Uptime

Datorföreningen Updates nyhetsblad

Redaktör: Bjarni Juliusson

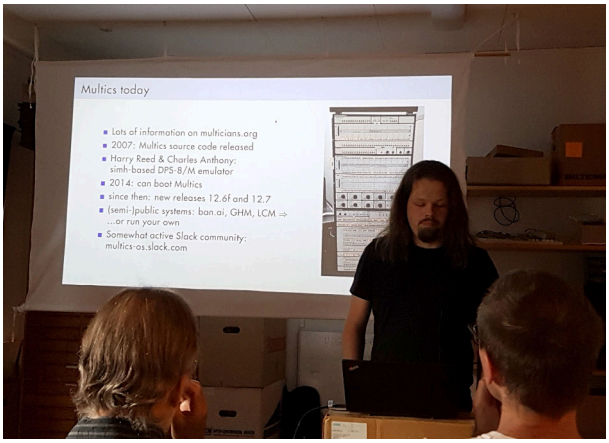
Nummer 3 2022

Den här gången blev det ett lite tjockare nummer, tack vare en mycket lärorik Månadens Dator. Förutom PDP-11-meck har arbetet fortsatt under sommaren med städning och sortering av hårdvara, minikonferens samt plenumsmöten—våra månatliga informella planerings- och uppföljningsmöten. Har du nyheter eller förslag till Uptime, eller har du skrivit något du vill dela med dig av, skicka ett brev till <uptime@dfupdate.se>.

Minikonferensen i juli

Den 16:e juli hölls Updates tredje minikonferens, den första efter Covid-19 och den första i Updates nya lokal. Fyra talare föreläste — två av dem på besök från utlandet, och tre av dem med liveuppvisningar. Vår egen Björn Victor demonstrerade användande av Chaosnet på en modern dator, Lars Brinkhoff demonstrerade det bland datorhistoriska fantaster namnkunniga men sällan skådade operativsystemet WAITS, och Angelo Papenhoff demonstrerade det dito Multics. Mark Kahrs talade om datormuseet Large Scale Systems Museum utanför Pittsburgh, där han hjälper till, och om SDS-940, en mindre välkänd datormodell från 1960-talet. Det var fyra intressanta presentationer, som visade upp olika personers och gruppers arbete med att tillgängliggöra historisk datorteknik för allmänheten.

Demonstrationen av Multics uppskattades särskilt av undertecknad; Multics är historiskt ett mycket inflytelserikt operativsystem, som dock var nära att gå förlorat efter att projektet lagts ner, och den sista körande installationen tagits ur drift 2000. Multics körde på en enda, särskilt utvecklad stordatorplattform, och efter 2000 fanns varken någon körbar hårdvara, någon emulator, eller någon offentligt tillgänglig distribution av operativsystemet. Företaget som vid den tiden ägde upphovsrätten visade ringa intresse för den nedlagda produkten. Tack vare entusiasters lobbying släpptes dock källkoden fri 2007, och 2014 kunde Multics återuppstå med hjälp av en emulator. Sedan dess har utvecklingen av operativsystemet fortsatts av hobbyister, och idag finns installationer igen tillgängliga för login på internet.



aap presenterar Multics

Konferensen besöktes av ett dussintal personer och visades även live online. Efter föreläsningarna följde en gemensam middag och umgänge, och dagen efter träffades en del av deltagarna igen för ett litet hackathon i Updates lokaler med gemensamt arbete på diverse datorrelaterade projekt.

Tack till Björn, Lars, Angelo och Mark för era intressanta och informativa bidrag, och tack till alla deltagare för ett par trevliga dagar. Välkomna åter till framtida konferenser!

Svartbäcksgatan 65

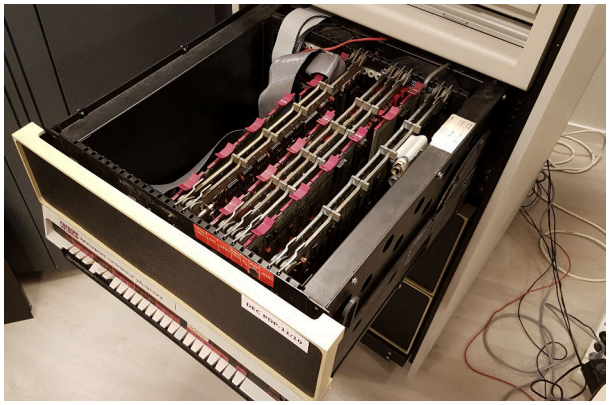
Köket/verkstan och utställningsrummet prioriteras och töms på lådor för att göra det möjligt att till fullo använda dessa rum för aktiviteter och utställningar. Lådorna packas i möjligaste mål om för att minska platsen de tar upp, och flyttas till delen av L-rummet där soffan står, som tills vidare får fungera som lagerutrymme för att frigöra de viktigare ytorna. Soffan hålls fri. Köket/verkstan har möblerats om lite för att förbättra användandet av borden.

Juli månads dator: PDP-11/10

Denna gång är det en minidator: Digital Equipment Corporations PDP-11/10, introducerad 1972 som lågprismodellen i den då nya PDP-11-familjen, en familj av ganska kompetenta 16-bitars minidatorer. Den följde på PDP-11/20, basmodellen, som introducerats 1970. Som prisjämförelse: en PDP-11/20 med 16Kbyte minne kostade cirka 14000 USD vid introduktionen; en PDP-11/10 med lika mycket minne cirka 6500 USD. Prestandaskillnaden var trots det ganska liten. PDP-11/10 lämpade sig bra som en liten dator för laboratoriebruk samt styr&mät-tillämpningar, och blev väldigt populär. Den har en minnescykeltid på en knapp mikrosekund, och en typisk instruktion tar ungefär 3–4µs, om det ger en känsla för hastigheten.

Var Updates exemplar kommer ifrån har vi inte lyckats ta reda på (hör av dig om du vet!), men den är bestyckad med en AR11, vilket är ett analoggränssnitt med ett antal in- och ut-kanaler och en timer. Den har även en dubbel 8-tums diskettstation av typ RX01 med en trasig drive, ett parallellportsgränssnitt typ DR11-C, en RL11 diskkontroller (dock ingen disk), samt 32Kbyte kärnminne. Datorn har också en terminallina och ett ROM-kort med bootloader, modell M9301-YA.

Datorn har visat sig vara i fungerande skick, förutom att den ena av de två RX01-drivarna (drive 1) inte fungerar alls. Eventuellt är den andra driven (drive 0) inte heller pålitlig enligt tidigare rapporter, men den har fungerat de senaste veckorna. Nog med ledigt utrymme finns i racket för att montera en RL01/RL02-disk att ansluta till RL11:n om vi skulle vilja det. Vi har flera exemplar av båda sorters diskdrive. Vi har inte undersökt om skenorna passar.



CPU-lådan utdragen; processorn är de två korten längst till höger

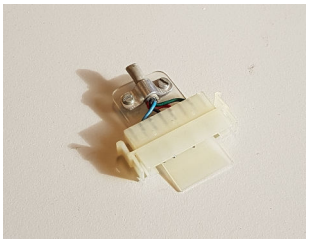
Datorn är uppbyggd av ett antal kretskort instuckna i en kortram i en CPU-låda. Korten kan vara av olika bredd; de bredaste är cirka 40cm och tar då upp hela bredden på kortramen. Själva processorn tar upp en del av kortplatserna, och andra kortplatser är lediga att användas för Unibus-expansionskort.

Processorn i PDP-11/10 heter KD11-B och tar upp två kortplatser. KA11, den processor som satt i föregångarmodellen PDP-11/20, tog upp tio kortplatser. Det var alltså en avsevärd miniatyrisering som ägt rum i den nya modellen, som till skillnad från sin föregångare använde sig av mikrokod.

CPU-lådan monteras på skenor i ett rack, och kan dras ut för att ge åtkomst till korten. På baksidan av lådan sitter ett nätaggregat som förser korten med ström, och på framsidan sitter en kontrollpanel för processorn. I racket monteras förutom CPU-lådan också andra enheter som ingår i datorn, t.ex. diskettstationer, bandstationer eller hålremsläsare. Kablaget mellan dessa enheter och processorn hänger bakom enheterna inuti racket. En strömfördelarlåda i botten av racket förser alla enheterna med ström via en gemensam strömbrytare kopplad till en nyckel på kontrollpanelen.

Två lappar sitter på datorns framsida. Den ena bär texten "2400 8N1", som är inställningarna som ska användas för konsolterminalen: 2400 baud, 8 databitar, ingen paritet, en stoppbit.

Det fysiska konsolinterfacet är av 20mA strömslingetyp, och kräver alltså en aktiv adapter för att kunna kopplas till de flesta modernare terminaler, t.ex. den VT320 vi valde att använda. Update har ingen sån adapter, men vol var vänlig nog att låna oss sin. Denna hade endast skruvkontakter, men vi hittade snabbt ett passande kontaktdon i komponentlagret vi fick av psykologiska institutionen.



Kontaktdon strömslinga

Med terminalen inkopplad och konfigurerad kan man slå på datorn. Det nästa man behöver göra är att tala om för den var i minnet den ska börja köra. Den andra lappen pekar ut startadressen i boot-romet: 173000. Detta ska läsas som ett oktalt tal, som förs in via spakarna på kontrollpanelen i motsvarande binära form: `1 111 011 000 000 000`. Därefter trycker man "load address", varefter man trycker "start". Datorn börjar nu köra programkoden i boot-romet.

På terminalen skrivs några nummer ut, följt av en prompt. Vid denna prompt kan man ange från vilken device man vill boota, och trycka enter.

Det enda sättet vi har att starta vår PDP-11/10 i nuläget är från diskett; datorn har inte förmåga att boota från RL-disk, trots RL11-kontrollern. Vi hittade en bootbar diskett med operativsystemet RT-11 version 2C från 1976, och flera disketter med diverse verktyg. 1976 är samma år som RL01 kom ut, och stöd för RL01 finns inte i denna version av RT-11. Det hade varit trevligt att kunna ha 5Mbyte lagring på en RL01 istället för de 256Kbyte som får plats på en RX01-diskett, även om vi inte kunde boota från den.

Det fanns förresten andra operativsystem för PDP-11 också. Redan 1970 fanns åtminstone två: DOS-11 kunde bootas från hårddiskar såsom RK03 eller RP03, eller från RX01-diskett, eller också från DECtape. Det kassettbaserade CAPS-11 kunde användas på ett minimalt PDP-11-system med endast en DECassette-bandspelare som lagring. Båda var väldigt enkla system. När RT-11 kom 1973 sågs det som en klar förbättring, och snart var det det mest populära systemet.

Hur som helst, vi stoppar i disketten i drive 0, skriver "DX" (stora bokstäver) vid prompten för att boota från RX01, och trycker enter. Datorn klonkar en stund, och så:

```
RT-11SJ    V02C-02H


.
```

"SJ" upplyser oss att detta är "Single-Job"-versionen av RT-11. Det finns även en RT-11FB, "Foreground/Background", som har stöd för assymmetrisk multitasking med två körande program. I SJ-versionen kör man bara ett program åt gången.

På våra disketter finns en texteditor, en assembler, en FORTRAN och en BASIC. Assembler på PDP-11 är väldigt trevligt att programmera, men FORTRAN var nog lika populärt, om inte populärare. Vi testade att skriva ett hello-world-program i assembler, och det fungerade fint. Vi anslöt också en högtalare till en av de analoga utkanalerna från AR11-kortet (vars dokumentation finns på Bitsavers) och skrev ett testprogram som genererade en sågtandsvåg med varierande tonhöjd. Det fanns ganska mycket processorcykler över. Potentiellt skulle man kunna skriva ett synthesizer-program som kan spela musik med olika vågformer etc.

Sammanfattningsvis är jag väldigt nöjd att vi valde vår PDP-11/10 som månadens dator. Det har varit en väldigt trevlig, charmfull dator att pyssla med. Jag kan mycket väl tänka mig att programmera mer på den i framtiden, men skulle då gärna åtminstone ha två fungerande diskettstationer, och ännu hellre en RL02. Tack till vol, bqt, kgs och aap för deras hjälp den senaste månaden. Nu är dock juli slut, och jag skjuter in stolen vid terminalbordet. Men jag vill gärna skriva lite till om RT-11 och dess verktyg...

Introduktion till RT-11

Det roligaste med Updates PDP-11/10 för mig har varit att lära mig att använda RT-11. Det är vad som har varit mest annorlunda mot allt jag är van vid. Låt se om jag kan kommunicera vad jag har lärt mig. **VARNING FÖR YTTERLIGA KVANTITETER ONÖDIG KUNSKAP!**

Först och främst är RT-11 ett väldigt litet, single-taskande system. Det är gjort för att kunna fungera på datorer med så lite som 16Kbyte minne. Den körande applikationen har, givetvis, direkt åtkomst till hela maskinen. Det finns ingen MMU, filskydd eller liknande.

Vid start presenteras användaren med en prompt bestående av en punkt. Detta är prompten för KMON, keyboard monitor, som är systemets skal. Härifrån kan man läsa och skriva i minnet, ladda och spara data mellan minnet och filer, starta program, sätta konfigurationsvariabler, etc. I praktiken är det vanligaste kommandot 'R', som laddar och kör ett program från fil.

```
RT-11SJ    V02C-02H

.R HELLO
HELLO, WORLD

.
```

Filer lagras i filsystem på devices, t.ex. diskettstationer, DECtape, etc. Varje device identifieras av ett namn bestående av två bokstäver och ett kolon. Exempel är `DX:` för RX01 (8-tumsdiskett) och `DP:` för RP02 (hårddisk). En del devices kan ha flera enheter, och dessa anges då med en siffra — `DX0:` för RX01 drive 0, `DX1:` för RX01 drive 1. Om siffran utelämnas avses implicit drive 0. Det

finns också devices som man kan läsa från eller skriva till, men som inte kan lagra filsystem, t.ex. radskrivaren LP: och terminalen TT:.

En fil anges med ett namn bestående av 6+3 tecken (stora bokstäver och siffror), med en punkt mellan dem, och devicenamnet direkt före: t.ex. DX:HELLO.SAV eller DP0:SAWTST.MAC. Inga katalognamn eller sökvägar förekommer, eftersom nästlade kataloger inte stöds. Utelämnas devicenamnet antas defaultdevicen DK:, som kan sättas att vara vilken device som helst (den är ett alias). Efter boot defaultar DK: till att vara bootdevicen SY: (också ett alias). Alias för devicenamn kan sättas med kommandot ASSIGN:

```
.ASSIGN DX1=DK
```

Från och med kommandot ovan avser alltså filnamn som UPTIME.TXT, utan devicenamn, filer på devicen DX1, på samma sätt som när man i MSDOS skriver A: som ett kommando för att välja defaultdevice och sedan kan skriva t.ex. MORE READ.ME för att läsa filen A:READ.ME.

RT-11 var för övrigt ett av de DEC-system som inspirerade Gary Kildall i designen av CP/M, operativsystemet på vilket MSDOS är baserat, från vilket förstås också Microsoft Windows ärvt sitt system med device+filnamn.

Nu kan man frestas att tro att vi enkelt kan lista innehållet på disketten med kommandot DIR, som i MSDOS, men det kan vi inte:

```
.DIR
?ILL CMD?

.
```

I senare versioner av RT-11 dyker det kommandot upp, men i version 2 måste vi använda oss av universalfilverktyget PIP:

```
.R PIP
*
```

Asterisken indikerar att PIP väntar på input. Kommandona för PIP är tämligen obskyra, men för att lista katalogen på defaultdevicen använder vi /L:

```
.R PIP
*/L

MONITR.SYS    46 20-NOV-75
EDIT  .SAV    19 20-NOV-75
MACRO .SAV    31 20-NOV-75
LINK  .SAV    25 20-NOV-75
PIP   .SAV    14 20-NOV-75
5 FILES, 135 BLOCKS
 345 FREE BLOCKS
*
```

Siffrorna bredvid filerna anger antalet block om 512 bytes de tar upp på disken. Alla filer tar upp sammanhängande raddor av block på disken. Dvs om du har "boxat in" en fil genom att skapa en till fil direkt efter den på disken, så kan du inte fortsätta att göra den första filen längre genom att skriva i den. Det går dock bra att skapa en ny fil med samma namn som en existerande fil och flytta över innehållet till den; när den nya filen stängs ersätter den den gamla, som istället blir ledigt

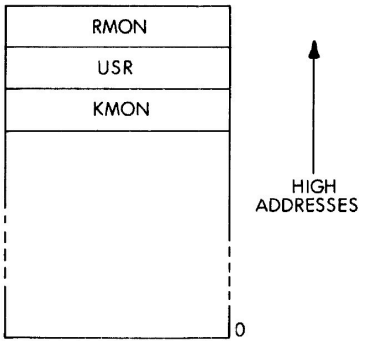
utrymme på disken. Med tiden tenderar disken att fyllas av hål där filer varit, och som är för små för att använda till nya filer. Då kör man PIP-kommandot /S för att flytta om filerna så att allt ledigt utrymme hamnar sist på disken.

Asterisken är förresten inte egentligen PIP:s prompt. Den tillhör CSI, command string interpreter, som är en del av operativsystemet, och används av applikationsprogram för att läsa in en kommandorad. Kommandoraden tar ett visst format, som används av alla systemverktygen:

```
UTFILER=INFILER/SWITCHAR
```

Upp till tre utfiler kan anges, separerade med kommatecken, och upp till sex infiler. Filändelserna kan utelämnas, och antar då defaults som är specifika per applikation. Efter ett utfilnamn kan man också lägga till [n] där n är ett antal block som ska reserveras för filen när den skapas. CSI hanterar öppnande och skapande av angivna filer helt automatiskt åt applikationen.

CSI ingår i en del av operativsystemet som heter USR, user service routine. Det är den delen som hanterar alla filsystemsrelaterade operationer. Denna del av operativsystemet är inte nödvändigtvis resident i minnet, utan laddas auotmatiskt in när applikationen gör ett systemanrop som kräver fil-I/O. Den residenta delen av systemet kallas RMON, och bor högst upp i minnet. USR kommer direkt under (om den är laddad i minnet för tillfället), och resten av minnet där under är tillgängligt för applikationen. KMON laddas i denna nedre del, och fungerar som ett applikationsprogram. RMON tar i SJ-versionen av RT-11 V2 upp cirka 3Kbyte minne, och USR cirka 4Kbyte.



Förutom dessa delar laddas också devicehanterare in i minnet vid behov. Detta sköts automatiskt av USR när applikationen gör systemanrop som kräver kommunikation med en device. Vill man undvika detta in- och urladdande av devicehanterare kan man manuellt ladda dem i minnet med KMON-kommandot LOAD, varefter de förblir residenta tills man gör UNLOAD. USR kan också låsas i minnet med systemanropet .LOCK.

Exekverbara program lagras i filer med ändelsen .SAV, så kallade save-filer. Dessa kan skapas med KMON-kommandot SAVE, som sparar en image av minnet i en fil. En save-fil kan laddas till minnet med GET, och en laddad image kan startas med START. Defaultstartadressen står i adress 40 (oktalt) i minnet, i en area med diverse systeminformation. Man kan också ange en startadress till START-kommandot. Kommandot R gör GET följt av START. I arean längst ner i minnet mellan adress 0 och 477 finns också bland annat interruptvektorer, jobbstatusflaggor, samt pekare till USR och toppen av applikationsminnet. Omedelbart ovanför finns applikationsprogrammets stack, som av default börjar vid adress 1000 (och växer neråt).

För att återgå till vår PIP-session; /L är alltså egentligen en switch, som anges efter en filspecifikation. Vi kan lista en annan device med t.ex. DX1:/L, vi kan ange filnamn före (med wildcards) om vi vill lista specifika filer, och vi kan ange diverse andra switchar för att göra andra saker än att lista katalogen: /S för "squeeze"-operationen som nämndes längre upp på sidan, /D med ett filnamn för delete, UTFIL=INFIL för att kopiera, osv.

Nu efter att vi har listat katalogen kan vi se att PIP (eller egentligen CSI) återigen har skrivit ut en *-prompt, och väntar på ett till kommando. Detta är normalt beteende i RT-11, och vi kommer tillbaka till KMON genom att trycka Ctrl-C:

```
 345 FREE BLOCKS
*^C

.
```

OK. Låt oss testa att skriva ett program! Det finns flera språk tillhands, men jag tänker välja

assembler, närmare bestämt MACRO, för att det såklart är det som är roligast att lära sig på en obekant datorplattform.

För att skriva och bygga ett assemblerprogram behöver vi tre verktyg: `EDIT`, `MACRO`, och `LINK`. Det helt klart roligaste av dessa är `EDIT`, och det är det vi börjar med. Vi ska skapa en källkodsfil med namnet `HELLO.MAC`, och skriva lite kod i den. (I terminalutskriften nedan är dollartecknen hur editorn ekar tangenten Escape, som används för att terminera kommandon i editorn.)

```
.R EDIT
*EWHELLO.MAC$$
*I      .MCALL ..V2..,.PRINT,.EXIT
        ..V2..
START:  .PRINT /HELLO, WORLD/
        .EXIT
        .END START
$$
*EX$$
.
```

Vi startar `EDIT`, som skriver ut en asterisk. Vi skriver in kommandot `EW` följt av ett filnamn, vilket sätter namnet på filen vi vill öppna för skrivning. Vi trycker escape för att avsluta filnamnet, och ett till escape för att köra kommandot. Editorn skriver ut en asterisk igen.

Vi skriver kommandot `I`, för Insert, och börjar skriva in vår kod. När vi har matat in allt vi vill trycker vi escape för att avsluta inmatningen av text, och ett till escape för att köra insert-kommandot. Vi får en till asterisk.

Assemblerkoden vi har skrivit är enkel: `.MCALL` är ett assemblerdirektiv som deklarerar att vi vill använda de följande systemmakrona. `..V2..` säger att vi använder version 2 av systemmakrona. `.PRINT` gör ett PRINT-systemanrop med en sträng. `.EXIT` gör ett EXIT-systemanrop, som avslutar programmet. Dessa tre är alltså makron och kommer från ett systembibliotek. `START:` är en label, `/HELLO, WORLD/` är en sträng, och `.END` är ett direktiv som avlutar assemblerfilen. Vi går vidare.

`EX` skriver bufferten till utfilen och avslutar editorn. KMON ger oss en punkt. Nu ska vi köra assemblern.

```
.R MACRO
*HELLO=HELLO
****** Q
3 000000          START:  .PRINT /HELLO, WORLD/
****** A
  000000 016700 .IIF NB </HELLO>,          MOV      /HELLO,%0
         000000'
ERRORS DETECTED: 2
FREE CORE: 5855.  WORDS

*
```

Assemblern ger oss en asterisk, och vi anger att utfilen ska heta "HELLO" och infilen ska heta "HELLO". Assemblern defaultar till ".OBJ" för utfilens ändelse, och ".MAC" för infilen. Vi hade också kunnat skriva `HELLO.OBJ=HELLO.MAC`.

Dock ser vi att vi har ett par fel i vår kod; assemblern har skrivit ut felmeddelanden. `Q` är "questionable syntax", och `A` är adresseringsfel. Ah: vi har klantat oss med parametern till `.PRINT`-makrot; vi kan inte bara skriva en sträng sådär, utan vi måste stoppa den i minnet nånstans och ange adressen. Vi får fixa vår kod. Vi trycker Ctrl-C för att avsluta assemblern, och startar editorn igen. Den här gången använder vi kommandot `EB` som öppnar en existerande fil för både läsning och skrivning (med backup).

```
*^C

.R EDIT
*EBHELLO.MAC$RG/$=JSKI#HELLO
$AIHELLO:       .ASCIZ $U$$
*B/L$$
        .MCALL ..V2.,,.PRINT,.EXIT
        ..V2..
START:  .PRINT #HELLO
        .EXIT
HELLO:  .ASCIZ /HELLO, WORLD/
        .END START
*EX$$

.
```

Denna bokstavssoppa kräver förklaring. `EBHELLO.MAC$` är som jag beskrev ovan. Därefter följer kommandot `R`, Read, som läser in första sidan av infilen. Editorn är nämligen sidorienterad, och man läser bara in en sida i taget i minnet. Sen redigerar man sidan, skriver den till utfilen, och läser in nästa sida, osv. Man kan inte gå tillbaka till en tidigare sida när man väl har gått vidare. Sidor delas från varandra i filen med form feed-tecken, och rekommenderas av manualen att vara 50-60 rader. Men det påverkar inte oss, vår fil är kort.

Kommandot `G` söker efter en text (avslutad med escape); här söker vi efter /-tecknet som börjar vår hello-sträng. Efteråt står markören efter detta hittade tecken i bufferten. Vi gör `=J`, som är Jump med parametern "=", som betyder "så många tecken bakåt som strängen vi just skrev i förra kommandot". I det här fallet var det bara ett tecken, så vi hade lika enkelt kunnat skriva `-1J`, men "=" är ett smidigt sätt att kunna referera tillbaka till texten man just sökt efter för att ta bort den eller ändra den utan att behöva räkna efter hur många tecken det är.

Nu står vi före /-tecknet. Där gör vi `S`, som är Save, som sparar resten av raden från där vi står i en kopieringsbuffert som editorn har. Därpå kör vi `K`, Kill, som tar bort resten av raden från där vi står, och så `I`, Insert, och den nya kodsnutten `#HELLO`, enter, och escape. (Tanken är att vi ska skapa en label längre ner som vi kallar "HELLO", och ange dess address som värde här. "#" i PDP-11-assembler anger en immediate.)

Nu står vi på raden med `.EXIT`. Vi gör Advance-kommandot `A` för att gå till början av nästa rad, och infogar där en label "`HELLO:`" följt av en tabb och strängdirektivet "`.ASCIZ`". Därefter klistrar vi in den undankopierade hello-strängen med Unsave-kommandot, `U`. Escape escape kör hela kommandoraddan, och vi får en asterisk.

Här ville jag se att koden blev rätt, vilket kan vara trevligt då och då eftersom man inte ser vad man gör medans man redigerar. Så jag använde kommandot `B` för att gå till början av bufferten, och `/L` som är List med parametern "/", som betyder "härifrån till slutet av bufferten". Hela bufferten skrivs således ut på terminalen. Koden ser rätt ut, så vi sparar och avslutar editorn med `EX`.

Sen kör vi assemblern igen:

```
.R MACRO
*HELLO=HELLO
ERRORS DETECTED: 0
FREE CORE: 5807.  WORDS

*
```

Aha! Nu gick det bra.

Då har vi fått en `HELLO.OBJ`, som vi kan mata till länkaren så att vi får en körbar `HELLO.SAV`:

```
*^C

.R LINK
*HELLO=HELLO


*
```

Det gick också bra, och länkaren har också defaultfiländelser, som vi kan se. Nu kan vi testa att köra vårt program:

```
*^C

.R HELLO
HELLO, WORLD

.
```

Och så enkelt är det! :)

Men där får det räcka. Jag skulle kunna skriva en massa om hur systemmakrona fungerar, vad det finns för utbud av systemanrop, hur RT-11 hanterar öppna filer och hur man allokerar minne, med mera, men den här artikeln är redan tillräckligt lång och jag måste gå och sova. Till den som läste ända hit: Tack för intresset!

Den kommande månadens dator

Augusti månads dator är vårt Asteroids-arkadspel, som vi ställt i utställningsrummet och gärna vill få i fungerande skick för gäster att spela på.

Närmast planerade aktiviteter

Inflyttningsfest kommer att hållas den 27:e augusti med mat och dryck. Vidare detaljer meddelas vid senare tillfälle.

Studiebesök på The Svedberg-laboratoriet som nu läggs ner äger rum den 24:e, för en förbokad grupp om 10 personer. Vill du anmäla intresse i händelse av återbud, klicka här. Observera att datumet redan är bestämt.

Plenumsmöte äger rum den 24:e augusti klockan 18:30.