CODEN:LUTFD2/(TFRT-5275)/1-052/(1982)

REALTIDSKÄRNA FÖR MOTOROLAS MC 68000 MIKROPROCESSOR

THOMAS BENGTSSON

INSTITUTIONEN FÖR REGLERTEKNIK
LUNDS TEKNISKA HÖGSKOLA

JUNI 1982

LUND INSTITUTE OF TECHNOLOGY
DEPARTMENT OF AUTOMATIC CONTROL
Box 725
S 220 07 Lund 7 Sweden

Document name
MASTER THESIS

Date of issue
June, 1982

Document number
CODEN:LUTFD2/(TFRT-5275)/1-052/(1982)

Author(s)
Tomas Bengtsson

Supervisor
Bengt Sundelius, Hilding Elmqvist

Sponsoring organization

Title and subtitle
Realtidskärna för motorolas MC 68000 mikroprocessor.
(Real-Time Kernel for Motorola MC 68000 Mikro Processor.)

Abstract

A realtime kernel in Pascal for a Motorola MC 68000 mikroprocessor has been implemented to be used on ASEA:s DS101 microcomputer system.
An arbitrary number of processes can be executing concurrently. The following primitives for process synchronization are available.

1. Priority
2. Semaphores
3. Message passing.
5. Monitors.
6. Waittime.
7. Vectoried interrupt.

The processes are realised as normal Pascal programs and linked together with the kernel. The kernel itself occupies about 4-5Kbyte machine-code, and is written in a combination of Pascal and assembler. An IO-routin that handles all input and output to a terminal, like in normal Pascal.
Finally I have been writing a floppydisc-handler that usees the kernel-primitives. It can handle transfer requests from many processes at the same time.

Key words
Realtimekernel, Floppydischandler, DSCA 114 Communication card.

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title

ISBN

Language
Swedish

Number of pages
52

Recipient's notes

Security classification

ASEA

Teknisk rapport
Technical Report

TR YLK 8124

Författare - Author
Thomas Bengtsson
Godkännare - Approved by
I Trulsson
Uppdragsgivare - Requested by
B Sundelius YLKX Sun

Titel - Title
Realtidskärna för Motorolas MC 68000 mikroprocessor.

Ex arb LTH

Från - From	Datum - Date	Reg.	Sida - Page
YLKX	81-12-15	5732	0

☐ Utredning, teoretisk undersökning - Analysis, theoretical investigation
☐ Provning, experim. undersökning - Test, experimental investigation
☐ Delrapport
☒ Slutrapport
Provning/undersökning avslutad
Test/investigation finished

Ordernr - Ref. No.
Debiteras ordernr
Pkl/Akl
Antal textsidor - No. of pages of text
Antal bilagesidor - No. of supplem. pages

Sammanfattning - Summary

En väl fungerande realtidkärna för MC 68000 processorn i ASEA:s DS 101 byggsystem har realiserats. Ett godtyckligt antal processer kan exekvera parallellt. Primitiv för hantering av följande synkroniseringsmekanismer har realiserats.

1. Prioriteter
2. Semaforer
3. Händelser
4. Meddelande via brevlådor
5. Monitorer
6. Väntetider
7. Vektoravbrott

Själva kärnan omfattar 4K maskinkod och är skriven i en kombination av assembler och Pascal.

Dessutom har en IO-rutin skrivits som hanterar all in- och utmatning till en SILENT-terminal, precis som i normal Pascal.

Slutligen har en floppydischandler utvecklats. Den har dock aldrig fulländats pga fel i ett kommunikationkort (DSCA 114).

Distribution
YLK, YLKS, YLKX (7)

Enbart sida 1 - Page 1 only
KSB CU, YL, YLF, YLB, JKE, YLM, YTK, YLKH, YLKM

Nyckelord - Ämnesord	Keywords	Övriga nyckelord
·Realtidskärna	·Realtimekernel	
·Floppydischandler	·Floppydischandler	
·DSCA 114 Kommunikationskort	·DSCA 114 Communication card	

Anm. Huvudregeln är att nyckelorden skall skrivas på svenska i vänstra kolumnen.
I kolumnen för keywords införs då så önskas engelska motsvarigheter till ämnesorden samt engelska uttryck utan svensk motsvarighet.
Med övriga nyckelord avses t ex materialbeteckning, produktbeteckning, leverantör, kund, etc.

6917 0300-BA (930) 80-04 8000 Ansvarig: KSB

INNEHÅLLSFÖRTECKNING

LITTERATURFÖRTECKNING

BILAGOR

1 Programlistningar
2 Kommunikationskortet DSCA 114
3 Anslutningsenheten DSTC 120
4 Floppydiscen
5 Programlistning Floppydischandler
6 ASC II-kod tabell

0 INLEDNING

Vid mina studier på LTH i ämnet "Datorer i reglersystem II" våren 1981 undervisades vi i användandet av en realtidskärna skriven i Pascal. Denna var i LUND implementerad på en LSI-11 minidator. Vid samtal på ASEA visade de sig intresserade av att realisera en liknade kärna för deras DS 101 mikrodatorsystem baserat på en MC68000 mikroprocessor. Jag beslutade därför att skriva en liknande kärna på ASEA.

Den största delen av kärnan var skriven i Pascal och kunde användas efter små modifieringar (program kod i ref 1). Assemblerdelarna fick emellertid skrivas om.

Vid kontakt med Hilding Elmqvist på Tekniska högskolan i LUND visade det sig att han redan hade skrivit rutinerna NEWPROCESS och RESUME för ovannämnda mikroprocessor. Dessa rutiner fick jag även tillgång till.

FÖRORD

Denna rapport avslutar ett examensarbete som fullbordar studierna på linjen Maskinteknik vid LTH i Lund.

Jag vill speciellt tacka min handledare Bengt Sundelius på ASEA, som med stor skicklighet ändrat i kompilatorn, samt givit inspirerande vägledning.

Jag vill även tacka Hilding Elmqvist vid institutionen för reglerteknik vid LTH i Lund, som ställt grundidéen till förfogande.

Västerås 81-12-11

Thomas Bengtsson

1. FÖRUTSÄTTNINGAR

1.1 Maskinvara

Maskinvaran är uppbyggd av komponenter ur ASEA:s nya DS 101 mikrodatorsystem. Processorkortet, DSPC 150, är uppbyggt kring Motorolas 16-bitars mikroprocessor MC 68000. På processorkortet finns dessutom 16 Kbyte PROM och en UART för kommunikation mot en SILENT-terminal.

DSMB 110 är ett minneskort med 128 Kbyte RAM-minne. För kommunikation mot IBM finns ett synkront interfacekort och ett modem.

För kommunikation mot floppydiscen finns ett DSCA 114 4-kanals asynkront kommunikationskort. Till detta är floppydiscen ansluten via en DSTC 120 anslutningsenhet. Det finns även en mindre floppydisc DSMC 110 inkopplad på bussen. Se fig nästa sida.

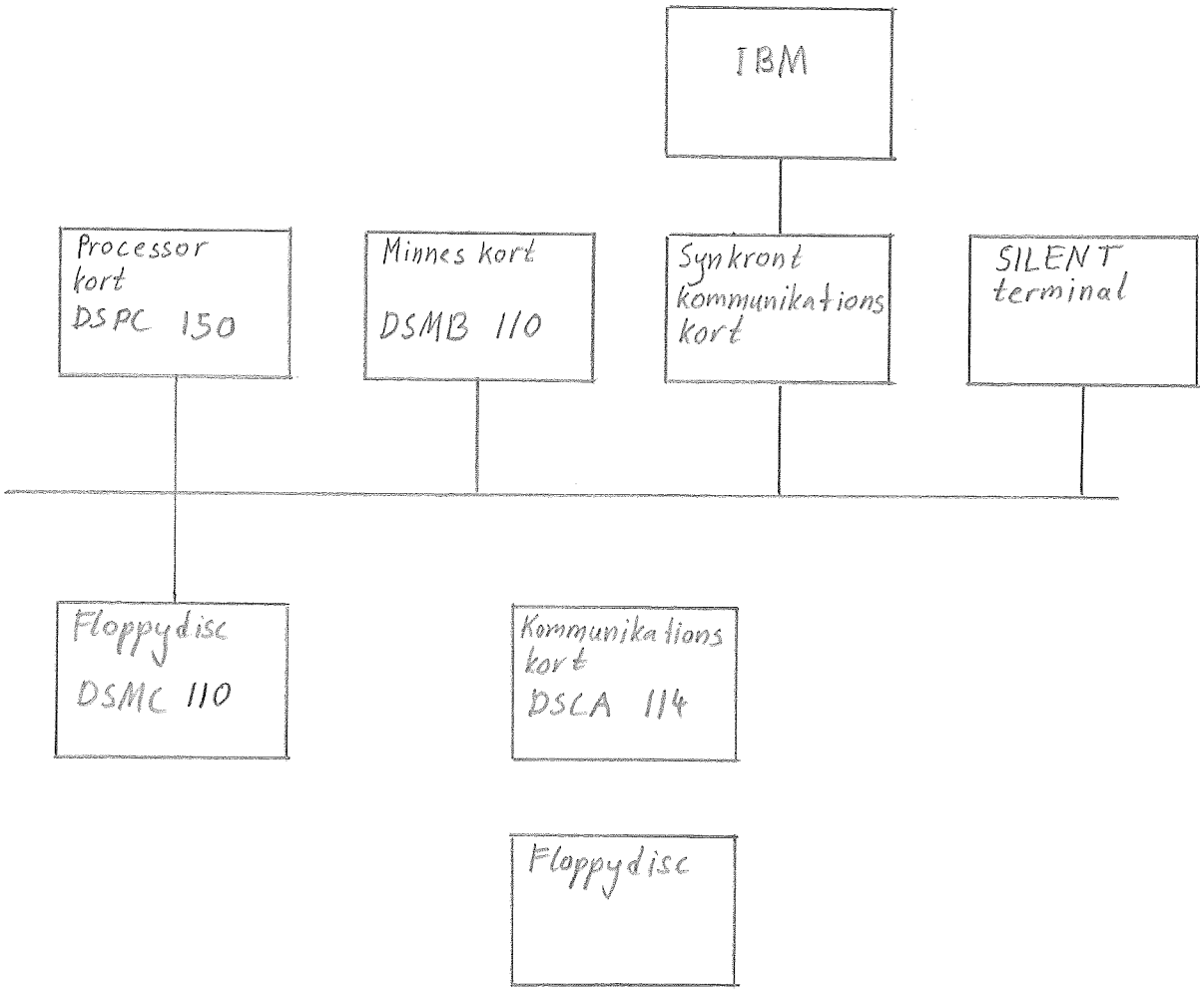


Fig 1 Maskinvarans uppbyggnad.

1.2 Utvecklingshjälpmedel

1.2.1 Editorn

Programutvecklingen har utförts på en IBM 3033 stordator. Editeringen är transaktionsorienterad. Detta innebär att man lägger upp en större mängd editeringar i en transaktion, som skickas till datorn för satsvis bearbetning. Detta system är bra vid omfattande ändringar.

IBM-maskinen har även haft en i mitt tycke för låg MTBF 3 timmar, en hel del längre stopp samt långa tider av snigelfart. Till våren skall emellertid systemet läggas om till timesharing.

1.2.2 Kompilering

Kompilatorn har använts i en på ASEA utvecklad Integer 4-mod dvs 4 bytes integer. Detta är praktiskt när man hanterar adresser, som på MC 68000 är på 32 bitar = 4 bytes.

Det har emellertid även medfört en hel del problem eftersom 3 svårupptäckta kompileringfel uppkom. Efter att jag upptäckt dem har de rättast med stor skicklighet av min handledare Bengt Sundelius.

Kompilatorn utför dels fullständig kompilering med utskrift av assembler och maskinkod, och dels enbart syntaxkontroll mot terminalen. Utskriften av maskinkod är mycket användbar när man exekverar i singlestep och med brytpunkter. Se kap 1.2.5 Debugger.

1.2.3 Assemblering

Assemblern har varit mycket bra och producerat väl valda felutskrifter.

1.2.4 Länkning

Ett exempel på länkfil finns i bil 7.

Alla program i realtidskärnan ligger för närvarande på jämna 100-tals adresser från och med 1000 H. Länkaren fungerade bra men gav den förtroendeingivande utskriften "Successfull termination of linkage" även om vissa externa referenser inte hade bytts ut. Samma förtroendeingivande utskrift fick man också, om man tagit till för lite adress-utrymme åt länkmodulerna. Detta innebär att maskinkoden för ett program hamnade ovanpå koden för ett annat.

1.2.5 MACSBUG

MACSBUG är namnet på Motorolas debuggingprogram för MC 68000. Under MACSBUG kan man läsa och skriva i minnesceller och register, sätta brytpunkter och köra singlestep. Man saknar dock möjligheten att sätta brytpunkter på Pascalnivå, och att kunna få en utskrift av i Pascal deklarerade variabler.

1.3 Avbrottssystemet på MC 68000

1.3.1 Avbrottsmasken

På mikrodatorn finns 7 stycken vektoriserade avbrottsnivåer. Statusregistret innehåller en 3-bitars mask som indikerar vilken prioritet processorn för tillfället har. Ett avbrott måste ha en prioritet högre än masken för att accepteras. Sätts masken till nivå 7 är avbrottssystemet frånslaget. I realtidskärnan finns för närvarande tre typer av avbrott.

1. Från intervallklockan (nivå 6)
2. Från terminalen (nivå 3)
3. Från kommunikationskortet (nivå 4)

Därför representeras enableinterrupt med att masken sätts till 2 och disableinterrupt med att masken sätts till 7. Programkod bil 1 s 1.

1.3.2 Processorns arbetstillstånd

Statusregistret innehåller även bit som markerar processorns status. Det finns två varianter, supervisorystate = SS och userstate = US. SS kännetecknas av att man har tillåtelse att manipulera statusregistret, och att man jobbar mot supervisorystackpointer = SSP. US kännetecknas av att man inte får ändra i statusregistret och att man jobbar mot USP.

I uppstarts programmet LOAD bil 1 s 1 sätts statusregistret till 2700 H, dvs man arbetar mot SSP och avbrottsmasken = 7.

1.3.3 Exception processing

Ett exception kan orsakas av i huvudsak 3 skäl:

Fall 1. Avbrott från yttre enhet

Fall 2. Exekvering av TRAP 0-15

Fall 3. Av otillåten operation ex: Dividera med 0
Udda long eller word adress
Ej implementerad adress
Ej implementerad instruktion

Endast fall 1 styrs av avbrottsmasken, de övriga är ovillkorliga.

Vid ett avbrott sker följande:

1. Nuvarande instruktion exekveras färdigt (Fall 1).
2. Jämför: MASK>=Avbrottsnivån => avbrottet behandlas ej (Fall 1).
3. MASK: = Avbrottsnivån (Fall 1).
4. SR och PC stackas på SSP (se kap 1.3.2)

5a. Ur avbrottsnivån beräknas en vektoradress (Fall 1).

5b. Ur TRAP-numret (0-15) beräknas en vekrotadress (Fall 2).

5c. Ur typen av fel bestämmer vektoradressen (Fall 3).

6. Innehållet på vektoradressen flyttas till PC och exekvereingen fortsätter nu i avbrottsrutinen.
7. Avbrottsrutinen avslutas att en RTE = "Return from exception" instruktion exekveras.

1.4 Intervallklockan

Intervallklockan ger avbrott på nivå 6. Det medför att avbrottsrutinens startadress skall skrivas till adressen 78 H se fig 2.

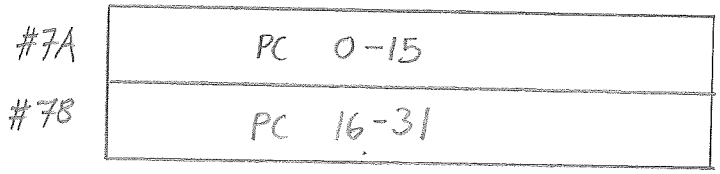


Fig 2 Definition av startadress.

Intervallklockan initieras som realtidsklocka genom att 35 H, 60 H och B6 H skrives till adressen FF0007 H. Se fig 3.

Fig 3 Initiering av intervallklockan.

Intervalltiden beräknas enligt:

T = (LSB * 0,02) + (MSB * 5) och fås i millisekunder.

I mitt fall har jag valt T = 50 millisekunder vilket ger LSB = 00H och MSB = 0AH.

Detta sköts av assemblerproceduren INITRTC bil 1 s 8, som anropas av processen CLOCK bil 1 s 24. Avbrott möjliggöres genom att sätta bit 1 i Controlregistret på adressen FF0029H.

Avbrotten kvitteras genom att man nollställer bit 12 i ett Flaggregister på adressen FF003EH, detta sköts av assemblerproceduren ACKRTC bil 1 s 8.

Fig 4 Definition av intervalltid.

1.5 Terminalanslutning

Terminalen är en SILENT 733ASR skrivare. Den är ansluten via en V24-kontakt till en UART på processorkortet. UART:en ger avbrott på nivå 3. Vektoradressen är 6CH. Avbrottsorsaken bestämmes av interrupt-enable-registret på adressen FF0023H.

Fig 5 Interruptenable registret.

Om bit 0 är ettställd, får man avbrott då data mottagits. Kvittens genom att läsa data.

Om bit 1 är ettställd, får man avbrott då data sänts. Kvittens genom att läsa Interrupt Identification registret på adressen FF0025H.

Data läses och skrives i Transmitter/Receiver registret på adressen FF0021H.

Programmering av UART:en ombesörjes av MACSBUG, och den används i realtidskärnans IO-rutin bil 1 s 10.
Kvitteringen av UART:ens avbrott sköts av proceduren DISABLE bil 1 s 13.
För närmare beskrivning av UART:en se bil 8.

2 REALTIDSKÄRNAN

2.1 Allmänt

Realtidsprogrammering sker för närvarande i assembler av två skäl, dels blir ett program i högnivåspråk ofta långsammare, och dels är det svårt eller oftast omöjligt att manipulera med processorns register.

Ett sätt att lösa snabbheten är genom att använda allt snabbare språk, kompilatorer och processorer.

En lösning på det andra problemet är att skriva ett litet operativsystem, som kan sköta en stor del eller till och med all registerhantering. I detta realtidsoperativsystem kan man bygga upp primitiv, som sköter de delar som kräver registerhantering, och kan anropas från ett högnivåspråk. En naturlig utveckling av detta operativsystem är att införa hantering av många samtidiga program (processer).

Börjar man studera problemet närmare (ref 1) finner man att det blir väldigt naturligt att dela upp programkoden i två delar. En Pascalnivå som sköter köhantering och en assemblernivå som sköter registerhantering. På assemblernivån kap 2.3 sköter man växlingen mellan olika processer, och på Pascalnivån kap 2.4 väljer man vilken process som skall få exekvera. Se fig 6.

Som högnivåspråk för mikrodatorprogrammering använder man Pascal på ASEA. Fördelen med Pascal är att man kan hantera köer på ett trevligt sätt. Nackdelen är att språket finns i många varianter på ASEA, vilket gör att programkoden blir maskinberoende.

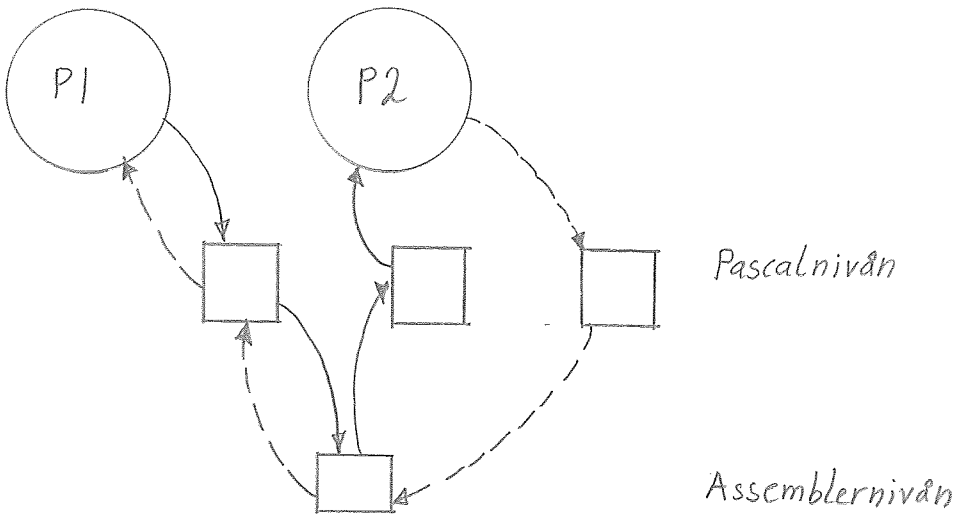


Fig 6 Kärnans uppbyggnad.

2.2 Minnesutnyttjande och registeranvändning

2.2.1 Runtimeorganisationen i Motorolas Pascal.

Minnesanvändningen visas i fig 7. Alla globala variabler adresseras relativt register A5. Variabler på mellanliggande nivåer nås via den statiska länken, SL, och lokala variabler adresseras relativt register A6. De olika SL stackas på adressen 8(A5) och uppåt. Register A4 används som tillfällig lagringsplats för den aktuella SL:en. De övriga registern D0-D6 och A0-A3 används vid utförandet av en Pascalsats.

På grund av ett fel i kompilatorn används egentligen inte register A3, men kärnan utgår ifrån att det används för säkerhets skull. Följaktligen innehåller endast register A4-A7 information efter en fullbordad Pascalsats.

2.2.2 Programstart

Göres kompileringen i A-mod länkas en uppstartsmodul till programmet.

```
XREF      STACKBAS
XREF      HEAPBAS
LEA.L     STACKBAS,A5
MOVE.L    A5,A6
MOVE.L    A5,A7
LEA.L     HEAPBAS,A0
MOVE.L    A0,4(A5)
BRA       MAIN
```

Vid B-mod finns ingen uppstartsmodul. Moden bestäms av styrkorten till kompilatorn.

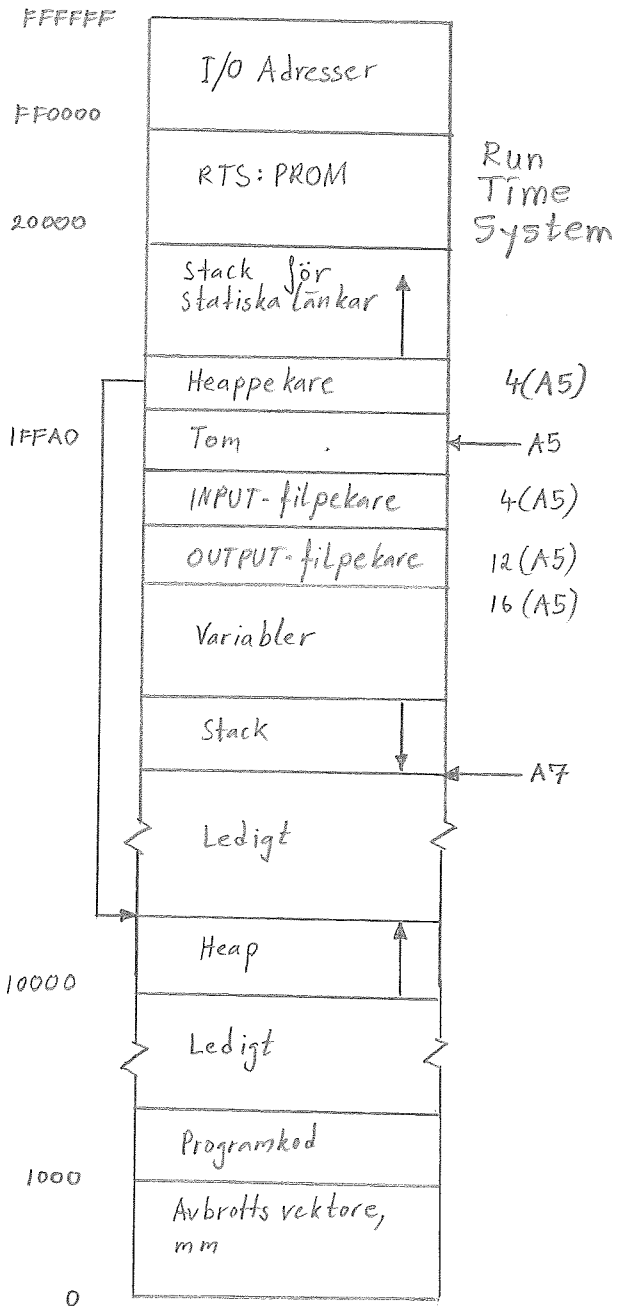


Fig 7 Minnesorganisationen i normal Pascal.

2.2.3 Processernas realisering

En grundläggande förutsättning för parallella processer i Pascal är att varje process har en egen stack. Detta införes genom att ge register A5, A6 och A7 ett lämpligt startvärde. För att även ge varje process en egen heap behöver man bara föra in en lämplig heapbas på adressen 4(A5) = Heappekaren se fig 7.

I den ursrungliga idéen ref 1 representeras en process av en parameterlös procedur. På ASEA föredrog man dock att representera en process som ett normalt Pascalprogram. Detta ger följande fördelar

1. Variablerna i monitorn blir bättre skyddade, kap 4.3.

2. Utskrift och inläsning blir enklare eftersom varje process har egna input och output filer.

och följande nackdelar.

1. Extern kompillering ger missad typkontroll vid anrop av externa procedurer.

2. Kräver större eftertänksamhet vid uppläggning av den globala dataarean.

För att ge alla processorna tillgång till globala data på ett smidigt sätt placeras dessa i ett KERNEL-record, detta placeras genom pascal instruktionen new i heapen. Därigenom behöver bara pekaren till detta record överföras till de olika processorna. Närmare bestämt till adressen -16(A5).

2.2.4 Minnesorganisation vid processanvändning

Organisationen finns i stora drag i fig 8 och i detalj för en process i fig 9. Observera att huvudprogrammet, CONCURRENT, finns på samma ställe som vid normal Pascal.

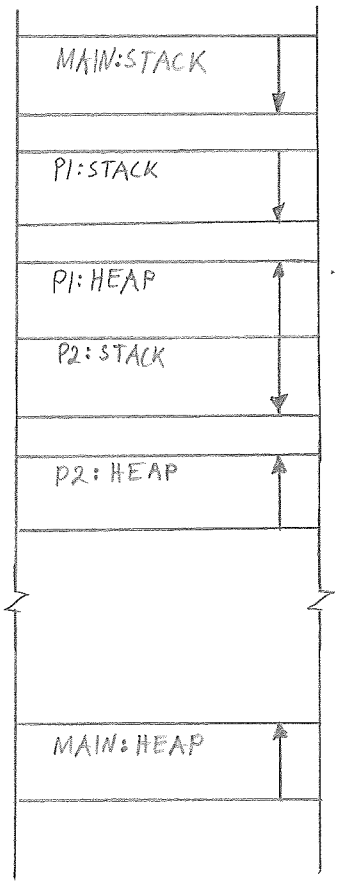


Fig 8 Minnesorganisationen vid användning av processer.

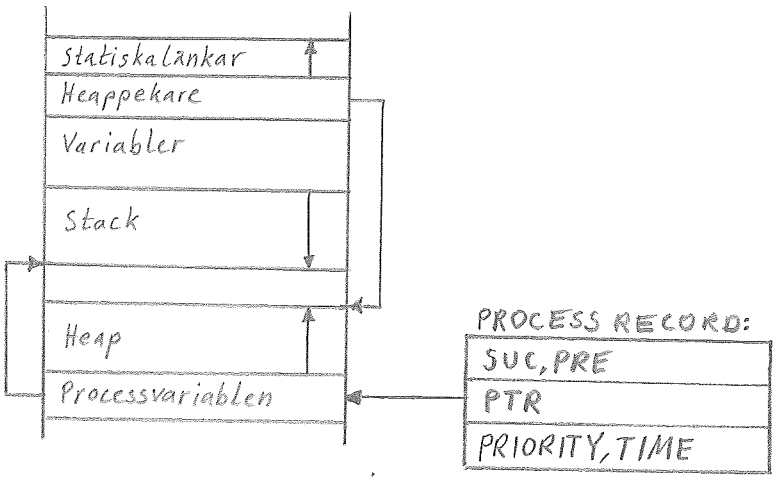


Fig 9a En väntande process's minnesorganisation.

Processvariabeln är en pekare till det uppstartsblock som placerades på stacken då processen avbröts. Startblocket innehåller en återhoppsadress till den rutin i kärnan som behandlade avbrottet, de undansparade registerna samt en återhoppsadress till den avbrutna processen. Se fig 9b.

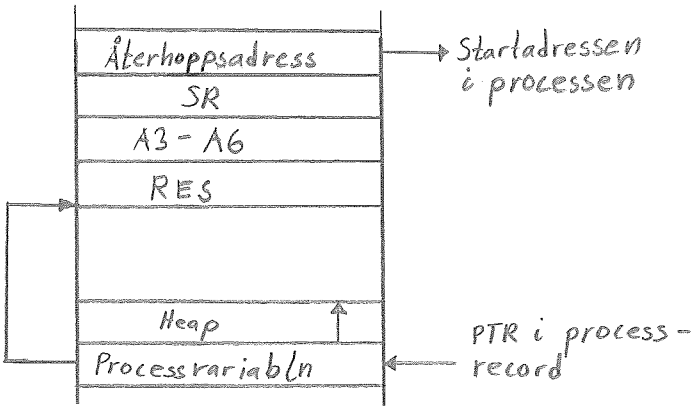


Fig 9b Exempel på startblock genererat av proceduren RESUME

2.3 Assemblernivån

I denna, den innersta nivån av realtidskärnan sker följande.

1. Initiering	INITNUCLEUS
2. Processer skapas	NEWPROCESS
3. Byte av process	RESUME
4. Avbrottshantering	IORESUME, INTRESUME
5. Överföring av pekare till brevlådor.	SENDADRESS, PUTADRESS

2.3.1 Processorn

MC 68000 är en processor med 16 bitars databuss och 24 bitars adressbuss. Den har 8 dataregister D0-D7 och 8 adressregister A0-A7.

Register A7 är stackpekare. Det finns 2 stycken A7, stackpekare, vilken man använder bestäms av statusregistret se kap 1.3.2. Processorn har även en 32-bitars programräknare.

Det finns fem data typer

1. Bitar	(1 bit)
2. BCD	(4 bitar)
3. Bytes	(8 bitar)
4. Words	(16 bitar)
5. Long words	(32 bitar)

och sex adresseringsmoder.

Register Direct Addressing Data Register Direct Address Register Direct	MOVE A0,D0	EA = Dn EA = An
Absolute Data Addressing Absolute Short Absolute Long	MOVE FF09,A0	EA = (Next Word) EA = (Next Two Words)
Program Counter Relative Addressing Relative with Offset Relative with Index and Offset	BRA LABEL	EA = (PC) + d_{16} EA = (PC) + (Xn) + d_8

Register Indirect Addressing
Register Indirect
Postincrement Register Indirect
Predecrement Register Indirect
Register Indirect With Offset
Indexed Register Indirect With Offset
	MOVE A0,-(A7)
	MOVE (A7)+,A0
	MOVE 6(A0,D0),A1
		EA = (An)
		EA = (An), An← An + N
		An← An - N, EA = (An)
		EA = (An) + d_{16}
		EA = (An) + (Xn) + d_8

Immediate Data Addressing Immediate Quick Immediate	MOVE #37,A0	DATA = Next Word(s) Inherent Data
Implied Addressing Implied Register	MOVE #37,SR	EA = SR, USP, SP, PC

Fig 10 Adresseringsmoder.

För närmare beskrivning se ref 5.

2.3.2 Dataarean

Assemblernivåns dataarea består bara av en variabel, en 4 bytes minnescell som kallas CURRENT (bil 1 s 1). CURRENT är en kopia av den exekverande processens processvariabel. Processvariablen är en pekare till processens uppstartadress och anger toppen på stacken.

2.3.3 Initiering

Initieringen sker genom ett anrop till INITNUCLEUS:

```
Procedure INITNUCLEUS (SREQ: Integer4; var FREETOP:
          Integer4; var MAIN:PTR);
```

Programkod bil 1 s 2. PTR är processvariabeln. Ett anrop till denna procedur medför att huvudprogramet, CONCURRENT, blir en process associerad med MAIN. Endast stackbehov SREQ behöver anges, hur det beräknas visas i kap 2.6.2. FREETOP ger tillbaka en pekare till gränsen mellan allokerat och ledigt minnesutrymme.

2.3.4 Skapa nya processor

För att en process skall kunna skapas krävs:

1. En startadress
2. A5: = A6: = A7: = Stackbasen (Kap 2.2.2)
3. 4(A5): = heapbasen (Kap 2.2.2)

Detta sköts normalt av en uppstartsmodul, men genom att kompilera programmet i B-mod genereras ingen uppstartsmodul. Punkt 1-3 sköts av en assemblerprocedur:

```
NEWPROCESS (STARTADR, STACKBAS, HEAPBAS: Integer4;
            var CHILD: PTR);
```

Programkod bil 1 s 3.

Denna procedur skapar en process med en minnesarea enligt fig 11. Den lägger även ut vissa adresser i processens globala dataarea via assembler proceduren PUTADRESS bil 1 s 9. Dessa adresser är:

1. Pekare till KERNELREC -16(A5).
2. Pekare till de olika brevlådorna konsekutivt med början på -20(A5).

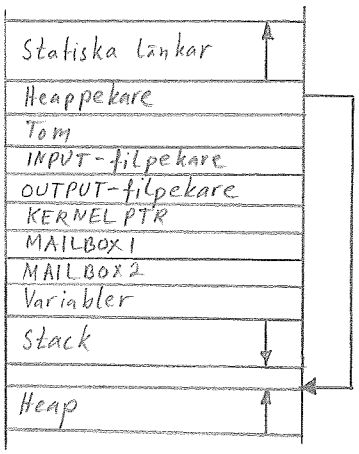


Fig 11 Detaljerad processbild.

Ur samma processinstans (startadress) kan många processor skapas med egna dataareor. CHILD är processvariablen med vilken den nya processen associeras.

2.3.3 Processbyte

Ett processbyte initieras av antingen ett anrop till ett primitiv i kärnan eller av ett avbrott.

1. RESUME	Normalt processbyte
2. IORESUME	Processen vill vänta på vektoravbrott
3. INTRESUME	Vid avbrott

Dessa 3 procedurer är symmetriska i det avseendet att de har en halva som bygger upp ett speciellt startblock, och en halva som utnyttjar startblocket vid återstarten av processen.

2.3.4 Normal återstart

Ett anrop enligt RESUME (PTR: Integar4), programkod bil 1 s 4, där PTR är processvariabeln. Detta innebär att den exekverande processen blir avbruten och processen associerad med PTR får starta.

2.3.5 Avbrottsstyrd processväxling

När kärnan vill att en process ska vänta på avbrott anropar den IORESUME (VECADR: Integer2; PTR: Integer4); bil 1 s 5. Detta innebär att processen associerad med PTR börjar exekvera och att den exekverande får vänta med stackinnehåll enligt fig 12.

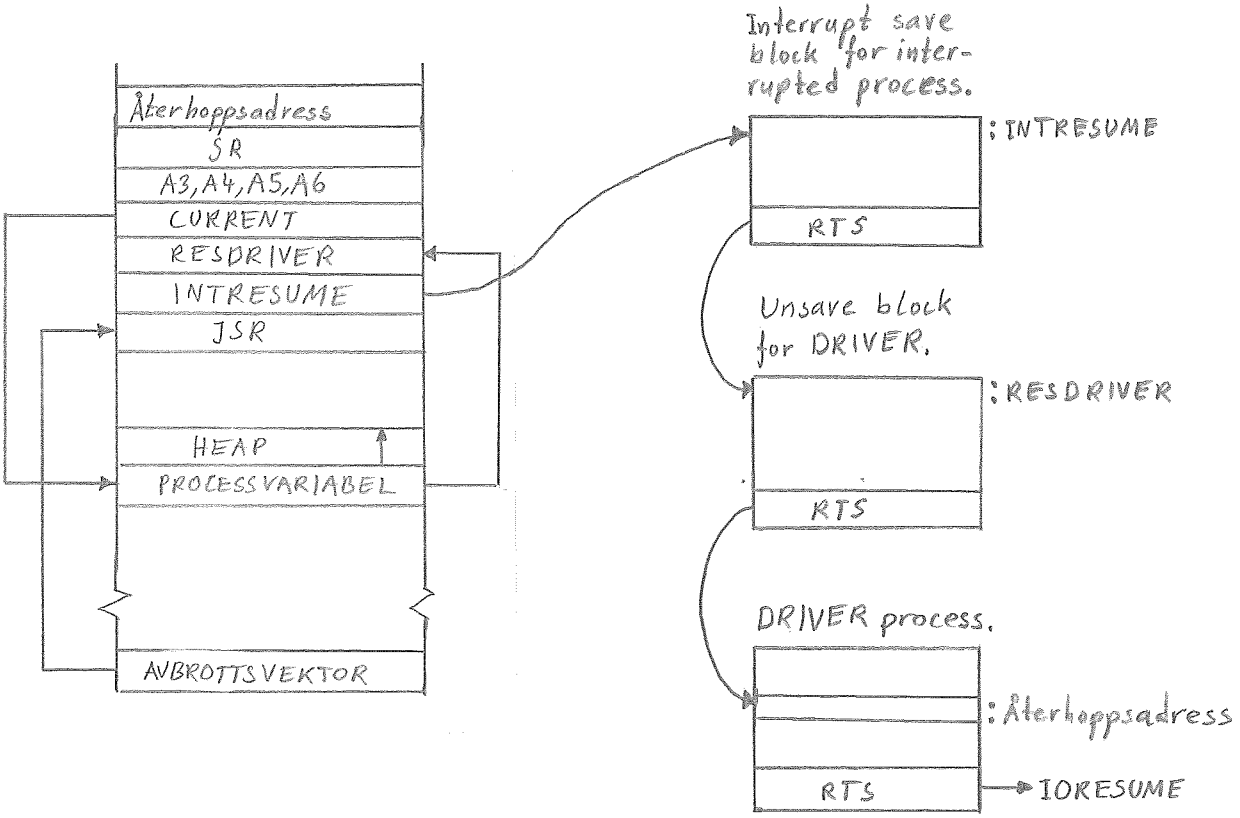


Fig 12 Stackinnehåll för process som väntar på avbrott.

För att interruptrutinen skall veta vilken process som skall återstartas lägger man upp instruktionen JSR INTRESUME på stacken. Detta innebär att återhoppsadressen till processen som väntade på avbrott stackas, dvs hamnar som om den vore en parameter vid ett procedur-anrop.

Procedure INTRESUME (IOSUSPENDED: PTR);

Programkod bil 1 s 6. I denna procedur stackas alla den avbrutna procedurens register, eftersom man inte vet hur långt den var i en Pascal instruktion. Stack-innehåll enligt fig 13.

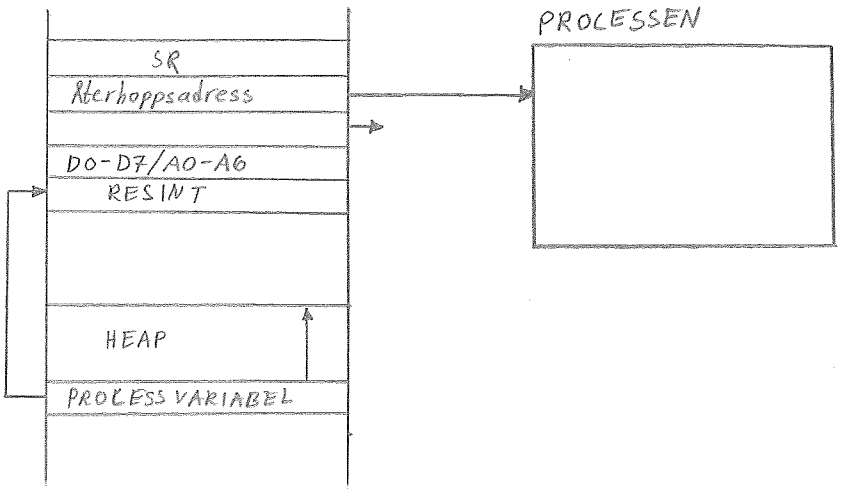


Fig 13 Stackinnehåll för en process, avbruten av ett avbrott.

2.4 Pascalnivån

På denna nivå realiseras de primitiv som användaren kan utnyttja.

För att förenkla fortsättningen skall jag börja med att beskriva kärnans uppbyggnad bil 1 s 16.

Varje process har en speciell dataarea PROCESSREC, varje semafor sitt SEMAFORREC, och varje händelse sitt EVENTREC. Med alla dessa dataareor är köer förknippade i vilka processrecords kan placeras in. Köerna implementeras som dubbelt länkade listor med listhuvudet av samma typ som elementen för att slippa hantera tomma listor. Se fig 14, 15.

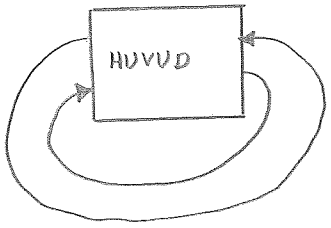


Fig 14 Tom lista.

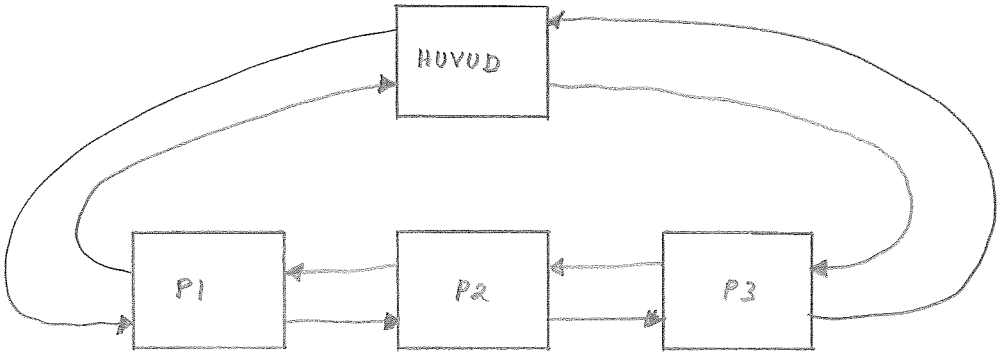


Fig 15 Lista med 3 väntande processer.

De köer som skapas vid initieringen i CONCURRENT är READYQUE och TIMEQUE. Köerna för händelser och semaförer kan skapas dynamiskt under exekveringen. Ett exempel på ett tillstånd i kärnan i fig 16.

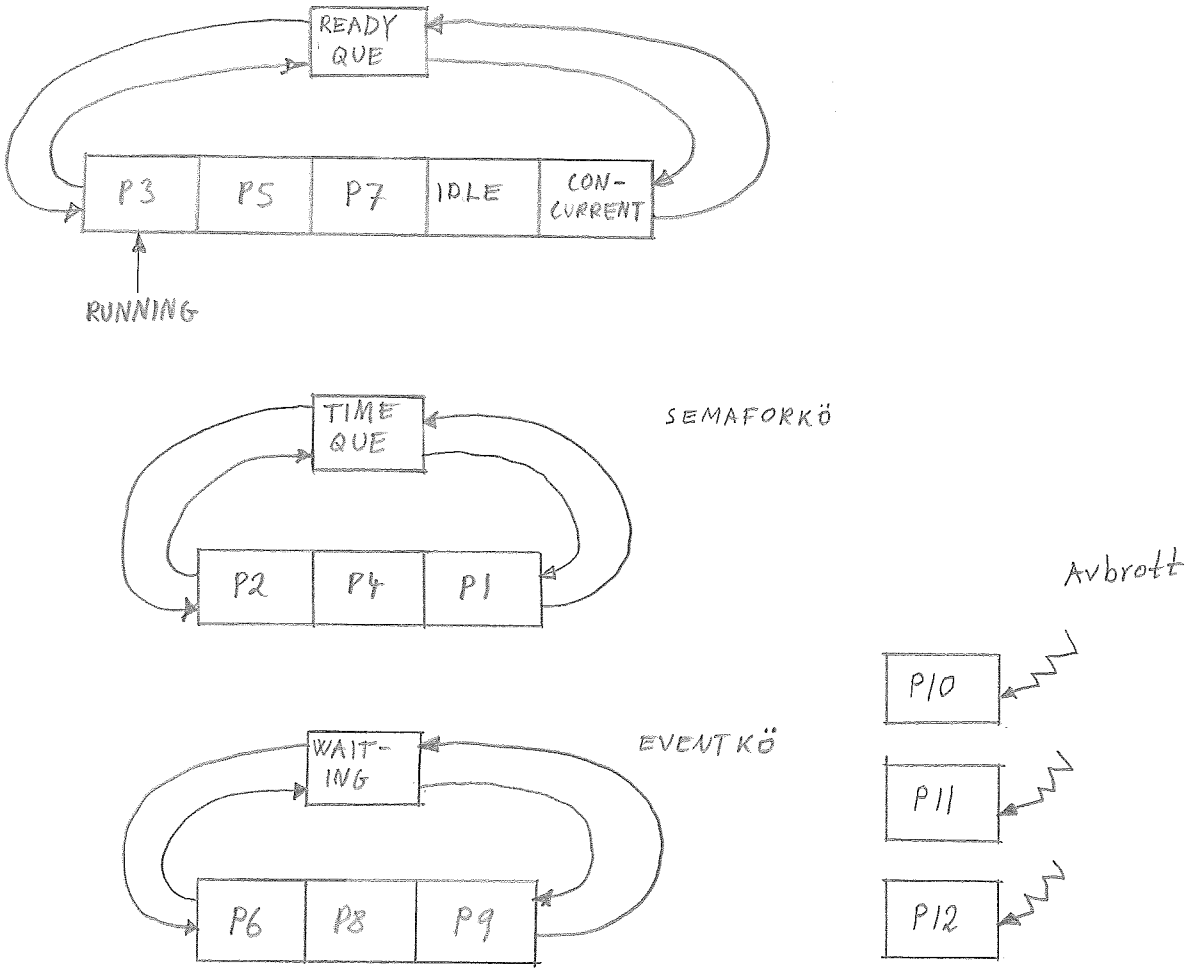


Fig 16 Köstruktur i kärnan.

För köhantering finns 3 interna procedurer.

1. PUT (Bil 1 s 17)
2. REMOVE (Bil 1 s 18)
3. PUTPRIORITY (Bil 1 s 17)

Beskrivning i samband med programkoden.

Det finns även procedur SCHEDULE bil 1 s 20 som de övriga primitiv anropar då de påverkat READYQUE. SCHEDULE går in och byter, om det inte är rätt process som skall börja exekvera.

2.5 Prioritetsbegreppet

Prioriteten används för att fastslå exekveringsordningen då många processer är körklara.

Ett högt värde på prioriteten exekverar före ett lågt. Är prioriteten större än eller lika med maxpriority = 1000 exekverar processen med avbrottssystemet frånslagit, exempel processen CLOCK. Processen IDLE har prioriteten noll, vilket medför att denna alltid exekverar då processorn är ledig.

Jag har valt att tolka negativa prioriteter som en identitet för en passiv process. En process med negativ prioritet ligger bakom IDLE: READYQUE, och kommer således aldrig att exekvera.

Följande primitiv påverkar en process's prioritet.

1. SETPRORITY
2. ACTIVATE
3. PASSIVATE

Se kap 4.5

2.6 Realiserade primitiv

En lista på alla primitiven finns i bil 1 s 38.

2.6.1 INITKERNEL (bil 1 s 18)

Denna procedur skapar själva kärnan och gör huvudprogrammet till en process. Man måste ange stackbehov, hur detta beräknas visas i kap 2.6.4. Denna procedur skapar även processerna IDLE och CLOCK.

2.6.2 CREATEPROCESS (bil 1 s 18)

Denna procedur skapar och startar en process av typen NR där nummer är index i en tabell över startadresser, PROCESSTAB. FINDPC (bil 1 s 7) hämtar fram en startadress utifrån ett process NR.

NR = 0	IDLE
NR = 4	CLOCK
NR = 8	FLOPPYDISCHANDLER
NR = 12, 16, 20 osv	för användarens processer.

Processernas prioritet sättes till 10.

Man anger även MEMREQ som anger hur mycket minnesbehov man vill ha allokerat för stack + heap. Minnes behovet beräknas enligt:

MEMREQ: = (Antalet filer) * 264 + (Antalet bytes dataarea) + (Antalet bytes heapbehov) + (80 bytes för strängar) + (ca 100 bytes för processorns stackoperationer)

2.6.3 IDLE (Bil 1 s 24)

Denna process sätter sin prioritet till noll och exekverar när processorn är ledig. NR = 0.

2.6.4 CLOCK (Bil 1 s 24)

Denna process uppdaterar väntetiden för den första processen i TIMEQUE, de andra processernas väntetider är relativt den förstas. Blir en eller flera väntetider noll överföres dessa processer till READYQUE. CLOCK aktiveras av intervallklockan var 50:de millisekund. NR = 4.

2.6.5 SETPRIORITY (Bil 1 s 19)

Denna procedur flyttar en process i READYQUE. Det finns 3 fall.

1. PRIORITY >= MAXPRIORITY => Exekverar med avbrottssystemet frånslaget.
2. MAXPRIORITY > PRIORITY >= 0 => Avbrotssystemet tillslagit.
3. PRIORITY < 0 => Den negativa prioriten ger processen en prioritet enligt vilken den kan återstartas av ACTIVATE.

2.6.6 ACTIVATE (bil 1 s 19)

Denna procedure tar en eller flera processer, som ligger i READYQUE med negativ prioritet, och återstartar dessa med en ny positiv prioritet. Detta kan användas för att återstarta en eller flera processer, som ligger i READYQUE med negativ prioritet.

2.6.7 PASSIVATE (bil 1 s 20)

Denna procedur tar en process ur READYQUE och placerar den med negativ prioritet i READYQUE. Dvs processen passiviseras.

2.6.8 INITSEM (bil 1 s 21)

En ny semafor skapas och dess räknar sätts till INITVAL.

2.6.9 WAIT (bil 1 s 21)

Är semaforens räknare lika med 1 fortsätter exekveringen, annars placeras processen i en väntekö.

2.6.10 SIGNAL (bil 1 s 21)

Finns det någon eller några processer i väntekön med högre prioritet kommer den första av dessa (högst prioritet) att starta, vid varje signal operation. Är kön tom markeras semaforen som ledig dvs COUNTER: = 1;

2.6.11 INITEVENT (bil 1 s 22)

INITEVENT skapar en kö över processer som väntar på händelsen med vilken event-varabeln är förknippad. För att få ömsesidig uteslutning då man ändrar en event-variabel finns det en skyddssemafor associerad med händelsen.

2.6.12 AWAIT (bil 1 s 23)

AWAIT innebär att processen placeras i DELAYEDQUE och semaforen markeras som ledig.

2.6.13 CAUSE (bil 1 s 22)

CAUSE innebär att alla processerna i DELAYEDQUE flyttas över till READYQUE dvs startas.

2.6.14 WAITTIME (bil 1 s 23)

Processen placeras i TIMEQUE. Väntetiden för alla de väntande, utom den första, är relativ. Den första processens väntetid är absolut. Ett tick är 50 msek. Maximal väntetid = 2^{32} 50 * 10^{-3} * 7 år.

2.6.15 WAITIO (bil 1 s 23)

För att göra denna procedur generellt användbar har jag tvingats ändra den i förhållande till grundidéen ref 1. Detta har medfört att användning kräver 5 steg.

1. Anropa DI.
2. Enable det avbrott du vill vänta på.
3. Anropa WAITIO.
4. Kvittera avbrottet du väntade på.
5. Anropa EI.

2.6.16 INITMAILBOX (bil 1 s 26)

Brevlådor skapas och pekare till dem överföres till processernas dataareor konsekutivt efter KERNELPTR av SENDADRESS (bil 1 s 26), för att underlätta användningen.

2.6.17 SENDMESSAGE (bil 1 s 26)

En pekare till ett meddelande överföres till en FIFO-kö. Pekarens NEXTMESS-referens måste vara "nil" första gången en pekare skickas.

2.6.18 RECEIVEMESSAGE (bil 1 s 26)

En pekare tages emot från FIFO-kön. Finns det inget meddelande i brevlådan väntar processen tills det kommer ett.

3 IN OCH UTMATNING TILL TERMINALEN

Vid anrop av readln eller writeln gör man ett hopp till runtimesystemet = RTS, som behandlar och omformar olika format till en följd av ASCII-tecken i INPUT eller OUTPUT filen. I RTS exekverar man sedan en TRAP 15 instruktion. Detta medför normalt ett hopp till IO-rutinerna i MACSBUG. MACSBUG sköter sedan in och utmatning. För att kunna få processorn att arbeta med annat i stället för att göra "busywait" på terminalen, har jag skrivit egna IO-rutiner. Genom att ändra den med TRAP 15 associerade vektorn på adressen BCH till startadressen för mina IO-rutiner behöver jag inte ändra i RTS. Detta göres av assembler proceduren LOAD i bil 1 s 1. LOAD ändrar TRAP-vektorn och gör sedan ett ovillkorligt hopp till PASS-START. Detta är ett läge som kompilatorn lägger upp i CONCURRENT vid A-mod på kompilatorn.

IO-rutinerna bil 1 s 10 låter nu processorn göra annat medan den väntar på UART:en.

Varje rad utskrives respektive inläses odelad. Vill man skriva ut många rader odelat får man införa en överordnad semaforer.

UART:en initieras av MACSBUG. Kvitteringen sker i assemblerprocedure DISABLE (bil 1 s 13).

För ömsesidig uteslutning vid utskrift av en rad utnyttjas:

1. INITPRINTER (Bil 1 s 27)
2. REQPRINTER (Bil 1 s 27)
3. RELPRINTER (Bil 1 s 27)

Innan RTS anropar mina IO-rutiner sätter den A5 och A6 som buffertpekare. För att kunna anropa primitiven enligt ovan samt WAITIO måste register A5 innehålla en process's stackbas. Detta värde finns på stacken och fiskas fram till en minnescell. När man skall eka ett inläst tecken finns A5 på ytterligare ett annat ställe, se stacknings förfarande i RTS. Vid anrop blir register D0 en parameter som talar om vad som skall utföras.

IO-rutinerna kan för närvarande användas precis som i "vanlig" Pascal.

4 ANVÄNDNING

4.1 Semaforer

Semaforer används framför allt för ömsesidig uteslutning. För alla andra användningsområde se ref 1.

Vid ömsesidig uteslutning måste räknaren initieras till 1.

ANROP: initsem (semafornamn, 1);

Användning i monitorprocedur för att nå data i monitor.

```
Procedure (* Entry *) GETDATA;
Begin
     wait (semafore);
     USE DATA;
     signal (semafor):,
end.
```

Data eller resurser kan nu användas med ömsesidig uteslutning garanterad. Glömmer man göra wait eller signal kan svårupptäckta fel uppstå. I semaforkön ligger processerna i prioritetsordning. Vid en signal operation kommer bara en, de första i kön, att få starta.

4.2 Händelser

Händelser används för att ge processerna en möjlighet att vänta selektivt.

Här beskrivs 3 användningsområden.

Fall 1 Vänta tills data i en monitor uppfyller ett visst villkor.

Fall 2 Vänta tills data i en monitor ändrats.

Fall 3 Vänta på besked av typen starta nu.

4.2.1 Fall 1

```
MONITOR:  var DATA = record
                       mutex: semafor;
                       change: event;
                       data: datatyp;
                     end;

I MONITOR PROCEDUR:
          wait (mutex);
          while ¬ condition do await (change);
          (*USE DATA*)
          cause (change);
          signal (mutex);

INITIERING:
          initsem (mutex, 1);
          initevent (change, mutex);
```

Varje process som ändrar i monitorn måste göra cause (change); i en monitor procedur:

```
          wait (mutex);
          cause (change);
          signal (mutex);
```

4.2.2 Fall 2

Förutsätter samma monitor som i fall 1.

```
ENTRY PROCEDUR:
          Procedur (* entry *) WAITDATA;
          Begin
              wait (mutex);
              await (change);
              (*USE NEW DATA*);
              signal (mutex);
          end.
```

4.2.3 Fall 3

```
MONITOR:  var DATA = record
                       mutex: semafor;
                       startorder: event;
                     end;

MONITOR PROCEDUR:
          Procedur  ( * ENTRY * ) WAITORDER;
          Begin
             wait (mutex);
             await (startorder);
             signal (mutex);
          end.

MONITOR PROCEDUR:
          Procedur (*ENTRY*) GIVEORDER;
          Begin
             wait (mutex);
             cause (startorder);
             signal (mutex);
          end.
```

Observera att event-variabeln använd som en gemensam resurs. Efter begin i en monitor procedur måste i verkligheten följa en with-sats.

```
with KERNEL^. DATA do
```

Förklaring se 4.7.3 KERNELREC. Exempel på monitor procedurer bil 1 s 23.

4.3 Monitorer

En monitorer är ett record med gemensamma data för många processer.

```
EX:     type  MONITOR = record
                  OK: Semafor;
                  DATA: Integer;
              end;

DATAACESS:        wait (OK);
                  (*COPY DATA*)
                  signal (OK);
```

För att minska risken att data läses utan att man gör wait, signal placeras satserna "DATAACESS" i en monitorprocedur. För att göra data globalt tillgängliga placeras monitorns vardeklaration i ett KERNELREC se kap 4.7.3. Se även monitor-exempel bil 1 s 36. Varje monitor har en procedur INITMONITOR, bil 1 s 36, som anropas i CONCURRENT och ger variablerna startvärde.

Monitorprocedurerna måste externkompileras om de skall användas av många olika processer.

4.4 Meddelande

Vid överföring av stora datamängder blir monitortekniken långsam. Det är då mer naturligt att överföra en pekare till ett record, ett meddelande, bil 1 s 36. Initieringen av globala brevlådor måste utföras i CONCURRENT. Pekare till brevlådorna överföres då automastiskt till konsekutiva minnesceller efter KERNELPTR. De sända meddelanderna placeras i en FIFO-kö. Om det finns något meddelande vid anrop av RECEIVEMESSAGE får man en pekare till det, annars får man vänta tills det kommer ett. SENDMESSAGE överför en pekare till brevlådans FIFO-kö.

Då ett meddelande skapas första gången med new måste dess NEXTMESS-referens sättas till "nil".

Från början kan t ex 10 meddelanden placeras i en global brevlåda kallad POOL. Alla processer kan sen ta ett tomt meddelande ur POOL:en fylla i lämpliga data och skicka det till en annan brevlåda. Mottagare läser meddelandet och lägger tillbaka det i POOL:en, därigenom går samma meddelanden runt hela tiden. Se fig 17.

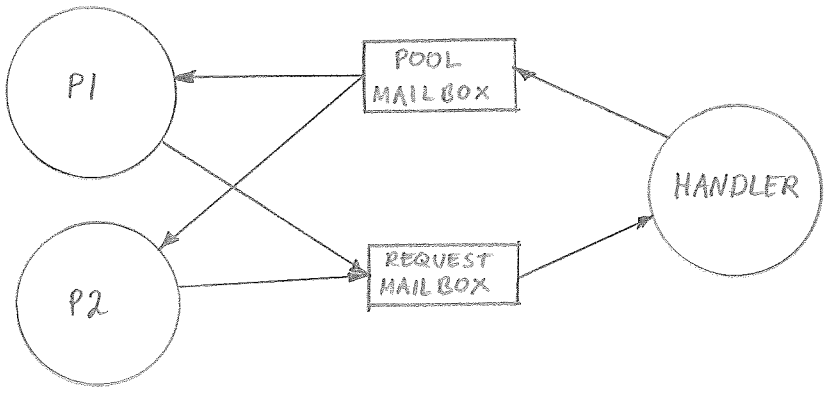


Fig 17 Meddelande överföring via POOL.

4.5 PASSIVATE och ACTIVATE

Dessa primitiv har införts för att eventuellt kunna användas i samband med simulering kap 6.1. De kan även användas för att starta och stoppa olika processer i READYQUE. Problemet vid PASSIVATE är att processen måste befinna sig i READYQUE. Jag har provat de ovannämnda primitiven dels för uppstart av processer på kommando bil 1 s 29, och dels som ett slags timeout bil 1 s 29. Dock rekommendera jag i första hand användningen av events vid synkronisering av typ uppstart.

Timeout realiseras lämpligen enligt ref 1. Tanken är att dessa primitiv skall arbeta mot HOLDQUE. Se kap 6.1

4.6 Initiering programmet CONCURRENT (bil 1 s 25)

EXEMPEL:

```
Program  CONCURRENT (INPUT, OUTPUT);
const  FLOPPYHANDLER = 8; P1 = 12; P2 = 16;
type
- INC PAGNT510
      (*Eventuella övriga type deklarationer*)
var   KERNEL: kernelptr; (*Måste finnas i alla
                           processer*)
      MAILBOX_1: mailbox; (*Måste finnas i alla
                           processer*)
      MAILBOX_2: mailbox; (*som använder glo-
                           bala brev-
      osv                  lådor*)
Procedur ..........; PASCAL; (*Externa pro-
                                cedurer*)
Begin
     new (KERNEL);
     initkernel;
     initprinter;
     createprocess (FLOPPYHANDLER, 1000);
     initmailbox (mailbox 1);
     initmailbox (mailbox 2);
     osv
     initmonitor_1;
     initmonitor_2;
     osv
     createprocess (P1, 1000);
     createprocess (P2, 1000);
     osv
     set priority (-1);(*Huvudprogrammet pas-
                        siviseras *)
end.
```

Obs. Huvudprogrammet är det enda program som kompileras i A-mod.

- INC PAGNT510 lägger in filen PAGNT510 som innehåller de globala deklarationerna.

4.7 Globala dataarean

Den globala dataarean finns på module PAGNT510, bil 1 s 34. I denna modul ingår 3 andra via INC-kommando. Dels PAGNT800 för monitordeklarationer, dels PAGNT511 för deklaration av meddelanden och brevlådor och dels PAGNT512 för ett KERNELREC.

Modulerna 800 och 511 kan båda strykas om de inte används. Modul 512 måste alltid finnas med. Uppläggningen av dataarean bör göras med stor noggrannhet.

4.7.1 Modulen PAGNT800 (bil 1 s 37)

Här typ deklarerar du dina monitorer.

EX:

```
mon_1 = record
          sem_1: semafor;
          data_1: datatyp_1;
        end;

mon_2 = record
          sem_2: semafor;
          data_2: datatyp_2;
        end;

osv
```

4.7.2 Modulen PGNT511 (bil 1 s 36)

Här typ deklarerar du dina meddelande.

EX:

```
messref  =^message;

message = record
            nextmess: messref; ( Alltid
                                 fält 1 )
            data : datatyp;
          end;
```

4.7.3 Modulen PAGNT512 (bil 1 s 37)

På denna modulen finns ett KERNELREC. Allt som skrives i detta record blir var-deklarerat och bildar den globala-dataarean. Genom att ta med, eller inte ta med denna modul bestämmer man om en process skall ha tillgång till globala dataarean. I regel tar man bara med modul 512 i sina entry procedurer.

```
EX:     KERNELREC = record
           RUNNING, READYQUE, TIMEQUE: Processptr;
           FREETOP: Integer4;
           PRINTER: resource;
           ( Slut på den obligatoriska delen )
           DATABAS_1: Mon_1;
           DATABAS_2: Mon_2;
           osv
        end;


        resource = record
          GUARD : semafor

        end;
```

4.8 Programstruktur

Uppläggning av dataarean sker i följande steg.

1. Skriv dina monitorer på en ny modul = monitormodulen. Kap 4.7.1.
2. Skriv dina meddelanden på ytterligare en ny modul = meddelandemodul. Kap 4.7.2.
3. Kopiera över PAGNT512 på en tredje ny modul = kernelmodul. Kompletteradenna enligt kap 4.7.3.
4. Se till att alla brevlådor är var-deklarerade i konsekutiv följd efter KERNELPTR, i de processer som använder dem, samt i CONCURRENT.
5. Kopiera PAGNT510 till en fjärde ny modul och ändra innehållet i den enligt nedan = global-datamodul.
   a) Byt PAGNT800 mot namnet på din monitormodul.
   b) Byt PAGNT511 mot namnet på din meddelandemodul.
   c) Byt PAGNT512 mot namnet på din kernelmodul.

Uppläggningen av dina processer enligt följande steg.

1. Ta med en lämplig mängd av dataarean t ex monitorer och meddelande modulerna. Ta med var-deklarationerna enligt kap 4.6.
2. Kompilera i B-mod.
3. Kopiera länkfilen PAGNT400, bil 7, och komplettera den med dina processer.
4. Utifrån ORG-kommandona beräknas startadresserna och placeras i PROCESSTAB bil 1 s 7.
5. Procedur deklarationerna för primitiv som skall anropas "tages" ur filen PAGNT500.
6. Starta exekveringen på adressen 1000 H.

I entryprocedurerna har man lämpligen med hela den globala datamodulen. Man får med denna data uppläggning ett bättre skydd av monitorerna.

Efter var och type deklarationerna enligt ovan kan egna deklarationer införas.

Vill man, för att spara utrymme, ej ha med filerna INPUT och OUTPUT deklareras lämpligen 4 icke använda "dummy"-minnesceller före KERNELPTR. KERNELPTR måste i alla program finnas på adressen -16(A5). Ett exempel på detta bil 1 s 24.

4.9 Länkning (bil 7)

Eftersom kärnan nu är fullt uttestad kan man ta bort alla org kommandon utom det första ORG Å1000. Man måste då ändra startadresserna för IDLE och CLOCK, de nya startadresserna finner man i utskriften från länkaren.

Vill man inte använda floppydischandlern kan modulerna 702, 802, 716, 808 tagas bort.

4.10 Felsökning

Ett bra sätt att felsöka på har visats sig vara genom att lägga in 4EFEH i maskinkoden där man vill stoppa. Genom att byta tillbaka kan man fortsätta exekveringen via singlestep t ex.

6917 0907-AA (999) 81-04 100 000 Ansvarig: POR

5 FLOPPYDISCHANDLERN

5.1 Floppydiscen

Ett utdrag ur en beskrivning av floppydiscen finns i bil 4.
Floppydiscen är inställd för autobaud.

5.2 Kommunikationskortet DSCA 114

Denna enhet har varit examensarbetets stora stötesten, beskrivning bil 2. Det kortet jag fick tag i var en med tvesamhet fungerande prototyp. Vissa kanaler fungerade bara efter upprepad initiering. Detta problem löstes med en typ av timeout bil 1 s 29. Den kanal jag slutligen började använda fungerade bra, utom vid mottagning av en sektor på 128 byte. Eftersom mottagningsbufferten är på 96 byte måste man flytta RAM (bil 4) i farten, om man vill ha avbrott efter 128 byte. Detta visade sig omöjligt, troligen spårar kortets processor ur. Denna programdel fick jag att fungera delvis genom att införa en väntetid istället för avbrott. Kortet görs för närvarande om. Programmeringen av kortets UART sköts av assembler proceduren INITUART bil 1 s 14.

5.3 Anslutningsenheten (DSTC 120)

Denna enhet används för att koppla floppydiscen via ett V-24 gränssnitt till DSCA 114. Mot floppydiscen har man en DB 25-kontakt.

Följande koppling visade sig lämplig.

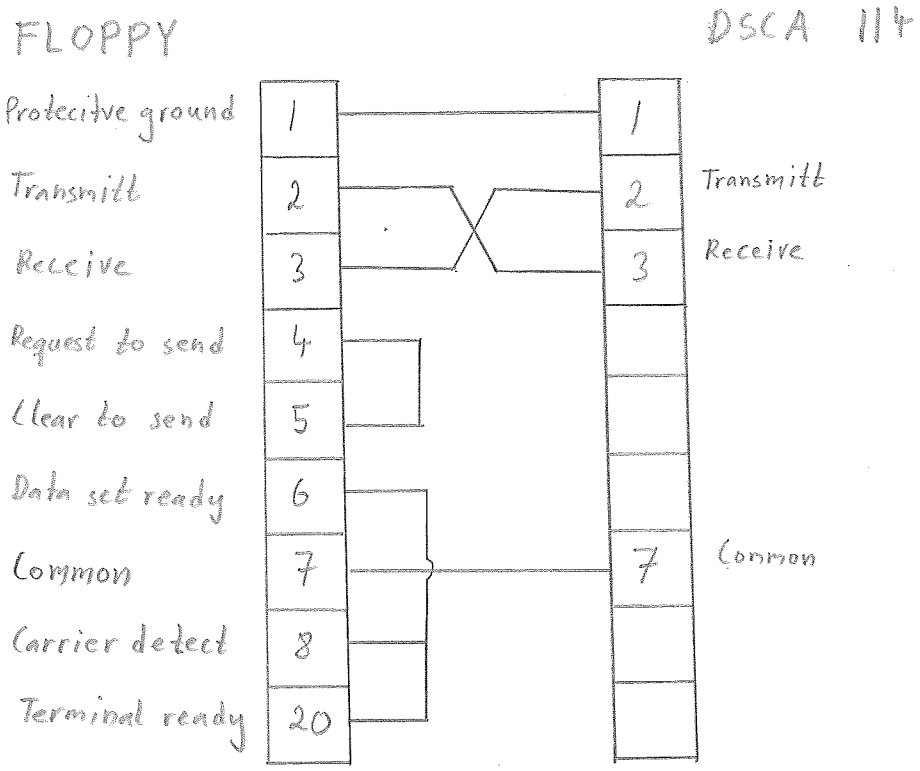


Fig 18 Floppydisc anslutning.

5.4 Användning

Floppydischandlerns programkod finns i bil 5.

Ett exempel på användning bil 1 s 31.

Handlern använder en assembler procedur MOVEBYTE, bil 1 s 14, vid kopiering av data. Vid omvandling Integer/ASCII används proceduren CONVERT bil 1 s 14.

Man skickar ett variant record till en brevlåda DISCREQUEST, där varianten anger kommando. När uppgiften är utförd kan man hämta en kvittens i brevlådan ACKNOWLEDGE. Innan floppydiscen har använts måste man skicka ett initieringsmeddelande, se programkoden.

6 UTVECKLINGSMÖJLIGHETER

6.1 Simulering

För simulering har processen SIMCLOCK bil 1 s 25 och primitivet HOLD utvecklats. SIMCLOCK låter processerna vänta en simulerad tid. Vid HOLD väntar processen i en HOLDQUE, man kan således simulera i reel och simulerad tid samtidigt. Även ACTIVATE och PASSIVATE skulle kunna operera på HOLDQUE, jfr SIMULA.

Dessa primitiv skulle kunna användas för simulering av digitala kretsar tillsammans med verkliga komponenter och MC 68000 CPU:n.

6.2 Kodoptimering

Med realtivt enkla medel skulle man kunna minska kärnans kodmängd och exekveringstid med ca 40 %, genom att optimera koden ur adressladdnings synpunkt.

Som det nu är laddas ofta adresser in i register trots att de redan finns där.

6.3 Debuggingrutin

Vid adressfel, stackfel mm hämtas vissa trapvektorer. För närvarande lägger MACSBUG in samma adress på dessa vektorer. På denna adress finns en rutin som skriver ut "TRAP ERROR". Därigenom förstörs information om var i programkoden felet inträffade och register innehåll. Det skulle i stället skrivas en liten rutin som skriver ut PC och de övriga registerna. En sådan rutin skulle spara mycket tid.

LITTERATURFÖRTECKNING

1. Datorer i reglersystem II: Hilding Elmqvist, Sven Erik Mattson, Gustaf Olsson. Kurslitteratu LTH, Lund 1981.

2. Structered concurrent programming: R C Holt, G S Graham. 1978.

3. The architecture of concurrent programming: Per Birch Jensen. 1977.

4. The logic design of multiple microprocessor systems: B A Bowen, R J A Buhr. 1980.

5. MC 68000 16-bit microprocessro. Users Manual: Motorola. 1980.

BILAGA 1

```
*LOAD ÄNDRAR TRAPVEKTORERNA SOM ANVÄNDS VID SYSTEM ANROP
*TILL KÄRNANS I/O-RUTINER
                XREF        PASSTART
LOAD:           NOP
                MOVE.L      ÄÅ38EC,ÅBC
                MOVE.W      ÄÅ2700,SR       *SUPERVISERY MODE
                JMP.L       PASSTART
*
```

```
*       PROCEDURE DI; PASCAL;
*       DISABLE INTERRUPT.
                XDEF        DI
DI:             ORI.W       ÄÅ700,SR
                RTS
*-------------------------------------------------------------------------
```

```
*       PROCEDURE EI; PASCAL;
*       ENABLE INTERRUPT.
                XDEF        EI
EI:             ANDI.W      ÄÅF2FF,SR
                RTS
*-------------------------------------------------------------------------
```

```
*  GLOBAL PEKARE TILL EXEKVERANDE PROCESS.
                XDEF        CURRENT
CURRENT:        DS.L        1
*-------------------------------------------------------------------------
```

```
*       PROCEDURE INITNUCLEUS(SREQ:INTEGER4;
*                             VAR FREE, MAIN:INTEGER4); PASCAL;
*
*       INPUT:      SREQ IS SIZE OF STACK
*
*       OUTPUT:     FREE IS ADDRESS OF UNALLOCATED STACK
*                   MAIN IS PROCESSPOINTER WHICH POINTS TO
*                   TO -4(HEAPBAS).
*
*
*       DESCRIPTION OF PROGRAM
*
*       1.    A7-SREQ -> FREE
*       2.    MAIN IS MADE TO POINT TO -4(HEAPBAS)
*       3.    CURRENT IS MADE TO POINT TO -4(HEAPBAS) THAT IS
*             TO SAY THAT THIS PROGRAM IS MADE CURRENT PROCESS.
*
*       WE HAVE AFTER ENTRY ON STACK
*        SREQ    VALUE          12(A7)
*        FREE    ADDRESS         8(A7)
*        MAIN    ADDRESS         4(A7)
*        PC      RETURN ADDR     0(A7)
*
            XDEF        INITNUCLEUS
            XREF        CURRENT
            XREF        HEAPBAS
INITNUCLEUS: MOVEA.L    A7,A0               GET STACKPOINTER
            SUB.L       12(A7),A0           A7-SREQ -> A0
*
            MOVE.L      8(A7),A1            GET ADDR OF FREE
            MOVE.L      A0,(A1)             STORE VALUE IN FREE
*
            LEA.L       HEAPBAS,A0
            SUBQ.L      Ä4,A0               GET ADDRESS BELOW HEAP
            MOVEA.L     4(A7),A1            GET ADDRESS OF PAR. MAIN
            MOVE.L      A0,(A1)             MAIN POINT TO -4(HEAPBAS
*
            LEA.L       CURRENT,A1          GET ADDRESS OF CURRENT
            MOVE.L      A0,(A1)             CURRENT POINT TO -4(HEAP
*
            MOVE.L      (A7),12(A7)         PICK UP RETURN ADDRESS
            ADD.L       Ä12,A7              DEALLOCATE PARAMETER SPA(
            RTS
            END
```

```
*       PROCEDURE NEWPROCESS(STARTADR,HEAPBAS,STACKBAS:INTEGER4;
*                            VAR CHILD:INTEGER4); PASCAL;
*
*       THIS BUILDS A NEWPROCESS-SAVE-BLOCK ON STACK.
*       CHILD IS POINTS TO  PROCESSVARIABLE IMMEDIATE BELOW HEAP
*       AFTERWARDS. PROCESSVARIABLE POINTS TO STARTADRESS OF PROCESS.
*
*
*
            XDEF        NEWPROCESS
            XREF        CURRENT
            XREF        PUTADRESS
NEWPROCESS: MOVE.W      SR,-(A7)
*
            ORI.W       ÄÅ700,SR            DISABLE INTERRUPT
*
*           BUILD NEWPROCESS-SAVE-BLOCK
*
*           PRESENT STACK ALLOCATIONS.
*           STARTADR        18(A7)
*           HEAPBAS         14(A7)
*           STACKBAS        10(A7)
*           CHILD            6(A7)
*           OLD PC           2(A7)
*           SR               0(A7)
*
            MOVEA.L     10(A7),A0           GET STACKBAS
            MOVE.L      18(A7),-(A0)        STARTADR -> STACK
            MOVE.L      10(A7),-(A0)        STACKBAS -> STACK
            LEA         RESCHILD,A1         ADDRESS OF RESCHILD -> A1
            MOVE.L      A1,-(A0)            RESCHILD -> STACK
            MOVEA.L     14(A7),A1           HEAPBAS  -> A1
            MOVE.L      A1,16(A0)           PUSH HEAPBAS ONTO STACK
*                                           ABOVE STARTADR
            SUBQ.L      Ä4,A1              -4(HEAPBAS) CONTAINS PROCESS
*                                           VARIABLE
            MOVEA.L     6(A7),A2            A2 IS ADDRESS OF CHILD
            MOVE.L      A1,(A2)             CONNECT CHILD TO PROCESSVAR
            MOVE.L      A0,(A1)             CONNECT PROCESSVAR TO
*                                           NEWPROCESS-SAVE-BLOCK
            MOVE.W      (A7)+,SR            RESTORE SR
            MOVE.L      (A7),16(A7)         GOTO RESCHILD
            ADD.L       Ä16,A7              DEALLOCATE PARAMETERS
            RTS
*
RESCHILD:   MOVEA.L    (A7)+,A4            STACKBAS -> A4
            MOVEA.L     A4,A5                       A5
            MOVEA.L     A5,A6                       A6
            SUB.L       Ä60,A7             ALLOCATE SPACE ON STACK
*                                          INIT OF KERNELPTR AND
            JSR.L       PUTADRESS          MAILBOX POINTERS.
            ADD.L       Ä60,A7
            ANDI.W      ÄÅF2FF,SR          ENABLE INTERRUPT
            RTS
            END
```

```
*       PROCEDURE RESUME(PROCESSPTR:INTEGER4); PASCAL;
*
*       THIS PROCEDURE CONTINUES THE EXECUTION OF A PROCESS THAT
*       HAS BEEN HALTED.
*
*       THE FOLLOWING ACTIONS ARE PERFORMED BY THIS ROUTINE.
*
*       1.  BUILD A RESUME-SAVE-BLOCK ON STACK FOR LATER RESUME
*           OF CURRENT PROCESS.
*
*       2.  SELECT AN EXISTANT RESUME-SAVE-BLOCK ON STACK BY THE
*           PARAMETER PROCESSPTR.
*
*       3.  START EXECUTION OF THE RESUME BLOCK TO RESTORE REGS.
*
*       4.  CONTINUE EXECUTION OF RESUMED PROCESS.
*
*       WHEN THIS PROCEDURE IS ENTERED WE HAVE TYPICALLY 4 RESUME-
*       SAVE-BLOCKS ON STACK AND WE BUILD NO 5.
*
*       A RESUME-SAVE-BLOCK LOOKS LIKE:
*
*       ADDR TO PROCESS             26(A7)
*       OLD PC  (RESUME)            22(A7)
*       OLD SR  (SAVED)             20(A7)
*       A3      (SAVED)             16(A7)
*       A4      (SAVED)             12(A7)
*       A5      (SAVED)              8(A7)
*       RES                          0(A7)
*
*
*       RES IS ADDRES OF RESUME-START-BLOCK WHICH IS A CODE PIECE
*       THAT CONTINUES EXECUTION OF THE PROCESS AT 26(A7).
*
*
*       WE HAVE THE FOLLOWING LINKS IN A RESUMED PROCESS:
*
*        ------------------
*       I PROCESS ADDR   I   26(A7)
*        I                I
*        I                I                PROCESSPTR-------------I
*        I                I                                       I
*        I                I                ------------------     I
*        I                I      HEAPBAS   I                I     I
*     I->I RES            I                I PROCESSVAR  -------I <I
*     I  ----------------- -                ------------------    I
*     I                                                          I
*     I                                                          I
*     I----------------------------------------------------------I
*
                XDEF        RESUME
                XREF        CURRENT
RESUME:         MOVE.W      SR,-(A7)          SAVE SR
                ORI.W       ÄÅ700,SR          DISABLE INTERRUPT
*
*               BUILD A NEW RESUME-SAVE-BLOCK ON STACK
                MOVEM.L     A3-A6,-(A7)       SAVE ADDRESS REGS
                LEA         RES,A0            GET ADDRESS OF START BLOCK
                MOVE.L      A0,-(A7)          PUSH IT ONTO STACK
                LEA.L       CURRENT,A0        GET ADDRESS OF CURRENT
                MOVEA.L     (A0),A2          -4(MINIHEAPBAS) -> A2
                MOVE.L      A7,(A2)           STORE START IN ABOVE ADDR
*
*               PREPARE PICK-UP OF OLD RESUME-SAVE-BLOCK
                MOVE.L      26(A7),(A0)       UPDATE CURRENT
                MOVEA.L     26(A7),A0         A0:= NEW PROCESSPTR
                MOVEA.L     (A0),A7           SWITCH TO NEW A7
                MOVEA.L     (A7)+,A0          RESUME THE PROCESS
                JMP         (A0)
*
RES:            MOVEM.L     (A7)+,A3-A6       RESTORE REGS
                MOVE.W      (A7)+,SR          RESTORE SR
                MOVE.L      (A7),4(A7)        RESTORE ENTRY POINT
                ADDQ.L      ÄÅ4,A7            TO CONTINUED PROCESS
                RTS
                END
```

```
*      PTR          6(A7)
*      VECADR       4(A7)
*      PC            (A7)
*PROCEDURE IORESUME(PTR:INTEGER4;VECADR:INTEGER2);
*RESUMES PTR OCH LÅTER DEN KÖRANDE VÄNTA PÅ VEKTOR AVBROTT.
                XDEF          IORESUME
                XREF          INTRESUME
                XREF          CURRENT
JSR             EQU           Å4EB9
IORESUME:       MOVE.W        SR,-(A7)          *SAVE CONTEXT FORDRIVER.
                ORI.W         ÄÅ700,SR
                MOVEM.L       A3-A6,-(A7)
                LEA.L         CURRENT,A0
                MOVE.L        (A0),-(A7)
                LEA           RESDRIVER,A1
                MOVE.L        A1,-(A7)
                MOVE.L        (A0),A2
                MOVE.L        A7,(A2)           *(CURRENT):=SP
                LEA.L         INTRESUME,A1      *BYGG JSR BLOCK PÅ STACKEN
                MOVE.L        A1,-(A7)
                MOVE.W        ÄJSR,-(A7)
                MOVE.L        Ä0,A1
                MOVE          36(A7),A1         *SET INTERRUPT VEKTOR
                MOVE.L        A7,(A1)
                MOVE.L        38(A7),(A0)       *RESUME PTR:PROCESSEN
                MOVEA.L       38(A7),A0
                MOVEA.L       (A0),A7
                RTS

RESDRIVER:      LEA.L         CURRENT,A0        *RESTORE CONTEXT FOR DRIVER
                MOVE.L        (A7)+,(A0)
                MOVEM.L       (A7)+,A3-A6
                MOVE.W        (A7)+,SR
                MOVE.L        (A7),6(A7)
```

```
*         PROCEDURE INTRESUM(IOSUSP:INTEGER4); PASCAL;
*
*         THIS PROCEDURE IS CALLED FROM STACK BY
*         'JSR.L  INTRESUME'  WHEN AN INTERRUPT OCCURS.
*
*       WE GET TO THIS JSR INSTRUCTION FROM INTERRUPT VECTOR
*       SPECIFIED IN IORESUME(PTR:INTEGER4;VECADR:INTEGER2).
*
*       INTERRUPT PROCESS-STACK CONTAINS THE FOLLOWING:
*
*       IOSUSP                    ADDRESS OF INTERRUPT ROUTINE
*       INTRESUME                 ADDRESS OF THIS ROUTINE
*       JSR INSTRUCTION
*
*
*
*         THIS PROCEDURE SAVES THE CONTEXT OF RUNNING PROCESS ON ITS
*         STACK AND RETURNS TO CORRESPONDING INTERRUPT DRIVER PROCESS.
*
*
              XDEF        INTRESUME
              XREF        CURRENT
*
INTRESUME:    MOVEM.L     D0-D7/A0-A6,-(A7)     SAVE ALL REGS
*
              LEA         RESINT,A0       PREPARE FOR CONT. AFTER INTERRUPT
              MOVE.L      A0,-(A7)        STORE RESUME ADDRESS
              LEA.L       CURRENT,A0
              MOVE.L      (A0),A0         ADDR(PROCESSVAR) ->A0
              MOVE.L      A7,(A0)
              MOVE.L      64(A7),A7       RTS TO INTERRUPT DRIVER !
              RTS

RESINT:       MOVEM.L     (A7)+,D0-D7/A0-A6  *RESTORE CONTEXT
              ADDQ.L      #4,A7
              RTE
              END
```

```
*         PROCEDURE FINDPC(VAR X:INTEGER4); PASCAL;
*
*         THE PROCESS NO X=0, 4, 8, ...   IS CONVERTED TO
*         ABSOLUTE 4 BYTES ADDRESS.
*
*         THE ADDRESSES ARE STORED INTERNALLY IN THIS ROUTINE
*         WHICH MUST BE UNIQUE TO EACH SYSTEM.
*
*
*         STACK ALLOCATION AFTER ENTRY.
*         ADDR(X)          4(A7)
*         OLD PC            (A7)
*
*
*
              XDEF          FINDPC
FINDPC:       MOVEA.L       4(A7),A1            ADDR(X)  -> A1
              MOVE.L        (A1),D0             CONT(X)  -> D0
              LEA           PROCESSTAB,A0       ADDR(PROCESSTAB) ->A0
              MOVE.L        0(A0,D0),(A1)       FETCH ADDRESS
              MOVE.L        (A7),4(A7)      MOVE RETURN ADDRESS
              ADDQ.L        Ä4,A7           DEALLOCATE PARAMETERLIST
              RTS
PROCESSTAB    DC.L          Å2700         *IDLE  NR=0
              DC.L          Å2800         *CLOCK NR=4
              DC.L          Å47CA         *FLOOPY NR=8
*CONTINUE HERE WITH YOUR PROCESS ADDRESSES
              DC.L          Å5000
              DC.L          Å3A00
              DC.L          Å3C5E
*-------------------------------------------------------------------------
```

DETTA SÄTT ATT ANGE STARTADRESSER ÄR ENDAST EN PROVISORISK LÖSNING. DET HELA KAN SKÖTAS AUTOMATISKT OM MAN SOM I REF 1 HAR TILLGÅNG TILL EN KOMPILATOR SOM KAN HA EN PROCEDUR SOM ARGUMENT I ETT PROCEDUR ANROP.
EN SÅDAN KOMPILATOR KOMMER ATT FINNAS PÅ ASEA TILL VÅREN 1982.

```
*INITRTC STARTAR REALTIDSKLOCKAN
MSB             EQU             ÅA          (*50MS. INTERVALLTID*)
LSB             EQU             Å0
IOBAS           EQU             ÅFF0000
TIMER           EQU             Å06
CONTROL         EQU             Å29
ICLX            EQU             Å01
                XDEF            INITRTC
INITRTC:        MOVE.L          ÄIOBAS,A0
                MOVE.B          ÄÅ35,TIMER(A0)          *INTERVALL
                MOVE.B          ÄÅ60,TIMER(A0)          *STALL
                MOVE.B          ÄÅB6,TIMER(A0)          *TIMEOUT
                MOVE.B          ÄLSB,ICLX(A0)
                MOVE.B          ÄMSB,ICLX(A0)   *T:=(LSB*20US)+(MSB*50MS)
                BSET            Ä1,CONTROL(A0)
                RTS
```

```
*PROCEDURE KVITTERA;
                XDEF            KVITTERA
KVITTENS        EQU             ÅFF003E
KVITTERA:       MOVEA.L         ÄKVITTENS,A1
                BCLR            Ä4,(A1)
                RTS
                END
```

```
*PROCEDURE SENDADRESS(NEWBOX:MAILBOX);
*FLYTTAR MAILBOX:S ADRESS TILL VEKTORN ADRESS.
                XDEF            SENDADRESS
SENDADRESS:     LEA             ANTAL,A0
                MOVE.L          (A0),D0
                LEA             ADRESS,A1
                MOVE.L          4(A7),0(A1,D0)
                ADDQ.L          Ä4,(A0)
                MOVE.L          (A7),4(A7)
                ADDQ.L          Ä4,A7
                RTS
```

```
*PROCEDURE PUTADRESS;
*FLYTTAR GLOBALPEKARE OCH BOX ADRESSER TILL PROCESSERNAS DATAAREA.
                XDEF            PUTADRESS
                XREF            HEAPBAS
PUTADRESS:      MOVEA.L         A5,A0
                SUB.L           Ä16,A0
                LEA.L           HEAPBAS,A1
                MOVE.L          A1,-(A0)
                CLR.L           D0
                LEA             ANTAL,A1
                LEA             ADRESS,A2
NEXT:           MOVE.L          0(A2,D0),-(A0)
                CMP.B           (A1),D0
                BEQ             OUT
                ADDQ.L          Ä4,D0
                BRA             NEXT
OUT:            RTS
ANTAL:          DC.L            Ä0
ADRESS:         DS.L            44
                END
```

```
*I/O RUTINER FÖR REALTIDSKÄRNAN
          XDEF        OUTDATA
          XREF        WAITIO
          XREF        EI
          XREF        DISABLE
UARTREG   EQU         ÅFF0021
          NOP
BUFFER    DS.B        Å128          WORKING AREA
          DS.B        20            STACK AREA
NYA5      DS.L        1
STOREA5   DS.L        1
INTIDREG  EQU         ÅFF0025
OUTDATA:  MOVEA.L     ÄINTIDREG,A0
          MOVE.B      (A0),D0
          BSR         OUTPUT              SEND A LINE
CRLF:     LEA         BUFFER,A5           LADDA BUFFERT ADR.
          MOVE.L      A5,A6
          MOVE        ÄÅ0D0A,(A6)+        PUT CR LF AT END OF BUFFER
*                                         SUBROUTINE OUTPUT
OUTPUT:   MOVEM.L     D0-D2/A0-A1,-(A7)   GET SOME WORKING ROOM
OUTP1     MOVEA.L     ÄUARTREG,A0         HÄMTA CONTROLREG. FÖR UART
OUTP2:    CMP.L       A6,A5               SEE IF AT OR BEYOUND EOL
          BMI.S       OUTP3
          MOVEM.L     (A7)+,D0-D2/A0-A1   RESTORE REGISTERS
          RTS                             END OF ROUTIN
OUTP3:    MOVE.B      (A5)+,D0            GRAB BYTE TO OUTPUT
          BSR.S       OUTCH1              GO PRINT IT
          BRA.S       OUTP2               GO TO ANOTHER
*                                         SUBROUTINE OUTCH1
OUTCH1:   BSR         OUTCH               GO PRINT D0
          TST.B       D0                  SEE IF NULL
          BEQ.S       OUTCHRTS            JUST END IF NULL
          CLR.L       D2                  CLEAR UPPER BYTES OF NULL LOOPCOUNT
          MOVE.L      ÄÅ051E,A1           FORM ADDRESS OF PADS
          MOVE.B      (A1),D2             DEFAULT NULL PADS
          CMP.B       ÄÅ0D,D0             SEE IF CR
          BNE.S       OUTCH2
          MOVE.L      ÄÅ0520,A1           FORM ADDRESS OF CR PADS
          MOVE.B      (A1),D2             NULLS AFTER CR
OUTCH2:   TST.L       D2                  SEE IF ANY PADDS TO BE SENT
          BEQ.S       OUTCHRTS            0=NONE
          CLR.L       D0                  0=NULL CHAR TO BE SENT
OUTCH3:   BSR         OUTCH               SEND A NULL
          SUB.L       ÄÅ01,D2             LOOP AROUND
          BNE         OUTCH3
OUTCHRTS: RTS


*                                         SUBROUTINE OUTCH
*SÄNDER ETT TECKEN
ENABLE    EQU         ÅFF0023
STATUS    EQU         ÅFF002A
OUTCH:    MOVEA.L     ÄENABLE,A2
          MOVE.B      ÄÅ2,(A2)
          MOVE.B      D0,(A0)
          MOVE.L      A5,-(A7)
          LEA         STOREA5,A5
          MOVE.L      (A5),A5
          MOVEM.L     D0-D2/A0-A1,-(A7)
          MOVE        ÄÅ6C,-(A7)
          JSR.L       WAITIO
          JSR.L       DISABLE
          MOVEM.L     (A7)+,D0-D2/A0-A1
          MOVE.L      (A7)+,A5
          RTS
*
```

```
*KALLAS VIA TRAP15 VID INLÄSNING FRÅN PRINTER
INDATA    MOVEM.L     D0-D3/A0-A2,-(A7)   FREE UP SOME WORK REG
          MOVEA.L     ÄINTIDREG,A0
          MOVE.B      (A0),D0
          MOVEA.L     ÄUARTREG,A0  HÄMTA UART REG.
          MOVE.B      (A0),D0
READBUF:  BSR         INCHNE       GO GET SOME DATA
          TST.B       D0           CHECK FOR NULLS
          BEQ.S       READBUF
  RB1:    MOVE        D0,-(A7)     SAVE FOR A WHILE
          BSR         OUTCH1       ECHO WHAT IS IN D0
          MOVE        (A7)+,D0     RESTORE IT
          CMP.B       ÄÅ0A,D0      SEE IF LINE FEED
          BEQ.S       READBUF      DON'T PUT IT IN BUFFER
          CMP.B       ÄÅ18,D0      SEE IF CTL X=CANCEL LINE
          BNE.S       RB2
          MOVE.B      ÄÅ5C,D0      SENT A SLASH
          BSR         OUTCH1
          MOVE.B      ÄÅ0D,D0      SEND A CARRIAGE RETURN
          BSR         OUTCH1
          MOVE.B      ÄÅ0A,D0      SEND A LINE FEED
          BSR         OUTCH1
          MOVE.L      A5,A6        START BUFFER OVER AGAIN
          BRA.S       READBUF
  RB2:    CMP.B       ÄÅ08,D0      SEE IF CTL H=BACKSPACE
          BNE.S       RB4
          CMP.L       A6,A5        SEE IF START OF BUFFER
          BEQ.S       RB3
          MOVE.B      -1(A6),D0    WHAT WAS LAST CHARACTER ****
          BSR         OUTCH1
*GO INTO BACKSPACE MODE
RB3:      MOVE.B      ÄÅ5C,D0      DO SEND A SLASH
          BSR         OUTCH1
RB31:     MOVE.B      -(A6),D0     GO BACK IN THE BUFFER
          CMP.L       A6,A5        CHECK POSITION
          BLE.S       RB32
          CLR.L       D0           PAST THE START OF THE BUFFER
          MOVE.L      A5,A6        WE ARE AT EH BEGINNING
RB32:     BSR         OUTCH1       GO PRINT WHAT BACKED OVER
RB34:     BSR         INCHNE
          TST.B       D0           SEE IF NUL
          BEQ.S       RB34
          CMP.B       ÄÅ08,D0      CHECK FOR ANOTHER BS CHAR
          BEQ.S       RB31
          CMP.B       ÄÅ7F,D0      DEL KEY
          BEQ.S       RB31
          MOVE        D0,-(A7)     PUSH CHAR FOR A MOMENT
          MOVE.B      ÄÅ5C,D0
          BSR         OUTCH1       GO PRINT SLASH TO END MODE
          MOVE        (A7)+,D0     UNSAVE CHAR JUST INPUT
          BRA         RB1          GO DECODE REGULAR CHAR
RB4:      CMP.B       ÄÅ7F,D0      DEL=BACKSPACE
          BEQ.S       RB3
          CMP.B       ÄÅ04,D0      SEE IF CTLD (REPRINT)
          BNE.S       RB5
          MOVE        ÄÅ0D,D0
          BSR         OUTCH1
          MOVE        ÄÅ0A,D0      PRINT LF
          BSR         OUTCH1
          MOVE.L      A5,-(A7)     SAVE ON STACK FOR A MOMENT
          BSR         OUTPUT       GO PRINT BUFFER
          MOVE.L      (A7)+,A5     GET BACK FROM STACK
          BRA         READBUF
RB5:      CMP.B       ÄÅ0D,D0      SEE IF AT END OF LINE
          BNE.S       RB6
          MOVE.B      ÄÅ0A,D0      GIVE LF
          BSR         OUTCH1
          MOVEM.L     (A7)+,D0-D3/A0-A2  PULL FROM STACK
          RTS
RB6:      MOVE.B      D0,(A6)+     SAVE DATA INTO BUFFER
          BRA         READBUF
          BSR         INCHNE
          BSR         OUTCH1
          RTS
*
```

```
INCHNE:   MOVEA.L     ÄENABLE,A1
          MOVE.B      ÄÅ1,(A1)
          MOVE.L      A5,-(A7)
          LEA         STOREA5,A5
          MOVE.L      (A5),A5
          MOVEM.L     D0-D2/A0-A1,-(A7)
          MOVE        ÄÅ6C,-(A7)
          JSR.L       WAITIO
          JSR.L       DISABLE
          MOVEM.L     (A7)+,D0-D2/A0-A1
          MOVE.L      (A7)+,A5
          MOVE.B      (A0),D0       LÄS TECKNET
          AND.B       ÄÅ7F,D0       DROP PARITY BIT
          RTS
*
*HIT KOMMER MAN VIA TRAP 15 VID I/O
          XREF        REQPRINTER
          XREF        RELPRINTER
          XREF        ERROR
EKA       EQU         Å22D80
TRAP15:   ORI.W       ÄÅ700,SR            *DI;
          MOVE.L      A5,-(A7)
          LEA         NYA5,A5
          MOVE.L      62(A7),(A5)
          CMP.L       ÄEKA,6(A7)
          BNE.S       REQUEST
          MOVE.L      12(A7),(A5)
REQUEST:  MOVE.L      (A5),A5
          MOVEM.L     D0-D1/A0-A1,-(A7)
          JSR.L       REQPRINTER
          ORI.W       ÄÅ700,SR            *DI
          LEA         STOREA5,A0
          MOVE.L      A5,(A0)
          MOVEM.L     (A7)+,D0-D1/A0-A1
          MOVE.L      (A7)+,A5
          MOVE.L      Å02(A7),A0
          MOVE        (A0)+,D0
          MOVE.L      A0,Å2(A7)
          CMP         ÄÅ01,D0
          BNE.S       NEXTTEST
          BSR         INDATA
          BRA.S       TRAPUT
NEXTTEST: CMP         ÄÅ02,D0
          BNE.S       FEL
          BSR         OUTDATA
          BRA.S       TRAPUT
FEL:      JSR.L       ERROR
TRAPUT:   MOVEA.L     ÄENABLE,A0
          MOVE.B      ÄÅ0,(A0)
          MOVE.L      A5,-(A7)
          LEA         STOREA5,A5
          MOVE.L      (A5),A5
          MOVEM.L     D0-D1/A0-A1,-(A7)
          JSR.L       EI
          JSR.L       RELPRINTER
          MOVEM.L     (A7)+,D0-D1/A0-A1
          MOVE.L      (A7)+,A5
          RTE
          END
```

```
* KVITTERAR UARTEN
          XDEF        DISABLE
UARTREG   EQU         ÅFF0021
INTIDREG  EQU         ÅFF0025
LINESTATUS EQU        ÅFF002B
DISABLE:  MOVEA.L     ÄLINESTATUS,A0
          MOVE.B      (A0),D0         *KVITTERA PARITETS FEL
          MOVEA.L     ÄINTIDREG,A0
          MOVE.B      (A0),D0         *KVITTERA TRANSMITT INTERUPPT
          AND.B       ÄÅ2,D0
          BEQ         DATAREADY
          RTS
DATAREADY: MOVEA.L    ÄUARTREG,A0
          MOVE.B      (A0),D0         *KVITTERA RECIVER INTERUPPT
          RTS
          END
```

```
*PROCEDURE INITUART;
                XDEF            INITUART
UARTBAS         EQU             ÅFF09E0
INITUART:       MOVE.L          ÄUARTBAS,A0
                BSET            Ä7,7(A0)        *SET BAUDRATE ENABLE
                MOVE.B          Ä16,1(A0)       *DIVISOR=16
                MOVE.B          Ä0,3(A0)        *BAUDRATE=9600
                MOVE.B          ÄÅ3,7(A0)       *EVEN PARITY 2 STOPBITS
                MOVE.B          ÄÅ7,9(A0)       *EJ SPLIT SPEED
                MOVE.B          1(A0),D0        *KVITTERA
                MOVE.B          11(A0),D0       *KVITTERA
                RTS
```

```
*PROCEDURE MOVEBYTE(VAR FROM,DEST:CHAR;ANTAL:INTEGER2);
*    DEST   10(A7)
*    FROM    6(A7)
*    ANTAL   4(A7)
                XDEF            MOVEBYTE
MOVEBYTE:       MOVEA.L         6(A7),A0
                MOVEA.L         10(A7),A1
                MOVE.W          4(A7),D0
NEXT:           MOVE.B          (A1)+,(A0)+     *COPY
                SUBQ.W          Ä1,D0
                BNE             NEXT
                MOVE.L          (A7),10(A7)
                ADD.L           Ä10,A7
                RTS
                END
```

```
*PROCEDURE CONVERT(REST,K:INTEGER2;VAR S:PACKED ARRAY(.1..3.) OF CHAR);
*OMVANDLAR FRÅN INTEGER TILL CHAR
*          REST    10(A7)
*          K        8(A7)
*          (S)      4(A7)
*          PC        (A7)
                XDEF            CONVERT
CONVERT:        MOVE.B          11(A7),D0
                MOVEA.L         4(A7),A0
                MOVE.W          8(A7),D1
                MOVE.B          D0,-1(A0,D1)
                MOVE.L          (A7),8(A7)
                ADDQ.L          Ä8,A7
                RTS
                END
```

```
(*DEKLARATIONER FÖR KÄRNANS PRIMITIV*)
CONST   MAXPRIORITY=1000;

TYPE    PROCESSPTR=ÖPROCESSREC;
        SEMAFOR=ÖSEMAFORREC;
        EVENT=ÖEVENTREC;

        KERNELPTR=ÖKERNELREC;

        PROCESSREC=RECORD
           SUC,PRE:PROCESSPTR;
           PTR:INTEGER4;
           PRIORITY,TIME:INTEGER4;
        END;

        SEMAFORREC=RECORD
           COUNTER:INTEGER4;
           WAITING:PROCESSPTR;
        END;

        EVENTREC=RECORD
           REENTRY:SEMAFOR;
           DELAYED:PROCESSPTR;
        END;

        RESOURCE=RECORD
           GUARD:SEMAFOR;
        END;

        KERNELREC=RECORD
           RUNNING,READYQUE,TIMEQUE:PROCESSPTR;
           FREETOP:INTEGER4;
           PRINTER:RESOURCE;
        END;


VAR KERNEL:KERNELPTR;
```

```
PROCEDURE PUT(P,Q:PROCESSPTR);
BEGIN
   PÖ.SUC:=Q;
   PÖ.PRE:=QÖ.PRE;
   QÖ.PREÖ.SUC:=P;
   QÖ.PRE:=P;
END.
```

```
PROCEDURE PUTPRIORITY(P,Q:PROCESSPTR);
(* THIS LIST PROCESSING FACILITY INSERTS A RECORD INTO A
   DOUBLE LINKED LIST SORTED IN ORDER TO PRIORITY VARIABLE IN
   THE RECORD *)

(* THE LIST IS ARRANGED ACCORDING TO DESCENDING PRIORITY *)

 (*   57     43    19          FROM LEFT TO RIGHT          *)
 (* THE SEARCH IS MADE IN THE LIST UNTIL WE FIND A PRIORITY
    LESS THAN THE GIVEN PRIORITY                           *)
VAR P1:PROCESSPTR; PRI:INTEGER4;
BEGIN
   PRI:=PÖ.PRIORITY;
   P1:=QÖ.SUC;
   WHILE (P1<>Q) AND (PRI<=P1Ö.PRIORITY) DO P1:=P1Ö.SUC;
   PUT(P,P1);
END.
```

```
PROCEDURE REMOVE(P:PROCESSPTR);
BEGIN
   WITH PÖ DO
      BEGIN
         PREÖ.SUC:=SUC;
         SUCÖ.PRE:=PRE;
      END;
END.
```

```
PROCEDURE INITKERNEL(STACKREQ:INTEGER4);
CONST CLOCKAREA=1000; IDLEAREA=1000;
      IDLE=0; CLOCK=4;
BEGIN WITH KERNELÖ DO
 BEGIN
   DI;
   NEW(RUNNING);
   NEW(READYQUE);
   READYQUEÖ.SUC:=RUNNING;
   READYQUEÖ.PRE:=RUNNING;
   RUNNINGÖ.SUC:=READYQUE;
   RUNNINGÖ.PRE:=READYQUE;
   (*SKAPAR TOM TIDSKÖ*)
   NEW(TIMEQUE);
   TIMEQUEÖ.SUC:=TIMEQUE;
   TIMEQUEÖ.PRE:=TIMEQUE;
   RUNNINGÖ.PRIORITY:=15;
   INITNUCLEUS(STACKREQ,FREETOP,RUNNINGÖ.PTR);
   CREATEPROCESS(IDLE,IDLEAREA);
   CREATEPROCESS(CLOCK,CLOCKAREA);
   WAITTIME(10);                    (*VÄNTA PÅ PRINTERN:S CR*)
 END;
END.
```

```
PROCEDURE CREATEPROCESS(NR:INTEGER4;MEMREQ:INTEGER4);
CONST STATICLINKAREA=44; (*MAX 10 NÄSTLADE PROC ANROP*)
      MAXPRIORITY=100;
VAR CHILD:PROCESSPTR;
    STARTADR,HEAPBASE:INTEGER4;
BEGIN WITH KERNELÖ DO
 BEGIN
   DI;
   HEAPBASE:=FREETOP-MEMREQ-STATICLINKAREA;
   NEW(CHILD);
   CHILDÖ.PRIORITY:=10;
   PUTPRIORITY(CHILD,READYQUE);
   FINDPC(NR);
   FREETOP:=HEAPBASE-4;
   NEWPROCESS(NR,HEAPBASE,HEAPBASE+MEMREQ,CHILDÖ.PTR);
   SCHEDULE;
   IF RUNNINGÖ.PRIORITY < MAXPRIORITY THEN EI;
 END;
END.
```

```
PROCEDURE SETPRIORITY(PRIORITY:INTEGER4);
BEGIN WITH KERNELÖ DO
 BEGIN
    DI;
    IF PRIORITY < RUNNINGÖ.PRIORITY THEN
    BEGIN
       RUNNINGÖ.PRIORITY:=PRIORITY;
       REMOVE(RUNNING);
       PUTPRIORITY(RUNNING,READYQUE);
       SCHEDULE;
    END
    ELSE  RUNNINGÖ.PRIORITY:=PRIORITY;
    IF RUNNINGÖ.PRIORITY > MAXPRIORITY THEN EI;
 END;
END.
```

```
PROCEDURE ACTIVATE(NR:INTEGER4;NEWPRIORITY:INTEGER4);
VAR    P:PROCESSPTR;
BEGIN WITH KERNELÖ DO
 BEGIN
    DI;
    P:=RUNNINGÖ.SUCÖ.SUC;
    WHILE (P <> READYQUE) AND (-NR<=PÖ.PRIORITY) DO
    BEGIN
       IF PÖ.PRIORITY=-NR THEN
       BEGIN
          REMOVE(P);
          PÖ.PRIORITY:=NEWPRIORITY;
          PUTPRIORITY(P,READYQUE);
       END;
       P:=PÖ.SUC;
    END;
    SCHEDULE;
    IF RUNNINGÖ.PRIORITY < MAXPRIORITY THEN EI;
 END;
END.
```

```
PROCEDURE PASSIVATE(NAME:INTEGER4);
(*PASSIVATES NAME IF IN TIMEQUE ELSE NOTHING*)
VAR P:PROCESSPTR;
BEGIN WITH KERNELÖ DO
 BEGIN
    DI;
    P:=TIMEQUEÖ.SUC;
    WHILE (PÖ.PRIORITY <> NAME) AND (P <> TIMEQUE) DO P:=PÖ.SUC;
    IF PÖ.PRIORITY=NAME THEN BEGIN PÖ.PRIORITY:=-NAME;
                                   REMOVE(P);
                                   PUTPRIORITY(P,READYQUE);
                             END;
    SCHEDULE;
    IF RUNNINGÖ.PRIORITY < MAXPRIORITY THEN EI;
 END;
END.
```

```
PROCEDURE SCHEDULE;
BEGIN WITH KERNELÖ DO
 BEGIN
   IF READYQUEÖ.SUC <> RUNNING THEN
   BEGIN
      RUNNING:=READYQUEÖ.SUC;
       (* START THE EXECUTION OF THE SELECTED PROCESS *)
      RESUME(RUNNINGÖ.PTR);
   END;
 END;
END.
```

```
PROCEDURE INITSEM(VAR SEM:SEMAFOR;INITVAL:INTEGER4);
BEGIN
   NEW(SEM);
   WITH SEMÖ DO
   BEGIN
      COUNTER:=INITVAL;
      NEW(WAITING);        (* ALLOCATE A PROCESSRECORD *)
      WAITINGÖ.SUC:=WAITING;    (*TOM TIDSKÖ*)
      WAITINGÖ.PRE:=WAITING;
   END;
END.
```

```
PROCEDURE WAIT(SEM:SEMAFOR);
BEGIN WITH KERNELÖ DO
 BEGIN
   DI;
   WITH SEMÖ DO
   BEGIN
      IF COUNTER>0 THEN COUNTER:=COUNTER-1
      ELSE
      BEGIN
         REMOVE(RUNNING);          (* THE  PROCESS  HAS TO WAIT *)
         PUTPRIORITY(RUNNING,WAITING); (* PUT IT INTO QUEUE *)
         SCHEDULE;
      END;
   END;
   IF RUNNINGÖ.PRIORITY < MAXPRIORITY THEN EI;
 END;
END.
```

```
PROCEDURE SIGNAL(SEM:SEMAFOR);
VAR P:PROCESSPTR;
BEGIN WITH KERNELÖ DO
 BEGIN
   DI;                        (* DISABLE INTERRUPT *)
   WITH SEMÖ DO
   BEGIN
      IF WAITING <> WAITINGÖ.PRE THEN
      BEGIN                              (* AT LEAST ONE WAITING *)
         P:=WAITINGÖ.SUC;                (* SKIP THE DUMMY PRCESS REC  *)
         REMOVE(P);                      (* TAKE THE PROCESSREC AWAY *)
         PUTPRIORITY(P,READYQUE);     (* PUT IT INTO READYQUEUE *)
         SCHEDULE;                       (* SELECT A NEW PROCESS FOR EXEC
      END
      ELSE COUNTER:=COUNTER+1;           (* NOBODY IS WAITING *)
   END;
   IF RUNNINGÖ.PRIORITY < MAXPRIORITY THEN EI;
 END;
END.
```

```
PROCEDURE INITEVENT(VAR E:EVENT;SEM:SEMAFOR);
BEGIN
   NEW(E);
   WITH Eö DO
   BEGIN
      REENTRY:=SEM;
      NEW(DELAYED);
      DELAYEDö.SUC:=DELAYED;
      DELAYEDö.PRE:=DELAYED;
   END;
END.
```

```
PROCEDURE AWAIT(E:EVENT);
VAR P:PROCESSPTR;
BEGIN WITH KERNELö DO
 BEGIN
   DI;
   REMOVE(RUNNING);
   PUT(RUNNING, Eö.DELAYED);
   WITH Eö.REENTRYö DO
   BEGIN
      IF WAITING <> WAITINGö.SUC THEN
      BEGIN
         P:=WAITINGö.SUC;
         REMOVE(P);
         PUTPRIORITY(P,READYQUE);
      END
      ELSE  COUNTER:=COUNTER+1;
   END;
   SCHEDULE;
   IF RUNNINGö.PRIORITY < MAXPRIORITY THEN EI;
 END;
END.
```

```
PROCEDURE CAUSE(E:EVENT);
VAR P:PROCESSPTR;
BEGIN WITH KERNELö DO
 BEGIN
   DI;
   WITH Eö DO
   BEGIN
      WHILE DELAYED <> DELAYEDö.SUC DO
      BEGIN
         P:=DELAYEDö.SUC;
         REMOVE(P);
         PUTPRIORITY(P,REENTRYö.WAITING);
      END;
   END;
   IF RUNNINGö.PRIORITY < MAXPRIORITY THEN EI;
 END;
END.
```

```
PROCEDURE WAITIO(VECADR:INTEGER2);
VAR    DRIVER:PROCESSPTR;
BEGIN WITH KERNELÖ DO
 BEGIN
    DI;
    DRIVER:=RUNNING;
    REMOVE(RUNNING);
    RUNNING:=READYQUEÖ.SUC;
    IORESUME(RUNNINGÖ.PTR,VECADR);
    PUTPRIORITY(DRIVER,READYQUE);
    SCHEDULE;
 END;
END.
```

```
PROCEDURE WAITTIME(T:INTEGER4);
VAR P:PROCESSPTR;
BEGIN WITH KERNELÖ DO
 BEGIN
    DI;
    REMOVE(RUNNING);
    P:=TIMEQUEÖ.SUC;
    WHILE (T > PÖ.TIME) AND (P <> TIMEQUE) DO
    BEGIN
       T:=T-PÖ.TIME;
       P:=PÖ.SUC;
    END;
    RUNNINGÖ.TIME:=T;
    PUT(RUNNING,P);
    IF P <> TIMEQUE THEN PÖ.TIME:=PÖ.TIME-T;
    SCHEDULE;
    IF RUNNINGÖ.PRIORITY < MAXPRIORITY THEN EI;
 END;
END.
```

```
(*ÅL+,O+,A4*)
PROGRAM IDLE;
(*TOM PROCESS*)
VAR S1,S2,S3,S4:INTEGER4;          (*FÖR ATT FÅ RÄTT ADRESS BERÄKNING*)
    KERNEL:INTEGER4;
PROCEDURE SETPRIORITY(PRIORITY:INTEGER4);PASCAL;
BEGIN
   SETPRIORITY(0);
   WHILE TRUE DO BEGIN END;
END.



PROGRAM CLOCK;
CONST MAXPRIORITY=1000;
TYPE
       PROCESSPTR=ÖPROCESSREC;
       SEMAFOR=ÖSEMAFORREC;
       EVENT=ÖEVENTREC;
       KERNELPTR=ÖKERNELREC;

       PROCESSREC=RECORD
          SUC,PRE:PROCESSPTR;
          PTR:INTEGER4;
          PRIORITY,TIME:INTEGER4;
       END;

       SEMAFORREC=RECORD
          COUNTER:INTEGER4;
          WAITING:PROCESSPTR;
       END;

       EVENTREC=RECORD
          REENTRY:SEMAFOR;
          DELAYED:PROCESSPTR;
       END;

       KERNELREC=RECORD
          RUNNING,READYQUE,TIMEQUE:PROCESSPTR;
          FREETOP:INTEGER4;
       END;

VAR    S1,S2,S3,S4:INTEGER4;   (*FÖR ATT FÅ RÄTT ADRESS*)
       KERNEL:KERNELPTR;
       P:PROCESSPTR;
       CLOCKINT:INTEGER2;
PROCEDURE EI;PASCAL;
PROCEDURE DI;PASCAL;
PROCEDURE SETPRIORITY(PRIORITY:INTEGER4);PASCAL;
PROCEDURE IORESUME(PTR:INTEGER4;VECADR:INTEGER2);PASCAL;
PROCEDURE INITRTC;PASCAL;
PROCEDURE REMOVE(P:PROCESSPTR);PASCAL;
PROCEDURE PUTPRIORITY(P,Q:PROCESSPTR);PASCAL;
PROCEDURE KVITTERA;PASCAL;
BEGIN WITH KERNELÖ DO
 BEGIN
    SETPRIORITY(MAXPRIORITY);
    REMOVE(RUNNING);
    CLOCKINT:=16Ä78;
    INITRTC;
    WHILE TRUE DO
    BEGIN
       RUNNING:=READYQUEÖ.SUC;
       IORESUME(RUNNINGÖ.PTR,CLOCKINT);
       KVITTERA;                          (*KVITTERA AVBROTTET*)
       P:=TIMEQUEÖ.SUC;
       IF P <> TIMEQUE THEN PÖ.TIME:=PÖ.TIME-1;
       WHILE (PÖ.TIME=0) AND (P <> TIMEQUE) DO
       BEGIN
          REMOVE(P);
          PUTPRIORITY(P,READYQUE);
          P:=TIMEQUEÖ.SUC;
       END;
    END;
 END;
END.
```

```
(*PAGNT150*)
(*ÅL+,O+,A4*)
PROGRAM SIMCLOCK(OUTPUT);
-INC PAGNT516

VAR     S1,S2:INTEGER4;          (*FÖR ATT FÅ RÄTT ADRESS*)
        KERNEL:KERNELPTR;
        FLAG:BOOLEAN;
        P:PROCESSPTR;
PROCEDURE EI;PASCAL;
PROCEDURE DI;PASCAL;
PROCEDURE SETPRIORITY(PRIORITY:INTEGER4);PASCAL;
PROCEDURE REMOVE(P:PROCESSPTR);PASCAL;
PROCEDURE PUTPRIORITY(P,Q:PROCESSPTR);PASCAL;
PROCEDURE SCHEDULE;PASCAL;
BEGIN WITH KERNELÖ DO
 BEGIN
    SETPRIORITY(1);
    FLAG:=TRUE;
    SIMTIME:=0;
    WHILE TRUE DO
    BEGIN
       P:=HOLDQUEÖ.SUC;
       SIMTIME:=SIMTIME+PÖ.TIME;
       IF P <> HOLDQUE THEN PÖ.TIME:=0
       ELSE IF FLAG THEN BEGIN  WRITELN(' HOLDQUE EMPTY');
                                FLAG:=FALSE;
                                SETPRIORITY(-1);
                         END;
       DI;
       WHILE (PÖ.TIME=0) AND (P <> HOLDQUE) DO
       BEGIN
          REMOVE(P);
          PUTPRIORITY(P,READYQUE);
          P:=HOLDQUEÖ.SUC;
       END;
       REMOVE(RUNNING);
       PUTPRIORITY(RUNNING,READYQUE);
       SCHEDULE;
       IF RUNNINGÖ.PRIORITY <> MAXPRIORITY THEN EI;
    END;
 END;
END.


PROCEDURE HOLD(T:INTEGER4);
VAR P:PROCESSPTR;
BEGIN WITH KERNELÖ DO
 BEGIN
    DI;
    REMOVE(RUNNING);
    P:=HOLDQUEÖ.SUC;
    WHILE (T > PÖ.TIME) AND (P <> HOLDQUE) DO
    BEGIN
       T:=T-PÖ.TIME;
       P:=PÖ.SUC;
    END;
    RUNNINGÖ.TIME:=T;
    PUT(RUNNING,P);
    IF P <> HOLDQUE THEN PÖ.TIME:=PÖ.TIME-T;
    SCHEDULE;
    IF RUNNINGÖ.PRIORITY < MAXPRIORITY THEN EI;
 END;
END.
```

```
PROCEDURE INITMAILBOX(VAR BOX:MAILBOX);
BEGIN
   NEW(BOX);
   SENDADRESS(BOX);
   WITH BOXö DO
   BEGIN
      FIRSTMESS:=NIL;
      INITSEM(MUTEX,1);
      INITEVENT(SEND,MUTEX);
   END;
END.
```

```
PROCEDURE SENDMESSAGE(BOX:MAILBOX;VAR MESS:MESSREF);
VAR M:MESSREF;
BEGIN WITH BOXö DO
 BEGIN
    WAIT(MUTEX);
    IF FIRSTMESS=NIL THEN FIRSTMESS:=MESS
    ELSE
    BEGIN
       M:=FIRSTMESS;
       WHILE Mö.NEXTMESS <> NIL DO M:=Mö.NEXTMESS;
       Mö.NEXTMESS:=MESS;
    END;
    CAUSE(SEND);
    SIGNAL(MUTEX);
 END;
END.
```

```
PROCEDURE RECEIVEMESSAGE(BOX:MAILBOX;VAR MESS:MESSREF);
BEGIN WITH BOXö DO
 BEGIN
    WAIT(MUTEX);
    WHILE FIRSTMESS=NIL DO AWAIT(SEND);
    MESS:=FIRSTMESS;
    FIRSTMESS:=FIRSTMESSö.NEXTMESS;
    SIGNAL(MUTEX);
 END;
END.
```

```
PROCEDURE INITPRINTER;
BEGIN
   WITH KERNELÖ.PRINTER DO
   BEGIN
      INITSEM(GUARD,1);
   END;
END.
```

```
PROCEDURE RELPRINTER;
BEGIN  SIGNAL(KERNELÖ.PRINTER.GUARD); END.
```

```
PROCEDURE REQPRINTER;
BEGIN WAIT(KERNELÖ.PRINTER.GUARD); END.
```

Dessa subprogram är extern kompilerade procedurer.

Detta är ett exempel på ett initieringsprogram.

```
(*ÅL+,O+,A4*)
PROGRAM CONCURRENT(INPUT,OUTPUT);
(*INITIALISERAR KÄRNAN OCH SKAPAR PROCESSER*)
CONST
-INC PAGNT680      (*GL. CONST DEKL.*)
TYPE
-INC PAGNT510      (*GL. TYPE DEKL.*)
VAR
-INC PAGNT670      (*GL. VAR DEKL.*)
-INC PAGNT870      (*ANV. VAR DEKL.*)
PROCEDURE CREATEPROCESS(NR:INTEGER4;MEMREQ:INTEGER4);PASCAL;
PROCEDURE INITKERNEL(STACKREQ:INTEGER4);PASCAL;
PROCEDURE SETPRIORITY(PRIORITY:INTEGER4);PASCAL;
PROCEDURE INITPRINTER;PASCAL;
PROCEDURE INITMAILBOX(VAR BOX:MAILBOX);PASCAL;
BEGIN
   NEW(KERNEL);
   INITKERNEL(1000);
   INITPRINTER;
   INITMAILBOX(DISCREQUEST);
   CREATEPROCESS(8,2000);
   CREATEPROCESS(12,2000);
   SETPRIORITY(-1);
END.
```

```
(*ÅL+,O+,A4*)
SUBPROGRAM READDATA(INPUT,OUTPUT);
(*ENTRY PROCEDUR*)
-INC PAGNT790
PROCEDURE READDATA(VAR Y:INTEGER);
BEGIN  WITH KERNELÖ.DATABAS DO
       BEGIN
          WAIT(OK);
          Y:=DATA;
          SIGNAL(OK);
       END;
END.
```

```
(*ÅL+,O+,A4*)
SUBPROGRAM NEWDATA(INPUT,OUTPUT);
(*ENTRY PROCEDURE*)
-INC PAGNT790
PROCEDURE NEWDATA;
BEGIN  WITH KERNELÖ.DATABAS DO
       BEGIN
          WAIT(OK);
          CAUSE(NEWS);
          SIGNAL(OK);
       END;
END.
```

```
(*PAGNT742*)
(*ÅL+,0+,A4*)
PROGRAM P1(INPUT,OUTPUT);
VAR KERNEL:INTEGER4;
    SVAR2:INTEGER2;
    SVAR:INTEGER4;
PROCEDURE SETPRIORITY(PRIORITY:INTEGER4);PASCAL;
PROCEDURE ACTIVATE(NR,NEWPRIORITY:INTEGER4);PASCAL;
PROCEDURE WAITTIME(TIME:INTEGER4);PASCAL;
BEGIN
   WHILE TRUE DO
   BEGIN
      WAITTIME(20);
      WRITELN('VÄLJ PROGRAM');
      READLN(SVAR2); SVAR:=SVAR2;
      IF SVAR=5 THEN ACTIVATE(5,10)
      ELSE IF SVAR=6 THEN ACTIVATE(6,70)
      ELSE WRITELN('5 ELLER 10 VAR DET');
      SETPRIORITY(-2);
   END;
END.
```

```
(*PAGNT802*)
(*ÅL+,0+,A4*)
PROGRAM WATCHDOG(INPUT,OUTPUT);
CONST NAME=1000;
VAR KERNEL:INTEGER4;
    CH:CHAR;
    ENABLE(.ORIGIN 16ÄFF08E3.):CHAR;
PROCEDURE DI;PASCAL;
PROCEDURE EI;PASCAL;
PROCEDURE WAITTIME(TIME:INTEGER4);PASCAL;
PROCEDURE SETPRIORITY(PRIORITY:INTEGER4);PASCAL;

BEGIN
   SETPRIORITY(-NAME);
   WHILE TRUE DO
   BEGIN
      WAITTIME(10);
      DI;
      CH:=ENABLE;
      IF CH <> CHR(00) THEN BEGIN ENABLE:=CHR(00);
                                  ENABLE:=CH;
                            END;
      EI;
   END;
END.
```

```
(*ÅL+,O+,A2*)
(*PAGNT808*)
PROGRAM DISCTESTER(INPUT,OUTPUT);
LABEL 999;
TYPE
-INC PAGNT510
VAR  KERNEL:KERNELPTR;
     DISCREQUEST:MAILBOX;
     ACK:MAILBOX;
     REQUEST:MESSREF;
     CH:CHAR;
     BUFF(.ORIGIN 16Ä6000.):PACKED ARRAY(.0..127.) OF CHAR;
     I,ANTALPOS,GRANS,B:INTEGER2;
PROCEDURE INITMAILBOX(VAR BOX:MAILBOX);PASCAL;
PROCEDURE SENDMESSAGE(BOX:MAILBOX; VAR MESS:MESSREF);PASCAL;
PROCEDURE RECEIVEMESSAGE(BOX:MAILBOX; VAR MESS:MESSREF);PASCAL;
BEGIN
   INITMAILBOX(ACK);
   NEW(REQUEST); REQUESTÖ.NEXTMESS:=NIL;
   REQUESTÖ.ACKNOWLEDGE:=ACK;
   REQUESTÖ.ADR:=16Ä6000;
   WHILE TRUE DO
   BEGIN
      WITH REQUESTÖ DO
      BEGIN
999:     WRITELN(' COMMAND? I:INIT F:FIND R:READ W:WRITE');
         READLN(CH);
         CASE CH OF
            'I':   BEGIN COMMAND:=INIT;
                         WRITELN(' SELECT MODE ',CHR(64),'=S B=D');
                         READLN(T);
                   END;
            'F':   BEGIN COMMAND:=FIND;
                         WRITELN(' SELECT DRIVE,TRACK,SEKTOR');
                         READLN(DRIVE,TRACK,SEKTOR);
                   END;
            'R':   COMMAND:=INP;
            'W':   BEGIN COMMAND:=OUTP;
                         FOR I:=32 TO 127 DO BUFF(.I.):=CHR(I);
                         FOR I:=0 TO 31 DO BUFF(.I.):=CHR(43);
                   END;
                   OTHERWISE: GOTO 999;
         END;
      END;
      SENDMESSAGE(DISCREQUEST,REQUEST);
      RECEIVEMESSAGE(ACK,REQUEST);
      WITH REQUESTÖ DO
      BEGIN
         CASE COMMAND OF
            XINIT: WRITE('  INIT SUCESSFUL');
            XFIND: WRITE('  FIND SUCESSFUL');
            XINP:  BEGIN
                      ANTALPOS:=0;
                      FOR I:= 0 TO 127 DO
                      BEGIN
                         ANTALPOS:=ANTALPOS+1;
                         CH:=BUFF(.I.);
                         IF CH < CHR(0) THEN
                          FOR B:=128 TO ORD(CH) DO WRITE(' ')
                         ELSE IF CH < CHR(32) THEN WRITE('.')
                         ELSE WRITE('CH');
                         IF ANTALPOS = 64 THEN WRITELN;
                      END;
                   END;
            XOUTP: WRITE('   WRITE OPERATION SUCESSFUL');
            ERROR: WRITE(' ERROR CODE:',S);
         END;(*CASE COMMAND*)
         WRITELN;
      END;
   END;
END.
```

```
TYPE    PROCESSPTR=ÖPROCESSREC;

        SEMAFOR=ÖSEMAFORREC;
        EVENT=ÖEVENTREC;
        KERNELPTR=ÖKERNELREC;

        PROCESSREC=RECORD
           SUC,PRE:PROCESSPTR;
           PTR:INTEGER4;
           PRIORITY,TIME:INTEGER4;
        END;

        SEMAFORREC=RECORD
           COUNTER:INTEGER4;
           WAITING:PROCESSPTR;
        END;

        EVENTREC=RECORD
           REENTRY:SEMAFOR;
           DELAYED:PROCESSPTR;
        END;

        RESOURCE=RECORD
           GUARD:SEMAFOR;
        END;

        MAILBOX=ÖPBOX;
        MESSREF=ÖMESSAGE;

        PBOX=RECORD
           FIRSTMESS:MESSREF;
           MUTEX:SEMAFOR;
           SEND:EVENT;
        END;

-INC PAGNT511      (*TYPE DEKL. FOR MESSAGE*)

-INC PAGNT800      (*TYPE DEKL. FOR MONITOR*)

-INC PAGNT512      (*KERNEL RECORD*)
```

```
MESSTYPE=(INIT,FIND,INP,OUTP,XINIT,XFIND,XINP,XOUTP,ERROR);

MESSAGE=RECORD
   NEXTMESS:MESSREF;
   ACKNOWLEDGE:MAILBOX;
   ADR:INTEGER4;
   CASE COMMAND:MESSTYPE OF
     INIT:          (T:CHAR);
     FIND:          (DRIVE,TRACK,SEKTOR:INTEGER2);
     INP,OUTP:      ();
     ERROR:         (S:PACKED ARRAY(.0..5.) OF CHAR);
   END;



    (*START MONITOR*)
    MONITOR=RECORD
       OK:SEMAFOR;
       NEWS:EVENT;
       DATA:INTEGER4;
    END;
    (*END MONITOR*)



     KERNELREC=RECORD
        RUNNING,READYQUE,TIMEQUE:PROCESSPTR;
        FREETOP:INTEGER4;
        PRINTER:RESOURCE;
        (*SLUT PÅ OBLIGATORISK DEL*)
        DATABAS:MONITOR;
     END;
```

```
PROCEDURE EI;PASCAL;
PROCEDURE DI;PASCAL;
PROCEDURE INITNUCLEUS(SREQ:INTEGER4;VAR FREE,MAIN:INTEGER4);PASCAL;
PROCEDURE NEWPROCESS(STARTADR,HEAPBASE,STACKBASE:INTEGER4;
          VAR CHILD:INTEGER4);PASCAL;
PROCEDURE IORESUME(PTR:INTEGER4;VECADR:INTEGER2);PASCAL;
PROCEDURE RESUME(P:INTEGER4);PASCAL;
PROCEDURE REMOVE(P:PROCESSPTR);PASCAL;
PROCEDURE PUTPRIORITY(P,Q:PROCESSPTR);PASCAL;
PROCEDURE PUT(P,Q:PROCESSPTR);PASCAL;
PROCEDURE SCHEDULE;PASCAL;
(*SLUT PÅ KÄRNANS INTERNA DEKLARATIONER*)
PROCEDURE SETPRIORITY(PRIORITY:INTEGER4);PASCAL;
PROCEDURE PASSIVATE(NAME:INTEGER4);PASCAL;
PROCEDURE ACTIVATE(NR,NEWPRIORITY:INTEGER4);PASCAL;
PROCEDURE CREATEPROCESS(NR:INTEGER4;MEMREQ:INTEGER4);PASCAL;
PROCEDURE INITKERNEL(STACKREQ:INTEGER4);PASCAL;
PROCEDURE INITSEM(VAR SEM:SEMAFOR;INITVAL:INTEGER4);PASCAL;
PROCEDURE WAIT(SEM:SEMAFOR);PASCAL;
PROCEDURE SIGNAL(SEM:SEMAFOR);PASCAL;
PROCEDURE INITEVENT(VAR E:EVENT;SEM:SEMAFOR);PASCAL;
PROCEDURE AWAIT(E:EVENT);PASCAL;
PROCEDURE CAUSE(E:EVENT);PASCAL;
PROCEDURE INITMAILBOX(VAR BOX:MAILBOX);PASCAL;
PROCEDURE SENDMESSAGE(BOX:MAILBOX;VAR MESS:MESSREF);PASCAL;
PROCEDURE RECEIVEMESSAGE(BOX:MAILBOX;VAR MESS:MESSREF);PASCAL;
PROCEDURE WAITIO(VECADR,STATUSREG:INTEGER4);PASCAL;
PROCEDURE WAITTIME(TIME:INTEGER4);PASCAL;
```

BIL 2

IDENTITET	U4.1.10.5	TYPBET	DSCA 114	S80-SPEC-155
BENÄMNING. ENG	25 t	Asyncronous Comm. module		
BENÄMNING. SV	25 t	Asynkronkommunikationsenhet		
FÖRFATTARE	L Nilsson YLKHD, 2648	ANSV	L Nilsson	GODK
FÖR YTTRANDE				

E

FÖR KÄNNEDOM

YLKH,YLK Danielsen,YLK Claesson,YLKHD Greijer,YLKHM,YLKHP,YLKSP,YLKSS Pauly,YLKMM, YLKTS,IKUA,IXU,OAH,NCC,NCCD,NCCG,YLKI,YLBK,YLBA,JKE,YLK Ögren,YLF,RKU

ENB. FÖRSÄTTBL. OCH SAMMANF.

E

S-80 SPEC NR	DATUM	ERSÄTTER S-80-SPEC NR	ERSÄTTER S-80-SPEC BLAD	REMISS-TID UTGÅR	FASTLAGD DATUM	ANMÄRKNING
032	79-05-20	-	-	79-05-01		
155	81-02-19	032			81-02-20	Redaktionella ändringar

DSCA 114 Asynkron kommunikationsenhet

ALLMÄNT

Typbeteckning:	DSCA 114
Beställningsnummer:	
Kanaler:	4 st helt oberoende asynkrona full duplex transmissionskanaler
Transmissionshastighet:	Kontinuerligt valbart upp till 38,4k baud "Split speed" möjlig
Dataformat:	5, 6, 7 eller 8 bitar; 1, 1 1/2 eller 2 stoppbitar
Paritet:	Ingen, udda eller jämn
Transmissionsutgång:	1) V24 med komplett modemhandskakning 2) Optoisolerad differentiell utgång med "carrier detect"
Datorinterface:	Blockbuffring. Valbart vektoriserat avbrottssystem. Karaktärsökmod.
Felövervakning:	Indikatorer för modemhandskakning och datatrafik. Slingtest av data och modemsignaler. Test av avbrottssystem.

INNEHÅLLSFÖRTECKNING

Appendix A

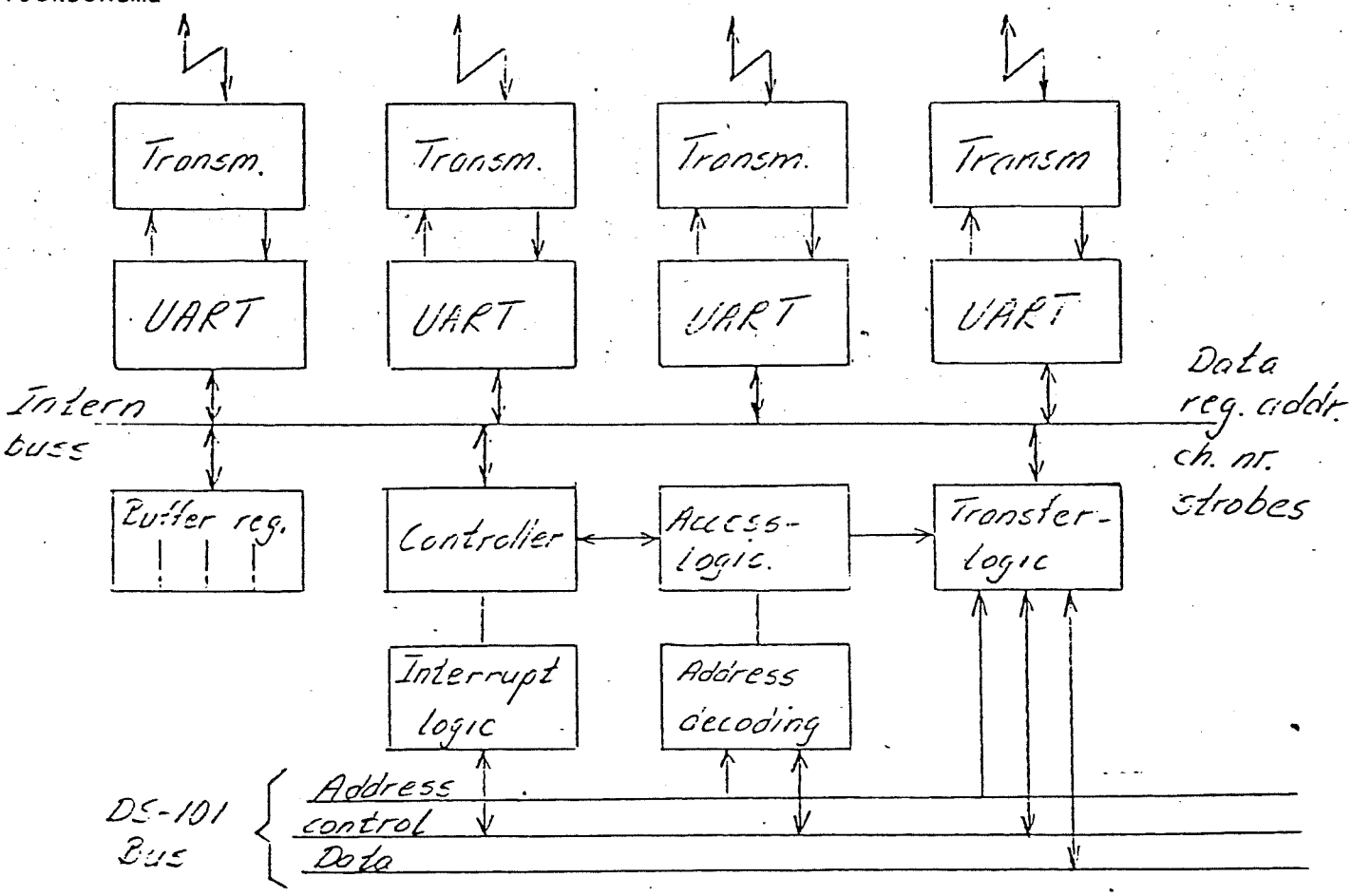
1
ANVÄNDNING

DSCA 114 är ett kretskort i ASEAs datorsystem DS-101C. Kortets 4 asynkrona kanaler är avsedda att användas mot periferienheter med asynkront seriesnitt t ex skrivmaskiner, bildskärmar, industriterminaler etc. Allmänt avser modulen möta trenden mot allt ökad användning av V24 i periferienheter med måttliga transmissionshastigheter. Kortet avses också att användas för asynkron dator/dator-kommunikation. Enhetens blockbuffringssystem medger relativt höga transmissionshastigheter utan att lasten på centraldatorn blir anmärkningsvärt hög. Kortet kan anslutas till alla modem och terminaler med V24 snitt. För att medge anslutning av terminaler på längre avstånd finns dessutom ett inbyggt modem för optoisolerad differentiell tvåtrådskommunikation kompatibel med ASEAs korthållsmodem KM1-D.

För att ytterligare nedbringa CPU-lasten finns ett avancerat helt programmerbart vektoriserat avbrottssystem som kan begära avbrott på en avbrottsnivå. Kortet är konstruerat för att möjliggöra långtgående funktionstest via "on-line"-testprogram.

2
FUNKTION

2.1
Blockschema



Figur 2.1 Principiell struktur

Nedanstående avsnitt är avsett att ge en snabb inblick i funktionen för att detaljer som sedan kommer att beröras skall kunna placeras in i sitt sammanhang.

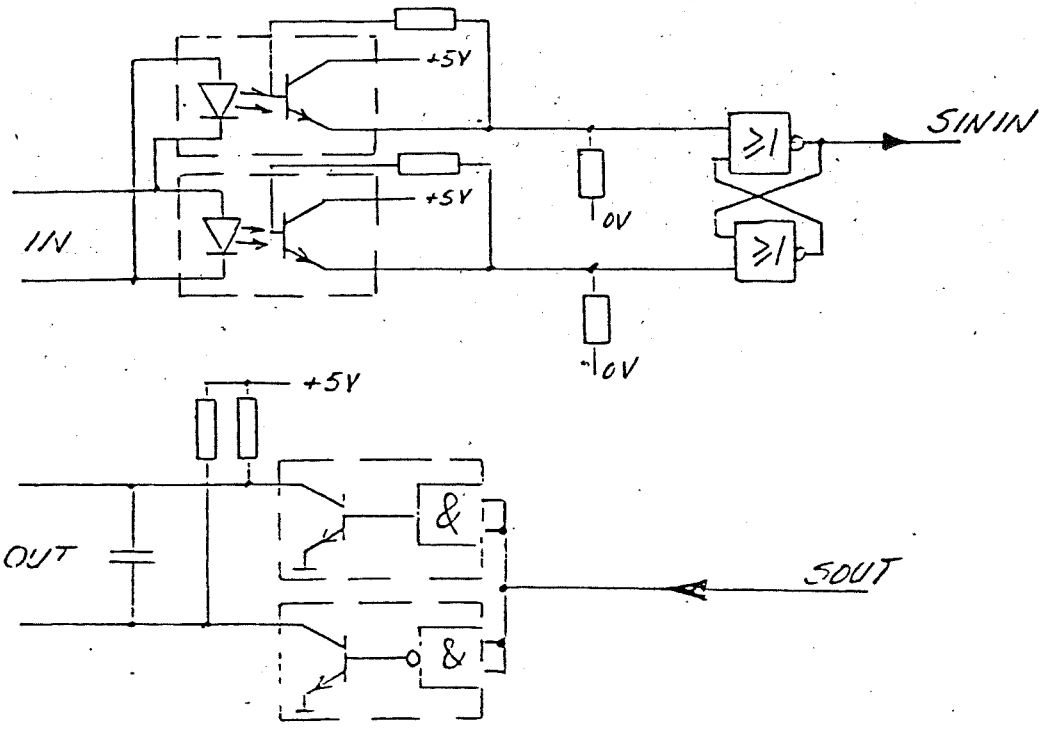
Seriekanaler

4 seriekanaler finns tillgängliga. De är helt oberoende av varandra och innehåller var för sig all nödvändig logik för följande funktioner:

- Parallell serieomvandling av data med hantering av start, stopp och paritetsbitar i den seriella dataströmmen.
- Felkontroll av inkommande data och service av parallell-buffertar.
- Programmerbar "Baud rate"-generator för transmissionshastigheter upp till 38.4 kbaud. Alla förekommande hastigheter därunder kan väljas.

2.2.2
Korthållsmodem

För längre avstånd - upp till 300 m - kan DSCA 114:s balanserade differentiella signalsnitt med optoisolering användes.
Se figur 2.3



Figur 2.3 Balanserat differentiellt signalsnitt

Följande modemsignal är kopplade mot modemet:

Received signal line detector

Om ström flyter i någon riktning av tillräcklig nivå är RSLD = TILL. RSLD är kopplad som en ellerfunktion mellan V24-snittet och korthållsmodemet.

2.2.3
Transmissionshastighet

Varje UART-funktion inkluderar en räknare som kan programmeras från huvudprocessorn för att ge önskad transmissionshastighet. Programmeringen tillgår så att bit D7 i register E7 sätts till 1 varefter ett 16 bits tal motsvarande önskad transmissionshastighet skrivs i register E1 och E3. Observera att inskrivningsordningen är vänd så att minst signifikant del skrivs i E1 och mest signifikant del i E3. Motsatt förhållande är annars det vanligast förekommande i DS-101. När talet skrivits in kan bit D7 i E7 åter nollställas. Ovanstående finns också beskrivet i Appendix A.

Talet som skall skrivas in kan beräknas med nedanstående ekvation.

Devisor = 153.600/baudrate.

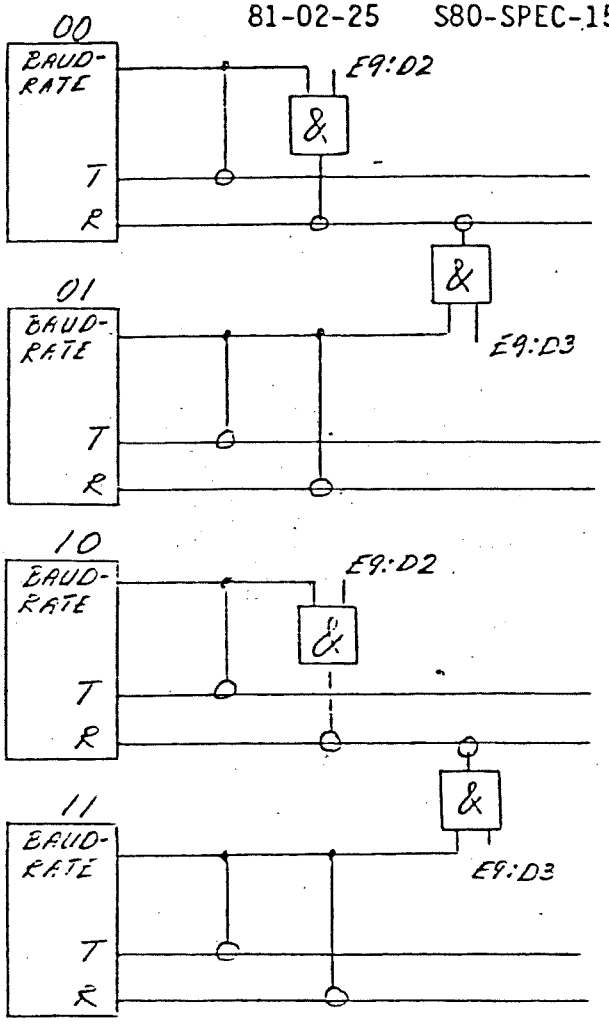
Värdet för ett antal vanliga hastigheter anges i figur 2.4.

Desired Baud Rate	Divisor Used to Generate 16x Clock	Percent Error Difference Between Desired & Actual
50	3072	-
75	2048	-
110	1396	0,026
134,5	1142	0,001
150	1024	-
300	512	-
600	256	-
1200	128	-
1800	85	0,392
2000	77	0,260
2400	64	-
3600	43	0,755
4800	32	-
7200	21	1,587
9600	16	-
19200	8	-
38400	4	-

Figur 2.4 Divisortal för vanliga transmissionshastigheter

"Split-speed" kan åstadkommas i begränsad omfattning. Om bit D2 i reg E9 sätts till 1 och D3 till 0 erhålls samma hastighet för mottagare som för sändare, nämligen inställd kanalhastighet. Om däremot D3=1 och D2=0 i reg E9 kommer mottagarkanalen att arbeta med den hastighet som närmast högre kanal arbetar med. Kanal 0 kan alltså ta emot med hastighet inställd på kanal 1 och kanal 2 med kanal 3:s hastighet. D2 och D3 i reg. E9 behöver ej sättas för kanal 1 och 3.

Denna egenskap kan vara värdefull mot modem med så kallad backkanal.



Figur 2.5 Arrangemang för "split-speed"

Kortets totala transmissionshastighet är begränsad till 44 kbaud sammanlagt för bägge riktningarna. Denna totala resurs kan fördelas på olika sätt som till exempel enligt nedan:

4 full duplex 4800 kbaud
4 halv duplex 9600 kbaud

2 full duplex 9600 kbaud
2 full duplex 1200 kbaud

2 halv duplex 19200 kbaud
2 halv duplex 2400 kbaud

2.3 UART

Alla interna register i UART är direkt avläsbar från huvudprocessorn. Den ligger med början på register E1 och udda register framåt.

Registerstrukturen finns beskriven i Appendix A.

Eftersom kontrollern läser av registern även den måste följande noga iakttas. Inga avläsningar sker från kontrollern utöver vad som nedan sägs.

- Vid alla avbrott läser kontrollern register E5 och omformar denna information till en avbrottsbegäran med en avbrottsvektor till huvudprocessorn som direkt anger orsaken. E5 är alltså alltid avläst en gång när huvudprocessorn får information om avbrott och skall således ej läsas igen. Vid "Linje status" avbrott skall huvudprocessorn läsa reg. EB.

- Vid dataavbrott överför kontrollern data till buffrarna där huvudprocessorn läser och skriver. Dataregistret E1 skall således endast användas för att sätta transmissionshastighet. Ingenting hindrar att huvudprocessorn läser och skriver data direkt till UART:en. Detta medför emellertid att buffertfunktionen ej kan garanteras.

- Alla modemsignaler sätts respektive avläses utan påverkan från kontroller. Signalerna är oberoende av datatrafiken. Alla ingående signaler kan fås att begära avbrott vid varje nivåväxling med undantag av uppringningssignalen (RI) som bara begär avbrott vid possitiv flank.

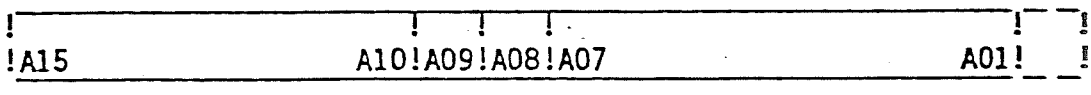
- UART:en intar vid uppstart det status som i APPENDIX A beskrivs som "RESET-status".

2.4 Adressdisposition

Som framgår av blockschemabeskrivning upptar kortet fyra I/O-adresser enligt specifikation för DS 100PBC-bussen. Varje I/O-adress omfattar 256 register som kan avläsas och skrivas i från bussen via en speciell accesslogik.

Kortets adress ställs in med en 6-ställig bygelgrupp som anger adress till en grupp om 4 konsekutiva I/O-adresser (de sex mest signifikanta adressbitarna av åtta). (Se kapitel 7.)

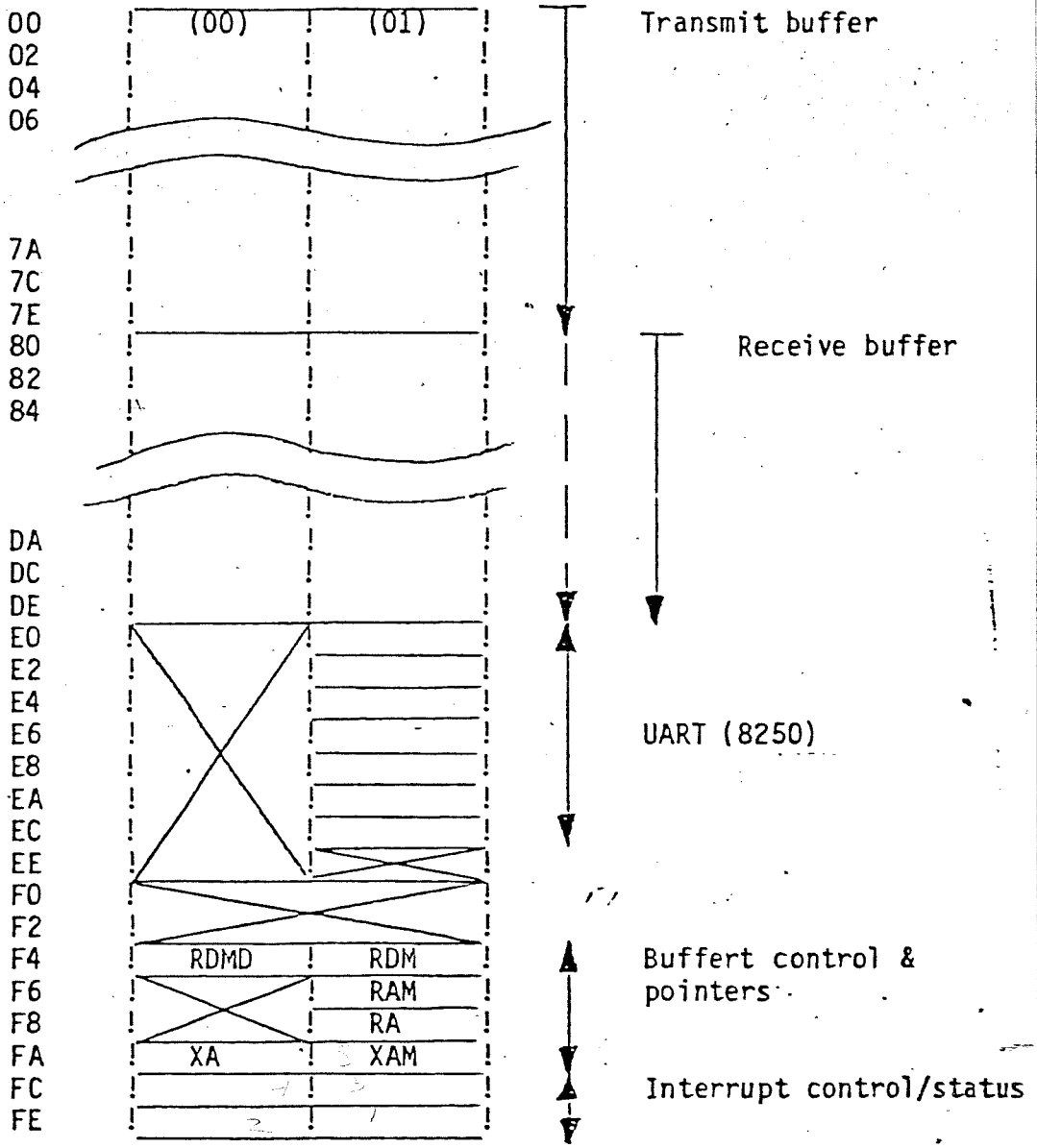
Adress till ett register i en kanal sätts alltså samman enligt nedan:



Kortadress (inställd med bygelgrupp) | Kanal-nummer (0-3) | Registeradress

Figur 2.6 Adressavkodning

Varje kanals adressfält om 256 byte kan via accesslogiken enbart nås via 8 bitsoperationer. Accesslogiken se till att UART och övriga register (realiserade med skriv/ läsminne)ser ut som ett kontinuerligt adressfält.



Figur 2.7 Registerdispossition per kanal

Adressfältet kan delas upp i följande delar:

1) #00-7F (DF)

Sändningsbuffer om 128 tecken som kan utökas till 224 om mottagarbuffert ej används.

2) #80-DF

Mottagarbuffer om 96 tecken.

3) #E1, E3, E5, E7, E9, EB, ED

Interna register i UART 8250.

4) #F4, F5, F7, F9, FA, FB

Pekare och övriga kontrollord för styrning av bufferthanteringen.

5) #FC-FF

Register av standardiserad natur för styrning och statusavläsning av i huvudsak avbrottssystemet.

Detaljerad beskrivning av dessa fält följer i närmast följande avsnitt.

2.5 Bufferthantering

Följande princip är grundläggande:

- Om bufferthanteringen är deaktiv, (genom att pekare som sätts från centralprocessern anger detta) påverkas ej UART:en av kortets kontrollerfunktion. I detta läge kan kortet ses som rena UART-funktionen och alla egenskaper reflekteras direkt mellan UART och DS 100PBC-buss.

- Om bufferthantering önskas i någon riktning anges detta genom att ställa upp buffertpekare varvid överföringen startas omedelbart. Härvid är det viktigt att notera nödvändigheten av att stoppa normal dataöverföring via avbrott från UART. Alltså när data önskas överföras mellan UART och buffer skall motsvarande avbrott vara avstängd i aktuell UART.

2.5.1 Sändning

Följande register styr överföringen:

Sändningsbuffer adress (XA)

Adress om 1 byte som anger vilket data som står i tur att sändas ut. XA kan läsas kontinuerligt för att kontrollera sändningens framåtskridande. Sändningsbufferten har normalt adress 00-7F (128 tecken) men kan utökas till 00-DF (224 tecken) om inte mottagarbufferten utnyttjas.

Sändningsbuffer adressgräns (XAM)

Adress om 1 byte som anger att sändningsbufferten är tom. Gränsen kan ställas till 7F eller om ej mottagarbufferten används till DF. När sista tecknet skickas ut till UART:en för sändning begärs avbrott av centralprocessorn. Om XA är större än XAM sänds inget tecken till UART:en mer. Initialt bör alltså XA alltid ställas större än XAM vilket DSCA 114 automatiskt gör vid INIT (XA=01, XAM=00).

Sändning går således till enligt nedanstående:

1) Lägg ut data i buffert.

2) Kontrolera om XAM = 00 om ej sätt XAM = 00

3) Ställ upp XA 00

4) Ställ upp XAM

Sändning pågår nu tills bufferten är tom och dataavbrott genereras med en vektor som anger sändning avslutat. (Samma vektor som fås vid direkt styrning av UART.)

2.5.2
Mottagning

Följande register styr mottagarbufferten:

Mottagningsadress (RA)

Adress om en byte som anger nästa tomma mottagarbuffert ledig för data. RA kan läsas kontinuerligt för att kontrollera antalet mottagna tecken. Räknaren räknas fram kontinuerligt runt mottagarbufferten:

80,81,,,,DE,DF,80,81,,,,

Bufferten blir således totalt 96 tecken. RA kan naturligtvis ställas till önskat startvärde före varje enskild sträng som önskas mottas, förslagvis till värdet 80.

Mottagningsadress test (RAM)

Adress om en byte som anger att avbrott skall begäras när motsvarande buffer fylls. Detta innebär emellertid ej att fyllning av bufferten upphör utan denna fortsätter enligt beskrivning av RA. Om en kontinuerlig sträng mottas där asynkronism gäller mellan teckenmottagning och ingripande från centralprocessorn rekommenderas ej ändring av RA i flykten eftersom data då kan föras in i bufferten och RA uppdateras utan att centralprocessorn kan notera detta. Om RAM ställs till 00 utläses inget data från UART till buffer. Detta kan användas om data önskas nonchaleras eller tas direkt via avbrott från UART.

Initialt står RA = RAM = 0 efter INIT.

Avbrottsvektor (reg FD)

Vektorn måste skrivas in i enheten via DS 100PBC-bussen innan enheten används för avbrottsgenerering. I princip kan centralenheten välja vilken information den önskar och senare utnyttja denna som den själv finner lämpligt. Vektorn läses automatiskt ut på bussen om enheten känner sig adresserad via en "IGO-signal". DONE och ADVER ges på samma sätt som en vanlig läsoperation.

Endast en kanal i taget kan begära avbrott på samma nivå. Detta märks emellertid ej från centralenheten. (Intern prioritering på kortet istället för prioritering via IGO/IGI på bussen är irrelevant.)

Vid avbrott förändrar enheten inskriven information i bit D00-D01 för att ange avbrottsorsak. Eftersom vektorn vanligen används som pekare i startadresstabell fås på detta sätt en automatisk vektorisering efter orsak. Bit D00-D01 kodas på samma sätt som UART:ens orsaksstatus och har således följande uttydning:

00: Modemstatusförändring.
01: Sändarreg tomt i UART, eller vid buffertanvändning, sändarbuffert tom.
10: Data mottaget till UART, eller vid bufferthantering, önskat antal tecken mottagna eller önskat söktecken funnet.
11: Linjestatusorsak.

För mer detaljerad beskrivning av ovanstående se tabell 5 i appendix 1 samt avsnitt 2.5 om bufferternas funktion.

Kontrollstatus ord (^{H}FF)

För varje kanal kan avbrottsnivån aktiveras. Detta görs genom att sätta bit D01, D02 enligt nedanstående:

D02, D01: 00 avbrott ej anslutet
01 nivå 1 ansluten
10 nivå 2 ansluten

Avbrottsnivån som gäller för kortet aktiveras separat för varje kanal genom att sätta bit D01 eller D02 i reg. FF.

D00=1 indikerar att avbrottsbegäran föreligger från kanalen ifråga. Denna bit skall nollställas av centralprocessorn efter det att avbrottsorsak är identifierad.

D07=1 indikerar att fel uppstått vid intagning i mottagarbuffer. Även denna indikering skall nollställas av centralprocessorn.

BIL 3

IDENTITET	U4.1.10.52	TYPBET	DSTC 120	S80-SPEC-153
BENÄMNING. ENG	25 t	Term.module-async.com.		
BENÄMNING. SV	25 t	Ansl.enhet-asynk.komm.		
FÖRFATTARE	L Nilsson YLKHD, 2648	ANSV	L Nilsson	GODK
FÖR YTTRANDE				

FÖR KÄNNEDOM

YLKH,YLK Danielsen,YLK Claesson,YLKHD Greijer,YLKHM,YLKHP,YLKSP,YLKSS Pauly,YLKMM, YLKTS,IKUA,IXU,OAH,NCC,NCCD,NCCG,YLKI,YLBK,YLBA,JKE,YLK Ögren,YLF,RKU

ENB. FÖRSÄTTBL. OCH SAMMANF.

S-80 SPEC NR	DATUM	ERSÄTTER S-80-SPEC NR	ERSÄTTER S-80-SPEC BLAD	REMISS-TID UTGÅR	FASTLAGD DATUM	ANMÄRKNING
046	79-09-03	-	-	79-09-12		reaktionella ändringar
153	81-02-19	046			81-02-19	

Ansl. enhet asynkr. komm. DSTC 120

INNEHÅLLSFÖRTECKNING

Anslutningsenhet för asynkronkommunikationsenhet DSCA 114.

Sammandrag

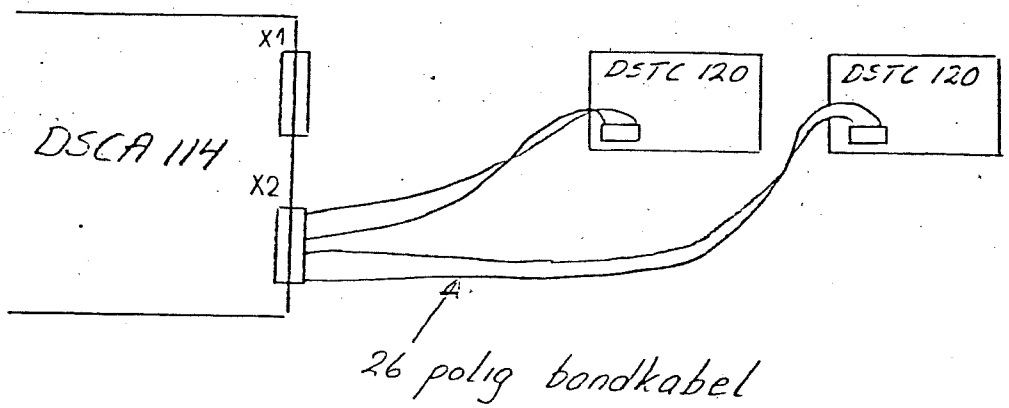
Typenhet	DSTC 120
Användning	Anslutning av asynkronkommunikations-enhet DSCA 114 till omgivningen via ett standardsnitt V24 till terminaler som floppydisk, skrivare m m.
Kanalantal	2 st (DSCA 114 har 4 kanaler).
Anslutning	- 26 polig bandkabel mot DSCA 114 (2 kanaler) - DB 25 kontakt/kanal som ger V24 ut. - Plint för anslutning av ASEAs korthållsmodem KM1-D.
Funktioner	Avstörning av signaler i balanserat strömsnitt.
Mekanik	1/4 19" anslutningsenhet (ca 80 x 120 mm).
Strömförsörjning	Ingen.

1
ANVÄNDNING

DSTC 120 används för att ansluta DSCA 114 mot yttre enheter som floppydisk, skrivare, m m. Två av DSCA 114's 4 kanaler kan anslutas per anslutningsenhet (se figur 1).

2
ANSLUTNINGSTEKNIK

Enheten ansluts med 26 polig bandkabel till DSCA 114 enligt figur 2.1.



Figur 2.1 Anslutningsskiss

Signalerna från DSCA 114 är av två slag:

1) Signalsnittet V24 med det elektriska snittet V28 för signalerna.

2) Balanserad strömslinga som ASEAs KM1-D.

- V24 signalerna ansluts på standard sätt till utgående DB25.

  Följande signaler finns:

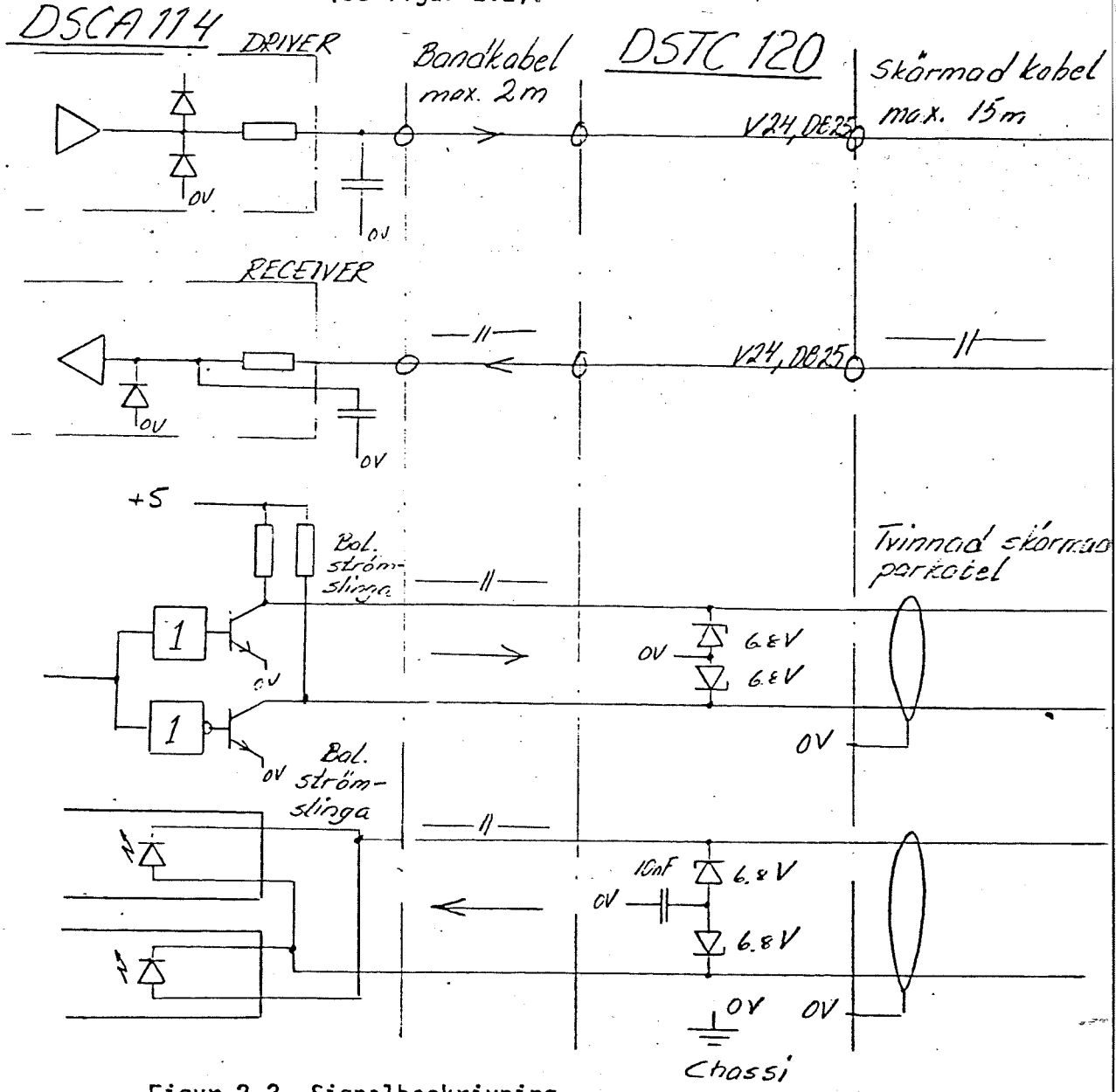
signal	riktning
RTS	→
CTS	←
DTR	→
RLSD	←
DSR	←
SININ	←
SOUT	→
OV	↔
RI	←

- Balanserad strömslinga ansluts till plintenhet (se figur 2.2).

  Kontakternas stiftkonfiguration finns beskriven i bilaga 1.

3
AVSTÖRNING

Signaler tillhörande balanserat strömsnitt avstörs med zenerdioder (se figur 2.2).



Figur 2.2 Signalbeskrivning

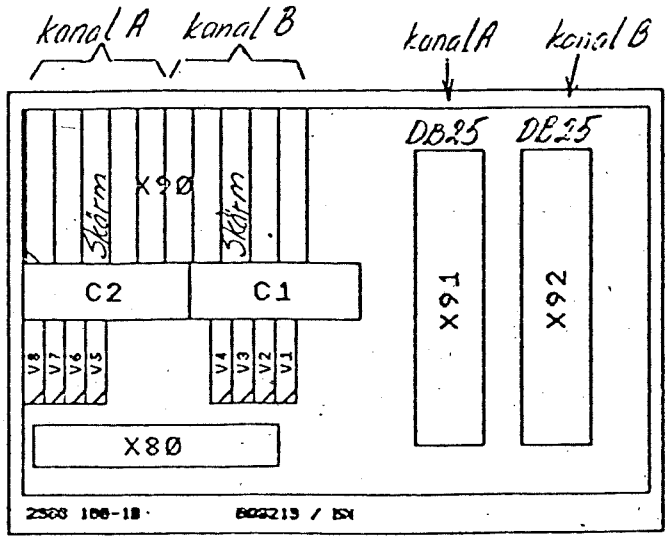
4
MEKANIK

Se figur 4.1.

X90: frånskiljbara plintar

X80: 26 polig bandkabelkontakt

X91, X92: DB 25-S



Figur 4.1 Anslutningsenhet, asynkronenhet

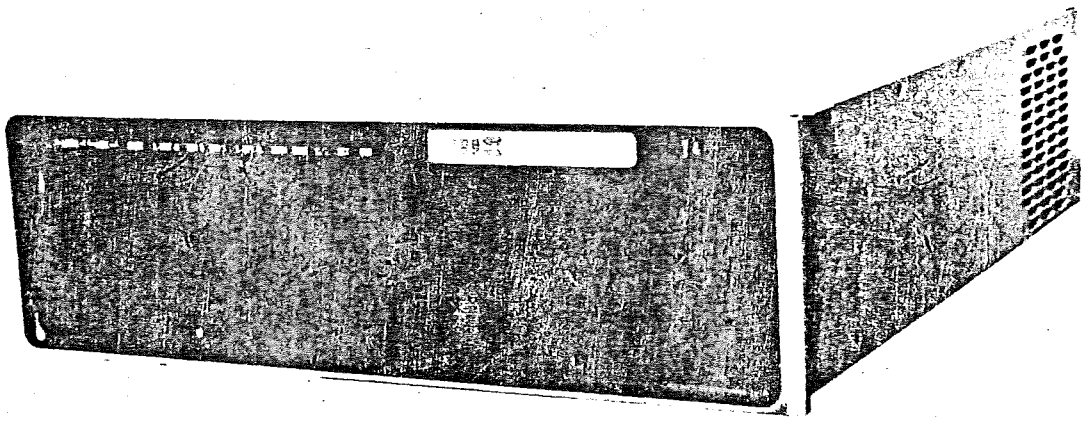
5
FÖRLÄGGNING

Det är av vikt att alla kablar förläggs så långt från processkablage som möjligt. Anslutningsenheten bör därför om möjligt ej placeras tillsammans med processkabelanslutningsenheter. V24 kablar bör överhuvud taget ej dras tillsammans med processkablage.

Scientific Micro Systems

SMS

FT0200I
OEM Manual



Floppy Disk

StorageSystem

TABLE 3-A. COMMON CONTROL COMMANDS

Function:	DRIVE SELECT
Keybd Entry*:	CTL O X where X=0 or 1
ASCII Char Seq:	SI X
Description:	Selects disk drive; returns primary status to host.

Function:	SEEK TRACK AND SECTOR
Keybd Entry:	CTL V MMNN where MM = track number and NN = sector number.
ASCII Char Seq:	SYN MMNN
Description:	Steps the head to track MM (00-76), stores sector NN or NNN for next operation. Primary status returns immediately following the command. Maximum sector number NN varies with the format type selected. When an invalid track or sector value is sent to the FT0200I it returns primary status but it does not move the head. Overlapped seek is provided in the FT0200I and one command can be queued while a seek is in process.

Function:	DONE
Keybd Entry:	CTL D
ASCII Char Seq:	EOT
Description:	Lights the DONE indicator. Causes DLE EOT response.

Function:	ERROR
Keybd Entry:	CTL X CTL D
ASCII Char Seq:	CAN EOT
Description:	Lights the ERROR indicator. Causes DLE EOT response.

Function:	PRIMARY STATUS
Keybd Entry:	CTL N
ASCII Char Seq:	SO
Description:	Provides six ASCII characters of status information I D T T S S R where: I= status character (reference section IV-E); D=0 when drive 0 is selected; D=1 when drive 1 is selected; T T= current track address (between 00 and 76); S S= current sector address (between 01 and 88 max); R= C/R turnaround character (optional). Reference Section VI-B.

* Keyboard entered commands require depression of the Control (CTL) key. While the CTL key is held depressed, enter the character which appears after CTL. This character should immediately be followed by the character(s) to be entered after the command. No spaces should appear between entries.

TABLE 3-A. COMMON CONTROL COMMANDS (continued)

Function:	RESET
Keybd Entry:	CTL Z
ASCII Char Seq:	SUB
Description:	Resets the system to internally strapped configuration, seeks both drives if ready to track 0, and sets the FT0200I to the Line mode. The Auto Baud initialization sequence must be repeated on the Local and Line ports when selected. Reset command is equivalent to operating the RESET switch. There is no transmitted response to this command. The host must delay commands for one second after Reset.
Function:	FORMAT
Keybd Entry:	CTL Y
ASCII Char Seq:	EM
Description:	Formats the diskette on the selected drive in the format specified by the latest INIT command following a reset. If there has been no INIT command prior to this command following a power up/reset, the format defaults to the selected format specified by the jumpers. Reference section VI-B. Primary status is returned when complete.
Function:	SECONDARY STATUS
Keybd Entry:	ESC CTL N
ASCII Char Seq:	ESC SO
Description:	Provides two characters of secondary status information. Reference section IV-F.
Function:	COPY
Keybd Entry:	ESC CTL S NNN where NNN (up to 999) is the number of sectors to be copied in the sector mode. NNN is not used in X-ON/X-OFF mode.
ASCII Char Seq:	ESC DC3 NNN
Description:	Allows all or any part of the information on Drive 1 to be copied onto drive 0. It's main use is in selective copy, offloading this task from the host. Selective copy is used to organize the data stored on one diskette onto another diskette. In the X-ON/X-OFF mode the copy stops after recording a read stop code. Primary status is returned when complete. This command is intended as an extension of the front panel copy when using a local terminal.*

*NOTE: Formats must be the same for both copy and copied fields. When recording a copy, the field of both drives must not span the boundary between track 0 and track 1 except in 3740 format.

TABLE 3-A. COMMON CONTROL COMMANDS (continued)

Function: BACKSKIP
Keybd Entry: CTL A
ASCII Char Seq: SOH
Description: Decrements the current sector by one. Sequential Backskip commands can be executed. Backskip occurs across track boundaries, i.e. current track and sector 0701 becomes 0626 for single density 128 byte sectors. Primary status is returned. Backskip may not span from track 1 to track 0 except in 3740 format.

Function: INIT
Keybd Entry: ESC CTL Z N where N is the desired mode, format and density.
ASCII Char Seq: ESC SUB N
Description: Provides the capability to select recording density, sector size and mode. No other parameters are affected. If no INIT command occurs after RESET, the FORMAT L and COPY L default to the strapped density and format. Primary status is returned. No other system states, values, pointers, track/sector positioning, drive selection are affected.

The third command byte determines the mode, recording density, and sector size. Any other characters than specified below are ignored.

ASCII CODE		RECORDING DENSITY	SECTOR SIZE
SECTOR MODE	X-ON/X-OFF		
@ *	0	SINGLE	128
A	1	SINGLE	256
B	2	DOUBLE	128
C	3	DOUBLE	256

Function: PAUSE
Keybd Entry: Break
ASCII Char Seq: Receive Data Spacing (0's)
Description: Interrupt FT0200I transmit while the receive data line is spacing (0's).*

*NOTE: May cause receiver overruns in the primary and secondary status response which may be ignored.

TABLE 3-B. SECTOR MODE COMMANDS

Function:	READ SECTOR
Keybd Entry:	CTL Q
ASCII Char Seq:	DC1
Description:	Reads the current sector, transmits the data to host, steps to next sequential sector, and returns Primary status to host. If parity is selected as odd or even, the appropriate parity bit is generated.

Function:	WRITE SECTOR
Keybd Entry:	CTL R X_1 X_2 ... X_n where X_1 X_2 ... X_n is data to be written and n = sector length.
ASCII Char Seq:	DC2
Description:	Writes the current sector with the received data, steps to next sequential sector, and returns Primary status to host. Eight bit binary data is always written on diskette; the parity bit is recorded.

Function:	WRITE SECTOR OF DELETED DATA
Keybd Entry:	CTL X CTL R X_1 X_2 ... X_n where X_1 X_2 ...X_n is data to be written and n = sector length.
ASCII Char Seq:	CAN DC2
Description:	Writes the current sector with a deleted data address mark and the received data, steps to the next sequential sector and returns Primary status to the host. Eight bit binary data is always written on the diskette; the parity bit is recorded.

Function:	BLOCK READ
Keybd Entry:	ESC CTL Q MSB LSB
ASCII Char Seq:	ESC DC1 MSB LSB
Description:	Reads data from successive sectors starting at the beginning of the current sector up to the byte count and returns Primary status to host, where MSB and LSB are the most and least significant bytes of the byte count. If an error condition is encountered, Primary status is returned upon detection of the error.

Function:	BLOCK WRITE
Keybd Entry:	ESC CTL R MSB LSB X_1 X_2 ... X_n where X_1 X_2 ... X_n is data to be written and n = value of MSB LSB.
ASCII Char Seq:	ESC DC2 MSB LSB
Description:	Writes data to successive sectors starting at the beginning of the current sector up to the byte count and returns Primary status to Host. If a buffer overrun is pending, two indications occur to temporarily inhibit host transmission. (Reference section IV-B.) If an error condition is encountered, Primary status is returned upon detection of the error.

E. PRIMARY STATUS

In response to a CTL N (ASCII character SO) command or after completion of most operations, the FT0200I returns a status character along with five other disk information characters. (See Table 3-A for the meaning of these five characters.) The significance of this status character is shown in Table 4.

TABLE 4. PRIMARY STATUS CHARACTER.

DISPLAYED CHARACTER	DRIVE STATUS	DISK STATUS
SP(1) ! " #	Drive 0 & Drive 1 NOT ready Drive 0 ready Drive 1 ready Drive 0 and Drive 1 ready	Normal status; no errors NOTE A Ready Error status is reported if the selected drive is not ready during a disk operation(2).
p q r s	Drive 0 & Drive 1 NOT ready Drive 0 ready Drive 1 ready Drive 0 & Drive 1 ready	Communications (Receiver Overrun) Error
x y z {	Drive 0 & Drive 1 NOT ready Drive 0 ready Drive 1 ready Drive 0 & Drive 1 ready	Communications (Frame/ Parity) Error
d e f g	Drive 0 & Drive 1 NOT ready Drive 0 ready Drive 1 ready Drive 0 & Drive 1 ready	Ready Error, Command or Disk Drive Error, Reference Secondary Status.
m n o	Drive 0 ready Drive 1 ready Drive 0 & Drive 1 ready	Disk Operation Error. Reference Secondary Status.
` a b c	Drive 0 & Drive 1 NOT ready Drive 1 NOT ready Drive 0 NOT ready Drive 0 & Drive 1 ready	FT0200I Command Error

NOTES:

(1) SP - SPACE advances carriage or cursor one position.
(2) Disk operations which cause command or Drive Error when the selected drive is not ready are Read, Write, Seek, Format, or Copy. Drive Select, Primary Status Request, Secondary Status Request, INIT, Set Read Stop Code and Backskip (within a track) commands are not disk operations. A backskip command which crosses track boundaries (a Backstep occurs) is a disk operation (implied Seek).

3000406/A

Primary Status Byte Format

BIT POSITION	0	1	2	3	4	5	6	7	
	PARITY	ERROR	PRIM.	COMM ERROR	O/C or R/F	DISK ERROR	DR1 RDY	DR0 RDY	DIRECTION OF SERIAL TRANSMISSION------>

Bit 0 PARITY: Varies according to parity selected for host terminal (parity=0 if no parity is selected).
Bit 1 ERROR: =1 if any FT0200I error occurs; =0 if no error present
Bit 2 PRIMARY: Always =1.
Bit 3 COMM ERROR: If =1 when bit 1=1, a communication error exists.
Bit 4 O/C or R/F: If =0 when bit 3=1, a receiver overrun error exists. If =1 when bit 3=1, a frame or parity error exists; If =0 when bit 5=1, a Command or Drive error exists; If =1 when bit 5=1, a Disk operation error exists;
Bit 5 DISK ERROR: If =1 when bit 1=1, a disk error exists.
Bit 6 DR1 RDY: =1 if disk drive 1 is in the system and ready to operate.
Bit 7 DR0 RDY: =1 if disk drive 0 is in the system and ready to operate.

Secondary Status Response

FIR where:

F Mode, format and density.

I Secondary Status Byte.

R Optional C/R turnaround character.

Mode, Format and Density character

Character 3 in the INIT command.
Character 1 in the Secondary Status Response.

ASCII CODE*		RECORDING	SECTOR
SECTOR MODE	X-ON/X-OFF	DENSITY	SIZE
@	0	SINGLE	128
A	1	SINGLE	256
B	2	DOUBLE	128
C	3	DOUBLE	256
*Any other characters received are ignored.			

```
(*ÅL+,0+,A2*)
(*PAGNT702*)
PROGRAM FLOPPY(INPUT,OUTPUT);
CONST  XA=16ÄFA;  XAM=16ÄFB;
       RA=16ÄF9;  RAM=16ÄF7;
       REC=16Ä80; ENABLESEND=2; ENABLEREC=1;
       DISABLE=16ÄFF;           ENABLE=16ÄE3;
       TRANSMITT=16ÄE1;
       VECADR=16Ä70;

-INC PAGNT514                              (*KERNEL TYPE DEKL.*)
-INC PAGNT511                              (*MESSAGE DEKLARATION*)
     BUFFERT=PACKED ARRAY(.0..255.) OF CHAR;
     TECKEN=PACKED ARRAY(.1..3.) OF CHAR;

VAR  KERNEL:KERNELPTR;
     DISCREQUEST:MAILBOX;
     REQUEST:MESSREF;
     B:INTEGER4;
     BUFF(.ORIGIN 16ÄFF0900.):BUFFERT;
     C:TECKEN;
     KVITTERA(.ORIGIN 16ÄBE09FD.):CHAR;
     I:INTEGER2;
     CH,SLASK:CHAR;
     WRONG:BOOLEAN;
(*PROCEDURES*)
PROCEDURE EI;PASCAL;
PROCEDURE DI;PASCAL;
PROCEDURE WAITIO(VEKTORADRESS:INTEGER2);PASCAL;
PROCEDURE INITUART;PASCAL;
PROCEDURE CONVERT(T,K:INTEGER2; VAR SVIT:TECKEN);PASCAL;
PROCEDURE MOVEBYTE(FROM,DEST:INTEGER4; ANTAL:INTEGER2);PASCAL;
PROCEDURE SENDMESSAGE(BOX:MAILBOX; VAR MESS:MESSREF);PASCAL;
PROCEDURE RECEIVEMESSAGE(BOX:MAILBOX; VAR MESS:MESSREF);PASCAL;
PROCEDURE WAITTIME(TIME:INTEGER4);PASCAL;

PROCEDURE SETINT(MODE:INTEGER2);
BEGIN
   BUFF(.DISABLE.):=CHR(16Ä42);              (*SYN*)
   BUFF(.ENABLE.):=CHR(MODE);
   WAITIO(VECADR);
   BUFF(.ENABLE.):=CHR(0);
   SLASK:=KVITTERA;
END;

PROCEDURE CHECKSTATUS(POS:INTEGER2);
VAR I:INTEGER2;
BEGIN
   WRITE('  STATUS:');
   FOR I:=POS TO POS+5 DO WRITE(BUFF(.I.));
   WRITELN;
END;

PROCEDURE SENDRECEIVE(SX,RX:INTEGER2);
(*SENDS SX BYTES SX<=270 AND RECEIVES RX BYTES RX<=96*)
BEGIN
   DI;
   BUFF(.XA.):=CHR(0);
   BUFF(.XAM.):=CHR(SX-1);
   BUFF(.RA.):=CHR(REC);
   BUFF(.RAM.):=CHR(REC+RX-1);
   SETINT(ENABLESEND);
   SETINT(ENABLEREC);
END;

PROCEDURE INITCOM;
BEGIN
   INITUART;
   BUFF(.0.):=CHR(16Ä0D);                  (*C/R*)
   BUFF(.1.):=CHR(65);                     (*A => AUTOBAUD*)
   SENDRECEIVE(2,1);
   EI;
   WRITELN(' ON LINE',BUFF(REC));
END;
```

```
PROCEDURE INITDISC;
BEGIN
   BUFF(.0.):=CHR(27);                          (*ESC*)
   BUFF(.1.):=CHR(26);                          (*CTRL Z*)
   BUFF(.2.):=REQUESTÖ.T;
   SENDRECEIVE(3,6);
   CHECKSTATUS(REC);
END;

PROCEDURE CHANGE(VAR TAL:INTEGER2);
(*OMVANDLAR INTEGER2 TILL ASCH II OCH GER ANTALET
  SIGNIFIKANTA SIFFROR I TAL*)
LABEL 999;
VAR REST,K,I:INTEGER2;
BEGIN FOR K:=1 TO 3 DO
 BEGIN
    REST:= TAL MOD 10;
    REST:=REST+48;
    CONVERT(REST,K,C);
    TAL:=TAL DIV 10;
    IF TAL=0 THEN GOTO 999;
 END;
999: FOR I:=K+1 TO 3 DO C(.I.):=CHR(48);   (*NOLLUTFYLLNAD*)
 TAL:=K;
END;

PROCEDURE LOCATE;
BEGIN WITH REQUESTÖ DO
   BEGIN
      BUFF(.0.):=CHR(15);                       (*CTRL O*)
      CHANGE(DRIVE);
      BUFF(.1.):=C(.1.);                        (*3:DE DIGIT*)
      SENDRECEIVE(2,6);                         (*SELECT DRIVE*)
      EI;
      CHECKSTATUS(REC);
      BUFF(.0.):=CHR(22);                       (*SYN*)
      CHANGE(TRACK);
      BUFF(.1.):=C(.2.);                        (*2:DE DIGIT*)
      BUFF(.2.):=C(.1.);                        (*3:DE DIGIT*)
      CHANGE(SEKTOR);
      BUFF(.3.):=C(.2.);                        (*2:DE DIGIT*)
      BUFF(.4.):=C(.1.);                        (*3:DE DIGIT*)
      SENDRECEIVE(5,6);                         (*SELECT TRACK AND SECTOR*)
      EI;
      CHECKSTATUS(REC);
   END;
END;
```

```
PROCEDURE READSEKTOR;
VAR I:INTEGER2;
    DELTA1,DELTA2:INTEGER4;
BEGIN WITH REQUESTÖ DO
   BEGIN
      BUFF(.TRANSMITT.):=CHR(17);          (*CTRL Q*)
      BUFF(.RA.):=CHR(REC);
      BUFF(.RAM.):=CHR(REC+63);            (*64-1*)
      BUFF(.DISABLE.):=CHR(16Ä42);
      BUFF(.ENABLE.):=CHR(1);
      WAITIO(VECADR);
      WAITIO(VECADR);                      (*DUBBELT AVBROTT*)
      BUFF(.ENABLE.):=CHR(0);
      DELTA1:=REC;
      MOVEBYTE(B+DELTA1,ADR,64);           (*LÄSER BÖRJAN*)
      SLASK:=KVITTERA;
      EI;
      DELTA1:=REC+64;
      DELTA2:=64;
      MOVEBYTE(B+DELTA1,ADR+DELTA2,32);    (*LÄSER MITTEN*)
      DELTA1:=REC;
      DELTA2:=96;
      MOVEBYTE(B+DELTA1,ADR+DELTA2,32);    (*LÄSER SLUTET*)
      CHECKSTATUS(REC+32);
   END;
END;

PROCEDURE WRITESEKTOR;
VAR I:INTEGER2;
    DELTA:INTEGER4;
BEGIN
   BUFF(.0.):=CHR(18);                     (*CTRL R*)
   DELTA:=1;
   MOVEBYTE(REQUESTÖ.ADR,B+DELTA,128);
   SENDRECEIVE(129,6);
   CHECKSTATUS(REC);
   EI;
END;

(*H U V U D P R O G R A M*)
BEGIN
   B:=16ÄFF0900;                           (*BUFFERT ADRESSEN*)
   INITCOM;
   WRONG:=FALSE;
   WHILE TRUE DO
   BEGIN
      RECEIVEMESSAGE(DISCREQUEST,REQUEST);
      WITH REQUESTÖ DO
      BEGIN
         CASE COMMAND OF
            INIT:  BEGIN INITDISC; COMMAND:=XINIT; END;
            FIND:  BEGIN LOCATE; COMMAND:=XFIND; END;
            INP:   BEGIN READSEKTOR; COMMAND:=XINP; END;
            OUTP:  BEGIN WRITESEKTOR; COMMAND:=XOUTP; END;
         END;   (*CASE COMMAND*)
         IF WRONG THEN BEGIN COMMAND:=ERROR; WRONG:=FALSE; END;
         SENDMESSAGE(ACKNOWLEDGE,REQUEST);
      END;
   END;
END.
```

4.1 CONVERSION TABLES

4.1.1 HEX-ASCII

B7 B6 B5 BITS / B4 B3 B2 B1	0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
	CONTROL		NUMBERS SYMBOLS		UPPER CASE		LOWER CASE	
0 0 0 0	0 NUL 0	20 DLE 16	40 Space SP 20 32	60 0 30 48	100 @ 40 64	120 P 50 80	140 ` 60 96	160 p 70 112
0 0 0 1	1 CNTR SOH A 1 1	21 CNTR DC1 Q 11 17	41 ! 21 33	61 1 31 49	101 A 41 65	121 Q 51 81	141 a 61 97	161 q 71 113
0 0 1 0	2 CNTR STX B 2 2	22 DC2 12 18	42 " 22 34	62 2 32 50	102 B 42 66	122 R 52 82	142 b 62 98	162 r 72 114
0 0 1 1	3 CNTR ETX C 3 3	23 DC3 13 19	43 # 23 35	63 3 33 51	103 C 43 67	123 S 53 83	143 c 63 99	163 s 73 115
0 1 0 0	4 EOT 4 4	24 DC4 14 20	44 $ 24 36	64 4 34 52	104 D 44 68	124 T 54 84	144 d 64 100	164 t 74 116
0 1 0 1	5 ENQ 5 5	25 NAK 15 21	45 % 25 37	65 5 35 53	105 E 45 69	125 U 55 85	145 e 65 101	165 u 75 117
0 1 1 0	6 ACK 6 6	26 SYN 16 22	46 & 26 38	66 6 36 54	106 F 46 70	126 V 56 86	146 f 66 102	166 v 76 118
0 1 1 1	7 BEL 7 7	27 ETB 17 23	47 ' 27 39	67 7 37 55	107 G 47 71	127 W 57 87	147 g 67 103	167 w 77 119
1 0 0 0	10 BS 8 8	30 CAN 18 24	50 (28 40	70 8 38 56	110 H 48 72	130 X 58 88	150 h 68 104	170 x 78 120
1 0 0 1	11 HT 9 9	31 EM 19 25	51) 29 41	71 9 39 57	111 I 49 73	131 Y 59 89	151 i 69 105	171 y 79 121
1 0 1 0	12 LF A 10	32 SUB 1A 26	52 * 2A 42	72 : 3A 58	112 J 4A 74	132 Z 5A 90	152 j 6A 106	172 z 7A 122
1 0 1 1	13 VT B 11	33 ESC 1B 27	53 + 2B 43	73 ; 3B 59	113 K 4B 75	133 [5B 91	153 k 6B 107	173 { 7B 123
1 1 0 0	14 FF C 12	34 FS 1C 28	54 , 2C 44	74 < 3C 60	114 L 4C 76	134 \ 5C 92	154 l 6C 108	174 | 7C 124
1 1 0 1	15 CR D 13	35 GS 1D 29	55 - 2D 45	75 = 3D 61	115 M 4D 77	135] 5D 93	155 m 6D 109	175 } 7D 125
1 1 1 0	16 CNTR SO N 14 E	36 RS 1E 30	56 . 2E 46	76 > 3E 62	116 N 4E 78	136 ^ 5E 94	156 n 6E 110	176 ~ 7E 126
1 1 1 1	17 CNTR SI O 15 F	37 US 1F 31	57 / 2F 47	77 ? 3F 63	117 O 4F 79	137 — 5F 95	157 o 6F 111	177 RUBOUT (DEL) 7F 127

BS = BACK SPACE

KEY octal 25 / hex 15 21 — NAK — ASCII character / decimal

4.0 REFERENCE