# Tentamen (Exempel)

## Datorteknik Y, TSEA28

| | |
|---|---|
| *Datum* | 2018-10-31 |
| *Lokal* | TER4 |
| *Tid* | 8-12 |
| *Kurskod* | TSEA28 |
| *Provkod* | TEN1 |
| *Kursnamn*<br>*Provnamn* | Datorteknik Y<br>Skriftlig tentamen |
| *Institution* | ISY |
| *Antal frågor* | 7 |
| *Antal sidor (inklusive denna sida)* | 7 |
| *Kursansvarig* | Kent Palmkvist, 1347, kentp@isy.liu.se |
| *Lärare som besöker skrivsalen*<br>*Telefon under skrivtiden* | Kent Palmkvist<br>1347, 013-281347 |
| *Besöker skrivsalen* | Ca kl 9 och kl 11 |
| *Kursadministratör* | |
| *Tillåtna hjälpmedel* | Inga |
| *Betygsgränser* | Preliminärt 0-20: U, 21-30: 3, 31-40: 4, 41-50: 5 |
| *Visning* | 14 November kl 13.00 – 14.00 i kursansvarigs 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 bland annat

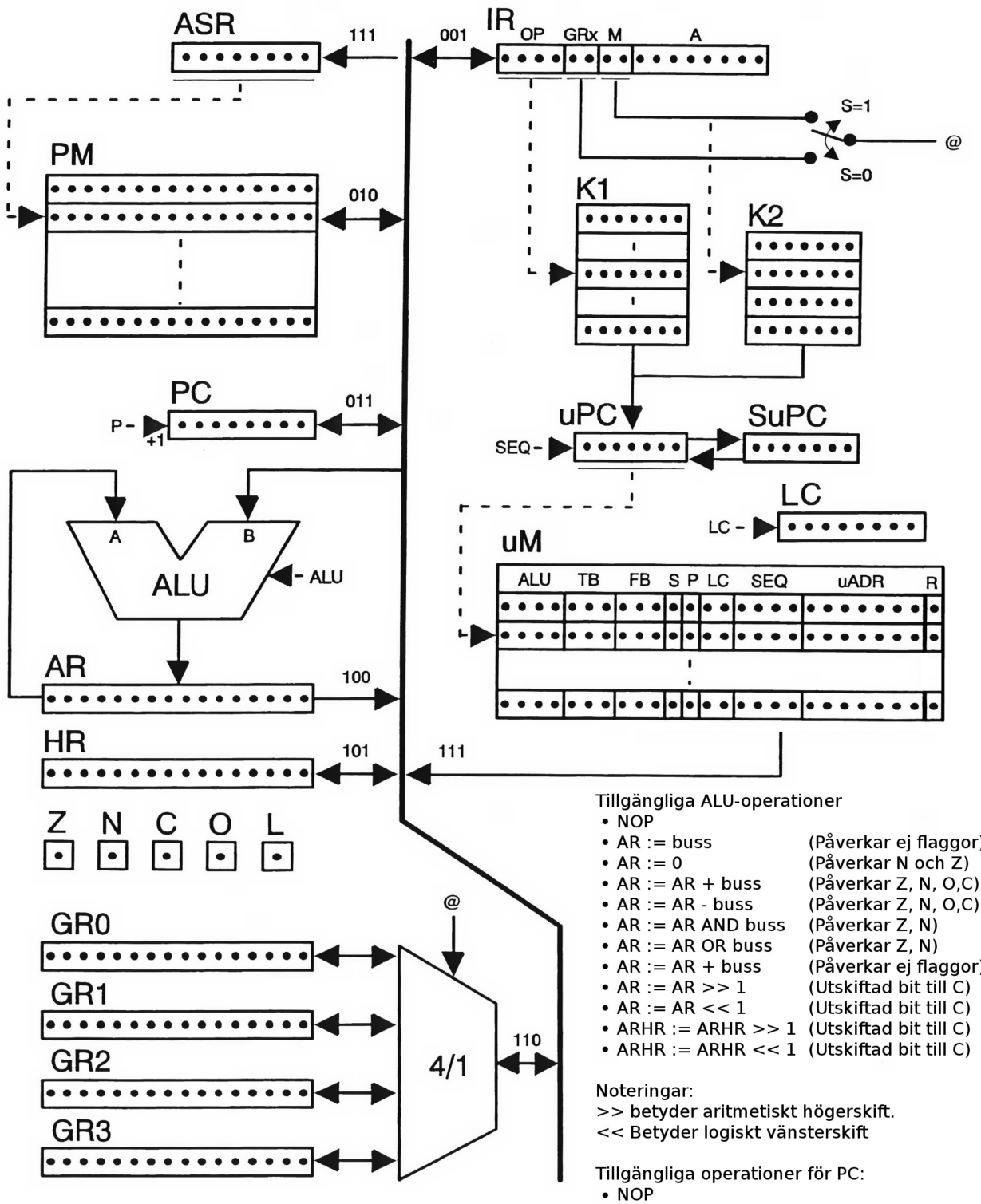
| addr | Mikrokod | Kommentar |
|---|---|---|
| 0 | ASR := PC, uPC := uPC+1 | |
| 1 | IR := PM, PC := PC+1, uPC := uPC+1 | |
| 2 | uPC := K2(M) | |
| 3 | ASR := IR, uPC := K1(OP) | ; direktadressering (M=00) |
| 4 | ASR := PC, PC := PC+1, uPC := K1(OP) | ; omedelbar adressering (M=01) |
| 5 | ASR :=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

a) (8p) Skriv mikrokod för instruktionen INCBRNZ GRx,A. Endast M=00 är tillåtet, och flaggorna får ändras av instruktionen. Instruktionen ska addera 1 till GRx, och om resultatet inte blev 0 ska ett hopp göras, där A anger hur långt hoppet ska göras relativt instruktionen efter INCBRNZ. A-fältet är ett 2-komplementstal. OP-code för INCBRNZ är 0xE.

Ange även adresserna för mikrokodsinstruktionerna.

**Exempel:** PM(0x10) = 0xE823 (motsvarar INCBRNZ GR2, 0x23), PC = 0x10, GR2 = 0x1234 Efter att INCBRNZ GR2,0x23 utförts ska GR2 = 0x1235 och PC = 0x34, dvs ett hopp till adress 0x34 har gjorts.

c) (2p) Fyll i K2 i binär form.




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) (2p) Vad skiljer en processorarkitektur av VLIW typ från en superskalar processorarkitektur?

(b) (2p) Vad är skillnaden mellan SRAM och DRAM?

(c) (2p) På vilket sätt kan virtuellt minne öka säkerheten i ett datorsystem?

(d) (2p) Varför är klockfrekvens ett olämpligt mått på en processors prestanda?

(e) (2p) Varför ger inte pipelining en ökning av prestanda som direkt motsvarar antal pipelinesteg som införs i processorn?

## Fråga 3: Assemblerprogrammering (10p)

Skriv en subrutin som kodar om en text med hjälp av kodnyckel som är 4 byte lång. Kodnyckelns första byte används för 1:a, 5:e, 9:e etc. tecknet i texten, andra byten i kodnyckeln används för 2:a, 6:e, 10:e etc. tecknet i texten, 3:e byten i kodnyckeln används för 3:e, 7:e, 11:e etc. tecknet i texten, och 4:e byten i kodnyckeln används för 4:e, 8:e, 12:e etc. tecknet i texten. Kodnyckelns byte ska xor:as med tecknet. De kodade tecknen ska sparas i minnet.

Kodnyckelns plats i minnet anges i r0, textens startadress anges i r1, längden på texten anges i r2 och platsen i minnet där den kodade texten ska lagras anges i r3. För 68000 anger A0 kodnyckelns plats, A1 anger textens startadress, textens längd anges i D0 och platsen där den kodade texten ska lagras anges i A2.

**Exempel**: r0 = 0x20001024, r1 = 0x20001004, r2 = 0x0000000a, r3 = 0x20001010

| Adress | Värde | Adress | Värde |
|---|---|---|---|
| 0x20001000 | 0x11 0x11 0x22 0x22 | 0x20001018 | 0x77 0x77 0x88 0x88 |
| 0x20001004 | 0x48 0x65 0x6a 0x20 | 0x2000101c | 0x55 0x55 0x66 0x66 |
| 0x20001008 | 0x68 0x6f 0x70 0x70 | 0x20001020 | 0x76 0x54 0x12 0x34 |
| 0x2000100c | 0x21 0x0a 0x00 0x00 | 0x20001024 | 0x01 0x02 0x04 0x08 |
| 0x20001010 | 0xab 0xcd 0x01 0x23 | 0x20001028 | 0xab 0xab 0xcd 0xcd |
| 0x20001014 | 0x20 0x00 0x10 0x20 | 0x2000102c | 0x23 0x32 0x44 0x44 |

När subrutinen är klar ska minnet på adress $20001010 och framåt se ut som nedan. De kodade tecknen är då 0x49 (0x48 XOR 0x01), 0x67 (0x65 XOR 0x02), 0x6e (0x6a XOR 0x04), 0x28 (0x20 XOR 0x08), 0x69 (0x68 XOR 0x01), 0x6d (0x6f XOR 0x02), 0x74 (0x70 XOR 0x04), 0x78 (0x70 XOR 0x08), 0x20 (0x21 XOR 0x01) och 0x08 (0x0a XOR 0x02).

| Adress | Värde |
|---|---|
| 0x20001010 | 0x49 0x67 0x6e 0x28 |
| 0x20001014 | 0x69 0x6d 0x74 0x78 |
| 0x20001018 | 0x20 0x08 0x66 0x66 |
| 0x2000101c | 0x77 0x77 0x88 0x88 |

## Fråga 4: Avbrott (8p)

En avbrottrutin ska skrivas, som först väntar på att bit 0 och 1 av byten på minnesadress 0x40001000 ska bli 1. När dessa båda bitar är 1 ska ett 32-bitars ord läsas av från minnesadress 0x40001010 och placeras i r0. Sedan ska subrutinen Subrutin2 (som definierats på annat ställe i koden) anropas. Denna subrutin påverkar register r0-r7. När subrutinen är klar ska avbrottsrutinen avslutas.

För 68000 gäller att D0 används istället för r0, och att D0-D7 påverkas av subrutinen Subrutin2.

**Exempel**: Värdet som läses från adressen 0x40001000 är 0x10. Adressen läses då en gång till och värdet är då 0x37. Register r0 laddas med värdet som lästs från adress 0x40001010 och subrutinen Subrutin2 anropas. Därefter avslutas avbrottsrutinen.

## Fråga 5: Aritmetik (6p)

(a) (2p) Skriv det hexadecimala talet 0x2e7 som ett 11-bitars binärtal.

(b) (2p) En 16-bitars dator beräknar summan 0x6223+0xbdac. Bestäm flaggornas värde (flaggor Z, C, N och O).

(d) (2p) Beräkna additionen av de två 2-komplementstalen ”110110101” och 011010111011110”. Beskriv svaret som ett 16-bitars binärtal i 2-komplements form.

## Fråga 6: Cache (6p)

Processorn i Raspberry Pi 3 har en 512 KByte (524288 bytes) level 2 cache med en cacheline längd av 64 byte. Denna cache är en gruppassociativ cache. Adressbussen är 32 bitar lång.

(a) (2p) Index bestäms med 7 bitar av adressen. Beräkna hur många vägar det finns i den gruppassociativa cachen.

(b) (2p) Hur många bitar av adressen används som tag?

(c) (2p) Antag cachen är tom. Hur många läsningar i sekvensen 0x20000000 + n*0x200 kan göras innan inläst data börjar kastas ut ur cachen?

# Kort repetition av M68000

| Mnemonic | Kort förklaring | Mnemonic | Kort förklaring |
|---|---|---|---|
| ADD | Addition | EXT | Sign extend |
| ADDX | Add with X flag | EXTB | Sign extend a byte to 32 bit |
| AND | Logic and | JSR | Jump to subroutine |
| ASL | Arithmetic shift left | LEA | Load effective address |
| ASR | Arithmetic shift right | LSL | Logic shift left |
| BCC | Branch on carry clear | LSR | Logic shift right |
| BCS | Branch on carry set | MOVE | Move |
| BEQ | Branch on equal | MULS | Signed multiplication |
| BGT | Branch on greater than | MULU | Unsigned multiplication |
| BLT | Branch on less than | NEG | Negate |
| BNE | Branch on not equal | NOP | No operation |
| BRA | Branch always | NOT | Bitwise logic invert |
| BSR | Branch to subroutine | OR | Logic OR |
| CLR | Clear | ROL | Rotate left |
| CMP | Compare (Destination - Source) | ROR | Rotate right |
| DIVS | Signed division | RTE | Return from exception |
| DIVU | Unsigned division | RTS | Return from subroutine |
| EOR | Logic XOR | SUB | Subtract |
| EXG | Exchange | TST | Set integer condition codes |

```
; Exempel på M68000 kod som kopierar 200 bytes från $2000 till $3000
    MOVE.L #$2000,A0
    MOVE.L #$3000,A1
    MOVE.B #50,D0
loop
    MOVE.L (A0)+,(A1)+
    ADD.B #-1,D0
    BNE loop
```

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

addr1: .field 0x2000,32
addr2: .field 0x3000,32

main:  ldr  r0,addr1
       ldr  r1,addr2
       mov  r3,#50
loop:
       ldr  r4,[r0],#4
       str  r4,[r1],#4
       subs r3,r3,#1
       bne  loop
```