Distanstentamen

OBS! Endast för studenter med LiU-ID abcdeX79, abcdeX88 eller abcdeX97

Datorteknik Y, TSEA28

Datum	2021-05-31
Lokal	Distanstentamen
Tid	14-18
Kurskod	TSEA28
Provkod	TEN1
Kursnamn *Provnamn*	Datorteknik Y Distanstentamen
Institution	ISY
Antal frågor	6
Antal sidor (inklusive denna sida)	6
Kursansvarig	Kent Palmkvist, Kent.Palmkvist@liu.se
Telefon/email under skrivtiden	Kent Palmkvist 013-281347 Kent.Palmkvist@liu.se
Tillåtna hjälpmedel	Föreläsningsmaterial, kursbok, laborationsanvisningar. Ingen kontakt med annan person än examinator är tillåten.
Betygsgränser	Preliminärt 0-20: U, 21-30: 3, 31-40: 4, 41-50: 5
Visning	Scannade rättade svar kommer läggas upp på inlämningsfunktionen i lisam

Viktig information

- Denna tenta ska endast lösas av studenter vars LiU-id slutar med två sista siffror som summerar till 16. Detta innebär alla LiU-ID av formen abcdeX79, abcdeX88 eller abcdeX97 (a-e bokstäver, X siffra).
- Inget namn eller personnummer behöver anges på inskickade papper eftersom inlämning i lisam automatiskt kopplar dokumentet till respektive students id.
- 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
- Svar inlämnas via lisams kursrum senast 18.20 Måndag 31 Maj 2021. Om problem uppstår med Lisam skickas motsvarande filer via email till Kent.Palmkvist@liu.se senast 18.30 Måndag 31 Maj 2021. För studenter med utökad skrivtid gäller sluttid 19.30 och inlämning senast 19.50.

Lycka till!

Fråga 1: Mikroprogrammering (11p)

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: 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 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, uPC := uPC+1	; indexerad adressering (M=11)
8	AR := GR3+AR, uPC := uPC+1, R:=1	
9	ASR := AR, uPC := K1(OP)	

(a) (6p) Skriv mikrokod som ersätter adressingsmode M=10 så den implementerar Indirekt med postincrement adressering. I indirekt med postincrement används värdet i GR3 som effektivadress och därefter uppdateras GR3 med summan av GR3 och värdet i instruktionens A-fältet. Bara A-fältets värde ska adderas. Opcode, GRx och M-bitarna i instruktionen ska inte påverka det nya värdet i GR3.

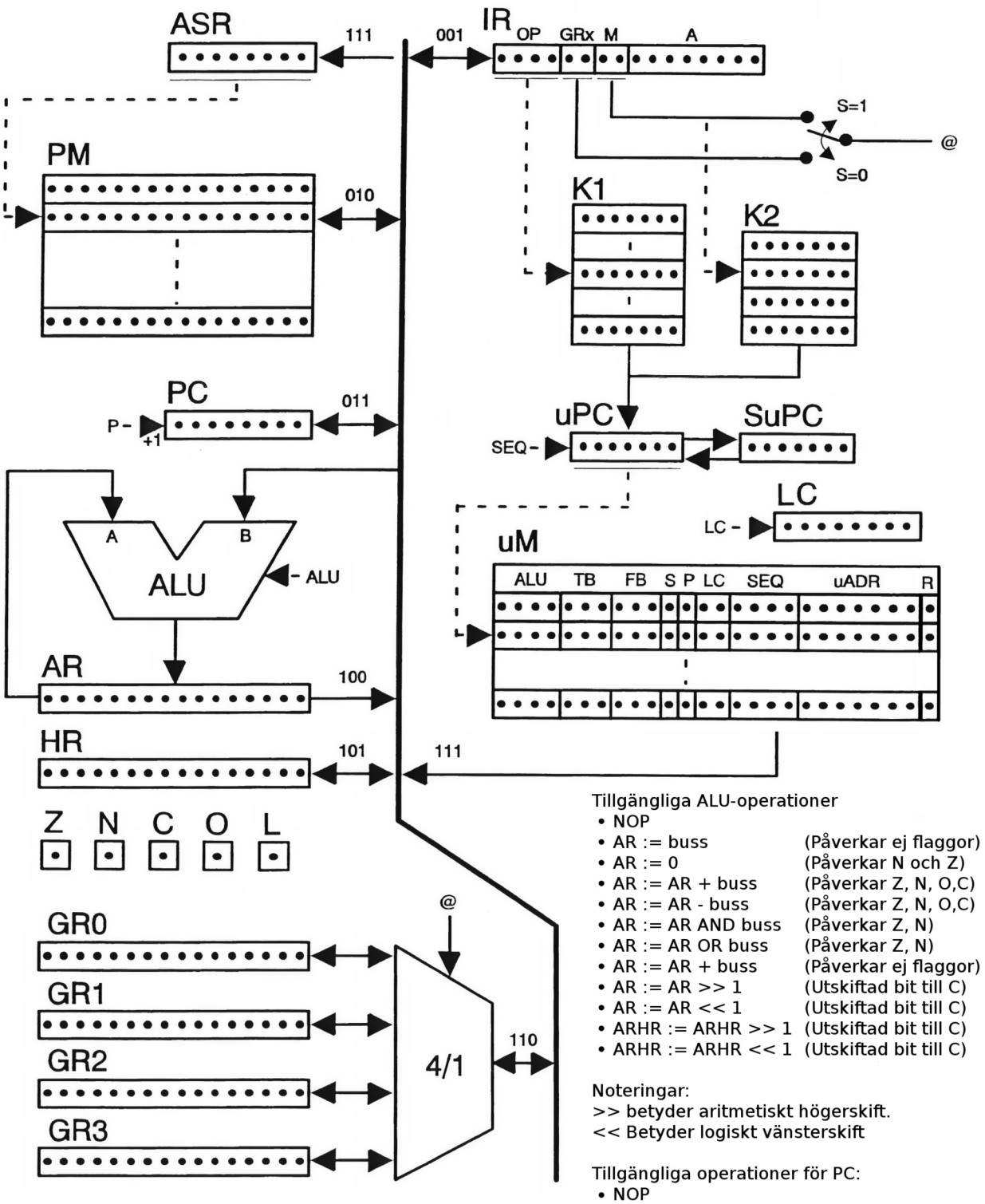
Exempel: Om GR3=0x97F8 och A-fältet=0x34 ska 0xF8 användas som effektiv adress, och GR3 ska uppdateras till summan 0x97F8+0x34=0x982C.

(b) (2p)

Ange innehåll i K2 efter att uppgift a) ersatt adresseringsmode M=01. Använd binärt format för både in och utsignaler. uPC är 7 bitar lång.

(c) (3p)

Skriv mikrokoden för instruktionen LOAD GRx,M,A. Denna instruktion laddar Grx-registret med det värde som finns i minnet på den adress som bestämts av adresseringsmode. Alla adresseringsmoder ska stödjas. Inga flaggor ska påverkas. Bestäm sedan hur många klockcykler det tar att utföra LOAD GR1,M=10,0x10. Använd adresseringsmoden implementerad i uppgift a).



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 (8p)

(a) (2p) Antag att en 4-vägs superskalär processor med 8 pipelinesteg kan klockas i 1 MHz. Antag sedan att en icke superskalär processor med 5 pipelinesteg kan klockas i 3 MHz. Vilken av dessa processorer kan nå högst genomströmningshastighet av instruktioner?

(b) (2p) Varför finns det olika instruktioner för multiplikation av 2-komplementstal respektive positiva heltal men inte separata instruktioner för addition av 2-komplementstal respektive positiva heltal? Motivera ditt svar.

(c) (2p) Varför är det ovanligt att mikroprogrammerade processorer innehåller branch prediction?

(d) (2p) Vad skiljer en processor av VLIW-typ från en processor av SIMD-typ?

Fråga 3: Assemblerprogrammering (10p)

Skriv en subrutin som läser värden i en tabell som startar i minnet på adress 0x10001234. I tabellen finns kodade värden som ska översättas, skickas till en subrutin kallad Subrutin1. och dessutom summeras. Summan ska sparas som ett 16-bitars 2-komplementstal på adress 0x10001232.

Subrutin1 förväntar sig det översatta 16-bitars värdet i register r0, och förstör register r2 och r3. Subrutin1 antas vara definerad på annan plats i koden (du ska inte skriva den subrutinen).

Varje kodat värde i tabellen består av 8 bitar. Det ska översättas till ett 16 bitars tal i 2-komplementsform. De fem mest signfikanta bitarna (bit 7 till bit 3) utgör ett värde i 2-komplementsform. De tre minst signifikanta bitarna (bit 2 till bit 0) anger hur många nollor som ska placeras till höger om 2-komplementstalet.

Det sista värdet i tabellen har värdet 0x00.

Exempel: Antag tabellen på adress 0x10001234 består av

Adress	Värde	Adress	Värde
0x10001234	0xab	0x10001236	0xc5
0x10001235	0x50	0x10001237	0x00

Subrutinen ska då översätta 0xab = 1010 1011_2 till 1111 1111 1010 1000_2 = 0xffa8 (10101_2 är negativt, lagt till 3 nollor till höger, som ett 16-bitars tvåkomplementstal) och anropa Subrutin1 med r0=0xffa8. Därefter ska 0x50 översättas till 0x0005 och Subrutin1 anropas med r0=0x0005, 0xc5 översättas till 0xff00 och Subrutin1 anropas med r0=0xff00. Summan blir 0xffa8+0x0005+0xff00=0xfead som placeras på adress 0x10001232.

Fråga 4: Avbrott (9p)

En avbrottsrutin ska samla in information från två GPIO-portar och sätta ihop dessa till ett nytt värde som skickas till en subrutin kallad Subrutin 2.

Från adress 0x40005000 ska en byte (och bara en byte) läsas in. Om både bit 0 och bit 1 är 1 ska ett 4-bitars värde från bit 6 till bit 3 kopieras in i register r0 på bitpositionerna bit 3 till bit 0. Om någon eller båda av bit 0 och bit 1 i värdet från 0x40005000 är 0 ska istället bit 7 placeras som bit 3 i r0, och bit 4 till bit 2 placeras som bit 2 till bit 0 i register r0.

Från adress 0x40007000 ska ett 16-bitars värde läsas in (bara 16 bitar får läsas och måste göras i en läsning) och bit 9 till 6 hos det inlästa värdet från adress 0x40007000 ska placeras i bitpositionerna bit 7 till bit 4 i register r0.

När de nya värdena lästs in i r0 (bit 7 till bit 4 beror av värdet i 0x40007000 och bit 3 till bit 0 beror av värdet i 0x40005000) ska sedan subrutinen Subrutin2 anropas. Denna är definierad på annan plats i koden (du behöver inte skriva den). Subrutinen förstör register r7 och r8.

Exempel: Avbrottet läser i en läsning 8-bitars värdet 0x12 från adress 0x40005000 och i en läsning värdet 0x12c4 från adress 0x40007000. Register r0 ska då sättas till 0x4b och subrutinen Subrutin2 anropas.

Fråga 5: Aritmetik (6p)

(a) (2p) Antag att en 8-bitars dator implementerar subtraktion A-B mha addition genom beräkningen A+(-B). Ställ upp additionen som binär addition utför subtraktionen 158-75 manuellt.

(b) (2p) Vilka värden får flaggorna Z, N, C och O i beräkningen i a).

(c) (2p) Ställ upp multiplikationen och beräkna produkten av de två 4-bitarstalen 0100 och 1011.

Fråga 6: Cache (6p)

En processor har en cache som har 64 byte långa cachelines. Adressbussen är 32 bitar lång. Cachen är en 2-vägs gruppassociativ cache. Index är 11 bitar.

(a) (2p) Hur många byte av data från primärminnet kan maximalt lagras i cacheminnet?

(b) (2p) Hur mycket extra minne (räknat i antal bitar) finns i cache om man räknar bort kopiorna av data från primärminnet?

(c) (2p) Antag cachen är tom. Hur många läsningar i sekvensen 0x12345678 + n*0x4000 (n=0,1,2,3,...) kan göras innan redan inläst data börjar kastas ut ur cachen?

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
```