



Tentamen med lösningsförslag

EDA482 Maskinorienterad programmering D

EDA483 Maskinorienterad programmering E

DAT017 Maskinorienterad programmering IT

DIT153 Maskinorienterad programmering GU

Fredag 23 augusti 2024, kl. 8.30 - 12.30

Examinatorer

Roger Johansson, tel. 772 57 29

Erik Sintorn, tel. 772 52 29

Kontaktpersoner under tentamen:

Roger Johansson, tel. 772 57 29

Erik Sintorn, tel. 772 52 29

Tillåtna hjälpmedel

Utgåvor som distribuerats inom ramen för kursen, häftet:

- *Quick Guide, Laborationsdator MD407 med tillbehör*

Inget annat än understrykningar ("överstrykningar") får vara införda i detta häfte.

Tabellverk eller miniräknare får ej användas.

Lösningar

Anslås senast dagen efter tentamen via kursens hemsida.

Granskning

Tid och plats anges på kursens hemsida.

Allmänt

Uppgifter 1 t.o.m 3 utgörs av flervalsfrågor.

Försök först lösa uppgifterna, så som du lärt dig i kursen, jämför sedan med flervalsalternativen och ange det alternativ du anser är korrekt. Avge dina flervalssvar på blanketten, längst bak i tentamens-tesen. Skicka med denna blankett med dina övriga lösningar. Högst ett svarsalternativ får anges i varje uppgift. Siffror inom hakparenteser anger möjligt intervall för poängbedömning.

Observera att, bland svarsalternativen finns alternativ som *kan* ge poängavdrag. Om du är mycket osäker på ditt svar bör du därför i stället välja att avstå.

För övriga uppgifter gäller:

Svar kan avges på svenska eller engelska.

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 getts under kursen.
- C-program är utformade enligt de råd och anvisningar som getts under kursen. I programtexterna skall raderna dras in så att man tydligt ser programmets struktur.

Betygsättning

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

Maximal poäng är 50 och tentamenspoäng ger slutbetyg enligt:

$20p \leq \text{betyg } 3 < 30p \leq \text{betyg } 4 < 40p \leq \text{betyg } 5$

Uppgift 1 (-1..4p)

Följande deklarationer är givna på "toppnivå".

```

short a;
short b(int,unsigned short);
int c;
unsigned short d(long);
long e;

```

Vi har nu tilldelningen:

```
a = b ( c, d(e) );
```

Vilken av följande implementeringar i ARM v6-kod är korrekt?

A	B	C	D
<pre> LDR R0,=e LDR R0,[R0] BL d LDR R0,=c LDR R0,[R0] BL b LDR R1,=a STRH R0,[R1] </pre>	<pre> LDR R0,=e LDR R0,[R0] BL d UXTH R0,R0 LDR R0,=c LDR R0,[R0] BL b LDR R1,=a STRH R0,[R1] </pre>	<pre> LDR R0,=e LDR R0,[R0] BL d MOV R1,R0 LDR R0,=c LDR R0,[R0] BL b LDR R1,=a STRH R0,[R1] </pre>	<pre> LDR R0,=e LDR R0,[R0] BL d UXTH R0,R0 MOV R1,R0 LDR R0,=c LDR R0,[R0] BL b SXTH R0,R0 LDR R1,=a STRH R0,[R1] </pre>
E	F	G	H
<pre> LDR R0,=e LDR R0,[R0] BL d UXTH R0,R0 MOV R1,R0 LDR R0,=c LDR R0,[R0] BL b SXTH R0,R0 LDR R1,=a STR R0,[R1] </pre>	<pre> LDR R0,=e LDR R0,[R0] BL d MOV R1,R0 LDR R0,=c LDR R0,[R0] BL b LDR R1,=a STR R0,[R1] </pre>	<pre> LDR R0,=e LDR R0,[R0] BL d UXTH R0,R0 LDR R0,=c LDR R0,[R0] BL b SXTH R0,R0 LDR R1,=a STR R0,[R1] </pre>	<pre> LDR R0,=e LDR R0,[R0] BL d LDR R0,=c LDR R0,[R0] BL b SXTH R0,R0 LDR R1,=a STR R0,[R1] </pre>

Uppgift 2 (-1..6p)

En C-funktion `contain` är definierad enligt listning till höger.

Ange det alternativ (A-H) som visar en korrekt implementering (dvs. hur funktionen ska kodas) i ARM v6 assemblerspråk.

```
int contain( char *s, char c )
{
    while( *s )
    {
        if( *s == c )
            return 1;
        s++;
    }
    return 0;
}
```

A	B	C	D
<pre>contain: B contain1 contain2: CMP R2,R1 BEQ contain3 ADD R0,R0,#1 B contain1 contain3: MOV R0,#1 BL LR contain1: LDR R2,[R0] CMP R2,#0 BNE contain2 MOV R0,#0 BX LR</pre>	<pre>contain: B contain1 contain2: CMP R2,R0 BEQ contain3 ADD R0,R0,#1 B contain1 contain3: MOV R0,#1 BL LR contain1: LDR R2,[R0] CMP R2,#0 BNE contain2 MOV R0,#0 BX LR</pre>	<pre>contain: B contain1 contain2: CMP R2,R0 BEQ contain3 ADD R0,R0,#1 B contain1 contain3: MOV R0,#1 BX LR contain1: LDRB R2,[R0] CMP R2,#0 BNE contain2 MOV R0,#0 BX LR</pre>	<pre>contain: B contain1 contain2: CMP R2,R1 BEQ contain3 ADD R0,R0,#1 B contain1 contain3: MOV R0,#1 BX LR contain1: LDRB R2,[R0] CMP R2,#0 BNE contain2 MOV R0,#0 BX LR</pre>
E	F	G	H
<pre>contain: B contain1 contain2: CMP R2,R1 BEQ contain3 B contain1 contain3: MOV R0,#1 BX LR contain1: LDR R2,[R0] CMP R2,#0 BNE contain2 MOV R0,#0 BX LR</pre>	<pre>contain: B contain1 contain2: CMP R2,R0 BEQ contain3 B contain1 contain3: MOV R0,#1 BX LR contain1: LDR R2,[R0] CMP R2,#0 BNE contain2 MOV R0,#0 BX LR</pre>	<pre>contain: B contain1 contain2: CMP R2,R0 BEQ contain3 B contain1 contain3: MOV R0,#1 BL LR contain1: LDRB R2,[R0] CMP R2,#0 BNE contain2 MOV R0,#0 BX LR</pre>	<pre>contain: B contain1 contain2: CMP R2,R1 BEQ contain3 B contain1 contain3: MOV R0,#1 BL LR contain1: LDRB R2,[R0] CMP R2,#0 BNE contain2 MOV R0,#0 BX LR</pre>

Uppgift 3 (-2..6p)

Konstruera en C-funktion som undersöker en parameter med avseende på antalet 0-ställda bitar. Funktionen deklarerar:

```
short bitcheck( unsigned int *p, int * num );
```

- `p` är en pekare till det värde som ska undersökas
- `num` är en pekare till en plats dit antalet 0-ställda bitar hos parametern `p`, ska skrivas

Returvärdet för funktionen ska vara skilt från 0 om antalet nollor hos parametern är jämt delbart med 2, annars ska returvärdet vara 0.

Sex olika programmerare får i uppgift att implementera funktionen. Resultaten blir likartade men bara en programmerare presterar ett helt korrekt resultat. De olika alternativen finns nedan, ange alternativet med den korrekta lösningen.

A	B
<pre>short bitcheck(unsigned int *p, int *num) { num = 0; while(*p) { if(*p) (num)++; *p >>= 1; } num = (short) ((sizeof(int) * 8) - num); return !(num & 1); }</pre>	<pre>short bitcheck(unsigned int *p, int *num) { num = 0; while(p) { if(p & 1) (num)++; p >>= 1; } num = (short) ((sizeof(int) * 8) - num); return !(num & 1); }</pre>
C	D
<pre>short bitcheck(unsigned int *p, int *num) { *num = 0; while(*p) { if(*p) (*num)++; *p >>= 1; } *num = (short) ((sizeof(int)) - *num); return !(*num & 1); }</pre>	<pre>short bitcheck(unsigned int *p, int *num) { *num = 0; while(*p) { if(*p) (*num)++; *p >>= 1; } *num = (short) ((sizeof(int)) - *num); return !(*num & 1); }</pre>
E	F
<pre>short bitcheck(unsigned int *p, int *num) { *num = 0; while(*p) { if(*p & 1) (*num)++; *p >>= 1; } *num = (short) ((sizeof(int) * 8) - *num); return !(*num & 1); }</pre>	<pre>short bitcheck(unsigned int *p, int *num) { num = 0; while(p) { if(p & 1) (num)++; p >>= 1; } num = (short) ((sizeof(int)) - num); return !(num & 1); }</pre>

Uppgift 7 (6p)

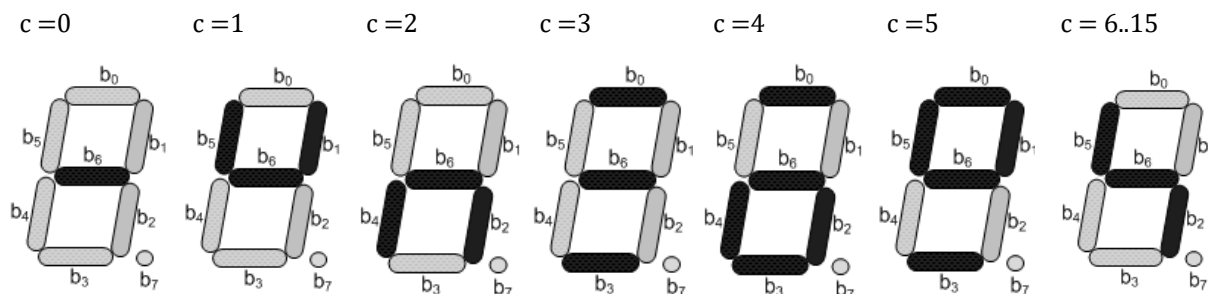
En sjusegmentsdisplay är ansluten till GPIO D (bit 7..bit 0) till MD407.

Utskrift sker med en funktion: `void outFig( unsigned char c );`

Parameter `c` anger vad som ska skrivas ut.

- Om parametern är i intervallet 0..15 ska motsvarande figur (se nedan) skrivas ut till displayen.
- Om parametern är otillåten (större än 15) ska displayen släckas.

Följande figurer ska genereras vid utskrift:



- Visa en funktion `void init_app(void)` som initierar port D för användning med displayen. Hänsyn behöver inte tas till eventuell användning av portens övriga bitar (1p).
- Visa funktionen `outFig(5p)`.

Följande makrodefinitioner kan förutsättas givna som pekare till respektive register.

`GPIO_D_MODER` , `GPIO_D_OTYPER`, `GPIO_D_ODRLOW`

Uppgift 8 (6p)

Visa en enkel implementering av USART-rutiner ("polling") för MD407. Funktionerna specificeras enligt följande:

```
void _iniuart( unsigned short baud)
parameter baud används för att initiera brr.
```

```
char _tstchar( void )
```

Funktionen `_tstchar` kontrollerar först om ett tecken anlänt till seriekretsen. Om så är fallet, returneras detta tecken av funktionen. Om inget tecken anlänt returneras 0.

Utmatningsrutinen `_outchar`, väntar tills sändarregistret är tomt, därefter skrivs tecknet till seriekretsens dataregister.

```
void _outchar( char c )
```

Du får anta att en datastruktur `struct usart` finns given (utformad enligt QuickGude). Skriv koden för USART1.

Uppgift 9 (4p)

- Varför används *kompilatorbibliotek* i GCC? (2p).
- Beskriv kortfattat, gärna med en figur, hur minnet används av en enkel applikation med dynamisk minneshantering (`malloc/free`). (2p)

Svarsblankett

Anonym kod

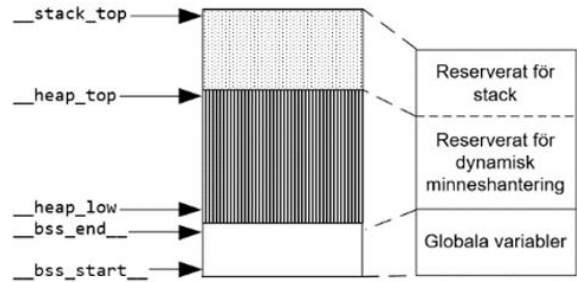
Maskinorienterad programmering, tentamen flervalsfrågor.

För varje fråga, kryssa i det alternativ du anser vara korrekt, eller avstå.
Lämna in denna blankett tillsammans med dina övriga lösningar.

Fråga	A	B	C	D	E	F	G	H
1	0	0	1	4	1	0	0	-1
2	0	0	2	6	0	0	-1	0
3	0	-1	2	1	6	-1		

Lösningsförslag

Uppgift 1 (-1..4p) Svarsalternativ D <pre> LDR R0,=e @ R0<-&e LDR R0,[R0] @ R0<-e BL d @ R0<-d(e) UXTH R0,R0 @ R0<-(unsigned short)d(e) MOV R1,R0 @ R1<-d(e) LDR R0,=c @ R0<-&c LDR R0,[R0] @ R0<-c BL b @ R0<-b(c,d(e)) SXTH R0,R0 @ R0<-b(c,d(e)) (1 LDR R1,=a @ R1<-&a STRH R0,[R1] @ a= b(c,d(e)) </pre> 1) REDUNDANT since only 16 bits are used in the assignment but formally ok.	Uppgift 2 (-1..6p): Svarsalternativ D <pre> @ int contain(char *s, char c) @ R0: parameter s @ R1: parameter c @ R0: Returvärde contain: B contain1 @ while(*s) contain2: CMP R2,R1 BEQ contain3 @ (*s==c)? ADD R0,R0,#1 @ s++ B contain1 contain3: MOV R0,#1 BX LR @ return 1 contain1: LDRB R2,[R0] CMP R2,#0 BNE contain2 @ (*s==0)? MOV R0,#0 BX LR @ return 0 </pre>
Uppgift 3 (-2..6p): Svarsalternativ E <pre> short bitcheck(unsigned int *p, int *num) { /* Vi räknar ettorna, det är enklast... */ *num = 0; while(*p) { if(*p & 1) (*num)++; *p >>= 1; } /* Antal nollor */ *num = (short) ((sizeof(int) * 8) - *num); return !(*num & 1); } </pre>	Uppgift 4a (2p) <pre> #define ctrl ((unsigned char *) 0xFF600000) #define ivr ((unsigned long *) 0xFF60000C) *(ctrl) = ...; </pre> Uppgift 4b (3+1p) <pre> typedef struct{ volatile unsigned char ctrl; volatile unsigned char status; short unused0; volatile unsigned short channel; short unused1; volatile unsigned int data; volatile unsigned int ivr; } SIGUNIT; ((SIGUNIT *) 0xFF600000)->data; </pre> Uppgift 4b (2p) <pre> #define RS (1<<0) #define EN (1<<3) #define IA (1<<6) #define IE (1<<7) </pre>
Uppgift 5 (5p) <pre> #define SYSTICK_IRQVEC \ ((volatile unsigned int *)0x2001C03C) void systick_init(void) { /* SystemCoreClock = 168000000 */ *SYSTICK_IRQVEC = systick_irq_handler; *STK_CTRL = 0; *STK_LOAD = (168000-1); *STK_VAL = 0; *STK_CTRL = 7; } </pre>	
Uppgift 6 (5p) <pre> *SYSCFG_EXTICR4 &= 0x0FFF; /* nollställ bitfält EXTI15 */ *SYSCFG_EXTICR4 = 0x3000; /* PD15->EXTI15 */ *EXTI_IMR = (1<<15); /* aktivera avbrott EXTI15 */ *EXTI_FTSR = (1<<15); /* aktivera trigger på negativ flank */ *EXTI_RTSR &= ~(1<<15); /* deaktivera trigger på positiv flank */ *NVIC_ISER1 = (1<<8); /* aktivera avbrott i NVIC */ </pre> Vektor nummer 40 (offset 0xE0)	

Uppgift 7a (1p) <pre>void init_app(void) { *GPIO_D_MODER = 0x00005555; *GPIO_D_OTYPER = 0x00000000; *GPIO_D_ODRLow = 0; }</pre> Uppgift 7b (5p) <pre>void outFig(unsigned char c) { unsigned char code; unsigned char segCode[]={ 0x40,0x62,0x54,0x49,0x5D,0x6B }; if(c >= 6){ code = 0x64; }else code = segCode[c]; *GPIO_D_ODRLow = code; }</pre>	Uppgift 8 (2+2+2=6p) <pre>#define USART1 ((struct usart *) 0x40011000) #define BIT_UE (1<<13) #define BIT_TE (1<<3) #define BIT_RE (1<<2) void _iniuart(unsigned short baud) { USART1->brr = baud; USART1->cr2 = 0; USART1->cr1 = BIT_UE BIT_TE BIT_RE; } #define BIT_RXNE(1<<5) #define BIT_TXE (1<<7) char_tstchar(void) { if((USART1->sr & BIT_RXNE)== BIT_RXNE) return (char) USART1->dr; return 0; } void_outchar(char c) { while ((USART1->sr & BIT_TXE)==0); USART1->dr = (unsigned short) c; }</pre>
Uppgift 9a (2p) För att tillhandahålla mjukvaruimplementerade funktioner av operationer som inte stöds med maskininstruktioner för den aktuella arkitekturen.	Uppgift 9b (2p)  The diagram illustrates the memory layout of a program. It shows a vertical stack of memory regions. From top to bottom, the regions are: Stack: Indicated by a dashed line and labeled "Reserverat för stack". It is bounded by <code>__stack_top</code> at the top and <code>__heap_top</code> at the bottom. Dynamic Memory: Indicated by a dashed line and labeled "Reserverat för dynamisk minneshantering". It is bounded by <code>__heap_low</code> at the top and <code>__bss_end</code> at the bottom. Global Variables: Indicated by a dashed line and labeled "Globala variabler". It is bounded by <code>__bss_start</code> at the top and <code>__bss_end</code> at the bottom. Arrows point to the boundaries of these regions: <code>__stack_top</code>, <code>__heap_top</code>, <code>__heap_low</code>, <code>__bss_end</code>, and <code>__bss_start</code>.