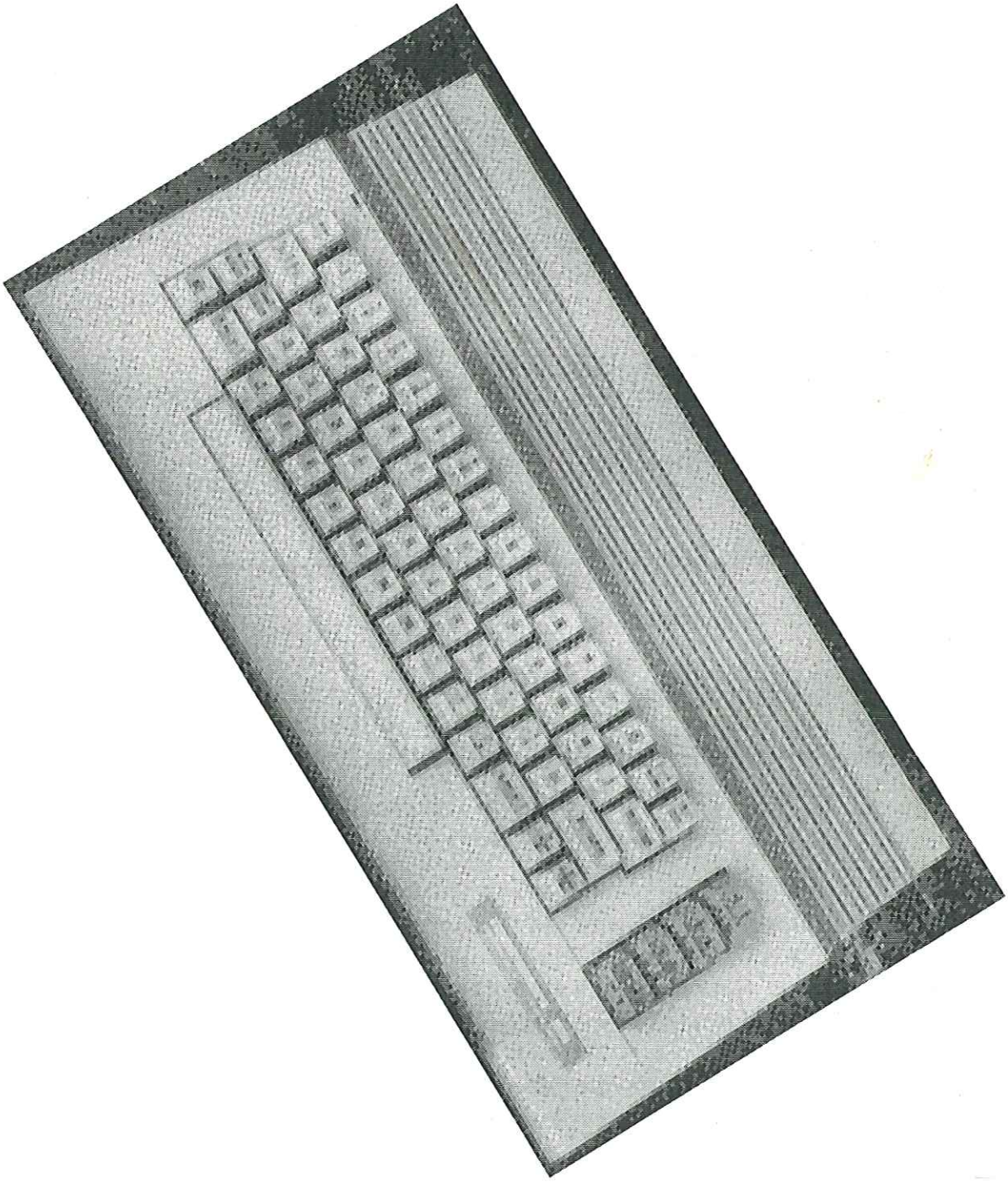
RELIK

Tidningen för 8-bitars datorer och dess användare

Nummer 2, Januari 1997 Pris: 20 SEK



INNEHÅLL

ARTIKLAR

Relik behöver fortfarande fler skribenter. Om Du har lust att skriva artiklar om Åtta-Bitars datorer och produkter kring dem, hör av dig till oss. Vi söker speciellt skribenter för andra datormärken än Commodore 64 och 128.

Relik övertar automatiskt copyright-rätten för alla artiklar som skickas in till tidningen. Att skicka in en artikel är detsamma som att godta detta. Som ersättning för en införd artikel får författaren ett gratisexemplar av det nummer som artikeln införs i.

RELIK

Relik är en tidning om åtta-bitars datorer, skriven av användare för användare. Arbetet är helt idéellt, och eventuellt överskott från försäljningen går till förbättringar av tidningen.

Redaktör: Jens Örtenholm
Ansvarig utgivare: Jens Örtenholm

Adress:
Relik
c/o Jens Örtenholm
Råbystigen 48
197 31 Bro

Telefon red: 070-5927747

WWW:
http://mercury.komvux.uppbro.se/~jens/relik.html

E-mail: jens.ortenholm@pc-programs.se

Postgiro: 770118-0239

Tidningen kostar 20 SEK inklusive porto inom Sverige. Betala in pengarna på postgirokontot och ange tydligt namn, adress och vilket nummer av tidningen du vill beställa.

PD-disketter kostar 30:- styck, eller 30:- för den första disketten och 10:- / styck för de övriga vid beställningar över 2 st. Betala in pengarna på postgirokontot och ange tydligt namn, adress och vilken/vilka disketter du vill ha.

Medverkande i detta nummer: Jens Örtenholm, Roland Vallgren, Anders Reuterswärd, Erik Paulsson och Alexander Hole.

Vad har hänt sedan senast?

Hej där! Så var vi då tillbaks igen, efter månader av försening. Nästa nummer var utlovat i November - så blev det icke. Det borde dessutom ha kommit ett nummer i December - inte heller det blev av stapeln. Nu är det i mitten av Januari, och saker och ting har äntligen börjat flyta igen.

Men vad är det då som har hänt, kan man undra?

Bara själva tanken på att producera en tidning för åtta-bitars datorer verkar föra otur med sig. Gamla goda Åtta Bitar bjöd på försening efter försening, och det var en av de saker som jag ville undvika med Relik, men redan vid andra numret så körde saker och ting ihop sig.

Vårt största problem har varit frånvaron av skribenter. Att sätta ihop en tidning helt på egen hand är inte bara jobbigt - det känns också fel på något sätt. Idén med en tidning (tycker jag) är att ett flertal artikelförfattare delar på ett gemensamt utrymme. Vad som framförallt har saknats är mer allmänna artiklar, någonting som inte riktigt är min starka sida (jag är mest intresserad av programmering). Till och med detta nummer innehåller flertalet tekniska artiklar, men det kommer vi att ändra på till nummer 3.

Denna (i mina ögon) nästan oändliga tidsrymd som förflutit sedan nummer 1 har lyckligtvis inte varit en overksam period. Vi har fått fler skribenter, och vi har spenderat massor av tid med samordning och planering. Dessa nya förmågor kommer att presenteras allt eftersom deras alster når in i tidningen. Som ni ser så har jag också jobbat lite granna på tidningens layout, frågan är bara om det är till det bättre eller till det sämre, men det var både roligt och utvecklande för mig så jag hoppas att ni ska trivas med det.

Ett annat problem som dök upp från ingenstans var att Nethosting, den web-service jag använt för att lägga upp mina (och tidningens) hemsidor på bestämde sig för att lägga ned verksamheten. Därför finns det i skrivande stund ingen web-sida för tidningen på nätet, men inom ett par dagar kommer jag att lägga upp sidorna (uppdaterade, till och med) tillfälligt på min skolas server, URL dit är http://mercury.komvux.uppbro.se/~jens/relik.html men just för tillfället har Swipnet strulat till den maskinen i sin nameserver, så fungerar det inte så beror det på det)..

Mitt i den här smeten så passade givetvis massor av människor på att beställa nummer 1, nummer 2 och även prenumerationer. Dessa har, till min stora skam, ännu inte skickats ut, men kommer skickas inom ett par dagar, och som plåster på såren för denna långa väntan så kommer alla dessa beställare få ett nummer gratis.

Så, nu hoppas jag att vi äntligen ska kunna göra oss fria från förseningarnas och försoffningarnas bojor! Med en avsevärt större redaktion så ser jag till och med möjligheter till att detta ska bli sanning, och jag vågar nästan lova att nummer 3 kommer någon gång i Februari (förmodligen i slutet på månaden).

Så lev väl, må bra och tappa inte hoppet!

Jens Örtenholm
Redaktör

ROS-64

Roland Vallgren

Det här är första delen i något som är tänkt att bli en artikelserie och ett designprojekt. Avsikten med designprojektet är att konstruera ett realtidsoperativsystem. Avsikten med artikelserien är mer diffus, men jag tycker att det kan vara intressant för fler än mig. Dessutom kanske jag kan se till att få lite mera gjort om jag måste få med en ny del till nästa nummer av tidningen.

I den här första delen handlar litet om bakgrunden till projektet. Härifrån kommer också en grundläggande funktion som kommer att användas rätt mycket.

För ett par år sedan knåpade jag ihop en rasteravbrottsrutin enligt en beskrivning i Datormagazin. Det var den rutin som öppnar ramen nertill och uppptill jag använde. Till detta behövdes en rutin som kunde arbeta med presentation i ramen. Den presentationen startades av avbrottsrutinen men fick inte avbrytas och startas igen av nya avbrott. Den rasteravbrottsrutin jag gjorde hade mycket snäva krav på svarstiden för ett avbrott så det gick inte att låta avbrotten vara avstängda för länge. I stället blockerades presentationsrutinen med en "semafor".

Roland Vallgren kan nås via email: roland.vallgren @mn.medstroms.se eller via redaktionen.

När den konstruktionen blev klar och det hade smält ett tag så funderade jag på om det inte skulle vara möjligt att konstruera ett realtidsoperativsystem till C64:an. Jag vet ännu inte om det är möjligt att göra, så vi får se vad resultatet blir. En del konstruktioner som behövs finns redan medan andra återstår att funderas ut. Enbart för nöjet att få tänka och programmera lite så gör jag det även om det skulle visa sig att det inte går.

Till att börja med blir det en del basfunktioner som skall gå att använda på vilken 6502 baserad dator som helst. När det blir dags att koda så blir det nog i C-64 kod. Senare borde det gå att göra en C-128 version som skulle kunna utnyttja en del av C-128:ans utökade hårdvara. MMU:n är det som intresserar mig i första hand. Eventuellt kan det bli tal om att göra en version till Z80:n i C-128:an, men dit är det rätt långt.

Nå, skall det inte börja någon gång? Jodå, TILL ATTACK!

"Semafor", visst stod det så? - undrar en läsare. Vad är det?

Så vitt jag vet så är en semafor en signalapparat som används för att dirigera tåg på järnvägen. Jag vet inte om semaforer används nuförtiden, sånt sköts väl elektroniskt.

Inom järnvägen använder man semaforer för att signalera till tågen när dom får köra eller inte. Den signalering som semaforen i operativsystemet skall efterlikna är att se till att inte mer än ett tåg är inne på samma spår samtidigt. När ett tåg lämnar ett spår så kan ett annat släppas in. Detta för att slippa kollisioner mellan tågen. I ett operativsystem kan det bli konflikt om flera program kör samma kod samtidigt eller förändrar samma data samtidigt. Detta kan man hindra med en semafor. Om ett program är inne i den kritiska koden så får andra program snällt vänta tills det blir ledigt. Ett annat exempel är att se till att inte flera program hanterar samma hårdvara samtidigt.

I C-128 BASIC finns ett exempel på kod som borde vara gjort med en semafor istället för en flagga.

Avbrottsrutinen i C-128 BASIC tar hand om ljud och de extra grafikfunktioner som BASIC:en tillhandahåller. Då det kan ta ganska lång tid att utföra alla dessa funktioner så finns det en viss risk att det hinner komma ett nytt avbrott. I första hand skulle man kunna tänka sig att stänga av avbrotten i processorn. I det här fallet går det inte, eftersom det finns andra viktiga avbrott som måste fungera.

Så här ser den intressanta delen ut: (Kopierat från boken ABACUS BASIC 7.0 INTERNALS)

```
; BASIC IRQ ROUTINE
A84D:  LDA $12FD     ;If
```

```
the BASIC IRQ
A850:  BEQ $A853     ;Is
already in progress
A852:  RTS
;Then exit the routine
A853:  INC $12FD
;Block the BASIC IRQ
A856:
..
Process the BASIC IRQ
..
; Exit BASIC IRQ
A9F0:  DEC $12FD
;Clear the IRQ in
progress flag
A9F3:  RTS           ;And
exit
```

På adress $12FD finns en flagga som används för att se till att enbart ett avbrott körs samtidigt. Om man tittar efter noga så ser man att hanteringen av flaggan har ett fel.

Vad händer om ett nytt avbrott inträffar efter instruktionen "LDA $12FD" men innan instruktionen "INC $12FD"?

Det nya avbrottet kommer inte att veta att det förra inte har hunnit sätta flaggan ännu. Man kan tycka att det inte tar särskilt lång tid innan flaggan sätts så det skall väl inte vara något problem. Här kanske det inte är det, men det kan finnas andra fall när det kan vara fatalt.

Så här kan koden göras om för att få den säkrare:

```
; IMPROVED BASIC IRQ
ROUTINE
A84D:  INC $12FD
;Test and set IRQ in
progress flag
A850:  BNE $A856
;Jump if not in progress
A852:  JMP $A9F0
;Exit if IRQ already in
progress
A855:  NOP           ;Not
needed
A856:
..
Process the BASIC IRQ
..
; Exit BASIC IRQ
A9F0:  DEC $12FD
;Clear the IRQ in
progress flag
A9F3:  RTS           ;And
exit
```

Genom att göra "INC $12FD" direkt så uppnås två resultat med en enda instruktion. Flaggan förändras, samtidigt som ett resultat kan avläsas i processorns statusflaggor. Om flaggan är satt till #$FF innan INC så blir den noll ($00) efter INC instruktionen. I alla andra fall blir flaggan inte noll av INC instruktionen och hoppet till $A9F0 görs. Rutinen avslutas alltid med att instruktionen "DEC $12FD" utförs. Oberoende av vilken väg programmet går så måste flaggan vara oförändrad när rutinen körts.

Om man studerar det här noga kan man dock finna att det kan bli problem om det kommer 256 avbrott innan vi har hunnit återställa flaggan.

För att slippa detta så kan ytterligare en förbättring göras:

```
; MORE IMPROVED BASIC
IRQ ROUTINE
A84D:  LSR $12FD
;Test and reset IRQ not
in progress flag
A850:  BCS $A856
;Jump if not in progress
A852:  RTS
;Exit the IRQ if already
in progress
A853:  NOP           ;Not
needed
A854:  NOP           ;Not
needed
A855:  NOP           ;Not
needed
A856:
..
Process the BASIC IRQ
..
; Exit BASIC IRQ
A9F0:  INC $12FD     ;Set
the IRQ not in progress
flag
A9F3:  RTS           ;And
exit
```

Här skall flaggan vara initierad till ett ($01). Då finns det en enda bit som är satt till ett i $12FD. När instruktionen "LSR $12FD" exekveras så kommer ettan att flyttas in i carryflaggan. Om carry inte blir ett så betyder det att avbrottsrutinen redan pågår. Om det inte fanns någon etta i $12FD så ändras ingenting och därför behövs ingen återställning.

Efter att avbrottsrutinen är klar så måste flaggan återställas. Här görs det med INC $12FD eftersom en ensam etta markerar att flaggan är ledig.

Med de här modifieringarna så har flaggan i $12FD blivit en semafor. Med en sån här semafor kan tillgången till en resurs styras. I ett generellt fall kan en semafor användas för fler än en resurs. Här betyder resurs något som flera program kan ha behov av att använda

Så här kan en generell semafor med fler än en resurs se ut:

```
; GENERIC SEMAPHORE

; Claim one resource
CLAIM  LSR SEMAPHORE
       BCC DENIED

;Access granted, execute
the protected code
GRANTED ..

; Leaving protected
code, release resource
RELEASE SEC
        ROL SEMAPHORE
        RTS

; Access denied, take
required action
DENIED ..
```

I byten SEMAPHORE kan antalet resurser väljas genom att sätta det antal bitar man önskar. Den klarar alltså mellan en och åtta resurser. Vid behov av fler resurser kan en semafor med INC / DEC användas istället.

Jag har försökt använda benämningen semafor här. När man tittar på olika processorer så brukar hanteringen av semaforer skötas med så kallade "test and set"

instruktioner.

Moderna processorer kan ofta kopplas in i flerprocessorsystem. I sådana system är behovet av väldefinierade "test and set" instruktioner av stor betydelse för att styra kommunikation mellan olika program. När en sådan instruktion utförs så meddelar processorn detta i hårdvara så att andra processorer vet vad som är på gång.

En "test and set" instruktion skulle kunna skrivas så här i 6502 assembler:

```
; TEST AND SET

        SEI
        LDA SEMAPHORE
        BEQ BUSY
        DEC SEMAPHORE
        CLC
        BCC DONE
BUSY    SEC
DONE    CLI
```

Så mycket kod blir det av något som bara är en instruktion.

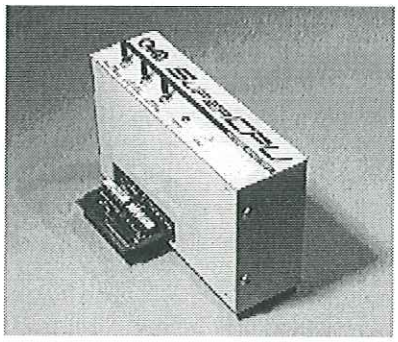
Nästa del kommer att behandla:

*** Något om oberoende processer**
*** Kommunikation mellan processer**
*** Projekt ROS-64,**
RealtidsOperativsystem-64

BROTKASTEN CD - 278 MB PROGRAM!

Lider du brist på bra shareware-program till din 64:a eller 128:a? Har du tillgång till en PC med CD-läsare? Då är Brotkasten CD troligen något för dig. Här finns otroliga 278 MB program packade på en CD-skiva, varav de flesta till 64:an. Alltsammans ligger i så kallat D64-format, det vill säga diskettsidor som är läsbara med 64-emulatorn C64GS, eller konverterbara till 64-format. Enklaste sättet att använda sig av programmen för vanligt folk, som nu inte kör 64Net eller något annat avancerat system, är att kopiera önskade filer till disketter och sedan konvertera filerna till 64:an med hjälp av Big Blue Reader (eller något shareware-program, varav ett finns med på CD:n). Sedan använder man 64-programmet D64-xtra, som också finns med på CD:n, för att omvandla D64-filerna till användbart format. En kanske lite omständlig procedur, men inte svårt. Och väl värt det, om man vill komma åt innehållet på denna guldgruva till CD. Och vad består detta innehåll av? Jo, massor av program från varjehanda källor, till exempel 64er. Här hittar man också en hel del CP/M-program. Dessutom finns till exempel 30 Mb GEOS-program, varav åtskilliga inte ens finns i världens största GEOS-bibliotek... (i alla fall inte ännu..). Givetvis finns de två stora 64-emulatorerna för PC, samt användbara DOS- och Windows-program för att hantera innehållet på skivan. För de med mera speciella intressen finns även PC- och Amiga-emulatorer för andra format, som klassikerna ZX81, Spectrum, Apple II, Texas och Atari, samt en del programvara för dessa, men detta ligger utanför intresset hos denna tidning. Många av programmen är tyska och skivan är tysk, vilket innebär att en hel del av instruktioner och dylikt är på detta lands språk. Det går att använda det hela utan att kunna tyska, med hjälp av erfarenhet och sunt förnuft, men givetvis underlättar det med lite språkkunskaper. Det här är ett enkelt och billigt sätt att komma över massor med program. Testexemplar från: DinoSoft. Pris: 350:-

Anders Reuterswärd



Bilden föreställer ett exemplar av CMD:s SuperCPU 64

SuperCPU

För ett par år sedan så satt jag och dagdrömde. Drömmen bestod av att min C64:a som jag satt och knappade på vid tillfället skulle innehålla en snabbare processor och mer internminne. Det var avsaknaden av dessa två detaljer som gjorde att jag gled mer och mer över till PC-maskinerna, eftersom mina program då gjorde sitt jobb på ett par sekunder istället för minuter på C64:an. Nu inser jag att min dröm nästan har blivit sann. CMD har utvecklat SuperCPU:n.

Först av allt så vill jag påpeka att jag inte haft möjlighet att testa SuperCPU:n, denna artikel baserar sig alltså på information som finns tillgänglig från CMD själva. Jag tyckte ändå det kunde vara värt att skriva en del om den, framför allt för de som inte har tillgång till internet och själva kan läsa den information som finns på CMD:s hemsidor.

SuperCPU:n pluggas in i cartridgeporten på C64:an eller C128:an. De viktigaste komponenterna är först och främst självklart CPU:n, en W65C816S processor på 20 MHz, men den innehåller även 128 Kb statiskt DRAM (av samma typ som används som cache på nyare PC-moderkort), 128 Kb ROM samt en krets kallad CPLD som sköter kommunikationen mellan datorn och den nya processorn.

Att den pluggas in i cartridgeporten betyder inte att den porten blir upptagen, SuperCPU:n har nämligen en så kallad pass-thru port på baksidan där man kan stoppa in den cartridge man önskar. Alla cartridgar fungerar dock inte med SuperCPU:n, vilket heller inte alla program gör. Man kan fortfarande använda sådana cartridgar i pass-thru porten genom att manuellt ställa om SuperCPU:n till 1 MHz läge (alltså stänga av turboeffekten), vilket också får gamla program att fungera. Vissa tillägg som t.ex. REU, RAMLink, Swiftlink, GeoRAM med flera fungerar också i turboläge.

SuperCPU:n har tre omkopplare för på/av, JiffyDOS på/av och hastighetsinställning (1/20 MHz, med andra ord en switch för ifall 20 MHz processorn ska användas eller inte). En resetknapp finns också, samt en liten lysdiod som visar ifall turbo är på eller avslaget. JiffyDOS switchen finns eftersom JiffyDOS Kernal ROM är inbyggt i SuperCPU:n, vilket ger möjlighet till snabb seriell kommunikation med JiffyDOS-drivar.

Det finns även ytterligare en port på SuperCPU:n, en expansionsport som i framtiden ska kunna användas till antingen ett C-128 kort (jag är inte riktigt klar över vad det är, men jag antar att det har någonting med den kommande C-128 versionen av SuperCPU:n att göra) eller ett RAM-expansionskort. Båda dessa kort ska ha möjlighet till RAM-expansion mellan 1 och 16 MB DRAM i SIMM-format.

Det främsta användningsområdet för SuperCPU:n tror jag kommer bli tillsammans med GEOS. Enligt CMD så är den nämligen i stort sett 100% kompatibel med GEOS, och de som använder GEOS kan ju tänka sig vilket fartlyft det kan bli... GEOS är ju ett system som hårdvaran i C64:an aldrig riktigt räckt till för, det har ju minst sagt gått lite snigligt ibland, men med SuperCPU:n så kan jag tänka mig att mycket arbete som gjorts i skolor eller på arbeten i ordbehandlingsväg etc istället kommer att göras hemma på C64:an i GEOS.

Som tidigare nämnts så är den färdiga versionen av SuperCPU:n enbart till för en C64:a alternativt en C128:a i 64-läge. En C128 version är dock utannonserad, och enligt CMD:s hemsida så ska den komma i sista kvartalet av 1996, där vi i och för sig befinner oss nu, men ingen information om den finns tillgänglig på CMD:s hemsidor.

Efter att ha läst igenom CMD:s information och skrivit denna artikel, så kan jag bara sluta mig till att SuperCPU:n är någonting som i alla fall jag själv kommer sätta bland de högre posterna på önskelistan, även om jag vet att jag inte kommer få någon. Jag kan tänka mig att de flesta C64-entusiaster också skulle vilja ha en, och många kommer säkerligen att köpa en. För vem har inte önskat att diverse program, och kanske framförallt GEOS, skulle gå sisådär 19 MHz snabbare då och då?

Enligt CMD så ligger priset för SuperCPU:n på 199 dollar, och C128-versionen kommer enligt samma källa kosta ungefär 299 dollar.

Jens Örtenholm

REDAKTIONEN

VILL PASSA PÅ ATT ÖNSKA ER ALLA EN GOD JUL OCH ETT GOTT NYTT ÅR I EFTERSKOTT

B128

Erik Paulsson

(Denna artikel var ursprungligen menad för Åtta Bitar)

Vadå B128? C128 heter det väl? Många känner nog inte till att Commodore faktiskt både utvecklade och lanserade en B128.

I diskussionforumet comp.sys.cbm hittade jag häromdagen lite information om en av Commodores relativt okända datorserier. Första gången Commodore visade upp sin B128 var på "Summer Consumer Electronics Show" i Maj 1982. B128:an och dess storebröder var mycket olika Commodores storsäljare Vic-20 och C64. Commodore siktade med B128:an in sig p) mer "seriösa" användare, alltså en ren arbetsmaskin.

Tre olika modeller ingick i B serien, B128, B256 och B720. B256 är helt identisk med B128, enda skillnaden är att B256 har ytterligare 128k minne monterat i de tomma ram socklarna i B128.

B720 använde sig av samma moderkort som i B128/B256, men med ett löstagbart tangentbord och inbyggd skärm. B720 har ett strömlinjeformat hölje, enligt de som har sett den är den absolut snyggaste datorn någonsin. Dock verkar det som om B720 aldrig blev helt färdig utvecklad.

B128 var som sagt en ren arbetsmaskin och var utrustad därefter, 128k ram expanderbart till 1 megabyte, den hade också en 6550 UART med en bra RS-232 port för höghastighetsöverföring , en IEEE port, 80 kolumner monokrom, diskdrive 8050, och av någon anledning var den även utrustad med ett SID chip. Den hade ungefär samma Basic som 7.0 i C128 med avsaknad av diverse grafikkommandon och spritestöd.

Det finns inte lika mycket program att välja och vraka bland till B128 som till C64, men lite program finns dock. Hela Super serien SuperText,

Erik Paulsson har tidigare skrivit för Åtta Bitar och kan nås via redaktionen.

SuperBase etc.) gjordes även till denna dator. Ett par megabyte PD material lär också gå att få tag på.

Det finns inte så förfärligt många B128/256/720, men vissa källor uppger följande.
B128 35000 ex en annan källa uppger 15000 ex
B256 4000 ex
B720 1000ex

Då kvarstår bara frågan, vart tog den vägen? Prislappen på B128:an var från början satt lite väl högt $1700 ca 11500 kr (man bör också ha i åtanke att detta var 83-84) och samtidigt sjönk priserna på PC datorerna till ungefär samma nivå. B128:an var helt enkelt för dyr för Commodore att tillverka. Vissa anser också att avsaknaden av färg var en anledning till att B128:an aldrig blev en storsäljare.
Commodore vågade inte heller satsa på B128:an, utan satsade istället allt på C64, som i och för sig kan räknas som ett smart drag.
Så den dator som enligt många var en av de absolut mest avancerade 8-bits datorerna blev snabbt begravd.

Det finns också dem som menar på att C128 är direkt baserad på B128. Detta verkar dock inte troligt då olikheterna är större än likheterna. C128 gjordes med målet att kunna köras i 80 kolumner, och klara av CP/M, men samtidigt vara helt bakåt kompatibel med C64. B128:an saknar både Z80 processor, MMU och VIC-II chip. B128:ans minne är direkt expanderbart till 1 megabyte, det går att
expandera även C128:ans minne internt till 1 megabyte, men det är mycket omständligt. Ett Z80 processor kort till B128 var planerat men konstruktionen var helt olik den i C128 där MMU:n mappar in "rätt" processor, i B128 skulle detta skötas med hjälp av en CIA krets.

B:et i B128 står för Business, det sägs också att Commodore gjorde ytterligare en variant p) B128:an som kallades P128, d(r P:et står för Personal. Men det är en annan historia.

INTERNET LÄNKAR

WWW:

Alter:	http://www.geocities.com/SiliconValley/6645/
C65:	http://stekt.oulu.fi/~jopi/c65.html
Censor Design:	http://www.censor.net/
Central Coust Commodore Users Group:	http://www.slonet.org/~rtrissel/
CMD:	http://www.the-spa.com/cmd/
Commodore 8-bit Server:	http://www.hut.fi/~msmakela/cbm/
Commodore Computer Cult Corner:	http://www.ts.umu.se/~yak/cccc/
Cosmic Style:	http://www.tu-chemnitz.de/~dsc/c0smic/
DesTerm:	http://www.ionline.net/~mdesmond/desterm.html
Fairlight:	http://www.ludd.luth.se/~watchman/fairlight/
Flash Inc:	http://www.abc.se/%7Em9656/flashinc/
Fridge:	http://stratus.esam.nwu.edu/~judd/fridge/
In Medias Res:	http://www.kuai.se/~zike/index.html
Matthew Desmond's Homepage:	http://www.ionline.net/~mdesmond/ (mannen bakom DesTerm)
Novaterm:	http://www.eskimo.com/~voyager/novaterm.html
Omni / Revenge Homepage:	http://flash.lakeheadu.ca/~jgvotour/index.html
Propaganda:	http://www.algonet.se/~motley/propa.htm
Relik:	http://www.nethosting.com/~emotion/relik.html
Triad:	http://www.df.lth.se/~triad/triad/
Åtta Bitar:	http://www.mds.mdh.se/%7Edat95pkn/8bitar

FTP:
ftp.funet.fi/pub (Under katalogen /pub/cbm så finns kataloger med programvara för C64/C128)

Newsgroups:
comp.sys.cbm

LÄR DIG C64 ASSEMBLER

Jens Örtenholm

Så var det då dags för ytterligare en del i vår kurs för assembler-programmering på C64:an, närmare bestämt del 2. För att hänga med så bör du ha läst förra delen, som återfinns i Relik #1.

Till att börja med tänkte jag att vi skulle titta på några fler instruktioner, så att vi har någonting att leka med när vi ska programmera.

INX, INY (Increase X, Y): Dessa enkla instruktioner ökar helt enkelt värdet i X- respektive Y-registren med ett. De är speciellt användbara i slingor av olika slag (som du strax ska få se). Instruktionerna tar inga parametrar, som exempel: INX.

INC (Increase): Den här instruktioner ökar innehållet på en viss minnesadress med 1. T.ex. kan du använda instruktionen INC $1000 för att öka innehållet på adress $1000 med 1.

CPX, CPY (ComPare X, Y with): De här två instruktionerna glömde jag att ta med i förra numret, och de fungerar på samma sätt som CMP från förra artikeln, bara det att de jämför värdet i X med parametern.

BRK (Break): Den här instruktionen stoppar helt enkelt programkörningen.

JSR (Jump Sub-Routine): JSR används för att hoppa till en adress, utföra ett antal instruktioner och slutligen återvända och fortsätta på nästa rad. Anta att vi t.ex. har ett JSR $1000 i vårt program. Processorn kommer då köra koden som finns på adress $1000, och när den hittar en specialinstruktion så återvänder den till raden med vårt JSR och fortsätter att köra koden under detta.

RTS (Return from Sub-Routine): RTS är den specialinstruktion som man lägger in i sina subrutiner för att återvända från den när den är färdig.

CLC (Clear Carry): Denna instruktion nollställer carryflaggan (mer om den senare).

SEC (Set Carry): Den här instruktionen är raka motsatsen till CLC, och sätter alltså carryflaggan.

ADC (Add With Carry): ADC är en additionsinstruktion som ökar på värdet i ackumulatorn med ett direkt värde (t.ex. #$25) eller innehållet på en adress (t.ex. $1000). Carryflaggan påverkar dock värdet, hur och varför kan du se nedan i avsnittet om addition och subtraktion.

SBC (Subtract...Carry): SBC är en subtraktionsinstruktion som minskar värdet i ackumulatorn med ett direkt värde (t.ex. #$42) eller innehållet på en adress (t.ex. $1000). Carryflaggan påverkar dock värdet, hur och varför kan du se nedan i avsnittet om addition och subtraktion.

SLINGOR

Då ska vi passa på att omsätta våra kunskaper, både nya och gamla, i ett par vanliga programmeringstekniker. Till att börja med tänkte jag att vi skulle titta på slingor, eftersom det är en så vanlig företeelse i ett program. Slingor är helt enkelt en teknik för att utföra en serie instruktioner antingen ett visst antal gånger, eller tills ett visst förhållande uppstår. På så sätt så slipper man skriva långa rutiner som gör samma sak om och om igen t.ex. 10 gånger.

Slingor konstruerar man oftast med hjälp av våra indexregister, X eller Y. Man väljer en av dem som räknare, nollställer denna och räknar upp den för varje varv i slingan. I slutet på slingan så jämför man värdet i räknaren med hur många varv slingan ska köras. Om programmet inte har nått upp till rätt antal varv så kör vi helt enkelt ett varv till. Ett litet exempel är detta, som ökar färgen på ramkanten (vilket man gör med adress $D020) tio gånger:

```
                = $1000

                LDX #$00
igen            INC $D020
                INX
                CPX #10
```

Jens Örtenholm kan nås via email: jens.ortenholm@pc-programs.se, eller via redaktionen.

```
                BNE igen
evighet         JMP
evighet
```

Detta enkla program, som du ska skriva in i Turbo Assembler om du vill provköra det, börjar med att ange att koden ska läggas på adress $1000. Sedan nollställer vi vår räknare, X och efter denna instruktion så börjar slingan. Vi ökar innehållet i adress $D020 (alltså, ramkantens färg) med ett, och passar på nästa rad på att öka räknarens värde med ett, för att sedan jämföra X-registret med det decimala talet 10 (kom ihåg att bara ett # utan $ efter betyder att det är det decimala talet vi avser). Om X-registret inte var detsamma som 10 så hoppar vi tillbaka till den plats i koden som markeras med etiketten **igen**, och det gör vi med hjälp av BNE-instruktionen (BNE igen, hoppa om ej lika till etiketten igen). När X-registrets värde når 10 så kommer BNE-instruktionen inte att utföra någonting, och programmet kommer sedan stanna i en evighets-slinga med instruktionen JMP evighet som om och om igen kommer hoppa till samma JMP-instruktion hela tiden.

För att avbryta programmet, reseta datorn och starta om Turbo Assembler med SYS $9000, eller SYS 4096*9.

Som du säkert förstår så skulle vi enkelt kunna modifiera programmet så att slingan utfördes ända tills ett visst tillstånd inträffade. Följande exempel är nära nog identiskt:

```
                = $1000

igen            INC $D020
                LDA $D020
                CMP #10
                BNE igen
evighet         JMP
evighet
```

Istället för att använda X-registret som en räknare, så ökar vi innehållet i $D020 och hämtar sedan in värdet i ackumulatorn med instruktionen LDA $D020. När vi sedan har värdet där så undersöker vi det med CMP för att se om det har blivit 10. Om det inte har det, så kommer BNE-instruktionen hoppa tillbaka och utföra ytterligare en ökning och en jämförelse, annars hamnar vi i den enkla evighets-loopen.

Någonting som nybörjare i assembler ofta inte tänker på är att när man hämtar in ett värde i ett register, t.ex. med LDA, så förstörs det värde som fanns där innan. Om en slinga beror på att ett visst värde hela tiden finns i t.ex. X-registret, så får man inte skriva över det värdet med någonting annat, för då börjar programmet bete sig konstigt.

SUBRUTINER

Någonting som är väldigt viktigt när man programmerar är att man delar upp sitt program i små, lätthanterliga bitar. Detta gör man oftast genom att använda sig av så kallade subrutiner, d.v.s små programdelar som kan anropas med ett JSR från vilken annan del av programmet som helst. Programdelar som det kan vara lämpligt att göra till subrutiner är kodstycken som kommer användas på flera ställen i samma program (då kan man istället för att skriva in kod som utför samma sak på flera ställen bara anropa subrutinen) och brottstycken av en programbit som har blivit alldeles för lång för att man enkelt ska kunna se flödet i den.

Som exempel så kan vi anta att vi på flera ställen i vårt program kommer behöva anropa en rutin som ökar både ram- och skärmfärgen med ett steg. Nu är ju det en rutin som inte tar särskilt många rader kod (faktiskt bara två instruktioner), och som det egentligen är rätt onödigt att skriva en subrutin för, men vi låtsas att det är en bra idé bara för att få ett aning om hur det fungerar.

```
                = $1000

                JSR farg
                JSR farg
                JSR farg
evighet         JMP
evighet

farg            INC $D020
                INC $D021
                RTS
```

Som du ser så kommer vårt program att anropa den subrutin som anges med etiketten **farg** tre gånger, innan det ger sig in i en evighets-loop. Vi placerar vår subrutin alldeles nedanför genom att ange etiketten farg och sedan skriva vår kod. Koden ökar färgen på först ramkant och sedan bakgrund med ett, och återvänder sedan från rutinen med instruktionen RTS. Enkelt!

På alla de ställen i vårt program där vi vill öka färgerna så lägger vi alltså in en rad med instruktionen JSR farg.

För att illustrera den andra typen av rutin som det är lämpligt att göra en subrutin av, så tänkte jag beskriva ett exempel. Anta att vi har ett program vars huvudrutin består av en slinga som utför olika saker. Normalt skulle denna huvudrutin, inklusive slingan, vara flera tusen kodrader lång, och det skulle bli ganska svårt att överblicka och förstå vad den sysslade med. För att bryta ned huvudprogrammet så att det blev enklare att hantera så skulle man kunna bryta ut de uppgifter som slingan utförde, en och en, och lägga varje uppgift som en subrutin. På så sätt skulle man kunna förkorta huvudslingan till kanske runt 50 rader, och den skulle mest innehålla en massa JSR till subrutinerna. Det skulle göra att det blir enklare att se vad programmet sysslar med, samt att det blir en mindre del kod att felsöka om det är problem med någon enskild uppgift, då man bara behöver felsöka just den subrutinen.

ADDITION/SUBTRAKTION

Vad datorer är absolut bäst på är ju att räkna, och därför är det viktigt att kunna utföra beräkningar i sina program. Som en introduktion till matematik i assembler så tänkte jag att vi skulle ta en närmare titt på de instruktioner som C64:ans processor tillhandahåller för addition och subtraktion.

För att addera två tal, så ska man använda sig av instruktionen ADC, och får då placera det ena talet i

ackumulatorn och sedan addera med det andra. Vad man ibland kan glömma bort är att ADC faktiskt betyder Add With Carry, och att carry-flaggan alltså kommer adderas till resultatet. Det här fyller ett syfte, vilket vi snart ska titta på.

För att addera talen #$12 och #$13 så gör vi alltså på följande sätt:

```
          = $1000

            LDA #$12
            CLC
            ADC #$13
            STA $D020
evighet     JMP
evighet
```

Det här programmet stoppar först in det första talet i ackumulatorn med hjälp av instruktionen LDA #$12. Sedan använder vi instruktionen CLC (Clear Carry) för att nollställa carry-flaggan så att inte additionen kommer ge ett felaktigt resultat. Instruktionen ADC #$13 lägger sedan till #$13 till innehållet i ackumulatorn, och för att se vad det hela ger för resultat så stoppar vi in värdet som ramkants-färg, alltså i adress $D020. När det är färdigt så försätter vi programmet i en evighets-loop. Om det hela fungerar som det ska, så ska ramkantens färg ändras till grönt.

Carry-flaggans funktion är faktiskt ganska viktig. Anta att vi vill addera talen #$FF med #$02. Det skulle ge ett resultat vilket var större än #$FF, och alltså inte fick plats i en bytes minne. Carry-flaggan skulle då sättas, som en indikation på att vi hade gått över gränsen (ungefär som en minnessiffra i en matematisk uppställning), och resultatet skulle bli #$01. Genom att undersöka värdet i carry-flaggan efter additionen, så skulle vi kunna lagra talets andra del någon annanstans, och med andra ord jobba med 16-bitars tal. Som exempel:

```
          = $1000

            LDA #$00
            STA $D020
            STA $D021
            LDA #$FF
            CLC
            ADC #$02
            STA $D020
            BCC
inteover
            INC $D021
inteover    JMP
inteover
```

Det här exemplet börjar med att nollställa innehållet i adresserna $D020 och $D021, vilket kommer sätta ramkantens och bakgrundens färg till svart (för svart representeras av siffran 0). Sedan adderar vi på samma sätt som i förra exemplet talen #$FF och #$02. Resultatet placerar vi i $D020. Men efter detta så blir det lite annorlunda. Om vi hade gått över gränsen för vad ADC klarade av, så kommer carry-flaggan att sättas. Därför använder vi instruktionen BCC (hoppa om carry-flaggan EJ är satt) för att hoppa över en bit kod ifall det inte blivit något överslag. Har det däremot blivit det, så kommer raden INC $D021 att köras, och vi kommer alltså se det genom att bakgrundsfärgen ändras från svart. När detta är färdigt så hamnar vi återigen i en evighets-loop. Om du provkör programmet så ska både bakgrundsfärgen och ramfärgen ändras till vitt, vilket är helt korrekt eftersom resultatet blir #$01 (och #$01 representerar färgen vitt) och vi ökar $D021 från #$00 till #$01 om vi fick överslag.

Subtraktion fungerar på ett liknande sätt, men här använder vi oss av instruktionen SBC. Det intressanta med SBC är att här subtraheras den *inverterade* carry-flaggan. Alltså, för att subtrahera två små tal, t.ex. #$01 från #$02, så gör vi följande:

```
          = $1000

            LDA #$00
            STA $D020
            STA $D021
            LDA #$02
            SEC
            SBC #$01
            STA $D020
evighet     JMP
evighet
```

Det första vi gör är att sätta bakgrunds- och ram-färg till 0 (d.v.s svart), och sedan hämtar vi in talet 2 i minnet. För att subtrahera 1 från det, så **sätter** vi först carry-flaggan med instruktionen SEC och subtraherar sedan med SBC #$01. Anledningen till det är att carry-flaggan inverteras vid användning, så att om carry-flaggan är 0 så kommer ytterligare 1 att subtraheras från talet. Därför måste vi se till att carry-flaggan är satt vid vanliga subtraktioner. Resultatet kommer placeras i ramfärgen, och den ska bli vit eftersom resultatet ska bli #$01.

Nu ska vi ta en titt på vad som händer om vi t.ex. subtraherar #$02 från #$01. Vad som händer när vi går över gränsen är att carry-flaggan nollställs. Ett exempel:

```
          = $1000

            LDA #$00
            STA $D020
            STA $D021
            LDA #$01
            SEC
            SBC #$02
            STA $D020
            BCS ejover
            INC $D021
ejover      JMP ejover
```

Först sätter vi ram- och bakgrunds-färg till svart, och sedan hämtar vi in talet 1, sätter carry-flaggan och drar ifrån 2. Resultatet från den uträkningen placerar vi i $D020, och om carry-flaggan fortfarande är satt (d.v.s vi gick inte förbi gränsen) så hoppar vi till ejover. Annars ökar vi $D021, vilket indikerar att vi hamnade under noll, och vi hamnar sedan i evighets-loopen. Om du provkör programmet så kommer du se att bakgrundsfärgen blir vit, för att visa att vi slog över, och att ramfärgen blir grå, för att visa att resultatet blir #$FF.

Då var vi klara för den här gången. Som övning får du gärna skriva ett par program som räknar fram och tillbaka, och kom ihåg att du också kan använda t.ex. CMP för att kolla att du har fått önskat resultat.

Internet Uppkoppling

Erik Paulsson

Peter Karlsson har skrivit om vad som finns på Internet, jag tänkte skriva lite om hur du kopplar upp din C64/128 mot nätet, för det behövs faktiskt inga dyra PC maskiner för detta ändamål. ***(Denna artikel är ursprungligen avsedd för Åtta Bitar)***

Modemet
Till att börja med behöver du ett modem, då är frågan hur snabbt modem man ska köpa. På en C64 kan man utan Swiftlink bara köra i 2400 bps, på en C128 däremot kan man komma upp i 9600 bps med Desterm men Desterm saknar stöd för hårdvaruhandskakning vilket gör att det blir opålitligt i hastigheter över 2400 bps när man kör mot Internet, det går men man får en hel del "skräp" på skärmen vilket kan vara ganska irriterande. En ny version av Desterm är sedan länge utlovad men det verkar dröja innan den blir klar. Men man kan ju köra i 9600 bps när man läser email och surfar på World Wide Web och gå ned till 2400 bps när man behöver helt säkra överföringar som t.ex. när man plockar hem filer med FTP. Om man däremot vill komma upp i riktigt höga hastigheter måste man skaffa en Swiftlink som tillåter hastigheter ända upp till 38400 bps. Men då blir det genast dyrare. Om man vill prova på Internet och se efter vad det har att ge så räcker det alldeles utmärkt med 2400 bps.

SLIP
Under utveckling är ett program till C64 som används för att koppla upp sig med SLIP, med detta program kommer man med ett speciellt interface upp i 9600 bps. Än så länge finns bara en demoversion.

Utan Swiftlink
Om du väljer att koppla upp dig utan Swiftlink och nöjer dig med att köra i 2400 (9600/2400 på en 128:a) så behöver du förutom modemet ett modem interface, sådana finns att köpa från Sandinge's. Men jag föredrar faktiskt ett hembyggt. Detta hembyggda interface görs med en 7404 krets som kostar ca 5-6 kr, en bit kabel (flatkabel duger bra), en Userport kontakt och en 25 eller 9 polig D-sub. Om du vill kan du montera alltihop innanför datorns hölje och slipper på så vis kostnaden för en Userport kontakt eftersom det då är enklast att löda fast sladdarna direkt på kortet.
Om du däremot väljer att göra interfacet helt externt så kan jag rekommendera att du lägger 7404 kretsen i en av kontakternas kåpor, det är lite trångt men enklast eftersom man gör kabel och interface i ett, köper du däremot ett vanligt modeminterface så måste du skaffa en kabel också. Oavsett om du gör en extern kabel eller monterar alltihop i datorn så skall det kopplas enligt följande kopplingsschema:
Userport In IC7404 Ut 25 D-sub 9

M - 11 10 - 2 / 3
B&C - 4 3 - 3 / 2
A - 7 7 - 7 / 5
2 - 14
Bygla 4 &5 / 7 & 8

Om allt fungerar så är det dags att titta på vilken Internet leverantör man skall välja. Vilken spelar ingen större roll, men var mycket noggrann med att kolla att de verkligen ger möjlighet att koppla upp sig mot ett UNIX skal, alla leverantörer ger nämligen inte denna möjlighet. Det finns många att välja bland, det enklaste är att köpa en Internet tidning och ta en titt på vad som finns i närheten. Själv har jag ett "konto" hos Algonet där det går alldeles utmärkt att koppla upp sig mot ett UNIX skal. Algonet bygger hela tiden ut sitt system och sätter upp nya modempooler så att alla ska kunna nå dem med närsamtal, denna utbyggnad av modempoolerna beräknas vara klar i Juni. Om du vill ha mer information kan du ringa Algonet på tel: 08-587 587 00, Fax: 08-587 587 30.

Börja surfa!
När du har både modem och Internet abonnemang så är det bara att sparka igång terminalprogrammet (förslagsvis NovaTerm 9.5 eller Desterm 2.00) och köra igång! Om du behöver hjälp med att få tag på nödvändiga komponenter eller terminalprogram eller om du bara undrar något i största allmänhet så är du välkommen att höra av dig till undertecknad.

Erik Paulsson har tidigare skrivit för Åtta Bitar och kan nås via redaktionen.

Listning: DIVISION

Alexander Hole

```
          *= $1000
nemn1     = $0336
answ1     = $0337
answ2     = $0338
answ3     = $0339
answ4     = $033a
answ5     = $033b
answ6     = $033c
answ7     = $033d
answ8     = $033e
stor1     = $033f
nemn2     = $0341
stor2     = $0342
nemn3     = $0343
nemn4     = $0344
stor3     = $0345
stor4     = $0346
stor5     = $0348
stor6     = $0349
stor7     = $034a
stor8     = $034b
nemn5     = $034c
nemn6     = $034d
nemn7     = $034e
nemn8     = $034f
telj1     = $0350
telj2     = $0351
telj3     = $0352
telj4     = $0353
sign1     = $0354
sign2     = $0355
sign3     = $0356
;--------------------
;-        *      32 bitars      *         -
;-        *  fixpoint division  *         -
;-        * med 64 bitars svar  *         -
;-        *   för + och - tal    *         -
;-        *        av            *         -
;-        *   Alexander Hole     *         -
;-        *  Februari  1996      *         -
;--------------------
          lda #$00       ;nollstellning
          sta stor1      ;av
          sta stor2      ;n0dvendiga
          sta stor3      ;register
          sta stor4      ;
          sta stor5      ;stor=jemf0relse
          sta stor7      ;    buffert
          sta stor7      ;
          sta stor8
          sta answ1      ;
          sta answ2      ;answ=svar
          sta answ3      ;register med
          sta answ4      ;decimaler i
          sta answ5      ;form utav
          sta answ6      ;binaler
          sta answ7      ;
          sta answ8
          sta nemn5      ;nemnarens
          sta nemn6      ;outnyttjade
          sta nemn7      ;32bits se nedan!
          sta nemn8
          lda #$08       ;her placeras
          sta nemn1      ;legsta byten,
          lda #$00       ;her nest legsta
          sta nemn2      ;osv.....
          lda #$00       ;dessa 32 bitar
          sta nemn3      ;(nemn1-4)
          lda #$00       ;er nemnaren!!!
          sta nemn4
          lda #$04       ;och i dessa byte
          sta telj1      ;placeras
          lda #$00       ;teljarens
          sta telj2      ;fyra bytes
          lda #$00       ;(telj1-4)
          sta telj3      ;der telj1 er
          lda #$00       ;legst!!!!!!
          sta telj4
          lda #$00       ;0 betyder att
          sta sign1      ;nemn er +,1=-
          lda #$00       ;dito f0r telj
          sta sign2
          jmp lam        ;pga begrensning
                         ;bmi:ar krevs ett
nepp2     jmp nepp       ;hopp f0rlengning
                         ;f0r kommande bmi
lam       ldx #$3f       ;#$3f = 64 bitar

          asl telj1      ;her skiftas
          rol telj2      ;teljaren
loop      rol telj3      ;in i
          rol telj4      ;bufferten
          rol stor1      ;f0r att kolla om
          rol stor2      ;teljaren er
          rol stor3      ;mindre eller
          rol stor4      ;st0rre en
          rol stor5      ;nemnaren
          rol stor6
          rol stor7
          rol stor8
          lda stor8      ;her jemf0rs
          cmp nemn8      ;buffertens
          bmi nepp2      ;(stor1-8)
          lda stor7      ;atta bytes
```

Alexander Hole har tidigare skrivit för Åtta Bitar och kan nås via redaktionen.

```
            cmp nemn7       ;med nemnarens
            bmi nepp        ;n0dvendiga
            lda stor6       ;atta bytes
            cmp nemn6       ;
            bmi nepp        ;var utav vi
            lda stor5       ;anvender fyra
            cmp nemn5       ;f0r att holla
            bmi nepp        ;oss till de
            lda stor4       ;regler som
            cmp nemn4       ;vi lert oss
            bmi nepp        ;i skolan
            lda stor3       ;om divisions
            cmp nemn3       ;precision....
            bmi nepp        ;
            lda stor2       ;det er lett
            cmp nemn2       ;att anvenda alla
            bmi nepp        ;64 bitar i nemn
            lda stor1       ;ge nemn5-8 ett
            cmp nemn1       ;annat verde en
            bmi nepp        ;noll!!!!!!!!!!
            lda stor1
            sec
            sbc nemn1       ;om talet i
            sta stor1       ;bufferteb er
            lda stor2       ;st0rre en
            sbc nemn2       ;talet i
            sta stor2       ;nemnaren so
            lda stor3       ;ska man ta bort
            sbc nemn3       ;so mycket som
            sta stor3       ;er resonabelt
            lda stor4       ;ur bufferten
            sbc nemn4       ;
            sta stor4       ;
            lda stor5       ;OBSERVERA!
            sbc nemn5       ;talet
            sta stor5       ;i bufferten
            lda stor6       ;kan var st0rre
            sbc nemn6       ;en vad teljaren
            sta stor6       ;er, eftersom
            lda stor7       ;fyra bytes
            sbc nemn7       ;roteras in i
            sta stor7       ;atta stycken
            lda stor8       ;Det er detta
            sbc nemn8       ;som g0r bla deci
            sta stor8       ;-maler m0jligt
            sec             ;Her tends eller
            jmp rols        ;eller slecks
nepp        clc             ;carry beroende
rols        rol answ1       ;po resultatet
            rol answ2       ;av jemf0relserna
            rol answ3       ;ovan
            rol answ4       ;
            rol answ5       ;svaret leggs
            rol answ6       ;binert i
            rol answ7       ;answ1-8
            rol answ8
            dex             ;
            bmi loop2       ;tillbaka till
            jmp loop        ;start po loopen
loop2                       ;
                            ;her bereknar om
            lda sign1       ;talet er + eller
            eor sign2       ;-,nu f0rsor ni
            sta sign3       ;vel varf0r jag
                            ;valde 0 som +
                            ;och 1 som -!!!
;--------------------
;-       * 64 bitars tal till BCD    *    -
;-      *           av                *    -
;-       *     Alexander Hole          *    -
;-       *       Teori bakom           *    -
;-      *            av                *    -
;-       *          Simon              *    -
;-       *      Kristoffersson         *    -
;--------------------
bcd01     = $0357     ;bytar f0r minsta
bcd02     = $0358     ;enheten
bcd03     = $0359     ;---------
bcd04     = $035a     ;Det tomma utrymmet
bcd05     = $035b     ;her anvender jag
bcd06     = $035c     ;nu till att f0rklar
bcd07     = $035d     ;a varf0r,ner man
bcd08     = $035e     ;koverterar binera
bcd09     = $035f     ;tal med binaler til
bcd10     = $0360     ; BCD so blir svaret
bcd11     = $0361     ;inte riktigt exakt!
bcd12     = $0362     ;
bcd13     = $0363     ;Den minsta talet
bcd14     = $0364     ;som kan beskrivas
bcd15     = $0365     ;med divisions prog
bcd16     = $0366     ;ovan er(se efter
bcd17     = $0367     ;kommentar)
scd01     = $0378     ;---------
scd02     = $0377     ;bytar f0r BCD svar
scd03     = $0376     ;---------
scd04     = $0375     ;1/(2 upph0jt i 32)
scd05     = $0374     ;vilket inte er ett
scd06     = $0373     ;exakt decimal tal,
scd07     = $0372     ;och logiskt nog
scd08     = $0371     ;so er det so ,att
scd09     = $0370     ;so lenge man
scd10     = $036f     ;avrundar utgongs
scd11     = $036e     ;verdena kommer
scd12     = $036d     ;even svaret vara
scd13     = $036c     ;lite fel!Desto fler
scd14     = $036b     ;siffror man har des
scd15     = $036a     ;to bettre svar,men
scd16     = $0369     ;man moste beherska
scd17     = $0368     ;sig!!! mao inte 60k
;---------;decimal avrundning:)
          lda #$39      ;
          sta bcd01     ;her har vi
          lda #$65      ;talet !!!!
          sta bcd02     ;
          lda #$43      ;dvs
          sta bcd03     ;
          lda #$06      ;1/(2 upph0jt i 32)
          sta bcd04     ;
          lda #$83
          sta bcd05
          lda #$32
          sta bcd06
          lda #$02
          sta bcd07
          lda #$00
          sta bcd08     ;
          sta bcd09     ;
          sta bcd10     ;t0mning av
          sta bcd11     ;0vriga BCD
          sta bcd12     ;lagrings
          sta bcd13     ;adresser
          sta bcd14     ;
          sta bcd15     ;
          sta bcd16     ;
          sta bcd17
          sta scd01
          sta scd02
          sta scd03
          sta scd04
          sta scd05
          sta scd06
          sta scd07
          sta scd08
          sta scd09
          sta scd10
          sta scd11
          sta scd12
          sta scd13
          sta scd14
          sta scd15
          sta scd16
          sta scd17
```

```
          ldx #$3f       ;64 bittar!
loop3
          lsr answ8      ;
          ror answ7      ;
          ror answ6      ;kolla om non
          ror answ5      ;bit i det
          ror answ4      ;binera svaret
          ror answ3      ;er satt
          ror answ2
          ror answ1
          bcs hepp2
          jmp hepp
hepp2
          sed
          lda scd01      ;om det er det
          clc            ;so ska man
          adc bcd01      ;addera den
          sta scd01      ;bittens verde
          lda scd02      ;till
          adc bcd02      ;BCD svaret
          sta scd02      ;
          lda scd03      ;anmerk po att
          adc bcd03      ;jag her
          sta scd03      ;anvender nonting
          lda scd04      ;riktigt koolt!
          adc bcd04      ;Nemligen den
          sta scd04      ;inbyggda BCD
          lda scd05      ;funktionen
          adc bcd05      ;f0r 6510 CPU
          sta scd05      ;
          lda scd06      ;Monga kellor
          adc bcd06      ;seger att denna
          sta scd06      ;funktion er
          lda scd07      ;buggig och inte
          adc bcd07      ;funkar
          sta scd07      ;ordentligt po
          lda scd08      ;vissa 6510 CPU
          adc bcd08      ;men alla 6510or
          sta scd08      ;jag testat har
          lda scd09      ;bara lite
          adc bcd09      ;standard buggar
          sta scd09      ;
          lda scd10      ;inga riktiga
          adc bcd10      ;horda reknefel!
          sta scd10      ;
          lda scd11      ;
          adc bcd11      ;
          sta scd11
          lda scd12
          adc bcd12
          sta scd12
          lda scd13
          adc bcd13
          sta scd13
          lda scd14
          adc bcd14
          sta scd14
          lda scd15
          adc bcd15
          sta scd15
          lda scd16
          adc bcd16
          sta scd16
          lda scd17
          adc bcd17
          sta scd17
hepp
          sed            ;her har vi SED
          lda bcd01      ;igen
          clc            ;(SEt Decimal mode)
          adc bcd01      ;
          sta bcd01      ;
          lda bcd02      ;her bereknas de
          adc bcd02      ;binara bittarnas
          sta bcd02      ;BCD verden
          lda bcd03      ;
          adc bcd03      ;den legsta har
          sta bcd03      ;ungefer
          lda bcd04      ;
          adc bcd04      ;2.3*10 upph0jt
i-10
          sta bcd04      ;
          lda bcd05      ;och nesta dubbla
          adc bcd05      ;
          sta bcd05      ;so den her delen
          lda bcd06      ;
          adc bcd06      ;dubblar i
princip
          sta bcd06      ;
          lda bcd07      ;det f0rra talet
          adc bcd07      ;
          sta bcd07      ;bdc01-17=2*bcd01-
17
          lda bcd08      ;
          adc bcd08      ;
          sta bcd08
          lda bcd09
          adc bcd09
          sta bcd09
          lda bcd10
          adc bcd10
          sta bcd10
          lda bcd11
          adc bcd11
          sta bcd11
          lda bcd12
          adc bcd12
          sta bcd12
          lda bcd13
          adc bcd13
          sta bcd13
          lda bcd14
          adc bcd14
          sta bcd14
          lda bcd15
          adc bcd15
          sta bcd15
          lda bcd16
          adc bcd16
          sta bcd16
          lda bcd17
          adc bcd17
          sta bcd17      ;
          cld            ;man for inte
          dex            ;gl0mma stenga
          bmi loop32     ;av BCD lege
          jmp loop3      ;eftersom BASICen
loop32                   ;och KERNEL inte
                         ;arbetar i det
                         ;leget!
                         ;inte min
assembler
                         ;heller och detta
                         ;har lett till
                         ;att jag fott
skriva
                         ;det her monga
                         ;gonger extra!
;--------------------
          rts
;--------------------
;Om du har en maskinkods monitor
;so tryck "m 0368" so for du de f0rsta
;atta bytesen och "m 036f" f0r nesta
;och "m 0377" f0r sista byten!
;(av svaret alltso,i decimaltal!!!!)
;Om svaret er plus eller minus hittar
;du i byte $0356.
;Tack till Simon f0r teorin bakom BCD
;rutinen!
```

LISTA PÅ BBS:ER MED INNEHÅLL FÖR ÅTTA-BITARS DATORER

Namn:	*Sysop:*	*Nummer:*	*Datorer:*
Antidote	Taper / Triad	042-76416	64/128
Fosie BBS	Natas	040-269767	64/128
The Studio	Jerry / Triad	0159-31991	64/128

*Utrustning: C64, 2400, 8 MB RamLink (10*1581)*

Warez Aquarium	Sledge / FLT	08-371360	64/128

*Utrustning: C64, 2400, CMD HD 540 MB / JiffyDOS, 1581, C*Base 3.2*

WinterMute	Emotion	Tf. Nere	64/128

Utrustning: Pentium 133, Courier v.34+, 4GB HD, CD-ROM
Hemsida: http://www.nethosting.com/~emotion/winterm.html