# Tentamen

## Datorteknik Y, TSEA28

| | |
|---|---|
| *Datum* | 2023-05-30 |
| *Lokal* | TERF,TER3,TER2,U3 |
| *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* | Ulrika Ericsson, 013 282379 |
| *Tillåtna hjälpmedel* | Inga |
| *Betygsgränser* | Preliminärt 0-20: U, 21-30: 3, 31-40: 4, 41-50: 5 |
| *Visning* | Måndag 19/6 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 du sett tidigare har följande förändring gjorts:
Mikroprogrammerings-ordet 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 := K1(OP) | ; direktadressering (M=00) |
| 4 | ASR := PC, uPC := uPC + 1 | ; omedelbar adressering (M=01) |
| 5 | PC := PC + 1, uPC := K1(OP) | |
| 6 | ASR := IR, uPC := uPC+1 | ; indirekt adressering (M=10) |
| 7 | ASR := PM, uPC := K1(OP) | |
| 8 | AR := IR, uPC := uPC+1 | ; indexerad adressering (M=11) |
| 9 | AR := GR3+AR, uPC := uPC+1, R:=1 | |
| 0xa | ASR := AR, uPC := K1(OP) | |

(a) (8p) Skriv mikrokod för instruktionen CMPSKIPGT GRx, M,A. Instruktionen jämför GRx med minnesvärdet på minnesadressen definierad av adresseringsmoden. Om GRx är större än minnesvärdet ska instruktionerna på de två följande adresserna hoppas över, annars ska nästa instruktion startas som vanligt. Antag att värdet i GRx och minnesvärdet är på tvåkomplementsform.

Alla adresseringsmoder ska stödjas. Flaggor får påverkas. Ange adresserna för mikrokodsinstruktionerna som absoluta tal.

**Exempel1:** PM(0x20)=0xf542 och GR2 = 0x2345. Om instruktionen CMPSKIPGT GR2,00,0x20 ligger på adress 0x45 och utförs, så ska sedan nästa instruktion som utförs vara den på adress 0x48.

**Exempel2:** GR3=0x37, PM(0x3c)=0x7542, GR1 = 0x2333. När instruktionen CMPSKIPGT GR1,11,5 ligger på adress 0xde och utförs, så ska sedan nästa instruktion som utförs vara den på adress 0xdf.

(b) (2p) Antag index 6 i K1 innehåller startadressen till mikrokoden för CMPSKIPGT. Ange det hexadecimala värdet för maskininstruktionen CMPSKIPGT GR1,10,0x5a.

## Fråga 2: Allmän teori (10p)

(a) (2) Ange två viktiga egenskaper som skiljer SRAM från DRAM.

(b) (2) Kan styrkonflikter lösas med hjälp av data forwarding? Motivera ditt svar

(c) (2) Varför är en fullt associativ cache dyrare att bygga (effekt och areamässigt) än en direktadresserad cache av motsvarande storlek?

(d) (2) Kan en addition i en processor ge flaggkombinationen N=1, Z=1? Motivera ditt svar.

(e) (2) Vilken processor kan utföra flest instruktioner per sekund: en 7-stegs pipelinad processor med 5 MHz klockfrekvens eller en 2-vägs superskalär processor med 3 MHz klockfrekvens?

## Fråga 3: Assemblerprogrammering (10p)

I minnet ligger två tabeller. Den första tabellen innehåller 32-bitars värden där de 4 mest signifikanta bitarna är ett index och de resterande 28 bitarna är ett positivt heltal. Den andra tabellen består av 16 stycken 32-bitars värden. Det första värdet i den andra tabellen har index 0.

Skriv en subrutin som går igenom alla värden i den första tabellen. För varje värde ska det positiva heltalet baserat på bit 27 till bit 0 av värdet adderas till 32-bitars heltalet som finns i andra tabellen på det index definierat av bit 31 till bit 28 i värdet från tabell 1.

Första tabellens start anges i register r0, antal värden i första tabellen anges i register r1, och startadressen till andra tabellen anges i register r2.

**Exempel**: r0 = 0x20001004, r1=0x00000004, r2=0x20001018

Före subrutinanrop

| Adress | Värde |
|---|---|
| 0x20001004 | 0x10123456 |
| 0x20001008 | 0x21010203 |
| 0x2000100c | 0x03222222 |
| 0x20001010 | 0x22334455 |
| 0x20001014 | 0xabcdef01 |
| 0x20001018 | 0x00224488 |
| 0x2000101c | 0x44556677 |
| 0x20001020 | 0x21314151 |
| 0x20001024 | 0x3a3b3c3d |

Efter subrutinanrop

| Adress | Värde |
|---|---|
| 0x20001004 | 0x10123456 |
| 0x20001008 | 0x21010203 |
| 0x2000100c | 0x03222222 |
| 0x20001010 | 0x22334455 |
| 0x20001014 | 0xabcdef01 |
| 0x20001018 | 0x034466aa |
| 0x2000101c | 0x44679acd |
| 0x20001020 | 0x246587a9 |
| 0x20001024 | 0x3a3b3c3d |

Subrutinen kommer gå igenom första tabellen med start på adressen 0x20001004. Första värdet (0x10123456) består av index 1 och heltalet 0x00123456, andra värdet (0x21010203) består av index 2 och heltal 0x01010203, tredje värdet (0x03222222) av index 0 och heltal 0x03222222 och fjärde värdet består av index 2 och heltal 0x02334455. I tabell två läggs då på index 0 (adress 0x20001018) heltalet 0x034466aa (0x00224488+0x03222222), på index 1 läggs heltalet 0x44679acd (0x44556677+0x00123456) och på index 2 läggs värdet 0x246587a9 (0x21314151+0x01010203+0x02334455).

## Fråga 4: Avbrott (8p)

Skriv en avbrottsrutin som läser ett 16-bitars värde (positivt heltal) från adress 0x40001008 (får bara läsas en gång). Bit 3 till bit 0 i 16-bitars värdet ska sedan inverteras (bitvärde 0 blir 1 respektive bitvärde 1 blir 0). Om det nya värdet är större än 0x0123 ska subrutinen Subrutin1 anropas med värdet placerat i register r0. Slutligen ska bit 7 till bit 0 av orginalvärdet (innan bit 3 till bit 0 inverterades) skrivas som ett 8-bitars värde till adress 0x40001010.

Subrutin1 förstör register r5 och r6. Subrutin1 finns definierad på annan plats och ska inte skrivas. Inga andra avbrottsrutiner får startas medan denna avbrottsrutin utförs.

**Exempel**: Avbrottsrutinen läser i en läsning det 16-bitars värdet 0xabdc från adress 0x40001008. Bit 3 till bit 0 inverteras vilket ger väret 0xabd3 som placeras i r0. Eftersom värdet 0xabd3 > 0x0123 så anropas subrutinen Subrutin1. Slutligen skrivs värdet 0xdc som en 8-bitar skrivning till adress 0x40001010 och avbrottsrutinen avslutas sedan.

## Fråga 5: Aritmetik (6p)

(a) (2p) En 8-bitars processor utför additionen 0x86 + 0x7a. Ange resulterande värden på flaggorna Z, C, N, och O.

(b) (2p) Ställ upp den binära additionen av de två 2-komplementstalen $10111_{2C}$ och $1011_{2C}$. Utför sedan additionen och ange summan. Summan ska anges med minsta möjliga antal bitar som fortfarande kan beskriva summan korrekt som ett 2-komplementstal.

(c) (2p) Skriv det decimala värdet -23 som ett 8-bitars 2-komplementstal i hexadecimal form.

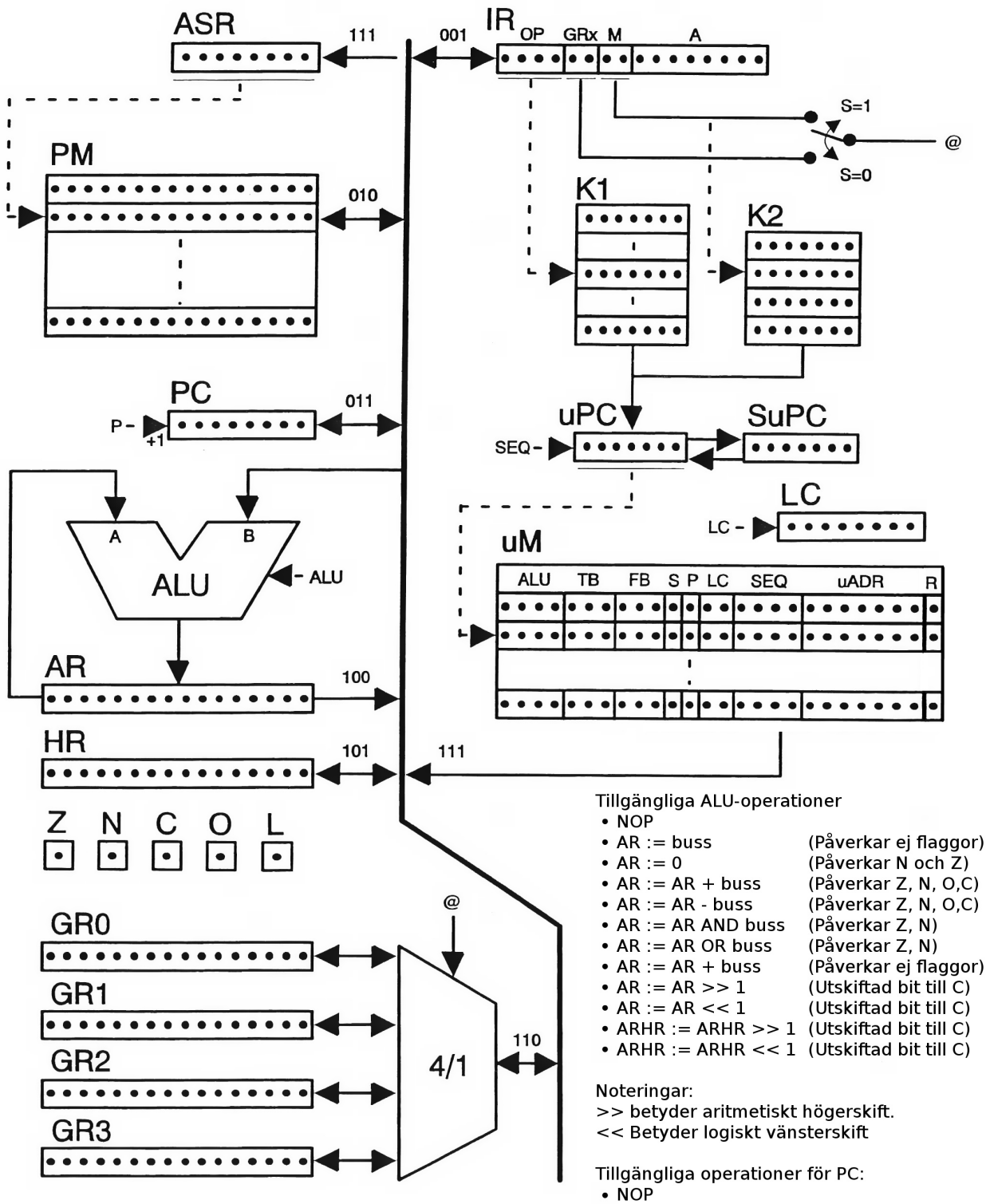
## Fråga 6: Cache (6p)

En processor har en 4-vägs gruppassociativ cache. Adressbussen är 32 bitar lång. Varje cacheline är 64 byte lång. 7 bitar används som index av cachen.

(a) (2p) Vilken storlek har denna cache?

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

(c) (2p) Antag cachen är tom. Hur många läsningar i sekvensen 0x12345678 + n*0x1800 (n = 0,1,2,3,...) kan göras innan redan inläst data börjar kastas ut ur cachen?




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)*

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
```