# Tentamen

## Datorteknik Y, TSEA28

| | |
|---|---|
| *Datum* | 2022-05-31 |
| *Lokal* | FE241, TER1, TER2 |
| *Tid* | 14-18 |
| *Kurskod* | TSEA28 |
| *Provkod* | TEN1 |
| *Kursnamn*<br>*Provnamn* | Datorteknik Y<br>Skriftlig tentamen |
| *Institution* | ISY |
| *Antal frågor* | 6 |
| *Antal sidor (inklusive denna sida)* | 6 |
| *Kursansvarig* | Kent Palmkvist, 1347, Kent.Palmkvist@liu.se |
| *Lärare som besöker skrivsalen*<br>*Telefon under skrivtiden* | Kent Palmkvist<br>1347, 013-281347 |
| *Besöker skrivsalen* | Ca kl 15 och kl 17 (+/- 30 minuter) |
| *Kursadministratör* | |
| *Tillåtna hjälpmedel* | Inga |
| *Betygsgränser* | Preliminärt 0-20: U, 21-30: 3, 31-40: 4, 41-50: 5 |
| *Visning* | 17/6 kl 13-14 i examinators kontor |

## Viktig information

- Hexadecimala värden anges antingen som 0x1234 eller $1234
- För de uppgifter där du måste göra uträkningar är det viktigt att du redovisar alla relevanta mellansteg
- I de uppgifter där du ska skriva assembler- eller mikrokod är det viktigt att du kommenterar koden
- Om du i en viss uppgift ska förklara något, tänk på att du gärna får rita figurer för att göra din förklaring tydligare
- Uppgifterna i tentan är inte ordnade efter svårighetsgrad
- Skriv inga svar i detta häfte!

## Lycka till!

## Fråga 1: Mikroprogrammering (10p)

I figur 1 återfinns en bekant datormodell som du ska mikroprogrammera i denna uppgift.

Jämfört med den modell ni sett tidigare har följande förändring gjorts: Mikroprogrammeringsordet har utökats med ytterligare en bit (kallad R). Om R=0 fungerar datorn precis som tidigare. Om R=1 sätts styrsignaler som styr multiplexern vid GR0-GR3 till 3 (dvs register GR3 väljs), oavsett vad IR innehåller.

Notera att det är viktigt att du skriver vilken additionsoperation du valt (den som påverkar flaggorna eller den som inte påverkar flaggorna). (Om du inte skriver något speciellt kommer jag att anta att du ej vill modifiera flaggorna).

Mikrokodsminnnet innehåller i denna datormodell bland annat

| addr | Mikrokod | Kommentar |
|---|---|---|
| 0 | ASR := PC, uPC := uPC+1 | |
| 1 | IR := PM, PC := PC+1, uPC := uPC+1 | |
| 2 | uPC := K2(M) | |
| 3 | ASR := IR, uPC := uPC+1 | ; direktadressering (M=00) |
| 4 | uPC := K1(OP) | |
| 5 | ASR := PC, uPC := uPC + 1 | ; omedelbar adressering (M=01) |
| 6 | PC := PC + 1, uPC := K1(OP) | |
| 7 | ASR := IR, uPC := uPC+1 | ; indirekt adressering (M=10) |
| 8 | ASR := PM, uPC := K1(OP) | |
| 9 | AR := IR, uPC := uPC+1 | ; indexerad adressering (M=11) |
| 0xA | AR := GR3+AR, uPC := uPC+1, R:=1 | |
| 0xB | ASR := AR, uPC := K1(OP) | |

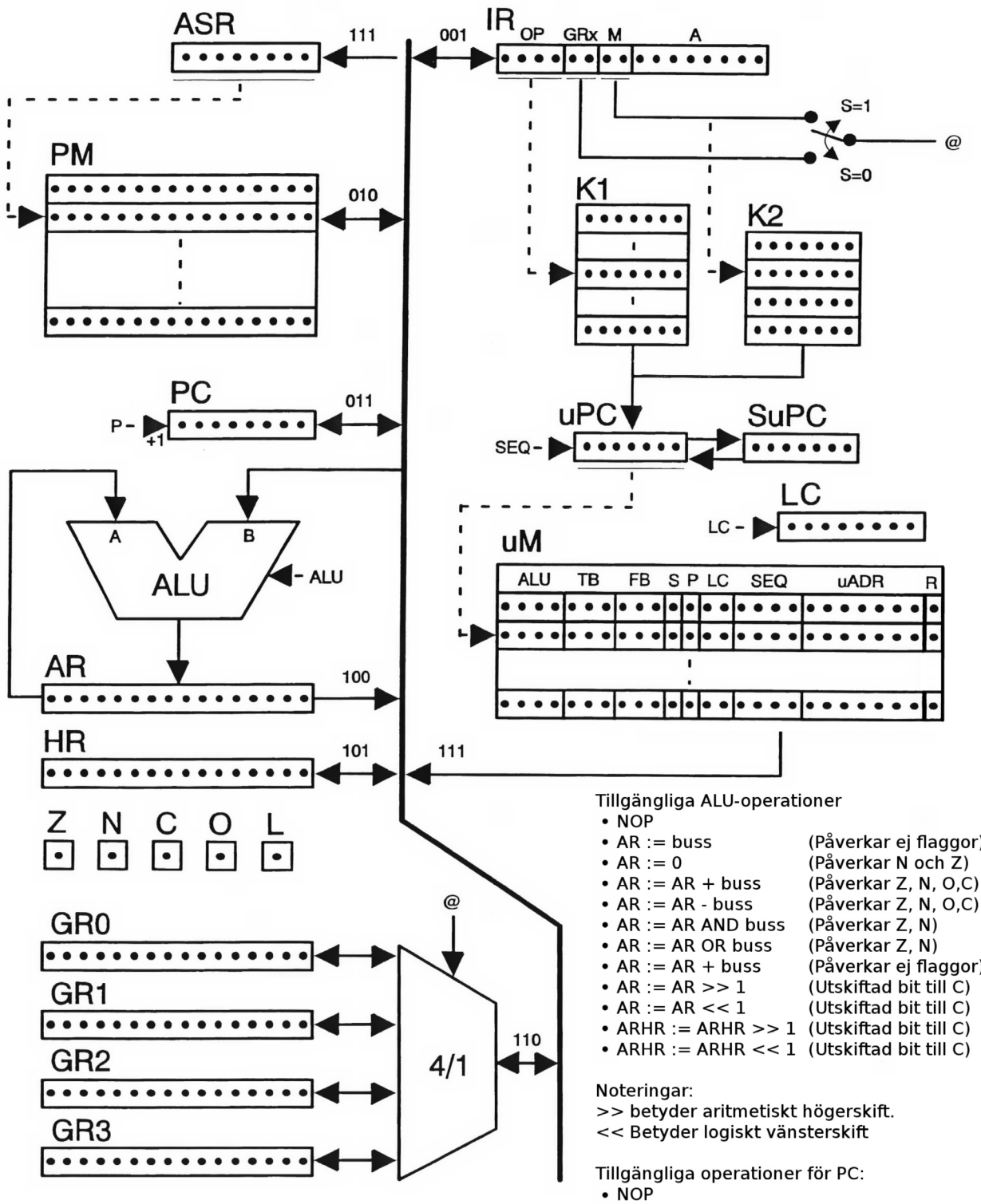
(a) (8p) Skriv mikrokod för instruktionen MAXGR3 GRx,M,A. Instruktionen ska jämföra GRx med värdet i PM på den adress som definieras av adresseringsmode (alla adresseringsmoder ska stödjas), och det största av dessa två tal ska placeras i GR3. Indata antas vara 16-bitars tal i två-komplementsform. Flaggor får påverkas.

Ange även adresserna för mikrokodsinstruktionerna som absoluta tal.

**Exempel1:** GR1 = 0x0123, PM(0x20)=0x4321. När instruktionen MAXGR3 GR1, 00,0x20 körs ska resultatet bli GR3 = 0x4321.

**Exempel2:** GR2 = 0x5678, PM(0x22)=0x0033, PM(0x33)=0x9809. När instruktionen MAXGR3 GR2, 10,0x22 körs ska resultatet bli GR3=0x5678.

(b) (2p) Ange innehållet i K2 i binär form, med binär representation av både adress och data.




Tillgängliga ALU-operationer
- NOP
- AR := buss (Påverkar ej flaggor)
- AR := 0 (Påverkar N och Z)
- AR := AR + buss (Påverkar Z, N, O,C)
- AR := AR - buss (Påverkar Z, N, O,C)
- AR := AR AND buss (Påverkar Z, N)
- AR := AR OR buss (Påverkar Z, N)
- AR := AR + buss (Påverkar ej flaggor)
- AR := AR >> 1 (Utskiftad bit till C)
- AR := AR << 1 (Utskiftad bit till C)
- ARHR := ARHR >> 1 (Utskiftad bit till C)
- ARHR := ARHR << 1 (Utskiftad bit till C)

Noteringar:
>> betyder aritmetiskt högerskift.
<< Betyder logiskt vänsterskift

Tillgängliga operationer för PC:
- NOP
- PC := PC + 1 (PC räknas upp med ett)
- PC := buss

Tillgängliga operationer för LC:
- NOP
- LC räknas ned med ett
- LC laddas från uADR

Tillgängliga operationer för uPC:
- uPC := uPC + 1 (uPC räknas upp med ett)
- uPC := K1(OP)
- uPC := K2(M)
- uPC := 0
- uPC := uADR
- uPC := uADR om (valfri) flagga är 1, annars uPC+1
- uPC := uADR om (valfri) flagga är 0, annars uPC+1

Under varje klockcykel kan även en av följande enheter skicka data från bussen: IR, PM, PC, AR, HR, GRx eller uM. En av följande enheter kan också ta emot data ifrån bussen varje klockcykel: IR, PM, PC, AR, HR, GRx eller ASR Viktigt: När uM (de 16 minst signifikanta bitarna) skickas ut till bussen kan enbart en ALU-operation och/eller en bussöverföring göras. uPC räknas också automatiskt upp med ett i detta fall.

Signalen S väljer om M eller GRx-fältet ska användas till att adressera GR0-GR3 (S=0 innebär att GRx-fältet adresserar GR0-GR3). Slutligen används signalen R=1 för att tvinga styrsignalen @ att bli 3.

*Figur 1: En välkänd datormodell (Björn Lindskog 1981)*

## Fråga 2: Allmän teori (10p)

(a) (2) Vad skiljer en processor av VLIW typ från en superskalär processor?

(b) (2) Motivera varför alla datorer innehåller icke-volatilt primärminne.

(c) (2) Vilken typ av konflikt löser Data forwarding?

(d) (2) Kan ett subrutinanrop skapa en styrkonflikt? Motivera ditt svar.

(e) (2) Vad betyder förkortningarna RISC respektive CISC ?

## Fråga 3: Assemblerprogrammering (10p)

I minnet ligger en tabell med 16-bitars tal lagrad, där bit 15-4 anger ett värde i tvåkomplementsform, och bit 3-0 (dvs minsta nibble) används som en typ av markering. Skriv en subrutin som hittar alla tal som har den markering som anges i register r2 och letar reda på det största talet och placerar detta i register r1 (dvs tar bort markeringen och teckenförlänger värdet till 32 bitars längd). Tabellens start anges i register r0 och antal värden i tabellen anges i r1. Största talet ska placeras i register r1 och därefter ska subrutinen Subrutin1 anropas. Subrutin 1 förstör registervärden i r0, r1 och r2. Subrutin1 definieras på annan plats och ska inte skrivas. Slutligen ska det största värdet placeras i r0 och subrutinen avslutas.

**Exempel**: r0 = 0x20001004, r1 = 0x00000006, r2 = 0x0000000a

| Adress | Värde | Adress | Värde |
|---|---|---|---|
| 0x20001000 | 0x123a | 0x2000100a | 0xef01 |
| 0x20001002 | 0x5678 | 0x2000100c | 0x5542 |
| 0x20001004 | 0xabe1 | 0x2000100e | 0x006a |
| 0x20001006 | 0x344a | 0x20001010 | 0x2f01 |
| 0x20001008 | 0xc42a | 0x20001012 | 0x2145 |

Subrutinen testar de 6 värdena på adresserna 0x20001004 – 0x2000100e, identifierar att 2:a (0x344a), 3:e (0xc42a) och 6:e (0x006a) värdet har rätt markering (högra nibble är a). För dessa värden med rätt markering tas markeringarna bort och teckenförlängs (0x00000344, 0xfffffc42, 0x00000006) och största talet hittas (0x00000344). Subrutin1 anropas då med r1=0x00000344, och när Subrutin1 är klar sätts r0 till det största värdet (0x00000344) och subrutinen avslutas därefter.

## Fråga 4: Avbrott (8p)

En kylpump styrs av en avbrottsrutin. Skriv denna avbrottsrutin. Avbrottsrutinen ska läsa ett 32-bitars värde som innehåller 2 sensorvärden från adress 0x40001000 (med en läsning). Bit 5-0 är första sensorns värde (positivt heltal) och bit 21-16 är 2:a sensorns värde (positivt heltal).

Om första sensorns värde är större än andra sensorns värde ska subrutinen Subrutin2 anropas.

Inga andra avbrott får ske medan detta avbrott hanteras.

Subrutin2 förstör register r5 och r7. Subrutin2 finns definierad på annan plats och ska inte skrivas.

**Exempel**: Avbrottsrutinen läser i en läsning det 32-bitars värdet 0x1E234568 från adress 0x40001000. Det första sensorvärdet är då 0x28 och den andra sensorns värde är 0x23. Då den första sensorns värde är större än den första sensorn anropas subrutinen Subrutin2.

## Fråga 5: Aritmetik (6p)

(a) (2p) Översätt decimaltalet -19 till ett 8-bitars tal i 2-komplementsform. Beskriv värdet som ett hexadecimalt tal.

(b) (2p) Beräkna en 8-bitars differens som resultat av subtraktionen av det 4-bitars 2-komplementstalet $0111_{2C}$ från det 6-bitars tvåkomplementstalet $111010_{2C}$. (dvs beräkna $111010_{2C}$-$0111_{2C}$). Utför beräkningen genom att skriva om subtraktionen till addition genom omskrivningen A-B = A + (-B), samt ställ upp den binära additionen och utför beräkningen. Svaret ska vara på formen av ett 8-bitar 2-komplements tal.

(c) (2p) En 8-bitars processor utför additionen 0x75+0x53. Ange flaggornas värden (Z,C,N,O).

## Fråga 6: Cache (6p)

En processor har en gruppassociativ cache som har 8 byte långa cachelines. Adressbussen är 32 bitar lång. Cachens storlek är 32768 byte. 8 bitar av adresssen används som index.

(a) (2p) Bestäm cachens associativitet, dvs hur många olika vägar har cachen?

(b) (2p) Hur många adressbitar används som tag?

(c) (2p) Antag cachen är tom. Hur många cachemissar fås om 6 läsningar av adresserna enligt sekvensen 0x1234567 + n*0x06 (n=0,1,2,3,4,5) görs? Antag processorn utför läsning av varje adress med instruktionen LDRB.

Kort repetition av ARM Cortex-M4

| Mnemonic | Kort förklaring | Mnemonic | Kort förklaring |
|---|---|---|---|
| ADR | Address | CPSID | Disable interrupt |
| ADD, ADDS | Addition | CPSIE | Enable interrupt |
| ADDC,ADDCS | Addition with C flag | EOR,EORS | Logic XOR |
| AND,ANDS | Logic AND | LDR | Load register |
| ASR,ASRS | Arithmetic shift right | LDRB,LDRSB | Load register byte |
| B | Branch | LDRH,LDRSH | Load register halfword |
| BCC,BLO | Branch on carry clear | LSL,LSLS | Logic shift left |
| BCS,BHI | Branch on carry set | LSR,LSRS | Logic shift right |
| BEQ | Branch on equal | MOV,MOVS | Move |
| BGE | Branch on greater than or equal | MUL,MULS | Multiply |
| BGT | Branch on greater than | MVN | Move negative |
| BL | Branch and link | NOP | No operation |
| BLT | Branch on less than | ORR,ORRS | Logic OR |
| BLE | Branch on less than or equal | POP | Pop |
| BLS | Branch on lower or same | PUSH | Push |
| BLT | Branch on less than | ROR | Rotate right |
| BMI | Branch on minus | SBC,SBCS | Subtraction with C flag |
| BNE | Branch on not equal | STR | Store register |
| BPL | Branch on plus | STRB | Store register byte |
| BVC | Branch on overflow clear | STRH | Store register halfword |
| BVS | Branch on overflow set | SUB,SUBS | Subtraction |
| BX | Branch and exchange | TST | Test |
| CMP | Compare (Destination - Source) | | |

```
; Exempel på ARM Cortex-M4 kod som kopierar 200 bytes från 0x2000 till 0x3000

main:  mov  r0,#(0x2000 & 0xffff)
       movt r0,#(0x2000 >> 16)
       mov  r1,#(0x3000 & 0xffff)
       movt r1,#(0x3000 >> 16)
       mov  r3,#50
loop:
       ldr  r4,[r0],#4
       str  r4,[r1],#4
       subs r3,r3,#1
       bne  loop
```