MOP Demo 4

Assemblerprogrammering

X.9

Skriv en subrutin ASCHEX i assemblerspråk som översätter ett 7-bitars ASCII-tecken för en hexadecimal siffra (0-9 eller A-F) till motsvarande binära tal (00000000)2-(00001111)2. Vid anrop av subrutinen innehåller register A ASCII-tecknet i bitarna b6-b0 medan b7 är en checkbit med okänt innehåll. Vid återhopp skall det binära talet finnas i register A. Om innehållet i register A vid anropet ej är ASCII-tecknet för en hexadecimal siffra så skall talet (11111111)2 finnas i register A vid återhopp. Endast register A och register CC får vara förändrade vid återhopp från subrutinen.

Lösning: X.9

Analys av problemet: Ett ”klassiskt” kodkonverteringsproblem, omvandla ”ASCII”- till HexaDecimala- tecken. Ett beprövat sätt att lösa detta är att använda det aritmetiska sambandet mellan ASCCI och HexaDecimala siffror/tecken. Genom att titta i kodlistan för ASCII ser man att ASCCI värdet för 1 ges av 30HEX + 1, ASCII för A ges av 37HEX+AHEX.

```
ASCHEX   ANDA    #$7F     Nollställ bit 7

         SUBA    #$30     Bilda binärtal om ASCII-tecken för decimal siffra
         BLO     FEL      Ej hexsiffra
         CMPA    #9       Decimal siffra?
         BLS     EXIT     Siffra 0-9

         SUBA    #$07     Bilda tal för hexsiffra A-F
         CMPA    #$0A     Kolla undre gräns
         BLO     FEL      Ej hexsiffra

         CMPA    #$0F     Kolla övre gräns
         BLS     EXIT     Hexsiffra

FEL      LDAA    #$FF     Signalera felaktigt hextal med A=FFH

EXIT     RTS
```

```
*Uppgift X.10
*I ett datorsystem med processorn CPU12 behövs en subrutin, STRCONV, som
omvandlar en sträng
*med ASCII-tecknen för hexadecimala siffror i minnet till en annan sträng
med motsvarande
*8-bitars dataord. ASCII-strängen består av ett jämnt antal ASCII-tecken
med dataordet 0
*tillagt sist. Principen för omvandlingen framgår av följande exempel:
*
*ASCII-strängen 42H, 35H, 37H, 38H, 36H, 32H, 41H, 33H, 0
*ger binära strängen 10110101, 01111000, 01100010, 10100011.
*
*Skriv subrutinen STRCONV. Vid anrop av subrutinen skall adressen till
ASCII-strängen finnas
*i X-registret och adressen till den binära strängen i Y-registret. Då en
korrekt omvandling
*har genomförts skall carryflaggan nollställas före återhopp. Om ett
otillåtet ASCII-tecken
*upptäcks skall omvandlingen avbrytas och carryflaggan ettställas före
återhopp.
*Endast flaggregistret får vara förändrat vid återhopp. ASCII-tecknen är
lagrade i bit6-bit0
*på varje adress i textsträngen. Bit7 i textsträngens dataord har värdet 0.
*Textsträngen avslutas med datavärdet 0.
********************************************************************************
*************
* Huvudprogram för test av subrutin STRCONV

        ORG     $1000
        LDS     #$2FFF          BOS
        LDX     #ASCSTR         Pekare till ASCII-sträng
        LDY     #BINSTR         Pekare till binärsträng

        JSR     STRCONV         Omvandla från ASCII till binärsträng

STOP    BRA     STOP            Ok om C=0, fel annars

        ORG     $1020
ASCSTR  FCS     "CF2FFFCE100ECD104016106020FE434632464646" En felfri ASCII-
sträng
        FCB     0               Slutmarkering

        ORG     $1060
BINSTR  RMB     32              Plats för binärsträng
*
```

```
*******************************************************************************
**************
*       Subrutin STRCONV
*
        ORG     $1080
STRCONV PSHD                    Spara reg på stack
        PSHX
        PSHY

ASCLOOP LDAA    1,X+            Hämta ASCII-tecken. Öka pekare
        CLC                     Nollställ felflagga i C
        BEQ     STREX           Strängslut, avsluta

        JSR     ASCHEX          Omvandla ASCII till binär nibble
        CMPA    #$FF            Felaktig hexsiffra?
        BEQ     ERROR           Hoppa ut och sätt felflagga

        LSLA                    Flytta nibble till vänster
        LSLA
        LSLA
        LSLA
        TFR     A,B             Spara vänsterjusterad nibble

        LDAA    1,X+            Hämta nästa ASCII-tecken. Öka pekare
        JSR     ASCHEX          Omvandla ASCII till binär nibble
        CMPA    #$FF            Felaktig hexsiffra?
        BEQ     ERROR           Hoppa ut och sätt felflagga

        ABA                     Ej fel, kombinera nibbles
        STAA    1,Y+            Skriv dataord i binärsträng och öka pekare
        BRA     ASCLOOP

ERROR   SEC                     Sätt felflagga
STREX   PULY                    Återställ register
        PULX
        PULD
        RTS

*******************************************************************************
***************
*       Subrutin ASCHEX
*
ASCHEX  ANDA    #$7F    Nollställ bit 7
        SUBA    #$30    Bilda binärtal om ASCII-tecken för decimal siffra
        BLO     FEL     Ej hexsiffra
        CMPA    #9      Decimal siffra?
        BLS     EXIT    Siffra 0-9
        SUBA    #$07    Bilda tal för hexsiffra A-F
        CMPA    #$0A    Kolla undre gräns
        BLO     FEL     Ej hexsiffra
        CMPA    #$0F    Kolla övre gräns
        BLS     EXIT    Hexsiffra
*
FEL     LDAA    #$FF    Signalera felaktigt hextal med A=FFH
*
EXIT    RTS
```

5. En process skall styras med hjälp av en dator med processorn CPU12. I samband med detta skall en flödesgivare som är ansluten till en av datorns inportar med den symboliska adressen SENS läsas av en gång per sekund. Det avlästa värdet är ett 8-bitars tal utan inbyggt tecken och är ett mått på flödet i en rörledning. Eftersom datorn normalt är upptagen med beräkningsarbete för processtyrningen, skall avbrott användas för avläsningen av flödesgivaren.
Det finns en binär signal med den konstanta frekvensen 250 Hz tillgänglig för generering av avbrott.
Föreslå en koppling för avbrottsgenerering på processorns IRQ'-ingång. D-vippor, NAND- och NOT-grindar får användas. Det finns inga andra avbrottskällor i systemet.
Skriv avbrottsrutinen IRQRUT som läser av flödesgivaren en gång per sekund och placerar det avlästa värdet på adressen DE00H i minnet.
Skriv ett avsnitt av huvudprogrammet där IRQ-avbrott initieras. IRQ-vektorn antas vara placerad i RWM på adresserna FFF2H och FFF3H. Avbrottsrutinen börjar på den symboliska adressen IRQRUT. Assemblerspråk för processorn CPU12 skall användas.

Analys av problemet: En yttre flödessensor skall av läsas periodiskt. Detta löser vi genom att anpassa avläsningstiden med hjälp av den yttre klockan. Klockan ger avbrott 250 ggr/sek (250 Hz). Vi kan använda en ”räknare” som räknar 250 pulser/avbrott mellan varje avläsning (en per sekund). Det avlästa värdet skall placeras på adress DEOOH. Vi skall bygga koden för klockhantering och avläsning i en avbrottsrutin som skall placeras på den symboliska adressen IRQRUT (assemblatorn ger den fysiska adressen).
Vi skall även skriva ett huvudprogram som förbereder processorn för att ta emot och tillåta avbrott.
Hårdvarulösning: Vi kan använda en D-vippa för att ”fånga” klockan och ”resetta” vippan med en ”stor” NAND-grind.

Tenta 100308. Uppgift 4:

4. Avbrottssystemet i CPU12 skall användas till att köra två olika program, ProgA och ProgB, pseudoparallellt. Programmen som utgörs av evighetsslingor har startadresserna 1000H resp. 2000H. De använder var sin stack med BOS på adresserna 2000H resp. 3000H. En händelsedetektor som genererar binärpulser skall också ge avbrott. Vid varje sådant avbrott skall en 8-bitars variabel EVENT ökas med ett.

Man har tillgång till en digital binär signal med frekvensen 100 Hz. En oanvänd inport som får användas finns på adressen INPORT. CS-signalen för inporten är inte tillgänglig.

a) Rita ett kopplingsschema som visar hur man kan generera avbrott med frekvensen 100 Hz och då pulser kommer från händelsedetektorn. Det får förutsättas att adressområdet 8000H – 8FFFH är tomt. Det finns inga övriga avbrottskällor att ta hänsyn till i systemet! Processorns IRQ'-ingång skall användas. D-vippor, NAND- och NOT-grindar får användas. **(2p)**

b) ProgA inleds med en del som bara körs vid starten och fortsätter sedan med en evighetsslinga. I den första delen av ProgA görs nödvändiga initieringar av avbrottssystemet för de två avbrottskällorna och för processbytet vid varje 100Hz-avbrott. Det förutsätts att systemet startar med reset av processorn och att ProgA då startas. Skriv programmet ProgA. Utrymme för globala variabler finns på adresserna 1E00H- 1E0FH. **(4p)**

c) Skriv avbrottsrutinen som uppdaterar variabeln EVENT och utför processbyte. **(3p)**

Alla odefinierade symboliska adresser ovan är definierade på annat ställe i programmet. Assemblerspråk för processorn CPU12 skall användas. Radkommentarer skall finnas!

2. Ett datorsystem enligt figur 1 har CS-logiken nedan.

Visa hur adressrummet utnyttjas, dvs adressintervall för samtliga enheter anslutna till bussarna.
Visa också hur man kan koppla in ytterligare en minnesmodul med 16 kByte ROM till detta system.
Vad är oftast skälet till att man i vissa delar av minnet använder RWM och i andra ROM?

2. Lösning:

ROM:	A	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
2^{14}= 16k Start: C000H =		1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Slut: FFFFH =		1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
		CS		14 st													

RWM:	A	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
2^{13}= 8k Start: 4000H =		0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Slut: 5FFFH =		0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1
		CS			13 st												

UTPORT:	A	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Adr:		0	0	0	0	0	0	0	1	-	-	-	-	-	-	-	-
										8st							
Start: 0100H =		0	0	0	0	0	0	0	1	0	0	0				0	0
Slut: 01FFH =		0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1
						CS											

Samma utport ”finns” på 2^8 olika adresser.

INPORT:	A	15	14	13	12	11	10	9	8	7	6	5	8st		1	0
Adr:		0	0	0	0	0	0	1	0	-	-	-	-	-	-	-	-
Start: 0200H =		0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
Slut: 02FFH =		0	0	0	0	0	0	1	0	1	1	1	1	1	1	1	1
						CS											

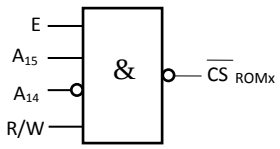
Samma inport ”finns” på 2^8 olika adresser.

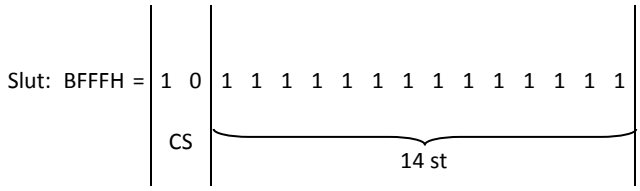
Inkoppling av ytterligare en 16k ROM-modul:

Det krävs att det finns en ledig adresslucka med konstanta $A_{15}A_{14}$ för att rymma 16k.

$A_{15}A_{14}$ = 10 är ledig.

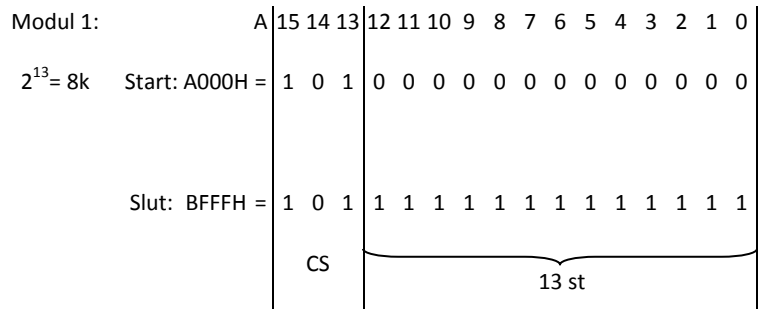
ROMx:	A	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
2^{14}= 16k Start: 8000H =		1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0



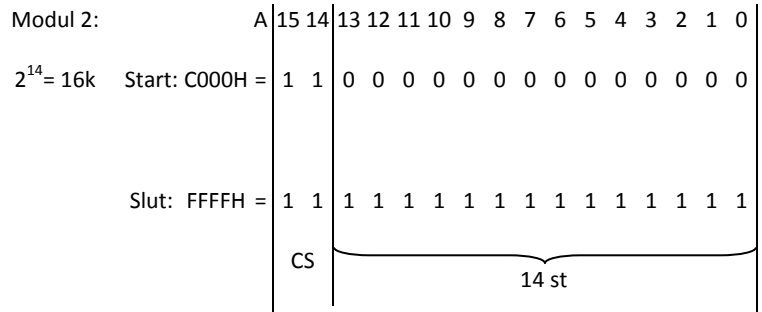


RWM används för variabler, dvs ändringsbara data och för program under utvecklingsskedet. ROM används för program som inte ändras och för konstanter. RWM är flyktigt, dvs det tappar informationen vid spänningsbortfall. ROM är icke flyktigt, dvs det behåller informationen vid spänningsbortfall.

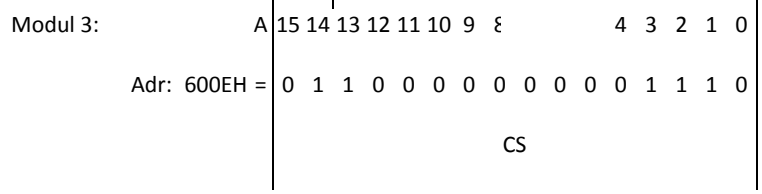
3.Lösning:



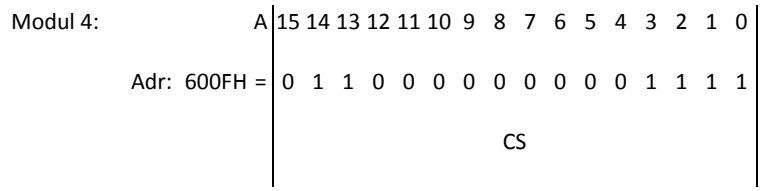
Detta bör vara en RWM-modul med kapaciteten 8kbyte. (RWM eftersom R/W-signalen saknas vid CS-avkodningen.)



Detta bör vara en ROM-modul med kapaciteten 16kbyte. (ROM eftersom R/W-signalen finns med vid CS-avkodningen.)



Detta är en inport. (Endast en adress och modulen aktiveras vid läsning i ”minnet”.)



Detta är en utport. (Endast en adress och modulen aktiveras vid skrivning i ”minnet”.)

E-signalen finns med i CS-avkodningen eftersom adressbitarna har giltiga värden endast när E = 1.