att stimulera hjärtverksamheten. Detta styrs från datorn via ett 8-bitars register på adressen 0D02. I bitarna 7-4 anger man hur många volt (mellan 0-15 volt) den elektriska stöten skall ha och för att generera själva stöten skriver man en etta till bit nummer 0.

Skriv i C ett program som övervakar patienten. Programmet skall en gång var 5:e sekund kontrollera att patientens hjärta slagit under de senaste 5 sekunderna. Om så inte är fallet skall patienten ges en elektrisk stöt på 6 volt. Dessutom skall alarm ges till sjukvårdspersonalen genom att en färdigskriven parameterlös funktion `call_nurse` anropas.

Du *måste* använda dig av funktionen `monitor` från deluppgift a. (Det måste du göra även om du inte löst den deluppgiften.) Du får använda dig av alla de makrodefinitioner du skrev i den sista laborationen. Du får också förutsätta att avbrottrutinen för att hantera realtidsklockan samt funktionen `init_clock` från laborationen finns. Din uppgift är alltså att skriva `main`-funktionen och de funktioner som initierar sensorn och hanterar avbrott från den. Du måste också skriva de två funktioner som skall anropas från funktionen `monitor`. För enkelhets skull får du anta att det finns en färdig avbrottsrutin `heart_trap` skriven i assembler. Det enda denna funktion gör är att anropa en funktion med namnet `heart_inter`. Du måste alltså själv skriva funktionen `heart_inter`. Du kan däremot inte förutsätta att adressen till avbrottsrutinen är inlagd i avbrottsvektorn. Det måste du göra själv. Du får förutsätta att alla styrregister är både läs- och skrivbara så att det inte är nödvändigt att använda skuggregister.

(7 p)

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