CHALMERS

Institutionen för data- och informationsteknik
Avdelningen för datorteknik

TENTAMEN

KURSNAMN	**Digital- och datorteknik**
PROGRAM:	**Elektro** **Åk 1/ lp 4**
KURSBETECKNING	**EDA216**
EXAMINATOR	**Lars-Eric Arebrink**
TID FÖR TENTAMEN	**2013-06-03 kl 14.00 – 18.00**
HJÄLPMEDEL	**Av institutionen utgiven** **”Instruktionslista för FLEX- eller FLIS-processorn” (INS1)** **Tabellverk eller miniräknare får ej användas.**
ANSV LÄRARE: **Besöker tentamen**	**Lars-Eric Arebrink, tel. 772 5718** **vid flera tillfällen**
ANSLAG AV RESULTAT	**När rättningen är färdig anslås resultatet med anonyma koder och tid för granskning på kursens hemsida.**
ÖVRIG INFORM.	**Tentamen omfattar totalt 60 poäng.** **Den som är inskriven före HT 2012 kan använda FLEX-processorn istället för FLIS-processorn vid lösning av assembleruppgifter!** **Onödigt komplicerade lösningar kan ge poängavdrag.** **Svar på uppgifter skall motiveras.**
BETYGSGRÄNSER.	**Betyg 3: 24 poäng** **Betyg 4: 36 poäng** **Betyg 5: 48 poäng**
SLUTBETYG	**För slutbetyg 3, 4 eller 5 på kursen fordras betyg 3, 4 eller 5 på tentamen och godkända laborationer.**

1. I uppgift a-h nedan används 7-bitars tal X, Y, S och D. Aritmetiska operationer utförs på samma sätt och flaggor sätts på samma sätt som i ALU'n i bilaga 1. För tal med tecken används 2k-representation.

$X = 010\ 1011_2$ och $Y = 101\ 1111_2$.

a) Vilket talområde måste X, Y, S och D tillhöra om de tolkas som tal utan tecken? **(1p)**

b) Vilket talområde måste X, Y, S och D tillhöra om de tolkas som tal med tecken? **(1p)**

c) Visa med penna och papper hur räkneoperationen S = X + Y utförs i en 7-bitars ALU. **(1p)**

d) Vilka värden får flaggbitarna N, Z, V och C vid räkneoperationen i c)? **(1p)**

e) Visa med penna och papper hur räkneoperationen D = X – Y utförs i en 7-bitars ALU. **(1p)**

f) Vilka värden får flaggbitarna N, Z, V och C vid räkneoperationen i e)? **(1p)**

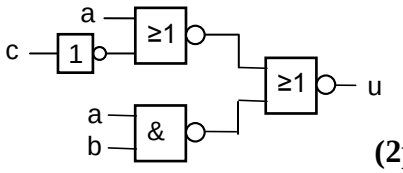
g) Tolka bitmönstren X, Y, S och D som tal *utan* tecken och ange deras decimala motsvarighet. Avgör och motivera med hjälp av flaggorna om resultaten S och D är korrekta eller felaktiga. **(1p)**

h) Tolka bitmönstren X, Y, S och D som tal *med* tecken och ange deras decimala motsvarighet. Avgör och motivera med hjälp av flaggorna om resultaten S och D är korrekta eller felaktiga. **(1p)**

i) Översätt (packa) det decimala talet –56,375 till ett 32-bitars flyttal enligt flyttalsstandarden IEEE 754 (dvs 23 bitar av mantissan). Ge det packade flyttalet på binär och hexadecimal form. **(2p)**

2.

a) Ge ett "minimalt" boolesk SP-uttryck för u i grindnätet till höger.



(2p)

b) En boolesk funktion f(a,b,c,d) har karnaughdiagrammet till höger. Realisera funktionen med så få grindar som möjligt. NAND-grindar med valfritt antal ingångar, XOR-grindar och NOT-grindar får användas. Endast insignalerna a, b, c och d finns tillgängliga. Rutor med - representerar "dont care"-termer som får användas vid behov.

ab \ cd	00	01	11	10
00	0	1	1	0
01	1	0	-	0
11	-	1	-	0
10	-	-	1	0

(4p)

3.

a) Konstruera en JK-vippa med hjälp av en SR-vippa och standardgrindar. **(3p)**

b) Konstruera en 4-bitars autonom uppräknare. (Räknesekvens: 0, 1, 2, …, 14, 15, 0, 1, 2, …) Numrera tillstånden $q_3q_2q_1q_0$. T-vippor, AND-grindar med valfritt antal ingångar, XOR-grindar och NOT-grindar får användas.

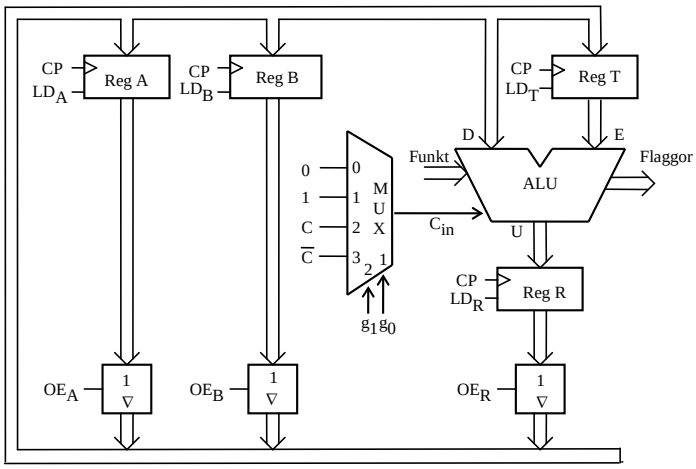
Ledning: Det finns ett enkelt villkor som avgör om en bit i det binära talet $(q_3q_2q_1q_0)_2$ skall ändras från 0 till 1 (eller tvärt om) vid uppräkningen! **(6p)**

4. I datavägen till höger innehåller register A och B från början värdena 24_{10} resp. 44_{10} på binär form. Därefter ges styrsignaler och klockpulser enligt tabellen nedan.

Rita av tabellen!

Fyll sedan i RTN-beskrivning samt hexadecimalt registerinnehåll för alla klockpulsintervall. Vid laddning av ett register skall det nya innehållet ”synas” på raden efter laddningen i tabellen.

ALU-funktioner: Se bilaga 1!



CP	Styrsignaler (=1)	RTN	Reg A	Reg B	Reg T	Reg R
1	OE_A, LD_T		18	2C	?	?
2	OE_B, f_3, f_1, LD_R					
3	OE_R, f_3, f_1, f_0, LD_R					
4	OE_R, f_3, f_1, f_0, LD_R					
5	OE_R, f_3, f_2, LD_R					
6	OE_R, f_1, f_0, LD_R					
7	OE_R, f_3, g_0, LD_R					
8	OE_R, LD_B					
9	?					

(1+3p)

5. Figuren på sidan 45 i INS1 visar hur FLIS-datorn är uppbyggd. På sidorna 43 och 44 i INS1 visas hur ALU’ns funktion väljs med styrsignalerna f_3 - f_0 och C_{in}.

I tabellerna nedan visas styrsignalerna för två olika EXECUTE-sekvenser för två olika maskininstruktioner, en för FLIS- och en för FLEX-processorn.

State	S-term	RTN-beskrivning	Styrsignaler (=1)
Q_4	$Q_4 \cdot I_{xx}$		MR, LD_T, INC_{PC}
Q_5	$Q_5 \cdot I_{xx}$		MR, g_{13}, LD_T
Q_6	$Q_6 \cdot I_{xx}$		OE_A, f_2, f_1, f_0, g_3, g_2, LD_{CC},NF

a) Rita en tabell med ”State”- och RTN-kolumner enligt ovan (FLISP) eller nedan (FLEX) för **en** av instruktionerna och fyll i RTN-beskrivningen! Förklara vilken assemblerinstruktion som beskrivs!

Som alternativ kan du använda FLEX-datorn som visas i bilaga 3 med ALU'n i bilaga 1 och tabellen nedan istället för FLIS-datorn.

State	S-term	RTN-beskrivning	Styrsignaler (=1)
Q_5	$Q_5 \cdot I_{xx}$		OE_X, LD_T
Q_6	$Q_6 \cdot I_{xx}$		OE_B, f_3, f_1, LD_R
Q_7	$Q_7 \cdot I_{xx}$		OE_R, LD_{MA}
Q_8	$Q_8 \cdot I_{xx}$		MR, LD_A, NF

(2p)

5(forts.)

b) Instruktionen ”Add memory bytes” nedan skall implementeras för FLIS- eller FLEX-processorn med hjälp av styrenheten med fast logik. Instruktionen består av fyra ord. Den utför addition av dataorden M(Adr1) och M(Adr2). Summan placeras som M(Adr3). Flaggorna skall påverkas som vid en vanlig addition.

OPKOD
Adr1
Adr2
Adr3

Register som är synliga för programmeraren får ej påverkas, förutom CC. Operationskoden DF_{16} skall användas.

ADMB Adr1,Adr2,Adr3, RTN: M(Adr1) + M(Adr2) → M(Adr3)
Flags → CC

Gör en tabell liknande tabellen i uppgift a) för den efterfrågade EXECUTE-sekvensen. **(4p)**

6. Besvara kortfattat följande frågor rörande FLIS- eller FLEX-processorn.

a) Förklara varför det kan vara en fördel att använda instruktionen BRA Adr istället för JMP Adr. **(1p)**

b) Före de villkorliga hoppen BMI och BGT i ett program utförs en subtraktion som påverkar flaggorna. Förklara vad som händer för vardera hoppet om subtraktionen före är $65_{16} - 97_{16}$. **(2p)**

c) Översätt subrutinen till höger till maskinkod på hexadecimal form och visa hur den placeras i minnet. Det skall framgå hur offset för branch-instruktionerna beräknas. **(3p)**

```
INLP    EQU     5
;
        ORG     $A0
DELAY   PSHX
        PSHA
;
LOOP    LDX     #INLP
XLOOP   LEAX    –1,X
        CMPX    #0
        EXG     X,SP
        EXG     X,SP
        BPL     XLOOP
;
        INCA
        BNE     LOOP
;
DELX    PULA
        PULX
        RTS
```

d) Subrutinen i c) anropas i huvudprogrammet nedan.

```
        ORG     $10
        LDSP    #$FA
        LDA     #–10
        BSR     DELAY
STOP    BRA     STOP
```

Hur lång tid tar subrutinen att köra från och med BSR till och med RTS om FLIS-(FLEX)-processorn klockas med frekvensen 1MHz? **(3p)**

7. I minnet i ett datorsystem med FLIS- eller FLEX-processorn finns ett datablock, BLOCK1, med ett antal 8-bitars dataord. BLOCK1 avslutas med ett dataord med värdet 0, som bara förekommer som slutmarkering.

Skriv en subrutin MODIFY i assemblerspråk som läser varje dataord från BLOCK1, nollställer bit 7 och 2, ettställer bit 5, inverterar bit 4 och sedan skriver det nya dataordet i ett nytt datablock, BLOCK2, som också skall avslutas med ett dataord med värdet 0. BLOCK1 skall inte påverkas av subrutinen.

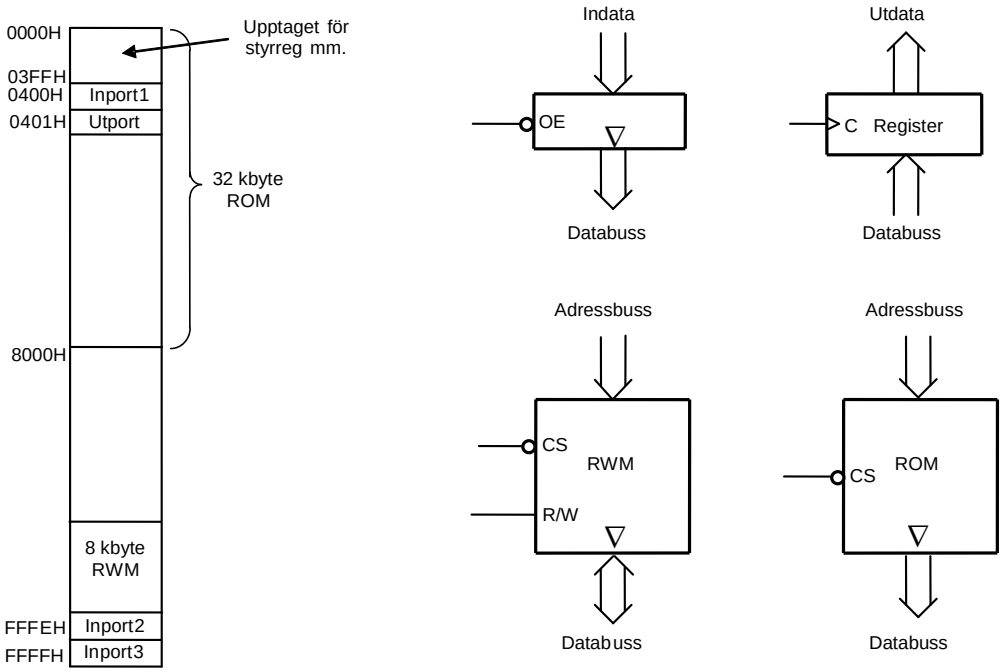
Subrutinen MODIFY skall dessutom räkna alla dataord i BLOCK1 med bit 7 = 1 och placera detta antal i A-registret före återhopp.

För FLISP gäller att startadresserna till BLOCK1 och 2 skall finnas i X-registret resp. Y-registret vid anrop av subrutinen.

För FLEX skall startadresserna till BLOCK1 och 2 finnas i X-registret resp. B-registret vid anropet.

Endast A-registret och flaggregistret får vara förändrade vid återhopp. För full poäng skall programmet vara ”korrekt” radkommenterat. **(8p)**

8. Ett mikrodatorsystem skall konstrueras med CPU12, 1 st 32 kbyte ROM-kapsel, 1 st 8 kbyte RWM-kapsel, 3 st 8-bitars "three-state"-buffertar som inportar och ett 8-bitars register som utport. Se figuren nedan! (Adresser som anges som xxxxH är hexadecimala värden.)



Signaler från processorn: D_7-D_0, A_{15}- A_0, E, R/W

RWM-modulen skall placeras så att den upptar de sista 8k adressena i adressrummet, förutom de sista två adresserna, där inportarna nr 2 och 3 är placerade. Till höger visas principen för de olika modulerna. Samtliga moduler utom utporten aktiveras med låg nivå (0) på respektive CS- eller OE-ingång. Utportens klockingång har positiv flanktriggning.

Till vänster i figuren framgår att adressområdet $0\text{-}400_{16}$ är upptaget. Det innebär att ROM-kapseln inte får aktiveras i detta område.

För att undvika busskollisioner måste Inport2 och Inport3 prioriteras före RWM-kapseln.

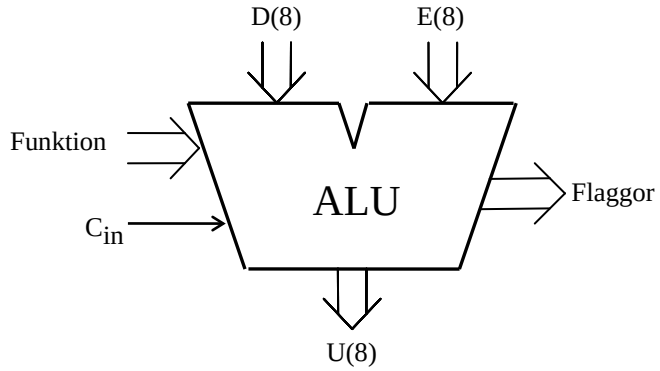
Uppgift:
Använd lämpliga signaler från processorn för att konstruera CS-logiken för ROM-modulen, RWM-modulen, inportarna och utporten. Använd fullständig adressavkodning. Endast grundläggande logikgrindar med valfritt antal ingångar får användas. Det användbara adressintervallen för de två minnesmodulerna skall anges i hexadecimal form.

Ledning: Adressområdet som omfattar de första 400_{16} adresserna kan betraktas som en modul med 400_{16} olika adresser.

(8p)

Bilaga 1

ALU:ns funktion (FLEX-ALU'n)



ALU:ns **logik-** och **aritmetikoperationer** på indata **D** och **E** definieras av ingångarna **Funktion (F)** och $\mathbf{C_{in}}$ enligt tabellen nedan. **F** = $\mathbf{(f_3, f_2, f_1, f_0)}$.

I kolumnen Operation förklaras hur operationen utförs.

f_3 f_2 f_1 f_0	U = f(D,E,C_{in})	
	Operation	**Resultat**
0 0 0 0	bitvis nollställning	0
0 0 0 1		D
0 0 1 0		E
0 0 1 1	bitvis invertering	D_{1k}
0 1 0 0	bitvis invertering	E_{1k}
0 1 0 1	bitvis OR	D OR E
0 1 1 0	bitvis AND	D AND E
0 1 1 1	bitvis XOR	D XOR E
1 0 0 0	D + 0 + C_{in}	D + C_{in}
1 0 0 1	D + FFH + C_{in}	D – 1 + C_{in}
1 0 1 0		D + E + C_{in}
1 0 1 1	D + D + C_{in}	2D + C_{in}
1 1 0 0	D +E_{1k} + C_{in}	D – E – 1 + C_{in}
1 1 0 1	bitvis nollställning	0
1 1 1 0	bitvis nollställning	0
1 1 1 1	bitvis ettställning	FFH

Carryflaggan (C) innehåller minnessiffran ut (carry-out) från den mest signifikanta bitpositionen (längst till vänster) om en aritmetisk operation utförs av ALU:n.

Vid **subtraktion** gäller för denna ALU att
C = 1 om lånesiffra (borrow) uppstår och
C = 0 om lånesiffra inte uppstår.

Carryflaggans värde är 0 vid andra operationer än aritmetiska.

Overflowflaggan (V) visar om en aritmetisk operation ger "overflow" enligt reglerna för 2-komplementaritmetik.

V-flaggans värde är 0 vid andra operationer än aritmetiska.

Zeroflaggan (Z) visar om en ALU-operation ger värdet noll som resultat på U-utgången.

Signflaggan (N) är identisk med den mest signifikanta biten (teckenbiten) av utsignalen U från ALU:n.

Half-carryflaggan (H) är minnessiffran (carry) mellan de fyra minst signifikanta och de fyra mest signifikanta bitarna i ALU:n.

H-flaggans värde är 0 vid andra operationer än aritmetiska.

I tabellen ovan avser "**+**" och "**–**" **aritmetiska operationer**. Med t ex $\mathbf{D_{1k}}$ menas att samtliga bitar i **D** inverteras.

Bilaga 2

Assemblerspråket för FLEX- och FLIS-processorn.

Assemblerspråket använder sig av mnemoniska beteckningar liknande dem som processorkonstruktören MOTOROLA (FREESCALE) specificerat för maskininstruktioner för mikroprocessorerna 68XX och instruktioner till assemblatorn, s k pseudoinstruktioner eller assemblatordirektiv. Pseudoinstruktionerna listas i tabell 1.

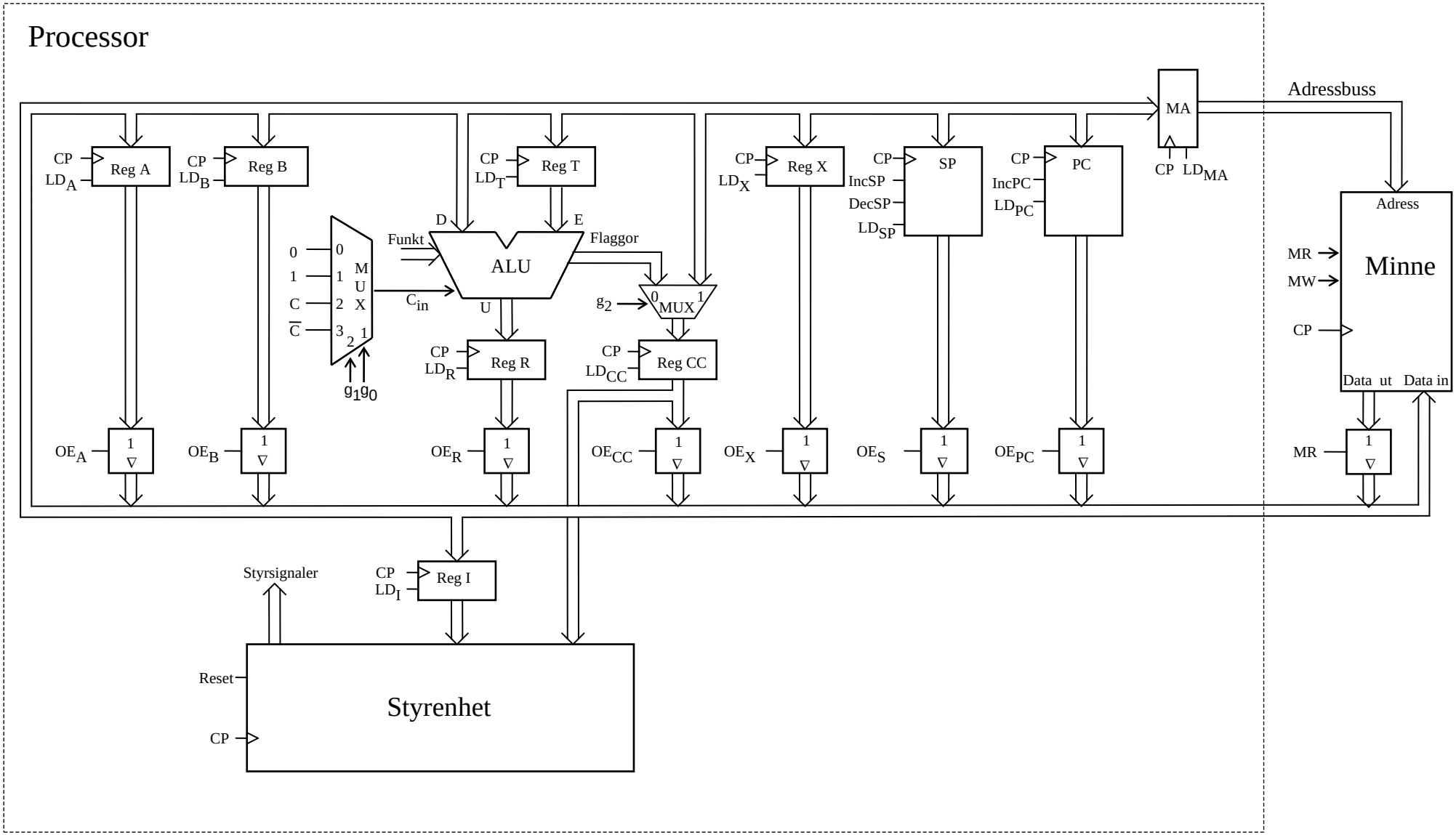
Tabell 1

Direktiv	Förklaring
ORG N	Placerar den efterföljande koden med början på adress N. (ORG för ORiGin = ursprung)
L RMB N	Avsätter N bytes i följd i minnet (utan att ge dem värden), så att programmet kan använda dem. Följden placeras med början på adressen L. (RMB för Reseve Memory Bytes)
L EQU N	Ger symbolen L konstantvärdet N. (EQU för EQUates = beräknas till)
L FCB N1,N2	Avsätter en byte för varje argument i följd i minnet. Respektive byte ges konstantvärdet N1, N2 etc. Följden placeras med början på adressen L. (FCB för Form Constant Byte)
L FCS ”ABC”	Avsätter en byte för varje tecken i teckensträngen ”ABC” i följd i minnet. Respektive byte ges ASCII-värdet för A B C, etc. Följden placeras med början på adressen L. (FCS för Form Character String)

Tabell 2 7-bitars ASCII

0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1	$b_6b_5b_4$ $b_3b_2b_1b_0$
NUL	DLE	SP	0	@	P	`	p	0 0 0 0
SOH	DC1	!	1	A	Q	a	q	0 0 0 1
STX	DC2	“	2	B	R	b	r	0 0 1 0
ETX	DC3	#	3	C	S	c	s	0 0 1 1
EOT	DC4	$	4	D	T	d	t	0 1 0 0
ENQ	NAK	%	5	E	U	e	u	0 1 0 1
ACK	SYN	&	6	F	V	f	v	0 1 1 0
BEL	ETB	‘	7	G	W	g	w	0 1 1 1
BS	CAN	(	8	H	X	h	x	1 0 0 0
HT	EM	)	9	I	Y	i	y	1 0 0 1
LF	SUB	*	:	J	Z	j	z	1 0 1 0
VT	ESC	+	;	K	[Ä	k	{ä	1 0 1 1
FF	FS	,	<	L	\Ö	l	|ö	1 1 0 0
CR	GS	-	=	M	]Å	m	}å	1 1 0 1
S0	RS	.	>	N	^	n	~	1 1 1 0
S1	US	/	?	O	_	o	RUBOUT (DEL)	1 1 1 1

Bilaga 3 (Observera att bilden visar FLEX-datorn)



Figur 1. Datorn FLEX.