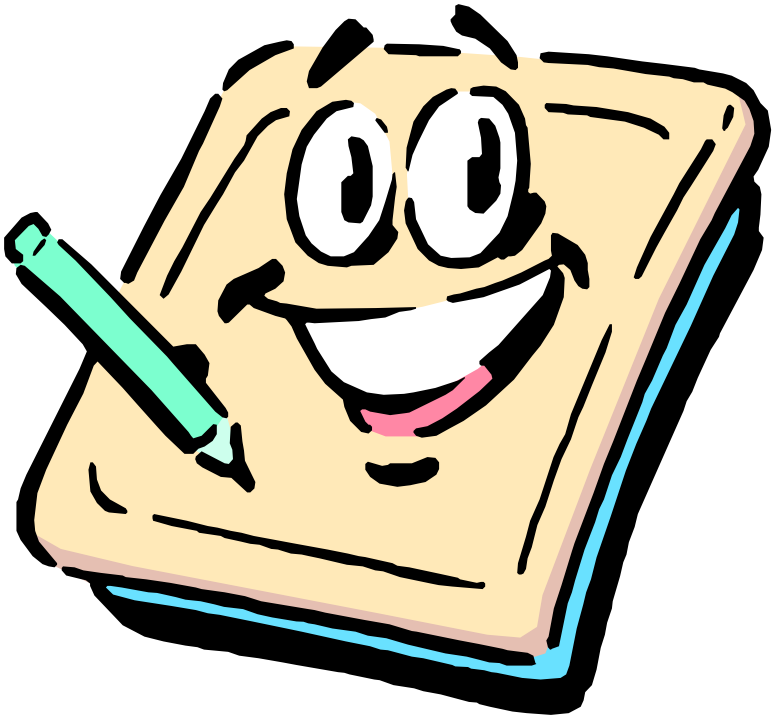
Övningsuppgifter i Mikrodatorteknik 4p/5p



Benny Thörnberg

Mittuniversitetet
Inst. för Informationsteknologi och medier

Hösten 2005

1 Exekvering av assemblerkod

1.1 Statusflaggors beteende

Vad blir C-, N- och Z- flaggornas värden samt D0:s värden efter varje instruktion i följande program (redovisa beräkningar).

```
a) MOVE.B       #$C5,D0
   ADDI.B       #$3B,D0
   MOVE.B       #$5F,D0
   ADDI.B       #$F1,D0
   EORI.B       #$80,D0
   ASR.B        #2,D0


b) MOVE.B       #$95,D0
   ASR.B        #3,D0
   SUBI.B       #$43,D0
   ADDI.B       #$F7,D0
   EORI.B       #$A6,D0
   ORI.B        #$82,D0


c) CLR.B        D0
   OR.B         #$E5,D0
   ASR.B        #3,D0
   ADD.B        #$04,D0
   ADD.B        #$57,D0
   EORI.B       #$A8,D0
```

1.2 Hoppinstruktioners beteende

Du ska bestämma exekveringsordningen för instruktionerna i följande program, redogör också varför vissa hopp utförs och vissa inte (motivera med registrens och statusflaggornas värden). Tänk på att vissa instruktioner kan utföras flera gånger.

```
a) L1: MOVE.W   #$A0A0,D1
   L2: BRA      L4
   L3: LSL.W    #1,D1
   L4: BMI      L3
   L5: ANDI.W   #$BEBF,D1
   L6: BEQ      L9
   L7: ASL.W    #$5,D1
   L8: BCS      L12
   L9: EORI.W   #$8000,D1
   L10:ASR.W    #$4,D1
   L11:BMI      L7
   L12:BRA      L12


b) L1: CLR.W    D1
   L2: BRA      L8
   L3: ASR.W    #2,D1
```

```
   L4: BCC       L7
   L5: EORI.W    #$D2D2,D1
   L6: BMI       L3
   L7: NOP
   L8: BEQ       L5
   L9: MOVE.B    #$FF,D1
   L10:BPL       L12
   L11:BMI       L12
   L12:BRA       L12


c) L1: EOR.W     D1,D1
   L2: BNE       L8
   L3: ADD.W     #$CCCC,D1
   L4: BMI       L8
   L5: ADD       #$E667,D1
   L6: BCS       L10
   L7: BRA       L10
   L8: LSR.W     #3,D1
   L9: BCS       L4
   L10:BEQ       L12
   L11:BNE       L12
   L12:BRA       L12
```

1.3 Implementera följande C-kod mha 68000 assembler

Variablerna a och b representeras med 16-bitars tal.

```
a) while(a>10) {
       a = a - 1;
       b = b & 0x33
   }


b) for(a=0; a<20; a++)  {
       b = b + 34;
   }
```

c) if (a>10)
    b = b & 0x5555;
else
    b = b | 0x5555;

2 Adressering och minneshantering

2.1 Adresseringsmetoder I

Vad kommer D3 och A3 att innehålla efter följande instruktion?
Innehållet i minnet framgår av tabellen. D3 = $00000000 innan instruktionen, A3 är definierad i varje del fält (Koden består av enskilda instruktioner inte ett program).

Adress (Hexadecimal)	Data i minnet (Hex)
4000	39
4001	45
4002	A6
4003	F6
4004	D4
4005	AA
4006	67
4007	89
4008	34
4009	22
400A	00
400B	23
400C	16
400D	24
400E	90
400F	87

a) A0= $4008

I. MOVE.W $400C, D3
II. MOVE.B (A0), D3
III. MOVE.W -(A0), D3
IV. MOVE.L -(A0), D3
V. MOVE.W (A0)+, D3
VI. MOVE.L (A0)+, D3
VII. MOVE.W #$400E, D3
VIII. MOVE.B -7(A0), D3

b) A1 = 4004$

I. MOVE.W #$4002, D3
II. MOVE.L (A1), D3
III. MOVE.B (A1), D3
IV. MOVE.B -(A1), D3
V. MOVE.L -(A1), D3
VI. MOVE.L (A1)+, D3
VII. MOVE.W $4008, D3
VIII. MOVE.B -3(A1), D3

c) A2 = $400C

I. MOVE.B $400E, D3
II. MOVE.W #$4006, D3
III. MOVE.L -(A2), D3
IV. MOVE.W (A2)+, D3
V. MOVE.B -5(A2), D3
VI. MOVE.B (A2)+, D3
VII. MOVE.B -(A2), D3
VIII. MOVE.L (A2), D3

2.2 Adresseringsmetoder II

Fyll i tabellen på nästa sida, tabellen skall representera minnets innehåll efter varje instruktion. Varje minnesposition är initierad till noll före varje instruktion (Koden består av enskilda instruktioner inte ett program).

a) D3 = $12345678, A0 = $400C

```
I.    MOVE.B   #$45, (A0)
II.   MOVE.W   D3, (A0)
III.  MOVE.L   D3, -4(A0)
IV.   MOVE.W   #$345, -(A0)
V.    MOVE.L   #$45, (A0)+
VI.   MOVE.B   D3, $400B
VII.  MOVE.W   D3, -(A0)
VIII. MOVE.L   #$345632, -8(A0)
```

b) D3 = $87654321, A1 = $4008

```
I.    MOVE.L   #$7E, $4000
II.   MOVE.W   #$1245, -(A1)
III.  MOVE.B   #$7E, -(A1)
IV.   MOVE.L   D3, (A1)+
V.    MOVE.W   D3, $4002
VI.   MOVE.B   D3, (A1)+
VII.  MOVE.W   #$324, -6(A1)
VIII. MOVE.L   D3, -(A1)
```

c) D3 = $01234567, A2= $4004

```
I.    MOVE.W   #$9F6, (A2)
II.   MOVE.W   D3, -(A2)
III.  MOVE.L   D3, -(A2)
IV.   MOVE.L   #$342, $400C
V.    MOVE.B   #$3F, (A2)+
VI.   MOVE.W   #$5F2, -2(A2)
VII.  MOVE.B   D3, 9(A2)
VIII. MOVE.L   #$4FD, (A2)
```

Benny Thörnberg Tel: 060-148917
Email: benny.thornberg@miun.se

Adress (Hexadecimal)	Data i minnet (Hex)								
	-	I	II	III	IV	V	VI	VII	VIII
4000	00								
4001	00								
4002	00								
4003	00								
4004	00								
4005	00								
4006	00								
4007	00								
4008	00								
4009	00								
400A	00								
400B	00								
400C	00								
400D	00								
400E	00								
400F	00								

2.3 Deklaration av minnesobjekt

Vilka adresser kommer varje label att representera (för tecken ersätts med ASCII kod).

Kod: a

```
        ORG     $5500
A       DC.B    $45
B       DS.B    7
C       DS.W    1
D       DC.W    $454
E       DC.L    $7889
F       DC.W    $678
G       DC.B    'Hello',0
```

Kod: b

```
        ORG     $5000
A       DC.B    $7F
B       DS.B    1
C       DS.W    3
D       DC.L    $332
E       DS.W    2
F       DC.L    $54322
G       DC.B    'Kalle',0
```

Kod: c

```
        ORG     $6000
A       DS.B    5
B       DC.B    $23
C       DS.W    3
D       DC.L    $234
E       DS.L    2
F       DC.W    $FD4
G       DC.B    'Anka',0
```

Strukturerad programmering

2.4 68000's datastack och subrutiner

Rita stacken efter varje instruktion i programmet nedan (I-VII). Anta att SP=$8000 innan instruktion I.

a)

```
I)      MOVE.L  D0, -(SP)
II)     MOVE.W  D1, -(SP)
III)    MOVE.B  D2, -(SP)
IV.i)   BSR     kalle                   IV.ii) kalle    RTS
V)      MOVE.B  (SP)+, D2
VI)     MOVE.W  (SP)+, D1
VII)    MOVE.L  (SP)+, D0
```

b)

```
I)      MOVE.B  D0, -(SP)
II)     MOVE.W  D1, -(SP)
III)    MOVE.L  D2, -(SP)
IV.i)   BSR     kalle                   IV.ii) kalle    RTS
V)      MOVE.L  (SP)+, D2
VI)     MOVE.W  (SP)+, D1
VII)    MOVE.B  (SP)+, D0
```

c)

```
I)      MOVE.W  D0, -(SP)
II)     MOVE.L  D1, -(SP)
III)    MOVEM.B D2, -(SP)
IV.i)   BSR     kalle                   IV.ii) kalle    RTS
V)      MOVE.B  (SP)+, D2
VI)     MOVE.L  (SP)+, D1
VII)    MOVE.W  (SP)+, D0
```

Parameteröverföring I

Implementera parameteröverföringen för funktionsdeklarationen. Skriv även koden för att anropa funktionen. **long int är 32 bitar och short int är 16 bitar bred**.

a)

```
long int tjatte(short int a, long int *b, short int c) {
   …
}
```

a och *b överförs via stacken
c som register (D0)
och returvärdet som register (D0)

b)

```
long int knatte(short int *a, long int, long int c) {
   …
}
```

a och b överförs via stacken
c som register (D0)
och returvärdet som register (D0)

c)

```
long int fnatte(short int a, short int *b, short int *c) {
   …
}
```

a och b överförs via stacken
c som register (D0)
och returvärdet som register (D0)

2.5 Parameteröverföring II

Rita stacken före funktionsanropet samt strax innan exekveringen av ’…’ i funktionen.

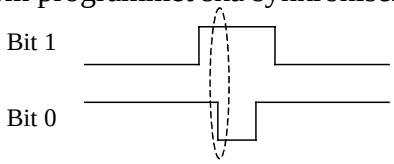
Benny Thörnberg Tel: 060-148917
Email: benny.thornberg@miun.se

3 Minnen och I/O

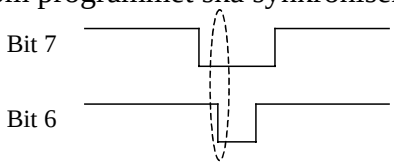
3.1 Synkronisering av I/O

Skriv ett program som synkroniserar mot en speciell händelse (specificerad i uppgiften) på en in port (IN_EVENT) och räknar antalet händelser som inträffar. Programmet skriver antalet till en ut port (OUT_EVENTS). Deklarera alla variabler och symboler på ett riktigt och logiskt sätt. Utportens beteende är endast definierat för skrivacesser. Alla I/O portar är 8-bitar.

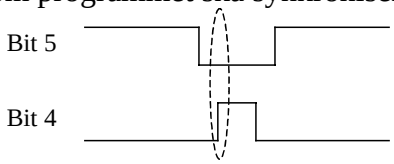
a) IN_EVENT ligger på adress $800000
OUT_EVENT ligger på adress $800001
Händelsen som programmet ska synkroniseras mot är:



b) IN_EVENT ligger på adress $FF0000
OUT_EVENT ligger på adress $FF0001
Händelsen som programmet ska synkroniseras mot är:



c) IN_EVENT ligger på adress $F000
OUT_EVENT ligger på adress $F001
Händelsen som programmet ska synkroniseras mot är:



a)

3.2 Sätt samman ett RAM minne

Sätt samman en minnesbank som är konstruerad av flera mindre minnesobjekt. Signaler till minnesbanken ska vara A(adress), D(data) samt CS*. Minneskapslarna som används för att konstruera minnesbanken har samma signaler som den färdiga minnesbanken har. Där CS* signalen är aktivt låg. Du ska indikera vilka adress/data-signaler som ska kopplas till de olika minneskapslarna.

a) Minnesbanken ska vara 16Mbytes stort organiserat med en 16 bitars databuss. Minnesbanken sätts samman med hjälp av minneskomponenter som har 4-bitars databuss och 21 bitars adressbuss.

b) Minnesbanken ska vara 2Mbytes stort organiserat med en 32 bitars databuss. Minnesbanken sätts samman med hjälp av minneskomponenter som har 4-bitars databuss och 18 bitars adressbuss.

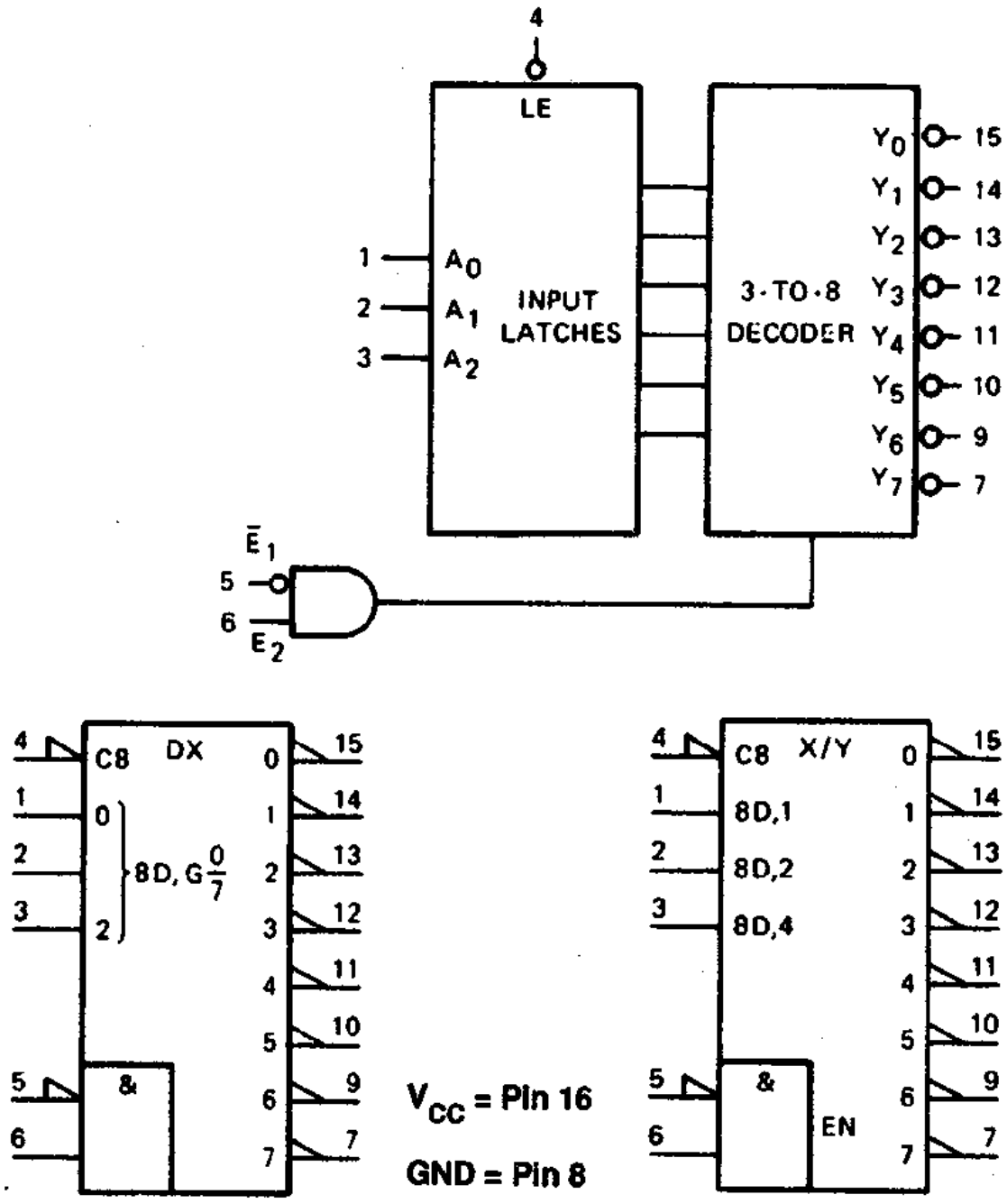
c) Minnesbanken ska vara 4Mbytes stort organiserat med en 8 bitars databuss. Minnesbanken sätts samman med hjälp av minneskomponenter som har 4-bitars databuss och 19 bitars adressbuss.

4 Datorsystem

4.1 Adressavkodare

Konstruera en adressavkodare (fullständig adressavkodning) som realiserar adresskartan definierad i uppgiften. Implementera med hjälp av logiska grindar och avkodare, se Figur 1.

Kod	Komponent	Adressområde
a)	ROM1	00 0000 - 01 FFFF
	ROM2	02 0000 - 03 FFFF
	RAM	04 0000 - 07 FFFF
	KOMP_1	80 0000 - 80 00FF
	KOMP_2	81 0000 - 81 00FF
	KOMP_3	82 0000 - 82 000F
	KOMP_4	83 0000 - 83 000F
b)	ROM1	00 0000 - 03 FFFF
	ROM2	04 0000 - 07 FFFF
	RAM	08 0000 - 0F FFFF
	KOMP_1	F0 0000 - F0 0003
	KOMP_2	F0 0004 - F0 0007
	KOMP_3	F0 0008 - F0 000B
	KOMP_4	F0 000C - F0 000F
c)	ROM1	00 0000 - 0F FFFF
	ROM2	10 0000 – 1F FFFF
	RAM	20 0000 - 2F FFFF
	KOMP_1	30 0000 - 30 00FF
	KOMP_2	40 0000 - 40 00FF
	KOMP_3	50 0000 - 50 00FF
	KOMP_4	60 0000 - 60 00FF



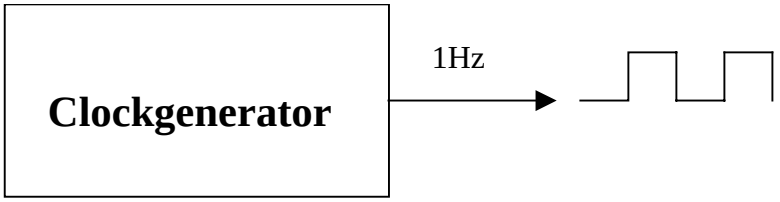
Figur 1

Benny Thörnberg
Tel: 060-148917
Email: benny.thornberg@miun.se

4.2 Avbrottshantering

Ni skall förse ett mikrodatorsystem som baseras på en Motorola 68000 CPU med ett tidtagarur som kan mäta tiden i 0 till 65535 sekunder. Tidtagningen skall vara baserad på en avbrottsrutin där avbrottet genereras av en extern klockgenerator med frekvensen 1Hz, se Figur 2.

a) Använd komponenterna i Figur 4 plus valfria grindar för att konstruera hårdvaran som ansluter klockgeneratorn i Figur 2 till mikrodatorsystemets avbrottshantering, nivå 6. För att göra anslutningen enkel skall avbrottet baseras på ett så kallat autovektoravbrott.



Figur 2

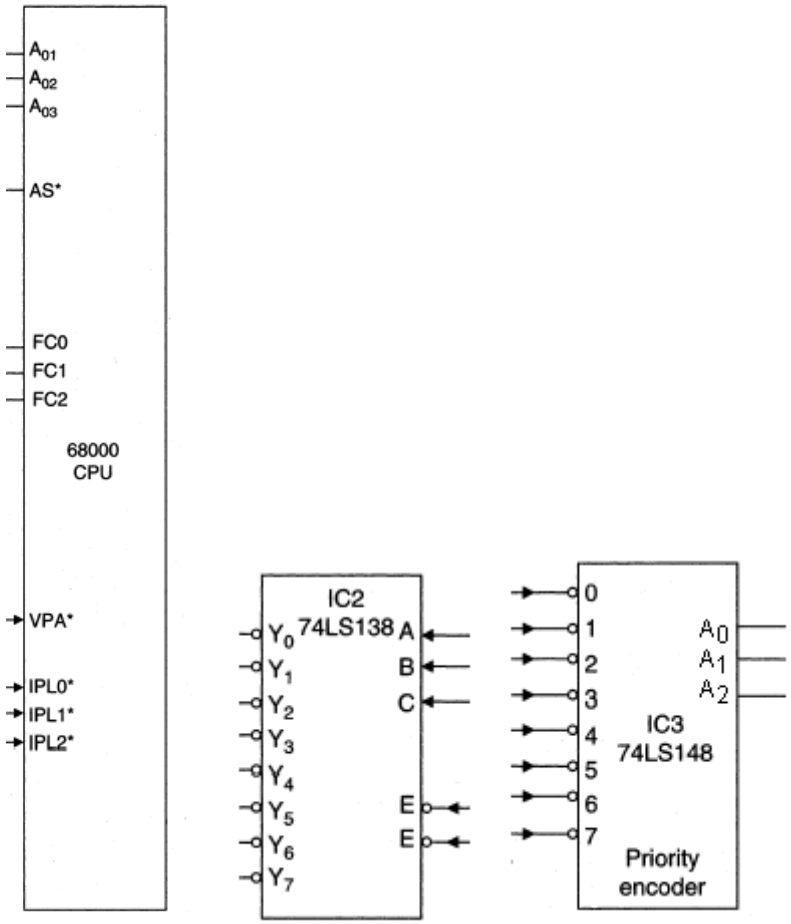
b) Skriv en avbrottsrutin som implementerar tidtagaruret. Start , Stopp och Nollställning av uret sker från ett huvudprogram genom att skriva till minnescellen Control. När bit0 ettställs så startar uret och stannar när bit0 nollställes. När bit 7 ettställs blir klockans värde noll. Se Figur 3. Tidtagarurets aktuella värde skall kunna avläsas i minnesceller enligt Figur 3.

Address	Timer variable								
4000	Control	c	x	x	x	x	x	x	s/st
4001	Counter High order byte	B15	B14	B13	B12	B11	B10	B9	B8
4002	Counter Low order byte	B7	B6	B5	B4	B3	B2	B1	B0

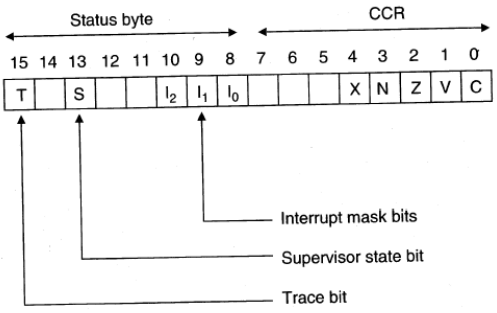
Figur 3

c) Antag att systemregistret Vector Base Register=0, Skriv ett huvudprogram som initierar avbrottsfunktionen, avbrottstabellen skall alltså uppdateras med en pekare till avbrottsfunktionen.

Benny Thörnberg Tel: 060-148917
Email: benny.thornberg@miun.se



Figur 4



FC2	FC1	FC0	Memory Access Type
0	0	0	Undefined—reserved
0	0	1	User data
0	1	0	User program
0	1	1	Undefined—reserved
1	0	0	Undefined—reserved
1	0	1	Supervisor data
1	1	0	Supervisor program
1	1	1	1ACK space (*CPU space*)

Table 6.2 The 68000's exception vector table

Vector Number	Vector (hex)	Address Space	Exception Type
0	000	SP	Reset—initial supervisor stack pointer
—	004	SP	Reset—initial program counter value
2	008	SD	Bus error
3	00C	SD	Address error
4	010	SD	Illegal instruction
5	014	SD	Divide by zero
6	018	SD	CHK instruction
7	01C	SD	TRAPV instruction
8	020	SD	Privilege violation
9	024	SD	Trace
10	028	SD	Line 1010 emulator
11	02C	SD	Line 1111 emulator
12	030	SD	(Unassigned—reserved)
13	034	SD	(Unassigned—reserved)
14	038	SD	(Unassigned—reserved)
15	03C	SD	Uninitialized interrupt vector
16	040	SD	(Unassigned—reserved)
⋮	⋮	⋮	⋮
23	05C	SD	(Unassigned—reserved)
24	060	SD	Spurious interrupt
25	064	SD	Level 1 interrupt autovector
26	068	SD	Level 2 interrupt autovector
27	06C	SD	Level 3 interrupt autovector
28	070	SD	Level 4 interrupt autovector
29	074	SD	Level 5 interrupt autovector
30	078	SD	Level 6 interrupt autovector
31	07C	SD	Level 7 interrupt autovector
32	080	SD	TRAP #0 vector
33	084	SD	TRAP #1 vector
⋮	⋮	⋮	⋮
47	0BC	SD	TRAP #15 vector
48	0C0	SD	(Unassigned—reserved)
⋮	⋮	⋮	⋮
63	0FC	SD	(Unassigned—reserved)
64	100	SD	User interrupt vector
⋮	⋮	⋮	⋮
255	3FC	SD	User interrupt vector